Module C3

Transformations géométriques avec Python 3
Nouvelle édition

Luc Baron, ing., Ph.D.
Professeur titulaire

6 février 2024

Département de génie mécanique
Polytechnique Montréal

Table des matières

Chapitre 1

Transformations en 2D

Dans ce chapitre, nous introduisons les transformations géométriques de translation, de rotation et d'homothétie en 2D. La bibliothèque `numpy` de Python est utilisée pour manipuler les vecteurs et les matrices.

1.1 Translation et rotation

Dans l'espace Cartésien de dimension 2, la position du point B est définie par le vecteur $\mathbf{b}$ dont les coordonnées (b_x, b_y) sont mesurées selon les axes x et y du référentiel $\mathcal{B}$. Tel que montré à la Fig. 1.1, le référentiel $\mathcal{B}$ est déplacé d'un vecteur $\mathbf{p}$ et tourné d'un angle θ dans le référentiel $\mathcal{A}$.

La position du point B dans $\mathcal{A}$ est donnée par le vecteur $\mathbf{a}$

$$\mathbf{a} = \mathbf{p} + \mathbf{R}\mathbf{b}, \quad \mathbf{a} \equiv [a_x, a_y]^T, \quad \mathbf{b} \equiv [b_x, b_y]^T, \tag{1.1}$$

où $\mathbf{p} = [p_x, p_y]^T$ est le vecteur position de l'origine de $\mathcal{B}$ relativement à l'origine $\mathcal{A}$ exprimé dans $\mathcal{A}$, et $\mathbf{R}$ est la matrice de rotation qui exprime l'orientation de $\mathcal{B}$ relativement à $\mathcal{A}$

$$\mathbf{R} = \begin{bmatrix} c & -s \\ s & c \end{bmatrix}, \quad s \equiv \sin\theta, \quad c \equiv \cos\theta, \tag{1.2}$$

c'est-à-dire la matrice de rotation 2×2 autour de la normale au plan xy dans le sens trigonométrique.

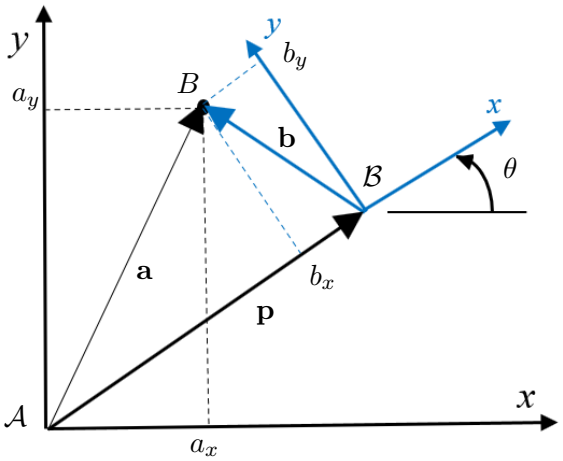


FIGURE 1.1 – Coordonnées du point B dans le référentiel $\mathcal{A}$ et $\mathcal{B}$

Les vecteurs positions $\mathbf{a}_i$ et $\mathbf{b}_i$ des points de l'ensemble $S = \{B_i\}_1^m$ peuvent être assemblés en 2 matrices $2 \times m$

$$\mathbf{A} \equiv [\mathbf{a}_1 \mathbf{a}_2 \cdots \mathbf{a}_m], \quad \mathbf{B} \equiv [\mathbf{b}_1 \mathbf{b}_2 \cdots \mathbf{b}_m], \tag{1.3}$$

afin de réécrire eq.(1.1) comme un système de m équations vectorielles de dimension 2, c'est-à-dire

$$\mathbf{A} = \mathbf{p} + \mathbf{RB} \tag{1.4}$$

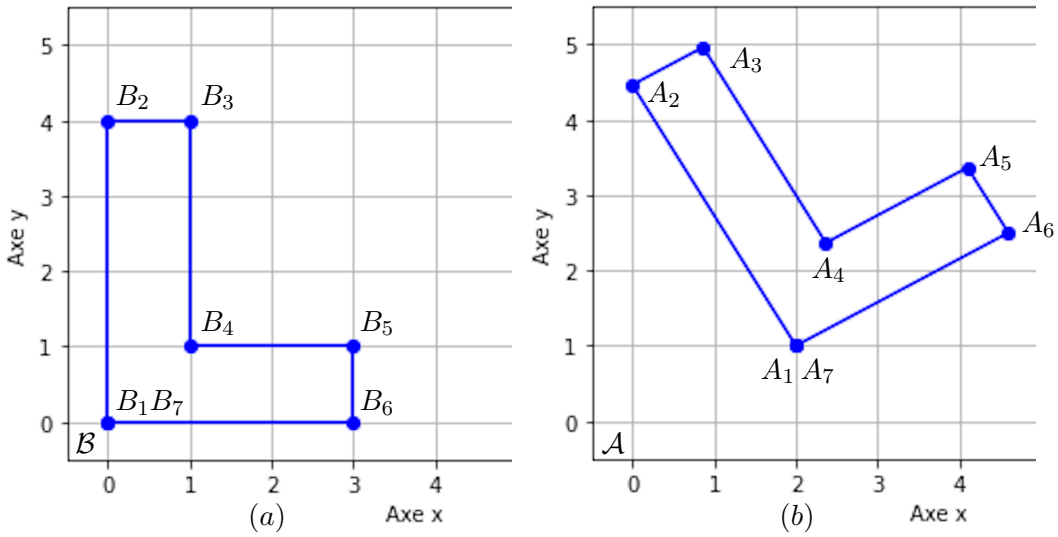


FIGURE 1.2 – Coordonnées des points $\{(B_i)\}_1^7$ dans $\mathcal{B}$ (à gauche) et dans $\mathcal{A}$ (à droite)

Soit l'ensemble S des coordonnées des points B dans $\mathcal{B}$ permettant de tracer le contour fermé de la lettre L montrée à la Fig. 1.2(a)

$$S = \{(0,0), (0,4), (1,4), (1,1), (3,1), (3,0), (0,0)\}_{\mathcal{B}}$$

écrit sous forme matricielle

$$\mathbf{B} = \begin{bmatrix} 0 & 0 & 1 & 1 & 3 & 3 & 0 \\ 0 & 4 & 4 & 1 & 1 & 0 & 0 \end{bmatrix}$$

Le référentiel $\mathcal{B}$ est déplacé dans $\mathcal{A}$ à la coordonnée $(2,1)$ avec une rotation de 30 deg. Ainsi, la position des points B_i est calculée dans $\mathcal{A}$ par

$$\mathbf{A} = \mathbf{p} + \mathbf{RB} = \begin{bmatrix} 2 \\ 1 \end{bmatrix} + \begin{bmatrix} c & -s \\ s & c \end{bmatrix} \begin{bmatrix} 0 & 0 & 1 & 1 & 3 & 3 & 0 \\ 0 & 4 & 4 & 1 & 1 & 0 & 0 \end{bmatrix}$$

avec $s = \sin(30) = 1/2$ et $c = \cos(30) = \sqrt{3}/2 = 0.866$, ce qui donne

$$\mathbf{A} = \begin{bmatrix} 2 & 0.000 & 0.866 & 2.366 & 4.098 & 4.598 & 2 \\ 1 & 4.464 & 4.964 & 2.366 & 3.366 & 2.500 & 1 \end{bmatrix}$$

Le résultat du déplacement du référentiel $\mathcal{B}$ dans $\mathcal{A}$ est montré à la Fig. 1.2(b). Le script 1 définit la fonction `Rot(t)` qui retourne la matrice de rotation d'un angle `t` en degré. Les coordonnées des points B_i sont assignées à la matrice `B` aux lignes 8 et 9. Le vecteur colonne de déplacement `p` est défini à la ligne 10. Les coordonnées des points B_i dans $\mathcal{A}$, c'est-à-dire la matrice `A`, sont calculées à la ligne 11. La matrice de rotation `Rot(30)` est multipliée par la matrice `B`, puis chacune des colonnes est additionnée à `p` pour donner `A`.

