Module C3

Manipulation de matrices et vecteurs avec Python 3

Timothée Duruisseau
Richard Nguyen The Phat

Department de génie mécanique
École Polytechnique Montréal
Canada
20 octobre 2021

Objectifs du module

À la fin du module, vous devriez maitriser la manipulation des composantes des matrices que ce soit un composant ou d'une colonne ou de rangée à travers les applications de transformations géométrique tel que la translation, rotation, l'homothétie, l'affinité et ainsi que la duplication de fichier 3D STL.

Table des matières

Chapitre 1

Chapitre 1 : Semaine C3 sur fichier STL

1.1 Introduction

Le chapitre C3 a pour but de vous faire pratiquer davantage sur les manipulations matricielles et cette fois ci dans des applications plus concrètes tel que sur les fichiers STL.

Un fichier STL, (surface tesselation language ou standard triangle language) est un fichier ASCII ou binaire utilisé dans l'environnement 3D. Une des applications courant des fichiers STL est dans l'impression 3D, votre modèle CAD est exporté dans un fichier standard dans lesquels les logicielles peuvent lire vos fichiers 3D et par la suite l'imprimer. D'autres domainse comme, l'industrie de cinéma et d'animation peuvent aussi utiliser les fichiers très similaire aux STL. Finalement, un des gros industrie qui peuvent utiliser les fichiers STL est l'industrie de jeux vidéo pour générer des environnements ou personnages, etc et de les animer par la suite.

Dans le contexte du cours MEC1315, nous allons traiter principalement les fichiers STL en format ASCII, bien qu'il y ait un moyen de convertir de la version ASCII en binaire, ou vice versa, nous ne consacrons pas de temps là-dessus. Un fichier STL ASCII est caractérisé par les éléments suivants, au besoin référez vous aux code de couleurs pour les identifiers. :
Une ouverture de fichier
Une déclaration d'une facette et de vecteur normal et ainsi que ses composantes de vecteur orthonormal
Une déclaration de 3 vertex (ou points) et de ses composants x,y,z
Une fermeture du fichier

```
solid name
facet normal n_i n_j n_k
  outer loop
    vertex v1_x v1_y v1_z
    vertex v2_x v2_y v2_z
    vertex v3_x v3_y v3_z
  endloop
endsolid name
```

Par analogie, on peut considérer qu'un objet STL c'est un objet dans l'espace composé de maillages de facettes triangulaire délimités par ses vertex(points).

La figure 1.1 affiche un extrait de code STL ASCII d'un train en jouet et de sa représentation 3D dans le logicielles Meshlab. Les facettes triangulaires et les vertex sont visibles.

On constate que le train en jouet STL est finalement la combinaison d'une multitudes de facettes

train_jouet.stl

```
solid ASCII
  facet normal -9.822309e-01 -1.848253e-01 -3.258969e-02
    outer loop
      vertex   -5.856887e+00 -3.939693e+00 2.454498e+00
      vertex   -5.818827e+00 -4.143084e+00 2.460879e+00
      vertex   -5.856887e+00 -4.000000e+00 2.796518e+00
    endloop
  endfacet
  facet normal -9.803997e-01 -1.950136e-01 -2.803873e-02
    outer loop
      vertex   -5.856887e+00 -4.000000e+00 2.796518e+00
      vertex   -5.818827e+00 -4.143084e+00 2.460879e+00
      vertex   -5.818827e+00 -4.191342e+00 2.796518e+00
    endloop
  endfacet
  facet normal -9.803997e-01 -1.950136e-01 2.803873e-02
    outer loop
      vertex   -5.856887e+00 -4.000000e+00 2.796518e+00
      vertex   -5.818827e+00 -4.191342e+00 2.796518e+00
      vertex   -5.818827e+00 -4.143084e+00 3.132158e+00
    endloop
  endfacet
  facet normal -8.288296e-01 -5.538062e-01 7.962531e-02
    outer loop
      vertex   -5.818827e+00 -4.143084e+00 3.132158e+00
      vertex   -5.818827e+00 -4.191342e+00 2.796518e+00
      vertex   -5.710441e+00 -4.353553e+00 2.796518e+00
    endloop
  endfacet
  facet normal -8.288296e-01 -5.538062e-01 7.962531e-02
    outer loop
      vertex   -5.818827e+00 -4.143084e+00 3.132158e+00
      vertex   -5.710441e+00 -4.353553e+00 2.796518e+00
      vertex   -5.710441e+00 -4.298725e+00 3.177858e+00
    endloop
  endfacet
  facet normal -5.516423e-01 -8.255911e-01 1.187021e-01
    outer loop
      vertex   -5.710441e+00 -4.298725e+00 3.177858e+00
```

