École des Ponts
ParisTech

Reconstruction de scènes 3D et Optimisation de maillages

Romain CHAUSSONNIER, Waël DOULAZMI, Matias ETCHEVERRY, Alexis GADONNEIX, Ayoub RHIM

Sous la direction de Florent LAFARGE (INRIA - Titane) et Pascal MONASSE (Imagine)

École nationale des Ponts et Chaussées, Champs-sur-Marne, France

Février - Mai 2021

Decreasing the computational cost of Computer Graphics applications requires a concise way to represent complex 3D datasets. This paper presents an automated pipeline to generate concise yet faithful mesh to approximate 3D objects. We first generate a 3D point cloud from a dataset of images using state-of-the art digitization techniques. We then process to generate complex meshes that we ultimately look to simplify. Our approach focus on piece-wise planar structures such as buildings or vehicles. We first detect planar shapes in the meshes and then process to simplify the mesh by collapsing co-planar edges. We proceed to obtain optimized meshes conserving the main structure while using a few faces. We eventually went from a mesh with more than a million faces to one using few hundreds.

Keywords : Surface approximation, surface reconstruction, polygonal surface mesh, piecewise planar structures, mesh simplification, 3D reconstruction.

De nombreuses applications en vision par ordinateur nécessitent de représenter des objets 3D à l'aide de structures de données légères. Cet article présente une méthode automatisée de génération de maillages simplifiés à partir de collections d'images. Nous générons d'abord un maillage à l'aide des images que nous simplifions ensuite en détectant les faces coplanaires puis en supprimant les arrêtes internes aux plans détectés. Notre approche permet de simplifier des maillages dans un temps raisonnable là ou les méthodes déjà implémentées divergent en complexité pour des maillages lourds [1].

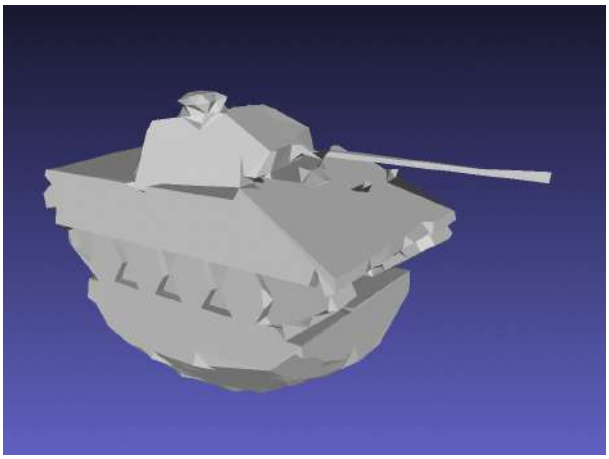

Figure 1: *Résultat final sur le dataset Panther*

Table des matières

Introduction

La reconstruction 3D est une branche riche et active de la vision par ordinateur et les méthodes et applications sont nombreuses. On peut citer parmi les applications la modélisation d'objets pour les effets spéciaux de cinéma, l'animation ou le jeu vidéo, la création de modèles destinés à l'impression 3D ou à la production industrielle, des applications de cartographie 3D, et bien d'autres encore.

Selon l'application envisagée, les critères de choix des méthodes varient. On peut rechercher de la précision dans la représentation de l'objet, de la régularité dans les formes utilisées pour le maillage, une taille réduite à l'essentiel, certaines propriétés topologiques précises, etc.

Notre objectif est ici de proposer un maillage simple, c'est à dire comportant peu de faces, d'objets 3D dits "structurés". Cela signifie que nos objets sont majoritairement composés de formes planaires et cela fait donc sens de vouloir alléger leur représentation. Les structures étudiés sont par exemple des véhicules ou des bâtiments.

Le point de départ des algorithmes est ici un ensemble de photos d'objets 3D capturés sous différents angles. Une partie des jeux de données nous a été fournie par l'INRIA : une ruine de temple, une façade, une monument comportant une statue, un char d'assaut et un phare. Mais il suffit d'un appareil photo à focale fixe pour se constituer un jeu de donnée (comme le camion Mercedes photographié à proximité de l'ENPC). À partir de ces photos et en suivant un certain nombre d'étapes algorithmiques, nous pouvons générer un nuage de point dense de l'objet puis obtenir un maillage fin de notre sujet. Nous allons ensuite chercher, à partir du nuage de point ou du maillage, à obtenir un maillage simple mais aussi fidèle que possible pour un nombre de faces donné. Il faudra pour cela exploiter la structure planaire par morceau de nos objets et tenter de conserver un maximum d'information au fil de la simplification, tout en se conformant à des réalités matérielles telles que le temps d'exécution ou la mémoire vive de nos ordinateurs.

L'état de l'art dans le domaine est aussi varié que le nombre d'applications ou de contraintes que l'on peut imaginer. Concernant la phase de génération du nuage de points et du maillage dense, les méthodes utilisées sont à la pointe pour les objectifs visés et seront détaillées au fil du rapport. Pour la simplification quant à elle, les méthodes de *slicing* [2] utilisées avec des solvers linéaires performants produisent les meilleurs résultats pour des objets simples (peu de composantes planaires). Pour des objets plus complexes ou pour garder un contrôle plus fin sur le niveau de détail voulu, l'approche cinétique de Bauchet et Lafarge [3] produisent des résultats de qualité similaires avec un coût de calcul réduit. Cette méthode ne dispose pour l'instant pas d'implémentation (commerciale ou openSource) et est trop complexe pour le cadre du projet de département, c'est pourquoi nous ne nous y intéresserons pas ici.

1 Reconstruction d'un maillage dense

1.1 Obtention d'un nuage de points dense

La construction du nuage de points d'une scène 3D à partir d'un dataset d'images associées est rendue possible par les outils issus de la SFM (Structure From Motion). Cette technique se décompose en 2 grandes étapes :

- **La recherche de correspondances** : recherche des points d'intérêts pour chaque image du dataset et création du graphe de scène.
- **Reconstruction incrémentale** : calcul de positions dans l'espace des caméras associées à chaque image, ainsi que d'une partie des points d'intérêts de chaque image.

1.1.1 Recherche de correspondances

Figure 2: *Quatre des 380 prises du camion*

On considère un dataset d'entré, constitué d'images d'une même scène 3D. Chaque image représente une partie de la scène 3D qu'on souhaite numériser. L'étape de correspondance permet d'obtenir un «graphe de scène», où chaque sommet du graphe correspond à une image du dataset, et où une arête relie 2 sommets lorsque les deux images associées ont un nombre de points d'intérêts communs géométriquement compatibles suffisant. Pour une paire d'images reliées dans ce graphe, on dispose également de la correspondance géometriquement compatible entre les points d'intérêts des deux images. Plus précieusement, la recherche de correspondance se décompose en 3 étapes : la Feature Extraction, le Feature Matching, et la Geometry Verification.

Feature Extraction :

Pour chaque image, on va calculer les points d'intérêts et les descripteurs associés.

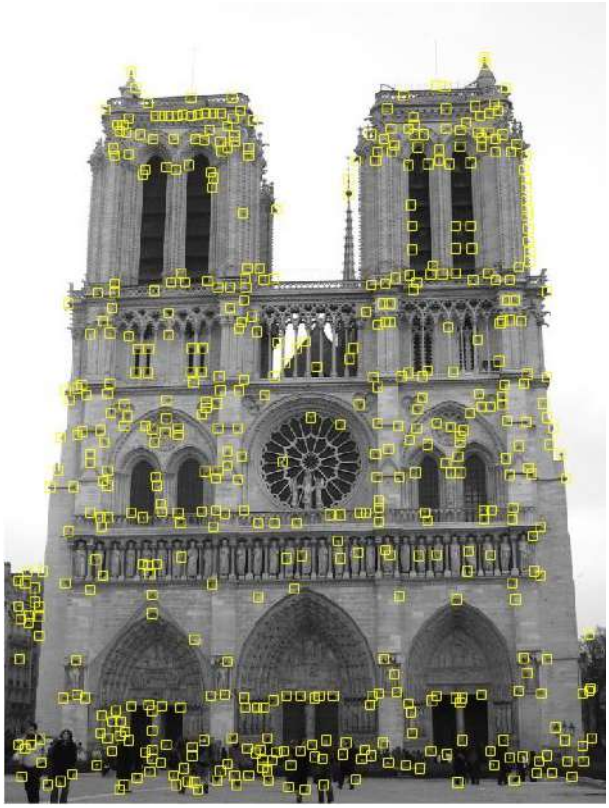

Figure 3: *Points d'intérêts détectés par SIFT sur Notre-Dame. Ils ne correspondent pas nécessairement à la notion intuitive de point remarquable.*

Feature Matching :

L'objectif de cette étape est d'associer les images par paires selon leur nombre de points d'intérêts ayant des descripteurs proches. On obtient alors en sortie, une première version du graphe de scène, où chaque image est un sommet et où une arête est mise lorsque le nombre de points d'intérêts ayant des descripteurs assez proches est assez élevé. Cependant, cette étape ne garantit pas que chacun des points de chaque paire de points d'intérêts qu'on a précédemment associés correspondent aux même points géométriques 3D. Ainsi, cette étape ne garantit pas que deux images reliées sont effectivement des images représentant la même région de la scène 3D (elle ne s'overlappent pas nécessairement). C'est pour cela qu'il est nécessaire de s'assurer de la compatibilité géométrique entre chaque paire d'images reliées par une arête.