SCRIPT 1
Calcul les coordonnées des points B_i suite à un changement de référentiel

```
import numpy as np
def Rot(t):           #Matrice de rotation 2 x 2 en 2D
    t = np.pi*t/180   #Conversion de degre a radian
    s, c = np.sin(t), np.cos(t)
    R = np.array([[c,-s],
                  [s, c]])
    return R
B = np.array([[0, 0, 1, 1, 3, 3, 0],
              [0, 4, 4, 1, 1, 0, 0]])
p = np.array([2,1]).reshape(2,1)  #p doit etre un vecteur colonne
A = p + Rot(30).dot(B)
print('A=',A)
```

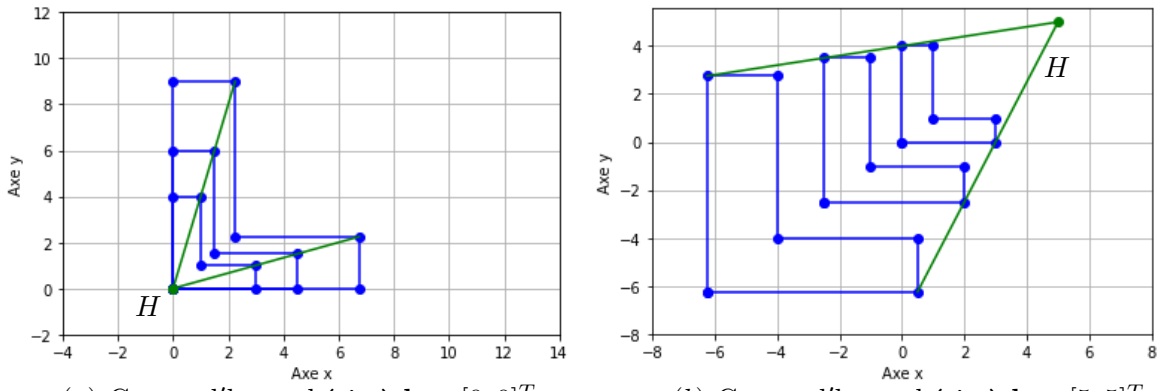
```
A= [[2.000 0.000 0.866 2.366 4.098 4.598 2.000]
 [1.000 4.464 4.964 2.366 3.366 2.500 1.000]]
```

1.2 Homothéties

Une homothétie est une transformation géométrique par agrandissement $f > 1$ ou réduction $f < 1$ qui produit une mise à l'échelle. Elle se caractérise par un centre d'homothétie H invariant, c'est-à-dire qu'il ne bouge pas, et un facteur d'échelle f. Ainsi, les points s'éloignent du centre d'homothétie H lorsque l'on agrandit ou se rapprochent du centre d'homothétie H lorsque l'on réduit. Les coordonnées des points résultants d'une homothétie f de centre H se calculent par

$$\mathbf{A} = f(\mathbf{B} - \mathbf{h}) + \mathbf{h} = f\mathbf{B} + (1 - f)\mathbf{h} \tag{1.5}$$

où $\mathbf{h}$ est la position du centre d'homothétie H et $f > 0$.



(a) Centre d'homothétie à $\mathbf{h} = [0, 0]^T$ (b) Centre d'homothétie à $\mathbf{h} = [5, 5]^T$

FIGURE 1.3 – Homothéties récurrentes de la lettre L avec $f = 1.5$

Le script 2 utilise l'éq.(1.5) pour calculer l'homothétie récurrente des coordonnées de la lettre L pour $f = 1.5$ avec un centre d'homothétie à l'origine $\mathbf{h} = [0, 0]^T$ et à $\mathbf{h} = [5, 5]^T$. La procédure est récurrente parce que la seconde homothétie est appliquée sur le résultat de la première, soit un agrandissement de $f * f = 1.5 * 1.5 = 2.25$. Lorsque $\mathbf{h} = [0, 0]^T$, l'éq.(1.5) se réduit à $\mathbf{A} = f\mathbf{B}$, c'est-à-dire une homothétie de f relativement à l'origine comme montrée à la Fig. 1.3(a).

SCRIPT 2
Homothéties récurrentes de la lettre L avec $f = 1.5$ (répétée 2 fois) de centre $\mathbf{h} = [5, 5]^T$

```
import numpy as np
import matplotlib.pyplot as plt
def Rot(t):           #Matrice de rotation 2 x 2 en 2D
    t = np.pi*t/180   #Conversion de degre a radian
    s, c = np.sin(t), np.cos(t)
    R = np.array([[c,-s],
                  [s, c]])
    return R
A = np.array([[0, 0, 1, 1, 3, 3, 0],
              [0, 4, 4, 1, 1, 0, 0]])
h = np.array([5,5]).reshape(2,1) #Position de centre d'homothetie
f = 1.5                          #Facteur d'echelle
plt.plot(A[0,:],A[1,:],'o-b')
for i in range(1,3):             #Boucle 2 fois
    A = f*A + (1-f)*h            #Homothetie de A avec f et h
    plt.plot(A[0,:],A[1,:],'o-b') #Trace les vecteurs de A
print('A=',A)
```

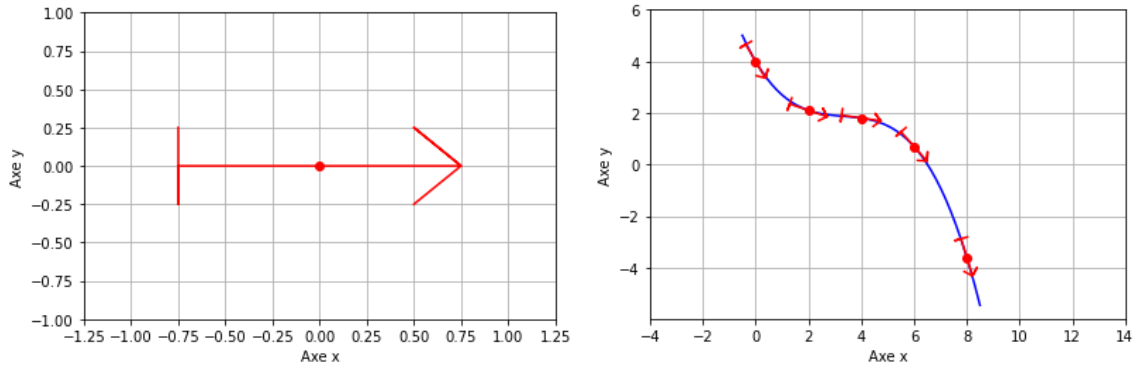
```
A= [[-6.25 -6.25 -4. -4.   0.5  0.5  -6.25]
 [-6.25 2.75  2.75 -4.  -4.   -6.25 -6.25]]
```

Exercice 1 : Tracer un polynôme de degré 3 avec 5 flèches tangentes

Soit le polynôme de degré 3

$$p_3(x) = -1/20\ x^3 + 1/2\ x^2 - 7/4\ x + 4 = y$$

Tracer en bleu le polynôme $p_3(x)$ entre $x = [-0.5, 8.5]$. Ajouter en rouge le symbole flèche tangent à la courbe aux points $x = [0, 2, 4, 6, 8]$, tel que montré à la Fig. 1.4.



