

École Publique d'ingénieries et d'ingénieurs en 3 ans

Rapport

ANALYSE TOPOLOGIQUE DE MOUVEMENTS DE GRAINS DANS UNE SÉQUENCE D'IMAGES 3D

—

PROJET INDUSTRIEL

Tutoré par Yukiko KENMOCHI & Julien MENDES-FORTE

Février 2025

Léo LEFEBVRE
Hugo LEMOISSON

Jonathan PALISSE
Elliot OLIVÉ



www.ensicaen.fr

Remerciements

Nous remercions Yukiko KENMOCHI et Julien MENDES-FORTE, chercheurs au GREYC et enseignants à l'ENSICAEN, pour nous avoir fait confiance en nous délégant ce projet.

Nous remercions également Gustavo PINZON FORERO, géomécanicien et l'un des principaux acteurs de ce projet, pour nous avoir fourni différents retours au cours de ce dernier.

Nous remercions Lysandre MACKE, doctorante en géométrie digitale, qui a produit le code et les outils à l'origine de ce projet lors de son stage au GREYC.

Table des matières

Remerciements	1
Table des matières	2
Introduction	3
I. Gestion de projet	4
1) Présentation et organisation de l'équipe	4
2) Objectifs et outils du projet	5
II. Travail Effectué	6
1) Segmentation	6
a) MaxTree	6
b) MinTree	7
c) Méthodes de segmentation	9
Segmentation de grains par Watershed	9
Segmentation de grains par Successive Watershed	11
2) Suppression des parties indésirables	13
a) Suppression des membranes latérales	13
b) Suppression des plateaux	13
b.1) Effacement direct sur les images	13
b.2) Assistance par watershed	14
3) Détection des contacts	15
4) Visualisation 3D interactive	17
a) Lecture des composantes et outils nécessaires à l'affichage	17
b) Affichage évolution d'un ou plusieurs grains dans le temps	18
III. Regard critique et perspectives d'amélioration	21
1) Organisation au sein du groupe et du projet	21
2) Futures fonctionnalités	21
Conclusion	22
Références	23
Annexes	24

Introduction

L'analyse des séquences d'images tridimensionnelles (3D) joue un rôle clé dans de nombreux domaines scientifiques, notamment en science des matériaux, en médecine et en biologie. Dans ces disciplines, la capacité à détecter et à quantifier les changements structurels au fil du temps est essentielle pour mieux comprendre les phénomènes dynamiques. En particulier, l'étude des matériaux granulaires, tels que les assemblages de particules, repose sur l'observation fine de leur évolution sous contrainte mécanique.

Dans ce contexte, la tomographie à rayons X permet d'acquérir des séquences d'images 3D détaillées de ces matériaux¹, offrant ainsi une vision précise de leur déformation et de leur réorganisation interne. Parmi les transformations observées, les changements topologiques (qui reflètent les modifications de la connectivité et de l'agencement des grains) sont d'une importance capitale. Toutefois, ces aspects restent encore peu étudiés, bien que leur compréhension soit cruciale pour modéliser des phénomènes complexes tels que l'apparition de bandes de cisaillement ou l'évolution des réseaux de contacts entre particules.

L'objectif de ce travail est d'analyser les changements topologiques intervenant lors du mouvement des grains dans une séquence d'images 3D. En quantifiant ces variations et en proposant de nouvelles méthodes d'identification, nous visons à améliorer la compréhension des dynamiques granulaires et à fournir des outils plus performants pour l'étude des matériaux déformables. Cette séquence d'images consiste, dans nos futurs exemples, de vidéos d'une cinquantaine d'images 3D.

Ce rapport a pour objectif de vous présenter comment notre équipe a pu entreprendre ce projet industriel. Nous l'avons divisé en trois parties.

Premièrement, nous analyserons comment la gestion de projet a été conçue par notre équipe. Dans la suite de ce rapport, nous détaillerons l'ensemble des travaux réalisés. Pour mener à bien cette étude, plusieurs étapes ont été mises en place afin d'extraire et d'analyser différents paramètres issus des images 3D. Dans un premier temps, nous avons appliqué des méthodes de segmentation permettant d'identifier et de labelliser les grains présents dans les images. Ensuite, afin de faciliter le travail des futurs chercheurs et étudiants, nous avons développé des ressources spécifiques permettant de filtrer et de supprimer les éléments indésirables des séquences 3D. Nous nous sommes ensuite intéressés à la récupération des points de contact entre les grains en exploitant des techniques de squelettisation, essentielles pour analyser leur connectivité et leur évolution au fil du temps. Enfin, une fois l'ensemble des données collectées (coordonnées, labels, points de contact, etc.), nous les avons intégrées dans un programme développé spécifiquement pour leur visualisation, dont nous détaillerons le fonctionnement dans ce rapport. Dernièrement, nous allons apporter un regard critique sur la gestion de projet et sur les perspectives qui peuvent être envisagées dans ce projet industriel en termes de fonctionnalités.

¹Annexe I

I. Gestion de projet

Dans cette première partie, nous allons détailler la gestion de projet que nous avons mise en place. Premièrement, nous analyserons l'organisation de l'équipe et du projet. Deuxièmement, nous allons analyser comment ont évolué les objectifs et les outils.

1) Présentation et organisation de l'équipe

Pour ce projet, nous étions une équipe de quatre personnes :

- Jonathan PALISSE - Chef de projet
- Léo LEFEBVRE - Gestionnaire de Version
- Hugo LEMOISSON - Développeur
- Elliot OLIVÉ - Développeur

Il est important de noter que nous nous sommes orientés vers une organisation agile du projet. Par conséquent, nos rôles dans l'équipe étaient loin d'être fixes. Par exemple, toutes les personnes du projet ont eu un moment la casquette de développeur. Cependant, un seul rôle n'a pas été partagé au sein de l'équipe : celui du chef de projet. Ce rôle avait pour objectif de faire la connexion entre le client/tuteur, Mme KENMOCHI, et le reste de l'équipe.

Dans le contexte de l'agilité, avec l'accord du client et en raison du cadre scolaire (projets parallèles, examens, etc.). Nous avons décidé d'effectuer des sprints d'environ une semaine, ce qui permet de garder le rythme qui nous a été imposé : le mercredi était dédié au projet industriel. Avec l'outil Trello, nous déposions nos cartes répondant à différentes fonctionnalités à ajouter. Nous n'avons pas implémenté un système de niveau de difficulté, chacun choisissait l'activité qu'il souhaitait effectuer.

À la fin de chaque sprint, une réunion a été organisée avec le client pour obtenir les feedbacks des solutions apportées aux demandes apportées à la précédente réunion. Durant cette réunion, tous les éléments clés sont pris en notes et de nouvelles cartes d'activités ou de précédentes sont modifiées pour lancer le prochain sprint de la semaine.

Pour centraliser le code et permettre à notre chef de projet de suivre en direct les avancements du projet. Nous avons créé un dépôt GitLab.

2) Objectifs et outils du projet

Dans cette partie, nous allons faire un point détaillé sur les objectifs et les outils mis en place pour mener à bien notre projet. Suite à plusieurs échanges avec notre client, nous avons identifié un objectif central : extraire et analyser divers paramètres issus d'images de grains et en proposer une visualisation en trois dimensions de manière claire et exploitable. Ces paramètres incluent notamment les coordonnées précises des grains, leurs points de contact, les forces de contact, ainsi que d'autres caractéristiques pertinentes pour l'étude menée.

Afin d'atteindre cet objectif, nous avons opté pour le langage Python comme principal outil de développement. Ce choix s'explique par la richesse de son écosystème de bibliothèques adaptées à notre problématique. En particulier, nous avons utilisé la bibliothèque Polyscope, spécialisée dans la visualisation interactive en 3D, pour représenter graphiquement les grains et leurs interactions. Par ailleurs, nous avons également eu recours au langage C, notamment pour tirer parti de la bibliothèque Pink, développée par l'ESIEE Engineering. Pink est une bibliothèque spécialisée en traitement d'images, et son intégration nous a permis d'optimiser certaines opérations, notamment celles liées aux points de contacts.

Le traitement des images constituant une étape cruciale du projet, nous avons également exploité un code Python fourni par Lysandre MACKE, qui repose sur la bibliothèque Higra. Higra est une bibliothèque dédiée aux structures hiérarchiques en morphologie mathématique.

En complément de ces outils de développement, plusieurs logiciels externes ont été mobilisés pour faciliter l'exploitation des données. Notamment, Fiji (basé sur ImageJ) a été utilisé pour la visualisation des images brutes. Ces images originales, fournies sous forme de vidéos de 54 frames, représentent un volume de données conséquent, atteignant parfois plusieurs centaines de gigaoctets. En raison de cette volumétrie importante, leur stockage et leur manipulation ont nécessité l'utilisation de disques durs externes dédiés.

Par ailleurs, afin de pouvoir tester et affiner nos méthodes dans l'affichage visuel sans dépendre immédiatement de nos propres jeux de données, nous avons pu exploiter des paramètres déjà calculés sur une vidéo d'un cylindre, résultats d'un travail de thèse antérieur en lien avec notre sujet [1].

Enfin, nous avons défini principalement ces différents tâches :

- Segmenter les grains
 - Labelliser les grains
 - Récupérer les coordonnées des grains
- Supprimer les parties indésirables
- Détecter les contacts
- Visualiser de manières claires et concises les grains avec différents paramètres possibles.

C'est dans la deuxième partie de ce rapport que ces différentes tâches seront expliquées.

II. Travail Effectué

Dans cette deuxième partie du rapport, nous présentons le travail accompli tout au long du projet. Tout d'abord, nous présentons la partie segmentation des grains pour attribuer leur label. Deuxièmement, la partie suppression des parties indésirables à l'image d'origine. Troisièmement, différents traitements en lien avec les points de contacts des grains. Et dernièrement la visualisation des grains dans un modèle 3D avec différents paramètres modifiables par l'utilisateur.

1) Segmentation

La segmentation des grains visait à les séparer, leur attribuer des labels afin de les compter et d'analyser leurs caractéristiques. Nous présentons ici le processus suivi pour obtenir une segmentation efficace.

Choix de l'algorithme

Pour réaliser cette segmentation, nous avons envisagé l'utilisation de l'algorithme du watershed. Cependant, cette méthode nécessite des prérequis, comme les components tree que nous allons présenter.

Component Tree

Un component tree est un arbre formé à partir d'une image en niveau de gris tel que :

- Les noeuds de l'arbre correspondent à une composante connexe de l'image
- Un noeud N_1 est fils d'un autre noeud N_2 si la composante connexe correspondante à N_1 est inclus dans la composante connexe correspondante à N_2 .
- A chaque noeud N est associé une altitude

Le component tree permet donc de transformer une image en niveau de gris en arbre de ses composantes connexes afin de la traiter. Des exemples de composants Tree sont les MaxTree et les MinTree.

a) MaxTree

Le MaxTree d'une image I correspond à un component tree considérant les ensembles de niveaux supérieurs $I_k = \{x / I(x) \geq k\}$ pour définir l'altitude. En effet, dans ce cas, $altitude(N) = \max(\{k \in \mathbb{Z} / N \in N_{I_k}\})$. L'inégalité de I_k peut être stricte. Ainsi :

- Plus une composante possède un niveau bas, plus elle sera proche de la racine
- Les maxima locaux de l'image correspondent à des feuilles.

