



Travail pratique de synthèse
M2 TechMed, EUPI

Stratégie de construction de design de stent

Nassim HAMRI
Tristan RENAUD

Janvier 2024

Table des matières

1	Contexte	1
2	Méthodes	1
2.1	Stent et maille	1
2.1.1	Définitions	1
2.1.2	Les éléments élémentaires	2
2.1.2.1	Stratégie de maillage	2
2.1.2.2	Format de données	3
2.2	Spécifications des entrées et des données disponibles	4
2.2.1	Entrées	4
2.2.2	Données disponibles	4
2.2.2.1	Mailles et connecteurs	4
2.2.2.2	Critères de longueur et diamètre	4
2.2.3	Sorties	4
2.2.3.1	Format CSV	5
2.2.3.2	Format IGES	5
2.3	Structuration du programme	5
2.3.1	Programmation orientée objet	5
2.3.1.1	Stent	5
2.3.1.2	Couronne	7
2.3.1.3	Maille	8
2.3.2	Exportation	10
2.3.2.1	Format CSV	10
2.3.2.2	Format IGES	10
2.3.3	Interaction utilisateur	11
2.3.3.1	Interface graphique	11
2.3.3.2	Invite de commande	11
3	Résultats	11
3.1	Interaction utilisateur	12
3.1.1	Interface graphique	12
3.1.2	Invite de commande	12
3.2	Exportations de stents	13
4	Discussion	14
4.1	Manque de données	14
4.2	Manque de validation	15
5	Conclusion	16

Annexe	17
A Mise à jour du programme	17
A.1 Gestion de dossier	17
A.2 Modification du code source	17
A.2.1 Classe Stent	17
A.2.2 Classe Couronne	19
B Exemple de fiche technique constructeur	21

1 Contexte

Ce travail autour de la reconstruction de design de stent s'inscrit dans un cadre plus vaste : le projet VSTENT coordonné par M. Laurent Sarry et soutenu par l'agence nationale de la recherche (ANR-22-CE19-0014). Ce projet vise la simulation temps-réel du déploiement d'un stent dans les artères coronaires à partir des données patient en peropératoire. L'objectif étant d'aider le médecin à éviter un sur-déploiement ou au contraire un sous-déploiement du stent qui peuvent entraîner des risques délétères pour le patient (pouvant nécessiter de déposer un nouveau stent quelques mois ou années après). En effet, la force à appliquer pour un déploiement optimal est très variable : ce sont seulement des années d'expérience qui permettront au praticien d'être à peu près sûr de la force à appliquer.

Pour mener ce projet à bien, une des étapes est de pouvoir obtenir un modèle simulé du stent utilisé par le médecin. Or, il n'existe pas de modèle simulé disponible et encore moins de modèle spécifique à chaque opération, c'est-à-dire : avec un fabricant, un modèle, un diamètre et une longueur spécifiques. Notre projet a donc pour but de pouvoir générer un modèle complet du stent en fonction de ces paramètres.

2 Méthodes

2.1 Stent et maille

2.1.1 Définitions

Dans un premier temps, il nous a fallu bien définir le domaine et les termes utilisés dans ce projet. Pour rappel, les stents étudiés sont des endoprothèses vasculaires visant à maintenir la paroi vasculaire ouverte. Ce sont des tubes maillés constitués de schémas répétitifs.

L'élément élémentaire d'un stent est la maille qui va se répéter dans la longueur et dans la circonférence du stent. On peut également définir la couronne, qui est l'élément circonférentiel qui va se répéter en longueur : elle est généralement constituée de deux à trois mailles. Enfin, entre ces couronnes, on trouve généralement des connecteurs qui servent à maintenir la cohésion de l'ensemble. Les connecteurs entre les mailles au sein d'une même couronne sont généralement des segments au sein des mailles elles-mêmes qui fusionnent entre eux.

2.1.2 Les éléments élémentaires

Notre projet de modélisation en simulation d'un stent nécessite un acquis fondamental : la maille et ses connecteurs. Ce sont ces éléments qui devront être répétés, éventuellement translatés, mais aussi redimensionnés, pour obtenir un stent complet. Mais pour cela, il nous est nécessaire d'avoir une maille et ses connecteurs.

C'est donc à ce niveau qu'intervient le groupe travaillant sur la génération de modèle géométrique de maille de stent. À l'issue de leur travail, un modèle de stent doit être généré avec ses connecteurs associés.

2.1.2.1 Stratégie de maillage Du fait de notre dépendance au travail fourni par ce groupe, il a été nécessaire de discuter de la stratégie adoptée. En effet, le problème de construction peut être vu sous différents angles en fonction des éléments élémentaires à générer pour reconstruire le stent. Nous avons vu principalement deux stratégies différentes de considérer le modèle :

- **La maille comme élément élémentaire avec ses connecteurs :** ici, le plus petit élément du stent est la maille entière avec ses quatre connecteurs (fusionnés ou non à la maille) dans les quatre directions de l'espace.

