Optimisation des supports pour la fabrication additive

Alexis Vallade

▶ To cite this version:

Alexis Vallade. Optimisation des supports pour la fabrication additive. [Rapport de recherche] CMAP Ecole Polytechnique; CNRS. 2021. hal-03464750

HAL Id: hal-03464750
https://hal.science/hal-03464750v1

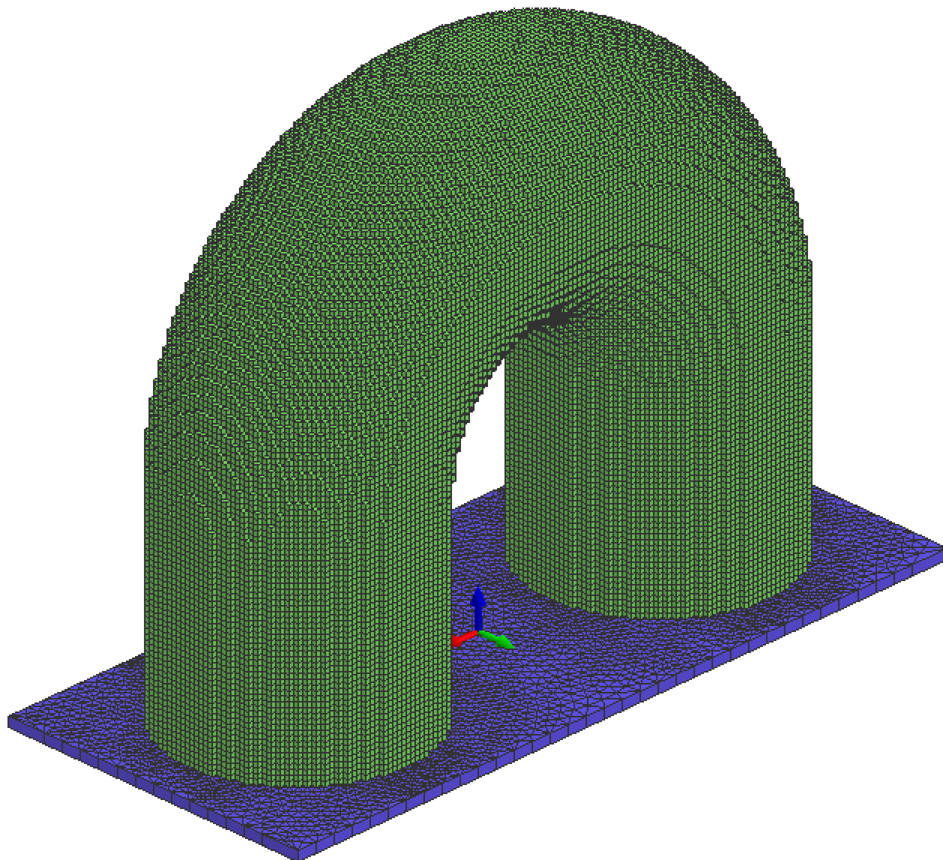
Submitted on 5 Jan 2022

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Optimisation des supports pour la fabrication additive
Projet SOFIA

Vallade Alexis



CMAP
École polytechnique
14 octobre 2021

Table des matières

I Support des parties en overhang **5**

I.1 Problématique 5

I.2 Détection des parties en overhang 6

I.3 Optimisation de supports soutenant les parties en overhang 6

I.4 Exemples 7

I.4.1 Table en T 7

I.4.2 Tube en U 11

II Détection des parties inaccessibles **15**

II.1 Problématique 15

II.2 Algorithme de détection 15

II.2.1 Description de l'algorithme 15

II.3 Optimisation du code 17

II.3.1 Méthode de ségrégation 17

II.4 Piste d'amélioration 21

II.4.1 Pondération des points d'observations 21

II.4.2 Parallélisation du code 22

II.5 Exemple 23

III Optimisation des supports à partir d'un calcul AM **25**

III.1 Problématique 25

III.2 Méthode proposée 25

III.2.1 Mise en place du calcul AM 25

III.2.2 Passage du calcul AM au calcul d'optimisation des supports 26

III.2.3 Optimisation des supports 27

III.3 Exemples 28

III.3.1 Toupie 28

III.3.2 Tube en U 29
III.4 Couplage avec la méthode géométrique de support des parties en overhang . . 31

A Scripts de détection des zones inaccessibles (getInaccessibleNodes.py) 38

B Scripts de détection des zones en overhang (getOverhang.py) 41

Introduction

Le projet ??SoFIA, « Solution pour la Fabrication Industrielle Additive métallique », est un programme de recherche appliquée à la fabrication additive métallique. Le programme a pour objectif de contribuer au développement de la fabrication additive métallique, en travaillant notamment sur la recherche de poudres, sur le développement d'équipements ou encore sur l'évolution des procédés. Ce rapport s'inscrit dans le cadre de ce projet, dans l'évolution des procédés, plus particulièrement sur l'optimisation des supports utilisés lors de la fabrication additive.

Le travail présenté dans ce rapport s'inscrit dans ce cadre. J'ai été recruté pour une durée de 3 ans au sein du CMAP afin de transférer les différents développements effectués au CMAP sur l'optimisation des supports dans les logiciels commerciaux d'ESI Group. L'optimisation de forme et de topologie est une technique permettant d'obtenir automatiquement la forme optimale d'une structure [1].[5]. C'est l'outil idéal pour optimiser les supports utilisés lors de la fabrication additive.

Dans ce rapport, trois pistes seront présentées. La première concerne l'optimisation de support pour les parties en overhang, la seconde traite de la détection des parties inaccessibles et enfin la dernière partie abordera l'optimisation de support en partant d'une simulation de fabrication additive.

Nous avons travaillé avec les outils ESI Group [12]. Les modèles et calculs ont été fait avec SYSTUS 2020 et TOPAZE 2020 [9] [17] . Le postraitement a été réalisé sur Visual-Viewer. Sauf mention contraire, les illustrations de ce rapport sont tirées de ces outils, ou bien des schémas réalisés par nos soins.

Remerciements

Je tiens, en premier lieu, à remercier Grégoire Allaire du CMAP de m'avoir encadré au sein du CMAP, à l'école Polytechnique, dans le cadre du projet SoFIA ainsi que Beniamin Bogosel pour m'avoir expliqué ses différents travaux.

Le travail présenté dans ce document a été réalisé avec les outils d'ESI Group. Je tiens à remercier Thomas Abballe pour son suivi hebdomadaire ainsi que Philippe Mourgue pour m'avoir aidé à plusieurs reprises sur des parties techniques.

Chapitre I

Support des parties en overhang

I.1 Problématique

La fabrication additive (Adictive Manufacturing) permet la fabrication de structures complexes ne pouvant pas être fabriquées autrement [4]. Cependant, elle a aussi des contraintes de fabrication, comme souligné dans la littérature [3, 7, 8, 10, 11, 13, 14, 15, 16, 18, 19, 20, 22, 23, 24]. La contrainte de surplomb (overhang) est l'une de ces contraintes. Les composants avec de petits angles par rapport à l'horizontal ou des éléments de suspension peuvent se déformer, s'affaisser lorsqu'ils sont fabriqués à l'aide de faisceaux laser ou d'électrons (comme on peut le voir sur la figure I.1). Pour garantir une bonne impression des pièces il est nécessaire de supporter ces zones quit sont dites "en overhang".

Une méthode, proposée dans [2], permet de s'assurer que ces zones soient supportées. Cette partie traite du portage des développements sur ce sujet dans l'environnement d'ESI Group.

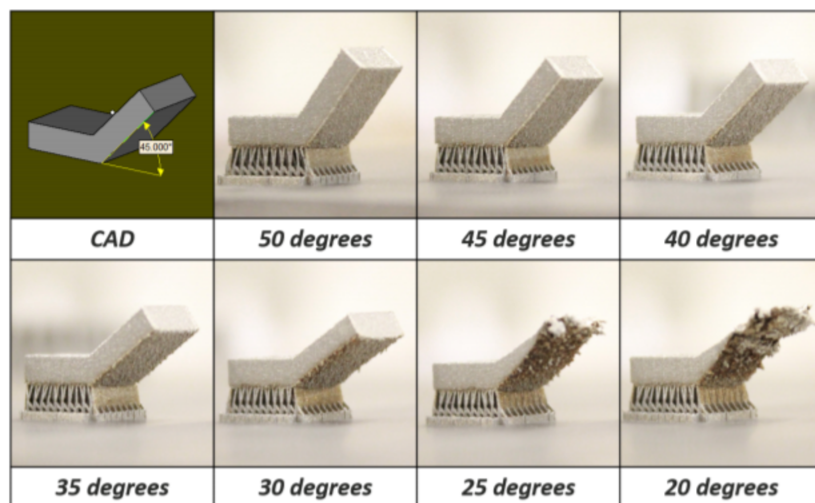


Figure I.1 – Impression d'une pièce avec différents angles d'inclinaison (https://www.additivemanufacturing.media/blog/post/7-helpful-numbers-quantify-design-rules-for-am)

I.2 Détection des parties en overhang

Un script en python a été réalisé dans le but de détecter les zones en overhang (voir annexe B. L'utilisateur indique sur quel maillage il veut réaliser l'opération ainsi que l'angle par rapport à l'horizontal à partir duquel on considère qu'une surface est en overhang.

La méthode consiste à comparer l'angle fourni par l'utilisateur et l'angle formé entre la normale de chaque élément surfacique et l'horizontale. Si l'angle calculé est inférieur à l'angle fourni par l'utilisateur, l'élément surfacique est en overhang.

Sur la figure I.2 on peut observer un résultat obtenu. Cet exemple est constitué d'un tube en U et d'une baseplate. La baseplate correspond à une plaque métalique sur laquelle on construit la structure, celle ci est horizontale. Les zones en overhang sont indiquées en vert.

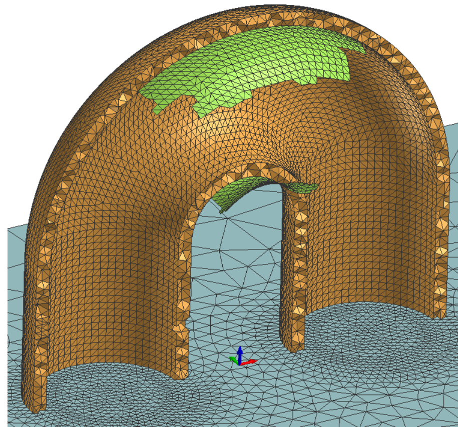


FIGURE I.2 – Zone en overhang sur un tube en U

I.3 Optimisation de supports soutenant les parties en overhang

Afin de forcer la présence de supports sur les surfaces en overhang, et ainsi éviter les problèmes de fabrication, une méthode de pseudo gravité est proposée ici suivant [3].

Une fois les surfaces en overhang détectées (schéma I.3), des forces verticales y sont appliquées (schéma I.4). On résout ensuite un problème d'optimisation. On peut au choix, soit minimiser le volume avec une compliance à ne pas dépasser, ou minimiser la compliance avec un volume à ne pas dépasser. On obtient alors un support couvrant la totalité de la surface en overhang (schéma I.5).

La compliance, qui représente le produit de la force et du déplacement sur un domaine, est calculée uniquement sur le support. Ce calcul est détaillé dans la partie 3.1 de l'article [2].

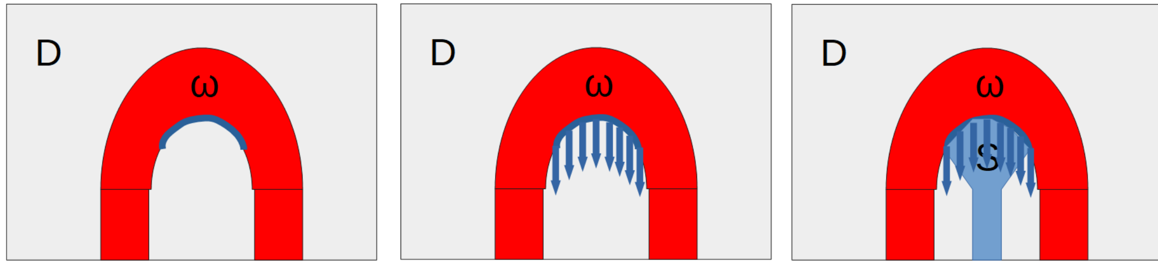


Figure I.3 – Détection des zones en overhang

Figure I.4 – Forces verticales imposées sur les surfaces en overhang

Figure I.5 – Support minimisant la compliance

I.4 Exemples

I.4.1 Table en T

La baseplate est composée de 5523 noeuds et 24792 éléments tetrahèdres.

La table possède au total 3947 nœuds et 16181 éléments tetrahèdres.

Le design space a 13317 nœuds et 66746 éléments tetrahèdres.

Le maillage de ces différentes parties est visible sur la figure I.6.