Le MaxTree permet ainsi de hiérarchiser l'ensemble des composantes connexes par niveau en mettant en avant les maxima locaux de l'image, qui seront représentés par des feuilles comme l'illustre la figure suivante :

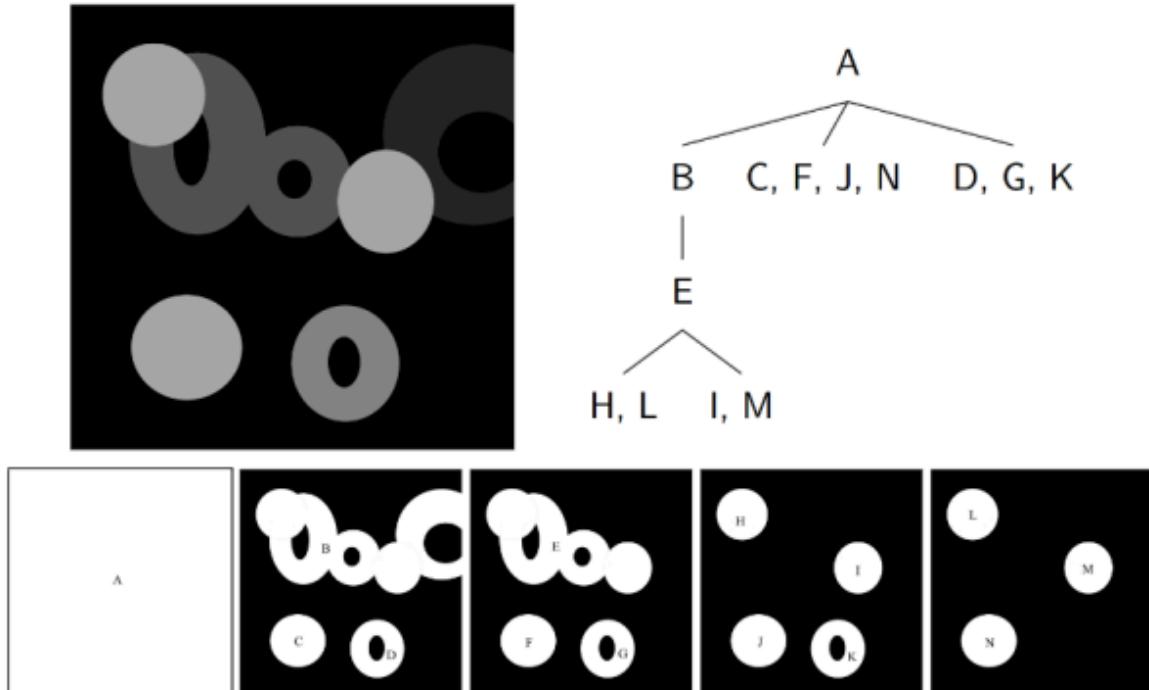


Figure 1.1 : MaxTree

b) MinTree

Le MinTree d'une image I correspond à un component tree considérant les ensembles de niveaux inférieurs $I_k = \{x / I(x) < k\}$ pour définir les composantes connexes de l'image. En effet, dans ce cas, $\text{altitude}(N) = \min(\{k \in \mathbb{Z} / N \in N_{I_k}\})$. L'inégalité de I_k peut être large. Par conséquent :

- Plus une composante possède un niveau élevé, plus elle sera proche de la racine
- Les minima locaux de l'image correspondent à des feuilles.

A l'inverse du MaxTree, le MinTree permet de mettre en avant les minimas locaux de l'image. En pratique, le MinTree d'une image correspond au MaxTree appliqué à son inverse, on parle d'arbre dual. Sur la figure suivante, le MinTree présente 5 feuilles indiquant ainsi 5 minima de l'image originale, respectivement 5 extrema dans l'image inverse.

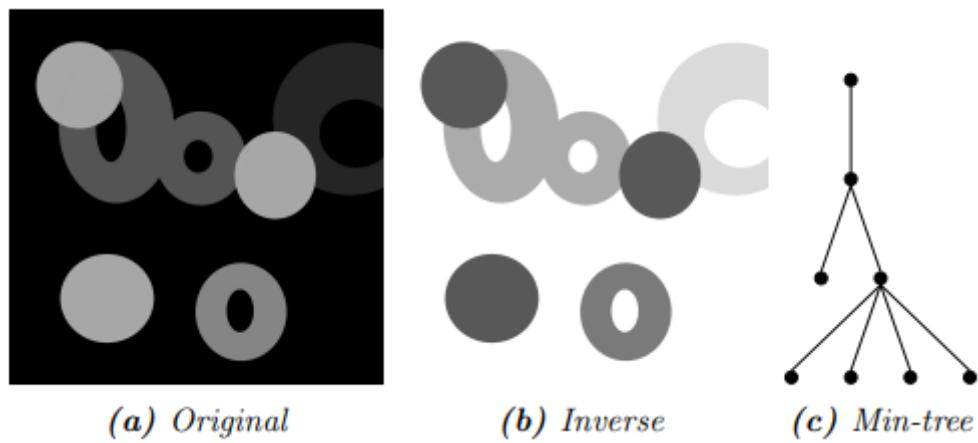


Figure 1.2 : MinTree [2]

L'algorithme MinTree (ou arbre des minima) est une structure utilisée pour extraire les composantes connectées des régions sombres d'une image en niveaux de gris. Il repose sur la construction d'un arbre des minima qui relie les pixels d'intensité croissante, permettant ainsi d'identifier les zones d'intérêt correspondant aux noyaux des grains.

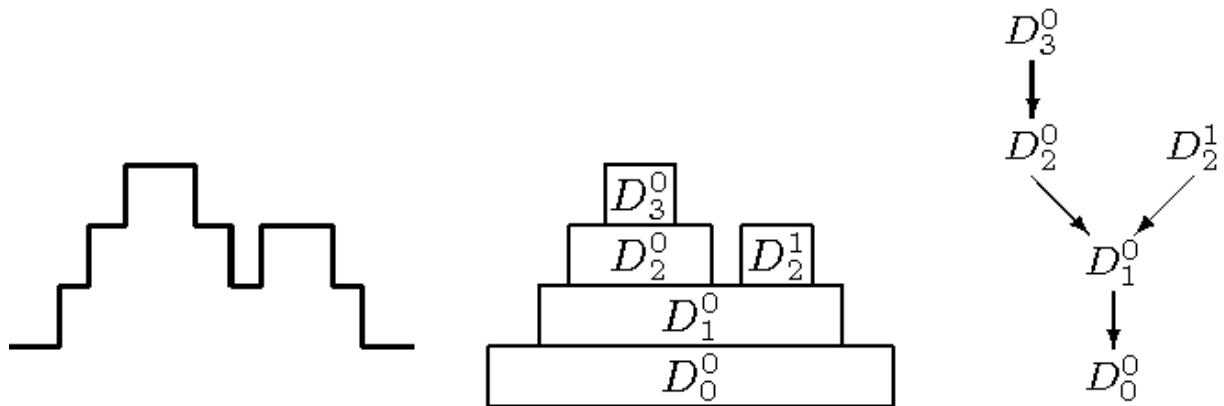


Figure 1.3 : Illustration MinTree

Étapes de l'algorithme MinTree

1. Construction de l'arbre des minima :
 - L'image en niveaux de gris est analysée pour identifier les bassins de minima locaux.
 - Ces bassins sont reliés entre eux pour former une hiérarchie d'intensité.
 2. Extraction des composantes pertinentes :
 - En fonction de critères de taille ou de contraste, certaines régions sont sélectionnées comme noyaux potentiels des grains.
 - Ces noyaux serviront de points de départ pour la segmentation finale.

En utilisant l'algorithme MinTree, nous avons pu obtenir des marqueurs précis pour chaque grain, facilitant ainsi l'application du watershed et garantissant une segmentation efficace et robuste².

c) Méthodes de segmentation

Segmentation de grains par Watershed

Une fois l'étape d'extraction de marqueurs réalisée à l'aide du MinTree, nous avons appliqué l'algorithme du watershed pour segmenter les grains de manière efficace. Cette méthode de segmentation repose sur une analogie avec le relief topographique, où une image est interprétée comme une surface dont les variations d'intensité correspondent à des altitudes.

Le principe du watershed consiste à considérer les minima locaux de l'image comme des bassins, puis à simuler une montée progressive du niveau d'eau dans ces bassins. Lorsque deux bassins en expansion se rencontrent, une ligne de partage des eaux est tracée pour marquer la frontière entre eux. Cette approche est particulièrement adaptée à la segmentation des grains, car elle permet de séparer les grains adjacents qui seraient autrement fusionnés en une seule composante.

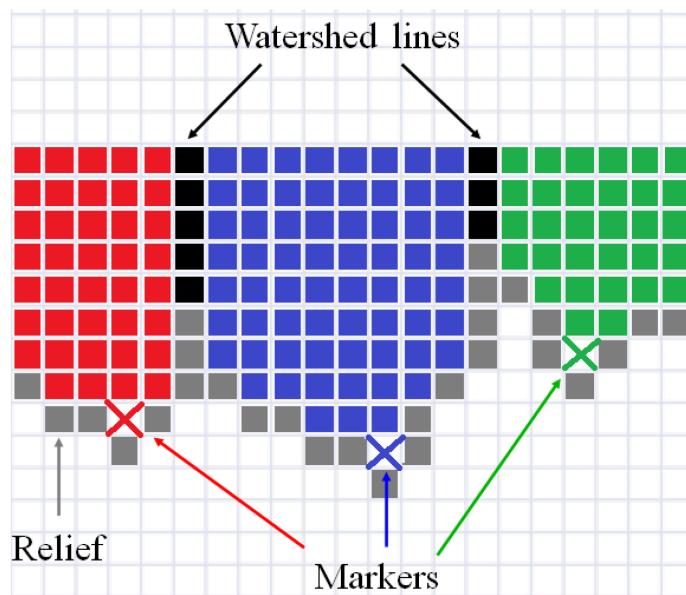


Figure 1.4 : Illustration Watershed [3]

² Annexe II

Prétraitement des images

Avant d'appliquer l'algorithme watershed, plusieurs étapes de prétraitement ont été nécessaires afin d'optimiser la segmentation :

1. Binarisation de l'image : l'image a été convertie en niveaux de gris et un seuillage a été appliqué pour distinguer les grains du fond.
2. Carte de distance : une carte de distance a été calculée à partir de l'image binarisée, en attribuant à chaque pixel une valeur correspondant à sa distance par rapport au fond.
3. Inversion de la carte de distance : pour correspondre au principe du watershed, la carte de distance a été inversée, transformant ainsi les centres des grains en minima locaux.
4. Utilisation des marqueurs : les noyaux extraits précédemment à l'aide du MinTree ont été utilisés comme marqueurs initiaux pour lancer l'algorithme.

Application de l'algorithme Watershed

L'algorithme watershed a ensuite été appliqué sur la carte de distance inverse en utilisant les marqueurs définis auparavant. Cette approche a permis d'obtenir une segmentation précise en séparant efficacement les grains adjacents.

Résultats et évaluation

Les résultats de segmentation ont été visualisés en attribuant une couleur unique à chaque grain segmenté³. Une inspection visuelle a montré que l'algorithme parvenait à distinguer correctement la majorité des grains, en particulier ceux situés au centre de l'image. Toutefois, certaines limitations ont été observées :

- Fusion de grains aux bords de l'image : certains grains situés près des bords de l'image ont tendance à être fusionnés malgré la présence de marqueurs. C'est d'autant plus vrai si on travaille sur un extrait de l'image. En effet, si le noyau d'un grain est en dehors de la section prise de l'image, alors il ne pourra pas être segmenté et sera confondu avec un autre grain⁴.
- Sensibilité au bruit et aux artefacts : la segmentation peut être influencée par les imperfections de l'image, nécessitant un prétraitement plus approfondi⁵.
- Dépendance aux marqueurs : la qualité des marqueurs initiaux joue un rôle clé dans la précision de la segmentation. Ce point implique le premier point.

³ Annexe III

⁴ Annexes IV et V

⁵ Annexe V

Améliorations possibles

Pour améliorer les résultats de la segmentation, plusieurs pistes peuvent être envisagées:

- Affinement des marqueurs : appliquer des techniques de correction pour mieux positionner les marqueurs.
- Filtrage avancé : utiliser des méthodes de correction des artefacts et un seuillage adaptatif pour améliorer la binarisation.
- Post-traitement : appliquer une érosion légère pour éviter l'inclusion de bruit dans la segmentation finale.

En conclusion, l'algorithme watershed, combiné à l'extraction de noyaux via le MinTree, a permis d'obtenir une segmentation robuste et efficace des grains.

Améliorations explorées

Afin d'améliorer le watershed précédent, nous avons tenté d'appliquer une transformation top-hat lors de la binarisation dans le but de diminuer l'impact du bruit par éclairage non-uniforme. Ce que nous appelons éclairage non-uniforme est un phénomène qui fait que le haut de l'image est plus clair que le bas de l'image par exemple. Cela pose un problème lorsqu'on essaie d'appliquer la binarisation avec un seuillage. En effet, le seuillage va supprimer soit trop en bas de l'image, soit pas assez en haut de l'image⁶. [\[4\]](#)

Nous avons donc implémenté cette transformation, cependant le temps de calcul explosait sur une petite section de l'image. C'est pourquoi nous ne sommes pas passés à l'échelle avec cet algorithme.

Néanmoins, nous ne sommes pas parvenus à comprendre pourquoi le temps d'exécution était si long et il conviendrait donc d'essayer de ré-implémenter cet algorithme. En effet, une erreur s'est peut-être glissée dans notre code.

Segmentation de grains par Successive Watershed

Afin d'améliorer la segmentation des grains, nous avons mis en place une approche innovante que nous avons appelée Successive Watershed. Cet algorithme vise à limiter les effets négatifs de l'éclairage non-uniforme et à éliminer les artefacts dus aux objets en contact avec les grains d'intérêt, comme les cylindres en haut et en bas de l'image.