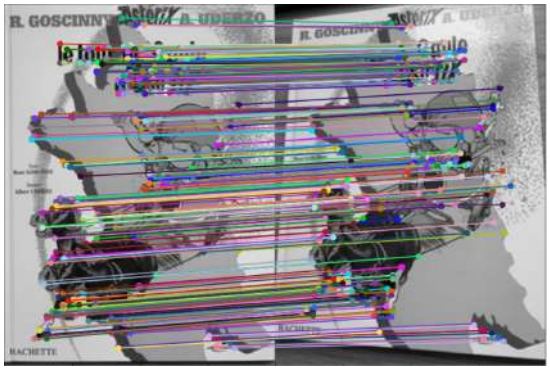

Figure 4: *2 images d'une même scène sur lesquelles on a detecté des points d'intérêt. Chaque segment de couleur relie un point d'intérêt de l'image de gauche, et un autre point d'intérêt de l'image de droite.*

Geometry Verification :

Cette étape vérifie, pour chaque paire d'images connectées du graphe de scène, s'il est possible de trouver une transformation géométrique qui envoie un nombre de points d'intérêt suffisant de la première image, sur la deuxième. Si c'est le cas, l'arête du graphe de scène est conservée, sinon, cela signifie que les deux images ne correspondent pas à la représentation de la même région 3D de la scène, donc les points d'intérêts matchés ne correspondant pas aux même points 3D de la scène réel : l'arête est supprimée. On répète l'opération pour toutes les paires d'images reliées par une arête dans le graphe de scène. En sortie, on obtient un graphe de scène épuré, où chaque paire d'images reliées contient des points d'intérêts communs géometriquement compatibles suffisants.

1.1.2 Reconstruction incrémentale

C'est à partir de ce graphe de scène, et de la correspondance des points d'intérêts pour chaque paire d'images reliées par une arête, qu'on détermine les positions des caméras associées à chaque image, ainsi que les positions des points d'intérêts géometriquement compatibles dans l'espace. Ce sont ces derniers qui constitueront le nuage de points de la scène 3D. On dit que la reconstruction est incrémentale, car les positions des points et des caméras sont calculées les unes à partir des autres, image par image.

Initialisation :
Comme la reconstruction est incrémentale, les calculs de positions doivent être précis, sans quoi les erreurs s'accumuleraient sur tous les calculs ultérieurs. L'étape d'initialisation consiste à déterminer les images pour lesquelles on fait le moins d'erreurs. Ces images correspondent typiquement à des sommets vivant dans une région dense du graphe de scène. Une fois que cette étape est effectuée, l'algorithme incrémental va pouvoir commencer : à chaque itération, on va choisir une image parmi le dataset et caluler sa pose (Image Registration), puis on va calculer la position des nouveaux points (Triangulation), puis on s'assure que cette estimation de position est précise, en la changeant au besoin (Bundle Adjustment).

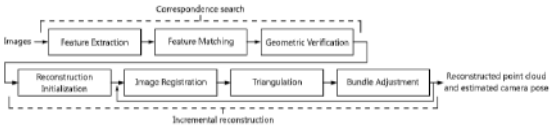

Figure 5: *Schéma de principe [4] illustrant le fonctionnement d'un pipeline SFM.*

Image Registration :

Comme dit précédemment, cette étape va consister à choisir une image dans le dataset qui n'a pas été traitée, et qui présente des points d'intérêts communs avec les images géométriquement compatibles. Grâce à cette correspondance, on est capable d'estimer la position de la caméra dans l'espace, à partir de la position des points et des caméras précédentes.

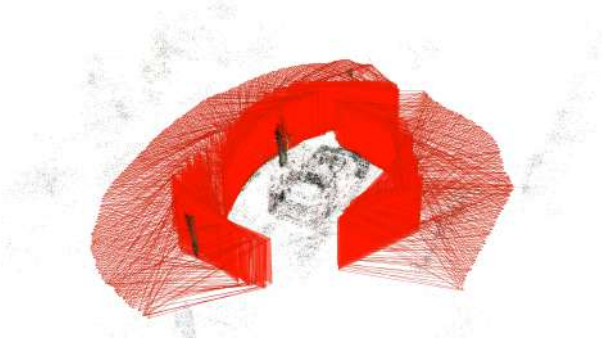

Figure 6: *Construction du nuage de points et positionnement des angles de vue*

Triangulation :

Lorsque la caméra associée à la nouvelle image a été positionnée dans l'espace, il reste à calculer les positions dans l'espace des nouveaux points d'intérêts, qui ne sont associés à aucun point d'intérêt des images précédentes.

Bundle Adjustment :

Cette étape permet de limiter l'impact des erreurs d'estimation des positions 3D des points et des caméras. Ceci est d'autant plus important que la reconstruction est incrémentale : chaque erreur se propage et perturbe la qualité des poses suivantes. La technique consiste d'abord à reprojeter les points 3D nouvellement calculés. Pour un point 3D donné, la reprojection se fait sur les plans des caméras qui contiennent le point d'intérêt qui représente le point 3D. Le Bundle Adjustment consiste ensuite à modifier les positions des points et des caméras pour minimiser la distance entre les points projetés, et les points d'intérêts correspondants.

RANSAC

Dans toutes les estimations précédentes, certains points peuvent être aberrants (outliers) et peuvent donc perturber les résultats. Pour une estimation robuste, on utilise souvent un algorithme type RANSAC. Cette classe d'algorithme permet d'estimer les paramètres d'un modèle malgré la présence d'outliers dans les données d'entrées.

1.2 Notre Pipeline de reconstruction

Pour chaque étape de la SFM, plusieurs algorithmes sont disponibles. Le choix de ces algorithmes influe fortement la qualité des résultats. Un tel choix d'algorithmes successifs est appelé pipeline de SFM. Il existe plusieurs outils proposant des pipelines SFM, comme Colmap, OpenMVS, Theia, VisualSfm etc.

Il existe plusieurs pipelines de reconstruction de scènes 3D open source. Parmi eux, on retrouve COLMAP, OpenMVG, Theia, etc. Chacun propose un ensemble d'algorithmes pour réaliser chacune des étapes de SFM décrites précédemment. Durant ce projet, nous avons dû choisir un des pipelines OpenSource existant pour reconstruire les nuages de points 3D à partir de nos datasets d'images.

Les pipelines OpenMVG et Colmap se distinguent par leur grande diversité d'algorithmes. En analysant plusieurs Benchmark, nous avons conclu que l'utilisation de COLMAP était plus avantageuse au vu des datasets que nous avons. On y voit notamment dans [4] que Colmap estime bien mieux les positions des caméras que OpenMVG, mais qu'il génère moins de points 3D. Étant donné le caractère structuré de notre dataset, les formes qu'on étudie sont assez régulières, et donc il vaut mieux privilégier la précision des positions des points quitte à en avoir en moindre nombre par la pipeline de reconstruction. Par ailleurs, le benchmark [5] vient étayé ce choix en montrant, sur des datasets structurés similaires aux nôtres, que COLMAP donne des meilleurs résultats en terme de précision.

Nous nous sommes donc orientés vers un pipeline de reconstruction de type COLMAP pour générer notre nuage de points.

Table 1. Incremental SfM pipelines algorithm comparison.

