Documentation

SCAD_NoName

Présentation

SCAD_NoName est une bibliothèque pour OpenSCAD. Elle ajoute des modules pour faciliter la CAO de pièces complexes destinées à l'impression 3D.

SOMMAIRE

- Sommaire **OK**

- Présentation **A FAIRE**
- BasicForms.scad **OK**
- Basics.scad **OK**
- Bevel_Chamfer.scad
- Bezier.scad **OK**
- Constants.scad **OK**
- Gears.scad **EN COURS**
- Holes.scad **OK**
- Matrix.scad **OK**
- RenardSeries.scad **OK**
- Spring.scad **OK**
- Thread.scad **OK**
- Transforms.scad **OK**
- Vector.scad **OK**

***EN COURS:** code en cours.

rot= ROT_Top : rotation à appliquer au [NOM], constante ROT_* ou un vecteur correspondant à [rotX, rotY, rotZ],

BasicForms

Dépendances

```
use<Basics.scad>
use<Matrix.scad>
use<Transforms.scad>
include<Constants.scad>
```

Présentation

Regroupe les différents modules correspondants aux différentes formes basiques.

---

colorCube, module

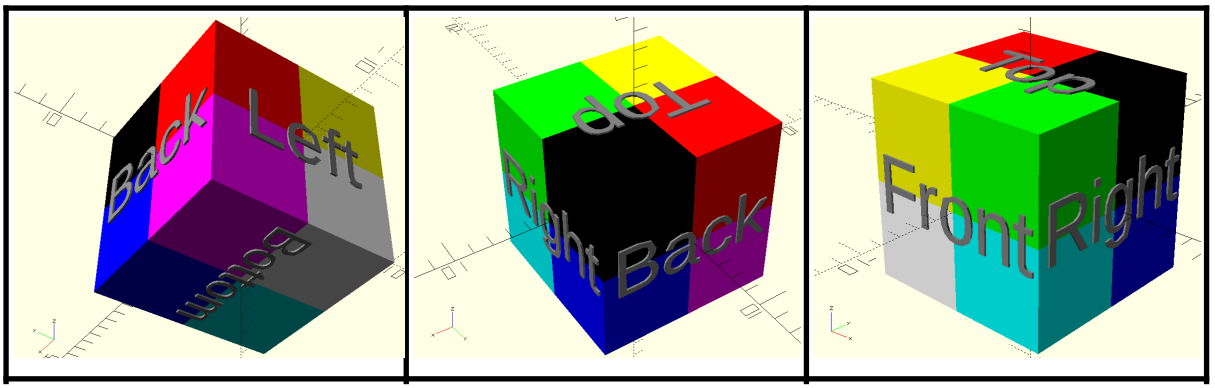
Description :

Créer un cube coloré centré en [0, 0, 0].

colorCube(**size= 1** : taille du cube)

Exemple :

Ajoute sur la face correspondante d'un colorCube de taille 10, le nom relatif à la constante de rotation appliquée. (***writeOnFace()*** est une fonction de **Basics.scad**)



Code :

```
writeOnFace(pos= [0, 0, 5], text= "Top", color= "grey", size= 3, valign=
"center", halign= "center")
writeOnFace(pos= [0, 5, 0], text= "Back", color= "grey", size= 3,
valign= "center", halign= "center", rot= ROT_Back + [0, 0, 180])
writeOnFace(pos= [0, -5, 0], text= "Front", color= "grey", size= 3,
valign= "center", halign= "center", rot= ROT_Frt)
writeOnFace(pos= [5, 0, 0], text= "Right", color= "grey", size= 3,
valign= "center", halign= "center", rot= ROT_Rgt + [0, 0, 90])
writeOnFace(pos= [-5, 0, 0], text= "Left", color= "grey", size= 3,
valign= "center", halign= "center", rot= ROT_Lft + [0, 0, -90])
writeOnFace(pos= [0, 0, -5], text= "Bottom", color= "grey", size= 2,
valign= "center", halign= "center", rot= ROT_Bot)
   colorCube(10);
```

---

chamferCube, module

Description :

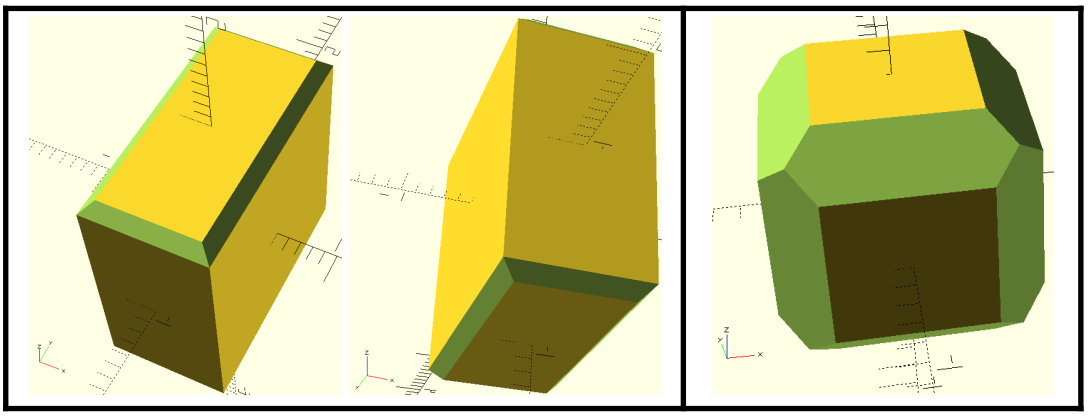
Créer un cube chanfreiné de taille [X, Y, Z]. La taille des chanfreins doit être inférieure à $\frac{min([X,Y,Z])}{2}$. “edges” est un vecteur contenant les constantes pour identifier les arêtes (énumérées dans **Constants.scad, EDGES_***).

chamferCube(

chamfer= 0.1 : taille du chanfrein,
size= [1, 1, 1] : taille du cube,
pos= [0, 0, 0] : position du cube,
rot= ROT_Top : rotation à appliquer au cube chanfreiné, constante ROT_* ou un vecteur correspondant à [rotX, rotY, rotZ],
edges= EDGE_All : vecteur des arêtes à chanfreiner,
center= false : centre ou non la pièce)

Exemple :

1. Créer un cube de taille [1, 2, 2], chanfreiné (0.1) au Dessus et en Dessous.
2. Créer un cube de taille [1, 1, 1], chanfreiné (0.2) sur toutes ses arêtes.



Code :

```
// Exemple 1:
chamferCube(size= [1, 2, 2], edges= [EDGE_Top, EDGE_Bot], center= true);

// Exemple 2:
chamferCube(chamfer= 0.2, center= true);
```

---

chamferCylinder, module

Description :

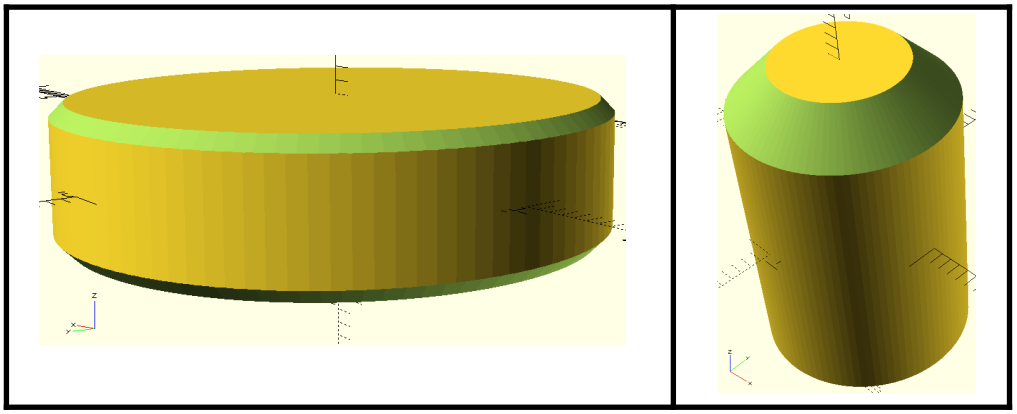
Créer un cylindre chanfreiné. La taille du chanfrein doit être inférieure à $\frac{r}{2}$. “edges” est un vecteur pouvant contenir seulement les constantes **EDGES_Top** et **EDGES_Bot** pour identifier les arêtes (énumérées dans **Constants.scaad**).

chamferCylinder(

h= 1 : hauteur du cylindre,
r= 1 : rayon du cylindre,
chamfer= 0.1 : taille du chanfrein,
chamferAng= 45 : angle du chanfrein [0, 60]°,
fn= 100 : nombre de segments du cylindre,
pos= [0, 0, 0] : position du cylindre,
rot= ROT_Top : rotation à appliquer au cylindre chanfreiné , constante ROT_* ou un vecteur correspondant à [rotX, rotY, rotZ],
edges= EDGE_All : vecteur d’arêtes à chanfreiner,
center= false : centre ou non la pièce)

Exemple :

1. Créer un cylindre de rayon 2, hauteur 1 et fn 100, chanfreiné (0.1) sur toutes ses arêtes.
2. Créer un cylindre de rayon 1, hauteur 3 et fn 100, chanfreiné (0.4) au Dessus.



Code :

```
// Exemple 1:
chamferCylinder(r= 2, center= true);

// Exemple 2:
chamferCylinder(h= 3, chamfer= 0.4, edges= [EDGE_Top], center= true);
```

***bevelCube*, module**

Description :

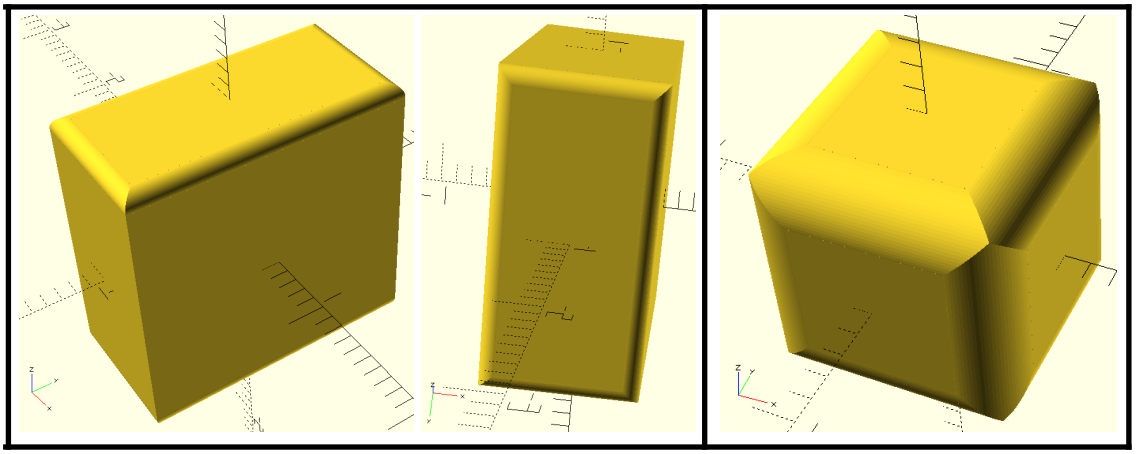
Créer un cube de taille [X, Y, Z] avec des congés. La taille des congés doit être inférieure à $\frac{min([X,Y,Z])}{2}$. “edges” est un vecteur contenant les constantes pour identifier les arêtes (énumérées dans **Constants.scad, EDGES_***).

bevelCube(

size= [1, 1, 1] : taille du cube,
bevel= 0.1 : rayon du congé,
fn= 20 : nombre de segments pour le cylindre du congé,
pos= [0, 0, 0] : position du cube,
rot= ROT_Top : rotation à appliquer au cube avec des congés, constante ROT_* ou un vecteur correspondant à [rotX, rotY, rotZ],
edges= EDGE_All : vecteur d'arêtes pour les congés,
center= false : centre ou non la pièce)

Exemple :

1. Créer un cube de taille [1, 2, 2], congé (0.1) avec un précision de 20, au Dessus et en Dessous.
2. Créer un cube de taille [1, 1, 1], congé (0.2) avec un précision de 20, sur toutes ses arêtes.