⁶ Annexe VI

Principe de l'algorithme

L'idée principale du Successive Watershed repose sur une segmentation locale, appliquée sur des régions centrées autour de chaque grain détecté. Plutôt que d'exécuter le watershed sur l'image complète, ce qui entraîne des erreurs de segmentation dues aux variations d'éclairage et au contact des grains avec des objets non considérés. Nous procédons en plusieurs étapes :

1. Extraction des centroïdes : à partir des données initiales, nous identifions les centroïdes de chaque grain et les enregistrons dans un fichier CSV.
2. Découpage en sous-images (crops) : pour chaque centroid détecté, nous extrayons une sous-image centrée sur le grain d'une taille définie (qui doit être la plus faible possible pour avoir l'exécution la plus rapide, mais suffisamment grande pour éviter les effets de bords). Cette taille est choisie pour être environ deux fois plus grande que la taille maximale des grains, garantissant que le grain d'intérêt est bien isolé.
3. Application du Watershed : un watershed est appliqué uniquement sur la sous-image, en utilisant une carte des noyaux et un masque contenant les labels associés à ces grains.
4. Récupération du grain segmenté : une fois le watershed exécuté, nous ne retenons que le grain situé au centre de l'image.
5. Reconstruction de l'image segmentée : la segmentation des différents grains est réinjectée dans une image de même taille que l'originale, garantissant ainsi une cohérence globale.

Cette approche permet de réduire l'impact des artefacts liés à l'éclairage et aux objets en contact, car chaque grain est traité indépendamment dans sa propre région.

Résultats et performances

Par manque de temps et de ressources, nous n'avons pu tester le Successive Watershed que sur des images de petites tailles par rapport aux réelles données. En effet, il faut une RAM conséquente pour traiter une image de 6 Go, de plus il faut une puissance de calcul convenable. Ça reste une grande limitation de cet algorithme de segmentation. Cependant, les résultats actuels sont satisfaisants, et l'algorithme ne demande qu'à être utilisé sur l'image originale⁷.

L'approche Successive Watershed pourrait donc montrer une amélioration significative dans la segmentation des grains. En traitant chaque grain indépendamment, nous pourrions :

- Réduire les erreurs dues aux variations d'éclairage, en isolant chaque grain du contexte global.
- Supprimer les objets parasites, notamment les parties du cylindre en contact avec certains grains.
- Obtenir une segmentation plus fine, en concentrant l'algorithme sur des zones restreintes.

Cependant, cette méthode peut être légèrement plus coûteuse en temps de calcul, car le watershed est exécuté plusieurs fois sur des sous-images au lieu d'une seule fois sur l'image complète. Une optimisation pourrait être envisagée pour améliorer l'efficacité du processus. En

⁷ Annexes IV et V

effet, l'algorithme est facilement parallélisable au niveau de la boucle for qui applique un watershed dans le secteur de chaque centroid⁸.

2) Suppression des parties indésirables

Les images que nous avons reçues pour le projet sont des images brutes sans traitement préalable. De ce fait, certaines parties ne correspondant pas aux grains, telles que les plateaux et les membranes les encerclant, sont considérées indésirables.

a) Suppression des membranes latérales

Afin de pouvoir se déplacer librement tout en étant retenus, les grains sont entourés d'une fine membrane plastique pouvant se contorsionner de façon imprévisible⁹. Afin de la faire disparaître en affectant les grains un minimum, nous avons choisi d'opérer des transformations morphologiques telles qu'une ouverture ou une fermeture. Étant donné la nature fine et filaire des membranes, de telles opérations accompagnées d'un élément structurant rectangulaire suffisamment petit permettent de ne pas affecter l'essentiel des grains. Elles font cependant disparaître les noyaux des grains quelle que soit la taille de l'élément structurant, et il est donc nécessaire de traiter ceux-ci avant de supprimer les membranes.

Nous avons effectué ces opérations grâce à l'outil de visualisation d'images 3D Fiji présenté plus tôt, et en particulier grâce au plugin MorphoLibJ. [\[5\]](#)

Après application, nous avons remarqué que les membranes ne pouvaient disparaître entièrement sans causer des dégâts irréversibles sur les grains¹⁰. Pour éviter cela, nous avons déterminé qu'un élément structurant de 3 pixels de large pour 1 pixel de haut (puis un élément similaire vertical) suivi d'un seuil en fonction du niveau de gris rendait un des meilleurs résultats possibles.

Les images sont codées en niveau de gris en 16 bits, et chaque pixel possède donc une valeur allant de 0 à $65535 = 2^{16} - 1$, de noir à blanc respectivement. De façon générale, c'est-à-dire pour les suppressions présentées dans ce rapport, un seuil à 2^{15} semble donner les meilleurs résultats¹¹.

b) Suppression des plateaux

Les grains sont posés sur un plateau épais horizontal immobile et sont compressés par un second plateau placé au-dessus de ces premiers. Ces deux plateaux sont de grands cylindres blancs verticaux qui se distinguent à l'œil facilement des grains. Il n'existe cependant pas de solution explicite pour les détecter informatiquement. Afin de les retirer des images, nous avons mis en place plusieurs solutions.

b.1) Effacement direct sur les images

Dans un premier temps, nous avons traité les images directement, étant donné que nous connaissons les positions approximatives et la forme des plateaux. Avec cette méthode, nous traitons les deux cylindres séparément. L'image 3D étant facilement décomposable en une

⁸ Annexe VI

⁹ Annexe VII

¹⁰ Annexe VII

¹¹ Annexe VII

suite d'images 2D représentant les couches de profondeur, nous traitons ici chacune des images 2D séparément. La difficulté de cette méthode est que chaque image ne possède pas nécessairement de plateaux, ou seulement une petite partie de ceux-ci.

Le cylindre inférieur est relativement simple à traiter car il est toujours posé au sol et n'a donc pas d'angle particulier. Pour pallier la difficulté précédente, nous parcourons horizontalement chaque image à la hauteur où le cylindre inférieur devrait se trouver jusqu'à trouver un pixel blanc (c'est-à-dire un pixel dont la valeur est supérieure au seuil défini plus tôt). Si aucune pixel blanc n'est trouvé, alors le cylindre inférieur est considéré comme non présent sur cette couche de profondeur. Si un pixel est trouvé, alors nous parcourons verticalement pour déterminer la hauteur du cylindre.

Afin de ne pas surcharger les calculs, les cylindres sont considérés ici comme des parallélogrammes sur chacune des couches. Une fois la hauteur et les extrémités trouvées, il est donc simple de recolorer l'ensemble des pixels contenus dans ce parallélogramme en noir.

Le cylindre supérieur est plus difficile à détecter, car il est potentiellement à n'importe quelle hauteur, et peut être à un certain angle qui rend plus difficile la détection des extrémités horizontalement. Sa détection est plus robuste et ne sera pas davantage détaillée dans ce rapport.

Étant donné la solution plus simple trouvée ultérieurement et les défauts de la solution actuelle, nous n'avons d'ailleurs pas eu le temps d'implémenter entièrement cette méthode. Un exemple de résultat est disponible en annexe. Nous pouvons remarquer que les coefficients directeurs ne sont pas toujours parfaits et certains grains sont parfois partiellement obfusqués¹².

b.2) Assistance par watershed

Afin notamment de détecter plus facilement les contours des cylindres, notre nouvelle méthode est d'utiliser le watershed successif présenté précédemment. Chaque noyau de grain possède un label précis, et le watershed successif se concentre tour à tour sur chacun des grains en découpant une partie de l'image centrée sur ce grain. Cette découpe est suffisamment large pour que n'importe quel grain soit compris entièrement à l'intérieur, mais ce ne serait pas le cas d'un cylindre. Si le "grain" dépasse du bord de l'image, il s'agit donc d'un des cylindres.

Dans ce cas, le label de ce noyau est modifié en un label spécial correspondant aux cylindres, et la partie du cylindre incluse dans la coupe est effacée. Le reste de l'effacement n'est pas essentiel, car il est par définition suffisamment loin de tout grain. Cette méthode assure qu'aucun grain n'est juxtaposé à un cylindre, ce qui facilite grandement sa différenciation des parties importantes de l'image. Ce label spécial peut ensuite être utilisé par exemple lors d'une visualisation 3D des images afin retirer ou afficher facilement tous les points originellement enregistrés ne représentant pas des grains¹³.

¹² Annexe VIII

¹³ Partie 4 : [Visualisation 3D interactive](#)

À titre d'exemple, l'une des annexes¹⁴ montre une utilisation de cet algorithme afin d'effacer un plateau ajouté artificiellement. Contrairement à précédemment, celui-ci est trouvé sans avoir besoin de parcourir l'image grâce au label qui lui a été fourni. Il suffit alors de recouvrir la ou les région(s) délimitée(s) par watershed possédant ce label.

Veuillez noter que cet exemple est réalisé sur un extrait de l'image à cause de la taille conséquente (près de 6 Go pour une image entière) de l'originale, d'où l'utilisation de parties réduites et l'ajout artificiel d'un plateau. Bien qu'il eût été préférable de prendre cet exemple sur un plateau réel de l'image, extraire une de ces parties demanderait d'avoir l'image complète en mémoire, ce qui n'était pas possible. Nous avons donc dû nous contenter d'un extrait qui nous était déjà disponible et qui se trouvait à l'écart des plateaux.

3) Détection des contacts

Un des principaux enjeux du projet est de détecter les contacts entre les grains et si possible la force de ces contacts. Cette étape est donc dépendante de la segmentation et la labellisation des grains. Plusieurs méthodes de détection de contacts ont été implémentées, d'une part afin d'essayer de réduire le temps d'exécution, et d'autre part afin d'avoir des méthodes viables selon les données disponibles. Ne sachant pas à l'avance quels résultats il serait possible d'obtenir pour la labellisation des grains, différentes stratégies ont été envisagées : certaines méthodes s'appuient sur une labellisation des grains entiers, tandis que d'autres se basent uniquement sur la labellisation des noyaux. Cette différence influence directement la manière dont les contacts entre grains sont identifiés.

a) Méthode naïve

La première méthode implantée fut une méthode naïve, reposant sur une labellisation complète des grains. Cette dernière étant simple à implémenter, elle nous a permis d'obtenir des résultats dignes de confiance, servant de référence afin d'évaluer les résultats des autres méthodes.

Cette méthode consiste à parcourir l'image entière et, pour chaque voxel labellisé, à examiner ses voisins afin de déterminer si un autre label est touché. De cette manière, aucun contact ne peut être oublié ni détecté de manière erronée.

Les résultats obtenus sont fiables, mais le temps de calcul est particulièrement long, car la recherche de contacts prend en compte l'intégralité des voxels labellisés.

b) Utilisation d'un squelette

La seconde méthode implantée, reposant elle aussi sur une labellisation complète des grains, consiste à squelettiser¹⁵ [6] l'image des grains grâce à la bibliothèque Pink [7]. Cela permet d'accélérer la recherche de contacts en utilisant le squelette obtenu. Ce dernier sert simplement à accélérer le parcours de l'image, car seuls les voxels du squelette sont pris en compte pour détecter les contacts.

En effet, l'opération de squelettisation ne modifie pas la topologie de l'image, et le programme de squelettisation prend en argument les voxels que nous voulons garder dans le squelette. Dans notre cas, ces voxels sont ceux faisant partie des noyaux des grains, détectés

¹⁴ Annexe IX

¹⁵ Annexe X

grâce au minTree. Ces deux arguments nous permettent donc de penser que ce squelette garde en lui tous les contacts.

En pratique, quelques contacts ne sont plus détectés en utilisant ce squelette, étant trop faibles pour être gardés lors de la squelettisation. La détection de contacts se faisant toujours en comparant les labels d'un voxel et de ses voisins, l'introduction de faux contacts n'est donc pas possible pour cette méthode. Le temps de calcul est significativement plus court qu'avec la méthode naïve, au prix d'une légère dégradation des résultats.

c) Utilisation des noyaux labellisés

La troisième et dernière méthode utilisée repose uniquement sur la labellisation des noyaux des grains. Cette méthode utilise encore une fois le squelette de l'image des grains afin de détecter les contacts. Les données d'entrée sont l'image des grains, les noyaux détectés grâce au minTree ainsi qu'un voxel labellisé par noyau de grain.

Dans un premier temps, les labels sont diffusés aux noyaux entiers. Ensuite, les labels sont diffusés dans le squelette jusqu'à se rencontrer. Le squelette étant composé en théorie des noyaux des grains ainsi que d'arêtes reliant les noyaux des grains en contact. L'idée est donc que les labels se rejoignant sont censés être ceux étant en contact.