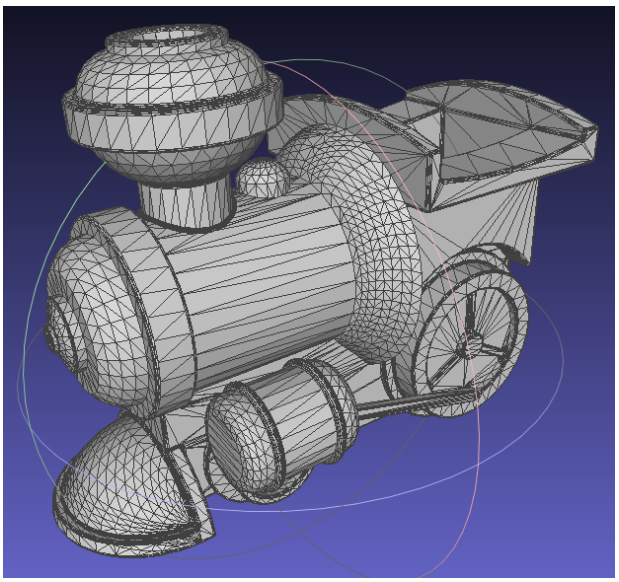

FIGURE 1.1 – General caption.

positionnés et orienté dans l'espace. À prioris, un fichier STL ASCII possède une multitude de vertex dupliqués, ce qui explique aussi pourquoi ce type de fichier grandi très vite en taille dû aux répétitions. Pur illustrer ce concept, considérer les facettes F1, F2 ... F5 qui ont un vertex commun V1. V1 sera alors dupliqués autant de fois dans le fichier STL ASCII qu'il y a de facette comme le montre à la figure 1.2 :

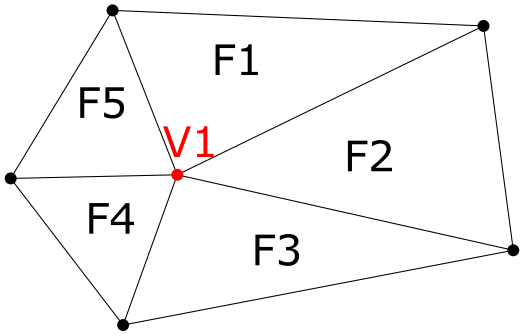

FIGURE 1.2 – Exemple d'un polygone formé par facettes triangulaires.

Les logicielles 3D doivent donc développer une technique pour gérer les facettes, les vertex et les vecteurs normaux afin d'éliminer les redondances. Un des moyens possibles est de travailler avec la matrices de connectivités de facettes, de vertex et de vecteur normaux.

On appelle une matrice de connectivité, une matrice dont ses éléments sont délimités par ses éléments de frontières. Prenons un exemple de maillage simplifié de la figure 1.3 pour illustrer le concept.

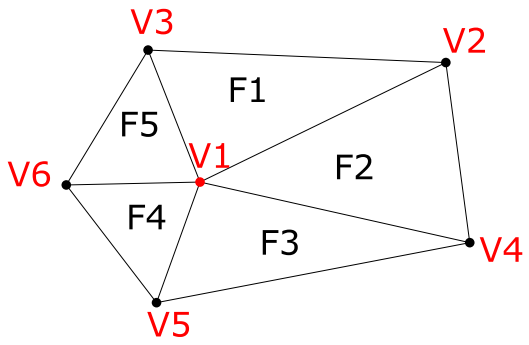

FIGURE 1.3 – Exemple d'un polygone formé par facettes triangulaires.