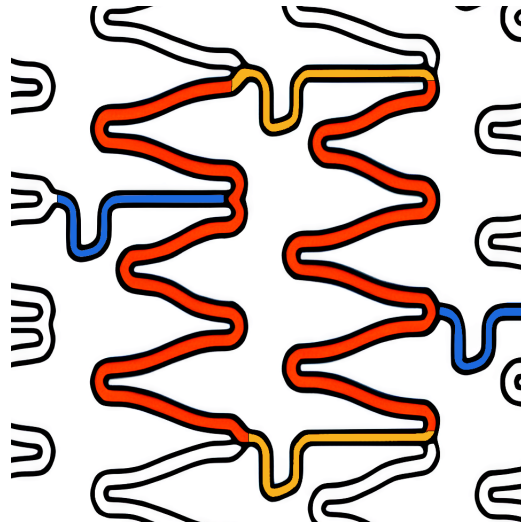


FIGURE 1 – Maille comme élément élémentaire

Ici, on a donc la maille complète comme élément élémentaire en rouge et orange. En orange, on a les connecteurs haut et bas qui sont fusionnés à la maille et en bleu les connecteurs latéraux.

- **Des segments de maille comme éléments élémentaires** : ici, la maille se retrouve divisée en plusieurs segments qui formeront un ensemble d'éléments élémentaires.

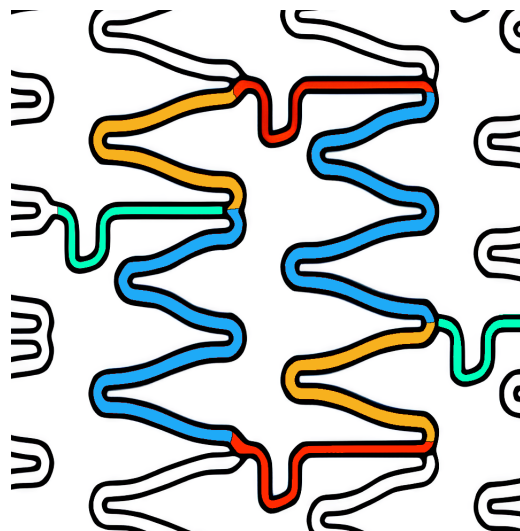


FIGURE 2 – Segments de maille comme éléments élémentaires

Cette fois, les éléments élémentaires ne sont plus que des segments de la maille, de même que pour les connecteurs.

D'après la structure des stents à générer, les deux stratégies nous ont semblé compatibles. Nous avons fait le choix de la première solution qui nous a semblé être, notamment, plus simple à mettre en place en réduisant le nombre d'éléments à répéter.

2.1.2.2 Format de données Nous avons aussi dû convenir d'un format de données dans lequel le premier groupe exporte ses données et dans lequel nous l'importons. Nous avons donc décidé d'utiliser un simple format .csv où la première colonne correspond aux coordonnées en abscisse et en ordonnées pour la seconde. Chaque ligne correspond alors à un point qui forme une arête avec ses voisins.

Par ailleurs, nous avons fait le choix d'un fichier pour la maille complète ainsi que quatre autres fichiers pour les quatre connecteurs.

2.2 Spécifications des entrées et des données disponibles

2.2.1 Entrées

Les entrées du programme sont simplement le nom du fabricant du stent, le modèle, sa longueur et son diamètre. À partir de cela, le programme doit donc être en mesure de créer le stent complet.

Nous avons choisi de concevoir une interface simple qui permet à l'utilisateur de récupérer le fichier .csv correspondant à la maille souhaitée. Il peut ensuite renseigner le diamètre et la longueur souhaités pour lancer l'exportation.

2.2.2 Données disponibles

2.2.2.1 Mailles et connecteurs Les fichiers et mailles et connecteurs associés pour chaque fabricant et modèle sont disponibles dans des sous-répertoires. À l'heure actuelles, seuls les modèles Synergy, Ultimaster Nagomi et Ultimaster Tensei du fabricant Terumo nous ont été fournis.

2.2.2.2 Critères de longueur et diamètre La dernière information qu'il nous manque pour mener à bien la reconstruction sont les critères de déformations et ajouts de mailles en fonction de la longueur et du diamètre. Cette information est cruciale afin de pouvoir coller au mieux à la réalité du stent utilisé par le chirurgien. Cela doit nous parvenir des différents constructeurs, mais, en l'état, sur l'ensemble des documentations qui ont été communiquées, aucune ne précise ces informations.

Nous ne savons donc pas le nombre de mailles par couronne et le nombre de couronnes par stent nécessaires pour les stents. Et donc nous ne pouvons pas non plus régulariser la taille des mailles.

2.2.3 Sorties

La sortie n'est autre que le stent complet généré à partir des entrées dans le corps de notre programme. Le nom du fichier comprend l'ensemble des informations du stent : fabricant, modèle, diamètre et longueur (ex : Stent_Terumo_Synergy_3.0_28.0). Concernant les formats, nous avons décidé d'exporter sous deux formats :

- Un premier format .csv très générique.
- Un second format de conception assistée par ordinateur.

2.2.3.1 Format CSV Pour rappel, le format *comma-separated values* est un format d'exportation largement utilisée grâce à sa simplicité de mise en forme et de lecture. Sa structure est composée de :

- Virgules : pour séparer les colonnes au sein d'une ligne.
- Retour à la ligne : pour séparer chaque nouvelle ligne.

Dans notre cas, nous avons choisi de réaliser deux exports. Un premier fichier contient les coordonnées de l'ensemble des points tel que (X, Y) dans l'ordre de construction. Dans le second fichier, une ligne contient deux points pour former une arête telle que (X1, Y1, X2, Y2). L'ensemble du stent étant vu comme une simple succession d'arêtes.

