Guide d'utilisation :

Optimisation et perfectionnement d'un outil informatique pour simuler le procédé de fabrication additive LBM à l'aide de librairies sous python

Guérin Léo

Table des matières

1 – Mise en place

Après avoir entièrement téléchargé le code hébergé sur github, il faut lancer votre application spyder ou idle. Ensuite entrer l'instruction :

```
from STL import mesh
```

Si l'instruction renvoie une erreur, c'est que la librairie dont vous avez besoin pour utiliser mon programme n'est pas installée. Deux solutions existent :

- La première, la plus appropriée, est de taper l'instruction suivante :

```
pip install numpy-stl
```

- La seconde est de taper l'instruction suivante :

```
pip install STL
```

Une fois l'installation terminée la mise en place pour utiliser ce programme est finie.

2 – Fichier STL

Les fichiers au format *STL* sont les seuls fichiers utilisables pour mon programme. Ces fichiers peuvent être téléchargés depuis différents sites (`https://www.thingiverse.com/`) ou peuvent être créés.

3 – Lancement de mon programme

3.1 – Extraction des données d'un fichier STL

Pour utiliser mon programme il faut d'abord taper les instructions suivantes pour extraire les informations du fichier à découper :

Exemple 3.1.

```
from load import load_mesh

mesh = load_mesh("mon_fichier.stl","binaire")
```

La fonction load_mesh permet d'extraire les informations du fichier *STL* mon_fichier.slt, l'extraction est un mode binaire (il existe un autre mode d'extraction, ASCII, mais il n'est pas très utilisé). La variable mesh contiendra les informations sous le bon format pour pouvoir exécuter la suite de la simulation.

3.2 – Simulation du procédé LBM

Pour simuler le procédé *LBM*, il faut taper importer la fonction slice_objet contenue dans le fichier CuttingTools.py avec l'instruction suivante :

```
from CuttingTools import slice_objet
```

Puis mettre les entrées suivantes dans l'ordre dans la fonction slice_objet :

- mesh (type list) : variable contenant les informations du fichier STL chargé avec la fonction load_mesh
- width (type int ou float) : épaisseur des couches
- coupe_au_centre (type bool) : par défaut mettre False
- axe (type str) : axe de découpe ("x", "y" ou "z")
- rotation_angle (type int ou float) : angle en degré correspondant à la rotation du remplissage à chaque nouvelle couche
- N_facette_cordon (type int) : nombre de facettes pour la résolution des cordon (mettre au moins trois)
- rayon_cordon (type int ou float) : rayon des cylindres représentant les cordons
- overlap_entre_cordons (type int ou float) : espace entre chaque cordon
- overlap_entre_cordons_contour (type int ou float) : espace entre le cordon définissant le contour de la tranche et entre les cordons à l'intérieur du contour de la tranche
- type_cordon (type str) : "cylindrique" pour avoir des cordons cylindriques ou "clavette", ou "ellipse" pour avoir des cordons légèrement aplatis
- avec_extremite (type bool) : si on veut les extrémités des cordons
- type_maille (type list) : le premier item correspond au type de la maille, "lineaire", "losange", "hexagonale","aleatoire". Le second correspond au caractère croisé ou non , "croisee" ou "decroisee"
- parametre_de_maille (type int ou float) : paramètre de la maille (ne pas mettre zéro)
- holographique (type bool) : si on veut une surface holographique
- parametre (type int ou float) : paramètre de la maille holographique (ne pas mettre zéro)

Exemple 3.2. Voici un exemple pour une simulation

```
from load import load_mesh

mesh = load_mesh("mon_fichier.stl","binaire")

from CuttingTools import slice_objet

resultat,liste_coordonnees_cordons = slice_objet(mesh,0.1,False,"z"
    ,30,12,0.1,0,0,"cylindrique",True,["lineaire","decroisee"],1,False,1)
```

La sortie resultat contiendra les informations sur l'objet après simulation. La sortie liste_coordonnees_cordons contiendra les coordonnées des cordons sous un certain format :

format de liste_coordonnees_cordons :

$$[\underbrace{[\ldots\ldots\ldots\ldots]}_{\text{première couche}},\underbrace{[\ldots\ldots\ldots\ldots]}_{\text{seconde couche}},\ldots,\underbrace{[\ldots\ldots\ldots\ldots]}_{N\text{ième couche}}]$$

Le format pour chaque couche est le suivant :

$$[\underbrace{[\ldots\ldots\ldots\ldots]}_{\text{coordonnées des cordons formant le contourde la couche}},\underbrace{[\ldots\ldots\ldots\ldots]}_{\text{coordonnées des cordons ne formants pas le contour de la couche}}]$$