(a) Symbole de flèche à tracer (b) Courbe avec 5 symboles tangents

FIGURE 1.4 – Symbole de flèche et courbe avec 5 symboles tangents

Solution :

Dans le référentiel de la flèche, les coordonnées des points sont relevées directement à partir de la

Fig. 1.4(a). Il est plus facile de faire une seule séquence de 7 points, soit

$$\mathbf{B} = \begin{bmatrix} -0.75 & -0.75 & -0.75 & 0.75 & 0.50 & 0.75 & 0.50 \\ 0.25 & -0.25 & 0.00 & 0.00 & 0.25 & 0.00 & -0.25 \end{bmatrix} \tag{1.6}$$

Aucun changement d'échelle n'est requis. Seul une rotation de la flèche est nécessaire pour obtenir les tangences à la courbe. Sachant que l'angle d'inclinaison θ est relié à la la pente, nous avons

$$\tan\theta = \frac{dy}{dx} = \frac{d}{dx}p_3(x) = -(3/20)\ x^2 + (2/2)\ x - (7/4)$$

L'angle d'inclinaison θ est donné par

$$\theta = \arctan(-(3/20)\ x^2 + x - (7/4))$$

Le centre de la flèche est l'origine du référentiel $\mathcal{B}$. Insérer l'origine du référentiel $\mathcal{B}$ à un point de la courbe permet de tracer la flèche centrée sur ce point. La rotation de $\mathcal{B}$ autour de son origine permet d'aligner l'axe x (la flèche) selon l'inclinaison θ de la courbe. La fonction `Symbole(x,y,t)` trace la flèche centrée sur (x, y) avec une inclinaison θ. Les coordonnées des points de la flèche se calculent par

$$\mathbf{A} = \mathbf{p} + \mathbf{RB}$$

où $\mathbf{R} = \text{Rot}(\theta)$ est la matrice de rotation d'un angle θ.

SCRIPT 3
Exercice 1 : Trace le polynôme de degré 3 avec 5 flèches tangentes

```
import numpy as np
import matplotlib.pyplot as plt
def Rot(t):           #Matrice de rotation en 2D avec t en radian
    s, c = np.sin(t), np.cos(t)
    R = np.array([[c,-s],[s, c]])
    return R

def Symbole(x,y,t):   #Trace un symbole a (x,y) avec un angle t
    B = np.array([[-0.75, -0.75, -0.75, 0.75, 0.5, 0.75, 0.5],
                  [ 0.25, -0.25,    0,   0,  0.25,   0, -0.25]])
    p = np.array([x,y]).reshape(2,1)
    A = p + Rot(t).dot(B)
    plt.plot(x,y,'or',A[0,:],A[1,:],'-r')

a = np.array([-0.05, 0.5, -1.75, 4]) #Coefficients du polynome de degre 3
x = np.linspace(-0.5,8.5,50)
y = np.polyval(a,x)
plt.plot(x,y,'-b')                   #Tracer le polynome
xp = np.arange(0,10,2)               #Points xp = 0, 2, 4, 6, 8
yp = np.polyval(a,xp)
for i in range(5):                   #Pour chacun des xp, yp
    t = np.arctan(-(3/20) * xp[i]**2 + xp[i] - (7/4)) #Calcul l'angle t
    Symbole(xp[i],yp[i],t)
plt.show()
```

1.3 Formulation transposée

La solution des problèmes de transformation géométrique en 2D se calcule par

$$\mathbf{A} = \mathbf{p} + \mathbf{RB} \tag{1.7}$$

où $\mathbf{A}$ et $\mathbf{B}$ sont des matrices de 2 lignes par m colonnes. Cette formulation (horizontale) utilise donc des matrices $\mathbf{A}$ et $\mathbf{B}$ sous forme de tableaux de plusieurs colonnes. Il est possible d'utiliser une formulation, dite verticale, sous forme transposée du problème, afin d'utiliser des tableaux de plusieurs lignes plutôt que plusieurs colonnes. La transposée de l'éq.(1.7) est

$$\mathbf{A}^T = \mathbf{p}^T + \mathbf{B}^T\mathbf{R}^T \tag{1.8}$$

où $\mathbf{A}^T$ et $\mathbf{B}^T$ sont des matrices de m lignes par 2 colonnes. Cette formulation verticale utilise donc des matrices $\mathbf{A}^T$ et $\mathbf{B}^T$ sous forme de tableaux de plusieurs lignes. La matrice de rotation transposée est postmultipliée plutôt que prémultipliée. Il est parfois plus facile de traiter certains problèmes de transformation géométrique avec cette formulation transposée (verticale) comme nous le ferons au chapitre suivant.

En général, l'éq.(1.7) s'écrit en *formulation transposée* par

$$\mathbf{U} = \mathbf{p}^T + \mathbf{VR}^T, \tag{1.9}$$

où $\mathbf{U} \equiv \mathbf{A}^T$ et $\mathbf{V} \equiv \mathbf{B}^T$ sont des matrices verticales $m \times 2$ des coordonnées des points dans les référentiels $\mathcal{A}$ et $\mathcal{B}$, respectivement, et $\mathbf{R}^T$ la matrice de rotation transposée qui postmultiplie $\mathbf{V}$.

Rappels :

La transposé d'une somme de matrices est la somme des matrices transposées.

$$(\mathbf{U} + \mathbf{V})^T = \mathbf{U}^T + \mathbf{V}^T \tag{1.10}$$

La transposé d'un produit de matrices est le produit des matrices transposées dans l'ordre inverse.

$$(\mathbf{UV})^T = \mathbf{V}^T\mathbf{U}^T \tag{1.11}$$

Une matrice de rotation est orthogonale, son inverse est sa transposées.

$$\mathbf{R}^{-1} = \mathbf{R}^T \quad \rightarrow \quad \mathbf{R}^{-1}\mathbf{R} = \mathbf{R}^T\mathbf{R} = \mathbf{1} \text{ et } \det(\mathbf{R}) = 1 \quad \forall\theta \tag{1.12}$$

Chapitre 2

Transformations en 3D

Dans ce chapitre, nous utilisons les transformations géométriques de translation, de rotation et d'homothétie avec la bibliothèque `numpy` de Python pour manipuler la position des vertex d'objets 3D. Nous utilisons la bibliothèque `MEC1315_STL.py` pour lire et écrire des fichiers STL.

