Reconnaissance de bâtiments à partir de nuages de points 3D

Justine Basselin

UNIVERSITÉ DE LORRAINE

École doctorale IAEM Lorraine

Reconnaissance de bâtiments à partir de nuages de points 3D

THÈSE

présentée et soutenue publiquement le 14 décembre 2022

pour l'obtention du

Doctorat de l'Université de Lorraine

(mention informatique)

par

Justine Basselin

Thèse co-encadrée par M. Dmitry Sokolov, M. Nicolas Ray et M. Hervé Barthélémy

Composition du jury

Président :	M. Emmanuel Jeandel	Université de Lorraine - LORIA
Rapporteurs :	Mme. Hyewon Seo	Université de Strasbourg - ICUBE
	M. Bruno Vallet	Université Gustave Eiffel - LASTIG
Examinateur :	M. Emmanuel Jeandel	Université de Lorraine - LORIA
Invité :	M. Frédéric Thomas	RhinoTerrain

Laboratoire Lorrain de Recherche en Informatique et ses Applications — UMR 7503

Remerciements

Tout d'abord, je tiens à remercier mes directeurs de thèse, qui ont rendu ces trois années passées dans l'équipe Pixel des plus enrichissantes. Merci à Nicolas et Dmitry pour leur temps, leur patience et leur disponibilité, ainsi que pour leurs précieux conseils durant les pauses café quotidiennes. Je tiens également à remercier les membres du jury qui ont accepté d'évaluer mes travaux. Je suis très honorée de votre participation et vous suis très reconnaissante de votre temps et de votre expertise. Merci à Hyewon Seo et Bruno Vallet pour leur rapport sur mon manuscrit, ainsi qu'à Emmanuel Jeandel pour avoir pris part au jury de soutenance. Je tiens à exprimer ma reconnaissance envers toute l'équipe RhinoTerrain, en particulier envers Fred pour son encadrement tout au long de cette thèse, et Guillaume pour sa communication originale et son lancé précis de paquets de mouchoirs. Je tiens également à remercier chaleureusement Hervé, même s'il ne s'est impliqué que sur une partie du projet. Je lui suis très reconnaissante pour les discussions enrichissantes que nous avons eues ensemble. Je tiens à remercier chaleureusement les membres de l'équipe Pixel avec qui j'ai eu le plaisir de partager ces dernières années : David, Yoann, François, Laurent, David et Dobrina. Je tiens également à remercier tout particulièrement mes co-bureaux, Etienne aka le maître du jugement et Guillaume le meme lord, pour leur soutien tout au long de cette période. Je souhaite remercier également toutes les personnes que j'ai pu rencontrer durant ma thèse, notamment Léo, Juliette, Jimmy. Je tiens également à remercier tous les stagiaires que j'ai eu la chance de côtoyer durant ma thèse. Je remercie également toutes les personnes qui ont contribué à rendre mon séjour à Nancy mémorable, en particulier la bande des loulous (Raphy, Val, Quentin, Ariane). Mais également à distance avec la bande des gendicapées (Matthou, Gary, Alex, Romain, Emily, Yoni et Alex). Je tiens à remercier Lucie et Claire pour m'accompagner depuis quelques années déjà et a qui je dois beaucoup. Je tiens à remercier une nouvelle fois Guillaume, pour m'avoir accompagné et soutenu au long de cette thèse. Pour finir, un très grand merci à ma mère, à mes tantes et à mes sœurs pour leur soutien inconditionnel tout au long de ces longues années d'études.

Table des matières

Introduction

Au début du XV^{e} siècle, un concours public invitait tous les artistes de Toscane à soumettre une proposition architecturale de dôme pour la cathédrale de Santa Maria del Fiore à Florence, dont la construction avait été laissée en suspens durant presque cinquante années. Pour participer, les compétiteurs devaient produire une miniature ou un plan, permettant au jury de mieux assimiler la solution proposée. Seize artistes répondirent au concours en présentant des maquettes en bois ou en briques d'argiles. *In fine*, le projet de Filippo Brunelleschi fut retenu, car il prévoyait l'édification du dôme sans utiliser de cintre [1] ce qui permettait de contourner l'obstacle du manque de bois qui prohibait jusqu'alors sa construction. Toutefois, sa solution était si singulière et inhabituelle qu'on lui commanda une seconde maquette pour en vérifier la faisabilité. Après la réalisation de cette seconde maquette, son projet fut validé et la construction put débuter.

Plus récemment, à la fin des années 1940, John Reber proposait de modifier complètement la baie de San Francisco pour créer un réservoir côtier destiné à fournir de l'eau douce pour l'irrigation des fermes. En 1953, le *US Army Corps of Engineers* recommanda la création d'un modèle réduit pour mesurer les conséquences d'une telle construction, notamment pour tester la résistance aux marées des barrages qui étaient nécessaires au projet. La maquette fut réalisée, et coûta près de 2,5 millions de dollars pour une superficie d'environ 6000 m^2. Cet investissement ne fut pas vain, puisque diverses analyses furent effectuées et montrèrent que les barrages ne créeraient pas de lacs d'eau douce, mais des bassins d'évaporation, et que les marées généreraient des courants dangereux voire des inondations. Cette maquette suffit à montrer que le projet était désastreux, tant sur le plan économique que sur le plan urbain, et grâce à sa réalisation au préalable, le projet Reber put être évité.

Depuis toujours, les maquettes constituent un outil idéal pour étudier des projets encore hypothétiques en permettant leur visualisation. Elles sont une expression claire et concrète d'une idée dont l'aperçu peut être partagé et débattu. Avec l'évolution des sciences et technologies, les maquettes sont rapidement devenues des outils d'expérimentation permettant de démontrer la faisabilité de projets urbains. Aujourd'hui, grâce aux avancées de l'informatique, il est possible de réaliser ces expériences pour un coût temporel et financier moindre en simulant des phénomènes physiques sur des représentations numériques en trois dimensions des domaines étudiés.

1. Le cintre est une charpente de soutien provisoire. C'est une structure sur laquelle les pierres d'un arc ou d'une voûte sont posées durant la construction. Tant que la clé de voûte n'est pas insérée, l'arc n'a aucune résistance. Les pierres ont donc besoin d'un support pour être maintenues dans leur position [Amsellem et al., 2015].

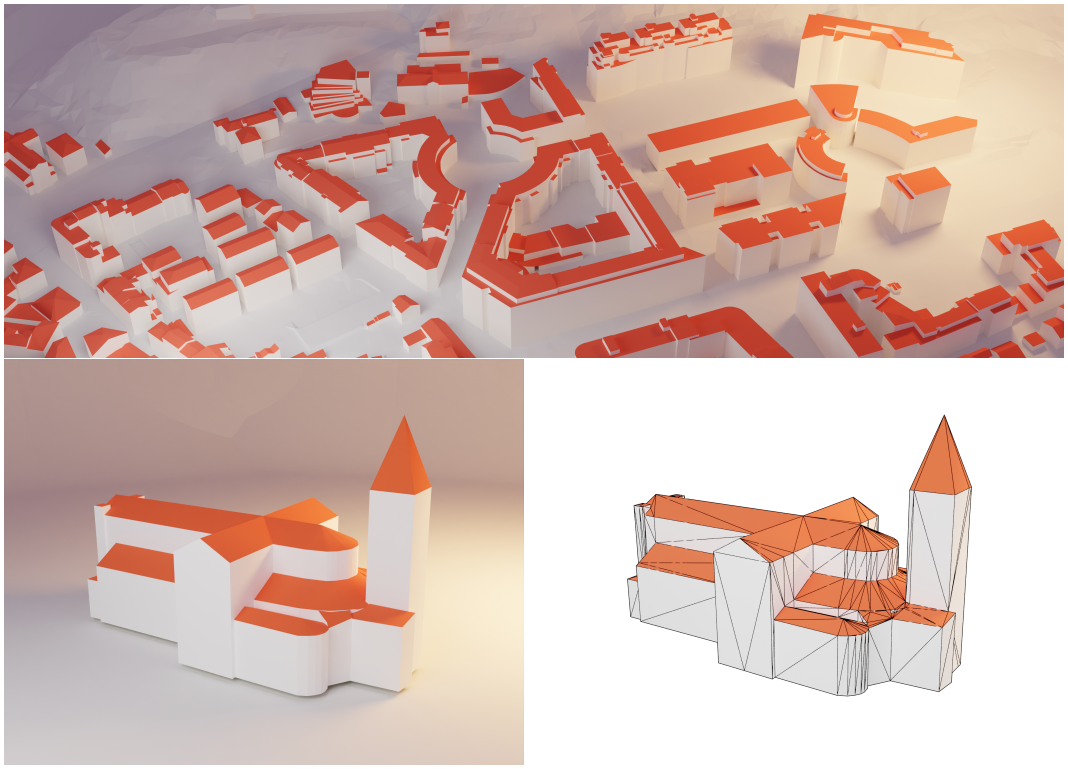


FIGURE 1 – **En haut** : illustration d'une partie de la maquette numérique urbaine du 9e arrondissement de la ville de Lyon [métropole, 2022a]. **En bas** : à gauche, rendu d'un bâtiment appartenant à la même maquette et, à droite, son maillage associé.

Dans le cas de la modélisation numérique de villes, ces représentations sont nommées maquettes ou modèles numériques urbains et, de nos jours, de plus en plus de municipalités souhaitent s'en doter (voir illustration en figure 1). Ces modèles leur permettent tout d'abord de réaliser des simulations de projet urbain. En effet, en retirant ou en ajoutant des blocs à leur maquette, les municipalités peuvent observer l'impact d'un nouveau projet sur leur territoire et en contrôler l'intégration dans le paysage (par exemple, en vérifiant la hauteur des volumes bâtis). Selon Christelle Gibon [2], responsable des projets innovants pour la métropole de Rennes, l'une des forces de ces maquettes est également leur capacité à renouer le dialogue lors de négociations citoyennes autour d'un projet urbain, où il est souvent difficile d'obtenir des retours. De plus, très récemment, quelques municipalités se tournent vers ces maquettes dans le contexte de la résilience urbaine, car elles permettent d'obtenir des informations inaccessibles au moyen d'un plan traditionnel en deux dimensions. Ces informations sont nécessaires à la réalisation de simulations environnementales, comme la prévision des risques d'inondation, ou encore l'estimation d'ombre portée sur des bâtiments, l'étude de la propagation du bruit dû à la circulation dans un quartier, mais aussi la prédiction de l'exposition solaire reçue par une toiture pour évaluer s'il est intéressant d'y installer un panneau solaire.

Mais qu'est-ce qu'un modèle numérique urbain ? Au même titre que les cartes,

2. Cette information est issue d'une interview que nous a accordée Christelle Gibon réalisée en décembre 2021.

les modèles numériques urbains sont une approximation du monde réel (§1).

Ils restituent avec une certaine fidélité leur homologue réel :

— D'un point de vue géographique. Les modèles sont géoréférencés : ils restituent avec une précision centimétrique la position de l'objet réel dans des repères dédiés. Pour ce faire, les modèles sont souvent reconstruits à partir de numérisations obtenues grâce à la télédétection laser (ou LiDAR §2), ou par photogrammétrie.

— D'un point de vue géométrique. Les caractéristiques sont modélisées à un certain niveau de détail où les éléments constituant le modèle peuvent être simplifiés ou omis. Généralement, pour un bâtiment, la forme globale est restituée : sa toiture, ses murs, parfois ses ouvertures, mais il n'est pas utile, par exemple, de représenter ses tuiles. Les différents objets composant un modèle numérique urbain sont représentés par un maillage, c'est-à-dire une discrétisation structurée de l'objet (§3).

Aujourd'hui, la génération automatique de maillage à partir de relevés LiDAR demeure une question ouverte. Pour cause, générer des modèles numériques urbains n'est pas sans difficulté. Il faut, à partir d'acquisitions, produire un modèle géométrique structuré conciliant proximité aux données terrains et simplicité tout en restant compatible avec les exigences de la simulation numérique. Si les technologies de numérisation, telles que la télédétection par balayage laser, ont fait de nombreux progrès ces dernières décennies, l'informatique en général et le domaine du traitement de la géométrie numérique en particulier sont très en retard par rapport aux avancées de l'électronique. Malgré le haut degré de maturité atteint par les techniques de numérisation, les solutions efficaces pour effectuer le traitement de données bruitées et de densité variable sont rares et peu adaptées à la complexité et à la masse des données (§4).

Des outils d'assistance semi-interactifs émergent sur le marché, mais ils ne répondent pas suffisamment aux besoins, tant en termes de précision que d'efficacité. Aujourd'hui, la création d'un modèle numérique à partir de tels relevés est longue, fastidieuse et toujours en partie manuelle [Agugiaro, 2014, Stoter et al., 2016]. Ce manuscrit, fruit du partenariat entre le laboratoire LORIA et la société RhinoTerrain, vise à présenter des méthodes facilitant la génération de maquettes numériques urbaines pour la simulation numérique.

Organisation du manuscrit

Dans la suite de cette introduction, nous présentons les notions nécessaires à la compréhension du manuscrit en explorant les tenants et aboutissants de la génération de maquettes numériques urbaines (§1) : des acquisitions nécessaires à sa génération (§2) permettant d'obtenir des nuages de points, à ses utilisations. Nous mettons en évidence différents éléments de géométrie algorithmique (§3) qui seront utilisés au cours de ce manuscrit. Enfin, nous faisons un état de l'art des méthodes de reconstruction de maquettes numériques urbaines à partir de nuages de points issus d'acquisitions LiDAR aéroporté (§4). Nous invitons le lecteur à lire ces sections avant d'entreprendre la lecture des chapitres suivants.

Partie I : Reconstruction de modèle numérique urbain

Dans cette partie, nous abordons les différentes contributions de cette thèse à la reconstruction de modèle numérique urbain. La littérature de la reconstruction de tel modèle se découpant en deux grandes familles de méthodes, nous avons exploré les méthodes dites basées sur les données dans le chapitre 1. Ensuite, nous présentons une seconde méthode dans le chapitre 2, dite basée sur les modèles, permettant d'ajuster des surfaces paramétriques définies par nos soins dans une bibliothèque de modèles de toitures, à des nuages de points bruités ou lacunaires. Cette approche variationnelle permet l'ajustement de la surface au nuage de points. L'originalité de cette approche provient de l'utilisation d'un diagramme de Voronoï restreint à la surface, permettant le calcul d'une fonction de distance entre la surface et le nuage de points sous forme close.

Partie II : Accélération et transfert industriel

Dans une seconde partie, nous abordons quelques problématiques soulevées par la méthode d'ajustement de surface. Tout d'abord, nous nous intéressons dans le chapitre 3 à l'accélération du calcul de diagramme de Voronoï dans un cas plus général. Nous proposons une solution permettant de calculer dans un temps aujourd'hui record des diagrammes de Voronoï et des diagrammes de puissance sur carte graphique. Dans le chapitre 4, nous proposons une méthode plus rapide d'ajustement de surface à un nuage de points en réalisant une approximation de la fonction de distance définie dans le chapitre 2. Enfin, ce manuscrit conclut sur la présentation du pipeline de manipulation de nuage de points pour la génération de maquette numérique urbaine que nous proposons aux utilisateurs du logiciel Rhinocéros, présenté dans le chapitre 5. Pour cela, nous présentons quelques méthodes utiles, mise en place au sein d'un plugin du logiciel Rhinocéros avec lequel travaille l'entreprise RhinoTerrain pour la reconstruction de maquette urbaine.

Publications associées à la thèse

- (chapitre 2) Une méthode de reconstruction de surface de toiture basée sur les données à partir de nuage de points issus de relevés LiDAR dans le journal WSCG.
  Dobrina Boltcheva, Justine Basselin, Clément Poull, Hervé Barthélemy et Dmitry Sokolov "Topological-based roof modeling from 3D point clouds" WSCG 2020.
- (chapitre 3) Une méthode d'ajustement de surface paramètriques à des nuage de points issus de relevés LiDAR dans le journal Computer-Aided Design.
  Guillaume Coiffier, Justine Basselin, Nicolas Ray et Dmitry Sokolov "Parametric Surface Fitting on Airborne LiDAR Point Clouds for Building Reconstruction" Computer-Aided Design, 2021.
- (chapitre 4) Une méthode de calcul d'intégrale sur diagramme de puissance sur le GPU dans le journal Computer Graphics Forum.
  Justine Basselin, Laurent Alonso, Nicolas Ray, Dmitry Sokolov, Sylvain Lefebvre et Bruno Lévy "Restricted power diagrams on the GPU" Computer Graphics Forum, 2021.

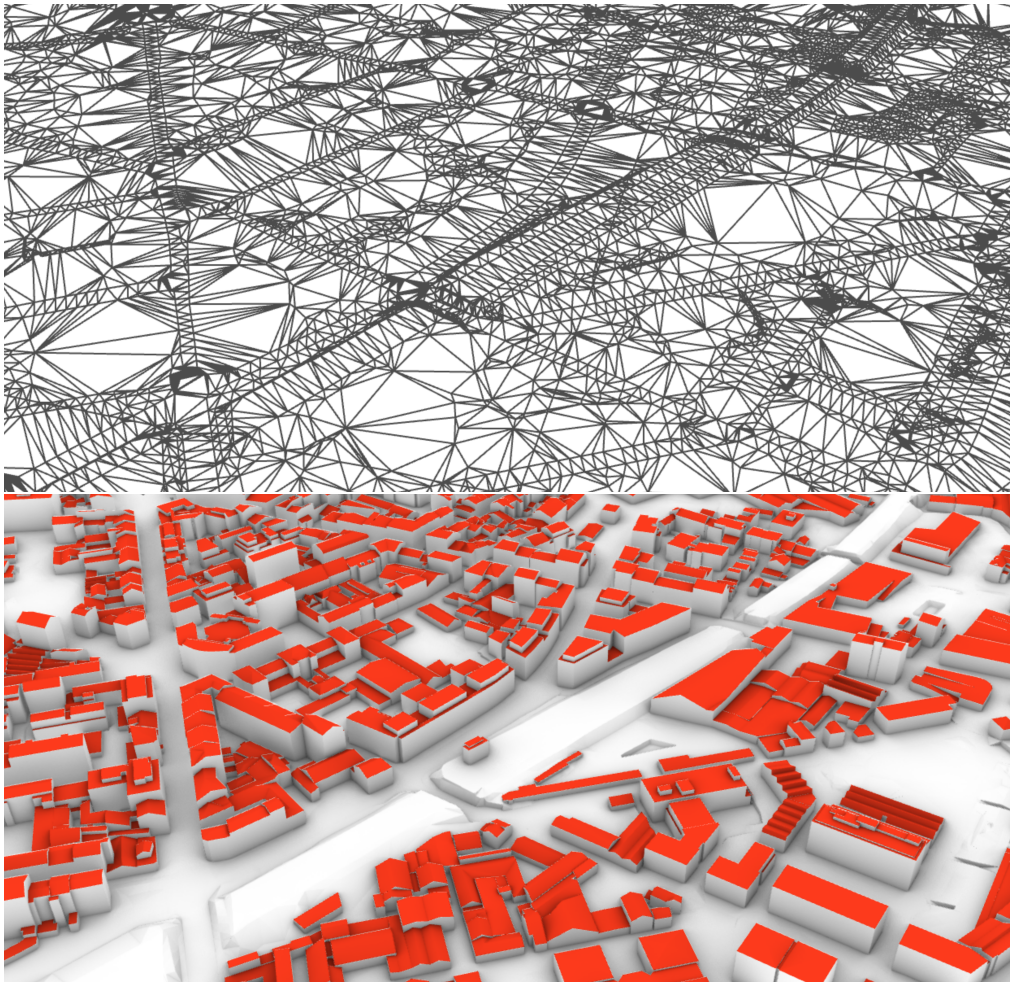


FIGURE 2 – Illustration d'une maquette numérique urbaine du 9e arrondissement de la ville de Lyon [3] modélisé à l'aide des logiciels RhinoCapture et RhinoCity de l'entreprise RhinoTerrain. **En haut :** maillage du terrain (MNT), **en bas :** maquette numérique urbaine, contenant le MNT ainsi que ses bâtiments.

1 Maquette numérique urbaine

Il existe différentes définitions pour les modèles numériques urbains [Biljecki, 2017], nous présentons ici sa définition la plus répandue, qui est celle proposée par l'Open Geospatial Consortium (OGC).

1.1 Qu'est-ce qu'une maquette numérique urbaine ?

Une maquette numérique urbaine (MNU), ou modèle numérique urbain, est une approximation géométrique tridimensionnelle du monde réel : elle reproduit, avec un certain niveau de détail, les éléments de son homologue réel (voir figure 2).

Les éléments urbains courants et l'environnement global avoisinant sont restitués grâce à des maillages. Ces maillages représentent le plus souvent : les bâtiments,

3. Les données sont issues de la page web de la Métropole du Grand Lyon [métropole, 2022a].

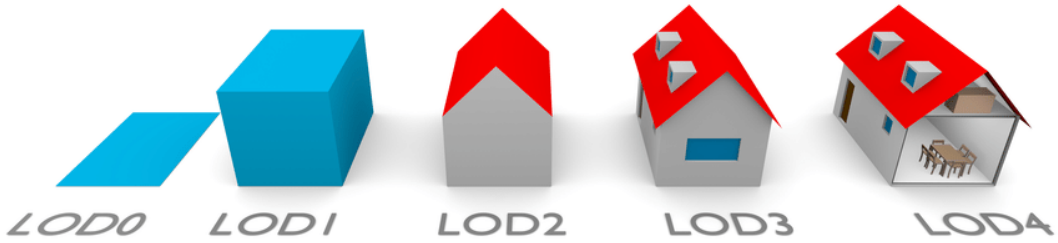


FIGURE 3 – Les cinq LODs de la norme CityGML 2.0. Chaque niveau de détail indique la complexité ainsi que la proximité du modèle à la réalité terrain. Image issue de la thèse de Filip Biljecki [Biljecki, 2017].

les routes, la végétation, les voies ferrées, les rivières, les ponts, que nous pouvons observer dans un quartier, une ville ou encore une métropole entière. Un modèle numérique urbain est dit géoréférencé, car chaque élément de la maquette possède des coordonnées géographiques. C'est l'un des fondements des systèmes d'informations géographiques (SIG), dont le modèle numérique urbain est une partie intégrante.

Outre le stockage de données purement géométriques, certains modèles conservent aussi des informations sémantiques sur les différents objets de la scène 3D, tels que des noms, des identifiants, des usages, *etc.* Ils permettent également de contenir des relations hiérarchiques ou topologiques entre les différents objets. Le maintien et le développement d'une grande pluralité d'informations quantifient l'utilité d'une maquette numérique urbaine. Une telle quantité d'informations engendre des problèmes en matière de stockage ou encore de synchronisation de données. Différents formats de données ont été mis en place pour formaliser cette masse de données, tels que le très répandu format CityGML [Gröger et al., 2012] défini par l'Open Geospatial Consortium (OGC), depuis peu le CityJson [Ledoux et al., 2019], ou encore le format de base de données 3dCityDB.

1.2 Niveau de détail

Le niveau de détail d'un modèle numérique urbain, c'est-à-dire le degré de proximité entre le modèle virtuel et son homologue réel, se nomme LOD pour *Level Of Details* (voir figure 3). Le LOD varie suivant l'application à laquelle la maquette se destine, la précision de l'acquisition qui a permis de la produire et les données disponibles. En effet, chaque niveau de détails possède des avantages et des inconvénients. Par exemple, une grande précision impliquera des données de taille conséquentes plus difficiles à stocker et à maintenir.

Le niveau le plus simple est le niveau LOD0 qui est essentiellement constitué d'une surface représentant le terrain (voir figure 2). Cette surface est nommée modèle numérique de terrain (MNT). Le niveau LOD1 contient des bâtiments représentés par des prismes dotés de toits plats. Le LOD2 présente des surfaces de toitures et des objets de végétation. Le LOD3 désigne des modèles architecturaux avec des structures de mur et de toit détaillées, des balcons, de la végétation basse. Enfin, LOD4 complète ajoute au LOD3 des structures situées à l'intérieur des bâtiments. Par exemple, les bâtiments sont composés de pièces, de portes intérieures, d'escaliers et de meubles.

Récemment, un raffinement de ces niveaux de détails a été présenté [Biljecki et al.,

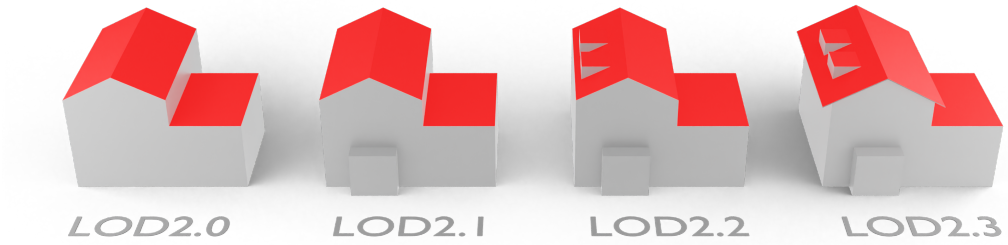


FIGURE 4 – Exemple de modèles de bâtiments LOD2 raffinés de la nouvelle spécification définie par Biljecki et al. [Biljecki et al., 2016c].

2016c]. Le but de cette nouvelle spécification est d'améliorer la définition géométrique des éléments contenus dans un MNU en proposant 16 niveaux de détails différents, mais surtout de lever les imprécisions de la norme définie par l'OGC.

Dans le cadre de cette thèse, nous nous concentrons sur la reconstruction de maquette numérique urbaine au niveau de détail LOD2 comme défini par l'Open Geospatial Consortium (OGC). Ce choix sera plus amplement discuté dans la sous-section 3.2. Pour le niveau de détail LOD2, Biljecki et al. spécifient quatre niveaux de détails pour les modèles LOD2 :

LOD2.0. Les modèles de ce niveau sont représentés par des prismes dotés d'une toiture. ils peuvent également être dotés de garages si ceux-ci mesurent plus de 4 mètres de côté et au moins 10 m^2.

LOD2.1. Ce niveau ajoute au niveau de détail précédent les extensions de bâtiments (si celles-ci mesurent plus de 2 mètres de côtés et 2 m^2). L'intérêt de ce niveau de détail est de capturer la géométrie des murs du bâtiment, y compris les renfoncements.

LOD2.2. Ce niveau est similaire au niveau LOD2.1, à la différence que des superstructures telles que des lucarnes ou de larges cheminées sont ajoutées au modèle. Ce niveau de détail est bénéfique aux simulations d'ensoleillement, car la géométrie de la toiture est entièrement restituée.

LOD2.3. Ce niveau nécessite la modélisation des débords des toitures si leur longueur est supérieure à 20 centimètres. Cette représentation peut avoir un intérêt si la simulation demande la géométrie exacte des murs, comme le calcul du volume d'un édifice.

Dans ce manuscrit, nous souhaitons reconstruire des maquettes numériques urbaines au niveau de détail LOD2.2. Pour les obtenir, il est nécessaire de se baser sur des acquisitions géoréférencées de terrain. Ce point est abordé dans la section suivante.

2 Numérisation par acquisition LiDAR aéroporté

Les systèmes d'acquisition par balayage laser se sont développés de manière fulgurante ces dernières décennies et se sont démocratisés durant le milieu des années

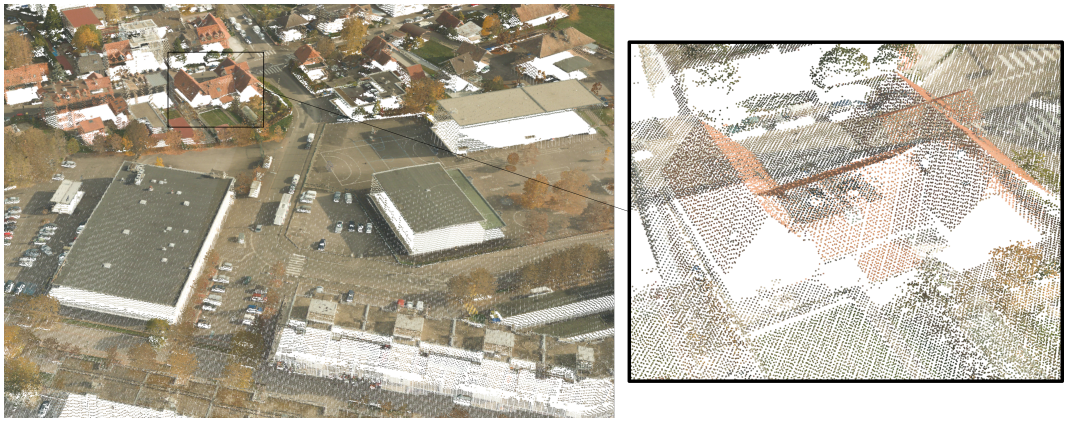


FIGURE 5 – Nuage de points obtenu par acquisition LiDAR aéroporté d'une partie de la banlieue de Strasbourg. Données issues de la page web de l'OpenData de Strasbourg [Strasbourg, 2016].

1990. Cette technologie est le fruit d'un long travail de développement datant du milieu du XX^{e} siècle. À cet égard, il est intéressant de souligner que la NASA a joué un rôle pionnier dans son développement [Petrie and Toth, 2018]. Notamment, grâce à ses activités de cartographie en Arctique datant du début des années 1960 ou encore à ses expéditions sur la Lune, durant les missions Apollo 15 à 17, où des scanners étaient embarqués à bord des vaisseaux.

Aujourd'hui la télédétection par laser ou LiDAR permet la production de nuages de points d'une résolution spatiale jusqu'à maintenant jamais atteinte par des relevés terrestres traditionnels (tachéomètres) en seulement quelques heures (voir figure 5). Ces différents avantages poussent les collectivités à produire de plus en plus de ces données. C'est ainsi que le LiDAR est rapidement devenu le moyen d'acquisition le plus répandu dans le domaine de la géo-information [Shan and Toth, 2018].

Consciente du récent développement des technologies dites de télédétection et des différents avantages offerts par les capteurs LiDAR, la société RhinoTerrain qui propose déjà des outils d'aide à la reconstruction de maquettes numériques urbaines à partir d'orthophotographies a choisi de se tourner vers ce moyen d'acquisition. RhinoTerrain souhaite proposer aux collectivités des solutions permettant la reconstruction de MNU à partir de ces données et cette thèse constitue les prémices d'un nouveau projet de traitement de données issues d'acquisitions par scanners laser.

2.1 Les principes de base de la télémétrie laser

Le LiDAR, acronyme de l'expression anglaise *Light Detection And Ranging*, soit en français *détection et estimation de la distance par la lumière*, permet de mesurer la distance d'un émetteur A à une cible B avec une grande précision. Pour cela, un signal est émis en direction d'un objet (par exemple un bâtiment, de la végétation, *etc.*) et réfléchi par celui-ci dans toutes les directions ou absorbé (selon la nature du point de contact). Une fraction du signal est renvoyée dans la direction du récepteur.

Ce signal de retour est une version atténuée et retardée du signal émis ; l'analyse de ses caractéristiques permet de déterminer sa distance à l'émetteur. Cette distance peut être estimée par la méthode dite du *Time of Flight (TOF)* [Shan and Toth,

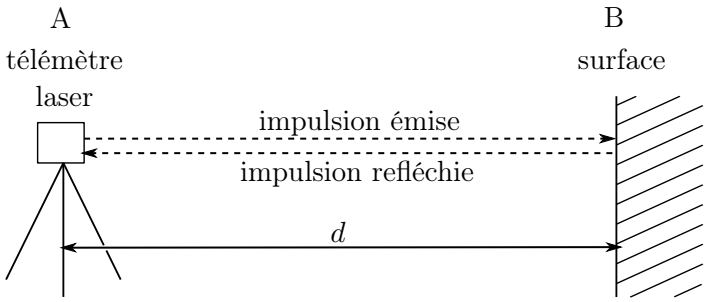


FIGURE 6 – Fonctionnement d'un télémètre laser utilisant le principe du *Time of Flight (TOF)*.

2018]. Elle consiste à envoyer une impulsion laser en direction d'un objet (B), puis de mesurer le temps d'aller-retour entre l'instant de l'émission de cette impulsion laser en un point A et l'instant de réception de son écho maximal après réflexion sur une surface B (voir figure 6).

Fonctionnement du LiDAR ALS

Il convient de distinguer les trois manières existantes permettant le déploiement des capteurs LiDAR : les scanners laser terrestres (Terrestrial Laser Scanning, TLS) utilisant un trépied, les scanners laser aéroportés, transportés par avion (Airborne Laser Scanning, ALS) et récemment les scanners laser mobiles (Mobile LiDAR Systems, MLS), comprenant des capteurs montés sur des véhicules au sol (y compris des bateaux) ou encore des systèmes compacts transportables installés dans des sacs à dos. Dans ce manuscrit, nous nous concentrons sur les acquisitions aéroportées, car elles offrent une couverture spatiale et une densité de points suffisamment grande pour la génération de maquette numérique urbaine.

Les relevés ALS se caractérisent par une grande précision (erreur variant entre 0.05 à 0.2 mètre verticalement et de 0.2 à 0.6 mètre horizontalement) et une très haute résolution avec un échantillonnage variant entre 1 et 30 pts.m^{-2} [Passalacqua et al., 2015].

Le fonctionnement du LiDAR ALS se décompose en un système à trois éléments :
— d'un émetteur laser à émission pulsée, dont la longueur d'onde se situe dans le proche infrarouge ;
— de miroirs, oscillants ou rotatifs sur lesquels les rayons laser sont réfléchis ;
— d'un récepteur mesurant l'écho lumineux.

Dans le cas précis des systèmes aéroportés, le LiDAR est embarqué par un aéronef, auquel est ajouté un réseau GPS ainsi qu'une centrale inertielle [Reporting, 2007]. Ces deux éléments permettent de déduire la position de la source laser, la direction et l'inclinaison de l'avion. Grâce à ces informations, le système permet de recueillir pour chaque signal émis des positions (x,y,z) géoréférencées.

Afin de couvrir toute la zone rapidement, les impulsions sont réfléchies par un miroir oscillant vers la surface terrestre le long d'une droite horizontale perpendiculaire à la trajectoire de vol de l'avion (voir figure 7). L'ensemble de ces impulsions permet de relever le profil d'élévation, nommé fauchée au sol. Enfin, l'avancée de l'avion

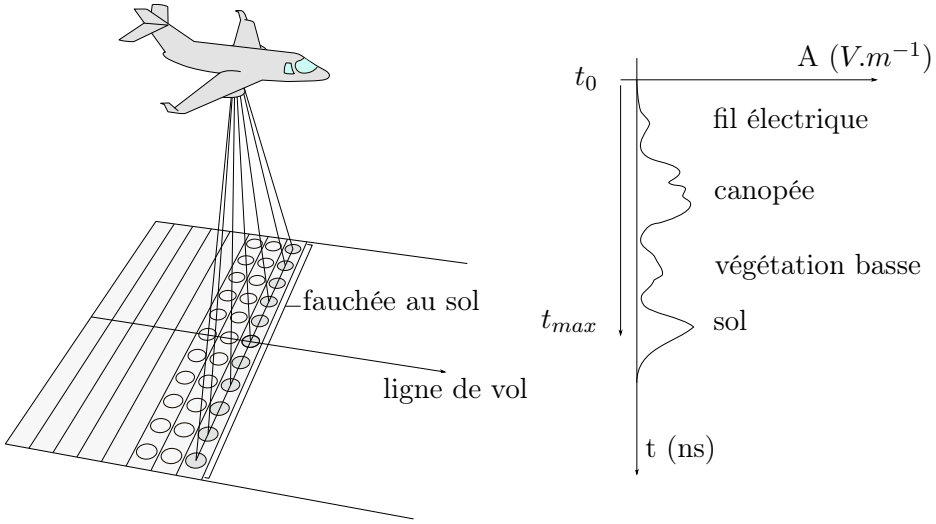


FIGURE 7 – **À gauche** : le nuage de points obtenu grâce à une acquisition LiDAR aéroporté est composé de différents profils d'élévation perpendiculaire à la trajectoire l'avion (ou ligne de vol). Ces profils sont obtenus à l'aide d'un mécanisme de miroir oscillant permettant de dévier le signal en différents points le long de la ligne de profil. La largeur des profils d'élévation est nommée fauchée au sol. **À droite** : amplitude en fonction du temps du signal reçu après réflexion sur plusieurs cibles. Ici, l'impulsion s'est réfléchie en partie sur différents objets, illustrés par l'ensemble des pics d'amplitude élevée, ces pics sont nommés échos.

produisant un ensemble de fauchées permet d'obtenir la 3^{e} dimension du nuage de points LiDAR.

Attributs des nuages de points LiDAR

En supplément des positions, l'analyse de l'amplitude des signaux reçus permet de distinguer plusieurs pics pour un même signal : ce sont les échos (voir figure 7). Ces échos apportent des informations supplémentaires nommées attributs : intensité, numéro de retour, valeurs de classification des points (bâtiment, végétation, route, pont, *etc.*), points situés à la limite de la ligne de vol, valeurs RVB (rouge, vert et bleu), heure GPS, angle et direction du balayage. Ces différents attributs sont énumérés dans le format LAS [for Photogrammetry and (ASPRS), 2008], dont une description est disponible en tableau 1 :

2.2 Précision de mesure

Les nuages de points issus de relevés LiDAR ALS possèdent certaines spécificités inhérentes à la manière dont ils ont été acquis. Il convient de les distinguer afin de proposer un traitement adapté à ces données.

Tout d'abord, les nuages de points issus de relevés LiDAR sont principalement composés de points représentant les surfaces visibles depuis l'aéronef (voir figure 8). Ces points décrivent les toitures des bâtiments, la canopée de la végétation, la surface des routes, *etc.* Le nuage de points s'apparente pratiquement à une carte de hauteur.

Attribut	Description
Intensité	Pour un point donné, l'intensité permet de quantifier l'amplitude d'un écho, elle est principalement fonction de la réflectivité de l'objet scanné. L'intensité est une mesure relative, chaque acquisition peut avoir des valeurs différentes.
Numéro de retour	De la même manière que le nombre de retour, le numéro de retour quantifie l'ordre dans lequel un écho a été reçu. Par exemple, le numéro 1 correspond au premier écho reçu, la canopée d'un arbre, le dernier correspond au sol situé en dessous de ce même arbre (voir figure 7).
Nombre de retour	Pour une impulsion laser, plusieurs retours ou échos peuvent être obtenus. Ce scalaire indique le nombre de retours total qu'une impulsion a fourni.
Classification	Les points post-traités peuvent être affiliés à une classe d'objet, par exemple : route, bâtiment, végétation haute ou basse, *etc.*
Booléen de direction de balayage	Ce booléen indique la direction dans laquelle le miroir du scanner se déplaçait au moment de l'impulsion de sortie. Un bit de valeur 1 correspond à une direction de balayage positive, et un bit de valeur 0 à une direction de balayage négative.
RVB	Cet attribut correspond à la couleur d'un point donné, quantifié avec les couleurs rouge vert et bleu.
Angle de balayage	Cette valeur exprime l'angle auquel l'impulsion laser a été envoyée, il peut varier entre -90° et 90°, mais en pratique se situe généralement entre -35° et 35°.
Heure GPS	Heure à laquelle le point laser a été émis. L'heure est estimée grâce au "temps GPS". Le point de départ (heure zéro) de l'heure GPS est défini actuellement au 6 avril 2019, puis le numéro de la semaine (le nombre de semaines depuis l'heure zéro) et le nombre de secondes écoulées dans cette semaine permettent d'évaluer le temps.
Bord de la ligne de vol	Le bit de données à une valeur de 1 uniquement lorsque le point se trouve à la fin d'une ligne de balayage.

TABLE 1 – Partie 1. En plus des positions x, y et z, des attributs sont stockés, ci-dessus la première partie des attributs établie par la norme LAS 1.2 [for Photogrammetry and (ASPRS), 2008].

En effet, la fauchée au sol ainsi que la trajectoire régulière de l'aéronef impliquent que les points sont distribués homogènement dans le plan xy. Puisque l'acquisition est réalisée depuis le ciel, certains éléments du paysage urbain peuvent ne pas être représentés, dus notamment aux occultations : même en variant le sens des prises de vue, grâce aux lignes de vol parallèles, les endroits fortement confinés ne peuvent pas être relevés par cette méthode d'acquisition. Par exemple, dans les banlieues pa-

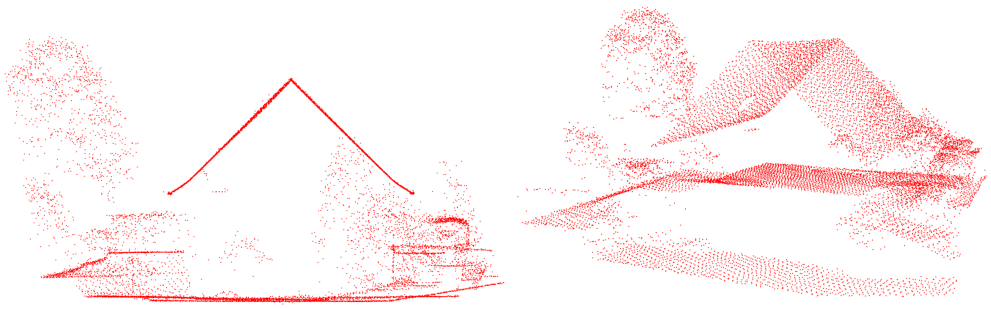


FIGURE 8 – Exemple de nuage issu de données LiDAR. Le nuage de points capture uniquement les données vues de haut et s'apparente à une carte de hauteur.

villonnaires, il est très probable de rencontrer des toitures partiellement occultées par de la végétation. Cet élément est particulièrement problématique puisqu'il est difficile, d'extrapoler les données manquantes en se basant sur ces uniques informations lacunaires.

La densité du nuage de points peut largement varier sur l'ensemble d'un nuage de points. En effet, la résolution du nuage de points dépendant de la trajectoire effectuée par l'aéronef, il est tout à fait possible que le nuage soit plus dense par endroits. Certaines zones rurales sont également moins denses, tout simplement parce qu'il n'est pas nécessaire d'avoir beaucoup de précision pour l'acquisition d'un champ, par exemple.

La précision d'une acquisition joue également un rôle important, en particulier lorsqu'il faut construire une maquette numérique la plus proche de la réalité possible. Une des principales limitations de l'acquisition LiDAR est la variabilité de cette même précision. Le capteur peut produire des données aberrantes et des données bruitées par plusieurs facteurs intrinsèques (capteur du signal, bruit du capteur et système d'amplification) et extrinsèques (milieux de propagation de l'onde laser, distance, réflectance de la surface, angle d'incidence) au laser. Certaines erreurs peuvent être facilement évitées en s'assurant que les conditions d'acquisitions soient optimales. Pour cela, les campagnes d'acquisitions se déroulent par temps peu humide, c'est-à-dire sans couche nuageuse ni pluie, car cela peut atténuer le signal, mais aussi sans neige, car celle-ci peut fausser les relevés altimétriques. Pour autant, il demeure d'autres sources d'erreurs plus complexes à contrôler.

Les erreurs les plus critiques portent le plus souvent sur un désalignement angulaire entre le scanner LiDAR et le système de navigation et/ou de position [Reporting, 2007]. Les désalignements sont souvent causés par l'erreur humaine lors du vol, ou lors de la calibration. La précision peut également beaucoup varier en fonction de la qualité de la trajectoire effectuée par l'avion. Selon, l'ASPRS (*American Society for Photogrammetry and Remote Sensing*), une erreur d'alignement de l'aéronef suivant ses axes de rotation peut avoir une conséquence dramatique sur sa précision en altitude. Des imprécisions peuvent également se produire dans le cas où le réseau de base DGPS au sol est erroné, entrainant des problèmes importants lors du recalage des données obtenues avec d'autres acquisitions existantes. La précision est également hautement liée à l'altitude de vol. L'aéronef peut avoir une altitude plus ou moins importante pour augmenter ou diminuer la largeur de fauchée au sol. Toutefois, plus l'avion à une altitude élevée, plus l'empreinte du laser augmente au sol, car un laser

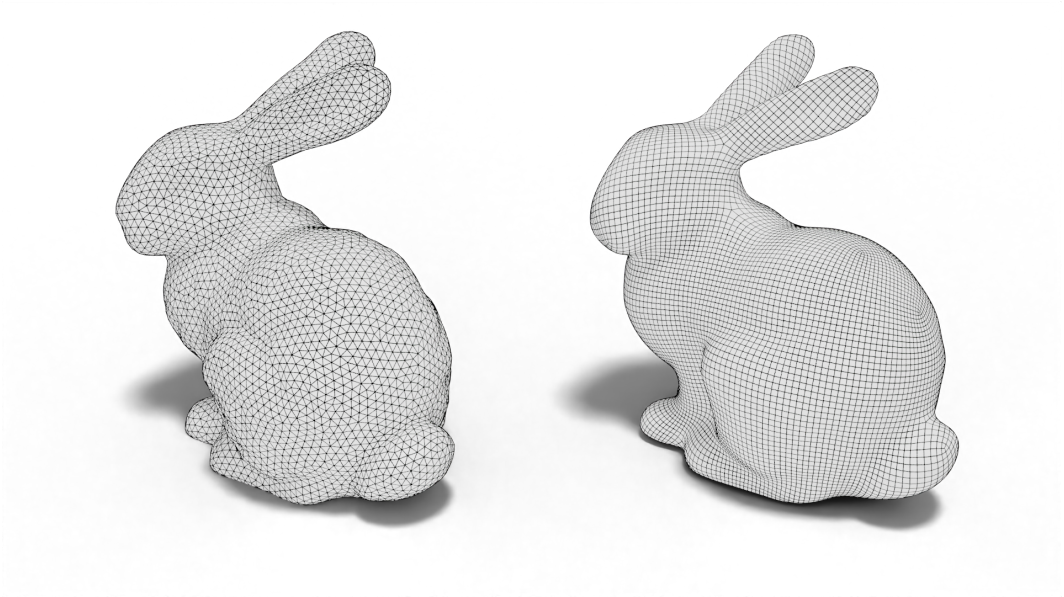


FIGURE 9 – Les maillages sont une manière de représenter des objets tridimensionnels numériquement. Ici, l'objet à gauche est représenté par des triangles, nommé faces et décrit par un triplet de points nommés sommets. L'ensemble des sommets sont liés par des arêtes. Le maillage à gauche est une quadrangulation où les faces sont représentées par des quadrilatères.

n'est pas parfaitement focalisé.

Nous avons présenté le moyen d'acquisition que nous utiliserons pour la génération de données terrain géoréférencées sous la forme de nuage de points. Dans la section qui suit, nous posons le problème suivant : comment transformer notre nuage de points en un maillage exploitable pour la simulation numérique et la visualisation ?

3 Du nuage de points au maillage : formulation du problème

Dans le cas le plus général, en géométrie, la définition d'un nuage de points est relativement simple : il s'agit d'un ensemble de coordonnées discrétisant un espace. L'espace auquel appartiennent ces points est fréquemment $\mathbb{R}^2$ ou $\mathbb{R}^3$. Un nuage de points est dépourvu de toute structure, c'est-à-dire qu'il n'existe aucune relation de voisinage entre deux points voisins. C'est justement cette structure qui est nécessaire à la plupart des traitements algorithmiques, tels que les simulations numériques. Il convient donc de transformer ces nuages de points en une structure géométrique plus adaptée, que l'on nomme maillage.

Le maillage est l'une des structures de données les plus utilisées dans les domaines du graphisme, de la simulation, de l'approximation, de la reconstruction d'objets 3D, *etc.* Un maillage est illustré en figure 9. Il existe différentes représentations explicites de celui-ci, dont les propriétés varient suivant le domaine d'application. Plus formellement, il est défini par :

— **ses sommets** connectés deux à deux par des arêtes.

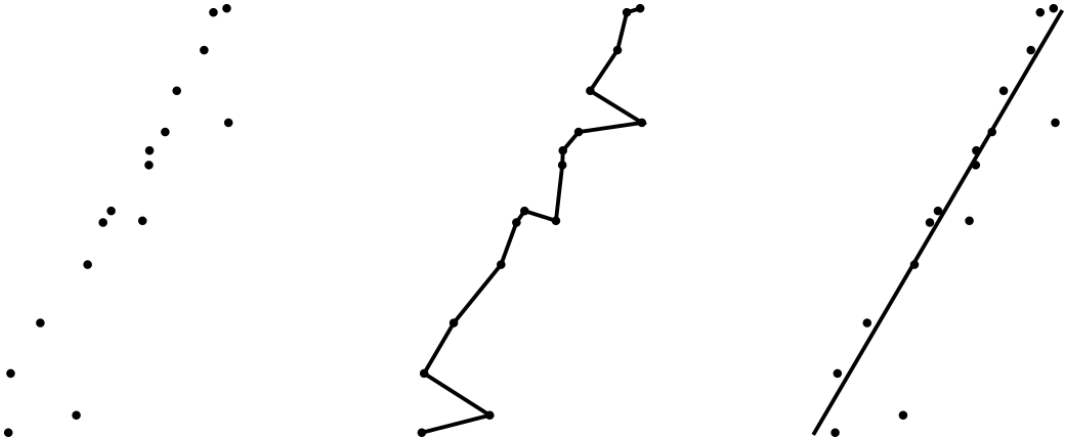


FIGURE 10 – Différence entre interpolation et approximation d'une courbe à un nuage de points. À gauche un nuage de points que nous souhaitons représenter par une courbe, au centre une interpolation de ce nuage de points où l'ensemble de l'information du nuage de points est capturée, y compris le bruit de mesure, tandis que dans le cas de l'approximation (à droite), nous cherchons à ajuster au mieux la courbe au nuage de points.

— **ses arêtes** situées à la frontière des faces.
— **ses faces** constituées d'un ensemble de sommets. Par défaut, une face sera constituée d'un nombre arbitraire de sommets. Généralement, les faces sont triangulaires, le maillage se nomme alors triangulation.

Pour représenter des volumes, il est possible d'utiliser des maillages volumiques où les sommets sont reliés par des polyèdres (le plus souvent des tétraèdres).

Comme nous avons pû le voir dans la section précédente, puisque nos nuages de points sont issus de relevés LiDAR, c'est-à-dire de mesures physiques, ils sont naturellement bruités. Il est donc préférable de chercher une surface approximante qui permettra de réduire le bruit, plutôt qu'une interpolation (voir figure 10). De manière plus simple, nous souhaitons ajuster une surface au plus proche des points, plutôt que de considérer que notre surface doit passer par tous les points.

Cependant, la reconstruction de surface à partir d'un nuage de points par approximation est un problème mal posé [4], par exemple, un nombre infini de surfaces peut passer par le nuage de points. Il convient donc de délimiter notre problème afin d'en faciliter sa résolution.

3.1 Hypothèses pour la reconstruction

Comme expliqué précédemment, la société RhinoTerrain souhaite produire des maquettes numériques urbaines LOD2.2 (voir figure 2) à partir de numérisations issues de relevés LiDAR. Ce choix découle, d'une part, du fait que LOD2 est le niveau de détail le plus demandé par nos clients. D'autre part, car, les relevés aériens

4. Un problème bien posé, dont la définition découle des travaux du mathématicien Jacques Hadamard, doit posséder les propriétés suivantes : une solution existe, la solution est unique, et enfin, elle dépend de façon continue des données. Un problème est dit mal posé s'il ne satisfait pas l'une de ces propriétés.

ne disposent pas assez de relevés fiables au niveau des façades pour permettre une modélisation d'un niveau de détail supérieur (LOD2.3 par exemple). *A contrario*, ces numérisations fournissent une bonne représentation des toitures des bâtiments, nous nous concentrons donc sur la restitution de maillages de toitures, que nous extruderons verticalement pour en obtenir les murs.

Pour être parfaitement opérationnels, ces maillages doivent respecter un ensemble de contraintes, ils doivent être topologiquement et géométriquement justes ainsi que sémantiquement riches.

RhinoTerrain a mis au point un cahier des charges strict pour ces maillages :

- Pour une toiture donnée, chaque face du maillage représentant la toiture doit représenter dans son intégralité, une primitive de bâtiment, c'est-à-dire un pan de toit.
- Chaque face du maillage doit être planaire. Cette condition est nécessaire pour les simulations de potentiel solaire.
- Les pans de toits doivent suivre des contraintes d'alignement ou encore de perpendicularité. Par exemple, la faitière d'une toiture doit être parallèle avec ses gouttières, et la gouttière doit être perpendiculaire à l'arbalétrier.
- Chaque face d'un maillage de bâtiment doit posséder une orientation z-top, c'est-à-dire posséder une normale orientée vers l'axe z positif.
- Les maillages ne doivent pas présenter de trous.
- Les maillages ne doivent pas présenter d'auto-intersections ni d'intersections avec les autres maillages de bâtiment.
- Les maillages doivent représenter l'intégralité de la toiture existante, même si le nuage de points est lacunaire.

L'ensemble de ces contraintes seront traitées dans la suite des chapitres présentant des méthodes de reconstruction. Dans la suite de cette section, nous explorons les raisons qui ont conduit RhinoTerrain à vouloir reconstruire des maquettes numérique urbaine LOD2. Pour cela, nous abordons les différentes applications visées par l'entreprise.