	Feature Extraction	Feature Matching	Geometric Verification	Image Registration	Triangulation	Bundle Adjustment	Robust Estimation
COLMAP	SIFT [11]	Exhaustive Sequential Vocabulary Tree [14] Spatial [11] Transitive [15]	4 Point for Homography [9] 5 Point Relative Pose [10] 7 Point for F-matrix [9] 8 Point for F-matrix [8]	P3P [9] EPnP [20]	sampling-based DLT [11]	Multicore BA [26] Ceres Solver [25]	RANSAC [9] PROSAC [7] LO-RANSAC [12]
OpenMVG	SIFT [11] AKAZE [17]	Brute force ANN [18] Cascade Hashing [11]	affine transformation 4 Point for Homography [9] 8 Point for F-matrix [8] 7 Point for F-matrix [9] 5 Point Relative Pose [10]	6 Point DLT [9] P3P [20] EPnP [20]	linear (DLT) [9]	Ceres Solver [25]	Max-Consensus RANSAC [9] LMed [21] AC-Ransac [13]
Theia	SIFT [11]	Brute force Cascade Hashing [11]	4 Point for Homography [9] 5 Point Relative Pose [10] 8 Point for F-matrix [8]	P3P [20] P6P / DLT [9] P4P [16] P3P [16]	linear (DLT) [9] 2-view [16] Midpoint [16] N-view [16]	Ceres Solver [25]	RANSAC [9] PROSAC [7] Arrsac [23] Ltvsac [24] LMed [21]
VisualSFM	SIFT [11]	Exhaustive Sequential Preemptive [22]	n/a	n/a	n/a	Multicore BA [26]	RANSAC [9]
Bundler	SIFT [11]	ANN [18]	8 Point for F-matrix [8]	DLT based [9]	N-view [16]	SBA [27] Ceres Solver [25]	RANSAC [9]
MVE	SIFT [11] + SURF [17]	Grid-wise + exhaustive Cascade Hashing [11]	8 Point for F-matrix [8]	P3P [20]	linear (DLT) [9]	own LM BA	RANSAC [9]

Figure 7: *Tableau [4] résumant les différents algorithmes disponibles pour chaque étape, pour chacun des pipelines open source que nous considérons. On remarque en particulier que COLMAP propose beaucoup d'algorithmes en comparaison avec OpenMVG*

Il est donc fondamental de sélectionner des algorithmes adaptés à notre dataset. En particulier, comme notre dataset est structuré, les algorithmes doivent être adaptés aux surfaces assez régulières. Nous avons identifié deux contraintes principales :

- Les zones planes : notre dataset comporte beaucoup de plans
- Les repeated Patterns : sur un plan, le même motif peut être répété

Le choix des algorithmes que nous avons fait a été conditionné par ces deux contraintes. Voici le pipeline que nous avons utilisé durant ce projet :

- Feature Extraction : SIFT
- Feature Matching : exhaustive
- Geometry Verification : 5 Point Relative Pose
- Image Registration : P3P
- Triangulation : Sampling-based DLT
- Bundle Adjustment : Ceres Solver
- Robust estimation : RANSAC

A l'issue de ce pipeline, nous avons pu obtenir un nuage de points, que nous avons densifié à

l'aide d'OpenMVS.

Figure 8: *Nuage après densification*

1.3 Création du maillage dense

Une fois le nuage de points dense obtenu, nous avons cherché à créer un maillage triangulaire à partir de celui-ci. Pour ce faire, nous avons choisi d'utiliser l'algorithme de poisson. Mais dans un premier temps nous avons effectué un traitement préalable du nuage de points pour avoir un meilleur contrôle et plus de régularité sur ce maillage qui servira par la suite de point de départ et de référence.

1.3.1 Preprocessing

Cette étape de pré-traitement intervient principalement pour deux raisons. D'une part obtenir un nuage de points plus régulier en lissant légèrement la surface et en retirant les outliers (points erronés). D'autre part pour échantillonner le nuage de points afin d'avoir une densité plus uniforme sur l'objet et pour que le nombre de faces ne soit pas trop élevé (et que les données soient plus facilement manipulables par la suite).

La première étape est réalisée à la main et ne peut pas vraiment être automatisée. Il s'agit d'isoler l'objet qui nous intéresse du reste de la scène. Par exemple une montagne en fond ne nous intéresse pas et compliquerait beaucoup la création du maillage. Cette étape a été réalisée à l'aide de l'outil `Meshlab` qui offre une interface visuelle puissante.

Figure 9: *Nuage de points avant nettoyage*

Figure 10: *Objet isolé manuellement avec Meshlab*

La suite du pré-traitement a été réalisée à l'aide de la bibliothèque C++ `CGAL` qui propose de nombreux algorithmes et structures de données pour traiter efficacement des problèmes de géométrie 2D ou 3D, notamment le package dédié au traitement des nuages de points [6]. L'étape suivante consistait à supprimer les outliers de notre nuage de points. Pour détecter ces derniers, nous avons regardé la distance moyenne à un nombre donné de voisins de chaque point. Si cette distance est au dessus d'un certain seuil, on supprime le point. Le seuil choisi était généralement autour de deux fois la distance moyenne séparant un point de ses voisins. L'algorithme considérait généralement qu'environ 3 à 5% des points étaient des outliers. Il faut néanmoins être vigilant, en effet, si l'on prend un seuil trop élevé, on risque de perdre de l'information dans les zones de faibles densités. Le compromis à effectuer dépend ici de la densité et de la qualité du nuage de point d'origine.

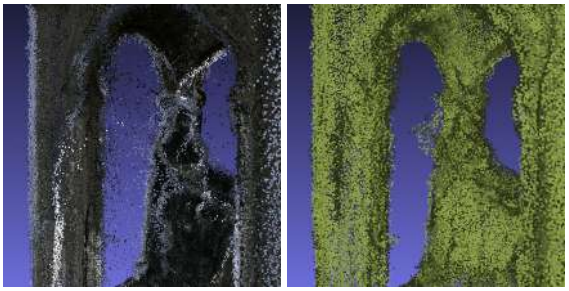

Figure 11: *A cause des reflets, des changements de luminosité ou autres, on voit apparaître un certain nombre d'outliers qu'il faut supprimer pour garantir un maillage fidèle à l'objet d'origine. Ici sur le dataset Francis.*

Afin d'améliorer la précision du maillage de

poisson réalisé ensuite, nous avons choisi d'utiliser une fonction de smoothing de CGAL. Pour garantir une bonne conservation des angles, nous avons utilisé le *bilateral smoothing*, un filtre non-linéaire préservant les arêtes. Ce lissage est léger pour éviter de perdre de l'information.

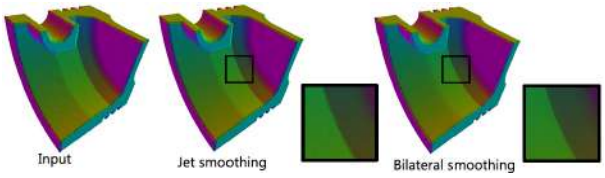

Figure 12: *Illustration de la méthode de bilateral smoothing*

La dernière étape du pré-traitement est l'échantillonnage du nuage de points. Cette étape est importante car elle permet d'homogénéiser le nuage de points et de lui donner une taille plus raisonnable que les millions de points obtenus à l'issu de la phase précédente. L'algorithme utilisé pour l'échantillonnage prend en entrée une longueur (généralement la distance caractéristique entre les points du nuage) et va découper l'espace en une grille de mailles proportionnelles à notre paramètre. Tous les points présents dans une même case seront fusionnés. L'avantage de cette approche est la très bonne conservation des zones de faible densité présentes sur les zones peu accessibles de notre objet (toits, intérieurs ...).

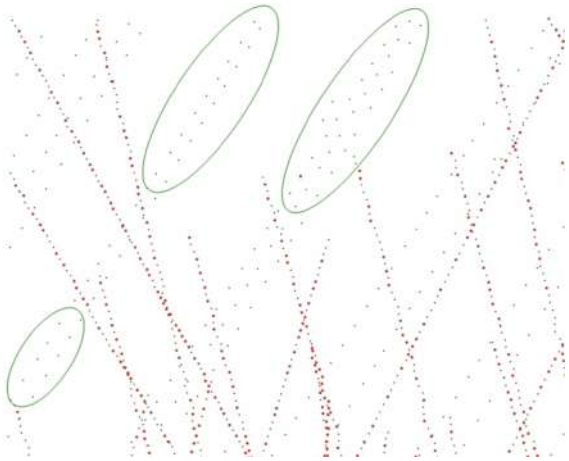

Figure 13: *Échantillonnage du nuage de points, les zones peu denses (en vert) sont conservées*

1.3.2 Obtention du maillage

Nous sommes maintenant prêts à créer un maillage qui servira de référence pour la suite du projet. Il existe plusieurs algorithmes pour construire un maillage à partir d'un nuage de point et nous avons choisi la reconstruction de Poisson [7]. Cette méthode repose sur le calcul d'une fonction implicite dont le gradient coïncide avec les normales du nuage de points. Elle reconstruit ensuite un maillage dont la régularité et la finesse peuvent être contrôlées par l'utilisateur. Les avantages de cette méthode par rapport aux autres candidats dans le domaine de la reconstruction (*Advancing front* et *Scale space*) est qu'elle produit des surfaces lisses, fermées et manifold (en français variété, c'est-à-dire localement semblable à $\mathbb{R}^2$), en plus d'offrir un bon contrôle. De plus, cette méthode est assez robuste au bruit et aux outliers qui auraient persisté après le pré-traitement.

