Manuale di utilizzo di

OpenSCAD

Programma opensource per disegnare in 3D

Tratto da: http://en.wikibooks.org/wiki/OpenSCAD_User_Manual
ttp://edutechwiki.unige.ch/en/OpenScad_beginners_tuhtorial

Traduzione e adattamento di Giuseppe Tamanini

Ultimo aggiornamento: 22/08/2023

OpenSCAD è un software per la creazione di solidi 3D oggetti CAD. Si tratta di un software libero ed è disponibile per GNU / Linux, MS Windows e Apple OS X.

A differenza di molti software liberi per la creazione di modelli 3D (come la ben nota applicazione Blender), OpenSCAD non si concentra sugli aspetti artistici della modellazione 3D, ma si concentra invece sugli aspetti CAD. È l'ideale per creare modelli 3D di parti di macchine ma non adatto alla creazione di film animati al computer.

OpenSCAD non è un software che permette di utilizzare il mouse per disegnare ma, richiede la digitazione di comandi relativamente semplici in un linguaggio di programmazione (script). Questo dà il pieno controllo del processo di modellazione e permette di cambiare facilmente qualsiasi fase del processo di modellazione, o anche per la produzione di disegni che sono definiti dai parametri configurabili.

OpenSCAD fornisce due principali tecniche di modellazione: In primo luogo, la geometria solida costruttiva (CSG) e l'estrusione di profili 2D in formato DXF dai quali è anche possibile leggere i parametri di progetto. Oltre a leggere i file DXF, OpenSCAD può anche leggere e creare modelli 3D in formati di file STL e OFF.

OpenSCAD non usa unità di misura come millimetri, centimetri o pollici. Se definisci un cubo ([10,10,10]) non ottieni un cubo di 10 per 10 per 10 cm o mm, ottieni solo un cubo di 10 per 10 per 10. Sarà lo slicer – il software che produce il file per la stampa 3D dell’oggetto – che decide che unità utilizzare quando carichi il modello esportato. Io usando Slic3r ora PrusaSlicer ottengo le unità in mm. Io tutto quello, ad ora, ho disegnato con OpenSCAD l’ho pensato in mm. In questa guida, quindi, potete considerare come unità di misura usata da OpenSCAD i millimetri.

OpenSCAD può essere scaricato dal http://openscad.org/. Sul sito ufficiale si possono trovare informazioni aggiuntive nella mailing list.

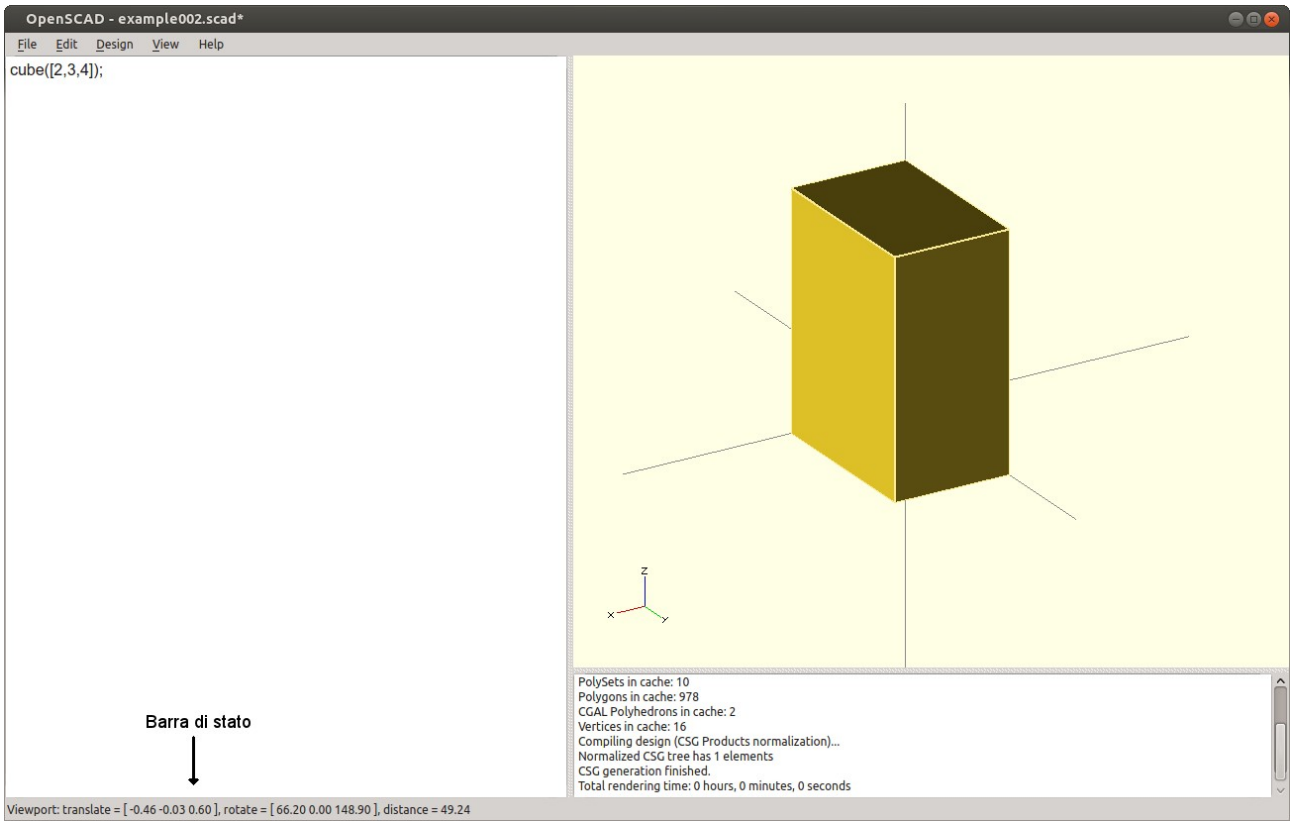
Per il nostro primo modello creeremo un semplice parallelepipedo da 2 mm x 3 mm x 4 mm. Nell'editor openSCAD, digitare la seguente riga di comando:

```
cube([2,3,4]);
```

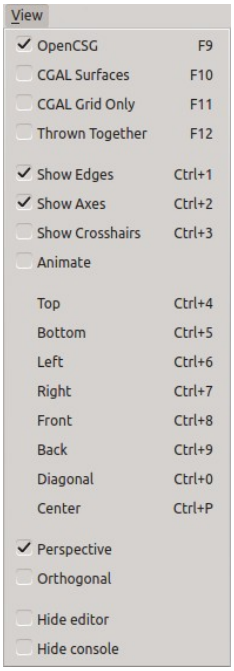
Quindi premere **F5** per avere un'anteprima grafica di quello che avete digitato.

È possibile modificare la vista dell'oggetto presente nella finestra di anteprima in tre modi:

- Trascinando l'oggetto con il tasto sinistro del mouse per ruotare la vista. Nella barra di stato cambieranno i valori di rotazione (rotate = ...).
- Trascinando con il tasto destro del mouse per spostare la vista. Nella barra di stato cambieranno i valori di (translate = ...).
- Usando la rotellina del mouse per ingrandire e rimpicciolire. In alternativa è possibile utilizzare i tasti + e -, o premendo il tasto destro del mouse trascinarlo tenendo premuto il tasto Maiusc. Nella barra di stato cambieranno i valori di (distance = …).



Menu View

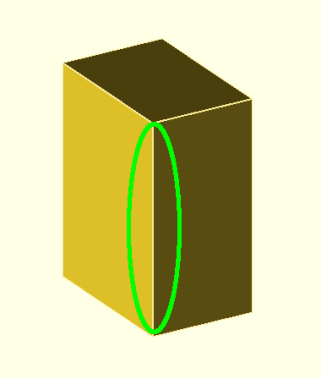


L'area di visualizzazione può essere configurata in modo da avere diversi metodi di rendering[1] e altre opzioni utilizzando il menu **Visualizza**. La maggior parte delle opzioni qui descritte sono disponibili anche utilizzando una combinazioni di tasti.

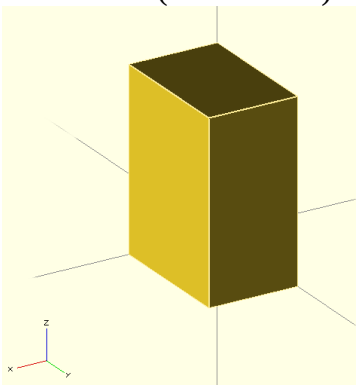
OpenCSG - Tasto **F9** (attivata all'avvio del programma)
In modalità rendering OpenCSG viene utilizzata la libreria OpenCSG contenente le avanzate funzionalità di OpenGL (2.0) per generare la vista dell'oggetto.
Questo metodo produce risultati istantanei, ma ha un frame rate (frequenza dei fotogrammi) basso quando si lavora con oggetti molto complessi.
Si noti che selezionando la modalità OpenCSG usando **F9** passerà l'ultima vista OpenCSG generato, senza ricaricare il codice sorgente. Se sono state fatte delle modifiche al codice sorgente utilizzare la funzione di compilazione (**F5**, che si trova nel menu **Design**) .

CGAL Surface (facce) – Tasto **F10**
GGAL Grid Only (solo griglia) - Tasto **F11**
Usano la modalità OpenGL semplice
Tralascio le spiegazioni perché queste due modalità visualizzano solo gli oggetti semplici.

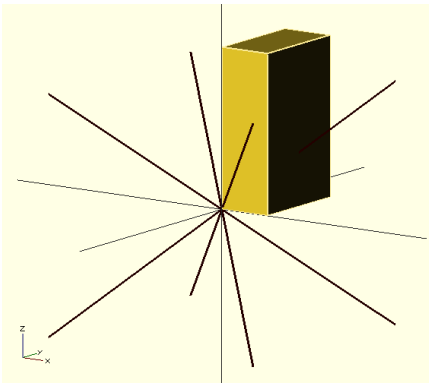
Show Edges (Mostra Bordi) – Tasti **Ctrl +1**
Se l'opzione **Show Edges** è attivata, oltre che le facce, verranno visualizzati anche i bordi con un colore chiaro.



Show Axes (Mostra Assi)- Tasti **Ctrl +2**



Se l'opzione **Show Axes** è attivata, verranno visualizzati gli assi di un sistema ortogonale tridimensionale. Inoltre, un indicatore più piccolo contenente i nomi degli assi (x in rosso, y in verde e z in blu) viene visualizzato nell'angolo in basso a sinistra dell'area di visualizzazione.



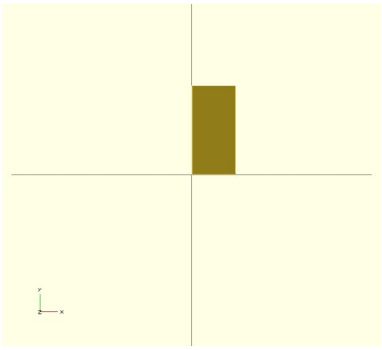
Show Crosshairs (Mostra Mirino) - Tasti **Ctrl +3**
Se l'opzione Show Crosshairs è attivata, il centro della finestra sarà indicato da quattro linee diagonali che puntano al centro del sistema di coordinate. Questo è utile per allineare l'area di visualizzazione di un particolare modello in modo da mantenerlo centrato sullo schermo durante la rotazione.

1 Il **rendering** è un termine della lingua inglese che in senso ampio indica la resa grafica, ovvero un'operazione compiuta da un disegnatore per produrre una rappresentazione di qualità di un oggetto o di una architettura (progettata o rilevata). In tempi relativamente recenti ha assunto un valore essenzialmente riferito all'ambito della computer grafica; dove identifica il processo di "resa" ovvero di generazione di un'immagine a partire da una descrizione matematica di una scena tridimensionale interpretata da algoritmi che definiscono il colore di ogni punto dell'immagine digitale. La descrizione è data in un linguaggio o in una struttura dati e deve contenere la geometria, il punto di vista, le informazioni sulle caratteristiche ottiche delle superfici visibili e sull'illuminazione.
-Tratto da Wikipedia -

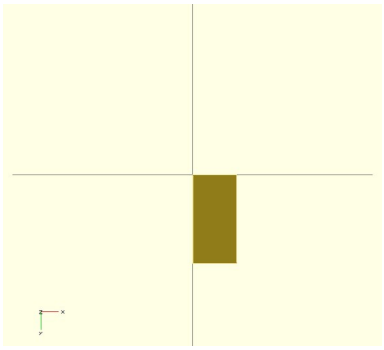
Animate (Animazione)
L'opzione Animate aggiunge la barra di animazione al bordo inferiore dello schermo. Non appena verranno introdotti i valori di FPS (fotogrammi al secondo) e steps (passi) (valori consigliati sono 10 e 100, rispettivamente), il campo time (tempo) viene incrementato di 1/Steps, FPS volte al secondo, fino a quando non raggiunge il valore 1, poi riparte da 0.

Vista

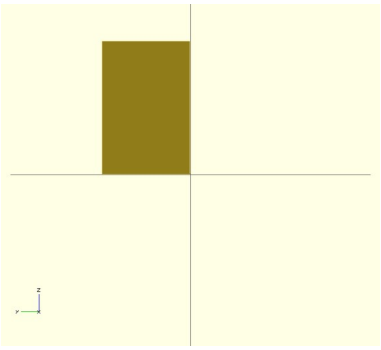
Top (da sopra) – Tasti **Ctrl + 4**



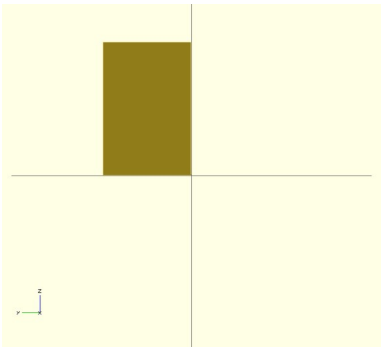
Bottom (dal basso) – Tasti **Ctrl + 5**



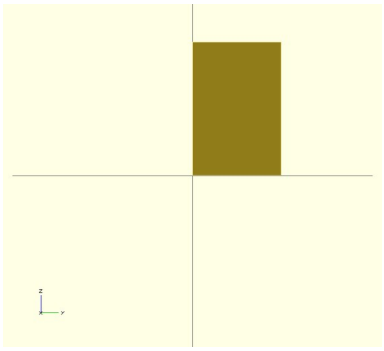
Left (da sinistra) – Tasti **Ctrl + 6**



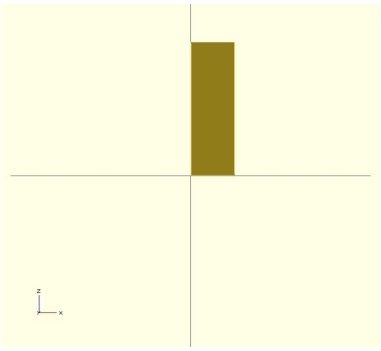
Right (da destra) – Tasti **Ctrl + 7**



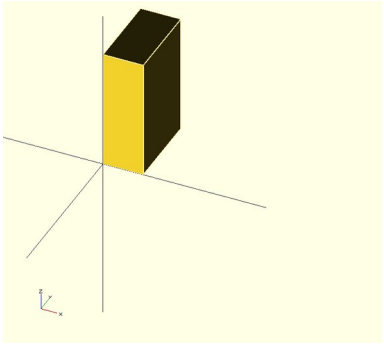
Front (da davanti) – Tasti **Ctrl + 8**



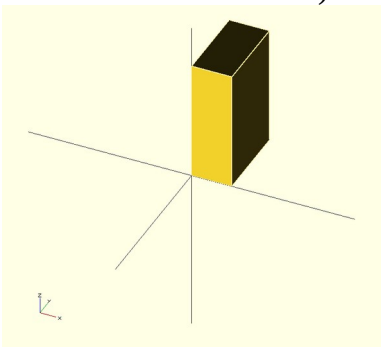
Back (da dietro) – Tasti **Ctrl + 9**



Diagonal (in diagonale) – Tasti **Ctrl + 0**
(visualizzazione proposta all'avvio del programma)



Center (al centro) – Tasti **Ctrl + P**
(posiziona il centro degli assi al centro dello schermo senza cambiare la vista)



Perspective
Per impostazione predefinita, la vista è in modalità Perspective (prospettiva), il che significa che i lati più lontani si vedranno più corti, come si vedesse dal vero con gli occhi o in una foto.