Pour ces deux items le format est le suivant :

$$[\underbrace{[\ldots\ldots\ldots\ldots]}_{\text{coordonnées du premier cordon}},\underbrace{[\ldots\ldots\ldots\ldots]}_{\text{coordonnées du cordon suivant}},\ldots]$$

Pour chaque cordon le format est le suivant :

$$[\underbrace{[x_1,y_1,z_1]}_{\text{coordonnées du point de départ du cordon}},\underbrace{[x_2,y_2,z_2]}_{\text{coordonnées du point de d'arrivée du cordon}}]$$

3.3 – Visualisation du résultat de la simulation

Première méthode

Une fois la variable resultat récupérée on peut voir l'aspect de l'objet après simulation du procédé *LBM*. Il suffit d'importer la fonction Create_fichier_stl contenue dans le fichier fonction_generation_fchier_stl.py :

Exemple 3.3.

```
from load import load_mesh

mesh = load_mesh("mon_fichier.stl","binaire")

from CuttingTools import slice_objet

resultat,liste_coordonnees_cordons = slice_objet(mesh,0.1,False,"z"
    ,30,12,0.1,0,0,"cylindrique",True,["lineaire","decroisee"],1,False,1)

from fonction_generation_fichier_stl impot Create_fichier_stl

Create_fichier_stl("nom_du_nouveau_fichier",resultat)
```

Il suffit ensuite d'ouvrir le fichier intitulé nom_nouveau_fichier.stl pour visualiser le résultat de la simulation.

Seconde méthode

Une fois la variable liste_coordonnees_cordons récupérée on peut voir l'aspect de l'objet après simulation du procédé *LBM*avec une sélection par couches . Tout d'abord il faut appliquer la fonction regneration_cordon (contenue dans le fichier CuttingTools.py) qui permet de reconstituer un objet mesh à partir des coordonnées des cordons. Il y a différentes entrées à saisir :

- coord_cordon (type list) : liste des coordonées des cordons
- cordon_contour (type bool) : si on veut faire apparaître les cordons formant le contour
- cordon_interieur (type bool) : si on veut faire apparaître les cordons qui sont à l'intérieur de la couche
- couche_min (type int ou str) : numéro de la première couche prise ou "tout" dans le cas où l'on veut l'intégralité de l'objet
- couche_max (type int ou str) : numéro de la dernière couche prise ou "tout" dans le cas où l'on veut l'intégralité de l'objet
- rayon (type int ou float) : rayon des cordons
- axe (type str) : axe orthogonal à la surface définie par le remplissage ("x", "y" ou "z")

Après avoir saisi ces entrées il faut importer la fonction Create_fichier_stl contenue dans le fichier fonction_generation_fchier_stl.py :

Exemple 3.4.

```
from load import load_mesh

mesh = load_mesh("mon_fichier.stl","binaire")

from CuttingTools import slice_objet,regeneration_cordon

resultat,liste_coordonnees_cordons = slice_objet(mesh,0.1,False,"z"
    ,30,12,0.1,0,0,"cylindrique",True,["lineaire","decroisee"],1,False,1)

mesh_a_visualiser = regeneration_cordon(liste_coordonnees_cordon,True,True
    ,2,50,0.1,"z")

from fonction_generation_fichier_stl impot Create_fichier_stl

Create_fichier_stl("nom_du_nouveau_fichier",resultat)
```

Il suffit ensuite d'ouvrir le fichier intitulé nom_nouveau_fichier.stl pour visualiser le résultat de la simulation.

4 – Autres fonctions utiles

4.1 – generer_cylindre_mesh

La fonction generer_cylindre_mesh, contenue dans le fichier fonction_generation_cordons.py, permet de générer un cordon en fonction de certaines caractéristiques. Cet objet une fois généré sera sous le format d'un mesh, utilisable par mes autres fonctions dont slice_objet et Create_fichier_stl. Pour utiliser cette fonction il y a plusieurs entrées à saisir :

- axe (type str) : axe correspond à l'une des directions principales de la base canonique ("x", "y", ou "z")
- rayon (type int ou float) : rayon du cordon
- hauteur (type int ou float) : correspond à la longueur du cordon
- centre_base (type list) : liste des coordonnées du centre du polygone sous le format [x,y,z]
- vecteur_directeur (type list) : liste des coordonnées de la normale de la base de centre "centre_base" , orienté vers l'intérieur du cordon, sous le format [x,y,z]
- N_facette_cordon (type int) : le nombre de sommets du polygone de la base du cordon
- type_cordon (type str) : "cylindrique" pour des cordons cylindriques, "clavette" pour des cordons plats, ou "ellipse" pour des cordons à sections elliptiques
- avec_dome (type bool) : pour avoir les extrémités ou non
- type_maille (type str) : correspond au type de maille, "lineaire", "losange", "hexagonale", "aleatoire"
- parametre_de_maille (type int ou float) : espace entre deux alternances de mailles
- orientation_maille (type str) : "+" si la maille est orientée à "gauche", "-" si la maille est orientée à "droite"
- coord_de_depart (type int ou float) : dans le cas d'un maillage cela devient l'origine du maillage