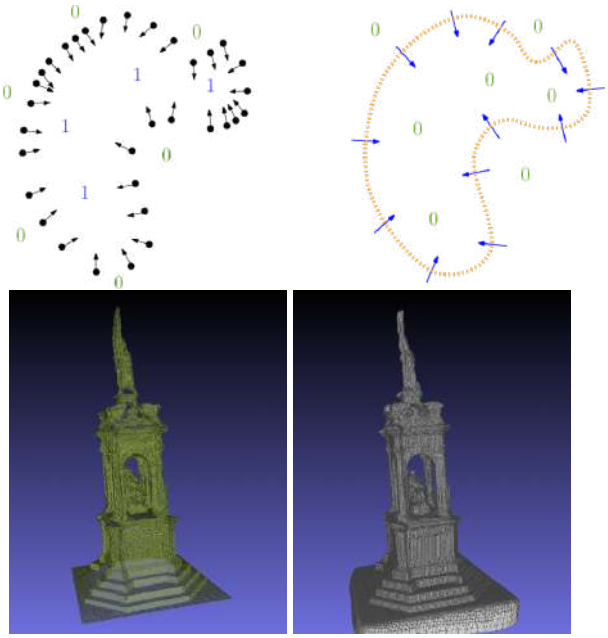

Figure 14: *Schéma de principe de la reconstruction de Poisson (indicatrice et fonction implicite) puis application de la méthode avec une profondeur d'octree de 7 sur le dataset Francis*

Les inconvénients de cette méthode quant à eux sont la nécessité d'avoir des normales fiables associées à nos points (ce qui est notre cas), le fait que les points d'origine ne se trouvent pas nécessairement sur la surface, et la faible capacité d'interpolation de la méthode. Ce dernier point est le plus gênant, en effet certaines zones qui n'apparaissent pas dans le nuage de point risquent d'être mal évaluées, laissant souvent place à des "bulbes" caractéristiques de la reconstruction de Poisson.

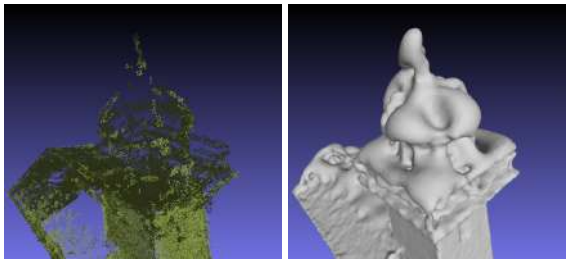

Figure 15: *Les zones peu denses du nuage de points sont mal interpolées, laissant apparaître ces "bulbes" pour prolonger naturellement les normales.*

Nous avions à notre disposition deux implémentations de cette méthode, l'une dans `CGAL` reposant sur une triangulation de Delaunay et l'autre dans `Meshlab` utilisant un octree. Les résultats sont très similaires mais nous avons choisi la seconde option qui offre un contrôle plus précis sur la finesse du maillage désirée et un outil visuel pour ajuster nos choix en temps réel (par opposition au C++ qui nécessite la compilation et l'exécution d'un code puis la visualisation dans un logiciel extérieur).

À ce stade, il est difficile d'évaluer la qualité de notre maillage par rapport au nuage de points ou à l'objet d'origine, en effet, la quasi-totalité des points du nuage (plus de $99,9\%$) se trouvent sur notre surface et nous ne disposons pas d'un modèle 3D de l'objet (*ground truth*) pour comparer. Nous devons donc nous baser sur la qualité des méthodes utilisées en observant des benchmarks et sur des critères plus subjectifs de ressemblance pour ajuster les paramètres.

2 Remaillage

Une fois qu'on a obtenu un maillage précis avec plusieurs dizaines de milliers de points, il faut simplifier ce maillage au maximum pour obtenir un objet avec des formes régulières. En effet, nous travaillons sur des objets qui peuvent se représenter facilement par des formes planaires, comme un camion.

2.1 Détection de plans au sein du maillage

Une fois notre maillage simplifié obtenu, il s'agit d'obtenir un ensemble de plans plus cohérent dans lesquels circonscrire notre structure. Chaque surface triangulaire du maillage sera associée à un de ces plans, auquel elle appartient plus vraisemblablement dans le monde réel. Deux méthodes de la bibliothèque `Cgal` peuvent être utilisées pour retrouver ces plans, *RANSAC* et *Region growing*. Nous utilisons les implémentations du package *Shape detection* [8].

2.1.1 Ransac

La méthode *RANSAC*, pour *RANdom SAmple Consensus* consiste à obtenir une série de plans depuis un ensemble de points, de manière aléatoire.
Notre maillage simplifié possède à la fois des points pertinents et bien positionnés (inliers) et des points aberrants (outliers). La méthode *RANSAC* détermine alors les meilleurs plans passant par ces points, de telle sorte que les outliers aient peu d'influence. Elle se décompose en une répétition de ces 4 étapes:

- Extraire aléatoirement un sous ensemble de points du maillage
- Associer le meilleur plan à ces points
- Estimer la qualité de ce plan sur toutes les données qui n'ont pas été extraites
- Conserver ce plan s'il est suffisamment fidèle au maillage.

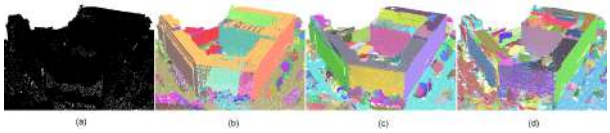

Figure 16: *Variation de RANSAC en diminuant ε la distance maximale au plan autorisée*

2.1.2 Region Growing

Par ailleurs, il existe également la méthode itérative *Region Growing*. Cette méthode a pour but de déterminer les meilleurs plans passant par nos points du maillage. Elle se décompose en 4 étapes:

- Extraire aléatoirement un point du maillage.
- Associer récursivement les plus proches voisins de ce point.
- Associer un plan à tous ces points
- Recommencer sur tous les points restants.

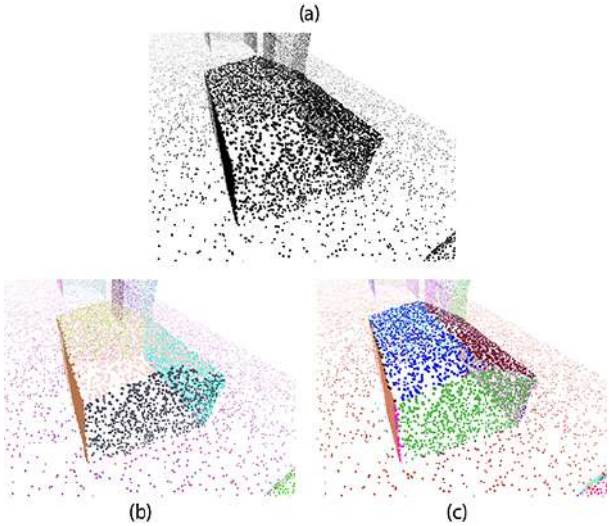

Figure 17: *Variation du résultat d'un Region growing en diminuant l'angle d'acceptation de points maximal*

2.1.3 Comparaison des deux méthodes

Dans chacun des cas, l'utilisateur rentre en entrée la précision de son modèle. Sur la méthode *RANSAC*, la précision est un seuil pour savoir si un plan peut être considéré comme interpolant, tandis que sur la méthode *Region Growing*, la précision est un seuil pour déterminer si un point peut être considéré comme un voisin.
Plus la précision est élevée, plus les points seront vus comme des outliers. Cette précision est en lien direct avec la complexité du modèle. Plus la complexité est élevée, plus le nombre de plans en sortie sera élevé. Il faut donc veiller à garder une complexité assez faible, pour limiter les itérations de notre algorithme et pour s'assurer d'obtenir des surfaces simples en sortie.

Nous pouvons ainsi noter les points forts et faibles de chaque méthode.
La méthode *RANSAC* est très robuste aux outliers. Cependant, elle augmente très en vite en complexité en temps avec le nombre de points dans le maillage. Par ailleurs, la méthode *Region Growing* est assez sensible au bruit mais peut directement s'implémenter à partir d'un maillage sur `Cgal`.

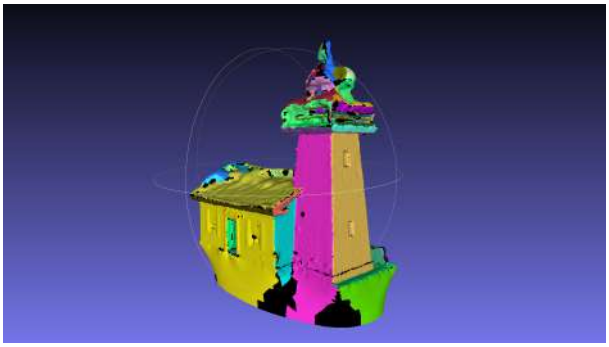

Figure 18: *Plans résultants d'un Region growing*

2.2 Slicing

La première méthode de remaillage que nous avons utilisée est un procédé de *slicing*. Cet algorithme fait office d'état de l'art pour les problèmes de reconstruction d'objets planaires par morceaux possédant un faible nombre de faces. Nous nous appuyons sur l'implémentation du package `Cgal` *Polygonal surface reconstruction* [1].