Code :

```
// Exemple 1:
bevelCube(size= [1, 2, 2], edges= [EDGE_Top, EDGE_Bot], center= true);

// Exemple 2:
bevelCube(chamfer= 0.2, center= true);
```

---

bevelCylinder, module

Description :

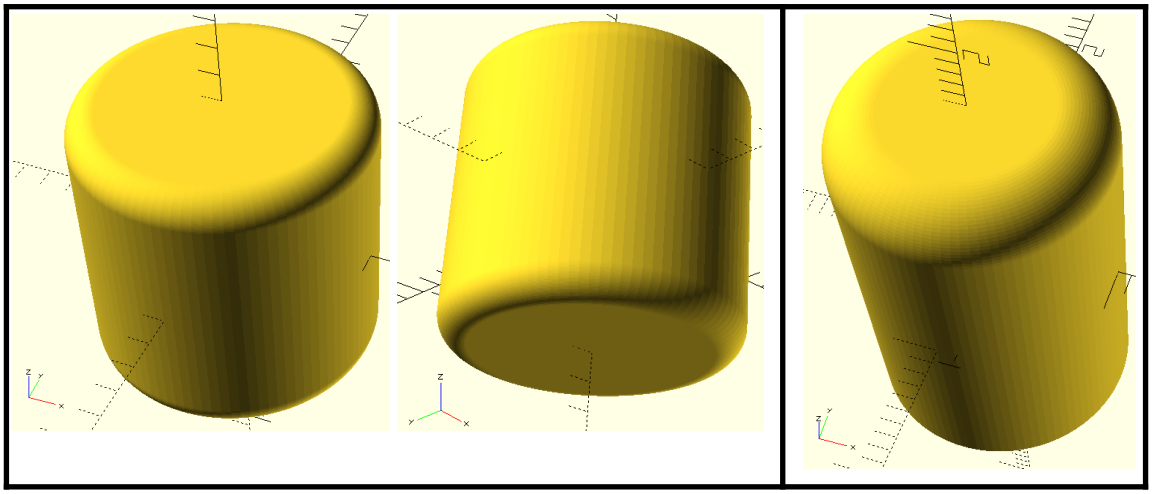
Créer un cylindre avec des congés. La taille des congés doit être inférieure à $\frac{r}{2}$. “edges” est un vecteur pouvant contenir seulement les constantes **EDGES_Top** et **EDGES_Bot** pour identifier les arêtes (énumérées dans **Constants.scaad**).

bevelCylinder(

h= 1 : hauteur du cylindre,
r= 1 : rayon du cylindre,
bevel= 0.1 : taille du chanfrein,
fn= 100 : nombre de segments du cylindre,
fnB= 100 : nombre de segments du cylindre pour le congé,
pos= [0, 0, 0] : position du cylindre,
rot= ROT_Top : rotation à appliquer au cylindre avec des congés, constante ROT_* ou un vecteur correspondant à [rotX, rotY, rotZ],
edges= EDGE_All : vecteur d’arêtes à chanfreiner,
center= false : centre ou non la pièce)

Exemple :

1. Créer un cylindre de rayon 0.5, hauteur 1 et fn, fnB= 20, avec un congé de taille (0.1) sur toutes ses arêtes.
2. Créer un cylindre de rayon 1, hauteur 3 et fn, fnB= 20, avec un congé de taille (0.4) au Dessus.



Code :

```
// Exemple 1:
bevelCylinder(r= 0.5, center= true);

// Exemple 2:
bevelCylinder(h= 3, bevel= 0.4, edges= [EDGE_Top], center= true);
```

---

linearPipe, module

Description :

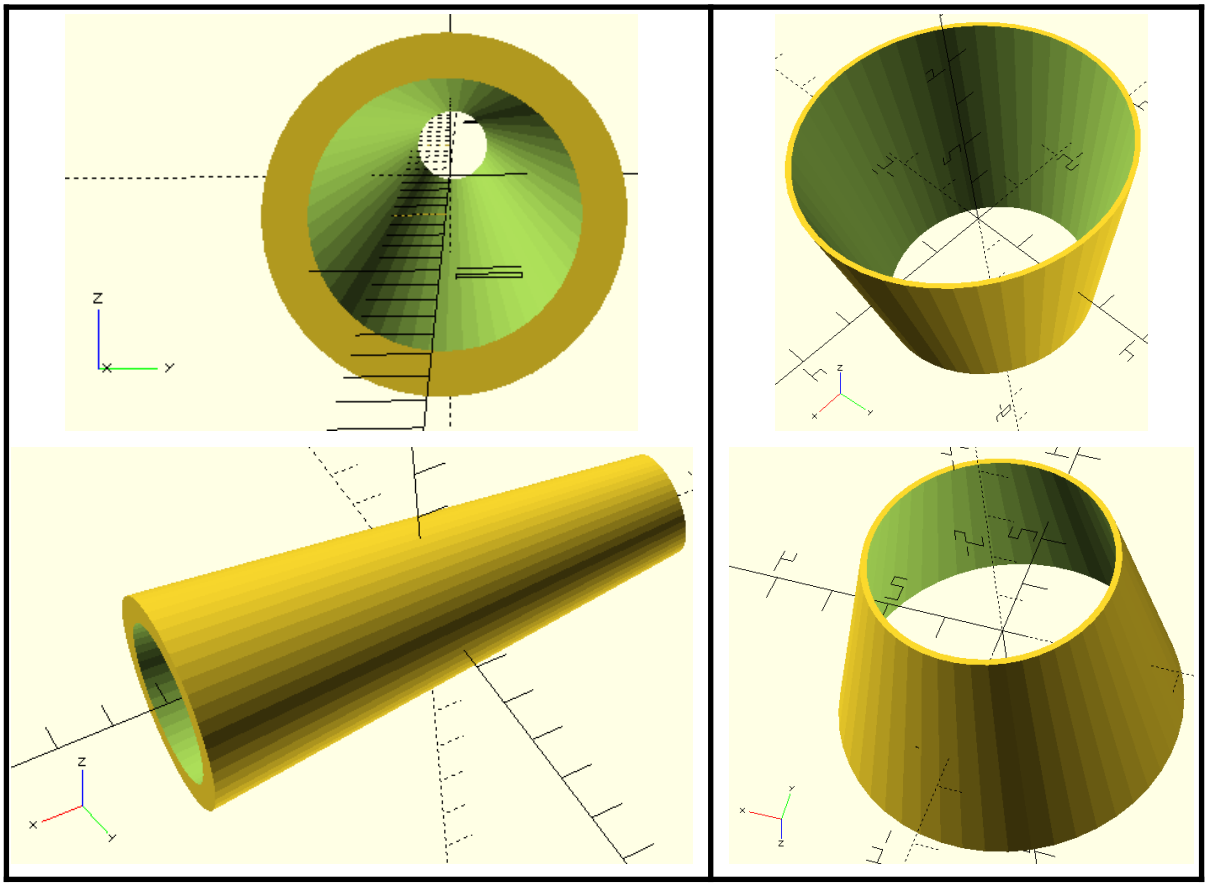
Créer un tuyau linéaire.

linearPipe(

r= 1 : rayon extérieur du cylindre inférieur,
thick= 0.1 : épaisseur,du tuyau,
r1= undef : si défini, représente le rayon du extérieur cylindre supérieur,
r2= undef : si défini, l'épaisseur n'est plus prise en compte, représente le rayon intérieur du cylindre supérieur,
h= 1 : hauteur du tuyau,
pos= [0, 0, 0] : position du tuyau,
fn= 50 : nombre de segments des cylindres,
rot= ROT_Top : rotation à appliquer au tuyau, constante ROT_* ou un vecteur correspondant à [rotX, rotY, rotZ],
center= false : booléen, centre ou non la pièce)

Exemple :

1. Créer un tuyau centré, orienté à Droite h= 10, thick= 0.5, r= 2 et r1= 1 (équivalent aux paramètres r= 2, r1= 1, r2= 0.5).
2. Créer un tuyau centré, h= 4, r= 2, r1= 3, r2= 2.9 (équivalent aux paramètres r= 2, r1= 1, thick= 0.1).



Code :

```
// Exemple 1:
linearPipe(r= 2, r1= 1, thick= 0.5, h= 10, center= true, rot= ROT_Lft,
center= true);

// Exemple 2:
linearPipe(r= 2, r1= 3, r2= 2.9, h= 4, center= true);
```

---

regularIcosahedron, module

Description :

Créer un icosaèdre régulier en fonction de la taille d'une de ses arêtes.

regularIcosahedron(

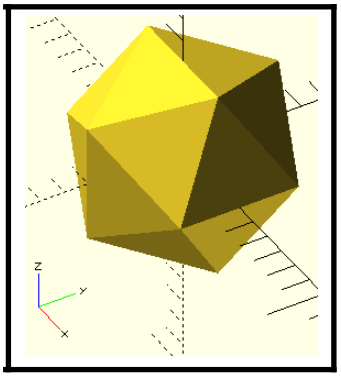
t= 1 : taille d'une arête de l'icosaèdre,

pos= [0, 0, 0] : position de l'icosaèdre,

rot= ROT_Top : rotation à appliquer à l'icosaèdre, constante ROT_* ou un vecteur correspondant à [rotX, rotY, rotZ])

Exemple :

Créer un icosaèdre régulier de taille d'arête 5.



Code :

```
regularIcosahedron(t= 5);
```

Basics

Dépendances

```
use<Transforms.scad>
include<Constants.scad>
```

Présentation

Basics contient des fonctions et modules utilitaires à la bibliothèque.

---

isDef, fonction

Description :

Test si une variable est définie. Principalement utilisé dans les modules pour tester si des paramètres sans valeur par défaut ont été initialisés.

isDef(**u** : variable quelconque)

Valeur de retour :

Booléen : True si la variable est définie, false sinon.

---

isUndef, fonction

Description :

Test si une variable n'est pas définie. Principalement utilisé dans les modules pour tester si des paramètres sans valeur par défaut n'ont pas été initialisés.

isUndef(**u** : variable quelconque)

Valeur de retour :

Booléen : True si la variable n'est pas définie, false sinon.

---

echoMsg, fonction

Description :

Affiche un message seulement si msg est défini. Il est utilisé pour afficher des messages dans une fonction.

echoMsg(**msg** : message, string)

Valeur de retour :

Booléen : True si la variable est définie, false sinon.

---

echoError, module

Description :

Affiche un message d'erreur avec un préfixe. Il est utilisé pour afficher des messages d'erreur dans un module.

echoError(

msg : message, string,

pfx : préfixe de l'erreur, par défaut "ERROR")

Valeur de retour :

String: Écrit dans la console un message d'erreur surligné en rouge avec "préfixe, message".

---

echoError, fonction

Description :

Affiche un message d'erreur avec un préfixe. Il est utilisé pour afficher des messages d'erreur dans une fonction.

echoError(

msg : message, string,

pfx : préfixe de l'erreur, par défaut "ERROR")

Valeur de retour :

String: Écrit dans la console un message d'erreur surligné en rouge avec "préfixe, message".

---

assertion, module

Description :

Affiche un message d'erreur si le test passé en paramètre n'est pas vrai. S'utilise pour restreindre ou tester des variables dans un module.

assertion(

succ : booléen si vrai n'exécute pas l'appel d'affichage

msg : message, string)

Valeur de retour :

String: Écrit dans la console le message surligné en rouge.

---

assertion, fonction

Description :

Affiche un message d'erreur si le test passé en paramètre n'est pas vrai. S'utilise pour restreindre ou tester des variables dans une fonction.

assertion(

succ : booléen si vrai n'exécute pas l'appel d'affichage

msg : message, string)

Valeur de retour :

String: Écrit dans la console le message surligné en rouge.

---

ifNullGetUnit, fonction

Description :

Si la valeur passée en paramètre est 0 ou indéfinie, renvoie 1 sinon la valeur. Très utile lors de la création de matrice de translation linéaires (cf Matrix.scad)

ifNullGetUnit(**value** : valeur à tester)

Valeur de retour :

Entier, réel, ... : Si la valeur vaut 0 ou indéfinie renvoie 1, sinon **value** .

---

regularDiameter, fonction

Description :

Donne le diamètre du cercle circonscrit d'un polygone régulier en fonction de la longueur d'un de ses côtés.

regularDiameter(

nbS : nombre de côtés du polygone (minimum 3),

lengthS : longueur d'un côté)

Valeur de retour :

Réel: Diamètre du cercle circonscrit du polygone.

---

regularRadius, fonction

Description :

Donne le rayon du cercle circonscrit d'un polygone régulier en fonction de la longueur d'un de ses côtés.

regularRadius(

nbS : nombre de côtés du polygone (minimum 3),
lengthS : longueur d'un côté)

Valeur de retour :

réel: Rayon du cercle circonscrit du polygone.

---

writeOnFace, module

Description :

Ajoute ou enlève un texte sur un module. Utilise la fonction ***text()*** de Openscad merci de vous référencer à sa documentation pour l'utilisation de ses paramètres :

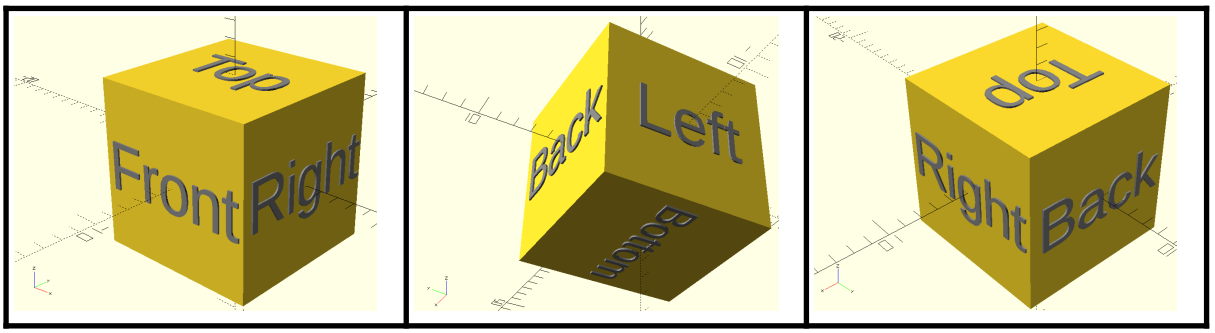
https://en.wikibooks.org/wiki/OpenSCAD_User_Manual/Text

writeOnFace(

pos : nombre de côtés du polygone (minimum 3),
color : couleur du texte, par défaut à "white" (utilise la fonction ***color()*** de Openscad),
text : paramètre de la fonction ***text()***,
size : paramètre de la fonction ***text()***,
font : paramètre de la fonction ***text()***,
halign : paramètre de la fonction ***text()***,
valign : paramètre de la fonction ***text()***,
spacing : paramètre de la fonction ***text()***,
direction : paramètre de la fonction ***text()***,
language : paramètre de la fonction ***text()***,
script : paramètre de la fonction ***text()***,
fn : nombre de segments pour représenter les lettres (paramètre de la fonction ***text()***),
h : hauteur du texte à ajouter ou à enlever,
rot : rotation à appliquer au texte, constante ROT_* ou un vecteur correspondant à [rotX, rotY, rotZ],
diff : booléen, si true effectue un enlèvement de matière, sinon l'ajoute)

Exemple :

Ajoute sur la face correspondante le nom relatif à la constante de rotation appliquée.