3.2 Applications visées par l'entreprise

Autrefois principalement utilisées pour la visualisation, les MNU sont aujourd'hui de plus en plus utilisés dans différents domaines et pour un large éventail de tâches. Dans un contexte où il devient de plus en plus important de se préparer à tout risque climatique, ces maquettes numériques représentent un outil formidable pour la réalisation d'analyses autrefois impossibles avec des cartes 2D. Ainsi, ces dernières années, un nombre croissant de villes se sont dotées ou ont cherché à se doter de telles maquettes [Stoter et al., 2016]. L'article [Biljecki et al., 2015] donne un aperçu complet de l'état de l'art de l'application des modèles de ville en 3D.

Estimation de potentiel solaire et des ombres portées

L'estimation de potentiel solaire, ou de cadastre solaire, est sûrement l'un des cas d'utilisation les plus importants des modèles numériques urbains (voir [Freitas et al., 2015, Martín et al., 2015] pour une revue complète du domaine). Le niveau d'illumination au cours des années est évalué sur chaque façade et toiture de bâtiment afin d'estimer le meilleur endroit pour placer des panneaux photovoltaïques (voir

FIGURE 11 – Cadastre solaire calculé avec le logiciel RhinoSolar d'une partie du 9e arrondissement de la métropole de Lyon, visualisation sur une carte 2D [métropole, 2022b]. Estimation donnée variant de bleu à rouge. Bleu signifie faible exposition solaire, orange, bonne exposition et rouge, excellente exposition.

figure 11). Ces simulations permettent également d'identifier des îlots de chaleurs urbains, c'est-à-dire des zones où la température est bien plus élevée localement. Au sein de l'entreprise RhinoTerrain, ce genre de simulation est réalisé avec des modèles LOD2. Dans la continuité des études solaires se trouvent les études d'ombres portées. En effet, si un bâtiment d'envergure, ou une éolienne vient à être construit, il est important de vérifier au préalable l'impact de sa construction en termes de luminosité sur les bâtiments avoisinants.

Simulation d'inondation

D'abord réalisée à travers des modèles d'élévation [Sahoo et al., 2006], l'estimation des risques d'inondations est un sujet qui a été largement abordé dans la littérature. Aujourd'hui, grâce à l'avènement du CityGML, ce type de simulation est désormais réalisée avec des MNU, le plus souvent en LOD1 [Varduhn et al., 2015, Gandini et al., 2020, Jang et al., 2021], améliorant ainsi la qualité des résultats obtenus, par rapport à la SIG 2D.

Estimation de la consommation d'énergie des foyers

Ces études portent sur l'estimation de consommation d'énergie, comme le chauffage, la climatisation ou les pertes de chaleur par pont thermique. Elles peuvent être réalisées avec des MNU en LOD2 [Agugiaro, 2016, Kaden and Kolbe, 2014].

Estimation de la propagation du bruit dans un environnement urbain

Les modèles numériques urbains présentent de réels atouts pour le calcul de niveau de bruit en zone urbaine. Différents articles se sont penchés sur le sujet en proposant des modèles de bruits adaptés à des représentations CityGML [Stoter et al., 2008, Kavisha et al., 2017]. En outre, des directives gouvernementales récentes poussent les villes à se doter de ces modèles. Par exemple, la directive 2002/49/CE établie par le Conseil et le Parlement européens [Directive, 2002], visant à évaluer le niveau de pollution sonore de différents lieux à fort trafic, nécessite des modèles 3D. Cette analyse peut être réalisée à partir de données SIG 2D, mais l'utilisation de MNU offre de meilleurs résultats. En effet, d'après Jarosław Kubiak and Radzym Ławniczak [Kubiak and Ławniczak, 2016], pour les bâtiments situés près de zones à fort trafic, le niveau sonore peut varier en fonction de la hauteur en raison de la réfraction.

Il existe bien d'autres applications, comme l'estimation de la propagation d'ondes radioélectriques où l'on estime la couverture Wifi en réfléchissant des ondes à partir des données publiques des positionnements de routeurs, ou la dynamique des fluides, pour estimer le flux de vent et donc l'intérêt de l'ajout d'éoliennes, ou encore la détection de changements pour la vérification cadastrale. L'article [Biljecki et al., 2015] cite pas moins de 29 applications différentes, faisant du MNU un outil pluridisciplinaire et polyvalent.

Ces différentes applications délimitent les besoins actuels des métropoles. Dans notre cas, nous souhaitons proposer des modèles numériques compatibles avec ces différentes analyses : de potentiels solaires, de nuisance sonore ou encore de simulation d'inondations. C'est donc pour cette raison que nous nous concentrons sur la génération de MNU LOD2. En effet, ce niveau de détails est pour le moment le niveau le plus utilisé pour toutes ces simulations. D'une part, car il constitue un bon compromis entre la complexité de la surface et sa proximité avec les données réelles. D'autre part, car il est facilement possible de passer d'un modèle en LOD2 à un modèle en LOD1 au besoin.

Dans la section suivante, nous présentons l'état de l'art de la reconstruction de maquette numérique urbaine LOD2 à partir de nuage de points ALS.

4 Littérature de la reconstruction de bâtiment

Comme nous avons pu le voir dans les sections précédentes, les maquettes numériques urbaines sont des objets qui ont gagné, au cours des dernières décennies, une attention croissante. Sans surprise, de nombreux travaux de recherche se sont donc concentrés sur la reconstruction automatique de telles maquettes. Mais la reconstruction de MNU à partir de données réelles demeure un problème particulièrement complexe, notamment dû à la qualité de ces mêmes acquisitions (occlusions, valeurs aberrantes, densité variable, *etc.*).

4.1 Portée de l'étude

Les maquettes numériques urbaines sont composées de multiples objets, parmi ces objets nous trouvons : des routes, des bâtiments, de la végétation, des voitures,

des fils électriques, *etc.* Dans cette section, nous nous consacrons à la littérature de la reconstruction de bâtiments. L'ensemble des articles cités dans cette bibliographie sont issus des communautés d'informatique graphique et de télédétection.

Dans cette étude, nous nous intéressons spécifiquement à la reconstruction de maquettes numériques urbaines de ville existante, toutefois, il existe un pan de la littérature analogue traitant de la modélisation procédurale de MNU dont une revue exhaustive est dressée par Ruben Smelik [Smelik et al., 2014]. En effet, la génération procédurale permet la création rapide de maquettes numériques urbaines, en se basant uniquement sur un ensemble de règles génératives [Biljecki et al., 2016b], mais, elle ne permet pas, par nature, la reconstruction exacte des villes préétablies.

Cette étude se concentre plutôt sur la reconstruction de bâtiment à partir de données issues d'acquisitions réelles, notamment les données issues d'acquisitions LiDAR aéroporté. Par conséquent, nous ne traiterons pas des algorithmes de traitement de données issues de TLS ou de MLS (voir section 2), car, ils diffèrent largement des algorithmes développés pour la télédétection aéroportée.

Plusieurs articles proposent une étude de ces différentes méthodes comme [Brenner, 2005, Haala and Kada, 2010, Musialski et al., 2013, Berger et al., 2017]. À notre connaissance, le dernier article présentant une revue complète du domaine est proposé par Wang et al. [R. Wang and Chen, 2018]. Dans cette section, nous passons en revue les méthodes de reconstruction de bâtiments de la littérature à partir de nuage de points.

Fondamentalement, il existe deux grandes approches pour la modélisation de bâtiments dans la littérature : l'approche paramétrique (ou basée sur des modèles) et l'approche non paramétrique (ou basée sur des données).

Les approches basées sur les modèles tentent d'ajuster certaines formes prédéfinies aux données, tandis que les approches guidées par les données tentent d'extraire les formes présentes dans celles-ci. Bien que les méthodes basées sur les modèles soient considérées comme relativement plus robustes par rapport aux méthodes basées sur les données, leurs performances sont limitées aux modèles connus [V. Verma and Hsu, 2006,Xu et al., 2017,Xiong et al., 2014a]. Les méthodes guidées par les données fonctionnent, en théorie, pour toutes les formes de toitures possédant des pans planaires. Depuis peu, des méthodes utilisant l'intelligence artificielle apparaissent dans la littérature, une revue de ces méthodes est proposée par Mehmet Buyukdemircioglu et al. [Buyukdemircioglu et al., 2022]. Ces méthodes ne permettent pas de respecter l'ensemble des contraintes définies par RhinoTerrain et ne seront donc pas présentées dans cette section.

Les sous-sections suivantes présentent les deux types de méthodes, d'abord les méthodes dites basées sur les données en sous-section 4.2, puis les méthodes dites basées sur les modèles en sous-section 4.3. En raison de la profusion d'articles dans le domaine de la reconstruction des bâtiments, cette section vise uniquement à en présenter un aperçu, en illustrant les différents courants par des articles bien choisis et n'a donc pas pour vocation d'être exhaustive.

4.2 Approche basée sur les données

Les méthodes dites basées sur les données sont issues du constat suivant : les bâtiments sont des objets construits de la main de l'homme, par conséquent, ils présentent des formes relativement simples et organisées sous forme de grandes régions

planaires. La plupart des travaux [Vosselman, 1999, Zhang et al., 2006, Elaksher et al., 2002, Dorninger and Pfeifer, 2008, Zhou and Neumann, 2008, Zhou and Neumann, 2010, Tarsha-Kurdi et al., 2008a, Lafarge and Mallet, 2012, Chen et al., 2014, Shahzad and Zhu, 2016, Martín-Jiménez et al., 2020] utilisent différents pipelines où chaque étape détecte et/ou combine des caractéristiques [Milde and Brenner, 2009, Xiong et al., 2014b] extraites du nuage de points. Pour expliquer ces méthodes, nous commençons par décrire les étapes d'un pipeline typique. Ce pipeline comprend les trois grandes étapes suivantes :

— Segmentation : les points du nuage appartenant aux bâtiments sont agrégés en régions planaires que l'on nomme segments. Ces segments sont constitués en s'assurant que chaque segment représente un pan de toit.
— Combinaison : les segments résultants sont ensuite combinés pour extraire les caractéristiques du bâtiment, pour cela des intersections entre les plans obtenus sont réalisées pour obtenir les lignes d'intersection, les contours des bâtiments, les coins, *etc.*
— Régularisation : à partir des données obtenues au travers des deux étapes précédentes, il cherche à recréer un modèle de toit conforme.

Un apperçu des trois grandes étapes du pipeline de reconstruction est présenté ci-dessous :

Segmentation du nuage de points en segments planaires

Une des premières étapes de la reconstruction de bâtiments à partir de nuages de points consiste à extraire des informations de ces nuages de points. En effet, si nous considérons un nuage de points d'une ville entière, il est plus commode d'extraire les points présentant de l'intérêt à la reconstruction pour la faciliter. Pour cela, nous pouvons segmenter le nuage d'entrée en sous-ensembles de points d'intérêts, qui devraient idéalement représenter des entités élémentaires (c'est-à-dire des plans), permettant de se réduire à des cas pouvant être traités indépendamment les uns après les autres. Nous pouvons classer les méthodes de reconstruction de bâtiments (basées sur les données) en quatre catégories en fonction de l'approche utilisée pour détecter les régions planes dans les nuages de points :

— **La transformée de Hough** [Hough, 1962] est l'une des premières méthodes utilisées pour la reconstruction de bâtiments [Maas and Vosselman, 1999a], et elle a été améliorée et ajustée de nombreuses fois [Huang and Brenner, 2011, Novacheva, 2008, Maltezos and Ioannidis, 2016].
— **RANSAC**, abréviation pour RANdom SAmple Consensus, est une méthode appliquée fréquemment [G. Forlani and Zingaretti, 2006, D. Chen and Liu, 2012]. Elle tend à être plus robuste que la transformée de Hough. Une comparaison entre ces méthodes est présentée dans [Tarsha-Kurdi et al., 2008b] pour la segmentation automatique de zones planes à partir de nuages de points. Toutefois, l'algorithme RANSAC n'est pas robuste au bruit en général [Le and Duan, 2017] mais fonctionne assez bien lorsque les primitives sont de grandes régions planes.
— L'approche par **croissance de régions** est largement utilisée pour sa simplicité et son efficacité. Contrairement à la transformée de Hough et à RANSAC, il s'agit d'une technique de segmentation locale. Dans [Alharthy and Bethel, 2004] la croissance de région est appliquée pour construire des seg-

ments planaires par une technique d'agrégation de cellules. D'autres méthodes de segmentation basées sur la croissance de région dans le contexte de la reconstruction de bâtiments en 3D sont, par exemple, utilisées dans les applications suivantes [Awrangjeb and Fraser, 2014, Abdullah et al., 2014, Sun and Salvaggio, 2013, Zhou and Neumann, 2012].

— Une autre option viable (mais coûteuse) consiste à effectuer une **optimisation globale**. Dans [Kim and Shan, 2011], les segments de toit planaires sont déterminés en minimisant un problème variationnel se basant sur la variation des normales des points.

Génération de modèle

Une fois qu'un ensemble de segments a été déterminé, des indices de modélisation peuvent être extraits pour la construction ultérieure de modèles de bâtiments 3D. Pour cela, les lignes d'intersections, les lignes de saut de hauteur et les contours des bâtiments sont fréquemment extraits. Bien que les contours de bâtiments puissent être considérés comme un type particulier de lignes de saut de hauteur, ils sont souvent récupérés dans une étape de traitement distincte.

Il existe plusieurs approches qui se concentrent sur la génération d'une ébauche de toiture et sa simplification ou sa régularisation (qui seront expliquées dans la sous-section suivante), c'est-à-dire une modification du modèle obtenu. Ces approches sont, par exemple, basées sur l'algorithme RANSAC [Jarzabek-Rychard, 2012], α-shape [Peter and Pfeifer, 2008, B. Albers, 2016, R.C. dos Santos and Carrilho, 2019], grilles structurées [Zhou and Neumann, 2012, Sun and Salvaggio, 2013], ou simplification de lignes comme Douglas-Peucker [Douglas and Peucker, 1973].

Les discontinuités de hauteur entre les segments adjacents sont recherchées dans l'article de G. Vosselman [Vosselman, 2000]. Dans [Rottensteiner et al., 2012], les contours sont estimés sur la base de tests statistiques. Un filtre appelé *compass line filter* (CLF) est proposé dans le document [G. Sohn and Tao, 2008]. Les lignes de crête sont généralement obtenues directement par l'intersection de deux plans qui sont dérivés d'une paire de segments adjacents. D'autres indices de modélisation sont, par exemple, extraits dans [Overby et al., 2004], où, l'intersection de tous les pans de toits est réalisée, indépendamment de leurs contiguïtés, puis les segments obtenus sont préservés ou non en fonction de leur distance au nuage de points. Une approche pour détecter les indices de modélisation sous forme de lignes de crête en utilisant RANSAC est présentée dans [Fan et al., 2014].

La plupart des méthodes de reconstruction basées sur les données génèrent un modèle polyédrique 3D directement à partir des indices de modélisation, [Rottensteiner et al., 2012, Novacheva, 2008, Peter and Pfeifer, 2008, Zhou and Neumann, 2012, Xiao et al., 2015]. Pour cela, les lignes extraites sont étendues et connectées entre elles de manière à ce que chaque surface de toit soit délimitée par une séquence fermée de segments de ligne connectés. La mise en œuvre d'une telle méthode de génération directe d'un modèle polyédrique pose toutefois de nombreux problèmes, car il existe généralement des ambiguïtés sur la manière dont les segments de ligne doivent être connectés pour garantir la planarité de chaque face polyédrique sans aucun vide entre les deux. Il existe des méthodes utilisant la triangulation 2.5D [Dong et al., 2017], mais elles ne garantissent pas la topologie correcte du modèle (saut de hauteur).

Régularisation du modèle obtenu

Comme les modèles de bâtiments dérivés des approches de reconstruction basées sur les données ressemblent de très près aux données d'entrées, elles sont par conséquent imparfaites. Plusieurs méthodes de régularisation et d'optimisation ont été développées pour pallier ce problème. Pour cela, la plupart des méthodes décrites ci-dessus intègrent l'orientation principale du bâtiment pour assurer l'orthogonalité et le parallélisme de certaines arêtes.

Un modèle d'ajustement qui tient compte de la topologie du bâtiment pour améliorer la forme d'un modèle de bâtiment est par exemple proposé dans [Rottensteiner, 2006]. Dans [Zhou and Neumann, 2012], la géométrie finale du toit est optimisée en fonction des contraintes découvertes, par exemple l'alignement des arêtes ou les sommets de même hauteur. Afin de déterminer automatiquement des contraintes indépendantes et cohérentes, un algorithme glouton est proposé dans [Boge et al., 2013] tandis que les bases de Grobner sont utilisées dans [Meidow and Hammer, 2016]. Dans [Sohn et al., 2013] et [Jarzabek-Rychard and Maas, 2017], un raffinement des modèles de bâtiments est proposé sur la base d'images aériennes à partir desquelles les bords des bâtiments sont détectés et incorporés dans le processus de reconstruction. Un processus de régularisation implicite dans le cadre de MDL en combinaison avec l'hypothèse et le test (HAT) est, par exemple, proposé dans [Jaewook et al., 2017].

4.3 Approche basée sur les modèles

Contrairement aux méthodes basées sur les données, les méthodes basées sur les modèles nécessitent de d'abord choisir un modèle, puis de faire correspondre ces modèles au nuage de points. Pour cela, ces méthodes utilisent une bibliothèque de modèles, comprenant des toits à deux pans, des toits-terrasses, des toits en pavillons, *etc.*

Par exemple, l'article [Maas and Vosselman, 1999b] propose d'estimer un indicateur largement utilisé en traitement d'images, nommé l'invariant de moment pour vérifier si un toit est à pignon ou non. Cette approche présente le défaut de demander énormément de calcul lorsqu'il faut inférer des structures se trouvant sur les toits. Dans leur article [Poullis and You, 2008], Poulis et You recherche les primitives géométriques des toits en explorant les contraintes de symétrie qui existent en architecture. Cependant, leur bibliothèque de modèle ne permet pas de reconstruire tous les modèles, et se généralise mal à d'autres types de toitures. Huang et al. [Huang et al., 2011a] effectue une modélisation générative pour construire le toit cible qui correspond aux données. La sélection des primitives de toit, ainsi que l'échantillonnage de leurs paramètres, est pilotée par la technique de Monte-Carlo par chaîne de Markov à saut réversible. Les primitives obtenues sont assemblées et, si nécessaire, fusionnées pour obtenir un toit entier. Toutefois, cette méthode use d'une régularisation peu robuste.

Henn et al. [Henn et al., 2013] proposent de reconstruire des bâtiments LOD2 à partir de nuages de points épars. Pour identifier le modèle de toit le plus probable, des méthodes d'apprentissage automatique supervisées (SVM) sont utilisées. Ils appliquent l'algorithme RANSAC pour estimer les paramètres du modèle. Toutefois, cette méthode nécessite une boîte englobante 2D du toit comme entrée.

Dernièrement, Zhang et al. [Zhang et al., 2021] ont proposé une méthode permettant de reconstruire un bâtiment à partir d'une distance de nuage de points à maillage. Cependant, ils utilisent un terme d'énergie imposant que l'air de l'enveloppe convexe du maillage obtenu soit inférieur à l'air de l'enveloppe convexe du nuage de points. Cela peut engendrer des reconstructions erronées sur les nuages de points lacunaires.

5 Objectifs

Malgré deux à trois décennies de recherche, la génération automatique de MNU demeure une question ouverte, pour laquelle il n'existe aujourd'hui aucune solution satisfaisante qui soit à la fois généraliste et automatique. Pour cause, générer des maquettes numériques urbaines n'est pas sans difficulté : il faut à partir d'acquisitions de données réelles bruitées et lacunaires, produire un modèle géométrique structuré, conciliant proximité aux données et simplicité, tout en restant compatible avec les exigences de la simulation numérique.

L'objectif de cette thèse est le développement d'outils d'assistances interactifs pour les modeleurs responsables de la création de maquettes numérique urbaine. Pour cela, nous souhaitons fournir des méthodes permettant la reconstruction automatique de bâtiments sous la supervision d'un opérateur humain. Notre démarche se qualifie de semi-automatique. Les outils que nous présentons dans ce manuscrit poursuivent les trois objectifs suivants :

— faciliter le travail du modeleur en améliorant l'ergonomie des outils proposés ;
— réduire le coût de production de modèle urbain en réduisant le temps de modélisation par bâtiment, afin de rendre plus accessibles les simulations numériques pour les municipalités ;
— produire des modèles géométriques de bâtiments les plus représentatifs de la réalité terrain et d'une qualité suffisante pour permettre la réalisation de simulation numérique.

Première partie

Reconstruction de modèle numérique urbain

L'objectif de cette première partie est de présenter deux méthodes de reconstruction de toiture pour la génération de maquettes numériques urbaines à partir de données LiDAR aéroporté.

Nous abordons dans le chapitre 1 de cette partie, une méthode dite basée sur les données, permettant la reconstruction de surface représentant les toitures des bâtiments d'une maquette numérique. L'originalité de l'approche consiste à considérer le problème d'un point de vue topologique, permettant de s'abstraire des problèmes liés aux intersections de plans des méthodes traditionnelles.

Puis, dans le chapitre 2 nous mettons en avant une méthode permettant d'ajuster des surfaces paramétriques à des nuages de points. Pour cela, nous décrivons un problème variationnel, où l'on minimise la distance entre le nuage de points et la surface.

Chapitre 1

Reconstruction de toitures par approche basée sur les données

Sommaire

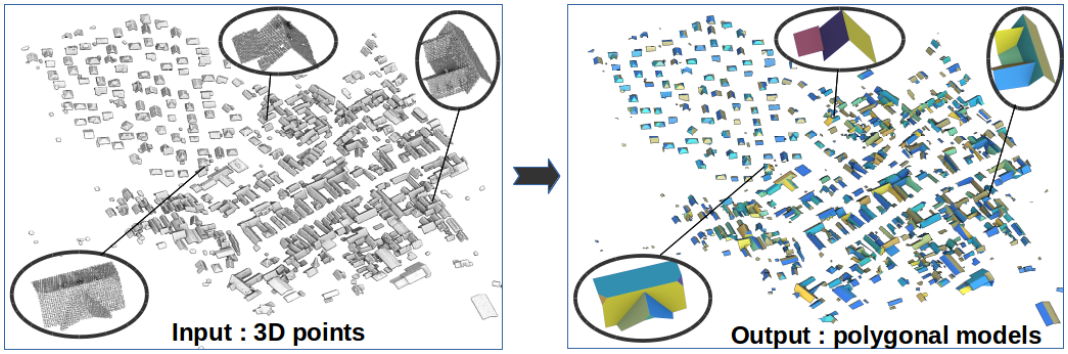


FIGURE 1.1 – Construction de modèles polygonaux de toits à partir de données LiDAR préalablement classées.

1.1 Contributions

La méthode prend en entrée un ensemble de nuages de points classifiés. Pour cela, nous préservons uniquement la classe bâtiment de la classification LAS/LAZ (définie dans le chapitre introductif). Chaque classe utilisée en données d'entrée correspond à la toiture d'un bâtiment individuel ou à un ensemble de toitures de bâtiments mitoyens (voir figure 1.1). En sortie, la méthode fournit un modèle de toit, sous la forme d'un maillage 3D où chaque région planaire est représentée par un polygone lié de manière cohérente à ces voisins. Cette méthode suit le pipeline des méthodes dites basées sur les données en améliorant les étapes de maillage et de régularisation. Notre originalité vient du fait que nous fixons en priorité les propriétés topologiques du modèle résultant, qui sont difficiles à recouvrer par des approches géométriques. Cette approche topologique rend notre solution plus simple et plus robuste que les méthodes existantes qui extraient le plus souvent les lignes d'intersection. Les résultats expérimentaux montrent que le pipeline proposé offre un taux de réussite élevé pour l'extraction des plans (94% de complétude, 92,7% d'exactitude, 90,8% de qualité) lorsque la densité de points est suffisamment élevée. En outre, cette approche combinatoire permet de traiter plus facilement les bâtiments mitoyens possédant des arêtes parallèles avec un saut de hauteur.

1.1.1 Vue d'ensemble

Cette méthode suit le pipeline d'une méthode basée sur les données et est décomposée en trois étapes : segmentation, modélisation, régularisation. L'étape de segmentation consiste à délimiter les pans de toiture grâce à un algorithme de segmentation combinant un estimateur de normale basé sur la transformée de Hough et une stratégie de croissance de région. Ensuite, chaque toiture est modélisée par un maillage 2D dont chaque pan est défini par une région planaire. Ce maillage permet de découvrir d'une part le contour du bâtiment et d'autre part le faîtage [5], sans calcul d'intersection de plans.

5. Pour rappel, le faîte est l'arête supérieure de la toiture qui forme une ligne où se rencontrent les deux pans d'un toit.

Le maillage encode directement les relations topologiques entre les pans voisins nous permettant de construire le modèle polygonal final de manière directe. La méthode est entièrement automatique, à l'exception de l'ajustement de quelques paramètres devant être choisis avec soin par un opérateur humain. La méthode proposée vise à construire des modèles de toit avec un niveau de détail LOD2, tel que défini dans la norme CityGML (voir chapitre introductif), c'est-à-dire des structures de toit détaillées ne possédant pas de superstructures (telles que des cheminées, des lucarnes, *etc.*).

Les trois grandes étapes de cette méthode sont les suivantes :

1. **Segmentation et clusterisation**, la normale en chaque point est calculée avec un estimateur de normale nommé transformée de Hough. Puis, grâce à une approche par croissance de région, les points voisins possédant des normales similaires sont rassemblés en un nuage de points représentant une zone planaire.
2. **Reconstruction**, ensuite, chaque point du toit est projeté dans le plan xy et son α-shape [Edelsbrunner et al., 1983] est construit à partir de la triangulation de Delaunay 2D de la toiture. Les arêtes constituant le contour initial du bâtiment sont extraites des triangles se trouvant sur le bord de la triangulation. Puis, nous extrayons les lignes de faîtage du toit à partir de l'ensemble des triangles du maillage dont au moins deux des sommets appartiennent à des pans différents. Cette approche nous permet de construire ensuite un modèle topologique précis (graphe) où chaque sommet correspond à l'intersection d'exactement trois versants du toit. Chaque arête correspond à l'intersection de plans adjacents et chaque facette correspond à un versant de toit.
3. **Régularisation**, enfin, nous régularisons le modèle obtenu en détectant les sauts de hauteurs parmi les faitières et en assurant la planarité des polygones.

La suite de la section est organisée comme suit : §1.1.2 décrit l'étape de détection des plans, puis §1.1.3 explique l'étape de génération du modèle, et enfin, §1.1.4 détaille l'étape de régularisation du modèle. La figure 1.2 fournit une représentation visuelle de la méthode et présente chaque étape de notre traitement (A à I).

1.1.2 Détection des régions planaires

Notre méthode de détection de versants de toits traite individuellement chaque groupe de bâtiments et prend en compte les coordonnées des points du nuage ainsi que leurs vecteurs normaux. Nous avons utilisé un algorithme basé sur la transformée de Hough fourni par Boulch [Boulch and Marlet, 2012] permettant d'approximer le plan tangent de chaque point en considérant ses 30 voisins les plus proches. Cette méthode est particulièrement bien adaptée à nos données, car elle ne lisse pas les vecteurs normaux des faîtes.

Ensuite, un *algorithme de croissance de région*, fourni par la bibliothèque [PCL, 2022], est utilisé pour fusionner les points qui sont suffisamment proches en termes d'écart de normales. Afin de calculer l'équation cartésienne de chaque segment plan détecté, nous calculons le vecteur normal au plan comme la moyenne des normales de ses points.

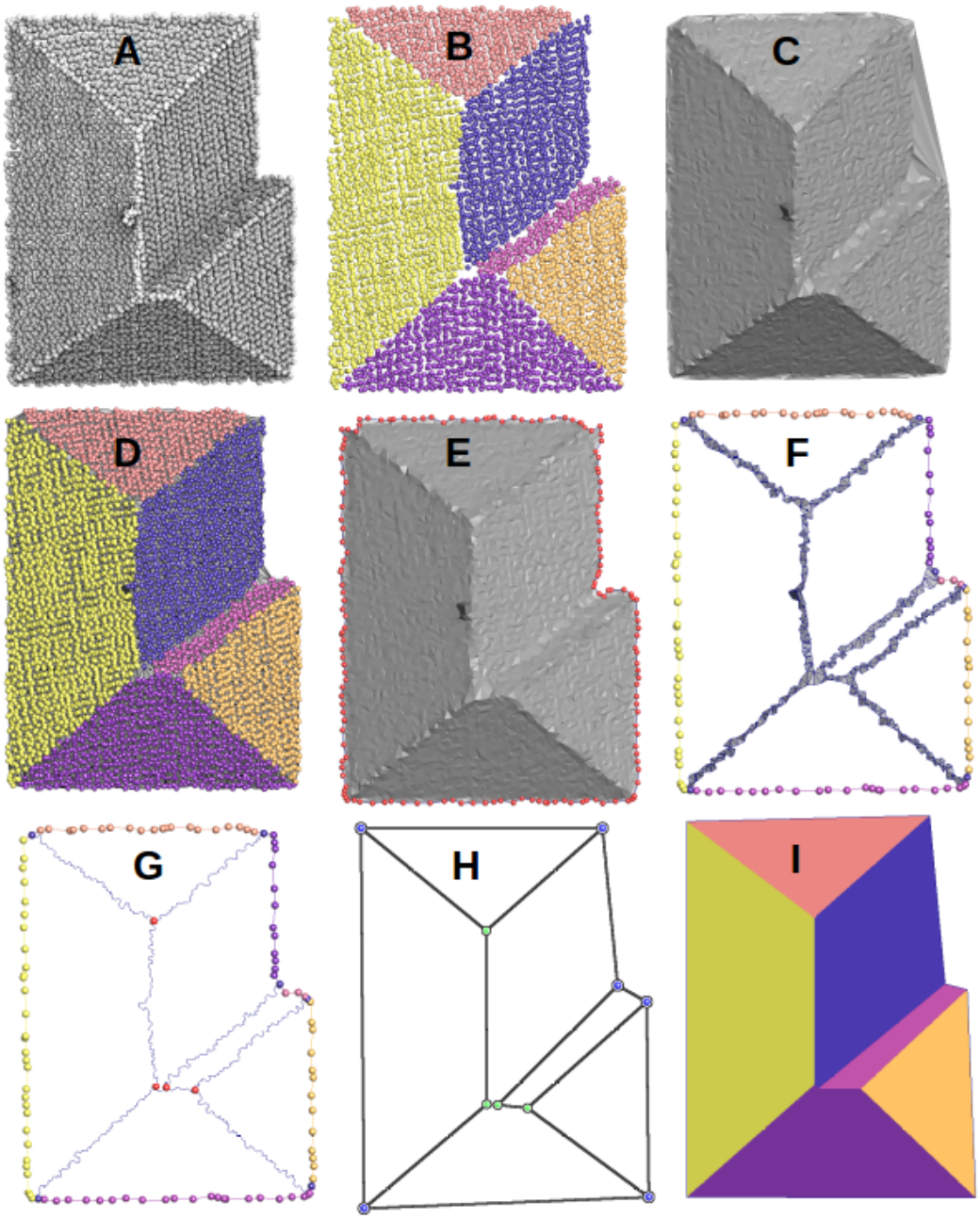


FIGURE 1.2 – **Pipeline détaillé.** A : nuage de points d'entrée. B : nuage de points labélisés, chaque point est étiqueté avec une valeur correspondant au versant planaire auquel il appartient. C : triangulation de Delaunay du nuage de points. D : α-shape de la triangulation et points labélisés. E : les sommets en rouge indiquent le contour du toit ; F : triangles des lignes de faîtage ; G : graphe topologique brut ; H : graphe simplifié ; I : maillage polygonal.

À la fin de cette étape, chaque nuage de points de toiture a ses points étiquetés avec une valeur correspondant au versant planaire auquel le point appartient, comme le montre la figure 1.2-B. Enfin, nous conservons les équations des plans estimés pour les utiliser dans une prochaine étape de la méthode.

1.1.3 Génération de modèles de toiture

À partir de la segmentation des versants de toits effectuée à l'étape précédente, nous nous intéressons à la construction d'un *descripteur de forme* pour la toiture. Pour cela, nous créons un graphe, dont les arêtes sont constituées des *bords extérieurs du toit*, ainsi que des *lignes de faîte* et dont les sommets, que l'on nomme *coins*, représentent l'intersection mutuelle des bords et des lignes de faîtage. Ici, nous nous intéressons à la génération d'un maillage triangulaire reliant tous les coins. Ensuite, ce maillage peut être utilisé pour extraire le modèle grossier du toit. Ce modèle initial sera optimisé géométriquement lors de la dernière étape, afin de fournir un modèle de toit valide.

Tout d'abord, nous projetons les points 3D d'entrée dans le plan xy en ignorant la coordonnée z. Comme les empreintes des bâtiments sont rarement des polygones convexes, ce maillage initial doit être *"creusé"* de l'extérieur afin de supprimer les triangles couvrant les zones concaves. Cela nous amène exactement à la définition de l'α-shape des points qui est habituellement calculée à partir du maillage de Delaunay en supprimant les triangles dont l'arête est plus longue qu'un paramètre $\alpha > 0$, [Edelsbrunner, 2010]. Indirectement, nous effectuons une approche pour creuser notre maillage telle qu'introduite dans [Duckham et al., 2008] et [Methirumangalath et al., 2015].

En pratique, nous avons implémenté un algorithme d'érosion et de détection de bordures. Nous commençons par construire la bordure convexe du toit sous la forme d'une liste d'arêtes, contenant uniquement les arêtes incidentes à un seul triangle dans la triangulation de Delaunay. Ensuite, nous considérons l'arête la plus courte et nous vérifions si elle est plus petite que la valeur seuil α. Si oui, l'arête est conservée et l'algorithme continue avec une de ses arêtes adjacentes. Sinon, l'arête est retirée de la liste de la bordure et nous ajoutons à sa place les deux autres arêtes du triangle considéré. L'algorithme s'arrête lorsque toutes les arêtes de la liste des bordures ont été vérifiées.

L'aspect le plus critique de cet algorithme consiste en la sélection d'une valeur optimale pour le paramètre α, puisqu'il dépend directement de la densité des points et du niveau de détail souhaité dans l'extraction de la bordure. Il est à noter qu'il existe une tentative récente [R.C. dos Santos and Carrilho, 2019] d'incorporer l'information de variation de densité dans l'algorithme dans le contexte particulier de la modélisation des bâtiments. À ce stade du traitement, nous disposons d'un maillage triangulaire dense qui correspond parfaitement aux données d'entrée, comme le montre la figure 1.2-D.

Notez que nous conservons les étiquettes des sommets calculées à l'étape précédente. Elles sont représentées par des couleurs différentes et indiquent à quel versant appartient chaque sommet. Cette information sera utilisée pour découvrir les lignes de faîte et les coins entre les plans du modèle polygonal. De plus, les arêtes du maillage constituant la bordure sont également marquées d'une valeur spécifique, leurs sommets sont indiqués en rouge sur la figure 1.2-E.

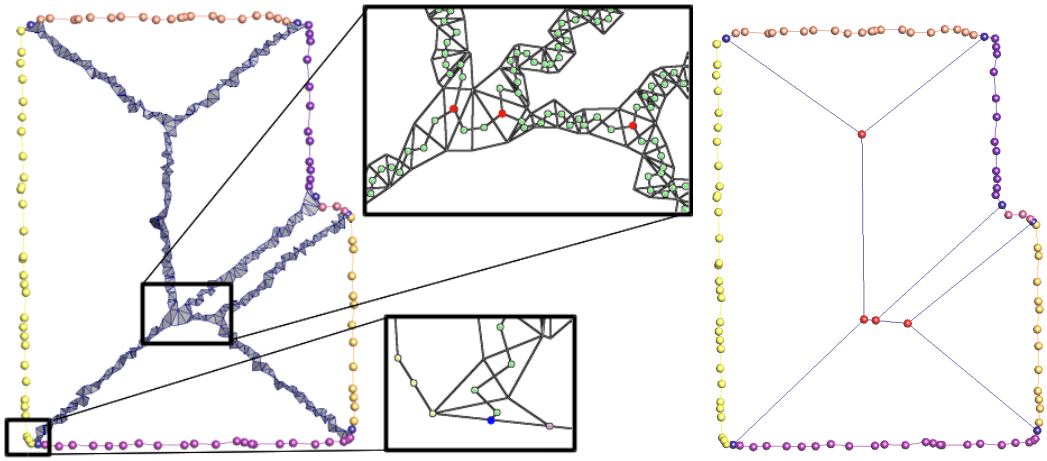


FIGURE 1.3 – À gauche, les arêtes du faîtage. Les sommets *coins* sont en rouge, le sommet *connexion* est en bleu. À droite, les arêtes simplifiées en supprimant les sommets *ordinaire*.

Nous supposons que l'empreinte du bâtiment, c'est-à-dire son contour n'est qu'une seule composante connexe. Dans le cas contraire, le traitement s'arrête et le segment de bâtiments doit être affiné afin de séparer correctement les différentes composantes non connexes.

Dans notre cas particulier, les lignes de faîtage sont définies par l'intersection de deux versants adjacents, tandis que les coins correspondent aux intersections d'exactement trois versants. Ces caractéristiques topologiques sont facilement accessibles à partir du maillage en tant qu'ensemble de triangles ayant plus de deux sommets d'étiquettes différentes, comme le montre la figure 1.2-F.

Cette simple observation nous permet de concevoir un algorithme combinatoire en 2D qui consiste en un simple parcours de maillage et ne nécessite aucun paramètre. Cela contraste fortement avec la plupart des méthodes de la littérature reposant généralement sur des calculs géométriques 3D (recherche de plans adjacents, calcul de la ligne d'intersection des plans, recherche des points d'extrémité de la ligne de faîte, *etc.*) qui sont connues pour être difficiles à mettre en œuvre de manière robuste [Devillers and Guigue, 2002].

En outre, en choisissant de travailler le plus longtemps possible en 2D, nous évitons de nombreuses configurations 3D complexes et reportons la prise en compte des sauts de hauteur de faîte à la dernière étape du pipeline, celle de la régularisation.

En pratique, il n'y a aucune difficulté technique à construire le graphe topologique à partir du maillage à condition que le maillage soit encodé dans une structure de données appropriée, par exemple la structure half-edges [Muller and Preparata, 1978, Kettner, 1999]. L'algorithme commence par ajouter au graphe les sommets et les arêtes de la bordure qui ont été préalablement étiquetés. Il ajoute ensuite les *half-edges* des triangles appartenant au faîtage, c'est-à-dire appartenant à deux ou trois versants. Cela consiste à ajouter un sommet au centre de chaque triangle et à relier les barycentres des triangles qui partagent une arête commune (voir le zoom sur la figure 1.3).

L'algorithme enregistre également, pour chaque barycentre, un attribut contenant les labels des points constituant le triangle. À partir de ces étiquettes, nous pouvons distinguer trois cas possibles :

— Si un sommet a exactement trois étiquettes, il est étiqueté comme *corner* du maillage et sera verrouillé et conservé dans le graphe.
— Si un sommet a exactement deux étiquettes, il est un étiqueté comme sommet *ordinaire* et pourra être supprimé pendant l'étape d'optimisation.
— Si un sommet possède une étiquette unique (les points sur la bordure ne possèdent pas d'étiquette), il s'agit d'un sommet de *connexion* et doit être préservé. Il est connecté à la bordure de la toiture en le projetant sur l'arête de la bordure la plus proche (qui est nécessairement l'une des arêtes du triangle auquel il appartient) (voir le zoom dans la figure 1.3).

Le graphe topologique brut résultant est illustré dans la figure 1.2-G où les *coins* sont représentés en rouge, tandis que les sommets de *connexion* sont en bleu. Les sommets *ordinaires* ne sont pas mis en évidence pour une meilleure visibilité, mais les arêtes entre chacun de ces sommets sont dessinées en bleu.

Remarquez qu'à ce stade du traitement, le graphe topologique est composé de nombreuses petites arêtes qui ont été extraites du maillage dense du α-shape. Cependant, nous sommes en mesure d'anticiper l'optimisation du modèle en supprimant simplement tous les sommets *ordinaires*, ce qui nous permet de simplifier les lignes des faîtages, comme le montre la figure 1.3.

1.1.4 Régularisation du modèle

Le modèle produit par les étapes précédentes possède déjà une topologie correcte, toutefois, il doit être optimisé géométriquement pour devenir un modèle de toit satisfaisant. Tout d'abord, nous devons simplifier la bordure du modèle, puis nous devons gérer la contrainte de planarité de ses faces et enfin, nous devons détecter les sauts de hauteur des faîtes.

Simplification du contour

Il existe plusieurs méthodes dans la littérature pour simplifier des polylignes. Si l'algorithme de Douglas-Peucker [Douglas and Peucker, 1973] est le plus connu, nous avons implémenté l'algorithme de Visvalingam [Visvalingam and Whyatt, 1992] pour sa simplicité. L'idée est très simple : il faut itérativement supprimer les points ayant le moins d'intérêt pour la polyligne en se basant sur des conditions locales. Pour cela, il suffit de calculer l'air du triangle formé par le point considéré et ses voisins immédiats et si le point présente la plus petite aire de la polyligne alors il est supprimé.

Optimisation de la géométrie du modèle

Nous disposons d'un maillage polygonal (non planaire) et des équations de plans cartésien. Chaque sommet de notre modèle à régulariser est incident à une, deux ou trois facettes.

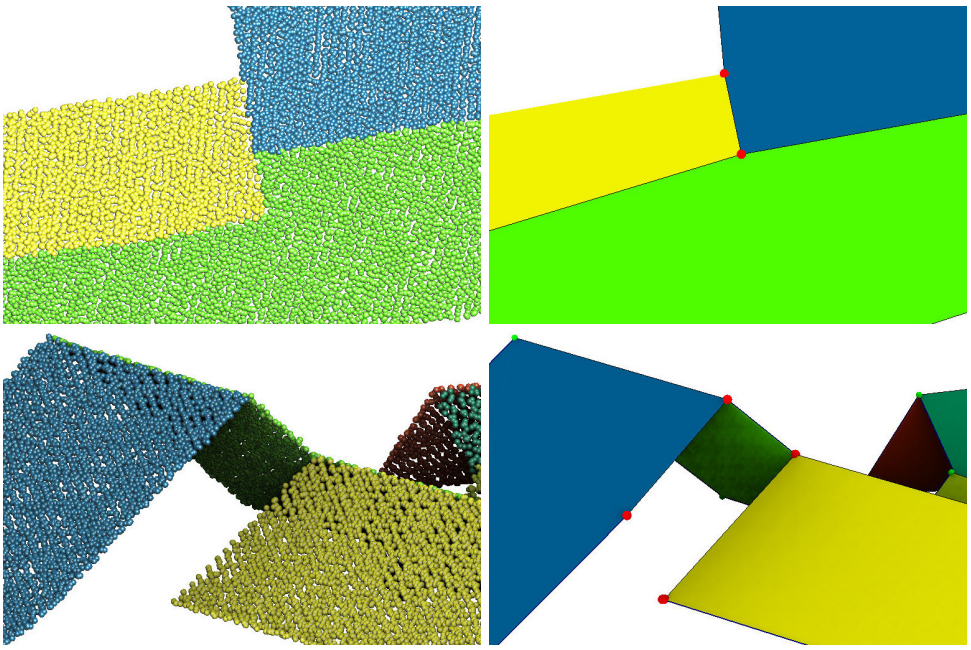


FIGURE 1.4 – Étapes de régularisation du modèle. **Ligne du haut :** nuage de points d'entrée vus du dessus et polygones 2D correspondants après la simplification. **Ligne du bas :** vue latérale des points d'entrée originaux et du modèle régularisé. Les points rouges montrent les sommets qui ont été dupliqués.

En fonction de la connectivité, nous déplaçons les sommets :
— *Une face incidente :* nous le projetons sur le plan correspondant.
— *Deux faces incidentes :* nous le projetons sur la ligne d'intersection.
— *Trois faces incidentes :* nous le déplaçons sur l'intersection des plans correspondants.

Bien que la projection soit conceptuellement simple, il faut faire attention. Reportez-vous à l'image en bas à gauche de la figure 1.4 pour une illustration. Les plans en jaune et en bleu ne se croisent pas, mais l'α-shape 2D ne permet pas de récupérer cette information. Considérons les sommets surlignés en rouge dans l'image supérieure droite de la figure ; si nous les projetons sur les deux plans simultanément, nous obtenons un résultat incohérent. Heureusement, il est très facile de détecter cette incohérence.

En effet, une incohérence peut être détectée grâce à la distance entre la position originale du point et son éventuelle nouvelle position. Si cette distance dépasse un certain seuil, alors il s'agit d'une erreur. Si le point a une valence de trois, alors parmi les plans adjacents, nous identifions une paire de plans qui semblent être les plus parallèles (ceux qui vont générer la plus grande incohérence). Et nous dupliquons le point de manière à séparer les deux plans. S'il a une valence de deux, nous dupliquons simplement le point et séparons les deux plans comme le montre l'image en bas à droite.

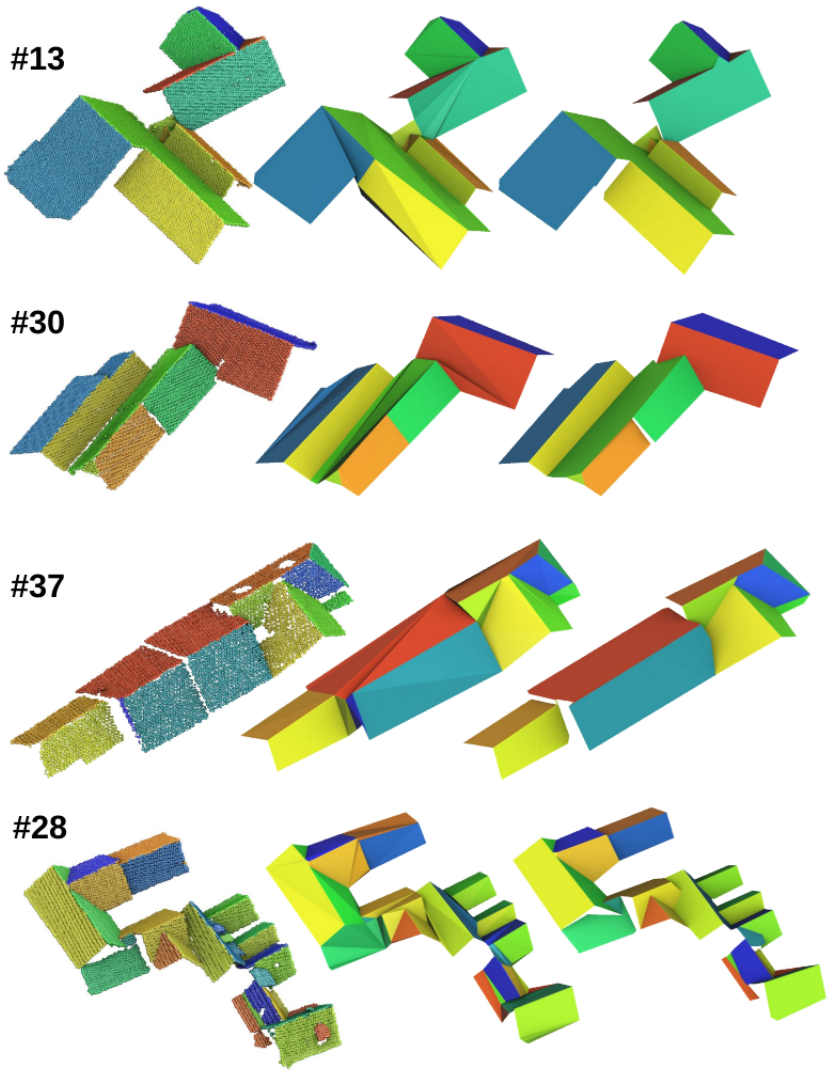


FIGURE 1.5 – Modèles possédant des faîtes à saut de hauteur. Première colonne : points annotés, deuxième colonne : graphe topologique, troisième colonne : modèle après régularisation.

Ainsi, l'étape de régularisation peut être vue comme la boucle suivante :

1. projection de tous les sommets sur les plans correspondants ;
2. s'il n'y a pas de configuration incohérente, arrêtez-vous ;
3. sinon dupliquer les sommets incohérents et passer à l'étape de projection.

Il est remarquable de constater que l'algorithme récupère les incohérences qui correspondent aux sauts de hauteur des lignes de faîte. La détection des sauts de hauteur des lignes de faîte est un problème difficile et notre approche combinatoire est plus robuste que les algorithmes basés sur la géométrie, comme les α-shape 3D ou les contours de toits individuels. La figure 1.5 présente quelques exemples de l'algorithme de régularisation.

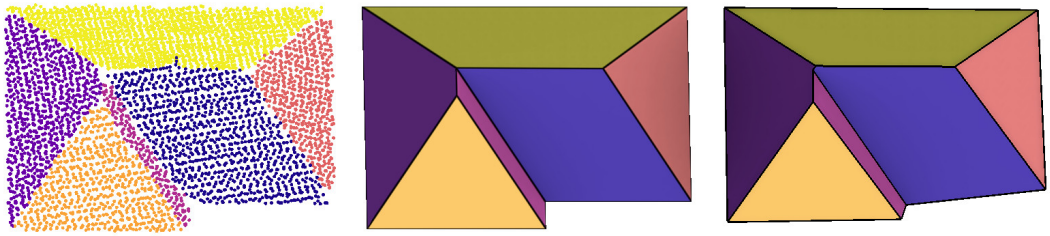


FIGURE 1.6 – **Gauche :** ensemble de points d'entrée étiquetés ; **Milieu :** modèle de toit construit manuellement ; **Droit :** reconstruction automatique.

Densité brute	Densité moyenne	Divergence du faisceau	Précision en x,y	Précision en z	Distance moyenne entre deux points
28 pts/m^2	30 pts/m^2	0.25 mrad	10 cm	6 cm	20 cm

TABLE 1.1 – Paramètres de l'acquisition LiDAR.

1.2 Résultats

1.2.1 Description du jeu de données

Le jeu de données utilisé pour nos tests a été capturé au-dessus de la ville de Breuschwickersheim de l'Eurométropole de Strasbourg en France à l'aide du scanner laser LMS Q780 de Riegl (voir tableau 1.1). Il couvre une surface d'environs 1000 $m \times$ 1000 m.

À partir de la classification du nuage de points, nous avons pu extraire un ensemble de 2.5M de points correspondant aux toitures des bâtiments de l'ensemble de la zone (figure 1.1). Ce nuage de points a été ensuite divisé en segment correspondant à des bâtiments individuels, ou des groupes de bâtiments mitoyens. Nous avons obtenu 445 segments en utilisant un algorithme d'extraction de segments par distance euclidienne.

Chaque segment résultant correspond au toit d'un seul bâtiment ou d'un ensemble de bâtiments adjacents (figure 1.1). Comme nous ne traitons pas le problème de détection de toiture au sein d'un nuage de points, nous supposons que la délimitation des bâtiments est correcte. Nous avons conservé les grands segments tels qu'ils ont été livrés, sans les couper manuellement. Cela nous permet de fournir des données d'entrée raisonnablement difficiles à notre méthode de modélisation des toits.

Pour avoir une donnée de référence, un opérateur humain a construit manuellement les modèles polygonaux pour chaque segment montré dans la figure 1.1 en utilisant le logiciel RhinoCapture [RhinoTerrain, 2022]. Un exemple de toit de référence est donné dans la figure 1.6.

1.2.2 Évaluation quantitative

Nous avons utilisé le système d'évaluation proposé dans [Rutzinger et al., 2009, Awrangjeb and Fraser, 2014] qui suppose que les toits sont représentés par des modèles polygonaux et que chaque toit individuel est constitué d'un ensemble de plans. Nous avons effectué une *évaluation basée sur l'objet au niveau du plan* puisque,

	C_m	C_r	Q_l
Moyenne	94.0	92.7	90.8
Médiane	100	100	100
Premier quartile	91.2	89.6	80.8

TABLE 1.2 – Évaluation de la précision de la détection des plans sur l'ensemble des données (445 segments de bâtiments).

rappelons-le, notre méthode ne traite que le problème de la modélisation des toits en détectant les plans dans des groupes de bâtiments déjà segmentés.

Dans l'évaluation basée sur les objets, *complétude* (C_m), *exactitude* (C_r), et *qualité* (Q_l) sont estimés en comptant le nombre de vrais positifs (VP), faux positif (FP) et faux négatifs (FN) dans les résultats extraits, comme suit :

$$C_m = TP/(TP + FN), \quad C_r = TP/(TP + FP), \quad Q_l = TP/(TP + FN + FP)$$

- La *complétude* indique le taux de détection et est le pourcentage d'entités dans les données de référence qui ont été détectées.
- L'*exactitude* indique dans quelle mesure les entités détectées correspondent aux données de référence et est étroitement liée au taux de faux négatif.
- La *qualité* fournit une mesure de performance composée qui équilibre *complétude* et *exactitude*.