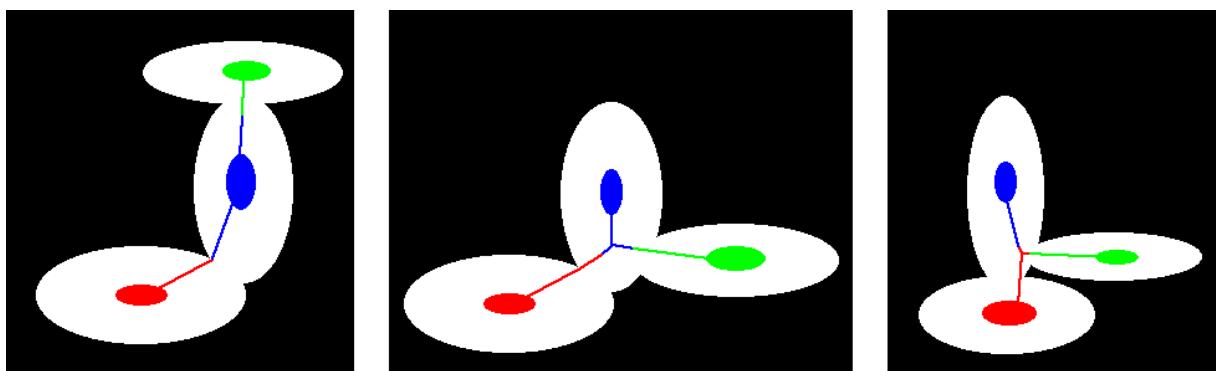


Figure 3.1 – Résultats possibles de diffusion dans le squelette

Comme illustré sur la figure ci-dessus, plusieurs cas sont possibles lors de la diffusion des labels : si une arête du squelette ne relie que deux labels sans fusionner avec une autre (grande majorité du temps), les deux labels concernés se rencontrent et le contact est détecté. Dans le deuxième cas, les deux arêtes fusionnent avant d'arriver au noyau bleu. Cependant, le label bleu est le plus proche de la jonction, il touche donc les deux autres labels comme prévu et les labels rouge et vert ne se touchent pas, comme prévu. Dans le dernier cas, le noyau le plus proche de la jonction n'est pas celui réellement en contact avec les deux autres. Le label se diffusant le premier jusqu'à la jonction (ici le rouge) n'est donc pas le bon, un faux contact va donc être détecté (rouge-vert) et un contact réel ne le sera pas (bleu-vert).

d) Calcul des forces de contact

Nous définissons la force d'un contact par le nombre d'érosions nécessaires pour le supprimer. Pour calculer cela, il faut donc itérativement éroder l'image des grains, puis appliquer une des méthodes présentées ci-dessus pour détecter les contacts encore présents. Ceci est itéré jusqu'à ce qu'il n'y ai plus aucun contact.

La méthode naïve présente le même temps de calcul à chaque itération, tandis que la seconde méthode, celle utilisant la squelettisation et les noyaux entièrement labellisés, présente

un avantage : le squelette est calculé une seule fois, et permet d'accélérer chaque itération de détection de contacts. Cependant, les grains étant des ellipsoïdes et non des sphères, la zone de contact entre deux grains ne se trouve pas obligatoirement entre les deux noyaux.

Cela implique que le squelette ne passe pas toujours par le centre de la zone de contact, ce qui peut entraîner une rupture du contact au niveau du squelette après un nombre d'érosions inférieur à celui attendu.

4) Visualisation 3D interactive

Dans cette partie du rapport, nous allons analyser comment nous avons construit la visualisation 3D des grains avec une composante temporelle. Premièrement, nous allons vous expliquer comment les différents paramètres nécessaires ont été analysés, par la suite comment l'affichage global a été géré et les possibilités que possède l'utilisateur pour modifier certains paramètres visuelles. De plus, d'autres prototypes seront mentionnés.

a) Lecture des composantes et outils nécessaires à l'affichage

En premier lieu, il est important d'avant afficher des données, d'obtenir ces différentes données lisibles par les programmes. Pour cela, nous nous sommes inspirés des données de la thèse du géomécanicien Pinzón Gustavo [\[1\]](#) pour concevoir les différents paramètres.

En premier lieu, nous avons utilisé un fichier des coordonnées de points¹⁶ pour chaque frame de la vidéo. Ce fichier texte contient pour chaque ligne, le label du grain, ces coordonnées en XYZ.

En deuxième lieu, nous avons utilisé un fichier des points de contacts des grains¹⁷ pour chaque frame de la vidéo. Ce fichier texte contient pour chaque ligne, le label du grain A et le label du grain B en contact.

Par la suite, via du code Python, ces fichiers sont traités pour obtenir une structure compréhensible par l'ordinateur pour les futurs traitements. Principalement une programmation orientée objet (l'objet étant le grain) pour une meilleure implémentation dans le code.

Enfin, pour afficher ces différentes composantes, il est nécessaire de trouver une bibliothèque qui pourra nous être utile. En effet, au lieu de perdre des ressources à créer notre propre logiciel d'affichage de graphiques, il existe déjà différents outils open-source qu'on peut importer à notre projet. Celui que nous avons choisi est Polyscope [\[8\]](#) en Python car :

- Demande très peu de puissance de calcul
- S'implémente facilement
- Permet d'ajouter des éléments personnalisables
- Permet de modification en temps réel du visuel

Nous avons testé d'autres outils comme Plotly 3D [\[9\]](#) mais il était bien trop gourmand en calcul pour notre contexte. En effet, l'objectif est de pouvoir afficher un graphe avec plus de 20 000 grains et plusieurs dizaines de milliers d'arêtes avec comme contrainte un temps de calcul des paramètres raisonnable et un taux de rafraîchissement supérieur à 30 images par secondes lors des déplacements de l'utilisateur au sein du graphique 3D généré.

¹⁶ Annexe XI

¹⁷ Annexe XII

b) Affichage évolution d'un ou plusieurs grains dans le temps

Avec ces différents paramètres en main, nous pouvons maintenant effectuer les différents calculs qui vont être utilisés à l'affichage dans Polyscope. Différents scripts Python ont été créés pour effectuer différents affichages. En premier lieu, le programme *polyscope_follow_grains_global* a été créé, celui-ci a comme objectif d'afficher de manière globale tous les grains au sein d'un fichier des coordonnées et de ses points de contacts. Tout cela avec une évolution dans le temps. Ce programme se lance en prenant le dossier avec les coordonnées de grains et ses contacts pour chaque image de la vidéo.

Après un traitement de quelques minutes, une nouvelle fenêtre s'ouvre¹⁸. Celle-ci contient plusieurs informations clés pour l'utilisateur. Sur l'écran principal, vous trouverez les différents grains répartis dans l'environnement 3D, le déplacement se fait principalement via la souris.

En premier lieu, sur la partie de gauche, toutes les informations nécessaires aux noeuds et aux arêtes de connexions :

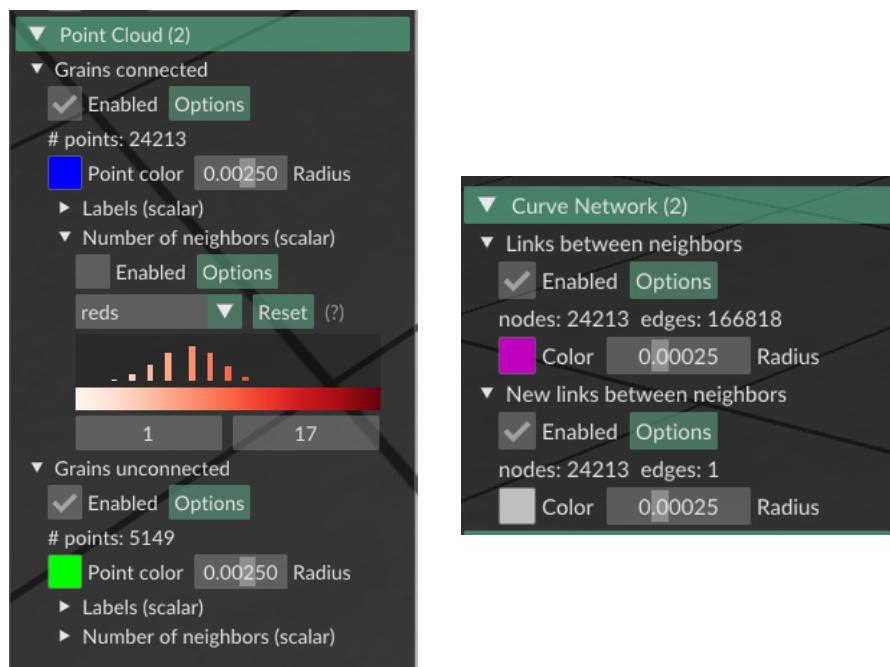


Figure 4.1 – Fenêtre de paramètres à gauche, gestion des noeuds et des arêtes.

Comme vous pouvez le voir sur la figure ci-dessus. Nous avons différents curseurs qui sont possibles d'être modifiés en temps réel par l'utilisateur. Dans la partie des noeuds correspondant aux grains, il y a deux parties distinctes : les grains connectés et les grains non-connectés. Cela signifie tout simplement les grains qui ont au moins un point de contact et les grains qui n'en ont pas. L'utilisateur peut cacher ou non une de ces catégories. Il peut modifier à ses envies les couleurs ou encore la taille des grains. Il a aussi la possibilité dans la section "Number of neighbors" d'activer une heatmap du nombre de points de contacts pour chaque grains¹⁹.

¹⁸ Annexe XIII

¹⁹ Annexe XIV

De plus, il y a la partie des arêtes, celle-ci est aussi divisée en deux parties : les arêtes présentes sur la frame précédente et actuelle puis les nouvelles arêtes présentes sur la frame actuelle²⁰. Comme avec les noeuds, les couleurs, l'épaisseur, etc. peut être modifié par l'utilisateur en temps réel.

Dans différents paragraphes, nous avons mentionné le terme "frame". Celui-ci correspond à une image de la vidéo. Pour gérer le paramètre temps sur Polyscope, une fenêtre personnalisée a été conçue :

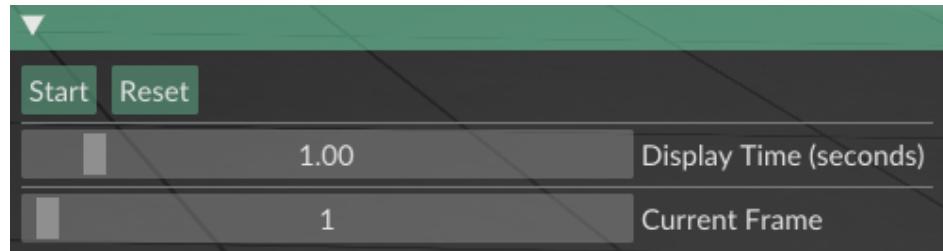


Figure 4.2 – Fenêtre de contrôle du temps de l'évolution du cylindre et de ses grains.

Celle-ci possède quatre actions qui peuvent être manipulées par l'utilisateur :

- Le bouton start : pour lancer l'animation.
- Le bouton reset : pour revenir à l'image 0 de la vidéo.
- Un curseur "Display time" : pour définir le temps que reste une frame à l'écran avant de passer à la suivante. Cela est défini en secondes.
- Un curseur "Current Frame" : pour définir la frame actuellement visible. L'utilisateur, si besoin, peut se déplacer à l'image qu'il souhaite.

En raison du support papier, il est difficile pour nous de vous afficher l'évolution fluide que cela peut faire. Par exemple, l'utilisateur peut aller de la frame 3 à 24 puis finalement à la frame 51²¹.

Lorsque l'utilisateur clique sur grain, différentes informations sont affichées :



Figure 4.3 – Fenêtre d'information d'un grain après avoir cliqué dessus.

Nous retrouvons différentes données clés : ces coordonnées en XYZ, le label du grain et le nombre de voisins qu'il possède.

²⁰Annexe XV

²¹Annexes XVI, XVII et XVIII

En nous basant sur le programme que nous avons construit, nous avons décidé d'en créer un second qui permet de choisir les labels que nous voulons suivre. Ce programme, s'appelant *polyscope_follow_grains_with_labels*, prend en paramètre le dossier avec les coordonnées de grains et ses contacts pour chaque image de la vidéo, puis dernièrement les numéros des labels que nous voulons sélectionner. Avec globalement les mêmes paramètres modifiables que le précédent programme, celui-ci se concentre plus précisément sur certains grains choisis par l'utilisateur²².

De plus, un autre programme a été créé : *polyscope_compare_grains*. Un prototype sur la possibilité de visualiser deux cylindres de grains en même temps et de comparer leurs différences²³. La logique est simple : nous prenons deux graphes représentant chacun un nuage de grains. Si un grain du graphe 1 possède les mêmes coordonnées qu'un grain du graphe 2, alors on le catégorise d'une certaines couleurs (ici verte), si le grain est unique au graphe 1, il possède alors une autre couleur, de même dans le cas où le grain est unique au graphe 2. Cela permet par exemple, de comparer l'évolution des grains entre deux simulations différentes.

Nous avons également observé, à partir des données fournies par le géomécanicien Gustavo PINZON FORERO, certains résultats visuels inhabituels. Par exemple, le grain présentant la plus grande connexité atteint une valeur de 17, ce qui semble peu probable dans un cas réel. De plus, en affichant les arêtes courantes et nouvelles, nous avons remarqué que, dans certaines frames, les résultats du géomécanicien montrent des connexions nouvellement délimitées de manière anormalement précise²⁴. Cela pourrait être lié au choix de l'algorithme utilisé pour détecter les points de contact et mesurer ces différentes connexions.