Pour chaque facettes est délimité par 3 vertex. Les matrices de connectivités de facettes (1), de vertex(2) et de vecteurs normaux(3) sont représenté par les éléments de la figure 1.4.

$$\underbrace{\begin{vmatrix} F_1 & V_1 & V_2 & V_3 \\ F_2 & V_1 & V_4 & V_2 \\ F_3 & V_1 & V_5 & V_4 \\ F_4 & V_1 & V_6 & V_5 \\ F_5 & V_1 & V_3 & V_6 \end{vmatrix}}_{1} \cdots\cdots \underbrace{\begin{vmatrix} V_1 & v_{1x} & v_{1y} & v_{1z} \\ V_2 & v_{2x} & v_{2y} & v_{2z} \\ V_3 & v_{3x} & v_{3y} & v_{3z} \\ V_4 & v_{4x} & v_{4y} & v_{4z} \\ V_5 & v_{5x} & v_{5y} & v_{5z} \\ V_6 & v_{6x} & v_{6y} & v_{6z} \end{vmatrix}}_{2} \cdots\cdots \underbrace{\begin{vmatrix} \vec{N_1} & n_1\vec{i} & n_1\vec{j} & n_1\vec{k} \\ \vec{N_2} & n_2\vec{i} & n_2\vec{j} & n_2\vec{k} \\ \vec{N_3} & n_3\vec{i} & n_3\vec{j} & n_3\vec{k} \\ \vec{N_4} & n_4\vec{i} & n_4\vec{j} & n_4\vec{k} \\ \vec{N_5} & n_5\vec{i} & n_5\vec{j} & n_5\vec{k} \end{vmatrix}}_{3}$$

FIGURE 1.4 – Matrices de facettes, de vertex et vecteurs orthonormaux

À titre d'exemple, la face F3 est donc délimité par les vertex V1,V5 et V6 et les coordonnées de ces vertex sont décrits dans la matrice (2). Similairement, pour la matrice (3) contient les composantes vectorielles de $\vec{N}_3$.

Avec la définition des ces 3 matrices, on élimine ainsi toute forme de redondance quand on doit traiter les fichiers de format STL. Évidement, d'un point de vu de programmation on ne va pas littéralement appeler F1,F2, V1, V2 ou N1, N2 et ainsi que ses sous-composants. On utilisera les index des matrices pour repérer les éléments. La déclarations des matrices auront la forme similaire à la figure 1.4. Si on désire, analyser la facette 4, on analysera la 4ieme ligne de la matrice F, si on désire analyser le vertex 5 on regardera la 5ieme ligne de la matrice V et ainsi de suite tel que montre à la figure 1.5

$$\underbrace{\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccc} & 1 & 2 & 3 \end{array} \\ F = \begin{vmatrix} V_1 & V_2 & V_3 \\ V_1 & V_4 & V_2 \\ V_1 & V_5 & V_4 \\ V_1 & V_6 & V_5 \\ V_1 & V_3 & V_6 \end{vmatrix} \end{array}}_{1} \quad \underbrace{\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \\ 6 \end{array} \begin{array}{c} \begin{array}{cccc} & 1 & 2 & 3 \end{array} \\ V = \begin{vmatrix} v_{1x} & v_{1y} & v_{1z} \\ v_{2x} & v_{2y} & v_{2z} \\ v_{3x} & v_{3y} & v_{3z} \\ v_{4x} & v_{4y} & v_{4z} \\ v_{5x} & v_{5y} & v_{5z} \\ v_{6x} & v_{6y} & v_{6z} \end{vmatrix} \end{array}}_{2} \quad \underbrace{\begin{array}{c} \\ 1 \\ 2 \\ 3 \\ 4 \\ 5 \end{array} \begin{array}{c} \begin{array}{cccc} & 1 & 2 & 3 \end{array} \\ N = \begin{vmatrix} n_1\vec{i} & n_1\vec{j} & n_1\vec{k} \\ n_2\vec{i} & n_2\vec{j} & n_2\vec{k} \\ n_3\vec{i} & n_3\vec{j} & n_3\vec{k} \\ n_4\vec{i} & n_4\vec{j} & n_4\vec{k} \\ n_5\vec{i} & n_5\vec{j} & n_5\vec{k} \end{vmatrix} \end{array}}_{3}$$

FIGURE 1.5 – Matrices de facettes, de vertex et vecteurs orthonormaux

1.2 Transformation Géométrique

1.2.1 Translation linéaire

La translation linéaire de modèle 3D nécessite seulement de modifier le contenu de la matrice de vertex V, puisque c'est cette matrice qui contiennent les coordonnées dans l'espace. Le code 1 montre la translation d'un train de 30 unités en x, -30 unités en y et 10 unités en z avec la fonction array([x,y,z]) de numpy.

