Contributions en traitements basés points pour le rendu et la simulation en mécanique des fluides.

Hassan Bouchiba

THÈSE DE DOCTORAT

de l'Université de recherche Paris Sciences et Lettres
PSL Research University

Préparée à MINES ParisTech

Contributions en traitements basés points pour le rendu et la simulation en mécanique des fluides.

École doctorale n°432

SCIENCES ET MÉTIERS DE L'INGÉNIEUR

Spécialité MATHÉMATIQUES, INFORMATIQUE TEMPS-RÉEL, ROBOTIQUE

COMPOSITION DU JURY :

Mme Marie-Paule Cani
Présidente du jury

M Tamy Boubekeur
Rapporteur

M Florent Lafarge
Rapporteur

M Mathieu Brédif
Examinateur

M Thierry Coupez
Examinateur

M Jean-Emmanuel Deschaud
Examinateur

M François Goulette
Examinateur

Soutenue par **Hassan Bouchiba**
le 5 juillet 2018

Dirigée par **François Goulette**
et par **Jean-Emmanuel Deschaud**

MINES ParisTech | PSL RESEARCH UNIVERSITY PARIS

Remerciements

Je souhaiterais tout d'abord, non seulement remercier, mais surtout dédier cette thèse à mes parents, Khadija et Fouzi. Sans tous les sacrifices qu'ils ont consentis, leur soutien et leur amour, rien de tout cela n'aurait été possible. Ce sont eux qui m'ont insufflé le goût des études, l'envie de comprendre, la soif de savoir, et ce sont eux qui m'ont permis de mener à terme ce projet. Merci maman, merci papa, sans vous rien n'aurait été possible.

Je souhaiterais également remercier mes encadrants Jean-Emmanuel Deschaud et François Goulette pour leur engagement et leur aide dans ce projet. Ils m'ont transmis leur savoir et m'ont été d'un soutien précieux pour mener à bien cette thèse. Merci à François de m'avoir donné la fantastique opportunité de pouvoir faire une thèse sous sa direction. Merci à Jean-Emmanuel pour son accompagnement, ses encouragements quotidiens et ses nombreux retours. Merci aussi pour son aide sans faille, pour les nombreuses discussions scientifiques que nous avons eues et ses apports précieux tout le long de cette thèse.

Je voudrais également remercier tout particulièrement ma fiancée Marie-Amandine pour son soutien inconditionnel et pour son aide précieuse. Elle a partagé ma vie durant ces années de doctorat, elle a vécu avec moi les moments difficiles comme les moments heureux de cette thèse. Elle m'a été d'un support très précieux dans les moments forts de cette aventure et n'a jamais cessé de croire en moi. Du fond du cœur je te remercie. Je remercie également mon petit frère, Younes, ma famille et ma belle famille, en particulier Alain, Pascale et Mathilde pour leur grande aide et leur soutien à toute épreuve. J'ai également une pensée pour mon grand père de cœur Jacky, ainsi qu'à tous nos proches qui nous ont quitté trop tôt.

Enfin un grand merci à tous mes collègues, travailler à leur côté a été un honneur et un plaisir. Je garderai un souvenir indélébile des repas au ministère qui auront rythmé mes journées et qui étaient le théâtre de discussions passionnées. Je remercie enfin mes amis qui m'auront soutenu tout au long de cette aventure. Merci également au lecteur qui, je l'espère, trouvera dans ce document au moins autant de plaisir que j'en ai eu à travailler sur ce projet.

Table des matières

Introduction

Cette thèse a été préparée au sein de l'équipe *"Nuages de points et modélisation 3D"* du laboratoire de robotique de l'école des Mines ParisTech (CAOR). Historiquement cette équipe a été l'une des premières à développer un système mobile de cartographie terrestre basé Lidar (F. Goulette et al., 2006). Aujourd'hui son activité est tournée vers le traitement des données nuages de points de toutes sortes (lasergrammétrie, photogrammétrie...). Les thématiques de recherche principales sont la segmentation, la classification de nuages de points, la reconstruction de surface, le rendu de nuages de points ainsi que le SLAM. L'équipe dispose également de plateformes d'acquisition mobile comme le prototype L3D2 (voir figure 1) qui, équipé de lidar, permet de produire des scans laser sur de grandes distances. C'est cette proximité avec la production de données qui nous a poussés à mettre en avant l'utilisation de données variées dans les différentes parties de cette thèse.

Figure 1 – Prototype L3D2 du laboratoire de robotique de l'école des Mines.

Cette thèse a fait l'objet d'une collaboration rapprochée avec l' *"Institut de Calcul Intensif"* de l'école Centrale Nantes pour le chapitre 2. Ce laboratoire récemment créé autour du super-calculateur *Liger* rassemble des spécialistes du calcul intensif et du calcul numérique appliqué à la simulation d'équations aux dérivées partielles pour la physique. Un stage a également été encadré durant cette thèse autour de la

thématique du rendu temps réel de nuages de points (chapitre 1) et de son intégration dans les simulateurs pour la formation des conducteurs dans le milieu ferroviaire.

L'objet d'étude de cette thèse est le nuage de points. Un nuage de points est un ensemble non ordonné de points. Chaque point contient au minimum sa position 3D à laquelle peuvent s'ajouter d'autres informations comme l'intensité de retour du capteur ou une couleur. Le nuage de points 3D est la donnée obtenue par la majorité des méthodes de numérisation surfacique actuelles ; chaque point représente un échantillon de la surface mesurée. La figure 2 donne un exemple de nuage de points produit par scan laser mobile.

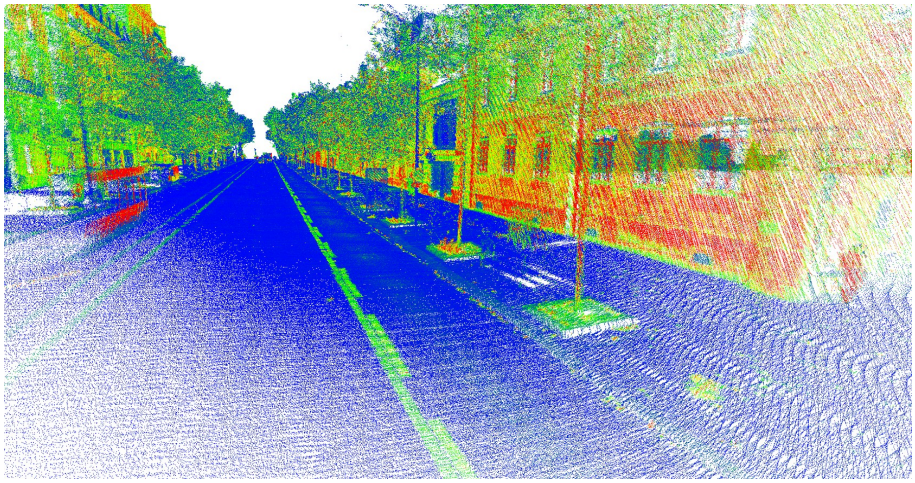


FIGURE 2 – Nuage de points de la façade de l'école des Mines, acquis dans le cadre de cette thèse avec le prototype L3D2.

Nous nous intéressons dans cette thèse à l'utilisation de nuages de points comme unique représentation explicite de surface. En effet, il est apparu tôt dans la communauté qu'une manière d'utiliser les données obtenues par numérisation 3D pour réaliser les traitements usuellement réalisés sur des géométries créées à la main, est d'avoir recours à la création d'un maillage. Cette conversion est communément appelée *reconstruction de surface* (un état de l'art est donné dans le chapitre 2). Hormis le fait que la reconstruction de surface pose de nombreux problèmes lorsqu'elle est appliquée à des données réelles, on peut également se questionner sur sa pertinence. En effet, il existe un certain nombre d'applications qui peuvent être directement appliquées à des nuages de points sans passer par une autre représentation intermédiaire. L'intérêt majeur de cette approche réside dans l'utilisation directe des données en sortie d'acquisition.

Cette approche représente le fil conducteur de cette thèse et de manière plus générale, elle constitue une philosophie de plus en plus répandue dans la littérature. A titre d'exemple, on peut citer *Point Morphology* (CALDERON et BOUBEKEUR, 2014) qui développe les outils classiques de la morphologie mathématique pour les appliquer directement à des nuages de points 3D.

L'utilisation directe de nuage de points pose néanmoins le problème de la création de nouveaux algorithmes spécifiques afin de tenir compte de leur caractère particulier. Nous porterons ainsi une attention particulière aux trois points suivants :

- **La polyvalence des données d'entrée :** Comme cela a été dit précédemment, les nuages de points peuvent provenir de différentes sources. Chacune présentant des particularités qui influent sur les caractéristiques du modèle 3D produit (bruit, répartition des données, occlusions...). Pour créer une méthode robuste, il est donc nécessaire d'en tenir compte.
- **La notion de surface :** La principale différence avec une surface représentée par un maillage est qu'un nuage de points ne présente pas de connectivité, ainsi un nuage de points ne définit pas une surface. Il en représente néanmoins un échantillonnage duquel la surface peut être déduite. Nous présenterons ainsi dans cette thèse, deux approches de définition de surface à partir de nuage de points sans passer par un maillage surfacique intermédiaire.
- **Gestion de la quantité de données :** Les nuages de points 3D produits de nos jours sont massifs. Cette quantité de données doit donc être gérée de manière spécifique par une organisation des données, que l'on désigne par le terme *"structuration"*. Cette problématique est par ailleurs exacerbée dans le cas du rendu temps réel de nuages de points, puisque le temps de traitement alloué à la création de chaque image est très limité.

Nous avons choisi de traiter dans cette thèse deux cas d'application qui sont le rendu et la simulation numérique. Ces deux applications, à priori différentes, font écho aux trois points cités plus haut. Elles donnent également une vue large du type de traitements directement applicables au nuage de points. Enfin, elles sont liées à des problématiques actuelles aussi bien sur le plan de la recherche que sur le plan industriel. Les applications directes des travaux présentés dans cette thèse peuvent être ainsi vues du point de vue applicatif, comme des alternatives peu coûteuses et de très haute qualité à la modélisation 3D manuelle. En effet, dans de nombreux cas, la production de contenu 3D est une tache complexe et chronophage, et l'utilisation d'objets ou de scènes numérisées permet d'y apporter une alternative rapide et fidèle à la réalité. A titre d'exemple, le rendu de nuages de points pourrait permettre d'intégrer des scènes virtuelles identiques à la réalité dans des environnements de simulation.

Cette thèse propose ainsi deux contributions :

- **Rendu de nuages de points bruts et massifs par opérateurs pyramidaux en espace image :** La première contribution proposée est une nouvelle méthode de rendu de nuages de points par opérateurs pyramidaux en espace image. Cette nouvelle méthode s'applique aussi bien à des nuages de points d'objets scannés, que de scènes complexes de taille arbitraire. Par ailleurs, elle s'applique à des nuage de points bruts et ne nécessite donc pas de précalculer

des attributs supplémentaires sur le nuage de points. La succession d'opérateurs en espace image permet alors de reconstruire en temps réel une surface et d'en estimer des normales, ce qui permet par la suite d'en obtenir un rendu par ombrage. De plus, l'utilisation d'opérateurs pyramidaux en espace image permet d'atteindre des fréquences d'affichage plus élevées d'un ordre de grandeur que l'état de l'art.

- **Simulation en mécanique des fluides en volumes immergés par reconstruction implicite étendue :** La deuxième contribution présentée est une nouvelle méthode de simulation numérique en mécanique des fluides en volumes immergés par reconstruction implicite étendue. La méthode proposée se base sur une nouvelle définition de surface implicite par moindres carrés glissants étendue à partir d'un nuage de points. Cette surface est alors utilisée pour définir les conditions aux limites d'un solveur Navier-Stokes par éléments finis en volumes immergés, qui est utilisé pour simuler un écoulement fluide autour de l'objet représenté par le nuage de points. Le solveur est interfacé à un mailleur adaptatif anisotrope qui permet de capturer simultanément la géométrie du nuage de points et l'écoulement à chaque pas de temps de la simulation.

Le plan de la thèse suit la description des deux contributions principales. Le chapitre 1 est consacré au rendu de nuages de points bruts et massifs. Le chapitre 2 traite de la simulation numérique en mécanique des fluides autour de nuages de points.

Chapitre 1

Rendu réaliste temps réel de nuages de points bruts

1.1 Introduction

1.1.1 Cadre général

Le rendu temps réel de nuages de points consiste à générer des vues artificielles d'un nuage de points 3D, ce qui permet à un utilisateur de se déplacer virtuellement dans un environnement numérisé. Le rendu de nuages de points peut être vu comme la première opération de traitement que l'on peut réaliser sur un nuage de points. Les intérêts du rendu de nuages de points sont multiples. On peut en effet le voir comme un outil de visualisation qui permet d'inspecter de manière interactive un nuage de points, dans le but par exemple de prendre des mesures ou de vérifier les résultats d'un algorithme. On peut le voir également comme une alternative au rendu polygonal pour intégrer du contenu numérisé dans une application 3D sans passer par une étape intermédiaire de reconstruction explicite de surface. Enfin, il s'agit d'un sujet de recherche ouvert qui présente de nombreuses applications industrielles présentes et futures. Nous présentons dans la suite de cette section deux domaines d'application du rendu de nuages de points : le patrimoine et les Systèmes d'Information Géographique (SIG) 3D.

Le patrimoine a été historiquement un des premiers domaines d'application, et reste toujours un domaine d'application très actif du rendu de nuages de points. L'application principale dans ce domaine réside dans la visualisation d'objets et de sites archéologiques. À titre d'exemple entre 1999 et 2000 l'équipe d'informatique graphique de l'université de Stanford *(Stanford University Computer Graphics Laboratory)* a mené le *Digital Michelangelo Project* (Marc Levoy, Pulli et al., 2000). Leur but était de numériser les principales statues du sculpteur Michel-Ange, et en particulier le David, à la galerie de l'Académie de Florence en Italie. Ce projet représentait en divers aspects un défi technologique, dans la mesure où il fallut adap-

ter toute la chaîne de production et de traitement de scans laser de l'époque pour scanner des statues de grande taille (7,5 m pour le David), ce que l'on peut voir sur la figure 1.1. Ce projet de grande envergure permit d'établir un jeu de données de référence qui est encore aujourd'hui largement utilisé dans la communauté. Les scans obtenus présentent pour certains plusieurs milliards de points 3D, ce qui représente plusieurs giga-octets de données. Encore aujourd'hui, les GPUs ne sont pas capables de traiter un tel nombre de primitives 3D de manière interactive. Et donc à fortiori, le matériel informatique de l'époque n'était pas capable de traiter une telle quantité de données directement. C'est justement ce qui a conduit à l'élaboration de l'algorithme de rendu QSplat (Szymon Rusinkiewicz et Marc Levoy, 2000), que nous détaillerons dans la section 1.2.

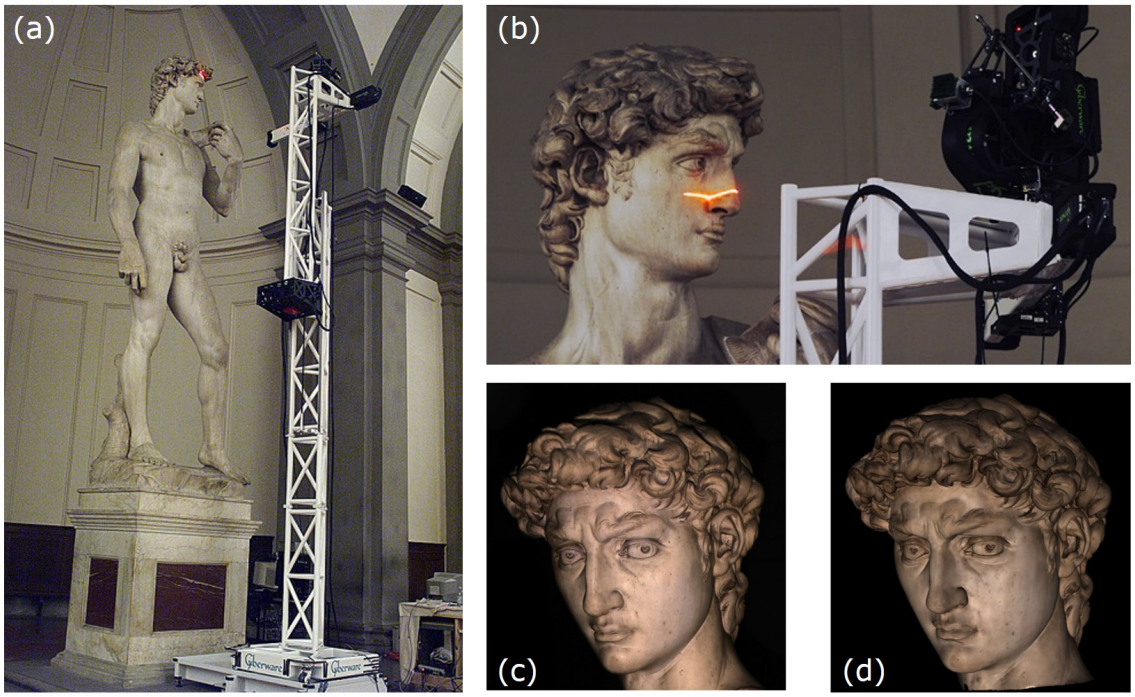


Figure 1.1 – (a) Structure employée pour le scan du David de Michel-Ange. (b) Zoom sur le scanner 3D Cyberware monté au sommet de la structure métallique. (c) Photographie du David. (d) Rendu obtenu à partir du modèle scanné. (Source : Digital Michelangelo Project (Marc Levoy, Pulli et al., 2000)).

Plus récemment le projet Terapoints mené par l'institut de recherche en informatique graphique de l'Université Technique de Vienne (*TU Wien*) a conduit à la création du logiciel Potree (Schuetz, 2016) (voir figure 1.2). Il s'agit d'un logiciel de visualisation de nuages de points massifs en ligne. Il implémente une version en client-serveur de l'algorithme *Instant Points* (Michael Wimmer et Scheiblauer, 2006) qui permet de visualiser en ligne des modèles contenant plusieurs milliards de points, avec notamment un record à 640 milliards de points pour un nuage de points complet des Pays-Bas (Martinez-Rubi, O. et al., 2015). En plus de son utilisation

pour la visualisation de sites archéologiques, ce logiciel montre l'intérêt du rendu de nuages de points massifs dans le domaine des SIG. L'application du rendu de nuages de points réside ici dans la visualisation pour la gestion de grands réseaux ou de larges infrastructures, par exemple pour les villes ou les collectivités. Cette application commence également à apparaitre dans les SIG plus traditionnels sous la forme de plugins commerciaux. On peut citer également le logiciel *CloudCompare* qui fait référence comme logiciel libre de visualisation et de traitement de nuages de points.

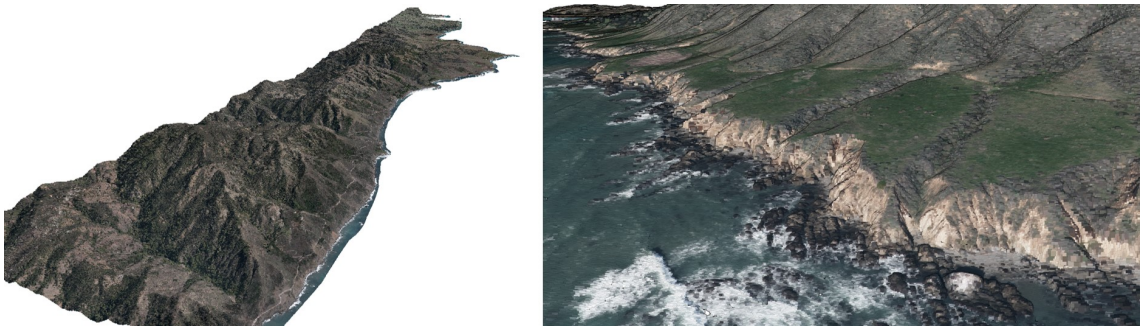


FIGURE 1.2 – Vues d'un nuage de points de la côte de la ville de San Simeon en Californie, obtenues à l'aide du logiciel Potree. Le nuage de points contient 17.7 milliards de points. Les données sont issues du site *Open Topography*.

1.1.2 Méthodes de rendu et applications

Du point de vue du résultat obtenu, on peut classifier les méthodes de rendu temps réel de nuage de points en 3 familles, illustrées sur la figure 1.3 :

- **Rendu expressif :** Les méthodes de rendu expressif [1] permettent de mettre en évidence certains attributs d'un modèle 3D, comme par exemple sa géométrie, sans nécessairement rechercher à produire un effet réaliste.
- **Rendu basé image :** Le rendu basé image est un domaine de recherche qui s'intéresse à la génération de nouvelles vues d'un objet ou d'une scène à partir d'une collection d'images. Le rendu de nuages de points colorisés peut en être rapproché du point de vue des résultats obtenus. L'éclairage de la scène est alors fixe ; il correspond à celui capturé lors de l'acquisition.
- **Rendu par ombrage :** Le rendu par ombrage consiste à synthétiser la réponse lumineuse d'une scène 3D à partir de sources de lumière virtuelles. Usuellement, lorsque l'on s'intéresse aux méthodes temps réel, cette réponse est approximée par des méthodes "physiquement plausibles" [2].

1. Le terme anglophone associé est NPR pour *Non Photorealistic Rendering*.
2. Ce qui est à nuancer avec l'apparition de méthodes d'illumination globale temps réel (LAURENT et al., 2016)

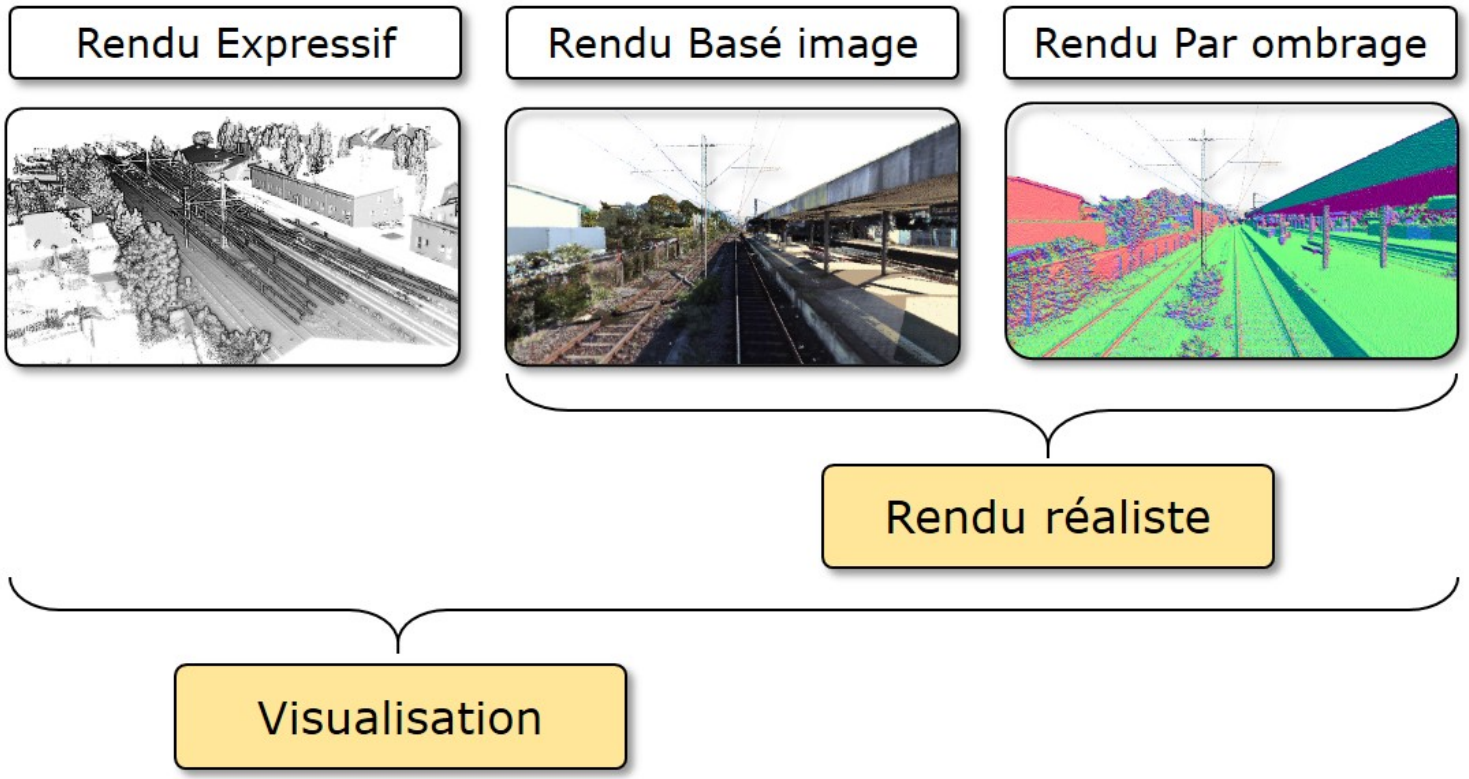


FIGURE 1.3 – Différentes méthodes de rendu de nuages de points et leurs applications. Le rendu expressif a été obtenu par l'algorithme d'*Eye Dome Lighting* (BOUCHENY, 2009) implémenté dans le logiciel CloudCompare. Le rendu par ombrage est représenté par une carte des normales (*normal map*) car c'est notamment celle-ci qui permet de calculer le rendu final à partir de sources de lumière virtuelles.

D'un point de vue applicatif, on peut alors diviser ces 3 catégories en 2 cas d'usage, également illustrés sur la figure 1.3 :

- **La visualisation :** La visualisation rassemble les applications liées à l'inspection d'un nuage de points dans le but d'en percevoir ou de mesurer les détails aussi bien liés aux attributs qu'il contient qu'à sa géométrie. Les deux applications présentées dans la sous-section 1.1.1 en font partie.
- **Le rendu réaliste :** Le rendu réaliste rassemble quant à lui les applications qui s'intéressent à la restitution d'objet ou d'environnements représentés par des nuages de points dans le but d'y immerger virtuellement un utilisateur. Parmi ces applications, on peut citer les moteurs graphiques des environnements de simulation ou de jeux vidéo. Un exemple d'application pour la simulation de conduite ferroviaire sera donné en conclusion de ce chapitre (figure 1.55).

Ces trois familles de méthodes posent des challenges algorithmiques notamment lorsqu'il s'agit de traiter des scènes réelles complexes. Dans la suite de ce chapitre, nous choisissons de nous placer dans le cadre du rendu basé image et du rendu par ombrage. Comme illustré sur la figure 1.3, ces deux familles de méthodes s'appliquent au rendu réaliste, ce que ne permet pas le rendu expressif. On pourra également noter que ces méthodes peuvent toutefois être utilisées à des fins de visualisation.

1.1.3 Objectifs et contraintes

1.1.3.1 Objectifs

Nous présentons dans ce chapitre une nouvelle méthode de rendu réaliste temps réel de nuages de points bruts et massifs qui satisfait les trois points suivants.

Efficacité L'efficacité de la méthode est mesurée au temps nécessaire pour produire chaque image. Il est communément admis que pour qu'une méthode soit considérée temps réel cette durée ne doit pas dépasser 33 ms, ce qui correspond à une fréquence d'affichage de 30 fps[3]. Cependant certaines applications peuvent nécessiter des fréquences d'affichage plus élevées, typiquement 60 fps (soit 16 ms) pour les simulateurs ou 120 fps (soit 8 ms) pour les casques de réalité virtuelle. C'est pour cette raison que nous choisissons de fixer le budget alloué par image à **10 ms**. Nous montrerons par la suite que pour satisfaire cette limite, il est nécessaire, d'une part de mettre en œuvre une méthode de structuration du nuage de points, et d'autre part de concevoir des algorithmes de traitement efficaces.

Qualité de rendu La méthode de rendu de nuages de points la plus basique consiste à rendre[4] chaque point par un pixel dans l'espace projectif de la caméra.

3. Pour *Frames Per Second* qui signifie "images par seconde".
4. Le verbe rendre, de l'anglais *render*, désigne dans ce chapitre la synthèse d'une image à partir d'un modèle 3D.

Cette méthode produit des artefacts illustrés sur la figure 1.4. Ces 3 artefacts sont dû au fait qu'un point ne possède pas d'extension spatiale. L'effet de surface "pleine" observable pour les parties du modèle qui sont loin du point de vue de la caméra est obtenu lorsque la distance entre deux points est inférieure à 1 pixel. Lorsque celle-ci est supérieure à 1 pixel on peut voir à travers le modèle ce qui nuit à la compréhension visuelle de la scène. La qualité de rendu est alors jugée relativement à la correction de ces artefacts : retrait des parties cachées, remplissage et antialiasing.

FIGURE 1.4 – Principaux artefacts observables en rendu de nuages de points. (1) Proche du point de vue, la densité du nuage de points projeté sur l'image est trop faible. (2) On peut voir à travers le modèle. (3) Aliasing.

Rendu par ombrage Le rendu par ombrage nécessite de calculer une surface pleine et d'en estimer des normales. La recherche de rendu par ombrage couplée à l'utilisation de nuages de points bruts (contrainte qui est détaillée plus bas), représente un défi. Nous montrerons en effet par une étude de l'état de l'art que très peu de méthodes y parviennent. La plupart des méthodes proposant un rendu par ombrage sur des nuages de points utilisent des attributs supplémentaires (normales ou tenseur géométrique local) précalculés sur chaque point du nuage.

1.1.3.2 Contraintes

Les contraintes suivantes délimitent le cadre de réflexion technique de la méthode proposée.

Utilisation de nuages de points bruts Le terme *"brut"* désigne un nuage de points qui contient au minimum la liste des coordonnées 3D des points, ce qui représente l'information qui est obtenue directement en sortie du processus de numérisation. Ainsi, même s'il peut contenir d'autres données, les méthodes développées dans ce chapitre n'utilisent que cette information. L'intérêt principal de développer des méthodes de rendu de nuages de points bruts réside dans la possibilité de les appliquer à tout type de données. En effet, nous montrerons par une étude de l'état de l'art que de nombreuses méthodes se basent sur l'utilisation d'informations supplémentaires comme les normales ou les densités locales du nuage de points. Ces informations ne sont pas produites directement par le processus d'acquisition et nécessitent donc d'être précalculées sur l'ensemble du nuage de points par un processus couteux en temps de calcul et qui implique nécessairement un choix lié aux paramètres de la méthode employée pour le calcul.

Variété des données utilisées Nous nous intéresserons à traiter une variété large de nuages de points. En effet, selon le type d'acquisition utilisée, le nuage de points obtenu peut être de nature très différente en termes de bruit, densité, anisotropie de l'échantillonnage et structures géométriques qu'il contient. En particulier, les nuages de points des scènes sont plus complexes que les nuages de points d'objets. La figure 1.5 présente quelques exemples de nuages de points utilisés dans ce chapitre. Par ailleurs, selon la méthode d'acquisition, les niveaux de bruits peuvent varier de quelques millimètres à plusieurs centimètres (figure 1.6). A la différence de beaucoup d'articles de l'état de l'art, nous accorderons donc une importance particulière à développer des méthodes qui permettent de traiter des scènes nuages de points.