Comme le montre le tableau 1.2, les métriques moyennes de complétude, d'exactitude et de qualité, calculées sur l'ensemble du jeu de données (445 nuages de points de bâtiments), étaient respectivement de 94%, 92,7% et de qualité 90,8%.

En plus de ces résultats quantitatifs, une analyse qualitative est également présentée via une visualisation (voir figure 1.7). Le tableau 1.3 donne les métriques d'évaluation détaillées pour les segments représentés sur les figure 1.5, figure 1.7 et figure 1.8. Tous les clusters correspondant à des bâtiments individuels ont été parfaitement reconstruits (voir figure 1.7). Les grands clusters correspondant à des bâtiments adjacents ont également été correctement traités (voir figure 1.7 :#10,#23). Dans le cas de clusters présentant des sauts de hauteurs (figure 1.5), certains vrais plans ont été manqués et certains plans de toit ont été mal reconstruits. Les quatre derniers clusters (figure 1.8) sont considérés comme des cas d'échec qui sont principalement dus à une mauvaise segmentation des plans.

1.3 Analyse et résultats

La méthode proposée vise à construire des modèles de toit avec un niveau de détail LOD2, tel que défini dans la norme CityGML [Gröger and Plümer, 2012]. Les petites superstructures de toit telles que les cheminées, les antennes et les divers composants bruyants sont pour la plupart supprimés par l'*algorithme de segmentation en plan* (étape 1). Cependant, certaines petites extensions de toit et la structure détaillée du toit ont été modélisées, comme le montre la figure 1.7 :#105.

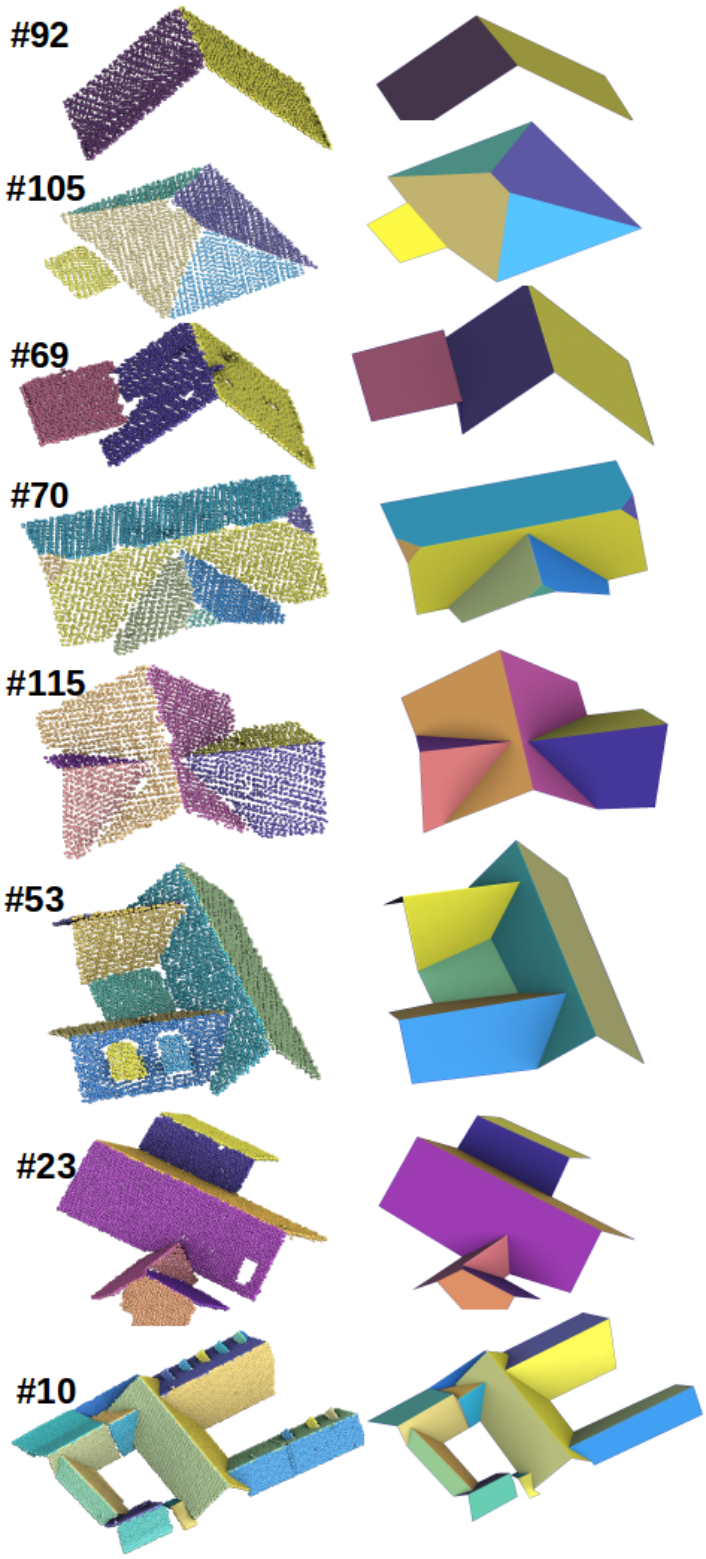


FIGURE 1.7 – Modèles résultants pour 8 segments.

#	#ref	VP	FP	FN	C_m	C_r	Q_l
92	2	2	0	0	100	100	100
105	5	5	0	0	100	100	100
69	3	3	0	0	100	100	100
70	7	7	0	0	100	100	100
115	6	6	0	0	100	100	100
53	7	7	0	0	100	100	100
23	8	8	0	0	100	100	100
10	18	18	0	0	100	100	100
13	13	13	1	0	100	92.9	92.9
30	9	8	0	1	88.8	100	88.8
37	13	11	1	2	84.6	91.7	78.6
28	19	18	1	1	94.7	94.7	90
151	7	7	4	0	100	63.6	63.6
214	3	3	2	0	100	60	60
215	9	8	0	1	88.9	100	88.9
256	2	2	1	0	100	66.7	66.7

TABLE 1.3 – Mesures d'évaluation détaillées pour les toits des figures (1.7, 1.5 et 1.8).

Les lucarnes sont des éléments de toit définissant de petits plans qui sont topologiquement *intérieur*s aux plans de toit primaires. Les grandes lucarnes sont reconnues et délimitées avec précision par notre algorithme de l'étape 1, comme le montre la figure 1.7 :#53. Mais ces structures sont généralement supprimées par l'algorithme de génération de modèle (étape 2).

Rappelons que notre *algorithme de génération de modèles* fonctionne en 2D, ce qui nous permet, d'une part, de concevoir un algorithme combinatoire robuste évitant de nombreux problèmes 3D, d'autre part, de reporter les problèmes d'intégration 3D dus aux sauts de hauteur. Cela rend notre méthode capable de traiter les toits à géométrie simple de manière assez efficace. Pour les toits plats simples, les toits à pignons et les toits en croupe, l'algorithme a atteint 100 % de complétude, d'exactitude et de qualité. Nos données contiennent 40 % de ce type de structure. Pour les toits à pans coupés, à pignons croisés, à croupe et à noue et les toits entrecroisés, la méthode a permis d'obtenir 98% de modèles parfaits. Ces formes constituent environ 30% de notre jeu de données (voir figure 1.7). Comme le montre la figure 1.5, la plupart des structures complexes avec des sauts de hauteur sont correctement optimisées par notre algorithme *régularisation de modèle*. Les 30% restants sont des structures complexes, 20% d'entre elles sont traitées avec succès et les modèles résultants sont acceptables pour une utilisation ultérieure (voir figure 1.7 :#10,#23), tandis que 10% de nos données sont des échecs réels qui sont principalement dus à une mauvaise segmentation des plans, comme le montre la figure 1.8.

1.4 Conclusion

Dans ce chapitre, nous avons présenté un pipeline pour la reconstruction de toits de bâtiments à partir de données ponctuelles LiDAR préclassifiées. La méthode prend en entrée un groupe de points 3D correspondant au toit d'un bâtiment individuel. Elle fournit un modèle polygonal 3D où chaque parcelle de toit planaire est représentée

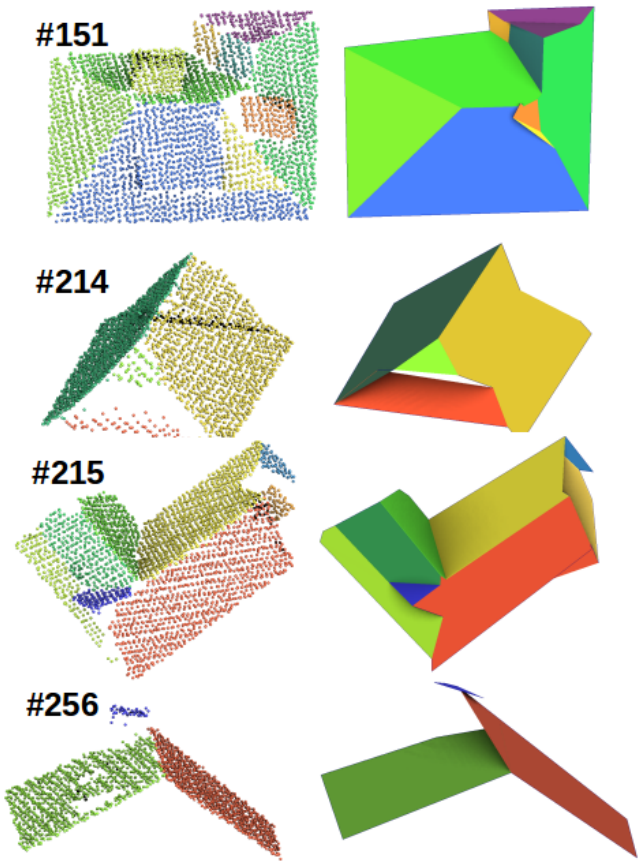


FIGURE 1.8 – Exemples de cas d'échec dus à une segmentation incorrecte du plan.

par un polygone plat et où les intersections des faîtes et des coins sont topologiquement cohérentes. La principale contribution est une approche combinatoire qui est plus robuste que les méthodes géométriques essayant de fixer les lignes d'intersection. Grâce aux informations combinatoires, nous pouvons facilement reconstruire les bords des marches pendant la phase de régularisation. Des tests expérimentaux ont montré que le pipeline proposé traite avec succès des données d'entrée difficiles. Il atteint un taux de détection des plans de 94% et un taux de qualité de 90%.

Pour autant, même si les toitures ont été reconstruites, elles ne respectent pour la plupart pas les critères nécessaires à la réalisation d'une bonne simulation. Ce genre de résultats peuvent se retrouver assez facilement dans un ensemble de modèles numériques urbain reconstruit automatiquement [Ledoux, 2018]. En effet, ces modèles présentent différentes anomalies récurrentes d'après les articles [Biljecki et al., 2016a, Ledoux, 2018]. Nous pouvons citer les erreurs suivantes, le mot surface fait référence à un versant de toiture :

— Les surfaces définissant les plans de toits possèdent moins de trois points ;
— Les surfaces possèdent des points dupliqués ;
— La surface présente un trou ;
— La surface intersecte une autre surface ;
— La surface est mal orientée ;
— La surface présente de longs triangles, d'air négligeable.

Nous proposons donc d'aborder le problème avec un différent point de vue dans le chapitre suivant.

Chapitre 2

Ajustement de surface paramétrique à des nuages de points

Sommaire

Dans le chapitre précédent, nous avons constaté que la reconstruction d'un modèle numérique urbain est possible au moyen des approches basées sur les données. Cependant, certains modèles de bâtiments obtenus avec ce type d'approche présentent divers problèmes suffisamment contraignants pour rendre les MNU inexploitables pour la simulation numérique. Notons que les problèmes les plus courants sont les suivants : des toits dont les versants ne sont pas planaires, mal alignés ou dont les sommets ne sont pas connectés, ou encore l'absence de certains éléments constitutifs des bâtiments (voir section 1.3). De plus, bien que les bâtiments semblent très réguliers lorsqu'ils sont visualisés, ils présentent différents défauts dus au bruit naturellement présent dans les nuages de points LiDAR. Par exemple, les sommets représentant le faîtage du toit n'ont pas la même hauteur où les toits à pignons ne sont plus symétriques, *etc.* Certains pans de toits peuvent également ne pas être reconstruits pour cause de données lacunaires. Enfin, les maillages obtenus sont difficilement modifiables par un utilisateur lambda, tandis qu'un maillage contraint serait plus facilement contrôlable et modifiable interactivement grâce à des outils de modification locale en s'appuyant sur ces mêmes contraintes.

D'après Biljecki et al. [Biljecki et al., 2016a], il n'existe malheureusement pas de solution universelle permettant de réparer automatiquement ces maquettes numériques erronées, même si quelques propositions ont été faites [Zhao et al., 2013]. Il serait donc bénéfique de se concentrer sur la gestion de ces erreurs durant la phase de création des maillages, en empêchant, par exemple, certaines configurations problématiques.

Fort de ces informations, nous avons créé une nouvelle méthode de reconstruction de bâtiment à partir de nuage de points LiDAR, en mettant l'accent sur la robustesse aux données aberrantes. Pour cela, nous mettons à profit les compétences de la société RhinoTerrain dont la reconstruction semi-automatique à partir de modèle de toiture est le cœur de métier. À l'aide de leur bibliothèque de modèles, nous injectons de l'information là où elle est bruitée ou lacunaire et reconstruisons ainsi la ville telle qu'un architecte l'aurait élaborée. De plus, cette méthode a été conçue pour être intégrée au sein d'un logiciel et être la plus interactive possible, tout en limitant le nombre de paramètres utilisés. Nous proposons donc une alternative qui considère la reconstruction du toit comme un problème d'optimisation unique. L'avantage de cette approche est que la détermination de chaque caractéristique prend directement en compte le fait que l'objet que nous recherchons est une instance d'un modèle de toit donné, que nous supposons connu. Dans notre cas, le modèle de toit est une surface polygonale soumise à certaines contraintes comme la planarité du toit, l'alignement des gouttières ou l'horizontalité du faîte du toit (voir figure 2.2).

Dans ce chapitre, nous étendons le travail de [Nivoliers et al., 2014] à l'ajustement de surface paramétrique à des nuages de points. Notre idée consiste à minimiser une fonction de distance définie entre une toiture et un nuage de points : nous souhaitons que la surface soit aussi proche que possible de chaque point du nuage de points et que les points du nuage soient également proches de la surface.

Nous avons mis au point une méthode variationnelle d'ajustement de surface paramétrique à des nuages de points (voir figure 2.1). Ce chapitre est organisé de la manière suivante : après un bref aperçu des travaux connexes §2.1, nous formulons l'ajustement de modèle de toit comme un problème d'optimisation puis nous développons notre méthode de résolution §2.2. Enfin, le comportement de la méthode est évalué afin d'établir ses limites §3.4.

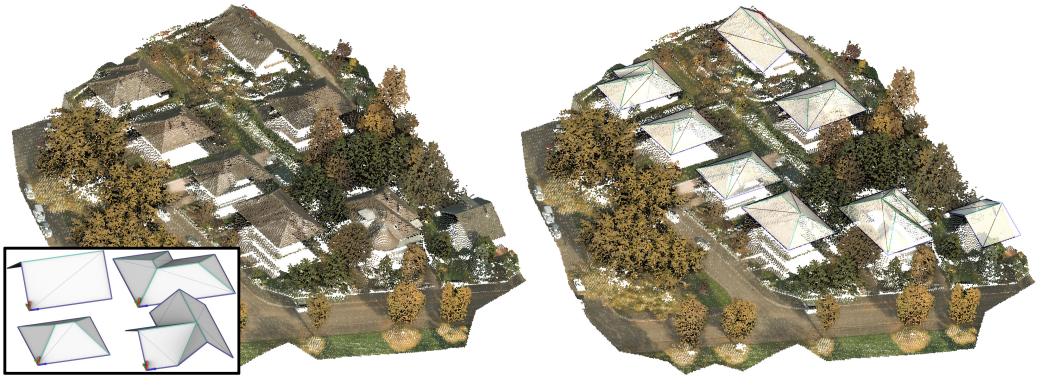


FIGURE 2.1 – **Ajustement de modèles de toit par notre méthode**. Un nuage de points LiDAR de la ville de Strasbourg est présenté à gauche [Strasbourg, 2016]. À droite, des modèles de toits paramétriques sont ajustés aux données correspondantes dans le nuage de points par notre méthode. Ici, chaque modèle a été sélectionné par un opérateur humain et initialisé à proximité d'une toiture dans le nuage de points, puis ajusté sur l'ensemble du nuage. Les points appartenant à des objets autres que des toits, comme le sol ou les arbres, sont considérés comme des valeurs aberrantes par notre méthode. Le cadre inférieur gauche présente quelques exemples de modèles de toit utilisés.

2.1 Fondements et état de l'art

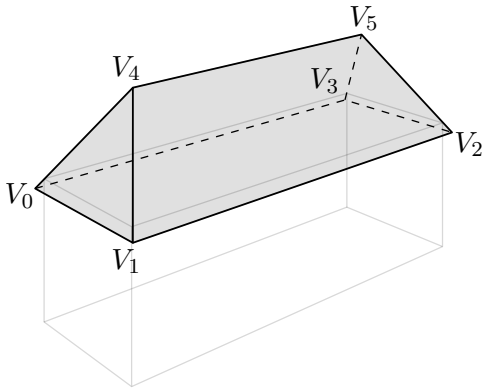
Dans cette section nous présentons une formulation de notre problème ainsi que la littérature des méthodes d'ajustement de surface à des nuages de points.

2.1.1 Formalisation du problème

Nous appelons modèle paramétrique un maillage triangulé défini par l'utilisateur et soumis à diverses contraintes géométriques. Bien que ce cadre soit large et puisse s'adapter à une grande variété de problèmes, nous nous limiterons à l'étude de la reconstruction de bâtiments. Nos modèles s'apparentent à des toitures et leur maillage associé est homéomorphe à un disque [6]. Les contraintes géométriques sur ces modèles reflètent la planarité des pans de nos toitures, l'alignement des gouttières, l'horizontalité des faîtières, *etc.* Elles peuvent être exprimées comme des égalités ou des inégalités reliant les coordonnées des sommets du modèle (voir figure 2.2).

Les données que nous utilisons sont des acquisitions LiDAR brutes de régions d'intérêt (par exemple un quartier). En plus du toit que nous souhaitons ajuster, les nuages de points contiennent d'autres structures, telles que des cheminées, des antennes, des arbres, des surfaces au sol ou même d'autres toitures. La méthode d'ajustement doit donc être robuste aux valeurs aberrantes naturellement présentes dans le nuage de points.

6. En topologie, un homéomorphisme est une application bijective continue dont la bijection réciproque est également continue. En d'autres termes, nous souhaitons ici que toutes nos surfaces soient équivalentes par déformation continue à un disque.



$$\begin{cases} (V_3 - V_0) = (V_2 - V_1) \\ (V_3 - V_0) \cdot (V_1 - V_0) = 0 \\ (V_3 - V_0) = \lambda(V_5 - V_4), \quad 0 < \lambda < 1 \end{cases}$$

FIGURE 2.2 – Un modèle de toit est une surface polygonale (dans cet exemple, il possède deux faces triangulaires (V_0, V_1, V_4) et (V_3, V_2, V_5), et deux faces trapézoïdales (V_1, V_2, V_5, V_4) et (V_0, V_3, V_5, V_4)). Il peut s'adapter à différents nuages de points de toits en modifiant la position de ses sommets et en respectant un ensemble de contraintes. Chacune des contraintes exprime dans l'ordre : les vecteurs $\overrightarrow{V_0V_3}$ et $\overrightarrow{V_1V_2}$ sont égaux, puis, les vecteurs $\overrightarrow{V_0V_3}$ et $\overrightarrow{V_0V_1}$ sont perpendiculaires et enfin les vecteurs $\overrightarrow{V_0V_3}$ et $\overrightarrow{V_4V_5}$ sont colinéaires et la faitière est plus courte.

2.1.2 Notations

Nous détaillons dans cette sous-section les notations utilisées dans ce chapitre.

Nuage de points

P	le nuage de points fixé ;
p	un point appartenant à P ;
$\pi_P(q)$	le point le plus proche d'un point q appartenant à S dans P ;

Surface

S	la surface mobile à ajuster ;
q	un point appartenant à S ;
V_i	le sommet i de S ;
V_{ij}	la coordonnée d'indice j du sommet i ;
f	une face de S ;
$\pi_S(p)$	le point le plus proche de p sur S ;

Optimisation

V	la concaténation des V_i de S en un vecteur de taille $\mathbb{R}^{3n}$;
R	matrice de rotation de la transformation rigide de la base locale de S vers la base canonique de $\mathbb{R}^3$;
T	vecteur de translation de la transformation rigide de la base locale de S vers la base canonique de $\mathbb{R}^3$;
S_i	coordonnées d'un sommet V_i exprimé dans la base locale de S ;
M	matrice des contraintes linéaires de S ;
Y	ensembles de variables réduites, tel que $V = MY$;

2.1.3 État de l'art : Reconstruction de surfaces à partir de modèles

Dans cette section, nous passerons en revue les méthodes d'ajustement de surface contrainte, puisque nos modèles de toits sont contraints dus à la géométrie inhérente aux toits. Cette étude vient compléter l'état de l'art proposé dans le chapitre précédent qui était uniquement centré sur la reconstruction de toit à partir de données LiDAR.

Pour trouver la déformation d'un modèle, la plupart des travaux [Tam et al., 2013] reposent sur l'algorithme d'optimisation Iterative Closest Point (ICP). Le problème réside en la minimisation d'une distance définie entre une surface et un nuage de points, permettant de trouver une transformation rigide et optimale de la surface [Besl and McKay, 1992, Segal et al., 2009]. Il est également possible de considérer des déformations locales de la surface [Stoll et al., 2006], mais cela nécessite souvent l'instanciation de repères de correspondance par un utilisateur ou de les détecter automatiquement [Rusu et al., 2009].

Une transformation non rigide doit préserver la forme globale de la surface ; ainsi, la déformation doit être localement proche d'une isométrie, c'est-à-dire qu'elle peut être considérée localement comme une transformation rigide de la surface originale [Sorkine and Alexa, 2007, Sumner et al., 2007, Li et al., 2009]. Cette approche fonctionne assez bien pour les maillages denses, mais minimiser la déformation de la surface n'est pas un objectif valable pour toutes les applications (y compris la reconstruction de toits). Pour les modèles articulés, nous envisageons plutôt une transformation rigide par os [Pellegrini et al., 2008] et, pour les modèles CAO, [Buonamici et al., 2018] travaille avec une forme décrite par un ensemble de paramètres (rayon du trou, longueur du bord, *etc.*). Ceci est très proche de notre problème, nos gabarits et contraintes étant très similaires. Notre méthode présente l'avantage de gérer les nuages de points bruités et contenant des valeurs aberrantes, et a été conçue pour pouvoir utiliser des solveurs efficaces.

Une autre façon d'ajuster une surface est présentée dans [Nivoliers et al., 2014] : elle minimise une fonction de distance symétrique entre les deux surfaces et régulière. Elle peut être directement optimisée par L-BFGS [Liu and Nocedal, 1989]. Notre méthode est assez similaire, mais supporte les contraintes du modèle de toit ainsi que le rejet des valeurs aberrantes grâce à une approche robuste de M-estimateur.

2.2 Contributions

Dans cette section, nous décrivons notre problème comme la minimisation d'une fonction d'énergie. Cette fonction est composée de la somme de deux termes définis respectivement dans les sous-sections § 2.2.2 et § 2.2.3. Les contributions suivantes ont fait l'objet d'une publication [Coiffier et al., 2021].

2.2.1 Vue d'ensemble

Pour pouvoir formuler le problème comme une optimisation, nous avons besoin de définir une distance entre le nuage de points P et une surface S.

Une distance naturelle lorsque l'on estime une distance entre deux objets géomé-

triques est la distance de Hausdorff, définie par la formule suivante :

$$H(S,P) := \max\left(\sup_{\mathbf{q}\in S}\inf_{\mathbf{p}\in P}||\mathbf{p}-\mathbf{q}||,\ \sup_{\mathbf{p}\in P}\inf_{\mathbf{q}\in S}||\mathbf{p}-\mathbf{q}||\right)$$

La distance de Hausdorff est un maximum entre deux termes : la minimisation de cette distance assure que chaque point de la surface S n'est pas trop loin du nuage de points P, et réciproquement que chaque point de P n'est pas trop loin de la surface. Cependant, en présence de valeurs aberrantes, la distance de Hausdorff est arbitrairement grande. De plus, nous souhaitons que notre fonction objectif soit au moins de classe C^1 pour utiliser un optimiseur efficace, ce qui n'est pas le cas de la distance de Hausdorff.

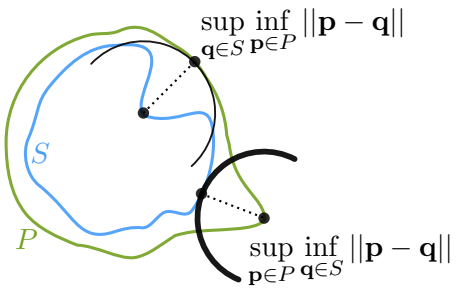


FIGURE 2.3 – Illustration de la distance de Hausdorff.

Dans notre problème, le nuage de points P est fixé. Nous proposons une énergie se comportant d'une manière similaire à la distance de Hausdorff sous la forme d'une énergie $E(S)$ permettant de mesurer la distance entre une surface (mobile) S et un nuage de points (fixe) P dont la minimisation conduit à l'ajustement final.

Nous définissons l'énergie comme suit :

$$E(S) := \mathcal{F}(S) \ + \ \gamma\, \mathcal{G}(S) \tag{2.1}$$

où $\mathcal{F}$ représente la distance du nuage de points à la surface, $\mathcal{G}$ la distance de la surface au nuage de points et le paramètre γ le poids de $\mathcal{F}$ par rapport à $\mathcal{G}$ (voir sous-section 2.2.3). Pour définir les fonctions $\mathcal{F}(S)$ et $\mathcal{G}(S)$, nous avons besoin d'introduire deux fonctions de projection. Soit $\mathbf{q}$ un point de la surface S, et $\mathbf{p}$ un point du nuage de points P. Nous définissons les fonctions de projection suivantes :

— $\pi_P(\mathbf{q}) := \text{argmin}_{\mathbf{p}\in P}\,||\mathbf{p}-\mathbf{q}||$ est la projection de $\mathbf{q}$ dans le nuage de points, c'est-à-dire le point le plus proche de $\mathbf{q}$ dans P ;
— $\pi_S(\mathbf{p}) := \text{argmin}_{\mathbf{q}\in S}\,||\mathbf{q}-\mathbf{p}||$ est la projection de $\mathbf{p}$ sur la surface, c'est-à-dire le point le plus proche sur S de $\mathbf{p}$.

La figure 2.4 présente les deux fonctions de projection. Nous allons définir rapidement les deux termes $\mathcal{F}(S)$ et $\mathcal{G}(S)$ comme les intégrales de ces projections, elles peuvent être estimées de manière efficace grâce à des diagrammes de Voronoï restreints.

2.2.2 Estimation de la distance de P à S

Ajuster une primitive surfacique à un nuage de points peut être considéré comme un problème spécifique de régression en trois dimensions. Contrairement aux primitives simples comme les plans ou les sphères, nos modèles de toit ne peuvent pas

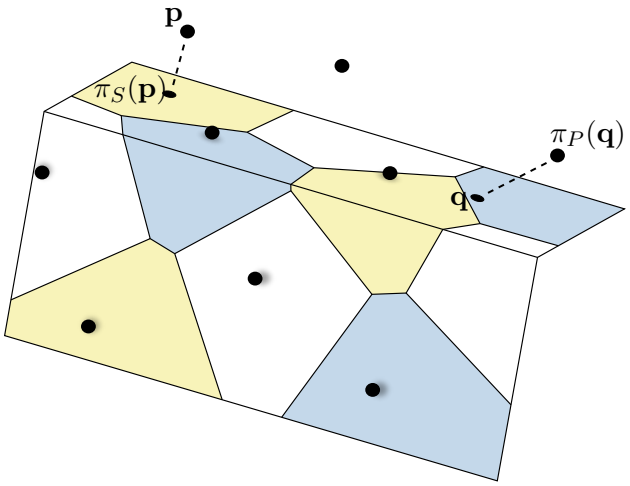


FIGURE 2.4 – Notre énergie mesure la distance entre le nuage de points P (points noirs) et une surface S (le toit à deux pans). Pour calculer la distance, nous avons besoin des fonctions de projection π_P et π_S donnant respectivement, le point le plus proche du nuage de points et de la surface. Ensuite, nous intégrons les fonctions pour chaque point de la surface (respectivement du nuage de points). Les cellules du diagramme de Voronoï restreint sont représentées en couleur.

être ajustés à l'aide d'une expression de forme close. Cela dit, estimer la somme des distances aux carrés entre les points $\mathbf{p}$ du nuage de points P et leur projection $\pi_S(p)$ sur S est une façon naturelle d'attirer notre surface vers le nuage de points. Mais cette fonction présente deux inconvénients majeurs : elle n'est pas dérivable en tout point et tout comme la distance d'Hausdorff, elle est arbitrairement grande lorsque le nuage de points possède des valeurs aberrantes éloignées.

La question de la robustesse aux valeurs aberrantes pour un problème de régression a été largement étudiée dans divers travaux sur les M-estimateurs et méthode des moindres carrés robustes [Holland and Welsch, 1977]. Albert Beaton et John Tukey ont introduit un M-estimateur, le biweight de Tukey, abréviation de bisquare weight [Beaton and Tukey, 1974] défini par :

$$\psi : x \mapsto \begin{cases} x(1-\frac{x^2}{\delta^2})^2 & \text{si } |x| < \delta \\ 0 & \text{sinon.} \end{cases}$$

Nous définissons notre premier terme d'énergie $\mathcal{F}$, tel que :

$$\mathcal{F}(S) := \sum_{p \in P} \Psi\left(||p - \pi_S(p)||^2\right) \tag{2.2}$$

avec, Ψ une primitive de la fonction biweight de Tukey, telle que :

$$\Psi : x \mapsto \begin{cases} 1-(1-\frac{x^2}{\delta^2})^3 \text{ si } |x| < \delta \\ 1 \text{ sinon} \end{cases} \tag{2.3}$$

Contrairement à de nombreuses autres fonctions utilisées pour les M-estimateurs, le biweight de Tukey est de classe C^2 et correspond donc à nos exigences de dériva-

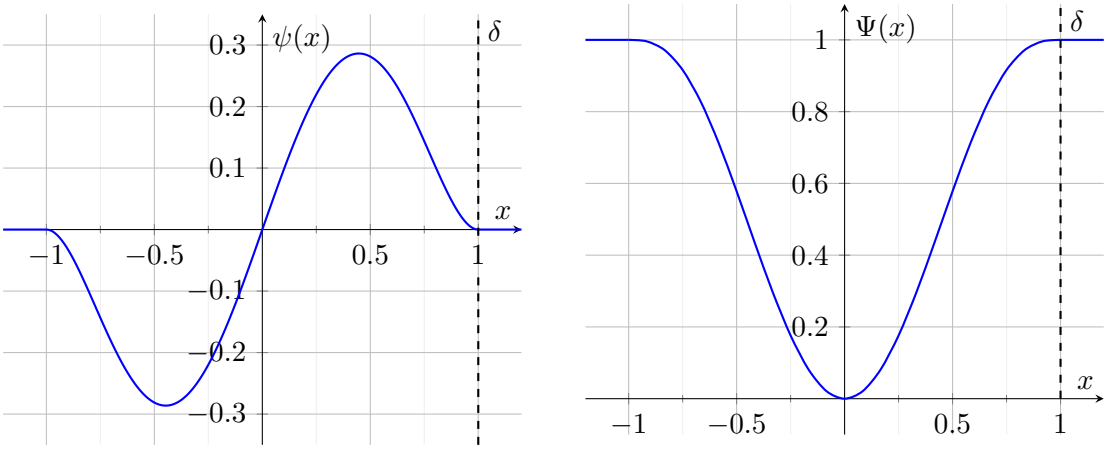


FIGURE 2.5 – Fonction biweight de Tukey ψ pour $\delta = 1$ (à gauche) et sa primitive Ψ, tel que $\Psi(0) = 0$ (à droite).

bilité. La fonction biweight ψ présente plusieurs propriétés intéressantes dans notre cas. Premièrement, $\Psi(0) = 0$ et $\psi(0) = 0$, ce qui signifie que l'énergie associée aux points proches de la surface est faible et a un faible gradient.

Deuxièmement, $\Psi(x) = 1$ et $\psi(x) = 0$ pour tout $x > \delta$, ce qui signifie que δ définit naturellement un seuil pour les points à considérer comme aberrants et ne contribuant pas à l'évolution de l'énergie. En pratique, δ est estimé comme la distance moyenne entre les points et leurs k voisins les plus proches du nuage de points (et nous obtenons empiriquement les meilleurs résultats en prenant $k = 10$). La minimisation du terme $\mathcal{F}(S)$ garantit que le modèle est étiré pour capturer toute l'étendue des points du toit sans être sensible aux points non pertinents et trop éloignés.

Cependant, en partant d'une position où chaque point de P est à une distance de la surface S supérieure à la valeur δ, le gradient sera nul et il ne permettra pas de converger sans l'ajout d'un autre terme dans la fonction d'énergie. Il convient donc d'ajouter un second terme afin de pallier à ce problème de convergence.

2.2.3 Terme symétrique : distance de S à P

Alors que la minimisation du terme $\mathcal{F}$ permet de trouver une position et une orientation satisfaisante de la surface S, l'approche par régression ne permet pas d'ajuster les bords de la surface à la limite du toit compris dans le nuage de points (voir figure 2.6). En effet, les parties de la surface éloignées du nuage ne sont pas pénalisées par le terme $\mathcal{F}$. Pour éviter cette situation, nous ajoutons un second terme $\mathcal{G}$ à notre fonction d'énergie, définie comme suit :

$$\mathcal{G}(S) := \int_S ||q - \pi_P(q)||^2 dS(q) \tag{2.4}$$

Ce terme d'énergie est inspiré de l'algorithme *Voronoï Square Distance Minimization (VSDM)* [Nivoliers et al., 2014], où une énergie similaire est utilisée pour ajuster une surface à une autre surface. Comme le montre la figure 2.6, le terme $\mathcal{G}$ seul, tout comme le terme $\mathcal{F}$, ne peut pas mener à une solution satisfaisante étant donné notre problème. En effet, les points de P qui ne sont pas considérés comme les points les plus proches d'un point dans S n'ont aucune influence sur l'énergie, et

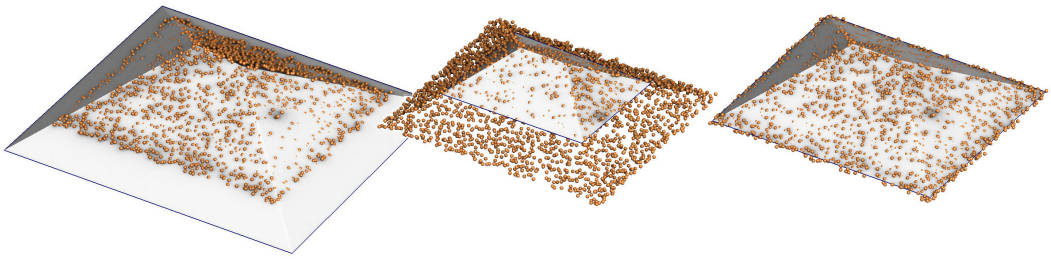


FIGURE 2.6 – Nous avons besoin des termes $\mathcal{F}(S)$ et $\mathcal{G}(S)$ dans l'énergie $E(S)$, sinon l'appariement est non optimal. **Gauche :** $\operatorname{argmin} \mathcal{F}$; **milieu :** $\operatorname{argmin} \mathcal{G}$; **droite :** $\operatorname{argmin} E$.

de fait, ils ne seront donc jamais pris en compte dans un processus de minimisation. En conséquence, le terme $\mathcal{G}$ ne capturera pas correctement les bords du toit à l'intérieur du nuage de points, et pourrait même atteindre un minima global, où $\mathcal{G} = 0$, en déplaçant le toit vers un seul point de P. Les deux termes jouent donc un rôle complémentaire dans la solution globale.

Le paramètre γ règle les poids relatifs des deux termes de l'énergie. Pour être indépendant de l'échelle globale ou de la densité des nuages de points, nous fixons $\gamma = 1$ pour notre nuage de points de résolution 30 pts/m^2, et conservons le même rapport pour les nuages de points plus ou moins denses. Cette valeur a été utilisée pour générer tous les résultats, à l'exception de la figure 2.10 et de la dernière ligne de la figure 2.14 où le réglage permet de capturer le bord du toit.

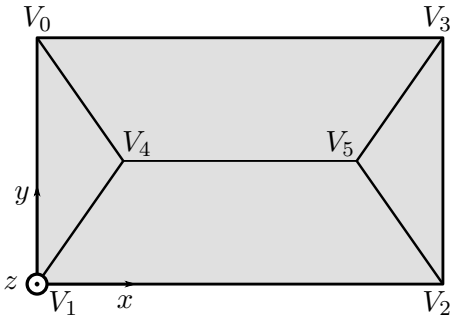
Il est intéressant de noter que bien que les deux termes de l'énergie soient définis comme des sommes de distances sur chaque point de P (pour $\mathcal{F}$) et de S (pour $\mathcal{G}$), leur expression diffère naturellement de la même manière que P et S sont deux objets différents, l'un étant un ensemble discret de points et l'autre une surface continue. De plus, remarquons que nous nous plaçons dans un contexte où les valeurs aberrantes ne peuvent apparaître que dans P alors que toute la surface de S doit être considérée comme une partie pertinente à ajuster sur les points. Cela implique logiquement que la robustesse des valeurs aberrantes ne s'exprime que dans le terme $\mathcal{F}$ et non dans $\mathcal{G}$.

En résumé, étant donné une surface mobile S et un nuage de points P fixe, nous avons défini l'énergie $E(S)$ dont la minimisation ajuste S à une partie de P. En pratique, S est un maillage triangulé dont la connectivité est fixée et dont la géométrie est définie par la position de ses sommets. Nous définissons la position du i-ème sommet S_i dans une base locale du modèle telle que :

$$S_i := RV_i + T$$

où $V_i = (V_{x_i}, V_{y_i}, V_{z_i})^T \in \mathbb{R}^3$, avec $R \in \mathbb{R}^{3\times3}$ une matrice de rotation et $T \in \mathbb{R}^3$ un vecteur de translation. R et T représentent une transformation rigide de cette base locale vers la base canonique de $\mathbb{R}^3$.

Notre objectif est de déformer le modèle pour maximiser sa vraisemblance au nuage de points en optimisant la position de ses coordonnées, notées $V \in \mathbb{R}^{3n}$,



$$\begin{cases} V_{x_0} = V_{x_1} \\ V_{x_3} = V_{x_2} \\ V_{y_1} = V_{y_2} \\ V_{y_4} = V_{y_5} \\ V_{y_0} = V_{y_3} \\ V_{x_0} < V_{x_4} < V_{x_5} < V_{x_3} \\ V_{y_1} < V_{y_4} < V_{y_0} \end{cases}$$

FIGURE 2.7 – Modèle de toit de la figure 2.2 où les contraintes ont été exprimées sous forme d'égalités et d'inégalités entre les coordonnées dans leur base locale. Ici, la toiture est contrainte afin de réduire le nombre de degrés de liberté c'est-à-dire le nombre de variables à l'optimisation, mais aussi afin de contraindre la toiture à présenter certaines propriétés géométriques. Pour autant, il n'est pas obligatoire de contraindre la surface dans notre algorithme.

c'est-à-dire la concaténation des V_i ainsi que pour la transformation rigide (R, T) du modèle. Nous réécrivons alors l'équation (2.1) en :

$$E(V, R, T) \;:=\; \mathcal{F}(V, R, T) \;+\; \gamma\, \mathcal{G}(V, R, T) \tag{2.5}$$

2.2.4 Injection de contraintes géométriques

Nous n'avons pas encore intégré les contraintes géométriques imposées à la surface, de sorte que notre problème d'optimisation se réécrit tel que :

$$\underset{V,R,T}{\operatorname{argmin}}\; E(V, R, T) \quad \text{soumis à des contraintes sur } V. \tag{2.6}$$

Dans cette partie, nous montrons d'abord comment ajouter ses contraintes géométriques (§ 2.2.4) via une réduction de variables, puis nous détaillons le calcul de l'énergie (2.5) ainsi que son gradient à l'aide de diagrammes de Voronoï restreints (§ 2.2.5), ce qui nous permet d'utiliser un solveur quasi newtonien.

Comme le montre la figure 2.2, nos modèles de toit doivent respecter diverses contraintes imposées sur leurs sommets (planarité des pans du toit, alignement des gouttières, horizontalité des faîtières, *etc.*). Toutes ces conditions peuvent être exprimées sous forme d'équations liant les coordonnées V de S. Dans notre optimisation, nous supportons des contraintes linéaires sur V qui prennent la forme :

$$\sum_{i,j} \alpha_{ij} V_{ij} = 0 \text{ ou } \sum_{i,j} \beta_{ij} V_{ij} \geqslant 0$$

.

Notez que certaines contraintes sont aisément linéarisables si nous considèrons les coordonnées des sommets de notre surface S, par exemple l'égalité des vecteurs des gouttières, alors que d'autres ne le sont pas, tel que l'orthogonalité ou la colinéarité. La séparation de la transformation rigide de base locale R, T de la déformation du maillage V est une pratique courante pour gérer ces cas non linéaires [Sorkine and Alexa, 2007, Sumner et al., 2007, Huang et al., 2011b]. Cela donne à toutes les

contraintes considérées une forme linéaire lorsqu'elles sont exprimées dans la base locale.

Nous montrons comment nous modélisons les contraintes présentées dans la figure 2.2. Tout d'abord, nous définissons la base locale de sorte que $\overrightarrow{V_1V_2}$ soit colinéaire à l'axe Ox, à savoir $V_{1y} = V_{2y}$ et $\overrightarrow{V_1V_0}$ soit colinéaire avec l'axe Oy avec $V_{1x} = V_{0x}$. Les contraintes peuvent alors être exprimées comme suit (voir figure 2.7) :

— $\overrightarrow{V_1V_2} = \overrightarrow{V_0V_3}$ donne $V_{1x} = V_{0x}$, $V_{2x} = V_{3x}$, $V_{1y} = V_{2y}$ et $V_{0y} = V_{3y}$;
— $\overrightarrow{V_1V_2} \cdot \overrightarrow{V_1V_0} = 0$ peut être développé en deux équations $V_{1x} = V_{0x}$ et $V_{1y} = y_{2y}$;
— La colinéarité $\overrightarrow{V_4V_5} = \lambda\overrightarrow{V_1V_2}$, $0 \leq \lambda \leq 1$ donne $V_{4y} = V_{5y}$ et $V_{0x} < V_{4x} < V_{5x} < V_{3x}$.

Égalités linéaires Les m contraintes linéaires exprimées sur les $3n$ variables de V peuvent être écrites dans une matrice C de taille $m \times 3n$ telle que $CV = 0$ (par exemple, dans la figure 2.7, $m = 5$). À partir de là, notre objectif est de calculer un ensemble de $3n - m$ variables réduites Y indépendantes les unes des autres, ainsi qu'une transformation entre Y et V telle que pour toutes affectations de Y, le V obtenu par cette transformation satisfasse $CV = 0$. À cette fin, nous cherchons une relation linéaire de la forme :

$$V = MY \tag{2.7}$$

où M est une matrice de taille $3n \times (3n - m)$ de rang maximal. Donc, $CMY = 0$ quelque soit $Y \in \mathbb{R}^{3n-m}$. Pour M, nous choisissons une base orthonormée du noyau de C. Cela la rend facilement calculable par une décomposition QR de C.

Inégalités linéaires En plus des égalités linéaires, nous voulons empêcher le modèle de toit d'avoir une position irréaliste, telles qu'empêcher les intersections des faces. Ces positions sont souvent le résultat de l'inversion de deux coordonnées. Compte tenu de l'initialisation et des paramètres fixés sur notre toit, une telle inversion n'est jamais nécessaire pour atteindre les minima globaux de l'énergie.

C'est ici que les contraintes d'inégalité de forme générale $\sum_{i,j} \beta_{ij} V_{ij} \geqslant 0$ prennent tout leur sens. Bien qu'elles ne soient pas strictement nécessaires, elles agissent comme une régularisation sur l'énergie en supprimant les minima locaux associés à des positions non réalistes de la surface. Comme nous ne sommes pas intéressés par les solutions où deux sommets coïncident, ces inégalités ne sont pas considérées comme strictes, mais plutôt comme une ligne directrice où $\sum_{i,j} \beta_{ij} V_{ij}$ devrait être suffisamment grand.

Étant donné que les expressions d'inégalité ne doivent pas être strictement respectées, nous choisissons de les traiter en ajoutant un terme supplémentaire à la fonction d'énergie.

Pour une contrainte de la forme $\sum_{i,j} \beta_{ij} V_{ij} \geqslant 0$, nous ajoutons à l'énergie E un terme barrière logarithmique exponentiel $\mathcal{I}$ tel que :

$$\mathcal{I}(V) = a \log(1 + \exp(-b \sum_{i,j} \beta_{ij} V_{ij})), \text{ where } a, b > 0 \tag{2.8}$$

La fonction $x \mapsto \log(1 + \exp(-x))$, nommée *softplus*, est une approximation dérivable de la fonction rectifieur linéaire $x \mapsto \max(-x, 0)$ souvent utilisée en régression et en apprentissage automatique (voir figure 2.8).

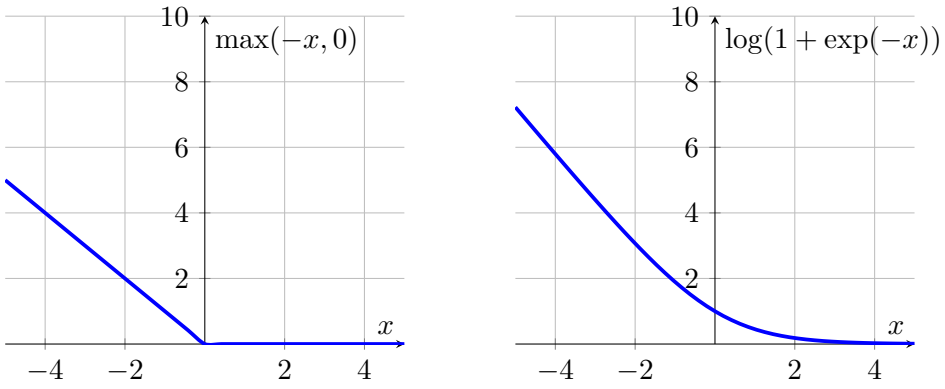


FIGURE 2.8 – Fonction $\max(-x, 0)$ à gauche et fonction softplus avec $a = 1$ et $b = 1$, à droite.

Le réglage des paramètres a et b nous permet de contrôler la pente et l'amplitude du terme de barrière. Avoir une grande valeur pour a permet de supprimer efficacement les minima locaux de E qui présentent des retournements de la surface ou des positions inacceptables. La valeur de b contrôle l'ampleur du gradient de $\mathcal{I}$. Si b est trop grand, les solutions avec des sommets très proches les uns des autres ne sont efficacement pas éliminées. Toutefois, si b est trop faible, la pente du gradient, lorsque les variables ne respectent pas la contrainte, pourrait ne pas être suffisante pour l'optimiseur.

En pratique, nous prenons $a = 10^5$ et $b = 3$. Ici, le paramètre b ne dépend pas de la taille du modèle, car les toitures sont rarement très petites. Cependant, si nous souhaitions fournir un algorithme plus généraliste, le paramètre b devrait dépendre de la taille du modèle.

En conclusion, en tenant compte des contraintes d'égalités et d'inégalités dans le système, notre problème de minimisation sous contraintes équation 2.6 peut être entièrement énoncé comme suit :

$$\underset{Y,R,T}{\operatorname{argmin}} \left\{ \mathcal{F}(MY, R, T) + \gamma\, \mathcal{G}(MY, R, T) + \sum_k \mathcal{I}_k(MY) \right\} \tag{2.9}$$

2.2.5 Minimisation de l'énergie

Les termes $\mathcal{F}$, $\mathcal{G}$ et $\mathcal{I}$ de l'équation 2.9 sont de classe C^2, ce qui est un résultat non trivial pour $\mathcal{G}$ [Liu et al., 2009]. Nous proposons de minimiser cette énergie en utilisant des méthodes quasi newtoniennes du second ordre comme l'algorithme L-BFGS [Liu and Nocedal, 1989]. Ceci nécessite l'estimation de la valeur de l'énergie E et de son gradient ∇E.

Dans toute cette section, nous adoptons la convention suivante pour les dérivées partielles : si $u = (u_x, u_y, u_z)$ est un point de $\mathbb{R}^3$ et f est une fonction à valeur réelle dépendante de u_i, nous noterons $\frac{\partial f}{\partial u}$ le vecteur $\left(\frac{\partial f}{\partial u_x}, \frac{\partial f}{\partial u_y}, \frac{\partial f}{\partial u_z}\right)$.

Nous commençons par calculer le gradient selon les coordonnées $S_i := RV_i + T$ des sommets de la surface S exprimé dans la base globale. Par abus de notations, nous désignerons par S le vecteur issu de la concaténation des S_i lorsque cela n'introduit

pas d'ambiguïté. Nous commençons par calculer $\nabla_S E$, et le gradient de E en fonction des variables réduites (Y, R, T) sera obtenu ultérieurement via la règle de la chaîne.

Estimation de $\mathcal{F}$ Le calcul du premier terme d'énergie $\mathcal{F}$ (équation 4.2) est simple : la distance au carré $||p - \pi_S(p)||^2$ peut être calculée avec une distance point-triangle en itérant sur chaque triangle de S, avant d'être introduite dans la fonction Ψ (équation 4.3).

Pour le gradient $\nabla_S \mathcal{F}$, étant donné un point $p \in P$, nous obtenons le vecteur :

$$\frac{\partial \mathcal{F}}{\partial \pi_S(p)} = 2(p - \pi_S(p))\psi\left(||p - \pi_S(p)||^2\right) \in \mathbb{R}^3$$

où ψ est la dérivée de Ψ. En notant $f(V_1, V_2, V_3)$ la face triangulaire de S contenant $\pi_S(p)$, et $(\lambda_1, \lambda_2, \lambda_3)$ les coordonnées barycentriques de $\pi_S(p)$ dans f, la contribution de p au gradient $\nabla_S \mathcal{F}$ ne concerne que les coordonnées des V_i avec :

$$\frac{\partial \mathcal{F}}{\partial V_i} = 2\lambda_i(p - \pi_S(p))\psi\left(||p - \pi_S(p)||^2\right), \tag{2.10}$$

et ces expressions doivent être additionnées pour chaque $p \in P$.

Estimation de $\mathcal{G}$ Pour le second terme $\mathcal{G}$ (équation 2.4), nous suivons [Nivoliers et al., 2014] et calculons le diagramme de Voronoï de P restreint à S. Désignons par Ω_p la cellule de Voronoï du site $p \in P$. Le diagramme partitionne S en régions polygonales $\Omega_p \cap S$ où le point le plus proche $\pi_S(p)$ est constant. L'intégrale peut donc être décomposée comme suit :

$$\mathcal{G} = \int_S ||q - \pi_P(q)||^2 dS(q) = \sum_{p \in P} \int_{\Omega_p \cap S} ||p - q||^2 dS(q) \tag{2.11}$$

Pour un $p \in P$ donné, nous triangulons la région $\Omega_p \cap S$ en triangles $t(t_1, t_2, t_3)$ et nous pouvons exprimer la valeur analytique de l'intégrale sur chaque triangle en utilisant le théorème 2.1 de [Lasserre and Avrachenkov, 2001] :

$$\mathcal{G}_t = \frac{aire(t)}{6} \sum_{1 \le i \le j \le 3} (t_i - p)(t_j - p) \quad \text{et} \quad \mathcal{G} = \sum_{p \in P} \sum_{\substack{t \in \\ \Omega_p \cap S}} \mathcal{G}_t$$

L'expression de $\mathcal{G}_t$ dépend des coordonnées des sommets t_k, qui ne sont pas des coordonnées de V en general, pour l'expression du gradient de $\mathcal{G}_t$ se référer à [Nivoliers et al., 2014].

Les valeurs et les gradients des termes de pénalisation $\mathcal{I}$ doivent être calculés en utilisant les coordonnées V dans la base locale. Leurs expressions sont simples.

