Projet spécial LOG791

Département de génie logiciel et des TI

Développement d'un algorithme de découpage par couche à diamètre variable de modèle 3D.

Aptus Tech.

Auteurs

Jacob Cossette
COSJ08079801

Professeur superviseur

Camille Coti

Date

18 décembre 2024

Suivi des changements

*__A__ – Ajouté **M** – Modifié **S** – Supprimé

NUMÉRO DE VERSION	DATE aaaa/mm/jj	NUMÉRO DE FIGURE, TABLE OU SECTION	A* M S	BRÈVE DESCRIPTION DU CHANGEMENT	NUMÉRO DE DEMANDE CHANGEMENT
1.0	2024/09/25		A	Publication Initiale	
2.0	2024/12/15		A	Publication Final	

TABLE DES MATIÈRES

TABLE DES FIGURES

1. PROBLÉMATIQUE ET CONTEXTE

L'entreprise Aptus Tech, participant à la cohorte d'automne 2024 du Centech, et moi-même, cherchons à implémenter un algorithme de découpage par couche à diamètre variable pour les modèles 3D. Le développement de la technologie de tête d'impression à diamètre variable d'Aptus Tech nécessite l'élaboration d'un nouvel algorithme de découpage lors de la préparation des modèles 3D pour l'impression. Ce nouveau type d'impression 3D présente plusieurs défis, notamment en ce qui concerne les logiciels de slicing existants et les algorithmes utilisés pour générer le découpage des surfaces et des structures internes des objets 3D. Actuellement, les technologies disponibles pour gérer ce type de découpage sont limitées, ce qui pousse l'équipe à explorer diverses solutions possibles.

2. OBJECTIFS DU PROJET

Tel qu'expliqué dans la problématique, l'objectif principal de ce projet est d'initier une recherche préliminaire en vue du développement d'un algorithme de découpage pour l'impression 3D à diamètre variable. Les objectifs spécifiques de ce document sont les suivants :

1. **Développement d'un algorithme de preuve de concept** : Concevoir et développer un algorithme capable de gérer la variation de l'épaisseur des

couches d'impression pour des objets simples tels que des cubes, des pyramides, et autres formes géométriques basiques.

2. **Création d'un prototype fonctionnel** : Développer un prototype qui, à partir d'un fichier d'objet 3D, pourra générer le découpage des couches d'impression en fonction de paramètres précis tels que la rapidité, la précision, et d'autres critères à définir. Ce prototype permettra d'évaluer les performances de l'algorithme dans un environnement proche des conditions réelles.

3. STRUCTURE DU DOCUMENT

La structure du document débutant par l'exploration des différentes techniques étudiées, incluant l'intersection des triangles et du plan pour la précision des découpes et la mesure des zones d'intérêts. On examine également les outils logiciels testés et envisagés, tels qu'OpenGL, Assimp et Slic3r, et leurs avantages et limitations pour ce projet. L'avancement des travaux est ensuite discuté, en concluant avec une évaluation des résultats actuels et les perspectives.

4. SOMMAIRE DES TECHNIQUES ÉTUDIÉES

Pour développer cet algorithme, il est important de définir les étapes clés du processus de découpage à diamètre variable. Voici une description des phases principales:

1. **Import du fichier STL** : Le processus commence par l'importation du fichier de l'objet 3D au format STL dans l'environnement logiciel de découpage. On utilise les types de formats ASCII STL où chaque sommet est représenté par trois vertex

   Un triangle T est donc défini par :

$$T = \{(v1, v2, v2), n\}$$

Où v1, v2, v3 représentent des vertex et n la normale.

2. **Définition des paramètres** : Plusieurs paramètres doivent être spécifiés, dont l'un des plus importants est la tolérance d'erreur maximale autorisée entre le modèle STL et les couches d'impression. Cette tolérance définit la précision requise pour chaque couche.

Tableau 1: Tableau des paramètres

Paramètre	Description	Impact
Épaisseur de couche maximale et minimale (layer_height_max)	Définis l'épaisseur des couches.	Plus les couches sont fines, plus la précision est élevée, mais le temps d'impression augmente.
Erreur maximale autorisée (Cusp Height) (max_error)	Limite l'erreur entre la surface imprimée et le modèle 3D réel.	Moins l'erreur tolérée est élevée, plus la précision augmente, mais le temps d'impression aussi.

3. **Génération d'un premier découpage** : Un premier découpage grossier est généré, basé sur les paramètres de l'utilisateur, comme le diamètre de la tête d'impression. Ce découpage servira de point de départ pour l'optimisation.

4. **Analyse de précision** : Une boîte de précision est ensuite appliquée pour identifier les zones où l'erreur entre la surface découpée et le modèle 3D dépasse un seuil prédéfini. Ces zones nécessitent une attention particulière et un redécoupage pour améliorer la précision de l'impression.