Orthogonal
Quando la modalità di visualizzazione viene impostata a Orthogonal (ortogonale), le lunghezze dei lati non dipenderanno dalla distanza (la vista simulerà una telecamera a distanza infinita). Questo è particolarmente utile in combinazione con le opzioni Top, Bottom, Left, Right, Front, Back, descritte precedentemente, perché l'immagine appare in 2D come quella che si vede in un disegno meccanico.

cube

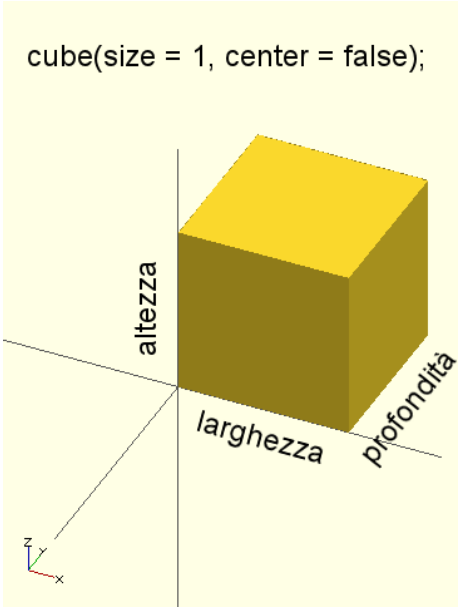
Disegna un parallelepipedo. I nomi degli argomenti, se riportati nell'ordine specificato, sono opzionali.

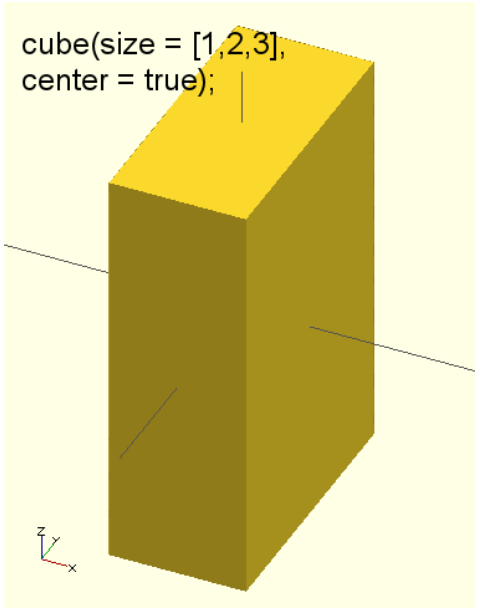
Parametri
size
Numero decimale - Se il numero è unico, il risultato sarà un cubo con lato di tale lunghezza. Se vengono indicati tre valori che corrispondono alle lunghezze dei lati sugli assi X, Y, e Z (larghezza,profondità,altezza). Se il parametro size non non viene indicato sarà disegnato un cubo di lato 1.

center
Booleano - Determina come verrà posizionamento dell'oggetto. Se è impostato a true (vero), il parallelepipedo viene centrato in (0,0,0). Altrimenti, il cubo è posto nel quadrante positivo con un angolo a (0,0,0). Se non viene indicato center verrà impostato a false (falso).

Esempi di utilizzo
cube(size = 1, center = false);
comando semplificato
cube(10)

cube(size = [1,2,3], center = true);
comando semplificato
cube([1,2,3],true);





Altri esempi:
cube(6,true);
cube([4,6,10]);

Le variabili speciali **$fa**, **$fs** e **$fn** impostano il numero di settori (lati di un poligono regolare) da utilizzare nel generare il cerchio:

$fa è l'angolo minimo del settore. Un cerchio può contenere massimo 360 frammenti: 360 diviso questo valore. Il valore predefinito è 12 (che disegna un poligono a 30 lati). Il valore minimo consentito è 0.01 (0,01). Un valore inferiore provocherà un errore.

$fs è la dimensione minima di un settore. Usando questa opzione i cerchi molto piccoli hanno un numero di settori inferiore a quello specificato usando $fa. Il valore predefinito è 2. Il valore minimo consentito è 0.01 (0,01). Un valore inferiore provocherà un errore.

$fn è in genere 0. Quando questa variabile ha un valore maggiore di zero, le due variabili precedenti vengono ignorate e il cerchio viene disegnato con questo numero di settori. Il valore predefinito è 0.

Maggiore è il numero di settori, maggiore è il consumo di memoria e CPU. A seconda del disegno, i valori $fn, di $fa e $fs devono essere mantenuti piccoli, almeno fino a quando il disegno non viene completato. Un $fn oltre 100 non è raccomandato o usato solo in circostanze specifiche, un valore inferiore a 50 è quasi sempre consigliabile.

sphere

Disegna una sfera all'origine del sistema di coordinate. Il nome dell'argomento è facoltativo. La sfera visualizzata appare, in base alla dimensione, con sfaccettature. Le variabili $fa, $fs e $fn permettono di migliorare la qualità di visualizzazione (risoluzione) dell'immagine.

Parametri
r
Numero decimale - Numero che indica il raggio della sfera. Se non indicato verrà usato il valore 1.

$fa
Angolo minimo del settore

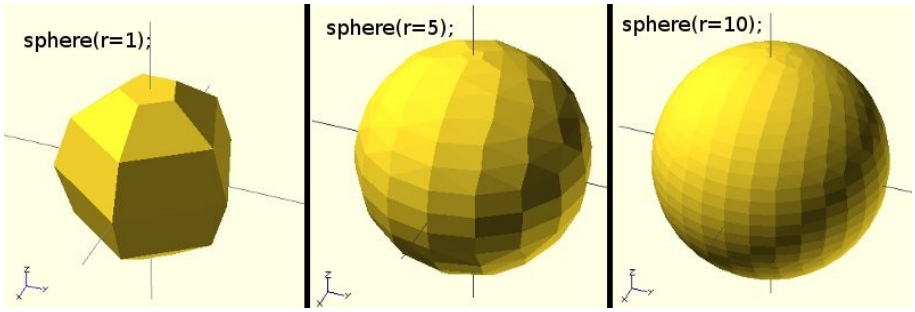
$fs
Dimensione minima del settore

$fn
Numero dei settori

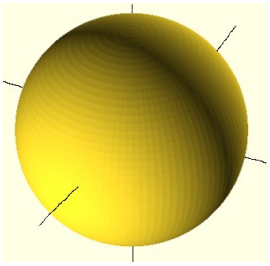
Esempi di utilizzo:
sphere(r = 1);
comandi semplificato:
sphere(1);

sphere(5);

sphere(10);



Altri esempi



Disegna una sfera di raggio di 2 mm ad alta risoluzione
sphere(2,$fn=100);

Disegna una sfera di raggio 2 mm ad alta risoluzione
sphere(2,$fs=0.01);

cylinder

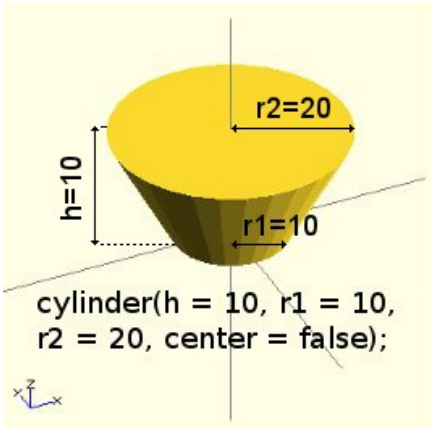
Disegna un cilindro o un cono. I nomi degli argomenti, se riportati nell'ordine specificato, sono opzionali. L'oggetto visualizzato appare, in base alla dimensione, con sfaccettature. Le variabili $fa, $fs e $fn permettono di migliorare la qualità di visualizzazione (risoluzione) dell'immagine.

Parametri
h
Numero decimale - Specifica l'altezza del cilindro. Se il parametro h non non viene indicato sarà disegnato un cilindro o un cono di altezza 1.

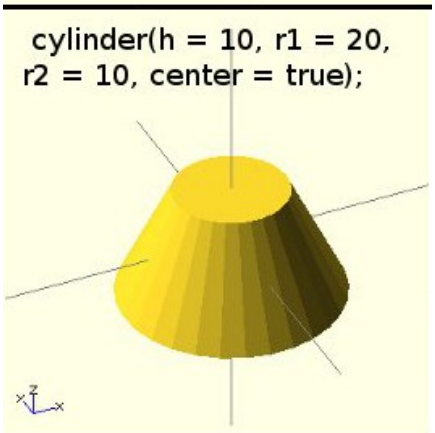
r1
Numero decimale - Specifica il raggio del cerchio di base del cono. Se non indicato verrà usato il valore 1.

r2
Numero decimale - Specifica il raggio del cerchio in cima al cono. Se non indicato verrà usato il valore 1.

r
Numero decimale - Specifica il raggio del cilindro. Se il parametro r non non viene indicato sarà disegnato un cilindro di raggio 1.



center
Boolean.
Determina come verrà posizionamento dell'oggetto. Se è impostato a true (vero) viene centrato in (0,0,0). Altrimenti posto nel quadrante positivo con un angolo a (0,0,0). Se non indicato center verrà impostato a false (falso).



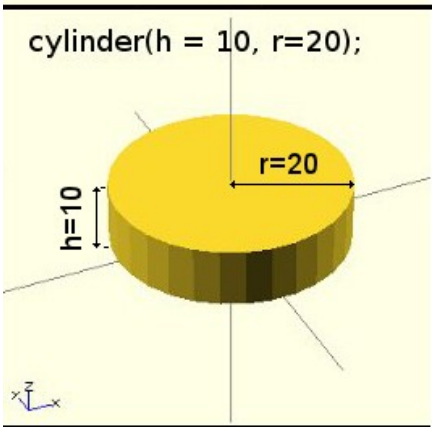
$fa
Angolo in gradi

$fs
Angolo in mm

$fn
Risoluzione

Esempi di utilizzo:
cylinder(h = 10, r1 = 10, r2 = 20, center = false);
comando semplificato
cylinder(10,10,20);

cylinder(h = 10, r1 = 20, r2 = 10, center = true);
comando semplificato
cylinder(10,20,10,true);



cylinder(h = 10, r=20);
cylinder(h = 10, r=20, $fs=6);

N.B.: Se si vuole disegnare un cilindro il comando non può essere semplificato.

polyhedron

Disegna un poliedro con un elenco di punti e un elenco di triangoli. I punti in elenco sono quelli dei vertici della forma, i triangoli sono quelli delle superfici del poliedro.

Sintassi
polyhedron(points = [[x, y, z], ...], triangles = [[p1, p2, p3..], ...], convexity = N);

Parametri
points
vettore - Specifica le coordinate [x,y,z] dei vertici della forma. I punti della serie saranno "etichettati" con un numero a partire dallo 0: il primo vertice elencato sarà il punto 0, il secondo il punto 1, il terzo il punto 2, l'ultimo sarà il punto (numero dei punti -1).

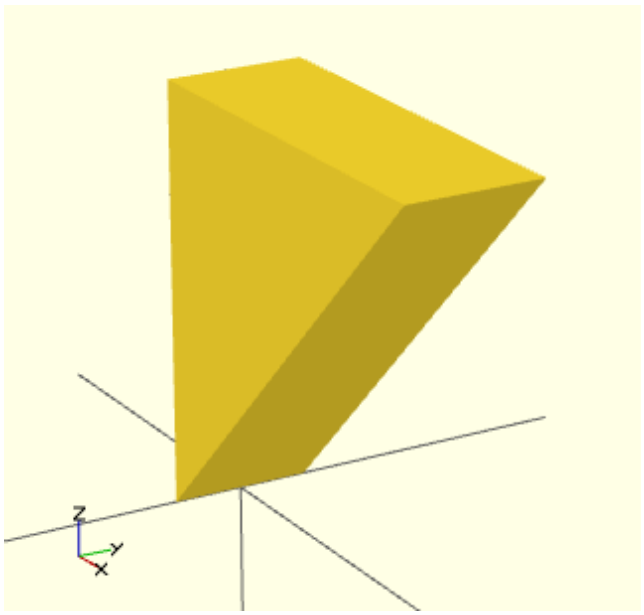
triangles
vettore – Specifica i punti dei vertici dei triangoli elencati in senso orario (Es. [0,1,2]) che serviranno a chiudere le varie facce del poliedro. Se la faccia è rettangolare bisognerà tracciare i due triangoli creati dalla diagonale.

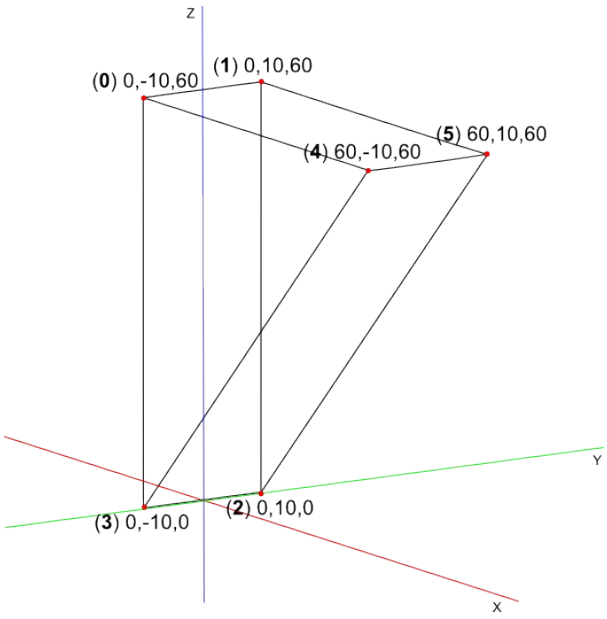
convexity
Questo parametro è necessario solo per visualizzare correttamente l'oggetto in OpenCSG e non avendo alcun effetto sulla resa del poliedro verrà tralasciato.

Esempio di utilizzo

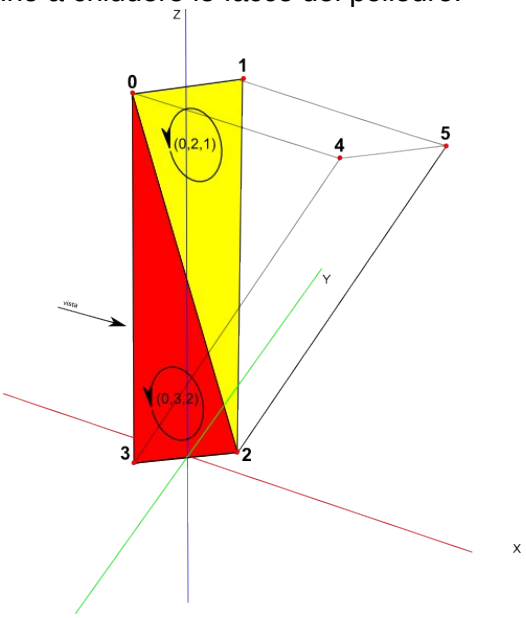
```
polyhedron (
        points = [ [0,-10,60], [0,10,60], [0,10,0], [0,-10,0], [60,-10,60], [60,10,60] ],
        triangles = [[0,3,2], [0,2,1], [3,0,4], [1,2,5], [0,5,4], [0,1,5], [5,2,4], [4,2,3] ]
                );
```

La serie di coordinate indicate dal parametro:
0 1 2 3 4 5
points = [[0,-10,60], [0,10,60], [0,10,0], [0,-10,0], [60,-10,60], [60,10,60] sono quelle dei sei vertici del poliedro che saranno automaticamente etichettate con i numeri da 0 a 5.

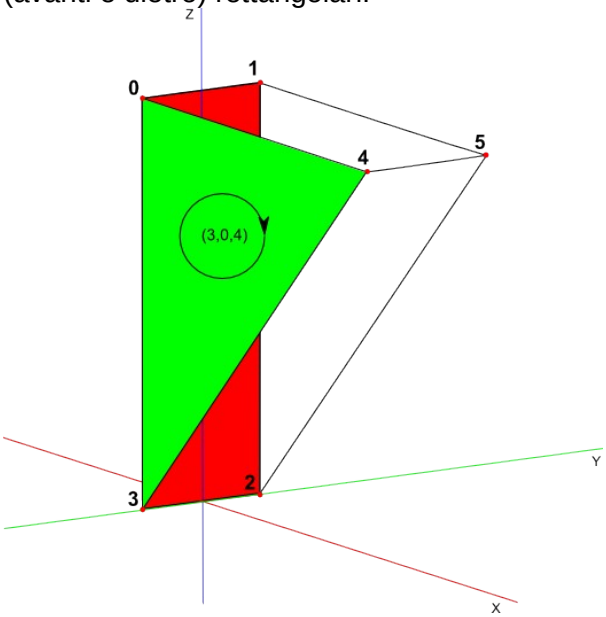


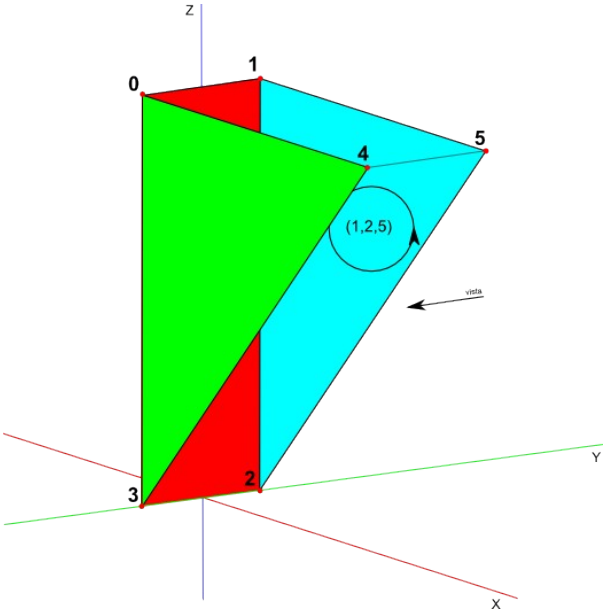


La serie dei punti specificate dal parametro triangles andranno a chiudere le facce del poliedro:
Le serie [0,3,2], [0,2,1] chiuderanno con due triangoli la faccia di sinistra. Il disegno seguente la riporta vista dall'interno e quindi la rotazione è rappresentata in senso antiorario.