Code :

```
writeOnFace(pos= [0, 0, 5], text= "Top", color= "grey", size= 3, valign= "center",
halign= "center")
writeOnFace(pos= [0, 5, 0], text= "Back", color= "grey", size= 3, valign= "center",
halign= "center", rot= ROT_Back + [0, 0, 180])
writeOnFace(pos= [0, -5, 0], text= "Front", color= "grey", size= 3, valign= "center",
halign= "center", rot= ROT_Frt)
writeOnFace(pos= [5, 0, 0], text= "Right", color= "grey", size= 3, valign= "center",
halign= "center", rot= ROT_Rgt + [0, 0, 90])
writeOnFace(pos= [-5, 0, 0], text= "Left", color= "grey", size= 3, valign= "center",
halign= "center", rot= ROT_Lft + [0, 0, -90])
writeOnFace(pos= [0, 0, -5], text= "Bottom", color= "grey", size= 2, valign= "center",
halign= "center", rot= ROT_Bot)
   cube(10);
```

---

factorial, fonction

Description :

retourne la factorielle d'un nombre.

factorial(**n** : entier)

Valeur de retour :

Entier: n!

---

choose, fonction

Description :

Calcule $\binom{n}{k}$.

choose(

n : coefficient supérieur,

k : coefficient inférieur)

Valeur de retour :

Retourne la valeur de $\binom{n}{k}$

Bevel_Chamfer

Dépendances

```
include<Constants.scad>
use<Basics.scad>
use<Transforms.scad>
use<Vector.scad>
```

Présentation

Bevel_Chamfer regroupe les modules permettant de générer des chanfreins et des congés à insérer à vos modules.

---

bevel, module

Description :

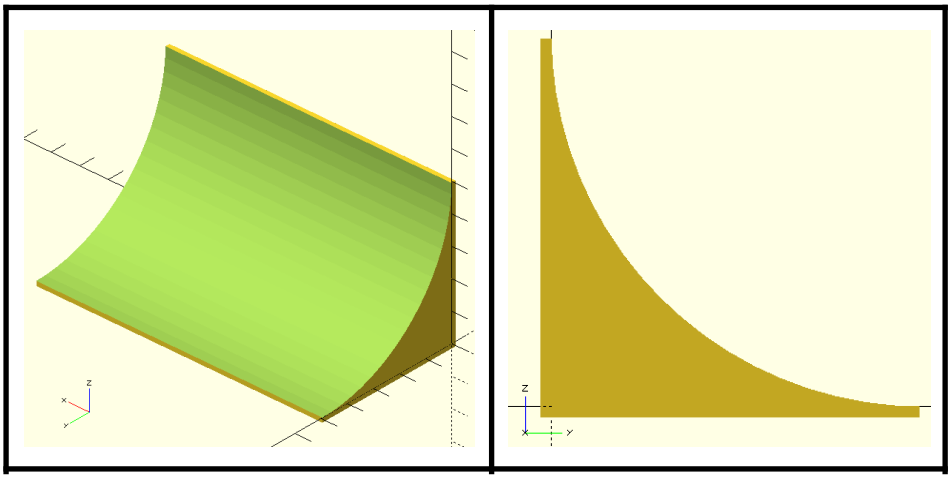
Créer un congé en fonction de son rayon et de sa longueur. Il dispose d'une excroissance de 0.1mm derrière et en dessous pour l'ajouter à votre pièce.

bevel(

r= 1 : rayon du congé,
length= 1 : longueur du congé,
fn= 20 : précision du cylindre pour représenter le congé,
pos= [0, 0, 0] : position finale du congé,
rot= ROT_Top : rotation à appliquer au congé, constante ROT_* ou un vecteur correspondant à [rotX, rotY, rotZ],
orient= ORIENT_1 : orientation à appliquer au congé, constante ORIENT_*,
center= false : centre ou non le congé)

Exemple :

Créer un un congé (r= 5, length= 10) orienté sur la gauche avec l'orientation n°2.



Code :

```
bevel(r= 5, length= 10, rot= ROT_Rgt, orient= ORIENT_2,);
```

---

cylindricalBevel, module

Description :

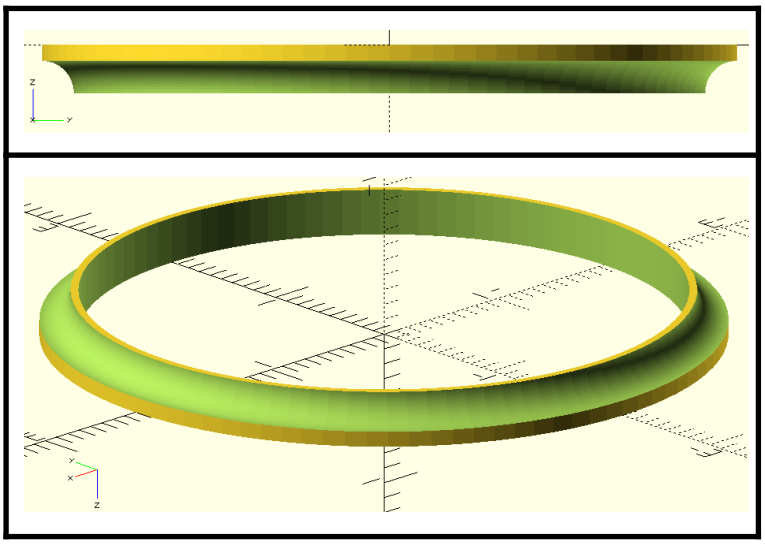
Créer un congé circulaire en fonction de son rayon, de sa longueur et de son rayon de base. Il dispose d'une excroissance de 0.1mm derrière et en dessous pour l'ajouter à votre pièce.

cylindricalBevel(

r= 1 : rayon du congé,
R= 1 : rayon de base où le congé doit être appliqué,
fn= 20 : précision du cylindre pour représenter le cylindre,
fnB= 20 : précision du cylindre pour représenter le congé,
pos= [0, 0, 0] : position finale du congé,
rot= ROT_Top : rotation à appliquer au congé cylindrique, constante ROT_* ou un vecteur correspondant à [rotX, rotY, rotZ],
center= false : centre ou non le congé)

Exemple :

Créer un un congé cylindrique (r= 0.2, R= 2) orienté vers le bas.



Code :

```
cylindricalBevel(r= 0.2, R= 2, fn= 100, rot= ROT_Bot);
```

---

chamfer, module

Description :

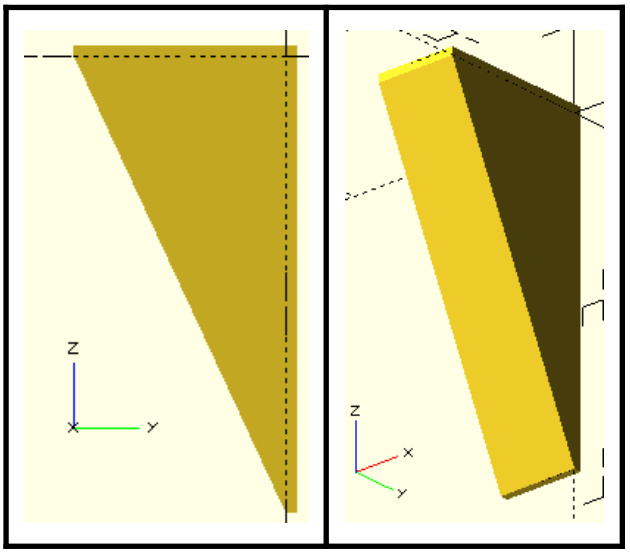
Créer un chanfrein en fonction de son angle et de sa longueur. Il dispose d'une excroissance de 0.1mm derrière et en dessous pour l'ajouter à votre pièce. L'angle du chanfrein doit être compris entre 20 et 60°.

bevel(

chamfer= 1 : longueur du chanfrein,
length= 1 : longueur du chanfrein,
chamferAng= 1 : angle du chanfrein, doit appartenir à [20, 60]°,
fn= 20 : précision du cylindre pour représenter le chanfrein,
pos= [0, 0, 0] : position finale du chanfrein,
rot= ROT_Top : rotation à appliquer au chanfrein, constante ROT_* ou un vecteur correspondant à [rotX, rotY, rotZ],
orient= ORIENT_1 : orientation à appliquer au chanfrein, constante ORIENT_*,
center= false : centre ou non le chanfrein)

Exemple :

Créer un un chanfrein (chamfer= 4, length= 1, chamferAng= 25) orienté sur la gauche avec l'orientation n°3.



Code :

```
chamfer(chamfer= 4, chamferAng= 25, rot= ROT_Lft, orient= ORIENT_3)
```

---

cylindricalChamfer, module

Description :

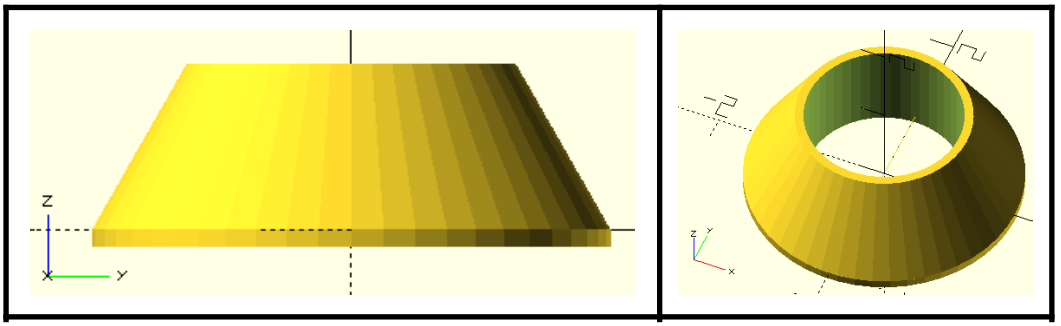
Créer un chanfrein circulaire en fonction de son angle, de sa longueur et de son rayon de base. Il dispose d'une excroissance de 0.1mm derrière et en dessous pour l'ajouter à votre pièce. L'angle du chanfrein doit être compris entre 20 et 60°.

cylindricalChamfer(

chamfer= 1 : longueur du chanfrein,
chamferAng= 1 : angle du chanfrein, doit appartenir à [20, 60]°,
R= 1 : rayon de base où le chanfrein doit être appliqué,
fn= 20 : précision du cylindre pour représenter le chanfrein,
pos= [0, 0, 0] : position finale du chanfrein,
rot= ROT_Top : rotation à appliquer au chanfrein cylindrique, constante ROT_* ou un vecteur correspondant à [rotX, rotY, rotZ],
center= false : centre ou non le congé)

Exemple :

Créer un un chanfrein cylindrique (chamfer= 1, chamferAng= 60, R= 1) orienté vers le haut.



Code :

```
cylindricalChamfer(chamferAng= 60, fn= 50);
```

Bezier

Dépendances

```
use<Transforms.scad>
use<Basics.scad>
use<Vector.scad>
include<Constants.scad>
```

Présentation

Bezier contient des fonctions et des modules pour tracer des courbes de Bézier. Les points de contrôles sont apparents uniquement lors de l'aperçu, ils n'apparaissent plus lors du rendu.

---

bernstein, fonction

Description :

Retourne le polynôme de Bernstein

$$B_k^n(t) = \binom{n}{k} t^k (1-t)^{n-k}$$

bernstein(

k: index,

n : index max,

t : précision ([0, 1]))

Valeur de retour :

Retourne le polynôme de Bernstein.

---

pointBezier, fonction

Description :

Calcule la représentation paramétrique en fonction de t pour la courbe de Bézier.

$$P(t) = \sum_{i=0}^{n} (B_i^n(t) P_i)$$

pointBezier(

pts: (P_i) vecteur de points de contrôles,

n : index max,
t : précision ([0, 1]),
k= 0 : paramètre de récursion)

Valeur de retour :

Retourne la représentation paramétrique en fonction de t pour la courbe de Bézier.