5. **Génération des couches intermédiaires** : Aux endroits identifiés comme critiques (où la précision ne satisfait pas les attentes), des couches intermédiaires sont ajoutées. Cela permet d'augmenter la définition et la qualité de l'impression sur ces zones.

6. **Génération du code G optimisé** : Enfin, un code G optimisé est généré, prenant en compte le chemin de la tête d'impression et les ajustements apportés pour améliorer la qualité d'impression dans les zones critiques.

4.1 Intersection des triangles du plan

Pour chaque arête qui croise le plan de découpe dans un triangle donné, nous trouvons le point d'intersection entre l'arête et le plan en utilisant l'interpolation linéaire. Supposons que les sommets V1 et V2 d'un triangle soient situés respectivement au-dessus et en dessous du plan de découpe. En appliquant l'interpolation linéaire, nous calculons le point d'intersection *p* sur cette arête.

La formule d'interpolation est la suivante :

$$P = v_1 + t \cdot (v_2 - v_1) \qquad (1)$$

P représente le point de croisement, v1 et v2 des vertex du triangle, t est le coefficient pour localiser le point sur l'arête.

Où :

$$t = \frac{z_i - z_1}{z_2 - z_1} \qquad (2)$$

Z1 et z2 représentent la composante z des vertex, zi la composante z du plan courant.

En identifiant tous les points d'intersection des arêtes de chaque triangle d'un fichier STL, on peut ensuite relier ces points pour former le contour de la couche. Ce contour, sous forme de polygone, représente la limite de la section transversale de l'objet à cette hauteur.

Ces contours servent alors de base pour générer le G-code, qui définit la limite du chemin que suivra la tête d'impression pour remplir la couche. En utilisant cette approche, il devient également possible de calculer l'erreur entre les contours des couches générées à chaque hauteur, en comparant la distance des points du contour actuel par rapport aux contours théoriques de l'objet 3D, permettant ainsi un ajustement dynamique de l'épaisseur des couches pour maximiser la précision.

4.2 Mesure des zones d'intérêts

Le paramètre principal de cet algorithme est l'erreur maximale autorisée entre les couches d'impression, mesurée par la *cusp height, est* un concept discuter par K. Mani & al (1998) [1]**.** La *cusp height* correspond à la hauteur de l'ondulation ou de la déviation entre deux couches successives d'impression par rapport à la surface idéale du modèle 3D. En d'autres termes, il s'agit de la distance entre le point le plus éloigné d'une couche et sa position théorique sur le modèle numérique. Ce paramètre est crucial, car une *cusp height* trop élevée peut entraîner des irrégularités visibles sur la surface imprimée, compromettant ainsi la qualité de l'objet final.

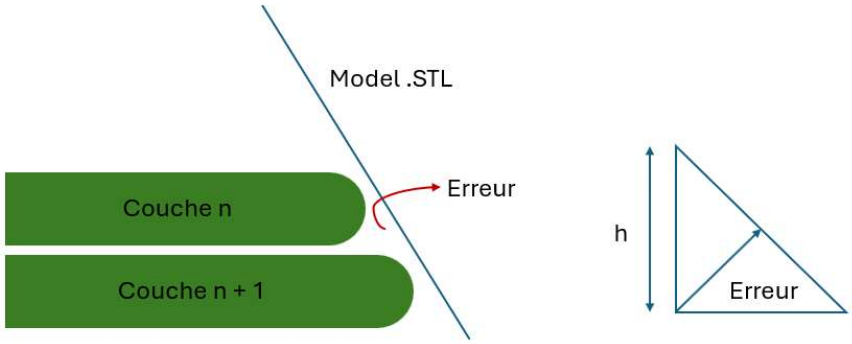

Figure 1: Représentation du cusp height

En fixant une erreur maximale via la *cusp height*, l'algorithme identifie les zones où cette déviation dépasse le seuil défini, permettant d'ajuster localement l'épaisseur des couches. Cela améliore la précision des détails critiques et minimise les irrégularités de surface. Une gestion efficace de la *cusp height* est essentielle pour obtenir une impression de haute qualité, en particulier dans les zones où des détails fins ou des courbes complexes sont présents. En ajustant la *cusp height*, l'algorithme peut optimiser la précision des surfaces tout en maintenant une vitesse d'impression raisonnable.

$$h = r \cdot (1 - \cos(\frac{\theta}{2}) \qquad (3)[1]$$

- h : Cusp height.
- r : Rayon local de courbure de la surface.

- θ : Angle entre deux points adjacents du contour.

Bien que d'autres approches existent dans la littérature pour calculer l'erreur relative entre les couches d'impression et le modèle STL, celle-ci a été choisie pour sa simplicité.