Nuages de points massifs Nous admettrons qu'un nuage de points est qualifié de massif lorsque sa taille dépasse 100 millions de points. Il s'agit là de la limite au-delà de laquelle, pour un rendu par force brute sur un GPU haut de gamme récent, la fréquence d'affichage descend en dessous de 30 fps. Cette valeur est obtenue en considérant que la capacité de traitement d'un GPU est de l'ordre de 3 milliards de points par seconde. Cette valeur est extraite du tableau 1.1 qui sera détaillé dans la section 1.3. En revanche, étant donné un point de vue, seule une partie de l'ensemble des points du nuage contribue à la production de l'image finale. Ainsi, pour traiter des nuages de points massifs, il est nécessaire de développer des méthodes d'organisation des données que l'on appelle "structuration" afin de sélectionner les points suffisants au rendu à chaque image et en temps réel.

Exploration immersive L'exploration immersive désigne ici la génération de vues *"de l'intérieur"* dans les scènes de nuages de points ou bien des vues *"de près"* pour les objets. Ce type d'exploration nécessite de développer des méthodes de rendu robustes aux scènes qui présentent de fortes différences de profondeurs. C'est le cas par exemple de la vue générée sur la figure 1.4 où l'on observe aussi bien des points

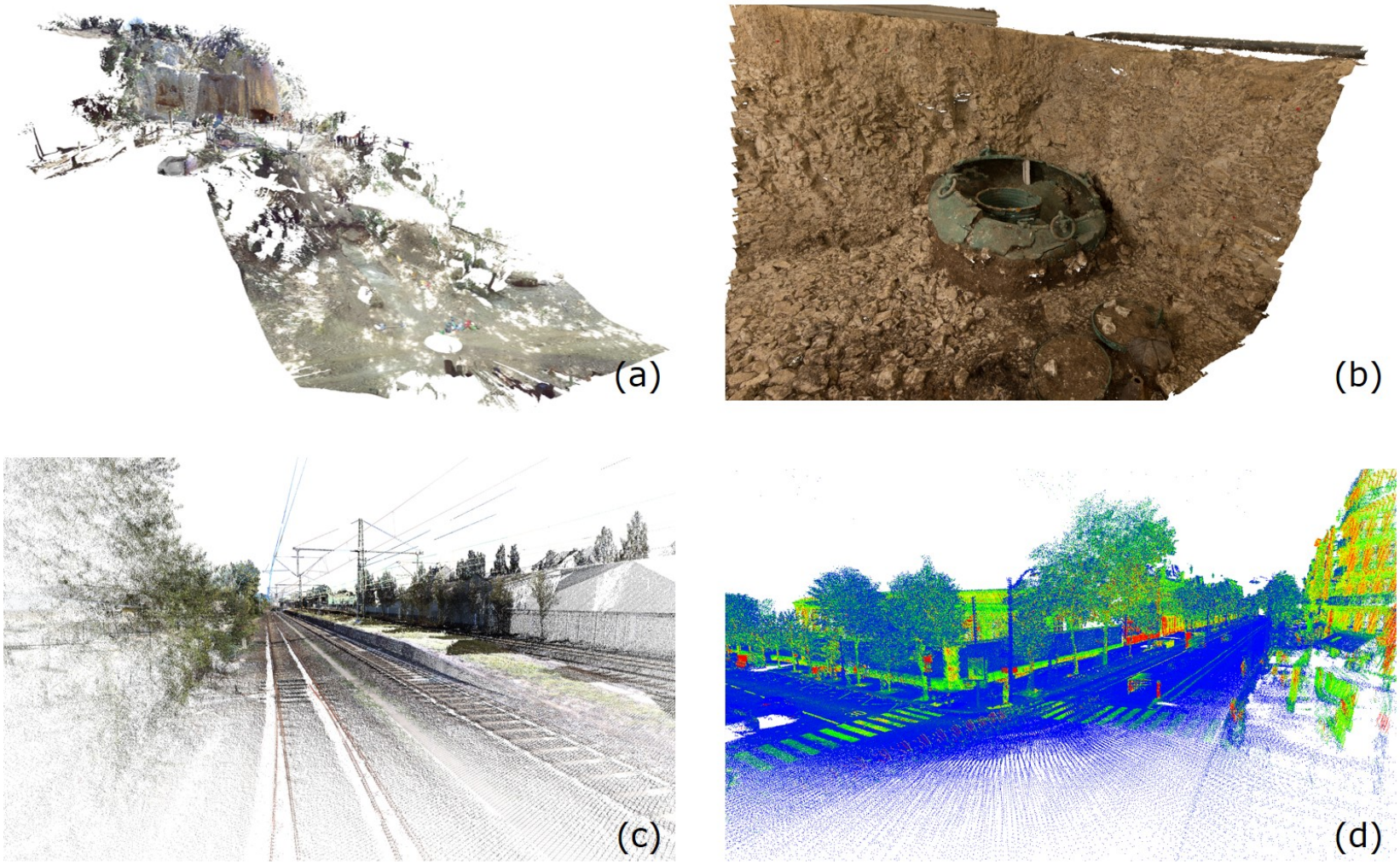


FIGURE 1.5 – Exemples de nuages de points traités dans ce chapitre. (a) Scan fixe d'une tombe étrusque. (b) Nuage obtenu par photogrammétrie d'un chaudron retrouvé dans une tombe Celte. (c) Scan mobile d'une voie de chemin de fer avec un capteur lidar mono-fibre RIEGL (précision millimétrique). (b) Scan mobile d'une rue Parisienne réalisé dans le cadre de cette thèse à l'aide d'un capteur lidar multifibres Velodyne HDL32E (précision centimétrique).

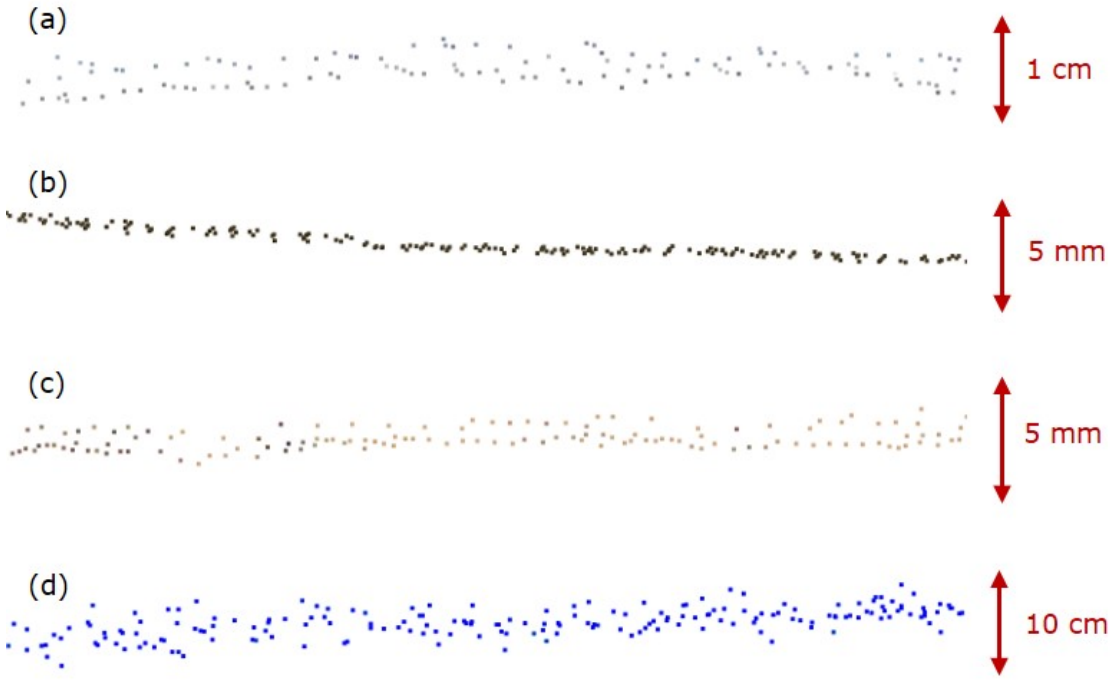


FIGURE 1.6 – Illustration des différents niveaux de bruit des nuages de points de la figure 1.5 par coupes transversales de zones planes.

proches à quelques centimètres, que des points à plus de 500 mètres du point de vue virtuel.

1.1.4 Contributions et plan

Ce chapitre présente la première contribution de cette thèse. Il présente ainsi une nouvelle méthode de rendu de nuages de points bruts par opérateurs pyramidaux en espace image. Cette méthode permet d'atteindre des fréquences d'affichage plus élevées que les méthodes de l'état de l'art tout en proposant un rendu de haute qualité dont le résultat est adapté à l'utilisation ultérieure d'algorithmes de rendu par ombrage. Par ailleurs, la méthode proposée gère des scènes variées de taille arbitraire. Elle s'applique à des nuages de points bruts et ne nécessite donc pas d'attributs autres que la position 3D des points. Elle gère également les scènes avec de fortes différences de profondeur, garantissant ainsi une qualité de rendu constante quelque-soit le point de vue virtuel de la caméra employé, notamment en vue immersive.

Après un état de l'art général autour du rendu temps réel de nuages de points massifs dans la section 1.2, nous traitons le problème de la gestion de la quantité de données dans la section 1.3. Enfin, la section 1.4 est consacrée à la méthode de rendu en espace image.

1.2 État de l'art

1.2.1 Structuration de nuages de points pour le rendu

Un premier problème lié au rendu de nuages de points réside dans la gestion de la quantité de données à traiter. Ce problème est exacerbé par l'utilisation de points à la place de facettes car l'impression de surface est obtenue par la densité de points, ce qui implique que les modèles contiennent généralement plus de primitives géométriques. Une solution à ce problème consiste à structurer le nuage de points.

La structuration de nuages de points consiste à organiser efficacement en mémoire les informations portées par un nuage de points afin d'en faciliter le traitement. C'est un sujet de recherche vaste qui n'est, ni limité au rendu de nuages de points, ni limité aux données nuages de points. En effet, le problème de structuration se pose lorsque l'on désire calculer des voisinages dans un nuage de points. Pour cela, on peut utiliser une structure de *kd-tree* (Moore, 1991). Celle-ci permet de passer d'une complexité de l'algorithme de recherche naïf en n (où n est le nombre de points du nuage) à une complexité en $log(n)$, au prix d'un prétraitement de complexité $nlog(n)$ mais qui n'est réalisé qu'une seule fois si le nuage est statique.

L'*octree* est une autre structure plus générale qui est largement utilisée dans la communauté. Il s'agit d'un arbre de partition spatiale où chaque nœud est divisé régulièrement en 8 fils selon les trois directions de l'espace. La figure 1.7 illustre ce procédé de construction en 2D (l'arbre obtenu est appelé *quadtree*). Il existe de nombreuses variantes d'*octree* qui présentent chacune des spécificités selon les contraintes liées à l'utilisation finale de la structure (exemple : faible empreinte mémoire, mise à jour fréquente...). (Elseberg et al., 2013) présentent des exemples d'algorithmes classiques accélérés par une variante d'*octree*.

(Szymon Rusinkiewicz et Marc Levoy, 2000) ont été les premiers à proposer un algorithme permettant de visualiser des nuages de points de très grande taille. Il est intéressant de noter que leur but premier était en réalité de rendre des maillages de très grande taille. Pour cela ils synthétisent l'information surfacique locale de chaque nœud du maillage par un surfel [5] discoïde, qui peut être vu comme un point 3D augmenté d'informations supplémentaires : normale et rayon. Leur motivation principale réside dans le fait que la gestion de niveaux de détails est intrinsèquement plus aisée avec un nuage de points qu'avec un maillage, puisqu'elle peut être réalisée par un sous-échantillonnage adéquat.

Ils utilisent alors une hiérarchie de sphères englobantes qu'ils calculent sur l'ensemble du nuage de points. Cette structure permet en plus de compresser l'information portée par chaque point sur 6 octets à l'aide d'un encodage récursif. Ainsi la position réelle d'un nœud est calculée à la volée lors du parcours de l'arbre. La figure 1.8 montre que la structure de données est stockée sur le disque par un parcours en

5. Élément surfacique, aussi connu sous le nom de splat, composé d'un disque (ou d'une ellipse orientée) et d'une normale.

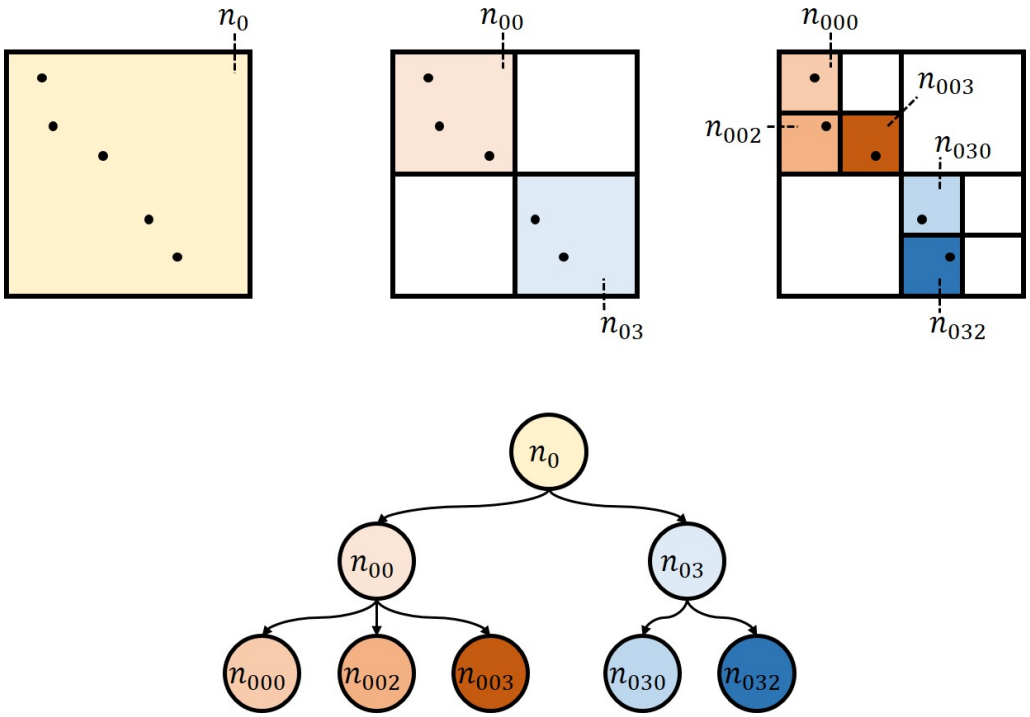


FIGURE 1.7 – Exemple de *quadtree*. (En haut) Représentation spatiale des trois niveaux successifs de l'arbre. Les cases en couleur représentent les nœuds de l'arbre. Les cases en blanc représentent les nœuds vides (qui ne contiennent pas de points). (En bas) Représentation arborescente de l'arbre.

largeur[6]. La gestion du chargement depuis le disque dur pour des nuages de points qui ne rentrent pas en RAM est assuré par l'utilisation de la mémoire virtuelle gérée par le système d'exploitation (*memory mapped files*).

La structure de données permet alors, par un algorithme de rendu très compact, de rendre le nuage de points en adaptant la résolution à la distance avec le point de vue tout en occultant les parties hors du cône de vue (*furstum culling*). L'adaptation du niveau de détail est réalisée de manière simple en arrêtant la récursion dans l'arbre lorsque la taille d'un nœud projeté à l'écran est inférieure à une certaine taille en pixels. Cette dernière constitue un paramètre qui peut être ajusté pour contrôler le compromis entre vitesse et qualité d'affichage.

On peut noter que l'utilisation des normales dans cet algorithme n'est pas nécessaire, de même que la structure de sphères englobantes qui peut être remplacée par n'importe quelle structure de volumes englobants (notamment un *octree*). Cette publication séminale reste aujourd'hui une référence dans le domaine. Son principal inconvénient est que le GPU est largement sous-exploité dans la mesure où pour chaque nouvelle image il est nécessaire de construire la liste des points nécessaires au rendu et de l'envoyer au processeur graphique. A noter que cette remarque est à

6. Traduction de l'anglais de *Breadth First Order (BFO)* qui s'oppose au "Parcours en profondeur" pour *Depth First Oder (DFO)*

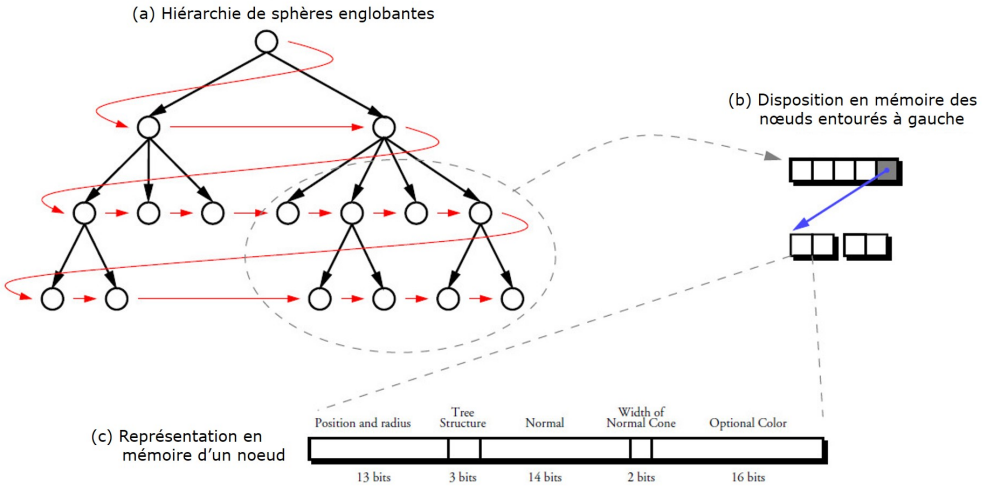


FIGURE 1.8 – Représentation en mémoire de la structure de données de QSplat. (a) L'arbre est stocké par un parcours en largeur. (b) Le lien entre un groupe de nœuds parents et leurs nœuds fils est établi par un seul pointeur. Ce pointeur n'est présent que si tous les nœuds parents sont des feuilles de l'arbre. (c) Un nœud occupe 48 bits, grâce à un processus de quantification. (Figure reproduite et traduite de (Szymon RUSINKIEWICZ et Marc LEVOY, 2000))

replacer dans le contexte des versions anciennes des API 3D où le pipeline graphique n'était pas programmable.

(DACHSBACHER et al., 2003) proposent l'algorithme *Sequential Point Trees (SPT)* qui sélectionne les points à rendre directement sur le GPU, par opposition à *QSplat* où la sélection est réalisée sur le CPU. Leur idée est de pousser à l'extrême la réduction des points sélectionnés par niveaux de détails en intégrant un critère de sélection géométrique. Ainsi la récursion est stoppée lorsque le splat du nœud interne représente suffisamment ses fils, ce qui permet par exemple d'arrêter la récursion plus tôt pour les régions planaires. Cette structure de données est représentée sur la figure 1.9.

SPT ne permet néanmoins pas d'éliminer les objets hors du cône de vue (*frustum culling*). De plus, la sélection du LOD sur le GPU présentée dans l'article présente un surcoût en calcul car elle est réalisée après un calcul dans le *geometry shader*.

SPT et *QSplat* sont donc deux méthodes antagonistes qui ont donné suite à de nombreuses autres méthodes qui ont tenté de tirer parti des bénéfices de chacune. C'est le cas de (Michael WIMMER et SCHEIBLAUER, 2006) avec *Instant Points*, implémentée dans le logiciel Potree, présenté en introduction. *Instant Points* repose sur deux innovations : la première consiste à simplifier la structure de données *SPT* pour permettre la sélection uniquement sur un critère de distance au point de vue, ce qui est illustré sur la figure 1.9. Ainsi la sélection du niveau de détail est plus grossière qu'avec *SPT* mais elle reste moins coûteuse en temps de calcul puisqu'elle se factorise en un unique calcul sur le CPU du nombre de points à afficher. La deuxième innovation est d'ajouter à cette structure une structure d'arbres imbriqués,

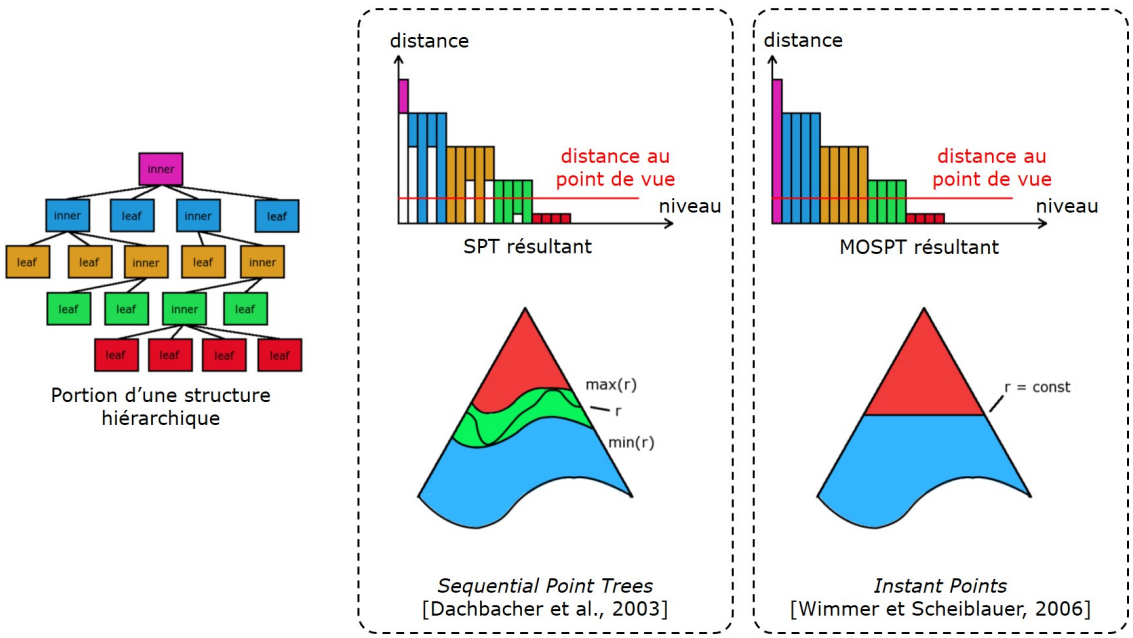


FIGURE 1.9 – Comparaison entre les structures de données *SPT* et *Instant Points*. *SPT* sélectionne chaque point sur un critère géométrique. *Instant Points* sélectionne les points sur le même critère basé uniquement sur la distance au point de vue. (Figure reproduite, modifiée et traduite de (Michael WIMMER et SCHEIBLAUER, 2006))

représentée sur la figure 1.10. Cette structure permet de réaliser le retrait des parties hors du cône de vue.

On notera que de la même manière que la sélection du LOD, le retrait des parties hors du cône de vue est approximée par rapport à *QSplat*. Cette stratégie permet néanmoins de charger des parties entières du jeu de données sur le GPU par un système de cache et donc d'éviter de mettre à jour l'intégralité de la liste des points à afficher à chaque nouvelle image. On notera également que contrairement aux autres méthodes *Instant Points* décrit explicitement l'organisation des fichiers pour la gestion *out-of-core* des jeux de données qui ne rentrent pas en RAM.

Instant Points représente aujourd'hui la méthode de référence pour la structuration de nuages de points pour le rendu. Elle présente néanmoins un inconvénient qui est qu'elle doit être recalculée à chaque fois que le nuage de points est modifié. Nous proposons une méthode dérivée qui apporte une solution simple à ce problème dans la section 1.3.

D'autres méthodes ont suivi *Instant Points*. Parmi elles, on distingue (ELSEBERG et al., 2013) qui présente une structure d'*octree* unifiée pour le rendu et le traitement, sans proposer de gestion explicite du caractère *out-of-core*. (WAND et al., 2008) proposent une structure de données intéressante qui permet en plus d'éditer des nuages de points massifs de manière interactive. Leur structure implique néanmoins des contraintes fortes sur son intégration dans une chaine de traitements complexe.

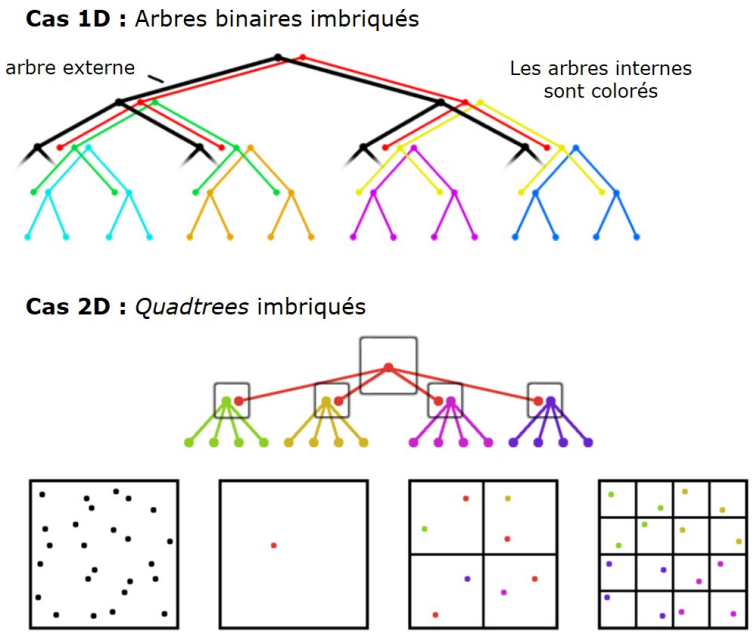


FIGURE 1.10 – Schématisation de la structure d'arbres imbriqués utilisée pour le retrait des parties hors du cône de vue dans *Instant Points*. (Figure reproduite, modifiée et traduite de (Michael WIMMER et SCHEIBLAUER, 2006))

À titre de remarque, plus récemment de nouvelles méthodes de structuration sont apparues dans la communauté. A notre connaissance, elles n'ont toutefois pas encore été appliquées au rendu de nuage de points. (KÄMPE et al., 2013) ont introduit une méthode de compression basée sur l'utilisation de graphes acycliques pour éliminer les redondances dans un *octree*. Cette méthode permet d'atteindre des taux de compression sans pertes impressionnant permettant le stockage sur GPU de jeux de données massifs. Elle trouve son application dans le rendu par lancer de rayon. (GOBBETTI et MARTON, 2005) et (CRASSIN, NEYRET, LEFEBVRE et al., 2009) ont introduit deux méthodes similaires permettant également d'accélérer le *ray-tracing* de modèles 3D par une structure de données partiellement stockée sur GPU. Ces méthodes présentent un intérêt supplémentaire, par rapport aux méthodes présentées précédemment, dans leur capacité à gérer les parties cachées par occlusion (*occlusion culling*).

1.2.2 Rendu de nuages de points

Le rendu de nuage de points trouve son origine dans le rendu d'effets volumétriques comme la fumée ou le feu. Les premiers travaux qui s'intéressent à l'utilisation de points pour la modélisation et le rendu d'objets solides sont : (CSURI et al., 1979) qui évoquent l'idée d'utiliser des points comme primitive de rendu, et (Marc LEVOY et WHITTED, 1985) traitent le cas particulier des surfaces différentiables.

Nous présentons dans cette section les différentes méthodes de rendu de nuages de points. La figure 1.11 illustre quatre catégories de méthodes de rendu de nuages de points, réparties selon le type de données qu'elles prennent en entrée. Le rendu basé image[7] consiste à générer des vues artificielles à partir d'une collection de photos sans passer par une reconstruction de surface. Bien que loin, d'un point de vue technique, des méthodes présentées ici, ce type de rendu en est proche d'un point de vue du résultat. En particulier dans le cas du rendu d'un nuage de points colorisé à partir d'une collection de photos, ou parce qu'il a été produit par photogrammétrie. C'est pourquoi nous ne détaillerons pas ces méthodes dans cette partie. Les trois autres catégories : rendu polygonal, rendu points avec attributs et rendu points bruts, sont détaillées par la suite.

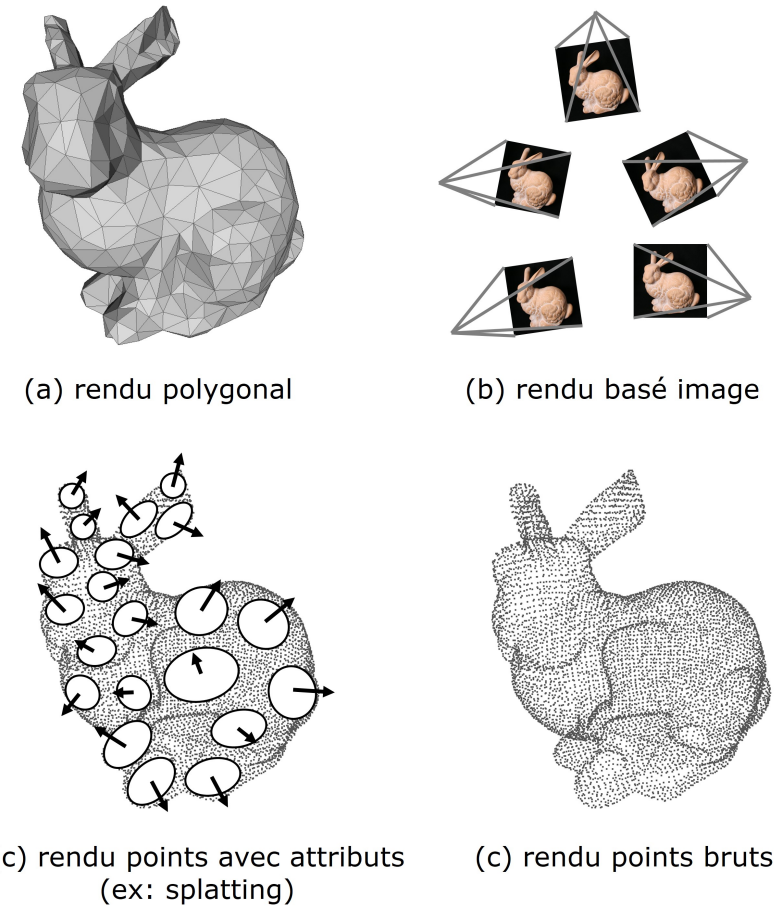


FIGURE 1.11 – Différentes méthodes de rendu par points, illustrées par les données qu'elles prennent en entrée.

7. *Image Based Rendering (IBR)* en anglais)

1.2.2.1 Rendu par reconstruction de surface

Une première approche pour améliorer l'aspect visuel du rendu d'un nuage de points consiste à construire un maillage surfacique à partir celui-ci. Ce processus est désigné par le terme *reconstruction de surface*. (Kazhdan, Bolitho et al., 2006 ; Kazhdan et Hoppe, 2013) en sont deux exemples. Un état de l'art plus complet est donné dans le chapitre 2.

Bien que ces méthodes permettent de se placer directement dans le cadre plus classique de rendu de modèles polygonaux, elles ne permettent pas d'apporter une solution satisfaisante au problème du rendu de nuages de points. Ces méthodes impliquent généralement un prétraitement coûteux. De plus, d'un point de vue théorique, elles sont basées sur le postulat que le nuage de points est localement surfacique, ce qui n'est pas nécessairement le cas lorsqu'il s'agit de nuages de points réels ; ceux-ci présentent souvent des régions volumiques (comme les arbres) ou linéiques (comme les grillages). De plus ces méthodes génèrent souvent des géométries très détaillées ce qui n'apporte pas plus d'information que dans le nuage de points initial.

Parmi les méthodes de reconstructions de surface, (Boubekeur et al., 2005) présentent une approche qui consiste à générer une triangulation grossière qui est enrichie par des textures de normales (*normal map*) ainsi que des textures de déplacement (*displacement map*). Cette approche est plus cohérente avec l'utilisation des GPU par les moteurs de rendu. Elle reste néanmoins peu adaptée au rendu de scènes complexes pour les mêmes raisons que les méthodes basées sur une reconstruction de surface explicite.