***bezierCurve*, module**

Description :

Place fn children()/points pour représenter la courbe de Bézier correspondante aux points de contrôles pts.

bezierCurve(

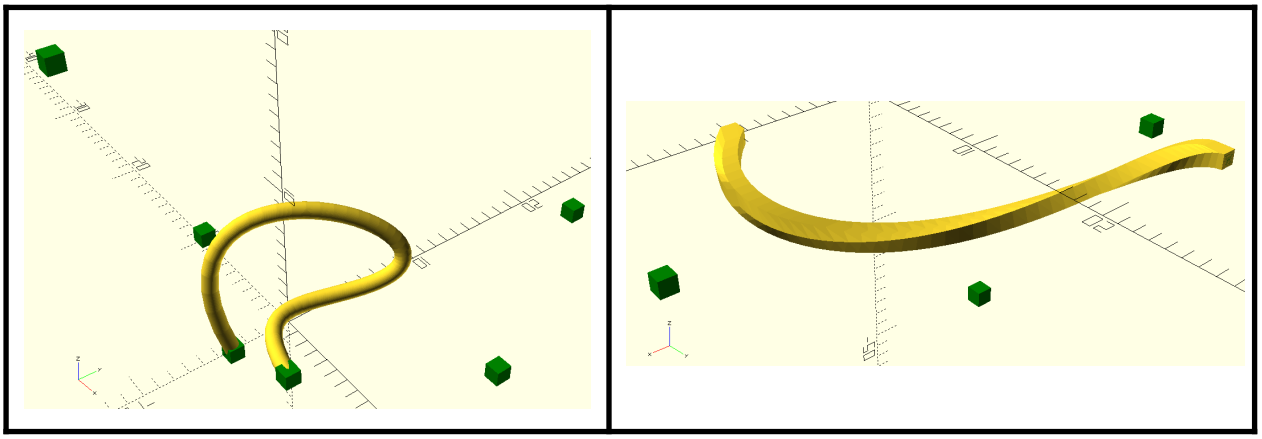
pts : (P_i) vecteur points de contrôles,

fn= 10 : nombre de children()/ points pour représenter la courbe,
ang= undef : vecteur d'angles de rotations [rotX, rotY, rotZ] à appliquer entre le premier et le dernier children())

Exemple :

1. Trace la courbe de Bézier correspondante avec des sphères de rayon 0.5 aux points de contrôles 2*pts et pour 50 segments, les cubes vert représentent les points de contrôle.

2. Trace la courbe de Bézier correspondante avec des cubes de rayon 0.5 aux points de contrôles 10*pts et pour 20 segments, les cubes vert représentent les points de contrôle.



Code :

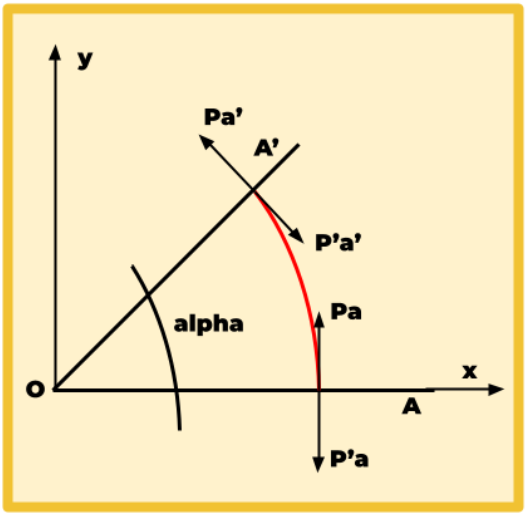
```
// Exemple 1:
pts1 = [[0,-2,0], [-2,-5,10], [3,9,2], [6,3,1], [-3,-1,2], [3,-2,1.5]];

// Points de contrôles
for(i = [0 : len(pts1) - 1]){

      color("green")
            mTranslate(2*pts1[i])
                  cube(1, $fn= 50, center= true);
}

bezierCurve(2*pts1, 50)
      sphere(0.5, $fn= 50);

// Exemple 2:
pts2 = [[1, 0, 0], [2, 1.1, 0], [0, 1.1, -1], [-1, 1.5, 0], [-1, 2, 0]];

// Points de contrôles
for(i = [0 : len(pts2) - 1]){

      color("green")
            mTranslate(10*pts2[i])
                  cube(1, $fn= 50, center= true);
}
bezierCurve(10*pts2, 20, ang= [180, 0, 0])
      cube(0.1, center= true);
```

---

bezierArcPts, fonction

Description :

Calcule la position de P et P' en fonction de A. Si hélicoïde, H est ajouté pour orienter la courbe.



bezierArcPts(

alpha : angle de l'arc de cercle]0, 180],
r : rayon du cercle,
A : premier point de l'arc de cercle,
helicoide : booléen si vrai applique H,
H : correction (utilisé pour ***bezierArcCurve()***))

Valeur de retour :

Retourne le vecteur [P, P'] suivant A.

---

bezierArcCurve, module

Description :

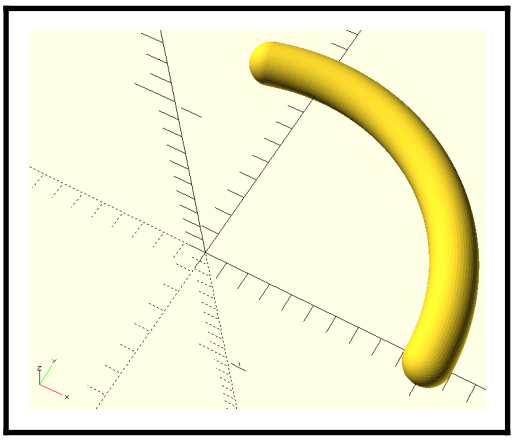
Trace une courbe de Bézier circulaire à partir du poir A.

bezierArcCurve(

A= [1, 0, 0] : point de départ de l'arc de cercle,
alpha= 45 : angle de l'arc de cercle]0, 180]
r= 1 : rayon du cercle,
fn= 10 : nombre de children()/ points pour représenter la courbe,
p= undef : pas (hauteur entre A et A'), 0 si indéfini,
rot= undef : vecteur d'angles de rotations [rotX, rotY, rotZ] à appliquer entre le premier et le dernier children(),
theta= [0, 0, 0] : si rot vrai applique le vecteur de rotations[rotX, rotY, rotZ])

Exemple :

Trace l'arc de cercle de 105° avec une courbe de Bézier avec des sphères de rayon 1 au premier point de contrôle A et avec 50 segments.



Code :

```
bezierArcCurve(alpha= 105, fn= 50)
      sphere(0.1, $fn= 50);
```

---

parametricPoint, fonction

Description :

Retourne le coordonnées paramétrique d'un point de la surface de bézier en (u, v).

$$p(u,v) = \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} B_i^{n-1}(u) B_j^{m-1}(v) M_{i,j}$$

bezierCircularPts(

M : matrice de points de contrôles,
n : nombre de lignes de la matrice M,
m : nombre de colonnes de la matrice M,
u : coordonnée de u,
v : coordonnée de v,
i= 0 : paramètre de récursion)

Valeur de retour :

Retourne le point de la surface paramétrique en fonction des coordonnées u et v.

---

bezierSurface, module

Description :

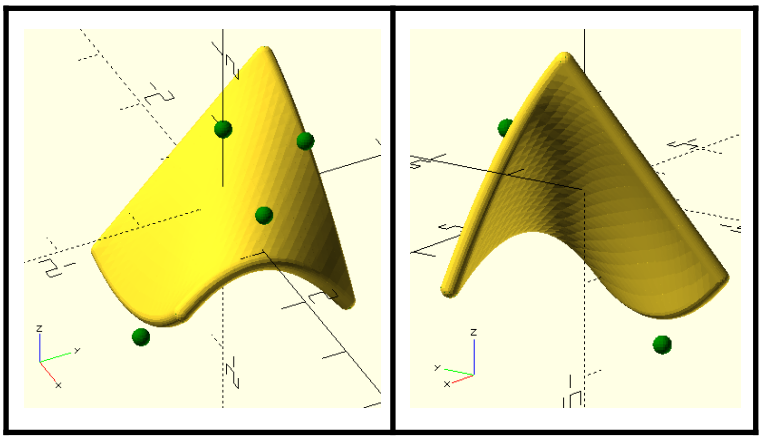
Trace une surface de Bézier à partir d'une matrice $M_{n,m}(\Re)$. Notez que la matrice doit-être au minimum carrée d'ordre 2. Les valeurs U et V ne sont utilisées indépendamment l'une de l'autre, que lorsqu'elles sont définies et appartiennent à]0, 1[; sinon c'est $\frac{1}{fn}$ qui détermine la précision.

bezierSurface(

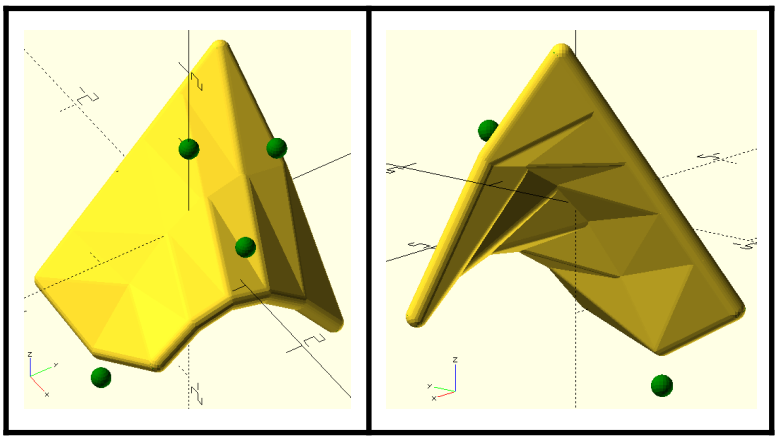
M : matrice de points de contrôles de la surface à tracer,
U= undef : précision entre deux points d'une ligne de M,
V= undef : précision entre deux points d'une colonne de M,
p= 0.5 : pas (hauteur entre chaques révolutions),
fn= 10 : nombre de children()/ points pour représenter la courbe sur une ligne /colonne)

Exemple :

1. Trace la surface de Bézier à partir de la matrice M, avec pour précision 1/20 pour U et V.



2. Trace la surface de Bézier à partir de la matrice M, avec pour précision U= 0.2 et V = 0.5.



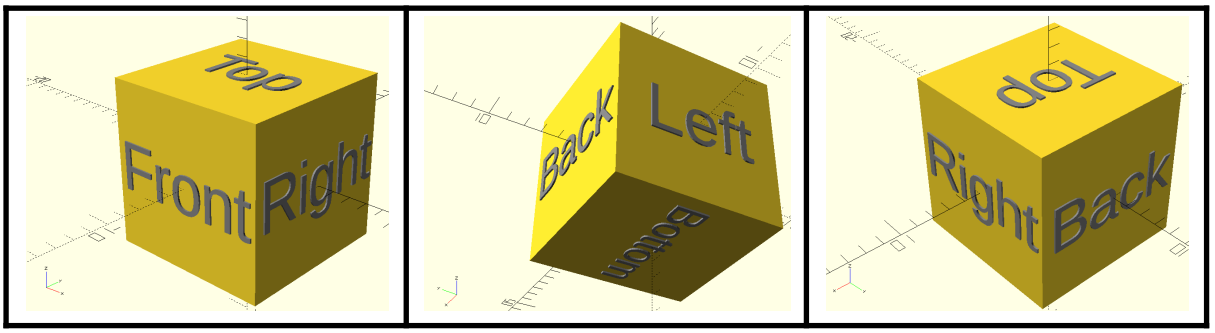
Code :

```
matrix= [[[-1,   1,  1],  [0, 1,  0.5],   [1, 1,  -1]],
         [[-1,   0,  0],  [0, 0,  1],     [1, 0,  0.5]],
         [[-1,   -1, -1], [0, -1, -1.5],  [1, -1, -0.5]]];

// Exemple 1:
bezierSurface(M= matrix,fn= 20)
   sphere(0.1, $fn= 20);

// Exemple 2:
bezierSurface(M= matrix, U= 0.2, V= 0.5)
   sphere(0.1, $fn= 20);
```

Constants

Présentation

Constants contient toutes les constantes qui peuvent-être utilisées avec certaines fonctions de la bibliothèque.
Le tableau suivant représente la position correspondante au suffixe des constantes.



*TRANS_**, constante

Description :

Constantes pouvant êtres utilisées pour effectuer des translations. Elle peuvent bien sûr être combinées : `3*TRANS_Top + TRANS_Lft = [-1, 0, 3];`

TRANS_

Null : nulle [0, 0, 0],
Top : Dessus [0, 0, 1],
Bot : Dessous [0, 0, -1],
Frt : Avant [0, -1, 0],
Back: Arrière [0, 1, 0],
Rgt: Droite [1, 0, 0],
Lft : Gauche [-1, 0, 0],
AllPos : Tous positifs [1, 1, 1],
AllNeg : Tous négatifs [-1, -1, -1])

*ROT_**, constante

Description :

Constantes pouvant êtres utilisées pour effectuer des rotation afin d'orienter une pièce. Elle peuvent bien sûr être combinées :

```
ROT_Top + ROT_Lft = [0, -90, 0];
```

TRANS_ **Top** : Dessus [0, 0, 0], (considéré comme étant nulle)
Bot : Dessous [180, 0, 0],
Frt : Avant [90, 0, 0],
Back: Arrière [-90, 0, 0],
Rgt: Droite [0, 90, 0],
Lft : Gauche [0, -90, 0])

---

*EDGE_**, constante

Description :

Constantes pouvant êtres utilisées pour identifier les arrête à affecter avec une fonction. Elle peuvent être combinées en étant énuméré dans un vecteur : `edges= [EDGE_Top, EDGE_BackLft];`

Attention : ces constantes désignent la matrice de transformation permettant de trouver les arêtes correspondantes à une face.