Calcul des gradients de la transformation rigide (R,T) et des variables locales V. Les coordonnées V_i dans la base locale sont définies à partir des coordonnées S_i de la base globale par une transformation rigide :

$$S_i = R(\theta)V_i + T \in \mathbb{R}^3 \tag{2.12}$$

Par souci de clarté, nous nous limitons aux rotations autour de l'axe vertical Oz. Cela est suffisant pour traiter notre principal cas d'application et la généralisation à

une rotation arbitraire paramétrée par trois angles d'Euler est simple. Soit θ l'angle de rotation le long de Oz et $T = (T_x, T_y, T_z)$ le vecteur de translation. Nous obtenons :

$$\frac{\partial E}{\partial V_i} = R_z(-\theta)\frac{\partial E}{\partial S_i} \tag{2.13}$$

Ce qui donne le gradient $\nabla_V E$ étant donné le gradient $\nabla_S E$ calculé précédemment. En outre, nous nous intéressons aux dérivées $\frac{\partial E}{\partial \theta}$ et $\frac{\partial E}{\partial T}$ par rapport à la transformation rigide.

Gradient de $\frac{\partial E}{\partial \theta}$

En appliquant la règle de la chaine, nous obtenons :

$$\frac{\partial E}{\partial \theta} = \sum_{i=1}^{n} \frac{\partial E}{\partial S_i} \cdot \frac{\partial S_i}{\partial \theta} = \nabla_V E \cdot \frac{\partial S}{\partial \theta}$$

Selon l'équation 2.12, nous avons :

$$S_i = R_z(\theta)V_i + T$$

Seul la matrice de rotation R depend de θ dans le vecteur S. La dérivée de la matrice de rotation R peut être obtenu grâce à sa forme exponentielle. Soit $u \in^3$ un vecteur unitaire, nous avons :

$$L_u = \begin{pmatrix} 0 & -z & y \\ z & 0 & -x \\ -y & x & 0 \end{pmatrix}$$

Quel que soit $\theta \in$, la matrice $R_u(\theta) = \exp(\theta L_u)$ est la rotation d'angle θ autour de l'axe u. À partir de cette expression, nous pouvons déduire :

$$\frac{\partial R_u(\theta)}{\partial \theta} = \frac{\partial \exp(\theta L_u)}{\partial \theta} = L_u \exp(\theta L_u) = L_u R_u(\theta)$$

Dans notre cas, nous considérons uniquement les rotations autour de l'axe Oz, car il n'est pas nécessaire pour nos applications de prendre en compte les autres axes de rotation, mais ce résultat est vrai pour tout vecteur u.

Soit L_z et $R_z(\theta)$ les matrices antisymétriques et de rotation de l'angle θ associé à l'axe $Oz = (0, 0, 1)$. Notez que $L_z = R_z(\frac{\pi}{2})$. En combinant ces propriétés avec l'équation 2.12, nous avons :

$$\frac{\partial S_i}{\partial \theta} = L_z R_z(\theta) V_i = L_z(S_i - T) = R_z(\frac{\pi}{2})(S_i - T)$$

Nous obtenons enfin :

$$\frac{\partial E}{\partial \theta} = \sum_i \frac{\partial E}{\partial S_i} \cdot R_z(\frac{\pi}{2})(S_i - T) \tag{2.14}$$

Cela nous permet de conclure pour l'axe Oz. Afin d'ajouter les axes Oy et Ox, il suffit d'appliquer les rotations dans un ordre fixe et d'insérer L_x et L_y à la bonne position dans l'équation 2.14.

Gradient de $\frac{\partial E}{\partial T}$

Soit μ un indice x, y ou z. L'expression de la règle de la chaine pour la translation T_μ s'écrit :

$$\frac{\partial E}{\partial T_\mu} = \sum_{i=1}^{n} \frac{\partial E}{\partial S_i} \cdot \frac{\partial S_i}{\partial T_\mu}$$

et selon l'équation 2.12 :

$$\frac{\partial S_i}{\partial T_x} = (1,0,0),\ \frac{\partial S_i}{\partial T_y} = (0,1,0),\ \frac{\partial S_i}{\partial T_z} = (0,0,1)$$

Nous obtenons :

$$\frac{\partial E}{\partial T} = \sum_{i} \frac{\partial E}{\partial S_i} \tag{2.15}$$

Calcul du gradient en fonction des variables réduites Y Avec les expressions ci-dessus, nous avons tous les ingrédients pour minimiser l'énergie en fonction des variables (V, θ, T), ce qui n'est qu'une moitié du problème puisqu'il ne prend pas en compte les contraintes d'égalité linéaire (2.7). Bien que le calcul de $\nabla_S E$ et de $\nabla_V E$ soit une étape nécessaire, nous sommes réellement intéressés par les dérivées partielles des $3n - m$ variables réelles de Y (en supposant que m est le nombre de contraintes d'égalité distinctes). Étant donnée l'équation 2.7 $\nabla_Y E$ peut être facilement calculée en utilisant la règle de la chaîne comme suit :

$$\nabla_Y E = M^\top \nabla_V E \tag{2.16}$$

qui se conclut sur le calcul de E et de ses gradients en fonction des variables pertinentes.

2.3 Résultats

Évaluer et comparer notre méthode est complexe pour deux raisons. Tout d'abord, l'ajustement d'un modèle de toit sur un ensemble de nuages de points est un problème mal posé puisque les informations de base ont été perdues pendant l'acquisition.

Deuxièmement, aucune métrique quantitative et globale n'a été établie pour évaluer la qualité d'un ajustement, et donc comparer objectivement les méthodes existantes.

Notre énergie E (définie par l'équation 2.6) est une tentative de définition d'une telle métrique. Afin de tester la qualité de notre métrique, nous définissons comme position de référence soit un modèle ajusté manuellement, soit le résultat de l'ajustement sur des données qui ont été nettoyées manuellement (comme dans le tableau 2.1 et le tableau 2.2). Nous évaluons ensuite la qualité de la position finale d'un ajustement en calculant la distance de Hausdorff entre ses sommets et les sommets de cette référence, que l'on appelle d. Bien que cette distance ne prenne en compte que les sommets du gabarit du toit et non la surface entière, les zones d'intérêt seront souvent les bords du toit, ce que cette métrique capture correctement. Dans toute cette section, la distance et les scores numériques donnés dans le texte et les figures correspondront à cette distance et seront donnés en mètres (m).

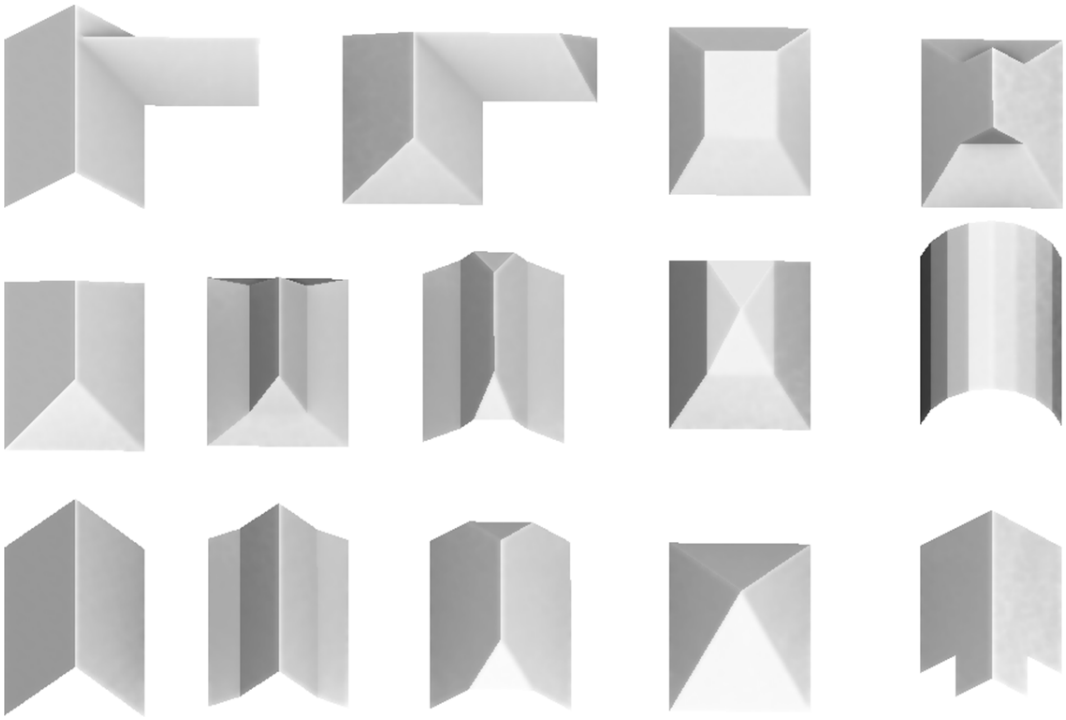


FIGURE 2.9 – Notre méthode utilise une bibliothèque de divers modèles de toits, ce qui nous permet de couvrir la majorité des cas du monde réel.

Nos nuages de points ont été sélectionnés à partir d'acquisitions LiDAR des villes de Breuschwickersheim et Strasbourg qui sont disponibles en ligne [Strasbourg, 2016]. Afin d'isoler l'effet de divers défauts sur les nuages de points, nous avons extrait à la main certains nuages de points de référence et éliminé manuellement différents ensemble de valeurs abbérantes afin de générer des distributions admettant la même solution optimale pour le placement du toit. En outre, des cas de test ont été sélectionnés pour des toits mitoyens dans le but de tester différents types de jonctions pour les toits composés (voir figure 2.10).

Nous nous basons sur une bibliothèque prédéfinie de modèles de toits (voir figure 2.9) disponibles sous forme de maillages triangulés sur lesquels nous définissons manuellement des contraintes géométriques. Ces modèles couvrent la majorité des toits détachés du monde réel, à l'exception des toits plats, qui peuvent être traités avec un modèle rectangulaire plat. Pour les formes de toit plus complexes, par exemple dans les zones urbaines denses, notre méthode considère le toit connecté comme un groupe de ces modèles et vise à les adapter séparément.

Dans cette section, nous évaluons la robustesse et la qualité de notre méthode sur deux classes de paramètres. Nous nous concentrons d'abord sur la qualité de l'ajustement en fonction du bruit présent dans le nuage de points et sa qualité. Ensuite, nous réalisons des expériences sur la position initiale des modèles, et la capacité de notre méthode à converger vers une bonne solution lorsque cette initialisation est éloignée de l'optimal. Enfin, nous discuterons des performances, des avantages et des cas d'échec de la méthode.

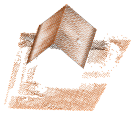
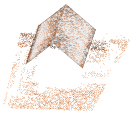
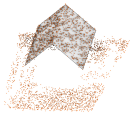
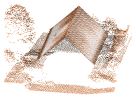
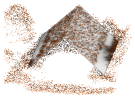
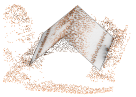
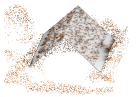
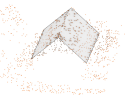
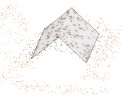
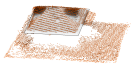
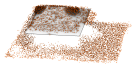
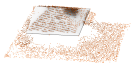
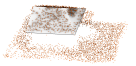
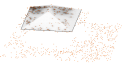
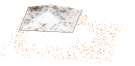
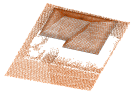
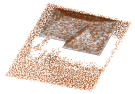
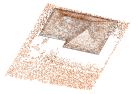
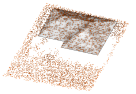
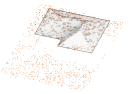
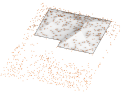
	100%	Bruité 100%	50%	Bruité 50%	10%	Bruité 10%
Toiture à 2 pans						
	Référence	$d = 0.27m$	$d = 6.80 \times 10^{-2}m$	$d = 0.20m$	$d = 0.53m$	$d = 0.44m$
Toiture à 2 pans et végétation						
	Référence	$d = 2.94m$	$d = 1.93m$	$d = 2.13m$	$d = 0.53m$	$d = 0.66m$
Toiture à 4 pans						
	Référence	$d = 0.44m$	$d = 2.84m$	$d = 4.24m$	$d = 2.40m$	$d = 2.24m$
Toiture en L et sol						
	Référence	$d = 0.43m$	$d = 0.34m$	$d = 0.46m$	$d = 0.95m$	$d = 0.87m$

TABLE 2.1 – Expérience montrant la robustesse de notre méthode suivant la qualité des nuages de points. *Bruité* indique la présence d'une distribution uniforme de bruit d'une amplitude de 30 cm ajoutée à la position de chaque point. Les pourcentages indiquent la proportion de points restant dans le nuage. Les nuages de points sont représentés en orange et nous représentons la position finale du modèle de toit sur le nuage de points. La position initiale du toit est définie en alignant le barycentre du nuage avec le barycentre du modèle, et en translatant le modèle de 3m vers le haut. La rotation était la même pour tous les cas de test, mais arbitraires. L'erreur d est estimée quantitativement comme la distance de Hausdorff entre le modèle ajusté et un modèle de référence qui est soit généré par l'homme, soit le résultat d'un ajustement sur le nuage de points nettoyé.

2.3.1 Impact de l'initialisation

Nous commençons par tester le comportement de convergence de la méthode en fonction de la qualité du nuage de points. Rappelons que nous avons conçu la fonction d'énergie pour qu'elle soit robuste aux valeurs aberrantes (voir sous-section 2.2.2), c'est-à-dire aux grandes portions du nuage de points qui ne correspondent pas à la forme spécifique du toit que nous voulons ajuster. Cela doit être vrai même si les valeurs aberrantes sont cohérentes, par exemple dans le cas d'un arbre ou d'un toit voisin. En outre, nous testons notre méthode sur des données altérées, afin de vérifier qu'elle est relativement indépendante de la densité du nuage de points, ainsi que robuste au bruit d'acquisition. Les tests sont donc répartis en deux catégories :

— Présence d'autres structures (antennes, volets, cheminées, sol, arbres, *etc.*) ;
— Présence d'autres toitures.

Acquisition bruitée et densité du nuage de points Le premier test que nous effectuons peut être considéré comme un contrôle de bon sens où nous évaluons que le résultat obtenu par notre méthode ne change pas, ou peu, lorsque la densité du nuage de points est modifiée et lorsque du bruit est ajouté au nuage de points. À cette fin, nous comparons dans le tableau 2.1 les résultats d'ajustement du même modèle avec la même initialisation sur un nuage de points qui a été artificiellement modifié. Trois configurations de décimation du nuage de points sont considérées : nuage de points complet, décimation aléatoire de 50% et décimation aléatoire de 90% des points. Le bruit artificiel ajouté suit une distribution uniforme avec une moyenne nulle et une amplitude maximale de 30 cm.

La position initiale du modèle a été obtenue en faisant correspondre les barycentres du nuage de points et du modèle, puis en appliquant une rotation fixe (mais arbitraire) de 0,2 radian et une translation vers le haut de 3 mètres. Cela garantit que la convergence ne bénéficie pas d'une position initiale trop proche de la position optimale. Commencer par le haut du nuage de points est également une pratique courante dans nos tests, puisque la plupart des points aberrants se trouvent sous ou à côté de la zone d'intérêt.

Les expériences présentées dans le tableau 2.1 ont été réalisées sur trois modèles de toits différents (un toit à deux pans, un toit à 4 pans et un toit en L) et trois nuages de points correspondants. Dans le cas du toit à deux pans, l'expérience a été menée sur un nuage présentant des arbres et sur le même nuage où les arbres ont été supprimés manuellement. Nous observons que dans le premier cas, les points bruyants des arbres sont capturés par le modèle, ce qui conduit à une grande imprécision sur les bords du toit. Dans un contexte où il y a moins de points aberrants près du toit (toit à deux pans sans arbres et toit en L), nous observons que l'écart varie de quelques centimètres à quelques dizaines de centimètres même avec 90% du nuage de points décimé, ce qui est acceptable étant donné l'étendue de la perturbation. Sur le toit à 4 pans, les scores de distance de Hausdorff sont élevés, car la perturbation du nuage de points conduit à placer la ligne de faite à la perpendiculaire de celle d'origine. Cela conduit à des résultats visuellement proches, mais nous considérons ces cas comme des échecs.

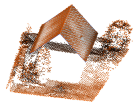
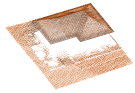
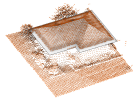
	Segmenté	Petites structures	Larges structures	Sol	Sol et végétation
Toiture à 2 pans					
	Référence	$d = 2.73 \times 10^{-5} m$	$d = 2.96 \times 10^{-3} m$	$d = 2.06 \times 10^{-5} m$	$d = 2.91 \times 10^{-5} m$
Toiture à 4 pans					
	Référence	$d = 8.37 \times 10^{-2} m$	$d = 9.30 \times 10^{-1} m$	$d = 1.93m$	$d = 2.60m$
Toiture en L					
	Référence	$d = 2.31 \times 10^{-2} m$	$d = 9.42 \times 10^{-2} m$	$d = 1.48 \times 10^{-2} m$	$d = 1.01m$

TABLE 2.2 – Expérience montrant la robustesse de notre méthode à diverses distributions de valeurs aberrantes présentes dans des jeux de données réels. Les nuages de points sont représentés en orange et nous représentons la position finale du modèle de toit sur le nuage de points. La position initiale du toit est définie en alignant le barycentre du nuage avec le barycentre du modèle, et en translatant le modèle de 3m vers le haut. La rotation était la même pour tous les cas de test, mais arbitraires. L'erreur d est définie comme dans le tableau 2.1.

Robustesse aux valeurs aberrantes contextuelle Dans la plupart des nuages de points LiDAR aéroportés du monde réel, l'acquisition est effectuée sur chaque objet présent dans le paysage. Bien que la technologie LiDAR moderne permette une forme de segmentation, la présence d'objets non pertinents dans le nuage est fréquente. Lors de l'ajustement d'un modèle de toit, les structures sur le toit comme les antennes ou les fenêtres ainsi que les structures adjacentes comme les arbres et les voitures doivent être ignorées.

Étant donné des nuages de points segmentés manuellement en différents "niveaux" de valeurs aberrantes, nous effectuons une optimisation avec notre méthode sur ces différentes versions. Nous comparons ainsi les performances avec et sans certaines classes de structures. Nous considérons quatre cas en plus de la référence parfaitement segmentée : les petites structures supplémentaires sur le toit (cheminées, antennes, *etc.*), les grandes structures supplémentaires, le sol et enfin le sol et les arbres.

Les résultats sont présentés dans le tableau 2.2. L'initialisation a été similaire pour tous les cas de test et définie comme dans l'expérience précédente : initialisation à 3m de hauteur du barycentre du nuage de points et une rotation de 0.2 radian. Pour chaque cas, nous mesurons la distance de Hausdorff par rapport au cas parfaitement segmenté. Nous observons que dans la grande majorité des cas, l'influence des valeurs aberrantes reste faible, avec une déviation observée dans notre métrique de l'ordre du centimètre. Sur le toit en croupe, nous observons que la présence d'une structure adjacente (comme un porche ou une véranda) entraîne une imprécision sur le bord. Les cas d'échec les plus remarquables se produisent sur les toits à 4 pans et les toits en croupe en L avec présence d'arbres et de sol, où un arbre touchant directement le bord du toit fait que l'algorithme considère ces points comme valides, ce qui conduit également à un grand décalage.

Groupes de toits et contiguïtés Dans les zones denses où les bâtiments sont adjacents, nous pouvons utiliser notre méthode pour ajuster un seul toit à la fois. Les autres toits présents dans le nuage de points doivent être considérés comme des valeurs aberrantes, et la seule donnée indiquant quel toit dans le nuage de points doit être ajusté est l'initialisation du modèle. Dans la figure 2.10, nous présentons le résultat obtenu pour un toit à deux pans et un toit en croupe croisée. Bien que des résultats corrects puissent être obtenus en réglant le paramètre γ et en commençant près de la position souhaitée, nous concluons que ces configurations restent particulièrement difficiles à optimiser avec notre méthode, car elles peuvent conduire à des solutions acceptables qui ne correspondent pas à la réalité (voir figure 2.14). En revanche, elles peuvent grandement bénéficier d'étapes de prétraitement telles que la segmentation des plans.

2.3.2 Robustesse et cas d'échec

Jusqu'à présent, les expériences ont été réalisées en utilisant une initialisation grossière au nuage de points donné. Or, la minimisation de notre fonction d'énergie est une tâche qui dépend fortement de la position de départ. Après avoir évalué notre méthode sur les variations du nuage de points, nous présentons ici une série de tests afin de valider que notre méthode permet de trouver la position optimale du toit pour des initialisations variées.

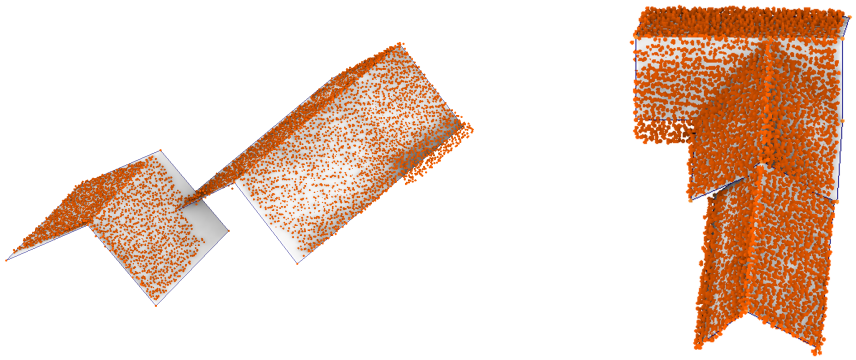
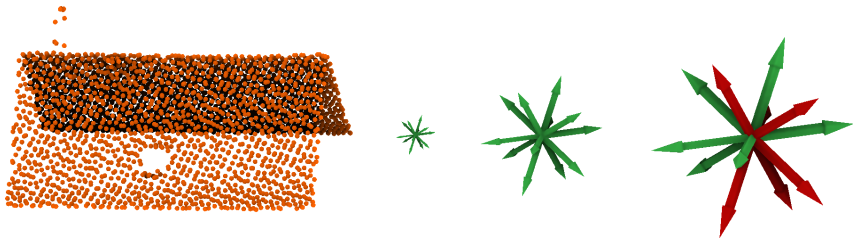


FIGURE 2.10 – Résultats d'ajustement obtenus sur des toits adjacents. À gauche : deux toits à deux pans. À droite : un toit en forme de T et un toit à deux pans simple. Lors de l'ajustement sur un toit spécifique, l'autre toit est considéré comme part des valeurs aberrantes.

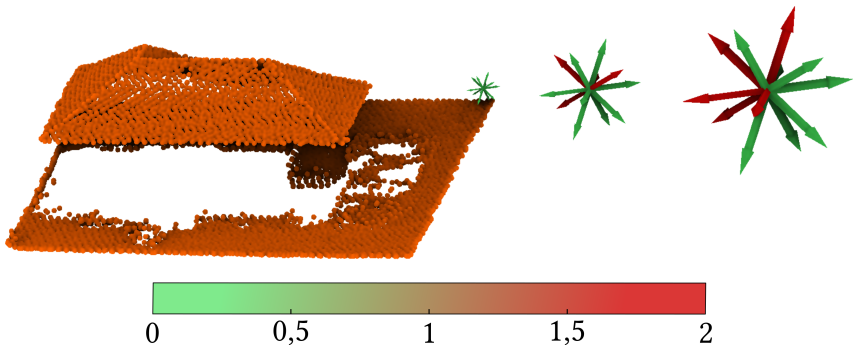
Rappelons que dans l'équation 2.9, nous optimisons notre fonction d'énergie selon trois ensembles de variables : la translation T, la rotation R et les variables réduites Y obtenues après avoir contraint les sommets à des contraintes linéaires. Nous présentons ici un protocole permettant d'évaluer la qualité de l'optimisation en fonction des valeurs initiales de ces trois ensembles de variables de manière indépendante.

Translation Nous initialisons le modèle de toit en appliquant une translation à la position de référence. Nous testons douze directions différentes uniformément réparties sur une sphère pour trois magnitudes différentes de translation à 1, 3, et 5 mètres de la position optimale (voir figure 2.11). À côté du nuage de points utilisé, nous représentons la translation sous forme de douze vecteurs de longueur correspondante. Les couleurs correspondent à la distance de Hausdorff calculée, 0m étant entièrement vert et 2m entièrement rouge. Nous observons qu'en pratique, les translations de faible amplitude ne posent aucun problème d'optimisation, alors que les translations de plus grande amplitude ont tendance à échouer. Lorsque le modèle est éloigné du nuage, il ne considère qu'une partie du nuage, car la plupart des points sont considérés comme aberrants et peuvent donc converger vers un ajustement partiel (voir figure 2.14).

Rotation Pour la composante rotation, nous initialisons le modèle avec dix angles différents autour de l'axe Z, centré sur le centre de la position optimale du toit. La disposition des résultats est représentée sur la figure 2.12, où les positions initiales forment l'anneau intérieur des modèles, et les positions finales correspondent à l'anneau extérieur. Nous observons que pour le toit à 2 pans, l'optimisation est capable de retrouver l'orientation correcte, quelle que soit l'orientation, alors que sur les toits non symétriques comme le toit en L, la meilleure position n'est obtenue que pour un angle de $2i\pi/n$ par rapport à l'optimal, avec $i \in [0, n-1]$ et $n = 10$. Un angle supérieur conduit généralement à un minimum local où le toit est inversé (voir figure 2.14).



Toiture à deux pans et petites structures.



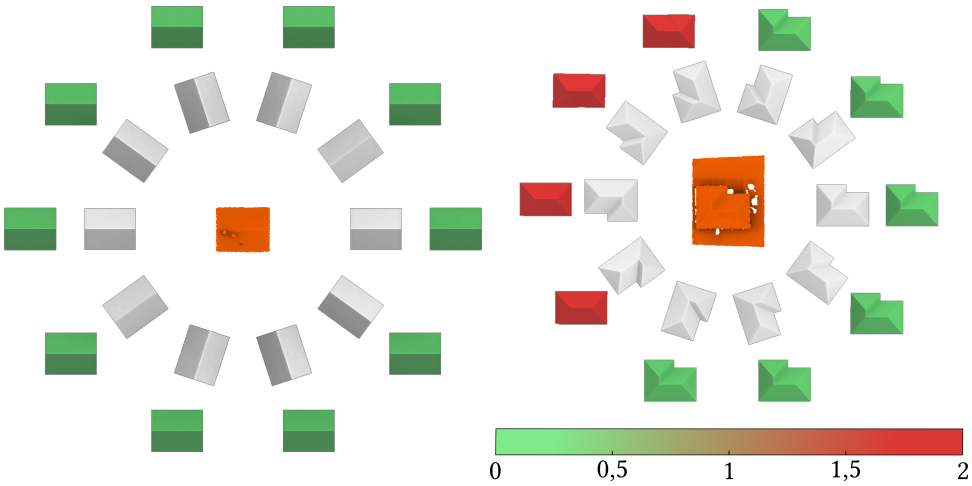
Toiture en L et sol.

FIGURE 2.11 – Expériences de translation sur un toit deux pans avec de petites structures et un toit en L avec du sol. La couleur des flèches représente le succès de notre méthode lorsqu'elle est initialisée à la position de référence, traduite par le vecteur correspondant.

Étirement Enfin, nous effectuons des tests en étirant le modèle de référence. L'étirement est appliqué par un facteur de [0.5, 1., 1.5] dans les directions x, y et z, conduisant à 27 cas différents. Sur la figure 2.13, nous présentons les résultats pour un toit en L avec sol. Nous observons que les cas d'échec se produisent pour les modèles rétrécis, qui ont tendance à converger vers une partie seulement du nuage de points.

2.3.3 Performances

Temps de fonctionnement Notre algorithme est implémenté en C++, et utilise le diagramme de Voronoï restreint ainsi que le solveur L-BFGS présents dans la bibliothèque Geogram [Inria, 2018]. Les expériences ont été réalisées sur un processeur Intel(R) Xeon(R) E-2276M (cadencé à 2,8 GHz). Le temps de convergence peut prendre jusqu'à 20 secondes pour les initialisations qui sont loin de la position optimale puisque le solveur a besoin de plus d'étapes pour atteindre la convergence, et le diagramme de Voronoï restreint prend également plus de temps à être calculé lorsque le modèle couvre également les cellules des aberrations. Lorsqu'une bonne initialisation est disponible, le temps descend à $2-3$ secondes par ajustement. La partie la plus longue de notre algorithme est le calcul du diagramme de Voronoï restreint (80% du temps d'exécution) nécessaire pour évaluer $\mathcal{G}$ et $\nabla_S \mathcal{G}$. Un nuage de points typique dans nos données contient 1000 à 10 000 points selon la distribution considérée des points aberrants.



(a) Toiture à deux pans et petites structures.

(b) Toiture en L et sol.

FIGURE 2.12 – Expériences de rotation. Le nuage de points est représenté en orange au milieu. Le premier toit à droite du nuage de points est la vérité terrain. Les autres toits dans l'anneau intérieur (gris) sont obtenus par rotation de la vérité terrain. Le deuxième anneau de toits montre notre résultat pour chaque initialisation. Quelle que soit la position initiale de rotation du modèle, le schéma de minimisation converge vers la position correcte. Anneau de gauche : toit à deux pans avec rotation. Anneau de droite : toit en L.

Cas d'erreurs fréquents Nous présentons des cas de réussite et d'échec dans l'ensemble de cette section. D'après nos expériences, nous avons observé que les échecs se produisent lorsque :

1. Le modèle de toit correspond à une partie ou à l'intégralité de la distribution des points aberrants. Cela se produit notamment lorsque l'initialisation est trop proche de la distribution (par exemple, pour un modèle initialisé dans le sol), ou si la distance minimale entre une aberration et un point pertinent est faible et que la fonction biweight (voir équation 4.3) n'est plus capable de distinguer l'une de l'autre ;
2. Le modèle de toit est initialisé dans une mauvaise orientation qui bloque l'optimisation dans un minimum local ;
3. Le nuage de points présente un toit adjacent. Bien que cela donne une énergie très faible et de bons résultats visuels, cela n'est pas souhaité dans la plupart des cas d'utilisation.

Ces trois cas sont présentés dans l'ordre dans la figure 2.14 et la figure 2.15. Heureusement, ils peuvent être facilement évités pour la plupart :

1. Considérer les initialisations qui sont au-dessus du nuage de points (ou une initialisation plus fine qui pourrait être disponible dans un pipeline de traitement) ;
2. Essai de la meilleure des deux initialisations où nous avons considéré deux angles initiaux α et $\alpha + \pi$;

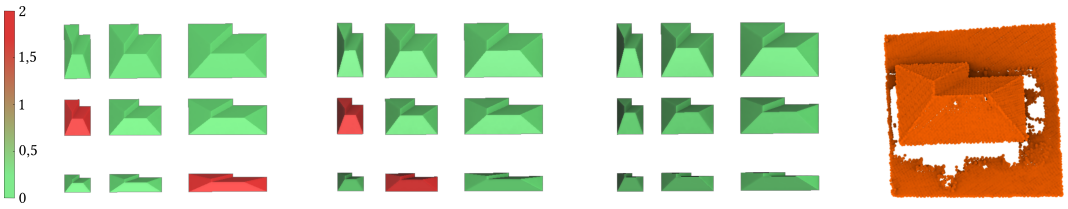


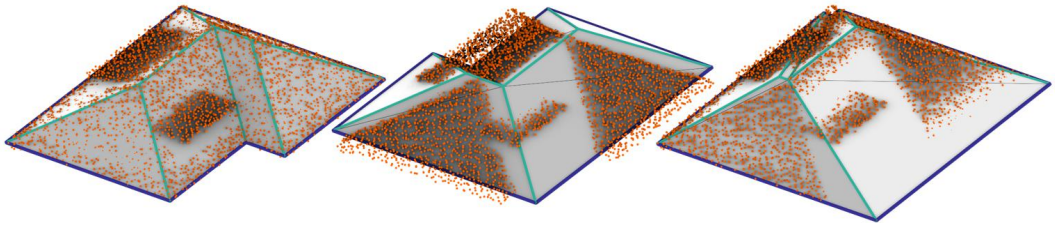
FIGURE 2.13 – Expériences d'étirement sur un toit en L avec sol. De gauche à droite, chaque ensemble de 9 toits a un facteur d'étirement z croissant de [0.5, 1., 1.5]. Le facteur d'étirement x augmente à l'intérieur de chaque ligne, et l'étirement y augmente dans chaque rangée. Par conséquent, le résultat central n'est pas étiré du tout et représente notre référence. Les gabarits montrés sont le toit **initialisation** et non sa position finale. La carte des couleurs va de 0m (vert) à 2m (rouge). Le nuage de points utilisé est représenté sur la droite.

3. Réglage du paramètre γ et donc diminution de la sensibilité aux points voisins.

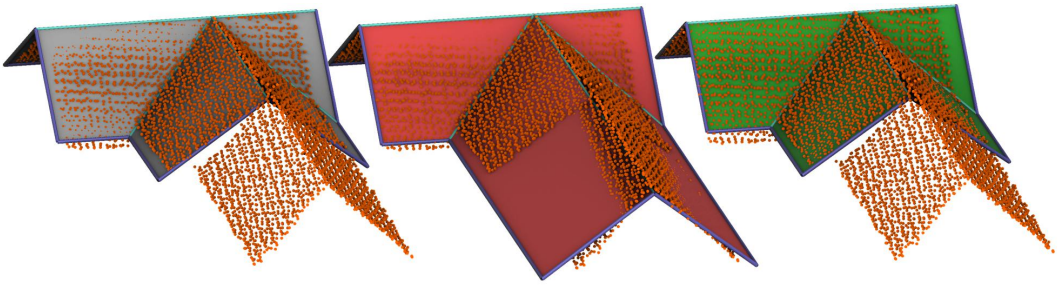
Comme prévu, les valeurs aberrantes proches de la surface sont difficiles à classer en utilisant uniquement une fonction de la distance au toit. Dans nos travaux futurs, nous prévoyons un meilleur comportement avec des fonctions d'énergie qui seraient anisotropes dans la direction normale du plan. Une autre direction future est d'ajuster plus d'un toit à la fois, afin de segmenter automatiquement un ensemble de toits adjacents.

Initialisé au toit blanc solide, notre algorithme trouve un minimum local (toit transparent) qui inclut les points au sol.

FIGURE 2.14 – Cas d'échec typique de notre algorithme.



Le toit en croix (gauche), initialisé par une rotation de π (milieu), a trouvé un minimum local qui ne correspond pas aux faces.



Le bord de la marche (à gauche) n'est pas détecté par notre algorithme (au milieu), mais la division de γ par 10 résout le problème (à droite).

FIGURE 2.15 – Cas d'échec typique de notre algorithme.

Une meilleure initialisation Les expériences détaillées dans la section 4.3.1 montrent que le bassin d'attraction du meilleur ajustement est assez large en général, même en présence de bruit et de valeurs aberrantes. Cependant, il est important de noter qu'un algorithme de reconstruction entièrement automatique nécessiterait une estimation initiale juste pour commencer, qui pourrait par exemple être calculée à partir d'empreintes ou de données cadastrales, d'un ajustement de plan, *etc.*, ou donné par un opérateur humain dans un contexte d'interaction homme-machine.

2.4 Conclusion

Dans ce travail, nous montrons que l'ajustement d'un toit doté de contraintes géométriques sur des nuages de points acquis par LiDAR peut être réalisé efficacement par l'optimisation d'une fonction d'énergie, en utilisant des optimiseurs quasi newtoniens comme L-BFGS et le calcul rapide d'un diagramme de Voronoï restreint. L'ajustement partiel du nuage de points, c'est-à-dire l'ajustement en présence de valeurs aberrantes, peut également être effectué. Notre méthode donne des résultats satisfaisants en tant que procédure d'ajustement autonome sur un panel varié de cas réels, en particulier pour les toits individuels. Dans les zones plus denses, l'intégration de cette méthode dans un pipeline avec des méthodes de pointe pour trouver une initialisation équitable et/ou appliquer un prétraitement de segmentation est loin d'être inutile.

Deuxième partie

Accélération et transfert industriel

L'objectif de cette seconde partie est de présenter un ensemble de méthodes permettant l'accélération des calculs et le transfert industriel des différentes méthodes de cette thèse.

Nous abordons dans le chapitre 3 de cette partie une méthode de calcul de diagramme de Voronoï et diagramme de puissance restreint sur le GPU permettant de calculer efficacement ces diagrammes pour des applications de simulation numérique.

Puis, dans le chapitre 4, nous présentons une optimisation de l'algorithme d'ajustement de surface à un nuage de points présenté dans le chapitre 2, permettant de calculer plus rapidement nos ajustements de surface.

Enfin dans le chapitre 5 nous montrons l'ensemble des développements réalisés permettant à un utilisateur d'exploiter une partie des méthodes proposé dans cette thèse.

Chapitre 3

Diagramme de puissance restreint à un volume optimisé sur le GPU

Sommaire

3.1 Motivations

Dans le chapitre précédent, une énergie a été définie puis minimisée afin d'ajuster une surface munie de contraintes géométriques à un nuage de points. Néanmoins, même si cet algorithme offre les résultats espérés, il nécessite un temps d'exécution élevé pour chaque ajustement. Celui-ci peut varier de 1 à 30 secondes où l'utilisateur doit attendre passivement la fin de la commande. Ce temps d'exécution est déraisonnable pour l'intégrer dans un logiciel pour que la méthode soit qualifiée d'interactive [Card et al., 1991]. Pour des raisons d'ergonomie logicielle, nous souhaitons réduire le temps d'exécution de notre algorithme. Une rapide analyse des performances montre que les multiples calculs de diagrammes de Voronoï restreints ont un impact considérable sur les performances de l'algorithme. En effet, plus de 80% du temps d'exécution est consacré à l'estimation de diagramme de Voronoï restreint. Il faut donc trouver une méthode permettant de diminuer drastiquement le temps de génération d'un diagramme de Voronoï.

Cette problématique est en lien avec des problématiques de l'équipe Pixel portant sur la simulation numérique, où il est également nécessaire de calculer des intégrales, mais cette fois-ci pour un volume donné. En effet, lorsque les simulations numériques font intervenir des objets géométriques complexes, il est nécessaire de les discrétiser. Les diagrammes de Voronoï étant contrôlés par un ensemble de points sont intéressants à cet effet en tant qu'outils de discrétisation. Ils peuvent être exploités, par exemple, pour simuler des processus de diffusion (Centroidal Voronoi Tessellations/Lloyd) [Wang, 2017, Lu et al., 2012], ou pour modéliser des mousses aléatoires, mais bien distribuées [Martínez et al., 2016].

Dans le cas de la simulation numérique, il est uniquement nécessaire d'estimer l'intégrale de la fonction d'énergie, ouvrant une porte pour l'optimisation, grâce à des algorithmes parallèles ne stockant pas de combinatoire [Ray et al., 2018]. Cependant, cet algorithme ne supporte pas la restriction à un domaine prédéfini, tel qu'une surface représentant une toiture. Nous souhaitons étendre l'algorithme de Ray et al. [Ray et al., 2018] pour répondre à ces besoins.

L'équipe Pixel s'intéresse également à une généralisation du diagramme de Voronoï, nommée diagramme de puissance, ou diagramme de Laguerre–Voronoi. Par rapport à un diagramme de Voronoï classique, un poids est ajouté à chaque racine afin de contrôler la taille relative de chaque cellule [7]. Un avantage du diagramme de puissance est son lien avec le transport optimal : étant donné un ensemble de racines et un domaine Ω, il existe un ensemble de poids, tel que la carte qui associe chaque racine à sa cellule de puissance restreinte à Ω est le transport optimal entre les racines et Ω. Cette propriété a été exploitée pour générer du bruit bleu [de Goes et al., 2012, Xin et al., 2016] et plus récemment pour renforcer l'incompressibilité en dynamique des fluides [de Goes et al., 2015a, Gallouët and Mérigot, 2017]. Nous choisissons donc de travailler sur une problématique plus large permettant le calcul de diagramme de puissance, pour répondre aux différentes problématiques de l'équipe.

La plupart d'entre nous disposent d'une ressource dotée d'une grande puissance, mais qui reste le plus souvent inexploitée : notre carte graphique. Dans cette partie, nous décrivons comment calculer efficacement des diagrammes de puissance restreints

7. Un diagramme de Voronoï est donc un diagramme de puissance dont tous les poids ont été fixés à un même poids.

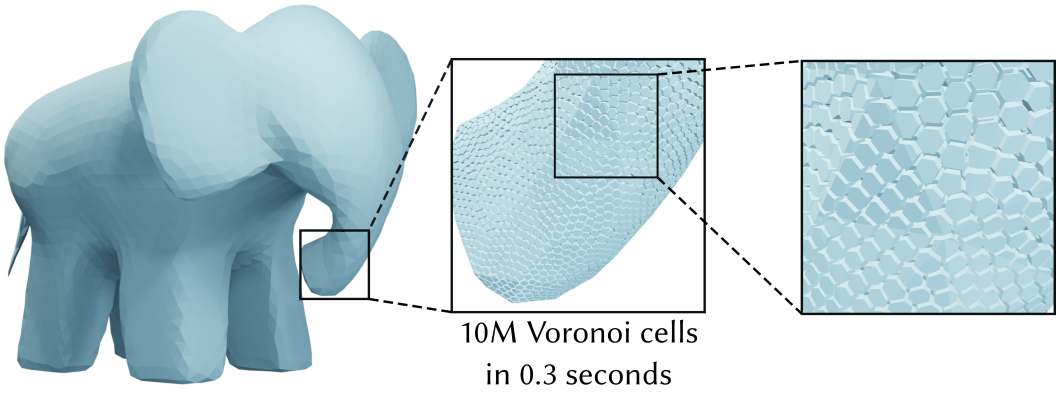


FIGURE 3.1 – Nous proposons une méthode de calcul de diagramme de Voronoï sur le GPU qui nous permet de calculer 10 millions de points restreints à une triangulation en 300 ms (avec une carte graphique Nvidia V100).

à un domaine, grâce au calcul sur processeur graphique afin de bénéficier de sa capacité de traitement parallèle. Pour l'ajustement de surface à nuage de points et les applications de simulation, les diagrammes de Voronoï/puissance sont construits uniquement pour calculer les intégrales. Par exemple, la *Centroidal Voronoi tessellation* (CVT), qui est un diagramme de Voronoï spécifique où les racines sont également le centre de masse de leur cellule, nécessite d'estimer le volume et le barycentre des cellules de Voronoï ; quant au transport optimal, il doit en supplément supporter les diagrammes de puissance et nécessite d'estimer la matrice de l'opérateur laplacien pondéré [de Goes et al., 2012]. Le calcul de ces valeurs est souvent un goulot d'étranglement, car elles sont nécessaires dans une boucle critique du processus d'optimisation.

Ce chapitre présente une méthode performante pour estimer des diagrammes de puissance sur un domaine restreint par une surface triangulée grâce à l'utilisation du GPU (voir figure 3.1). Dans un premier temps, nous présentons différentes méthodes qui permettent de calculer des diagrammes de Voronoï et des diagrammes de puissance en parallèle §3.2. Puis, nous détaillons notre méthode en prenant soin de rappeler les travaux précédents [Ray et al., 2018] sur lesquelles notre algorithme s'appuie §3.3. Enfin, nous évaluons les performances de notre méthode à travers différentes applications et discutons des résultats obtenus §3.4.

3.2 Fondements et état de l'art

Dans cette section, nous présentons l'état de l'art de la littérature qui permet le calcul de diagramme de Voronoï. Nous présentons ensuite les notations qui seront utilisées dans la suite de ce chapitre.

3.2.1 État de l'art : calcul de diagramme de Voronoï

Le calcul de la triangulation de Delaunay (c'est-à-dire le graphe dual du diagramme de Voronoï) est une tâche fondamentale en géométrie algorithmique. Dans la plupart des cas, elle est réalisée par une extension de l'algorithme de Bowyer-

Watson [Bowyer, 1981, Watson, 1981]. L'algorithme de Bowyer-Watson classique consiste à ajouter itérativement les racines à une triangulation de Delaunay déjà valide. Si la racine est contenue dans le cercle inscrit d'un ou plusieurs triangles de la triangulation, les triangles sont détruits. La cavité obtenue est à nouveau triangulée en incluant la racine considérée. Cet algorithme est difficilement parallélisable puisqu'il ajoute un à un des sommets au maillage. Lorsque la distribution des racines est homogène, il est possible d'explorer des solutions alternatives plus faciles à paralléliser, c'est l'objectif de la méthode que nous proposons.

Extensions de l'algorithme de Bowyer-Watson

La plupart des algorithmes étendent l'algorithme de Bowyer-Watson [Bowyer, 1981, Watson, 1981] (localisation de racines et triangulation de cavité) et s'appuient sur un tri spatial, comme l'utilisation de KD-tree [Amenta et al., 2003, Delage and Devillers, 2018] pour accélérer la phase de localisation des points les plus proches. Les implémentations parallèles sont optimisées pour des paramètres spécifiques, par exemple, pour calculer des maillages fins, il est possible de diviser l'ensemble de points par régions et de les distribuer sur plusieurs clusters [Rycroft, 2009, Nikos and Damian, 2003, Alexander and Rainald, 1995, De Cougny and Shephard, 1999, Gonzalez, 2016]. Lorsque le maillage tient dans la mémoire d'un seul ordinateur, des processeurs multicœurs avec des dizaines de threads peuvent éditer le maillage simultanément avec une proportion raisonnable de conflits [The CGAL Project, 2018, Batista et al., 2010, Inria, 2018, Remacle, 2017, Marot et al., 2018]. En effet, l'idée est d'allouer des zones où chaque processeur pourra éditer indépendamment le maillage. L'implémentation GPU peut être dérivée directement des stratégies CPU [Cao et al., 2014, Cao, 2014], mais l'édition simultanée d'une structure de données de maillage globale ne s'adapte pas très bien à un nombre élevé de processus et, en particulier, n'atteint pas une efficacité élevée sur les architectures GPU modernes.

Calculer indépendamment des cellules de Voronoï

Une approche différente consiste à calculer chaque cellule de Voronoï indépendamment comme l'intersection d'un ensemble de demi-espaces. Pour une racine S_i, nous savons que les points plus proches d'une racine S_j que de S_i n'appartiennent pas à la cellule de S_i. Cet ensemble est défini par un demi-espace $ax+by+cz+d>0$ ou $ax+by+cz+d<0$, où $ax+by+cz+d=0$ est le plan médian de S_iS_j. À partir de cette observation, nous pouvons définir la cellule de Voronoï $\mathcal{C}_i$ comme l'intersection des demi-espaces correspondants (voir figure 3.2).

Comme le diagramme de Voronoï, le diagramme de puissance, est généré par un ensemble de points ; cependant, contrairement au diagramme de Voronoï, ces points sont pondérés et ces poids influencent la taille des cellules. À l'instar des diagrammes de Voronoï, les cellules d'un diagramme de puissance sont des polyèdres convexes et peuvent également être considérées comme l'intersection de demi-espaces ; la seule différence est que ces demi-espaces ne sont pas générés par les plans médians, mais par des hyperplans radicaux (voir figure 3.3). Nous pouvons noter que même si les cellules d'un diagramme de puissance sont des polyèdres convexes, leur restriction à un domaine non convexe n'est pas nécessairement convexe, c'est également le cas pour les cellules de Voronoï. De plus, la restriction d'une cellule à un domaine peut même

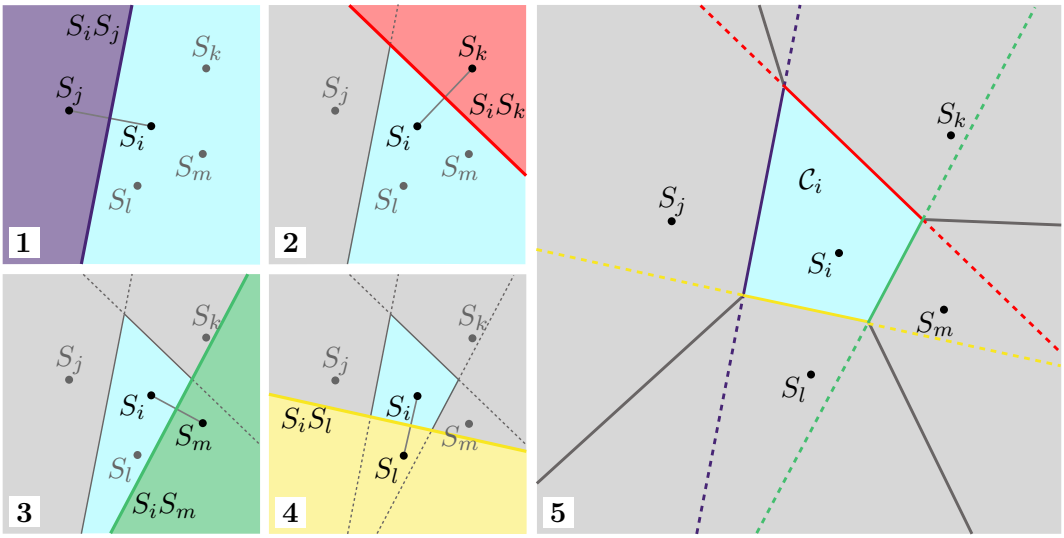


FIGURE 3.2 – La cellule de Voronoï $\mathcal{C}_i$ de la racine S_i (représenté en cyan dans l'image 5) peut être considérée comme l'intersection d'un ensemble de demi-plans représenté en couleur dans les images 1, 2, 3 et 4.

avoir des composantes non connexes, il faut donc faire particulièrement attention lorsque l'on calcule les intégrales.

Pour calculer les cellules de Voronoï (non restreintes) à l'aide de cette définition, chaque cellule est initialisée au domaine entier, puis est itérativement coupée par les demi-espaces générés par les autres racines. Heureusement, il n'est pas nécessaire de visiter l'ensemble des autres racines. En effet, nous pouvons visiter les racines voisines dans l'ordre croissant de distance à la racine concernée. Avant chaque découpe, nous calculons le rayon minimum d'une boule englobante centrée sur la racine et contenant la cellule. Si la distance entre la racine considérée et la racine voisine est supérieure à deux fois le rayon, alors le plan médian ne découpe pas la cellule et nous pouvons arrêter le processus. Cette condition est appelée **critère du rayon de sécurité** [Lévy and Bonneel, 2013]. Ce critère a été récemment amélioré dans l'article [Sainlot et al., 2017], afin de traiter des distributions de points non homogènes. Ce nouveau critère de validation par les coins permet d'avoir de meilleurs résultats lorsque les poids sont très éloignés du domaine de restriction, sinon il présente les mêmes performances que le rayon de sécurité. Nous avons opté pour l'utilisation du rayon de sécurité qui nous a paru mieux adapté à l'architecture GPU, mais il serait intéressant d'étudier l'approche par la validation par les coins.

Cette approche est facile à paralléliser, et elle surpasse les extensions de l'algorithme de Bowyer-Watson lorsque l'ensemble de points a une distribution homogène. Pour une implémentation sur GPU, il est préférable de se concentrer sur une structure de données plus compacte [Ray et al., 2018]. La restriction de ce dernier algorithme à un maillage tétraédrique a récemment été réalisée [Liu and Yan, 2019, Liu et al., 2020] en considérant l'intersection de chaque cellule de Voronoï avec ses tétraèdres voisins. Le choix des tétraèdres à intersecter avec la cellule de Voronoï considérée est complexe ; les auteurs ont choisi une heuristique basée sur leurs barycentres respectifs, ce qui conduit parfois à des erreurs de calcul. À l'inverse, notre algorithme est exhaustif et n'omet pas de paires de primitives à intersecter, évitant ainsi ces erreurs.

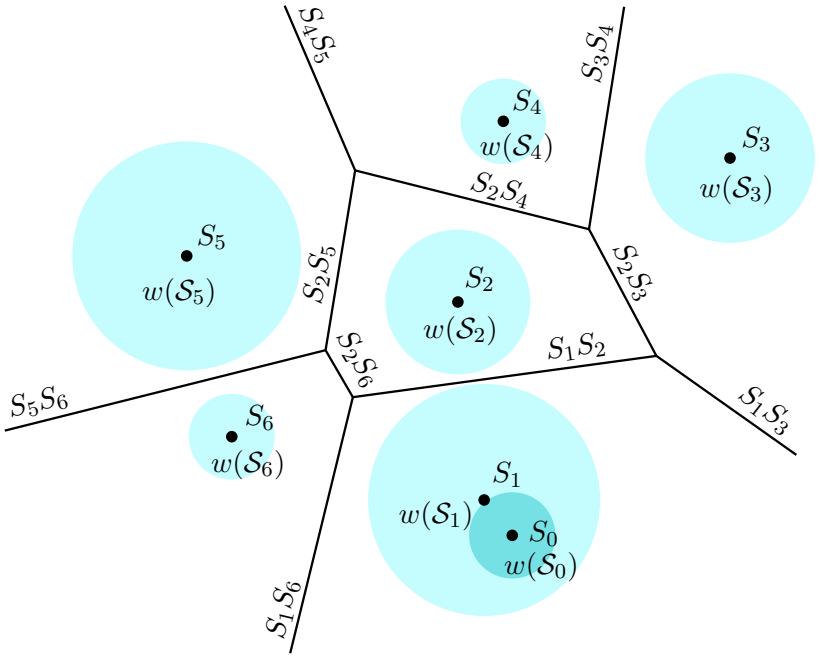


FIGURE 3.3 – En deux dimensions, une cellule du diagramme de puissance $\mathcal{C}_2$ de la racine S_2 peut être vue comme l'intersection de demi-plans générés par des axes radicaux (S_1S_2, S_2S_3, S_2S_4, S_2S_5, S_2S_6) définis par les cercles centrés sur leurs racines et dont les rayons correspondent à leurs poids respectifs (ici $w(\mathcal{S}_0) = 0.3$, $w(\mathcal{S}_1) = 0.8$, $w(\mathcal{S}_2) = 0.5$, $w(\mathcal{S}_3) = 0.6$, $w(\mathcal{S}_4) = 0.3$, $w(\mathcal{S}_5) = 0.8$ et $w(\mathcal{S}_6) = 0.3$).