Les méthodes de reconstruction de surface implicites peuvent également être utilisées pour rendre des nuages de points. Celles-ci sont souvent associées au rendu non temps réel par ray-tracing. (Adamson et Alexa, 2003) proposent une méthode pour calculer l'intersection entre un rayon et une surface définie par points (*PSS* pour *Point Set Surface*). Ici la surface est définie par l'ensemble des points fixes d'un opérateur de projection de $\mathbb{R}^3 \rightarrow \mathbb{R}^3$ qui est calculé à la volée ce qui ne permet pas d'envisager l'utilisation de cette méthode en rendu temps réel. Une autre approche consiste à calculer une représentation implicite par fonction scalaire signée sur une grille dense ou sur une structure de données creuse. Nous présentons une implémentation de la méthode de (Knoll et al., 2006) dans le chapitre suivant. Bien que l'utilisation du *ray-tracing* permette d'accéder à un ensemble très large d'effets visuels elle reste limitée pour le rendu temps-réel de nuages de points.

En conclusion, la reconstruction de surface explicite ou implicite n'est pas adaptée à notre problématique pour les trois raisons suivantes :

- **Prétraitement lourd :** les méthodes de reconstruction de surface nécessitent des prétraitements lourds sur les nuages de points. Elles ne sont donc pas envisageables pour le rendu de données brutes.

- **Robustesse vis à vis de nuages complexes :** les algorithmes de reconstruction de surface ne sont pas robustes aux artefacts ainsi qu'aux structures géométriques complexes que l'on peut trouver dans les relevés laser de scènes.
- **Redondance d'information :** tout comme le nuage de points, le maillage est une représentation explicite de surface. L'information de surface décrite par le maillage est déjà présente dans le nuage initial, ainsi le maillage peut être vu uniquement comme une représentation intermédiaire. D'où l'idée de se passer de ce dernier pour rendre directement un nuage de points.

1.2.2.2 Rendu par grossissement des points

Le rendu de nuages de points par grossissement des points consiste à rastériser chaque point par un carré ou un disque de taille fixée (en pixels) dans l'espace image. Cette approche est utilisée dans beaucoup de logiciels de visualisation de nuages de points comme *CloudCompare*. La figure 1.12 donne un exemple de ce type de rendu. Cette méthode simple permet de remplir l'espace entre les points, néanmoins l'effet produit est grossier.

FIGURE 1.12 – Rendu par grossissement de la taille des points. La taille des points est fixée à 15 pixels pour une résolution de 1280×748

Cette méthode peut être implémentée dans le *vertex shader* de la manière suivante :

```
1 gl_PointSize = point_size;
```

Cette implémentation est de plus très gourmande car elle instancie (point_size $*$ point_size) *fragment units* par point. Ce qui engendre une baisse notable de la

fréquence d'affichage. Typiquement, pour la vue présentée sur la figure 1.12, on passe de 250 fps pour une taille de 1 pixel à 27 fps pour 15 pixels (configuration : CPU : i7-4770 3.4 GHz, GPU : Nvidia GeForce GTX 1080 Ti).

Une autre approche consiste à rendre les points par des carrés ou des disques de taille fixée (en mètres) dans le repère de la scène, engendrant ainsi par effet de perspective des carrés ou des disques de taille variable dans l'espace image. Cette méthode est décrite sur le point (a) de la figure 1.13. C'est une des approches retenues par (Devaux et Brédif, 2016).

La qualité du rendu obtenu est meilleure qu'avec une taille de points fixée. De plus elle nécessite d'instancier moins de *fragment units* donc elle est plus efficace. Elle produit néanmoins toujours des artefacts grossiers et nécessite d'avoir une scène avec peu de variations de densité.

L'approche qui produit les meilleurs résultats par grossissement de points est celle de (Schütz et M. Wimmer, 2015) ; elle consiste à rendre les points par des paraboloïdes dans l'axe de la caméra, ce qui est illustré sur la figure 1.13. De cette manière l'effet grossier de superposition des points que l'on peut observer par un rendu par carrés ou disques plats est supprimé. Comme l'expliquent les auteurs, on peut rapprocher leur méthode de la création de cellules de Voronoï autour des points.

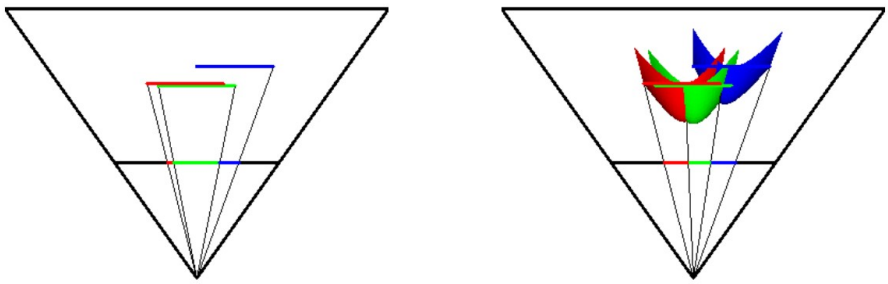


Figure 1.13 – Rendu par carrés de taille fixée dans le repère de la scène. (a) Tous les pixels d'un même point ont la même profondeur. (b) Variation de la profondeur des pixels en fonction de la distance au centre du carré. On remarque que les points ne se chevauchent plus sur le plan de la caméra sur le schéma (b). (Figure reproduite et traduite de (Schütz et M. Wimmer, 2015))

Les résultats obtenus par la méthode sont présentés sur la figure 1.14. Les résultats sont comparables en termes de lisibilité à ceux obtenus par les méthodes de splatting qui sont présentées dans le paragraphe suivant. Cependant, la faible qualité de la surface résultante ne permet donc pas de calculer un rendu par ombrage. A noter que (Xu et al., 2004) utilisent déjà une méthode similaire dans leur phase de retrait des parties cachées. Celle-ci est cependant moins aboutie que la méthode proposée par (Schütz et M. Wimmer, 2015).

En conclusion, les méthodes de rendu de nuages de points par grossissement des points sont intéressantes pour la visualisation car elles sont simples à implémenter.

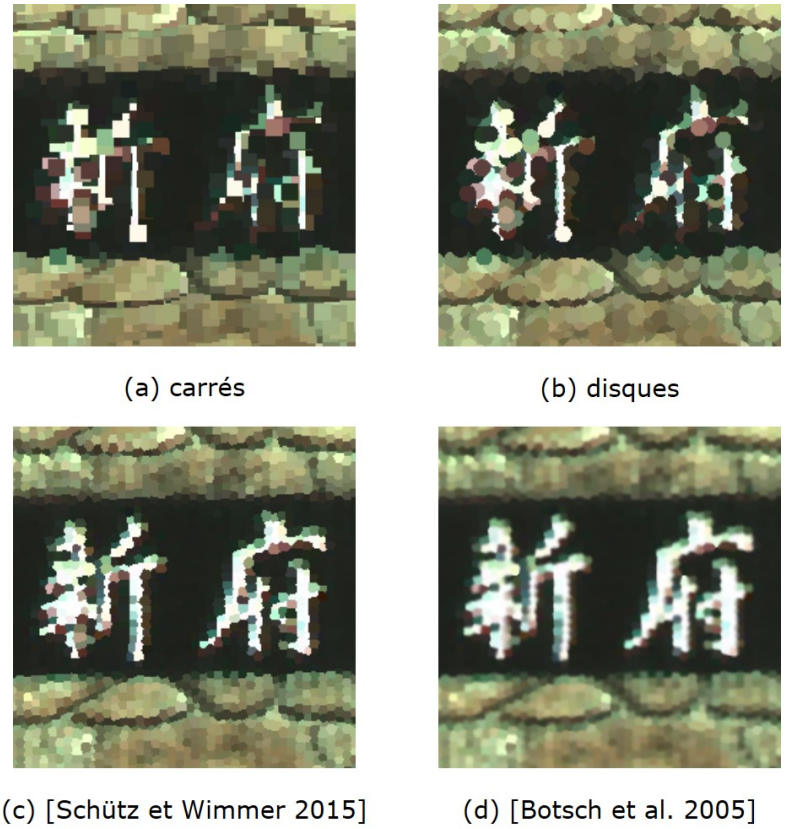


FIGURE 1.14 – Résultats obtenus par (SCHÜTZ et M. WIMMER, 2015) comparés au rendu (a) par carrés, (b) par disques et (d) par (M. BOTSCH et al., 2005). La méthode améliore la lisibilité du texte dans le nuage de points. (Figure reproduite et traduite de (SCHÜTZ et M. WIMMER, 2015))

Elles ne sont néanmoins pas adaptées à notre problématique du fait de la faible qualité de la surface produite.

1.2.2.3 Splatting

Le splatting consiste à rendre chaque point par un petit élément de surface ellipsoïdal que l'on nomme *surfel* ou *splat*. La figure 1.15 en donne un exemple. Cette approche est apparue relativement tôt dans la littérature. Son attrait réside principalement dans sa capacité à générer une surface sans avoir recours à un algorithme de reconstruction de surface, et donc sans supposer de connectivité entre les points.

L'idée originale d'utiliser des splats comme primitive de rendu est attribuée à (PFISTER et al., 2000). Le lecteur pourra se référer à (KOBBELT et Mario BOTSCH, 2004 ; SAINZ et PAJAROLA, 2004) pour un état de l'art détaillé des premières méthodes de splatting. (M. BOTSCH et al., 2005) proposent une méthode basée sur les évolutions récentes (pour l'époque) des GPUs. Le pipeline employé dans la méthode est reproduit sur la figure 1.16. Il se décompose en trois phases : la première

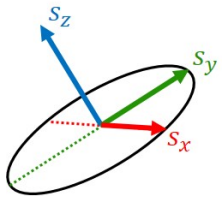


FIGURE 1.15 – Surfel : élément de surface orienté. Le surfel est un ellipsoïde dans le plan défini par son vecteur normal unitaire orienté. L'ellipsoïde peut être décrit par deux vecteurs qui portent son orientation ainsi que ses dimensions (demi petit et grand axe).

consiste à remplir un tampon de profondeur pour que, dans une deuxième phase, les attributs de la surface puissent être correctement interpolés entre les splats voisins. L'ombrage est alors réalisé dans un dernier temps sur la base du tampon géométrique (*G-Buffer*) ainsi créé. L'intérêt principal de cette méthode est qu'elle implémente une version efficace du filtre EWA (pour *Elliptical Weighted Average*) (GREENE et HECKBERT, 1986) qui permet la reconstruction d'une surface sans aliasing, garantissant ainsi une haute qualité géométrique et en termes d'attributs. Cette méthode représente aujourd'hui l'état de l'art du rendu de nuages de points par splatting.

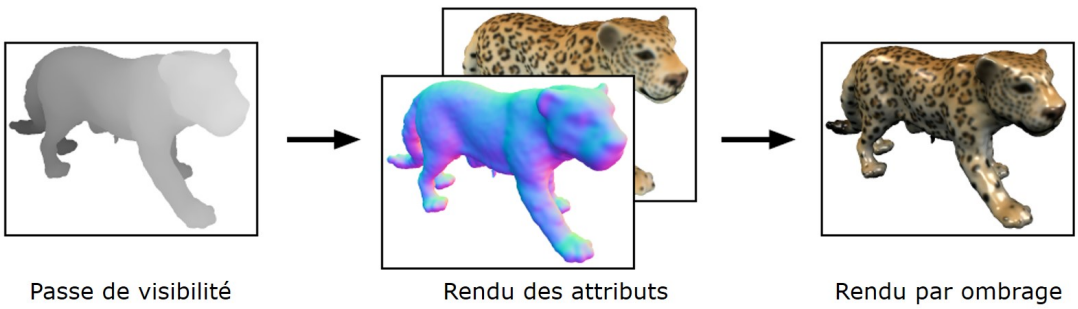


FIGURE 1.16 – Pipeline utilisé pour le rendu par splatting sur GPU. (Figure reproduite, modifiée et traduite de (M. BOTSCH et al., 2005))

Les méthodes de splatting sont intéressantes car elles permettent de gérer le problème des occlusions et de l'aliasing sans supposer de connectivité entre les points. Comme illustré sur la figure 1.17, la qualité de la surface produite est limitée, notamment au niveau des bords saillants et des zones de forte courbure. Elles ne permettent pas non plus de gérer les voisinages volumiques (végétation ...) ou linéiques (grilles, câbles ...). Elles ne peuvent donc pas être envisagées comme des méthodes unifiées de rendu pour les relevés laser complexes.

Le deuxième inconvénient de ces méthodes réside dans l'utilisation de normales et de rayons pour chaque point. Il existe des méthodes avancées pour calculer ces attributs supplémentaires (BOULCH et MARLET, 2012 ; BOULCH et MARLET, 2016). Bien que parallélisables, celles-ci sont gourmandes en ressources de calcul et ne sont donc pas envisageables pour le rendu de nuages de points massifs. Par ailleurs, pour

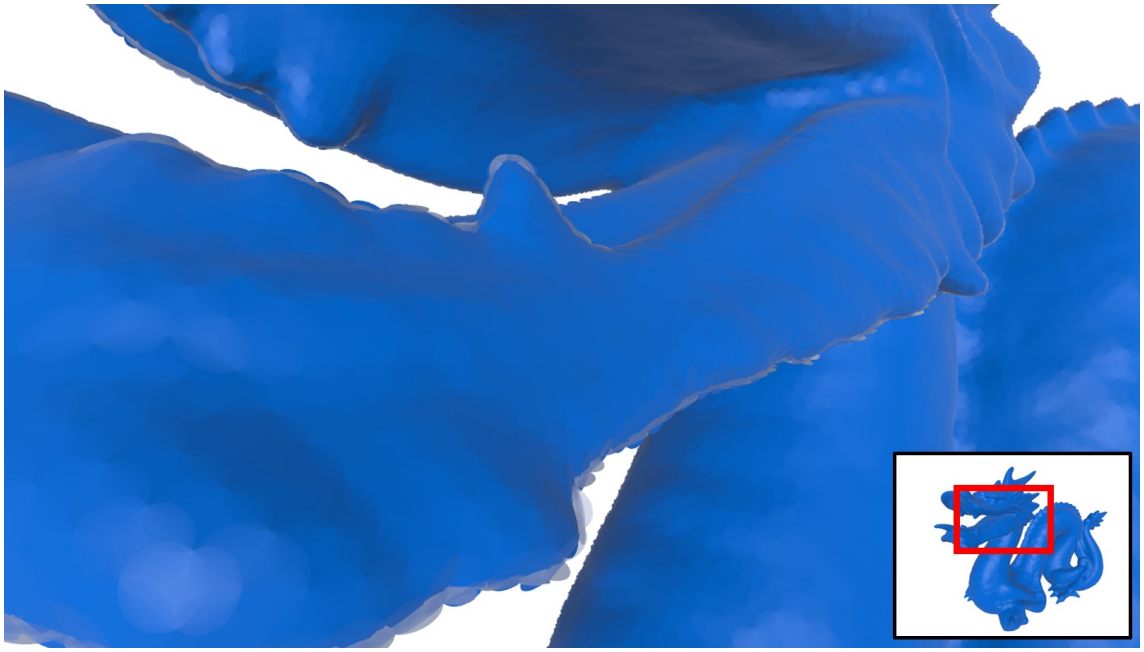


FIGURE 1.17 – Rendu par splatting du dragon de Stanford (M. BOTSCH et al., 2005). (En bas à droite) vue d'ensemble du modèle et de la zone d'intérêt. Les splats produisent des artefacts proche des bords saillants du modèle.

les mêmes raisons de voisinages qui ne sont pas nécessairement surfaciques, le calcul des normales est un problème mal posé pour les relevés laser complexes.

1.2.2.4 Rendu par visibilité analytique

Un des problèmes lié au rendu de nuages de points illustré sur la figure 1.4 est la possibilité de voir à travers le nuage de points des parties qui sont supposées être cachées. Il existe un ensemble de méthodes, que l'on désigne par les termes "visibilité" ou "retrait des parties cachées", qui s'intéressent à apporter une solution à ce problème précis.

(KATZ et al., 2007) proposent une méthode analytique simple et élégante pour supprimer les parties cachées d'un nuage de points. Le nuage de point complet est renversé par une transformation sphérique centrée sur le point de vue, comme on peut l'observer sur la figure 1.18. Le principe de la méthode réside alors dans le fait que les points visibles se situent sur l'enveloppe convexe du nuage de point transformé auquel on a ajouté le point de vue. Cette enveloppe représente une triangulation de la partie visible du nuage de points, elle peut ainsi être alors rendue donnant ainsi l'impression d'une surface remplie.

(MEHRA et al., 2010) en proposent une version robuste en relaxant les contraintes d'appartenance stricte à l'enveloppe convexe en introduisant une région tampon autour du nuage de point d'origine. Ils démontrent également l'intérêt de la méthode

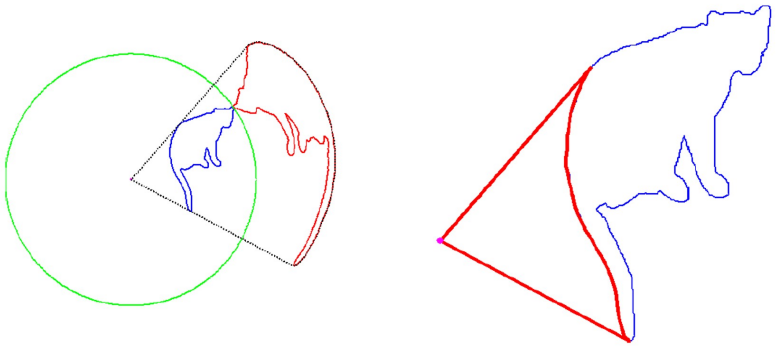


FIGURE 1.18 – Opérateur de retrait des parties cachées analytique de (KATZ et al., 2007). A gauche : transformée sphérique d'un nuage de point par rapport à un point de vue donné (centre de la sphère verte). A droite : transformée inverse sur l'enveloppe convexe du résultat de la première transformée et du point de vue ; seuls les points visibles subsistent. (Figure reproduite de (KATZ et al., 2007))

pour la reconstruction de surface à partir de nuages de points bruités. Bien que cette méthode simple et efficace ait connu un grand intérêt dans la communauté, elle reste inadaptée au cadre de notre étude dans la mesure où, d'une part elle nécessite l'ensemble du nuage de point. De ce fait, elle est difficilement applicable à des nuages de points massifs de scènes et reste ainsi cantonnée aux petits nuages de points d'objets circonscrits dans l'espace. D'autre part, elle n'est pas adaptée à une implémentation temps-réel.

(MACHADO E SILVA et al., 2014) en présentent une implémentation approximée sur GPU. Leur méthode est basée sur un calcul d'enveloppe convexe sur GPU par une discrétisation angulaire de l'espace projectif de la caméra. Bien que leur méthode améliore les performances de l'algorithme initial, elle reste très coûteuse en temps de calcul ce qui la rend également impossible à utiliser en temps réel sur des nuages de points massifs.

1.2.2.5 Rendu par traitements en espace image

Une autre manière de rendre un nuage de points est d'utiliser des traitements en espace image. Pour ce faire, le nuage de points est, dans un premier temps, rendu sans apriori sur la surface qu'il représente [8]. Cette étape est qualifiée de "passe géométrique". Les calculs de projection en perspective liés à cette étape sont détaillés dans l'annexe A. L'ensemble de textures obtenues sont alors traitées pour améliorer le rendu obtenu après la première étape.

L'intérêt de ces méthodes est multiple. Le premier réside dans l'indépendance en termes de complexité par rapport à la taille de la scène car les traitements ne dépendent que des dimensions des textures produites à la suite de l'étape géométrique. Le second intérêt est qu'il s'agit de la méthode privilégiée pour rendre des nuages de

8. Le plus souvent en produisant un fragment par point

points bruts, dans la mesure où l'étape géométrique ne nécessite pas d'apriori sur la surface. Il convient néanmoins de nuancer ce dernier point ; comme présenté dans la suite de cette section, certaines méthodes de rendu par traitements en espace image utilisent également des attributs supplémentaires pré-calculés (normales, densités, etc...).

(Grossman et Dally, 1998) sont les premiers à présenter un pipeline complet de rendu de nuages de points bruts en espace image. Il est tout d'abord intéressant de constater que leur motivation principale est de proposer une alternative au rendu basé image, trop gourmand en mémoire, ainsi qu'au rendu polygonal jugé moins efficace (à l'époque). La partie de rendu en espace image est divisée en deux temps : premièrement un opérateur de visibilité basé sur une pyramide d'images est employé pour résoudre les occlusions et labéliser les trous à remplir. Le remplissage des trous est ensuite effectué par *pull-push* (Gortler et al., 1996) (méthode décrite plus en détail dans la section 1.4). L'ombrage est effectué entre ces deux étapes à l'aide des attributs stockés dans chaque point (normales, couleur diffuse, couleur spéculaire, brillance). Bien que présentant des idées intéressantes, cette méthode reste néanmoins conçue pour traiter des images de profondeur synthétisées à partir de modèles polygonaux texturés et n'est donc pas adaptée au traitement de nuages de points issus de relevés laser massifs.

(Marroquim et al., 2007) propose également une méthode de rendu en espace image basée sur l'algorithme de *pull-push*. La méthode s'appuie sur les améliorations récentes de l'époque apportées aux méthodes de traitement de l'image par méthodes pyramidales (Strengert et al., 2006). Elle propose une alternative plus efficace au rendu par splatting car elle ne repose que sur une seule passe géométrique alors que les méthodes de splatting en nécessitent deux, comme cela est illustré sur la figure 1.16. L'utilisation d'une méthode pyramidale est intéressante pour la reconstruction en espace image car elle permet de synthétiser de l'information à différentes échelles de manière extrêmement efficace. Néanmoins, la méthode proposée ici n'est pas applicable aux nuages de points bruts puisqu'elle nécessite d'avoir accès aux mêmes attributs que ceux utilisés par les méthodes de splatting (normales et rayons), notamment pour gérer les occlusions comme on peut le voir sur la figure 1.19. Par ailleurs, nous montrerons dans la section 1.4 que la qualité de la carte de profondeur reconstruite par l'algorithme de *pull-push* n'est pas suffisante pour estimer des normales a posteriori.

Dans la même philosophie, (Farias, 2014) présente une alternative au splatting en mettant en œuvre l'algorithme de *jump flooding* (Rong et Tan, 2006). La méthode est intéressante dans la mesure où sa complexité est en $nlog(n)$ où n représente le nombre de points du *viewport*. Bien qu'elle permette de s'affranchir du calcul des rayons, elle est néanmoins basée sur l'utilisation de normales et donc non applicable aux nuages de points bruts.

(Preiner et al., 2012) ont introduit la méthode *Auto Splats* qui permet de faire du splatting sur des nuages de points bruts. Pour cela ils estiment les rayons et

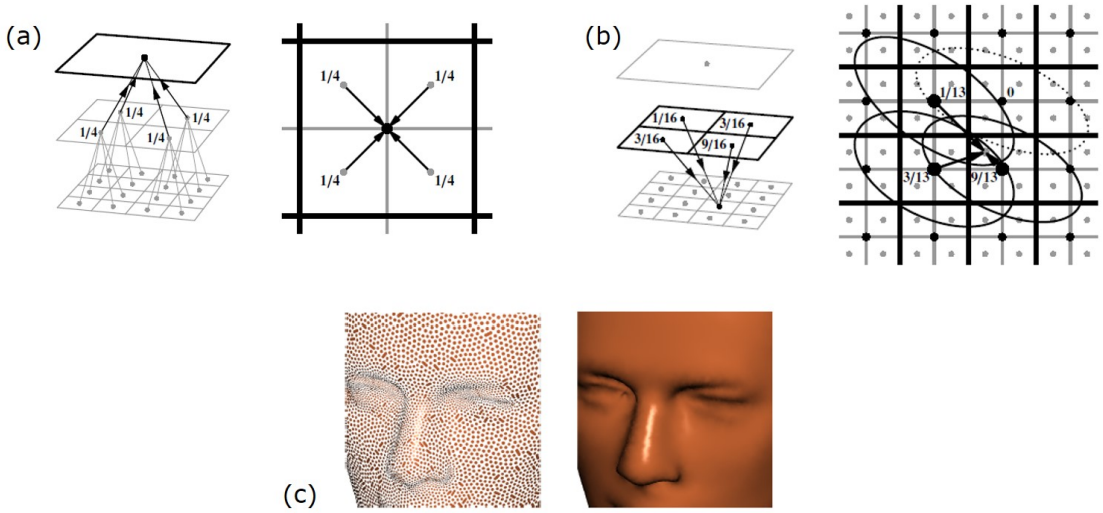


FIGURE 1.19 – Pipeline utilisé par (MARROQUIM et al., 2007). (a) phase de *pull* une image de résolution 2 fois plus faible est synthétisée à partir de l'image au niveau inférieur. (b) phase de *push* les informations manquantes dans le niveau inférieur sont interpolées à partir des informations du niveau supérieur. Les ellipses d'influence des points, calculées à partir de leurs rayons et normales, sont utilisées pour gérer les occlusions. (c) Résultat sur un nuage test. (Figure reproduite et modifiée de (MARROQUIM et al., 2007))

normales par une méthode exclusivement en espace image, basée sur un calcul de k plus proches voisins sur GPU. Une fois ces attributs calculés, le nuage de points est rendu par splatting par la méthode de (M. BOTSCH et al., 2005). Comme on peut le voir sur la figure 1.20, les résultats obtenus avec cette méthode sont comparables à ceux obtenus avec les meilleures méthodes de splatting sur une grande variété de nuages de points. La méthode souffre néanmoins des mêmes artefacts de rendu que les méthodes de rendu par splatting pour la gestion des voisinages non surfaciques ainsi que pour la gestion des arêtes vives.

(ROSENTHAL et LINSEN, 2008) utilisent une étape de remplissage avant de procéder au retrait des parties cachées. Leur approche est basée sur l'itération de filtres 3×3 sur l'ensemble des pixels de l'image. Ces filtres permettent de qualifier le voisinage de chaque pixel afin de décider s'il doit être rempli ou non. De ce fait cette méthode n'est pertinente que lorsque les nuages de points sont observés de suffisamment loin pour que la densité apparente à l'écran soit suffisante. Le retrait des parties cachées est alors effectué par comparaison de profondeur sur l'image résultante. Le caractère itératif de leur méthode présente l'inconvénient de mal gérer les vues rasantes pour lesquelles l'espace entre les points à l'écran peut être arbitrairement grand à cause de la projection perspective, ce qui nécessiterait un nombre d'itération élevé pour combler les trous ainsi produits. Leur méthode reste donc cantonnée à l'observation d'objets de densité uniforme.

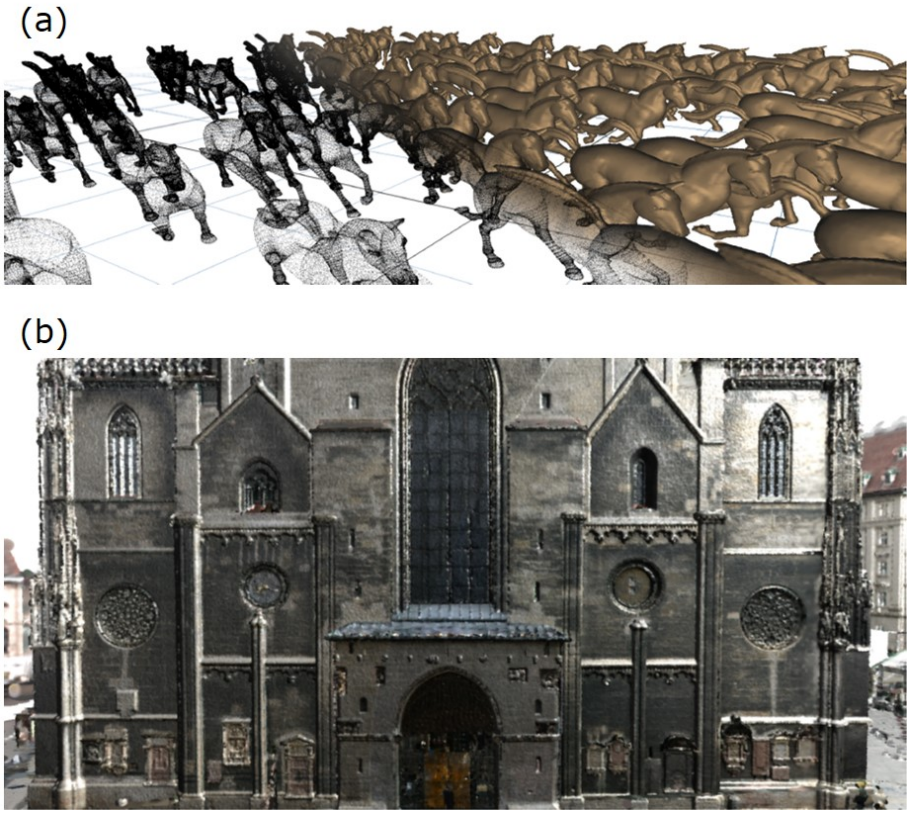


FIGURE 1.20 – Résultats obtenus par l'algorithme *Auto Splats* sur des nuages de points bruts. (a) Scène dynamique de 1 million de points. (b) Scan massif de cathédrale (10 million de points après sélection par niveau de détails). (Figure reproduite et modifiée de (PREINER et al., 2012))

(PINTUS et al., 2011) représente la référence la plus proche des travaux présentés dans ce chapitre dans la mesure où avec (PREINER et al., 2012) ils sont les seuls à traiter la problématique du rendu de nuages de points bruts. Ils proposent une méthode basée uniquement sur des calculs en espace image. Leur pipeline est similaire à (GROSSMAN et DALLY, 1998) ; ils enchainent retrait des parties cachées et reconstruction (ou remplissage). Le retrait des parties cachées est basé sur un calcul d'occlusion dans l'axe de la caméra virtuelle. Il permet de labéliser les parties à remplir dans l'image. Celles-ci sont ensuite remplies par un filtre médian itéré. Pour les mêmes raisons citées plus haut, cette approche n'est pas adaptée à la reconstruction d'une surface de haute qualité géométrique utilisable ensuite pour un rendu par ombrage. Par ailleurs, bien qu'ils présentent des résultats sur des jeux de données complexes, ils ne traitent pas explicitement du problème de l'exploration interactive avec des scènes comportant de grandes différences de profondeur. De plus leur méthode ne permet pas de réaliser un rendu par ombrage du modèle car elle n'estime pas les normales à la surface.