2.1 Objet STL

Le format de fichier STL, initialement introduit pour la stéréolithographie (STL pour STereo-Lithography) est maintenant largement utilisé pour la simulation de la production, la numérisation et l'impression 3D, ainsi qu'en animation 3D. Ce format décrit la géométrie de surface d'un objet en 3D par une série de triangles interconnectés, appelées facettes. Chaque facette est définie par une normale orientée vers l'extérieur de l'objet et 3 sommets, appelés vertex. Le sens de la numérotation des vertex donne la normale selon la règle de la main droite. Cette règle est utilisée par plusieurs logiciels pour recalculer les normales suite à un étirement non homogène ou une simple rotation. Les fichiers STL peuvent être enregistrés selon deux types de format : texte (code ASCII) ou binaire. Les fichiers textes sont faciles à traiter avec n'importe quel langage de programmation, mais peuvent être très volumineux si la géométrie de l'objet est complexe. Les fichiers binaires sont beaucoup moins volumineux, et donc,

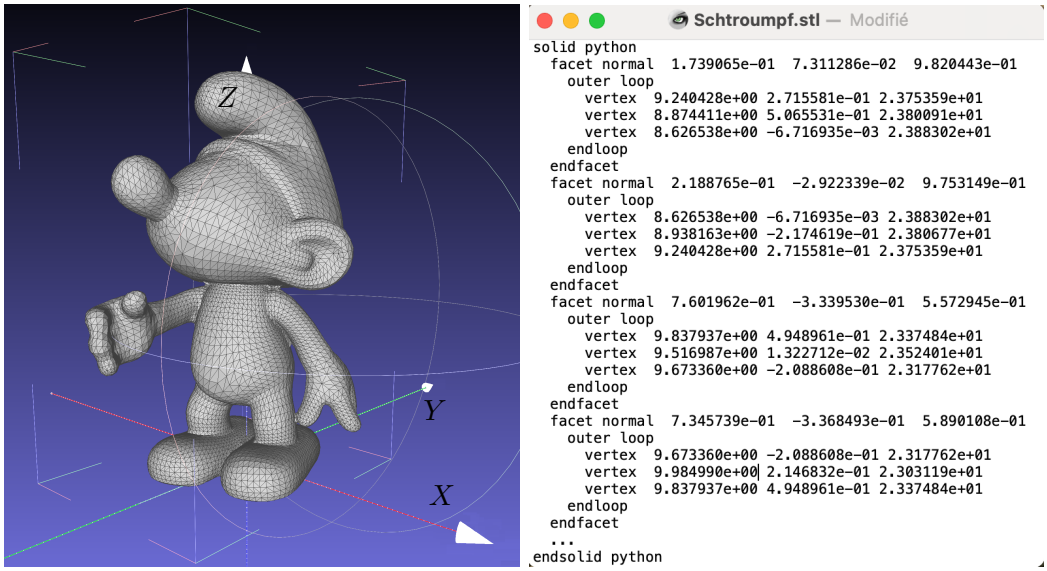
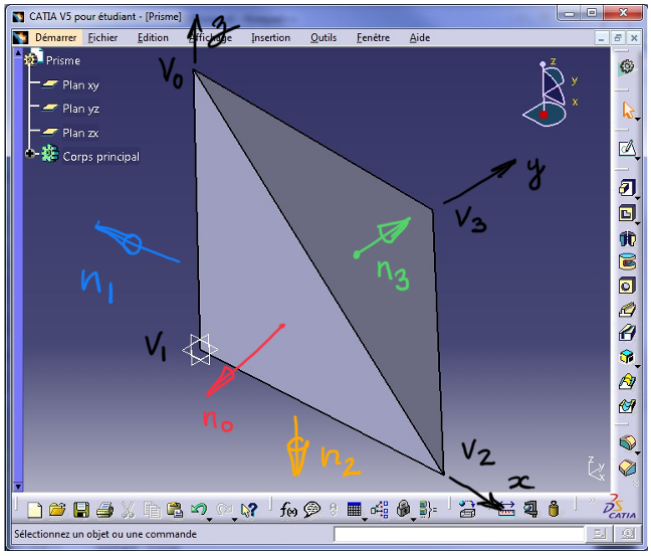


FIGURE 2.1 – Affichage des 18246 facettes et 9121 vertex du fichier `Schtroumpf.stl`

sont les plus utilisés. De plus, ils permettent de contenir un code de couleur RGB de 16 bits associé à chaque facette. Un élément additionnel utile à plusieurs applications.




```
solid CATIA STL
  facet normal  0.0 -1.0  0.0   %f0 et n0
    outer loop
      vertex    0.0  0.0  1.0   %v0
      vertex    0.0  0.0  0.0   %v1
      vertex    1.0  0.0  0.0   %v2
    endloop
  endfacet
  facet normal -1.0  0.0  0.0   %f1 et n1
    outer loop
      vertex    0.0  1.0  0.0   %v3
      vertex    0.0  0.0  0.0   % -> v1
      vertex    0.0  0.0  1.0   % -> v0
    endloop
  endfacet
  facet normal  0.0 0.0 -1.0    %f2 et n2
    outer loop
      vertex    1.0  0.0  0.0   % -> v2
      vertex    0.0  0.0  0.0   % -> v1
      vertex    0.0  1.0  0.0   % -> v3
    endloop
  endfacet
  facet normal  0.5773503  0.5773503  0.5773503 %f3 et n3
    outer loop
      vertex    0.0  0.0  1.0   % -> v0
      vertex    1.0  0.0  0.0   % -> v2
      vertex    0.0  1.0  0.0   % -> v3
    endloop
  endfacet
endsolid CATIA STL
```

FIGURE 2.2 – Affichage des 4 facettes et 4 vertex d'un prisme et le contenu du fichier `Prisme.stl`

La Fig. 2.2 illustre les 4 facettes et 4 vertex d'un prisme créé par CATIA, ainsi que le contenu du fichier STL correspondant. Notons la présence des commentaires (identifié par le symbole %) à la fin de certaines lignes. Chaque facette est définie par un bloc débutant par `facet` et se terminant par `endfacet`. Il contient un vecteur normal à la facette, ainsi que 3 vecteurs positions des vertex. Le fichier `Prisme.stl` contient 4 facettes (et donc 4 normales) et 12 vertex (4 facette x 3 vertex/facette). Cependant, chaque facette a toujours 2 vertex partagés avec une facette juxtaposée. Ainsi, il est d'usage de retirer, à la lecture du fichier STL, la répétition des vertex. Pour le prisme, seuls les vertex de `v0` à `v3` sont nécessaires pour positionner les facettes de `f0` à `f3` de normales `n0` à `n3`.

La Fig. 2.3 présente la structure des matrices **F**, **V** et **N** une fois les répétitions de vertex retirées. Les matrices **F** et **N** sont de dimension `nf x 3` où `nf` est le nombre de facettes. La ligne i de **N** est la normale de la facette i de **F**. La matrice **V** est de dimension `nv x 3` où `nv` est le nombre de vertex. Les matrices **V** et **N** contiennent les coordonnées x (colonne 0), y (colonne 1) et z (colonne 2) des vertex et des normales. La matrice **F** ne contient pas de coordonnées, mais plutôt les numéros de ligne de **V** où se trouvent les coordonnées des vertex. En résumé, la facette `F[i,:]` a la normale `N[i,:]` et les numéros de vertex `a=F[i,0]`, `b=F[i,1]` et `c=F[i,2]`, dont les coordonnées des vertex sont données par `V[a,:]`, `V[b,:]` et `V[c,:]`. Habituellement, nous avons `nf` >> `nv`.