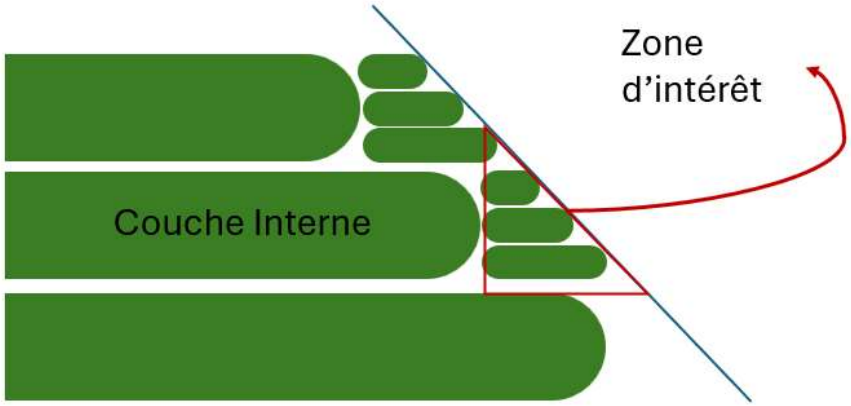

Figure 2: Représentation des répartitions des couches d'impression

Une fois les zones d'intérêt définies, un algorithme de découpage précis est appliqué à ces zones, principalement sur les bordures de l'objet. L'un des défis consiste à s'assurer que le diamètre d'impression dans ces zones soit un multiple des couches internes pour éviter un décalage entre les couches, ce qui pourrait entraîner des erreurs d'alignement.

Par exemple, si l'utilisateur choisit un diamètre d'impression de 0,3 mm, les zones nécessitant une précision accrue pourraient être imprimées à 0,1 mm, de façon à

garantir que l'addition des couches atteigne 0,3 mm, évitant ainsi tout décalage ou chevauchement des couches.

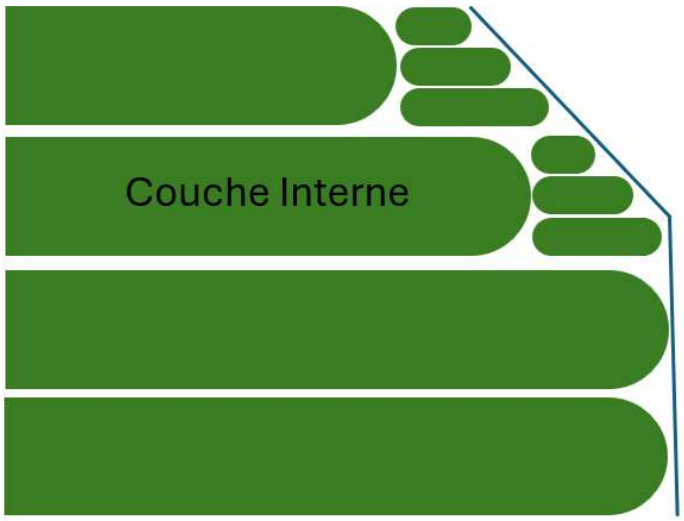

Figure 3: Représentation du modèle complet

Le concept de zones d'intérêt permet d'accélérer le processus d'impression dans les zones moins critiques tout en améliorant la précision dans les zones complexes. Cela réduit le besoin de reprises et de redécoupages dans certaines parties de l'objet, optimisant ainsi la vitesse et la qualité globale de l'impression.

5. MÉTHODOLOGIE EXPÉRIMENTALE

Dans cette section, nous détaillons les différentes approches et outils explorés pour le développement d'une plateforme de lecture de fichiers STL et l'implémentation

d'algorithmes de découpage. Voici un aperçu des techniques étudiées jusqu'à présent.

5.1 Open GL

Au départ, l'utilisation d'OpenGL a été envisagée pour sa flexibilité et son contrôle précis sur le rendu graphique et l'exécution des algorithmes. L'objectif initial était de créer une interface graphique permettant d'afficher les fichiers STL dans une fenêtre dédiée, étape essentielle au développement d'algorithmes de découpage. Bien que cette approche semblât simple, plusieurs défis techniques ont émergé :

- **Erreur OpenGL (GLFW error 65542: WGL)** : Dès le lancement, une incompatibilité avec la carte graphique NVIDIA GTX 680 a généré des erreurs liées à OpenGL, empêchant le bon fonctionnement de l'application. L'erreur provenait de l'absence de support pour les nouvelles versions d'OpenGL par le matériel disponible. En l'absence de solutions sans frais supplémentaires, il a été nécessaire de changer de machine. [2]

En développant l'interface, trois librairies principales ont été utilisées : GLEW [3], GLFW [4], et GLM [5]. Bien que ces librairies aient été intégrées avec succès, la création des fichiers CMake pour l'importation de ces dépendances fut utilisée pour

simplifier l’installation du projet. Les scripts CMake utilisés sont disponibles dans le dépôt du projet.[1]

Créer un visionneur STL performant capable de gérer de grands fichiers et des fonctionnalités basiques (translation, rotation, etc.) s'est avéré plus complexe que prévu. Compte tenu des objectifs ambitieux du projet et du temps limité, l'option d'OpenGL a finalement été abandonnée pour se concentrer sur des solutions existantes mieux développées.