Cette fonction renvoie deux sorties : mesh qui sera une liste renfermant les informations sur le cordon et qui sera exploitable par d'autres fonctions, l'autre sortie est liste_coord_centre qui contient les coordonnées du point de départ et d'arrivée du cordon.

Exemple 4.1. Exemple avec visualisation

```
from fonction_generation_cordons import generer_cylindre_mesh

mesh,coordonnees = generer_cylindre_mesh("z",3,10,[0,0,0],[0,1,0],12,"
    ellipse",False,"lineaire",1,"+",0)

from fonction_generation_fichier_stl import Create_fichier_stl

Create_fichier_stl("cordon",mesh)
```

En ouvrant le fichier cordon.stl qui vient d'être crée on obtient ceci :

4.2 – Fichier fonction_tool_space.py

Ce fichier contient deux fonctions utiles pour effectuer des transformations spatiales :

rotate_mesh

Cette fonction permet d'effectuer une rotation de l'objet autour d'un axe et pour un certain angle. Voici les différentes entrées :

- mesh (type list) : liste sous le format défini par la fonction load_mesh représentant l'objet
- axe (type str) : axe selon lequel on effectue la rotation
- angle_en_deg (type int ou float) : angle en degré de la rotation

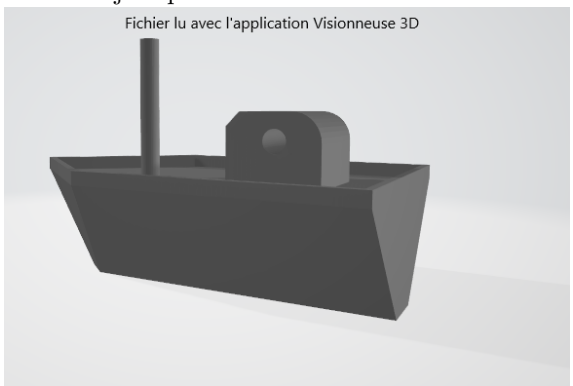
Exemple 4.2.

Voici l'objet à retourner :

Le code :

```
from load import load_mesh

mesh = load_mesh("bateau.stl","binaire")

from fonction_tool_space import rotate_mesh

mesh_apres_rotation = rotate_mesh(mesh,"x",180)

from fonction_generation_fichier_stl import Create_fichier_stl

Create_fichier_stl("bateau_apres_rotation",mesh_apres_rotation)
```

Voici l'objet après rotation :



translation_mesh

Cette fonction permet de translater un objet dans l'espace. Voici les différentes entrées :

- mesh (type list) : liste sous le format défini par la fonction load_mesh représentant l'objet
- translation_selon_x (type int ou float) : translation selon l'axe x
- translation_selon_y (type int ou float) : translation selon l'axe y
- translation_selon_z (type int ou float) : translation selon l'axe z

Exemple 4.3.

```
from load import load_mesh

mesh = load_mesh("bateau.stl","binaire")

from fonction_tool_space import translation_mesh

mesh_apres_translation = translation_mesh(mesh,1,0,6.5)

from fonction_generation_fichier_stl import Create_fichier_stl

Create_fichier_stl("bateau_apres_translation",mesh_apres_translation)
```

4.3 – Calcul_centre_gravite

Cette fonction contenue dans le fichier fonction_centre_gravite.py permet de calculer le centre de gravité de l'objet après simulation à partir de la sortie liste_coordonnees_cordons de la fonction slice_objet :

Exemple 4.4.

```
from load import load_mesh

mesh = load_mesh("mon_fichier.stl","binaire")

from CuttingTools import slice_objet

resultat,liste_coordonnees_cordons = slice_objet(mesh,0.1,False,"z"
    ,30,12,0.1,0,0,"cylindrique",True,["lineaire","decroisee"],1,False,1)

from fonction_centre_gravite impot Calcul_centre_gravite

coordonnees_centre_gravite = Calcul_centre_gravite(liste_coordonnees_cordons
    )
```

4.4 – Fichier fonction_generation_fichier_stl.py

Create_fichier_stl

Cette fonction permet à partir d'un objet mesh de créer un fichier au format *STL* représentant l'objet en 3D.

Exemple 4.5.