Pour conclure cette partie sur l'affichage 3D sur Polyscope au fil du temps. La bibliothèque est très enrichissante mais possède quelques lacunes principalement dans sa liberté de conception. Dans sa version Python, comme elle met en avant l'accessibilité, la bibliothèque a dû sacrifier en retour sa complexité. Par exemple, il n'est pas possible après lancement de polyscope de supprimer un nuage de points, nous pouvons seulement la cacher avec l'option "enabled". La problématique est que cette option est aussi manipulable par l'utilisateur, c'est-à-dire que l'utilisateur peut facilement afficher des choses que nous voulions pas montrer à l'origine. C'est pour cela que nous aurions dû prendre la version C++ de Polyscope qui est bien plus libre et avec une communauté plus active. Pour autant, la version Python nous a permis rapidement d'obtenir des résultats intéressants sans partir dans une structure trop complexe à notre niveau.

²² Annexe XIX

²³ Annexe XX

²⁴ Annexe XIX

III. Regard critique et perspectives d'amélioration

Dans cette troisième et dernière partie de ce rapport, nous allons effectuer un regard critique sur nos actions passées et ainsi concevoir une ou plusieurs perspectives d'amélioration.

1) Organisation au sein du groupe et du projet

Dans l'ensemble, l'organisation du projet a été satisfaisante durant tout le projet. Nous avons réussi à lier à la fois, la vie scolaire (études, examens, activités scolaires, etc.) et le projet de deuxième année. En général, les retours de notre client ont été satisfaisants.

Pour autant, cela peut s'améliorer, par exemple, en essayant de pousser la méthodologie agile encore plus loin. Nos différentes cartes sur Trello manquaient une composante : un niveau de complexité/difficulté, ce niveau correspondant soit au temps nécessaire estimé pour effectuer la tâche, soit au nombre de lignes de code nécessaires. De plus, ces cartes pourront avoir une description avec une base commune : "En tant qu'utilisateur..." .

Dernièrement, notre projet manque de stabilité dans le sens où très peu de tests unitaires ont été construits. Pour gagner en robustesse, il aurait été intéressant de développer une plus grande couverture de tests et au mieux appliquer un TDD²⁵. La méthode TDD a comme objectif de concevoir la méthode de test de la fonctionnalité avant même de développer la fonctionnalité. Ce qui permet d'avoir un contrôle total de la fonction tout en ayant un minimum de recherche documentaire en amont.

2) Futures fonctionnalités

Par manque de temps, nous n'avons pas pu implémenter certaines fonctionnalités. Du côté de la segmentation, plusieurs pistes sont à travailler :

- Réessayer un algorithme à base de top-hat ou alors améliorer la binarisation avec de nouvelles méthodes
- Tester le Successive Watershed sur les images originales qui sont bien plus grosses
- Paralléliser l'exécution du Successive Watershed

Dans la partie détection de contacts :

- Parallélisation de la squelettisation
- Parallélisation de la diffusion de labels pour la dernière méthode

Dans la partie visualisation 3D interactive, en raison du manque de temps, nous n'avons pas pu implémenter une manière plus confortable de filtrer les labels à afficher. Actuellement, il est obligatoire de les écrire à la main via le terminal, ce qui est assez contraignant. C'est pour cela qu'il aurait été plus intéressant de construire un fichier txt ou csv contenant tous les labels auxquels l'utilisateur veut suivre. Ce fichier est par la suite envoyé au programme qui va afficher sur polyscope les différents grains.

²⁵ Test Driven Development

Conclusion

Pour conclure, ce projet a été enrichissant pour nous tous. Il a permis d'améliorer notre gestion de projet en appliquant la méthode agile dans un environnement pratique avec, par exemple, l'utilisation de sprints d'un mois et la coopération du client. Cela nous a permis de comprendre que c'est une méthode efficace pour notre situation : groupe de quatre personnes, projet étudiant de quelques mois. Cependant, nous sommes conscients que cette organisation a été perfectible. Par exemple, nous n'avons pas utilisé un système de niveau de difficulté pour les différentes cartes que nous avons assignées. De plus, ces cartes manquaient de standardisation dans leur description : "En tant qu'utilisateur, je..." .

Après avoir défini une organisation cohérente pour notre gestion de projet, nous avons pu réfléchir sereinement aux différents objectifs de notre projet. L'analyse topologique des mouvements de grains dans une séquence d'images 3D nous a permis d'explorer des méthodes avancées de segmentation, de suppression d'éléments indésirables et de détection des contacts entre particules. Nous avons également développé une visualisation 3D interactive, essentielle pour interpréter nos résultats et faciliter leur exploitation future.

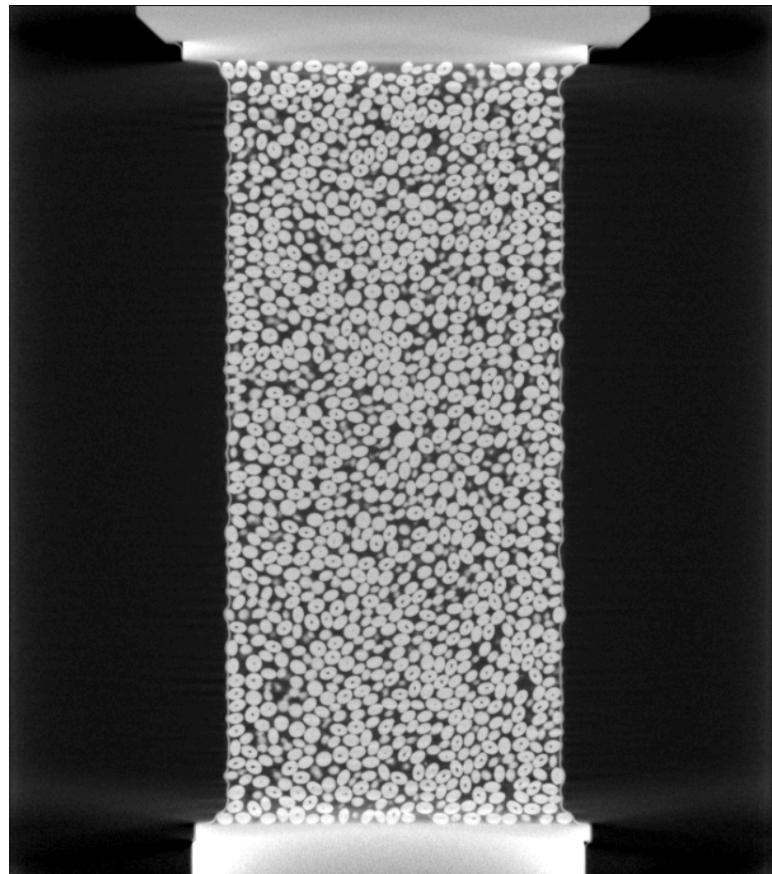
Grâce à une approche agile et à une répartition efficace des tâches, nous avons su répondre aux attentes initiales tout en proposant des améliorations et en anticipant certaines limitations. Toutefois, des perspectives d'amélioration subsistent, notamment en ce qui concerne l'optimisation des algorithmes de segmentation et de détection des contacts. L'application d'une méthode plus robuste de binarisation, le test du Successive Watershed sur des images de grande taille ou encore la parallélisation de certaines étapes du traitement pourraient permettre d'affiner nos résultats et de réduire les temps de calcul.

Ce projet nous a permis de consolider nos compétences en traitement d'images et en développement, tout en nous confrontant aux défis liés à l'analyse de données complexes. Il constitue un complément solide à la base déjà concrète de Lysandre MACKE pour d'éventuels travaux futurs et pourra être enrichi par des contributions ultérieures.

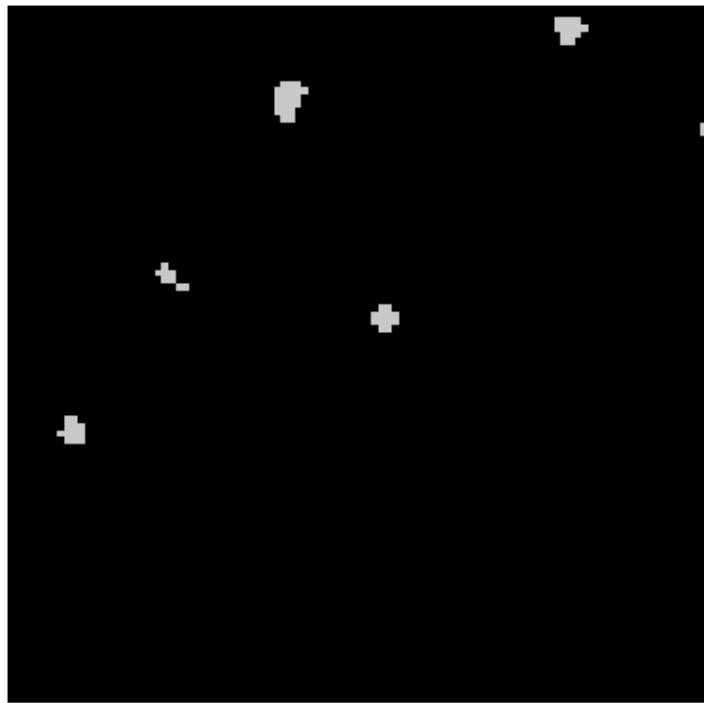
Références

[1]	Thèse "Experimental investigation of the effects of particle shape and friction on the mechanics of granular media" contenant différents paramètres sur les grains dans une vidéo : https://zenodo.org/records/8014905
[2]	Article Component Tree M. WILKINSON : https://www.semanticscholar.org/paper/A-fast-component-tree-algorithm-for-high-images-and-Wilkinson/d2b62a552a9fec0b8a2663d87fe78b026dcf69d7?utm_source=direct_link
[3]	Marker-Controlled Watershed for Segmentation of Images Encyclopedia MDPI : https://encyclopedia.pub/entry/24796
[4]	Algorithme Watershed : Ligne de partage des eaux (segmentation) – Wikipédia
[5]	Plugin MorphoLibJ pour ImageJ et Fiji : https://imagej.net/plugins/morpholibj
[6]	Michel Couprie, Gilles Bertrand. Discrete Topological Transformations for Image Processing. Brimkov, Valentin E. and Barneva, Reneta P. Digital Geometry Algorithms, 2, Springer, pp.73-107, 2012, Lecture Notes in Computational Vision and Biomechanics, 978-94-007-4174-4. ff10.1007/978-94-007-4174-4_3ff. ffhal-00727377 https://hal.science/hal-00727377
[7]	Bibliothèque Pink : https://perso.esiee.fr/~coupriem/Pink/doc/html/
[8]	Bibliothèque Python Polyscope : https://polyscope.run/py/
[9]	Bibliothèque Plotly 3D : https://plotly.com/python/3d-charts/

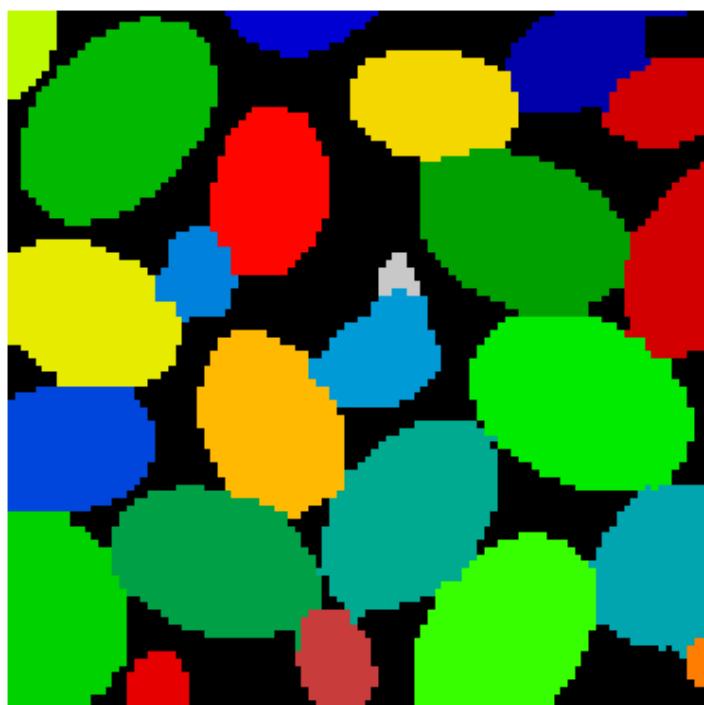
Annexes



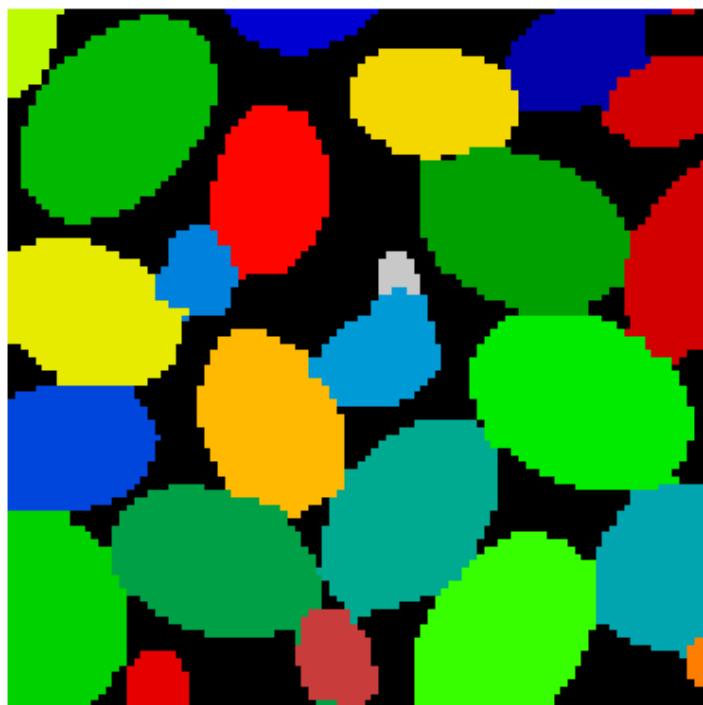
Annexe I - Exemple d'image de découpe d'une image 3D du cylindre