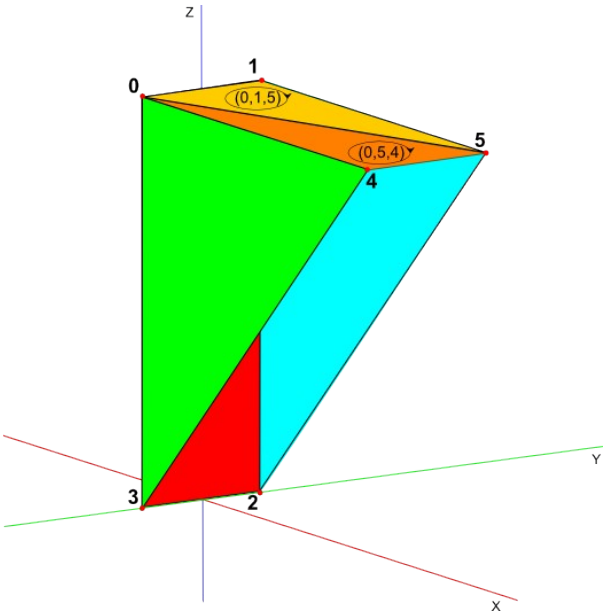


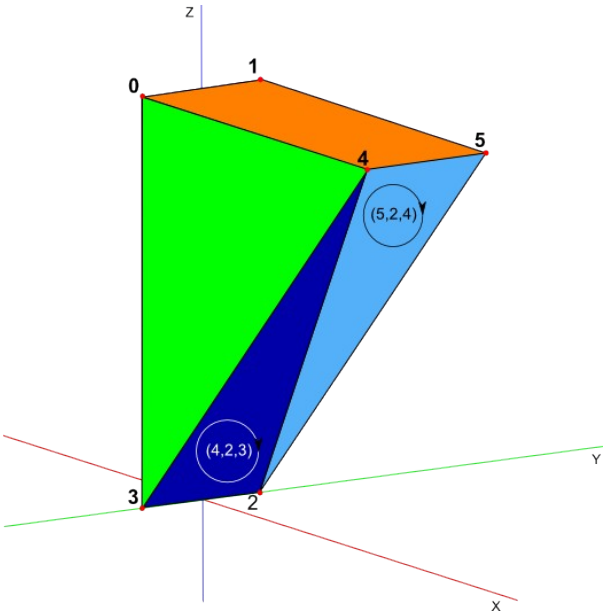
Le due serie seguenti [3,0,4], [1,2,5] chiudono le due facce (avanti e dietro) rettangolari.



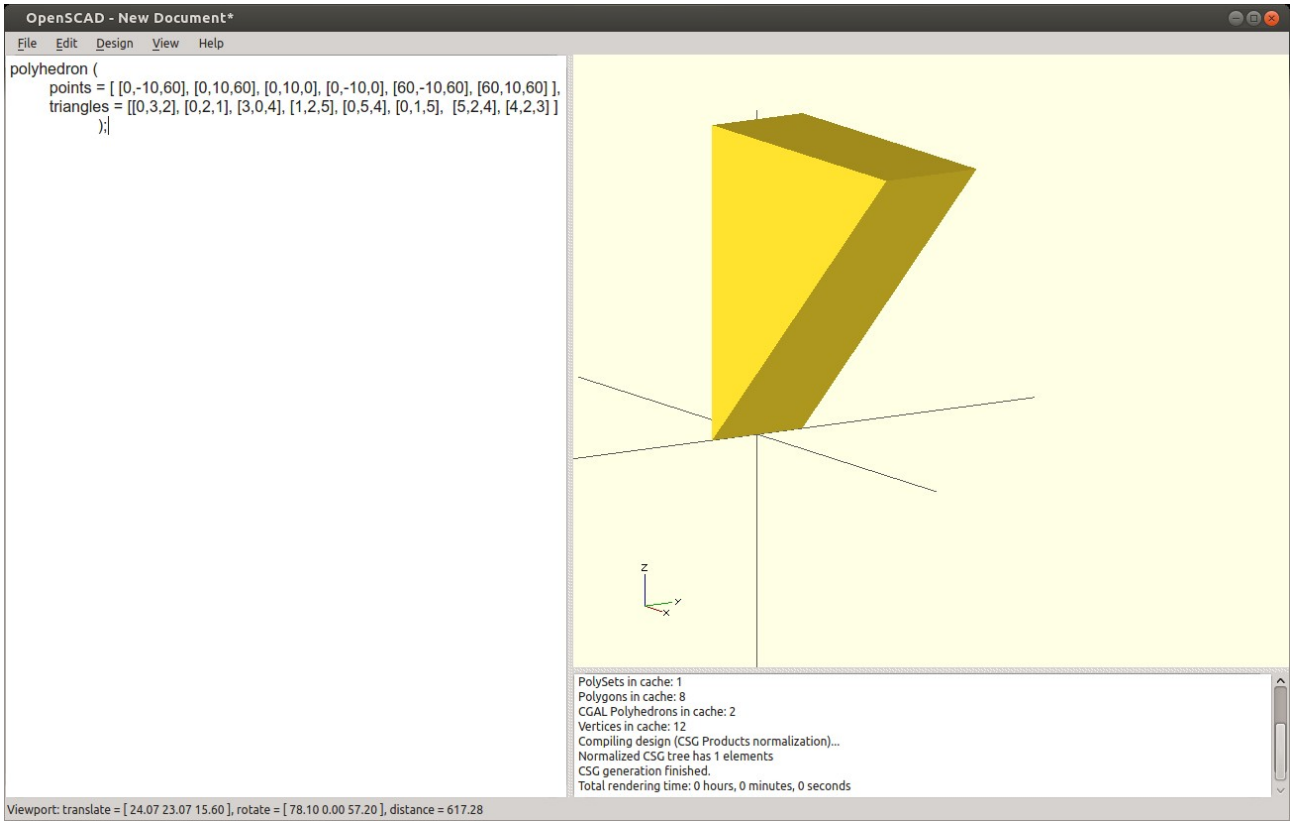


Le serie [0,5,4], [0,1,5] e [5,2,4], [4,2,3] chiuderanno, ciascuna, con due triangoli le facce in alto e a destra.





Ecco la finestra del programma con il poliedro completato:



Comandi di trasformazione

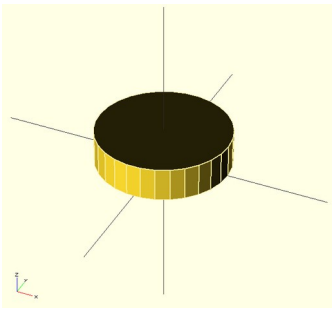
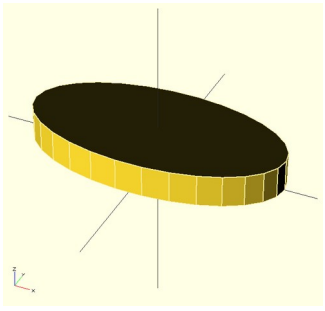
Scale
Scala gli elementi sspecificati utilizzando il vettore specificato. Il nome dell'argomento è facoltativo.

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

Esempio di utilizzo:ì

Creare un ovale di larghezza doppia della profondità.

scale (v=[2,1,1]) cylinder(h=10, r=20);
comando semplificato
scale ([2,1,1]) cylinder(h = 10, r=20);

rotate

Sintassi

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

Ruota gli elementi specificati in base ai valori indicati dal vettore [x,y,z]. I nomi degli argomenti sono opzionali.

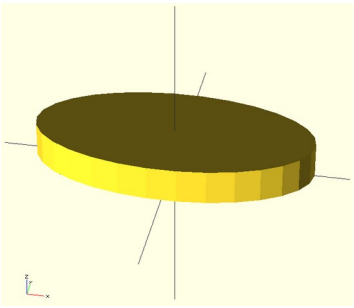
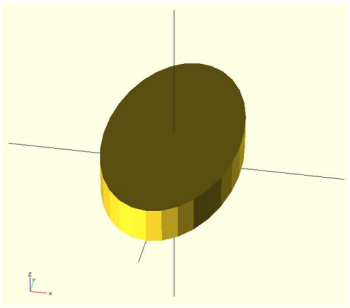
Parametri

a
Vettore – Specifica la rotazione su ogni singolo asse in gradi [deg_x,deg_y,deg_z].

Esempio di utilizzo

Ruotiamo di 90 gradi in senso antiorario sull'asse z l'ovale creato nell'esempio precedente.

rotate(a=[0,0,90]) scale(v=[2,1,1]) cylinder(h=10, r=20);
comando semplificato
rotate([0,0,90]) scale([2,1,1]) cylinder(h=10,r=20);

translate

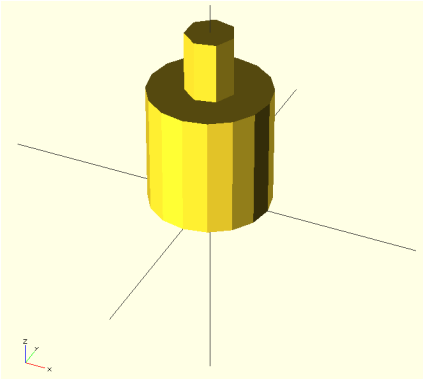
Sintassi
translate(v = [x, y, z]) { ... }

Transla (muove) gli elementi specificati in base ai valori indicati dal vettore [x,y,z]. Il nome dell'argomento è opzionale. Le parentesi graffe possono essere omesse se l'unico elemento da traslare viene scritto di seguito.

Parametri
v
Vettore – Specifica lo spostamento in mm su ogni singolo asse [x,y,z].

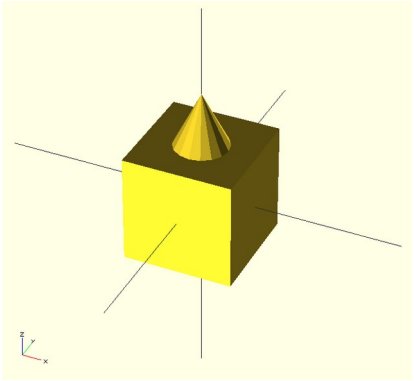
Esempi di utilizzo

Disegna un cilindro di raggio 5 e altezza 10, poi appoggiamo sopra di esso un altro cilindro di raggio 2 e altezza 5.



```
cylinder(r=5, h=10);
translate(v=[0,0,10]) {
    cylinder(r=2, h=5);
}
```
comando semplificato
```
cylinder(r=5, h=10);
translate([0,0,10]) cylinder(r=2, h=5);
```

Disegna un cubo di lato 20, poi appoggiamo sopra un cono di raggio 5 e altezza 10.



```
cube(size=20, center=true);
translate(v=[0,0,10]) {
    cylinder(r1=5,r2=0,h=10);
}
```
comando semplificato
```
cube(20,true);
translate([0,0,10]) cylinder(r1=5,r2=0,h=10);
```

mirror

Sintassi
mirror([x, y, z]) { ... }

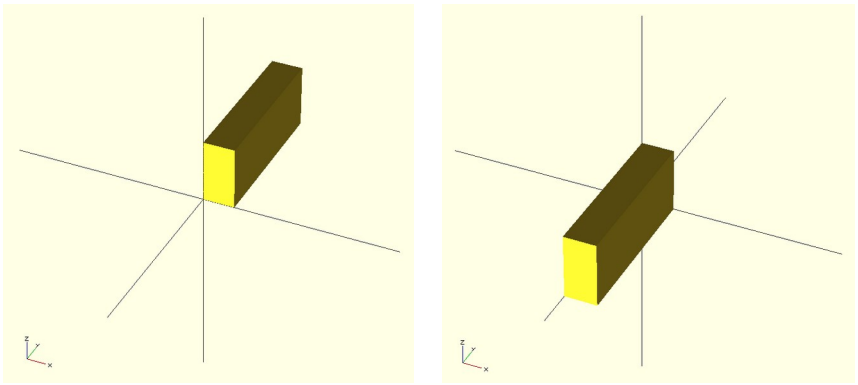
Riflette gli elementi specificati rispetto all'origine degli assi. L'argomento a specchio () è il vettore normale su quell'aereo. Le parentesi graffe possono essere omesse se l'unico elemento da traslare viene scritto di seguito.

Paramentri
[x, y, z]
è una matrice dove va indicato il valore 0 (No) o 1 (Sì)

Esempio di utilizzo

Disegna un parallelepipedo di larghezza 10, profondità 50 e altezza 20 specchiato (ruotato di 180°) sull'asse y

```
mirror([0,1,0]) {
    cube(size=[10,50,20]);
}
```
comando semplificato
```
mirror([0,1,0]) cube(size=[10,50,20]);
```



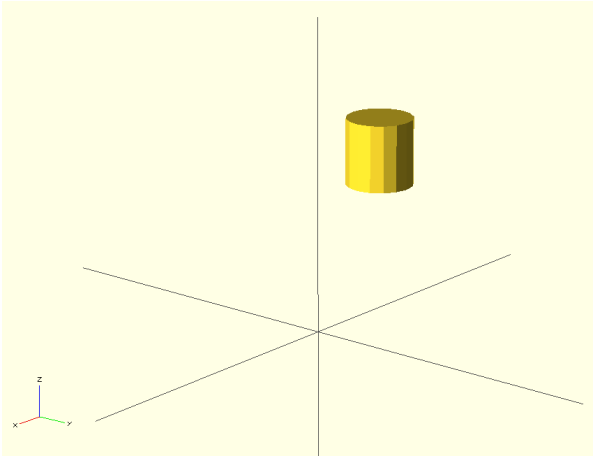
multmatrix

Moltiplica la geometria degli elementi specificati in base ad una matrice di trasformazione da 4x4 elementi.

Sintassi
multmatrix(m = [...]) { ... }

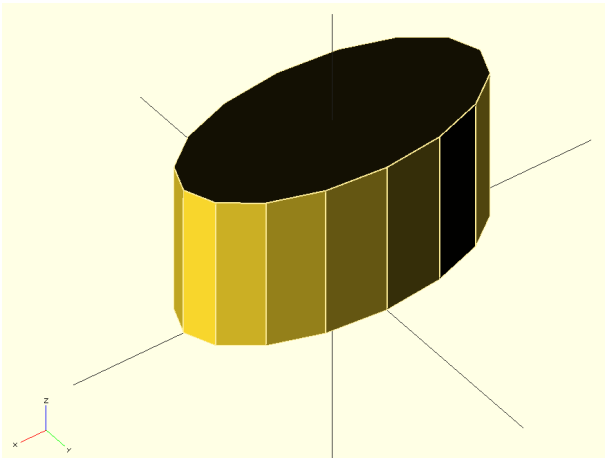
Esempi:
Disegna un cilindro di raggio 5 e altezza 10 traslato di [10, 20, 30]):

```
multmatrix(m = [ [1,0,0,10],[0,1,0,20],[0,0,1,30],[0,0,0,1] ]) cylinder(r=5,h=10);
```



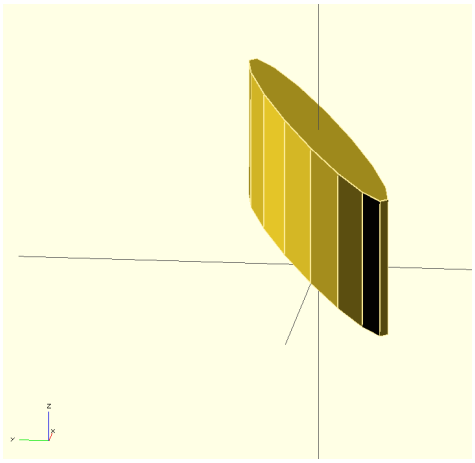
Lo stesso cilindro non traslato ma con profondità (lungo l'asse x) raddoppiata.

```
multmatrix(m = [ [2,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0,1] ]) cylinder(r=5,h=10);
```



Disegna il solito cilindro in modo che appaia affettato a 45°.