EDGE_* **Top** : Dessus [[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, 1], [0, 0, 0, 1]],
Bot : Dessous [[1, 0, 0, 0], [0, 1, 0, 0], [0, 0, 1, -1], [0, 0, 0, 1]],
Frt : Avant [[1, 0, 0, 0], [0, 1, 0, 1], [0, 0, 1, 0], [0, 0, 0, 1]],
Back : Arrière [[1, 0, 0, 0], [0, 1, 0, -1], [0, 0, 1, 0], [0, 0, 0, 1]],
Rgt : Droite [[1, 0, 0, 1], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]],
Lft : Gauche [[1, 0, 0, -1], [0, 1, 0, 0], [0, 0, 1, 0], [0, 0, 0, 1]])

Attention : ces constantes désignent la matrice de transformation permettant de trouver une arête, notez que certaines arêtes sont en double mais ont été laissées afin de laisser toutes les combinaisons possibles.

EDGE_Top* **Frt** : Au Dessus et en Avant, EDGE_Top*EDGE_Frt,
Back : Au Dessus et en Arrière, EDGE_Top*EDGE_Back,
Rgt : Au Dessus et à Droite, EDGE_Top*EDGE_Rgt,
Lft : Au Dessus et à Gauche, EDGE_Top*EDGE_Lft)

EDGE_Bot* **Frt** : En Dessous et en Avant, EDGE_Bot*EDGE_Frt,
Back : En Dessous et en Arrière, EDGE_Bot*EDGE_Back,
Rgt : En Dessous et à Droite, EDGE_Bot*EDGE_Rgt,
Lft : En Dessous et à Gauche, EDGE_Bot*EDGE_Lft)

EDGE_Back* **Top** : En Arrière et au Dessus, équivalent à EDGE_TopBack,
Bot : En Arrière et au Dessous, équivalent à EDGE_BotBack,
Rgt : En Arrière et à Droite, EDGE_Back*EDGE_Rgt,
Lft : En Arrière et à Gauche, EDGE_Back*EDGE_Lft)

EDGE_Frt* **Top** : En Avant et au Dessus, équivalent à EDGE_TopFrt,
Bot : En Avant et au Dessous, équivalent à EDGE_BotFrt,
Rgt : En Avant et à Droite, EDGE_Frt*EDGE_Rgt,
Lft : En Avant et à Gauche, EDGE_Frt*EDGE_Lft)

EDGE_Rgt* **Top** : À Droite et au Dessus, équivalent à EDGE_TopRgt,
Bot : À Droite et au Dessous, équivalent à EDGE_BotRgt,
Frt : À Droite et en Avant, équivalent à EDGE_FrtRgt,
Back : À Droite et en Arrière, équivalent à EDGE_BackRgt)

EDGE_Left* **Top** : À Gauche et au Dessus, à EDGE_TopLft,
Bot : À Gauche et au Dessous, équivalent à EDGE_BotLft,
Frt : À Gauche et en Avant, équivalent à EDGE_FrtLft,
Back : À Gauche et en Arrière, équivalent à EDGE_BackLft)

Attention : cette constante applique la fonction sur toutes les arêtes.
EDGE_All = [EDGE_FrtLft, EDGE_TopLft, EDGE_BotLft]

Pour les ressorts, vous serez amené à utiliser les constantes pour définir le sens d'enroulement :

CLOCKWIRE : sens horaire
ANTICLOCKWIRE : sens trigonométrique

Pour les engrenages, vous serez amené à utiliser les constantes pour définir le sens du pas :

RIGHT_HAND : pas à droite
LEFT_HAND : pas à gauche

Pour les moletages, vous serez amené à utiliser les constantes pour définir le type de moletages:

HORIZONTAL : moletage horizontale
VERTICAL : moletage vertical

Gears

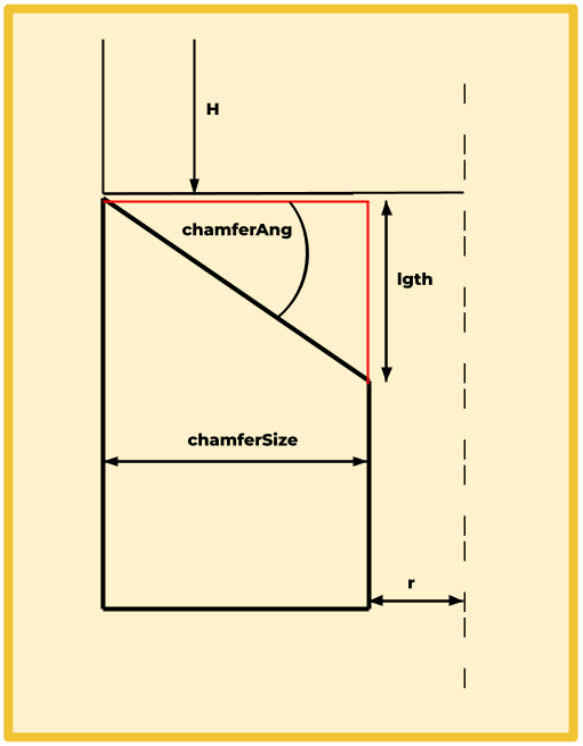
Holes

Dépendances

```
use<Transforms.scad>
use<Basics.scad>
include<Constants.scad>
```

Présentation

Holes contient les fonctions de perçage de la bibliothèque.



$chamferAng \in]0,\ 90°[$

---

hole, module

Description :

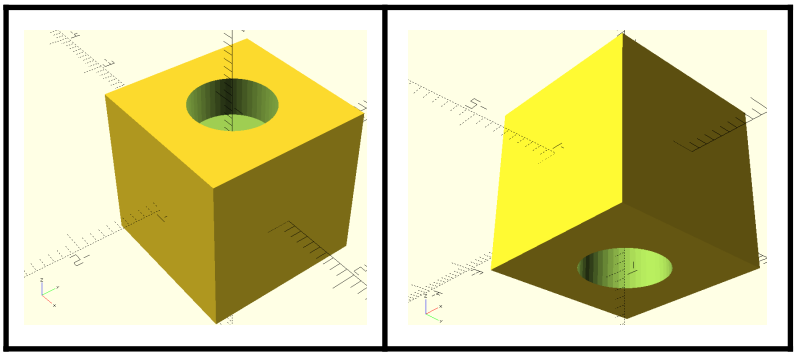
Perce dans la première pièce les trous suivants. Cette fonction utilise la fonction ***children()***, le premier children en paramètre sera la pièce percée. Les children énumérées ensuite sont les perçages à effectuer.

hole(

pos : vecteur énumérant les position des différents trous à percer,
rots : vecteur énumérant les orientation des différents trous à percer)

Exemple :

Perce au Dessus et au Dessous d'un cube(taille: 2) avec deux cylindres(rayon: 0.5, h= 0.5) aux positions respectives des faces et rotations (au Dessus, au Dessous).



Code :

```
hole([[0, 0, 0.5], [0, 0, -0.5]], [ROT_Top, ROT_Bot]){

   cube(2, center= true);

   cylinder(r= 0.5, h= 0.5 + 0.01, $fn= 50);

   cylinder(r= 0.5, h= 0.5 + 0.01, $fn= 50);
}
```

---

cylinderHole, module

Description :

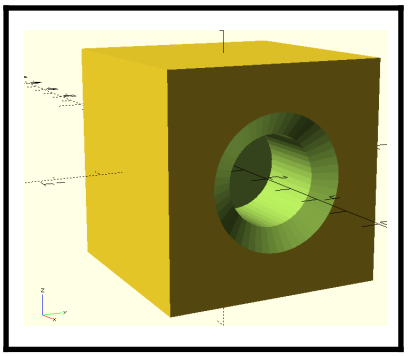
Perce dans la pièce passée en children(), un cylindre chanfreiné ou non.

cylinderHole(

pos= [0, 0, 0] : position du cylindre à percer,
r= 1 : rayon du cylindre,
h= 1 : profondeur du perçage,
fn= 50 : nombre de segments du cylindre,
chamfer= false : booléen, si true effectue un chanfrein, par défaut à false,
chamfersize= 0.1 : largeur du chanfrein,
chamferAng= 30 : angle du chanfrein,
rot= ROT_Top : rotation à appliquer au trou, constante ROT_* ou un vecteur correspondant à [rotX, rotY, rotZ],
H= 0 : hauteur du cylindre de “pré-perçage”, de base à 0)

Exemple :

Perce sur la Droite d’un cube(taille: 5) un cylindre(rayon: 1, h= 2) (rotations à Droite) chanfreiné à 30° et de longueur 0,5.



Code :

```
cylinderHole(pos= [2.5, 0, 0], r= 1, h= 2, fn= 50, chamfer= true,
chamferSize= 0.5, rot= ROT_Rgt)
   cube(5, center= true);
```

---

cubeHole, module

Description :

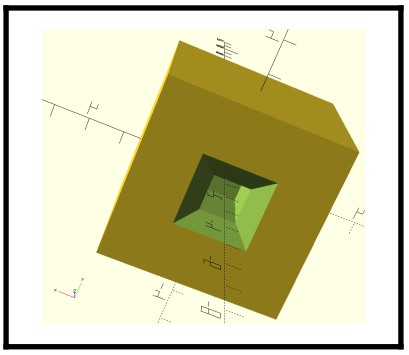
Perce dans la pièce passée en children() un carré chanfreiné ou non.

cubeHole(

pos= [0, 0, 0] : position du carré à percer,
c= 1 : taille des côtés du carré,
h= 1 : profondeur du perçage,
chamfer= false : booléen, si true effectue un chanfrein, par défaut à false,
chamfersize= 0.1 : largeur du chanfrein,
chamferAng= 30 : angle du chanfrein,
rot= ROT_Top : rotation à appliquer au trou, constante ROT_* ou un vecteur correspondant à [rotX, rotY, rotZ],
H= 0 : hauteur du cylindre de “pré-perçage”, de base à 0)

Exemple :

Perce en Dessous d’un cube(taille: 5) un carré(côté: 1, h= 2) (rotations en Dessous) chanfreiné à 30° et de longueur 0,1.



Code :

```
cubeHole(pos= [0, 0, -2.5], c= 1, h= 2, chamfer= true, chamferSize= 0.5,
rot= ROT_Bot)
   cube(5, center= true);
```

---

squareHole, module

Description :

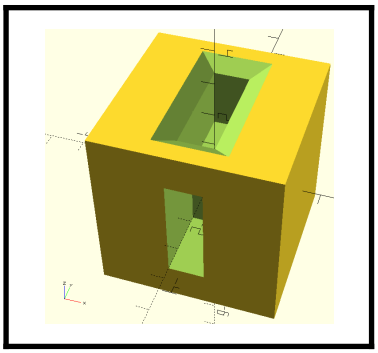
Perce dans la pièce passée en children() un carré avec comme côté le vecteur [x, y], chanfreiné ou non.

squareHole(

pos= [0, 0, 0] : position du carré à percer,
size= [1, 1] : taille [x, y] des côtés du carré,
h= 1 : profondeur du perçage,
chamfer= false : booléen, si true effectue un chanfrein, par défaut à false,
chamfersize= 0.01 : largeur du chanfrein,
chamferAng= 30 : angle du chanfrein,
rot= ROT_Top : rotation à appliquer au trou, constante ROT_* ou un vecteur correspondant à [rotX, rotY, rotZ],
H= 0 : hauteur du cylindre de "pré-perçage", de base à 0)

Exemple :

Perce en Dessous et à l'Avant d'un cube(taille: 5) respectivement un carré(côté: [1, 3], h= 2) (rotations en Avant, en Dessus) avec seulement au dessus un chanfrein à 30° et de longueur 0.5.