La méthode prend en entrée un nuage de points ou un maillage (de préférence orienté) et se décompose en plusieurs étapes :

- **Détection de plans** : On utilise d'abord l'une des méthodes de détection de primitives décrites précédemment. Les primitives extraites sont des plans 2D orientés. C'est lors de cette étape que l'on va contrôler le nombre de faces présentes dans le maillage final.

- **Création des faces candidates** : L'étape vise à créer un ensemble de faces potentielles pour notre maillage final. La génération de ces faces suit un procédé assez visuel. On prolonge tous les plans jusqu'aux limites de la *bounding box* (parallépipède de volume minimal contenant tout notre maillage) puis on découpe ces plans en faces en créant une arrête lorsque deux plans s'intersectent et un sommet lorsque 3 plans ou plus s'intersectent.

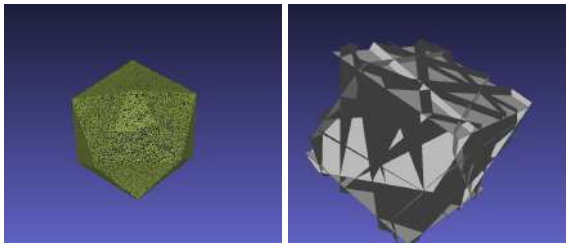

Figure 19: *Le nuage de points pour un objet simple (balle) et les faces candidates, c'est à dire les plans prolongés puis découpés selon les intersections.*

- **Sélection des faces** : La dernière étape est la plus importante mais aussi la plus complexe et coûteuse. L'objectif est de choisir, parmi les faces candidates, un ensemble optimal de faces polygonales (non nécessairement triangulaires) tout en garantissant une surface topologiquement correcte. Ce sous-ensemble est trouvé comme solution d'un problème d'optimisation linéaire sous contraintes en nombres entiers, dit MILP [2]. En l'occurrence nous avons utilisé le solver opensource `SCIP` supporté par le package `Cgal`.

Le principal atout de cette méthode est son optimalité pour un nombre de primitives donné. Cette méthode correspond parfaitement à nos objectifs de minimisation du nombre faces et de préservation de la structure. Néanmoins, la troisième étape décrite demande de résoudre un MILP avec un nombre de variables qui croît exponentiellement en fonction du nombre de plans détectés. Même les techniques les plus efficaces (génération de colonnes) et les solvers les plus performants ne peuvent résoudre le problème au delà d'une centaine de primitives. Dans notre cas nous ne pouvions excéder une vingtaine de plans ce qui rend la méthode très peu flexible et contrôlable et non-adaptée à notre data set.

Figure 20: *Le slicing fonctionne très bien sur des objets simples comme la balle mais sur le dataset temple par exemple, il faudrait plus de plans pour avoir des résultats fidèles et ce n'est pas possible à cause de l'explosion de la complexité*

2.3 Variationnal Shape Approximation

Une des méthodes de remaillage déjà implémentée dans `CGAL` est : *Triangulated Surface Mesh Approximation* [9]: elle repose sur la détection de primitives planaires (*planar proxies*) en utilisant la méthode décrite par *Pierre Alliez* dans son article *Variationnal Shape Approximation* [10].

L'algorithme se déroule en 4 étapes :

- **Choix de noyaux** (*seeds*) : Les seeds sont des faces de bases qui nous serviront pour le clustering consistant à associer chaque face à un proxy. On peut les sélectionner de manière aléatoire ou hiérarchique. Nous avons choisi la méthode hiérarchique qui permet d'éviter les minima locaux.[9]

- **K-means clustering** sur les faces avec les noyaux sélectionnés. La grande différence avec Ransac et *region growing* est que toutes les faces sont associées à une primitive.

- **Simplification des proxies** : lorsque deux proxies appartiennent au même plan, on les fusionne, au contraire lorsque dans une proxy les points ne sont pas coplanaires (à une marge d'erreur près) on split le proxy. On recommence ces opérations jusqu'à convergence de l'algorithme.

- **Sélection des sommets et remaillage** : Une fois que chaque face est associée à un proxy, on sélectionne les sommets adjacents à au moins 3 régions (à l'intérieur du maillage) ou deux régions à la frontière du maillage. D'autres sommets sont sélectionnés avec des critères plus fins. Ensuite on reconstruit un maillage à l'aide d'une triangulation de Delaunay sur les sommets sélectionnés.

Un paramètre sur lequel nous avons joué pour régler la finesse des résultats est le nombre maximal de proxies détectées.

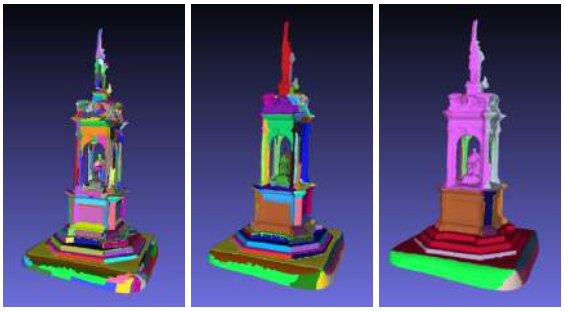

Figure 21: *Proxies détectés sur la statue de Francis, avec 500, 100 puis 20 proxies, on observe qu'avec 20 proxies l'algorithme ne détecte plus le haut de la statue. A 100 proxies le résultat est encore satisfaisant, au niveau des escaliers notamment.*

On peut observer les maillages obtenus :

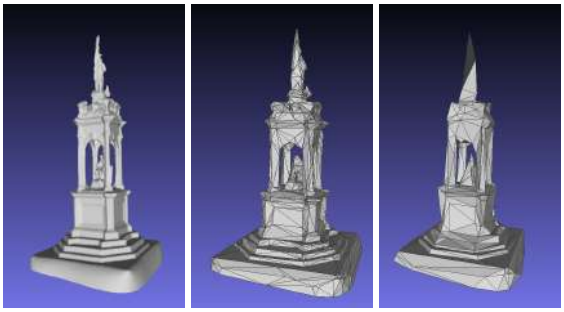

Figure 22: *Maillages générés par l'algorithme VSA sur Francis, de gauche à droite : le maillage avant simplification (136 606 faces), le résultat avec 500 proxies (2051 faces), le résultat avec 100 proxies (576 faces).*

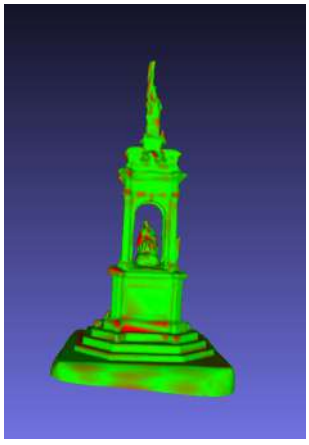

Figure 23: *Visualisation de la distance de Haussdorf entre le maillage généré avec 100 proxies et le maillage original (cf. Section 2.5). Cette représentation permet de mettre en évidence les zones perdues.*

Le principal défaut de cette méthode est son manque de propriétés topologiques : les surfaces générées ne sont pas des 2-variétés, certaines faces s'intersectent, et sont mal orientées. Ce problème se voit bien sur un maillage très simple comme celui de cet ourson :

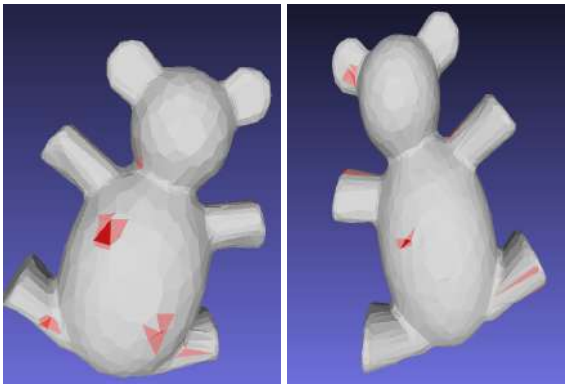

Figure 24: *Maillage généré par le VSA sur un maillage d'ourson. Sont sélectionnées en rouge les faces qui s'intersectent. Les faces en rouge foncé sont mal orientées.*

Enfin on peut observer les résultats de VSA sur un objet avec plus de surfaces planaires comme le camion :