De surcroît, nous évitons la tétraédrisation du domaine, en considérant uniquement une surface triangulée délimitant les frontières de notre domaine. Ces différents éléments et un algorithme d'ordonnancement efficace nous permettent d'être 100 fois plus rapide que l'algorithme proposé par Liu et al. [Liu et al., 2020]. Nous surpassons également les implémentations CPU pour le cas spécifique des distributions homogènes de racines. Notez qu'il existe également une implémentation GPU pour les diagrammes de Voronoï 2D restreints [Fei et al., 2014] permettant la restriction à une surface triangulée (et non à un volume). Cette restriction permet de découper directement chaque triangle du domaine sans calculer explicitement les cellules de Voronoï. Dans notre cas, nous nous concentrons sur une approche volumique permettant d'être exploitée par l'équipe pour des simulations numériques de fluide ou encore de mousse.

3.2.2 Notations

Nous détaillons dans cette sous-section les notations utilisées dans ce chapitre (voir figure 3.4).

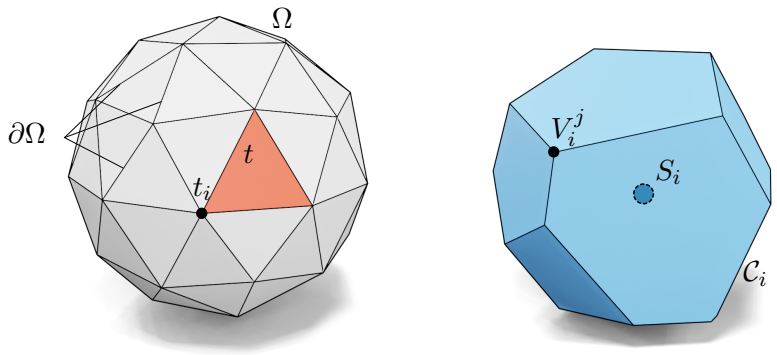


FIGURE 3.4 – À gauche un exemple de domaine de restriction représenté par la triangulation d'une icosphère. Dans cet exemple, le domaine de restriction Ω est convexe, mais il ne doit pas nécessairement l'être. À droite, en bleu, une cellule de Voronoï/cellule de puissance et ses notations.

Paramètres

K	nombre maximal de plus proches voisins ;
P	nombre maximal de plans coupant délimitant une cellule $\mathcal{C}_i$;
V	nombre maximal de sommets d'une cellule $\mathcal{C}_i$;

Domaine et structure spatiale

Ω	domaine de restriction ;
$\partial\Omega$	surface triangulée décrivant la limite du domaine Ω ;
t	triangle de $\partial\Omega$;
t_i	sommet de t ;
gd	grille régulière du domaine Ω ;
gr	grille des racines S_i ;

Cellule de Voronoï ou cellule de diagramme de puissance

S_i	racine de Voronoï ou de diagramme de puissance ;
$\mathcal{C}_i$	cellule associée à la racine S_i ;
V_i^j	sommet j de la cellule $\mathcal{C}_i$;
$w(\mathcal{S}_i)$	poids associé à la racine S_i ;
$\mathcal{V}(\mathcal{C}_i)$	ensemble des tétraèdres issus de la décomposition de $\mathcal{C}_i$;
v_i^k	tétraèdre k de $\mathcal{V}(\mathcal{C}_i)$;
$\|\|v_i^k\|\|$	le volume signé du tétraèdre v_i^k ;
$box(\mathcal{C}_i)$	la boîte englobante de la cellule $\mathcal{C}_i$;

Décomposition et calcul d'intégrale (voir section 3.3.4)

O	le coin de coordonnées minimales de $box(\mathcal{C}_i)$
$\mathcal{U}$	ensemble de tétraèdres formés par (O, t_i)
$\mathcal{U}'$	ensemble de tétraèdres modifiés (infinie)
u	tétraèdre de $\mathcal{U}'$

3.3 Contributions

Tout d'abord, nous présentons une vue d'ensemble de l'algorithme proposé, puis nous présentons les requêtes des plus proches voisins et le calcul des cellules de Voronoï de [Ray et al., 2018]. Nous montrons ensuite comment supporter les diagrammes de puissance et calculer la matrice de l'opérateur laplacien pondéré. Nous montrons également comment restreindre les intégrales au domaine de simulation. Enfin, nous présentons les performances de notre algorithme dans des cas d'applications représentatifs et nous discutons de ses limites. Cet algorithme a fait l'objet d'une publication [Basselin et al., 2021].

3.3.1 Vue d'ensemble

L'algorithme proposé améliore les performances par rapport à l'état de l'art :

- Nous obtenons une accélération allant d'un facteur 2 à 3 pour les diagrammes de Voronoï non restreints, en comparaison avec l'algorithme de [Ray et al., 2018], en introduisant une nouvelle stratégie d'ordonnancement.
- Notre algorithme est au moins 100 fois plus rapide que l'algorithme GPU de [Liu et al., 2020] pour l'intégration sur un domaine restreint, grâce à l'évaluation directe des intégrales sur le domaine, au lieu d'intersection des cellules avec un maillage tétraédrique.

Il présente les caractéristiques suivantes :

- Le domaine de simulation est délimité par une surface triangulée.
- Il prend en charge de grands ensembles de données et des distributions irrégulières d'ensembles de points (comme le bruit blanc) en consommant moins de mémoire que les travaux précédents.
- Les diagrammes de puissance sont pris en charge, avec certaines limitations sur la distribution des poids, en accord avec nos observations sur la simulation des fluides incompressibles.
- De nouvelles intégrales sont prises en charge : par exemple, l'opérateur laplacien pondéré qui est nécessaire pour les simulations de transport optimal et de fluide semi-discret.

Notre algorithme (voir algorithme 1) prend en entrée un ensemble de racines pondérées (points 3D et poids) et un domaine défini par une surface triangulée délimitant sa frontière. La sortie est un ensemble d'intégrales calculées sur le diagramme de puissance 3D restreint au domaine. Dans nos expériences, nous avons testé les intégrales nécessaires à l'algorithme de Lloyd et à la simulation de fluide par particules de puissance [de Goes et al., 2015a]. En effet, la simulation de fluide peut être effectuée en partitionnant l'espace en un ensemble de cellules de puissance et en considérant que les particules de fluide sont représentées par des cellules de puissance en mouvement. Cette simulation nécessite les volumes et les barycentres des cellules du diagramme de puissance restreint, ainsi que la matrice de l'opérateur laplacien pondéré.

Tout d'abord, nous précalculons sur le CPU une structure de données d'accélération pour accélérer les accès locaux à la géométrie du domaine (*grille_du_domaine* ou *gd*, algorithme 1, ligne 1). Cela permet un calcul très efficace des intégrales restreintes au domaine, comme expliqué dans §3.3.4.

```
   Input : float4 racines[#racines] ; // racines (coordonnées et poids)
   Input : triangulation ∂Ω ; // domaine
   Input : int K, P, V ; // hyperparamètres
   Output : float4 resultat[#racines] ; // intégrales (volume, barycentre, etc.)
 1 gd ← grille_du_domaine(∂Ω) ; // §3.3.4
 2 gr ← grille_des_racines(racines) ; // §3.3.3
 3 a_traiter ← {1, .., #racines} ;
 4 while a_traiter ≠ ∅ do
 5 |   int s ← taille_paquet(K) ;
 6 |   échec ← ∅ ;
 7 |   for paquet ∈ diviser(_traiter, s) do
 8 |   |   int knn[s][k] ← estimer_knn(gr, paquet) ; // §3.3.3
 9 |   |   resultat.mettre_à_jour(gd, paquet, knn, échec) ; // §3.3.3, §3.3.4
10 |   end
11 |   (K, P, V) ← 1.5(K, P, V) ;
12 |   a_traiter ∪ échec ;
13 end
14 gr.permuter(resultat) ; // Annule la réorganisation effectuée en §3.3.3
```

Algorithme 1 : Vue d'ensemble de l'algorithme

Ensuite, nous initialisons sur le GPU la structure de données de recherche spatiale pour les racines (*grille_des_racines* ou *gr*, voir algorithme 1, ligne 2). Après cela, nous commençons à évaluer les intégrales : nous récupérons les plus proches voisins de chaque racine (voir algorithme 1, ligne 8), puis nous les utilisons pour calculer les intégrales sur la restriction de la cellule au domaine (voir algorithme 1, ligne 9).

3.3.2 Stratégie d'ordonnancement

Pour traiter toutes les racines, nous introduisons une nouvelle stratégie d'ordonnancement qui optimise à la fois le temps d'exécution et la consommation mémoire GPU. Le processus d'ordonnancement global se déroule en deux boucles imbriquées :

La boucle externe (voir algorithme 1, ligne 4). L'idée derrière la boucle externe est de faire varier les paramètres de notre algorithme à chaque itération, de manière à réduire le temps d'exécution. En effet, la première itération est conçue pour être très rapide et traiter un grand nombre de cellules, mais peu parfois échouer à traiter quelques cellules. Alors que les itérations suivantes sont pensées pour être plus sûres, mais plus lentes et ainsi assurer le traitement des cas les plus complexes (mais moins nombreux). Pour cela, nous conservons les racines qui n'ont pas pu être traitées dans une pile ; puis nous les réévaluons lors de passes ultérieures.

Plus spécifiquement, dans notre algorithme, un compromis entre vitesse et taux de réussite est donné par trois paramètres K, P, V introduits dans [Ray et al., 2018] et détaillés dans §3.4. Nous initialisons les paramètres en fonction de la distribution des racines (et des poids pour le diagramme de puissance) afin de traiter la majorité des racines lors de la première passe. Chaque itération suivante augmente les paramètres K, P, V jusqu'à ce que les intégrales de toutes les cellules soient correctement évaluées. Nous avons observé que la simple mise à l'échelle de K, P, V par un facteur 1,5 à chaque itération était efficace dans les différents cas de figure (figure 3.13). No-

tez qu'il est plus facile d'évaluer les intégrales pour les cellules qui n'intersectent pas la frontière du domaine (§3.3.4), que pour les cellules qui sont coupées par la frontière (§3.3.4). Exécuter ces cas en même temps obligerait à exécuter des branches divergentes, ce qui est inefficace sur les architectures SIMD comme les GPU. Par conséquent, nous divisons la première passe en deux sous-étapes. Dans la première sous-étape, nous ne considérons que les cellules se trouvant entièrement à l'intérieur ou à l'extérieur du domaine, tandis que la deuxième sous-étape considère le reste des cellules. Dans la première sous-étape, nous calculons l'ensemble des cellules, et nous écartons celles dont la boîte englobante (*grille_du_domaine* dans l'algorithme 1, ligne 1) intersecte la limite du domaine.

La boucle interne (voir algorithme 1, ligne 7). Le calcul des intégrales sur une cellule nécessite d'abord d'obtenir les plus proches voisins de la racine, puis de construire la cellule et d'évaluer les intégrales. Bien qu'il soit possible d'exécuter toutes les étapes dans un même thread, il est plus efficace de précalculer les plus proches voisins de toutes les racines dans un grand tableau [Ray et al., 2018] (voir algorithme 1, ligne 8), car plus de threads peuvent être lancés simultanément sans forcer l'accès à la mémoire globale, dont l'accès est lent (seule la mémoire partagée est utilisée). La construction de la cellule et l'évaluation des intégrales sont ensuite calculées à partir de ce tableau (voir algorithme 1, ligne 9).

Un inconvénient de cette solution (observé dans [Ray et al., 2018]) est que le tableau des plus proches voisins ne tient pas dans la mémoire pour certains paramètres. Cette situation s'aggrave avec la matrice de l'opérateur laplacien pondéré qui consomme encore plus de mémoire. Notre solution consiste à traiter les racines par lots de mémoire maximale. Le nombre de racines par lot est déterminé par le paramètre K et la mémoire disponible sur le GPU (voir algorithme 1, ligne 5).

3.3.3 Calcul des cellules de diagramme de puissance

Dans cette sous-section, nous montrons comment calculer les diagrammes de puissance non restreints. Tout d'abord, nous revisitons le cas des diagrammes de Voronoï obtenus en calculant chaque cellule à partir de ses racines voisines. Nous montrons ensuite comment l'étendre aux diagrammes de puissance.

Requête des k plus proches voisins

Pour calculer une cellule de Voronoï d'une racine donnée $\mathcal{S}$, nous devons trouver ses k plus proches voisins. Pour ce faire, nous utilisons un algorithme [Ray et al., 2018] (voir figure 3.5) :

Si nous supposons une distribution homogène des racines dans l'espace, il est possible d'avoir une structure de recherche spatiale efficace que nous appelons *gr* ou *grille_des_racines* (voir algorithme 1, ligne 2). L'idée est de définir une grille régulière (voxel) dans l'espace, afin de pouvoir visiter efficacement toutes les racines dans un voxel donné. En pratique, cela se fait en réordonnant les points et en calculant un tableau d'offsets de voxel. Nous avons également besoin d'un tableau d'indirection pour rétablir l'ordre original des racines.

Grâce à cette structure de recherche spatiale, nous pouvons trouver rapidement les k plus proches voisins d'une racine donnée $\mathcal{S}$. L'idée est d'obtenir les racines à partir du voxel qui contient $\mathcal{S}$, puis de visiter les voxels voisins en anneaux concen-

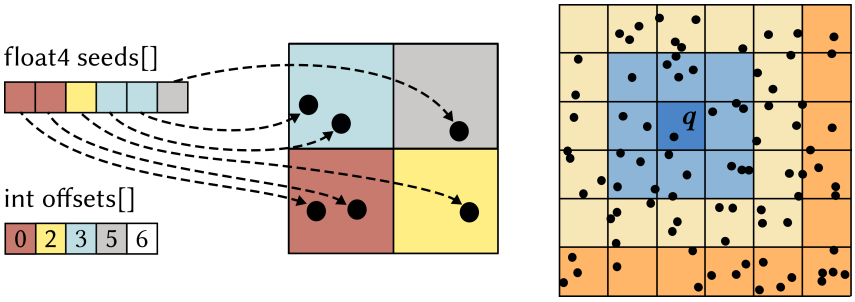


FIGURE 3.5 – **Gauche** : afin d'accéder rapidement à tous les points d'un voxel, nous trions l'ensemble des points par leur identifiant du voxel correspondant. **À droite** : pour trouver les k plus proches voisins pour un point $\mathcal{S}$ situé dans le voxel bleu foncé, nous visitons tous les voxels voisins en anneaux concentriques.

triques jusqu'à ce que l'on soit sûr d'obtenir les k plus proches voisins. La recherche s'arrête lorsque l'anneau le plus proche non visité est plus éloigné que les k voisins que nous avons déjà trouvés. Dans cet algorithme, le tableau des points voisins est trié par ordre croissant de distance à la racine $\mathcal{S}$.

La différence entre notre implémentation et [Ray et al., 2018] est la manière d'effectuer nos requêtes : au lieu de demander les k plus proches voisins pour toutes les racines en une seule fois, nous effectuons plusieurs requêtes. Cela permet de demander différents nombres de voisins (boucle externe) pour mieux s'adapter à la difficulté de calcul de chaque cellule et de garder le résultat stocké dans une mémoire partagée rapide (boucle interne).

Cellules convexes

Une cellule de Voronoï (diagramme de puissance) est un polyèdre convexe qui peut être vu comme une intersection d'un certain nombre de demi-espaces. Pour représenter un polyèdre convexe, nous utilisons la structure de données introduite dans [Ray et al., 2018]. Cette structure de données est compacte et bien adaptée à une implémentation sur GPU : elle consiste en un tableau d'équations de demi-espace (float4 ou double4) et un tableau de triplets d'entiers représentant les sommets du polyèdre. Chaque triplet stocke les indices de trois équations de demi-espace incidentes au sommet, énumérées dans le sens des aiguilles d'une montre.

Suivant [Ray et al., 2018], pour calculer la cellule de Voronoï de la racine $\mathcal{S}$, nous l'initialisons à la boîte englobante de toutes les racines. Ensuite, nous visitons les racines voisines et découpons itérativement la cellule par les demi-espaces correspondants. Puisque les voisins sont triés par distance croissante à la racine $\mathcal{S}$, nous pouvons arrêter de couper la cellule lorsque le critère du rayon de sécurité est satisfait, c'est-à-dire lorsque le voisin non visité le plus proche est plus éloigné de $\mathcal{S}$ que deux fois la distance maximale des sommets de la cellule à $\mathcal{S}$ (il est alors garanti que le plan médian de la racine plus éloignée ne coupe pas la cellule).

La routine clé ici est de couper la cellule convexe par un demi-espace, elle peut être effectuée en trois étapes (voir figure 3.6) :

1. détection des sommets à supprimer en évaluant simplement la nouvelle équation du demi-espace sur chaque sommet,

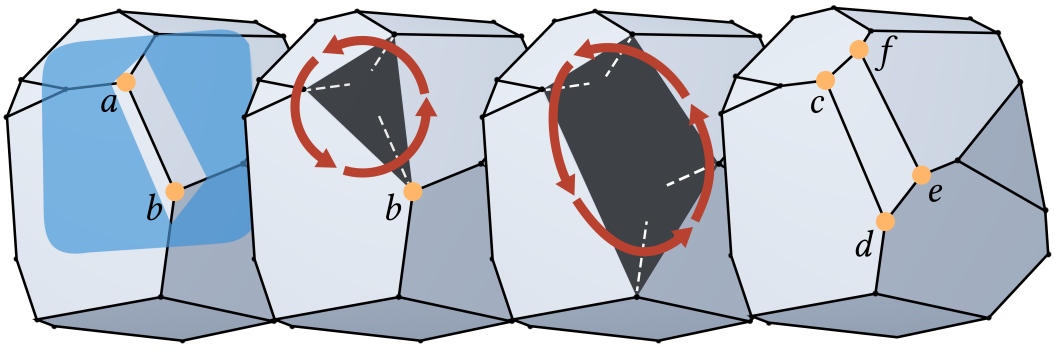


FIGURE 3.6 – Coupe d'un polyèdre convexe (en gris) par un demi-espace délimité par un plan (en bleu). La découpe est effectuée en trois étapes. Tout d'abord, nous détectons les sommets à supprimer (a et b). Ensuite, nous supprimons itérativement les sommets, créant ainsi un trou avec des bords pendants : nous supprimons le sommet a et nous initialisons la limite du trou (flèches rouges) comme étant le dual des arêtes pendantes. Ensuite, nous supprimons le sommet b et mettons à jour la limite du trou en conséquence. Enfin, nous créons la nouvelle facette et générons des sommets pour chaque arête pendante (en orange).

2. suppression itérative du tableau de sommets. Ils sont retirés dans un ordre qui permet de maintenir une représentation du trou (voir figure 3.6, flèches rouges) produit dans le maillage par le retrait des sommets. Cette représentation est une liste circulaire d'indices de demi-plans (stockés dans un tableau) telle que deux éléments consécutifs définissent une arête pendante.
3. ajout de la nouvelle équation de demi-espace au tableau et ajoute un nouveau sommet pour chaque arête pendante, le nouveau sommet étant simplement le triplet composé par le nouvel identifiant de demi-espace et des deux identifiants de demi-plans consécutifs dans la liste circulaire.

Cet algorithme est efficace sur GPU, car il ne manipule que des tableaux statiques et les relations entre les éléments sont faciles à mettre à jour : elles sont directement encodées dans la représentation des sommets.

Extension aux diagrammes de puissance

Un diagramme de puissance peut être calculé comme une intersection de demi-espaces, tout comme un diagramme de Voronoï. Cependant, nous considérons maintenant des hyperplans radicaux délimités par l'équation de chordale [Aurenhammer, 1987] : le point q est sur la chordale des racines $\mathcal{S}_i$ et $\mathcal{S}_j$ en respectant les poids $w(\mathcal{S}_i)$ et $w(\mathcal{S}_j)$ si :

$$2q \cdot (\mathcal{S}_j - \mathcal{S}_i) = w(\mathcal{S}_i) - w(\mathcal{S}_j) + |\mathcal{S}_j|^2 - |\mathcal{S}_i|^2$$

La modification de l'équation du demi-espace implique que le critère du rayon de sécurité n'est plus suffisant pour garantir l'exactitude de la découpe. En fait, en l'absence d'antécédents sur les poids, l'utilisation de calculs locaux n'est pas une option viable : par exemple, une racine avec un poids très élevé peut couvrir l'ensemble du domaine.

Heureusement, dans la plupart des applications (par exemple, la dynamique des fluides), les diagrammes de puissance sont souvent très similaires aux diagrammes de Voronoï, avec ce degré de liberté supplémentaire (les poids des racines) qui permet de translater les cellules ou d'ajuster le volume de certaines cellules. Dans les deux cas, les poids des racines voisines sont très similaires. De manière plus formelle, nous pouvons supposer que deux racines $\mathcal{S}_0, \mathcal{S}_1$ de poids w_0, w_1 sont telles que :

$$|w_0 - w_1| < 2\varepsilon|\mathcal{S}_0\mathcal{S}_1|$$

Sous cette hypothèse, ajouter ε à notre rayon de sécurité garantit que la cellule soit complètement calculée. Une fois le résultat calculé, nous pouvons vérifier que ε était suffisamment grand par une simple comparaison de volume. Si une cellule a été calculée sans prendre en compte un voisin qui aurait dû y participer, son volume sera trop grand. Cela permet de détecter que ε est trop faible lorsque la somme des volumes des cellules est supérieure au volume de Ω §3.5.

Pour la simulation de fluides, fixer ε à 0,2 fois le rayon de sécurité était suffisant pour parvenir à calculer toutes les cellules.

3.3.4 Intégration

La sous-section précédente détaille le calcule d'un diagramme de puissance non restreint, cellule par cellule. Dans cette sous-section, nous rappelons comment Ray et al. évaluent les intégrales. Puis nous explicitons notre contribution au calcul d'intégrale sur un domaine restreint.

Intégrer sur des cellules

Le principe est de calculer l'intégrale sur chaque cellule par une somme signée d'intégrales sur des tétraèdres [Lien and Kajiya, 1984], où les tétraèdres sont obtenus par projections orthogonales comme dans [Mullen et al., 2011].

Disons que nous avons une fonction $f(x) : \mathbb{R}^3 \to \mathbb{R}$ et pour la cellule $\mathcal{C}$ nous voulons calculer l'intégrale :

$$\int_{\mathcal{C}} f \, dV$$

L'idée principale est de décomposer le polyèdre $\mathcal{C}$ en un ensemble de tétraèdres $\mathcal{V}(\mathcal{C})$ et d'évaluer l'intégrale pour chaque tétraèdre. La difficulté réside dans le fait que la structure de données légère utilisée pour représenter le polyèdre ne permet pas d'inférer efficacement la connectivité entre les arêtes et les faces. Heureusement, puisque nous ne sommes pas intéressés par la connectivité, mais seulement par les intégrales sur la cellule, il existe un moyen de les calculer efficacement.

Supposons que nous divisions le polyèdre en pyramides : telles qu'une face de la cellule $\mathcal{C}_i$ est sa base et la racine S_i sa pointe, comme illustrée dans la figure 3.7. Cette pyramide peut à son tour être décomposée en un ensemble de tétraèdres en créant un tétraèdre par arête $V_i^j V_i^{j+1}$ de la base de la pyramide. Dans ce cas, les sommets du tétraèdre sont les sommets des arêtes V_i^j et V_i^{j+1}, la racine S_i et sa projection orthogonale H sur la base de la pyramide. Il est à noter que si la racine S_i (ou la projection H, ou la projection I) est à l'extérieur de la cellule certains tétraèdres auront un volume négatif, compensant ainsi le volume supplémentaire

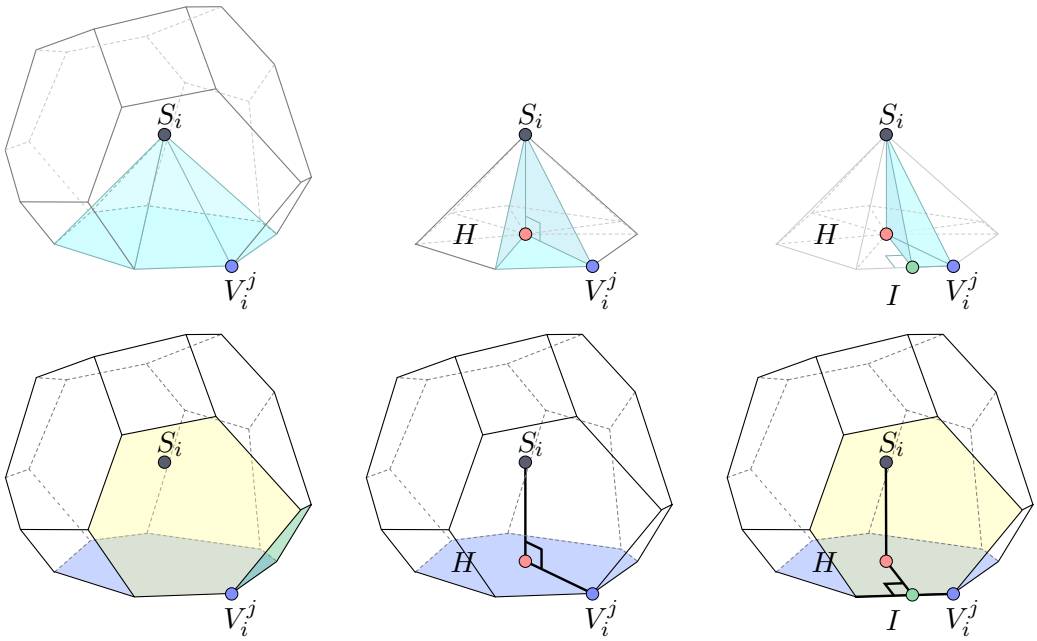


FIGURE 3.7 – **Ligne supérieure :** pour décomposer la cellule convexe en tétraèdres, nous la décomposons d'abord en pyramides associées à chaque face de la cellule (image de gauche). Ensuite, cette pyramide peut se décomposer en différents tétraèdres dont la base est un triangle, enfin ces tétraèdres peuvent à nouveau être subdivisés en deux. **Ligne supérieure :** Puisque nous utilisons une structure de données légère, il convient de retrouver ces informations à partir de la racine S_i et de l'ensemble de plans (en jaune, vert et bleu). Pour cela, nous projetons la racine S_i sur la face en bleu (point H). Ensuite, en projetant S_i sur l'intersection des plans jaune et bleu (point I, image de droite). À partir des points V_i^j, H, S_i et I il est possible de reconstruire un tétraèdre, en reproduisant cette construction pour tous les points V_i^j, nous pouvons obtenir la décomposition présentée ci-dessus.

des autres tétraèdres. Notez que la racine S_i ne peut être à l'extérieur de la cellule qu'avec des diagrammes de puissance. Chaque tétraèdre peut être divisé en deux tétraèdres en projetant S_i sur le bord de la facette (point I), car notre structure de données ne connaît pas les bords. Une fois de plus, le point I peut ne pas être sur le bord, mais une fois de plus, les tétraèdres de volume négatif équilibreront le volume supplémentaire des autres tétraèdres.

Avec cette approche, le problème de la connectivité persiste : nous devons savoir que V_i^j, V_i^{j+1} est une arête. Il existe un moyen de contourner le problème : si nous introduisons I, la projection orthogonale de la racine sur l'arête V_i^j, V_i^{j+1} ; alors le tétraèdre $(V_i^j, V_i^{j+1}, H, S_i)$ peut être vu comme une composition de deux tétraèdres (V_i^j, I, H, S_i) et (I, V_i^{j+1}, H, S_i).

De cette façon, nous pouvons calculer une décomposition $\mathcal{V}(\mathcal{C})$ sans restaurer la connectivité : pour chaque sommet du polyèdre, nous émettons six tétraèdres. Pour calculer les tétraèdres, il suffit de connaître la racine $\mathcal{S}$ et trois équations du plan incident au sommet. Ce n'est pas une décomposition minimale, mais elle peut être générée directement à partir de notre structure de données légère. Notez que les projections orthogonales H et I peuvent se trouver en dehors de la base de la

pyramide et de l'arête correspondante, mais les volumes signés s'équilibrent.

Ainsi, en ayant la décomposition $\mathcal{V}(\mathcal{C})$, l'intégrale peut être calculée comme suit :

$$\int_{\mathcal{C}} f\,dV = \sum_{v \in \mathcal{V}(\mathcal{C})} ||v|| \int_{v} f\,dV, \tag{3.1}$$

où $||v||$ est le volume signé du tétraèdre v.

Ceci est une conséquence directe du théorème de la divergence appliqué à un champ vectoriel radial centré sur la racine $\mathcal{S}$. Par exemple, si $f \equiv 1$, *i.e.* si nous souhaitons calculer le volume de la cellule, nous pouvons prendre un champ vectoriel égal à $\overrightarrow{\mathcal{S}P}/6$ au point P. En effet, le volume signé de v est égal au flux de ce champ à travers le triangle de v opposé à $\mathcal{S}$.

Dans la sous-section 3.4.2, nous détaillons l'évaluation des intégrales nécessaires à l'algorithme de Lloyd et à la simulation des fluides : le barycentre et le volume de la cellule, ainsi que l'aire de la frontière entre deux cellules.

Intégrer sur des cellules restreintes

Dans la sous-section précédente, nous avons montré comment intégrer une fonction f sur l'intersection entre un polyèdre $\mathcal{C}$ et la boîte englobante de l'ensemble des racines. Maintenant, nous nous intéressons à des domaines plus complexes : nous voulons remplacer la boîte englobante par un domaine Ω défini par une surface triangulée. Le problème consiste à intégrer la fonction f sur le domaine $\Omega \cap \mathcal{C}$.

Sans modifier radicalement notre méthode, la difficulté consiste à définir des demi-espaces pour intersecter notre cellule à partir d'un maillage arbitraire de la surface $\partial\Omega$. Pour y parvenir, nous proposons une décomposition exploitant une somme d'intégrales signées. Nous présentons d'abord l'idée de notre méthode sans optimisation, puis nous expliquons comment accélérer le processus en effectuant uniquement des calculs locaux.

Notre objectif est de calculer l'intégrale d'une fonction f sur l'intersection $\mathcal{C} \cap \Omega$, c'est-à-dire $\int_{\Omega \cap \mathcal{C}} f\,dV$. Nous voulons éviter le calcul direct de $\Omega \cap \mathcal{C}$, donc étudions d'abord comment calculer l'intégrale $\int_{\Omega} f\,dV$.

Le domaine Ω est représenté par une surface triangulée $\partial\Omega$; choisissons un point arbitraire O et définissons un ensemble de tétraèdres $\mathcal{U}$ comme suit : chaque triangle de $\partial\Omega$ crée un tétraèdre défini par trois points du triangle et le point O.

En utilisant le même argument que pour l'équation (3.1), nous pouvons écrire l'intégrale comme suit :

$$\int_{\Omega} f\,dV = \sum_{u \in \mathcal{U}} ||u|| \int_{u} f\,dV.$$

Cette décomposition est illustrée par la figure 3.8, où nous calculons le volume total du lapin de Stanford comme étant la somme des volumes des tétraèdres bleus ($||u|| > 0$) générés par les triangles dans $\partial\Omega_{in}$ moins la somme des volumes (non signés) des tétraèdres rouges ($||u|| < 0$) générés par $\partial\Omega_{out}$ (rouge).

Cette décomposition de Ω peut également être utilisée pour intégrer sur l'intersection $\mathcal{C} \cap \Omega$:

$$\int_{\mathcal{C} \cap \Omega} f\,dV = \sum_{u \in \mathcal{U}} ||u|| \int_{\mathcal{C} \cap u} f\,dV.$$

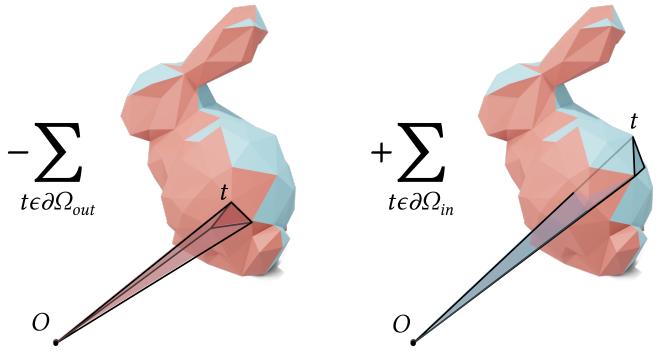


FIGURE 3.8 – Un exemple de calcul de volume : lorsque la normale du triangle t pointe vers le point O (à gauche), le volume du tétraèdre est retiré de la somme, sinon (à droite) il est ajouté.

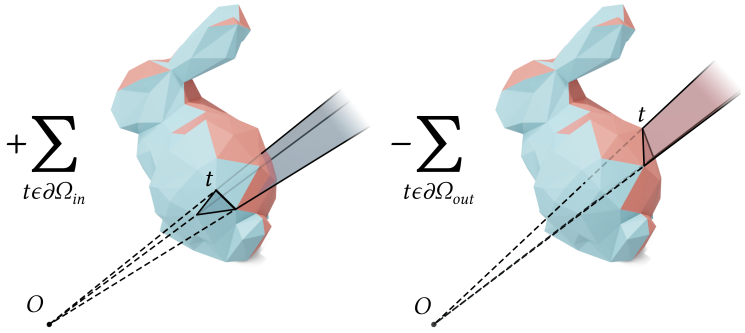


FIGURE 3.9 – Le volume du domaine peut également être calculé comme la somme des volumes signés des troncs de cône générés par les triangles et le point O.

Notez que tous les tétraèdres $u \in \mathcal{U}$ sont des polyèdres convexes, il est donc simple de calculer l'intersection $\mathcal{C} \cap u$: cela signifie simplement quatre intersections de demi-espaces à ajouter à la procédure de découpe décrite dans §3.3.3, et l'intégrale elle-même peut être calculée avec l'équation (3.1) :

$$\int_{\mathcal{C}\cap\Omega} f\,dV = \sum_{u\in\mathcal{U}} ||u|| \sum_{v\in\mathcal{V}(\mathcal{C}\cap u)} ||v|| \int_{v} f\,dV. \tag{3.2}$$

Alors que l'équation (3.2) nous permet de calculer l'intégrale sur la cellule convexe restreinte $\Omega \cap \mathcal{C}$, à l'aide de l'ensemble des tétraèdres $\mathcal{U}$, elle nécessite de nombreux calculs supplémentaires. Le domaine de simulation Ω est généralement défini par des milliers de triangles, ce qui rend peu pratique l'intersection de chaque cellule du diagramme de puissance avec tous les tétraèdres de $\mathcal{U}$. L'idée principale de l'optimisation est d'écarter prématurément les cellules vides $\mathcal{C} \cap u$ dans l'équation (3.2). En effet, il n'est pas intéressant de calculer des cellules de puissances se trouvant à l'extérieur du domaine. Le problème est que l'ensemble $\mathcal{U}$ ne nous permet pas de détecter facilement ces cellules. À cette fin, nous introduisons un ensemble modifié $\mathcal{U}'$. Si nous considérons le cône de pointe $O \notin \Omega$ qui serait divisé en deux régions par le triangle $t \in \partial\Omega$: le tétraèdre $u \in \mathcal{U}$ était la première moitié qui contient O, et maintenant nous considérons l'autre moitié (infinie) $u \in \mathcal{U}'$. Il peut être moins

intuitif de travailler avec $\mathcal{U}'$ qu'avec $\mathcal{U}$, mais ils partagent la même propriété clé : tout point en dehors du domaine Ω appartient à un nombre pair de polyèdres (où les intégrales s'annulent), alors que les points à l'intérieur du domaine appartiennent à un nombre impair de polyèdres contribuant à l'intégrale.

La figure 3.9 illustre le calcul du volume ($f \equiv 1$) avec l'ensemble $\mathcal{U}'$. Le volume total du lapin est égal à la somme signée des volumes des troncs de cône u générés par le point O et les triangles dans $\partial\Omega$. Ainsi, nous pouvons insérer $\mathcal{U}'$ au lieu de $\mathcal{U}$ dans l'équation (3.2) :

$$\int_{\mathcal{C}\cap\Omega} f\,dV = \sum_{u\in\mathcal{U}'} ||u|| \sum_{v\in\mathcal{V}(\mathcal{C}\cap u)} ||v|| \int_{v} f\,dV. \tag{3.3}$$

Notez que les cônes tronqués non bornés $u \in \mathcal{U}'$ ont un volume infini, cependant le signe est bien défini : pour un cône tronqué u généré par le triangle (t_0, t_1, t_2) et le point O, nous avons : $|u| = -\text{det3x3}(t_0 - O, t_1 - O, t_2 - O)$. Notez également que l'équation (3.3) suppose que $O \notin \Omega$; si ce n'est pas le cas, nous devons ajouter le volume de la cellule au résultat. Cela revient à créer un trou infinitésimal dans Ω pour en exclure O.

Les polyèdres dans $\mathcal{U}'$ sont convexes et délimités par 4 plans, il est donc simple de calculer l'intersection $u \cap \mathcal{C}$ pour $u \in \mathcal{U}'$. Utilisée telle quelle, l'équation (3.3) n'apporte pas un gain direct par rapport à l'équation (3.2), cependant elle ouvre une fenêtre pour une optimisation.

Optimisation pour les domaines complexes

L'objectif principal de cette sous-section est d'optimiser les calculs en les effectuant localement, c'est-à-dire en utilisant uniquement les triangles situés à proximité de la cellule $\mathcal{C}$. L'élément clé de cette optimisation est l'élimination rapide des cellules vides $\mathcal{C} \cap u$ dans l'équation (3.3).

Pour ce faire, nous calculons une boîte englobante $box(\mathcal{C})$ pour chaque cellule $\mathcal{C}$ (pas nécessairement une boîte minimale). Le point O est choisi pour être le coin de coordonnées minimales de la boîte et ainsi l'ensemble $\mathcal{U}'$ est défini par cellule. Notez que tout triangle $t \in \partial\Omega$ qui n'intersecte pas la boîte ($t \cap box(\mathcal{C}) = \emptyset$) produit un polyèdre u qui n'intersecte pas la cellule ($u \cap \mathcal{C} = \emptyset$). La figure 3.10 en fournit une illustration. Par conséquent, nous pouvons calculer l'équation (3.3) uniquement pour les polyèdres générés par les triangles intersectant $box(\mathcal{C})$.

En pratique, nous précalculons sur le CPU la structure de données d'accélération $gd \leftarrow grille\_du\_domaine(\partial\Omega)$ (voir algorithme 1, ligne 1). Nous plaçons le domaine Ω à l'intérieur d'une grille régulière de taille N^3. La résolution est choisie de manière à ce que chaque voxel ne soit intersecté que par quelques triangles. Pour chaque voxel de la grille, nous précalculons la liste des triangles dans $\partial\Omega$ qui intersectent le voxel, ainsi qu'un booléen indiquant si le coin de la coordonnée minimale du voxel est à l'intérieur de Ω ou non. Nous stockons les données dans une matrice et dans un vecteur de valeurs booléennes.

Avec cette structure de données, nous calculons des intégrales sur des cellules de puissance restreintes (voir algorithme 2). Tout d'abord (voir algorithme 2, ligne 1), nous choisissons la $box(\mathcal{C})$ comme étant la boîte englobante minimale de $\mathcal{C}$ composée des voxels de gd. Ensuite (voir algorithme 2, lignes 2 et 3), nous fusionnons la liste

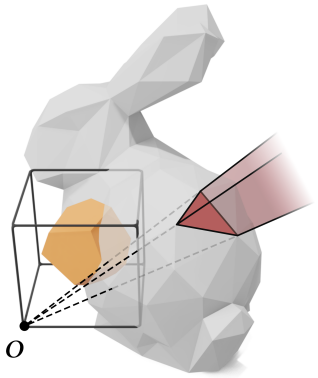


FIGURE 3.10 – Lorsque l'on intègre sur le polyèdre orange restreint au lapin, tout triangle (rouge) qui se trouve à l'extérieur de la boîte noire (contenant le polyèdre), génère un polyèdre (rouge) qui n'intersecte pas le polyèdre orange.

des triangles intersectant $box(\mathcal{C})$ et nous fixons le point O. Si O appartient à Ω, nous ajoutons l'intégrale sur $\mathcal{C}$ au résultat. Le reste de l'algorithme itère sur chaque triangle t, coupe $\mathcal{C}$ par les quatre demi-espaces qui définissent le cône tronqué créé par le triangle t et O, et ajoute l'intégrale correspondante si la normale de t pointe vers O et sinon la soustrait.

Input : gd ; // *grille*
Input : $\mathcal{C}$; // *cellule de diagramme de puissance*
Output : I ; // *Intégrales*
1 $box(\mathcal{C}) \leftarrow gr.sous\_grille\_englobante(\mathcal{C})$;
2 $T \leftarrow \bigcup_{(i,j,k)\in box(\mathcal{C})} gr[i,j,k].triangles$;
3 $O \leftarrow \min_{(i,j,k)\in box(\mathcal{C})} (i,j,k)$;
4 $I \leftarrow 0$;
5 **if** $O \in \Omega$ **then** $I \leftarrow integrer(\mathcal{C})$;
6 **for** $t \in T$ **do**
7 $\quad \mathcal{C}_{clip} \leftarrow \mathcal{C} \cap cone\_tronque(O,t)$;
8 $\quad I \leftarrow I - \det 3x3(t_0 - O, t_1 - O, t_2 - O) \times integrer\,(\mathcal{C}_{clip})$;
9 **end**

Algorithme 2 : Intégrales sur $\Omega \cap \mathcal{C}$

Le précalcul de la $grille\_du\_domaine(\Omega)$ n'est raisonnable que si nous devons calculer plusieurs diagrammes de puissance restreints à un domaine statique, dans ce cas, il introduit un temps supplémentaire largement compensé par le gain obtenu lors de la phase d'intégration.

3.4 Expériences de simulations

L'efficacité de notre algorithme dépend fortement de la régularité de la distribution des racines et des poids. Cette section est organisée comme suit. Nous expliquons d'abord comment nous avons ajusté les paramètres de l'algorithme. Puis nous

évaluons les performances de notre algorithme dans différents contextes, en faisant varier le domaine d'intégration, le type de distribution des racines et le type de diagramme (diagramme de Voronoï / diagramme de puissance). Enfin, nous concluons en présentant les limitations et quelques améliorations possibles.

Configuration des paramètres

Pour exécuter l'algorithme, nous devons choisir trois paramètres ; pour une performance maximale, nous les faisons varier en fonction des a priori sur les données dont nous disposons :

- Le paramètre K est le nombre de voisins qui est calculé pour chaque racine. Si K est grand, le temps d'exécution et la consommation de mémoire augmentent, mais s'il n'y a pas assez de voisins pour atteindre le rayon de sécurité, l'algorithme échoue à calculer la cellule.
- Le paramètre P est le nombre maximal de demi-espaces qui intersectent une cellule pendant sa construction. Il affecte la mémoire autorisée par thread et donc le nombre maximum de threads pouvant accéder simultanément à la mémoire partagée.
- Le paramètre V est le nombre maximal de sommets de la cellule pendant sa construction. Il a également un impact sur l'utilisation de la mémoire par thread, avec les mêmes conséquences sur les performances.

La meilleure situation pour notre algorithme serait un bruit bleu avec un poids constant par racine (i.e. un diagramme de Voronoï), mais en pratique les entrées ne sont pas toujours aussi triviales. Pour l'algorithme de Lloyd, la distribution des racines dans les premières itérations peut être un bruit blanc. Pour les simulations de fluides, le bruit est plus proche du bruit bleu, mais la différence de poids des racines des voisins nous oblige à considérer de plus grands voisinages.

Pour chaque situation, les paramètres K, P, V doivent être ajustés pour obtenir de meilleures performances. Leurs valeurs doivent être suffisamment grandes pour n'échouer que sur un petit pourcentage de cellules, et aussi faible que possible pour minimiser le temps d'interrogation de plus proches voisins (avec le paramètre K) et la consommation mémoire (pour tous les paramètres).

Nous avons trouvé nos paramètres par une approche expérimentale : nous exécutons notre algorithme avec des valeurs de K, P, V qui nous permettent de calculer toutes les cellules sans échec et tout en enregistrant la valeur requise de K, P, V pour chaque cellule. Cela nous a donné une distribution de K, P et V pour chaque configuration et nous avons fixé K, P, V pour que les cellules soient traitées avec succès dans environ 95 % des cas. En pratique, K varie beaucoup en fonction du type de racine et de la distribution des poids, mais P et V sont juste un peu plus bas pour le bruit bleu.

Nos cas de test utilisent les trois distributions de racines introduites dans [Ray et al., 2018] (bruit bleu, bruit blanc et grille perturbée) avec 10M de racines. La figure 3.11 fournit les histogrammes de K, P, V pour chaque distribution à la fois pour les diagrammes de Voronoï et les diagrammes de puissance (Laguerre-Voronoï). Les poids des diagrammes de puissance ont été calculés pour produire un volume égal pour toutes les cellules du diagramme.

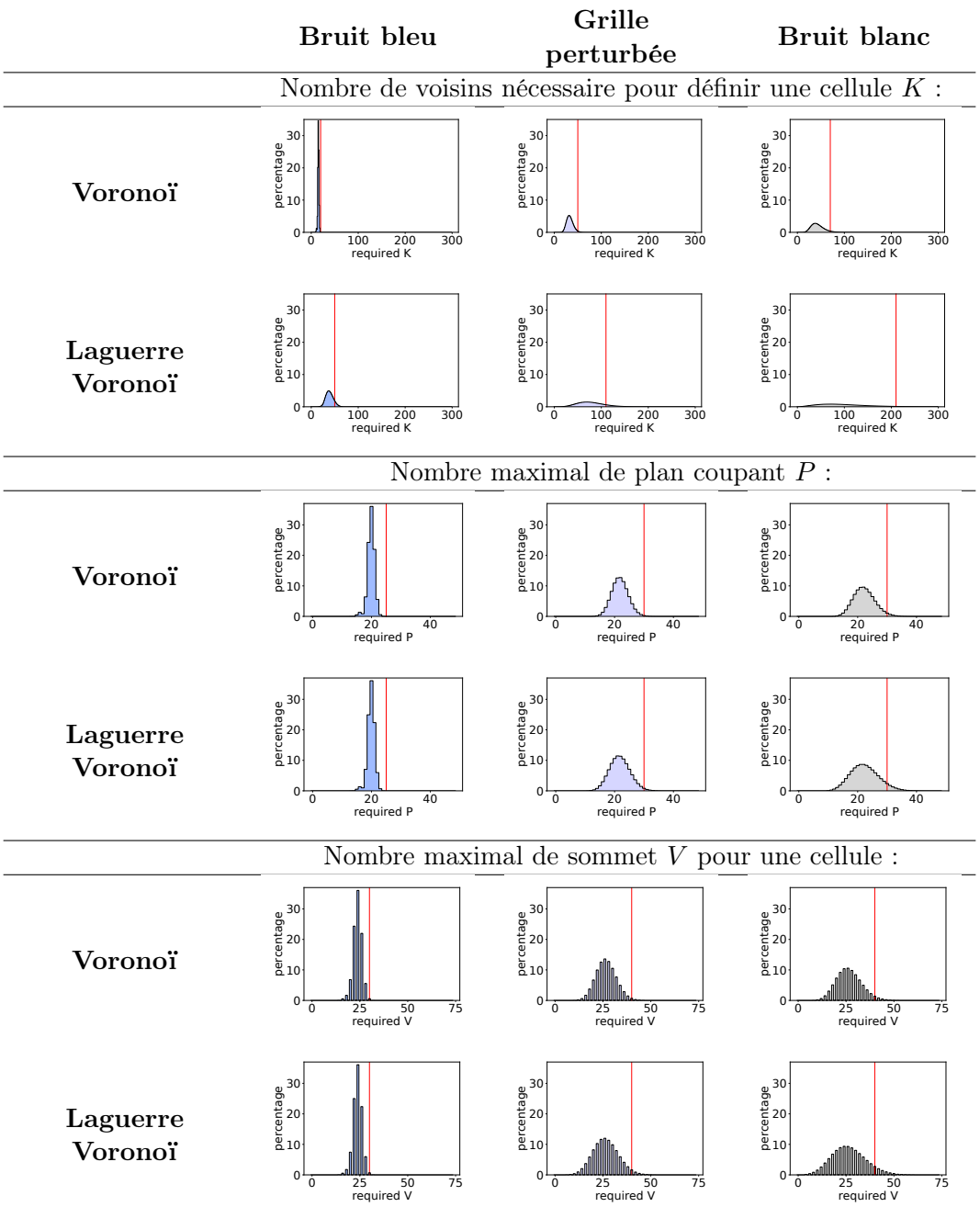


FIGURE 3.11 – Statistiques sur 10M de cellules de Voronoï et 10M de cellules de diagramme de puissance. **Les colonnes** correspondent à trois jeux de données générés : le bruit bleu, la grille perturbée et le bruit blanc. **Haut :** histogramme de la distribution du nombre de voisins nécessaires pour atteindre le critère de rayon de sécurité. **Milieu :** nombre maximal de plans lors de la découpe d'une cellule. **Bas :** nombre maximal de sommets lors de la découpe d'une cellule. Les lignes rouges montrent les valeurs des paramètres que nous avons choisis pour la première itération de l'algorithme.

3.4.1 Temps d'exécution

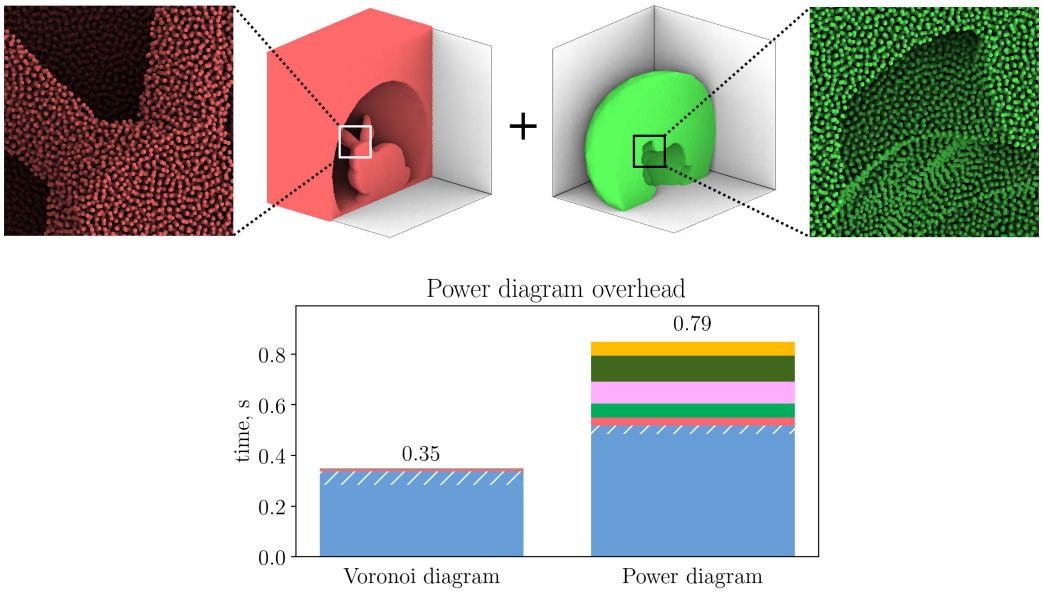


FIGURE 3.12 – Les temps nécessaires au calcul d'un diagramme de puissance. **Gauche :** Diagramme de Voronoï calculé sur 10M racines (bruit bleu), restreint au globe de neige du lapin. Ce diagramme comporte environ 4 millions de cellules de Voronoï non vides, les racines sont représentées en vert dans l'image du bas. Les racines produisant des cellules vides sont représentées en rouge. **Droit :** Diagramme de puissance, les poids des 4M racines sont calculés pour produire un volume égal pour toutes les cellules.

Ce travail étend l'état de l'art du calcul des diagrammes de Voronoï sur le GPU [Ray et al., 2018] qui suppose une distribution raisonnable des racines.

Notre implémentation OpenCL utilise des nombres à virgule flottante de 64 bits. Nous présentons des expériences réalisées sur une Nvidia Tesla V100 ; pour autant, cela peut parfaitement être réalisé avec un GPU grand public (ex. GTX1080) qui ne dispose pas de type double rapide. Sur une GTX1080, notre choix de nombres à virgule flottante 64 bits double grossièrement le temps d'exécution par rapport aux nombres flottants 32 bits.