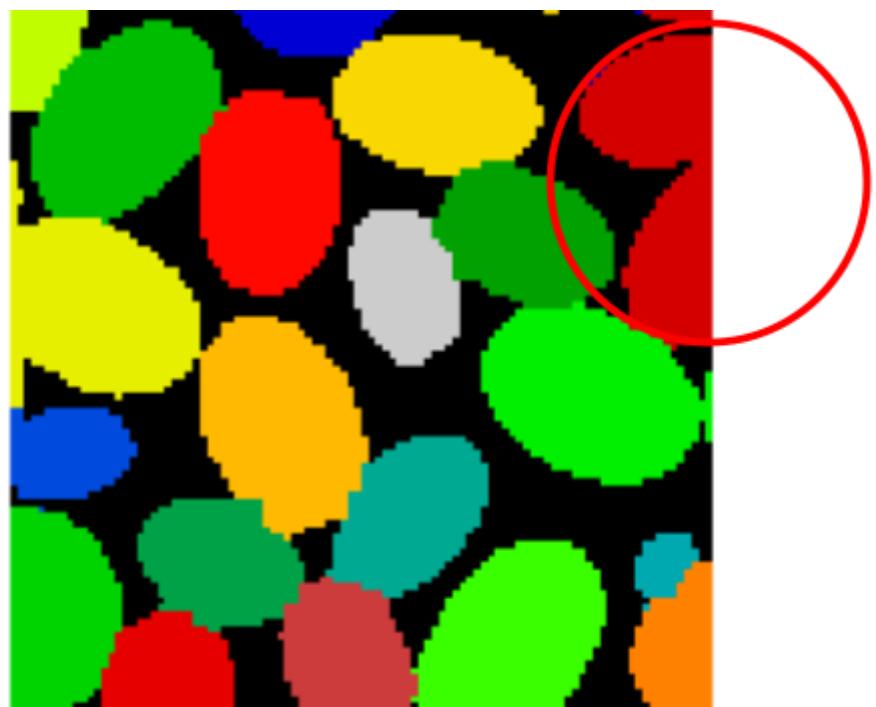
Annexe II – Résultat du MinTree sur un crop



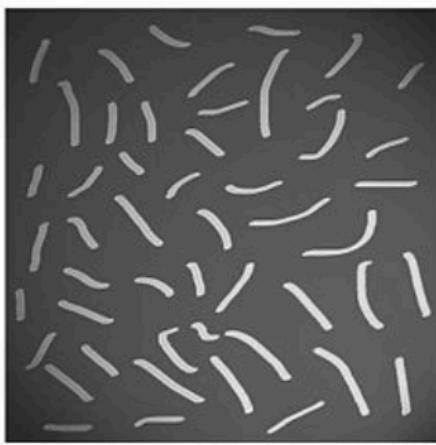
Annexe III – Résultat du Watershed sur un crop



Annexe IV - Résultat du Successive Watershed sur un crop



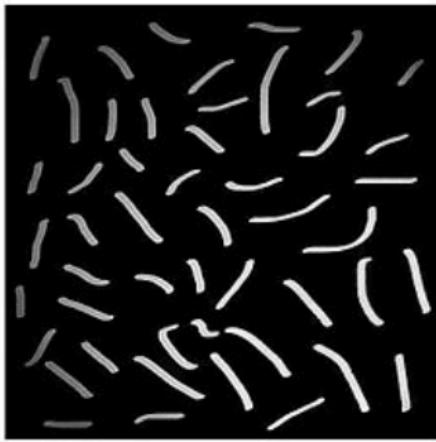
Annexe V - Grain qui se propage dans un autre à cause des effets de bord



Nonuniform lighting condition input image



Threshold a nonuniform lighting input image

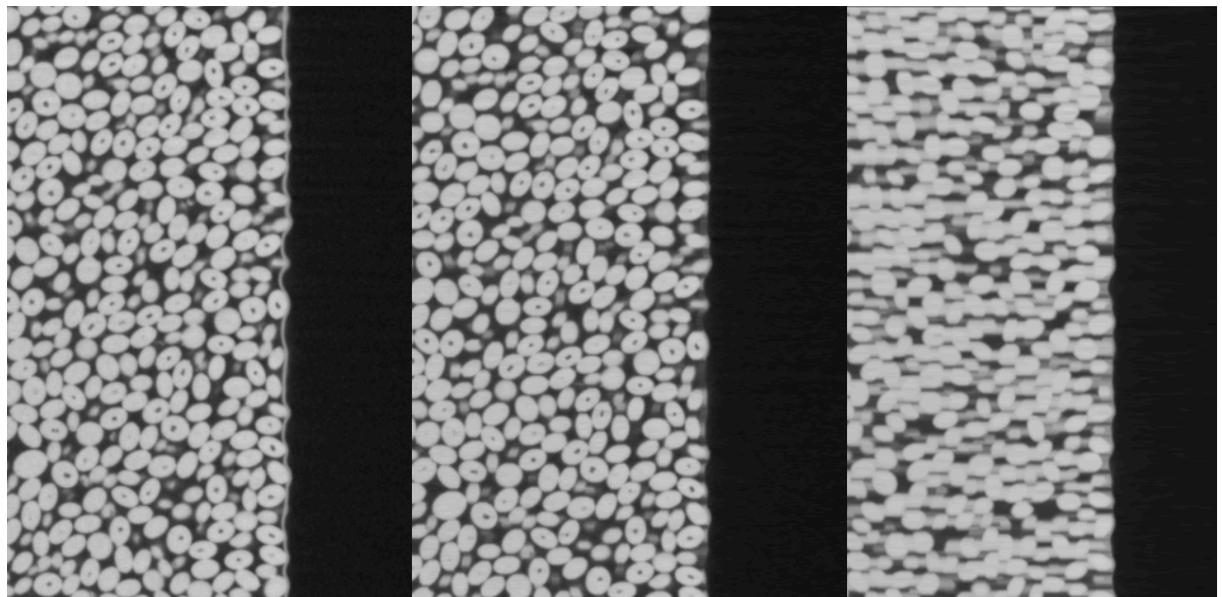


Top-hat transform applied to input image



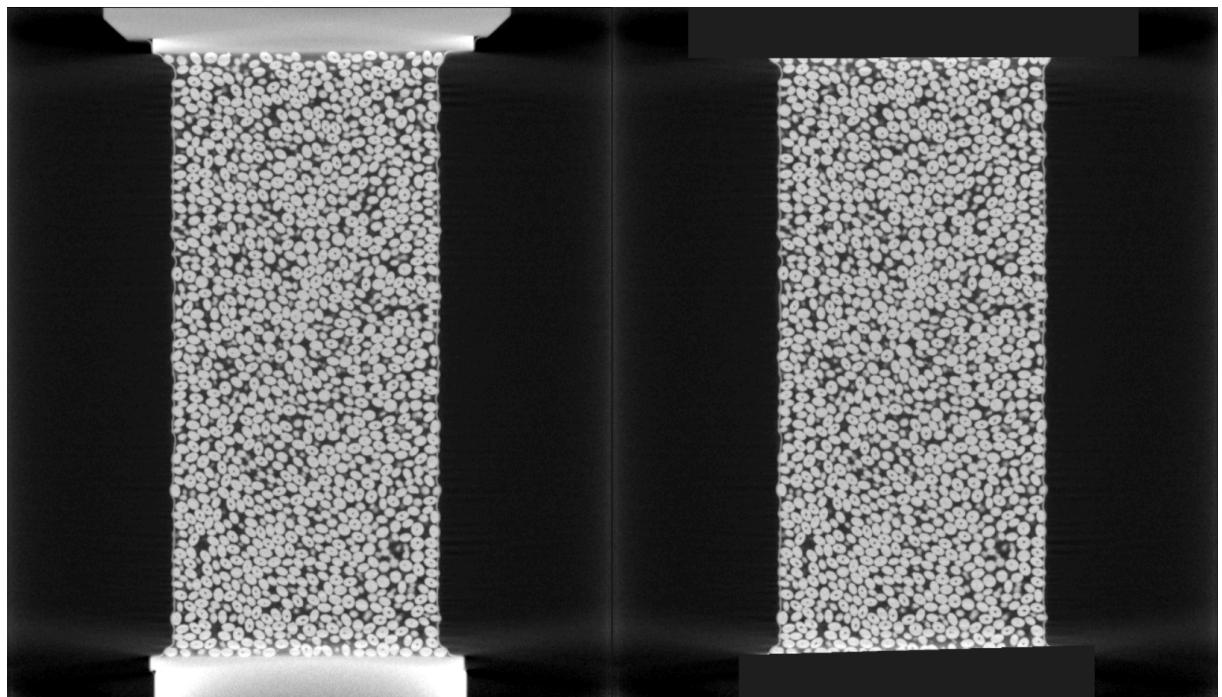
Threshold image after top-hat transform applied

Annexe VI - Top-hat Transform [\[4\]](#)

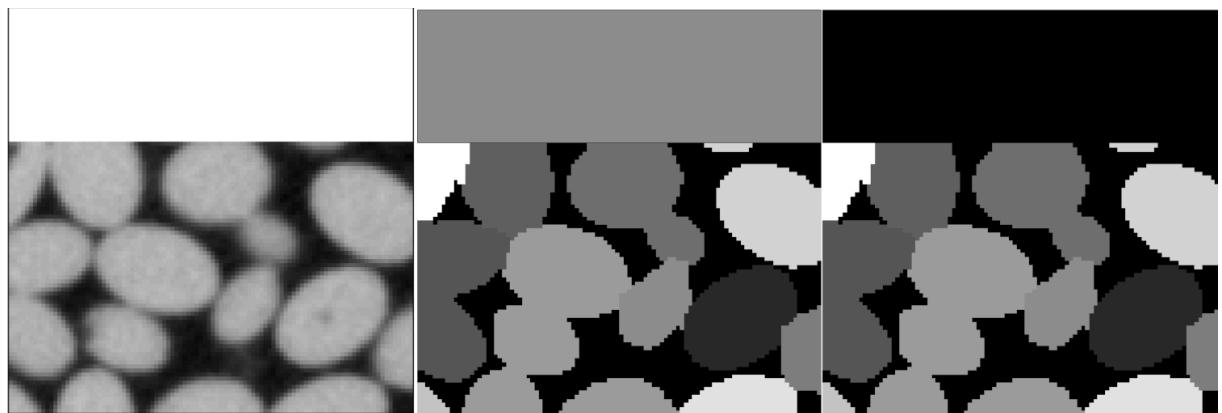


Annexe VII - Effacement des membranes latérales

Respectivement : image originale / élément structurant adapté / élément structurant trop large

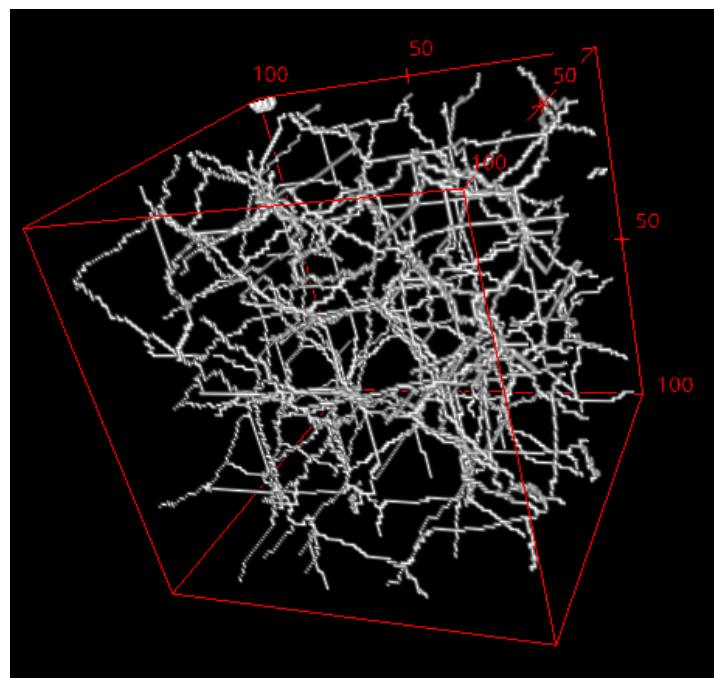


Annexe VIII - Effacement direct des plateaux sur une couche de profondeur



Annexe IX - Exemple d'effacement par assistance watershed

Respectivement : extrait de l'image originale avec plateau ajouté / watershed appliqué / plateau trouvé et retiré à partir de son label spécial