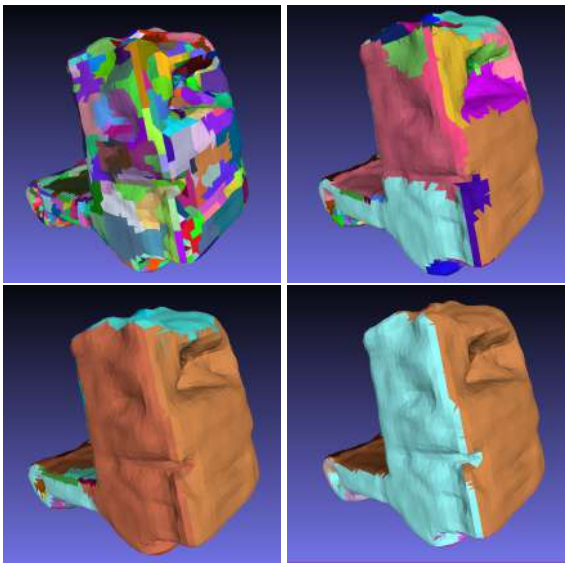

Figure 25: *Proxies détectés sur le camion, avec 1000, 100, 50 puis 20 proxies. On ne détecte plus le toit du camion avec 20 proxies. Le résultat avec 50 proxies est visuellement satisfaisant.*

On retient la sortie avec 50 proxies qui donne un maillage de 178 faces :

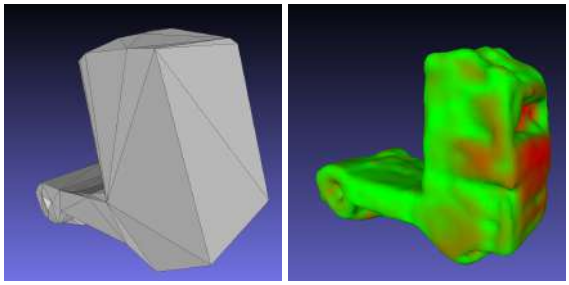

Figure 26: *Maillage généré avec 50 proxies et visualisation de la distance de Hausdorff associée. Le mesh perd les détails de la face avant (phare et pare-brise) tout en conservant une structure de camion.*

2.4 Approche personnalisée

Nous avons cherché à développer notre propre pipeline pour traiter le problème afin de pallier aux défauts des méthodes présentées ci-dessus : la complexité exponentielle du *slicing* et les mauvaises propriétés topologiques du VSA.

Ces deux approches sont constructives : les faces sont construites à partir de propriétés globales, à savoir les *proxies* détectés par *region growing* ou *clustering*. Notre approche repose sur le concept de décimation, le nombre de faces est réduit en supprimant itérativement des arrêtes du maillage d'origine. La méthode se divise en trois phases principales : la détection des primitives, la projection sur les primitives puis la décimation.

2.4.1 Description générale

Détection de proxies

Comme pour les algorithmes précédents, notre méthode commence par la détection de *proxies*, les composantes planaires dont la conservation est l'objectif principal du projet. Comme justifié précédemment, nous identifions les primitives à l'aide du *region growing*. Nous commençons en sélectionnant les grandes primitives, en imposant un nombre minimum de faces par primitive élevé et de grands écarts à la surface permis. Ces éléments correspondent aux grandes faces de notre objet.

Néanmoins, toutes les faces n'ont pas encore été attribuées. Les zones restantes correspondent aux détails de l'image, à certaines arrêtes et parfois à du bruit. Nous avons alors dû choisir entre deux approches pour traiter ces faces.

La première consistait à ne garder que le premier niveau de détail, c'est-à-dire chercher à rattacher les points non-attribués aux "grandes" primitives. Pour chaque face non-attribuée, on identifie les deux plans les plus proches. Si l'un est significativement plus proche (par exemple deux fois plus proche), la face est attribuée à cette primitive. Si par contre les deux plans sont à des distances similaires, on attribue les sommets aux deux plans pour qu'ils soient ensuite projetés sur l'intersection (cette démarche est justifiée par le fait qu'un certain nombre de faces non attribuées se situent sur des angles). Cette méthode efface le bruit et conserve les arrêtes et les plans principaux, cependant nous avons choisi de ne pas l'implémenter. Les principales raisons sont la perte totale des zones plus fines parfois importantes (dans notre cas le canon du char Panther et la statue du monument Francis) et les aberrations géométriques que cette méthode peut engendrer si un contrôle fin n'est pas effectué. En effet, puisque les plans sont infinis dans $\mathbb{R}^3$, ils traversent la surface et détecter le plan le plus proche peut causer de grosses incohérences, ainsi que de nombreuses faces qui intersectent la surface (dites *self-intersections*).

La seconde idée, que nous avons exploitée, consistait à effectuer de nouveau le *region growing* sur les faces non affectées mais cette fois avec des paramètres différents, en l'occurrence un nombre de faces minimum plus bas et un écart maximum à la surface plus fin. On peut ainsi trouver des faces correspondant à un niveau de détail plus fin mais qui ne sont pas assimilables aux premières primitives. On peut alors itérer ce procédé jusqu'à ce que toutes nos faces soient associées à une primitive. L'algorithme converge nécessairement puisque le nombre minimum de faces par région décroît jusqu'à 1. Les paramètres varient selon des fonction de type sigmoïdes, les variations s'accélèrent avec le temps pour donner la priorité aux premières primitives et accélérer l'algorithme.

Projection sur les primitives

Maintenant que toutes les faces sont attribuées à une primitive (de degrés de précision variables), nous allons projeter les faces sur le plan correspondant. Nous commençons pour cela par calculer le plan d'une primitive en prenant le meilleur plan au sens des moindres carrés à l'aide d'une fonction du package *Principal component analysis* [11]. On dira

qu'un sommet appartient à une primitive si une des faces qu'il forme appartient à la primitive, un sommet peut donc appartenir à plusieurs primitives (notamment au niveau des arrêtes). On parcourt maintenant les sommets de notre maillage. Si un sommet appartient à une seule primitive on le projette sur celle-ci, si il appartient à deux primitives on le projette sur l'intersection des deux plans (une droite), de même pour trois primitives (un point). Nous ne traitons pas spécifiquement le cas d'intersections plus nombreuses puisque c'est très rare (une vingtaine de points sur des dizaines de milliers) mais on pourrait par exemple choisir de projeter sur le point qui minimise la distance à tous les plans. Ces projections ne garantissent pas d'éviter les *self-intersections* mais notre choix des primitives fait qu'elles sont peu nombreuses et la plupart d'entre elles disparaîtront lors de l'étape de décimation.

Décimation

Jusqu'ici, nous n'avons pas simplifié le maillage; le nombre de faces est le même qu'au début. Nous utilisons un *edge collapse* pour réduire le nombre de faces. Des arêtes sont supprimées itérativement (entraînant la suppression des deux faces adjacentes). On choisit à chaque itération l'arête dont la suppression à le coût le plus faible (distance du maillage avec suppression par rapport au maillage sans suppression) [12].

2.4.2 Algorithme et implémentation

Notre algorithme fonctionne de manière hiérarchique: on détecte des grands plans avec une faible précision, puis, au fur et à mesure, on s'intéresse à des plans de plus en plus petits, jusqu'à ce que toutes les faces du maillage soit assignées à un plan.

Notations de bases

Plus précisément, on considère un maillage $MESH$ contenant N_face_Mesh faces triangulaires, tel que :

$$MESH = (face_i)_{i \in [N\_face\_Mesh]}$$

où $face\_i$ est une face triangulaire du $MESH$.
On va considérer $Nb\_face\_min_i = f(i)$ une suite qui indique le nombre de faces minimale en fonction de i. Typiquement, la fonction f est décroissante. Ce paramètre va indiquer le nombre de faces minimales des plans détectées à l'étape i.
Enfin, on considère un objet de type $vector$: $plane\_map$. Ce $vector$ est définit tel que :

$$plane\_map[face_i] = plan_j$$

En effet, la $plane\_map$ associe à une face, un plan associé, ou -1 si aucun plan n'est associé.
Puis on note $N\_Face\_non\_assigne$ le nombre de faces qui ne sont associé à aucun plans. Autrement dit, $N\_Face\_non\_assigne$ est le cardinal de l'ensemble $\{i | plane\_map[face_i] = -1\}$.

La fonction simplify

Nous avons ensuite définit une fonction $simplify(Nb\_face\_min\_i, plane\_map, MESH)$, qui prend un nombre de faces minimales $Nb\_face\_min\_i$, une $plane\_map$, et un $MESH$. Cette fonction va detecter des plans de $MESH$ composé d'un nombre de faces minimales $Nb\_face\_min\_i$, et modifie par effet de bord $plane\_map$. En sortie de cette fonction, on obtient alors une $plane\_map$ qui associe à chaque face, un plan, ou -1 si aucun plan n'a été détecté.

La fonction projection

Cette fonction $projection(plane\_map, MESH)$ permet de projeter les faces $face_i$ telle que $plane\_map[face_i] \neq -1$, sur les plans associés. Cette fonction modifie $MESH$ par effet de bord.

La fonction collapse

Enfin, la fonction $collapse(N\_Face\_finale, MESH)$ modifie $MESH$ par effet de bord, et permet d'obtenir un maillage avec N_Face_finale, grâce à l'algorithme de $edge\ collapse$ décrit précédemment.