Les résultats sont résumés dans la figure 3.13. Nous avons effectué les calculs sur trois ensembles de données (10M de racines) avec différents types de distributions de points (bruit bleu, grille perturbée, bruit blanc). La colonne la plus à gauche donne les performances de [Ray et al., 2018] (en virgule flottante 32 bits) ; la deuxième colonne donne les temps d'exécution de notre algorithme sans les nouvelles fonctionnalités : nous avons calculé les diagrammes de Voronoï sans restrictions de domaine. Notre stratégie d'ordonnancement permet d'utiliser de meilleurs paramètres pour la plupart des cellules, ce qui se traduit par un facteur d'accélération de 2 à 3 selon le type de distribution des racines.

Les 6 colonnes les plus à droite correspondent aux diagrammes de Voronoï restreints à différents domaines. Nous observons que l'ajout de la restriction introduit un surcoût raisonnable (typiquement inférieur à 15%). Notez que la première passe est

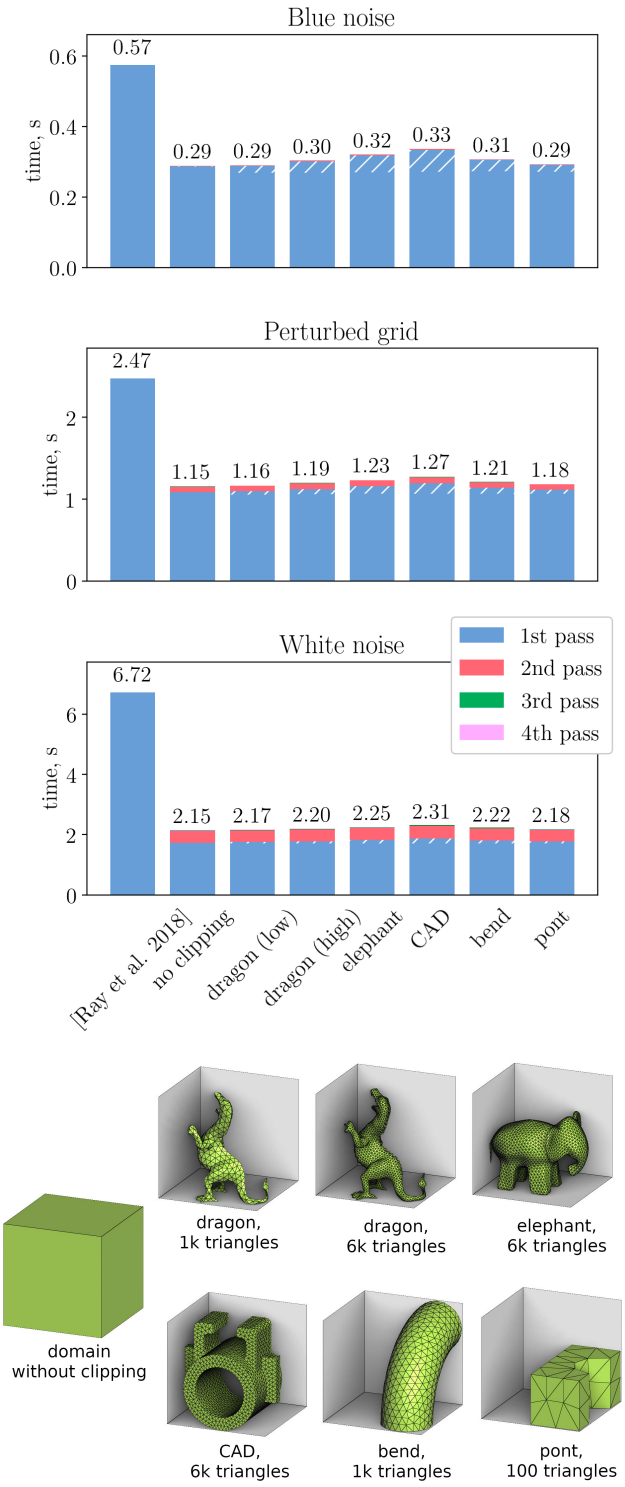


FIGURE 3.13 – Statistiques sur 10M racines de Voronoï avec distribution de bruit bleu **(en haut)**, situées sur une grille perturbée **(milieu)** et bruit blanc **(en bas)**. **La colonne la plus à gauche** correspond au temps d'exécution de l'algorithme [Ray et al., 2018] ; **la deuxième colonne** est notre résultat sans restriction ; **6 colonnes de droite** correspondent à différents domaines restreints.

effectuée en deux sous-étapes : dans la première sous-étape, nous calculons uniquement les cellules dont nous garantissons qu'elles se trouvent entièrement à l'intérieur ou à l'extérieur du domaine, et dans la deuxième sous-étape, nous calculons le reste des cellules. Le diagramme à barres montre la première sous-étape en bleu et la deuxième sous-étape en bleu hachuré.

Pour le calcul des diagrammes de puissance, nous devons augmenter le rayon de sécurité, dans nos expériences, nous l'avons augmenté d'un facteur 1.2 pour les simulations de fluides [de Goes et al., 2015a, Gallouët and Mérigot, 2017]. Cela peut sembler une différence mineure, mais cela augmente largement le nombre de voisins à considérer par $1{,}2^3$, ce qui double presque la région où les racines voisines peuvent participer à une cellule de puissance. Nous observons (voir figure 3.12) un impact sur les performances qui est proportionnel à l'augmentation de la taille du voisinage : presque un facteur de deux. Pour une variation plus importante des poids, une stratégie plus complexe serait nécessaire pour éviter de perdre en performances.

3.4.2 Applications

Nous montrons comment utiliser notre méthode pour deux applications : l'algorithme de Lloyd et la simulation de fluides incompressibles. Chaque cas nécessite d'évaluer des intégrales spécifiques.

Barycentre et volume de la cellule

Pour l'algorithme de Lloyd, seul le barycentre de chaque cellule doit être calculé. Il est égal à l'intégrale des coordonnées x, y et z sur le volume, divisée par le volume de la cellule. En pratique, pour chaque tétraèdre $v \in \mathcal{V}(\mathcal{C}_{clip})$, nous calculons son volume signé $\mathrm{vol}(v)$ et son barycentre $\mathrm{bary}(v)$, que l'on additionne au fur et à mesure. Alors le barycentre du polyèdre $\mathcal{C}_{clip}$ est donné par :

$$\mathrm{bary}(\mathcal{C}_{clip}) = \frac{\sum\limits_{v \in \mathcal{V}(\mathcal{C}_{clip})} \mathrm{vol}(v)\,\mathrm{bary}(v)}{\sum\limits_{v \in \mathcal{V}(\mathcal{C}_{clip})} \mathrm{vol}(v)}.$$

Les barycentres et les volumes de tous les polyèdres $\mathcal{C}_{clip}$ sont, à leur tour, additionnés à la volée pour produire le barycentre et le volume de la cellule $\mathcal{C}$ (voir algorithme 2, ligne 8).

Matrice de l'opérateur laplacien pondéré

Les simulations de fluides de type particules de puissance [de Goes et al., 2015b, Gallouët and Mérigot, 2017] imposent l'incompressibilité du fluide en calculant un transport optimal semi-discret. L'intégration sur un diagramme de puissance est le goulot d'étranglement de la simulation.

En plus du barycentre et du volume de la cellule, le transport optimal semi-discret [de Goes et al., 2012, Mérigot, 2011, Lévy, 2015, Kitagawa et al., 2016, Lévy and Schwindt, 2018] nécessite l'évaluation de l'opérateur laplacien pondéré. Pour évaluer cette matrice, pour chaque paire de cellules adjacentes, nous avons besoin de la surface de la frontière, de son barycentre et de la distance entre les racines correspondantes.

Nous pouvons représenter ces informations par une matrice creuse, fournissant toutes les valeurs nécessaires pour chaque paire de racines. Afin de calculer cette matrice, nous gardons la trace des racines voisines chaque fois que nous générons une équation de demi-espace. Cela nous oblige à maintenir un tableau de racines synchronisé avec le tableau d'équations de demi-espace. Une fois que la cellule est calculée, la distance entre la racine et ses voisins est facile à calculer. Comme précédemment, la décomposition des cellules en tétraèdres permet de calculer l'aire des frontières ainsi que les barycentres correspondants. En pratique, la symétrie de la matrice permet de calculer uniquement la partie supérieure de la matrice (lorsque l'identifiant de la racine voisine est plus grand que l'identifiant de la racine actuelle). La matrice de sortie est stockée par un tableau de taille fixe de coefficient non nul pour chaque racine.

3.5 Discussion

Comparé à d'autres algorithmes GPU

Pour le calcul des diagrammes de Voronoï non restreints, notre algorithme offre un gain de vitesse de deux fois supérieur à celui de l'algorithme [Ray et al., 2018]. L'implémentation récente des diagrammes de Voronoï restreints sur GPU [Liu et al., 2020] traite 60K racines par seconde (30K racines en 0,52 seconde) dans le cas le plus favorable. Le fait que nous travaillions directement avec la limite du domaine nous permet, avec un domaine de restriction similaire, de traiter 4300K racines par seconde (10M racines en 2.31 secondes). De plus, contrairement à [Liu and Yan, 2019, Liu et al., 2020] notre algorithme ne souffre pas d'erreurs de calcul dues à l'intersection avec des tétraèdres.

Enfin, nous fournissons toutes les caractéristiques nécessaires au calcul d'un transport optimal semi-discret dans un domaine (cellules de puissance, restriction à Ω et nouvelles intégrales).

Par rapport aux algorithmes du CPU.

Notre solution est un ordre de grandeur plus rapide pour nos cas d'utilisation. Cependant, notre algorithme est efficace pour certaines distributions de racines et de poids des diagrammes de puissance. Pour un usage général, les implémentations CPU restent une meilleure option. Sur un processeur Intel i7-6800K à 3,40 GHz de 12 cœurs, une implémentation multithread de Geogram [Inria, 2018] prend 41,9 secondes pour calculer le diagramme de puissance de la figure 3.12. Plus précisément, il faut 26,6 secondes pour calculer le diagramme de Laguerre et 15,3 secondes pour le restreindre au domaine. Les temps de calcul de CGAL [The CGAL Project, 2018] sont similaires à ceux de Geogram. Notre parallélisation permet de calculer en 0,79 seconde le diagramme de puissance restreint sur la Tesla V100.

Robustesse des prédicats.

Il s'agit d'un problème courant dans les algorithmes permettant d'estimer des diagrammes de puissance. Comme dans [Ray et al., 2018], un repli sur CPU est toujours possible en cas d'échec. En pratique, nous avons observé des échecs numériques

uniquement avec la virgule flottante 32 bits, la mise à niveau vers l'utilisation de nombre à 64 bits a résolu nos problèmes.

Précision d'intégration en virgule flottante.

Nous pouvons craindre que notre algorithme souffre de ce que l'on appelle *catastrophic cancellation* provenant de la somme et de la soustraction d'éléments de volume partitionnés. En effet, si le point O (se référer à la figure 3.10) était mal choisi (par exemple l'origine), cela pourrait être le cas. Pour chaque cellule, nous choisissons le point comme coin de coordonnées minimales de la boîte englobante de la cellule, ainsi le point n'est pas loin de la cellule, ce qui améliore grandement la précision.

Pour valider la précision de nos résultats, nous avons effectué un test où nous comparons nos résultats avec les résultats calculés sur un CPU. Nous avons calculé le volume et le barycentre de chaque cellule du diagramme de puissance pour l'ensemble des données présentées dans la figure 3.12. Ce calcul a été effectué à la fois sur le CPU (avec une implémentation de Geogram) et sur le GPU (avec notre méthode) avec des nombres à virgule flottante de 64 bits.

Nous avons donc deux ensembles de valeurs 64 bits $\left\{v_i^{CPU}\right\}_{i=1}^n$ et $\left\{v_i^{GPU}\right\}_{i=1}^n$, qui correspondent au volume des cellules non vides calculées respectivement sur le CPU et le GPU, où n représente le nombre de cellules non vides. Nous avons également deux ensembles de valeurs de triplet de 64 bits $\left\{b_i^{CPU}\right\}_{i=1}^n$ et $\left\{b_i^{GPU}\right\}_{i=1}^n$ correspondant aux barycentres des cellules.

De plus, nous avons calculé sur le CPU le volume du domaine V (`double`) et son barycentre B (64 bits). Ce calcul a été effectué directement sur le domaine. D'abord nous recalculons les invariants à partir des données du GPU et nous comparons la précision :

$$\frac{\left|\sum_i v_i^{GPU} - V\right|}{V} \approx 3 \cdot 10^{-15}$$

Ensuite, pour chaque coordonnée c, nous vérifions la différence entre les deux façons de calculer le barycentre du domaine :

$$\max_{c\in\{x,y,z\}} \frac{\left|\sum_i \frac{v_i^{GPU} b_i^{GPU}[c]}{V} - B[c]\right|}{B[c]} \approx 3 \cdot 10^{-14}$$

Ensuite, nous comparons deux diagrammes de puissance. D'abord, nous comparons les valeurs extrêmes de la discordance :

$$\frac{\max_i \left|v_i^{GPU} - v_i^{CPU}\right|}{V/n} \approx 6 \cdot 10^{-11}$$

$$\max_{c\in\{x,y,z\}} \max_i \frac{\left|b_i^{GPU}[c] - b_i^{CPU}[c]\right|}{B[c]} \approx 2 \cdot 10^{-6}$$

Nous terminons l'évaluation en calculant la discordance moyenne :

$$\frac{\sum_i \left|v_i^{GPU} - v_i^{CPU}\right|}{V} \approx 3 \cdot 10^{-12} \quad \max_{c\in\{x,y,z\}} \sum_i \frac{\left|b_i^{GPU}[c] - b_i^{CPU}[c]\right|}{nB[c]} \approx 2 \cdot 10^{-11}$$

Ainsi, notre algorithme GPU offre une précision suffisante pour le calcul numérique des intégrales dans le cas de nos applications de simulation numérique.

A priori sur les poids de diagramme de puissance.

Pour calculer les cellules de diagramme de puissance, nous pouvons nous assurer qu'un nombre suffisant de voisins sont visités en augmentant simplement les critères de rayon de sécurité utilisés pour les cellules de Voronoï. Cette stratégie s'est avérée efficace pour nos tests sur les simulations de fluides, mais elle repose sur une hypothèse forte sur les limites des variations de poids. D'autres applications peuvent avoir besoin d'une heuristique pour adapter localement ce rayon de sécurité, ou utiliser d'autres stratégies en fonction de leur propre variation de poids.

Notez qu'il existe un moyen d'affirmer l'exactitude du calcul du diagramme de puissance. Si le critère du rayon de sécurité ne reflète pas la variation des poids, cela signifie que certaines intersections de demi-espaces seront manquées. Dans ce cas, la somme des volumes des cellules du diagramme de puissance sera supérieure au volume du domaine.

3.6 Conclusion

Nous avons testé notre technique avec succès pour des problèmes d'échantillonnage ou encore pour la résolution de transport optimal semi-discret. Nos travaux permettent d'évaluer des intégrales sur un volume, définies par une triangulation, beaucoup plus rapidement que les solutions équivalentes sur CPU et nécessite moins d'a priori sur les distributions de racines et de poids que les précédents travaux réalisés sur GPU.

Chapitre 4

Accélération de la méthode d'ajustement d'une surface à un nuage de points

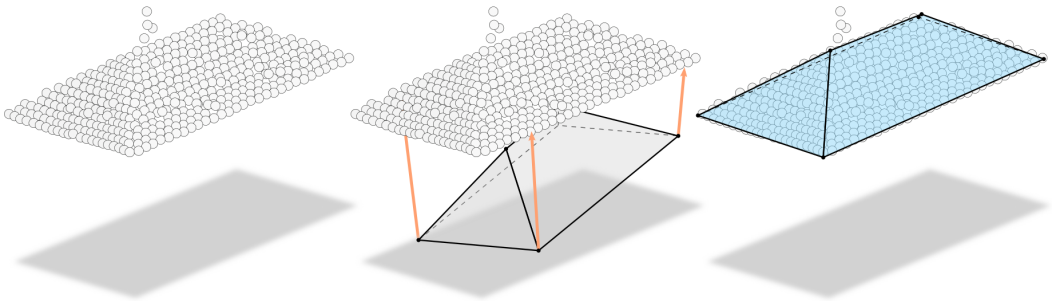


FIGURE 4.1 – Recherche de l'ajustement d'une surface à partir d'un nuage de points de toits à quatre pans : de gauche à droite, nuage de points brut ; initialisation de la surface et gradient (en orange) en chaque sommet de la surface ; résultat obtenu.

Sommaire

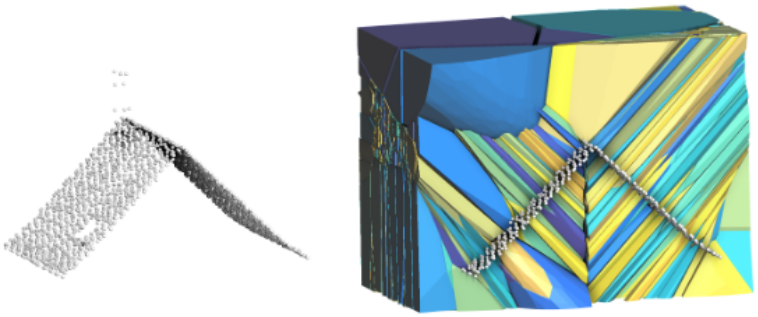


FIGURE 4.2 – Exemple de nuage de points à gauche et son diagramme de Voronoï à droite. Le nuage de points ressemble à une surface échantillonnée puis bruitée. Son diagramme de Voronoï présente des cellules anisotropes : elles sont allongées dans la direction de la normale à la surface, comme décrit par Merigot et al. [Mérigot et al., 2011].

Dans le chapitre 2, nous avons présenté une méthode permettant d'ajuster une surface soumise à des contraintes géométriques à un nuage de points. Malgré d'excellents résultats qualitatifs, cette méthode présente des temps d'exécution élevés prohibant son intégration dans le logiciel Rhinocéros. Afin de rendre possible son utilisation par la société RhinoTerrain, nous nous sommes intéressés à ce problème de performance.

Plusieurs profilages révèlent qu'en moyenne 80% du temps d'exécution global est dédié au calcul du diagramme de Voronoï restreint, indiquant clairement un goulot d'étranglement dans le bloc en lui-même. Dans le chapitre 3, nous nous sommes naturellement intéressés à l'optimisation de l'estimation d'intégrale sur des diagrammes de Voronoï restreints. Dans ce but, nous avons conçu un algorithme parallèle sur carte graphique permettant de calculer plus efficacement des diagrammes de puissance (généralisation des diagrammes de Voronoï). Cette méthode s'avère très performante pour les pipelines de simulation de l'équipe, où le calcul des diagrammes de puissance est également un goulot d'étranglement.

Fort de ces résultats, une solution se présente donc à nous. Elle consiste à remplacer notre calcul de diagramme de Voronoï de l'algorithme d'appariement par la version GPU introduite dans le chapitre précédent. Par la suite, nous discutons de l'intérêt de cette combinaison §4.1, puis nous présentons une approche plus adaptée à notre problème §4.2, enfin nous présentons les résultats obtenus §4.3 en fournissant une comparaison, en termes de temps d'exécution (voir sous-section 4.3.2) et de robustesse (voir sous-section 4.3.1) avec l'algorithme proposé dans le chapitre 2.

4.1 Pourquoi ne pas utiliser la version GPU ?

Comme indiqué précédemment, une solution naturelle serait d'utiliser notre optimisation du calcul du diagramme de Voronoï sur GPU pour l'ajustement de surface (cf. chapitre 2). Mais cette solution présente les deux défauts rédhibitoires suivants :

— D'un point de vue industriel, l'intégration de cette méthode dans nos logiciels nécessiterait que les clients disposent d'une carte graphique performante.

Même si les GPU tendent à se démocratiser et équipent aujourd'hui la majorité de nos ordinateurs, ils ne restent pas pour autant indispensables. De plus, les spécifications matérielles du logiciel pour lequel nous développons des plugins (Rhinocéros) recommandent uniquement à ses utilisateurs de posséder une carte vidéo compatible avec OpenGL 4.1. Par conséquent, RhinoTerrain ne souhaite pas utiliser la programmation GPU, car l'ensemble de ses utilisateurs ne disposent pas nécessairement d'une carte graphique.

— D'un point de vue technique, nos nuages de points ne répondent pas toujours aux critères d'homogénéité nécessaires à un calcul efficace des diagrammes de Voronoï sur GPU. Ils s'apparentent à des cartes de hauteur bruitées (voir figure 4.2) ; la distribution est homogène sur les axes x et y mais largement hétérogène sur l'axe z. Or, pour obtenir de meilleures performances, le calcul de diagramme de Voronoï restreint sur GPU nécessite en entrée une distribution variant du bruit bleu au bruit blanc. En effet, lorsque la distribution est relativement homogène, il est rapide de s'assurer que toutes les cellules du diagramme sont correctement calculées, grâce au critère du rayon de sécurité [Lévy and Bonneel, 2013]. Pour rappel, ce critère garanti qu'une cellule A ne partagera pas de face avec une cellule B voisine, si la racine de B est à une distance de A deux fois supérieures au rayon de la sphère englobante de A (voir figure 4.3). Une cellule peut alors être reconstruite en visitant un nombre limité de racines voisines : celles contenues dans son rayon de sécurité. Dans le cas des distributions homogènes, les cellules sont isotropes et peuvent être facilement approximées par une sphère centrée sur leur racine dont le rayon contiendra peu de racines voisines (voir la figure 4.3). Les cellules pourront donc être rapidement reconstruites en visitant peu de racines. Dans le cas de nos nuages de points, les cellules sont souvent anisotropes et le rayon de sécurité est d'autant plus élevé que la taille des sphères englobantes des cellules augmente. Donc, pour s'assurer que chaque cellule soit correctement calculée nous devons visiter bien plus de voisins, résultant en de mauvaises performances. Toutefois, puisque nos points sont très homogènes une fois projetés dans un plan x, y, il pourrait être envisageable d'appliquer l'algorithme GPU sur la projection de ces points.

Il est à noter qu'il existe aussi une autre solution conçue pour pallier ce problème. Elle a été récemment proposée par Sainlot et al. [Sainlot et al., 2017]. Pour cela, ils remplacent le critère de rayon de sécurité par un critère de validation dite par les coins. Cette méthode a été testée sur des ensembles de données destinés au transport optimal semi-discret, c'est-à-dire, des nuages de points issus de surface discrétisée. Nous n'avons donc aucune garantie de son efficacité sur nos distributions, à cause du bruit présent dans nos données. De plus, il faudrait également user d'une accélération GPU pour obtenir des performances raisonnables, cette solution n'est donc pas envisageable.

Pour des questions industrielles, nous préférons explorer une nouvelle stratégie, qui est présentée dans la suite de ce chapitre.

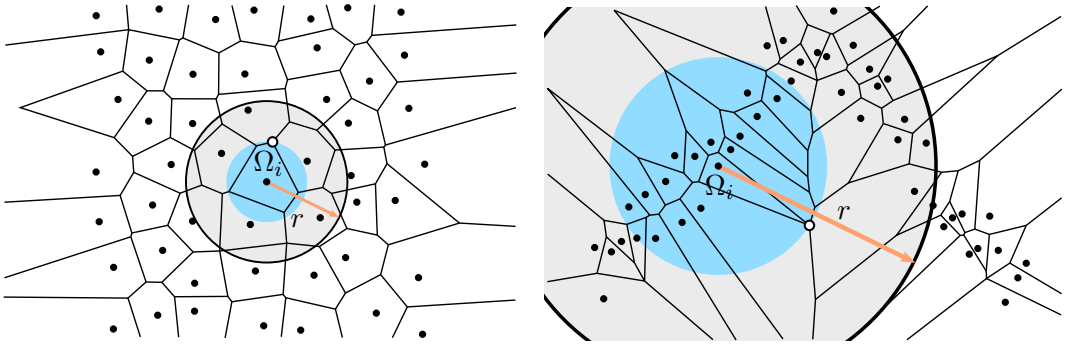


FIGURE 4.3 – À gauche une distribution homogène et son diagramme de Voronoï en deux dimensions, le rayon de sécurité, noté r est peu élevé, il faut donc peu de racines voisines (c'est-à-dire les racines contenues dans les deux cercles colorés) pour calculer les frontières de la cellule Ω_i. Dans le second cas, à droite, un cas symptomatique de notre algorithme, la distribution représente un nuage de points de toitures à deux pans. Nous pouvons voir qu'il faut un nombre plus élevé de racines voisines, dû au rayon de sécurité plus élevé, pour calculer les frontières de la cellule.

4.2 Approche proposée

L'idée est la suivante, plutôt que d'accélérer la génération des diagrammes de Voronoï, nous allons nous abstraire complètement de ceux-ci.

Dans la méthode présentée dans le chapitre 2, les diagrammes de Voronoï permettent de discrétiser notre surface pour estimer notre énergie, c'est-à-dire l'intégrale de la distance de la surface au nuage de points.

Nous choisissons de réaliser une approximation de cette intégrale en l'estimant à partir d'une discrétisation régulière de notre surface (voir sous-section 4.2.1). Cependant, cette énergie n'est pas dérivable, nous proposons alors de modifier la minimisation de l'énergie pour assurer localement et itérativement la dérivabilité de celle-ci (voir sous-section 4.2.2).

Cette énergie présente l'avantage de fournir des résultats aussi précis que l'énergie définie auparavant, et d'être bien plus rapide à estimer.

4.2.1 Approximation de l'énergie

Dans cette section, nous introduisons un nouveau terme d'erreur pour l'ajustement d'une surface triangulée à un nuage de points. Pour cela, nous nous inspirons des travaux du chapitre 2 en minimisant une approximation de l'énergie que nous avons définie auparavant. Nous commençons dans un premier temps par rappeler rapidement l'énergie précédemment présentée, puis nous proposons une approximation de celle-ci.

Énergie symétrique

Nous appelons S une surface à ajuster à un nuage de points P fixé. Le résultat de notre algorithme est une nouvelle géométrie de la surface S faisant correspondre S à P tout en respectant des contraintes géométriques sur les n sommets S_i, $i \in [\![0; n-1]\!]$ de S (par exemple, alignement ou orthogonalité de certaines arêtes de la surface).

Ces contraintes sont exprimées sous forme d'un nombre n_{eq} d'égalités telles que $\sum_{i=0}^{n} \alpha_i S_i = 0$ et n_{ineq} d'inégalités telles que $\sum_{i=0}^{n} \beta_i S_i > 0$. L'énergie tend vers zéro lorsque la surface est susceptible de correspondre au nuage de points et est strictement positive dans le cas contraire. Nous avons conçu une telle énergie que l'on nomme E et que nous soumettons à des contraintes.

Le problème s'exprime de la manière suivante :

$$\left| \begin{array}{l} \min\limits_{S} E(S) = \mathcal{F}(S) + \gamma\ \mathcal{G}(S) \\ \\ \text{sous les contraintes : } \begin{cases} A_{eq} S = 0 \\ \\ B_{ineq} S > 0 \end{cases} \end{array} \right. \tag{4.1}$$

où A_{eq} est une matrice d'égalités de taille $n_{eq} \times 3n$ et B_{ineq} est une matrice d'inégalités de taille $n_{ineq} \times 3n$. L'énergie est composée de deux termes symétriques $\mathcal{F}$ et $\mathcal{G}$ pondérés par un réel γ. Les termes $\mathcal{F}$ et $\mathcal{G}$ représentent respectivement la distance du nuage de points à la surface, ainsi que la distance de la surface au nuage de points.

Le **terme** $\mathcal{F}$ s'exprime comme la somme des distances au carré des points p du nuage à leur projeté sur la surface.

$$\mathcal{F}(S) := \sum_{p \in P} \Psi\left(||p - \pi_S(p)||^2\right) \tag{4.2}$$

où $\pi_S(\mathbf{p}) := \text{argmin}_{\mathbf{q} \in S} ||\mathbf{q} - \mathbf{p}||$ est la projection de $\mathbf{p}$ sur la surface, c'est-à-dire le point le plus proche sur S de $\mathbf{p}$ et Ψ permet de réguler le poids des points trop éloignés de la surface :

$$\Psi : x \mapsto \begin{cases} 1 - (1 - \frac{x^2}{\delta^2})^3 \text{ si } |x| < \delta \\ 1 \text{ sinon} \end{cases} \tag{4.3}$$

Le **second terme** $\mathcal{G}$ s'exprime comme la distance au carré de la surface S au point le plus proche dans le nuage de points.

$$\mathcal{G}(S) := \int_S ||q - \pi_P(q)||^2 dS(q) \tag{4.4}$$

où $\pi_P(\mathbf{q}) := \text{argmin}_{\mathbf{p} \in P} ||\mathbf{p} - \mathbf{q}||$ est la projection de $\mathbf{q}$ dans le nuage de points, c'est-à-dire le point le plus proche de $\mathbf{q}$ dans P. Il est à noter que l'ensemble des points de S pour lesquels le projeté $\pi_P(q)$ est constant forment une partition de la surface. Cette partition est le diagramme de Voronoï restreint du nuage de points à la surface. Désignons par Ω_p la cellule de Voronoï du point $p \in P$, le diagramme partitionne S en régions polygonales $\Omega_p \cap S$. L'intégrale définie dans l'équation 4.4 peut donc être décomposée comme suit :

$$\mathcal{G}(S) = \int_S ||q - \pi_P(q)||^2 dS(q) = \sum_{p \in P} \int_{\Omega_p \cap S} ||p - q||^2 dS(q) \tag{4.5}$$

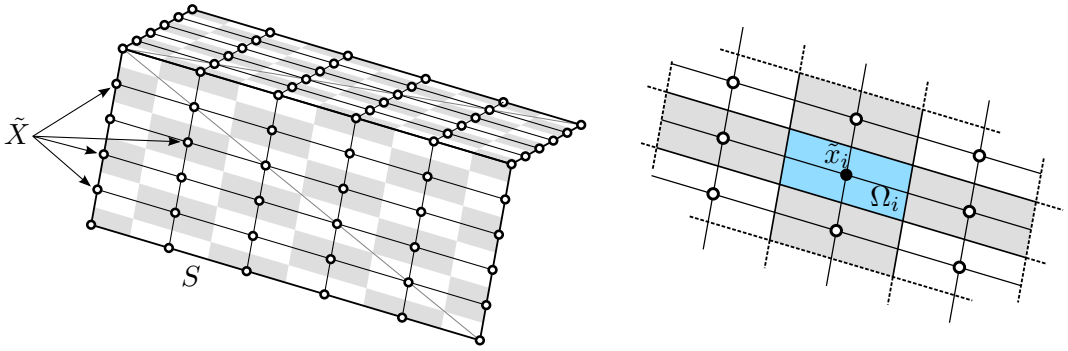


FIGURE 4.4 – À gauche : échantillonnage $\tilde{X}$ par grille régulière (6×6) de la surface S. À droite, un échantillon $\tilde{x}_i$ et sa cellule associée Ω_i.

.

L'intégrale sous forme close peut alors être calculée grâce au travail de [Nivoliers, 2012]. Ces termes ont chacun leur importance puisque l'un permet à la surface d'être attirée par le nuage de points et l'autre au nuage de points d'être attiré par la surface. Puisque ces termes sont définis sur des objets géométriques de nature différente, ils ont également une formulation différente.

L'estimation du terme $\mathcal{G}$ demande beaucoup de temps de calcul, nous proposons donc de remplacer le terme de l'équation 4.1 par une approximation de celui-ci.

Estimation de l'intégrale par subdivision

La méthode précédente présente les deux avantages suivants :

— Elle permet de déterminer pour chaque portion de la surface : le projeté $\pi_p(x)$ avec justesse, grâce à l'utilisation du diagramme de Voronoï du nuage de points restreint à la surface.

— Elle permet d'évaluer l'intégrale sous forme close en subdivisant les cellules de Voronoï restreintes en simplexes et en utilisant le théorème 2 proposé par Lasserre et Avrachenkov [Lasserre and Avrachenkov, 2001].

Notre objectif dans ce chapitre est d'améliorer drastiquement les performances de l'optimisation de l'énergie (d'au moins un ordre de grandeur) et pour cela, nous proposons de simplifier son expression.

Nous proposons d'effectuer une approximation de l'énergie en réalisant un échantillonnage $\tilde{X}$ de la surface. Pour chaque triangle t de la surface S, nous sélectionnons un sommet s_1, les vecteurs formés par ce sommet et les deux autres sommets s_2, s_3 du triangle forme une base. À partir de cette base, nous pouvons définir pour chaque triangle un échantillonnage comme une combinaison linéaire de ces deux vecteurs. Pour chaque triangle $t = (s_1, s_2, s_3)$, il est possible de définir un échantillon $\tilde{x}$ en trouvant trois réels α, β et γ tels que $\alpha + \beta + \gamma = 1$ et :

$$\tilde{x} = \alpha s_1 + \beta s_2 + \gamma s_3 \tag{4.6}$$

Les nombres α, β et γ sont les coordonnées barycentriques de $\tilde{x}$ par rapport aux trois sommets du triangle t qui le contient. Enfin, chaque échantillon contrôle une portion de la surface que l'on notera Ω_i (voir figure 4.4).

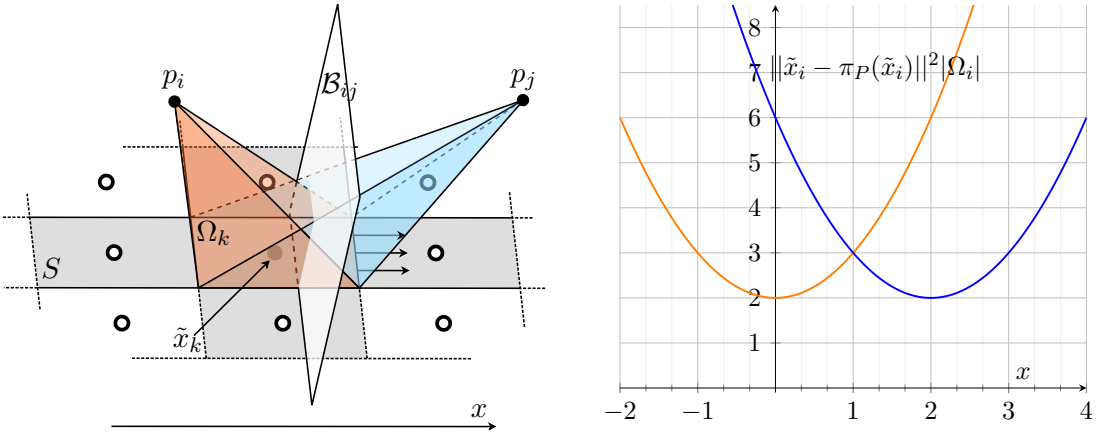


FIGURE 4.5 – À gauche : un échantillon $\tilde{x}_k$ et sa cellule associée Ω_k, le plus proche voisin de la cellule, p_i ou p_j varie en fonction du déplacement de la surface S sur l'axe x. Les points p_i et p_j définissent un plan médiateur $\mathcal{M}_{ij}$ si $\tilde{x}_k \in \mathcal{M}_{ij}$ le plus proche voisin de $\tilde{x}_k$ n'est pas unique. À droite : mesure de l'erreur de l'échantillon $\tilde{x}_k$ au point p_i en orange et à p_j en bleu, le minimum de ces deux fonctions est notre fonction d'énergie. La fonction n'est pas dérivable à l'intersection des deux courbes.

Nous obtenons un échantillonnage variant à chaque déformation de la surface. Il possède également la propriété de ne pas nécessiter de stockage en mémoire puisqu'il peut s'exprimer sous la forme de coordonnées barycentriques des sommets de chaque triangle. Enfin, cette solution est rapide et simple à implémenter.

L'**échantillonnage** de S permet de discrétiser l'intégrale sur S par la somme des distances au carré pondérée par l'aire de Ω_i, $|\Omega_i|$:

$$\begin{aligned} \mathcal{G}(S) &= \int_S ||q - \pi_P(q)||^2 dS(q) \\ &= \lim_{n \to +\infty} \sum_{i=0}^{n} ||\tilde{x}_i - \pi_P(\tilde{x}_i)||^2 |\Omega_i| \end{aligned} \tag{4.7}$$

L'approximation de $\mathcal{G}(S)$ est notée $\tilde{\mathcal{G}}(S)$:

$$\begin{aligned} \tilde{\mathcal{G}}(S) &\approx \mathcal{G}(S) \\ &= \sum_{\tilde{x}_i \in \tilde{X}} ||\tilde{x}_i - \pi_P(\tilde{x}_i)||^2 |\Omega_i| \end{aligned} \tag{4.8}$$

4.2.2 Continuité et dérivabilité

Comme exprimé précédemment, notre nouveau terme 4.8 utilise un échantillonnage de la surface $\tilde{X}$. La projection d'un échantillon vers le nuage de points s'exprime maintenant de la manière suivante :

$$\pi_P(\tilde{x}_i) = \underset{p \in P}{\operatorname{argmin}} ||\tilde{x}_i - p|| \tag{4.9}$$

En utilisant un tel échantillonnage de la surface, nous ne pouvons plus garantir l'injectivité de notre fonction de projection. En effet, il existe des configurations

dans lesquelles la projection de $\tilde{x}_i$ n'est pas unique. Soit p_i, p_j les deux points les plus proches de notre échantillon $\tilde{x}_i$ alors le projeté $\pi_P(\tilde{x}_i)$ n'est pas unique lorsque le plan médiateur $\mathcal{M}_{ij}$ défini par le couple de points p_i, p_j contient l'échantillon. Dans ce cas précis, la fonction n'est pas dérivable (voir figure 4.5). Cependant, il est à noter que la fonction 4.8 est de classe C^2 si aucun échantillon de $\tilde{X}$ n'appartient aux plans médiateurs définis par les points les plus proches des échantillons considérés. Nous pouvons le démontrer facilement en prouvant qu'au sein d'une cellule de Voronoï, l'énergie dépend uniquement de la racine de la cellule.

Pour proposer un algorithme robuste à tous types de configurations, nous choisissons de suivre un nouveau paradigme. Nous décidons de minimiser notre approximation par un processus itératif alternant deux étapes. Nous considérons les projetés $\pi_P(\tilde{x}_i)$ comme des constantes et minimisons notre énergie, puis nous estimons à nouveau des projetés.

Les étapes principales de l'algorithme sont les suivantes :

1. Spécifier une position initiale appropriée pour la surface à ajuster au nuage de points.
2. Estimer des échantillons pour la surface S. Pour chaque échantillon, associer un unique projeté constant, pour approximer localement notre fonction d'énergie.
3. Optimiser la position des différents sommets contraints de notre surface pour minimiser notre mesure d'erreur et obtenir une surface mise à jour.
4. Répéter les étapes 2 et 3 jusqu'à convergence, c'est-à-dire jusqu'à ce qu'un seuil d'erreur prédéfini soit atteint ou que la variation des sommets de notre surface tombe en dessous d'un seuil prédéfini.

Gradient de $\tilde{\mathcal{G}}$

Soit f une fonction scalaire et s_k un sommet de la surface S (k désignant un indice global de la surface), nous utilisons la notation suivante pour désigner la dérivée partielle de f par rapport au sommet s_k :

$$\frac{\partial f}{\partial s_k} = \left(\frac{\partial f}{\partial s_{k,x}}, \frac{\partial f}{\partial s_{k,y}}, \frac{\partial f}{\partial s_{k,z}} \right)$$

où $s_{k,x}, s_{k,y}, s_{k,z}$ sont les coordonnées du sommet s_k. Chaque dérivée partielle selon un sommet s_k forme une ligne de la matrice jacobienne de f. Considérons maintenant l'expression de la matrice jacobienne de la fonction $\tilde{\mathcal{G}}$ définie dans l'équation 4.8.

Le terme $\tilde{\mathcal{G}}$ peut s'exprimer comme une somme sur les triangles $t \in T$ de la surface S. Par construction, l'aire de chaque cellule associée à un échantillon peut s'exprimer comme une somme d'intersection $\Omega_{i|t}$ entre les cellules et les triangles de T :

$$\tilde{\mathcal{G}}(S) = \sum_{t \in T} \sum_{\tilde{x}_i \in t} ||\tilde{x}_i - \pi_P(\tilde{x}_i)||^2 |\Omega_{i|t}| \tag{4.10}$$

À partir de cette expression, le gradient de la fonction peut se scinder facilement en deux termes, la distance et la pondération par l'aire. Comme précédemment, la dérivée partielle de la distance s'exprime aisément grâce aux coordonnées barycentriques de l'échantillon : $\tilde{x}_i = \alpha_i s_k + \beta_i s_{k+1} + \gamma_i s_{k+2}$.

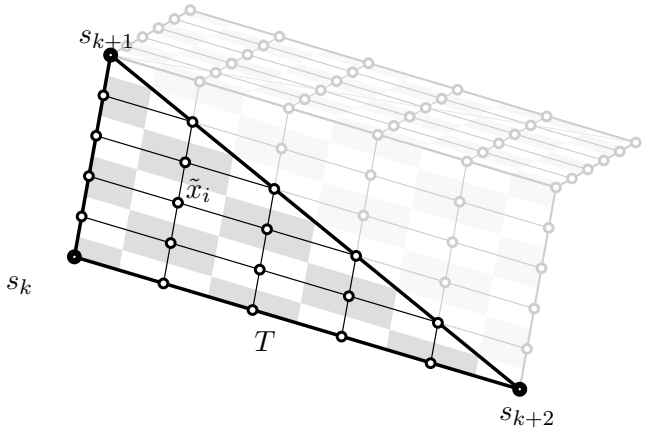


FIGURE 4.6 – Échantillonnage $\tilde{X}$ de la surface S, pour estimer le gradient de l'énergie nous considérons un triangle à la fois ; ici T et ses trois sommets (s_1, s_2, s_3).

Nous obtenons :

$$\frac{\partial ||\tilde{x}_i - \pi_P(\tilde{x}_i)||^2}{\partial s_k} = 2\alpha_i(\tilde{x}_i - \pi_P(\tilde{x}_i))$$

L'expression de la dérivée partielle de l'aire peut s'obtenir également facilement si nous considérons n_t le vecteur normal au triangle t composé de ses sommets $\{s_l, s_{l+1}, s_{l+2}\}$, indicés localement. L'aire de Ω_i s'exprime grâce à la norme de la normale n_t, telle que $||n_t|| = (s_{l+1} - s_l) \wedge (s_{l+2} - s_l)$, pondérée par un scalaire ω_i en fonction de la position de $\tilde{x}_i$.

Grâce aux propriétés de distributivité et d'anticommutativité du produit vectoriel nous obtienons pour la dérivée partielle de n_t :

$$\begin{cases} \frac{\partial n_t}{\partial s_l} = \mathbb{1} \wedge (s_{l+1} - s_{l+2}) \\ \frac{\partial n_t}{\partial s_{l+1}} = \mathbb{1} \wedge (s_{l+2} - s_l) \\ \frac{\partial n_t}{\partial s_{l+2}} = \mathbb{1} \wedge (s_l - s_{l+1}) \end{cases}$$

L'expression de la dérivée partielle peut alors être généralisée :

$$\frac{\partial n_t}{\partial s_l} = \mathbb{1} \wedge (s_{l+1} - s_{l+2})$$

Ici, nous considérons que les indices l sont évalués modulo 3. Nous obtenons l'expression de la dérivée partielle de $||n_t||$, telle que :

$$\frac{\partial ||n_t||}{\partial s_l} = \mathbb{1} \wedge (s_{l+1} - s_{l+2}) \times \frac{n_t}{||n_t||}$$

Enfin, il convient de pondérer l'aire $||n_t||$ pour obtenir la valeur réelle de Ω_i. Quatre types de configurations sont à distinguer :

1. $\tilde{x}_i$ est un sommet s_l, d'indice global minimum dans le triangle t, $\omega_i = 1/4$
2. $\tilde{x}_i$ est un sommet s_l, d'indice global non minimal dans le triangle t, $\omega_i = 1/8$
3. $\tilde{x}_i$ appartient à une arête du triangle T, $\omega_i = 1/2$
4. $\tilde{x}_i$ n'appartient à aucune arête du triangle T et n'est pas un sommet s_k, $\omega_i = 1$

Puis chaque coefficient ω_i est divisé par un coefficient d'échantillonnage $(N-1)^2$ avec N le nombre d'échantillons par arête de triangle. Nous obtenons enfin l'expression suivante pour le gradient de la fonction $\mathcal{G}(S)$:

$$\frac{\partial \tilde{\mathcal{G}}(S)}{\partial s_k} = \sum_{t \in \mathcal{T}(s_k), l \in [0,2]} \sum_{\tilde{x}_i \in t} \frac{\omega_i}{(N-1)^2} \left(2\alpha_k (\tilde{x}_i - \pi_P(\tilde{x}_i)) ||n_t|| + \mathbb{1} \wedge (s_{l+1} - s_{l+2}) \times \frac{n_t}{||n_t||} \right) \tag{4.11}$$

en définissant pour chaque s_k un voisinage de triangles t, $\mathcal{T}(s_k)$.

4.3 Comparaison des algorithmes d'appariement

Dans cette section, nous comparons les deux méthodes d'ajustement, la première estime exactement l'énergie grâce à un calcul de diagramme de Voronoï restreint (cette méthode est présentée dans le chapitre 2), la seconde approxime cette énergie en alternant plusieurs étapes d'optimisation en utilisant des projections constantes puis en sélectionnant de nouvelles projections.

Pour cela, nous utiliserons la base de données définie dans le chapitre 2, contenant des acquisitions LiDAR de la ville de Breuschwickersheim, en Alsace, en suivant une analyse similaire, consistant à tester les résultats de nos optimisations pour différentes initialisations et qualité de nuage de points.

Pour mesurer l'erreur de l'ajustement, nous utiliserons comme précédemment un modèle de référence, que nous avons ajusté manuellement au préalable (il s'agit du résultat de l'ajustement par l'algorithme du chapitre 2 à un nuage de points nettoyé manuellement) pour calculer la différence entre les deux modèles. Cette différence est obtenue grâce à la distance de Hausdorff entre ses sommets et les sommets de ce modèle de référence, nommé d.

Concernant les modèles de toits, nous nous basons sur la même bibliothèque (figure 2.9) disponible sous forme de maillage triangulé sur lequel nous définissons manuellement des contraintes géométriques. Enfin, nous discutons des performances, des avantages et des cas d'échec de la méthode.

Concernant l'équilibrage des deux énergies, nous choisissons de donner le même poids à chaque énergie pour traiter le maximum de modèles. Dans les faits, ajuster ce terme permet de traiter l'ensemble de la base de données.

4.3.1 Robustesse

En s'inspirant de la méthodologie du deuxième chapitre, nous commençons par tester la convergence de nos méthodes vers l'optimum en fonction de la qualité du nuage de points. Puisque, la nouvelle méthode est également conçue pour être robuste aux valeurs aberrantes, comme des nuages contenant de grandes quantités de points ne correspondant pas à la forme finale de la toiture, nous testons également sa robustesse aux nuages de points non segmentés. Enfin, nous estimons sa capacité à trouver l'optimum en fonction de son initialisation.

Qualité du nuage de points, impact sur les résultats

Acquisitions bruitées et densité du nuage de points Il est important de montrer que le bassin d'attraction du minimum de l'énergie correspondant à la position désirée est suffisamment large. Pour ce faire, nous vérifions que la position souhaitée est effectivement un minimum local en vérifiant que minimiser l'énergie depuis cette position n'a aucun effet. Ensuite, nous testons notre algorithme d'ajustement sur des nuages de points altérés. L'optimisation est réalisée sur les configurations suivantes : nuage de points bruité, décimation aléatoire de 50% et de 90% des points. Le bruit artificiel ajouté suit une distribution uniforme avec une moyenne nulle et une amplitude maximale de 50 cm.

Afin de vérifier l'impact de la qualité du nuage de points sur l'optimisation, notre surface est initialisée à son optimum. Pour rappel, la qualité des résultats est évaluée grâce à la distance de Hausdorff que l'on note d entre la surface optimum et le résultat de notre optimisation. Les différentes méthodes sont nommées suivant la nomenclature :

1. Appariement usant de diagramme de Voronoï restreint (voir chapitre 2) ;
2. Appariement alternant deux étapes (projection constante).

Les expériences ont été réalisées sur l'ensemble des nuages de points de notre base de données 4.3.1.

1. Toit deux pans, larges structures ;
2. Toit deux pans, sol ;
3. Toits mitoyens dissociés ;
4. Toits mitoyens concourent ;
5. Toit deux pans ;
6. Toit deux pans, petites structures ;
7. Toit deux pans, végétation et sol ;
8. Toit en L, larges structures ;
9. Toit en L, sol ;
10. Toit en L, végétation et sol ;
11. Toits en L, mitoyens dissociés ;
12. Toits en L, mitoyens concourent ;
13. Toit en L ;
14. Toit en L, petites structures.

Dans la majorité des cas, les résultats obtenus par l'algorithme à projections constantes produit des résultats similaires à la version présentée dans le chapitre 2 en figure 4.7. Dans le premier graphique, c'est-à-dire l'ajustement de la surface au nuage de points de référence, la quasi-totalité des surfaces restent à leur position optimale, exceptées pour les cas (c), (k) et (l), c'est-à-dire les toits mitoyens et contigus, qui est un cas d'erreur connu. En pratique, ces cas d'erreurs peuvent être facilement traités en augmentant le poids du terme d'énergie exprimé dans l' équation 4.10 pour l'algorithme à projections constantes ou dans l' équation 2.4 pour l'algorithme du chapitre 2.

L'ajout de 50 cm de bruit induit une plus grande distance entre le modèle de référence et le modèle résultant. Toutefois, une grande partie des résultats présentent un résidu inférieur à la valeur du bruit ajouté, ce qui est plutôt encourageant. Enfin, les algorithmes fournissent des résultats semblables pour les surfaces décimées si l'on considère qu'une distance inférieure à 50 cm est un succès.

Robustesse à l'initialisation

Il convient maintenant de vérifier la qualité des résultats en fonction de l'initialisation de la surface. Dans cette sous-section, nous procédons à des initialisations

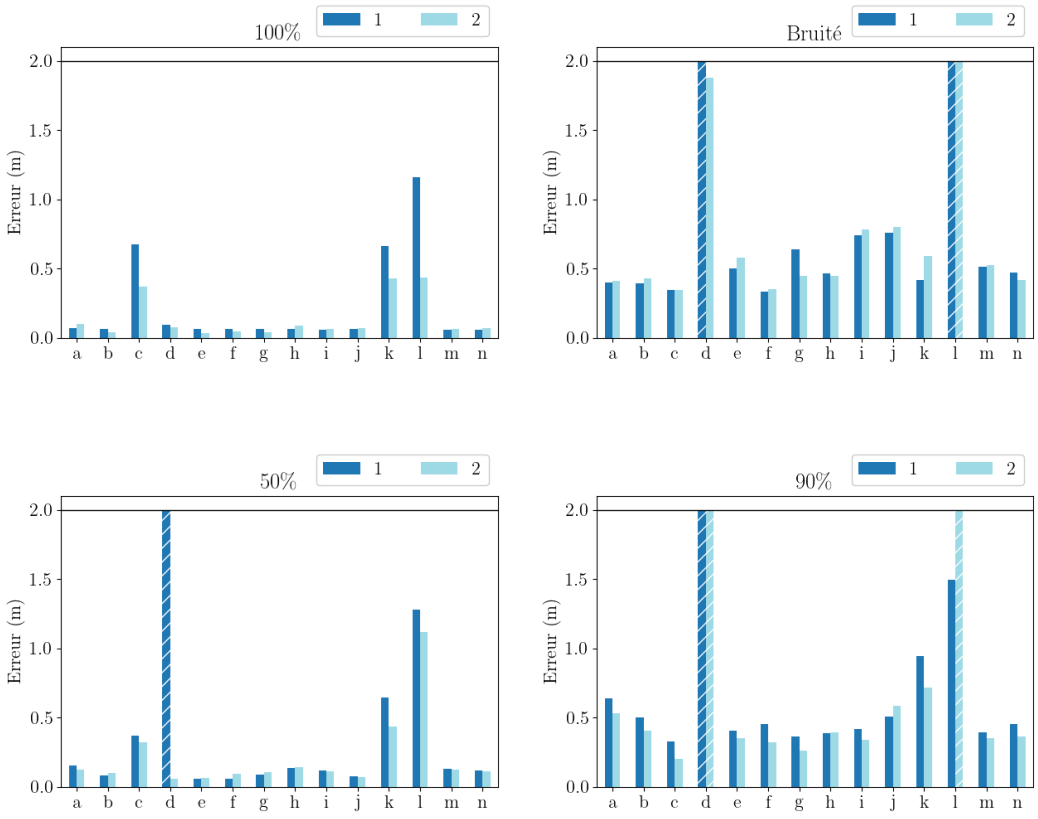


FIGURE 4.7 – Mesure de l'erreur sur l'ajustement d'une surface pour les deux différentes méthodes. En haut à gauche, l'optimisation est réalisée sur le nuage de points de référence, en haut à droite, sur un nuage bruité artificiellement. Enfin, les diagrammes du bas représentent l'ajustement à des nuages de points dont ont été retirés aléatoirement 90% à gauche et 50% à droite de points. Une erreur inférieure à 50 cm représente une surface considérée comme ajustée correctement. Les résultats supérieurs à 2 mètres (dont la barre est striée) ont été tronqués pour améliorer la lisibilité des figures.

translatées, tournées ou déformées afin de vérifier la convergence vers l'optimum.