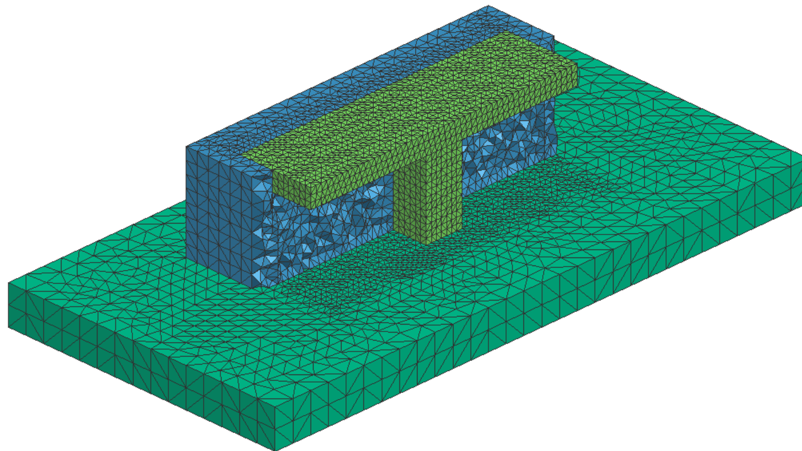


Figure I.6 – Maillage initial de la table pour le calcul d'optimisation

Les zones en overhang, détectées avec un angle de moins de 45° par rapport à l'horizontal, sont visibles sur la figure I.7.

Sur les surfaces en overhang, une force nodale est appliquée sur chaque nœud, et a pour valeur 1000 N. Ces forces sont visibles sur la figure I.8.

Minimisation de la compliance avec une fraction volumique de 20%

Dans un premier temps, le calcul d'optimisation est réalisé avec la minimisation de la compliance des supports en objectif et avec pour contrainte une fraction volumique de 20%. La

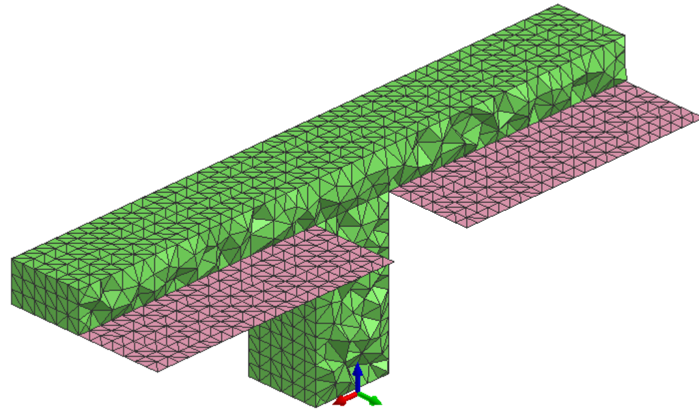


FIGURE I.7 – Surface en overhang pour la table

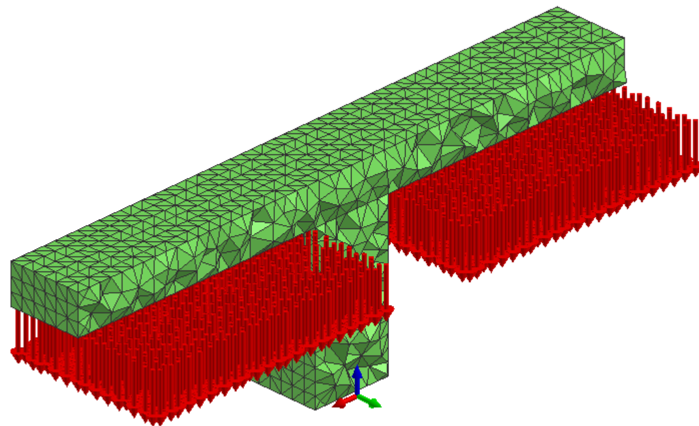


FIGURE I.8 – Forces verticales appliquées sur les surfaces en overhang pour la table

fraction volumique est définie comme le rapport entre le volume du support et le volume initial du design space.

Les supports obtenus sont visibles sur la figure I.9, l'évolution de la compliance en fonction du nombre d'itération est présentée sur la figure I.10 et celle du volume sur la figure I.11.

Minimisation de la compliance avec une fraction volumique de 5%

Le calcul d'optimisation est réalisé ici avec la minimisation de la compliance des supports en objectif avec pour contrainte une fraction volumique de 5%.

Les supports obtenus sont visibles sur la figure I.12, l'évolution de la compliance en fonction du nombre d'itération est présentée sur la figure I.13 et celle du volume sur la figure I.14.

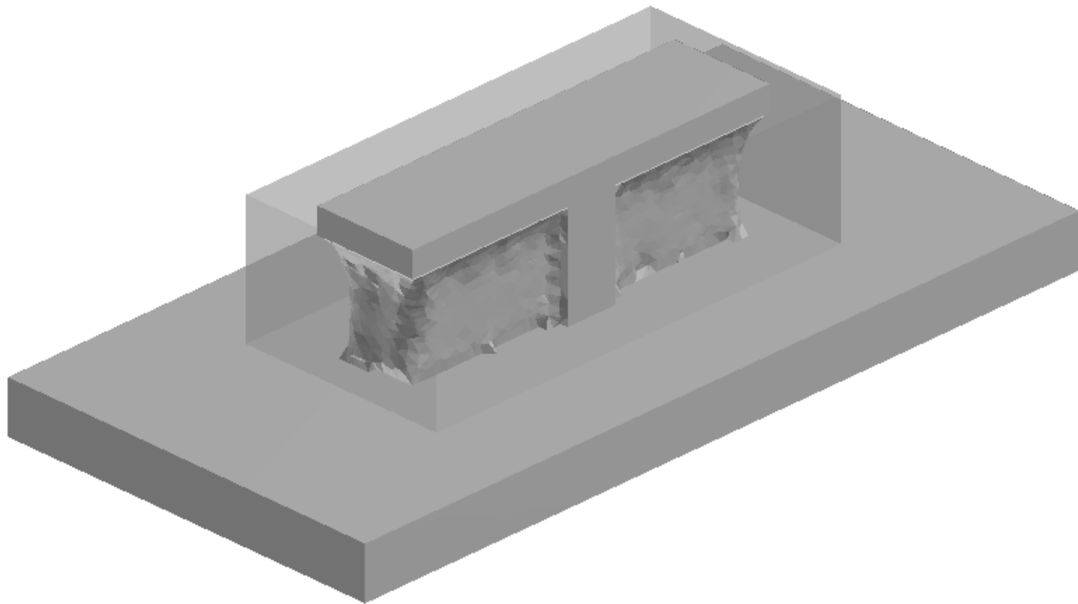


Figure I.9 – Supports obtenus pour la table, avec une fraction volumique cible de 20%

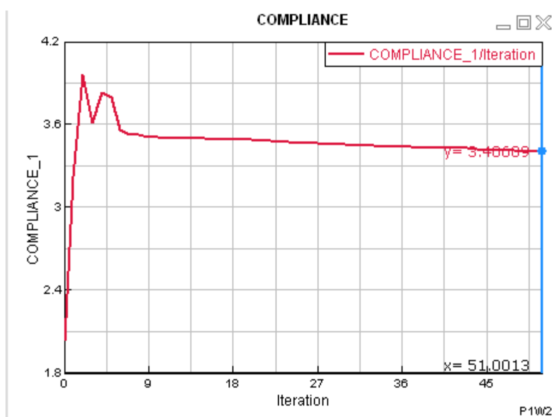


Figure I.10 – Évolution de la compliance en fonction du nombre d'itération

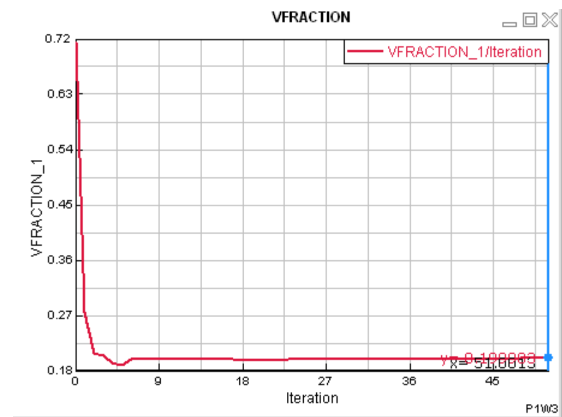


Figure I.11 – Évolution du volume des supports en fonction du nombre d'itération

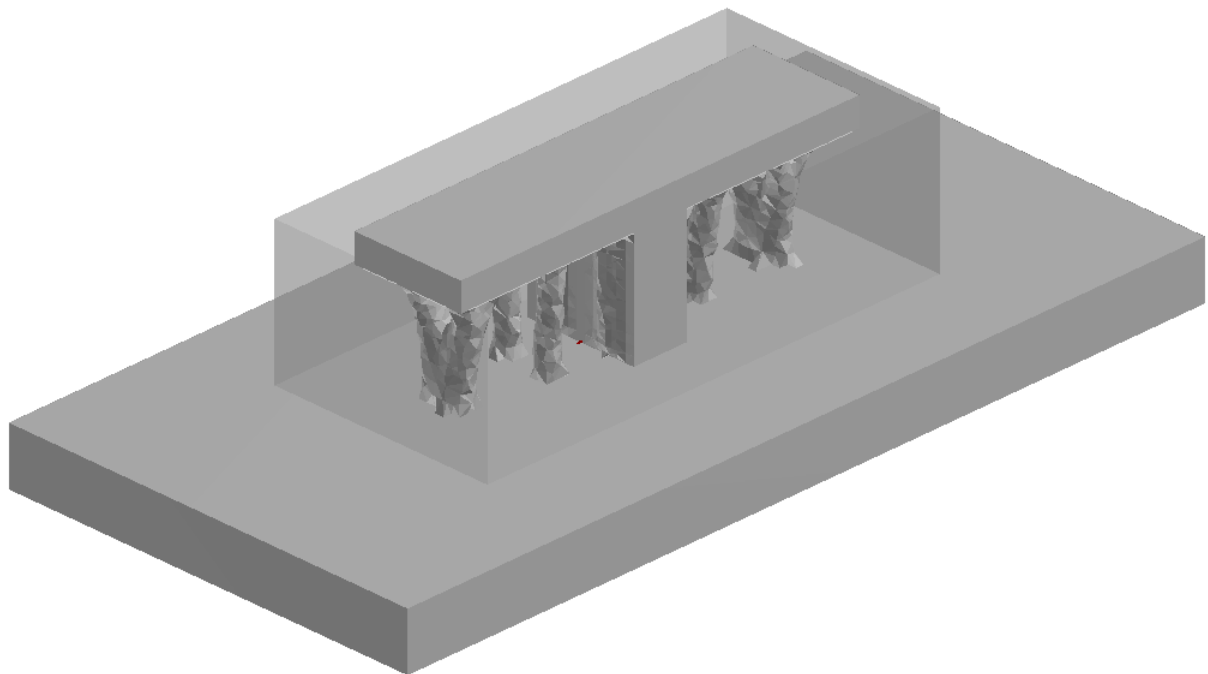


Figure I.12 – Supports obtenus pour la table, avec une fraction volumique cible de 5%

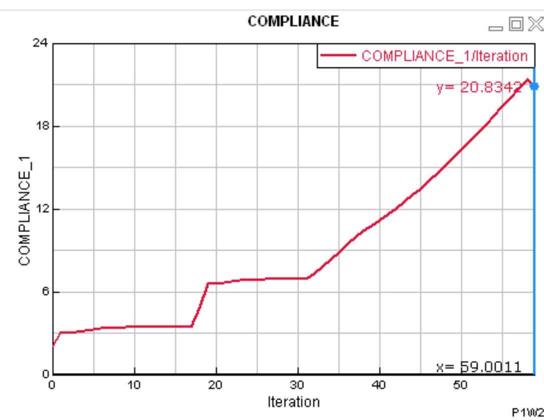


Figure I.13 – Évolution de la compliance en fonction du nombre d'itération

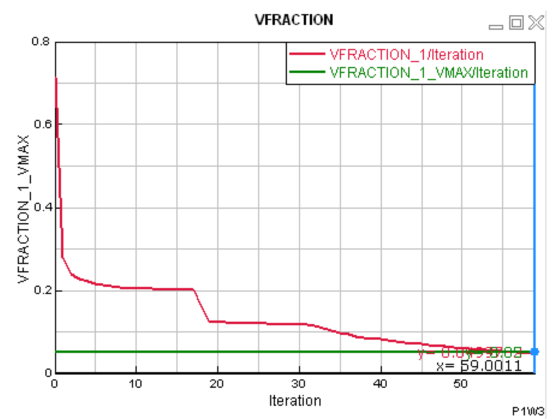


Figure I.14 – Évolution du volume des supports en fonction du nombre d'itération

I.4.2 Tube en U

La baseplate est composée de 8814 noeuds et 7631 éléments héxahèdriques.

Le tube possède au total 152545 nœuds et 112615 éléments héxahèdriques.

Le design space à l'intérieur du tube possède 178588 nœuds et 931680 éléments tetrahèdriques, celui de l'extérieur du tube a 178533 nœuds et 890003 éléments tetrahèdres.

Le maillage de ces différentes parties est visible sur la figure I.15.

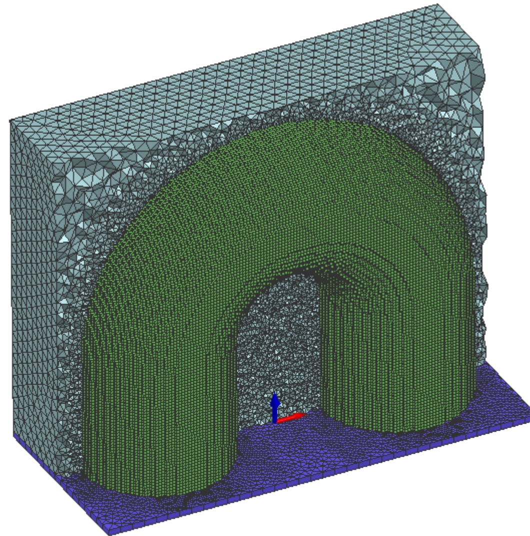


FIGURE I.15 – Maillage initial du tube pour le calcul d'optimisation

Les zones en overhang, détectées avec un angle de moins de 45° par rapport à l'horizontal, sont visibles sur la figure I.16.