SCRIPT 1
Exemple de code pour translater une géomtrie dans l'espace

```
1 import numpy as np
2 from MEC1315_STL import LireSTL,EcrireSTL ,Rx, Ry, Rz,EcrireSTLASCII, CalculNormal
3 #-------------------------------------------------------------------------------
4 F,V,N= LireSTL('Train_jouet.stl')
5 #Translation du train +30 en x, -30 en y, +10 en z
6 V=V+np.array([ 30 , -30 , 10 ])
7
8 EcrireSTLASCII('Train_translation.stl', F, V, N)
```

Le scripte 1 permet de produire le résultat de la figure 1.6, notez que le train n'est plus centré à l'origine.

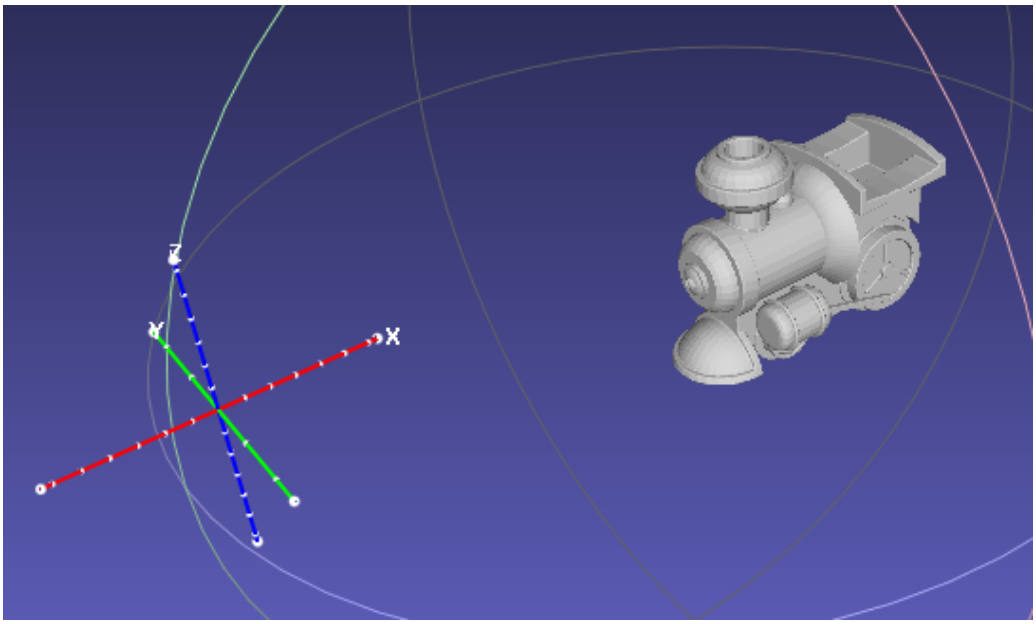

FIGURE 1.6 – Train en jouet ayant subit une translation

1.2.2 Rotation

La rotation d'un modèle STL autour d'un axe nécessite de modifier la matrice de vertex et de normaux. Il suffit de faire le produit matricielle de la matrice V et N par rapport à la matrice de

rotation. Les 3 matrices de rotation tournant dans le sens horaire (ou règle de la main gauche) sont définie de la manière tel que montre la figure 1.7. La multiplication matricielle se fait avec la fonction dot(matrice1,matrice2) de numpy tel que montre l'exemple de code 2 permet d'obtenir le résultat de la figure 1.8.

$$R_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & cos(\theta) & sin(\theta) \\ 0 & -sin(\theta) & cos(\theta) \end{bmatrix} R_y = \begin{bmatrix} cos(\theta) & 0 & -sin(\theta) \\ 0 & 1 & 0 \\ sin(\theta) & 0 & cos(\theta) \end{bmatrix} R_z = \begin{bmatrix} cos(\theta) & sin(\theta) & 0 \\ -sin(\theta) & cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