2.2.3.2 Format IGES Le format de conception par ordinateur utilisé est l'IGES *Initial Graphics Exchange Specification*. En effet, c'est ce format qui a été choisi par le doctorant travaillant sur la suite du projet. C'est un format textuel qui a l'avantage d'être assez universel. Cependant, il demeure peu utilisé, probablement du fait de son ancienneté. En effet, il est hérité de l'ère des cartes perforées et n'est plus mis à jour depuis 1996. Comme on le verra plus tard, cela a rendu l'exportation plus complexe.

2.3 Structuration du programme

Après avoir détaillé les entrées et sorties, vient donc la façon d'organiser et de construire notre programme pour y arriver. Puisque le programme doit faire face à un grand nombre de mailles différentes dont la majorité n'est pas encore disponible, il doit être le plus généralisable possible.

2.3.1 Programmation orientée objet

Comme nous l'avons vu, un stent peut facilement être vu comme la composition de couronnes dans sa longueur. Elles-mêmes composées de mailles dans leur circonférence. Cela nous a donc donné naturellement une organisation en trois classes :

- Stent
- Couronne
- Maille

2.3.1.1 Stent Cette classe est celle de plus haut niveau. Elle a pour données membres les caractéristiques techniques entrés par l'utilisateur. Elle est ensuite composée d'une liste d'objets couronnes avec leurs arêtes associées.

Lors de son initialisation, à l'aide de son constructeur, une boucle va permettre les instanciations : jusqu'au nombre de couronnes défini à l'aide des datasheets constructeurs. À la dernière itération de la boucle, un booléen permet de modifier l'instanciation pour obtenir une maille finale sans connecteur droit. Une instanciación appelle donc un nouvel objet couronne et lui fournit ensuite les caractéristiques techniques du stent et le booléen qui indique si la couronne est finale.

Enfin, une translation est appliquée dans la longueur et la largeur qui correspondent successivement au décalage vers la droite pour chaque nouvelle couronne et au décalage en hauteur pour le bon positionnement des connecteurs.

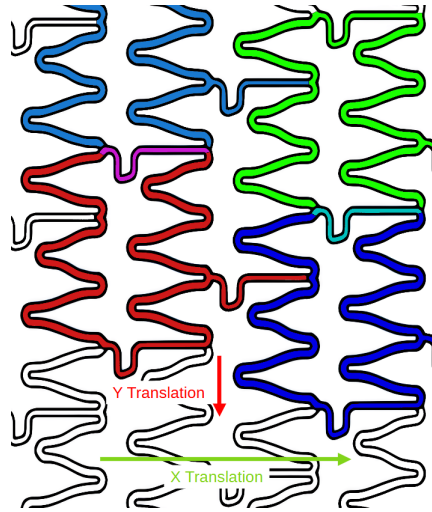


FIGURE 3 – Translation en ordonnée et abscisse entre deux couronnes

L'intérêt de la translation en Y est aidée de ce schéma : en effet, en fonction des modèles de maille, un décalage dans la hauteur peut être induit pour la connexion entre deux couronnes différentes. Il convient de noter que cette translation en ordonnée n'est pas systématique à tous les modèles, par exemple :

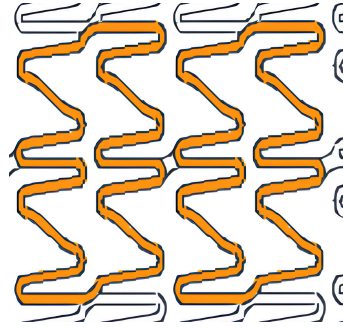


FIGURE 4 – Absence de translation en ordonnée

2.3.1.2 Couronne Dans cette classe de second niveau, on retrouve toujours comme données membres les caractéristiques données par la classe Stent et une liste d'objets couronnes avec leurs arêtes associées. Cette fois, une boucle instance des objets Maille en lui fournissant encore une fois les caractéristiques techniques du stent. Le nombre d'objets Maille instanciés dépend ici du constructeur et modèle (voir 2.2.2.2) : une décision est donc faite à ce niveau sur le nombre de mailles en circonférence.

Aussi, une translation de la hauteur de la maille est appliquée en ordonnée dans le plan à chaque nouvelle maille au sein d'une même couronne. Sans celle-ci, les mailles se superposeraient.

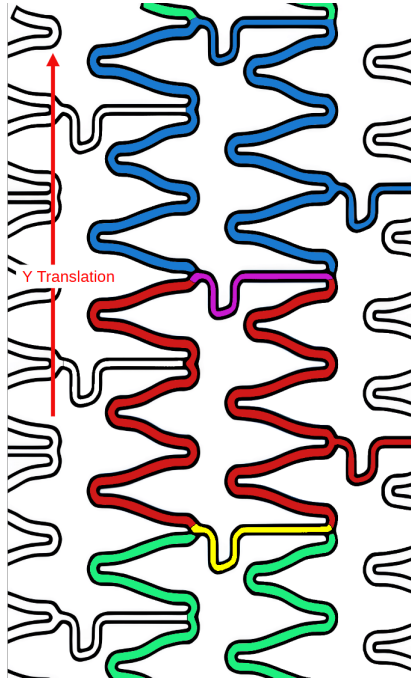


FIGURE 5 – Translation en ordonnée dans la couronne

On imagine ici une couronne composée de 3 mailles (verte, rouge et bleu) dans le plan (elle se replie sur elle-même pour former un tube, en 3D). Les segments jaune et violet correspondent aux superpositions des mailles. On comprend ici le rôle de cette translation en ordonnée.