Sur les surfaces en overhang, une force nodale est appliquée sur chaque nœud, et a pour valeur 1000 N. Ces forces sont visibles sur la figure I.17.

Support obtenu avec un objectif de minimisation du volume et une compliance max de 10 MPa.m

Dans un premier temps, le calcul d'optimisation est réalisé avec la minimisation du volume des supports en objectif avec pour contrainte une compliance maximum dans l'ensemble support + pièce de 10 MPA.m.

Les supports obtenus sont visibles sur la figure I.18, l'évolution de la compliance en fonction du nombre d'itération est présentée sur la figure III.21 et celle du volume sur la figure I.20.

Support obtenu avec un objectif de minimisation du volume et une compliance max de 20 MPa.m

Dans un second temps, le calcul d'optimisation est réalisé avec la minimisation du volume des supports en objectif avec pour contrainte une compliance maximum dans l'ensemble sup-

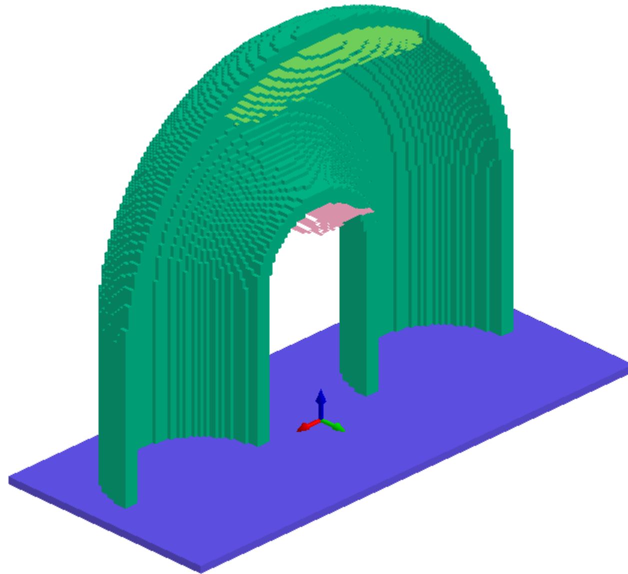


FIGURE I.16 – Surface en overhang pour le tube en U

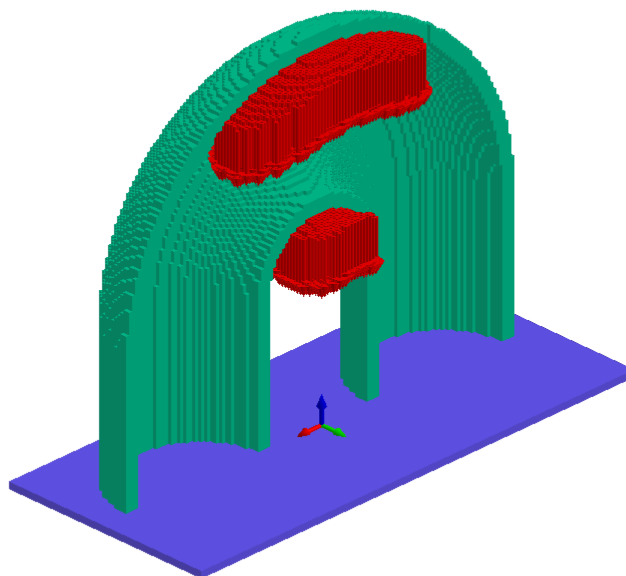
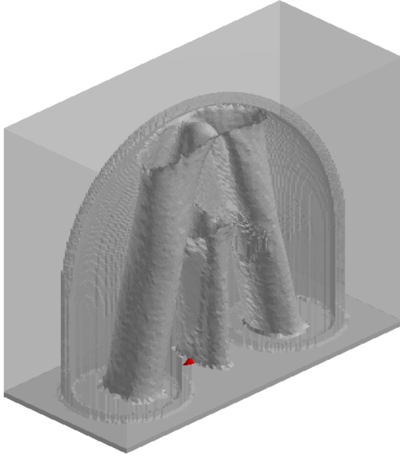


FIGURE I.17 – Forces verticales appliquées sur les surfaces en overhang pour le tube en U

port + pièce de 20 MPA.m.

Les supports obtenus sont visibles sur la figure I.21, l'évolution de la compliance en fonction du nombre d'itération est présentée sur la figure I.22 et celle du volume sur la figure I.23.



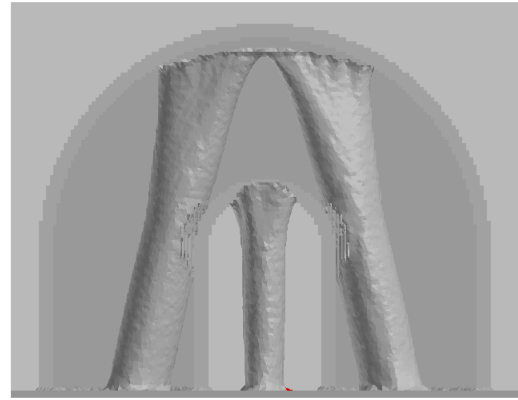


FIGURE I.18 – Supports obtenus pour le tube en U, avec une compliance cible de 10 Mpa.m

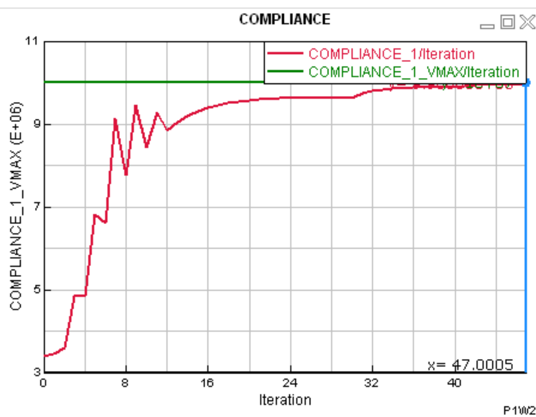


FIGURE I.19 – Évolution de la compliance en fonction du nombre d'itération

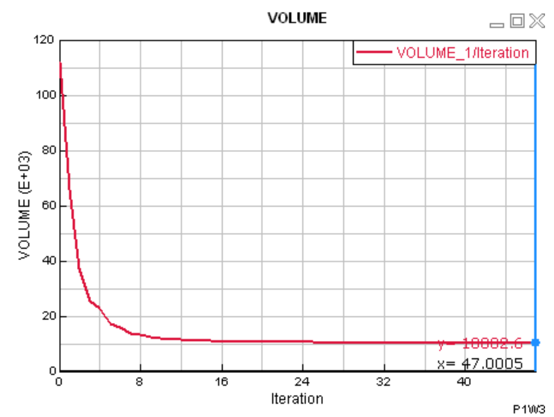
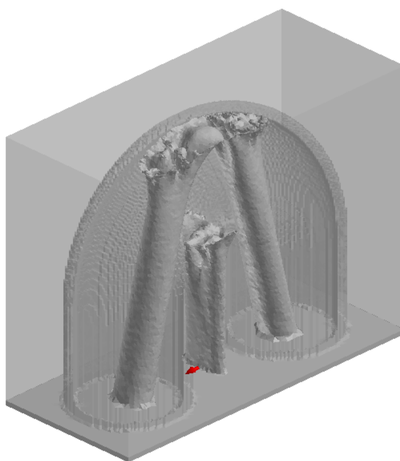


FIGURE I.20 – Évolution du volume des supports en fonction du nombre d'itération



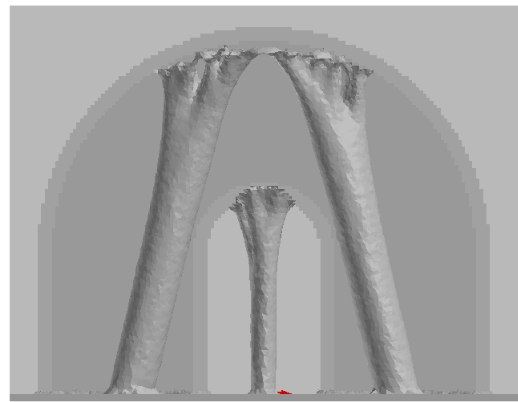


FIGURE I.21 – Supports obtenus pour le tube en U, avec une compliance cible de 20 Mpa.m

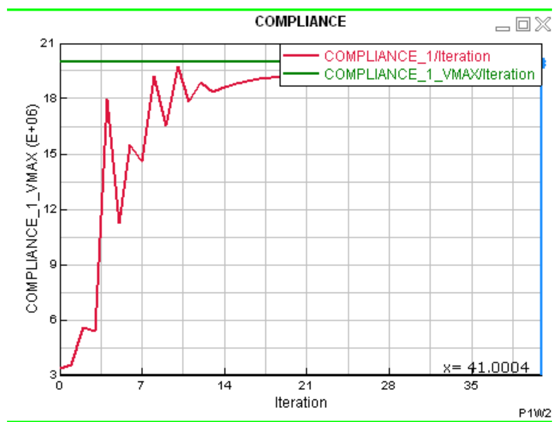


Figure I.22 – Évolution de la compliance en fonction du nombre d'itération

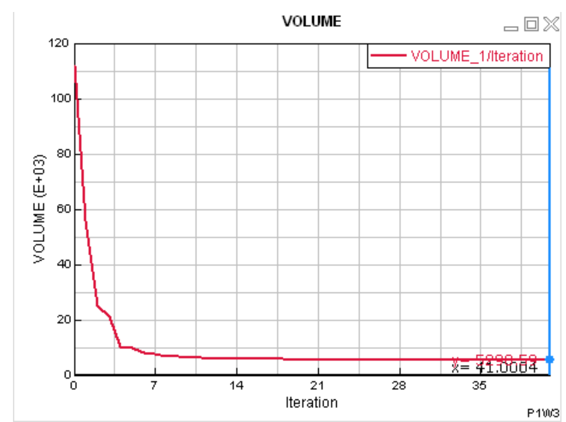


Figure I.23 – Évolution du volume des supports en fonction du nombre d'itération

Chapitre II

Détection des parties inaccessibles

II.1 Problématique

A la fin du processus d'impression 3D, les supports doivent être retirés. La surface de contact entre le support et la pièce doit être accessible depuis l'extérieur de la pièce afin de retirer au mieux celui-ci. La connaissance des régions de la pièce qui sont inaccessibles depuis l'extérieur est donc nécessaire dans le but d'éviter que le support y soit en contact. Cette partie traite du portage des développements de la méthode décrite dans [3], [2] sur ce sujet dans l'environnement logiciel d'ESI Group. Des résultats plus récents [6] ont montré qu'il était possible d'optimiser simultanément une pièce et son support en évitant que le support colle à une partie inaccessible de la pièce, ce chapitre ne traitera pas de ce développement.

II.2 Algorithme de détection

II.2.1 Description de l'algorithme

La méthode de détection des zones accessibles utilisée ici reprend l'idée du ray tracing. Un point du maillage est considéré accessible lorsque il existe un point d'observation en dehors de la pièce tel que le segment formé par ce point d'observation et le point à tester ne coupe aucun élément surfacique de la pièce.

Sur la figure II.1 on peut voir l'exemple d'une pièce Ω pour laquelle on veut vérifier que le point noté PT est accessible. Pour cela on dispose de 5 points d'observations, notés PO1 à PO5. Les 5 segments formés par les points d'observations et le point à tester sont tracés. On peut remarquer qu'il existe trois segments qui coupent au moins un élément surfacique (ceux des points PO3, PO4 et PO5). Les deux segments restants ne coupent pas d'élément surfacique, le point à tester est donc considéré comme accessible (via les points PO1 et PO2).

La figure II.2 reprend le même principe, mais avec un point à tester placé différemment. On observe que les 5 segments coupent au moins un élément surfacique de Ω, le point est donc considéré comme étant inaccessible depuis les points d'observations.

Il est important de noter ici que le placement des points d'observations est très important : il faut en placer de façon régulière autour de la pièce, ainsi qu'aux différentes entrées de la pièce (comme le point PO3 dans le cas de la pièce Ω).

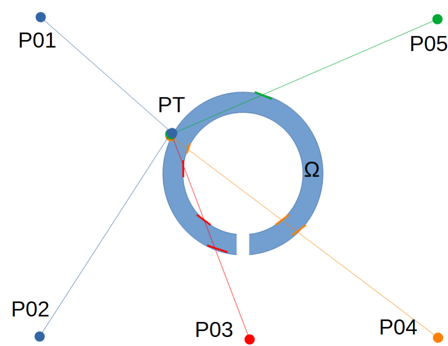


Figure II.1 – Point accessible

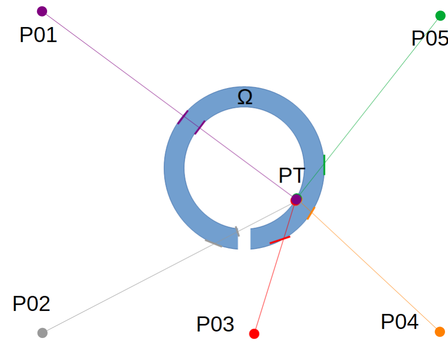


Figure II.2 – Point inaccessible

L'algorithme 1 présente la méthode développée. Afin de tester si un segment coupe un élément surfacique (un triangle), l'algorithme de Möller-Trumblore (voir [21]) est utilisé. Cet algorithme est utilisé dans des méthodes de ray tracing et il permet de déterminer efficacement si un segment coupe un triangle. Un des avantages de cet algorithme est qu'il ne nécessite pas de calculer le plan contenant le triangle, ce qui limite le nombre d'opération nécessaire à la détection.