Translation Comme réalisé dans le chapitre 2, nous initialisons le modèle de toit en appliquant une translation au modèle de référence. Nous testons douze directions différentes uniformément réparties sur une sphère. En pratique, les translations de faible amplitude ne posent aucun problème d'optimisation. Nous testons donc pour une translation de 3 mètres à la position optimale (voir figure 4.8). La figure 4.8 représente le nombre de translations donnant lieu à une optimisation réussie. Nous pouvons constater que ce nombre varie peu d'un algorithme à l'autre. Une nouvelle fois, les échecs d'optimisation concernent principalement les toits (c), (d), (k) et (l) et correspondent à des cas de toits mitoyens. Pour les échecs d'ajustement du cas (a), la surface s'est ajustée aux lucarnes présentes sur le toit dû à une initialisation

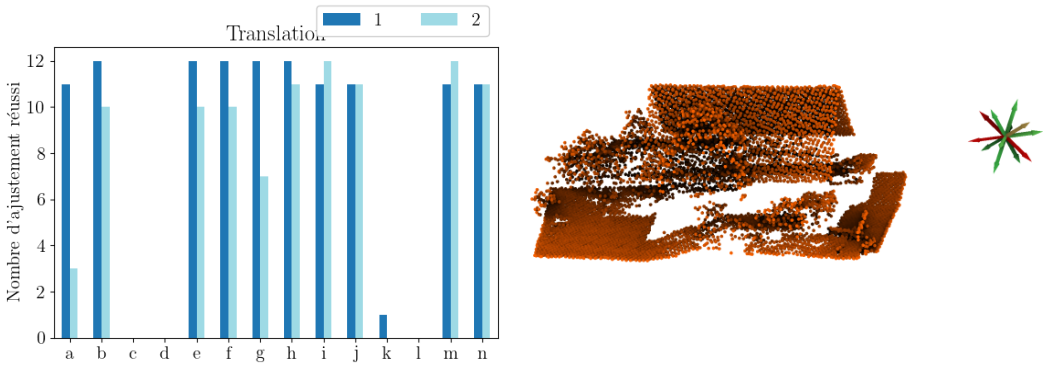


FIGURE 4.8 – À gauche : nombre de translations réussies (c'est-à-dire dont l'erreur est inférieure à 50 cm, chaque modèle peut avoir au plus 12 translations réussies). À droite : le nuage de points contenant un toit à deux pans et de la végétation (g), dont nous ajustons le modèle avec l'algorithme de projections constantes. Les flèches représentent le vecteur (à l'échelle) utilisé pour la translation de notre modèle à sa position d'initialisation. Les couleurs correspondent à la distance de Hausdorff calculée, 0 mètre étant entièrement vert et 2 mètres entièrement rouge.

peu favorable.

Rotation Pour la composante rotation, nous initialisons le modèle avec dix angles différents autour de l'axe Z, centré sur le barycentre du modèle de référence. La disposition des résultats est représentée sur la figure 4.9, où les positions initiales forment l'anneau intérieur des modèles, et les positions finales correspondent à l'anneau extérieur.

Nous observons que la méthode à projections constantes est mise en échec pour des cas simples, tel que l'ajustement a des toits deux pans (voir figure 4.9). Une des limites de l'algorithme apparaît lorsque le toit est initialisé orthogonalement à l'optimum, où l'optimisation conduit généralement à un minimum local.

Étirement Enfin, nous effectuons des tests en étirant le modèle de référence. L'étirement est appliqué par un facteur de [0.5, 1., 1.5] dans les directions x, y et z, conduisant à 27 cas différents, la surface est initialisée au barycentre du nuage de points. Nous pouvons constater dans la figure 4.10, que la méthode à projection constante obtient le plus grand nombre d'étirements réussi.

Résumé

En résumé, sur l'ensemble des 742 configurations testées, les méthodes ont fourni respectivement :

— 515 configurations de modèles ont convergé avec succès (distance de Hausdorff entre optimum et résultat inférieur à 0.2 mètre) pour la méthode utilisant un diagramme de Voronoï restreint.

— 507 configurations de modèles ont convergé avec succès pour la méthode des projections constantes.

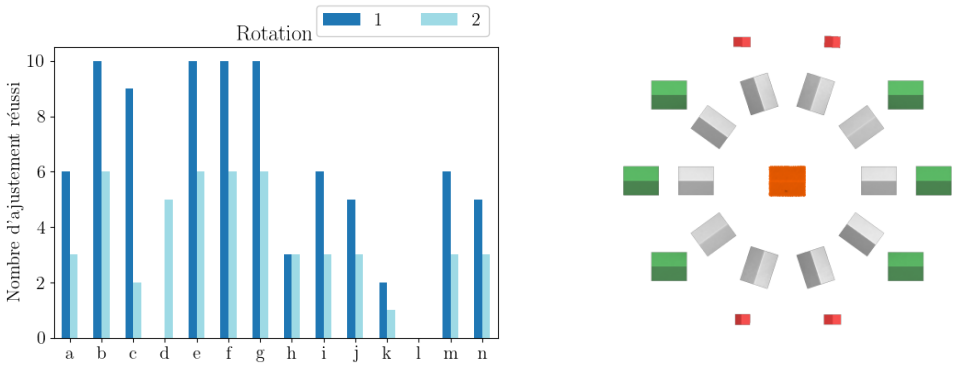


FIGURE 4.9 – À gauche : nombre de rotation réussie (dont l'erreur est inférieure à 50 cm, chaque modèle peut avoir au plus 10 rotations réussies). À droite : **Initialisation pivotée**, le modèle est initialisé à l'optimum puis tourné le long de l'axe Oz. Les positions initiales et finales du toit ont été décalées dans des cercles concentriques à des fins de visualisation. Du centre vers l'extérieur : le nuage de points, la position d'initialisation (en gris) et la position finale ajustée correspondante. Le code couleur varie de vert (distance 0 mètre) à rouge (distance 2 mètres).

Les deux méthodes présentent des résultats très similaires, mais la méthode à projections constantes semble plus sensible à l'initialisation en rotation et translation que l'algorithme proposé dans le chapitre 2.

4.3.2 Analyse temporelle

Dans cette sous-section, nous mesurons les temps d'exécution nécessaires à chaque algorithme pour attendre son critère de terminaison. Les algorithmes sont implémentés en C++, et utilise le diagramme de Voronoï restreint et le solveur L-BFGS présents dans la bibliothèque Geogram [Inria, 2018]. Les expériences ont été réalisées sur un processeur Intel(R) Xeon(R) E-2276M (cadencé à 2,8 GHz).

L'évaluation de notre méthode porte sur deux axes principaux : le temps d'exécution en fonction de la qualité du nuage de points et en fonction de la position initiale de la surface. Dans le premier cas, nous initialisons la surface à la même position que la surface de référence. Ici, nous pouvons constater que la méthode des projections constantes présente de meilleurs temps d'exécution. Cette méthode est un ordre de magnitude plus rapide que la méthode du chapitre 2 (voir figure 4.11).

Dans un second temps, nous avons évalué le temps d'exécution de l'optimisation en fonction de l'initialisation du modèle. Nous avons repris les initialisations en translation, rotation et étirement et nous avons réalisé une moyenne des temps d'exécution obtenus. Une seconde fois, la minimisation usant d'une approximation présente une accélération de un à deux degrés de magnitude comparée à l'algorithme présenté dans le chapitre 2 (voir figure 4.12).

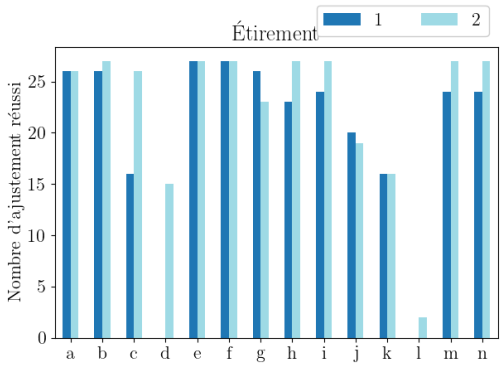


FIGURE 4.10 – Nombre d'étirements réussi (dont l'erreur est inférieure à 50 cm, chaque modèle peut avoir au plus 27 déformations réussies.

4.4 Conclusion

Dans ce travail, nous avons défini une méthode variationnelle permettant d'ajuster une surface à un nuage de points LiDAR. Cette méthode est issue de l'approximation de la méthode d'ajustement présenté dans le chapitre 2. En comparant cette méthode avec la méthode précédemment évoquée, nous avons conclu qu'elle offrait qualitativement des résultats similaires, dans un temps d'exécution bien inférieur. En termes de temps d'exécution, la méthode est plus rapide d'un à deux ordres de magnitudes. Toutefois, la méthode semble nécessiter une initialisation plus favorable que la méthode utilisant un diagramme de Voronoï. Cette méthode offre des résultats satisfaisants sur un panel varié de cas réels. Dans les zones plus denses, l'intégration de cette méthode dans un pipeline permettant de trouver une initialisation équitable et/ou appliquer un prétraitement de segmentation est loin d'être inutile.

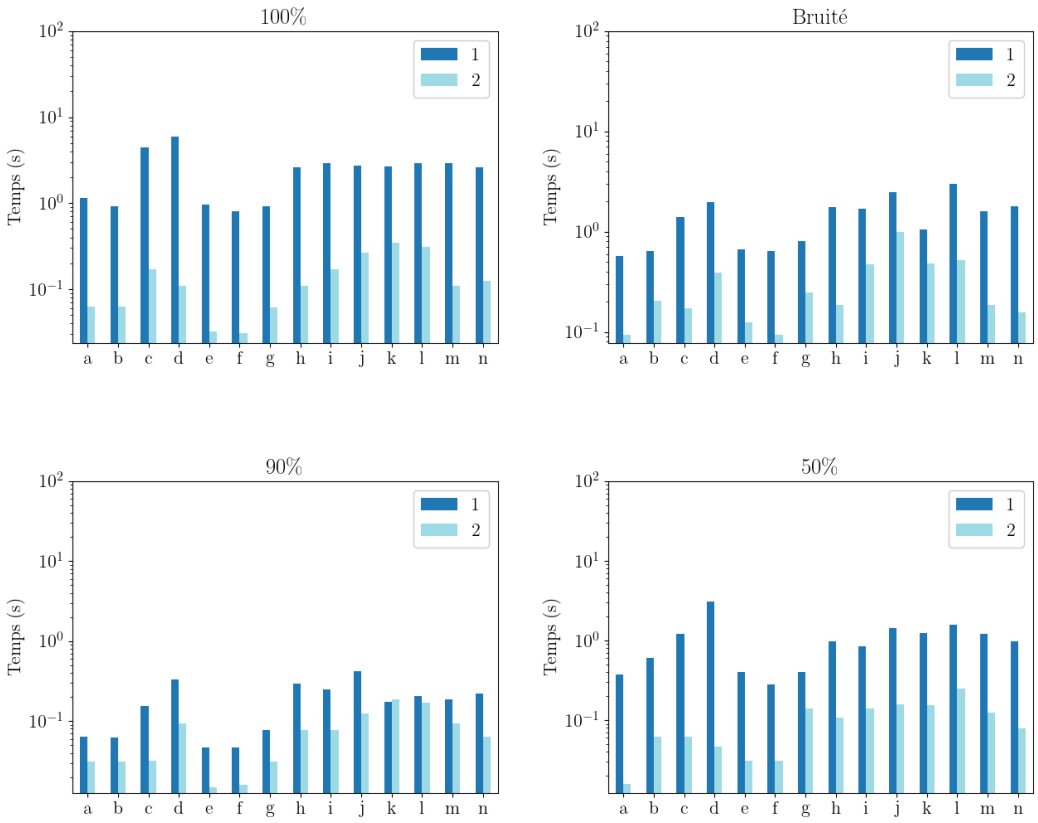


FIGURE 4.11 – Temps d'exécution moyen (échelle logarithmique) obtenus par les deux différentes méthodes. En haut à gauche, l'optimisation est réalisée sur le nuage de points de référence. En haut à droite, elle est réalisée sur un nuage bruité artificiellement, le bruit suit une distribution uniforme avec une moyenne nulle et une amplitude maximale de 50 cm. Enfin, les diagrammes du bas représentent l'ajustement à des nuages de points dont ont été retirés aléatoirement, 90% à gauche et 50% à droite, points.

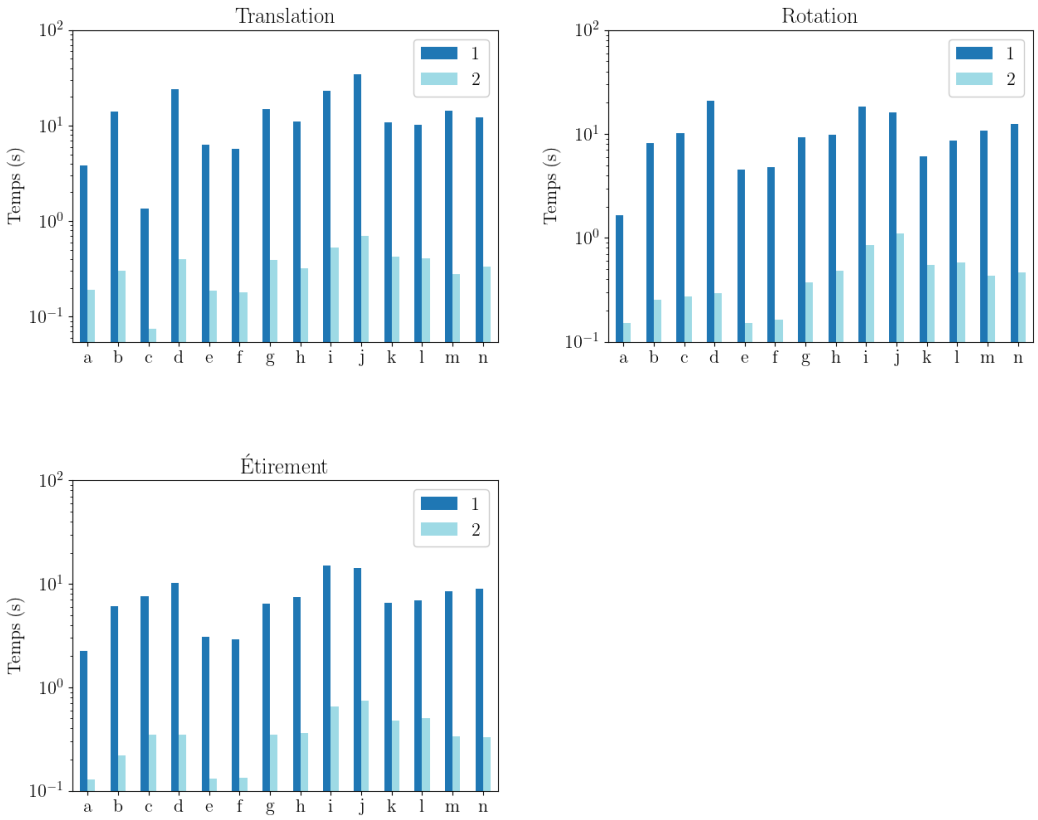


FIGURE 4.12 – Temps d'exécution (échelle logarithmique) obtenus pour les deux différentes méthodes. En haut à gauche, l'optimisation est réalisée avec une surface translatée de 3 mètres. En haut à droite, la surface est initialisée avec un angle variant entre 0 et 2π. Le diagramme du bas représente le temps d'exécution pour l'étirement de la surface dans les directions x, y et z.

Chapitre 5

Transfert industriel

Sommaire

Comme énoncé dans le premier chapitre introductif, la reconstruction de maillage approximant un nuage de points est un problème mal posé. Afin de stabiliser la solution du problème, un ensemble d'hypothèses réalistes est souvent imposé à la surface ou au nuage de points. Généralement, dans le cas de la reconstruction de toiture pour la génération de maquette numérique urbaine, nous supposons que la surface reconstruite est symétrique, linéaire par morceaux (c'est-à-dire composée de plans) ou encore correspondant à une primitive de forme connue. Concernant les nuages de points, des hypothèses sont posées sur leur densité d'échantillonnage, mais également sur le bruit contenu dans l'ensemble de points.

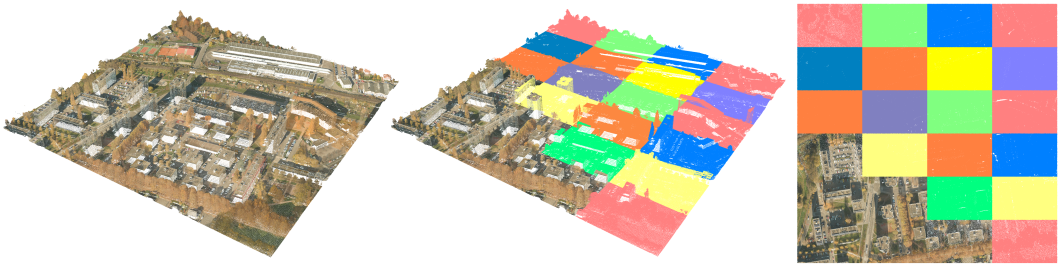
La prise en compte de ces hypothèses permet souvent de guider la reconstruction et donc de la faciliter. Toutefois, cela est souvent insuffisant, un utilisateur doit alors être impliqué afin d'apporter des indices pour faciliter la reconstruction. Dans ce chapitre, nous proposons une interface homme-machine ainsi que des outils logiciels, pour simplifier le travail des modeleurs et rendre, le problème mal posé de la reconstruction de surface à partir d'un nuage de points, traitable tout en contrôlant plus facilement son résultat. Pour cela, nous proposons de porter nos travaux de recherche au sein du logiciel Rhinocéros, en particulier l'ajustement d'une surface à un nuage de points, pour permettre aux utilisateurs de reconstruire plus facilement des toitures à partir de nuage de points. Toutefois, pour que ces travaux soient exploitables, il est nécessaire de mettre en place différents outils pour faciliter l'interaction des utilisateurs avec des nuages de points surdimensionnés, denses et non structurés.

Nous débutons par mettre en lumière les structures de données mises en place au sein du logiciel Rhinocéros pour afficher et interagir en temps réel avec les nuages de points §5.1. Puis, nous proposons un pipeline permettant aux modeleurs de reconstruire facilement les modèles de toitures §5.2.1.

5.1 Manipulation rapide des nuages de points

Comme expliqué au début de ce manuscrit, le développement d'outils d'acquisitions tels que le balayage laser (mobile, terrestre, aérien) ou la photogrammétrie a permis la production de nuages de points de plus en plus précis. En contrepartie, plusieurs millions à plusieurs milliards de points sont générés et nécessitent de grandes quantités de mémoire pour être stockés ou traités. Pour illustrer notre propos, imaginons que nous souhaitons réaliser l'acquisition d'une parcelle de 500 mètres de côtés, à cet effet nous choisissons une densité de 20 pts/m^2. Nous obtenons un nuage de points contenant environ 5 millions de points, représentés par des coordonnées x, y, z (`float`), des attributs : l'intensité (`unsigned short`), le numéro de retour (`3 bits`), le nombre de retour (`3 bits`), l'indice de direction de numérisation (`1 bit`), l'indice de ligne de fin de numérisation (`1 bit`), la classification (`unsigned char`), l'angle de numérisation (`char`), les données utilisateurs (`unsigned char`), l'identifiant d'origine du point (`unsigned short`), le temps GPS (`8 bytes`) et les couleurs (3 $\times$ `unsigned short`). Un rapide calcul montre que le stockage de cette dalle de nuage de points nécessite environ 170 Mo de mémoire.

Alors que le chargement d'une telle dalle, que nous pouvons considérer comme standard est bien maitrisé, l'importation massive de plusieurs dalles, ou d'une unique dalle de taille exceptionnelle, implique généralement un dépassement des capacités mémoire des machines de nos utilisateurs. Pour illustrer notre propos, nous pouvons



(a) (b) (c)

FIGURE 5.1 – De gauche à droite, (a) le nuage de points de référence, en vue perspective représentant la banlieue strasbourgeoise ; (b) nuage de points dont certaines dalles issues de la division sont représentées par des couleurs aléatoires ; (c) vue de haut du maillage de points.

citer le nuage de points de la ville de Nancy, stocké dans une unique dalle qui mesure un peu moins de 13 Go [Nancy, 2022] qui provoque en l'état l'interruption du logiciel Rhinocéros lors de son importation.

Pour travailler sur ces données parfois trop volumineuses, il est nécessaire de les traiter indépendamment sur des ensembles de données plus petits et assimilables par la machine d'un utilisateur. Plus formellement, Aggarwal et Vitter [Aggarwal and Vitter, 1988] définissent dans *External-Memory Model* une machine abstraite dotée de deux niveaux :

— le cache, où les opérations de lecture et d'écriture sont rapides, mais dont l'espace est limité ;
— le disque, où les opérations sont plus coûteuses, car plus lentes, mais dont l'espace est considéré comme pratiquement illimité.

Ce modèle exploite les assertions suivantes : les opérations de lecture et d'écriture sont beaucoup plus rapides sur le cache que sur le disque et la lecture de longs blocs contigus de mémoire est plus rapide qu'une lecture aléatoire sur le disque. Pour cela, le disque est partitionné en blocs de taille B pouvant être transférés en une seule fois, tandis que le cache de taille M peut stocker jusqu'à M/B blocs, avec $M \geq B$. Une opération d'entrée/sortie ou de transfert de mémoire consiste à déplacer un bloc de B éléments de la mémoire du disque sur le cache, et le temps d'exécution d'un algorithme est déterminé par le nombre de ces opérations d'entrée/sortie.

Aujourd'hui ce modèle demeure pertinent pour traiter des masses de données, même avec l'évolution des espaces de stockages plus performants tels que les SSD. Dans le cas des nuages de points, nous souhaitons partitionner nos données localement, d'une part pour que celles-ci puissent tenir en mémoire et d'autre part pour pouvoir interagir en temps réel avec les nuages de points.

Tout d'abord, nous souhaitons stocker les nuages sous forme de dalles dans des fichiers sur le disque. Ces données sont écrites dans des fichiers au format binaire, comme les formats standardisés LAS/LAZ. Les fichiers enregistrent uniquement les coordonnées des points et leurs différents attributs, ils permettent donc de résoudre le problème lié du stockage mémoire, mais ne facilite pas la réalisation de requête d'accès dans le nuage de points.

Contrairement aux images qui sont stockées de manières structurées et dont le

voisinage peut être déterminé simplement en interrogeant les pixels avoisinants, les nuages de points sont dépourvus de toute structure. Deux points pouvant être très proches spatialement peuvent avoir des indices très différents dans le vecteur les stockant. En raison de ce manque de structure, les opérations d'accès sur un nuage de points, et les traitements nécessitant de connaitre le voisinage d'un point peuvent-être chronophage. La manipulation de ces masses de données nécessite des structures permettant l'indexation spatiale, la détection efficace de collision ou encore l'élimination du rendu des objets non contenu dans le cône de vision. Pour cela, les nuages de points sont fréquemment partitionnés en sous-ensembles de données d'une taille plus raisonnable. Parmi ces structures nous comptons les octree, les quadtree, les kdtree, ou encore les R-tree. À notre connaissance, le logiciel Rhinocéros (versions 5, 6 et 7) ne fournit pas de structure de données pour le traitement efficace de nuage de points.

Pour répondre à ces problématiques, nous proposons de traiter les nuages de points en deux étapes :

1. **Division en sous-dalles** : le nuage de points est découpé en multiples dalles que l'on stocke sur le disque.
2. **Partition** : chaque dalle obtenue est structurée grâce à un octree.

5.1.1 Division en sous-dalles

La première phase dite de **division** prend en entrée un nuage de points ainsi que deux entiers définissant le nombre de colonnes et de lignes, définissant une grille qui servira à découper le nuage, et optionnellement une courbe rectangulaire représentant la boite englobante du nuage de points. Une illustration de la division est présentée dans la figure 5.1. Le résultat obtenu est un ensemble de fichiers stockés sur le disque contenant chacun un nuage de points dont les dimensions sont constantes, ainsi qu'un fichier au format xml recensant le nom des fichiers des nuages de points obtenus ainsi que leurs boites englobantes.

Les paramètres décrivant le nombre de colonnes et de lignes doivent être choisis avec soin de manière à obtenir des dalles suffisamment petites pour tenir en mémoire. Concernant les nuages de points avec une densité très variable, nous conseillons, s'il reste des dalles nécessitant une quantité de mémoire supérieure à la capacité de l'ordinateur, de reproduire l'opération de division récursivement jusqu'à obtenir le résultat souhaité. Il s'agit d'un choix technique permettant à l'utilisateur de s'adapter à la grande variabilité de densité des nuages de points obtenus par relevé LiDAR.

À partir de cette étape, l'utilisateur peut charger une partie des dalles en fonction de la quantité de mémoire disponible sur sa machine.

5.1.2 Partition en Octree

Bien que le quadtree semble être la structure de données la plus adaptée pour le traitement de nuage de points LiDAR aéroporté, la société RhinoTerrain souhaite une structure générale qui permet de prendre en compte tout type de nuages de points (y compris des nuages issus de relevés terrestres ou mobiles), nous avons implémenté un octree afin de stocker le nuage de points brut, car il s'agit de la structure de données la plus généraliste. Cette structure, très efficace, permet d'accéder, d'insérer et de supprimer rapidement des points avec une complexité dans le pire cas de $O(\log(n))$,

où n est le nombre de points du nuage. De plus, cette structure ne requiert pas une grande quantité de mémoire pour être stockée.

Un octree correspond à la partition récursive d'un volume d'espace cubique (voir figure 5.2). À partir d'un cube initial, ici la boite englobante du nuage de points, les cellules de l'octree sont formées en divisant le cube en 8 sous-cubes de même taille. Ces mêmes sous-cubes sont subdivisés à leur tour jusqu'à atteindre un critère de terminaison. Dans notre cas, nous avons choisi d'arrêter la subdivision lorsque les sous-cubes contiennent moins de 10000 points. Ce critère est modulable par l'utilisateur.

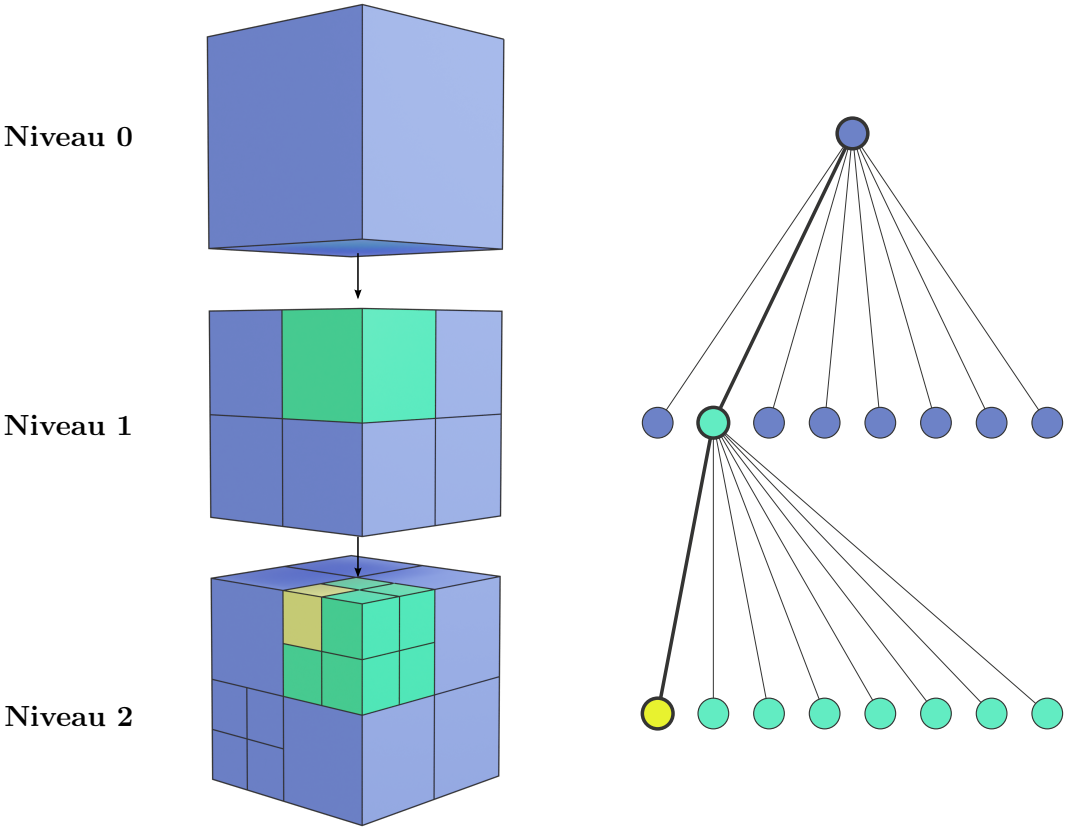


FIGURE 5.2 – Illustration d'un octree possédant 3 niveaux, le niveau 0 correspond à la boite englobante du nuage de point. Puis cette boite est découpée en 8 sous-boites, contenues dans le nœud parent sous forme de pointeur. Puis chaque nœud est à nouveau découpé en 8 nouveaux nœuds (voir niveau 2). La structure de données s'apparente à un arbre et permet un parcourt efficace de l'ensemble des points.

Nous venons de décrire les moyens mis en place pour stocker et accéder aux nuages de points plus facilement. Maintenant, nous décrivons dans la section qui

suit, les différentes méthodes que nous avons développées pour la reconstruction de maquette numérique urbaine.

5.2 Le plugin RhinoPointCloud

Nous proposons un plugin, nommé RhinoPointCloud, dont l'objectif est d'aider les professionnels de la création de modèles numériques urbains à rapidement modéliser un MNU à partir de nuage de points LiDAR aéroporté. Pour cela, nous proposons une suite d'outils permettant la segmentation des nuages de points et leur modélisation, variant de l'outil de reconstruction automatique à l'outil de reconstruction manuel qui permet de traiter les cas les plus complexes.

5.2.1 Aperçu du logiciel

Afin de permettre aux utilisateurs de produire les meilleures maquettes numériques, nous leurs proposons les méthodes suivantes, que l'on découpe en différentes étapes :

- **Segmentation**
  - Segmentation à partir de la classification ;
  - Segmentation à partir des cadastres ;
  - Segmentation à partir d'estimation des normales des points ;
  - Segmentation guidée par l'utilisateur ;

- **Choix des pans à reconstruire**
  - Guidé par l'utilisateur ;
  - Identification par clusterisation et graphe ;

- **Production du modèle 3D de toiture**
  - Ajustement par minimisation d'énergie ;
  - Création manuelle de la surface ;

Nous présentons les méthodes ci-dessus dans la suite de ce chapitre. Pour cela, nous considérons que nous avons à notre disposition un nuage de points d'une dalle de la banlieue de Strasbourg [Strasbourg, 2016], et nous analysons les différents cas de figure dans lequel l'utilisateur peut se trouver. Une partie des algorithmes ci-dessous utilise la partition en octree présenté précédemment, afin d'offrir des temps d'exécution plus courts à l'utilisateur. La prochaine section porte sur l'étape de segmentation qui est primordiale à tout traitement algorithmique.

5.2.2 Segmentation

Dans cette première section portant sur la segmentation, nous allons voir comment à partir d'un nuage de points brut, c'est-à-dire une dalle LiDAR de 500 mètres par 500 mètres, nous pouvons extraire des nuages de points de toiture. Pour cela, nous proposons un ensemble de méthodes de segmentation en fonction des données fournies par l'utilisateur.

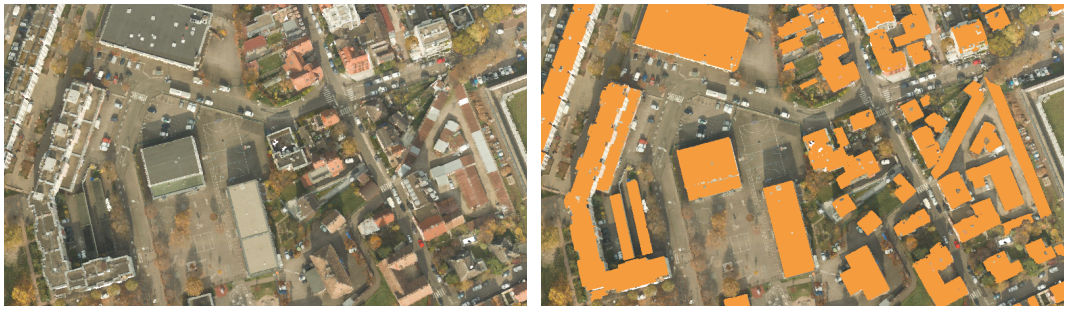


FIGURE 5.3 – Extraction des points correspondants à la classe "bâtiments" de la classification LAS. À gauche le nuage de points initial et à droite (en orange) le nuage de points contenant l'ensemble des points appartenant à des bâtiments au sens de la classification LAS.

Segmentation à partir de la classification

La première méthode est la plus simpliste, mais sans aucun doute l'une des plus efficaces, lors de l'import d'un fichier LAS, nous extrayons et stockons l'intégralité des attributs LiDAR, y compris la classification. Donc, si l'utilisateur importe un nuage de points doté d'une classification et que celle-ci contient la classe bâtiments telle que définie par le format Las/Laz, nous proposons une méthode permettant d'extraire les points de bâtiments (voir figure 5.3). Cette méthode est généraliste et permet d'extraire tout type de classe contenue dans la classification. Si les données ont été renseignées dans une nouvelle classe générée par l'utilisateur au sens LAS, celui-ci peut tout de même les récupérer.

Nous obtenons un nuage de points contenant l'intégralité des bâtiments de la dalle, il convient d'extraire les bâtiments individuels grâce à l'utilisation d'une clusterisation par distance euclidienne. Nous proposons une méthode à l'utilisateur pour réaliser une telle clusterisation avec un paramètre réglable, dont la valeur est fixée par défaut à deux fois la densité moyenne du nuage de points. Pour cela, la densité du nuage de points est estimée pour chaque point d_i, en se basant sur son voisinage local défini par une boule de rayon R centrée sur le point d'intérêt p_i, tel que :

$$d_i = \frac{1}{k} \sum_{j=0, j \neq i}^{k} ||p_i - p_j|| \tag{5.1}$$

avec k suffisamment grand pour atteindre les points contenus dans le rayon R. Il est parfois possible que certains points apparaissent en double dans le nuage de points, par conséquent, nous ignorons tous les points contenus dans la boule de rayon 10^{-5} mètre centrée sur le point considéré. La densité moyenne du nuage de points peut être calculée en réalisant la moyenne arithmétique de toutes les densités locales.

Cette méthode nous permet d'obtenir des segments de nuage de points contenant une à plusieurs toitures (si les bâtiments sont mitoyens).

Segmentation à partir des données cadastrales

Si l'utilisateur ne dispose pas d'un nuage de points classifié, mais possède des données cadastrales, nous proposons d'extraire les points contenus dans les courbes

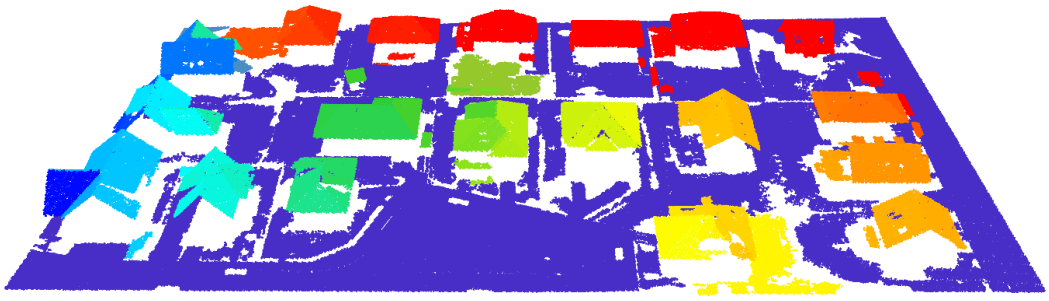


FIGURE 5.4 – Segmentation automatique du nuage de points, ici chaque couleur représente l'identifiant d'un nuage de points et chaque nuage de points représente une zone planaire.

cadastrales. Pour extraire les points, nous les projetons dans le plan (x,y), puis nous utilisons un algorithme de recherche d'inclusion de points dans un polygone 2D. Pour chaque point du nuage, nous lançons un rayon horizontalement dans la direction Ox, puis nous comptons le nombre d'arêtes qu'il intersecte. À chaque croisement, le rayon passe de l'intérieur à l'extérieur et vice-versa (théorème de Jordan [8]). Cela nous permet de déterminer si le point se trouve à l'intérieur ou à l'extérieur de la polyligne. Enfin, pour réduire le temps d'exécution de cet algorithme, nous utilisons l'octree défini précédemment, pour identifier les sous-cubes intersectant la polyligne et ainsi, réaliser le test d'inclusion sur un ensemble réduit de points.

Comme les polygones représentant les cadastres peuvent ne pas être exacts ou capturent l'empreinte au sol des bâtiments, ils peuvent ne pas contenir tous les points de la toiture. Étant donné que l'absence de ces points est bien plus préjudiciable à notre algorithme de reconstruction que de récupérer des points appartenant à une autre classe, nous choisissons de dilater les courbes cadastrales d'un mètre, afin de nous assurer que nous obtenons tous les points manquants.

Segmentation automatique

Si l'utilisateur possède un nuage de points sans attributs et ne fournit pas de données supplémentaires, nous segmentons automatiquement les zones planaires. Pour chaque point du nuage, nous calculons sa normale grâce à la transformée de Hough Randomisé (Randomized Hough Transform - RHT), dont l'algorithme est décrit par Dorit Borrmann et al. [Borrmann et al., 2011]. Nous choisissons cette méthode, car elle permet d'estimer rapidement et de manière robuste les normales du nuage de points. De plus, elle ne lisse pas les normales dans les zones à forte courbure, telles que les faîtages des toitures.

Puis, nous appliquons un algorithme de croissance de région en comparant pour chaque point sa normale à celle de ces voisins, si cette valeur est moins élevée qu'un certain seuil prédéfini γ alors le point est ajouté au nuage de points. Le voisinage contient tous les points à une distance R du point considéré. Comme précédemment, la valeur de R est fixée à deux fois la densité moyenne du nuage de points. Cette

8. Le complémentaire d'une courbe de Jordan dans un plan affine réel est formé d'exactement deux composantes connexes distinctes, l'une bornée et l'autre non. Toutes deux ont pour frontière la courbe de Jordan.

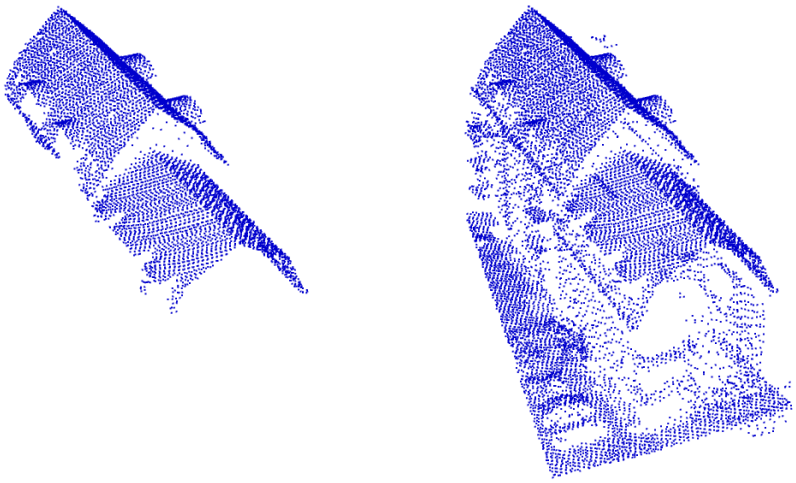


FIGURE 5.5 – Exemple d'omission dans la classification LAS/LAZ. Le nuage de points de gauche issus de la classe bâtiment de la classification LAS ne capture pas l'ensemble des points du toit, il manque les pans correspondants aux coyaux de la toiture. Nous proposons des outils manuels permettant à l'utilisateur de récupérer les données omises, dont le nuage de points résultant est présenté à droite.

méthode permet d'obtenir un nuage de points par pan de toiture. Une illustration du résultat est proposée dans la figure 5.4.

Segmentation manuelle

Enfin, nous proposons également à l'utilisateur de tracer des polylignes à la main lui permettant d'extraire les différents points présents dans cette même courbe. Pour cela, nous utilisons le même algorithme d'inclusion de points dans un polygone 2D que nous avons utilisé pour la segmentation à partir de cadastre. Cette méthode est particulièrement utile lorsque les méthodes présentées ci-dessus sont mises en échec (voir la figure 5.5).

5.2.3 Choix des pans à reconstruire et choix du modèle

À ce stade, l'utilisateur dispose de nuages de points représentant soit un pan de toit, soit un toit ou encore un ensemble de toits mitoyens. Nous proposons à l'utilisateur de sélectionner un ensemble de nuages de points à traiter, pour lui présenter tour à tour, en réalisant un zoom sur ceux-ci, afin de faciliter le traitement de l'ensemble des toitures.

L'utilisateur peut alors sélectionner un a plusieurs nuages de points (parmi les nuages avoisinants) afin de débuter la reconstruction. Nous proposons cela, pour permettre à l'utilisateur de sélectionner un ensemble de pans de toits. Cela est particulièrement utile, lorsqu'un nuage de points de toiture est réparti sur plusieurs dalles, ou lorsque la segmentation offre plusieurs pans de toits.

Pour chaque nuage de points, nous proposons à l'utilisateur de reconstruire un maillage de toiture à partir de la base de données présentée ci-dessous.

Base de données proposée

L'utilisateur dispose d'une base de données composée de 18 modèles de toitures prédéfinis, comme le montrent la figure 5.6 et la figure 5.7. La base de données est composée de toits divers, allant des toitures les plus communes présentées en figure 5.6, tels que les toits-terrasses, les toits à deux ou quatre pans, ainsi que des toitures plus rares présentées en figure 5.7. Chacun de ces modèles est contraint à respecter un ensemble de contraintes géométriques permettant d'obtenir un modèle conforme aux attentes de l'entreprise RhinoTerrain (planarité des faces, perpendicularité ou circularité de certaines arêtes). Pour rappel, la manière dont est contraint un modèle est décrite dans le chapitre 2 en sous-section 2.2.4.

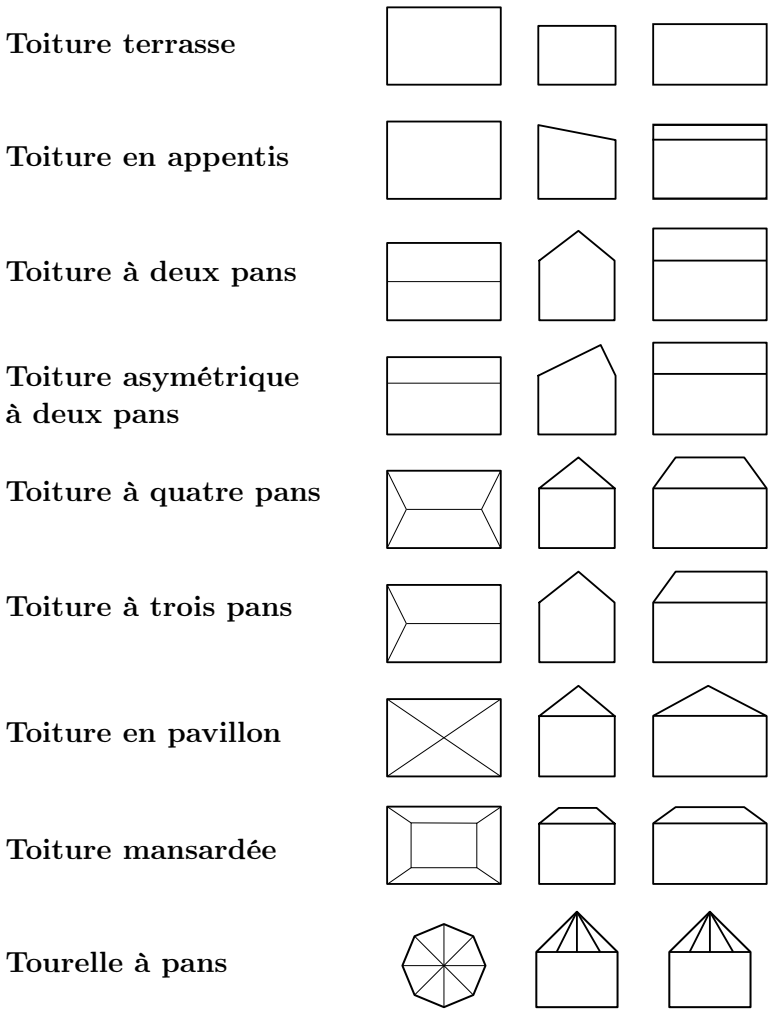


FIGURE 5.6 – Première partie de la bibliothèque de modèles des toitures proposées. À gauche est présentée une vue du haut de la toiture, au centre une vue de face et à droite une vue de côté.

Il est maintenant nécessaire de choisir un modèle, pour cela nous proposons deux méthodes de sélection.

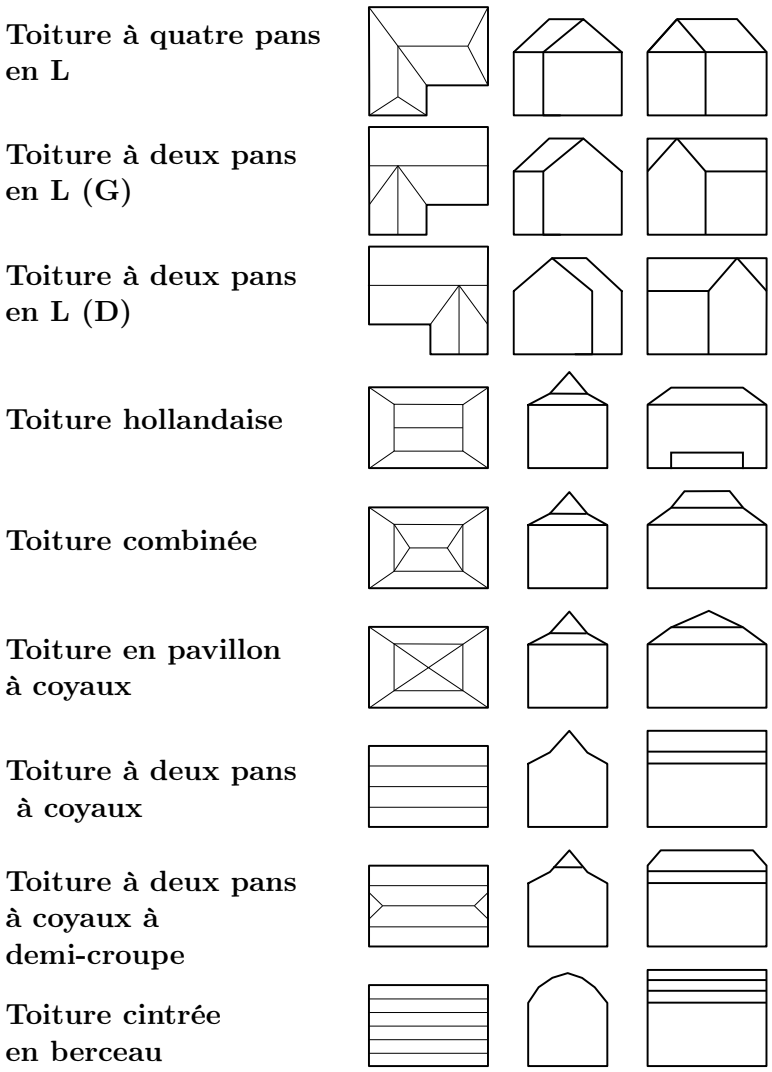


FIGURE 5.7 – Seconde partie de la bibliothèque de modèles des toitures proposées. À gauche est présentée une vue du haut de la toiture, au centre une vue de face et à droite une vue de côté.

Choix automatique ou manuel d'un modèle

La première méthode consiste à proposer automatiquement un modèle de toiture à l'utilisateur. Pour cela, nous cherchons un isomorphisme de graphes. Un isomorphisme de graphes est une bijection entre les sommets de deux graphes qui préserve les arêtes. Plus simplement, si nous considérons deux graphes non orientés, cela consiste à détecter si deux graphes sont exactement identiques en considérant l'indice des sommets du graphe comme muet.

Pour cela, chaque modèle présenté dans la base de données est doté de son propre graphe, dont les sommets représentent les faces du maillage et les arêtes représentent la connectivité de ces faces. Plus formellement, ce graphe est le dual de notre maillage.

Nous considérons l'union des nuages de points sélectionnés. Les pans de ce nouveau nuage sont extraits grâce à une croissance de région sur les normales du nuage

de points. Chaque pan du nuage de points représente un sommet de notre graphe. Les points du nuage sont alors labélisés par l'identifiant du sommet du graphe auxquels ils appartiennent. Enfin, pour chaque point du nuage, nous réalisons une recherche de ses k plus proches voisins, avec k faible (k=10). Si parmi ses voisins se trouve un point dont l'identifiant j est différent de celui i du point considéré, alors nous créons une arête dans le graphe entre les sommets i et j. En pratique, ce graphe est le dual du graphe topologique présenté en figure 1.2-G.

Nous obtenons deux graphes que nous pouvons comparer dans le but de proposer un modèle de toit par défaut à l'utilisateur pour la reconstruction.

Si le modèle de toit sélectionné est le bon alors, l'utilisateur valide le modèle, sinon l'utilisateur peut choisir par lui-même une toiture plus adéquate, parmi notre bibliothèque de modèles.

Si le nuage de points ne correspond à aucune toiture de la bibliothèque, l'utilisateur peut retourner à l'étape précédente de manière à réaliser une nouvelle segmentation des toitures (par exemple, si le nuage de points représente des toitures mitoyennes).

5.2.4 Production de modèles 3D

À cette étape, l'utilisateur a sélectionné un nuage de points et un modèle de notre base de données de toiture. Nous proposons alors d'ajuster ce modèle au nuage de points sélectionné. Pour cela, nous utilisons la méthode d'ajustement présenté dans le chapitre précédent.

Par défaut, les paramètres de l'ajustement sont réglés de manière à ce que la surface soit aux mieux recouvertes par le nuage de points. Toutefois, il existe des cas où l'utilisateur souhaite que la surface s'ajuste plus au nuage de points, quitte à ce que la surface ne soit pas entièrement recouverte, dans le cas par exemple des nuages de points lacunaires. Nous proposons donc à l'utilisateur de régler lui-même ce paramètre au cours de l'ajustement et nous lui présentons le résultat obtenu pour qu'il puisse le valider.

L'ensemble des modèles de toits obtenus peuvent être utilisés au sein des logiciels de la suite RhinoTerrain, permettant de réaliser les opérations booléennes nécessaires à la production de telles maquettes.

Nous présentons maintenant un ensemble de méthodes développées pour traiter les cas de toitures non usuelles, ou plutôt non compris dans la base de données.

Traitement des structures situées sur les toitures

S'il reste des structures sur le toit, comme, par exemple des lucarnes, nous les extrayons du nuage de points pour proposer à l'utilisateur de les reconstruire. Pour cela, nous extrayons les points se trouvant à une distance supérieure à deux fois la densité moyenne du nuage de points à la surface, puis nous ajoutons les points voisins, nous permettant d'obtenir les points appartenant à la fois à la structure sur la toiture. À partir de ce ou ces nuages de points, nous pouvons revenir à l'étape de choix du modèle puis réajuster et ainsi créer la superstructure du toit.

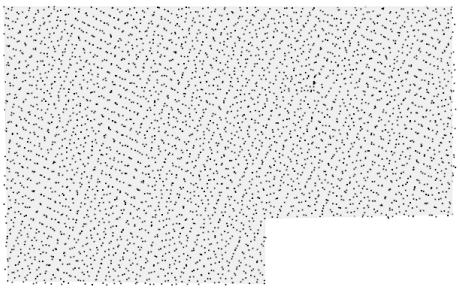


FIGURE 5.8 – Exemples de nuage de points de toitures-terrasses, dont le nuage de points possède un contour non rectangulaire. En gris, la surface reconstruite à partir de la bordure extraite automatiquement.