Code :

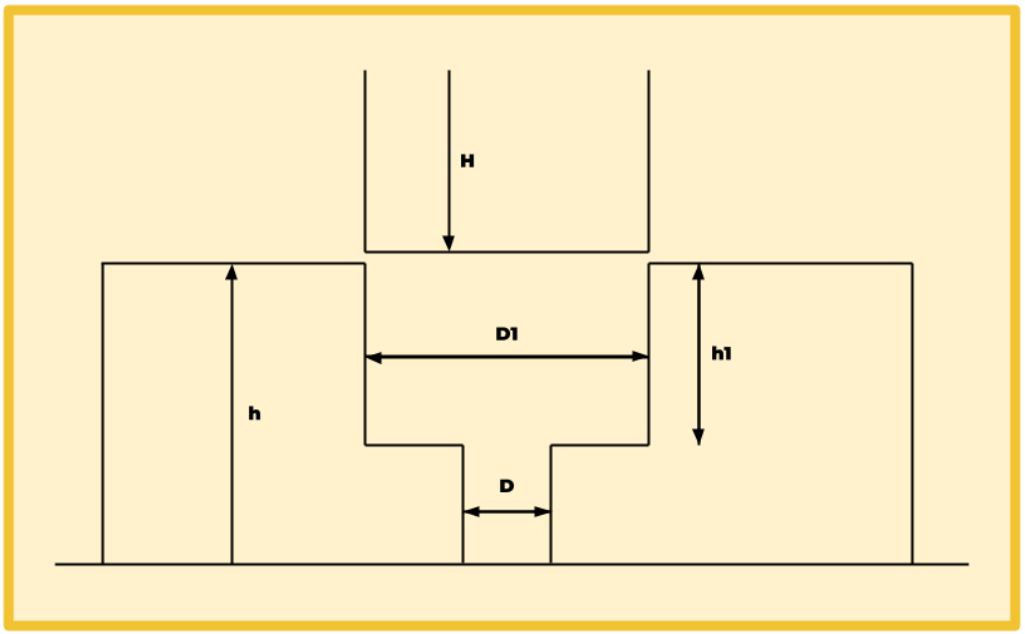
```
squareHole(pos= [0, -2.50, 0], size= [1, 3], h= 2, rot= ROT_Frt)
   squareHole(pos= [0, 0, 2.50], size= [1, 3], h= 2, chamfer= true,
chamferSize= 0.5)
       cube(5, center= true);
```

---

counterbore, module

Description :

Perce dans la pièce passée en children() un chambrage chanfreiné ou non (seulement l'angle au sommet).

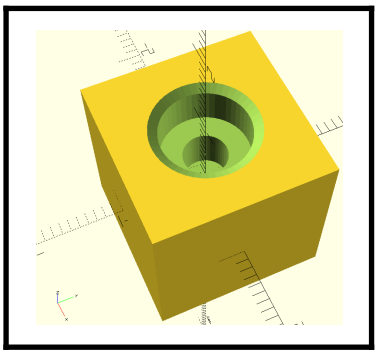


counterbore(

pos= [0, 0, 0] : position du chambrage,
D= 0.5 : diamètre du perçage,
h= 1 : profondeur du perçage,
D1= 1 : diamètre de la chambre,
h1= 0.5 : profondeur de la chambre,
fn=50 : nombre de segments du cylindre
chamfer= false : booléen, si true effectue un chanfrein,
chamfersize= 0.1 : largeur du chanfrein,
chamferAng= 30 : angle du chanfrein,
rot= Rot_Top : rotation à appliquer au chambrage, constante ROT_* ou un vecteur correspondant à [rotX, rotY, rotZ],
H= 0 : hauteur du cylindre de "pré-perçage", de base à 0)

Exemple :

Réalise au Dessus d'un carré(taille: 2) un chambrage(D= 0.5, h= 1, D1= 1, h1= 0.5) chanfreiné à un angle de 30° et un largeur de 0.1



Code :

```
counterbore(pos= [0, 0, 1], chamfer= true)
      cube(2, center= true);
```

---

cylindricalAxleHole, module

Description :

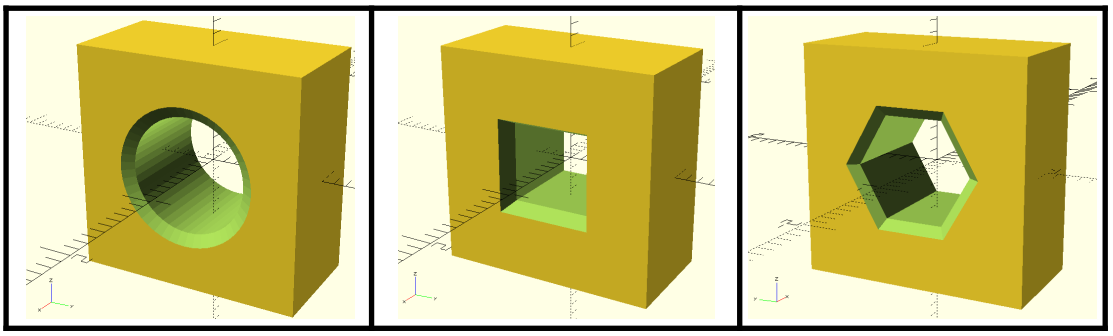
Perce dans la pièce passée en module (/"action()") un axe cylindrique avec comme côté le vecteur [x, y], chanfreiné ou non.

cylindricalAxleHole(

pos= [0, 0, 0] : position de la face à percer,
Daxe= 1 : diamètre de l'axe,
deltaD= 0 : jeu entre l'axe et le perçage (différence en le diamètre de perçage et le diamètre de l'axe, de base à 0),
h= 1 : épaisseur de la pièce,
fn= 50 : nombre de segments du cylindre,
rot= Rot_Top : rotation à appliquer au trou, constante ROT_* ou un vecteur correspondant à [rotX, rotY, rotZ],
chamfer= false : booléen, si true effectue un chanfrein, par défaut à false,
chamfersize= 0.1 : largeur du chanfrein,
chamferAng= 30 : angle du chanfrein,
edges= [EDGE_Top, EDGE_Bot] : arêtes du trou à chanfreiner de base [EDGE_Top, EDGE_Bot], si un seul chanfrein donner le vecteur [EDGE_*],
H= 0 : hauteur du cylindre de "pré-perçage", de base à 0)

Exemples :

1. Perce avec un différence de diamètre (deltaD= 0.02) sur la Droite d'un cube(taille= [1, 2, 2]) un cylindre(D= 1, h=1, fn= 50) chanfreiné seulement au Dessus (Dessus auquel on a appliqué une rotation sur la droite) à 30° sur 0.1.
2. Perce avec un différence de diamètre (deltaD= 0.02) sur la Droite d'un cube(taille= [1, 2, 2]) un cylindre(D= 1, h=1, fn= 4) chanfreiné seulement au Dessus (Dessus auquel on a appliqué une rotation sur la droite) à 60° sur 0.1.
3. Perce avec un différence de diamètre (deltaD= 0.02) sur la Droite d'un cube(taille= [1, 2, 2]) un cylindre(D= 1, h=1, fn= 6) chanfreiné seulement en Dessous (Dessous auquel on a appliqué une rotation sur la droite) à 30° sur 0.1.



Code :

```
// Exemple 1 :
cylindricalAxleHole(pos= [0.5, 0, 0], Daxe= 1, deltaD= 0.02, chamfer=
true, rot= ROT_Rgt, edges= [EDGE_Top])
   cube(size= [1, 2, 2], center= true);

// Exemple 2 :
cylindricalAxleHole(pos= [0.5, 0, 0], Daxe= 1, deltaD= 0.02, fn= 4,
chamfer= true, chamferAng= 60, rot= ROT_Rgt + [0, 0, 45], edges=
[EDGE_Top])
   cube(size= [1, 2, 2], center= true);

// Exemple 3 :
cylindricalAxleHole(pos= [0.5, 0, 0], Daxe= 1, deltaD= 0.02, fn= 6,
chamfer= true, rot= ROT_Rgt + [0, 0, 30], edges= [EDGE_Bot])
   cube(size= [1, 2, 2], center= true);
```

Matrix

Dépendances

```
use<Basics.scad>
include<Constants.scad>
```

Présentation

Matrix contient les fonctions permettant de créer différentes matrices de transformation linéaire. Elles sont utilisées pour de nombreuses fonctions et modules de cette bibliothèque. Notez que la dernière ligne n'est pas nécessaire pour le bon fonctionnement avec la fonction Openscad ***multmatrix()***, mais le 1 dans la dernière case de la matrice ne doit pas changer si vous modifiez cette dernière.

$$\begin{bmatrix} EchelleX & CisaillementX\,LeLongDeY & CisaillementX\,LeLongDeZ & TranslationX \\ CisaillementY\,LeLongDeX & EchelleY & CisaillementY\,LeLongDeZ & TranslationY \\ CisaillementZ\,LeLongDeX & CisaillementZ\,LeLongDeY & EchelleZ & TranslationZ \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

---

matrix, fonction

Description :

Retourne la matrice de transformation linéaire associée. Les paramètres scalX, scalY, scalZ sont protégés et donc prendront toujours la valeur 1 si la valeur passée en paramètre vaut 0 ou est indéfinie.

matrix(

scalX= 1 : Échelle suivant X,
scalY= 1 : Échelle suivant Y,
scalZ= 1 : Échelle suivant Z,
transX= 0 : Translation suivant X,
transY= 0 : Translation suivant Y,
transZ= 0 : Translation suivant Z,
shYalX= 0 : Cisaillement Y le long de X,
shZalX= 0 : Cisaillement Z le long de X,
shXalY= 0 : Cisaillement X le long de Y,
shZalY= 0 : Cisaillement Z le long de Y,
shXalZ= 0 : Cisaillement X le long de Z,
shYalZ= 0 : Cisaillement Y le long de Z)

Valeur de retour :

Matrice de transformation linéaire associée.

---

matrix, fonction

Description :

Retourne la matrice de transformation linéaire associée. Le paramètre scal= [X, Y, Z] est protégé et donc prendra toujours la valeur 1 si l'une des valeurs passée en paramètre vaut 0 ou est indéfinie.

matrix(

scal= [1, 1, 1] : Échelles suivant le vecteur [X, Y, Z],
trans= [0, 0, 0] : Translations suivant le vecteur [X, Y, Z],
shYalX= 0 : Cisaillement Y le long de X,
shZalX= 0 : Cisaillement Z le long de X,
shXalY= 0 : Cisaillement X le long de Y,
shZalY= 0 : Cisaillement Z le long de Y,
shXalZ= 0 : Cisaillement X le long de Z,
shYalZ= 0 : Cisaillement Y le long de Z)

Valeur de retour :

Matrice de transformation linéaire associée.

---

matRotX, fonction

Description :

Retourne la matrice de transformation linéaire associée à la Rotation suivant l'axe X. Notez que l'angle est en degré.

matRotX(**ang= 0** : angle de rotation suivant X)

Valeur de retour :

Matrice de transformation linéaire associée à la rotation suivant X.

---

matRotY, fonction

Description :

Retourne la matrice de transformation linéaire associée à la Rotation suivant l'axe Y. Notez que l'angle est en degré.

matRotY(**ang= 0** : angle de rotation suivant Y)

Valeur de retour :

Matrice de transformation linéaire associée à la rotation suivant Y.

---

matRotZ, fonction

Description :

Retourne la matrice de transformation linéaire associée à la Rotation suivant l'axe Z. Notez que l'angle est en degré.

matRotZ(**ang= 0** : angle de rotation suivant Z)

Valeur de retour :

Matrice de transformation linéaire associée à la rotation suivant Z.

---

matRot, fonction

Description :

Retourne la matrice de transformation linéaire associée à la Rotation suivant le vecteur [angX, angY, angZ]. Notez que les angles sont en degré.

matRot(**ang= [0, 0, 0]** : angles de rotation suivant le vecteur [angX, angY, angZ])

Valeur de retour :

Matrice de transformation linéaire associée à la rotation suivant X.

---

matTrans, fonction

Description :

Retourne la matrice de transformation linéaire associée à la Translation suivant le vecteur [X, Y, Z].

matTrans(**v= [0, 0, 0]** : vecteur de translation suivant [X, Y, Z])

Valeur de retour :

Matrice de transformation linéaire associée à la translation suivant le vecteur [X, Y, Z].

---

matScale, fonction

Description :

Retourne la matrice de transformation linéaire associée à la mise à l'échelle suivant le vecteur [X, Y, Z]. Le vecteur est protégé et donc ses valeurs prendront toujours la valeur 1 si celles passées en paramètre valent 0 ou sont indéfinies.

matScale(**v= [1, 1, 1]** : mise à l'échelle suivant le vecteur [X, Y, Z])

Valeur de retour :

Matrice de transformation linéaire associée à la mise à l'échelle suivant le vecteur [X, Y, Z].

***scaleEdge*, fonction**

Description :

Retourne la matrice de transformation linéaire associée à la Translation d'une arête par la multiplication avec un scalaire. Cette fonction est utilisée pour la réalisation de modules complexes. Peut aussi servir si vous souhaitez appliquer un scalaire à une matrice de translation.

Attention : cette fonction n'est pas prévue pour fonctionner sans un des deux paramètres. Aucun message d'avertissement programmé au préalable n'apparaîtra en cas de mauvaise utilisation de la fonction.

scaleEdge(

k : scalaire à appliquer à la matrice,

e : matrice linéaire de transformation)

Valeur de retour :

Matrice de transformation linéaire associée à la Translation d'une arête multipliée par un scalaire.

RenardSeries

Dépendances

```
use<Basics.scad>
```

Présentation

RenardSeries contient une fonction permettant d'obtenir les valeurs des séries de Renard.