[1] https://github.com/Jacob-Cossette/aptus-slicer2

5.2 Assimp

Afin de contourner certaines des limitations rencontrées avec OpenGL, nous avons exploré l'utilisation de la librairie Assimp (Open Asset Import Library). Cette librairie permet l'importation de fichiers 3D, dont les fichiers STL, et supporte plus de 40 formats différents, ce qui la rend très polyvalente. Assimp propose également une série d'outils mathématiques utiles pour le projet, comme la triangulation, l'extraction de vertex, et la gestion des normales et tangentes. [6]

Assimp possède une architecture bien développée avec des tests déjà intégrés dans le pipeline, facilitant l'ajout de nouvelles fonctionnalités. Nous avons envisagé d'implémenter un algorithme de découpage directement dans cette librairie afin d'exploiter sa capacité à visualiser les fichiers STL.

Bien que prometteuse, cette librairie nécessite l'implémentation complète de la génération d'un tool paths pour offrir une solution de découpage fonctionnelle. Alors, en raison des contraintes de temps, il est peu probable qu'Assimp puisse être utilisée comme solution finale. Elle pourrait néanmoins constituer un excellent outil pour le développement futur d'un logiciel de slicing complet, bien qu'elle ne soit pas parfaitement adaptée à nos besoins immédiats pour le développement des algorithmes de découpage.

5.3 Slic3r

Face aux limitations d'Assimp, nous avons exploré l'option d'utiliser un slicer open source existant, tel que Slic3r. Slic3r est un outil open source bien établi pour la découpe de modèles 3D en couches, avec un algorithme de génération de tool path déjà intégré.

L'utilisation d'un slicer open source comme Slic3r permet de se concentrer uniquement sur l'implémentation de l'algorithme de découpage sans avoir à développer toute l'infrastructure logicielle autour du rendu et de la visualisation des fichiers STL. Cela simplifie considérablement le projet. [7]

Cependant, cette approche présente certaines limitations, notamment un manque de contrôle sur l'environnement global et sur l'algorithme de génération du tool paths déjà intégré. Cela peut restreindre les ajustements et personnalisations que nous souhaitons apporter.

5.4 Solution sur mesure

Pour pallier les divers défis rencontrés par les solutions précédentes, on a opté pour l'utilisation d'un projet beaucoup plus simple afin d'expérimenter avec différents algorithmes. À cet effet, le code de l'algorithme de *slicing* développé par Aaron Schaer et Michelle Ross a été utilisé. Ce projet, programmé en Python, prend en entrée des fichiers STL en format ASCII et génère des fichiers G-code.

Dans le but de déterminer l'impact du *slicing* adaptatif, on a utilisé le site *Gcode.ws*, qui fournit un outil de visualisation couche par couche en 3D ainsi qu'une variété de métriques.

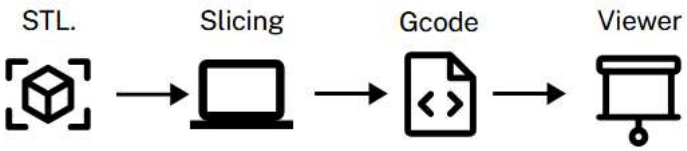

Figure 4: Pipeline pour la génération et visualisation

5.4.1 Détail de la solution

Plusieurs classes d'objets sont utilisées dans le programme. L'une d'entre elles, qui nous intéresse particulièrement, est la classe **Slice**, qui représente une tranche à une hauteur Z donnée du modèle. Elle contient toutes les informations dans l'espace

pour générer le code G. L'idée derrière notre modification était de définir une nouvelle dimension de cette tranche comme étant une subdivision, représentée par une série de points d'intersection entre les triangles du modèle STL et la hauteur Z_slice plus la moitié de l'épaisseur d'impression.

Tableau 2 : Objet du programme

Objet	**Description**
Point	Représente un point dans l'espace 3D avec des coordonnées x, y et z.
Line	Représente une ligne définie par deux points (p0 et p1).
Triangle	Représente un triangle défini par trois points (p0, p1, p2) et une normale (norm).
Slice	Représente une tranche d'un modèle 3D avec des périmètres, des sous-périmètres, des supports et des infills.

Pour éviter un *refactor* complet du fonctionnement du découpage, seule la prise en compte des périmètres a été envisagée. En effet, la génération du code se base sur les périmètres pour produire le remplissage intérieur de la pièce. Ainsi, en introduisant un nouveau sous-périmètre, il est possible de commencer le remplissage à partir de ce nouveau segment de manière simple et efficace.