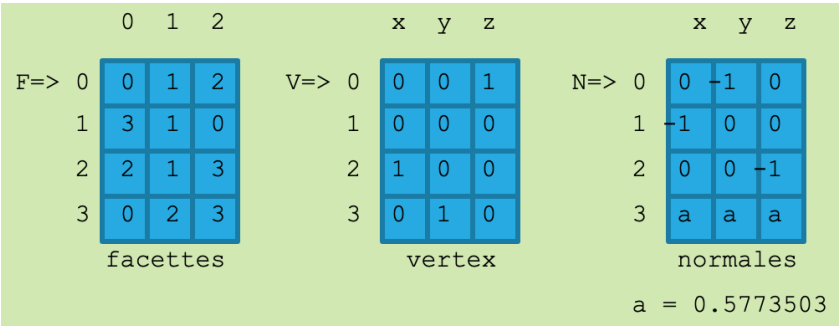
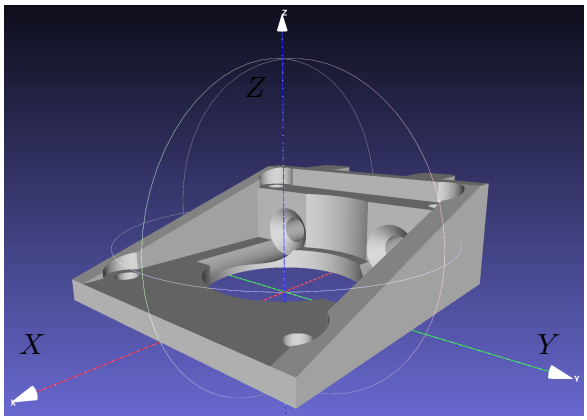


FIGURE 2.3 – Structure des matrices **F**, **V** et **N** après lecture du fichier `Prisme.stl`

2.2 Analyse

Le script 4 analyse un fichier STL. Il demande le nom du fichier, puis charge dans les matrices **F**, **V** et **N** le contenu du fichier en ayant retiré au préalable les répétitions des vertex. Il affiche le nombre de facettes et vertex, puis calcule les valeurs minimales et maximales des coordonnées des vertex selon les axes X, Y et Z. Les coordonnées du centre et les dimensions de l'objet 3D sont calculées à partir des valeurs limites. La Fig. 2.4 montre le résultat de l'exécution du script 4. Il affiche les dimensions de la pièce, ainsi que des matrices **F** (7478×3), **V** (3727×3) et **N** (7478×3).




```
Quel est le nom du fichier STL ? Nema17.stl
Lecture du fichier ... Nema17.stl
Analyse du fichier ... Nema17.stl
Nombre: facettes=7478 vertex=3727
X: min=-27.150000 max=21.150000
Y: min=-20.000000 max=20.000000
Z: min=0.000000 max=15.000000
Centre: x=-3.000000 y=0.000000 z=7.500000
Dimension: x=48.300000 y=40.000000 z=15.000000
F: (7478, 3) V: (3727, 3) N: (7478, 3)
```

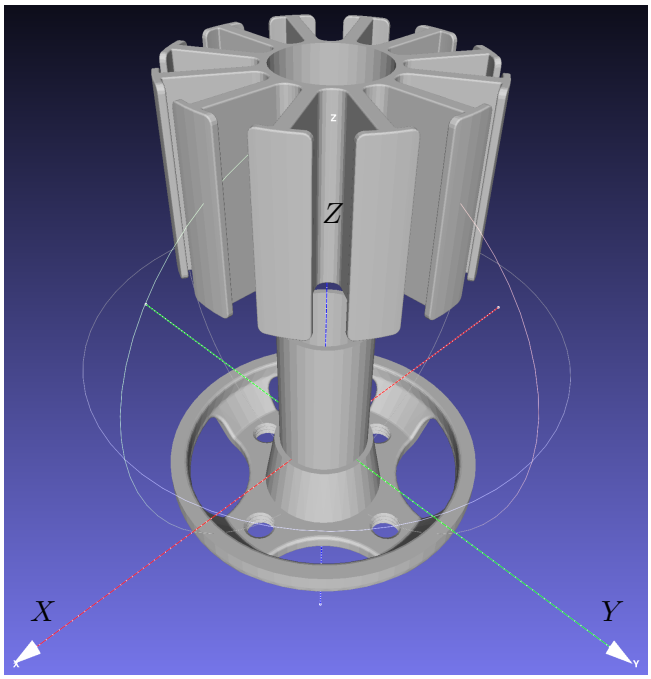
FIGURE 2.4 – Affichage de l'objet et résultat des mesures du fichier `Nema17.stl`

SCRIPT 4
Lecture et analyse d'un fichier STL - `C3_Mesure.py`

```
1  import numpy as np
2  import matplotlib.pyplot as plt
3  from   MEC1315_STL import *            #Acces aux fonctions STL
4  NomFichier = input('Quel est le nom du fichier STL ? ')
5  print('Lecture du fichier ...', NomFichier)
6  F,V,N = LireSTL(NomFichier)            #Lecture du fichier STL
7  print('Nombre: facettes=%d'% len(F),'vertex=%d'% len(V))
8  xmin, ymin, zmin = min(V[:,0]), min(V[:,1]), min(V[:,2])
9  xmax, ymax, zmax = max(V[:,0]), max(V[:,1]), max(V[:,2])
10 print('X: min=%f'% xmin,'max=%f'% xmax)
11 print('Y: min=%f'% ymin,'max=%f'% ymax)
12 print('Z: min=%f'% zmin,'max=%f'% zmax)
13 xc , yc, zc = (xmax+xmin)/2, (ymax+ymin)/2, (zmax+zmin)/2 #Centre de l'objet
14 print('Centre: x=%f'%xc,'y=%f'%yc, 'z=%f'%zc)
15 xd , yd, zd = (xmax-xmin), (ymax-ymin), (zmax-zmin) #Dimension de l'objet
16 print('Dimension: x=%f'%xd,'y=%f'%yd, 'z=%f'%zd)
17 print('F:', F.shape, 'V:', V.shape, 'N:',N.shape)   #Dimension de F,V,N
18 fig=plt.figure()   #---- Affichage du STL dans Python ----
19 ax=fig.add_subplot(1,1,1, projection='3d')          #Graphique 3D
20 ax.plot_trisurf(V[:,0],V[:,1],V[:,2], triangles=F)  #Vertex et numeros de facettes
```

2.3 Fusion

La fusion de fichiers STL requiert de déposer les objets STL dans un même référentiel. Ainsi, il faut empiler les matrices **F1**, **V1**, **N1** de l'objet 1 sur des matrices **F2**, **V2**, **N2** de l'objet 2. Les numéros de vertex de **F2** doivent alors être décalés du nombre de vertex de **V1**, tel que montré à la Fig. 2.6.