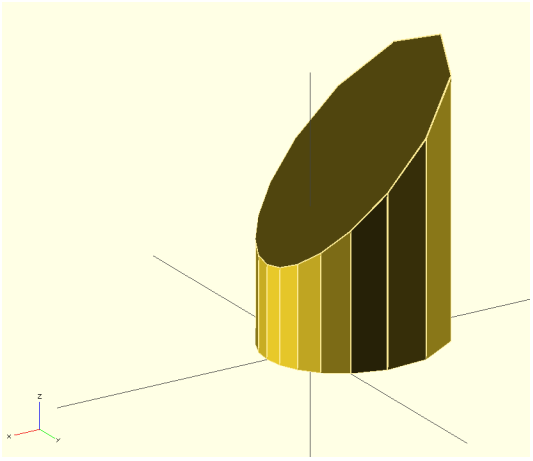
multmatrix(m = [[1,0,0,0],[0,1,0,0],[0,1,1,0],[0,0,0,1]]) cylinder(r=5,h=10);

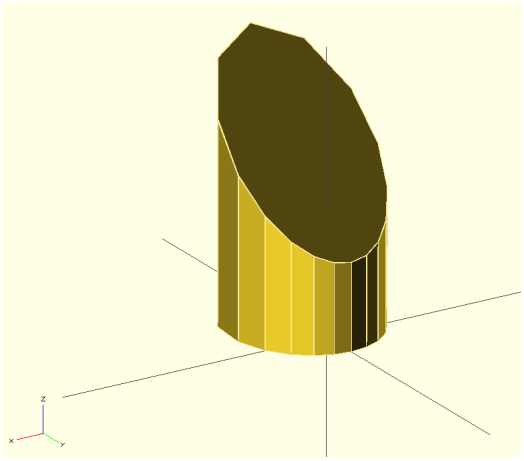


Modificando l'ultima serie di valori si possono ottenere deformazioni più complesse:

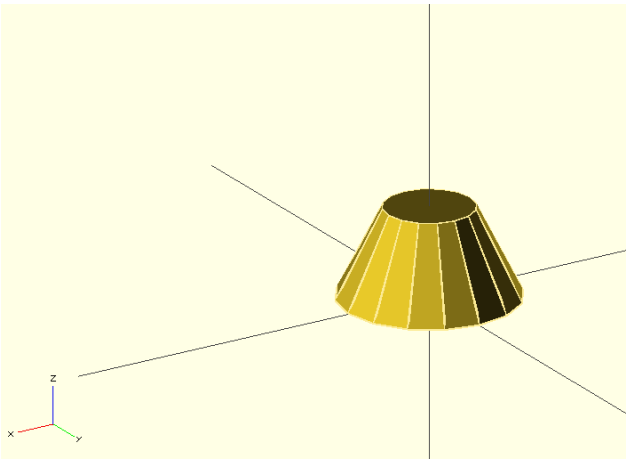
multmatrix(m = [[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0.1,0,1]]) cylinder(r=5,h=10);

multmatrix(m = [[1,0,0,0],[0,1,0,0],[0,0,1,0],[0.1,0,0,1]]) cylinder(r=5,h=10);





multmatrix(m = [[1,0,0,0],[0,1,0,0],[0,0,1,0],[0,0,0.1,1]]) cylinder(r=5,h=10);



color

Colora gli elementi specificati.

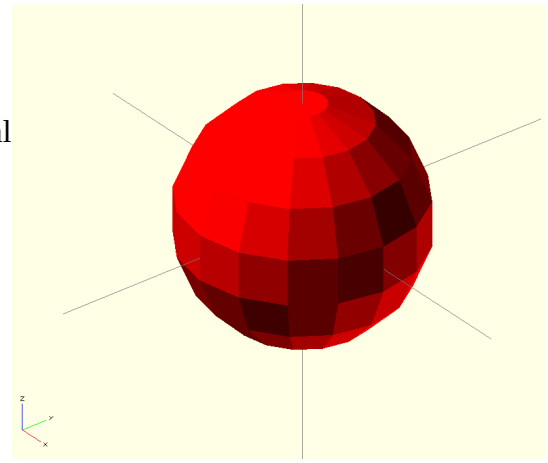
Sintassi
color(“colore”, trasparenza) { ... }

Parametri
Il nome del colore è tratto dalla lista del Wide Web Consortium SVG. Il valore della trasparenza (alpha value) è un numero decimale compreso tra 0 e 1, se non indicato vale 1 (opaco).

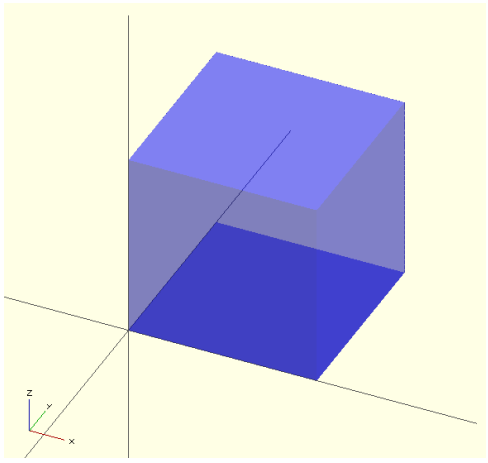
Purples	Blues	Greens	Yellows	Whites
Lavender	Aqua	GreenYellow	Gold	White
Thistle	Cyan	Chartreuse	Yellow	Snow
Plum	LightCyan	LawnGreen	LightYellow	Honeydew
Violet	PaleTurquoise	Lime	LemonChiffon	MintCream
Orchid	Aquamarine	LimeGreen	LightGoldenrodYellow	Azure
Fuchsia	Turquoise	PaleGreen	PapayaWhip	AliceBlue
Magenta	MediumTurquoise	LightGreen	Moccasin	GhostWhite
MediumOrchid	DarkTurquoise	MediumSpringGreen	PeachPuff	WhiteSmoke
MediumPurple	CadetBlue	SpringGreen	PaleGoldenrod	Seashell
BlueViolet	SteelBlue	MediumSeaGreen	Khaki	Beige
DarkViolet	LightSteelBlue	SeaGreen	DarkKhaki	OldLace
DarkOrchid	PowderBlue	ForestGreen	Browns	FloralWhite
DarkMagenta	LightBlue	Green	Cornsilk	Ivory
Purple	SkyBlue	DarkGreen	BlanchedAlmond	AntiqueWhite
Indigo	LightSkyBlue	YellowGreen	Bisque	Linen
DarkSlateBlue	DeepSkyBlue	OliveDrab	NavajoWhite	LavenderBlush
SlateBlue	DodgerBlue	Olive	Wheat	MistyRose
MediumSlateBlue	CornflowerBlue	DarkOliveGreen	BurlyWood	Grays
Pinks	RoyalBlue	MediumAquamarine	Tan	Gainsboro
Pink	Blue	DarkSeaGreen	RosyBrown	LightGrey
LightPink	MediumBlue	LightSeaGreen	SandyBrown	Silver
HotPink	DarkBlue	DarkCyan	Goldenrod	DarkGray
DeepPink	Navy	Teal	DarkGoldenrod	Gray
MediumVioletRed	MidnightBlue	Oranges	Peru	DimGray
PaleVioletRed	Reds	LightSalmon	Chocolate	LightSlateGray
	IndianRed	Coral	SaddleBrown	SlateGray
	LightCoral	Tomato	Sienna	DarkSlateGray
	Salmon	OrangeRed	Brown	Black
	DarkSalmon	DarkOrange	Maroon	
	LightSalmon	Orange		
	Red			
	Crimson			
	FireBrick			
	DarkRed			

Esempi:

Disegna una sfera di colore rosso di raggio 5.
color("red") sphere(5);



Disegna un cubo di lato 5 di colore blu con trasparenza al 50%.
color("Blue",0.5) cube(5);

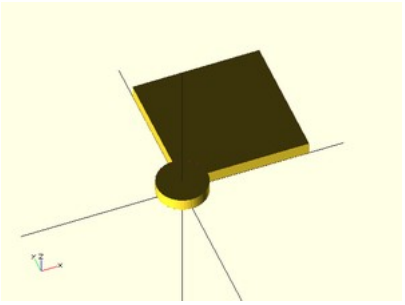


minkowski

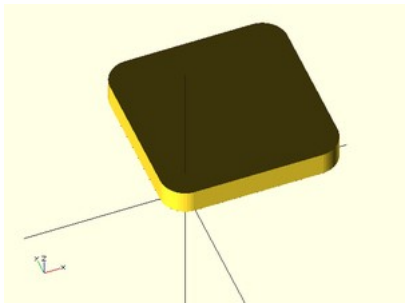
Permette di creare elementi complessi sommando due oggetti semplici.

Esempi:
Creare una forma quadrata con bordi arrotondati disegnando un quadrato e un cerchio.

```
cube([10,10,1]);
cylinder(r=2,h=1);
```

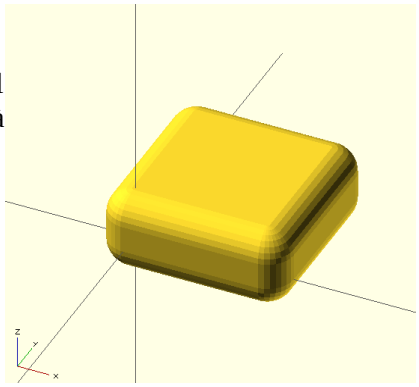


Ecco come la funzione minkowski crea il risultato aspettato:



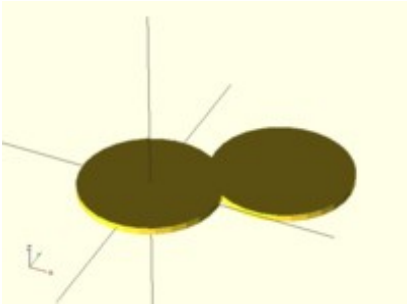
```
minkowski()
{
    cube([10,10,1]);
    cylinder(r=2,h=1);
}
```

Per ottenere bordi arrotondati su tutti gli angoli basterà usare, al posto del cerchio, una sfera (la variabile $fn specifica la risoluzione: più alto sarà questo valore e più tempo verrà impiegato per la generazione del disegno).



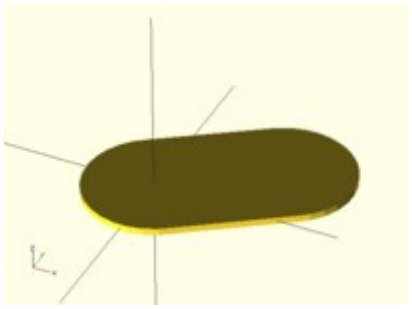
```
$fn=30;
minkowski() {
   cube([10,10,2]);
    sphere(2);
}
```

hull



Produce l'inviluppo convesso (involucro convesso) di due o più oggetti semplici.
Intuitivamente, l'inviluppo convesso di un insieme di punti è la forma che assumerebbe un elastico allargato in modo da contenere tutti i punti e poi lasciato libero di restringersi: un poligono che ha alcuni di quei punti come vertici e li contiene tutti.

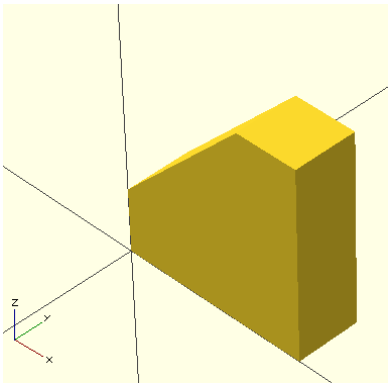
Esempi:
Crea l'inviluppo convesso di due cilindri.



```
hull() {
   cylinder(r=5,h=1);
   translate([15,10,0]) cylinder(r=5,h=1);
}
```

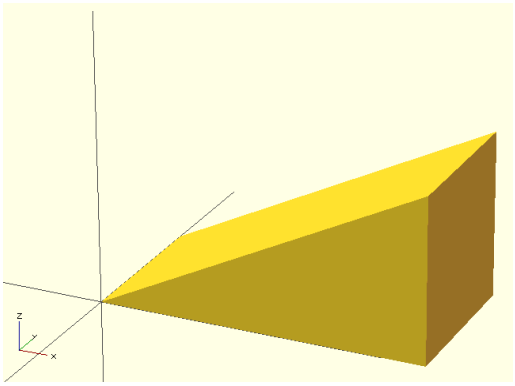
Crea l'inviluppo convesso di due parallelepipedi.

```
hull() {
cube([20,10,10]);
translate([20,0,0]) cube([10,10,30]);
}
```



Disegna un cuneo creando l'inviluppo convesso di due parallelepipedi di altezza minimale (un millesimo): uno che produce la superficie di base e il secondo quella in fondo (che appare in arancione)

```
hull() {
   cube([20,10,0.001]);
   translate([20,0,0]) rotate([0,-90,0]) cube([10,10,0.001]);
}
```

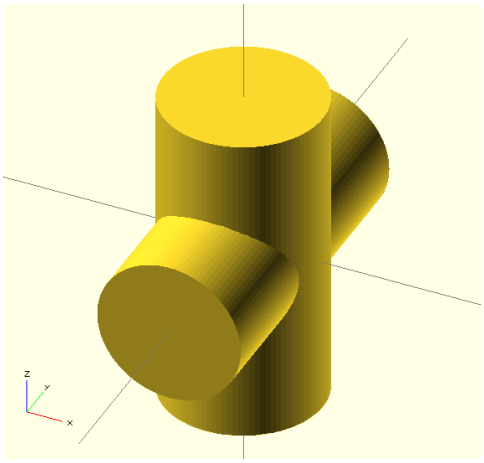


union

Fonde più oggetti. Il risultato dell'unione sarà sarà una forma (senza vuoti interni). Si ottengono così oggetti ideali per la stampa 3D. Il parametro $fn=100 imposta una risoluzione maggiore.

Esempio:
Fonde due cilindri.

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

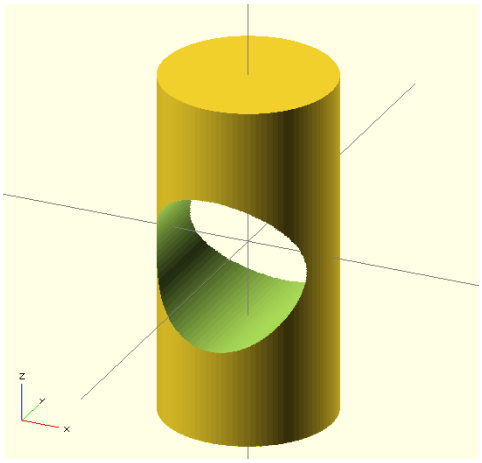


difference

Sottrae il secondo oggetto (e successivi) al primo.

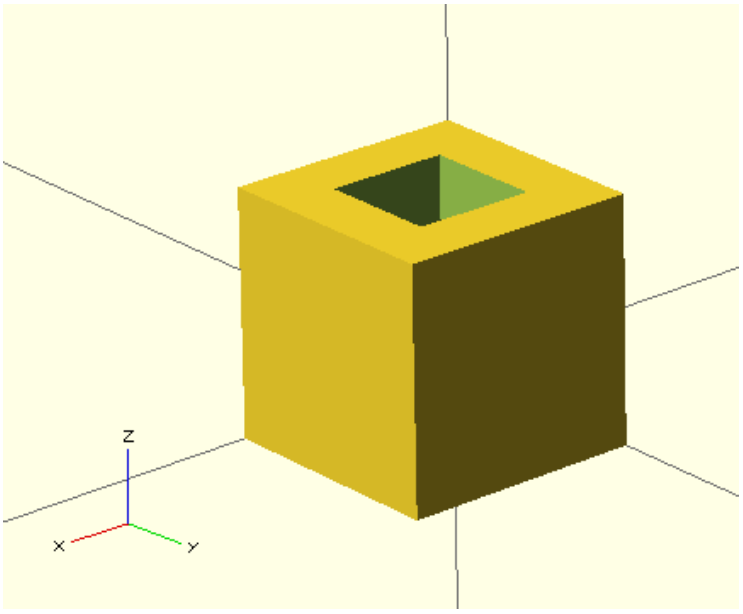
Esempi:
Come nell'esempio precedente solo che il secondo cilindro è sottratto al primo.

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



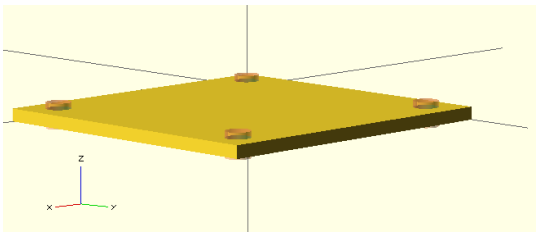
Disegna un cubo di lato 2 mm con un foro quadrato da 1 mm di lato. Il parallelepipedo a base quadrato che produce il foro viene fatto uscire da entrambi i lati di 0,5 mm. Il carattere # inserito davanti al comando permette di visualizzare, in colore arancio semitrasparente, l'oggetto che altrimenti non sarebbe visibile.