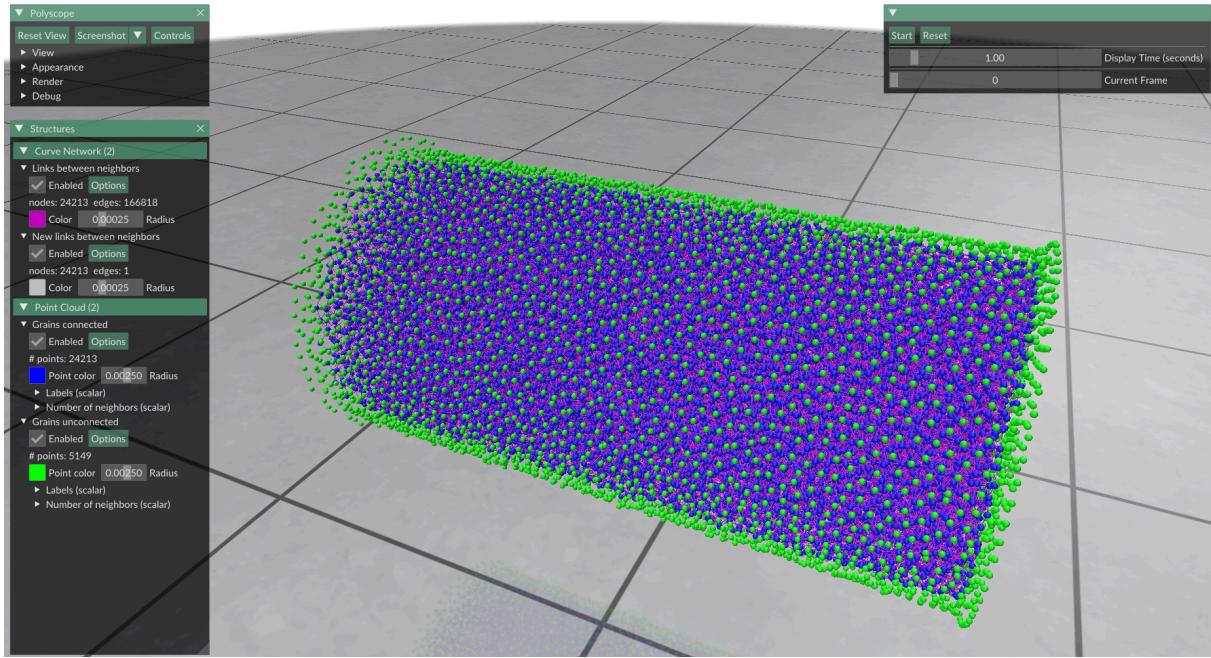
Annexe X - Squelette 3d utilisé pour la détection de contacts.

Label	Zpos	Ypos	Xpos
0.0000000	0.0000000	0.0000000	0.0000000
1.0000000	106.1302872	677.3010254	430.3721619
2.0000000	106.4360046	683.3963623	522.8596191
3.0000000	112.3213120	413.4475708	534.6060791

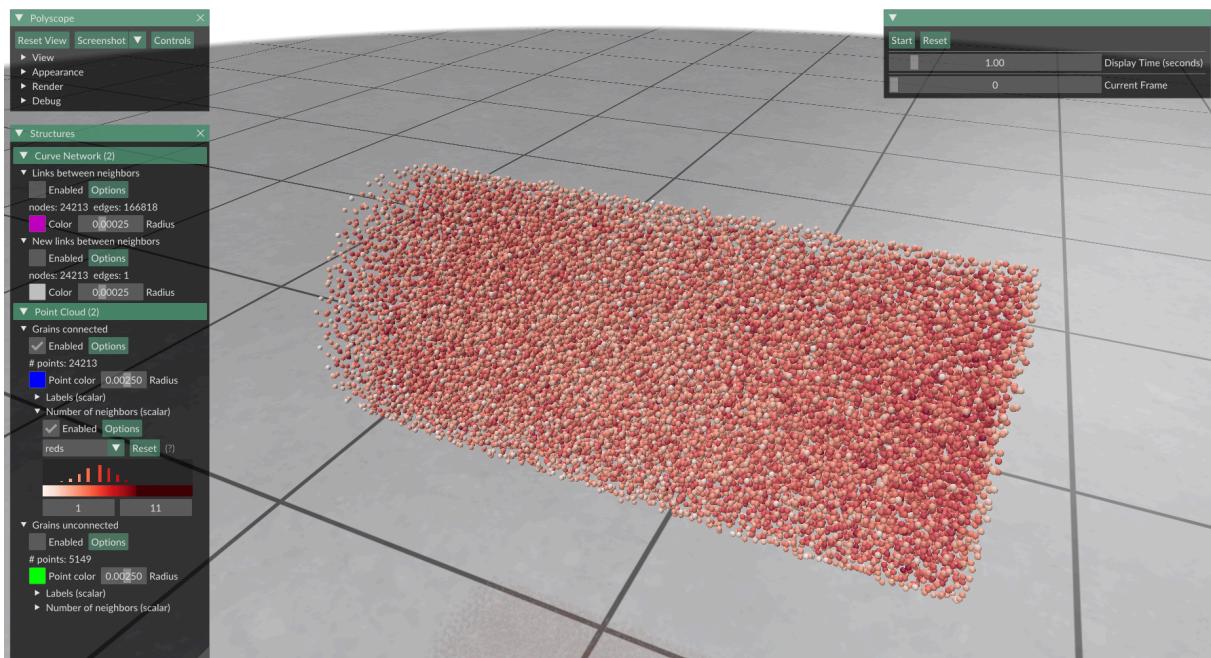
Annexe XI – Exemple fichier des coordonnées de centroids des grains

Label grain A	Label grain B
2788	2804
2784	2787
2786	2800
2777	2786

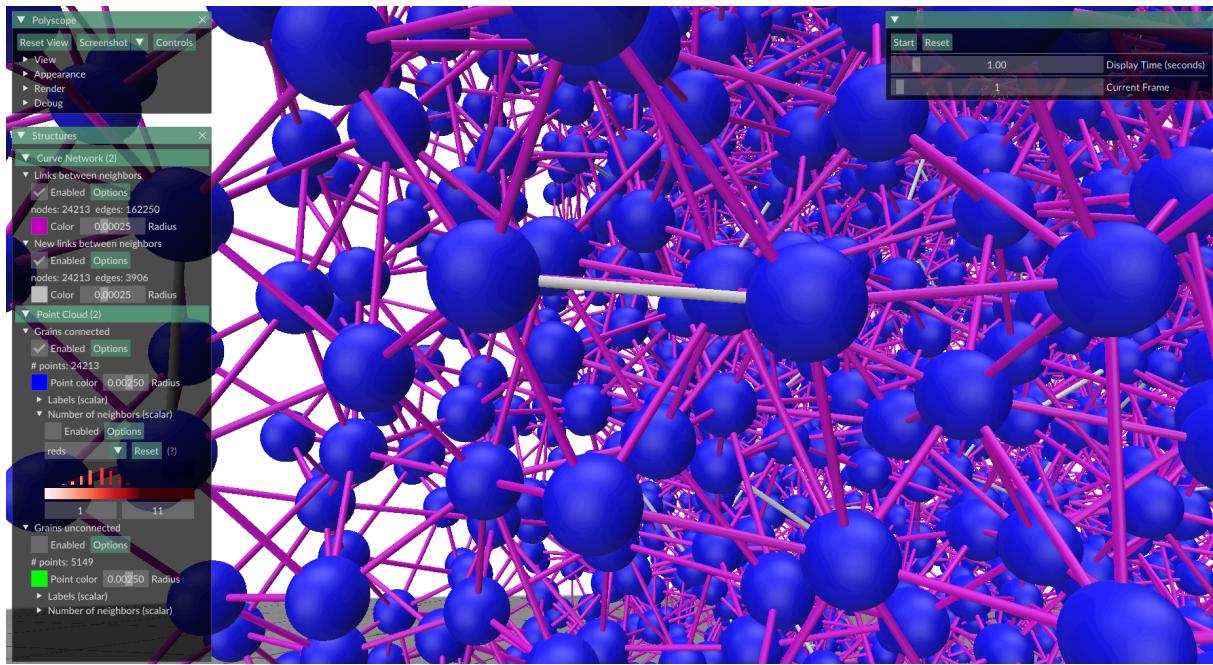
Annexe XII – Exemple de fichier des points de contacts entre les grains



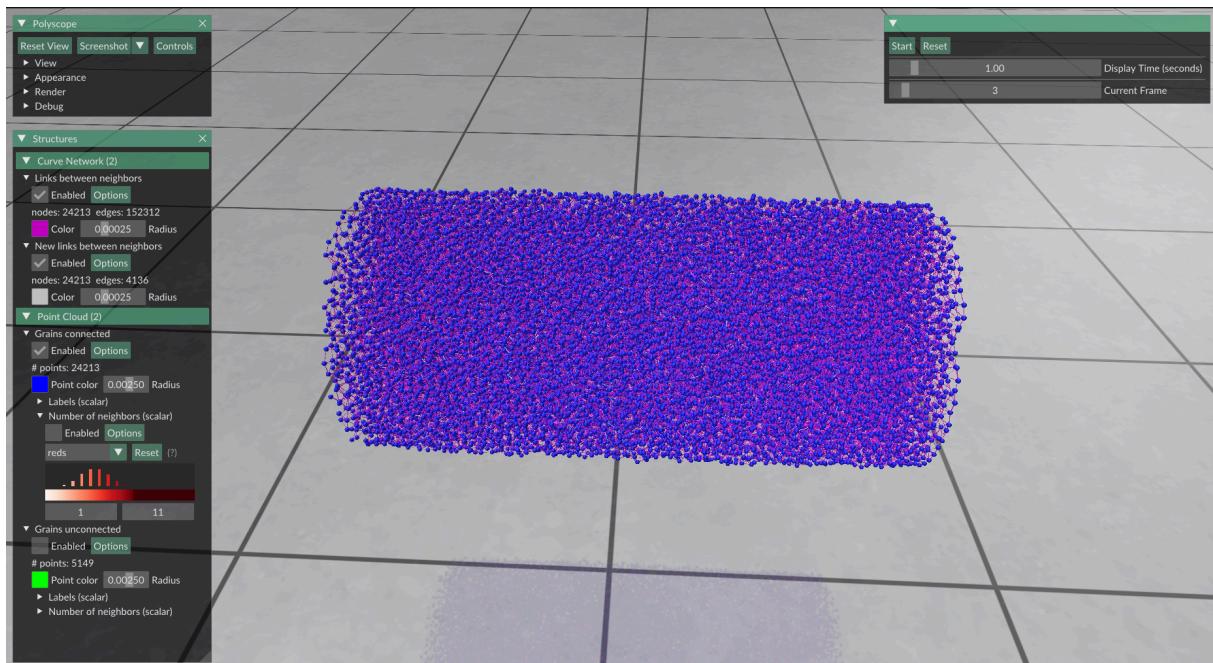
Annexe XIII - Affichage type sur Polyscope