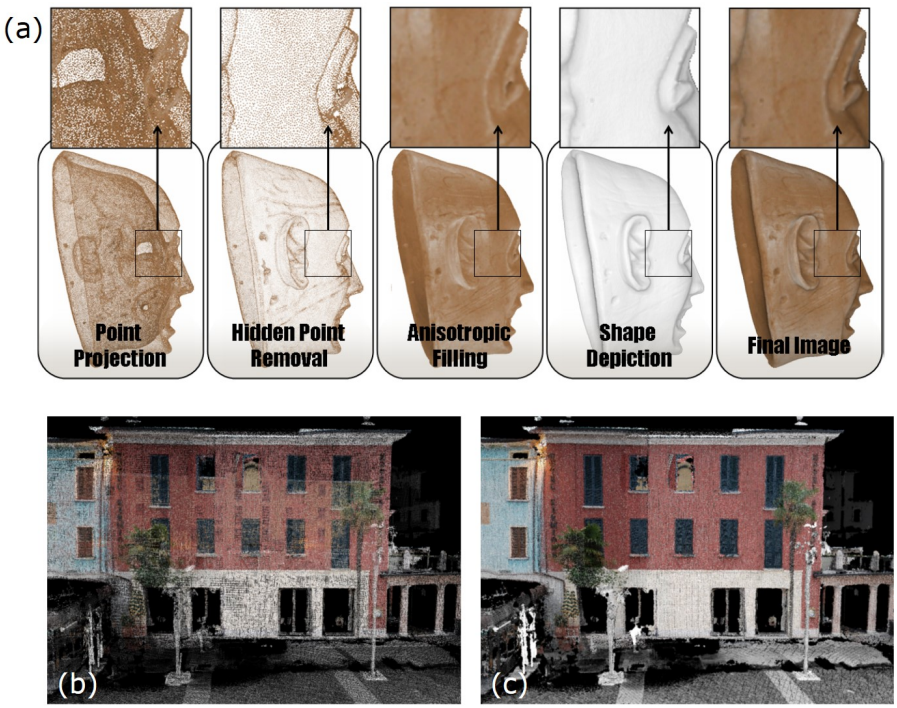


FIGURE 1.21 – (a) Pipeline utilisé dans (PINTUS et al., 2011). (b) Rendu par projection à 1 point par pixel. (c) Rendu par la méthode de (PINTUS et al., 2011). (Figure reproduite et modifiée de (PINTUS et al., 2011))

Les méthodes par traitements en espace image semblent être les plus adaptées aux contraintes que pose le rendu de nuages de points bruts. La figure 1.22 synthétise les principales approches mises en évidence par l'étude de l'état de l'art. L'approche qui semble la plus intéressante consiste à dissocier l'étape de retrait des parties cachées de l'étape de reconstruction comme (PINTUS et al., 2011). C'est le choix que nous avons retenu dans la section 1.4. Nous montrerons que cela permet d'obtenir un pipeline modulaire qui peut être utilisé de manière polyvalente et qui permet de séparer le problème en deux sous-problèmes distincts.

Dans le but d'obtenir une surface de haute qualité géométrique, l'approche de remplissage par pyramide d'images (MARROQUIM et al., 2007) semble plus intéressante que les méthodes par propagation (ROSENTHAL et LINSEN, 2008 ; PINTUS et al., 2011). En l'état, ces méthodes ne permettent toutefois pas d'atteindre une qualité de rendu suffisante pour un rendu par ombrage de nuages de points bruts[9].

9. Les méthodes de l'état de l'art qui les implémentent obtiennent un ombrage de bonne qualité car elles se basent sur l'utilisation des normales

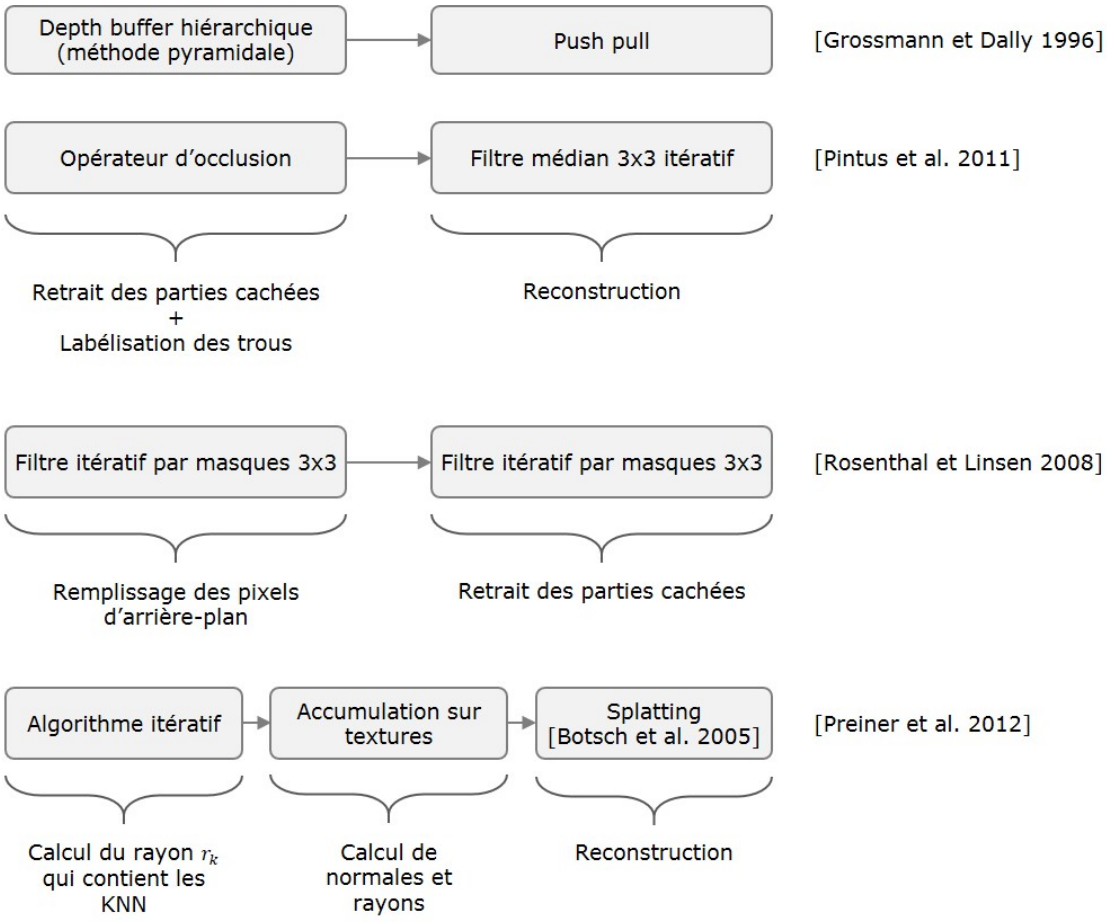


FIGURE 1.22 – Comparaison des différents pipelines de rendu de nuages de points bruts en espace image.

1.3 Structuration out-of-core de nuages de points massifs pour le rendu

Nous présentons dans cette section une nouvelle méthode de structuration *out-of-core* de nuages de points massifs pour le rendu. Le terme *out-of-core* désigne la capacité de la méthode à gérer des données qui potentiellement sont plus volumineuses que la mémoire RAM et qui de ce fait, ne peuvent être chargées simultanément par le programme. Cette méthode permet d'adresser une partie du problème de l'efficacité de la méthode de rendu sous la contrainte de l'utilisation de nuages de points massifs. Plus précisément, il s'agit de contrôler le temps de calcul nécessaire pour la passe géométrique en limitant le nombre de primitives géométriques traitées à chaque nouvelle image générée, et ce, de manière indépendante du nombre de points de la scène qui peut être arbitrairement grand.

1.3.1 Introduction

Le tableau 1.1 présente les différents ordres de grandeur liés au rendu de nuage de points. Ces chiffres montrent que :

- Sachant que le transfert des données entre périphériques de stockages impliqués dans le rendu est réalisé de la manière suivante : SSD → RAM → VRAM. Le stockage est donc limité par la taille du plus petit périphérique : la mémoire de la carte graphique, qui correspond en équivalent nuage de points à 4 km de rues.
- Les taux de transfert quant à eux, limitent la capacité à charger à la volée de nouveaux nuages de points. Le transfert depuis le disque vers la RAM est très limité, il correspond à 3 petits objets par seconde. Le taux de transfert depuis la RAM vers la RAM vidéo est bien plus élevé, il reste néanmoins faible lorsqu'on le ramène au temps total alloué à la production de chaque image : 10 ms. Cette valeur correspond d'ailleurs au temps de calcul total pour calculer chaque image et doit donc être partagé entre chargement et traitements. Ainsi par exemple pour une durée de 1 ms, on ne peut charger que 20 Mpts ce qui correspond à deux petits objets.
- On peut également transposer ce raisonnement aux ordres de grandeur liés à la capacité de traitement du GPU qui sont encore plus faibles : par exemple en 3 ms le GPU peut traiter au plus 1.17 Mpts.

En conclusion, il est donc inconcevable de pouvoir traiter un nuage de points arbitrairement grand, comme par exemple à l'échelle d'un quartier, par un algorithme de rendu basique qui consisterait à traiter tous les points par la carte graphique. Il est ainsi nécessaire d'avoir recours à un algorithme spécifique pour réduire en temps réel cette quantité de données à traiter, tout en limitant les transferts vers la mémoire graphique.

Ordres de grandeurs : tailles de nuages de points

Nuage de points	longueur	nombre de points	taille
Petit objet	-	10 Mpts	160 Mo
Rue	2 km	200 Mpts	3.2 Go
Quartier	100 km	10 Gpts	160 Go
Ville	1700 km	170 Gpts	2.7 To

Ordres de grandeurs : périphériques de stockage

Périphérique	taille	équivalent
VRAM (GPU)	8 Go	500 Mpts
RAM	16 Go	1 Gpts
SSD	1 To	62 Gpts

Ordres de grandeurs : taux de transfert entre périphériques

Bus de données	débit	équivalent
SSD $\rightarrow$ RAM	550 Mo/s	34 Mpts/s
RAM $\rightarrow$ GPU	320 Go/s	20 Gpts/s

Ordres de grandeurs : capacités de traitement

Processeur	capacité de traitement
CPU	60 Mpts/s
GPU	3.9 Gpts/s

TABLE 1.1 – Différents ordres de grandeur relatifs au rendu de nuages de points. On suppose qu'un point occupe 3*4+4 octets en mémoire, ce qui correspond à 3 coordonnées flottantes 3 canaux de de couleurs et un champ additionnel (intensité ou label). Les ordres de grandeur de capacité de traitement ont été mesurés par une implémentation basique d'un logiciel de rendu de nuages de points (configuration : Intel Core i7-7700K 4.2 GHz, 64 Go RAM, Nvidia GeForce GTX 1080).

Comme cela a été montré par une étude de l'état de l'art, l'ensemble des algorithmes d'accélération de rendu de nuage de points sont basés sur une organisation spatiale des données que l'on qualifie de "structuration". Ces derniers exploitent le fait qu'étant donné un point de vue virtuel, tous les points du nuage ne sont pas nécessaires pour le rendu. Ainsi pour rendre efficacement un nuage de points, il suffit de sélectionner uniquement les points qui contribuent au rendu, ce qui peut être mis en œuvre de trois manières :

- **Sélection par niveau de détail (LOD culling) :** par projection en perspective, plus une partie du modèle est éloignée de la caméra, moins elle est détaillée. Ainsi loin de la caméra il n'est pas nécessaire de rendre le nuage de point avec une résolution aussi haute que proche de la caméra.
- **Retrait de points hors du cône de la caméra (frustum culling) :** la caméra présente un cône de vue, ainsi tous les points qui se trouvent en dehors de ce cône ne sont pas nécessaires au rendu.
- **Retrait de points par occlusion (occlusion culling) :** les parties du modèle qui sont cachées par d'autres parties qui sont plus proches de la caméra ne contribuent pas au rendu. Ce dernier point est complexe à mettre en œuvre pour le rendu de nuages de points. Il ne sera pas traité dans cette partie.

Par une étude de l'état de l'art, il apparait que (Michael Wimmer et Scheiblauer, 2006) représente la méthode la plus avancée actuellement en termes de structures d'accélération de rendu de nuages de points. Elle présente néanmoins deux inconvénients : un phénomène de *popping*[10] survient lors du passage d'un niveau de détail[11] discret à l'autre. Ce phénomène, illustré sur la figure 1.23, induit une gêne visuelle pour l'utilisateur. Le deuxième inconvénient qu'elle présente réside dans le format de données qu'elle produit. Ce format ne permet pas de retrouver l'organisation initiale des fichiers qui ont servi au calcul de la structure de données.

Nous présentons ainsi dans cette partie une extension de cette méthode qui propose une solution aux deux limitations précédemment citées. La première réside dans la réduction du phénomène de popping. La deuxième réside dans le format de données de la structure proposée, celui-ci conserve la topologie des données utilisées pour la produire, permettant ainsi de traiter les données (tant que le traitement ne modifie pas l'ordre des points) sans avoir à recalculer entièrement la structure de données. Cela permet par exemple, une intégration transparente dans une chaine existante de traitements de nuages de points.

10. Le *popping* désigne l'apparition brutale d'objets dans la scène 3D, induisant ainsi une discontinuité au niveau de l'affichage

11. Le terme "niveau de détail" est abrégé par son acronyme anglophone LOD (pour *Level Of Detail*) dans la suite du document

FIGURE 1.23 – Phénomène de popping dû à l'utilisation de niveaux de détail discrets par (Michael WIMMER et SCHEIBLAUER, 2006). Les deux vues sont obtenues avec la version en ligne du logiciel Potree depuis deux points de vues proches (le budget de points utilisé est de 1.5 Mpts). On voit apparaitre et disparaitre les niveaux de détail discrets.

1.3.2 Structure de données

Après une description de la structure de données proposée, nous présentons les différents algorithmes mis en œuvre pour la construire et pour la rendre.

1.3.2.1 Le Raw Sequential Point Tree (RSPT)

Au cœur de notre structure de données se trouve le RSPT (pour *Raw Sequential Point Tree*). Cette structure fait écho au principe introduit par (DACHSBACHER et al., 2003) et simplifié par (Michael WIMMER et SCHEIBLAUER, 2006) [12] qui consiste à construire un arbre de partition spatiale sur un nuage de points et de stocker ensuite les différents niveaux de l'arbre les uns à la suite des autres en mémoire. La structure spatiale est alors oubliée, seule l'information de LOD est alors gardée.

La construction d'un RSPT se fait en deux étapes. La première consiste à calculer un octree sur le nuage de points d'entrée. L'octree contient un point par nœud (aussi bien les nœuds internes que les feuilles). L'algorithme 1.1 décrit sa construction. Ainsi à chaque création de nœud on choisit un point représentant dans la liste des points qui le composent. Ce choix peut être réalisé de différentes manières, le plus polyvalent étant de choisir le point le plus proche du centre du nœud. La récursion est arrêtée lorsque qu'il ne reste plus qu'un point dans la liste à traiter, ou bien l'on atteint une taille de feuille suffisamment petite. De cette manière on réalise également un sous-échantillonnage du nuage de points.

La deuxième étape de la construction consiste à récupérer la liste des points de chaque niveau de l'arbre, puis de les aligner séquentiellement en mémoire. La

12. (Michael WIMMER et SCHEIBLAUER, 2006) appellent leur structure MOSPT (pour *Memory Optimized Sequential Point Tree*)

Algorithme 1.1 : Construction d'un octree RSPT

Entrées : liste de points $\mathcal{P} = \{p_i, 0 \leqslant i < N\}$, taille de feuille s_{max}
Résultat : octree $\mathcal{O}$

```
// Calcul de la racine de l'arbre
(c_0, s_0) = cubeEnglobant(P)
N_0.feuille = faux
O.c = c_0                                   // centre du cube englobant
O.s = s_0                                   // taille du cube englobant
O.racine = N_0

// Construction récursive de l'arbre
construireOctreeRSTP(N_0, P, c_0, s_0)

Fonction construireOctreeRSTP(nœud N_k, liste de points P_k, centre c_k, taille s_k)
    si N_k.s < s_max ou |P_k| = 1 alors
        N_k.p = P_k[0]
        N_k.feuille = vrai
    sinon
        // Calcul du représentant du noeud
        p_k = plusProcheVoisin(c_k, P_k)
        P = P \ p_k
        N_k.p = p_k

        // Séparation de la liste de points en 8 listes filles
        C = separationEnOctants(c_k, P_k)

        // Création des noeuds fils non vides
        pour i = 0 à 7 faire
            si non estVide(C[i]) alors
                N_i = N_k.creeNoeudFils()
                c_i = c_k + (s_k/4)*offsetFils(i)
                construireOctreeRSTP(N_i, C[i], c_i, s_k/2)
```

particularité de la structure proposée est d'introduire à ce niveau une permutation aléatoire des points au sein de chacune de ces listes. Cette simple opération permet de considérer des niveaux de détail décimaux et non plus entiers, réduisant ainsi considérablement l'effet de *popping* en proposant une transition plus lisse entre niveaux de détail. Ce principe de construction est décrit sur la figure 1.24.

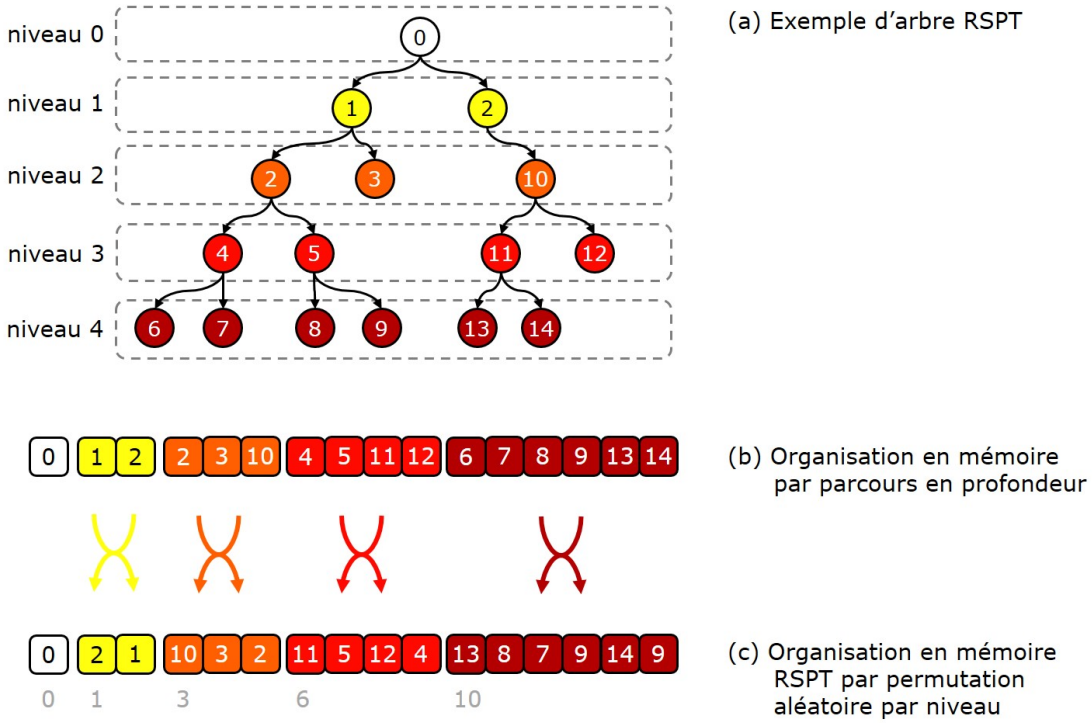


FIGURE 1.24 – Description du processus de construction d'un RSPT à partir d'un octree RSPT (représenté ici par un arbre binaire).

Le nuage de points organisé en RSPT, il suffit alors de garder en mémoire l'index des points au début de chaque niveau (représentés en gris sous le RSPT sur la figure 1.24). Pour rendre le nuage de points à un niveau de détail l donné, il suffit alors de rendre les N_l premiers points du nuage, où N_l représente l'index du premier point du niveau l. La figure 1.25 donne un exemple de RSPT, calculé sur une portion de scan lidar mobile et rendu à différents niveaux de détail.

Soient : t un seuil en pixels fixé par l'utilisateur, z la distance du RSPT à la caméra, s_0 la taille de la racine du RSPT, $\theta = \frac{\alpha}{2}$ le demi angle du champ de vision de la caméra et h la hauteur en pixels de la texture de rendu. Le niveau $l(z)$ de RSPT requis pour le rendu est alors obtenu par l'équation suivante :

$$l(z) = \frac{\log\left(\sqrt{3}hs_0\right) - \log\left(t\tan\theta z\right)}{\log\left(2\right)} \tag{1.1}$$

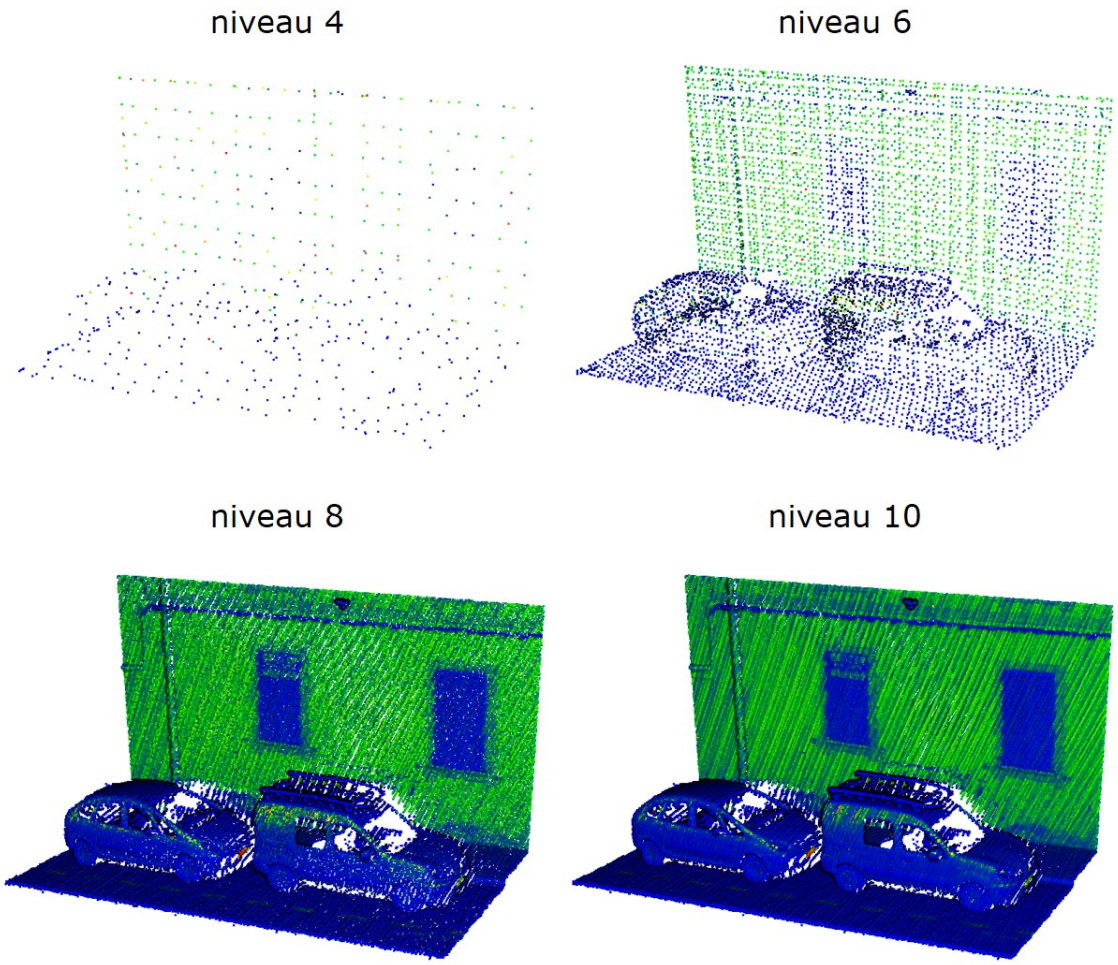


FIGURE 1.25 – Différents niveaux de détail d'un RSPT. Le RSPT a été calculé sur une portion d'un scan laser mobile de la ville d'Ajaccio. Le nuage contient 408,585 points. La taille à la racine est de 10.9 m et taille de feuille s_{max} a été fixée à 2cm. L'arbre contient 11 niveaux, 404,647 points et sa construction a pris 184 ms (CPU : Intel core i7-4770 3.4 GHz).

Les calculs qui permettent d'aboutir à ce résultat sont détaillés en annexe A. Le niveau de détail obtenu est décimal, il permet de calculer le nombre de points à afficher. Le paramètre t représente la distance en pixels entre deux points voisins une fois projetés à l'écran. En dessous de 1 px il n'y a donc pas de différence à l'écran. Le nombre de points rendu à chaque instant dépend directement de ce paramètre, il permet donc de donner un contrôle à l'utilisateur de la puissance de calcul nécessaire. La seule variable dans la formule est la distance du RSPT à la caméra z. Elle peut être calculée de différentes manières : de manière approchée, on peut la calculer à partir de la distance au centre du nœud racine, ou bien de manière exacte comme la distance à la surface du cube et mise à zéro à l'intérieur du cube.

Comme cela a été affirmé plus haut, le RSPT permet de gérer le LOD de manière globale sur l'ensemble d'un nuage de points. En revanche, il ne permet pas :

- Une gestion fine du LOD en différents points d'un nuage de points.
- Une gestion fine des parties hors du cône de la caméra.
- La gestion du chargement déchargement des données depuis le disque vers la RAM.

Ces trois problèmes sont d'autant plus importants lorsque le nuage de points est une scène massive. Pour y pallier, il est nécessaire d'encapsuler le RSPT dans une structure hiérarchique globale qui permet de gérer ces trois points. C'est ce qui est présenté dans le paragraphe suivant.

1.3.2.2 Structure externe d'octree et format

La structure proposée est basée sur une unique structure externe d'*octree*, à la différence de (Michael Wimmer et Scheiblauer, 2006) qui basent leur structure sur plusieurs *octrees* entrelacés. C'est cette propriété qui permet de garder la topologie de la données d'entrée, au détriment de la qualité de rendu en vue de loin du jeu de données. Ce dernier choix a été motivé par notre intérêt dans l'exploration interactive de scènes et donc en "vue de l'intérieur".

A la différence de l'*octree* implémenté pour la création des RSPT, décrit plus haut, cet octree est composé de feuilles de tailles fixes et seules les feuilles contiennent des données (comme illustré sur la figure 1.26). Par souci d'efficacité et d'empreinte mémoire, son implémentation est similaire à (Elseberg et al., 2013) : les centres et tailles de chaque nœud sont calculés à la volée à chaque parcours de l'arbre. Chaque feuille de l'arbre est un cube qui contient un RSPT.

La spécificité du format proposé réside dans le fait que les RSPTs voisins sont écrits sur le disque dans le même fichier. Ce dernier peut être écrit dans un format classique de nuage de points (ex : PLY ou LAS) qui peut être lu par n'importe quel autre programme externe. La répartition des RSPTs dans les différents fichiers est décrite par un fichier de configuration annexe qui est lu à l'initialisation du jeu de données. Cette innovation permet d'une part de charger en RAM plusieurs RSPT voisins à chaque fois ce qui a pour effet de pré-charger des parties du jeu de données

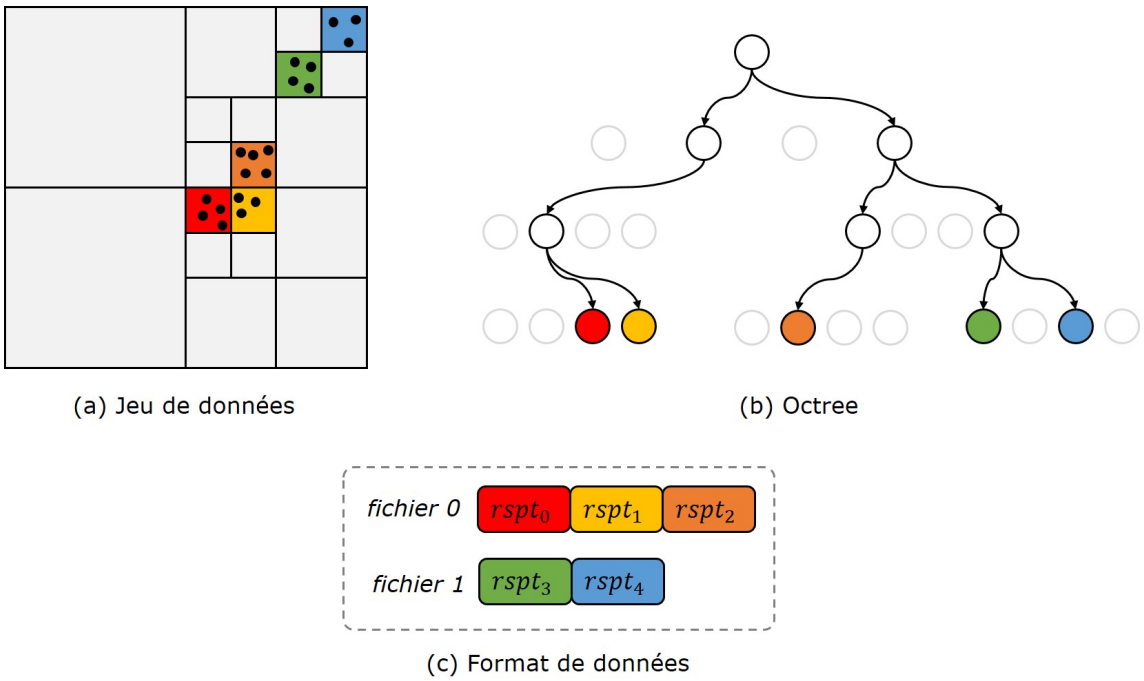


FIGURE 1.26 – Structure d'octree externe et format de données. (a) Jeu de données initial, les cases colorées représentent des RSPTs. L'octree est représenté spatialement, les cases grises représentent les nœuds vides de l'arbre. (b) Représentation arborescente de l'octree. Seuls les nœuds non vides sont instanciés. (c) Format de données résultat, les RSPTs sont stockés en mémoire dans des fichiers par proximité spatiale.

qui peuvent être potentiellement nécessaires par la suite. D'autre part cela permet d'obtenir des fichiers de nuage de points éditables tant que l'utilisateur ne modifie pas l'ordre et la position des points. Ainsi cela peut par exemple permettre de segmenter et classifier le jeu de données (ou une partie de celui-ci) sans nécessairement recalculer toute la structure de données. La figure 1.27 donne un exemple de résultat obtenu sur une portion d'un jeu de donnée traité. Dans ce cas, nous avons gardé pour chaque RSPT, l'information du nuage depuis lequel il a été créé, ce qui a permis par la suite de garder la même répartition de fichiers que dans le jeu de données initial. A noter qu'il ne s'agit pas de la seule stratégie de répartition des RSPTs en fichiers. On peut par exemple les répartir plus équitablement en constituant des groupes de taille équivalente par propagation de région.

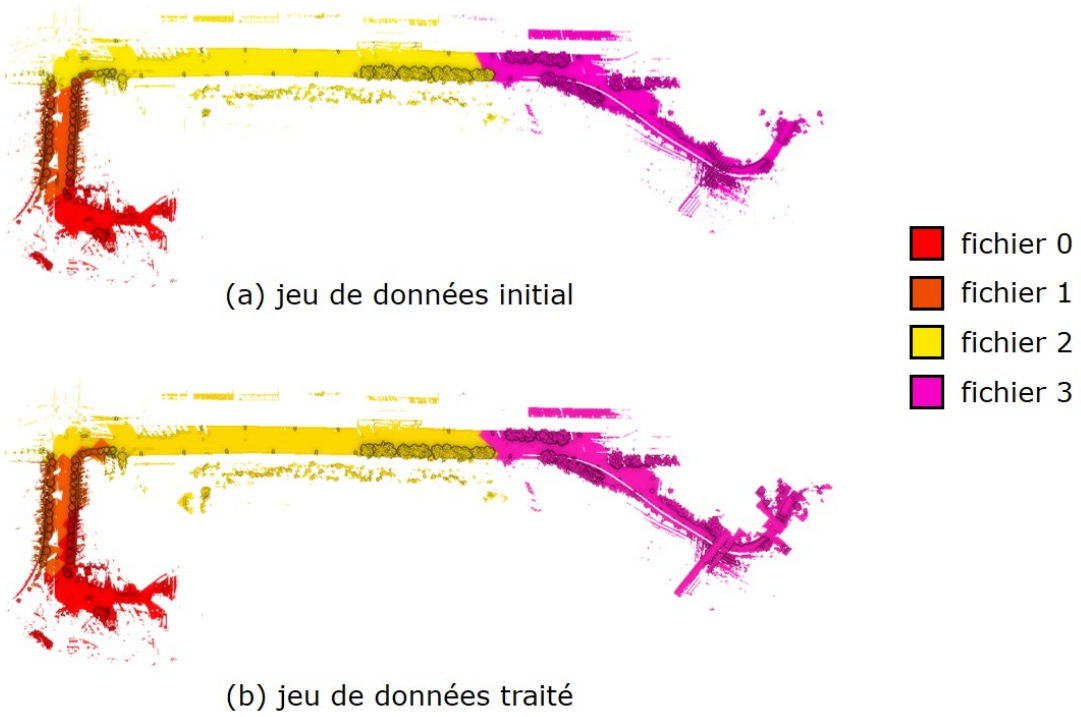


FIGURE 1.27 – Portion d'un jeu de données organisé avec notre structure de données. Sont affichés ici 4 nuages de points en vue de dessus. On remarque que les fichiers créés (b) sont très proches des fichiers initiaux (a).