2.3.1.3 Maille Il s'agit là de la classe de dernier niveau du programme avec pour données membres à nouveau les caractéristiques techniques du stent et les tableaux de points et d'arêtes associés. C'est dans cette dernière classe que sont importés les fichiers CSV du modèle de la maille utilisé (fabricant + modèle) et ses quatre connecteurs associés.

Connecteurs L'apposition des connecteurs se fait naturellement puisque le premier groupe nous renvoie des points tels que les connecteurs possèdent au moins un point de connexion sur la maille (coordonnées du point sur le connecteur et du point sur la maille identiques).

Mise à l'échelle Aussi, c'est à ce niveau qu'on applique les transformations géométriques pour mettre la maille et ses connecteurs à la bonne échelle. En effet, dans les fichiers d'entrées, l'échelle est totalement arbitraire

suite à l'échantillonnage le long des courbes de Bézier. On souhaite ici faire correspondre la taille du stent (diamètre et longueur) avec celle demandée par l'utilisateur. Cela nécessite de prendre en compte le nombre de couronnes et le nombre de mailles par couronne. Pour cette mise à l'échelle, on va simplement multiplier chaque point de la maille et des connecteurs par un facteur d'échelle et x et en y . On a donc e_x :

$$e_x = \frac{\text{longueur}}{nb_{couronne} * (\text{taille}_{maille} + \text{taille}_{connecteur_d})} \quad (1)$$

où :

- longueur représente la longueur rentrée par l'utilisateur
- $nb_{couronne}$ représente le nombre de couronnes du stent (défini d'après les datasheets constructeur)
- $\text{taille}_{maille} = |\max_x(\text{maille}) - \min_x(\text{maille})|$ représente la taille de la maille dans le sens de la longueur
- $\text{taille}_{connecteur_d} = |\max_x(\text{connecteur}) - \min_x(\text{connecteur})|$ représente la taille du connecteur droit dans le sens de la longueur

Et pour e_y :

$$e_y = \frac{\text{diametre} * \pi}{nb_{maille} * \text{taille}_{maille}} \quad (2)$$

où :

- diametre représente le diamètre rentré par l'utilisateur
- nb_{maille} représente le nombre de mailles par couronne (défini d'après les datasheets constructeur)
- $\text{taille}_{maille} = |\max_y(\text{maille}) - \min_y(\text{maille})|$ représente la taille de la maille dans le sens de la hauteur

Les divisions par la taille initiale des objets (maille et connecteur) permettent de normaliser la taille initiale arbitraire entre 0 et 1 puis le reste de l'équation, d'appliquer l'échelle souhaitée.

Routines de translation C'est aussi dans cette classe que sont définies nos deux fonctions de translation vues plus haut (XY et Y) comme elles sont appliquées sur les points de l'objet maille. Concernant leur implémentation :

- Translation XY : on l'utilise sur chaque maille d'une nouvelle couronne. Elle nécessite de connaître la distance de translation vers le haut et vers

la droite de la maille. Pour cela, on calcule :

$$T_{xx} = |\max_x(\text{connecteur}_{\text{droit}}) - \max_x(\text{connecteur}_{\text{gauche}})| \quad (3)$$

$$T_{yy} = |\max_y(\text{connecteur}_{\text{droit}}) - \max_y(\text{connecteur}_{\text{gauche}})| \quad (4)$$

On peut ensuite appliquer ces facteurs en additionnant aux points T_{xx} dans l'axe x et T_{yy} dans l'axe y .

- Translation Y : on l'utilise là dans chaque nouvelle maille d'une même couronne. Elle nécessite de connaître la hauteur d'une maille :

$$T_y = |\max_y(\text{connecteur}_{\text{haut}}) - \max_y(\text{connecteur}_{\text{bas}})| \quad (5)$$

Pour la i -ème maille d'une couronne, on additionne alors aux points $i * T_y$ selon y .

2.3.2 Exportation

2.3.2.1 Format CSV Comme expliqué, nous avons deux formats d'exportation. La première routine fait appel à la bibliothèque csv. On écrit un fichier en parcourant simplement les listes de points puis d'arêtes pour le second fichier. Chaque nouvelle ligne dans une liste correspond à une nouvelle ligne dans le fichier CSV.

2.3.2.2 Format IGES Le second format, IGES, est plus subtile à utiliser puisqu'il fait appel à une syntaxe d'écriture très précise. S'agissant d'un format de fichier textuel, il s'écrit comme un fichier texte au format ASCII. Pour rappel, ce format est standardisé sous la supervision de l'American National Standards Institute. Sa dernière version, la 5.3, date de 1996.

Probablement du fait de l'ancienneté du format, il n'existe pas de librairie officielle disponible sur les indexeurs de paquets comme PyPI. Nous avons trouvé une unique ressource sur github pour l'écriture de ce format. Cette librairie s'utilise simplement en permettant d'écrire des lignes (correspondant à nos arêtes). En revanche, l'exportation prenait près de dix minutes pour un fichier d'une dizaine de mailles. Il nous a donc fallu bien comprendre la structure du format pour comprendre le fonctionnement de la librairie.