Dans le cas où le maillage surfacique ne serait pas triangulaire, une étape sera nécessaire pour transformer le maillage en maillage triangulaire.

Algorithm 1: Détection des zones inaccessibles

```
input  : Liste de point à tester
         Liste de point d'observation
         Liste des éléments surfaciques
output: Liste des points accessibles
         Liste des points inaccessibles
for Chaque point de la surface (p1) do
    for Chaque point d'observation (p2) do
        estAccessible = Vrai;
        for Chaque triangle de la surface do
            if La droite formée par p1 et p2 coupe le triangle then
                estAccessible = Faux;
                On sort de la boucle sur les triangles;
            end
        end
        if estAccessible = Vrai then
            p1 est ajouté à la liste des éléments accessibles;
        end
    end
    if estAccessible = Faux then
        p1 est ajouté à la liste des éléments inaccessibles;
    end
end
```

II.3 Optimisation du code

Le calcul de l'intersection d'un triangle avec une droite demande environ une soixantaine d'opérations mathématiquse. Cette fonction, dans le pire des cas, peut être réalisée $n_{noeud\ a\ tester} * n_{noeud\ d'observation} * n_{triangle}$ fois. L'algorithme 1 se révéle alors bien trop lent. Nous proposons ici différentes méthodes afin de limiter le nombre d'opérations et ainsi réduire le temps de calcul.

II.3.1 Méthode de ségrégation

Une première approche dans l'amélioration de l'algorithme consiste à réduire le nombre de calculs d'intersections entre un segment et un triangle qui est nécessaire. Pour cela, deux approches de ségrégation sont proposés. Les méthodes de ségrégations proposées içi sont purement géométriques, elle permettent de savoir si le calcul de l'intersection est utile en réalisant une opération moins couteuse en terme de temps de calcul.

Ségrégation par une boîte

La première ségrégation proposée ici est une ségrégation par boite. Cette méthode consiste à vérifier si le triangle est inclus dans la boîte formée par le point à tester et le point d'obser-

vation, comme on peut le voir sur le schéma II.3.

Dans un premier temps, on stocke dans un vecteur les coordonnées minimale et maximale de chaque triangle, ce calcul est rapide et n'a besoin d'être réalisé qu'une fois.

Dans un second temps, avant le calcul d'intersection entre le segment et la droite, on vérifie que le triangle est bien inclus dans la box. Cette étape consiste simplement à comparer les coordonnées du triangle avec celle de la box formée par les deux points, elle ne nécessite donc que très peu de temps de calcul.

Si le triangle est dans la box alors on teste l'intersection, si il n'est pas dans la box on ne le teste pas. L'algorithme 2 décrit la totalité du processus.

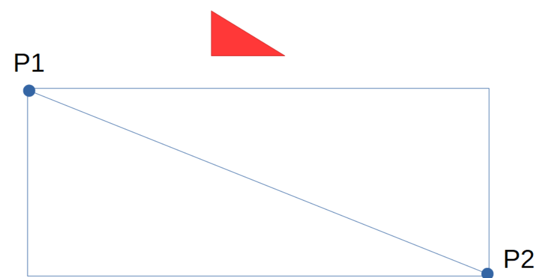


FIGURE II.3 – Ségrégation par une boite

Algorithm 2: Détection des zones inaccessibles

```
input  : Liste de point à tester
         Liste de point d'observation
         Liste des éléments surfaciques
output: Liste des points accessibles
         Liste des points inaccessibles
initialization;
for Chaque triangle de la surface do
  | Calcul des coordonnées min et max du triangle
end
for Chaque point de la surface (p1) do
  for Chaque point d'observation (p2) do
    estAccessible = Vrai;
    for Chaque triangle de la surface do
      if Le triangle n'est dans la boite formée par p1 et p2 then
        | On passe au triangle suivant;
      end
      if La droite formée par p1 et p2 coupe le triangle then
        estAccessible = Faux;
        On sort de la boucle sur les triangles;
      end
    end
    if estAccessible = Vrai then
      | p1 est ajouté à la liste des éléments accessibles;
    end
  end
  if estAccessible = Faux then
    | p1 est ajouté à la liste des éléments inaccessibles;
  end
end
```

En appliquant cette méthode sur l'exemple du tube en U, le nombre de calcul d'intersection entre un segment et le triangle est divisé par 14 environ, le temps de calcul lui est divisé par 5.

Ségrégation par une sphère

La seconde ségrégation proposée ici est une ségrégation par sphère. Cette méthode consiste à vérifier si le segment formée par le point d'observation et le point à tester coupe la sphère circonscrite au triangle, comme on peut le voir sur le schéma II.4.

Dans un premier temps, on stocke dans un vecteur le centre et le rayon de la sphère circonscrite au triangle. Ce calcul est rapide et n'a besoin d'être réalisé qu'une fois.

Dans un second temps, avant le calcul d'intersection entre le segment et le triangle, on vérifie que le segment coupe bien la sphère. Cette étape consiste simplement à calculer la distance entre le centre de la sphère et la droite, puis de la comparer au rayon de la sphère.

Cette opération, une fois optimisée, demande entre 7 et 20 opérations mathématiques, soit dans le pire des cas trois fois moins que le calcul d'intersection entre le triangle et le segment.

Si le segment coupe la sphere alors on teste l'intersection, si il ne coupe pas dans la sphère on ne le teste pas. L'algorithme 3 décrit la totalité du processus.

Cette ségrégation est bien plus coûteuse en terme de calcul que celle avec la box, mais elle retire un nombre de calcul d'intersection bien plus important.

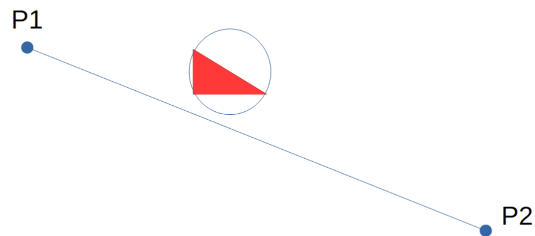


FIGURE II.4 – Ségrégation par une sphère

Algorithm 3: Détection des zones inaccessibles

```
input  : Liste de point à tester
         Liste de point d'observation
         Liste des éléments surfaciques
output: Liste des points accessibles
         Liste des points inaccessibles
initialization;
for Chaque triangle de la surface do
    Calcul de la sphère circonscrite au triangle
end
for Chaque point de la surface (p1) do
    for Chaque point d'observation (p2) do
        estAccessible = Vrai;
        for Chaque triangle de la surface do
            if La droite formée par p1 et p2 ne coupe pas la sphère circonscrite au triangle
              then
                On passe au triangle suivant;
            end
            if La droite formée par p1 et p2 coupe le triangle then
                estAccessible = Faux;
                On sort de la boucle sur les triangles;
            end
        end
        if estAccessible = Vrai then
            p1 est ajouté à la liste des éléments accessibles;
        end
    end
    if estAccessible = Faux then
        p1 est ajouté à la liste des éléments inaccessibles;
    end
end
```

En appliquant cette méthode sur l'exemple du tube en U, le nombre de calcul d'intersection entre un segment et le triangle est divisé par 750 environ, le temps de calcul lui est divisé par 2. Le calcul de distance entre une sphère et une droite demande un peu plus d'opération que la méthode de la box, le gain est ici moins important.

En combinant les deux méthodes, en commençant par la méthode de boite puis celle de la sphère, on réduit le temps de calcul final par 10. Ce script est visible dans l'annexe A.

II.4 Piste d'amélioration

II.4.1 Pondération des points d'observations

Les points d'observations ne sont pas forcement placés de façon optimale, afin d'améliorer l'ordre des points d'observations un système de pondération peut être mis en place. Au début

de l'algorithme tous les points ont un poids de 0. Lorsqu'un point de la surface est accessible par un point d'observation, ce point d'observation gagne une pondération de 1. Les points d'observations sont ensuite triés par poids (le tri peut se faire plus ou moins régulièrement en fonction du temps que prends cette étape).

Ainsi, les points permettant la validation de plusieurs nœuds sont mis en avant et traités en priorité, augmentant ainsi la probabilité de trouver rapidement un point d'observation qui correspond au point de la surface à tester.

L'algorithme 4 détaille cette procédure.

Algorithm 4: Détection des zones inaccessibles

```
input  : Liste de point à tester
         Liste de point d'observation
         Liste des éléments surfaciques
output: Liste des points accessibles
         Liste des points inaccessibles
initialization;
for Chaque point d'observation po do
    poids(po) = 0
end
for Chaque point de la surface (p1) do
    for Chaque point d'observation (p2) do
        estAccessible = Vrai;
        for Chaque triangle de la surface do
            if La droite formée par p1 et p2 coupe le triangle then
                estAccessible = Faux;
                On sort de la boucle sur les triangles;
            end
        end
        if estAccessible = Vrai then
            p1 est ajouté à la liste des éléments accessibles;
            poids(p2)+=1
        end
    end
    if estAccessible = Faux then
        p1 est ajouté à la liste des éléments inaccessibles;
    end
    triPointObservation()
end
```

II.4.2 Parallélisation du code

Afin d'améliorer le temps de calcul, certaines étapes peuvent être réalisées en parallèle. Plusieurs approches peuvent être réalisées.

Il serait par exemple judicieux de distribuer les points d'observations sur différents processeurs. Chaque processeur aurait alors à réaliser le calcul pour un couple point de la surface

/ point d'observation et il devra effectuer le calcul pour chacun des triangles. Lorsqu'un processeur trouve une triangle qui coupe la droite formée par le point de la surface et son point d'observation, il passe au point d'observation suivant, indépendamment des autres processeurs. Si le point d'observation associé à un processeur est considéré comme accessible, alors le calcul de chacun des processeurs doit s'arrêter, le point de la surface est considéré comme accessible et on passe au point de la surface suivant.

Une autre approche, similaire à la précédente, serait d'associer un nœud de la surface à un processeur, lorsque le processeur a fini son calcul, il passe au point suivant, indépendamment du calcul des autres processeurs.

II.5 Exemple

Afin d'illustrer l'algorithme, l'exemple d'un tube en U, visible sur la figure II.5, a été utilisé.

Le tube contient 14596 noeuds et 29192 triangles sur sa surface. Nous avons placé 25 points d'observations autour de la pièce ainsi qu'a proximité des points d'entrées du tube.

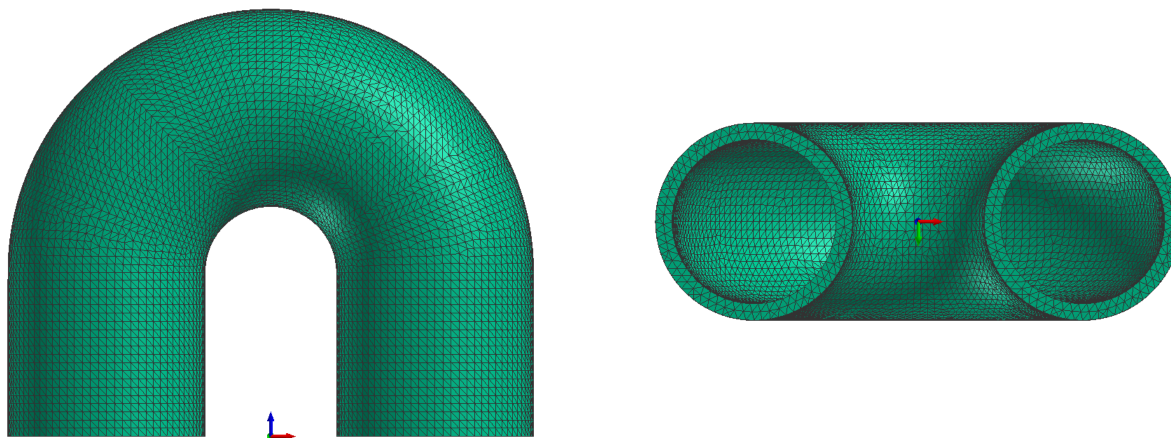


Figure II.5 – Exemple : cas d'un tube en U

La détection, sans aucune optimisation du code ni de la procédure, prend 7050 secondes pour être effectuée en totalité. Ce calcul demande d'effectuer environ 1.8 Milliards de fois l'opération de détection d'intersection entre un segment et un triangle.

Sur les 14596 nœuds de la surface, 825 sont détectés comme étant inaccessibles depuis l'extérieur. On peut observer ces nœuds sur la figure II.6.