La construction de l'octree est similaire à l'algorithme 1.1, à la différence que comme dans ce cas les nœuds internes de l'octree ne contiennent pas de données, il n'est pas nécessaire de choisir un représentant. De plus les tailles de feuilles sont fixées donc le critère d'arrêt ne porte que sur la taille du nœud. Comme la construction porte sur l'ensemble du jeu de données qui ne tient potentiellement pas en RAM, elle est réalisée à l'aide d'une structure spatiale creuse annexe. Cette structure est une table de hachage dont la clé est la position 3D sur la grille définie par la taille des feuilles de l'octree. Chaque case contient alors un fichier temporaire dans

lequel sont stockés les points au cours du traitement. La construction de l'octree à proprement parler est alors réalisée dans un deuxième temps en prenant en entrée la liste des cases non vides de la grille. La figure 1.26 illustre ce procédé.

L'algorithme consiste alors à sélectionner les RSPTs visibles, à savoir ceux qui sont contenus ou qui intersectent le cône de la caméra virtuelle. Cet algorithme peut être réalisé par un algorithme récursif similaire à (S. Rusinkiewicz et M. Levoy, 2001). Une fois la liste des RSPTs obtenue, il suffit de calculer pour chacun le nombre de points à rendre par la formule 1.1.

Le chargement des RSPTs de la RAM vers la mémoire graphique doit être fait sur le même thread que le rendu [13]. Ainsi pour ne pas détériorer la fréquence d'affichage, une limite sur le débit de données à charger est fixée. Ainsi, si un RSPT n'est pas disponible, car pas encore chargé en RAM ou sur la mémoire vidéo (ou VRAM), il n'est simplement pas rendu. Les RSPTs présents sur la VRAM sont également supprimés par un mécanisme de cache LRU [14] lorsque la quantité totale de VRAM occupée dépasse une limite fixée par le matériel (typiquement 11 Go pour une Nvidia GeForce 1080 Ti).

En ce qui concerne le chargement depuis le disque, celui-ci peut être réalisé sur un thread séparé. Nous avons mis en œuvre un algorithme de pré-chargement qui consiste simplement à charger les nuages de points contenus dans un rayon autour de la camera virtuelle. Le déchargement des données est également géré par un mécanisme de cache LRU. Un nuage est noté comme utilisé lorsque l'un de ses RSPTs est utilisé dans le thread principal de rendu. Les deux mécanismes de chargement sont illustrés sur la figure 1.28.

1.3.2.3 Gestion des parties lointaines

La gestion des parties lointaines représente la principale limitation de la structure proposée. En effet, comme décrit sur la figure 1.28, à chaque nouvelle image générée, les RSPTs sur la carte sont mis à jour par un mécanisme de cache LRU qui assure que la quantité de données présentes sur la carte graphique ne dépasse pas une limite fixée par l'utilisateur. Cela implique donc une limite sur l'horizon que peut voir l'utilisateur. Bien qu'il s'agisse d'une limitation par rapport à (Michael Wimmer et Scheiblauer, 2006), celle-ci ne pose pas de problème lorsque l'on considère l'exploration interactive d'une scène étant donné que le point de vue est pris proche du sol.

Une manière simple de pallier à ce problème est de créer un nuage de points global à partir d'un sous-échantillonnage global du nuage de points total. Il peut être créé lors de l'étape de création des RSPTS. La figure 1.29 montre le résultat associé à cette solution obtenu sur un scan laser mobile de la ville d'Ajaccio. Le

13. Il s'agit là d'une restriction de l'API graphique.
14. LRU est l'acronyme de *Last Recently Used* qui signifie le plus ancien à avoir été utilisé.

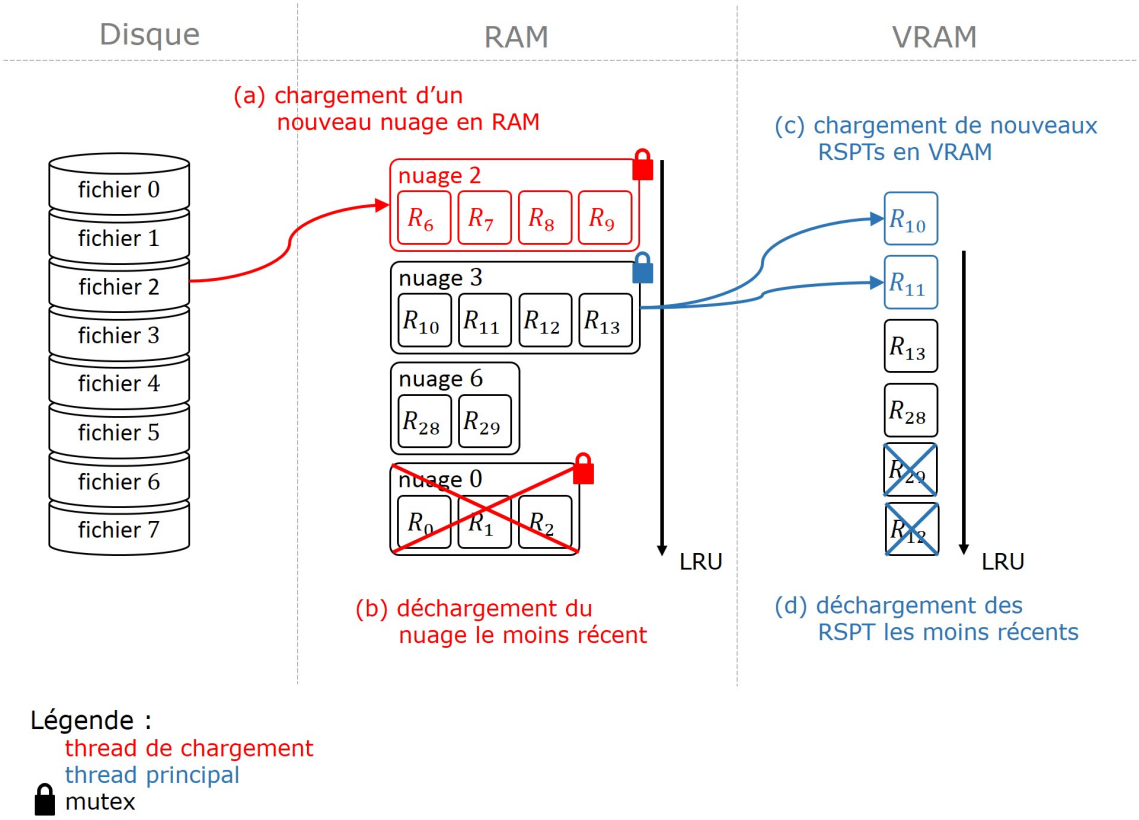


FIGURE 1.28 – Illustration du mécanisme de chargement et déchargement des données du disque vers la mémoire RAM puis vers la mémoire graphique. Ce mécanisme est réalisé sur deux threads illustrés en rouge pour le thread de chargement du disque vers la RAM et en bleu pour le chargement de la RAM vers la VRAM. Les RSPTs sont notés R_i.

nuage de fond permet de visualiser la partie du centre-ville qui n'est pas chargée sur le GPU.

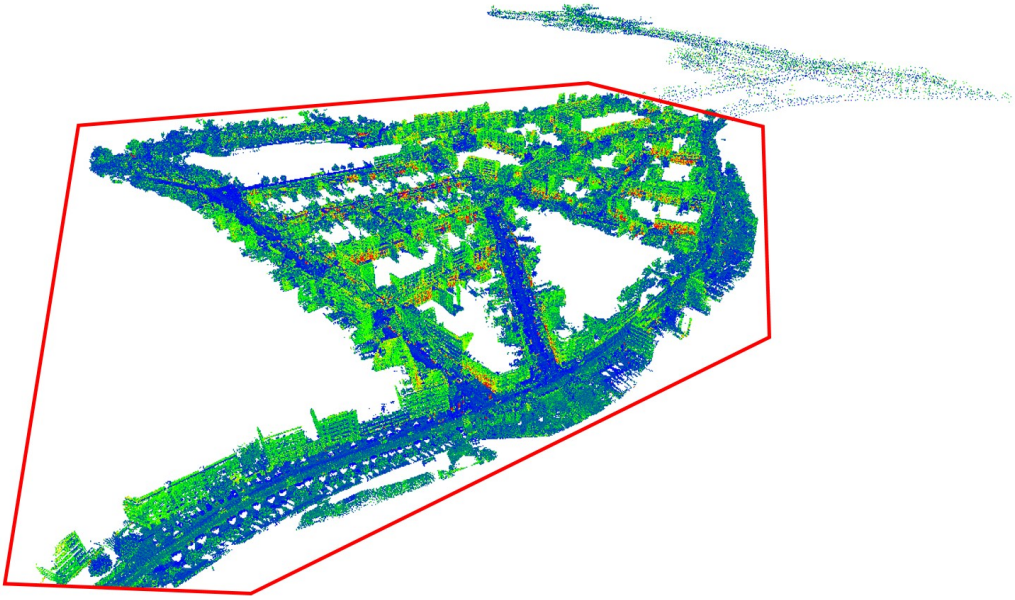


FIGURE 1.29 – Illustration de la gestion des parties lointaines par l'utilisation d'un nuage de fond. Les RSPTs rendus par la carte graphique sont entourés en rouge. La limite de chargement sur le GPU est fixée à 5 Go et celle en RAM à 10.0 Go. Le reste du nuage correspond au nuage de fond sous-échantillonné qui contient 14151 points.

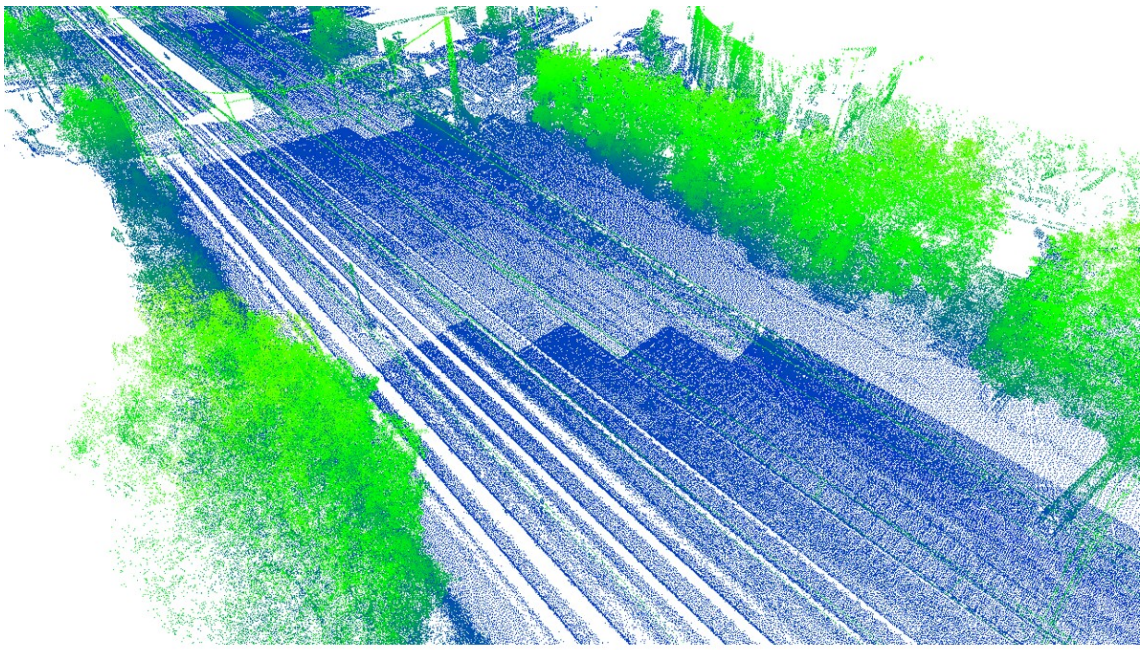
1.3.3 Résultats

1.3.3.1 Niveaux de détail continus

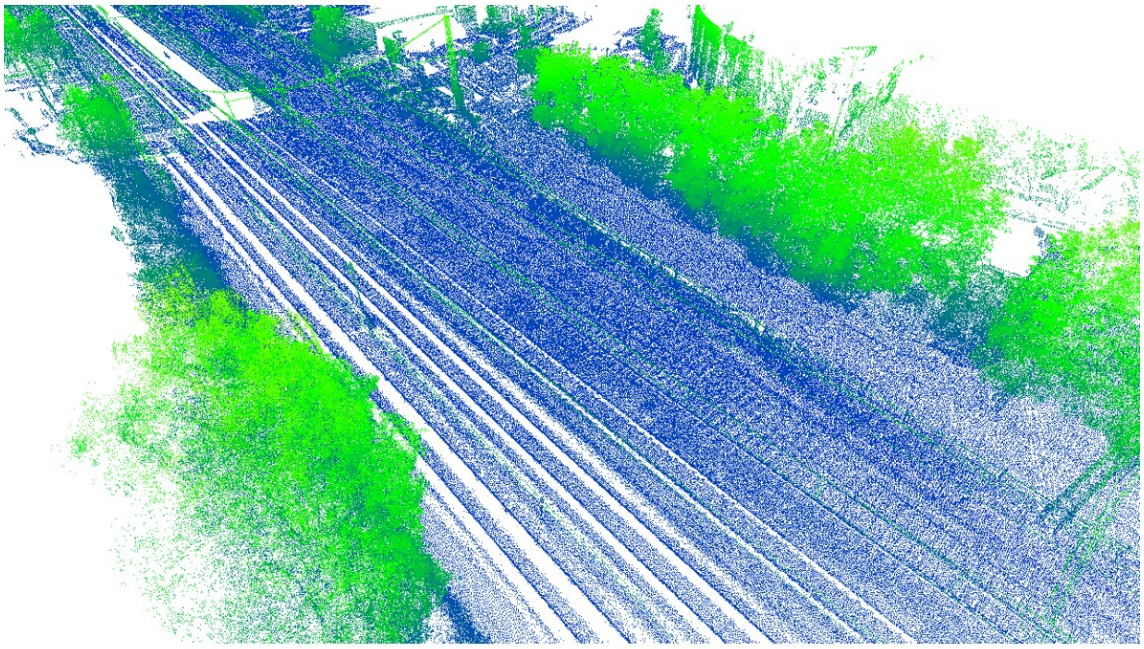
Le premier résultat présenté sur la figure 1.30 illustre la gestion de niveaux de détail décimaux. On peut voir sur la figure qu'elle permet une transition plus douce à l'écran, améliorant l'effet visuel au cours de la navigation. A noter que cette différence n'est observable qu'à des seuils t_{px} supérieurs à 0.5 pixels, puisqu'en dessous la résolution du nuage de points dépasse la résolution de l'écran. Cette remarque est cependant à nuancer par le fait que la navigation à de tels niveaux de seuils n'est envisageable que sur des configurations très performantes étant données qu'elles induisent un nombre de points rendus beaucoup plus élevé, d'où l'intérêt de la méthode proposée.

1.3.3.2 Temps de construction

La figure 1.31 présente les résultats obtenus sur 4 jeux de données différents. Ces jeux de données ont été choisis pour montrer que la structure proposée permet de



(a) Niveaux de détail entiers



(b) Niveaux de détail continus

FIGURE 1.30 – Gestion des niveaux de détail. (a) Gestion par niveaux entiers, on remarque la transition entre les niveaux. (b) Gestion par niveaux décimaux, la transition est lissée. Le seuil t_{px} a été choisi à 2 pixels pour accentuer l'effet.

traiter une large variété de scènes allant du scan laser fixe au scan laser mobile. Les temps de calculs associés au traitement de chaque scène ont été concaténés dans le tableau 1.2.

Les fréquences de calcul obtenues sont équivalentes aux fréquences d'acquisition des lidars modernes : entre 1 et 2 Mpts/s selon la complexité géométrique du jeu de données. Les temps de calcul ont été comparés sur les deux plus petits jeux de données à l'algorithme de conversion utilisé par le logiciel Potree (PotreeConverter). L'algorithme proposé est au moins deux fois plus rapide. A noter que les deux logiciels tournent sur un seul cœur de calcul.

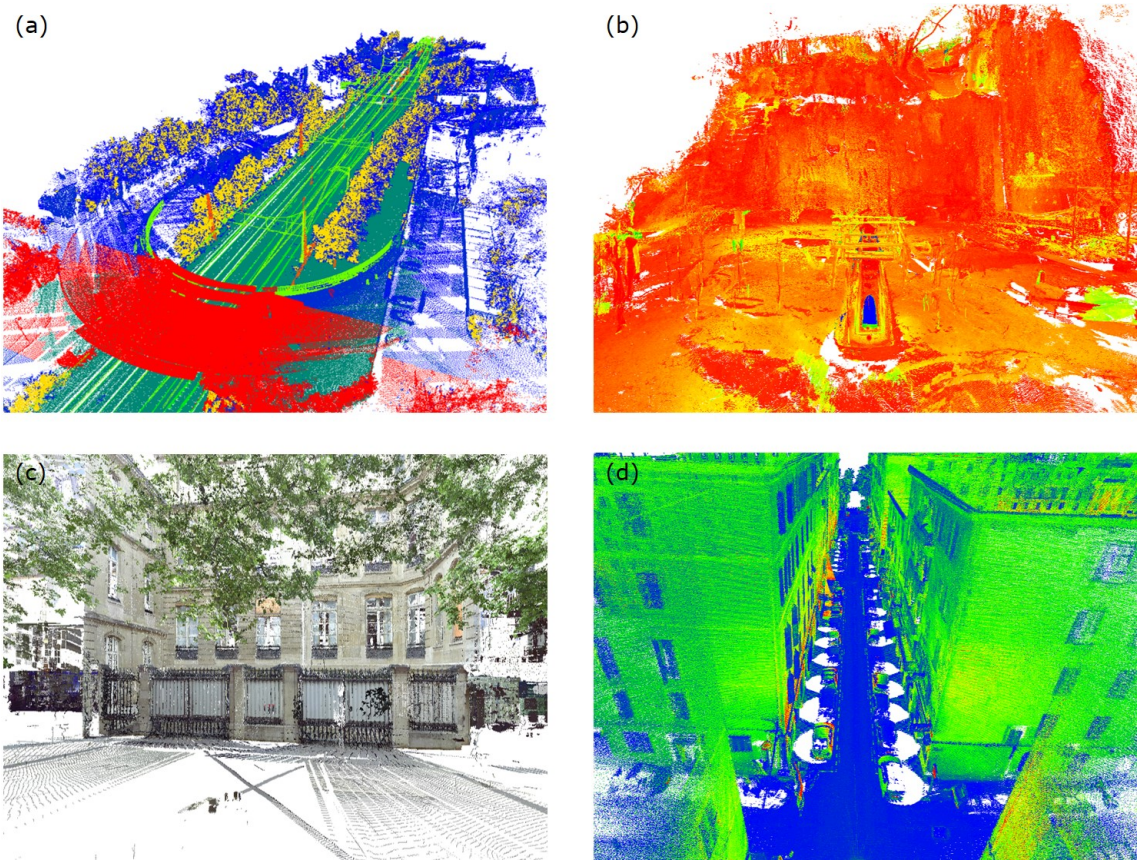


FIGURE 1.31 – Jeux de données utilisés pour illustrer la structure de données. (a) Nuage de points d'une portion de voie de chemin de fer obtenu par scan laser mobile. (Ce nuage de points a été gracieusement fourni par RIEGL Laser Measurement Systems). (b) Nuage de point d'une tombe étrusque, acquis par scan laser fixe. (c) Nuage de points de l'école des Mines ParisTech acquis également par scan laser fixe. (d) Nuage de points du centre-ville d'Ajaccio. Le nuage de points a été acquis par scan laser mobile à l'aide du prototype expérimental L3D2 du laboratoire de robotique de l'école des Mines ParisTech.

Jeu de données	RIEGL rail	Scalina	Mines ParisTech	Ajaccio
Nombre de points	35.5 M	227.4 M	2,218.2 M	1,891.8 M
Dimension	716.5 m	119 m	403 m	2.9 km
Champs*	ICL	IC	C	I
Taille	0.57 Go	3.55 Go	8.23 Go	32.7 Go
Taille de feuille	5 mm	1 cm	2 cm	2 cm
Traitement	**32.9 s**	**102.8 s**	1078.1 s	1684.1 s
Traitement (Potree)	66.4 s	715.4 s	-	-

TABLE 1.2 – Résultats obtenus pour la construction de quatre jeux de données.
* Champs contenus dans le nuage de points : **I :** intensité laser, **C :** couleur, **L :** classe (obtenue par l'algorithme de segmentation classification (ROYNARD et al., 2016)). (configuration : Intel Core i7-7700K 4.2 GHz, 64 Go RAM, Nvidia GeForce GTX 1080)

1.3.3.3 Rendu

La Figure 1.32 illustre l'évolution du nombre de points rendus en fonction du seuil t_{px}. On remarque que ce seuil permet d'influencer le nombre de points rendus par image et de la même manière la charge du GPU, influençant directement la durée de l'étape de projection géométrique. Il est intéressant de noter que la courbe présente un seuil, atteint à partir de $t_{px} = 1.5px$. Cela signifie que au-delà de ce seuil c'est le surcoût algorithme de recherche qui devient prépondérant par rapport au calcul de projection perspective sur le GPU.

Le tableau 1.3 synthétise les performances de l'algorithme sur les différents jeux de données. On peut noter que ses performances sont globalement stables et cohérentes avec les limites fixées en introduction. Si l'on regarde en détail, les jeux de données "Scalina", "Mines ParisTech" et "Ajaccio" présentent les durées maximales plus élevées que le jeu "RIEGL rail". Cela peut s'expliquer par leur configuration géométrique : ils présentent tous les trois des portions denses. Par exemple, le jeu de données "Mines ParisTech" est un bâtiment avec plusieurs pièces scannées. Ainsi, lorsqu'une pièce est dans le tronc de la caméra, elle est toujours envoyée au GPU même si elle est cachée derrière un mur. Il s'agit là d'une limitation inhérente aux méthodes de structuration basées rasterisation qui ne prévoient pas d'*occlusion culling*.

Jeu de données	RIEGL rail	Scalina	Mines ParisTech	Ajaccio
durée min (ms)	0.7	0.8	1.9	3.1
durée max (ms)	2.4	6.6	8.3	7.5
durée moyenne (ms)	1.8	4.5	5.5	4.5

TABLE 1.3 – Performances de l'algorithme de structuration mesurées en divers points de vues des quatre jeux de données. Les durées mesurées sont celles nécessaires pour calculer une image par projection perspective avec 1 pixel par point. Ces valeurs ont été obtenues avec $t_{px} = 1.0px$ et une résolution de 1208×720 (configuration : Intel Core i7-7700K 4.2 GHz, 64 Go RAM, Nvidia GeForce GTX 1080)

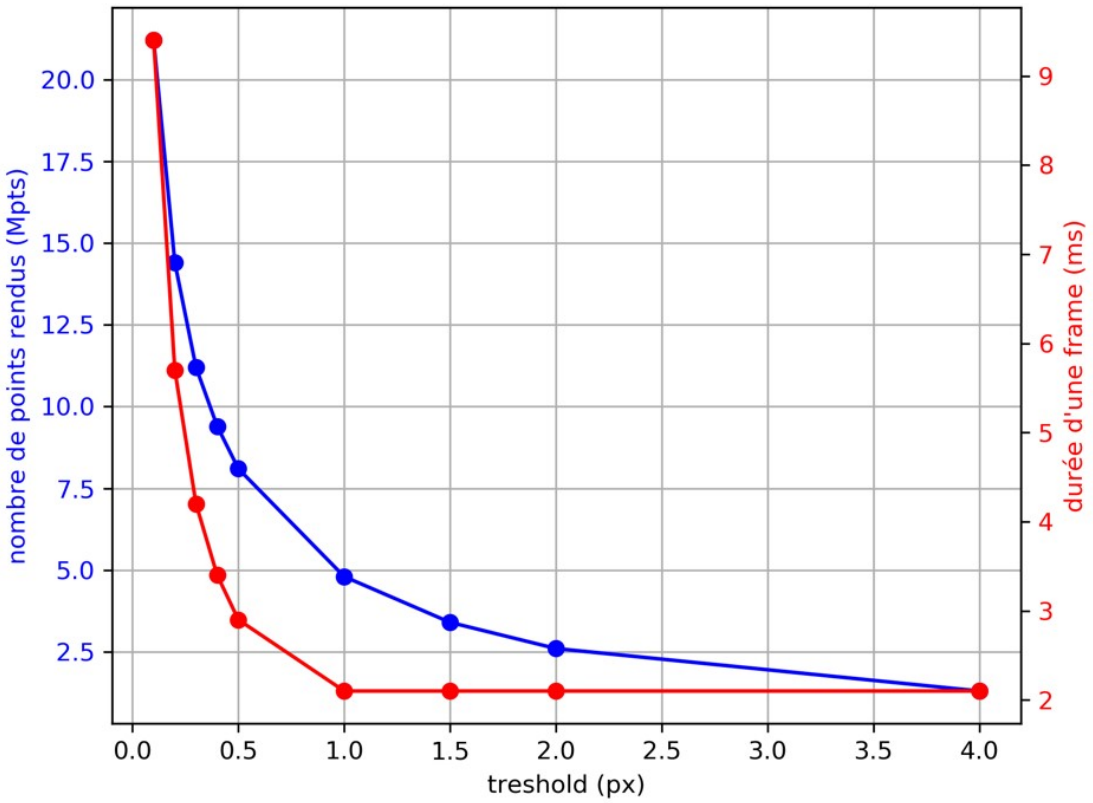


FIGURE 1.32 – Évolution du temps de calcul nécessaire pour calculer la passe géométrique d'une image (en rouge) et du nombre de points affichés (en bleu) en fonction du seuil t_{px}. Les valeurs ont été mesurées pour une vue statique du nuage de points RIEGL rail (configuration : Intel Core i7-7700K 4.2 GHz, 64 Go RAM, Nvidia GeForce GTX 1080).

1.4 Rendu de nuages de points par opérateurs pyramidaux en espace image

Nous présentons dans cette partie une nouvelle méthode de rendu de nuages de points basée sur des traitements en espace image. Cette méthode est basée sur un enchaînement d'opérateurs pyramidaux. Le terme pyramidal fait référence à l'utilisation de pyramides d'images. Ce point permet d'atteindre des performances de rendu élevées et indépendantes de la complexité de la scène ainsi que du point de vue.

1.4.1 Pipeline

La méthode proposée se base sur le pipeline décrit sur la figure 1.33.

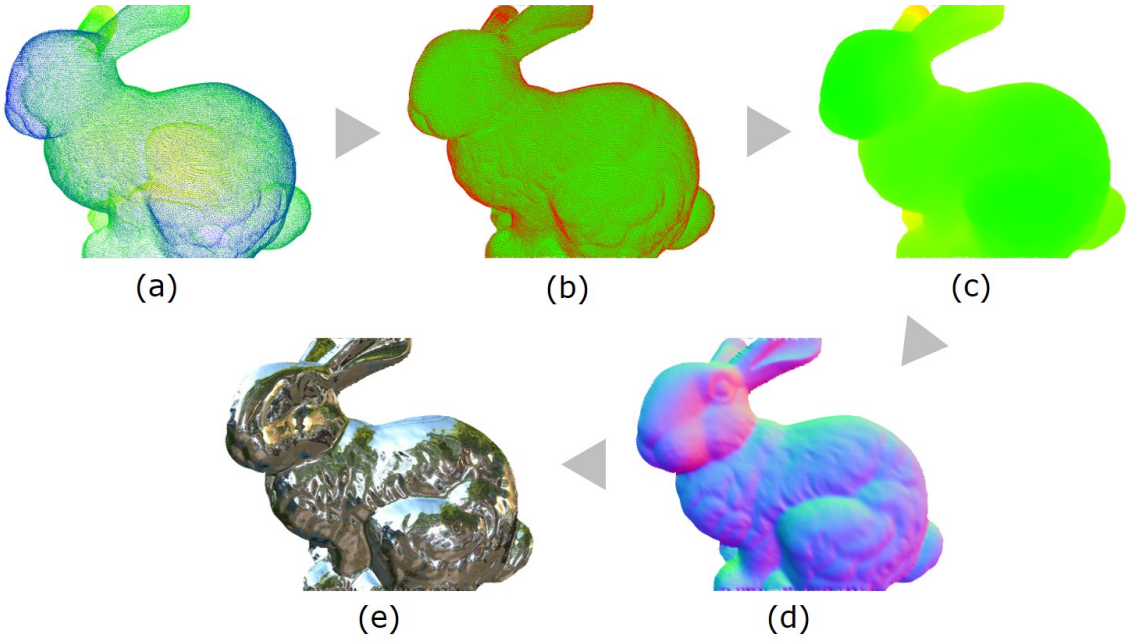


FIGURE 1.33 – Vue d'ensemble du pipeline proposé. (a) Unique passe géométrique avec un pixel par point. (b) Opérateur de retrait des parties cachées, en rouge : points du premier plan visibles, en vert : points cachés. (c) Reconstruction de la profondeur et des attributs. (d) estimation des normales. (e) Ombrage.

La méthode nécessite comme donnée d'entrée un *framebuffer*[15] qui contient une texture de profondeur et en option des textures d'attributs (comme par exemple une texture de couleur). Ce *framebuffer* est obtenu par une unique passe de rendu géométrique qui projette en vue perspective et rasterise par un pixel chaque point

15. Le terme *framebuffer* correspond à un ensemble de textures dans lesquelles peuvent écrire les *fragment shaders* à la suite d'un appel au pipeline de rendu.

du nuage. Les calculs relatifs à cette étape sont décrits dans l'annexe A. Il convient de noter que le fait de ne nécessiter qu'une seule passe géométrique (Schütz et M. Wimmer, 2015) couplé au rendu par un pixel par point est un avantage du pipeline proposé. En effet, les nuages de points sont composés de nombreuses primitives géométriques (typiquement plusieurs millions), la passe géométrique associée est ainsi coûteuse, d'où l'intérêt des méthodes qui ne nécessitent qu'une seule passe géométrique par opposition à celles qui en nécessitent plusieurs comme par exemple la méthode de splatting de (M. Botsch et al., 2005). Par ailleurs, le fait de rendre par un pixel par point réduit les accès concurrents en écriture au même pixel par plusieurs *fragment shaders*.