En l'occurrence, les deux sections majoritaires de ce format (Directory Entry et Parameter Data) correspondent à la définition du type d'objet (ici une ligne) et de ses paramètres (coordonnées). La librairie utilisait, pour stocker ces deux sections, un string auquel était ajouté une nouvelle chaîne de caractère à chaque itération (162 pour être précis). Les variables string

étant immutables en python, une nouvelle variable devait être créée à chaque itération pour un coût total $O(162n)$. En rempaçant simplement cette implémentation par une liste, on arrive sur un coût de $O(n)$. Le gain de temps moyen est donc d'un facteur 162 : pour une exportation initiale en 10 minutes, on passe à environ 4 secondes.

2.3.3 Interaction utilisateur

Finalement, la boucle *main* se situe au niveau de l'interaction utilisateur. Nous avons fait le choix de proposer deux types d'utilisation différente :

- Une interface graphique (*gui.py*)
- Via invite de commande (*no-gui.py*)

2.3.3.1 Interface graphique L'interface graphique, réalisée à l'aide de la bibliothèque *tkinter*, a l'avantage d'être très simple à utiliser par n'importe qui. L'utilisateur remplit d'abord le chemin vers le fichier CSV d'entrée (dans le dossier *Mailles/Constructeur/Modèle/fichier.csv*). Il peut ensuite entrer le diamètre et la longueur souhaités avant de cliquer sur ok. Une fenêtre s'ouvre pour indiquer que l'exportation est terminée.

Comme indiqué, cette solution est très simple à utiliser. Cependant, l'interface peut rajouter une lourdeur si l'on souhaite réaliser un grand nombre de requêtes (voire utiliser un script).

2.3.3.2 Invite de commande Dans cette seconde méthode, l'utilisateur lance le programme via une invite de commande (MS-DOS ou Bash, par exemple). C'est un peu moins évident à utiliser pour un utilisateur non habitué. Cela nécessite de connaître la syntaxe utilisée : *python no-gui.py constructeur modèle diamètre longueur* (ex : *python no-gui.py Terumo Ultimaster_Nagomi 3 24* pour générer un stent Ultimaster Nagomi du constructeur Terumo, de 3mm de diamètre et 24mm de long). Où *constructeur* et *modèle* correspondent aux sous-répertoires pour accéder à la maille. Cette implémentation peut permettre de lancer un grand nombre de requêtes via un simple script (voire d'organiser un pipeline avec un autre programme par exemple).

3 Résultats

Dans cette section, nous allons succinctement montrer les interfaces utilisateurs et quelques résultats de stents pouvant être obtenus.

3.1 Interaction utilisateur

3.1.1 Interface graphique

Comme expliqué, l'interface graphique est très simple à utiliser. L'utilisateur sélectionne le fichier CSV de la maille souhaitée contenue dans les sous-répertoires *Mailles/Constructeur/Modèle*. Il rentre ensuite le diamètre et la longueur souhaités et clique sur *ok*. L'export se lance automatiquement et une fenêtre indique à l'utilisateur lorsque c'est terminé. Ici, l'utilisateur a sélectionné le modèle *Ultimaster_Nagomi* du constructeur *Terumo* avec un diamètre de 3.0mm et une longueur de 33mm .