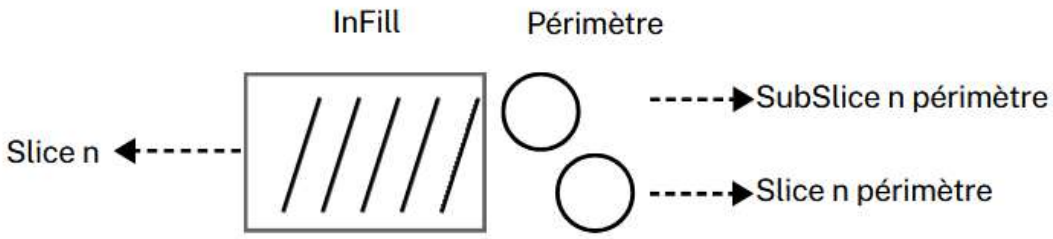

Figure 5: Représentation d'une Slice avec le diamètre adaptatif

Pour générer les sous-périmètres, des ajouts ont été faits à plusieurs fonctions. Premièrement, les attributs *sous-périmètre 1* et *sous-périmètre 2* ont été ajoutés au constructeur de *slice*. L'idée derrière cet ajout était d'introduire les dimensions nécessaires pour permettre plusieurs couches d'impression sur le périmètre. Un sous-périmètre supplémentaire a également été ajouté en prévision d'une troisième couche d'impression dans des ajouts futurs.

Il est à noter que cette configuration n'est pas encore disponible. Un nombre dynamique de sous-périmètres serait plus approprié dans une version finale, ce qui sera discuté dans la section *Limitation*.

Pour générer les sous-périmètres, des modifications ont été apportées au code. Dans un souci de modularité, un paramètre arg dans la commande précise l'utilisation de l'impression à diamètre dynamique. Lorsque ce paramètre est activé, la fonction separateSubSlices suit la même méthodologie que la fonction

separateSlices. Pour chaque périmètre et sous-périmètre, il y a interpolation des points d'intersection.

Si plus de deux points d'intersection sont rencontrés sur un triangle donné à un plan Z, ces points sont ajoutés au sous-périmètre. Ensuite, pour la génération dynamique du G-code, une section a été ajoutée pour la lecture des sous-périmètres et la génération des déplacements associés.

Tableau 3 : Paramètre d'entrée de slicing.py

Paramètre	**Description**
Fichier STL	Fichier STL qui sera généré en opération code G
Diamètre	Diamètre d'impression pour la génération du code G en mm.
Infill	Pourcentage de remplissage de l'intérieur.
Dynamic	Paramètre 1 ou 0 ou un1 représente un découpage dynamique en deux sous plan pour chaque slice.
python slicing.py [Fichier STL] [Diamètre] [Infill] [Dynamic]	

Pour utiliser la solution, il est recommandé d'utiliser un environnement Anaconda avec Python **3.8**. Les commandes ci-dessous et leurs arguments permettent d'appeler la génération du fichier gcode à partir d'un fichier STL.

6. RÉSULTATS

Les résultats présentés ci-dessous ont été obtenus à l'aide de plusieurs fichiers STL représentant des objets simples. L'objectif est de comparer les performances d'impression selon les indicateurs présentés dans le tableau 4. Afin d'assurer une conformité entre les résultats, les paramètres suivants ont été utilisés : un remplissage à 50 %, un coût de filament de 0,08 $/gramme, une vitesse d'extrusion de 15 mm/s et une vitesse de déplacement de 45 mm/s.

Lorsque l'on observe les résultats, on conclut que le découpage dynamique est plus performant qu'une impression à diamètre constant de 0,5 mm. Cependant, il est moins performant qu'un découpage à diamètre constant de 1 mm. Bien qu'il y ait une légère différence de performance, la méthodologie dynamique permet d'atteindre le niveau de précision d'une impression à 0,5 mm avec une augmentation de temps de seulement 18,54 %, contre une augmentation de 147,7 % pour passer d'un diamètre constant de 1 mm à 0,5 mm.

On remarque aussi l’augmentation radicale lorsque les objets sont plus complexes. Lorsque l’on compare les résultats entre du temps d’impression entre le cube et la sphère, on remarque une augmentation importante, encore plus marqué pour e fichier d’anneau.

	Cube			*Sphère*			*Anneau*		
Diamètre d'impression (mm)	0.5	1.0	Variable	0.5	1.0	Variable	0.5	1.0	Variable
Temps d'impression (s)	2097.2	1083	1146	11237.8	7497	9296	55094	13790	17350
Nombre de couches	40	20	20	40	20	20	20	10	10
Masse de l'objet (g)	2.2865	1.17	1.28	41.42	10.55	13.01	51.52	14.24	21.57
Utilisation de filament (mm)	766.41	391.41	463.85	16560	4220	5203	20598	5692.53	8624.01
Coût d'impression	0.1824	0.0936	0.1024	2.07	0.53	0.65	2.58	0.71	1.7256

Tableau 4 : Résultats des différents modèles testés.