L'étape suivante est un opérateur de retrait des parties cachées. Elle consiste, d'une part à retirer les points rendus qui sont supposés être occultés, et d'autre part à labelliser les trous à remplir pour l'étape suivante. Cette dernière consiste justement à les remplir par un opérateur pyramidal. S'en suit alors une étape de calcul des normales. A ce stade, le tampon géométrique (ou *G-Buffer*) obtenu contient pour chaque pixel : une couleur, une profondeur et un vecteur normal. On peut alors par la suite appliquer les méthodes classiques d'ombrage.

Le seul paramètre nécessaire pour l'ensemble de la méthode est une échelle métrique s_0 donnée par l'utilisateur. Celle-ci doit être de l'ordre de grandeur de l'échantillonnage du nuage de points.

1.4.2 Retrait des parties cachées

La première étape du pipeline consiste, étant donné un point de vue, à retirer les parties cachées du nuage de points. Ce problème est illustré par le point (2) sur la figure 1.4, ainsi que sur la figure 1.33.(a). On peut voir sur cette dernière par transparence la deuxième face du Stanford Bunny.

L'opérateur décrit dans cette section est uniquement basé sur des calculs en espace image sur la texture de profondeur. Il s'applique à l'image entière : aussi bien aux pixels du premier plan que ceux d'arrière-plan. Son principe est décrit sur la figure 1.34. Ainsi, de manière similaire à (Pintus et al., 2011), une valeur d'occlusion approximée est calculée à partir des pixels voisins du pixel considéré. Cependant, la méthode proposée présente deux avantages majeurs par rapport à (Pintus et al., 2011). Premièrement le voisinage est calculé de manière beaucoup plus efficace à partir d'une pyramide d'images, ce qui permet non seulement d'accélérer considérablement l'algorithme, mais également de détecter (et donc remplir par la suite) des trous d'une taille arbitrairement grande, ce qui permet de gérer avec un algorithme unifié les situations où le nuage de point est observé de très proche. Deuxièmement le voisinage est calculé de manière adaptative ce qui permet de produire de meilleurs résultats visuels pour les scènes qui comportent de grandes différences de profondeur. La valeur d'occlusion est alors comparée à un seuil afin de déterminer si le pixel est

visible ou pas. Le *framebuffer* est alors filtré en retirant les points du premier plan qui ne sont pas visibles.

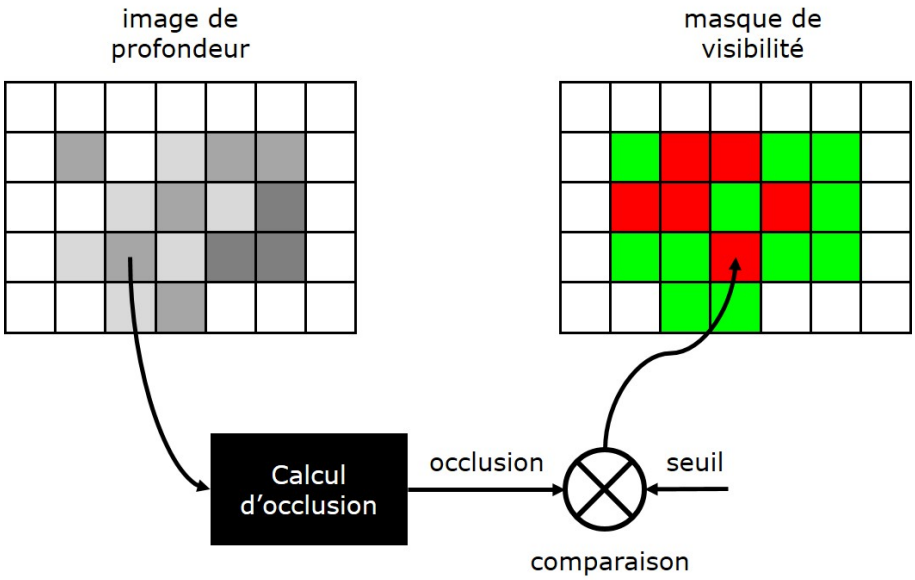


FIGURE 1.34 – Principe de l'opérateur de retrait des parties cachées par calcul d'occlusion.

1.4.2.1 Calcul d'occlusion approximée dans l'axe de la caméra

On suppose que l'on dispose d'une image de profondeur de dimensions (w, h), calculée à partir d'une étape de projection d'un nuage de points. On notera les valeurs de profondeur exprimées dans le repère de la caméra :

$$\begin{aligned} & z_{ij} \in [-f_c, 0] \\ & 0 \leqslant i < w, 0 \leqslant j < h \end{aligned} \tag{1.2}$$

Où f_c est la distance du plan *far* du tronc de projection au centre de la caméra virtuelle. Les valeurs de profondeur égales à f_c correspondent à l'*arrière-plan* de l'image, c'est à dire les pixels sur lesquels aucune géométrie n'a été projetée. On remarquera que les valeurs de profondeur sont négatives car on considère le repère canonique d'*OpenGL* [16].

On note $p_{ij} \in \mathbb{R}^3$ le point dans le repère caméra obtenu par l'opérateur de dé-projection décrit dans l'annexe A :

$$p_{ij} = \Pi^{-1}(i + 0.5, j + 0.5, z_{ij}) \tag{1.3}$$

Soit $f = (i, j)$ un pixel. On suppose que l'on dispose d'un ensemble de points $\mathcal{N} = \{n_k \in \mathbb{R}^3\}$. Ces points sont obtenus par dé-projection de pixels voisins du pixel

16. x pointe à droite, y pointe en haut et z pointe vers l'arrière

considéré. La méthode proposée pour les obtenir est discutée dans le paragraphe suivant. Nous décrivons dans la suite les calculs qui permettent d'obtenir $v(i,j,\mathcal{N})$ la visibilité du pixel f. On peut voir ce calcul comme une version formalisée et généralisée de l'opérateur proposé par (Pintus et al., 2011).

Soient $x, y \in \mathcal{R}^3$ deux points de l'espace dont les coordonnées sont exprimées dans le repère caméra. La visibilité du point x par rapport au point y est donnée par :

$$\omega(x,y) = \left(1.0 - \frac{y-x}{\|y-x\|} \cdot \frac{-x}{\|x\|}\right) \tag{1.4}$$

Cette quantité est proportionnelle (à un facteur 2π près) à l'angle solide délimité par le cône de sommet x, d'axe $-\vec{x}$ et qui contient le point y. Elle est comprise entre 0 et 2. Une valeur proche de 2 signifie que x est très peu occulté par y dans l'axe de la caméra. A l'opposé, une valeur proche de 0 signifie qu'il est très occulté dans l'axe de la caméra. La figure 1.35 schématise cette situation.

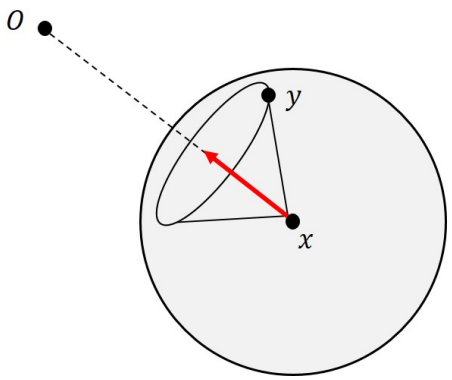


Figure 1.35 – Illustration de l'angle solide décrit par l'équation 1.4.

Afin de calculer $v(i,j,\mathcal{N})$ il faut alors combiner les différentes valeurs de visibilité $\omega(p_{ij}, n_k)$. Pour cela, une méthode simple consiste à diviser l'espace image autour du pixel f en N_s secteurs angulaires réguliers (typiquement $N_s = 8$).

Soit n_k un voisin de f dont les coordonnées sont $\{i+k, j+l\}$, l'index du secteur auquel il appartient, noté $\theta(n_k)$, est donné par l'équation suivante :

$$\begin{aligned} \theta(n_k) &= \left\lfloor \frac{N_s \psi(k,l)}{2\pi} \right\rfloor \\ \psi(k,l) &= \begin{cases} \alpha(k,l) & \text{si } \alpha(k,l) \geqslant 0 \\ \alpha(k,l) + 2\pi & \text{sinon} \end{cases} \\ \alpha(k,l) &= \operatorname{atan2}(l+0.5, k+0.5) \end{aligned} \tag{1.5}$$

La figure 1.36 donne un exemple de répartition des indices autour d'un pixel donné. Il convient de noter que le calcul de l'indice par l'équation 1.5 est assez lourd,

une optimisation consiste à pré-calculer ces valeurs dans une table de correspondance stockée dans une texture.

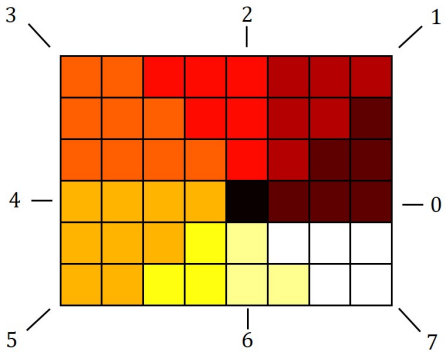


FIGURE 1.36 – Exemple de répartition des indices des secteurs autour du pixel noir pour $N_s = 8$.

Soit alors $\omega_m(i, j, \mathcal{N})$ la visibilité minimale sur chaque secteur, définie par :

$$\forall 0 \leqslant m < N_s,\ \omega_m(i, j, \mathcal{N}) = \min \{\omega(p_{ij}, n_k) \mid \theta(n_k) = m,\ n_k \in \mathcal{N}\} \tag{1.6}$$

La visibilité du pixel f est alors obtenue en moyennant ces valeurs sur l'ensemble des secteurs :

$$v(i, j, \mathcal{N}) = \frac{1}{N_s} \sum_{m=0}^{N_s-1} \omega_m \tag{1.7}$$

Un pixel est alors labellisé comme non visible si cette valeur dépasse un seuil, typiquement une valeur de 0.1 donne expérimentalement de bons résultats sur l'ensemble des tests réalisés. Par ailleurs, on peut remarquer que cette approche peut être vue comme un calcul d'occlusion ambiante basé horizon (BAVOIL et SAINZ, 2008), où la normale est remplacée par le vecteur point-caméra.

1.4.2.2 Calcul de voisinage pyramidal et adaptatif

L'opérateur de (PINTUS et al., 2011) est basé sur un voisinage exhaustif et de taille fixe autour de chaque pixel, ce qui pose deux problèmes majeurs :

- L'exhaustivité du voisinage implique que pour un rayon r_0 (en pixels) $(2r_0+1)^2$ pixels sont visités, ce qui rend l'opérateur très lent pour des grandes valeurs de r_0. Cela limite donc son utilisation à de petites valeurs de r_0 qui ne permettent pas de détecter des trous arbitrairement grands, en particulier lorsque l'on observe le nuage de points de près.
- La taille fixe du voisinage donne des résultats de faible qualité visuelle pour des scènes qui comportent de grandes différences de profondeur.

Nous introduisons donc dans ce paragraphe une nouvelle méthode pour calculer un voisinage pyramidal (et donc non exhaustif) et adaptatif.

Calcul d'une pyramide de nuages de points en espace image La première étape consiste à construire une hiérarchie de nuages de points à partir de la texture de profondeur en entrée. Pour ce faire, cette texture est d'abord déprojettée dans le repère caméra. Le résultat est alors stocké dans une texture (x, y, z) de nombres flottants. Cette texture correspond donc au niveau 0 de la hiérarchie de la pyramide.

Par la suite, étant donné un niveau l, chaque pixel (i^l, j^l) est obtenu en gardant le point avec la profondeur la plus petite en valeur absolue dans le voisinage $\left\{2i^l, 2i^l + 1\right\} \times \left\{2j^l, 2j^l + 1\right\}$ dans la texture au niveau $l-1$. Ce processus de construction est illustré sur la figure 1.37 et exemple de résultat obtenu est donné sur la figure 1.38.

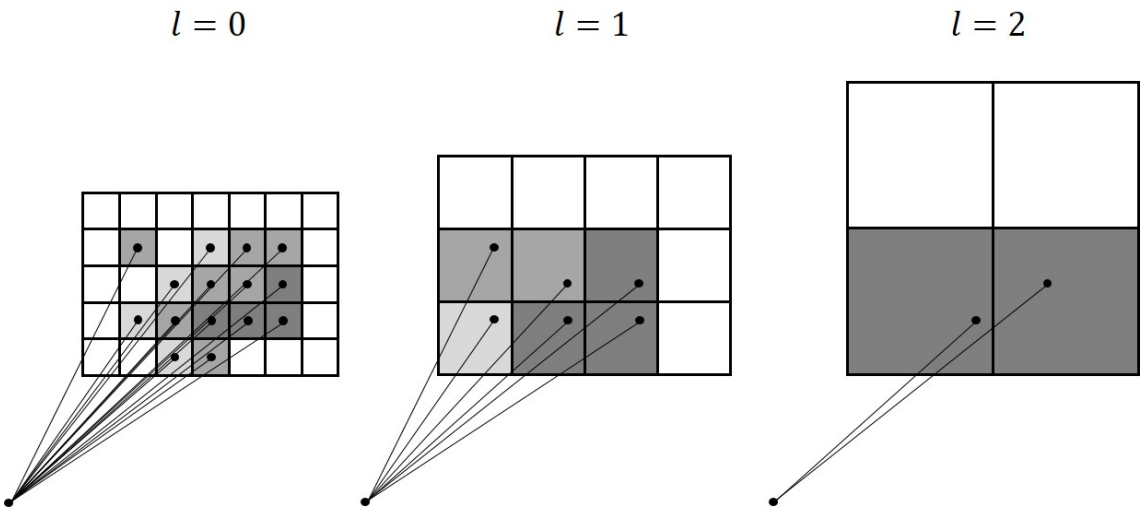


FIGURE 1.37 – Schéma de construction de la pyramide de nuages de points en espace image.

Calcul de voisinage pyramidal Cette pyramide contient alors une représentation multi-résolution du tampon de profondeur. Elle peut être interprétée comme un moyen efficace d'agréger l'information portée par des pixels voisins.

Ainsi, étant donné un pixel $\{i, j\}$, son voisinage multi-résolution dans la pyramide est composé, pour chaque texture au niveau l, des voisinages 3×3 centrés sur le pixel $\left\{\left\lfloor \frac{i}{2^l} \right\rfloor, \left\lfloor \frac{j}{2^l} \right\rfloor\right\}$ auxquels on a retiré ce pixel central. Ce schéma de construction est décrit sur la figure 1.39. Ce schéma peut contenir des points redondants d'un niveau à l'autre ce qui n'est pas un problème dans la mesure où le calcul d'occlusion garde uniquement le minimum de visibilité sur chaque secteur. On peut également remarquer que le pixel central de chaque niveau est effectivement inutile dans la mesure où il est systématiquement inclus dans les niveaux supérieurs.

Ce calcul de voisinage permet de réduire considérablement le nombre de points visités, comparativement à l'utilisation d'un voisinage exhaustif. Ainsi étant donné un rayon de voisinage r_0 en pixels la complexité de l'algorithme (assimilable au

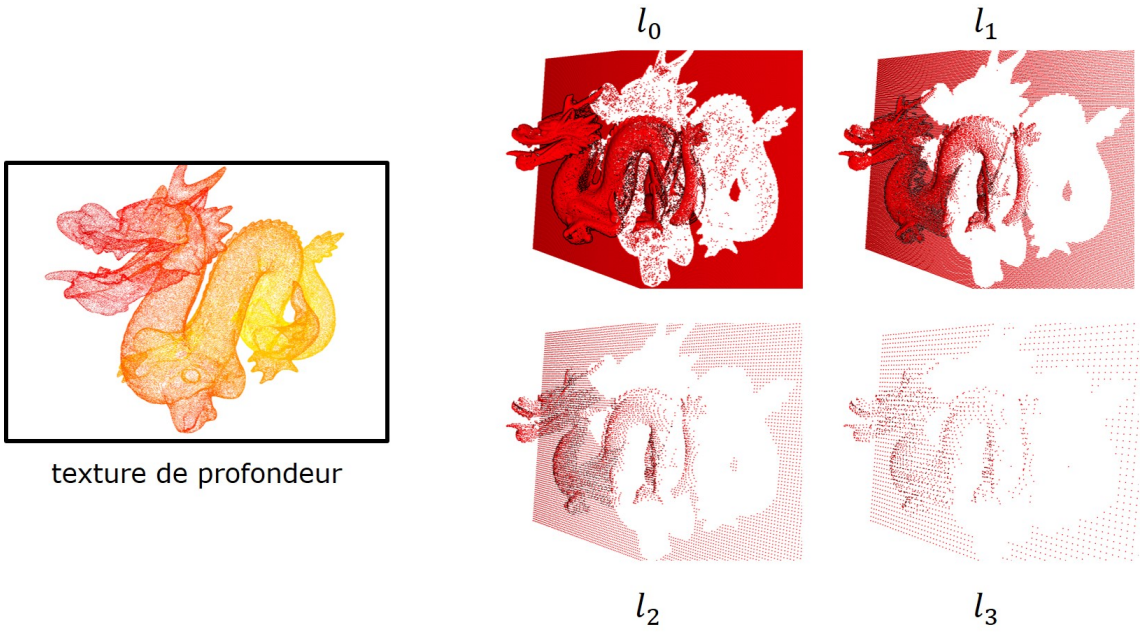


FIGURE 1.38 – Exemple de pyramide de nuage de points obtenue à partir de la texture de profondeur à gauche. Seuls les 4 premiers niveaux sont représentés ici. Le nuage de points est visualisé par un point de vue différent de celui de la caméra avec l'*eye-dome ligthing* du logiciel CloudCompare afin de renforcer les contours.

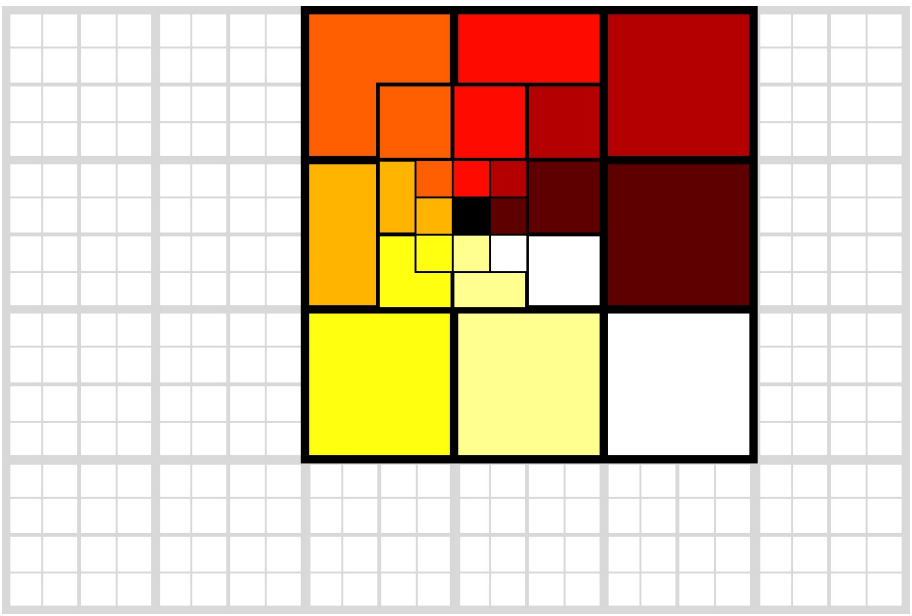


FIGURE 1.39 – Schéma de calcul de voisinage pyramidal autour du pixel noir sur 3 niveaux de pyramide. Les grilles associées à chaque niveau sont représentées avec des traits d'épaisseurs différentes. Les couleurs représentent l'index du secteur associé au pixel.

nombre de points visités dans le voisinage) exhaustif (Pintus et al., 2011) est $\sim r_0^2$ alors que celle de la méthode proposée est $\sim log(r_0)$.

Calcul de voisinage adaptatif La taille du voisinage influe sur le résultat de l'opérateur de visibilité de la manière suivante :

- Une grande taille de voisinage permet de détecter des trous de grande taille, ce qui présente un intérêt pour les parties de l'image qui sont proches de la caméra. En contrepartie, cela a pour effet de sur-détecter des trous dans les parties lointaines de l'image du fait de la baisse de résolution liée à la projection perspective. Cela a également pour effet de détecter plus de trous au niveau de la jonction entre le nuage de points et l'arrière-plan.
- Une petite taille de voisinage produit donc de meilleurs résultats pour les parties lointaines de l'image mais ne permet pas de boucher efficacement les trous proches de la caméra.

Cette analyse met en évidence le besoin d'avoir recours à un compromis, en particulier pour les scènes qui présentent de grandes différences de profondeur. Pour résoudre ce problème nous proposons d'avoir recours à un voisinage de taille adaptative. Ainsi, comme à ce stade du pipeline nous ne disposons pas encore d'une carte de profondeur reconstruite, nous utilisons une carte de profondeur de basse qualité z_c qui est obtenue en prenant la coordonnée z d'un niveau de la pyramide précédemment construite. En pratique le 4ème niveau de la pyramide donne de bons résultats dans la plupart des cas. La figure 1.40 en donne un exemple.

Ainsi, pour un pixel $\{i, j\}$ dont la profondeur grossière interpolée bilinéairement est $z_c(i, j)$, on obtient le rayon adaptatif r_{ij} et le niveau adaptatif l_{ij} suivants :

$$r_{ij} = \frac{s_{hpr} h}{2tan(\theta)} \frac{1}{z_c(i, j)} \tag{1.8}$$

$$l_{ij} = log(r_{ij})/log(2) \tag{1.9}$$

Où h est la hauteur en pixel du *viewport*, θ le demi angle du champ de vue vertical. s_{hpr} est un paramètre métrique qui peut être interprété comme étant la dimension d'un carré placé à une distance $z_c(i, j)$ de la caméra dans l'espace métrique de la scène. En pratique les expériences ont montré qu'une valeur de $10s_0$ donne de bons résultats pour une large variété de scènes. r_{ij} correspond donc au rayon d'un voisinage exhaustif et l_{ij} correspond à la profondeur équivalente dans la pyramide de nuages de points.

Un effet secondaire de l'utilisation d'un voisinage adaptatif est une légère amélioration en performance due au fait que la diminution du rayon de voisinage dans certaines parties de l'image induit une baisse du nombre de voisins à visiter.

La figure 1.41 montre les limites de l'utilisation d'une taille de voisinage fixe sur l'ensemble du *framebuffer*. Elle montre que l'utilisation d'une taille trop petite induit une sous-estimation des parties cachées au premier plan de la scène. Ce constat n'est

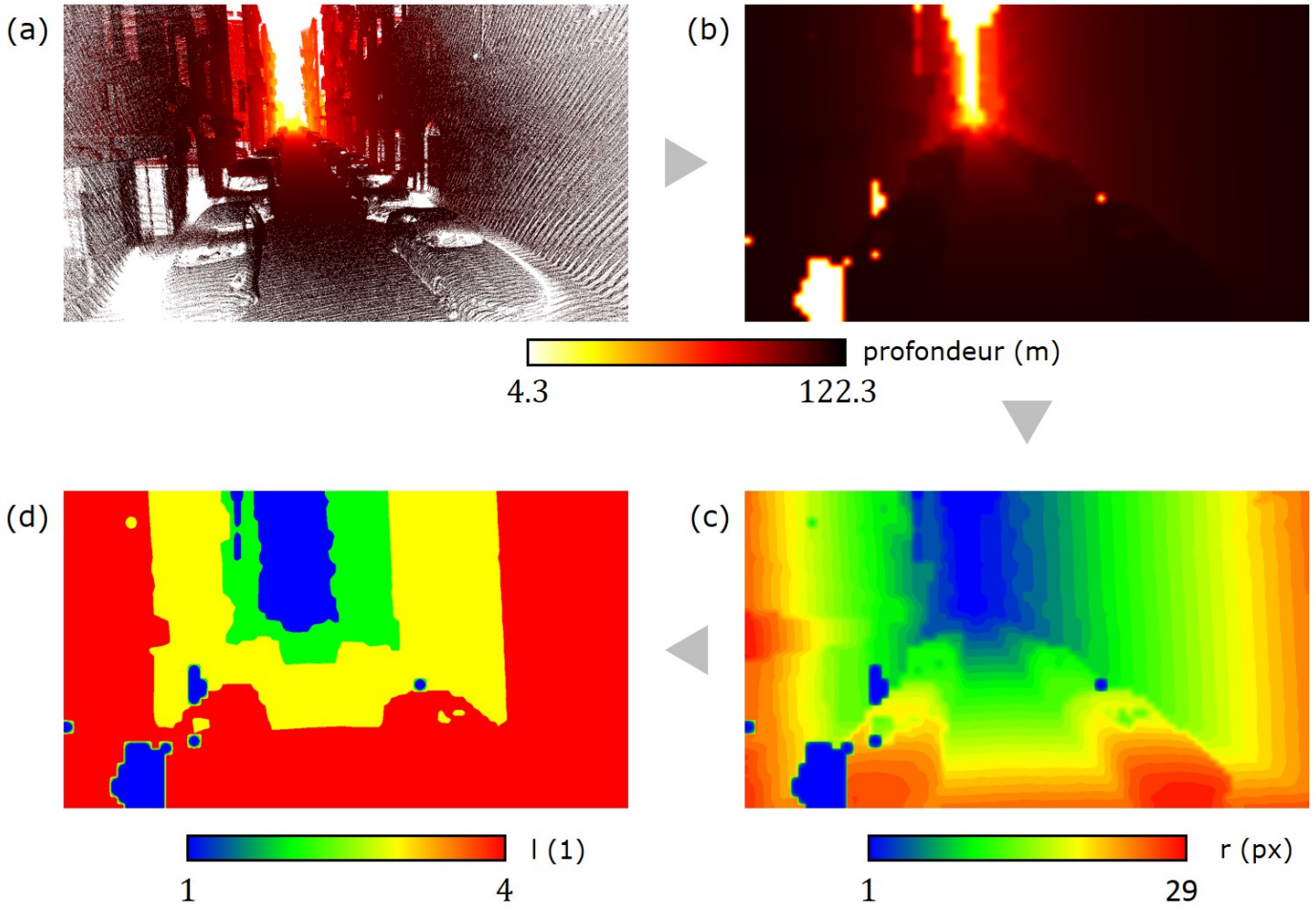


FIGURE 1.40 – Processus de calcul du niveau de voisinage adaptatif. (a) Tampon de profondeur initial. (b) profondeur des points au 4ème niveau de la pyramide interpolée bilinéairement. (c) Rayon de voisinage entier en pixels. (d) Niveau de voisinage pyramidal correspondant.

pas étonnant compte tenu du fait que la taille du voisinage définit la taille maximale des "trous" que la méthode permet de trouver. En revanche, une taille de voisinage trop grande induit une surestimation des parties cachées, notamment sous le pont au fond de la scène. L'utilisation d'un voisinage de taille adaptative permet de régler ces deux problèmes. Cet effet est exacerbé sur les scènes qui présentent de larges différences de profondeur, comme c'est le cas sur la figure 1.41 où les points proches du point de vue sont à quelques centimètres alors que le fond de la scène se situe à 200m du point de vue.

1.4.3 Remplissage par reconstruction en espace image

Après avoir retiré les parties cachées de la scène par l'opérateur présenté dans la section précédente, la deuxième étape du pipeline consiste à remplir les parties manquantes du *framebuffer*. Pour cela, nous nous basons sur l'algorithme du *pull push*.

Cet algorithme, introduit par (Gortler et al., 1996) et amélioré par (Kraus, 2009) a déjà été utilisé dans le contexte du rendu de nuage de points (Grossman et Dally, 1998 ; Marroquim et al., 2008). Cependant ces deux méthodes ne traitent pas le problème de la reconstruction d'un *framebuffer* complet (profondeur et attributs). Ils l'utilisent en effet uniquement pour reconstruire une couleur obtenue par un ombrage appliqué en chaque point à l'aide de normales. Par ailleurs (Marroquim et al., 2008) utilisent également les normales dans la phase de reconstruction. C'est pourquoi l'utilisation de cet algorithme pour le rendu de nuages de points bruts représente une contribution.

1.4.3.1 Remplissage par pull-push

Le *pull-push* est un algorithme d'interpolation de données éparse en 1D ou en 2D qui se décompose en deux phases. La première (*pull*) consiste à construire une pyramide d'images dont la résolution est récursivement divisée par deux. Durant cette phase, les pixels d'une image à un niveau donné sont synthétisés à partir de ceux de l'image au niveau plus fin par une moyenne pondérée sur une fenêtre de taille donnée. La deuxième (*push*) étape consiste à reconstruire l'information manquante (pixels de poids faible) de manière récursive en partant du niveau le plus grossier de la pyramide.

Nous utilisons ainsi la variante proposée par (Kraus, 2009). Soit x une quantité scalaire ou vectorielle échantillonnée sur une image. On note alors x_i^l le pixel d'index i au sein du niveau l d'une pyramide d'images. Le niveau 0 étant le niveau le plus fin et le niveau $L-1$ le plus grossier. Les dimensions des images à chaque niveau sont notées $\{W_l, H_l\}$. Par souci de simplicité, l'algorithme est décrit en 1D, en 2D la formulation est identique. Ces notations sont décrites sur la figure 1.42.

Pour la phase de *pull* (ou sous-échantillonnage), le niveau x^0 correspond à l'image de départ. Dans notre cas, il s'agit des composantes du frambuffer (profondeur et

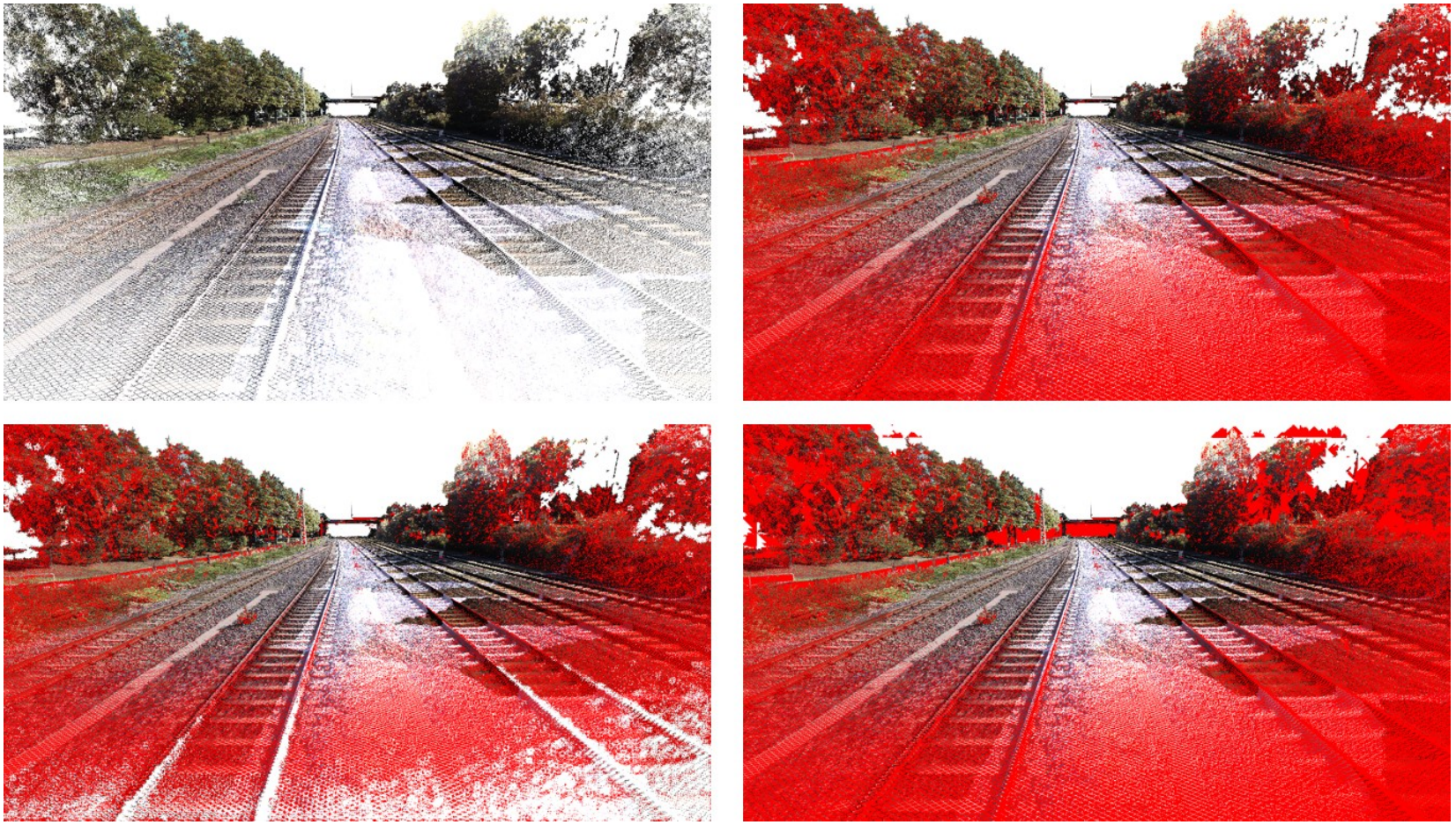


FIGURE 1.41 – Comparaison du voisinage fixe et du voisinage adaptatif. Les points cachés sont représentées en rouge. (En haut à gauche) Image initiale. (En haut à droite) voisinage adaptatif en utilisant la méthode proposée. (En bas à gauche) Voisinage fixe de rayon 5 pixels en utilisant (PINTUS et al., 2011). (En bas à droite) Voisinage fixe de rayon 25 pixels en utilisant (PINTUS et al., 2011).