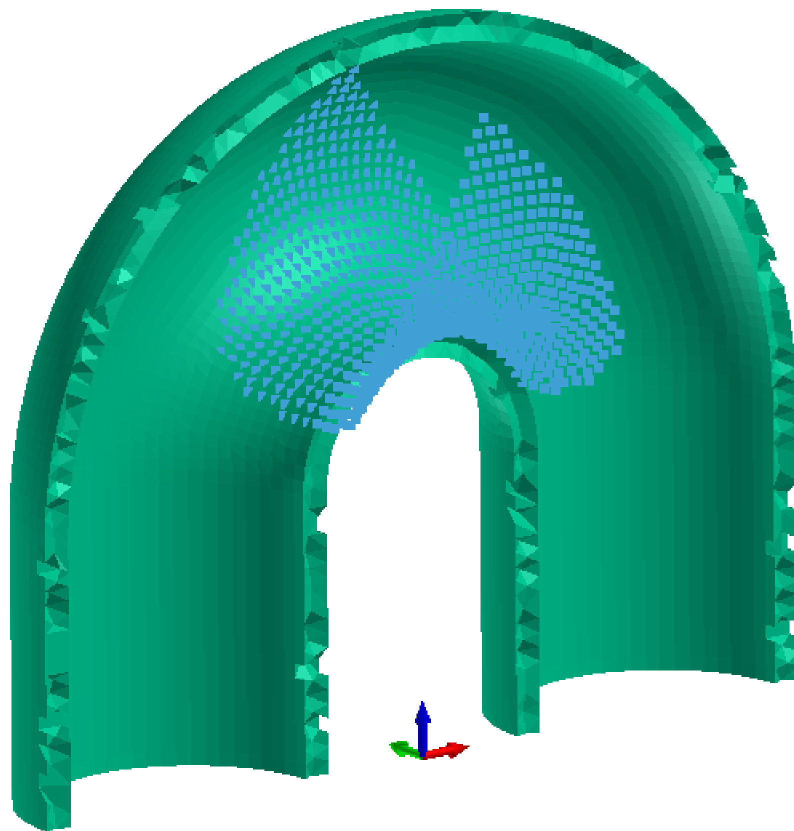


Figure II.6 – Nœuds du tube en U inaccessible depuis l'extérieur

Chapitre III

Optimisation des supports à partir d'un calcul AM

III.1 Problématique

Lors de la fabrication de pièces métalliques par fabrication additive, des déformations thérmomécaniques peuvent apparaitre lors du processus de fabrication. La simulation thérmo-mécanique du processus complet de fabrication additive, en prenant en compte la fabrication couche par couche est très consommatrice en temps de calcul.

Afin de réduire le temps de calcul lors du processus d'optimisation, il est nécessaire de trouver un modèle simplifié permettant d'avoir une réduction importante des temps de calcul.

III.2 Méthode proposée

Une méthode, proche de celle décrite dans la partie 3.2 de l'article [2], est utilisée ici dans l'environnement logiciel d'ESI Group. Celle-ci consiste à trouver un modèle statique équivalent au problème dépendant du temps. Pour cela, on réalise dans un premier temps un calcul précis, couche par couche, en prenant en compte les effets thérmo-mécaniques, puis on en déduit un chargement mécanique équivalent, ne dépendant cette fois ci plus du temps. Cette méthode est décrite dans cette section.

III.2.1 Mise en place du calcul AM

La première partie de la méthode consiste à réaliser une simulation numérique complète du processus de fabrication additive de la pièce. Ce calcul n'est réalisé qu'une fois, le temps de calcul peut être important ici.

Cette simulation est réalisée à l'aide d'ESI-AM, l'outil de fabrication additive d'ESI Group. Lors de cette simulation, on suppose que des supports sont présents dans une certaines zones, ces zones sont visibles en gris sur la figure III.1. Afin de représenter la présence de supports

sur ces zones, des ressorts sont accrochés sur chaque nœud de ces zones, ils permettent de prendre en compte une raideur du à la présence de support.

Les résultats obtenus lors de cette simulation servent de points de départ pour la méthode proposée ici. Une deuxième simulation numérique, cette fois-ci sans la présence des ressorts est aussi réalisée.

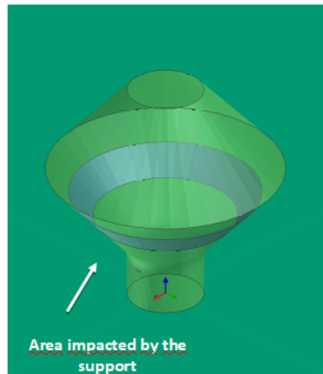


FIGURE III.1 – Définition de la zone de support

III.2.2 Passage du calcul AM au calcul d'optimisation des supports

Récupération de la déformation cible

Cette étape consiste à récupérer les déplacements obtenus lors du calcul AM avec la présence de ressorts sur les surfaces où sont supposés se placer les supports. Les supports représente l'effet mécanique dû à la présence de support dans cette zone. Ces déplacements nodaux sont stockés dans un fichier et serviront de déplacement cible lors du calcul d'optimisation. On notera ces déplacements U_g, cette opération est visible sur le schéma III.2.

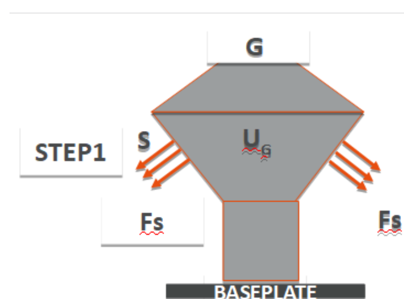


FIGURE III.2 – Récupération de la déformation cible

Récupération des forces internes

La seconde étape a pour but de déterminer les forces internes correspondant au chargement thermique. Pour cela, on charge les résultats AM obtenus sans la présence de ressorts, puis on stocke les déplacements obtenus. Ensuite, on effectue un calcul mécanique en imposant ces

déplacements sur chaque nœuds et on récupère les forces de réaction de chaque nœuds. On a ainsi transformé un calcul thérmomécanique en un modèle de mécanique statique. Ces forces sont stockées dans un fichier et sont notés IF_s. Cette étape est schématisée sur la figure III.3.

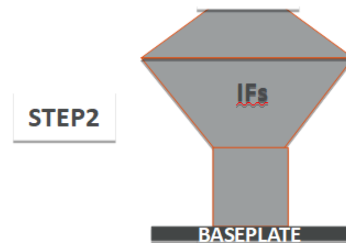


Figure III.3 – Récupération des forces internes

Exportation du maillage pour l'optimisation

La dernière étape consiste à mettre en place les forces internes IF_s sur les nœuds de la pièce, on retire ensuite tous les éléments du maillage autre que la pièce. On obtient alors le fichier visible sur la figure III.4. Le calcul d'optimisation étant un calcul mécanique linéaire, il est nécessaire à cette étape de vérifier que les déplacements obtenus correspondent bien à ceux observés lors du calcul AM (calcul thermique non linéaire). Si ceux-ci correspondent, la procédure est validée.

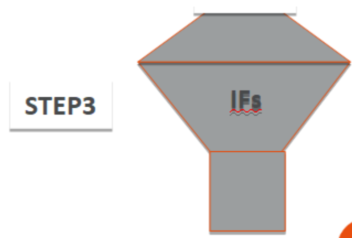


Figure III.4 – Export issue du calcul AM

III.2.3 Optimisation des supports

Lors de l'étape d'optimisation des supports, on cherche les supports ayant le volume le plus faible possible, permettant d'avoir une déformation de la pièce similaire à celle du calcul AM avec les ressorts.

L'état 1, représenté sur la figure III.5, est l'état de déformation que l'on souhaite obtenir. Au début du calcul d'optimisation, la présence de matière dans le design space réduit fortement la déformation de la pièce, comme on peut le voir sur la figure III.6, cet état intermédiaire est noté état 2. A la fin du processus d'optimisation, la déformation de la pièce, visible sur la figure III.7, est similaire à celle de l'état 1. Le support obtenu est alors celui recherché.

Dans le but de réaliser ce calcul, on doit importer le maillage issue du calcul AM sur un maillage conçu pour l'opération d'optimisation de support. Le maillage d'optimisation possède une baseplate différente de celle utilisée pour le calcul AM (les conditions aux limites

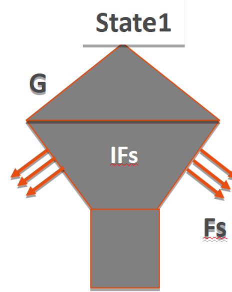


Figure III.5 – Déformation de la pièce souhaité

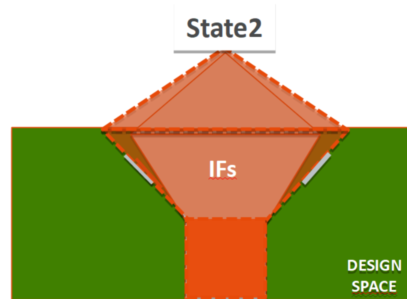


Figure III.6 – Déformation initiale de la pièce

étant imposées différemment) et celle-ci contient en plus le maillage du design space (qui n'existe pas sur le maillage AM). Le travail consiste à importer le maillage de la pièce issue du calcul AM, avec les forces internes, dans le maillage utile au calcul d'optimisation, contenant la nouvelle baseplate ainsi que le design space.

Le calcul AM ayant été réalisé avec des ressorts présents uniquement sur certaines surfaces (notées S), il est nécessaire que les supports obtenus lors du calcul d'optimisation soient collés à la pièce uniquement sur ces surfaces. Pour cela, tous les nœuds de surface de la pièce n'appartenant pas à S sont dédoublés. Ainsi, le support ne gagne rien mécaniquement en se collant à ces nœuds. Il est aussi nécessaire de ne pas dédoubler les nœuds de la pièce qui sont en contact avec la baseplate ainsi que ceux en commun entre la baseplate et la zone d'optimisation.

III.3 Exemples

III.3.1 Toupie

Le premier exemple présenté ici traite du cas d'une pièce en forme de toupie. Le maillage de la toupie est visible sur la figure III.8, la toupie est visible en rose et le design space en gris. La toupie est composée de 33300 hexaèdres et le design space de 71178 hexaèdres.

Les supports issues du calcul d'optimisation sont visibles sur la figure III.10.

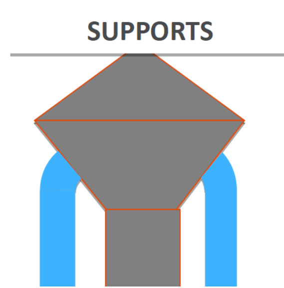


FIGURE III.7 – Déformation de la pièce avec les supports optimisés

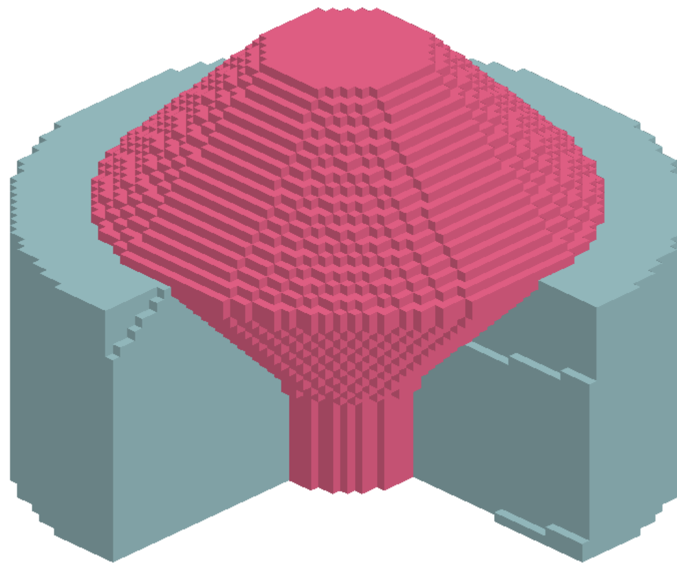


FIGURE III.8 – Maillage de la toupie utilisée pour le calcul d'optimisation en partant d'un calcul AM

III.3.2 Tube en U

Calcul AM

Le maillage utilisé pour le calcul AM, visible sur la figure III.11, est composé de deux parties, la pièce (le tube) et la baseplate.

Le tube est décomposé en 104 couches (représentant les couches lors de l'impression 3D), il possède au total 152545 nœuds et 112615 éléments héxahèdriques.

La baseplate possède 16436 noeuds et 11871 hexaèdres. Afin de limiter le nombre de nœud et d'élément sur la baseplate celle ci possède un maillage non conforme.

Sur le maillage du tube, des ressorts sont appliqués sur les surfaces en overhang ,représentées en rouge sur la figure III.12, ils permettent de représenter la présence de supports dans ces zones.

La baseplate ainsi que le tube ont pour propriétés mécaniques un module Young de 114 GPA, un coefficient de poisson de 0.342 et une limite d'élasticité de 900 MPa. Les ressorts ont une raideur de 1000 N/mm.

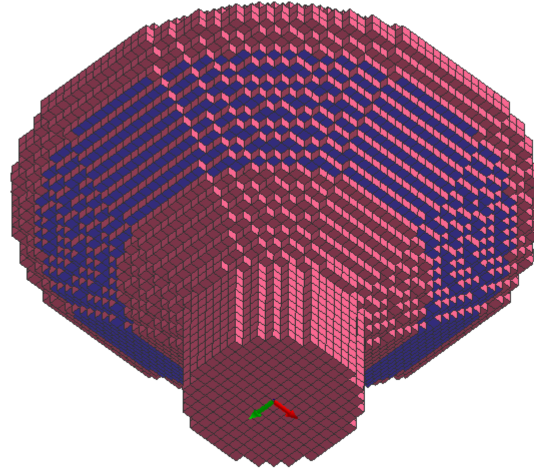


FIGURE III.9 – Zone de support prévue pour le cas de la toupie

Génération des fichiers pour l'optimisation des supports

Le maillage utilisé pour le calcul d'optimisation de support est composé de quatre parties, le tube (la pièce), la baseplate, le design space à l'intérieur du tube, le design space à l'extérieur du tube.

La baseplate est composé de 8814 noeuds et 7631 éléments hexaèdre.

Le tube est le même que celui utilisé pour le calcul AM avec les layers qui ont fusionné.