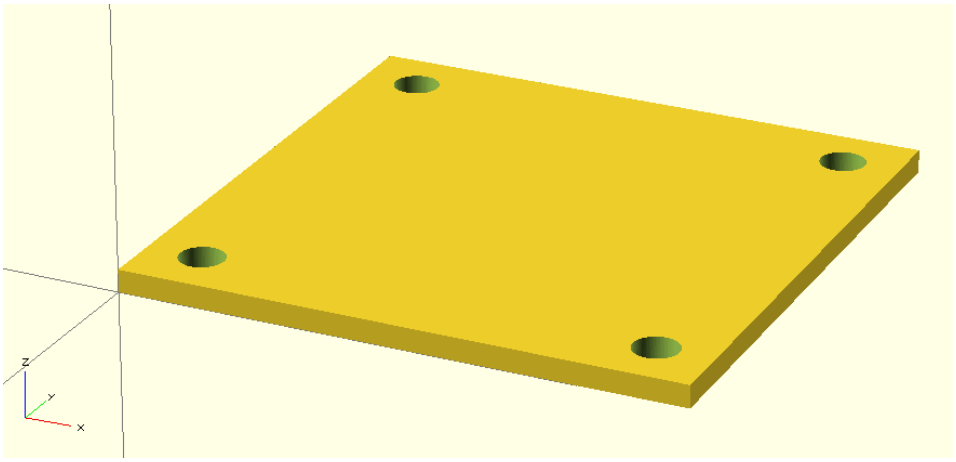
```
difference() {
    cube (size = [2,2,2]);
    translate ([0.5,0.5,-0.5]) cube (size = [1,1,3]);
}
```



Disegna una piastra da 50 per 50 e di spessore 2 con 4 fori da 4 con il centro distante 5 dai bordi.
I fori (cilindri di altezza 4 vengono fatti uscire di 1 per parte dai bordi della piastra)

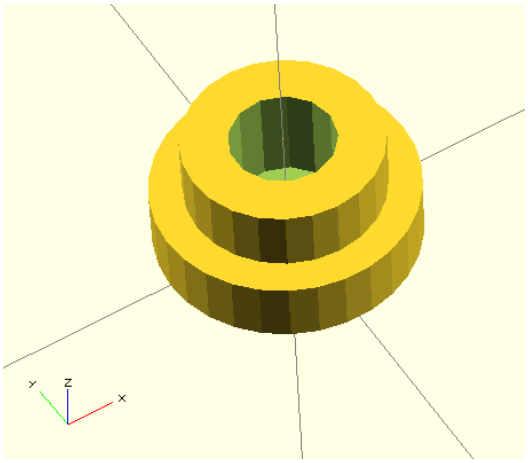
```
difference() {
    cube([50,50,2]);
    translate([5,5,-1]) cylinder(r=2,h=4,$fn=100);
    translate([45,5,-1]) cylinder(r=2,h=4,$fn=100);
    translate([5,45,-1]) cylinder(r=2,h=4,$fn=100);
    translate([45,45,-1]) cylinder(r=2,h=4,$fn=100);
}
```





Disegna un piedino per una barra liscia da 8 mm. Il foro entra per 3 mm anche nel primo cilindro: quello più largo: ecco perché ho usato la funzione union per creare un oggetto unico prima di produrre il foro all'interno. In questo esempio non ho usato il parametro $fn per aumentare la risoluzione del disegno.

```
difference() {
    union() {
        cylinder(r=10,h=5);
        translate([0,0,5]) cylinder(r=7.5,h=5);
    }
    translate([0,0,2]) cylinder(r=4,h=9);
}
```

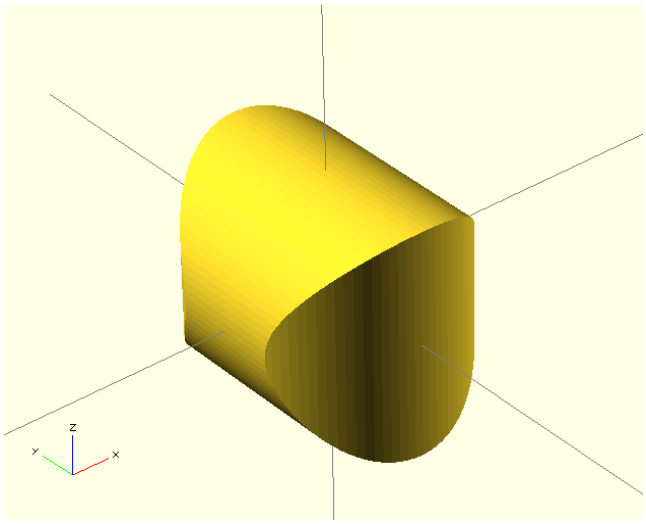


intersection

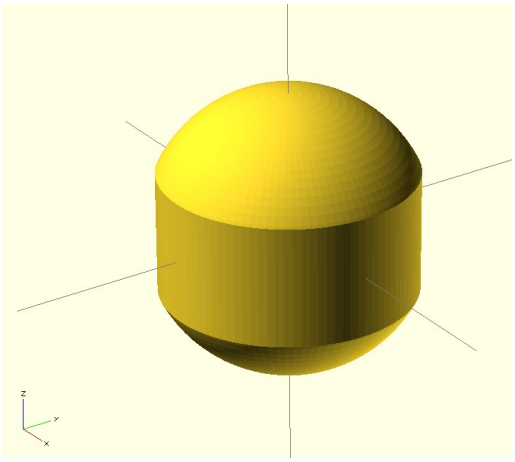
Crea l'intersezione di più oggetti mantenendo tutte le parti che si sovrappongono.

Esempi:
Crea l'intersezione tra due cilindri, il primo di diametro 2 e il secondo 1,8) disposti a 90° uno dall'altro. L'altezza è di 4 ma funziona con qualsiasi valore superiore a 2.

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



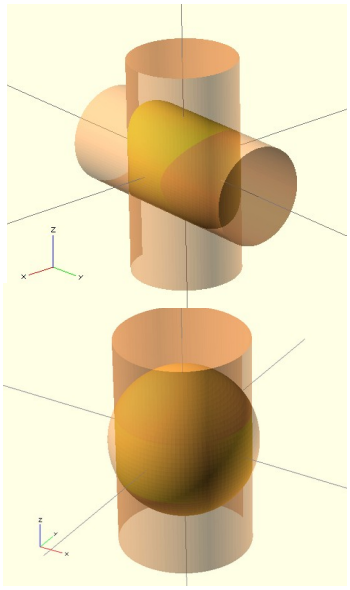
Crea l'intersezione tra una sfera di raggio 1 e un cilindro di raggio 0,9 e altezza 3.



```
intersection() {
   sphere(r=1,center=true,$fn=100);
   cylinder(r=0.9,h=3,center=true,$fn=100);
}
```

Caratteri di controllo

I cosiddetti caratteri di controllo sono molto utili, durante la scrittura di un programma, per evidenziare alcuni tra gli oggetti disegnati e, in particolare quando si usano i comandi union, difference e intersection per vedere quelli altrimenti invisibili.



Modificatore di controllo
L'oggetto sarà sarà disegnato in rosa trasparente

I due esempi precedenti dove si vedono completamente i due oggetti.

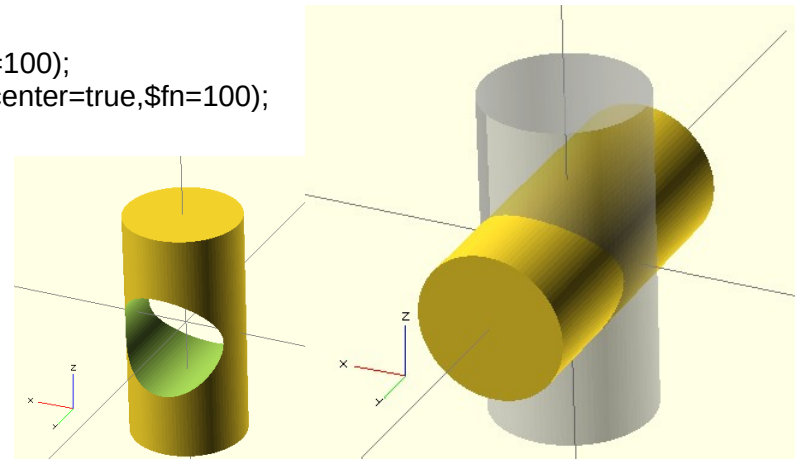
```
intersection() {
   #cylinder(r=1,h=4,center=true,$fn=100);
   #rotate([90,0,0]) cylinder(r=0.9,h=4,center=true,$fn=100);
}

intersection() {
   #sphere(r=1,center=true,$fn=100);
   #cylinder(r=0.9,h=3,center=true,$fn=100);
}
```

% Modificatore di trasparenza

Mostra gli oggetti in grigio chiaro trasparente e le trasformazioni applicate, ma non si vedrà il risultato.

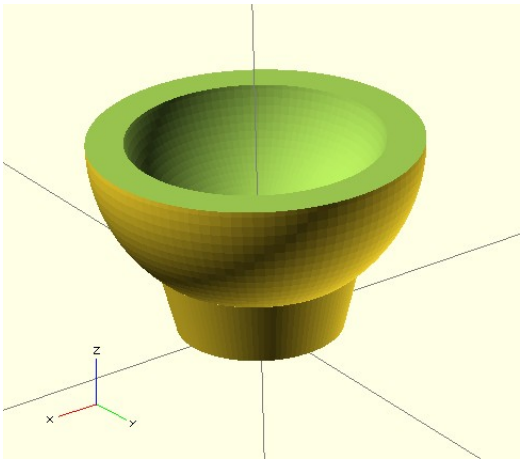
Ecco riproposto il primo esempio svolto con l'uso del comando difference dove in trasparenza si vede il cilindro che produce il foro.

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

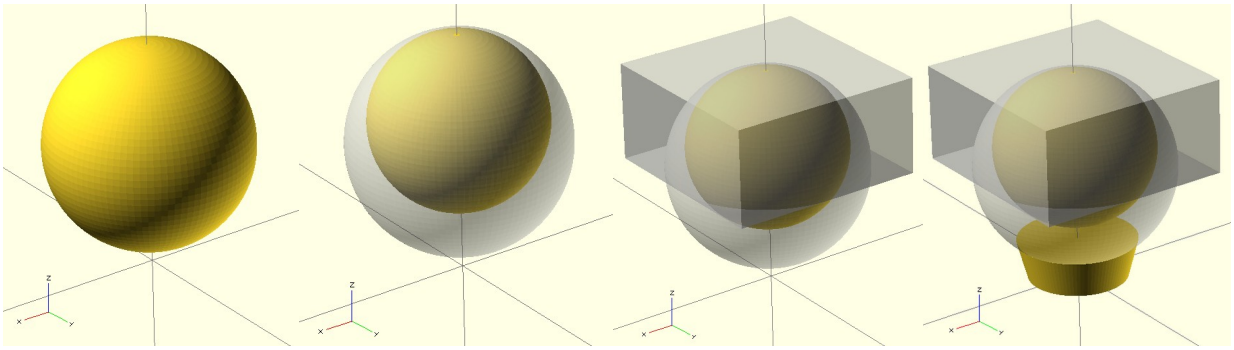


Esempi

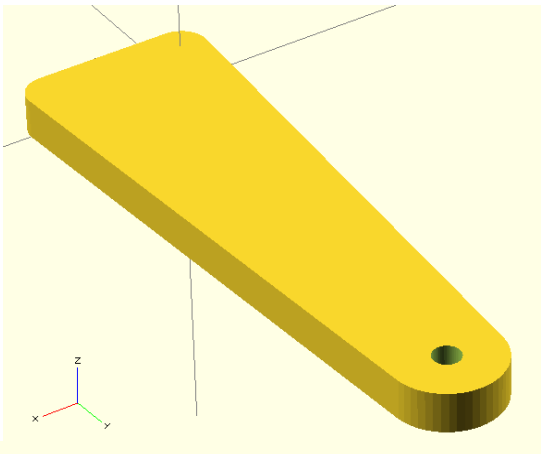
Disegna una scodella

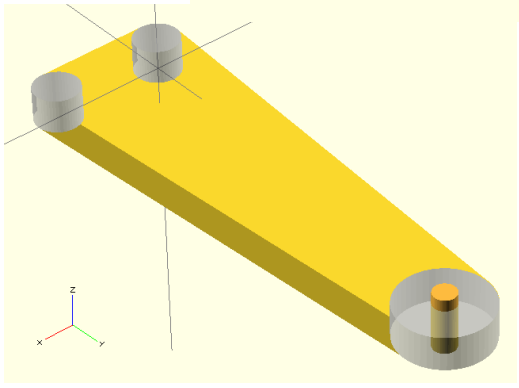


```
// Definisce la risoluzione per tutto il disegno
$fn=100;

//Sposta il punto di disegno
translate ([0,0,12]) {
        difference() {
    //Disegna la sfera esterna
    sphere(10);
    //Crea l'interno disegnando un'altra sfera più piccola
    translate([0,0,2]) sphere(8);
    //Taglia la sfera disegnando un parallelepipedo
    translate([0,0,5]) cube(size=[20,20,10],center=true);
        }
}
// Disegna il tronco di cono come base
translate([0,0,2])cylinder (h=4,r1=5,r2=6,center=true);
```



Disegna un semplice portachiavi





```
$fn = 50;
difference () {
   hull () {
      //Crea l'inviluppo di tre cilindri
      translate([0,0,0]) cylinder(r=2,h=3);
      translate([10,0,0]) cylinder(r=2,h=3);
      translate([5,40,0]) cylinder(r=4,h=3);
   }
   //Produce il foro
   translate([5,38,-1]) cylinder(r=1,h=5);
}
```

Commenti

OpenSCAD utilizza un linguaggio di programmazione per creare i modelli che vengono poi visualizzati sullo schermo. I commenti servono per lasciare note all'interno del codice (come aiuto per se stesso o per chi vuole imparare OpenSCAD o migliorare il programma), che descrive come funziona il codice e/o ciò che fa. I commenti non vengono valutati dal compilatore.

OpenSCAD per i commenti usa lo stile C++

Su una riga:
```
//Disegna una sfera di raggio 10
```

Sulla stessa riga del codice:
```
sphere(10); //Disegna una sfera di raggio 10
```

Su più righe:
```
/*
   Disegna una sfera
   di raggio 10
*/
```

Variabili

Le variabili in OpenSCAD sono semplicemente dei nomi a cui viene assegnato un valore numerico. Come se si usasse un cassetto contrassegnato dal nome della variabile con all'interno un numero. Usando il nome della variabile si ottiene il valore.

Esempi (i commenti possono essere tralasciati):

```
raggio=10; //Definisce la variabile raggio alla quale viene assegnato il valore 10
sphere(raggio); //Disegna una sfera di raggio uguale al contenuto della variabile raggio

altezza=25; //Definisce la variabile altezza alla quale viene assegnato il valore 25
//Disegna un parallelepipedo di larghezza 10, profondità 20
//e altezza 25 (Il valore della variabile altezza)
cube([10,20,altezza],center=true);
//Si sposta di 10 sull'asse y e disegna un cilindro di raggio 5
//e altezza 25 (Il valore della variabile altezza)
translate([0,10,0]) cylinder(r=5,h=altezza,center=true);
```

Vettori

Le variabili possono essere raggruppate in vettori all'interno di parentesi quadre. I vettori sono utili quando si vogliono memorizzare coordinate (X,Y,Z) o dimensioni (larghezza, profondità e altezza).

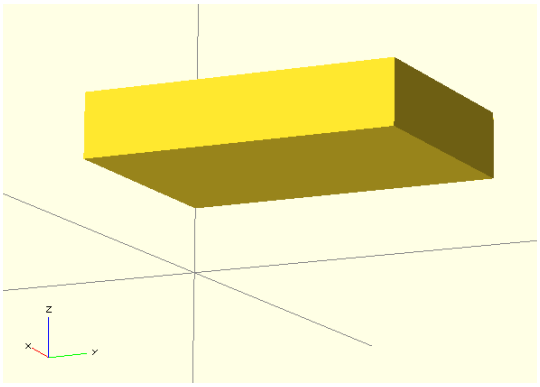
Esempio:

```
dimensioni=[60,85,20]; //Definisce un vettore dimensioni contenente i valori [60,85,20]
cube(dimensioni); // Disegna un parallelepipedo di dimensioni uguali ai contenuti del vettore vettori
```

Si può usare uno dei tre valori contenuti in un vettore usando vettore[numero] (dove numero corrisponde a 0 per estrarre il primo, 1 per il secondo e 2 per il terzo).

Esempio

```
dimensioni=[64,89,18]; //Definisce un vettore dimensioni contenente i valori [64,89,18]
//Si sposta alle coordinate [0,0,18], dove 18 è il terzo valore del vettore dimensioni
//e disegna un parallelepipedo di dimensioni [64,89,18] uguali ai contenuti del vettore dimensioni
translate([0,0,dimensioni[2]]) cube(dimensioni);
```



Stringhe

Devono essere racchiuse fra doppi apici (“). Quando si vuole scrivere le doppie virgolette o una barra rovescia basta anteporre una barra rovescia (\")e (\\) rispettivamente).
Altri caratteri speciali sono: tabulazione (\t), ritorno a capo (\r) e nuova riga (\n).

Esempio:

```
echo(" La veloce volpe marrone \tsalta \"sopra\" il lento cane.\rLa veloce volpe marrone. \nIl \\lento\\ cane.");
```

Stringa visualizzata:

```
ECHO: "La veloce volpe marrone    salta "sopra" il lento cane.
La veloce volpe marrone.
Il \lento\ cane."
```

Strutture di controllo

For (loop)

Scorre i valori all'interno di in un vettore o di un intervallo.

Sintassi:
for (nome_variable=intervallo o vettore) {
.....
}

Parametri
nome_variabile
nome della variabile da usare nel ciclo for.

intervallo:
[minimo:massimo] Sono due numeri che identificano gli estremi dell'intervallo (es. [0:5]).