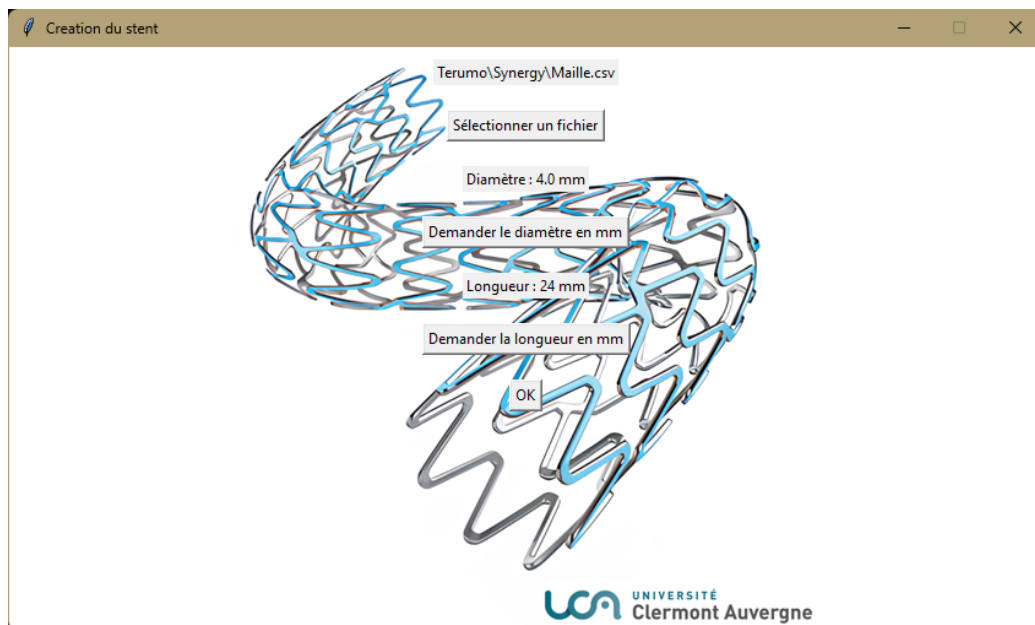


FIGURE 6 – GUI

3.1.2 Invite de commande

La seconde interface est donc directement en invite de commande. L'ajoute de l'option *-v* ou *-verbose* permet d'afficher les données du stent.

```
Anaconda Powershell Prompt x + v
(base) PS C:\Users\Tristan\Desktop\Stent_design_and_construction-main> python .\no-gui.py Terumo Synergy 4 24 --verbose
stent créé
Le fabricant est: Terumo
Le model est: Synergy
Le diamètre est: 4.0
La longueur est: 24.0
fichier exporté
```

FIGURE 7 – Invite de commande

3.2 Exportations de stents

En l'état, nous avons le modèle de maille pour les trois stents Synergy, Ultimaster Nagomi et Ultimaster Tansei du fabricant Terumo. On a réalisé trois exports pour chacun de ces stents avec des paramètres différents :

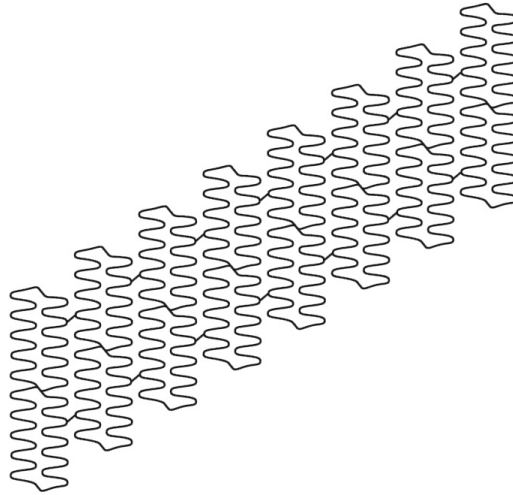


FIGURE 8 – Stent Terumo Synergy, diamètre $3mm$, longueur $24mm$

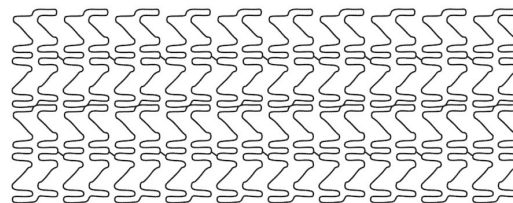


FIGURE 9 – Stent Terumo Ultimaster Nagomi, diamètre $4mm$, longueur $33mm$

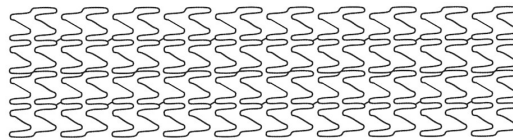


FIGURE 10 – Stent Terumo Ultimaster Tansei, diamètre $4mm$, longueur $50mm$

Les résultats sont cohérents avec ce que l'on attend (longueur et diamètre respectés).

En revanche, du fait du manque d'informations dans les datasheets constructeur, pour les stents de petites longueurs les formes des mailles ne semblent plus cohérentes.

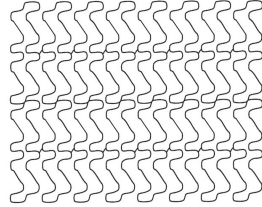


FIGURE 11 – Stent Terumo Ultimaster Tansei, diamètre $2.25mm$, longueur $9mm$

En effet, comme on le voit ici, les mailles sont complètement déformées car 8 couronnes doivent tenir dans $9mm$ (contre $50mm$ sur l'exemple plus haut).

4 Discussion

Dans l'ensemble, nous sommes plutôt satisfaits des résultats obtenus par notre programme. Cela étant, il reste certaines limitations et difficultés rencontrées que nous allons essayer d'explorer ici.

4.1 Manque de données

La première grande difficulté à laquelle nous avons fait face et qui a forcément un impact sur l'algorithme que nous avons implémenté est le manque crucial de données. On parle ici de deux éléments :

- Modèles géométriques de mailles (résultat de l'autre groupe de travail sur lequel on dépend).
- Caractéristiques techniques du constructeur.

Pour le premier élément, la difficulté rencontrée a surtout été de devoir implémenter l'algorithme "à l'aveugle". C'est-à-dire d'implémenter un algorithme sans pouvoir le tester sur des mailles. Cela a surtout été vrai les premières semaines durant lesquelles le premier groupe n'était pas en mesure de nous fournir de modèle de maille. Par la suite, il nous a fourni le modèle de la maille Synergy (Terumo). Par ailleurs, notre programme est supposé pouvoir générer n'importe quel stent à partir du moment où nous avons le modèle d'entrée. Cela nous a obligé à adopter une programmation

générique qui puisse se généraliser au mieux. Cela implique notamment de ne pas implémenter d'exceptions en fonction de tel ou tel stent.