---

normalNumberSerie, fonction

Description :

Calcule la n-ième valeur de la série de Renard.

narmalNumberSerie(

R : numéro de série,
n : indice à calculer,
B : 1, 10 ou 100, correspond au dimension allant de 1 à 10, 10 à 100 et 100 à 500)

Valeur de retour :

La n-ième valeur de la série de Renard.

---

renardSerie, fonction

Description :

Retourne la n-ième valeur de la série de Renard.

renardSerie(

R : numéro de série,
n : indice à calculer,
B : 1, 10 ou 100, correspond au dimension allant de 1 à 10, 10 à 100 et 100 à 500)

Valeur de retour :

La n-ième valeur de la série de Renard.

Spring

Dépendances

```
use<Bezier.scad>
use<Transforms.scad>
use<Basics.scad>
use<Vector.scad>
include<Constants.scad>
```

Présentation

Spring regroupe les modules permettant de créer des ressorts de compression et des ressorts en spirale.

---

helicoid, module

Description :

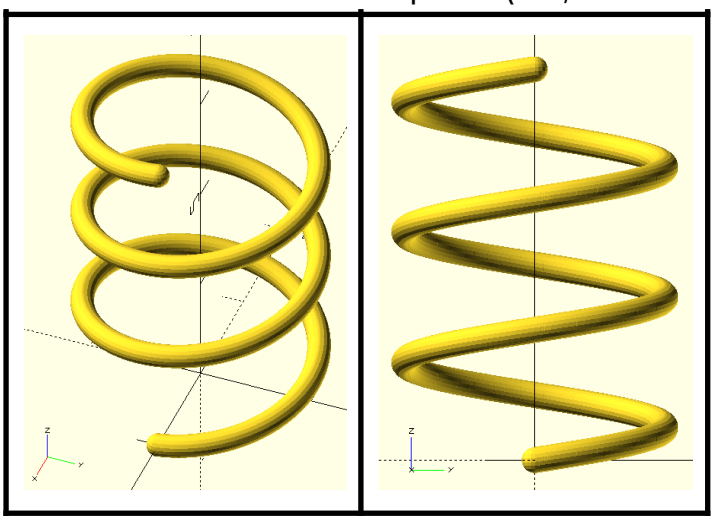
Créer une hélice du module passé en paramètre.

helicoid(

A= [1, 0, 0] : point de départ de l'hélice, appartient au cercle de rayon r,
r= 1 : rayon de l'hélice,
nbTurn= 1 : nombre de révolution,
p= 1 : pas de révolution,
fa= 1 : précision angulaire,
pos= [0, 0, 0] : position finale,
rot= ROT_Top : rotation à appliquer à l'hélicoïde, constante ROT_* ou un vecteur correspondant à [rotX, rotY, rotZ])

Exemple :

Créer l'hélice d'une sphère (r= 1, nbTurn= 3).



Code :

```
helicoid(r= 1, nbTurn= 3, p= 1)
      sphere(0.1, $fn= 20);
```

---

circularCompressionSpring, module

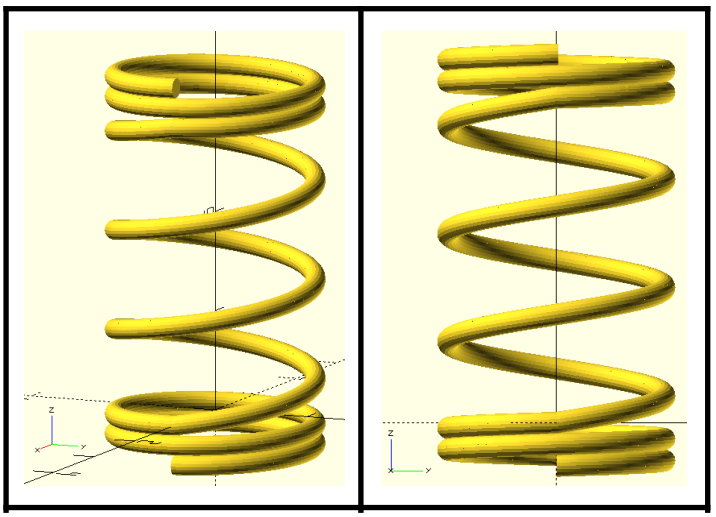
Description :

Créer un ressort de compression.

circularCompressionSpring(

A= [0, 0, 0] : point de départ de l'hélice, appartient au cercle de rayon r,
nbTurn= 1 : nombre de révolution,
r= undef : rayon de base du ressort,
p= undef : pas de révolution,
fn= 20 : nombre de segment pour représenter chaque quart de ressort,
startEnd= false : si true, ajoute le début et la fin du ressort,
nbTurn= 1 : sens de rotation du ressort CLOCKWIRE ou ANTICLOCKWIRE,
pos= [0, 0, 0] : position finale,
rot= ROT_Top : rotation à appliquer au ressort, constante ROT_* ou un vecteur correspondant à [rotX, rotY, rotZ])

Exemple :



Code :

```
circularCompressionSpring(A= [1, 0, 0], r= 0.1, nbTurn= 3, nbTurnStart=
2, p= 1, fa= 1);
```

---

spiralSpring, module

Description :

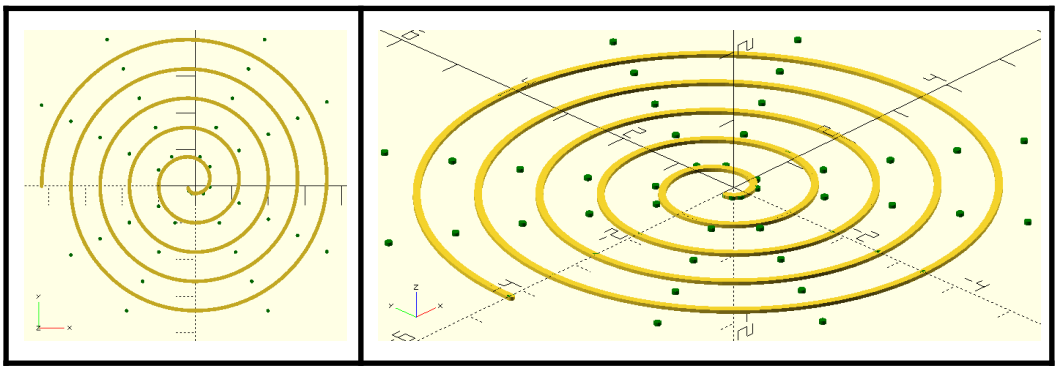
Créer un ressort en spirale. il faut définir soit le rayon départ de la spirale r, soit le pas p.

spiralSpring(

A= [1, 0, 0] : point de départ du ressort, appartient au cercle de rayon r,
nbTurn= 1 : nombre de tours pour la spirale,
r= undef : rayon de départ de la spirale,
p= undef : pas de la spirale,
fn= 20 : nombre de segment pour représenter un quart de spirale,
pos= [0, 0, 0] : position finale,
rot= ROT_Top : rotation à appliquer au ressort, constante ROT_* ou un vecteur correspondant à [rotX, rotY, rotZ])

Exemple :

Créer un ressort en spirale(r= 0.2, nbTurn= 5, direction= ANTICLOCKWIRE) d'un cylindre.



Thread

Dépendances

```
use<Transforms.scad>
use<Basics.scad>
use<Matrix.scad>
include<Constants.scad>
```

Présentation

Thread contient les fonctions de création de filetages et de taraudages ISO Triangulaire et Trapézoïdaux

---

ISOTraingularThread, module

Description :

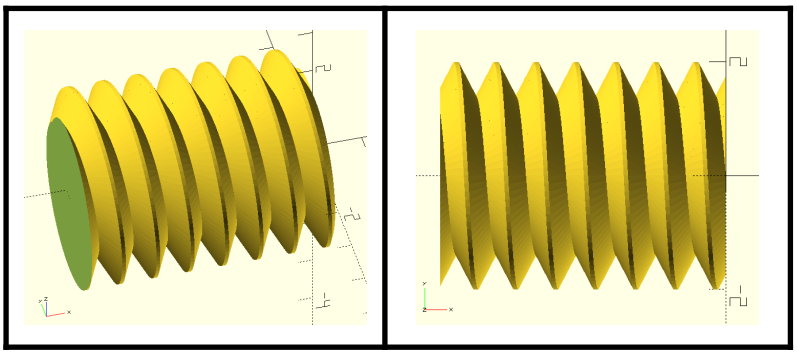
Créer un filetage ISO Triangulaire à la taille souhaitée.

ISOTriangularThread(

D= 1 : diamètre nominal du filetage,
p= 0.1 : pas,
h= 1 : hauteur du filetage,
fa= 1 : précision angulaire de révolution,
pos= [0, 0, 0] : position du filetage,
rot= ROT_Top : rotation à appliquer au filetage, constante ROT_* ou un vecteur correspondant à [rotX, rotY, rotZ],
gap= 0 : jeu pour ajuster le filetage, (+ augmente le diamètre, - le diminue),
center= false : centre ou non le filetage)

Exemple :

Créer un filetage (D= 4, p= 0.7, h= 5, fa= 4) orienté sur la gauche.



Code :

```
ISOTriangularThread(D= 4, p= 0.7, h= 5, fa= 4, rot= ROT_Lft);
```

ISOTraingularThreadTap, module

Description :

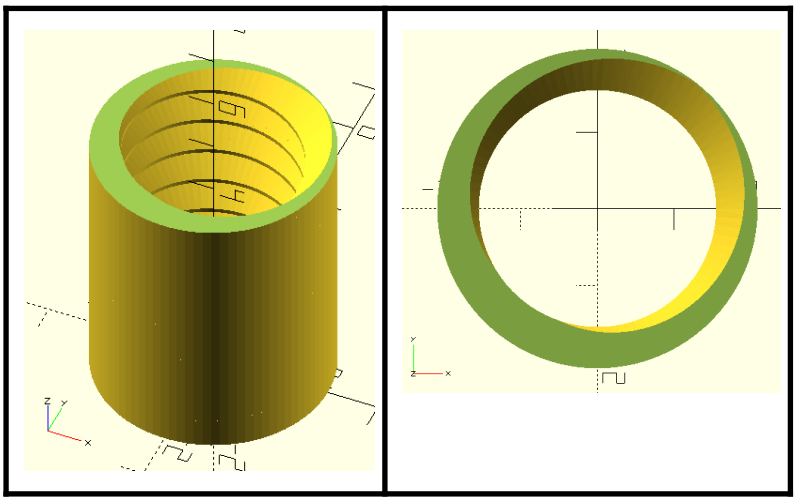
Créer un taraudage ISO Triangulaire à la taille souhaitée.

ISOTriangularThreadTap(

D= 1 : diamètre nominal du taraudage,
p= 0.1 : pas,
h= 1 : hauteur du taraudage,
fa= 1 : précision angulaire de révolution,
pos= [0, 0, 0] : position du taraudage,
rot= ROT_Top : rotation à appliquer au taraudage, constante ROT_* ou un vecteur correspondant à [rotX, rotY, rotZ],
gap= 0 : jeu pour ajuster le taraudage, (+ augmente le diamètre, - le diminue),
center= false : centre ou non le filetage)

Exemple :

Créer un taraudage (D= 4, p= 0.7, h= 5, fa= 4).



Code :

```
ISOTriangularThreadTap(D= 4, p= 0.7, h= 5, fa= 4);
```

---

trapezoidalThread, module

Description :

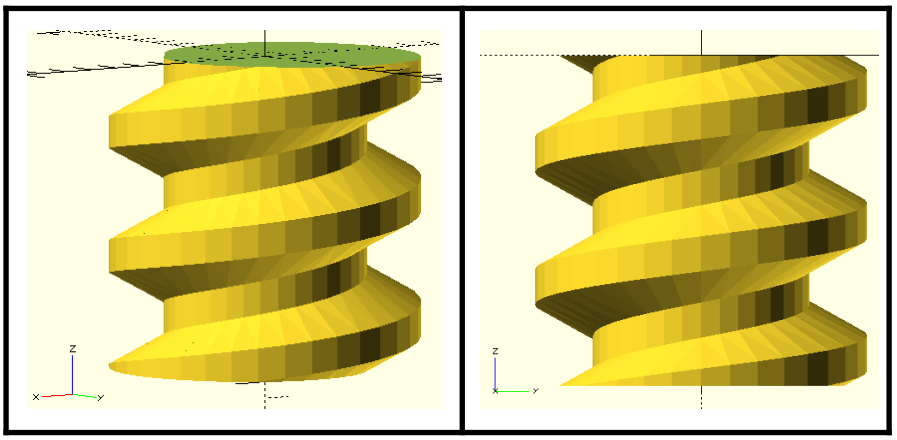
Créer un filetage trapézoïdal à la taille souhaitée.

trapezoidalThread(

D= 1 : diamètre nominal du filetage,
p= 0.12 : pas,
h= 1 : hauteur du filetage,
fa= 1 : précision angulaire de révolution,
pos= [0, 0, 0] : position du filetage,
rot= ROT_Top : rotation à appliquer au filetage, constante ROT_* ou un vecteur correspondant à [rotX, rotY, rotZ],
gap= 0 : jeu pour ajuster le filetage, (+ augmente le diamètre, - le diminue),
center= false : centre ou non le filetage)

Exemple :

Créer un filetage (D= 1, p= 0.12, h= 1, fa= 10) orienté vers le bas.