Vettore:
Un elenco di numeri tra parentesi quadre (es. [1,2,10,40.5]).

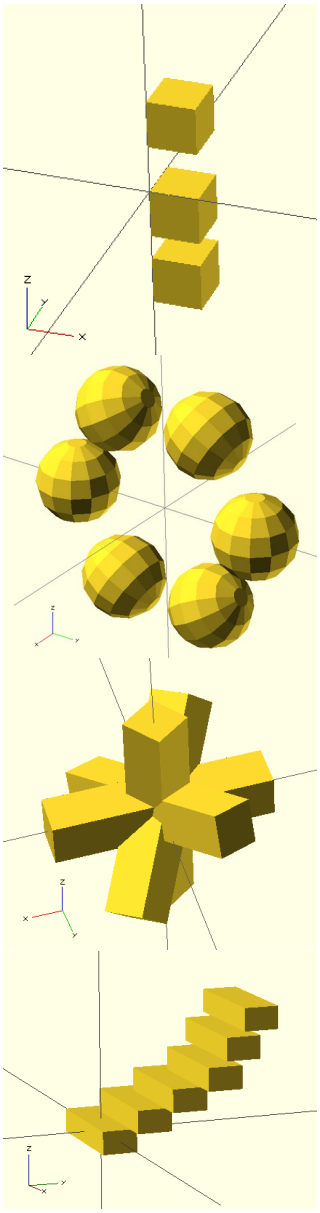
Esempi:

```
for (z=[-1,1,-2.5]) {
    translate([0, 0, z])
        cube(size=1,center=false);
}
```

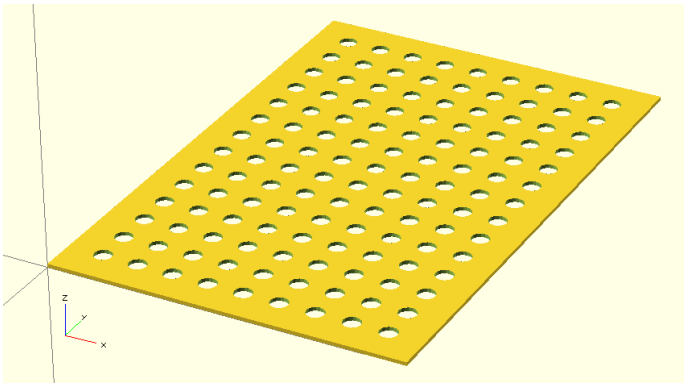
```
for(i=[0:5]) {
rotate([i*360/6,0,0])
translate( [0,10,0] )
sphere(r=4);
}
```

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

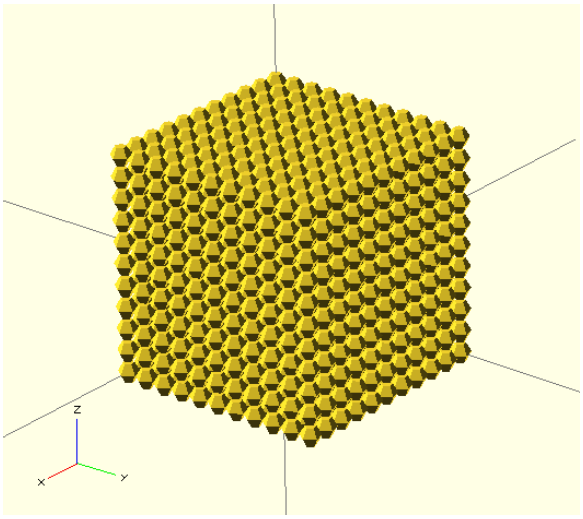
```
for(i=[[0,0,0],
    [10,12,10],
    [20,24,20],
    [30,36,30],
    [20,48,40],
    [10,60,50]])
{
    translate(i)
    cube([50,15,10],center=true);
}
```



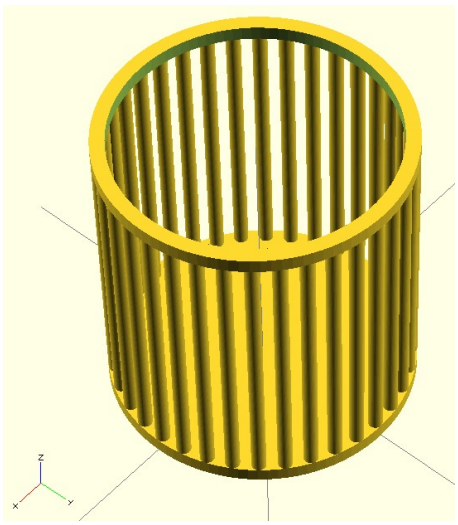
```
difference() {
    cube([200,300,2]);
        for (x=[1:9]) {
            for (y=[1:14]) {
                translate([x*20,y*20,-1])
                    cylinder(r=5,h=4);
            }
    }
}
```



```
for(x=[0:10]) {
    for(y=[0:10]) {
        for(z=[0:10]) {
            translate([x,y,z]) sphere(0.5);
        }
    }
}
```

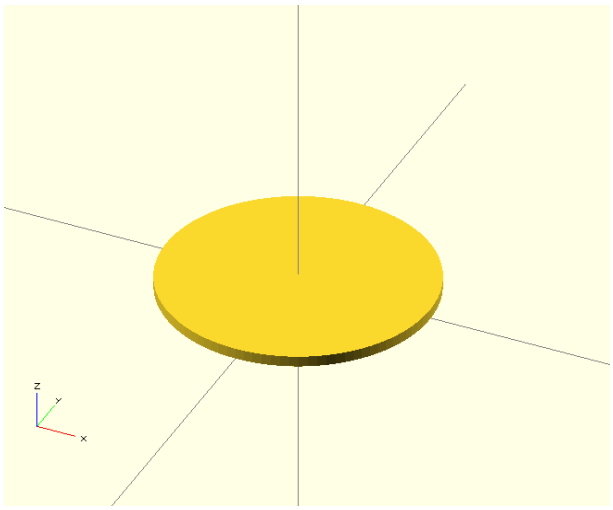


```
$fn=100;
cylinder(r=52.5,h=5);
for(i=[0:35]) {
    rotate([0,0,i*10])
    translate([0,50,5])
        cylinder(r=2.5,h=100);
}
difference() {
        translate([0,0,105]) cylinder(r=52.5,h=5);
        translate([0,0,104]) cylinder(r=47.5,h=7);
}
```

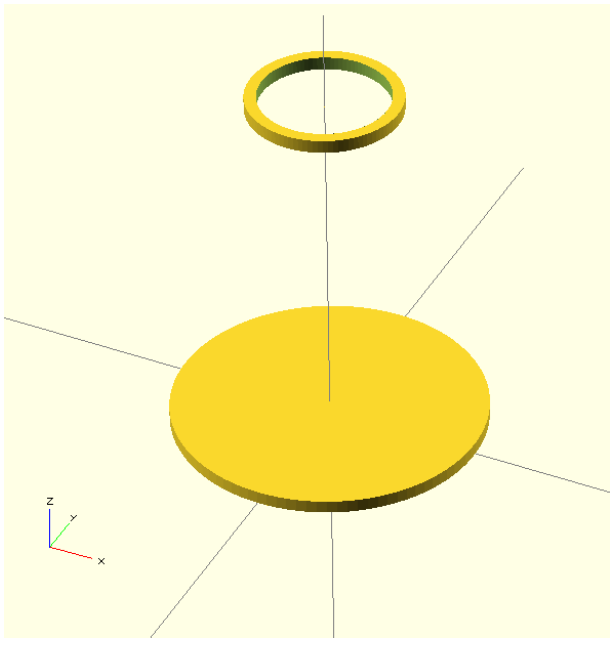


Disegnamo un fac-simile – molto fac e poco simile – **della Torre di Tesla**

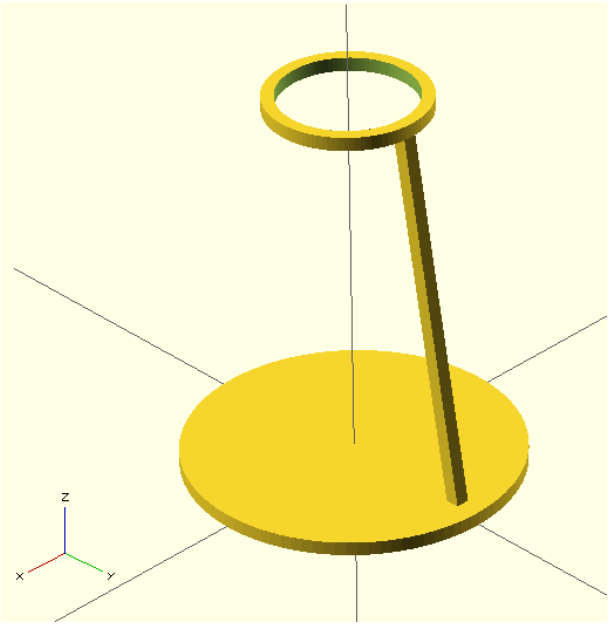
```
$fn=100;
cylinder(r=60,h=5); //base cilindrica di raggio 60 e altezza 5 mm
```



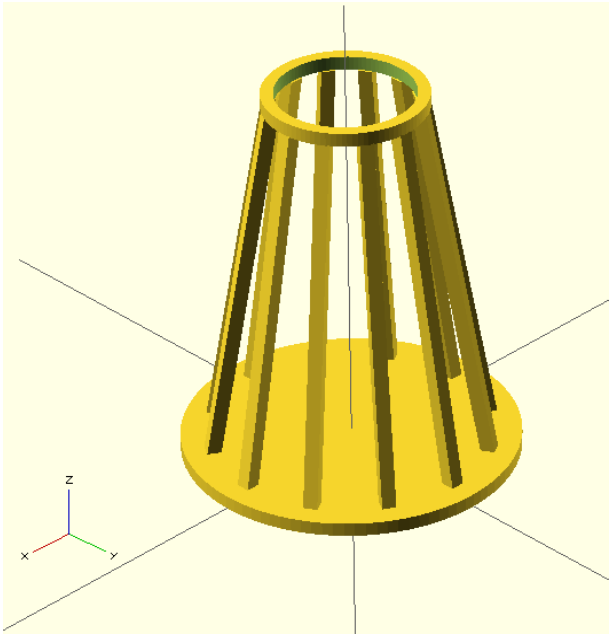
```
//anello in alto di raggio 29 e altezza 5 con foro di raggio 24,5 mm
difference() {
	translate([0,0,138]) cylinder(r=29,h=5); //cilindro pieno
	translate([0,0,137]) cylinder(r=24.5,h=7); //cilindro che produce il foro
}
```



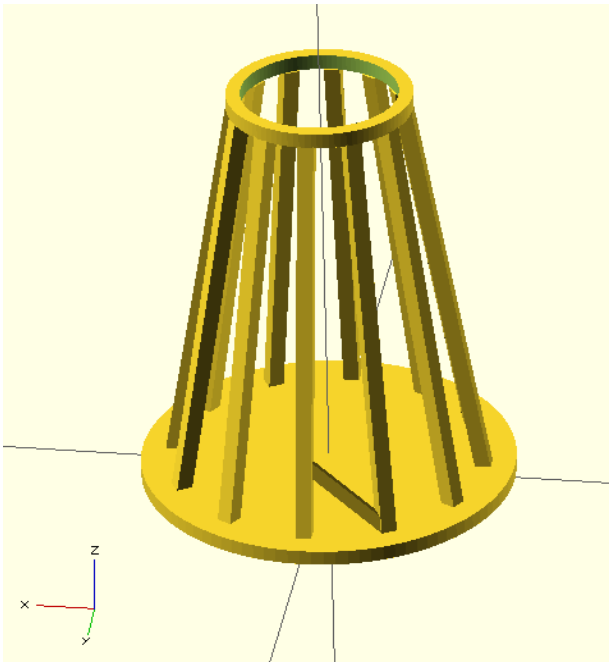
```
//prima barra quadra esterna inclinata verso l'interno di 10°
translate([-2.5,47.5,4]) rotate([10,0,0]) cube([5,5,136]);
```




```
//12 barre quadre esterne a distanza di 30° (i*30) e inclinate verso l'interno di 10°
for(i=[0:11]) {
	rotate([0,0,i*30]) {
		translate([-2.5,47.5,4]) rotate([10,0,0]) cube([5,5,136]);
	}
}
```

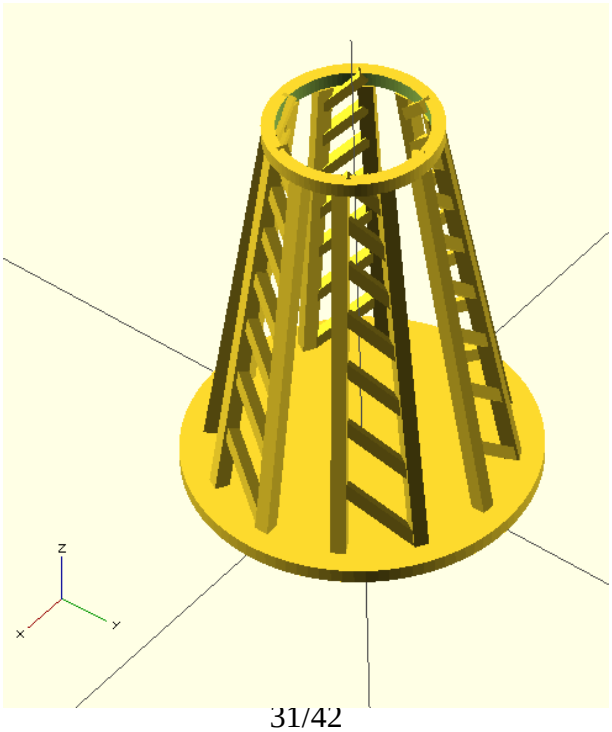


//traversa a 45° inclinata di 10° verso l'interno e ruotata di 15 gradi perché il punto centrale è spostato di 15° rispetto alle barre verticali.
rotate([10,0,15]) translate([-0.75,48.5,9]) rotate([0,45,0]) cube([4,4,34],center=true);

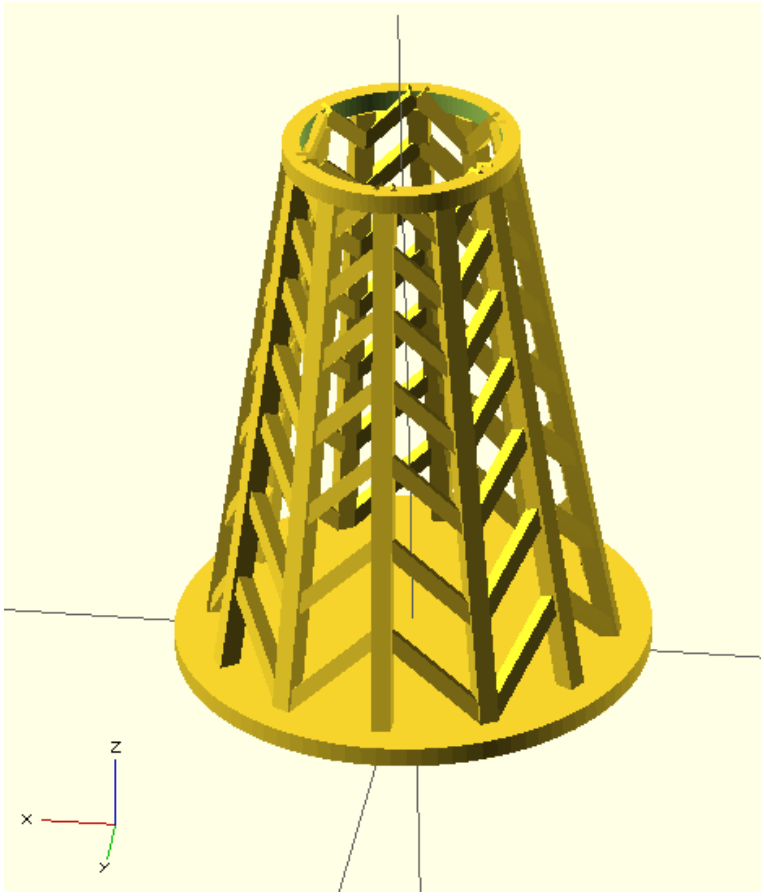


//traverse (a 45°) 6 in orizzontale a 60° per 7 in verticale a distanze proporzionali precalcolate e contenute nella matrice (x)