Pour la seconde, il s'agit du manque d'informations techniques de la part des constructeurs sur les fiches techniques à notre disposition. En effet, comme observé sur le stent de la figure 11, nous n'avons pas à disposition les indications sur les facteurs de réduction ou augmentation du nombre de mailles par couronne et de couronne par stent en fonction du diamètre et de la longueur. En l'état, un stent de $9mm$ ou un stent de $50mm$ ont le même nombre de couronnes. Donc forcément, cela induit une déformation importante des mailles dans le sens de la longueur. Cela est également vrai dans la circonférence. En effet, entre un stent de diamètre $2mm$ et un second de $4mm$, on pourrait imaginer que le nombre de mailles est doublé par exemple. Mais en l'état, nous n'avons pas cette information. Ainsi, pour les modèles du fabricant Terumo à disposition, nous avons utilisé le peu de données disponibles pour l'ensemble des modèles qu'il propose, mais cela reste encore largement insuffisant (voir annexe B). Cela étant, l'implémentation que nous avons réalisé permettrait de facilement mettre à jour le code source si ces données venaient à être connues plus tard (voir annexe A).

4.2 Manque de validation

Secondement, il a pu être parfois difficile de bien comprendre ce qu'on attendait de nous, notamment au niveau des fichiers d'exportation. C'est une problématique que nous avons rencontrée surtout sur la fin du projet. Nous aurions aimé pouvoir être assurés que le résultat qu'on obtient correspond à l'attendu et est utilisable pour la suite du projet ANR.

Dans un premier temps, nous nous étions focalisés sur l'exportation du fichier CAO (IGES) suite à des échanges avec M. Sarry qui nous indiquait que ce serait l'idéal pour s'articuler avec un programme réalisé par un doctorant qui prenait un fichier IGES en entrée.

Début janvier, une réunion avec M. Blaysat et M. Boldron nous a même permis de savoir qu'au moins visuellement, les résultats étaient bien cohérents. Nous n'avons en revanche pas eu de retour sur l'utilisation du modèle. Aussi, ils nous ont cette fois demandé une exportation d'un fichier des arêtes du stent. Nous avons donc implémenté de quoi réaliser un export en CSV en plus du format IGES.

5 Conclusion

En résumé, ce travail va, on l'espère, pouvoir contribuer positivement au projet ANR VSTENT grâce à la modélisation possible de n'importe quel stent à partir de paramètres génériques.

A ce stade, il reste déterminant de pouvoir obtenir des informations complètes sur la manière dont les fabricants construisent leurs stents.

Dans cette perspective, nous avons essayé de rendre notre programme modulaire permettant l'ajout ultérieur de toute information utile reçue de la part des fabricants.

Ce projet nous a également permis d'améliorer nos compétences en programmation orientée objet sur python. Cela nous a aussi permis de renforcer des soft skills comme le travail d'équipe au long cours avec les capacités de communication que cela implique ainsi que l'adaptabilité en fonction des quelques retours reçus, notamment.

Table des figures

1	Maille comme élément élémentaire	2
2	Segments de maille comme éléments élémentaires	3
3	Translation en ordonnée et abscisse entre deux couronnes	6
4	Absence de translation en ordonnée	7
5	Translation en ordonnée dans la couronne	8
6	GUI	12
7	Invite de commande	12
8	Stent Terumo Synergy, diamètre $3mm$, longueur $24mm$	13
9	Stent Terumo Ultimaster Nagomi, diamètre $4mm$, longueur $33mm$	13
10	Stent Terumo Ultimaster Tansei, diamètre $4mm$, longueur $50mm$	13
11	Stent Terumo Ultimaster Tansei, diamètre $2.25mm$, longueur $9mm$	14
12	Arborescence de répertoires adoptée	17
13	Extrait de fiche de synthèse technique Terumo	21

Annexe

A Mise à jour du programme

Dans cette section, nous allons préciser la façon dont peut être mis à jour le programme pour utiliser de nouveaux modèles et préciser les conditions de variation du nombre de mailles par couronne et couronnes par stent.

A.1 Gestion de dossier

Premièrement, nous avons construit notre programme pour qu'il aille chercher le fichier du modèle souhaité (.csv) en suivant précisément l'arborescence de répertoire relative (à partir du dossier source du programme) suivante : *Mailles/Constructeur/Modèle/fichier_maille.csv* où **Constructeur** est le nom exact du constructeur du stent (ex : Terumo) et **Modèle** le nom du modèle (remplacer les espaces par un tiret bas par exemple : Ultimaster_Nagomi).

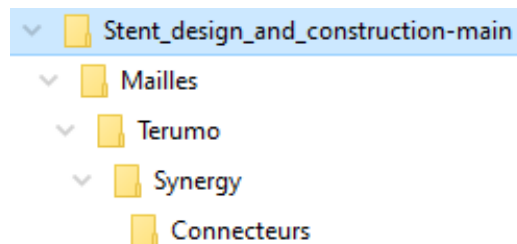


FIGURE 12 – Arborescence de répertoires adoptée

Finalement, le sous dossier **Connecteurs** contient les quatre connecteurs cardinaux du modèle de la maille nommés *Connecteur_bas.csv*, *Connecteur_gauche.csv*, *Connecteur_haut.csv* et *Connecteur_droit.csv*.