Algorithme et approche hiérarchique

L'idée est la suivante : on crée une $plane\_map$, initialisée à -1 pour toutes les faces. Puis on applique la fonction $simplify$ et $projection$ successivement pour chaque passage de boucle. Enfin, on diminue le nombre de faces minimales pour la détection de plans. Ainsi, l'algorithme converge nécessairement et on s'arrête lorsqu'il n'y a plus de faces non assignée à un plan.

Algorithm 1 Algorithme de simplification personalisé

Result: Approche personnalisée
for *i entre 0 et* N_face_mesh **do**
  $plane\_map[face_i] = -1$
end
$i = 0$
while $N\_Face\_non\_assigne \neq 0$ **do**
  simplify($Nb\_face\_min\_i$, $plane\_map$, $MESH$)
  projection($plane\_map$, $MESH$)
  $i = i + 1$
end
collapse(N_Face_finale, $MESH$)

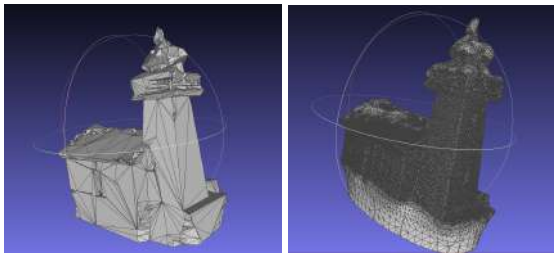

Figure 27: *Résultat obtenu pour le maillage lighthouse : image de départ (à droite) et image resultante (à gauche). On remarque que le nombre de faces a drastiquement diminué*

2.5 Comparaison des différentes approches

2.5.1 Critères quantitatifs

Nous avons testé et introduit un certain nombre d'approches pour simplifier notre maillage. Cependant, il est important d'introduire des critères quantitatifs pour comparer nos approches. L'objectif du projet était de simplifer le maillage tout en conservant sa structure globale et les critères que nous avons retenu sont les suivants :

- **Nombre de faces** : C'est le critère le plus naturel étant donné notre objectif. Le nombre de faces polygonales (en pratique triangulaires dans la plupart des méthodes) est une valeur que l'on va chercher à minimiser. Les maillages initiaux comptent entre 50 et 500 mille faces et notre objectif est de descendre à quelques centaines.

- **Distance de Hausdorff** : Pour mesurer la qualité géométrique de notre maillage simplifié par rapport à un maillage de référence, nous avons choisi d'utiliser la distance de Hausdorff implémentée dans le logiciel `Meshlab`. En pratique, on a besoin que d'une partie de la distance de Hausdorff qui fonctionne en théorie de manière symétrique; la formule utilisée ici pour calculer la distance d'un maillage X à un autre maillage Y s'écrit :

$$\sup_{x \in X} \inf_{y \in Y} \mathrm{d}(x, y) \qquad (1)$$

  Le logiciel sélectionne un certain nombre de points d'un premier maillage (pour nous le maillage de référence noté X) et calcule la distance au second maillage (maillage simplifié noté Y). Dans le cas particulier de la simplification de maillages, les points sélectionnés peuvent être les sommets du maillage de plus grande densité. Cette distance est en réalité calculée localement (par faces) puis moyennée sur l'ensemble de la surface.
  Les résultats peuvent être visualisés en appliquant un filtre coloré au maillage de référence. Par exemple sur la figure 28, on peut observer le maillage de référence de la structure Panther, un maillage très simplifié (150 faces) et la représentation colorée de la distance de Hausdorff. On observe en particulier l'erreur élevée au niveau du canon qui a disparu lors de la simplification.

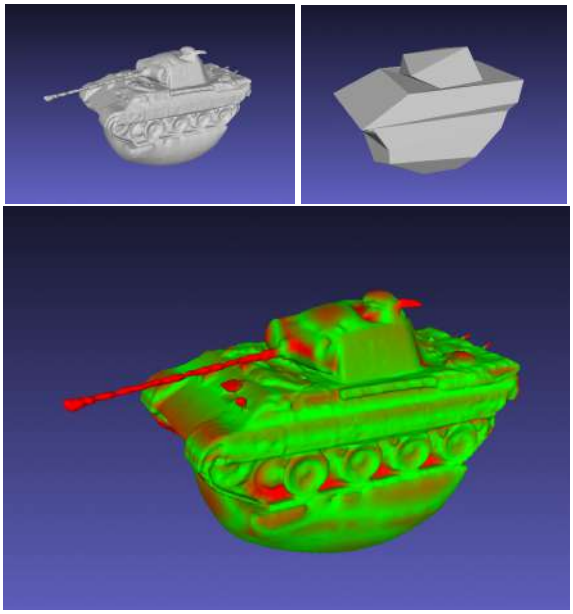

Figure 28: *Visualisation de la distance de Hausdorff. En vert les distance faibles, en rouge les distances élevées*

- **Temps de calcul** : Ce facteur est plus matériel mais revêt en pratique une grande importance pour savoir quel algorithme est applicable dans quel contexte et sur quel type de données. Le temps de calcul peut être lié à la complexité algorithmique des méthodes utilisées ou au volume de mémoire vive nécessaire. Ce dernier point est important à prendre en compte lorsqu'on manipule des structures de plusieurs millions de faces.

Ces trois critères sont ceux qui ont été pris en compte en priorité, nous avons choisi de ne pas nous focaliser sur les autres critères classiques d'évaluation de maillages comme l'isotropie ou la taille des arêtes.

Nous avons également choisi de ne pas interdire strictement les *self-intersections*, seulement de les limiter. Pour certaines applications ultérieures

(impression 3D, réalisation de patrons etc...), ce critère peut néanmoins avoir de l'importance.

Enfin, il est important de remarquer que les critères retenus ne reflètent pas exactement l'objectif. En effet nous voulons préserver la "structure de l'objet" ce qui comprend notamment les principales surfaces planes et les arêtes anguleuses. Il n'existe pas de mesure mathématique rigoureuse permettant d'évaluer ces critères mais il sont néanmoins importants à prendre en compte, ne serait-ce que qualitativement, dans l'appréciation de la méthode.

2.5.2 Comparaisons des méthodes de remaillage

Comme expliqué dans la section dédiée, la méthode de *slicing* n'est pas viable dans notre cas à cause de l'explosion du temps de calcul dès qu'on augmente le nombre de plans détectés. Cette méthode n'a donc pas été comparée quantitativement aux autres et a simplement été écartée.

Les trois méthodes comparées sur le graphe de la figure 29 sont le VSA, un simple *edge collapse* depuis le maillage d'origine, et notre méthode. Les résultats utilisés sont ceux pour le dataset Panther.

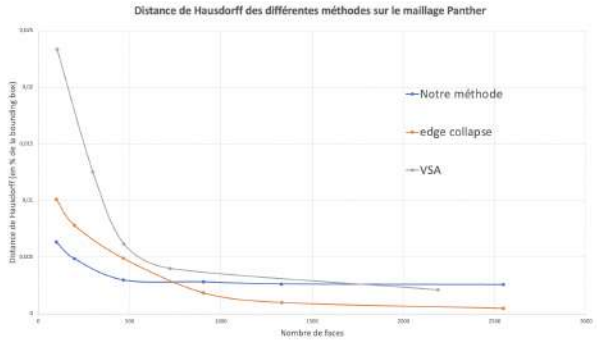

Figure 29: *Comparaison d'un edge collapse simple, du VSA et de notre méthode*

Pour un nombre élevé de faces, notre méthode fournit les plus mauvais résultats tandis que *edge collapse* produit les meilleurs maillages. C'est logique puisque *edge collapse* choisit successivement les opérations les moins coûteuses. Les résultats du *edge collapse* simple sont à nuancer légèrement, la distance de Hausdorff est faible puisque c'est celle qui est minimisée localement à chaque itération, cependant les angles et les surfaces planes sont moins bien conservés.

Le VSA quant à lui fournit les moins bons résultats dès qu'on abaisse le nombre de faces. La nature constructive de l'algorithme ne lui permet pas d'atteindre d'aussi bons résultats que les méthodes de décimation. Notons cependant que cet algorithme conserve assez bien la notion subjective de "structure" dont on a parlé ci-dessus bien qu'elle garantisse peu de propriétés topologiques (*2-manifold, watertight, self-intersections*).

Notre méthode fournit les meilleurs résultats sur un faible nombre de faces. La courbe de distance comprend un plateau au cours duquel les arêtes décimées appartiennent à des faces : puisqu'on a déjà projeté, le coût est nul. Ensuite la courbe suit la même tendance que pour un *edge collapse* simple, en suivant la trajectoire de moindre pente à chaque itération.