(a) Couvercle et rotor centrés sur l'axe z

```
Quel est le nom du fichier STL ? Rotor.stl
Lecture du fichier ... Rotor.stl
Analyse du fichier ... Rotor.stl
Nombre: facet=18300 vertex=9150
X: min=-10.991930 max=10.991930
Y: min=-10.991930 max=10.991930
Z: min=20.000000 max=33.500000
Centre: x=0.000000 y=0.000000 z=26.750000
Dimension: x=21.983860 y=21.983860 z=13.500000
F: (18300, 3) V: (9150, 3) N: (18300, 3)

Quel est le nom du fichier STL ? Couvercle.stl
Lecture du fichier ... Couvercle.stl
Analyse du fichier ... Couvercle.stl
Nombre: facet=8142 vertex=4055
X: min=-13.907690 max=13.925000
Y: min=-13.920670 max=13.920670
Z: min=-7.000000 max=13.500000
Centre: x=0.008655 y=0.000000 z=3.250000
Dimension: x=27.832690 y=27.841340 z=20.500000
F: (8142, 3) V: (4055, 3) N: (8142, 3)
```

(b) Dimensions des objets STL

FIGURE 2.5 – Affichage de `Fusion.stl` résultant de la fusion de `Rotor.stl` et `Couvercle.stl`

SCRIPT 5
Fusion des fichiers `Rotor.stl` et `Couvercle.stl` en un seul fichier STL - `C3_Fusion.py`

```
1  import numpy as np
2  from  MEC1315_STL import *
3  print('Lecture du fichier Rotor.stl ...')
4  F1,V1,N1 = LireSTL('Rotor.stl')  #Lecture du fichier STL
5  print('Lecture du fichier Couvercle.stl ...')
6  F2,V2,N2 = LireSTL('Couvercle.stl') #Lecture du fichier STL
7  nv1 = len(V1)           #Nombre de vertex dans V1
8  F2 = F2 + nv1           #Decalage des numeros de vertex pour F2
9  F = np.vstack((F1,F2)) #Empiler les facets F1 et F2
10 V = np.vstack((V1,V2)) #Empiler les vertex V1 et V2
11 N = np.vstack((N1,N2)) #Empiler les normales N1 et N2
12 print('Ecriture du fichier Fusion.stl ...')
13 EcrireSTLASCII('Fusion.stl',F,V,N) #Ecriture du fichier
```

```
Lecture du fichier Rotor.stl ...
Lecture du fichier Couvercle.stl ...
Ecriture du fichier Fusion.stl ...
```

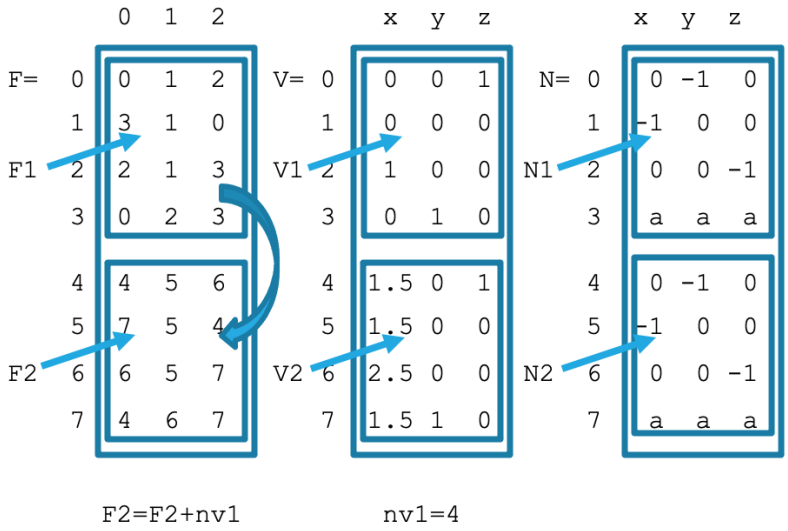


FIGURE 2.6 – Empilement de **F1**, **V1**, **N1** sur **F2**, **V2**, **N2** avec décalage des numéros de **F2**

2.4 Déplacement

Selon la formulation transposée, la translation et la rotation des vertex d'un objet STL se calculent en 3D de la même façon d'en 2D

$$\mathbf{U} = \mathbf{p}^T + \mathbf{V}\mathbf{R}^T \tag{2.1}$$

où $\mathbf{U}$ et $\mathbf{V}$ sont des matrices verticales $m \times 3$ des coordonnées des vertex dans les référentiels $\mathcal{A}$ et $\mathcal{B}$, respectivement, et $\mathbf{R}^T$ la matrice de rotation 3×3 transposée qui postmultiplie $\mathbf{V}$. En 3D, les matrices de rotation transposée sont définies par

$$\mathbf{R}_x^T \equiv \begin{bmatrix} 1 & 0 & 0 \\ 0 & c & s \\ 0 & -s & c \end{bmatrix}, \quad \mathbf{R}_y^T \equiv \begin{bmatrix} c & 0 & -s \\ 0 & 1 & 0 \\ s & 0 & c \end{bmatrix}, \quad \mathbf{R}_z^T \equiv \begin{bmatrix} c & s & 0 \\ -s & c & 0 \\ 0 & 0 & 1 \end{bmatrix}, \tag{2.2}$$

avec $s = \sin\theta$ et $c = \cos\theta$. L'angle θ doit être exprimé en radian. Les fonctions `RxT(theta)`, `RyT(theta)` et `RxT(theta)` de la bibliothèque `MEC1315_STL.py` permettent d'obtenir directement les 3 matrices de rotation en format transposée.

2.4.1 Translation

Dans l'exemple précédent, le rotor est centré sur l'axe z, le dessous à une hauteur de 20 mm. Le couvercle est également centré sur z, le dessus de l'épaulement à une hauteur de 0 mm. Afin d'assembler les deux objets, il faut déplacer le rotor vers le bas de 20 mm. Le script 6 contient la fonction `Translation(objet,p)`, où `objet` est la liste des matrices **F**, **V**, **N** et `p` le vecteur de déplacement. Il contient également la fonction `Groupe(objet1,objet2)` qui permet de regrouper deux objets STL en une seul. Ainsi, nous avons :

1. lire les fichiers `Rotor.stl` et `Couvercle.stl` ;
2. translation du rotor de -20 mm en z avec la fonction `Translation()` ;
3. regroupe le couvercle et le rotor en un seul objet STL avec la fonction `Groupe()` ; et finalement,
4. enregistre l'objet STL dans le fichier `Noyau.stl`.

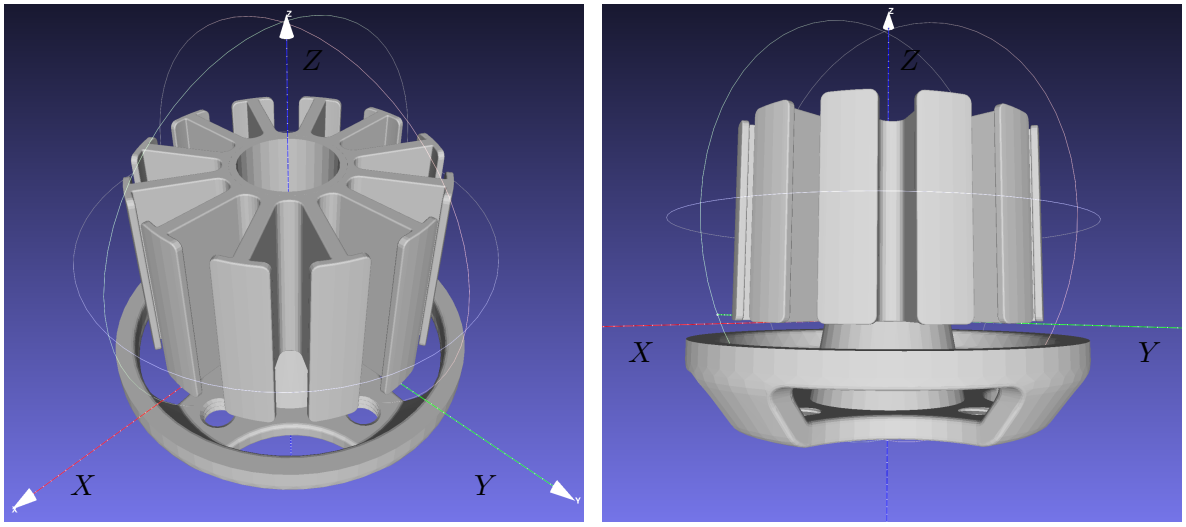


FIGURE 2.7 – Assemblage du rotor sur le couvercle par une translation de -20 mm en z du rotor