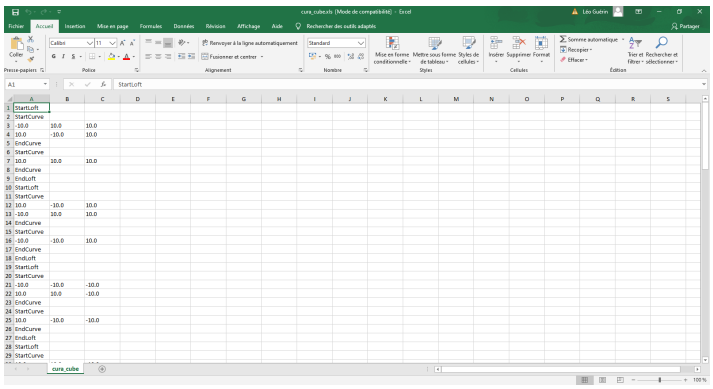
```
from fonction_generation_fichier_stl import Create_fichier_stl

Create_fichier_stl("nom du fichier",mesh)
```

Create_fichier_catia

Cette fonction permet à partir de la sortie liste_coordonnees_cordons de la focntion slice_objet, du rayon des cordons, du type de cordon, du nombre de facettes souhaité et de l'axe de découpe de générer un fichier excel lisible par le logiciel Catia pour générer des surfaces.

Exemple 4.6.

```
from load import load_mesh

mesh = load_mesh("mon_fichier.stl","binaire")

from CuttingTools import slice_objet

resultat,liste_coordonnees_cordons = slice_objet(mesh,0.1,False,"z"
    ,30,12,0.1,0,0,"cylindrique",True,["lineaire","decroisee"],1,False,1)

from fonction_generation_fichier_stl import Create_fichier_catia

Create_fichier_catia("nom du fichier",liste_coordonnees_cordons,rayon_cordon
    ,type_cordon,N_facettes,axe)
```



lecture

Cette fonction permet de lire un fichier :

Exemple 4.7.

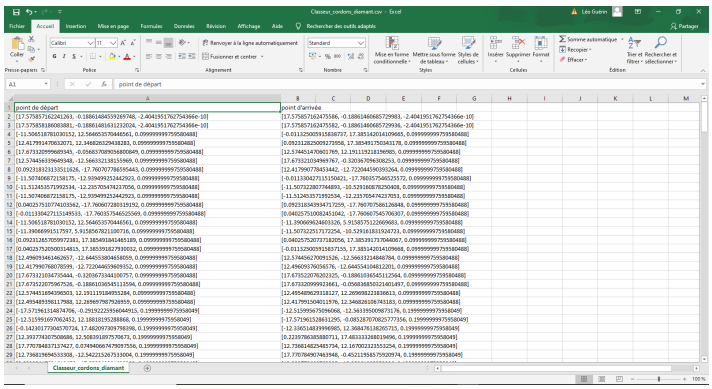
```
from fonction_generation_fichier_stl import lecture

lecture("nom du fichier.extension")
```

Create_fichier_csv

Cette fonction crée à partir de la sortie liste_coordonnees_cordons de la fonction slice_objet un tableau excel (format .csv) où la première colonne correspond aux coordonnées du point de départ de chaque cordon et la seconde aux coordonnées du point d'arrivée de chaque cordon :

Exemple 4.8.

```
from load import load_mesh

mesh = load_mesh("mon_fichier.stl","binaire")

from CuttingTools import slice_objet

resultat,liste_coordonnees_cordons = slice_objet(mesh,0.1,False,"z"
    ,30,12,0.1,0,0,"cylindrique",True,["lineaire","decroisee"],1,False,1)

from fonction_generation_fichier_stl impot Create_fichier_csv

Create_fichier_csv("nom du fichier",liste_coordonnees_cordons,rayon_cordon)
```



Create_fichier_python

Cette fonction permet de créer une copie d'un programme python sans accentuations et c cédille. J'ai mis un nom par défaut pour cette copie : aaaiuvheivuheivh.py pour être sûr de ne pas écraser un fichier python déjà existant mais ce nom peut être changé.

Exemple 4.9.

```
from fonction_generation_fichier_stl import Create_fichier_python

```

```
3 Create_fichier_python("mon_fichier_python_d'origine.py")
```

compte_ligne_fonction

Cette fonction permet de compter le nombre de lignes de codes d'une fonction (hors commentaires et sauts de ligne) contenue dans un fichier python.

Exemple 4.10.

```
1 from fonction_generation_fichier_stl import compte_ligne_fonction
2
3 nombre_de_lignes_de_code = compte_ligne_fonction("mon_fichier_python.py","
      ma_fonction")
```