Le design space à l'intérieur du tube possède 178588 nœuds et 931680 éléments tetrahèdriques, celui de l'extérieur du tube a 178533 nœuds et 890003 éléments tetrahèdres.

Les propriétés mécaniques utilisées sont les mêmes que pour le calcul AM, à l'exception que le calcul est désormais purement élastique, il n'y a donc plus de limite d'élasticité. Le matériau ersatz dans le designspace à un module Young 1000 fois inférieur à celui du tube.

Optimisation des supports

L'objectif à minimiser dans cet exemple est le volume du support avec pour contrainte une différence entre le déplacement cible et celui obtenu à ne pas dépasser pour les nœuds se trouvant dans les surfaces supportées.

Résultats

Sur les figures III.14 et III.17 on peut observer les résultats obtenus après optimisation. Une tolérance différente concernant la différence entre le déplacement cible et le déplacement obtenu a été utilisée, celle-ci est représentée par la valeur epsilon.

Epsilon, qui représente le critère de déplacement, est la norme de la différence entre la valeur cible de déplacement et la valeur obtenue pour une liste de nœuds données. Nous ne comparons ici uniquement les déplacements verticaux, suivant z.

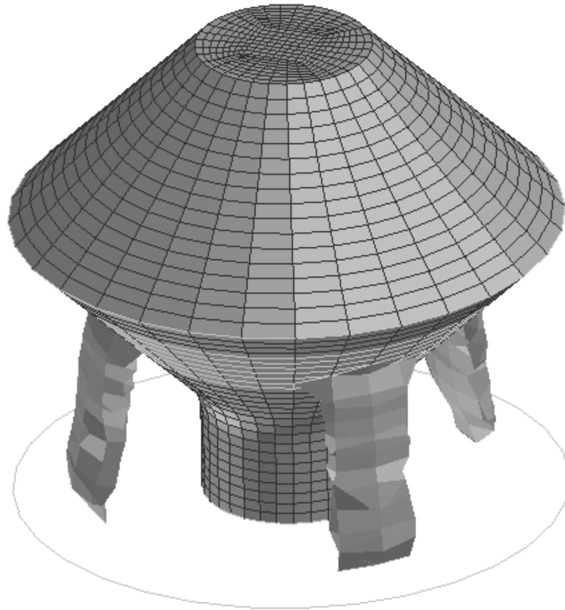


FIGURE III.10 – Supports issus du calcul d'optimisation en partant d'un calcul AM

III.4 Couplage avec la méthode géométrique de support des parties en overhang

Un des inconvénients içi est que le support ne colle pas totalement à la surface en overhang, la pièce ne sera donc pas imprimable. La combinaison de ce calcul avec un calcul prenant en compte les surfaces en overhang comme présentée en première partie de ce rapport permet de palier au problème.

Le résultat d'un tel calcul est visible sur la figure III.20. On observe que les supports obtenus sont similaires à ceux obtenus précédemment mais que cette fois ci la surface en overhang est totalement supportée.

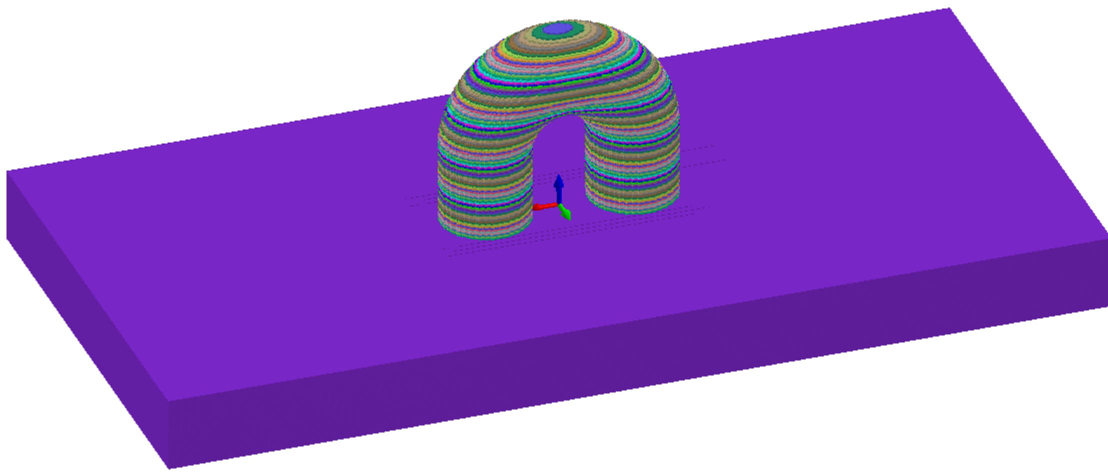


Figure III.11 – Maillage initial du tube pour le calcul AM

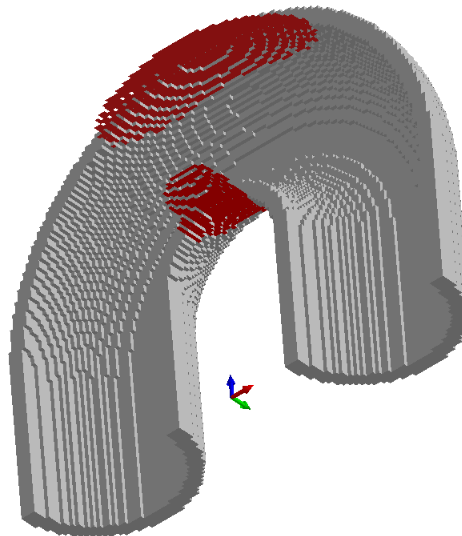


Figure III.12 – Zone d'application des ressorts

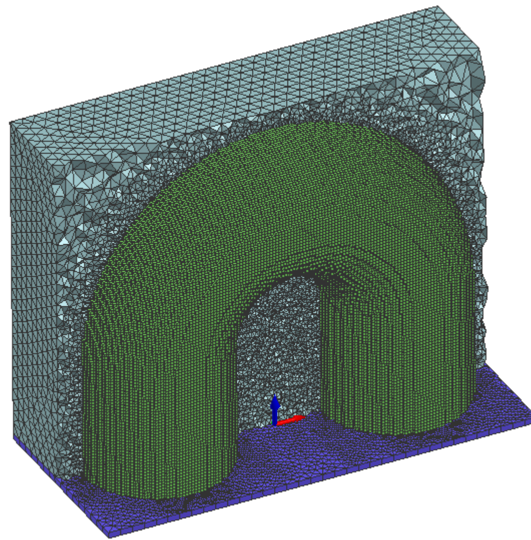


Figure III.13 – Maillage initial du tube pour le calcul d'optimisation

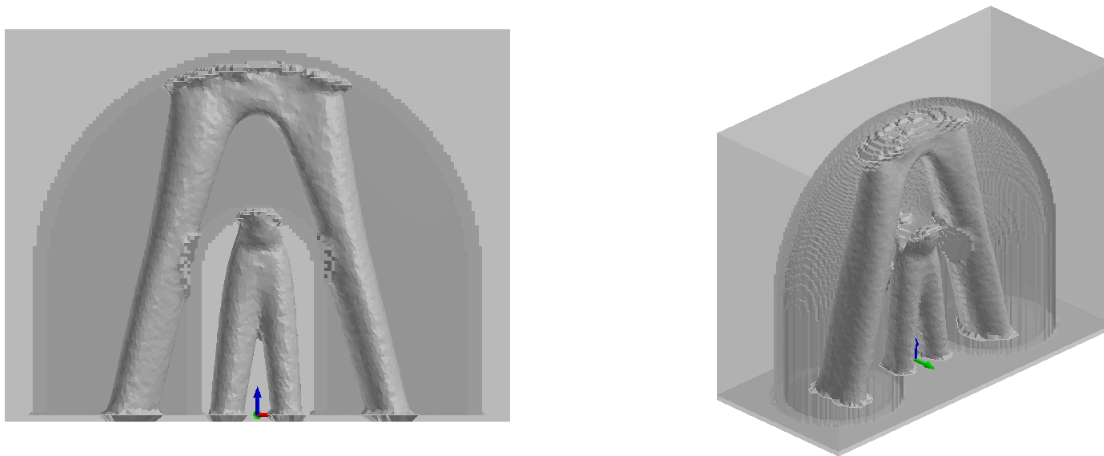


Figure III.14 – Supports obtenus dans le cas d'un tube en U, avec un epsilon de 4

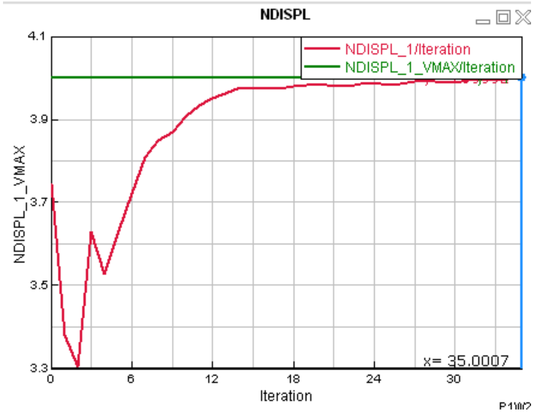


Figure III.15 – Évolution du delta de déplacement vis à vis des déplacements cibles en fonction du nombre d'itération

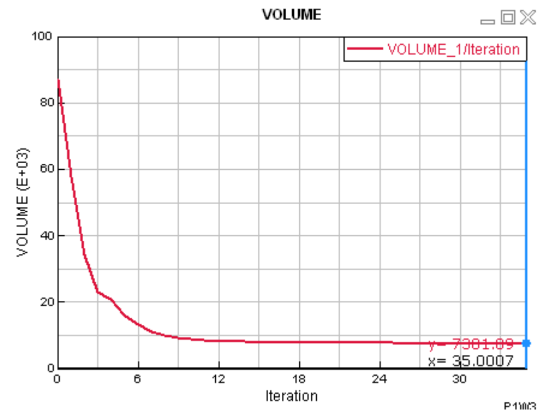
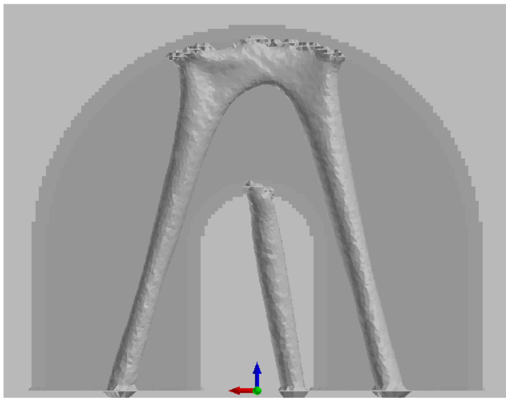


Figure III.16 – Évolution du volume des supports en fonction du nombre d'itération



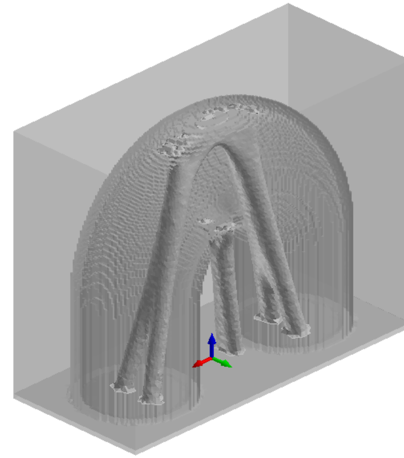


Figure III.17 – Supports obtenus dans le cas d'un tube en U, avec un epsilon de 5

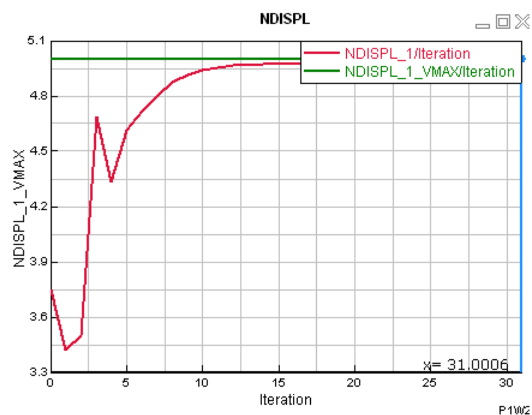


Figure III.18 – Epsilon en fonction du nombre d'itération

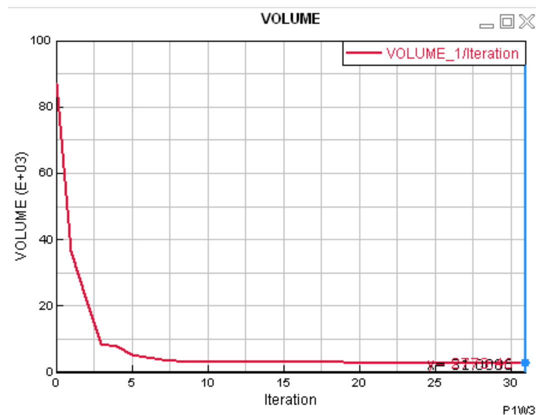
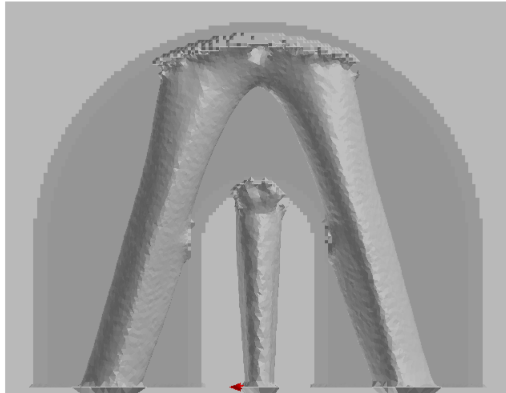