Code :

```
trapezoidalThread(p = 0.4, fa= 10, rot= ROT_Bot);
```

***trapezoidalThreadTap*, module**

Description :

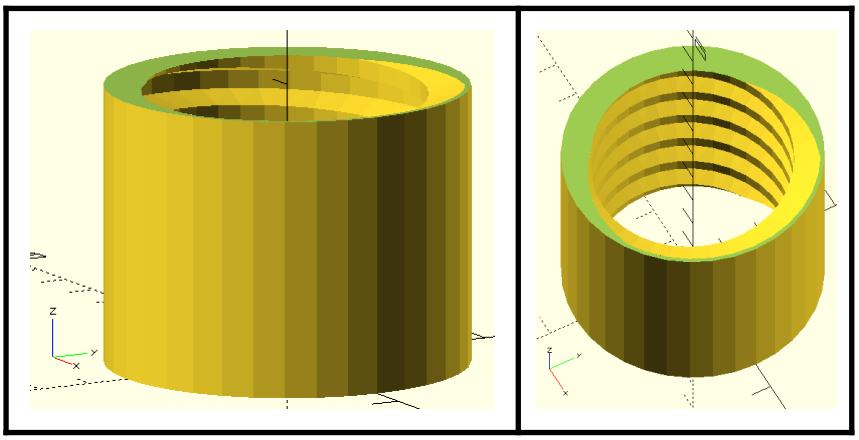
Créer un taraudage trapézoïdal à la taille souhaitée.

trapezoidalThreadTap(

D= 1 : diamètre nominal du taraudage,
p= 0.1 : pas,
h= 1 : hauteur du taraudage,
fa= 1 : précision angulaire de révolution,
pos= [0, 0, 0] : position du taraudage,
rot= ROT_Top : rotation à appliquer au taraudage, constante ROT_* ou un vecteur correspondant à [rotX, rotY, rotZ],
gap= 0 : jeu pour ajuster le taraudage, (+ augmente le diamètre, - le diminue),
center= false : centre ou non le filetage)

Exemple :

Créer un taraudage (D= 6, p= 1, h= 5, fa= 10).



Code :

```
trapezoidalThreadTap(D= 6, p=1, h=5, fa= 10);
```

***knurling*, module**

Description :

Créer un moletage horizontal, vertical ou en diamant à la taille souhaitée.

Attention :

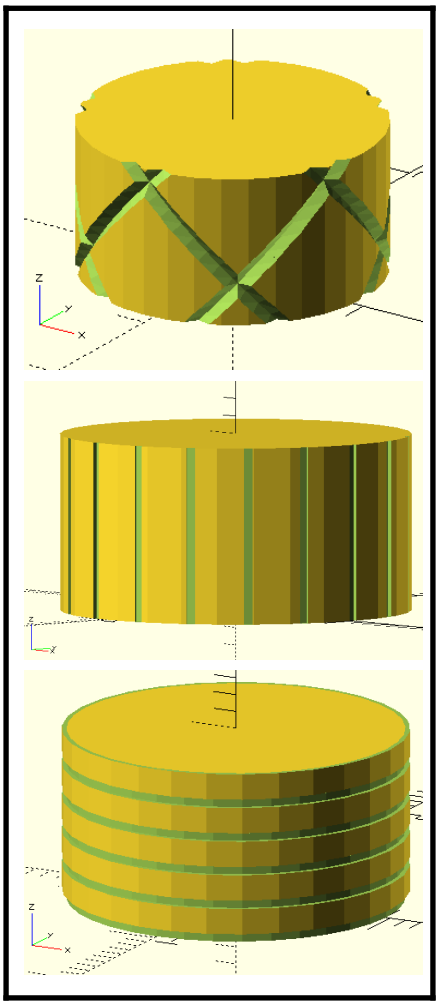
- La forme du moletage et la pièce à moleter doivent être placées de façon à ce que les parties inférieures soient sur le plan XY orienté vers Z+.
- Le pas (p) est utilisé lorsque l'orientation est définie. Il est utilisé pour définir la distance entre deux rainures HORIZONTALES.
- Le nombre de modules (moduleNb) est utilisé lorsque ang ou orient(= VERTICAL) sont définis pour déterminer le nombre de répétitions circulaires à effectuer.
- Si vous souhaitez réaliser un moletage vertical d'une longueur précise, pensez à soustraire la hauteur de la pièce de moletage de la hauteur totale. La première pièce a été placée sur l'axe X avec une translation de r, l'autre sera placée selon une rotation Z anti-blocage de la première pièce.

knurling(

r= 1 : diamètre de la pièce à moleter,
h= 1 : hauteur du moletage,
p= 0.1 : pas seulement si orient est défini,
moduleNb= 4 : si ang est défini, représente le nombre de rainures du moletage (utilisé uniquement avec ang et orient= VERTICAL),
ang= undef : si défini représente l'angle du moletage en diamant [20, 60]°,
orient= undef : si défini avec les constantes VERTICAL ou HORIZONTAL, réalise le moletage associé,
fa= 1 : précision angulaire de révolution,
pos= [0, 0, 0] : position du taraudage,
rot= ROT_Top : rotation à appliquer au taraudage, constante ROT_* ou un vecteur correspondant à [rotX, rotY, rotZ])

Exemple :

1. Créer un moletage en diamant avec une angle de 45°.
2. Créer un moletage vertical avec 10 modules.
3. Créer un moletage horizontal avec un pas de 0.2.



Code :

```
// Exemple 1
knurling(h= 1.2, ang= 45, moduleNb= 10, p = 0.4, fa= 5){

   mTranslate([0, 0, 0.05])
       cylinder(r= 1, h=1, $fn= 360/10);

   mTranslate([0.01, 0, 0])
       rotZ(90)
           rotX(45)
               cube([0.001, 0.1, 0.1], center= true);
}

// Exemple 2
knurling(r= 1, h= 1 - 0.05, p= 0.2, orient= VERTICAL, moduleNb= 19, fa=
10){

   cylinder(r= 1, $fn= 30);
   mTranslate([-0.025, -0.025, 0]) cube(0.05);
}

//Exemple 3
knurling(r= 1, h= 1, p= 0.2, orient= HORIZONTAL, moduleNb= 6, fa= 10){

   cylinder(r= 1, $fn= 30);
   mTranslate([-0.025, -0.025, 0]) cube(0.05);
}
```

Transforms

Dépendances

```
include<Constants.scad>
use<Basics.scad>
use<Matrix.scad>
```

Présentation

Transforms contient les modules de transformation basiques pouvant-être appliqués à un autre module. Ils sont réalisés à partir des matrices de transformation linéaires (cd Matrix.scad).

---

rotX, module

Description :

Applique une Rotation suivant l'axe X aux modules suivant son appel. Notez que l'angle est en degré.

rotX(**ang= 0** : angle de la rotation suivant X)

Valeur de retour :

Tourne suivant X les modules suivant son appel.

---

rotY, module

Description :

Applique une Rotation suivant l'axe Y aux modules suivant son appel. Notez que l'angle est en degré.

rotY(**ang= 0** : angle de la rotation suivant Y)

Valeur de retour :

Tourne suivant Y les modules suivant son appel.

---

rotZ, module

Description :

Applique une Rotation suivant l'axe Z aux modules suivant son appel. Notez que l'angle est en degré.

rotZ(**ang= 0** : angle de la rotation suivant Z)

Valeur de retour :

Tourne suivant Z les modules suivant son appel.

---

mRotate, module

Description :

Applique les Rotations suivant le vecteur [angX, angY, angZ] aux modules suivant son appel. Notez que les angles sont en degré.

mRotate(**ang= [0, 0, 0]** : angle de la rotation suivant le vecteur [angX, angY, angZ])

Valeur de retour :

Tourne suivant le vecteur [angX, angY, angZ] les modules suivant son appel.

---

mScale, module

Description :

Applique les mises à l'échelle suivant le vecteur [X, Y, Z] aux modules suivant son appel. Notez que les valeurs du vecteur qui valent 0 ou indéfinies seront remplacées

mTranslate(**k= [1, 1, 1]** : mise à l'échelle suivant le vecteur [X, Y, Z])

Valeur de retour :

Mise à l'échelle suivant le vecteur [X, Y, Z] des modules suivant son appel.

---

transform, module

Description :

Applique une matrice de transformation linéaire aux modules suivant son appel. Notez que vous pouvez passer en paramètres des opérations de matrices.

transform(**m= matScale(v= [1, 1, 1])** : matrice de transformation linéaire)

Valeur de retour :

Transformation des modules suivant son appel suivant une matrice de transformation linéaire.

Vector

Dépendances

```
use<Basics.scad>
```

Présentation

Vector contient des fonctions mathématiques pour des vecteurs.

---

makeVector, fonction

Description :

Retourne le vecteur mathématique $\overrightarrow{uv}$. Les vecteurs peuvent être de taille quelconque mais identiques.

makeVector(
u= [0, 0, 0] : premier vecteur
v= [0, 0, 0] : second vecteur)

Valeur de retour :

Retourne le vecteur $\overrightarrow{uv}$.

---

middleVector, fonction

Description :

Retourne les coordonnées du milieu de $\overrightarrow{uv}$. Les vecteurs peuvent être de taille quelconque mais identiques.

middleVector(
u= [1, 1, 1] : premier vecteur
v= [1, 1, 1] : second vecteur)

Valeur de retour :

Retourne les coordonnées du milieu de $\overrightarrow{uv}$.

---

mod, fonction

Description :

Retourne le module d'un vecteur 3D.

mod(**v= undef** : vecteur 3D)

Valeur de retour :

Retourne le module du vecteur 3D.

---

angleVector, fonction

Description :

Retourne l'angle orienté des vecteurs (de taille n) $\widehat{uv}$.

angleVector(

u= [1, 0, 0] : premier vecteur

v= [0, 1, 0] : second vecteur)

Valeur de retour :

Retourne l'angle orienté des vecteurs $\widehat{uv}$.

---

matVectRotX, fonction

Description :

Retourne la matrice de rotation suivant l'axe X pour un vecteur 3D. L'ange est en degré et la rotation orientée.

matVectRotX(**ang= 0**: angle de rotation suivant l'axe X)

Valeur de retour :

Retourne la matrice de rotation suivant l'axe X pour un vecteur 3D.

---

applyVectRotX, fonction

Description :

Applique la rotation suivant l'axe X au vecteur 3D. L'ange est en degré et la rotation orientée.

applyVectRotX(

v= [1, 1, 1]: vecteur à tourner

ang= 0: angle de rotation suivant l'axe X)

Valeur de retour :

Applique la rotation suivant l'axe X au vecteur 3D.

---

matVectRotY, fonction

Description :

Retourne la matrice de rotation suivant l'axe Y pour un vecteur 3D. L'ange est en degré et la rotation orientée.

matVectRotY(**ang= 0**: angle de rotation suivant l'axe Y)

Valeur de retour :

Retourne la matrice de rotation suivant l'axe Y pour un vecteur 3D.

---

applyVectRotY, fonction

Description :

Applique la rotation suivant l'axe Y au vecteur 3D. L'ange est en degré et la rotation orientée.

applyVectRotY(

v= [1, 1, 1]: vecteur à tourner

ang= 0: angle de rotation suivant l'axe Y)

Valeur de retour :

Applique la rotation suivant l'axe Y au vecteur 3D.

---

matVectRotZ, fonction

Description :

Retourne la matrice de rotation suivant l'axe Z pour un vecteur 3D. L'ange est en degré et la rotation orientée.

matVectRotZ(**ang= 0**: angle de rotation suivant l'axe Z)

Valeur de retour :

Retourne la matrice de rotation suivant l'axe Z pour un vecteur 3D.

---

applyVectRotZ, fonction

Description :

Applique la rotation suivant l'axe Z au vecteur 3D. L'ange est en degré et la rotation orientée.

applyVectRotZ(
v= [1, 1, 1]: vecteur à tourner
ang= 0: angle de rotation suivant l'axe Z)

Valeur de retour :
Applique la rotation suivant l'axe Z au vecteur 3D.

***matVectRot*, fonction**

Description :
Retourne la matrice de rotation suivant le vecteur [angX, angY, angZ] pour un vecteur 3D. L'ange est en degré et la rotation orientée.

matVectRotX(**ang= [0, 0, 0]**: angle de rotation suivant le vecteur [angX, angY, angZ])

Valeur de retour :
Retourne la matrice de rotation suivant le vecteur [angX, angY, angZ] pour un vecteur 3D.

***applyVectRot*, fonction**

Description :
Applique la rotation suivant le vecteur [angX, angY, angZ] au vecteur 3D. L'ange est en degré et la rotation orientée.

applyVectRotX(
v= [1, 1, 1]: vecteur à tourner
ang= [0, 0, 0]: angle de rotation suivant le vecteur [angX, angY, angZ])

Valeur de retour :
Applique la rotation suivant le vecteur [angX, angY, angZ] au vecteur 3D.