Ce que l'on espère avec notre méthode (et qui fonctionne pour des objets structurés comme ceux étudiés), c'est que projeter sur les composantes planaires principales permette d'atteindre un état de meilleur qualité que de simplement suivre la direction de moindre pente. Le coût est élevé lorsqu'on garde beaucoup de faces mais les résultats s'améliorent lorsque celui ci s'approche du nombre de plans (en ordre de grandeur). Mais le choix du nombre de plans n'est pas anodin et c'est l'objet du graphe de la figure 30.

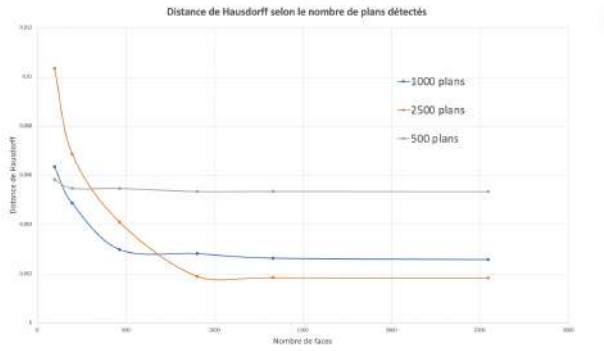

Figure 30: *Comparaison des résultats de notre méthode en fonction du nombre de plans détectés*

On observe sur le graphe 30 que si on augmente le nombre de plans détectés (en jouant sur des paramètres tels que le nombre minimal de voisins, la distance maximale tolérée au plan, etc.), le plateau atteint une meilleure qualité (distance plus basse). C'est logique puisqu'on s'est autorisé plus de plans pour approximer l'objet. Mais en même temps, le plateau se décale vers la droite, c'est à dire que les suppressions d'arêtes de coût nul seront moins nombreuses (c'est cohérent

également puisqu'il y a plus d'arêtes qui délimitent les plans).

Il faut donc trouver le bon nombre de plans à viser selon la structure d'origine et le nombre de faces désiré en sortie. Notre pipeline détermine des paramètres automatiquement mais les meilleurs résultats sont atteints en ajustant ces paramètres manuellement.

3 Conclusion

Notre projet consistait en la reconstruction 3D de scènes à caractère structuré, c'est-à-dire avec une grande géométrie planaire, en des maillages optimisés. L'enjeu était donc double. Il s'agissait à la fois de créer une pipeline commençant à la prise de photos du sujet, et aboutissant à l'obtention d'un maillage satisfaisant, mais aussi de déterminer en quoi consistait ce "satisfaisant". Le maillage devait être optimisé, c'est-à-dire simple, mais tout en conservant l'aspect initial de notre structure. Il fallait trouver un juste milieu entre le simple et le trop simple.

Pour la toute première partie de la pipeline, consistant en la création du nuage de points, les considérations de simplicité n'entraient pas en compte. Seule la qualité et la précision du nuage compte. C'est ce pourquoi nous nous sommes tournés vers les algorithmes de la pipeline COLMAP, offrant un meilleur positionnement des caméras que OpenMVS. Pour la phase suivante, en revanche, où l'on cherche à obtenir le maillage qui sera manipulé dans toute la suite, les considérations d'optimisation entrent en compte. Le nuage est nettoyé grossièrement de ses outliers, qui perturbent l'interpolation, lissé pour une bonne conservation des angles, et échantillonné afin de n'avoir pas à traiter trop d'informations. Ces étapes permettent d'obtenir un nuage plus affiné, traitable en un temps raisonnable, et donnant un maillage plus léger après application de l'algorithme de Poisson. Un simple RANSAC sur un mesh réalisé sans ces simplifications préalables demandait des dizaines d'heures de calcul sur nos ordinateurs, rendant impossible notre progression.

Une fois cette base de maillages obtenue, le temps de traitement n'était plus une contrainte d'optimisation de première importance. Nos différents algorithmes étaient discriminés en fonction du nombre de faces produites, et de la distance de Hausdorff entre le maillage de sortie et le celui de référence. Trois techniques s'opposent : un simple Edge Collapse, le Slicing et le VSA, auxquelles s'ajoutent notre propre approche. Déterminer une bonne méthode de remaillage caractérisait le coeur de notre projet. Si le Slicing était idéal pour le nombre de faces en sortie, et la préservation de la structure, la résolution de sa dernière étape était d'une trop grande complexité. L'edge collapse, quant à lui, ne permettait pas une grande préservation des surfaces planes de la structure. Enfin le VSA fournissait de plus mauvais résultats en distance de Hausdorff que notre approche pour un faible nombre de faces à l'origine, et inversement pour un grand nombre de faces.

Nous pouvons donc nous enorgueillir d'une pipeline donnant des résultats comparables à l'état de l'art selon les critères déterminés par nos soins. Et une telle méthode présente un grand intérêt à l'heure actuelle, en city modelling par exemple.

Remerciements

Nous tenons à remercier Pierre Alliez et Florent Lafarge de l'Inria pour avoir proposé ce sujet, et plus particulièrment Florent Lafrage pour son accompagnement régulier tout au long du projet ainsi que ses précieux conseils. Nous remercions également l'équipe du département *Ingénierie Mathématique et Informatique* de l'école des Ponts, Pascal Monasse, Eric Duceau, Sandrine Guillerm et Aurélie Mortier pour les conseils avisés.

Nous tenons également à remercier tous les développeurs qui contribuent à la bibliothèque CGAL.

Bibliographie

[1] Liangliang Nan. "Polygonal Surface Reconstruction". In: *CGAL User and Reference Manual*. 5.2.1. CGAL Editorial Board, 2021. URL: https://doc.cgal.org/5.2.1/Manual/packages.html#PkgPolygonalSurfaceReconstruction.

[2] Liangliang Nan and Peter Wonka. "PolyFit: Polygonal Surface Reconstruction from Point Clouds". In: *2017 IEEE International Conference on Computer Vision (ICCV)*. 2017, pp. 2372–2380. DOI: 10.1109/ICCV.2017.258.

[3] Jean-Philippe Bauchet and Florent Lafarge. “Kinetic Shape Reconstruction”. In: *ACM Transactions on Graphics* (2020). DOI: 10.1145/3376918. URL: https://hal.inria.fr/hal-02924409.

[4] Simone Bianco, Gianluigi Ciocca, and Davide Marelli. “Evaluating the Performance of Structure from Motion Pipelines”. In: *Journal of Imaging* 4.8 (2018). ISSN: 2313-433X. DOI: 10.3390/jimaging4080098. URL: https://www.mdpi.com/2313-433X/4/8/98.

[5] Arno Knapitsch et al. “Tanks and Temples: Benchmarking Large-Scale Scene Reconstruction”. In: *ACM Trans. Graph.* 36.4 (July 2017). ISSN: 0730-0301. DOI: 10.1145/3072959.3073599. URL: https://doi.org/10.1145/3072959.3073599.

[6] Pierre Alliez et al. “Point Set Processing”. In: *CGAL User and Reference Manual*. 5.2.1. CGAL Editorial Board, 2021. URL: https://doc.cgal.org/5.2.1/Manual/packages.html#PkgPointSetProcessing3.

[7] Michael Kazhdan, Matthew Bolitho, and Hugues Hoppe. “Poisson Surface Reconstruction”. In: *Symposium on Geometry Processing*. Ed. by Alla Sheffer and Konrad Polthier. The Eurographics Association, 2006. ISBN: 3-905673-24-X. DOI: 10.2312/SGP/SGP06/061-070.

[8] Sven Oesau et al. “Shape Detection”. In: *CGAL User and Reference Manual*. 5.2.1. CGAL Editorial Board, 2021. URL: https://doc.cgal.org/5.2.1/Manual/packages.html#PkgShapeDetection.

[9] Pierre Alliez, David Cohen-Steiner, and Lingjie Zhu. “Triangulated Surface Mesh Approximation”. In: *CGAL User and Reference Manual*. 5.2.1. CGAL Editorial Board, 2021. URL: https://doc.cgal.org/5.2.1/Manual/packages.html#PkgTSMA.

[10] David Cohen-Steiner, Pierre Alliez, and Mathieu Desbrun. *Variational Shape Approximation*. Research Report RR-5371. INRIA, Nov. 2004, p. 29. URL: https://hal.inria.fr/inria-00070632.

[11] Pierre Alliez, Sylvain Pion, and Ankit Gupta. “Principal Component Analysis”. In: *CGAL User and Reference Manual*. 5.2.1. CGAL Editorial Board, 2021. URL: https://doc.cgal.org/5.2.1/Manual/packages.html#PkgPrincipalComponentAnalysisD.

[12] Fernando Cacciola, Mael Rouxel-Labbé, and Baskın Şenbaşlar. “Triangulated Surface Mesh Simplification”. In: *CGAL User and Reference Manual*. 5.2.1. CGAL Editorial Board, 2021. URL: https://doc.cgal.org/5.2.1/Manual/packages.html#PkgSurfaceMeshSimplification.