Figure III.19 – Évolution du volume des supports en fonction du nombre d'itération



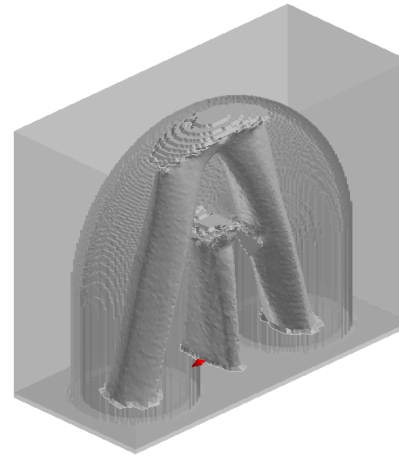


Figure III.20 – Supports obtenus dans le cas d'un tube en U, avec un support complet des surfaces en overhang

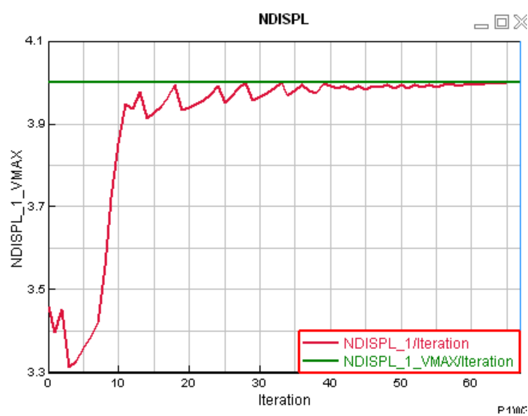


Figure III.21 – Epsilon en fonction du nombre d'itération

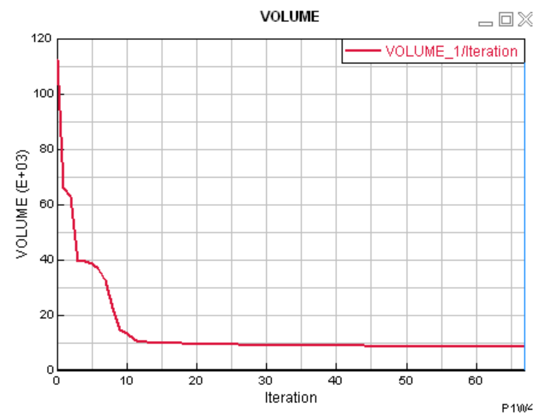


Figure III.22 – Évolution du volume des supports en fonction du nombre d'itération

Bibliographie

[1] Grégoire Allaire. *Conception optimale de structures*, volume 58 of *Mathématiques & Applications (Berlin) [Mathematics & Applications]*. Springer-Verlag, Berlin, 2007.

[2] Grégoire Allaire, Martin Bihr, and Beniamin Bogosel. Support optimization in additive manufacturing for geometric and thermo-mechanical constraints. *Structural and Multidisciplinary Optimization*, 61(6) :2377–2399, April 2020.

[3] Grégoire Allaire and Beniamin Bogosel. Optimizing supports for additive manufacturing. *Struct. Multidiscip. Optim.*, 58(6) :2493–2515, 2018.

[4] C. Barlier and A. Bernard. *Fabrication additive - Du Prototypage Rapide à l'impression 3D.* Dunod, Paris, 2016.

[5] Martin P. Bendsøe and Ole Sigmund. *Topology Optimization.* Springer Berlin Heidelberg, 2004.

[6] Martin Bihr. Optimisation topologique du couple pièce/support pour la fabrication additive sur lit de poudre, 2021. à paraitre.

[7] Simone Cacace, Emiliano Cristiani, and Leonardo Rocchi. A level set based method for fixing overhangs in 3D printing. *Appl. Math. Model.*, 44 :446–455, 2017.

[8] F. Calignano. Design optimization of supports for overhanging structures in aluminium and titanium alloys by selective laser melting. *Materials & Design*, 64 :203–213, 2014.

[9] Sandrine Dischert. Dream up your most innovative lightweight designs with topology optimization. Technical report, ESI Group, 2021.

[10] Jérémie Dumas, Jean Hergel, and Sylvain Lefebvre. Bridging the gap : Automated steady scaffoldings for 3d printing. *ACM Trans. Graph.*, 33(4) :98 :1–98 :10, July 2014.

[11] M.X. Gan and C.H. Wong. Practical support structures for selective laser melting. *Journal of Materials Processing Technology*, 238 :474 – 484, 2016.

[12] ESI Group. `https://www.esi-group.com/`.

[13] Kailun Hu, Shuo Jin, and Charlie C.L. Wang. Support slimming for single material based additive manufacturing. *Computer-Aided Design*, 65 :1 – 10, 2015.

[14] Xiaomao Huang, Chunsheng Ye, Siyu Wu, Kaibo Guo, and Jianhua Mo. Sloping wall structure support generation for fused deposition modeling. *The International Journal of Advanced Manufacturing Technology*, 42(11) :1074, Aug 2008.

[15] Ahmed Hussein, Liang Hao, Chunze Yan, Richard Everson, and Philippe Young. Advanced lattice support structures for metal additive manufacturing. *Journal of Materials Processing Technology*, 213(7) :1019 – 1026, 2013.

[16] Yu-Hsin Kuo, Chih-Chun Cheng, Yang-Shan Lin, and Cheng-Hung San. Support structure design in additive manufacturing based on topology optimization. *Struct. Multidiscip. Optim.*, 57(1) :183–195, 2018.

[17] Damien Lachouette and Philippe Conraux. Harmonic response optimization using esi topaze. In *CSMA 2019*, 2019.

[18] Matthijs Langelaar. Topology optimization of 3d self-supporting structures for additive manufacturing. *Additive Manufacturing*, 12(Part A) :60 – 70, 2016.

[19] Matthijs Langelaar. Combined optimization of part topology, support structure layout and build orientation for additive manufacturing. *Structural and Multidisciplinary Optimization*, 57(5) :1985–2004, 2018.

[20] Amir M. Mirzendehdel and Krishnan Suresh. Support structure constrained topology optimization for additive manufacturing. *Computer-Aided Design*, 81 :1 – 13, 2016.

[21] Tomas Möller and Ben Trumbore. Fast, minimum storage ray-triangle intersection. *Journal of Graphics Tools*, 2(1) :21–28, 1997.

[22] K. Mumtaz, P. Vora, and N. Hopkinson. A method to eliminate anchors/supports from directly laser melted metal powder bed processes. *Proc. Solid Freeform Fabrication Symposium, Sheffield*, pages 54–64, 2011.

[23] G. Strano, L. Hao, R. M. Everson, and K. E. Evans. A new approach to the design and optimisation of support structures in additive manufacturing. *The International Journal of Advanced Manufacturing Technology*, 66(9) :1247–1254, Jun 2013.

[24] J. Vanek, J. A. G. Galicia, and B. Benes. Clever support : Efficient support structure generation for digital fabrication. *Computer Graphics Forum*, 33(5) :117–125, 2014.

Annexe A

Scripts de détection des zones inaccessibles (getInaccessibleNodes.py)

```
#!/usr/bin/env python

workFile=r"C:\Users\Alexis\fichier.ASC"
refAngle=45.0
refNormal=[0.0,0.0,-1.0]
refVec=""
model=""
null=''


def defineRefNormal():
    ''' Build the reference vector used to test overhang
    '''

    global refNormal
    global refVec

    origin = VgPoint3()
    origin.x = 0.0 ; origin.y = 0.0 ; origin.y = 0.0 ;

    atRefNormal= VgPoint3()
    atRefNormal.x = refNormal[0]
    atRefNormal.y = refNormal[1]
    atRefNormal.z = refNormal[2]

    ret, refVec = VistaDb.MathComputeVector(origin,atRefNormal)
    ret, refVec = VistaDb.MathNormaliseVector(refVec)
    print(refVec)
    return ""


def isOverHang(elm_normal):
    '''
    Arguments:
        elm_normal: Vista 3D Point storing a normal vector
    Functionality:
```

```
37            Decides if the element overhang
38        Return Values:
39            1 if the element overhang, and 0 otherwise
40        '''
41        global refVec
42        global refAngle
43
44        cosAngle= elm_normal.x*refVec.x + elm_normal.y*refVec.y + elm_normal.z*
      refVec.z
45        angle=math.acos(cosAngle)*180.0/math.pi
46        return (angle<=refAngle)
47
48
49    def selectOverhangElements(selEleList):
50        '''
51        Arguments:
52            selEleList: List of Elements to test
53        Functionality:
54            Split the list of elements in two : overhang and correct
55        Return Values:
56            overhangEleList: list of elements with overhang
57            okEleList      : list of elements without overhang
58        '''
59        global model
60
61        # If you wan to select it
62        #hostutl.DeSelectAll(model)
63        #selEleList = hostutl.GetEntityUsingEntitySelector(model,r"2D Elements"
      ,"By",r"2D Elements",[],"Select 2D Elements to test.")
64
65        overhangEleList=[]
66        okEleList=[]
67        for element in selEleList:
68            ret, elm_normal = VistaDb.ElementComputeNormal(element)
69            #ret, elm_normal = VistaDb.MathNormaliseVector(elm_normal)
70            if isOverHang(elm_normal[0]):
71                overhangEleList.append(element)
72            else:
73                okEleList.append(element)
74
75        print("Nb Elements overhang: "+str(len(overhangEleList)))
76        print("Nb Elements ok: "+str(len(okEleList)))
77        return overhangEleList, okEleList
78
79
80    def createCollector(toKeepEleList, toRemoveEleList):
81        '''
82        Arguments:
83            toKeepEleList: list of elements with overhang
84            toRemoveEleList      : list of elements without overhang
85        Functionality:
86            Create a collector with the element to keep, and remove the rest
87        Return Values:
88            None
89        '''
```

```
90        global model
91        NewColl= VistaDb.ModelCreateGroupForGivenEntities(model,len(
      toKeepEleList),toKeepEleList,"OVERHANG_"+str(refAngle))
92        VistaDb.ModelDestroyElements(model, len(toRemoveEleList),
      toRemoveEleList)
93        VExpMngr.RefreshExplorer(r"")
94
95
96
97  loadFile(workFile)
98  defineRefNormal()
99  partSkin, elemSkin=createSkinElements2()
100 overhangEleList, okEleList=selectOverhangElements(elemSkin)
101 createCollector(overhangEleList, okEleList)
102 }
```