A.2 Modification du code source

A.2.1 Classe Stent

La première modification a lieu dans la classe Stent et permet de gérer le nombre de couronnes à utiliser.

```
1 class Stent:
2     """
```

```

3      le stent est une liste de liste ou chaque liste correspond
      ↪ a une couronne
4      '''
5  def __init__(self, constructeur, modele, longueur,
      ↪ diametre, interface=None):
6      self.interface = interface
7      # TERUMO
8      if constructeur == "Terumo" and modele in ["Synergy",
      ↪ "Ultimaster_Nagomi", "Ultimaster_Tansei"]:
9          if longueur in [9, 12, 15, 18, 21, 24, 28, 33, 38,
      ↪ 44, 50]:
10             if diametre in [2.0, 2.25, 2.50, 3.0, 3.50,
      ↪ 4.0, 4.5]:
11                 if 2.0 <= diametre <= 3.0:
12                     nbr_couronne = 8
13
14                 if 3.5 <= diametre <= 4.5:
15                     nbr_couronne = 10
16
17             else:
18                 if self.interface == True:
19                     self.constructeur = 0
20                     return None
21                 else:
22                     raise Exception("Le diametre ne fait
      ↪ pas partie des diametres
      ↪ autorises.\n Le diametre choisi
      ↪ etait : {}".format(diametre))
23
24         else:
25             if self.interface == True:
26                 self.constructeur = 1
27                 return None
28
29             else:
30                 raise Exception("La longueur ne fait pas
      ↪ partie des longueurs autorisees.\n La
      ↪ longueur choisie etait :
      ↪ {}".format(longueur))
31
32     else:

```

```

33         raise Exception("Le constructeur et/ou le modele
        ↪ n\'existe pas.\n Le constructeur choisi était :
        ↪ {}\n Le modèle choisi était :
        ↪ {}".format(constructeur, modele))
34
35     # AUTRES CONSTRUCTEURS      #
36     #
37     #
38     #
39     # AUTRES CONSTRUCTEURS      #

```

Les lignes 8 à 33 du morceau de code extrait du début de la classe Stent permettent d'implémenter les spécificités liées au fabricant Terumo. On vérifie que la longueur et le diamètre entrés sont acceptés par le fabricant pour le modèle utilisé (informations dans la documentation technique). Puis, on peut conditionner le nombre de couronnes utilisé pour modéliser le stent (dépend des informations dans la documentation technique).

A.2.2 Classe Couronne

Dans un second temps, il faut également modifier la classe Couronne pour indiquer le nombre de mailles dans la couronne :

```

1         liste_modele_2_mailles = ["Synergy",
        ↪ "Ultimaster_Nagomi", "Ultimaster_Tansei"]
2         liste_modele_3_mailles = []
3         if modele in liste_modele_2_mailles:
4             couronnes(2)
5         elif modele in liste_modele_3_mailles:
6             couronnes(3)
7
8         # AUTRES NOMBRE DE MAILLES
9         #
10        # ex : Si Boston_scientific & diamètre >=4 alors
        ↪ couronnes = 4
11        #
12        # if constructeur == "Boston_scientific" and diametre
        ↪ >= 4:
13        #     couronnes(4)
14        #
15        # AUTRES NOMBRE DE MAILLES

```

Ici, les trois modèles Synergy, Ultimaster_Nagomi et Ultimaster_Tansei possèdent deux mailles par couronne dans tous les cas. Un jeu de conditions peut permettre de gérer des cas plus compliqués comme indiqué en commentaire du code (lignes 12 et 13).

B Exemple de fiche technique constructeur

On présente ici succinctement le seul extrait de fiche technique fabricant qui nous donne quelques informations sur la manière de construire les stents.

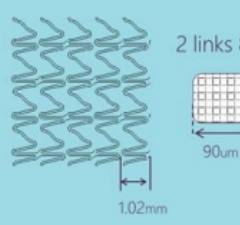
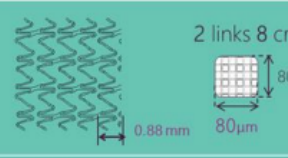

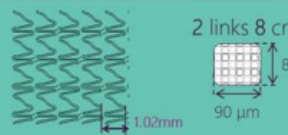
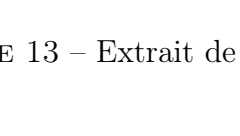
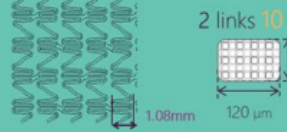
	<i>Ultimaster Tansei</i>	<i>Ultimaster Nagomi</i>
New SV (Small Vessel) Design Φ 2.00-2.50 mm		
MV (Middle Vessel) Design Φ 2.75-3.00 mm		
New LV (Large Vessel) Design Φ 3.50-4.50 mm		

FIGURE 13 – Extrait de fiche de synthèse technique Terumo

D'après notre interprétation, il est indiqué que pour le modèle Ultimaster Tansei, le stent utilise 8 couronnes et 2 mailles par couronnes pour l'ensemble des diamètres. Pour le modèle Ultimaster Nagomi, le stent utilise 8 couronnes et 2 mailles par couronnes pour des diamètres inclus dans $[2.00; 3.00]mm$ et 10 couronnes pour des diamètres inclus dans $[3.50; 4.50]mm$.

Ce sont donc ces informations qu'on a utilisées pour les conditions entre les lignes 11 et 15 de l'annexe A.2.1. On a supposé que le modèle Synergy fonctionnait comme le modèle Tensei. On n'a pas d'information sur les spécifications en longueur.