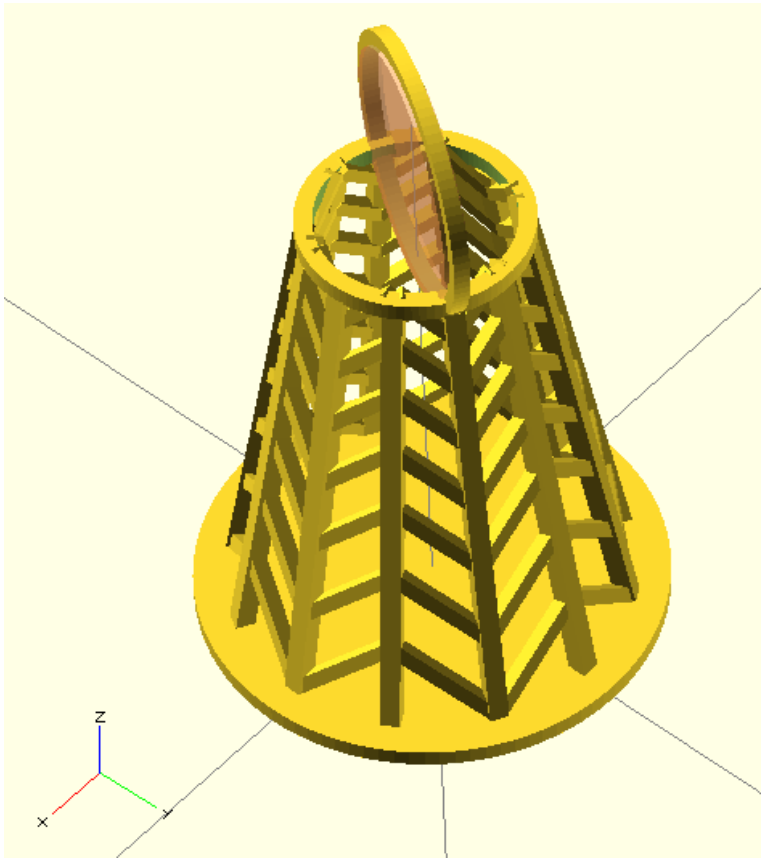
```
x=[0,25,48,68,87,104,120];
for (j=[0:6]) {
	//traverse (a 45°)
	for(i=[0:5]) {
		rotate([10,0,i*60+15]) {
			translate([-0.75,48.5,9+x[j]]) rotate([0,45,0]) cube([4,4,34-
j*2.5],center=true);
		}
	}
}
```



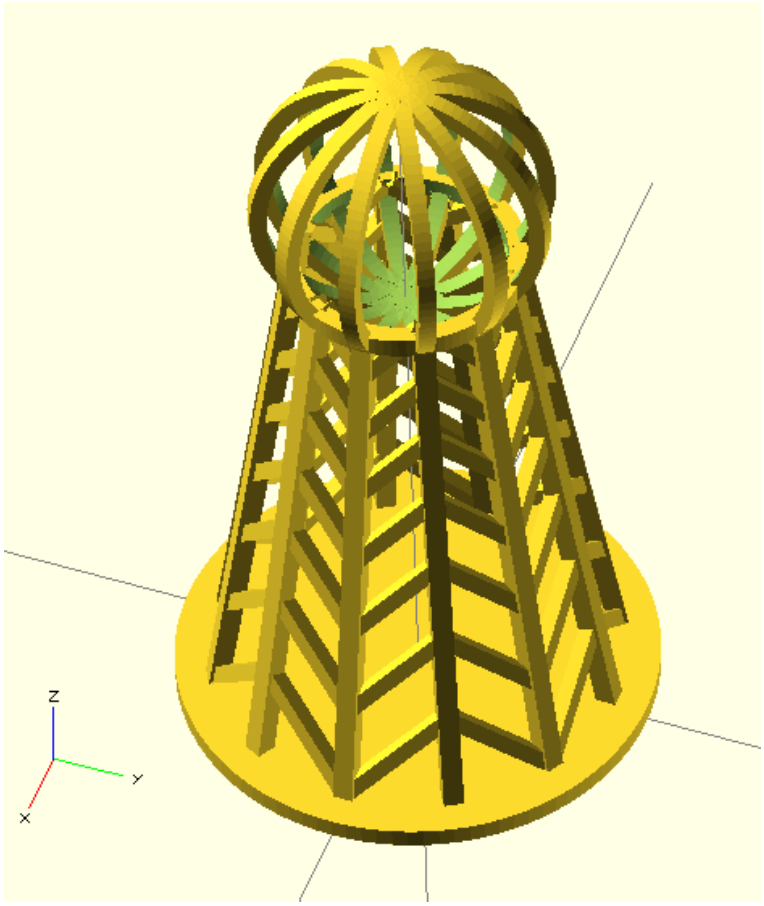
```
//aggiungiamo anche le traverse a -45°
x=[0,25,48,68,87,104,120];
for (j=[0:6]) {
	//traverse (a 45°)
	for(i=[0:5]) {
		rotate([10,0,i*60+15]) {
			translate([-0.75,48.5,9+x[j]]) rotate([0,45,0]) cube([4,4,34-
j*2.5],center=true);
		}
	}
	//traverse (a -45°)
	for(i=[0:5]) {
		rotate([10,0,i*60-15]) {
			translate([0.75,48.5,9+x[j]]) rotate([0,-45,0]) cube([4,4,34-
j*2.5],center=true);
		}
	}
}
```



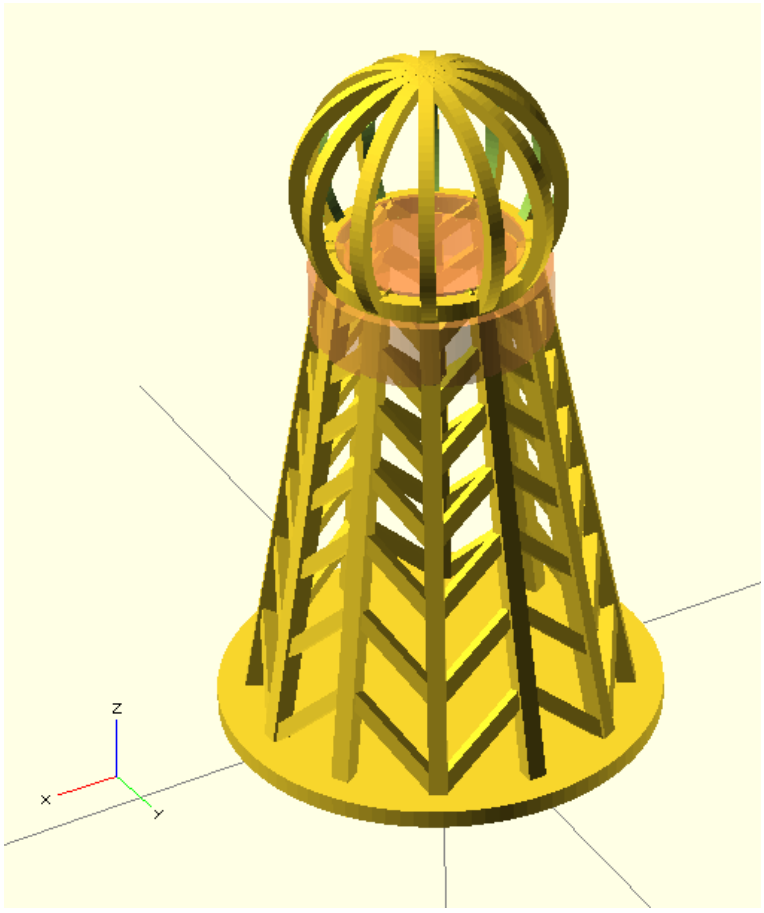
```
//anello che produce la cupola
difference() {
	//disegna il cilindro pieni
	rotate([30,90,0]) translate([-160,0,-2]) cylinder(r=36,h=4);
	//cilindrio che crea il foro producendo l'anello
	#rotate([30,90,0]) translate([-160,0,-3]) cylinder(r=32,h=6);
}
```



```
//12 anelli che producono la cupola
difference() {
	//disegna i cilindri pieni
	union() {
		for(i=[0:11]) {
			rotate([i*30,90,0]) translate([-160,0,-2]) cylinder(r=36,h=4);
		}
	}
	//cilindri che creando il foro producono gli anelli
	union() {
		for(i=[0:11]) {
			rotate([i*30,90,0]) translate([-160,0,-3]) cylinder(r=32,h=6);
		}
	}
}
```



```
//togliamo la parte in basso della cupola.
//12 anelli che producono la cupola
difference() {
	//disegna i cilindri pieni
	union() {
		for(i=[0:11]) {
			rotate([i*30,90,0]) translate([-160,0,-2]) cylinder(r=36,h=4);
		}
	}
	//cilindri che creando il foro producono gli anelli
	union() {
		for(i=[0:11]) {
			rotate([i*30,90,0]) translate([-160,0,-3]) cylinder(r=32,h=6);
		}
	}
	//cilindri che tagliano la parte di cupola che altrimenti uscirebbe sotto l'anello
	#translate([0,0,120]) cylinder(r=32,h=18);
	#translate([0,0,137]) cylinder(r=24.5,h=7);
}
```



Il programma completo

```
$fn=100;
cylinder(r=60,h=5); //base cilindrica
//anello in alto
difference() {
	translate([0,0,138]) cylinder(r=29,h=5); //cilindro pieno
	translate([0,0,137]) cylinder(r=24.5,h=7); //cilindro che produce il foro
}
//12 barre quadre esterne a distanza di 30° (i*30) e inclinate verso l'interno di 10°
for(i=[0:11]) {
	rotate([0,0,i*30]) {
		translate([-2.5,47.5,4]) rotate([10,0,0]) cube([5,5,136]);
	}
}
//traverse (a 45°) 6 in orizzontale a 60° per 7 in verticale a distanze proporzionali precalcolate e contenute nella matrice (x)
x=[0,25,48,68,87,104,120];
for (j=[0:6]) {
	//traverse (a 45°)
	for(i=[0:5]) {
		rotate([10,0,i*60+15]) {
			translate([-0.75,48.5,9+x[j]]) rotate([0,45,0]) cube([4,4,34-j*2.5],center=true);
		}
	}
	//traverse (a -45°)
	for(i=[0:5]) {
		rotate([10,0,i*60-15]) {
			translate([0.75,48.5,9+x[j]]) rotate([0,-45,0]) cube([4,4,34-j*2.5],center=true);
		}
	}
}
//12 anelli che producono la cupola
difference() {
	//disegna i cilindri pieni
	union() {
		for(i=[0:11]) {
			rotate([i*30,90,0]) translate([-160,0,-2]) cylinder(r=36,h=4);
		}
	}
	//cilindri che creando il foro producono gli anelli
	union() {
		for(i=[0:11]) {
			rotate([i*30,90,0]) translate([-160,0,-3]) cylinder(r=32,h=6);
		}
	}
	//cilindri che tagliano la parte di cupola che altrimenti uscirebbe sotto l'anello
	translate([0,0,120]) cylinder(r=32,h=18);
	translate([0,0,137]) cylinder(r=24.5,h=7);
}
```

Comandi di estrusione di oggetti 2D

linear_extrude

L'estrusione lineare è un'operazione che prende un oggetto 2D come input e genera un oggetto 3D come risultato. In OpenSCAD l'estrusione viene sempre eseguita sulla proiezione (ombra) dell'oggetto 2d sul piano xy e lungo l'asse Z; quindi, se ruoti o applichi altre trasformazioni all'oggetto 2D prima dell'estrusione, quella che verrà estrusa è la sua ombra.
Sebbene l'estrusione sia lineare lungo l'asse Z, è disponibile un parametro di torsione che fa ruotare l'oggetto attorno all'asse Z durante l'estrusione verso l'alto. Questo può essere utilizzato per ruotare l'oggetto durante l'estrusione per creare una torsione o produrre un'estrusione elicoidale .

Sintassi
```
linear_extrude(height = 5, center = true, convexity = 10, twist = -fanrot, slices = 20, scale = 1.0, $fn = 16) {...}
```

Parametri

height
Numero decimale positivo che indica l'altezza dell'estrusione.

center
Opzionale. Se true, l'estrusione verrà centrata sull'asse Z.

convexity
Come già specificato è utile solo per la visualizzazione in anteprima dell'oggetto. Normalmente 10 è il valore più indicato.

twist
È opzionale. Specifica l'angolo di torsione dell'estrusione in gradi.

slices
È opzionale. $fn indica la risoluzione sull'asse Z. Per capirne l'utilizzo vedere l'esempio specifico seguente.

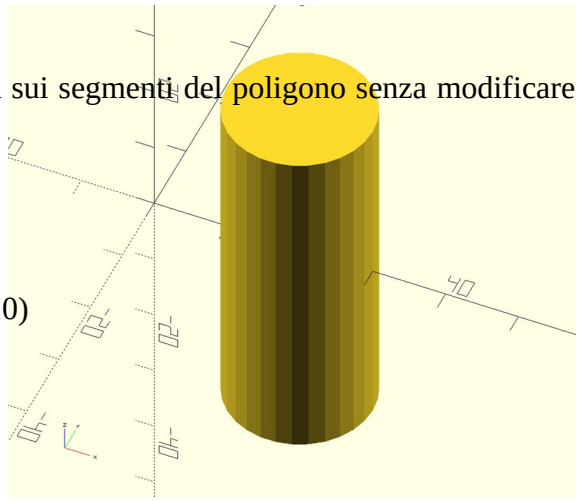
scale
È opzionale. Numero decimale. Se indicata ridimensiona la forma 2D di questo valore linearmente durante estrusione in modo che 'oggetto possa essere espanso o contratto.

$fn
È opzionale e indica la risoluzione, quindi anche la qualità dell'estrusione.

segments
È opzionale. Simile a slices ma con l'aggiunta di punti sui segmenti del poligono senza modificare la forma del poligono.

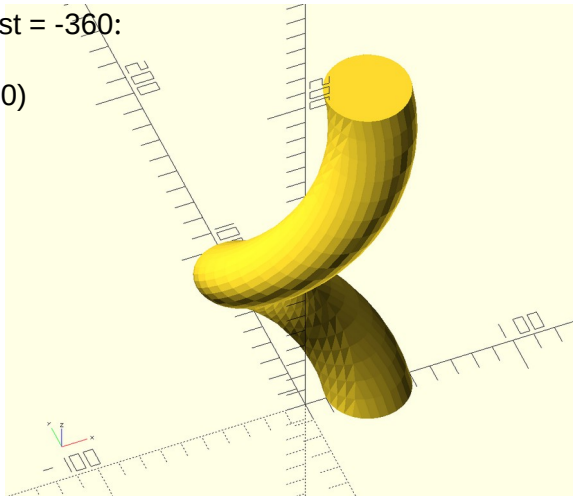
Esempio con l'opzione center = true:

```
linear_extrude(height = 50, center = true, convexity = 10)
        translate([20, 0, 0]) circle(r = 10);
```

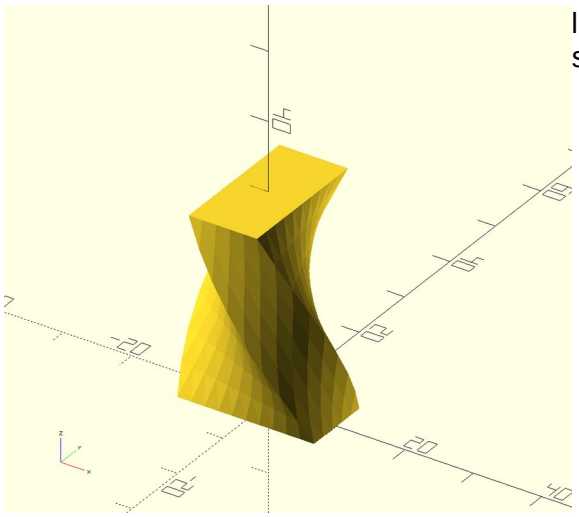


Togliamo l’opzione center e aggiungiamo l’opzione twist = -360:

```
linear_extrude(height = 200, convexity = 10, twist = -360)
        translate([30, 0, 0]) circle(r = 20);
```



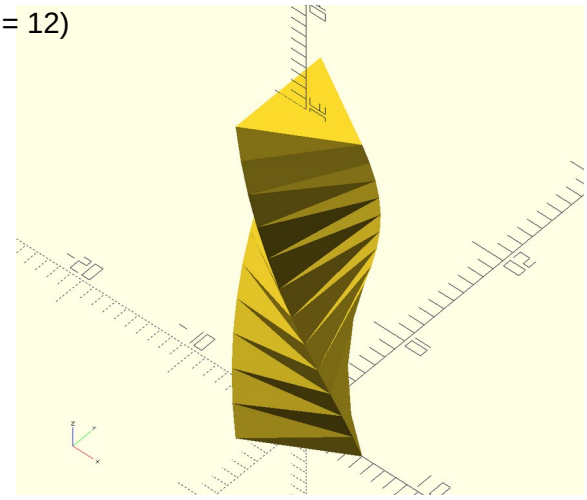
Ora vediamo un altro esempio che utilizza il comando square per estrudere un rettangolo:



```
linear_extrude(height = 30, twist = 90)
square([20, 10], center = true);
```