FIGURE 1.7 – Matrices de rotation dans le sens horaire de rotation

SCRIPT 2
Exemple de code pour rotation d'une géomtrie dans l'espace

```
import numpy as np
from MEC1315_STL import LireSTL,EcrireSTL, Rz,EcrireSTLASCII, CalculNormal
#-------------------------------------------------------------------------------
F,V,N= LireSTL('Train_jouet.stl')
#Rotation de pi/2 autour de l'axe des Z dans le sens horaire
angle=np.pi/2
R=Rz(angle)
V=np.dot( V , R)
N=np.dot(N , R)
#V=V.dot(R)
#N=N.dot(R)

EcrireSTLASCII('Train_rotation.stl', F, V, N)
```

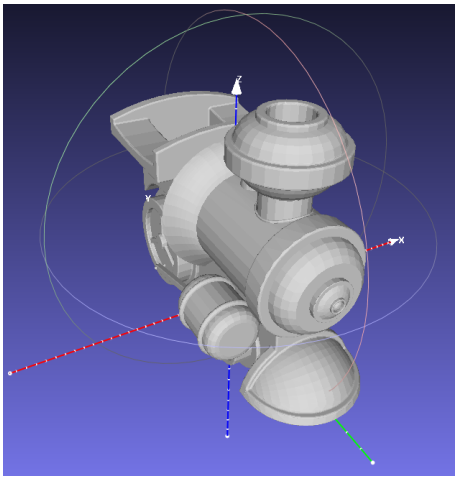

FIGURE 1.8 – Train en jouet pivoté autour de l'axe de z

1.2.3 Changement d'échelle

Dans le cas de changement d'échelle il existe deux possibilité, le changement d'échelle uniforme qu'on appelle Homothétie par rapport à l'origine, ou le changement non uniforme (grossissement ou

rétrécissement dans une direction particulière) appelé affinité en direction x, y ou z par rapport à l'origine. En principe, appliqué une homothétie ou affinité sur un objet qui n'est pas centré à l'origine, va aussi influencer sa position. Puisque le rayon origine-centroïde se voit aussi multiplier. Pour simplement changer l'échelle de l'objet sur lui même, il va falloir le centré à l'origine, appliquer la transformation et le remettre à sa position initiale.

Homothétie

L'homothétie nécessite d'appliquer un facteur d'échelle sur les composants du vertex seulement. Par exemple si on veut grossir de 3 fois la taille, il suffit de multiplier la matrice V par 3 tel que montre le code 3 permettant d'obtenir le résultat de la figure

Script 3
Exemple de code pour appliquer une homothétie sur une géométrie dans l'espace

```
import numpy as np
from MEC1315_STL import LireSTL,EcrireSTL ,Rx, Ry, Rz,EcrireSTLASCII, CalculNormal
#-------------------------------------------------------------------------------
F,V,N= LireSTL('Train_jouet.stl')
V=3*V

EcrireSTLASCII('Train_homothetie.stl', F, V, N)
```

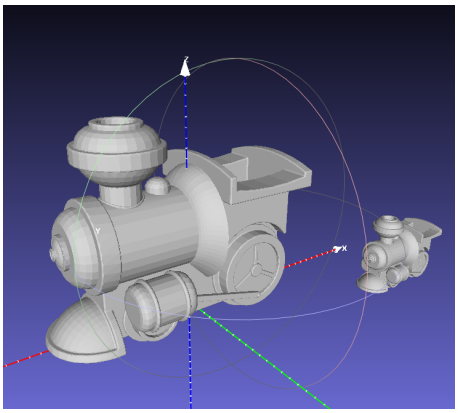

Figure 1.9 – Train en jouet avec un facteur d'échelle de 3

Affinité vectorielle

L'affinité vectorielle est une application de transformation qui change la dimension d'un objet selon une orientation particulière, par exemple, un étirement. L'affinité nécessite d'appliquer un coefficient sur la composante x ou y ou z. La matrice de vecteurs orthonormaux doit être en plus re-calculé. Par exemple, si on veut étirer 3 fois dans la direction x d'un objet, le code 4 permet d'obtenir le résultat de la figure 1.10. Similairement pour modifier l'échelle sur la direction y ou z, il suffit de remplacer l'indexe V[:,0] par V[:,1] ou par V[:,2].