Extraction des contours de bâtiments pour la reconstruction

Dans les nuages de points, nous pouvons parfois rencontrer des toitures-terrasses dont le contour ne correspond pas aux bâtiments de notre base de données (voir la figure 5.8). Ces bâtiments doivent être reconstruits en restituant du mieux possible le contour, pour cela nous proposons deux méthodes. La première méthode est automatique, mais suppose que le nuage de points n'est pas lacunaire et consiste à estimer au mieux le contour du bâtiment. Pour ce faire, nous nous inspirons de la méthode développée dans le chapitre 1, en estimant le contour du nuage de points en calculant son α-shape, puis en simplifiant son contour à l'aide de l'algorithme de Douglas-Peucker [Douglas and Peucker, 1973], car l'algorithme donne de très bons résultats lorsque la simplification implique une courbe contenant très peu de points. Puisque l'algorithme de Douglas-Peucker est conçu pour les courbes non fermées, nous divisons notre courbe en deux en sélectionnant deux points les plus proches des coins de la boite englobante minimale du nuage de points. Enfin, l'ensemble des segments obtenus sont régularisés de manière à ce qu'il s'ajuste au mieux au nuage de points du contour.

Toutefois, si le nuage de points est lacunaire, nous proposons à l'utilisateur une méthode pour tracer directement à la main la polyligne définissant le contour de la toiture, tout en contraignant cette polyligne à appartenir au plan du nuage de points.

5.3 Résultats et limites

À partir de cet ensemble de méthodes, nous avons entrepris la reconstruction d'une partie de la banlieue ouest de Strasbourg [Strasbourg, 2016]. Nous avons choisi cette zone en particulier, car elle comporte une végétation haute assez développée (soit des occlusions), ainsi que différents types de bâtiments. En effet, cette banlieue est constituée de bâtiments pavillonnaires, mais aussi des bâtiments mitoyens et enfin différents immeubles, elle représente donc un bon ensemble pour tester notre logiciel.

Un ensemble de 546 structures a été restitué, ce compte comprend tout type de reconstruction, lucarne incluse. Une illustration des résultats obtenus est présentée dans la figure 5.9. Les bâtiments ont été pour la plupart reconstruits à partir des méthodes automatiques, mais quelques bâtiments ont nécessité des traitements

FIGURE 5.9 – Exemple d'une reconstruction de la banlieue ouest de Strasbourg [Strasbourg, 2016] réalisée avec l'ensemble d'outils proposés dans RhinoPointCloud. En haut, un quartier pavillonnaire, en bas, un quartier doté de bâtiments mitoyens.

spécifiques.

Limites

Il demeure dans les nuages de points, des bâtiments remarquables (voir figure 5.10), qu'il est pour l'instant impossible de traiter avec le logiciel RhinoPointCloud. Toutefois, nous pensons que ce genre de bâtiments sont hors cadre des bâtiments que nous devons traiter.

5.4 Conclusion

La conception de maquette numérique de ville est une tâche difficile, pour laquelle l'entreprise RhinoTerrain pense qu'il faut impliquer un utilisateur pour obtenir des résultats pertinents. Dans ce chapitre, nous avons présenté une partie des développements que j'ai pu réaliser durant cette thèse pour permettre de faciliter cette tâche. L'ensemble de ces commandes ainsi que l'interface semblent de prime abord intéressés de nombreux partenaire, le logiciel ne devrait donc pas tarder à entrer en phase de bêta test, nous pourrons donc réaliser différents tests d'ergonomie nous permettant

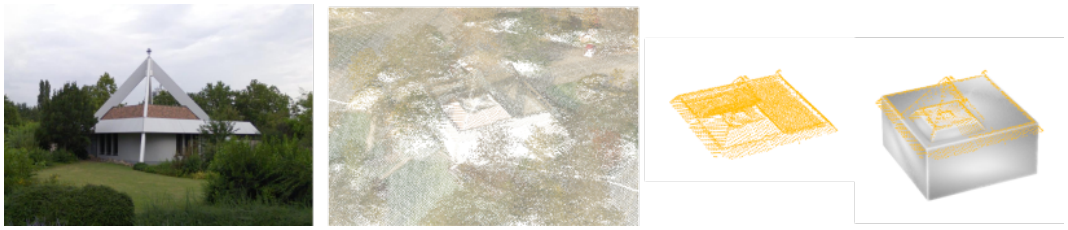


FIGURE 5.10 – Exemple de limite de l'algorithme de reconstruction sur une église de style contemporain. Le logiciel n'offre aucune solution permettant de reconstruire la charpente en béton se trouvant au-dessus de cet édifice.

de comprendre les points faibles de notre pipeline et de les fixer.

Concernant ce logiciel, nous sommes plutôt confiants des résultats qu'il pourrait apporter à nos clients, car ce pipeline fait partie des rares à placer la qualité des modèles générée en avant, grâce à son approche semi-automatique.

Conclusion

Bilan des contributions

Au cours de ce manuscrit, nous avons abordé deux approches différentes pour la reconstruction de maquette numérique urbaine à partir de nuage de points issus de scanner laser LiDAR aéroporté. Une première basée sur les données et une seconde basée sur les modèles sont présentées dans la première partie 5. Puis dans une seconde partie 2.4 nous avons présenté différentes méthodes pour accélérer l'ajustement de surface à un nuage de points. Enfin, nous avons illustré les différentes fonctionnalités implémentées au sein du logiciel Rhinocéros pour permettre à des utilisateurs d'exploiter une partie des méthodes présentées dans cette thèse. Un résumé des contributions est disponible ci-dessous :

Une méthode basée sur les données pour le calcul de maquette numérique urbaine proposant de reconstruire des maillages de toitures à partir de données LiDAR aéroporté classifiées. Cette méthode extrait les points correspondants aux toitures dans le nuage de points, puis à l'aide d'un estimateur de normale issue de la transformée de Hough, agrège les points en ensemble de plans. La spécificité de cette méthode est qu'elle permet de retrouver la connectivité des différents plans grâce à une approche purement géométrique. En effet, la connectivité est extraite grâce à la connectivité observée dans le maillage de Delaunay du nuage de points, rendant la méthode plus robuste. Ces travaux sont décrits dans le chapitre 1 et publié dans [Boltcheva et al., 2020].

Une méthode basée sur les modèles pour le calcul de maquette numérique urbaine permettant de reconstruire des toitures en ajustant une surface, issue d'une base de données de toits prédéfinis, à un nuage de points. Cette méthode minimise une énergie symétrique inspirée de l'énergie de Hausdorff c'est-à-dire, une énergie entre le nuage de points et sa surface et réciproquement une énergie entre la surface et le nuage de points. Cette énergie peut être définie grâce au diagramme de Voronoï restreint à la surface et estimée sous forme close. Cette énergie est contrainte par des contraintes d'égalités et d'inégalités, permettant de conserver certaines propriétés géométriques sur nos surfaces. Enfin, l'énergie obtenue est minimisée rapidement grâce à l'algorithme Broyden–Fletcher–Goldfarb–Shanno (BFGS). Cette méthode se distingue des méthodes de la littérature, car la densité du nuage de points n'a pas d'impact sur ses résultats, en effet l'information injectée par l'utilisateur en choisissant le modèle adéquat permet de pallier le manque d'information dans un nuage de points peu dense. Ces travaux sont décrits dans le chapitre 2 et publié dans [Coiffier et al., 2021].

Une méthode de calcul d'intégrale sur des diagrammes de Voronoï et diagrammes de puissance sur le GPU démontrant d'une grande efficacité pour

le calcul en parallèle de ces structures dans le cadre de la simulation numérique. Cette méthode s'inspire de l'article [Ray et al., 2018] qui encode les diagrammes de Voronoï comme un ensemble de plans coupants et étend l'article au calcul de diagramme de puissance restreint. Pour cela, nous avons développé une nouvelle stratégie d'ordonnancement pour utiliser de manière ingénieuse la mémoire de la carte graphique. L'évaluation des intégrales a également été repensée pour faciliter son calcul : nous considérons localement l'ensemble des triangles intersectant notre cellule. En itérant sur les triangles de ce domaine, nous pouvons calculer l'intégrale de chaque cellule comme une somme de volume signée de cônes tronqués. Cette méthode a été testée sur différentes simulations numériques et comparée aux algorithmes GPU, elle se place comme la méthode la plus rapide et comparé aux algorithmes du CPU, notre solution est un ordre de grandeur plus rapide pour nos cas d'utilisation. Ces travaux sont décrits dans le chapitre 3 et publié dans [Basselin et al., 2021].

L'accélération de la méthode d'appariement de surface à nuage de points et son transfert industriel a été présentée et permet d'accélérer d'un ordre de grandeur en moyenne la méthode précédemment présentée. Cette méthode réalise une approximation de l'énergie de la surface au nuage de points en réalisant une discrétisation de la surface. Cette optimisation permet de produire une méthode plus facilement réimplémentable et permet d'obtenir des résultats similaires à la méthode d'ajustement présentée dans le chapitre 2. L'ensemble de la méthode a été testé sur différents ensembles de données et a été implémenté dans le logiciel Rhinocéros. Un pipeline de traitement a été développé autour de cette méthode permettant de rendre le problème mal posé de la reconstruction de maquette numérique urbaine traitable et contrôlable par l'utilisateur. Ce pipeline alterne les phases de segmentations et d'ajustement tout en garantissant à l'utilisateur à tout moment de repasser en traitement manuel. Ces travaux sont présentés dans les chapitre 4 et chapitre 5 et ne sont pour l'instant pas publiés.

Les chapitres précédemment présentés proposent des méthodes différentes permettant de traiter efficacement le problème de la génération de maillage pour la simulation numérique. Nous pensons avec force que la méthode d'ajustement de surface à nuage de points est aujourd'hui la méthode la plus efficace pour produire des maquettes numériques urbaines dépourvues d'erreurs dans leur maillage. Nous soutenons également qu'une méthode assistée par l'utilisateur permet de produire de meilleurs modèles ne nécessitant pas de phase contraignante de nettoyage après production.

Publications associées à la thèse

- [Basselin et al., 2021] Basselin, Justine and Alonso, Laurent and Ray, Nicolas and Sokolov, Dmitry and Lefebvre, Sylvain and Lévy, Bruno, Restricted power diagrams on the GPU, 2021.
- [Coiffier et al., 2021] Coiffier, Guillaume and Basselin, Justine and Ray, Nicolas and Sokolov, Dmitry, Parametric Surface Fitting on Airborne LiDAR Point Clouds for Building Reconstruction, 2021.
- [Boltcheva et al., 2020] Boltcheva, Dobrina and Basselin, Justine and Poull, Clément and Barthélemy, Hervé et Sokolov, Dmitry, Topological-based roof modeling from 3D point clouds, 2020.

Les travaux effectués durant mes trois années de thèse semblent être une prémisse prometteuse pour la reconstruction de maquette numérique urbaine à partir de nuage de points LiDAR. Une première preuve de concept a été implémentée au sein du logiciel Rhinocéros et devrait passer dans quelque temps en version bêta pour que des utilisateurs puissent utiliser nos outils. Pour autant, il reste encore différents travaux à réaliser, par exemple parvenir à identifier à partir d'une maquette préexistante et d'un nouveau nuage de points les nouvelles toitures à reconstituer.

Une première piste pour la suite de ces travaux serait la mise en place d'algorithmes, basés sur les données et contraints capables de traiter l'ensemble des cas les plus simples de manière plus rapide, la qualité des nuages de points permettant en effet à ces modèles d'être retrouvés très rapidement. Ces résultats pourraient ensuite être présentés à l'utilisateur pour que celui-ci valide les maillages obtenus au fur et à mesure.

Pour autant, nous continuons de penser qu'il est important de reconstruire des maquettes numériques urbaines les plus exactes possibles. Dans cette optique, utiliser une méthode entièrement automatique n'est aujourd'hui pas une solution viable, notamment à cause de la variabilité des données LiDAR. De plus, un modèle de grande qualité sera toujours plus facilement réutilisable dans le futur pour la maintenance des données des villes, car il pourra être mis à jour automatiquement et quasi instantanément à partir des nouvelles données BIM produites par les autorités locales lors de la conception, de la construction, de la rénovation et de la démolition des bâtiments.

Bibliographie

[Abdullah et al., 2014] Abdullah, S., Awrangjeb, M., and Lu, G. (2014). Lidar segmentation using suitable seed points for 3d building extraction. *Int. Arch. of the Phot., R. Sensing & S. I. Sciences*.

[Aggarwal and Vitter, 1988] Aggarwal, A. and Vitter, Jeffrey, S. (1988). The input/output complexity of sorting and related problems. *Communications of the ACM*, 31(9) :1116–1127.

[Agugiaro, 2014] Agugiaro, G. (2014). From sub-optimal datasets to a citygml-compliant 3d city model : Experiences from trento, italy. *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences*, 2(4).

[Agugiaro, 2016] Agugiaro, G. (2016). Energy planning tools and citygml-based 3d virtual city models : experiences from trento (italy). *Applied Geomatics*, 8(1) :41–56.

[Alexander and Rainald, 1995] Alexander, S. and Rainald, L. (1995). Three-dimensional parallel unstructured grid generation. *International Journal for Numerical Methods in Engineering*, 38(6) :905–925.

[Alharthy and Bethel, 2004] Alharthy, A. and Bethel, J. (2004). Detailed building reconstruction from airborne laser data using a moving surface method. *Inter. Archives of Photogrammetry and Remote Sensing*, 35.

[Amenta et al., 2003] Amenta, N., Choi, S., and Rote, G. (2003). Incremental constructions con BRIO. In *Proceedings of the 19th ACM Symposium on Computational Geometry, San Diego, CA, USA, June 8-10, 2003*, pages 211–219.

[Amsellem et al., 2015] Amsellem, G., Grubert, M., and Rabier, B. (2015). *La maquette : un outil au service du projet architectural : actes du colloque qui s' est tenu les 20-21 mai 2011 à la Cité de l'architecture et du patrimoine*. Éditions des Cendres. Les maquettes en bois du dôme de Brunelleschi au Museo dell'Opera del Duomo à Florence : architecture, techniques et projet aux XVe et XVIe siècles.

[Aurenhammer, 1987] Aurenhammer, F. (1987). Power diagrams : Properties, algorithms and applications. *SIAM Journal on Computing*, 16(1) :78–96.

[Awrangjeb and Fraser, 2014] Awrangjeb, M. and Fraser, C. (2014). An automatic and threshold-free performance evaluation system for building extraction techniques from airborne lidar data. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 7(10) :4184–4198.

[Awrangjeb and Fraser, 2014] Awrangjeb, M. and Fraser, C. (2014). Automatic segmentation of raw lidar data for extraction of building roofs. *Remote Sensing*, 6 :3716–3751.

[B. Albers, 2016] B. Albers, M. Kada, A. W. (2016). Automatic extraction and regularization of building outlines from airborne lidar point clouds. *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences*, 41.

[Basselin et al., 2021] Basselin, J., Alonso, L., Ray, N., Sokolov, D., Lefebvre, S., and Lévy, B. (2021). Restricted power diagrams on the gpu. In *Computer Graphics Forum*, volume 40, pages 1–12. Wiley Online Library.

[Batista et al., 2010] Batista, V. H., Millman, D. L., Pion, S., and Singler, J. (2010). Parallel geometric algorithms for multi-core computers. *International Journal for Numerical Methods in Engineering*, 43(8) :663–677.

[Beaton and Tukey, 1974] Beaton, A. E. and Tukey, J. W. (1974). The fitting of power series, meaning polynomials, illustrated on band-spectroscopic data. *Technometrics*, 16(2) :147–185.

[Berger et al., 2017] Berger, M., Tagliasacchi, A., Seversky, L. M., Alliez, P., Guennebaud, G., Levine, J. A., Sharf, A., and Silva, C. T. (2017). A survey of surface reconstruction from point clouds. *Comput. Graph. Forum*, 36(1) :301–329.

[Besl and McKay, 1992] Besl, P. J. and McKay, N. D. (1992). A method for registration of 3-d shapes. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(2) :239–256.

[Biljecki, 2017] Biljecki, F. (2017). Level of detail in 3d city models.

[Biljecki et al., 2016a] Biljecki, F., Ledoux, H., Du, X., Stoter, J., Soon, K. H., and Khoo, V. (2016a). The most common geometric and semantic errors in citygml datasets. *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences*, 4.

[Biljecki et al., 2016b] Biljecki, F., Ledoux, H., and Stoter, J. (2016b). Generation of multi-lod 3d city models in citygml with the procedural modelling engine random3dcity. In *1st International Conference on Smart Data and Smart Cities : 30th UDMS*, pages 51–59. ISPRS.

[Biljecki et al., 2016c] Biljecki, F., Ledoux, H., and Stoter, J. (2016c). An improved lod specification for 3d building models. *Computers, Environment and Urban Systems*, 59 :25–37.

[Biljecki et al., 2015] Biljecki, F., Stoter, J., Ledoux, H., Zlatanova, S., and Çöltekin, A. (2015). Applications of 3d city models : State of the art review. *ISPRS International Journal of Geo-Information*, 4(4) :2842–2889.

[Boge et al., 2013] Boge, M., Meidow, J., and Bulatov, D. (2013). Extraction and refinement of building faces in 3d point clouds. *The Inter. Society for Optical Engineering*, 8892 :88920V.

[Boltcheva et al., 2020] Boltcheva, D., Basselin, J., Poull, C., Barthélemy, H., and Sokolov, D. (2020). Topological-based roof modeling from 3d point clouds.

[Borrmann et al., 2011] Borrmann, D., Elseberg, J., Lingemann, K., and Nüchter, A. (2011). The 3d hough transform for plane detection in point clouds : A review and a new accumulator design. *3D Research*, 2(2) :1–13.

[Boulch and Marlet, 2012] Boulch, A. and Marlet, R. (2012). Fast and robust normal estimation for point clouds with sharp features. *Computer Graphics Forum*, 31 :1765–1774.

[Bowyer, 1981] Bowyer, A. (1981). Computing dirichlet tessellations. *Comput. J.*, 24(2) :162–166.

[Brenner, 2005] Brenner, C. (2005). Building reconstruction from images and laser scanning. *International Journal of Applied Earth Observation and Geoinformation*, 6(3) :187 – 198.

[Buonamici et al., 2018] Buonamici, F., Carfagni, M., Furferi, R., Governi, L., Lapini, A., and Volpe, Y. (2018). Reverse engineering of mechanical parts : A template-based approach. *Journal of Computational Design and Engineering*, 5(2) :145 – 159.

[Buyukdemircioglu et al., 2022] Buyukdemircioglu, M., Kocaman, S., and Kada, M. (2022). Deep learning for 3d building reconstruction : A review. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 43 :359–366.

[Cao, 2014] Cao, T.-T. (2014). *Fundamental Computational Geometry on the GPU*. PhD thesis, National University of Singapore.

[Cao et al., 2014] Cao, T.-T., Nanjappa, A., Gao, M., and Tan, T. S. (2014). A GPU accelerated algorithm for 3d delaunay triangulation. In *Symposium on Interactive 3D Graphics and Games, I3D '14, San Francisco, CA, USA - March 14-16, 2014*, pages 47–54.

[Card et al., 1991] Card, S. K., Robertson, G. G., and Mackinlay, J. D. (1991). The information visualizer, an information workspace. In *Proceedings of the SIGCHI Conference on Human factors in computing systems*, pages 181–186.

[Chen et al., 2014] Chen, D., Zhang, L., Mathiopoulos, P. T., and Huang, X. (2014). A methodology for automated segmentation and reconstruction of urban 3-d buildings from als point clouds. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 7(10) :4199–4217.

[Coiffier et al., 2021] Coiffier, G., Basselin, J., Ray, N., and Sokolov, D. (2021). Parametric surface fitting on airborne lidar point clouds for building reconstruction. *Computer-Aided Design*, 140 :103090.

[D. Chen and Liu, 2012] D. Chen, L. Zhang, J. L. and Liu, R. (2012). Urban building roof segmentation from airborne lidar point clouds. *Inter. Journal of Remote Sensing*, 33 :6497–6515.

[De Cougny and Shephard, 1999] De Cougny, H. L. and Shephard, M. S. (1999). Parallel refinement and coarsening of tetrahedral meshes. *International Journal for Numerical Methods in Engineering*, 46(7) :1101–1125.

[de Goes et al., 2012] de Goes, F., Breeden, K., Ostromoukhov, V., and Desbrun, M. (2012). Blue noise through optimal transport. *ACM Trans. Graph.*, 31(6).

[de Goes et al., 2015a] de Goes, F., Wallez, C., Huang, J., Pavlov, D., and Desbrun, M. (2015a). Power particles : an incompressible fluid solver based on power diagrams. *ACM Trans. Graph.*, 34(4) :50 :1–50 :11.

[de Goes et al., 2015b] de Goes, F., Wallez, C., Huang, J., Pavlov, D., and Desbrun, M. (2015b). Power particles : An incompressible fluid solver based on power diagrams. *ACM Trans. Graph.*, 34(4) :50 :1–50 :11.

[Delage and Devillers, 2018] Delage, C. and Devillers, O. (2018). Spatial sorting. In *CGAL User and Reference Manual*. CGAL Editorial Board, 4.12.1 edition.

[Devillers and Guigue, 2002] Devillers, O. and Guigue, P. (2002). Finite Precision Elementary Geometric Constructions. Technical Report RR-4559, INRIA.

[Directive, 2002] Directive, E. (2002). Directive 2002/49/ec of the european parliament and the council of 25 june 2002 relating to the assessment and management of environmental noise. *Official Journal of the European Communities, L*, 189(18.07) :2002.

[Dong et al., 2017] Dong, C., Wang, R., and Peethambaran, J. (2017). Topologically aware building rooftop reconstruction from airborne laser scanning point clouds. *IEEE Transactions on Geoscience and Remote Sensing*, PP.

[Dorninger and Pfeifer, 2008] Dorninger, P. and Pfeifer, N. (2008). A comprehensive automated 3d approach for building extraction, reconstruction, and regularization from airborne laser scanning point clouds. In *Sensors*.

[Douglas and Peucker, 1973] Douglas, D. and Peucker, T. (1973). Algorithms for the reduction of the number of points required to represent a digitized line or its caricature. *Inter. J. for Geographic Information and Geovisualization*, 10(2) :112–222.

[Duckham et al., 2008] Duckham, M., Kulik, L., Worboys, M., and Galton, A. (2008). Efficient generation of simple polygons for characterizing the shape of a set of points in the plane. *Pattern Recognition*, 41 :3224–3236.

[Edelsbrunner, 2010] Edelsbrunner, H. (2010). Alpha shapes - a survey. *Tessellations in the Sciences*.

[Edelsbrunner et al., 1983] Edelsbrunner, H., Kirkpatrick, D., and Seidel, R. (1983). On the shape of a set of points in the plane. *IEEE Transactions on information theory*, 29(4) :551–559.

[Elaksher et al., 2002] Elaksher, A. F., Bethel, J. S., et al. (2002). Reconstructing 3d buildings from lidar data. *International Archives Of Photogrammetry Remote Sensing and Spatial Information Sciences*, 34(3/A) :102–107.

[Fan et al., 2014] Fan, H., Yao, W., and Fu, Q. (2014). Segmentation of sloped roofs from airborne lidar point clouds using ridge-based hierarchical decomposition. *Remote Sensing*, 6 :3284–3301.

[Fei et al., 2014] Fei, Y., Rong, G., Wang, B., and Wang, W. (2014). Parallel l-bfgs-b algorithm on gpu. *Computers & Graphics*, 40 :1 – 9.

[for Photogrammetry and (ASPRS), 2008] for Photogrammetry, A. S. and (ASPRS), R. S. (2008). Las specification version 1.2. bethesda, maryland. *American Society for Photogrammetry and Remote Sensing Publication*.

[Freitas et al., 2015] Freitas, S., Catita, C., Redweik, P., and Brito, M. C. (2015). Modelling solar potential in the urban environment : State-of-the-art review. *Renewable and Sustainable Energy Reviews*, 41 :915–931.

[G. Forlani and Zingaretti, 2006] G. Forlani, C. Nardinocchi, M. S. and Zingaretti, P. (2006). Complete classification of raw lidar data and 3d reconstruction of buildings. *Pattern analysis and appli.*, 8(4) :357–374.

[G. Sohn and Tao, 2008] G. Sohn, X. H. and Tao, V. (2008). Using a binary space partitioning tree for reconstruction polyhedral building models from airbome lidar data. *Photogrammetric Engineering and Remote Sensing*, 74 :1425–1440.

[Gallouët and Mérigot, 2017] Gallouët, T. O. and Mérigot, Q. (2017). A Lagrangian scheme à la Brenier for the incompressible euler equations. *Foundations of Computational Mathematics*.

[Gandini et al., 2020] Gandini, A., Garmendia, L., Prieto, I., Álvarez, I., and San-José, J.-T. (2020). A holistic and multi-stakeholder methodology for vulnerability assessment of cities to flooding and extreme precipitation events. *Sustainable Cities and Society*, 63 :102437.

[Gonzalez, 2016] Gonzalez, R. E. (2016). Paravt : Parallel voronoi tessellation code. *Astronomy and Computing*, 17 :80–85.

[Gröger and Plümer, 2012] Gröger, G. and Plümer, L. (2012). Citygml–interoperable semantic 3d city models. *ISPRS Journal of Photogrammetry and Remote Sensing*, 71 :12–33.

[Gröger et al., 2012] Gröger, G., Kolbe, T. H., Nagel, C., and Häfele, K.-H. (2012). Ogc city geography markup language (citygml) encoding standard.

[Haala and Kada, 2010] Haala, N. and Kada, M. (2010). An update on automatic 3d building reconstruction. *ISPRS Journal of Photogrammetry and Remote Sensing*, 65(6) :570 – 580.

[Henn et al., 2013] Henn, A., Gröger, G., Stroh, V., and Plümer, L. (2013). Model driven reconstruction of roofs from sparse lidar point clouds. *ISPRS Journal of Photogrammetry and Remote Sensing*, 76 :17 – 29. Terrestrial 3D modelling.

[Holland and Welsch, 1977] Holland, P. W. and Welsch, R. E. (1977). Robust regression using iteratively reweighted least-squares. *Communications in Statistics-theory and Methods*, 6(9) :813–827.

[Hough, 1962] Hough, P. (1962). Method and means for recognizing complex patterns. *US patent*, 3(6).

[Huang and Brenner, 2011] Huang, H. and Brenner, C. (2011). Rule-based roof plane detection and segmentation from laser point clouds. In *Joint Urban Remote Sensing Event*, pages 293–296. IEEE.

[Huang et al., 2011a] Huang, H., Brenner, C., and Sester, M. (2011a). 3d building roof reconstruction from point clouds via generative models. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 16–24.

[Huang et al., 2011b] Huang, H., Brenner, C., and Sester, M. (2011b). 3d building roof reconstruction from point clouds via generative models. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pages 16–24.

[Inria, 2018] Inria, p. A.-P. (2018). Geogram : a programming library of geometric algorithms. `http://alice.loria.fr/software/geogram/doc/html/index.html`.

[Jaewook et al., 2017] Jaewook, J., Yoonseok, J., and Gunho, S. (2017). Implicit regularization for reconstructing 3d building rooftop models using airborne lidar data. *Sensors*, 17.

[Jang et al., 2021] Jang, Y.-H., Park, S. I., Kwon, T. H., and Lee, S.-H. (2021). Citygml urban model generation using national public datasets for flood damage simulations : A case study in korea. *Journal of Environmental Management*, 297 :113236.

[Jarzabek-Rychard, 2012] Jarzabek-Rychard, M. (2012). Reconstruction of building outlines in dense urban areas based on lidar data and address points. *Inter. Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 39 :121–126.

[Jarzabek-Rychard and Maas, 2017] Jarzabek-Rychard, M. and Maas, H.-G. (2017). Geometric refinement of als-data derived building models using monoscopic aerial images. *Remote Sensing*, 9 :282.

[Kaden and Kolbe, 2014] Kaden, R. and Kolbe, T. H. (2014). Simulation-based total energy demand estimation of buildings using semantic 3d city models. *International Journal of 3-D Information Modeling (IJ3DIM)*, 3(2) :35–53.

[Kavisha et al., 2017] Kavisha, K., Ledoux, H., Commandeur, T., Stoter, J., and Kavisha, K. (2017). Modelling urban noise in citygml ade : Case of the netherlands. In *12th 3D Geoinfo Conference*, volume 4. ISPRS.

[Kettner, 1999] Kettner, L. (1999). Using generic programming for designing a data structure for polyhedral surfaces. *Computational Geometry*, 13(1) :65–90.

[Kim and Shan, 2011] Kim, K. and Shan, J. (2011). Building roof modeling from airborne laser scanning data based on level set approach. *ISPRS Journal of Photogrammetry and Remote Sensing*, 66(4) :484–497.

[Kitagawa et al., 2016] Kitagawa, J., Mérigot, Q., and Thibert, B. (2016). A newton algorithm for semi-discrete optimal transport. *CoRR*, abs/1603.05579.

[Kubiak and Ławniczak, 2016] Kubiak, J. and Ławniczak, R. (2016). The propagation of noise in a built-up area (on the example of a housing estate in poznań). *Journal of Maps*, 12(2) :231–236.

[Lafarge and Mallet, 2012] Lafarge, F. and Mallet, C. (2012). Creating large-scale city models from 3d-point clouds : a robust approach with hybrid representation. *International journal of computer vision*, 99(1) :69–85.

[Lasserre and Avrachenkov, 2001] Lasserre, J. B. and Avrachenkov, K. E. (2001). The multi-dimensional version of $\int_b^a x^p dx$. *The American Mathematical Monthly*, 108(2) :151–154.

[Le and Duan, 2017] Le, T. and Duan, Y. (2017). A primitive-based 3d segmentation algorithm for mechanical cad models. *Computer Aided Geometric Design*, 52 :231–246.

[Ledoux, 2018] Ledoux, H. (2018). val3dity : validation of 3d gis primitives according to the international standards. *Open Geospatial Data, Software and Standards*, 3(1) :1–12.

[Ledoux et al., 2019] Ledoux, H., Arroyo Ohori, K., Kumar, K., Dukai, B., Labetski, A., and Vitalis, S. (2019). Cityjson : A compact and easy-to-use encoding of the citygml data model. *Open Geospatial Data, Software and Standards*, 4(1) :1–12.

[Lévy, 2015] Lévy, B. (2015). A numerical algorithm for l2 semi-discrete optimal transport in 3d. *ESAIM : Mathematical Modelling and Numerical Analysis*, 49(6) :1693–1715.

[Lévy and Bonneel, 2013] Lévy, B. and Bonneel, N. (2013). Variational anisotropic surface meshing with voronoi parallel linear enumeration. In Jiao, X. and Weill, J.-C., editors, *Proceedings of the 21st International Meshing Roundtable*, pages 349–366, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Lévy and Schwindt, 2018] Lévy, B. and Schwindt, E. L. (2018). Notions of optimal transport theory and how to implement them on a computer. *Computers & Graphics*, 72 :135–148.

[Li et al., 2009] Li, H., Adams, B., Guibas, L. J., and Pauly, M. (2009). Robust single-view geometry and motion reconstruction. *ACM Trans. Graph.*, 28(5) :1–10.

[Lien and Kajiya, 1984] Lien, S. and Kajiya, J. T. (1984). A symbolic method for calculating the integral properties of arbitrary nonconvex polyhedra. *IEEE Computer Graphics and Applications*, 4(10) :35–42.

[Liu and Nocedal, 1989] Liu, D. C. and Nocedal, J. (1989). On the limited memory bfgs method for large scale optimization. *Math. Program.*, 45(1–3) :503–528.

[Liu et al., 2020] Liu, X., Ma, L., Guo, J., and Yan, D.-M. (2020). Parallel computation of 3d clipped voronoi diagrams. *IEEE Transactions on Visualization and Computer Graphics*, PP :1–1.

[Liu and Yan, 2019] Liu, X. and Yan, D.-M. (2019). Computing 3d clipped voronoi diagrams on gpu. In *SIGGRAPH Asia 2019 Posters*, SA '19, pages 9 :1–9 :2, New York, NY, USA. ACM.

[Liu et al., 2009] Liu, Y., Wang, W., Lévy, B., Sun, F., Yan, D.-M., Lu, L., and Yang, C. (2009). On centroidal voronoi tessellation—energy smoothness and fast computation. *ACM Transactions on Graphics (ToG)*, 28(4) :1–17.

[Lu et al., 2012] Lu, L., Lévy, B., and Wang, W. (2012). Centroidal voronoi tessellation of line segments and graphs. In *Computer Graphics Forum*, volume 31, pages 775–784. Wiley Online Library.

[Maas and Vosselman, 1999a] Maas, H. and Vosselman, G. (1999a). Two algorithms for extracting building models from raw laser altimetry data. *J. of Photogrammetry and Remote Sensing*, 54(2) :153 – 163.

[Maas and Vosselman, 1999b] Maas, H.-G. and Vosselman, G. (1999b). Two algorithms for extracting building models from raw laser altimetry data. *ISPRS Journal of photogrammetry and remote sensing*, 54(2-3) :153–163.

[Maltezos and Ioannidis, 2016] Maltezos, E. and Ioannidis, C. (2016). Automatic extraction of building roof planes from airborne lidar data applying an extended 3d randomized hough transform. *ISPRS Annals of Photogrammetry, Remote Sensing and Spatial Information Sciences*, III-3 :209–216.

[Marot et al., 2018] Marot, C., Pellerin, J., and Remacle, J.-F. (2018). One machine, one minute, three billion tetrahedra. `https://arxiv.org/abs/1805.08831`.

[Martín et al., 2015] Martín, A. M., Domínguez, J., and Amador, J. (2015). Applying lidar datasets and gis based model to evaluate solar potential over roofs : a review. *AIMS Energy*, 3(3) :326–343.

[Martínez et al., 2016] Martínez, J., Dumas, J., and Lefebvre, S. (2016). Procedural Voronoi Foams for Additive Manufacturing. *ACM Transactions on Graphics*, 35 :1 – 12.

[Martín-Jiménez et al., 2020] Martín-Jiménez, J., Del Pozo, S., Sánchez-Aparicio, M., and Lagüela, S. (2020). Multi-scale roof characterization from lidar data and aerial orthoimagery : Automatic computation of building photovoltaic capacity. *Automation in Construction*, 109 :102965.

[Meidow and Hammer, 2016] Meidow, J. and Hammer, H. (2016). Algebraic reasoning for the enhancement of data-driven building reconstructions. *J. of Photogrammetry and Remote Sensing*, 114 :179–190.

[Mérigot, 2011] Mérigot, Q. (2011). A multiscale approach to optimal transport. *Comput. Graph. Forum*, 30(5) :1583–1592.

[Mérigot et al., 2011] Mérigot, Q., Ovsjanikov, M., and Guibas, L. J. (2011). Voronoi-Based Curvature and Feature Estimation from Point Clouds. *IEEE Transactions on Visualization and Computer Graphics*, 17(6) :743 – 756.

[Methirumangalath et al., 2015] Methirumangalath, S., Parakkat, A., and Muthuganapathy, R. (2015). A unified approach towards reconstruction of a planar point set. *Computers & Graphics*, 51.

[Milde and Brenner, 2009] Milde, J. and Brenner, C. (2009). Graph-based modeling of building roofs. In *Proceedings of the 12th AGILE Conference on GIScience, Hannover, Germany (on CD-ROM)*.

[métropole, 2022a] métropole, G. L. (2022a). Page web des données des acteurs du territoire de la métropole de lyon. `https://data.grandlyon.com/`.

[métropole, 2022b] métropole, G. L. (2022b). Page web du cadastre solaire sur grand lyon. `https://cadastresolaire.grandlyon.com/`.

[Mullen et al., 2011] Mullen, P., Memari, P., de Goes, F., and Desbrun, M. (2011). Hot : Hodge-optimized triangulations. In *ACM SIGGRAPH 2011 Papers*, SIGGRAPH '11, New York, NY, USA. Association for Computing Machinery.

[Muller and Preparata, 1978] Muller, D. and Preparata, F. (1978). Finding the intersection of two convex polyhedra. *Theoretical Computer Science*, 7(2) :217 – 236.

[Musialski et al., 2013] Musialski, P., Wonka, P., Aliaga, D. G., Wimmer, M., Van Gool, L., and Purgathofer, W. (2013). A survey of urban reconstruction. In *Computer graphics forum*, volume 32, pages 146–177. Wiley Online Library.

[Nancy, 2022] Nancy (2022). Nuage de points de nancy. `https://www.datagrandest.fr/data4citizen/visualisation/information/?id=3d-modele-numerique-delevation-mne-50-cm-2020-metropole-du-grand-nancy`.

[Nikos and Damian, 2003] Nikos, C. and Damian, N. (2003). Parallel delaunay mesh generation kernel. *International Journal for Numerical Methods in Engineering*, 58(2) :161–176.

[Nivoliers, 2012] Nivoliers, V. (2012). *Phd thesis : Échantillonnage pour l'approximation de fonctions sur des maillages*. PhD thesis, INRIA.

[Nivoliers et al., 2014] Nivoliers, V., Yan, D.-M., and Lévy, B. (2014). Fitting polynomial surfaces to triangular meshes with voronoi squared distance minimization. *Engineering with Computers*, 30(3) :289–300.

[Novacheva, 2008] Novacheva, A. (2008). Building roof reconstruction from lidar data and aerial images through plane extraction and colour edge detection. *Inter. Society for Photogrammetry and Remote Sensing*, pages 51–56.

[Overby et al., 2004] Overby, J., Bodum, L., Kjems, E., and Iisoe, P. (2004). Automatic 3d building reconstruction from airborne laser scanning and cadastral data using hough transform. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, XXXIV.

[Passalacqua et al., 2015] Passalacqua, P., Belmont, P., Staley, D. M., Simley, J. D., Arrowsmith, J. R., Bode, C. A., Crosby, C., DeLong, S. B., Glenn, N. F., Kelly, S. A., et al. (2015). Analyzing high resolution topography for advancing the understanding of mass and energy transfer through landscapes : A review. *Earth-Science Reviews*, 148 :174–193.

[PCL, 2022] PCL (2022). Pcl. `pointclouds.org`.

[Pellegrini et al., 2008] Pellegrini, S., Schindler, K., and Nardi, D. (2008). A generalisation of the icp algorithm for articulated bodies. In *Proceedings of the British Machine Vision Conference*, pages 87.1–87.10. BMVA Press. doi :10.5244/C.22.87.

[Peter and Pfeifer, 2008] Peter, D. and Pfeifer, N. (2008). A comprehensive automated 3d approach for building extraction, reconstruction, and regularization from airborne laser scanning point clouds. *Sensors*, 8.

[Petrie and Toth, 2018] Petrie, G. and Toth, C. K. (2018). Introduction to laser ranging, profiling, and scanning. In *Topographic laser ranging and scanning*, pages 1–28. CRC Press.

[Poullis and You, 2008] Poullis, C. and You, S. (2008). Photorealistic large-scale urban city model reconstruction. *IEEE transactions on visualization and computer graphics*, 15(4) :654–669.

[R. Wang and Chen, 2018] R. Wang, J. P. and Chen, D. (2018). Lidar point clouds to 3-d urban models : a review. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 11(2) :606–627.

[Ray et al., 2018] Ray, N., Sokolov, D., Lefebvre, S., and Lévy, B. (2018). Meshless voronoi on the gpu. In *SIGGRAPH Asia 2018 Technical Papers*, SIGGRAPH Asia '18, pages 265 :1–265 :12, New York, NY, USA. ACM.

[R.C. dos Santos and Carrilho, 2019] R.C. dos Santos, M. G. and Carrilho, A. (2019). Extraction of building roof boundaries from lidar data using an adaptive alpha-shape algorithm. *IEEE Geoscience and Remote Sensing Letters*, 16(8) :1289–1293.

[Remacle, 2017] Remacle, J.-F. (2017). A two-level multithreaded delaunay kernel. *Computer-Aided Design*, (85) :2–9.

[Reporting, 2007] Reporting, H. A. (2007). Asprs lidar guidelines.

[RhinoTerrain, 2022] RhinoTerrain (2022). Rhinocapture. `www.rhinoterrain.com`.

[Rottensteiner, 2006] Rottensteiner, F. (2006). Consistent estimation of building parameters considering geometric regularities by soft constraints. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 34.

[Rottensteiner et al., 2012] Rottensteiner, F., Trinder, J., Clode, S., and Kubik, K. (2012). Automated delineation of roof planes from lidar data. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 36.

[Rusu et al., 2009] Rusu, R. B., Blodow, N., and Beetz, M. (2009). Fast point feature histograms (fpfh) for 3d registration. In *2009 IEEE International Conference on Robotics and Automation*, pages 3212–3217.

[Rutzinger et al., 2009] Rutzinger, M., Rottensteiner, F., and Pfeifer, N. (2009). A comparison of evaluation techniques for building extraction from airborne laser

scanning. *J. of Selected Topics in Applied Earth Observations and Remote Sensing*, 2(1) :11–20.

[Rycroft, 2009] Rycroft, C. (2009). Voro++ : A three-dimensional voronoi cell library in c++. Technical report, Lawrence Berkeley National Lab.(LBNL), Berkeley, CA (United States).

[Sahoo et al., 2006] Sahoo, B., Chatterjee, C., Raghuwanshi, N. S., Singh, R., and Kumar, R. (2006). Flood estimation by giuh-based clark and nash models. *Journal of Hydrologic Engineering*, 11(6) :515–525.

[Sainlot et al., 2017] Sainlot, M., Nivoliers, V., and Attali, D. (2017). Restricting voronoi diagrams to meshes using corner validation. In *Computer Graphics Forum*, volume 36, pages 81–91. Wiley Online Library.

[Segal et al., 2009] Segal, A., Hähnel, D., and Thrun, S. (2009). Generalized-icp. In Trinkle, J., Matsuoka, Y., and Castellanos, J. A., editors, *Robotics : Science and Systems*. The MIT Press.

[Shahzad and Zhu, 2016] Shahzad, M. and Zhu, X. X. (2016). Automatic detection and reconstruction of 2-d/3-d building shapes from spaceborne tomosar point clouds. *IEEE Transactions on Geoscience and Remote Sensing*, 54(3) :1292–1310.

[Shan and Toth, 2018] Shan, J. and Toth, C. K. (2018). *Topographic laser ranging and scanning : principles and processing*. CRC press.

[Smelik et al., 2014] Smelik, R. M., Tutenel, T., Bidarra, R., and Benes, B. (2014). A survey on procedural modelling for virtual worlds. In *Computer Graphics Forum*, volume 33, pages 31–50. Wiley Online Library.

[Sohn et al., 2013] Sohn, G., Jung, J., Jwa, Y., and Armenakis, C. (2013). Sequential modelling of building rooftops by integrating airborne lidar data and optical imagery : preliminary results. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, II-3/W1.

[Sorkine and Alexa, 2007] Sorkine, O. and Alexa, M. (2007). As-rigid-as-possible surface modeling. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing*, SGP '07, page 109–116, Goslar, DEU. Eurographics Association.

[Stoll et al., 2006] Stoll, C., Karni, Z., Rössl, C., Yamauchi, H., and Seidel, H.-P. (2006). Template Deformation for Point Cloud Fitting. In Botsch, M., Chen, B., Pauly, M., and Zwicker, M., editors, *Symposium on Point-Based Graphics*. The Eurographics Association.

[Stoter et al., 2008] Stoter, J., De Kluijver, H., and Kurakula, V. (2008). 3d noise mapping in urban areas. *International Journal of Geographical Information Science*, 22(8) :907–924.

[Stoter et al., 2016] Stoter, J., Vallet, B., Lithen, T., Pla, M., Wozniak, P., Kellenberger, T., Streilein, A., Ilves, R., and Ledoux, H. (2016). State-of-the-art of 3d national mapping in 2016. *Int. Arch. Photogramm. Remote Sens. Spatial Inf. Sci*, 41 :B4.

[Strasbourg, 2016] Strasbourg (2016). Lidar data. `https://data.strasbourg.eu/explore/dataset/odata3d_lidar/information`.

[Sumner et al., 2007] Sumner, R. W., Schmid, J., and Pauly, M. (2007). Embedded deformation for shape manipulation. In *ACM SIGGRAPH 2007 Papers*,

SIGGRAPH '07, page 80–es, New York, NY, USA. Association for Computing Machinery.

[Sun and Salvaggio, 2013] Sun, S. and Salvaggio, C. (2013). Aerial 3d building detection and modeling from airborne lidar point clouds. *Selected Topics in Applied Earth Observations and Remote Sensing, IEEE Journal of*, 6 :1440–1449.

[Tam et al., 2013] Tam, G. K. L., Cheng, Z., Lai, Y., Langbein, F. C., Liu, Y., Marshall, D., Martin, R. R., Sun, X., and Rosin, P. L. (2013). Registration of 3d point clouds and meshes : A survey from rigid to nonrigid. *IEEE Transactions on Visualization and Computer Graphics*, 19(7) :1199–1217.

[Tarsha-Kurdi et al., 2008a] Tarsha-Kurdi, F., Landes, T., and Grussenmeyer, P. (2008a). Extended ransac algorithm for automatic detection of building roof planes from lidar data.

[Tarsha-Kurdi et al., 2008b] Tarsha-Kurdi, F., Landes, T., and Grussenmeyer, P. (2008b). Extended ransac algorithm for automatic detection of building roof planes from lidar data. *The Photogrammetric Journal of Finland*, 21 :97–109.

[The CGAL Project, 2018] The CGAL Project (2018). *CGAL User and Reference Manual*. CGAL Editorial Board, 4.12.1 edition.

[V. Verma and Hsu, 2006] V. Verma, R. K. and Hsu, S. (2006). 3d building detection and modeling from aerial lidar data. *Computer Society Conference on Computer Vision and Pattern Recognition*, 2 :2213 – 2220.

[Varduhn et al., 2015] Varduhn, V., Mundani, R.-P., and Rank, E. (2015). Multiresolution models : Recent progress in coupling 3d geometry to environmental numerical simulation. *3D geoinformation science*, pages 55–69.

[Visvalingam and Whyatt, 1992] Visvalingam, M. and Whyatt, J. (1992). *Line generalisation by repeated elimination of the smallest area*. Cartographic Information Systems Research Group.

[Vosselman, 1999] Vosselman, G. (1999). Building reconstruction using planar faces in very high density height data. *International Archives of Photogrammetry and Remote Sensing*, 32(3 ; SECT 2W5) :87–94.

[Vosselman, 2000] Vosselman, G. (2000). Building reconstruction using planar faces in very high density height data. *Int. Arch. Photogramm. Remot. Sens*, 32.

[Wang, 2017] Wang, L. (2017). *Algorithms and Criteria for Volumetric Centroidal Voronoi Tessellations*. PhD thesis, Université Grenoble Alpes.

[Watson, 1981] Watson, D. (1981). Computing the n-dimensional delaunay tessellation with application to voronoi polytopes. *Comput. J.*, 24(2) :167–172.

[Xiao et al., 2015] Xiao, Y., Wang, C., Li, J., Zhang, W., Xi, X., Wang, C., and Dong, P. (2015). Building segmentation and modeling from airborne lidar data. *International Journal of Digital Earth*, 8(9) :694–709.

[Xin et al., 2016] Xin, S.-Q., Lévy, B., Chen, Z., Chu, L., Yu, Y., Tu, C., and Wang, W. (2016). Centroidal power diagrams with capacity constraints : computation, applications, and extension. *ACM Trans. Graph.*, 35(6) :244 :1–244 :12.

[Xiong et al., 2014a] Xiong, B., Elberink], S. O., and Vosselman, G. (2014a). A graph edit dictionary for correcting errors in roof topology graphs reconstructed from point clouds. *ISPRS Journal of Photogrammetry and Remote Sensing*, 93 :227 – 242.

[Xiong et al., 2014b] Xiong, B., Elberink, S. O., and Vosselman, G. (2014b). A graph edit dictionary for correcting errors in roof topology graphs reconstructed from point clouds. *ISPRS Journal of photogrammetry and remote sensing*, 93 :227–242.

[Xu et al., 2017] Xu, B., Jiang, W., and Li, L. (2017). Hrtt : A hierarchical roof topology structure for robust building roof reconstruction from point clouds. *Remote Sensing*, 9 :354.

[Zhang et al., 2006] Zhang, K., Yan, J., and Chen, S.-C. (2006). Automatic construction of building footprints from airborne lidar data. *Geoscience and Remote Sensing, IEEE Transactions on*, 44 :2523 – 2533.

[Zhang et al., 2021] Zhang, W., Li, Z., and Shan, J. (2021). Optimal model fitting for building reconstruction from point clouds. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 14 :9636–9650.

[Zhao et al., 2013] Zhao, Z., Ledoux, H., and Stoter, J. (2013). Automatic repair of citygml lod2 buildings using shrink-wrapping. In *8th 3DGeoInfo Conference & WG II/2 Workshop, Istanbul, Turkey, 27–29 November 2013, ISPRS Archives Volume II-2/W1*. ISPRS.

[Zhou and Neumann, 2008] Zhou, Q.-Y. and Neumann, U. (2008). Fast and extensible building modeling from airborne lidar data. In *Proceedings of the 16th ACM SIGSPATIAL international conference on Advances in geographic information systems*, page 7. ACM.

[Zhou and Neumann, 2010] Zhou, Q.-Y. and Neumann, U. (2010). 2.5d dual contouring : A robust approach to creating building models from aerial lidar point clouds. In Daniilidis, K., Maragos, P., and Paragios, N., editors, *Computer Vision – ECCV 2010*, pages 115–128, Berlin, Heidelberg. Springer Berlin Heidelberg.

[Zhou and Neumann, 2012] Zhou, Q.-Y. and Neumann, U. (2012). 2.5d building modeling by discovering global regularities. In *2012 IEEE Conference on Computer Vision and Pattern Recognition*, pages 326–333.

Résumé

La numérisation d'objets réels est de plus en plus utilisée dans des domaines tels que l'urbanisme, l'architecture, la gestion des catastrophes et la sécurité intérieure. Des outils d'acquisition tels que les scanners aériens de détection et de télémétrie par la lumière (LiDAR) permettent de produire des représentations numériques de villes entières sous la forme de nuages de points 3D échantillonnant les surfaces des objets dans l'environnement. Malgré le haut degré de maturité atteint par les techniques de numérisation, les solutions informatiques efficaces pour le prétraitement et la reconstruction à partir de ces mesures sont rares et mal adaptées à la complexité de l'environnement. Aujourd'hui, le processus de création d'un modèle numérique à partir de ces données est long, fastidieux et essentiellement manuel. Dans ce processus de rétroconception, l'opérateur humain dessine manuellement les éléments du modèle 3D au plus près du nuage de points.

Bien que des efforts importants aient été déployés pour développer des méthodes automatiques et semi-automatiques, qui apparaissent actuellement sur le marché, aucune solution proposée jusqu'à présent ne répond à toutes les exigences industrielles en termes de précision, d'exactitude et d'efficacité. En effet, la reconstruction de modèles de bâtiments en 3D est une tâche complexe qui nécessite un flux de travail composé de plusieurs étapes de traitement telles que la classification, l'extraction de contours, la segmentation, la reconnaissance de caractéristiques, la génération et la vérification d'hypothèses, la modélisation et la construction géométriques, l'ajustement et le raffinement. De plus, les modèles reconstruits doivent respecter un certain nombre de contraintes structurelles (planéité des segments de toit, arêtes de toit horizontales, symétrie, etc. Malgré les connaissances acquises, il existe encore un nombre important de problèmes non résolus provenant de : lacunes dans les données (dues à des occlusions ou à des réflexions et absorptions indésirables) ; bruit et valeurs aberrantes ; résolution limitée et densité de points variable ; grande variabilité et complexité des formes de bâtiments dans les zones urbaines, pour n'en nommer que quelques-uns. Dans ce travail, nous abordons le problème particulier de la construction (création) de modèles de toit 3D polygonaux à partir de données ponctuelles LIDAR préalablement classées.

Mots-clés: Maquette numérique urbaine, reconstruction, nuage de points, LIDAR ALS.

Abstract

Digitization of real-world objects is increasingly used in fields such as urban planning, architecture, disaster management, and homeland security. Acquisition tools such as Light Detection and Ranging (LiDAR) airborne scanners are used to produce digital representations of entire cities in the form of 3D point clouds sampling the surfaces of objects in the environment. Despite the high degree of maturity reached by the digitizing techniques, efficient computing solutions for pre-processing

and reconstruction from these measurements are scarce and poorly adapted to the complexity of the environment (complex structures of buildings and entire cities). Today, the process of creating a digital model from these data is time-consuming, tedious, and essentially manual. In this reverse engineering process, the human operator manually draws the elements of the 3D model as close as possible to the point cloud. Although significant efforts have been made to develop automatic and semi-automatic methods, which are currently appearing on the market, no solution proposed so far meets all industrial requirements in terms of precision, accuracy and efficiency. Indeed, the reconstruction of 3D building models is a complex task that requires a workflow composed of several processing steps such as classification, contour extraction, segmentation, feature recognition, hypothesis generation and verification, geometric modeling and construction, adjustment and refinement. In addition, the reconstructed models must meet a number of structural constraints (flatness of roof segments, horizontal roof edges, symmetry, etc.). Despite the knowledge gained, there are still a significant number of unsolved problems arising from : data gaps (due to occlusions or unwanted reflections and absorptions) ; noise and outliers ; limited resolution and variable point density ; high variability and complexity of building shapes in urban areas, to name a few. In this work, we address the particular problem of constructing (creating) polygonal 3D roof models from previously classified LIDAR point data.

Keywords: 3d city model, reconstruction, point cloud, LIDAR ALS.