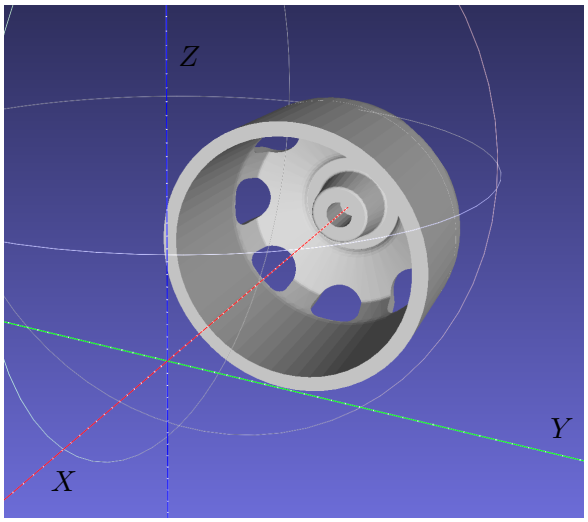
SCRIPT 6
Translation du rotor de -20 mm en z et fusion avec le couvercle - `C3_Translation.py`

```
import numpy as np
from  MEC1315_STL import *
def Translation(objet,p):
    F, V, N = objet[0], objet[1], objet[2]
    p = np.array(p).reshape(1,3)          #p doit etre un numpy array 1 x 3
    U = p + V                             #Translation U = p^T + V
    return [F,U,N]
def Groupe(objet1,objet2):                #Groupement de 2 objets STL
    objet2[0]= objet2[0] + len(objet1[1]) #Decalage des numeros de vertex
    F = np.vstack((objet1[0],objet2[0]))  #Empile les matrices Fi,Vi,Ni
    V = np.vstack((objet1[1],objet2[1]))
    N = np.vstack((objet1[2],objet2[2]))
    return [F,V,N]
print('Lecture du fichier Couvercle.stl ...')
Couv  = LireSTL('Couvercle.stl')    #Lecture du fichier STL
print('Lecture du fichier Rotor.stl ...')
Rotor = LireSTL('Rotor.stl')        #Lecture du fichier STL
Rotor = Translation(Rotor,[ 0, 0, -20])
F,V,N = Groupe(Rotor,Couv)          #Groupe 2 objets
print('Ecriture du fichier Noyau.stl ...')
EcrireSTLASCII('Noyau.stl',F,V,N)  #Ecriture du fichier
```

```
Lecture du fichier Couvercle.stl ...
Lecture du fichier Rotor.stl ...
Ecriture du fichier Noyau.stl ...
```

2.4.2 Rotation

La Fig. 2.8 montre le carter centré sur l'axe x entre -25 mm et -53 mm. Afin de pouvoir assembler le carter sur le moteur qui lui est centré sur l'axe z, il faut d'abord faire une rotation de $+\pi/2$ autour de l'axe y, puis une translation de -26.1mm en z. Le script 7 permet d'effectuer ces opérations. La Fig. 2.9 montre le résultat de l'assemblage.



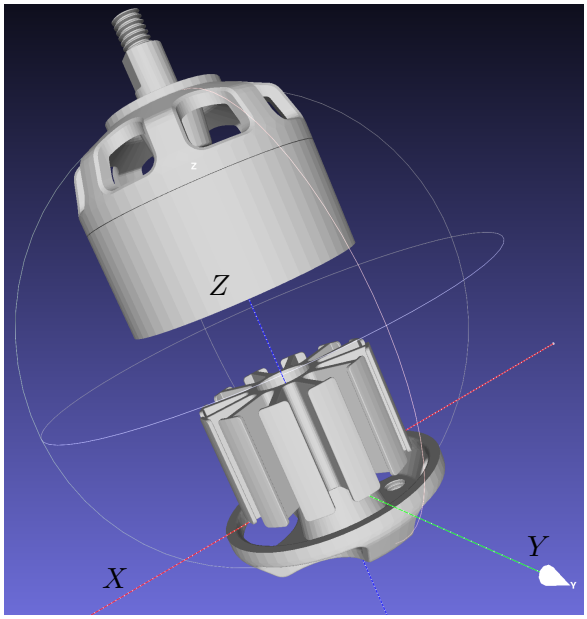
(a) Carter centrés sur l'axe x

```
Quel est le nom du fichier STL ? Carter.stl
Lecture du fichier ... Carter.stl
Nombre: facettes=9488 vertex=4673
X: min=-58.000000 max=-25.000000
Y: min=-13.925000 max=13.925000
Z: min=-13.925000 max=13.925000
Centre: x=-41.500000 y=0.000000 z=0.000000
Dimension: x=33.000000 y=27.850000 z=27.850000
F: (9488, 3) V: (4673, 3) N: (9488, 3)
```

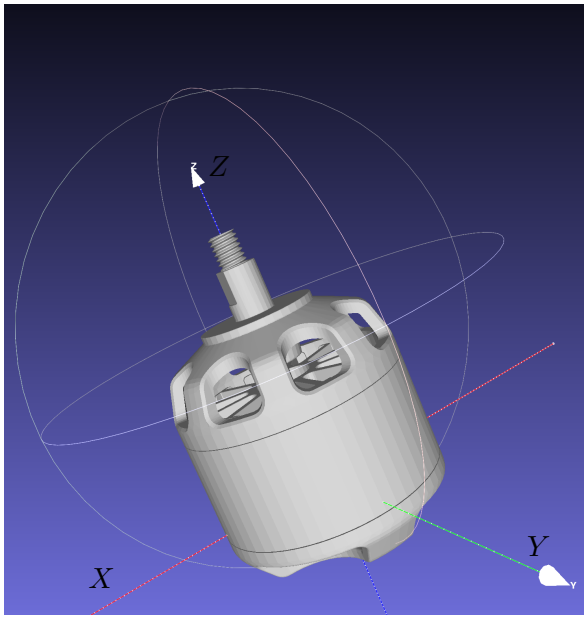
(b) Dimensions de l'objet STL

FIGURE 2.8 – Définition du carter dans son référentiel.

La fonction `RotationY(objet,p,theta)` permet le déplacement d'un objet du vecteur $\mathbf{p}$ et une rotation $\mathbf{R}_y^T(\theta)$ avec la formulation transposée du déplacement $\mathbf{U} = \mathbf{p}^T + \mathbf{V}\mathbf{R}_y^T$, tel que montré à la ligne 6 du script 7. Bien que `objet` soit une liste de 3 `numpy array`, cette fonction n'est qu'une implantation simpliste d'une fonction de déplacement en 3D qui devraient être plus générale.