Annexe B

Scripts de détection des zones en overhang (getOverhang.py)

```
1
2
3  #!/usr/bin/env python
4
5  refAngle=45.0
6  refNormal=[0.0,0.0,-1.0]
7  refVec=""
8  model=""
9  null=''
10 from VgPoint3 import *
11 from VgPoint2 import *
12 from VgMatrix import *
13
14 import time
15
16 import numpy as np
17 import math
18
19 import VScn
20
21 def circumSphereTriangle(triangleList):
22     circumSphereList = []
23     A = np.array(([0.0,0.0,0.0],[0.0,0.0,0.0],[0.0,0.0,0.0]))
24     B = np.array([0.0,0.0,0.0])
25     C = np.array([0.0,0.0,0.0])
26     x1 = 0.0
27     y1 = 0.0
28     z1 = 0.0
29     x2 = 0.0
30     y2 = 0.0
31     z2 = 0.0
32     x3 = 0.0
33     y3 = 0.0
34     z3 = 0.0
35     for triangle in triangleList:
36         x1 = triangle[0][0]
```

```
37          y1 = triangle[0][1]
38          z1 = triangle[0][2]
39          x2 = triangle[1][0]
40          y2 = triangle[1][1]
41          z2 = triangle[1][2]
42          x3 = triangle[2][0]
43          y3 = triangle[2][1]
44          z3 = triangle[2][2]
45          A[0][0] = 2*(x2-x1)
46          A[0][1] = 2*(y2-y1)
47          A[0][2] = 2*(z2-z1)
48          A[1][0] = 2*(x3-x1)
49          A[1][1] = 2*(y3-y1)
50          A[1][2] = 2*(z3-z1)
51          A[2][0] = ((y2-y1)*(z3-z1))-((y3-y1)*(z2-z1))
52          A[2][1] = ((z2-z1)*(x3-x1))-((z3-z1)*(x2-x1))
53          A[2][2] = ((x2-x1)*(y3-y1))-((x3-x1)*(y2-y1))
54          B[0] = (x2*x2+y2*y2+z2*z2)-(x1*x1+y1*y1+z1*z1)
55          B[1] = (x3*x3+y3*y3+z3*z3)-(x1*x1+y1*y1+z1*z1)
56          B[2] = x1*(((y2-y1)*(z3-z1))-((y3-y1)*(z2-z1)))+y1*(((z2-z1)*(x3-x1
        ))-((z3-z1)*(x2-x1)))+z1*(((x2-x1)*(y3-y1))-((x3-x1)*(y2-y1)))
57          C = np.linalg.solve(A, B)
58          circumSphereList.append([C[0],C[1],C[2],(C[0]-x1)*(C[0]-x1)+(C[1]-
        y1)*(C[1]-y1)+(C[2]-z1)*(C[2]-z1)])
59      return circumSphereList
60
61  def intersectTriangleSegment(triangle,pointToTest,epsilon,edge_1,edge_2,h,s
        ,q,a,f,u,v,t,segment_vector):
62
63      if(triangle[0]==pointToTest):
64          return False
65      if(triangle[1]==pointToTest):
66          return False
67      if(triangle[2]==pointToTest):
68          return False
69      for i in range(3):
70          edge_1[i] = triangle[1][i] - triangle[0][i]
71          edge_2[i] = triangle[2][i] - triangle[0][i]
72
73      h[0] = segment_vector[1]*edge_2[2] - segment_vector[2]*edge_2[1]
74      h[1] = segment_vector[2]*edge_2[0] - segment_vector[0]*edge_2[2]
75      h[2] = segment_vector[0]*edge_2[1] - segment_vector[1]*edge_2[0]
76      a=edge_1[0]*h[0]+edge_1[1]*h[1]+edge_1[2]*h[2]
77      if(a > -epsilon and a < epsilon):
78          return False
79
80      f = 1.0/a
81      for i in range(3):
82          s[i] = pointToTest[i] - triangle[0][i]
83      u=s[0]*h[0]+s[1]*h[1]+s[2]*h[2]
84      u*=f
85      u = 0
86      for i in range(3):
87          u=f*s[i]*h[i]+u
88      if (u<0.0 or u>1.0):
```

```
89            return False
90
91        q[0] = s[1]*edge_1[2] - s[2]*edge_1[1]
92        q[1] = s[2]*edge_1[0] - s[0]*edge_1[2]
93        q[2] = s[0]*edge_1[1] - s[1]*edge_1[0]
94        v=segment_vector[0]*q[0]+segment_vector[1]*q[1]+segment_vector[2]*q[2]
95        v*=f
96        if (v<0.0 or u+v>1.0):
97            return False
98        t=edge_2[0]*q[0]+edge_2[1]*q[1]+edge_2[2]*q[2]
99        t*=f
100       if (t>epsilon):
101           return True
102       else:
103           return False
104
105   def preparationBox(pointToTest,pointObservation,max_segment,min_segment):
106       max_segment[0] = max([pointToTest[0],pointObservation[0]])
107       max_segment[1] = max([pointToTest[1],pointObservation[1]])
108       max_segment[2] = max([pointToTest[2],pointObservation[2]])
109       min_segment[0] = min([pointToTest[0],pointObservation[0]])
110       min_segment[1] = min([pointToTest[1],pointObservation[1]])
111       min_segment[2] = min([pointToTest[2],pointObservation[2]])
112       return
113
114   def preparationBubble(pointToTest,pointObservation,norme,
          segment_vector_norme,segment_vector):
115       segment_vector[0] = pointObservation[0]-pointToTest[0]
116       segment_vector[1] = pointObservation[1]-pointToTest[1]
117       segment_vector[2] = pointObservation[2]-pointToTest[2]
118       norme = math.sqrt(segment_vector[0]*segment_vector[0]+segment_vector
          [1]*segment_vector[1]+segment_vector[2]*segment_vector[2])
119       segment_vector_norme[0] = segment_vector[0]/norme
120       segment_vector_norme[1] = segment_vector[1]/norme
121       segment_vector_norme[2] = segment_vector[2]/norme
122       return
123
124   def testBox(maxVector,minVector,min_segment,max_segment):
125       if(maxVector[0]<min_segment[0] or maxVector[1]<min_segment[1] or
          maxVector[2]<min_segment[2]):
126           return True
127       elif(minVector[0]>max_segment[0] or minVector[1]>max_segment[1] or
          minVector[2]>max_segment[2]):
128           return True
129       return False
130
131   def testBubble(center_to_segment,circumSphereList,pointToTest,
          segment_vector_norme,h,norme,jj):
132       center_to_segment[0] = circumSphereList[jj][0]-pointToTest[0]
133       center_to_segment[1] = circumSphereList[jj][1]-pointToTest[1]
134       center_to_segment[2] = circumSphereList[jj][2]-pointToTest[2]
135       h[0] = center_to_segment[1]*segment_vector_norme[2] - center_to_segment
          [2]*segment_vector_norme[1]
136       if(h[0]*h[0]>circumSphereList[jj][3]):
137           return True
```

```
138         h[1] = center_to_segment[2]*segment_vector_norme[0] - center_to_segment
        [0]*segment_vector_norme[2]
139         if(h[1]*h[1]>circumSphereList[jj][3]):
140             return True
141         h[2] = center_to_segment[0]*segment_vector_norme[1] - center_to_segment
        [1]*segment_vector_norme[0]
142         if(h[2]*h[2]>circumSphereList[jj][3]):
143             return True
144         norme = h[0]*h[0]+h[1]*h[1]+h[2]*h[2]
145         if(norme>circumSphereList[jj][3]):
146             return True
147         return False
148 
149 
150         return SkinPart
151 
152 
153 
154 
155 def selectInaccessibleElements(selEleList,selEleListDesign):
        '''
156 
157         Arguments:
158             selEleList: List of Elements to test
159         Functionality:
160             Split the list of elements in two : overhang and correct
161         Return Values:
162             overhangEleList: list of elements with overhang
163             okEleList       : list of elements without overhang
        '''
164 
165         global model
166 
167         # If you wan to select it
168         #hostutl.DeSelectAll(model)
169         #selEleList = hostutl.GetEntityUsingEntitySelector(model,r"2D Elements"
        ,"By",r"2D Elements",[],"Select 2D Elements to test.")
170 
171         overhangEleList=[]
172         okEleList=[]
173         rayList=[]
174         pointToTestList=[]
175         pointObservationList=[]
176         triangleList=[]
177         triangleList_0=[]
178         triangleList_1=[]
179         triangleList_2=[]
180         triangle_0=[0.0,0.0,0.0]
181         triangle_1=[0.0,0.0,0.0]
182         triangle_2=[0.0,0.0,0.0]
183         rayList.append(1.0)
184         rayList.append(2.0)
185         i=0
186         j=0
187         k=0
188         ret,num_part,n_list_part = VistaDb.ModelGetNodesFromElementList(len(
        selEleList),selEleList)
```

```
189     ret,num_design,n_list_design = VistaDb.ModelGetNodesFromElementList(len
    (selEleListDesign),selEleListDesign)
190     observationList
    =[[-50,-20,-10],[-50,-20,60],[-50,20,-10],[-50,20,60],[50,-20,-10],[50,-20,60],[50,2
191     observationList.append([0.00001,-20.0,-10.0])
192     observationList.append([0.00001,-20.0,60.0])
193     observationList.append([0.00001,20.0,-10.0])
194     observationList.append([0.00001,20.0,60.0])
195     observationList.append([-50,0.00001,-10])
196     observationList.append([-50,0.00001,60])
197     observationList.append([50,0.00001,-10])
198     observationList.append([50,0.00001,60])
199     observationList.append([-50,-20,25])
200     observationList.append([-50,20,25])
201     observationList.append([50,-20,25])
202     observationList.append([50,20,25])
203     observationList.append([20,0.00001,0.00001])
204     observationList.append([-20,0.00001,0.00001])
205
206
207
208     print("Il y a ", num_part, " noeuds sur la surface de la piece")
209
210     edge_1=[0.0,0.0,0.0]
211     edge_2=[0.0,0.0,0.0]
212     h=[0.0,0.0,0.0]
213     s=[0.0,0.0,0.0]
214     q=[0.0,0.0,0.0]
215     center_to_segment = [0.0,0.0,0.0]
216     segment_vector=[0.0,0.0,0.0]
217     segment_vector_norme=[0.0,0.0,0.0]
218     norme=0.0
219     epsilon=0.0000001
220     a=0.0
221     f=0.0
222     u=0.0
223     v=0.0
224     t=0.0
225     maxVectorListX = []
226     minVectorListX = []
227     maxVectorListY = []
228     minVectorListY = []
229     maxVectorListZ = []
230     minVectorListZ = []
231
232     for point in n_list_part:
233         pointToTestList.append([VistaDb.GetVgPoint3Value(point,"Point").x,
    VistaDb.GetVgPoint3Value(point,"Point").y,VistaDb.GetVgPoint3Value(point
    ,"Point").z])
234     for point in n_list_design:
235         pointObservationList.append([VistaDb.GetVgPoint3Value(point,"Point"
    ).x,VistaDb.GetVgPoint3Value(point,"Point").y,VistaDb.GetVgPoint3Value(
    point,"Point").z])
236     for triangle in selEleList:
```

```
237            ret , nbNode , lstNodes=VistaDb . ElementGetNodeList ( triangle )
238            triangleList . append ([[ VistaDb . GetVgPoint3Value ( lstNodes [ 0 ] , " Point " )
        . x , VistaDb . GetVgPoint3Value ( lstNodes [ 0 ] , " Point " ) . y , VistaDb .
        GetVgPoint3Value ( lstNodes [ 0 ] , " Point " ) . z ] ,[ VistaDb . GetVgPoint3Value (
        lstNodes [ 1 ] , " Point " ) . x , VistaDb . GetVgPoint3Value ( lstNodes [ 1 ] , " Point " ) . y ,
        VistaDb . GetVgPoint3Value ( lstNodes [ 1 ] , " Point " ) . z ] ,[ VistaDb .
        GetVgPoint3Value ( lstNodes [ 2 ] , " Point " ) . x , VistaDb . GetVgPoint3Value (
        lstNodes [ 2 ] , " Point " ) . y , VistaDb . GetVgPoint3Value ( lstNodes [ 2 ] , " Point " ) . z
        ]])
239        for ii in range ( len ( triangleList ) ) :
240            minX = min ( triangleList [ ii ][0][0] , triangleList [ ii ][1][0] ,
        triangleList [ ii ][2][0])
241            minY = min ( triangleList [ ii ][0][1] , triangleList [ ii ][1][1] ,
        triangleList [ ii ][2][1])
242            minZ = min ( triangleList [ ii ][0][2] , triangleList [ ii ][1][2] ,
        triangleList [ ii ][2][2])
243            maxX = max ( triangleList [ ii ][0][0] , triangleList [ ii ][1][0] ,
        triangleList [ ii ][2][0])
244            maxY = max ( triangleList [ ii ][0][1] , triangleList [ ii ][1][1] ,
        triangleList [ ii ][2][1])
245            maxZ = max ( triangleList [ ii ][0][2] , triangleList [ ii ][1][2] ,
        triangleList [ ii ][2][2])
246            maxVectorListX . append (maxX)
247            minVectorListX . append (minX)
248            maxVectorListY . append (maxY)
249            minVectorListY . append (minY)
250            maxVectorListZ . append (maxZ)
251            minVectorListZ . append (minZ)
252
253        circumSphereList = circumSphereTriangle ( triangleList )
254
255        max_segment =[0 ,0 ,0]
256        min_segment =[0 ,0 ,0]
257
258        print ( " Il y a " , len ( triangleList ) , " triangles sur la surface de la
        piece " )
259
260        tps_total_bubble = 0.0
261        tps_total_bubble_pre = 0.0
262        tps_total_box = 0.0
263        tps_total_box_pre = 0.0
264        tps_total_triangle_inter = 0.0
265        tps_add_pts = 0.0
266        tps_tmp_1 = 0.0
267        tps_tmp_2 = 0.0
268
269        tmp_bool = True
270
271        tps_ini = time . clock ()
272        i =0
273        j =0
274        ii =-1
275
276        for ii in range (1000) :
277            pointToTest = pointToTestList [ ii ]
```

```
278 #     for pointToTest in pointToTestList:
279 #         ii+=1
280         for pointObservation in observationList:
281             isAccessible = True
282             preparationBox(pointToTest,pointObservation,max_segment,
    min_segment)
283             preparationBubble(pointToTest,pointObservation,norme,
    segment_vector_norme,segment_vector)
284             segment_vector[0] = pointObservation[0]-pointToTest[0]
285             segment_vector[1] = pointObservation[1]-pointToTest[1]
286             segment_vector[2] = pointObservation[2]-pointToTest[2]
287             jj=-1
288             for triangle in triangleList:
289                 jj=1+jj
290                 if(maxVectorListX[jj]<min_segment[0] or maxVectorListY[jj]<
    min_segment[1] or maxVectorListZ[jj]<min_segment[2] or minVectorListX[jj
    ]>max_segment[0] or minVectorListY[jj]>max_segment[1] or minVectorListZ[
    jj]>max_segment[2]):
291                     continue
292                 if(testBubble(center_to_segment,circumSphereList,
    pointToTest,segment_vector_norme,h,norme,jj)):
293                     continue
294                 i=1+i
295                 if(intersectTriangleSegment(triangle,pointToTest,epsilon,
    edge_1,edge_2,h,s,q,a,f,u,v,t,segment_vector)):
296                     isAccessible=False
297                     break
298             if(isAccessible):
299                 okEleList.append(n_list_part[ii])
300                 break
301         if(isAccessible==False):
302             overhangEleList.append(n_list_part[ii])
303     tps_total2 = time.clock()-tps_ini
304     print("Temps de calculs box pre  : ",tps_total_box_pre)
305     print("Temps de calculs box : ",tps_total_box)
306     print("Temps de calculs inter : ",tps_total_triangle_inter)
307     print("Temps de calculs  : ",tps_total2)
308     print("On a réalisé  : ",i," calcul de triangle")
309     return overhangEleList,okEleList
310 
311 
312 loadFile(workFile)
313 partSkin, elemSkin=createSkinElements()
314 partSkin_design, elemSkin_design=createSkinElements()
315 overhangEleList, okEleList=selectInaccessibleElements(elemSkin,
    elemSkin_design)
316 createCollector(overhangEleList, okEleList)
317 
318 
319 
320 #VE.DeleteWindow()
```