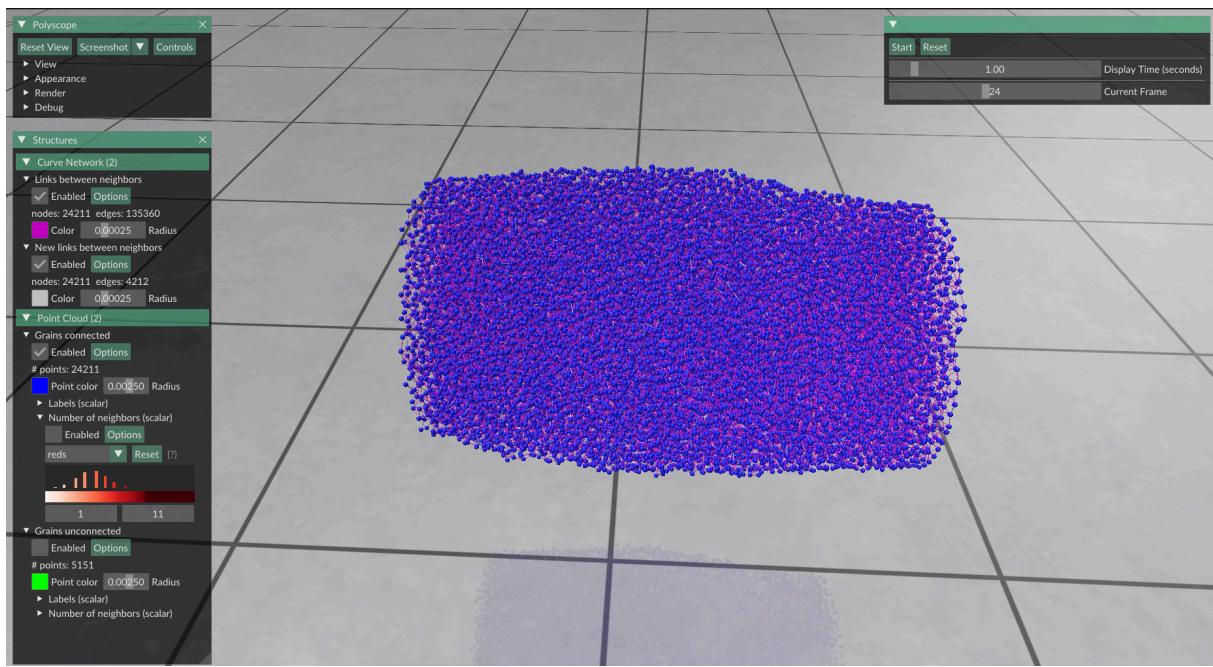
Annexe XIV - Affichage heatmap du nombre des points contacts pour chaque grain



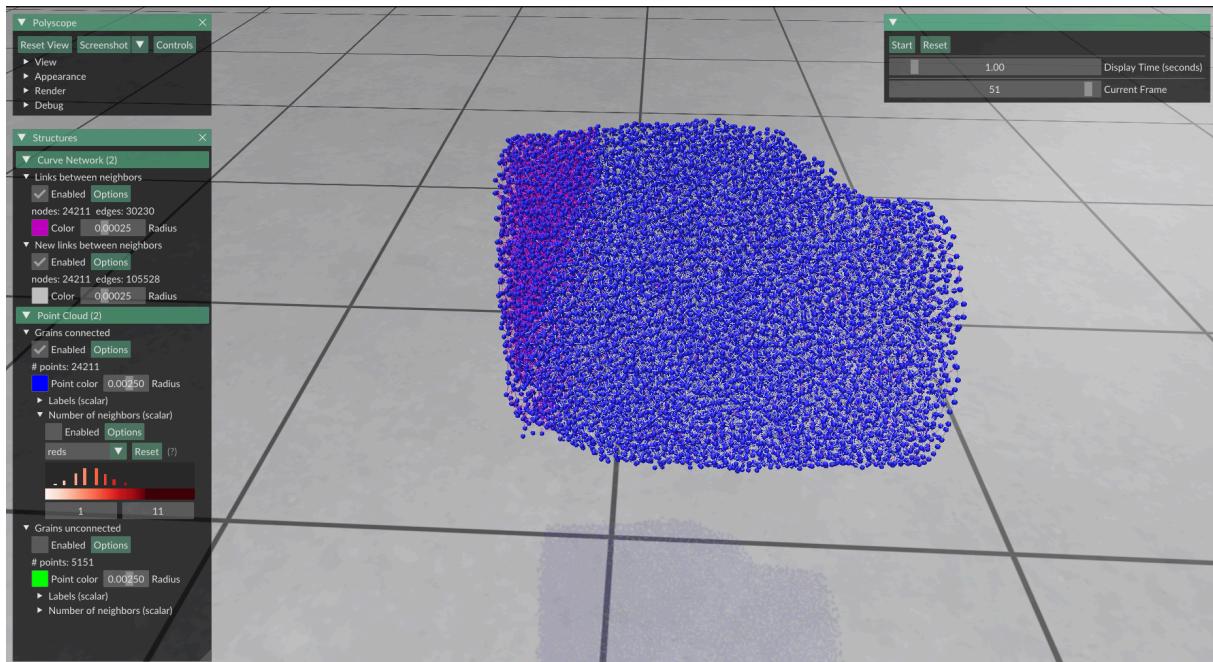
Annexe XV - En blanc : nouvelle arête à la frame 1. En violet : arête présent à la frame 0 et la frame 1.



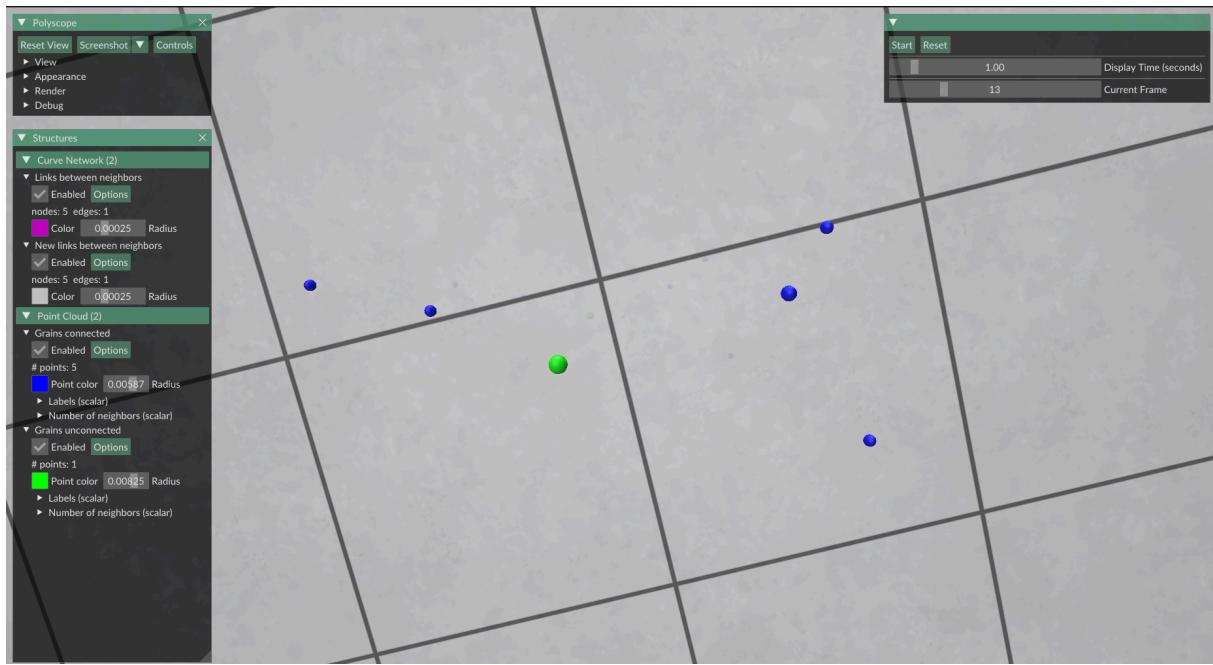
Annexe XVI - Affichage polyscope du cylindre à la frame 3.



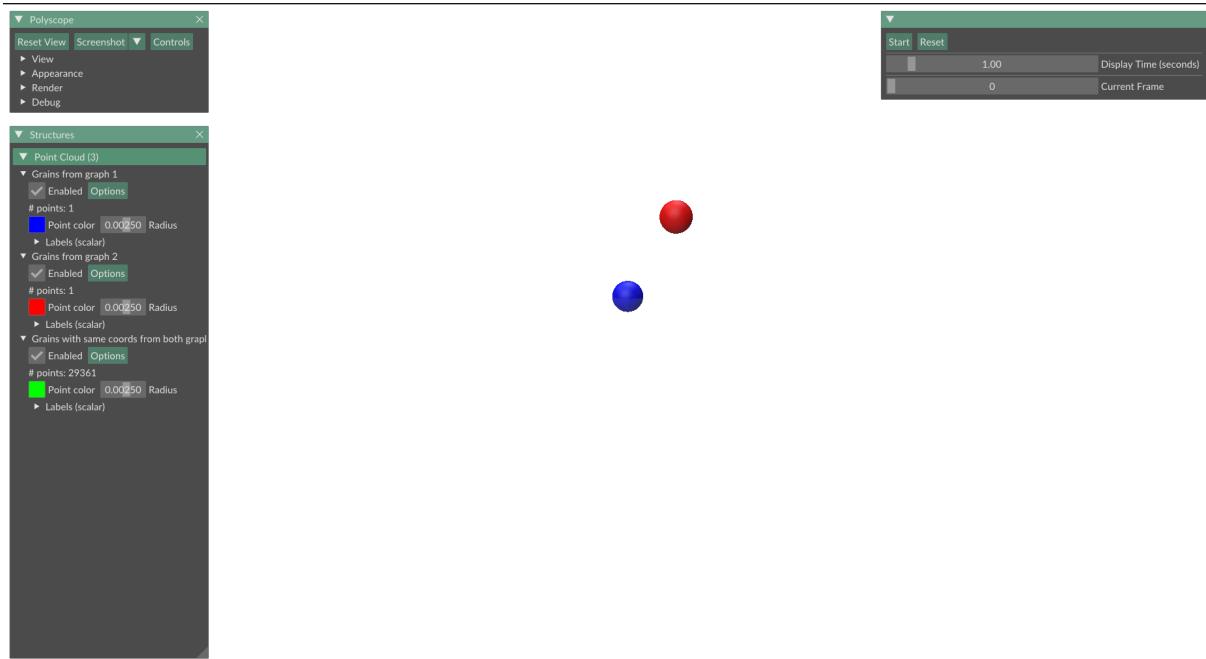
Annexe XVII - Affichage polyscope du cylindre à la frame 24.



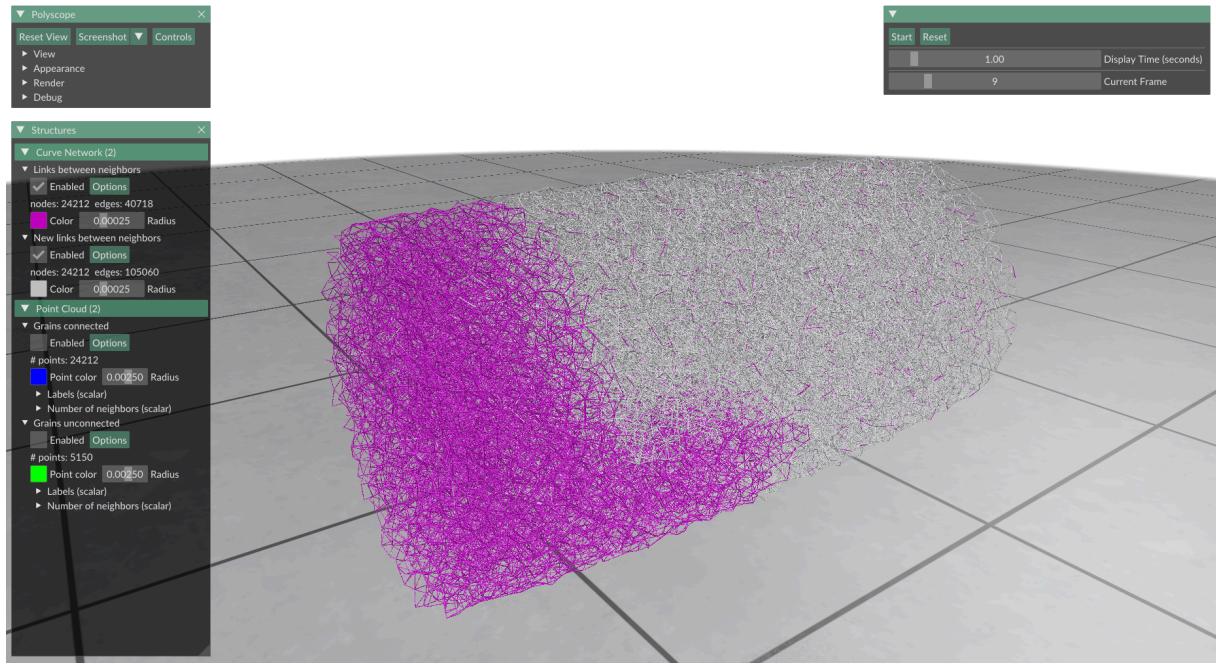
Annexe XVIII - Affichage polyscope du cylindre à la frame 51.



Annexe XIX - Affichage des grains choisis préalablement par l'utilisateur en lançant le programme.



Annexe XX - Affichage deux graphes au fil du temps. Vert : même grain dans les deux graphes. Bleu : grain unique dans le graphe 1. Rouge : grain unique dans le graphe 2.



Annexe XXI – Affichage des points de contacts existant à la frame actuelle et la précédente (violets) et affichage des nouveaux points de contacts (blancs). Exemple de résultat étrange avec les données de la thèse de Gustavo PINZON FORERO.