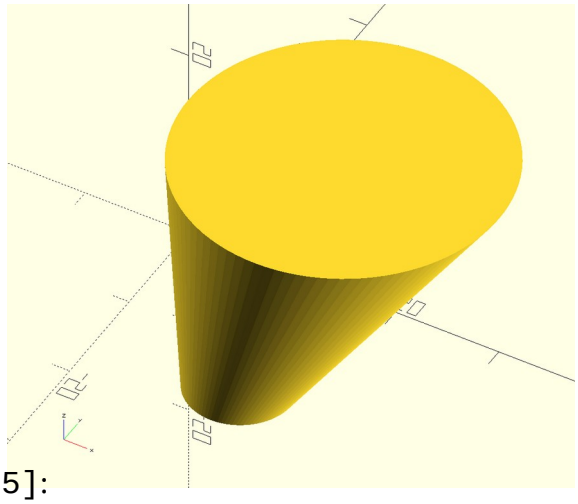
Esempio che mostra la differenza dell’uso dei parametri $fn e slices.
Il parametro slices definisce il numero di punti intermedi lungo l'asse Z dell'estrusione. Se viene omesso il suo valore predefinito aumenta con il valore di twist. Inoltre, il parametro $fn aggiunge lati al poligono estruso. Per vedere bene l’effetto il valore specificato in segment deve essere un multiplo di quelloin $fn. In questo esempio ho usato 3 e 12.

```
linear_extrude(height = 30, twist = 120, $fn = 3, slices = 12)
        circle(r = 5, center = true);
```



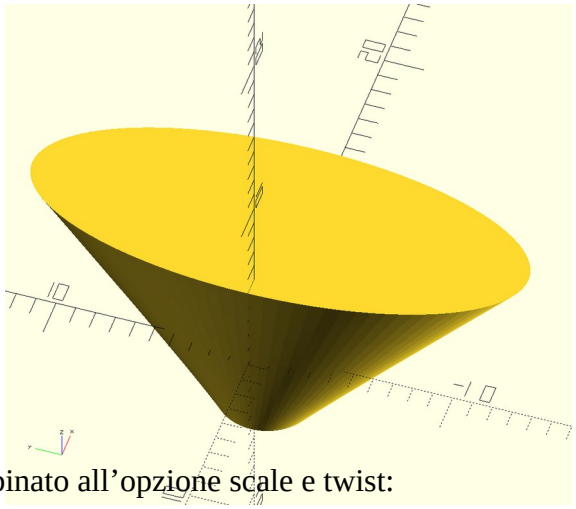
Ora vediamo un esempio che utilizza l'opzione scale. L'opzione scale ridimensiona la forma 2D in base a questo valore rispetto all'altezza dell'estrusione. La scala può essere uno scalare o un vettore:

```
linear_extrude(height = 30, center = true, convexity = 10, scale = 3, $fn = 100)
        translate([5, 0, 0]) circle(r = 5);
```



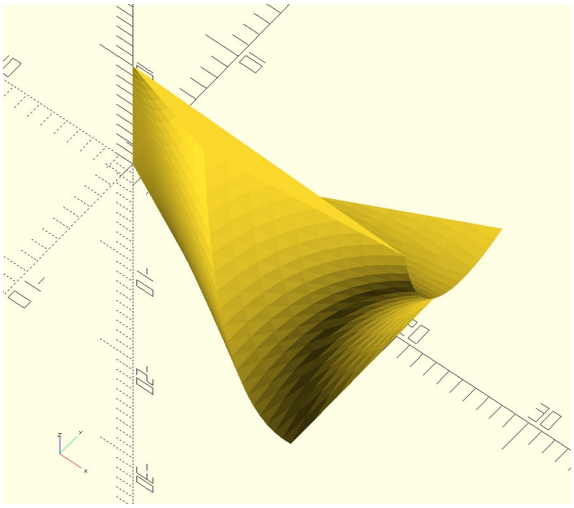
Esempio che scala gli assi X e Y in modo diverso `[2, 5]`:

```
linear_extrude(height = 10, center = true, convexity = 10, scale = [2, 5], $fn=100)
        translate([2, 0, 0])  circle(r = 2.5);
```



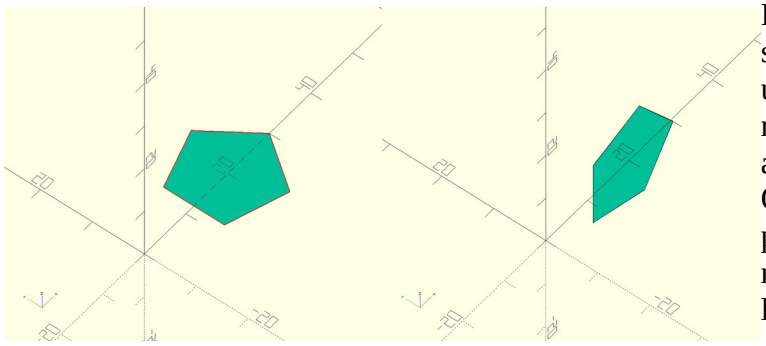
Un ultimo esempio che utilizza il comando polygon abbinato all'opzione scale e twist:

```
linear_extrude(height = 10, scale = [1, 0.1], slices = 20, twist = 45)
        polygon(points=[[0, 0], [20, 10], [20, -10]]);
```

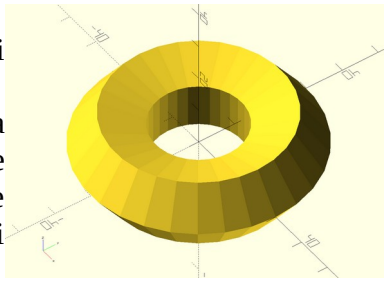


rotate_extrude

L'estrusione di rotazione ruota una forma 2D attorno all'asse z per formare un solido con una simmetria rotazionale.

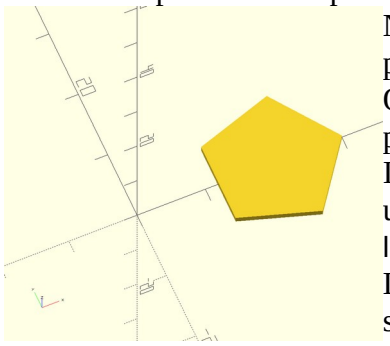


Partendo da un pentagono traslato di 20 sull'asse X, la forma che rotate_extrude userà per creare il toroide sarà quella ruotata +90 gradi (senso antiorario) attorno all'asse X.
Quindi, noi disegniamo la forma sul piano X-Y poi sarà OpenSCAD a ruotarla automaticamente per creare l'estrusione.



Il solido risultante viene posizionato in modo che il suo asse di rotazione si trovi lungo l'asse Z.
Proprio come con linear_extrude, l'estrusione viene sempre eseguita sulla proiezione del poligono 2D sul piano XY. Trasformazioni come rotate, translate, ecc. applicate al poligono 2D prima dell'estrusione modificano la proiezione del poligono 2D sul piano XY e quindi modificano anche l'aspetto dell'oggetto 3D finale.

- Una traslazione in Z del poligono 2D non ha alcun effetto sul risultato (come anche la proiezione non è influenzata).
- Una traslazione in X aumenta il diametro dell'oggetto finale.
- Una traslazione in Y si traduce in uno spostamento dell'oggetto sull'asse Z.
- Una rotazione attorno all'asse X o Y distorce la sezione trasversale dell'oggetto finale, poiché anche la proiezione al piano XY è distorta.



Non preoccuparti, se OpenScad visualizza poligoni 2D con una piccola altezza nella direzione Z, facendo apparire un oggetto 3D.
Quando OpenSCAD esegue l'estrusione verrà utilizzata solo la sua proiezione (vedi disegno in cima alla pagina).
Il comando rotate_extrude non può essere utilizzato per produrre un'elica o filettature a vite ma possono essere fatte solo con linear_extrude usando il parametro twist.
La forma 2D deve trovarsi completamente sulla destra (consigliata) o sul lato sinistro dell'asse Y, cioè tutti i vertici della forma devono avere x >= 0 o x <= 0. Se non vengono rispettate queste regole il programma provoca un errore.

Sintassi:
rotate_extrude(angle = 360, convexity = 10) {...}

Parametri:

angle
(richiede la versione 2019.05) il valore predefinito è 360. Specifica il numero di gradi di estrusione a partire dall'asse X positivo. La direzione di estrusione è antioraria, quindi per estrudere in senso orario si deve indicare un angolo negativo

convexity
Come già specificato è utile solo per la visualizzazione in anteprima dell'oggetto. Normalmente 10 è il valore più indicato.

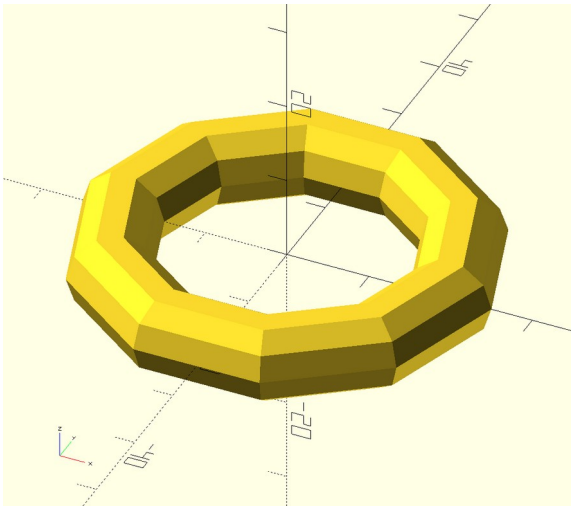
$fa
Opzionale. Angolo minimo (in gradi) di ogni frammento.

$fs
Opzionale. Lunghezza minima di ogni frammento lungo la circonferenza.

$fn
Opzionale. Numero fisso di frammenti a 360 gradi. Un valore di 3 o più rende vano l'uso delle due precedenti.

Esempi:

```
rotate_extrude(convexity = 10, $fn = 10)
        translate([20, 0, 0]) circle(r = 5);
```



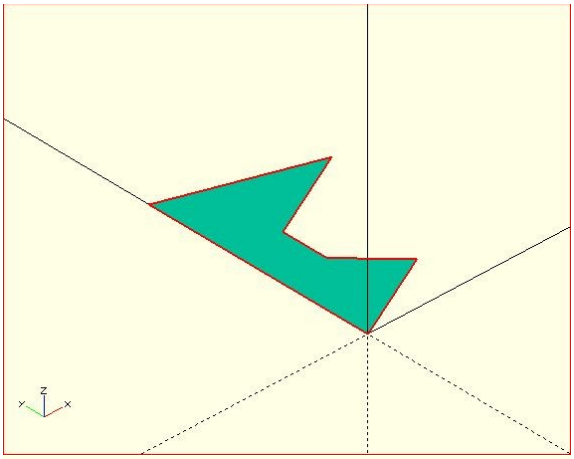
$fa = 10 può essere sostituito con $fa = 36:

```
rotate_extrude(convexity = 10, $fa = 36)
        translate([20, 0, 0]) circle(r = 5);
```

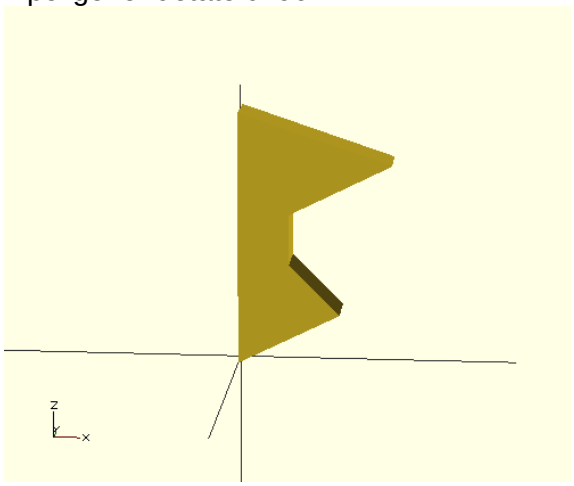
Estrusione di un poligono
L'estrusione può essere eseguita anche su poligoni con punti definiti dall'utente.

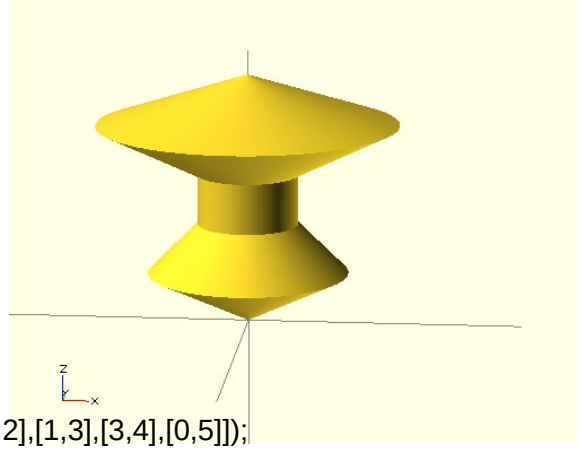
Ecco un semplice poligono e la sua estrusione rotazionale a 200 passi. (È mostrato anche ruotato di 90 gradi per capire meglio come avviene l'estrusione).

Il poligono origine:
`polygon(points=[[0, 0],[2, 1],[1, 2],[1, 3],[3, 4],[0, 5]]);`



Il poligono ruotato di 90°:



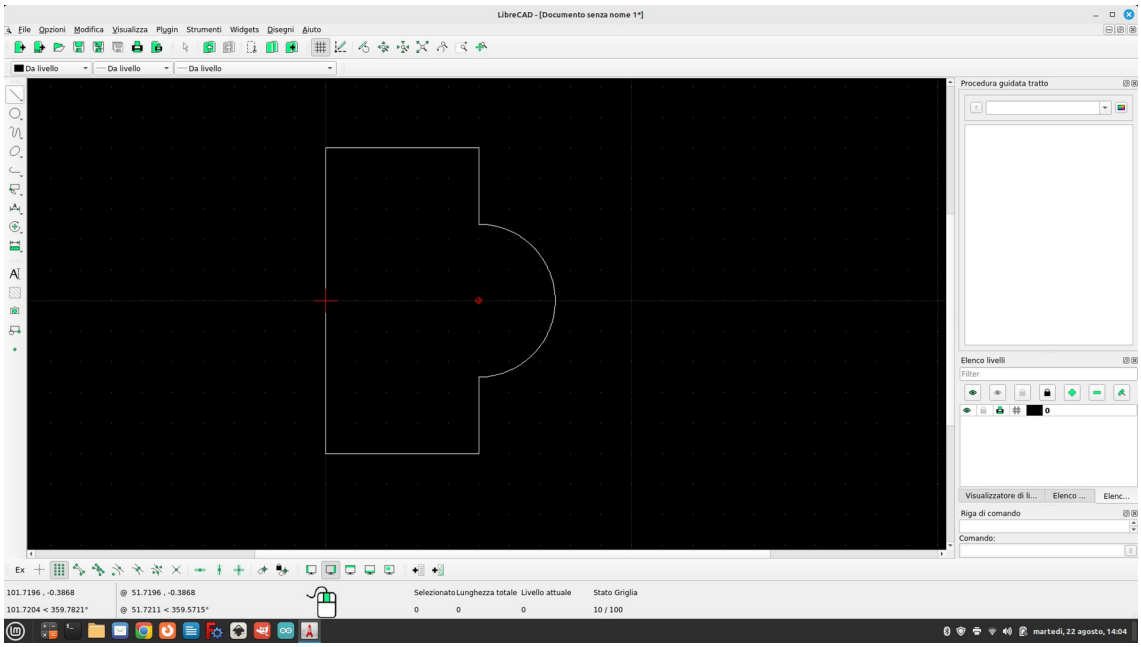


`rotate_extrude($fn=200), polygon(points=[[0,0],[2,1],[1,2],[1,3],[3,4],[0,5]]);`

Estrusione di file DXF

Il comando import() insieme ai comandi extrusion permette di estrudere oggetti 2D salvati in formato DXF producendo oggetti 3D.

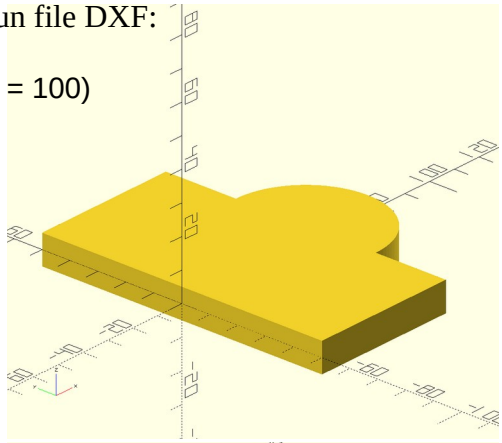
Per produrre l'esempio seguente ho disegnato una semplice figura chiusa, altrimenti il comando di estrusione non funzionerà, usando il software CAD opensource multipiattaforma LibreCAD.
Perché il comando funzioni il file DXF e quello di OpenSCAD dovranno essere salvati nella stessa cartella.



Linear Extrude

Esempio di estrusione lineare di un oggetto 2D importato da un file DXF:

```
linear_extrude(height = 10, center = true, convexity = 10, $fn = 100)
   import (file = "ExtrudeDxf.dxf");
```



Rotate Extrude

Esempio di estrusione rotazionale di un oggetto 2D importato da un file DXF.

```
rotate_extrude(height = 10, center = true, convexity = 10, $fn
= 100)
   import (file = "ExtrudeDxf.dxf");
```