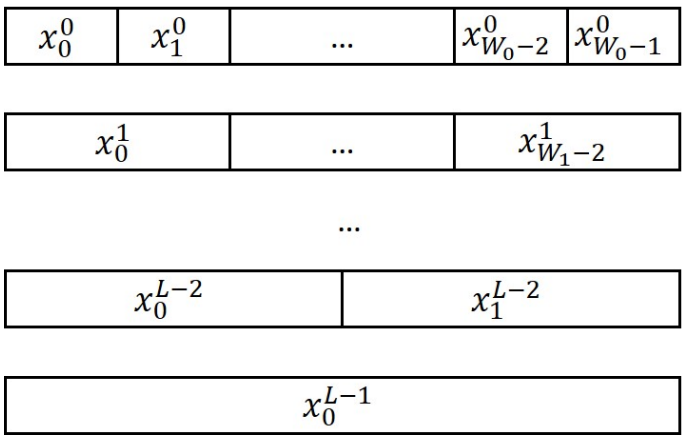


FIGURE 1.42 – Notations utilisées pour décrire les pyramides d'images dans l'algorithme de *pull-push* illustrées en 1D. (Figure inspirée de (KRAUS, 2009)).

attributs) après retrait des parties cachées. On y associe un poids w^0 qui vaut 1 pour les pixels du premier plan et 0 pour ceux de l'arrière-plan. On définit le filtre $\widetilde{h} = \left\{\widetilde{h}_{-1}, \widetilde{h}_0, \widetilde{h}_1, \widetilde{h}_2\right\}$. Différents choix peuvent être pris pour la taille de ce filtre ainsi que ces valeurs. En pratique nous prendrons pour la suite $\widetilde{h} = \left\{\frac{1}{3}, 1, 1, \frac{1}{3}\right\}$. En effet, l'expérience montre qu'un filtre de taille 4 produit de meilleurs résultats sur l'interpolation de la profondeur par rapport à un filtre de taille 2, comme utilisé par (MARROQUIM et al., 2008). L'influence sur l'interpolation de la couleur est en revanche moins marquée. La pyramide est alors construite à partir des équations suivantes (illustrées par la figure 1.43) :

$$\begin{aligned}
\widehat{w}^0 &= \min\left(w^0, 1\right), \ \widehat{x}^0 = \widetilde{w}^0 x^0 \\
w_i^{l+1} &= \sum_{k=-1}^{2} \widetilde{h}_k \widehat{w}_{2i+k}^l \\
\widehat{w}_i^{l+1} &= \min\left(w_i^{l+1}, 1\right) \\
\widetilde{x}_i^{l+1} &= \frac{\widehat{w}_i^{l+1}}{w_i^{l+1}} \sum_{k=-1}^{2} \widetilde{h}_k \widetilde{x}_{2i+k}^l
\end{aligned} \tag{1.10}$$

La phase de *push* (ou sur-échantillonnage) consiste à remonter la pyramide à partir des niveaux les plus bas pour reconstruire les niveaux les plus hauts. Pour cela on utilise un filtre $h = \{h_{-1}, h_0, h_1, h_2\}$. Dans ce cas, le choix du filtre est beaucoup plus important que pour la phase précédente. En effet, (KRAUS, 2009) montrent par l'expérience que le filtre $\left\{\frac{1}{4}, \frac{3}{4}, \frac{3}{4}, \frac{1}{4}\right\}$ qui correspond à une interpolation par B-spline quadratique, donne les meilleurs résultats. Les équations correspondantes (illustrées par la figure 1.44) sont les suivantes :

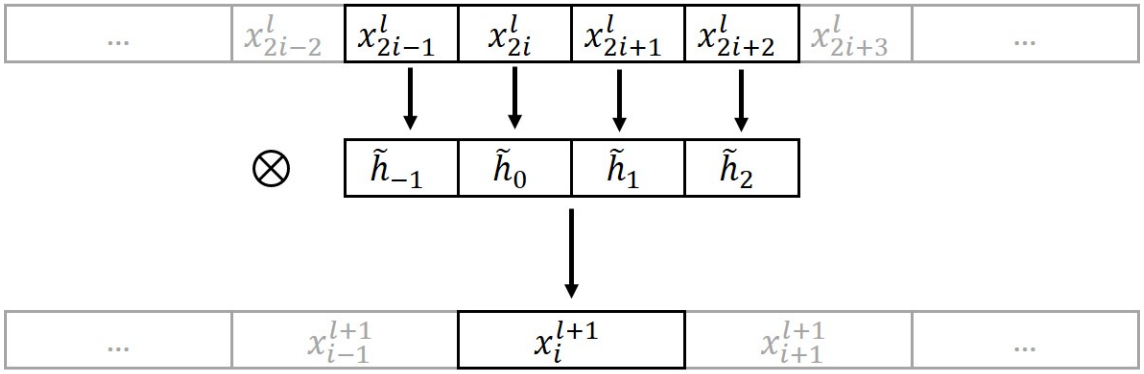


FIGURE 1.43 – Phase de *pull* de l'algorithme de *pull-push*.

$$\begin{aligned} \chi^l_i &= \sum_k h_{i-2k} x^{l+1}_k \\ x^l_i &= \chi^l_i (1 - \widehat{w}^l_k) + \widetilde{x}^l_i \end{aligned} \tag{1.11}$$

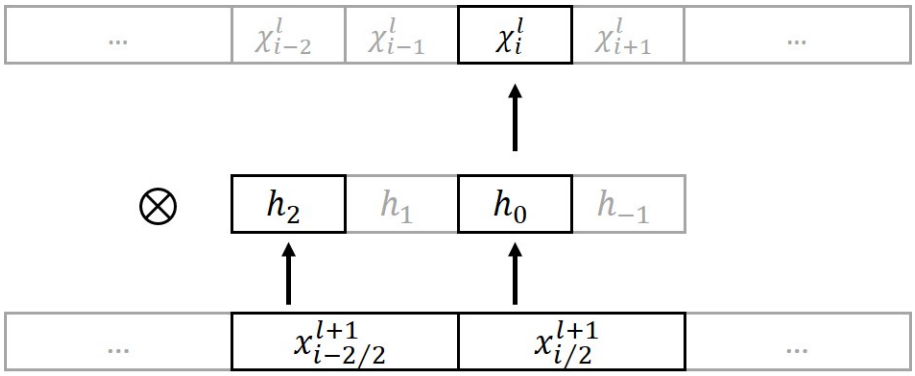


FIGURE 1.44 – Phase de *push* de l'algorithme de *pull-push* (illustré pour i pair).

La figure 1.45 illustre le procédé complet de *pull-push* sur un exemple concret. On remarque notamment que le poids tend rapidement vers la valeur 1 sur l'ensemble de l'image, ce qui est une des hypothèses prises par (KRAUS, 2009) pour établir les équations simplifiées 1.11.

1.4.3.2 Composition avec l'opérateur de retrait des parties cachées

Une différence importante par rapport aux différents articles de l'état de l'art qui mettent en œuvre une étape de remplissage après une passe de retrait des parties cachées (par exemple (PINTUS et al., 2011)) est que le remplissage est réalisé sur

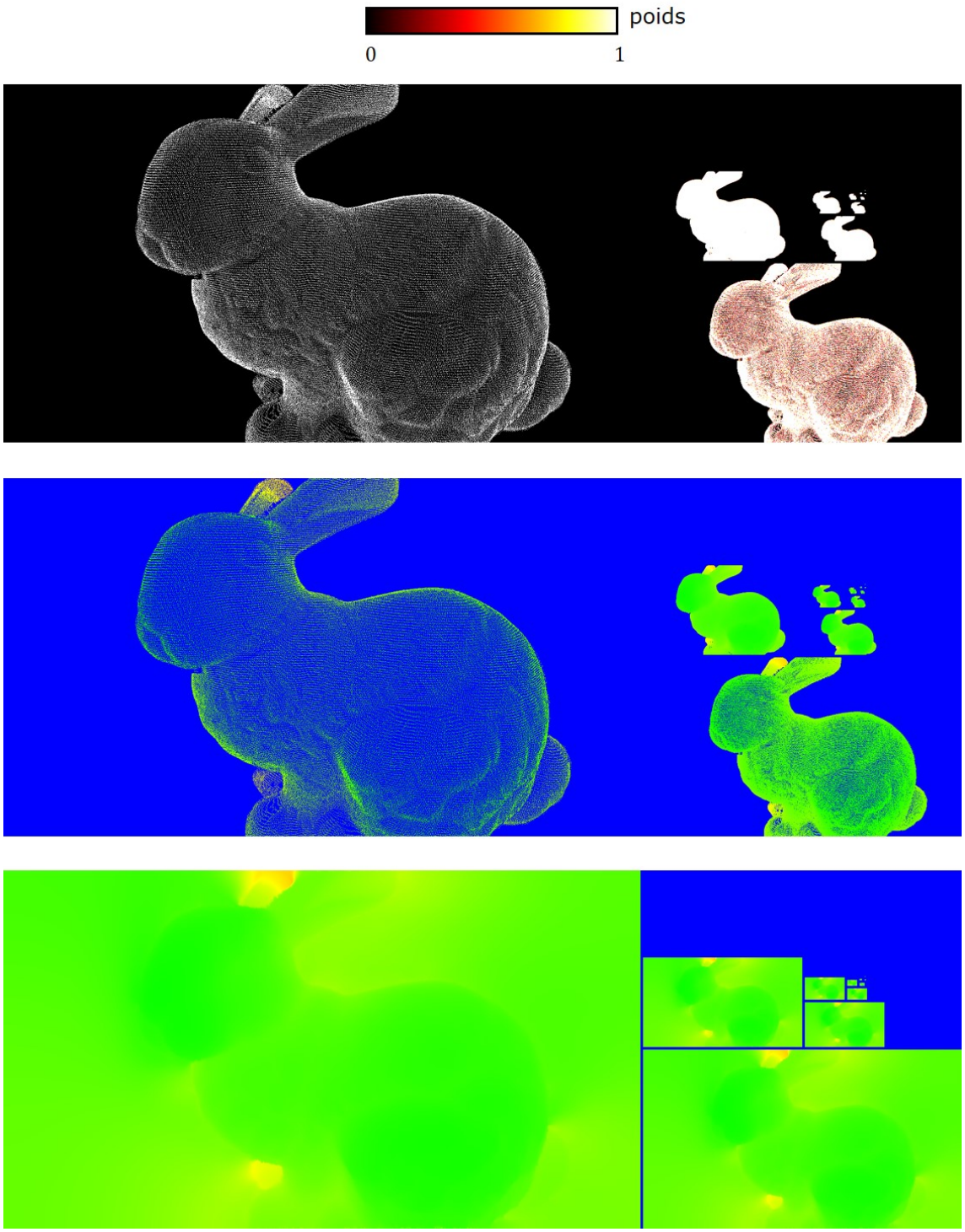


FIGURE 1.45 – (en haut) pyramide de poids $\widehat{w}$ dans la phase de *pull*. (au milieu) pyramide de $\widetilde{x}$ durant la phase de *pull*. (en bas) x durant la phase de *push*. Dans ce cas, x représente la profondeur du pixel.

l'ensemble de l'image et non pas uniquement sur les parties de l'image qui sont labellisées comme visibles par l'algorithme de visibilité. En pratique cela se traduit par le fait que le poids initial w^0 dans la phase de pull est choisi égal à 0 pour tous les pixels de l'arrière-plan (et non pas seulement pour les pixels non visibles).

Ainsi, l'image est complètement remplie après l'algorithme de remplissage. Afin de rétablir l'arrière-plan, il faut donc appliquer un masque d'arrière-plan en composant le masque de l'algorithme de visibilité et le masque du *framebuffer*. Ce masque présente les points de l'arrière-plan qui sont effectivement visibles après l'algorithme de visibilité. Il est illustré sur la figure 1.46. Ce masque est alors soustrait à l'image remplie pour obtenir la profondeur finale.

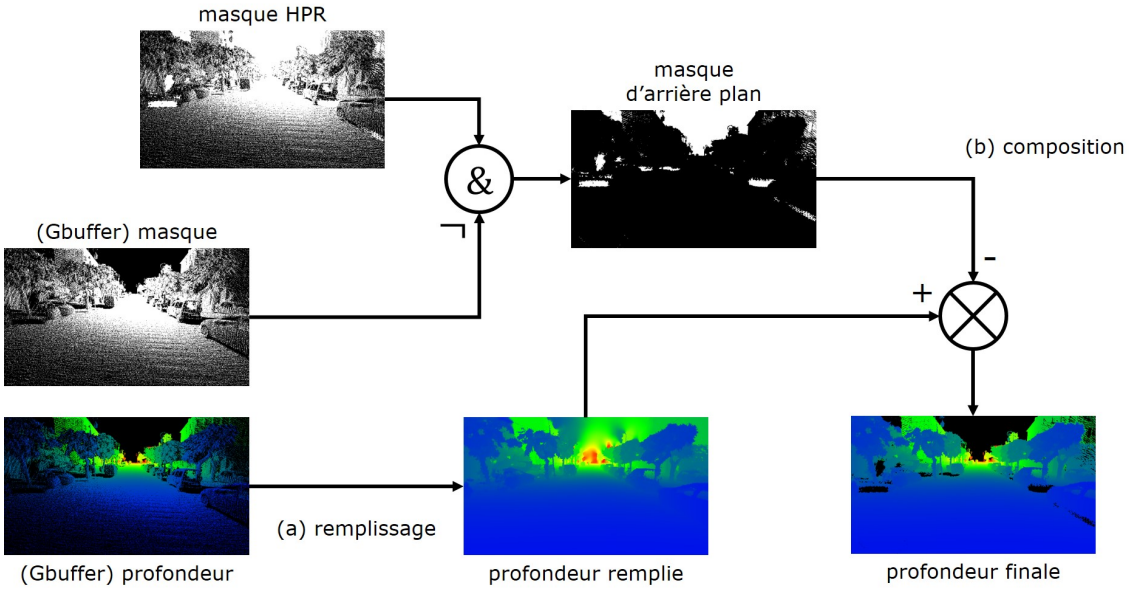


FIGURE 1.46 – Composition des résultats de l'algorithme de visibilité et de l'algorithme de remplissage.

Ce procédé permet notamment de reconstruire une surface cohérente même aux endroits où le filtre de visibilité a échoué pour détecter des trous, comme c'est le cas dans les régions de faible échantillonnage.

1.4.4 Calcul de normales en espace image et ombrage

Après les deux étapes précédentes : retrait des parties cachées et remplissage on obtient un *framebuffer* avec une texture de profondeur ainsi que des textures d'attributs remplies. Afin de pouvoir appliquer divers algorithmes de ré-éclairage (ou *shading*) classique il manque la donnée essentielle des normales à la surface. Il convient également de noter que la seule méthode de rendu de nuage de points bruts qui permet d'obtenir cette information est Auto-Splat (PREINER et al., 2012). Cette méthode est néanmoins très coûteuse en temps de calcul.

Il est possible de calculer des normales à partir d'une texture de profondeur remplie par un simple produit vectoriel. En notant $p_{i,j} \in \mathbb{R}^3$ le point 3D dans le repère de la caméra obtenu par déprojection du pixel dont la profondeur est z_{ij}, la normale n_{ij} peut être obtenue facilement par :

$$\begin{aligned} v &= [p_{i+1,j} - p_{i-1,j}] \wedge [p_{i,j+1} - p_{i,j-1}] \\ n_{ij} &= \frac{v}{\|v\|} \end{aligned} \tag{1.12}$$

Ce schéma simple pose un problème dans le cas du pipeline proposé pour deux raisons. La première réside dans le fait que les nuages de points en entrée peuvent être bruités, ce qui produit inévitablement des surfaces reconstruites bruitées puisque l'algorithme de pull-push est interpolant [17]. La deuxième réside dans l'algorithme de *pull-push* qui ne permet pas de reconstruire une surface lisse même lorsque les données d'entrée sont parfaites. Ce point est illustré sur la figure 1.47 où l'algorithme de remplissage est appliqué à un nuage de points synthétique de plan. Cet effet est d'autant plus important que les trous à remplir dans la surface sont importants. Il est donc nécessaire d'avoir recours à un algorithme de filtrage pour pallier à ce problème.

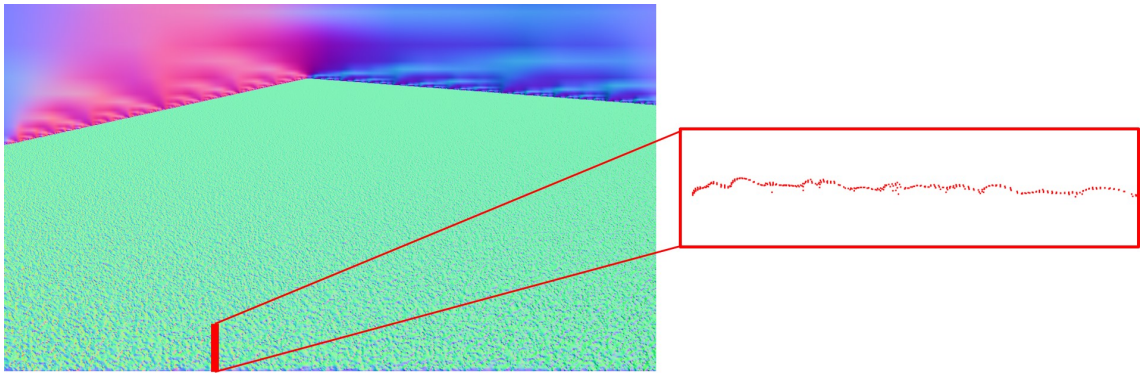


FIGURE 1.47 – Bruit généré par l'algorithme de pull-push. L'algorithme est appliqué à un nuage de points synthétique de plan (sans bruit ajouté). Les normales sont estimées directement à partir de la profondeur reconstruite. La coupe à droite montre le bruit sur la surface reconstruite qui cause les artefacts visibles sur les normales.

Une méthode pour résoudre ce problème serait d'utiliser l'algorithme 3D classique basé sur une analyse en composante principale des points dans le voisinage de chaque pixel (HOPPE et al., 1992). Cette approche pose cependant le problème d'être gourmande lorsque le voisinage est grand. Ainsi nous introduisons une nouvelle méthode efficace pour le calcul des normales basée sur des traitements pyramidaux.

Pour ce faire, nous calculons les normales sur la pyramide de texture de profondeur remplies obtenue durant la phase de *push* de l'algorithme précédent, à l'aide de

17. on pourrait imaginer un algorithme approximant en prenant des poids initiaux différents de 1, cependant cette piste semble donner des résultats de qualité moyenne.

la formule simple 1.12. On obtient ainsi une pyramide multi-résolution de normales. On remarquera que les niveaux les plus grossiers contiennent une information de normales lissée.

Ensuite pour chaque pixel de profondeur reconstruite z_{ij} on utilise l'équation 1.9 pour obtenir un niveau $l(z_{ij})$ décimal. La seule différence consiste à prendre un paramètre d'échelle s_n au lieu de s_{hpr} qui définit la taille adaptative de voisinage désirée. Ce paramètre peut être pris en première approximation de l'ordre de s_0. Ce niveau décimal permet de déterminer deux niveaux adjacents dans la pyramide :

$$l_{ij} \leqslant l(z_{ij}) \leqslant l_{ij} + 1 \tag{1.13}$$

La normale du pixel $\{i, j\}$ est alors obtenue par interpolation tri-linéaire, c'est à dire par interpolation linéaire des normales obtenues dans chaque niveau l_{ij} et $l_{ij} + 1$ par interpolation bilinéaire à la position du pixel $\{i, j\}$ projeté dans chaque texture grossière. Le résultat est une carte des normales lissée efficacement et de manière adaptative, ce qui permet de conserver les bords saillants du modèle initial tout en lissant le bruit. Un exemple de pyramide de normales et du résultat obtenu est donné sur la figure 1.48.

Les résultats d'ombrages donnés dans le paragraphe suivant sont obtenus par application de matcaps. Il s'agit de textures sphériques introduites par le logiciel ZBrush qui permettent simplement d'appliquer un effet d'ombrage prédéfini. Cette approche simple permet de montrer que l'algorithme présenté permet de reconstruire une surface de grande qualité (profondeur et normales) qui rend possible le rendu par ré-éclairage de modèles nuages de points bruts. Il serait néanmoins intéressant de mettre en œuvre d'autres algorithmes d'illumination globale comme le calcul d'occlusion ambiante en espace image (ou SSAO).

1.4.5 Résultats

1.4.5.1 Performances du retrait des parties cachées

Le tableau 1.4 montre que l'opérateur de visibilité introduit est au moins un ordre de grandeur plus efficace que l'état de l'art. Le temps de traitement est constant sur l'ensemble des jeux de données et est de l'ordre de grandeur du temps requis pour la passe géométrique.

1.4.5.2 Performances globales

La figure 1.49 donne le détail des performances de l'algorithme proposé. Les différentes valeurs mesurées correspondent aux vues présentées sur les figure 1.50, 1.51, 1.53 et 1.54. On peut tout d'abord observer que les performances du pipeline en espace image (c'est à dire l'ensemble des passes sauf la passe de projection) sont stables. La méthode proposée est donc bien indépendante de la complexité de la scène et du point de vue choisi. Le retrait des parties cachées est le traitement le

FIGURE 1.48 – (En haut à gauche) normales brutes calculées à partir du niveau 0 de la pyramide après reconstruction par l'algorithme de *pull-push*. (En haut à droite) normales lissées par notre opérateur pyramidal. (En bas) pyramide de normales calculée à partir de la pyramide reconstruite de l'algorithme de *pull-push*.

Jeu de données	passe géométrique	(PINTUS et al., 2011)	méthode de visibilité proposée
Scalina	1.4 ms	58 ms (25)	**2.7 ms**
Statue Asiatique	2.0 ms	21 ms (15)	**2.1 ms**
Ajaccio	11.4 ms	21.6 ms (15)	**2.3 ms**

TABLE 1.4 – Comparaison du temps de calcul par image de l'opérateur de retrait des parties cachées entre, PINTUS et al., 2011 (entre parenthèses le rayon équivalent en pixels), et la méthode proposée. Le temps de traitement de la passe géométrique est donné à titre indicatif. (configuration : Intel Core i7-7700K 4.2 GHz, 64 Go RAM, Nvidia GeForce GTX 1080)

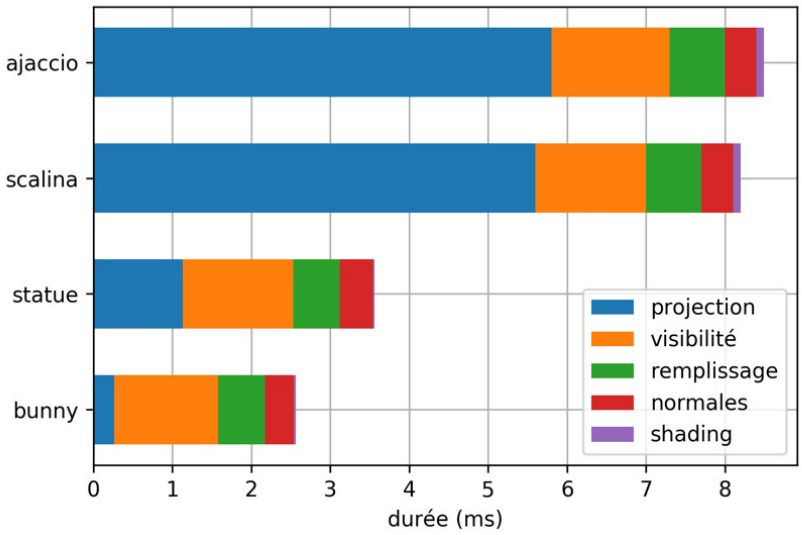


FIGURE 1.49 – Durées des différentes parties de l'algorithme proposé sur quatre jeux de données. (configuration : Intel Core i7-7700K 4.2 GHz, 64 Go RAM, Nvidia GeForce GTX 1080)

plus gourmand. On peut toutefois remarquer qu'il est du même ordre que les autres, ce qui n'est pas le cas des opérateurs de retrait des parties cachées de la littérature, comme nous l'avons montré plus haut. On peut remarquer que l'étape d'ombrage proposée est négligeable. En effet, l'ombrage par matcap est très simple car il revient à un accès à une valeur dans une texture. Enfin, les durées totales mesurées sont bien inférieures à la limite des 10 ms par image fixée en introduction, ce qui fait de la méthode proposée, une méthode performante et adaptée à des applications très demandeuses en ressources graphiques.

1.4.5.3 Rendu différé par ombrage de nuages de points bruts

Les figures 1.50, 1.51, 1.52, 1.53 et 1.54 illustrent la capacité de la méthode à gérer une large variété de nuages de points, allant du nuage de points d'objet au nuage de points bruité obtenu par scan laser mobile. Les images ont été rendues à une résolution de 1280×720 avec la configuration suivante : Intel Core i7-7700K 4.2 GHz, 64 Go RAM, Nvidia GeForce GTX 1080.

La qualité des normales permet de réaliser un ombrage par rendu différé (ou *deferred shading*) sur l'ensemble des nuages de points étudiés. Cette remarque est également valable pour des nuages de points acquis par scan laser mobile avec un lidar Velodyne comme c'est le cas sur la figure 1.54. Ce type de lidar produit des données bruitées, ainsi la méthode de lissage des normales proposée ici permet non seulement de lisser les artefacts produits par l'algorithme de pull-push, mais également l'éventuel bruit présent dans le modèle initial. La méthode permet en outre de

faire ressortir certains détails géométriques capturés lors de l'acquisition mais qui ne sont pas visibles par la variation de l'intensité du capteur. De plus, la figure 1.52 montre un exemple de rendu de nuage de points coloré (car acquis par photogrammétrie), ce qui démontre la capacité de la méthode à gérer des nuages de points qui contiennent des attributs supplémentaires.

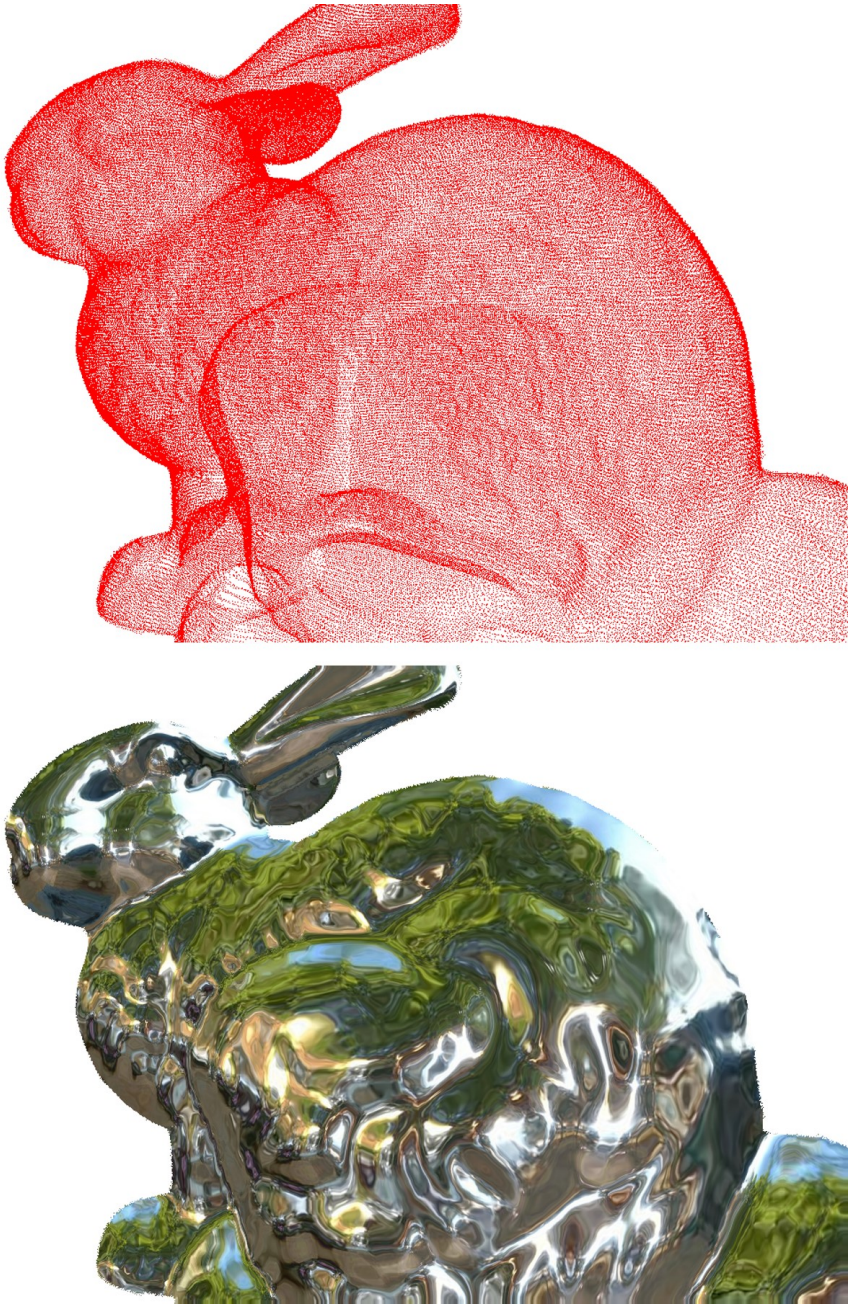


FIGURE 1.50 – Résultats de la méthode proposée sur le *Stanford Bunny*, 350k pts (En haut) résultat de la passe géométrique. (En bas) rendu par la méthode proposée.