6.1 Limitations

Malgré que la preuve de concept ait atteint certains objectifs pendant le projet, plusieurs limitations ont été rencontrées. Cette section s'intéresse aux limites de la solution actuelle d'un point de vue fonctionnel et opérationnel.

6.1.1 Opérationnel

Au niveau opérationnel, plusieurs limitations sont présentes. En effet, sans l'utilisation d'une imprimante *open source* avec un firmware *open source*, il sera complexe d'intégrer cette solution. Les opérations de variation de diamètre dans le G-code ne sont pas prises en charge par les firmwares actuels, nécessitant ainsi des adaptations pour exploiter pleinement les fichiers G-code générés.

Au-delà de ce problème, le manque de paramètres tels que le type de motif d'infill, l'épaisseur des surfaces et d'autres paramètres importants en impression 3D limite les possibilités de personnalisation. De plus, une plateforme d'impression doit idéalement inclure un visionneur intégré, ce qui n'est pas le cas de notre solution actuelle.

Bien que le site web *gcode.ws* [8] permette de visualiser les résultats pour évaluation, les fonctionnalités de notre pipeline restent limitées, et l'expérience

utilisateur n'est pas adaptée à une utilisation opérationnelle dans un environnement industriel d'impression 3D.

6.1.2 Fonctionnel

Plusieurs limitations dans les fonctionnalités de la solution actuelle sont présentes et devront être adressées pour fournir une solution complète. Dans cette sous-section, nous nous intéressons aux limitations techniques et explorons des solutions potentielles pour les résoudre.

***Overlap* entre le remplissage et les diamètres des slices**

Premièrement, nous constatons plusieurs chevauchements entre le remplissage interne (*infill*) et les diamètres des slices. Pour éliminer ces chevauchements, il serait possible de modifier la référence entre le périmètre du slice et le sous-périmètre afin d'éviter tout recouvrement durant l'impression. À la figure 6, une représentation d'un slice montre le remplissage interne ainsi que les traces d'impression des périmètres et sous-périmètres.

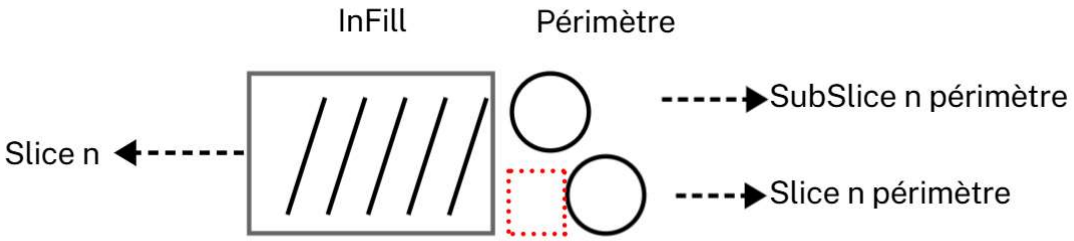

Figure 6: Représentation d'un plan n sous-remplie

Dans la configuration actuelle, un espace vide se forme entre l'impression du périmètre et le remplissage interne, représenté en rouge sur la figure. Cet espace compromet la solidité de l'impression. Des tests de compression et de traction seraient intéressants à réaliser pour évaluer les implications de ce vide.

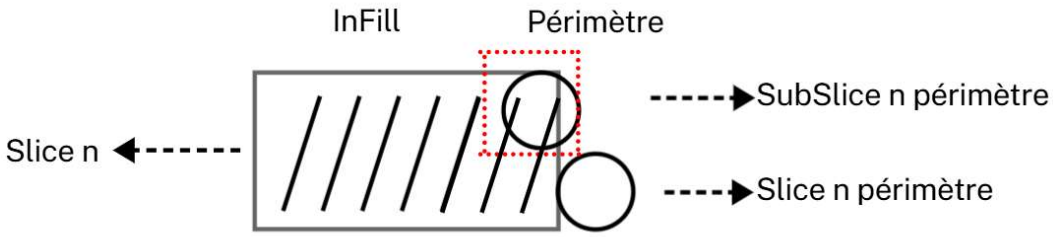

Figure 7: Représentation d'un plan n sur remplie

À l'inverse, la figure 7 illustre une situation différente : si la référence utilisée pour le remplissage interne correspond uniquement au périmètre initial, certaines portions du remplissage viennent se superposer au sous-périmètre. Cette superposition est problématique, car elle pourrait provoquer des erreurs d'impression majeures. Ce

type de problème peut survenir facilement. Par exemple, sur un même slice, une partie du périmètre initial peut correctement délimiter la zone de remplissage interne, tandis qu'une autre section superpose le sous-périmètre. Dans sa configuration actuelle, la solution ne permet pas de généraliser efficacement notre approche.