SCRIPT 4
Exemple de code pour appliquer une affinité vectorielle d'une géométrie dans l'espace

```
import numpy as np
from MEC1315_STL import LireSTL,EcrireSTL ,Rx, Ry, Rz,EcrireSTLASCII, CalculNormal
#-------------------------------------------------------------------------------
F,V,N= LireSTL('Train_jouet.stl')
#Translation du train +5 en x, -3 en y, +8 en z
V[ :,0]=V[ :,0]*3
N=CalculNormal( F, V )

EcrireSTLASCII('Train_affinite.stl', F, V, N)
```

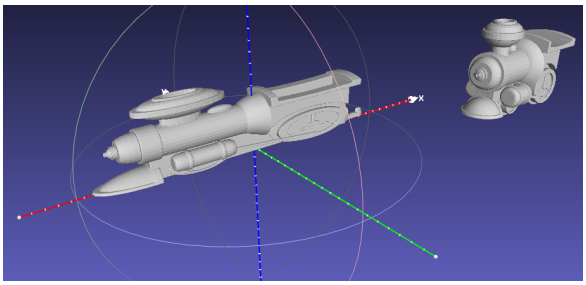

FIGURE 1.10 – Train en jouet étiré selon la direction des x

1.2.4 Duplication et ou ajout de plusieurs objets

Lorsqu'un objet doit être dupliqué ou vient s'ajouter un autre objet, il va falloir suivre les éléments :

- déterminer correctement l'ordre de l'objet de base (1) et de l'objet qui s'ajoute (2)
- déterminer le nombre de vertex de l'objet de base (1)
- concaténer les matrices de facettes de (1) et (2) avec ajout du nombre de vertex de l'objet (1) sur la matrice de facettes F2
- concaténer les matrices de vertex V1 et V2
- concaténer les matrice de vecteurs orthonormaux N1 et N2.

Le script 5 permet de cloner le train en jouet, de le faire translater et d'ajouter le chaton. Le tout format sois disant un seul corps STL tel que montre la figure 1.11

SCRIPT 5
Exemple de code pour dupliquer et ajouter un objet

```
import numpy as np
from MEC1315_STL import LireSTL,EcrireSTL ,Rx, Ry, Rz,EcrireSTLASCII, CalculNormal
#-------------------------------------------------------------------------------
F1,V1,N1= LireSTL('Train_jouet.stl')
nv1=len(V1) #Le nombre de vertex

F=np.vstack((F1, F1+nv1)) #Duplication, concaténation et ajout de nv1 sur l'objet F1
V2=V1 + np.array([0,0,100]) #Duplication (si on a avait pas translater, les deux objets
    seront superposé
V=np.vstack((V1,V2 ))
N=np.vstack(( N1,N1) )

F3,V3,N3=LireSTL('gato.stl')
nv=len(V)
V3=V3+np.array([100,0,0]) #Translation, sinon le chat sera coïncident aux objets

F=np.vstack( (F,F3+nv) )      #Concaténation et ajout de nv sur F3
V=np.vstack( (V,V3) )
N=np.vstack( (N,N3))


EcrireSTLASCII('Train_copie.stl', F, V, N)
```

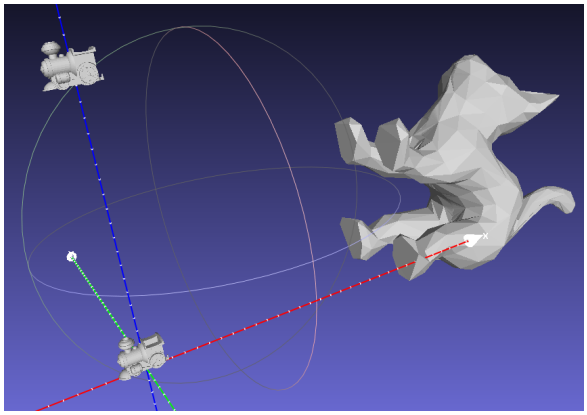

FIGURE 1.11 – Train en jouet fusionné avec un chat

1.2.5 Conclusion

En résumé, ce chapitre vous a introduit assez brièvement sur les fichiers STL et une méthode pour gérer les géométries tels que les faces, les vertex et les vecteurs normaux. Le point à retenir est que la matrice des facettes est une matrice de connectivité qui précise quels sont les vertex délimitant chaque facette. Lors des transformation, seul les matrices de vertex ou vecteurs normaux sont sujets à être modifié. La duplication ou l'ajout d'un objet stl supplémentaire nécessite de concaténer les matrices de facettes en tenant compte d'une constante, soit le nombre de vertex.