(a) Carter tourné de $+\pi/2$ autour de y



(b) Carter déplacé de -26.1 mm en z

FIGURE 2.9 – Assemblage préliminaire et final du moteur dans le fichier `Moteur.stl`

SCRIPT 7
Rotation du carter de $+\pi/2$ autour de y, puis translation de -26.1 mm en z et fusion avec le moteur - C3_Rotation.py

```
import numpy as np
from  MEC1315_STL import *
def RotationY(objet,p,theta):
    F, V, N = objet[0], objet[1], objet[2]
    p = np.array(p).reshape(1,3)          #p doit etre un numpy array 1 x 3
    U = p + V.dot(RyT(theta))             #Deplacement U = p^T + V Ry^T
    return [F,U,N]
def Groupe(objet1,objet2):                #Groupement de 2 objets STL
    objet2[0]= objet2[0] + len(objet1[1]) #Decalage des numeros de vertex
    F = np.vstack((objet1[0],objet2[0]))  #Empile les matrices Fi,Vi,Ni
    V = np.vstack((objet1[1],objet2[1]))
    N = np.vstack((objet1[2],objet2[2]))
    return [F,V,N]
print('Lecture du fichier Noyau.stl ...')
Noyau = LireSTL('Noyau.stl')      #Lecture du fichier STL
print('Lecture du fichier Carter.stl ...')
Carter = LireSTL('Carter.stl')    #Lecture du fichier STL
Carter = RotationY(Carter,[ 0, 0, -26.1],np.pi/2)
F,V,N  = Groupe(Noyau,Carter)     #Groupe 2 objets
print('Ecriture du fichier Moteur.stl ...')
EcrireSTLASCII('Moteur.stl',F,V,N) #Ecriture du fichier
```

```
Lecture du fichier Noyau.stl ...
Lecture du fichier Carter.stl ...
Ecriture du fichier Moteur.stl ...
```

2.5 Homothétie

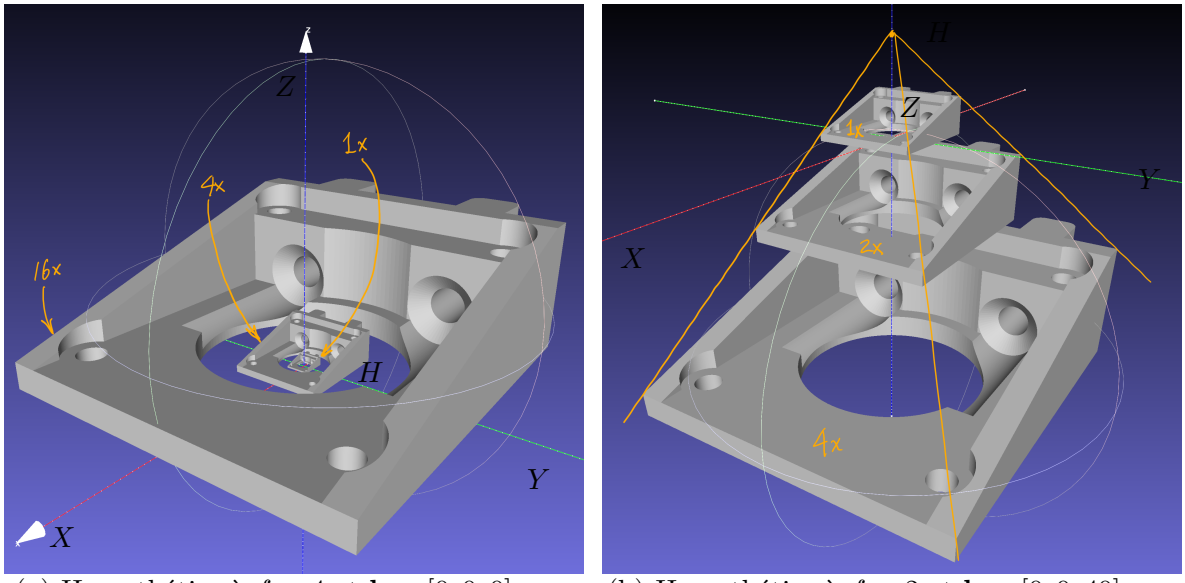
Comme en 2D, une homothétie est une transformation géométrique par agrandissement $f > 1$ ou réduction $f < 1$ qui produit une mise à l'échelle. Elle se caractérise par un centre d'homothétie H invariant, c'est-à-dire qu'il ne bouge pas. Les coordonnées des points résultants d'une homothétie se calculent de la même façon qu'en 2D, c'est-à-dire

$$\mathbf{U} = f(\mathbf{V} - \mathbf{h}^T) + \mathbf{h}^T = f\mathbf{V} + (1-f)\mathbf{h}^T \tag{2.3}$$

où $\mathbf{h}$ est la position du centre d'homothétie H et $f > 0$. Le script 8 utilise l'éq.(2.3) pour calculer l'homothétie des vertex de la pièce Nema17 pour $f = 2$ avec un point H à $\mathbf{h} = [0, 0, 40]$ mm. Chacune des pièces est 2 fois plus grande et 2 fois plus éloignée du point H que la précédente, tel que montré à Fig. 2.10(b). Si $\mathbf{h} = [0, 0, 0]$ mm et $f = 4$, alors le point H est à l'origine et chacune des pièces est 4 fois plus grande que la précédente, toujours centrées sur l'origine, tel que montré à la Fig. 2.10(a).

2.6 Conclusions

Les scripts présentés dans ce chapitre sont courts et spécifiques. Ils contiennent des fonctions, souvent simplistes, qui nécessitent une généralisation avant d'être incluses dans une bibliothèque, comme nous l'avons fait avec `MEC1315_STL.py` pour les fonctions STL.



(a) Homothéties à $f = 4$ et $\mathbf{h} = [0, 0, 0]$ mm (b) Homothéties à $f = 2$ et $\mathbf{h} = [0, 0, 40]$ mm

FIGURE 2.10 – Homothéties de la pièce Nema17 avec f et point H différents

SCRIPT 8
Homothéties du fichier `Nema17.stl` avec $f = 2.0$ de centre $\mathbf{h} = [0, 0, 40]$ mm

```
import numpy as np
from  MEC1315_STL import *
def Homothetie3D(objet,h,f):
    F, V, N = objet[0], objet[1], objet[2]
    h = np.array(h).reshape(1,3)    #h doit etre un numpy array 1 x 3
    U = f*V + (1-f)*h               #Formule de l'homothetie
    return [F,U,N]
def Groupe(objet1,objet2):                #Groupement de 2 objets STL
    objet2[0]= objet2[0] + len(objet1[1]) #Decalage des numeros de vertex
    F = np.vstack((objet1[0],objet2[0]))  #Empile les matrices Fi,Vi,Ni
    V = np.vstack((objet1[1],objet2[1]))
    N = np.vstack((objet1[2],objet2[2]))
    return [F,V,N]
print('Lecture du fichier Nema17.stl ...')
Nema1 = LireSTL('Nema17.stl')             #Lecture du fichier STL
Nema2 = Homothetie3D(Nema1,[0, 0, 40],2)
Nema3 = Homothetie3D(Nema2,[0, 0, 40],2)
F,V,N = Groupe(Groupe(Nema1,Nema2),Nema3) #Groupe 3 objets
print('Ecriture du fichier Homo3D.stl ...')
EcrireSTLASCII('Homo3D.stl',F,V,N)        #Ecriture du fichier
```

```
Lecture du fichier Nema17.stl ...
Ecriture du fichier Homo3D.stl ...
```