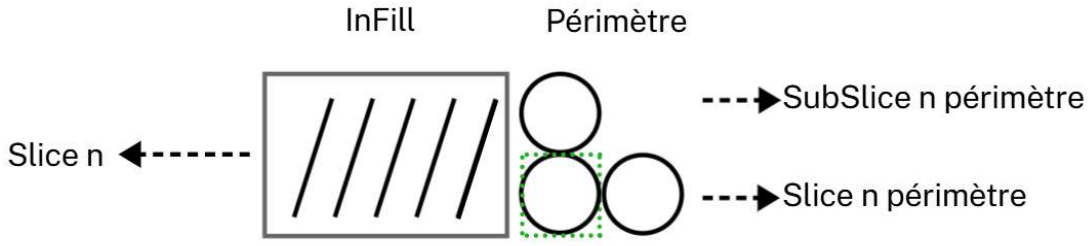

Figure 8 : Représentation d'un plan n sans anomalie

Une piste de solution pourrait consister à définir chaque périmètre et sous-périmètre à l'aide d'une épaisseur propre, composée de plusieurs lignes ajustées pour s'aligner correctement, comme illustré à la figure 9.

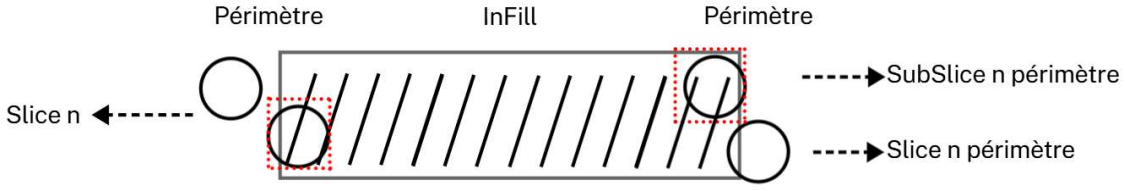

Figure 9: Représentation d'un plan n avec inversion de sous périmètre

Problèmes de temps de traitement

En termes de temps de traitement, certaines limitations sont également présentes. Par exemple, la génération des fichiers G-code est relativement lente malgré la simplicité des modèles. Pour un cube de 20 mm x 20 mm x 20 mm, le processus prend 32 secondes sur un processeur Intel Core i7-6600U. Avec l'augmentation de la complexité géométrique, comme pour une sphère, le temps de génération atteint 103 secondes et dans le cas de l'anneau on atteint 1457 secondes. Ce problème est amplifié par la complexité croissante des objets imprimés, ce qui pose un défi important dans des situations réelles.

L'une des solutions envisagées pour corriger cette problématique est de réécrire l'algorithme dans un langage plus optimisé que Python, comme le C++.

Problème optimisation de mouvement

Bien que la vitesse de déplacement soit considérablement rapide sur les imprimantes modernes, l'algorithme développé ne fournit pas une solution optimisée pour réduire les déplacements inutiles. En effet, comme on peut l'observer dans la Figure 10, plusieurs opérations de déplacement inverses sont effectuées sans réelle nécessité.

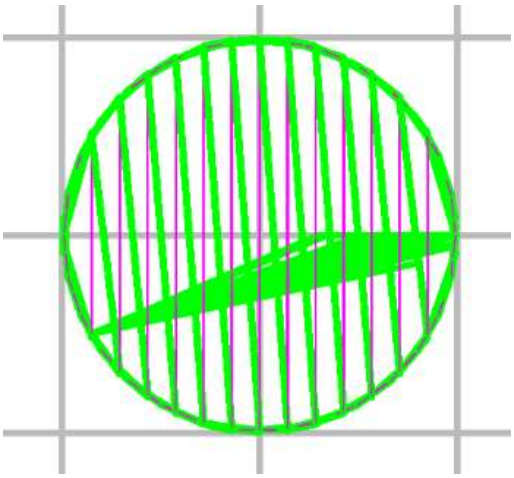

Figure 10: Déplacement pour l'impression d'une couche de sphère

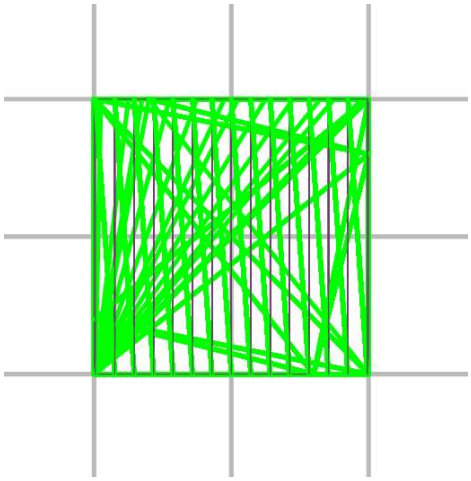

Figure 11: Déplacement pour l'impression d'une couche d'un cube

7. CONCLUSION

En résumé, jusqu'à présent, nous avons partiellement atteint les objectifs de ce travail. En effet, une revue de littérature a été réalisée pour comprendre les techniques modernes de développement d'algorithmes de découpage dynamique. Un prototype fonctionnel d'algorithme a été développé pour analyser les gains potentiels d'une impression à diamètre dynamique. Cependant, l'objectif de développer une plateforme de test pouvant être utilisée lors des essais d'impression a été atteint de manière partielle en raison de différentes limitations, telles que les chevauchements entre les couches, le temps de traitement lors de la génération du code G et, finalement, les défis liés à l'optimisation des mouvements.

Pour la suite, il serait pertinent de résoudre les limitations identifiées dans le projet développé, en particulier celles liées aux chevauchements entre le remplissage et les périmètres. Une autre piste de développement serait de finaliser le calcul de la *cusp height* afin d'améliorer dynamiquement la qualité de la surface.

8. BIBLIOGRAPHIES

[1] K. Mani, P. Kulkarni, and D. Dutta, “Region-based adaptive slicing,” *Comput.-Aided Des.*, vol. 31, no. 5, pp. 317–333, Apr. 1999, doi: 10.1016/S0010-4485(99)00033-0.

[2] “Error: “GLFW error 65542: WGL: The driver does not appear to...,” Intel. Accessed: Oct. 25, 2024. [Online]. Available: https://www.intel.com/content/www/us/en/support/articles/000058790/graphics.html

[3] “GLEW: The OpenGL Extension Wrangler Library.” Accessed: Oct. 25, 2024. [Online]. Available: https://glew.sourceforge.net/

[4] Luke’s Dev Tutorials, *Modern OpenGL Tutorial: Rendering a Colored Cube*, (Oct. 05, 2023). Accessed: Oct. 16, 2024. [Online Video]. Available: https://www.youtube.com/watch?v=-Vyq221eEmU

[5] “GitHub - g-truc/glm: OpenGL Mathematics (GLM).” Accessed: Oct. 25, 2024. [Online]. Available: https://github.com/g-truc/glm

[6] *assimp/assimp*. (Oct. 21, 2024). C++. Open Asset Import Library. Accessed: Oct. 21, 2024. [Online]. Available: https://github.com/assimp/assimp

[7] “Slic3r - Open source 3D printing toolbox.” Accessed: Oct. 25, 2024. [Online]. Available: https://slic3r.org/

[8] “gCodeViewer - online gcode viewer and analyzer!,” Online GCode Viewer. Accessed: Dec. 05, 2024. [Online]. Available: http://gcode.ws

ANNEXE 1 : INSTRUCTION D'INSTALLATION

```
### Slicer

Ce script Python découpe des modèles 3D à partir de fichiers STL pour générer
du G-code avec des infill (remplissage) et des supports.

### Crédits
Cet algorithme a été écrit par Aaron Schaer et Michelle Ross, avec l'aide de
Steven Hernandez, pour la fabrication numérique et adapter par Jacob Cossette
pour l'impression à diamètre dynamique.

### Utilisation
Notre script prend exactement trois entrées :

1. Le nom du fichier STL à découper
2. L'épaisseur souhaitée de chaque couche en millimètres
3. Le pourcentage de remplissage désiré, compris entre 0.0 et 1.0
4. L'instruction pour le type d'impression (0 ou 1)

Le script fonctionne de manière optimale avec Python 2.7.

Exemple d'utilisation :
```bash
python slicing.py cube.stl 0.08 0.2 0
```

GitHub
visiter   :
https://github.com/Nehri/slicing_algorithm Original
Visiter : https://github.com/Jacob-Cossette/Aptus_Slicing_Algo.git

---

Installation de l'environnement Conda
```

````
Pour installer l'environnement nécessaire à l'exécution du script, suivez les étapes ci-dessous :

1. **Installez Conda**
   Si vous n'avez pas encore installé Conda, vous pouvez télécharger et installer [Miniconda](https://docs.conda.io/en/latest/miniconda.html) ou [Anaconda](https://www.anaconda.com/products/distribution) selon votre préférence.

2. **Créez un nouvel environnement Conda**
   Ouvrez un terminal ou une invite de commande et créez un nouvel environnement Conda avec Python 2.7 :
   ```bash
   conda create -n slicer_env python=2.7
   ```

3. **Activez l'environnement**
   Une fois l'environnement créé, activez-le avec la commande suivante :
   ```bash
   conda activate slicer_env
   ```

4. **Exécutez le script**
   Une fois l'environnement configuré et les dépendances installées, vous pouvez exécuter le script Python :
   ```bash
   python slicing.py <nom_du_fichier_stl> <épaisseur_couche> <pourcentage_remplissage>
   ```
````

ANNEXE 2 : IMAGE DES FICHIERS STL ET DÉCOUPÉE.