FIGURE 1.51 – Résultats de la méthode proposée sur une statue Chinoise du 12ème siècle, acquise par photogrammétrie, 6M pts (En haut) résultat de la passe géométrique. (En bas) rendu par la méthode proposée

FIGURE 1.52 – Même modèle que la figure 1.51 rendu par la méthode proposée. Ici le rendu est réalisé par interpolation de la couleur de chaque point par l'algorithme de pull-push.

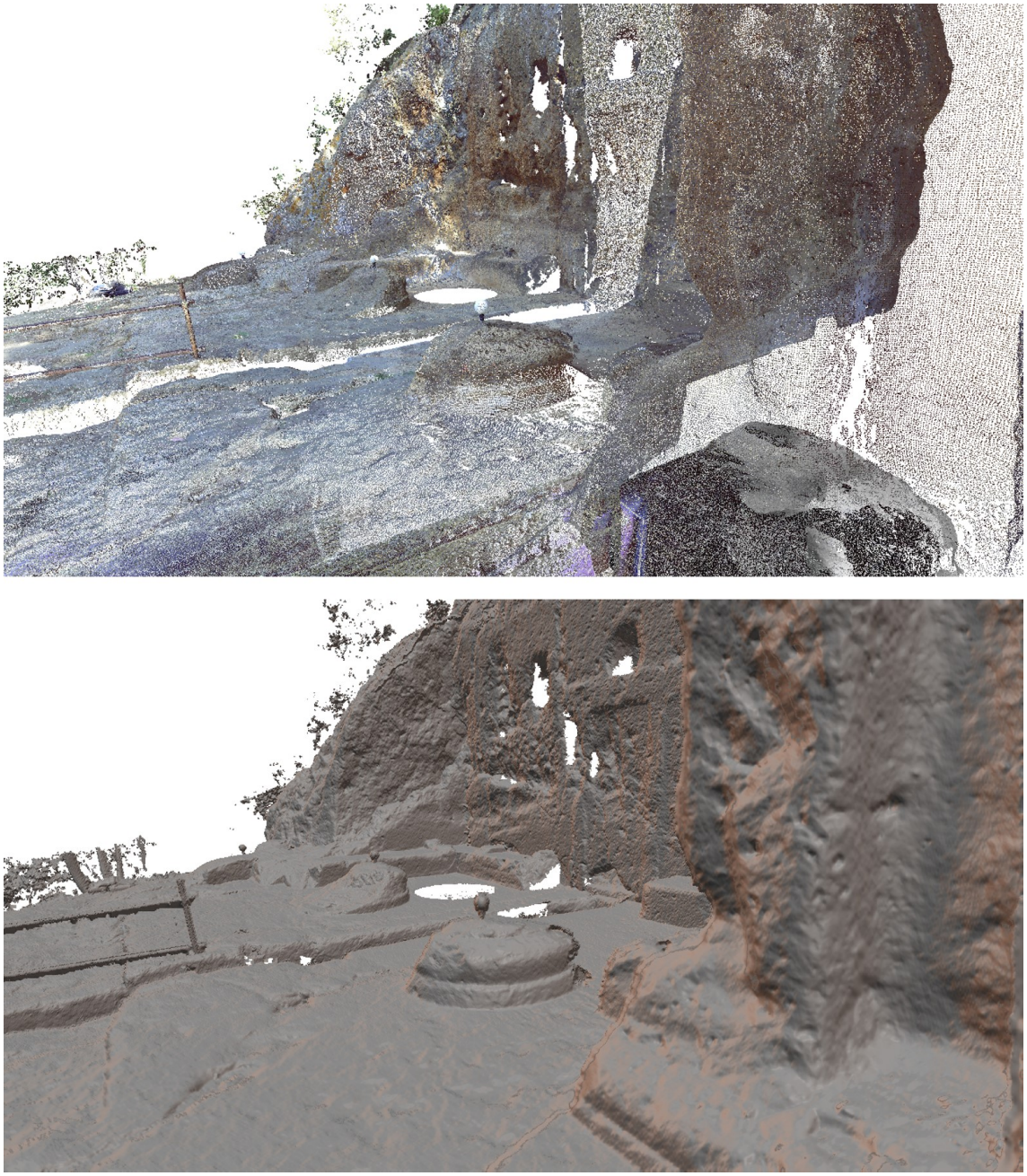


FIGURE 1.53 – Scan laser terrestre fixe de la tombe Étrusque Scalina, 73.6M pts. La méthode de structuration proposée dans la section précédente permet de réduire le nombre de points affichés durant la passe géométrique. (En haut) résultat de la passe géométrique. (En bas) rendu par la méthode proposée.

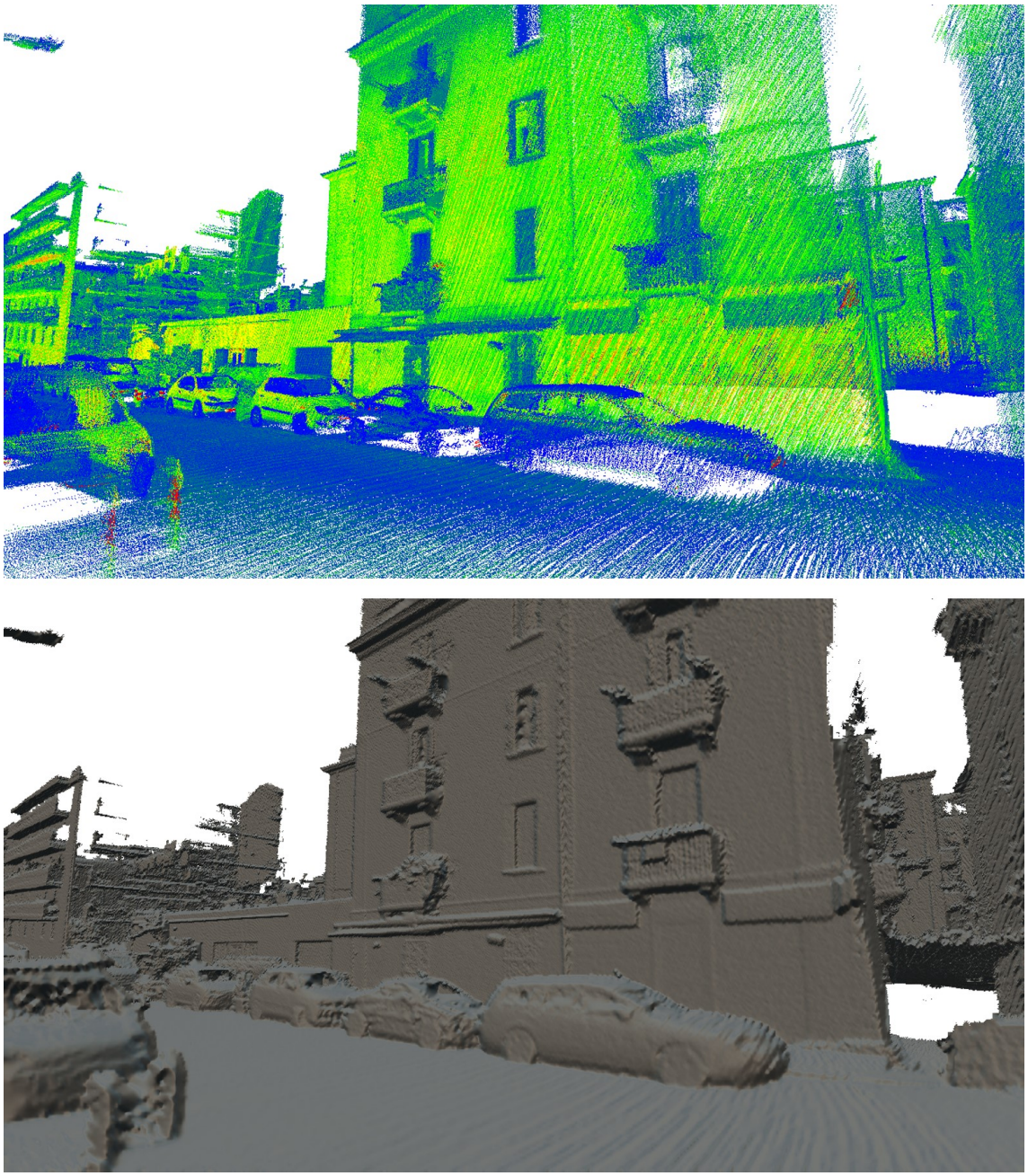


FIGURE 1.54 – Scan laser mobile de la ville d'Ajaccio, 2.7 milliards de points. le nuage de points présente un bruit de mesure de 2 à 3 cm. La méthode de structuration proposée dans la section précédente permet de réduire le nombre de points affichés durant la passe géométrique. (En haut) résultat de la passe géométrique. La couleur de chaque point correspond en fausses couleurs, à l'intensité de retour du laser. (En bas) rendu par la méthode proposée.

1.5 Conclusion

Nous avons proposé dans ce chapitre une nouvelle méthode de rendu de nuages de points. Cette méthode n'est basé que sur l'utilisation de l'information spatiale (x, y, z) de chaque point et ne requiert donc pas de calculer des normales sur l'ensemble du nuage de points affiché. Elle permet ainsi de rendre des nuages de points bruts. La méthode proposée est efficace, d'une part, grâce à l'utilisation d'une structure de données qui permet de limiter le nombre de primitives géométriques traitées par le GPU, et ce, de manière indépendante du nombre de points contenus dans le nuage de points. D'autre part, l'efficacité des traitements en espace image est liée à l'utilisation d'opérateurs pyramidaux. Ces derniers permettent de reconstruire en temps réel une surface et d'en estimer des normales, ce qui permet de réaliser en sortie de la méthode proposée un rendu par ombrage. Les performances de l'algorithme sont indépendantes de la complexité de la scène ainsi que du point de vue virtuel choisi, en particulier en exploration immersive. Par ailleurs, l'utilisation d'opérateurs adaptatifs, permet de gérer les scènes qui présentent de grandes différences de profondeur.

Les limites actuelles de la méthode résident d'une part dans la capacité de la structure de données à gérer le retrait des points par occlusion (*occlusion culling*). Les récentes méthodes de l'état de l'art qui sont basées sur du lancer de rayons sur GPU pourraient permettre de dépasser cette limitation. Du côté de la méthode de rendu, les limites principales résident dans la présence de légers scintillements au niveau des bords du modèle. La prise en compte de la cohérence temporelle pourrait y apporter une solution.

Finalement la méthode proposée permet d'atteindre des fréquences d'affichage très élevées en comparaison avec les méthodes de l'état de l'art. Ce dernier point pourra certainement ouvrir de nouvelles applications du rendu de nuages de points bruts comme par exemple l'utilisation dans les casques de réalité virtuelle, qui nécessitent des fréquences d'affichage de 120 Hz.

A titre d'illustration des applications industrielles possibles de la méthode proposée, la figure 1.55 montre un prototype de simulateur de conduite ferroviaire développé durant cette thèse.

FIGURE 1.55 – Vue du prototype de simulateur de conduite ferroviaire réalisé dans le cadre du stage de Raphaël Groscot encadré durant cette thèse. Ici le moteur de rendu de nuages de points est interfacé avec l'environnement de simulation Unity, dans lequel a été créé un scénario avec un train et des panneaux de signalisation virtuels.

Chapitre 2

Simulation numérique en mécanique des fluides à partir de nuages de points

2.1 Introduction

2.1.1 Cadre général

Dans de nombreux cas, la simulation numérique de phénomènes physiques tels que les écoulements de fluides, la propagation d'ondes acoustiques ou encore les transferts de chaleur, nécessite un modèle 3D de l'objet d'étude. Dans un contexte de conception industrielle, le modèle CAO de l'objet existe et peut être utilisé pour réaliser des simulations numériques. Néanmoins, lorsqu'il s'agit d'étudier des objets déjà existants, le modèle 3D de l'objet étudié doit être modélisé manuellement ce qui est une tâche chronophage et qui nécessite un savoir-faire technique pointu. C'est par exemple le cas pour l'étude de la ventilation dans les villes ou les bâtiments, ou la rétro ingénierie.

Les quarante dernières années ont connu l'essor des techniques de numérisation 3D. En particulier la cartographie mobile (F. Goulette et al., 2006) permet l'acquisition rapide, précise et détaillée d'environnements urbains à l'aide de véhicules circulant à vitesse normale dans le trafic. Les techniques de numérisation 3D représentent donc une alternative intéressante à la modélisation manuelle dans la mesure où elles permettent de produire des modèles 3D haute résolution. Néanmoins, ces techniques ne permettent pas d'obtenir directement un maillage surfacique variété et fermé de l'objet scanné. Au lieu de cela, elles produisent un ensemble de points bruités, non uniformément espacés, non connectés et échantillonnés sur sa surface que l'on appelle un nuage de points. Bien que celui-ci puisse être très dense, il ne comporte ni d'informations topologiques ni d'information de connectivité ce qui le rend inutilisable sous cette forme dans le cadre classique de la simulation numérique.

De plus les nuages de points acquis à l'aide des scanners actuels sont massifs dans la mesure où ces derniers produisent de l'ordre d'un million de points par seconde, ce qui ajoute une difficulté supplémentaire quant à leur utilisation.

2.1.2 Problématique

Le principal objectif de ce chapitre est de proposer une nouvelle méthode pour permettre d'utiliser les nuages de points comme géométrie pour définir les conditions aux limites dans les simulations numériques. Cela permettrait d'automatiser l'étape de modélisation manuelle requise autrement et ainsi permettre de nouveaux champs d'application pour simuler numériquement des phénomènes physiques autour des objets complexes. Bien que nous sommes convaincus que notre méthode pourrait s'appliquer à une large variété de simulations, nous avons fait le choix de nous concentrer sur la simulation d'écoulements transitoires de fluides incompressibles régis par les équations de Navier-Stokes en régime incompressible. Dans la suite de cet ouvrage, nous réfèrerons à la mécanique des fluides numérique par son acronyme anglophone CFD (pour *Computational Fluid Dynamics*).

La figure 2.1 illustre les différentes manières pour simuler un écoulement autour d'un objet réel. La première alternative, déjà présentée précédemment, consiste à créer un modèle CAO manuellement ou de manière semi-automatique à partir du nuage de points, des exemples plus concrets sont données dans la section 2.2. Cette méthode est fastidieuse et complexe, elle ne permet en général pas d'obtenir un modèle aussi fidèle et détaillé qu'un scan 3D. Cela peut s'expliquer par le fait que les modèles CAO sont construits à partir de primitives géométriques simples ou définies dans un espace paramétrique. Il s'agit donc d'une méthode limitée aux environnements structurés comme les bâtiments par exemple, ou aux objets industriels.

La figure 2.1 décrit également une autre alternative à la traditionnelle et fastidieuse modélisation CAO : un maillage surfacique peut être directement construit à partir d'un nuage de points, puis un maillage volumique peut être construit autour. Ce dernier est qualifié d'ajusté (ou *body-fitted* en anglais) puisqu'il ne couvre que la partie fluide du domaine d'étude. Ainsi la surface de l'objet se retrouve dans le bord du maillage volumique, ce qui permet de définir les conditions aux limites au niveau de la surface de l'objet (typiquement la vitesse est nulle sur la surface) de manière explicite dans le solveur utilisé pour calculer l'écoulement. Bien que de nombreux efforts aient été réalisés dans la communauté pour développer des méthodes pour construire automatiquement un maillage surfacique à partir d'un nuage de points (Hoppe et al., 1992 ; Amenta, Choi et al., 2001 ; Kazhdan, Bolitho et al., 2006) nous sommes convaincus qu'utiliser directement le nuage de points est bien plus prometteur que de passer par un maillage surfacique pour les raisons suivantes :

- Tout comme le nuage de points, le maillage surfacique est un choix pour représenter une surface de manière explicite. Néanmoins, dans notre cas le maillage surfacique, une fois construit, représente un choix fort sur la topologie de l'ob-

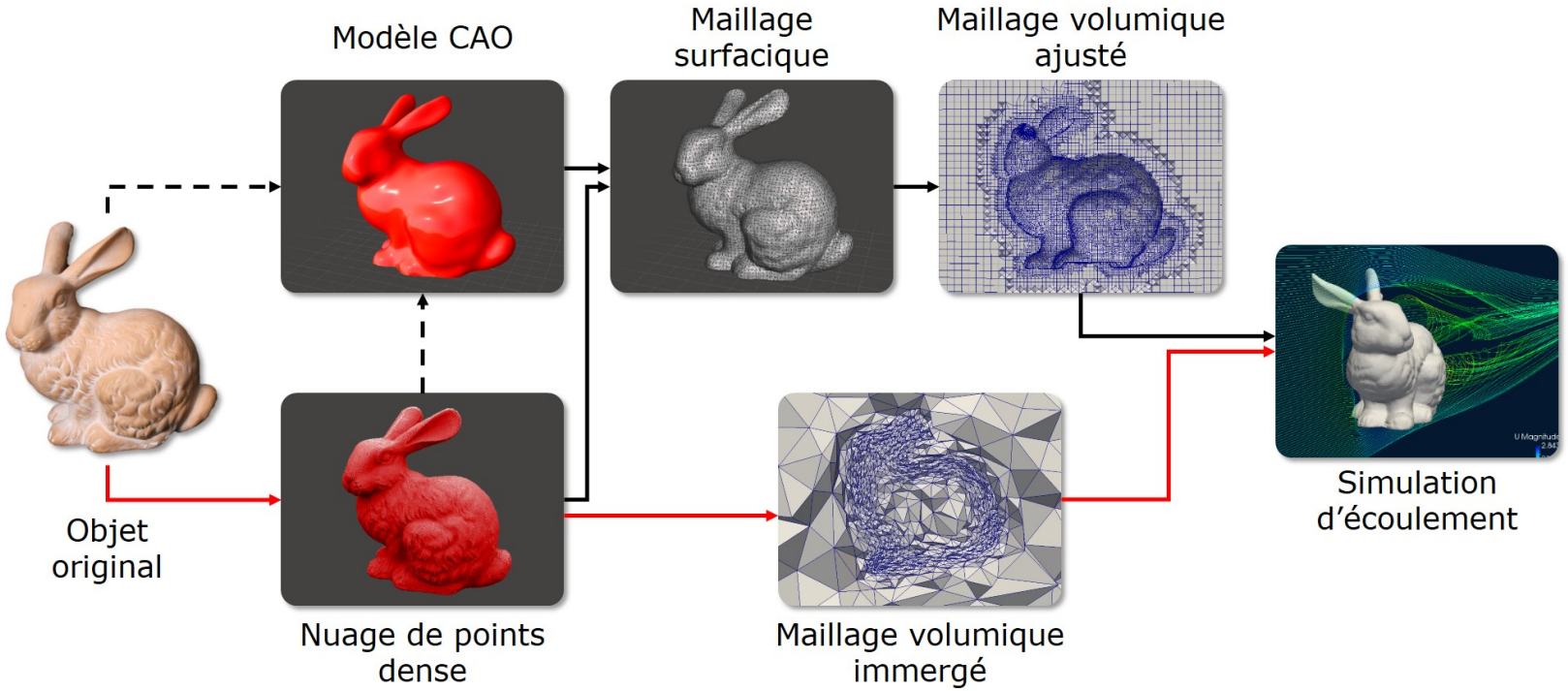


FIGURE 2.1 – Différentes méthodes pour simuler des écoulements autour d'objets réels. Les flèches en pointillés représentent les étapes manuelles ou semi-automatiques. Les flèches en **noir** représentent les *méthodes classiques*. Notre méthode est représentée par les flèches en **rouge**. Elle nécessite moins d'interaction de la part de l'utilisateur car le maillage volumique, qui sert de support pour la simulation numérique, est construit directement à partir du nuage de points original.

jet, alors que le nuage de points est la représentation la plus proche de la mesure étant donné qu'il s'agit de la sortie directe du processus de numérisation. De plus, pour les objets complexes, telles que les scènes en extérieur, les méthodes classiques de reconstruction de surface ne permettent pas de reconstruire de manière fiable un maillage cohérent.

- Les traditionnels maillages volumiques ajustés utilisés en CFD doivent être construits en respectant des règles précises au niveau de la forme et de la taille des cellules proches de la surface, afin de simuler au mieux les phénomènes de couche limite. En supposant qu'un tel maillage volumique est construit à partir d'un maillage surfacique, ces exigences doivent être également inférées dans le processus de construction du maillage surfacique, ce qui est possible lorsqu'il est construit à partir d'un modèle CAO mais beaucoup plus difficile lorsqu'il est construit à partir d'un nuage de points.

2.1.3 Contributions

Notre principale contribution est de proposer une méthode qui permet de simuler des écoulements autour de nuages de points 3D. Cette contribution représente un nouveau pas en avant vers l'automatisation de la simulation numérique. A notre connaissance, il n'existe pas de travaux similaires dans la littérature.

Cette contribution s'appuie sur deux points qui sont détaillés dans la suite de ce chapitre. Le premier point réside dans le fait que la seule représentation explicite de surface utilisée pour décrire la surface est un nuage de points 3D scanné. Comme on peut l'observer sur la figure 2.1 cela implique moins d'intervention de la part de l'utilisateur tout en préservant la précision géométrique maximale que présente le nuage de points dense.

Le deuxième point sur lequel s'appuie la contribution proposée est une nouvelle méthode de représentation implicite pour définir une surface à partir d'un nuage de points que nous avons appelée EIMLS (pour *Extended Implicit Moving Least Squares*). A la différence des représentations implicites de l'état de l'art, cette nouvelle représentation permet de définir une fonction implicite loin de la surface, ce qui n'est pas une propriété traditionnellement attendue lorsque l'on désire reconstruire explicitement une surface à partir d'un nuage de points mais qui est capital dans notre cas, lorsque l'on souhaite réaliser une simulation numérique en volumes immergés.

2.1.4 Méthode proposée

La figure 2.2 montre le jeu de données 2D que nous utilisons dans ce chapitre pour illustrer les fondements théoriques de notre méthode. Il a été obtenu en prenant une tranche du célèbre *Stanford Bunny* qui est un nuage de points 3D. Il est composé de points 2D ainsi que de vecteurs normaux 2D normalisés. Bien qu'il s'agisse d'un

modèle 2D, nous montrons dans la section 2.4 qu'il est bien représentatif d'un modèle 3D *réel* (par opposition à un modèle qui aurait été généré synthétiquement, par exemple par échantillonnage sur un maillage surfacique).

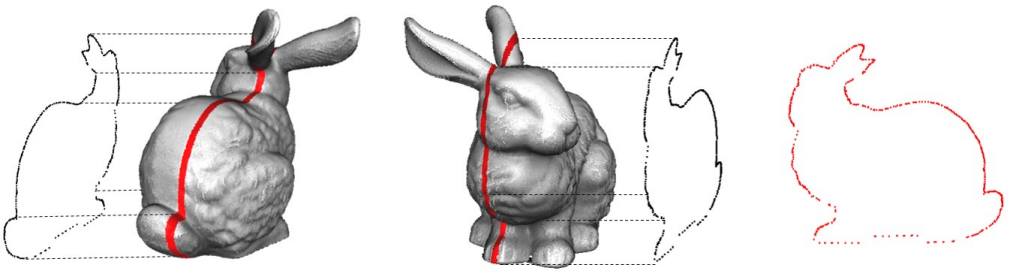


FIGURE 2.2 – Coupe 2D du *Stanford Bunny* utilisée comme jeu de données 2D dans ce chapitre. Cette coupe contient 557 points 2D avec normales. Les normales ont été calculées en 3D et ensuite projetées en 2D. Voir la section 2.3 pour plus de détails sur le jeu de données 3D et les traitements qui lui sont associés.

Notre méthode est décrite par la figure 2.3. Premièrement un nuage de points d-dimensionnel ($d = 2$ ou 3) avec normales **(a)** est utilisé pour calculer une fonction scalaire **(b)** qui représente implicitement la surface échantillonnée par le nuage de points. Deuxièmement un maillage volumique immergé anisotrope **(c)** peut être construit autour de la surface du modèle. Une fois obtenu, ce maillage sert alors de discrétisation spatiale pour une simulation d'écoulement autour de l'objet **(d)** par une résolution des équations Navier-Stokes incompressibles transitoires par éléments finis stabilisés. Durant la simulation, le maillage volumique est raffiné conjointement autour de l'écoulement et de la surface de l'objet grâce à une métrique d'erreur calculée *a-posteriori*. La fonction scalaire implicite est uniquement recalculée à chaque étape de remaillage aux nœuds du maillage. C'est alors une interpolation sur le maillage qui est utilisée dans la phase de calcul d'écoulements.

Nous présentons dans la section 2.3 comment les nuages de points 3D peuvent être acquis et prétraités afin de filtrer certains artefacts et calculer les normales orientées. Nous introduisons ensuite dans la section 2.4 notre nouvelle représentation implicite EIMLS pour calculer la fonction scalaire implicite à partir du nuage de points. Nous montrons après dans la section 2.5 comment adapter un maillage anisotrope autour d'une surface échantillonnée par points. Enfin, des résultats numériques obtenus sur des jeux de données 3D réels seront présentés dans la section 2.6.

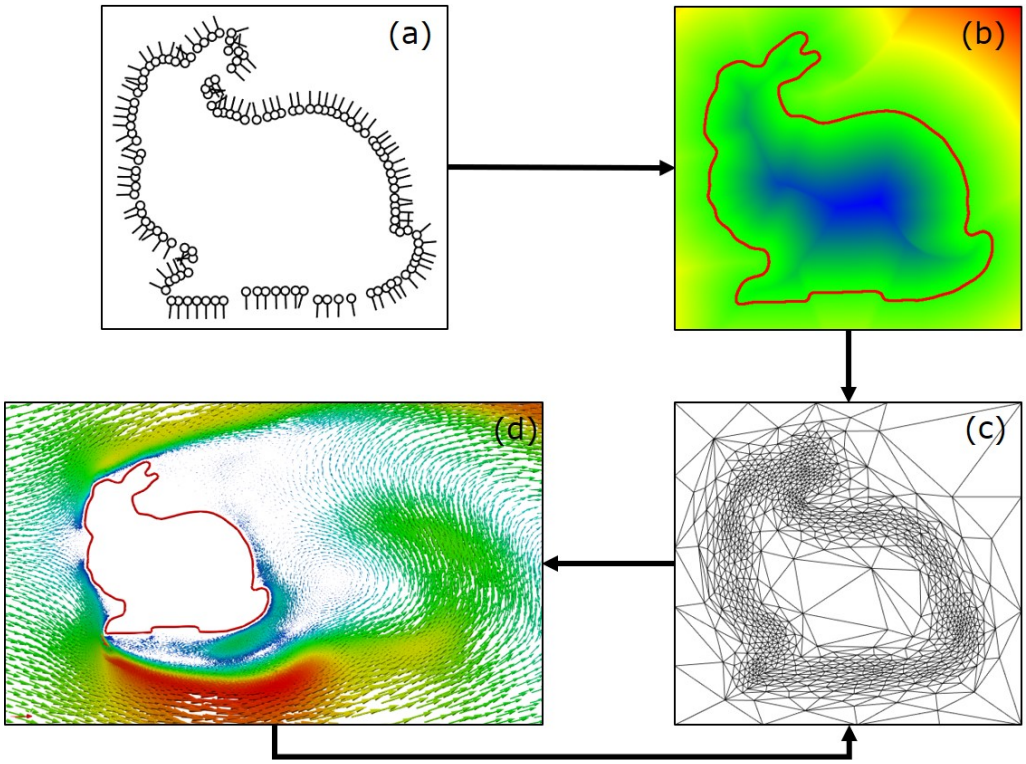


FIGURE 2.3 – Vue d'ensemble de notre méthode : (a) Nuage de points orienté avec normales. (b) Fonction scalaire implicite calculée par EIMLS. L'iso-zéro est représentée en **rouge**. (c) Maillage anisotrope adapté autour de la surface échantillonnée par le nuage de points (a). (d) Écoulement calculé par la méthode des éléments finis en volumes immergés.

2.2 État de l'art

La méthode proposée se situe à la croisée de plusieurs domaines scientifiques : la mécanique des fluides numérique en volumes immergés, la géométrie algorithmique et l'adaptation de maillage anisotrope. Il est donc complexe de fournir un panorama complet de tous les travaux qui lui sont connexes. C'est pour cette raison que nous avons pris le parti, dans cette section intitulée *"État de l'art"*, de ne présenter que les travaux connexes du point de vue de la problématique générale qui est la simulation numérique autour de nuages de points. Nous présentons par la suite dans chaque section des états de l'art spécifiques aux différents domaines associés.

Nous pouvons comparer dans un premier temps la méthode proposée à l'idée d'utiliser des points comme primitives pour la simulation numérique. Cette idée a été largement utilisée dans le contexte de l'animation basée physique. (Gross et Pfister, 2011) donnent un aperçu général de l'animation basée point. C'est dans ce contexte que (Müller et al., 2007) ont introduit la célèbre méthode PBD (pour *Position Based Dynamics*). L'idée de cette méthode est de remplacer le traditionnel formalisme basée force : une force induit une accélération qui modifie les vitesses qui modifient à leur tour les positions, par un formalisme basé position. Ainsi, en agissant directement sur les positions, la simulation résultante est bien plus stable notamment dans un contexte de simulation temps réel où les pas de temps utilisés sont contraints. Ce travail séminal a largement été repris dans la communauté de l'animation basée physique. Plus récemment (Macklin et al., 2014) ont proposé un solveur unifié pour les corps rigides, les corps mous, les tissus et les fluides avec des résultats temps réel très impressionnants, reproduits sur la figue 2.4. Bien que ces méthodes soient basées points, elles restent éloignées de notre travail dans la mesure où les points sont ici utilisés comme support de la simulation et non pour en définir les conditions aux limites.

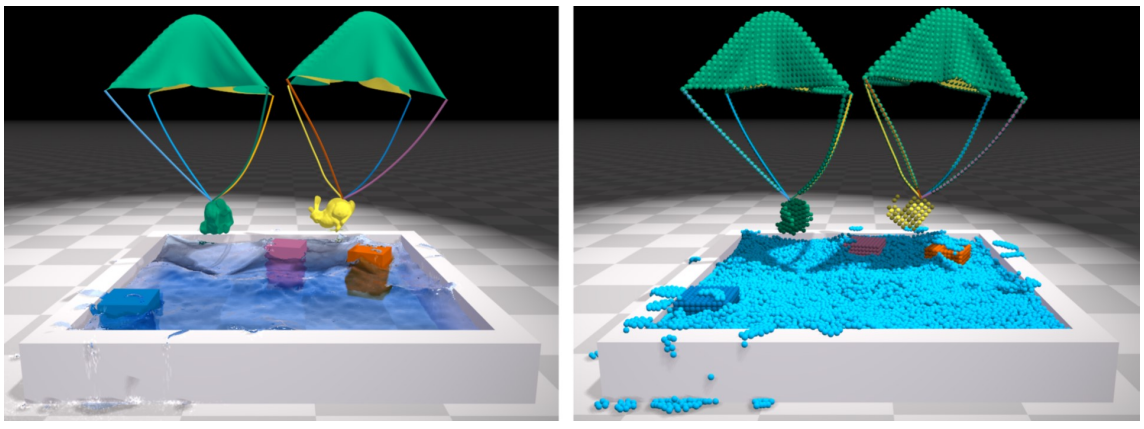


Figure 2.4 – Scène complexe simulée par PBD : des Stanford Bunny sont parachutés dans des piscines remplies d'eau. Tous les éléments de la scène interagissent les uns avec les autres au travers du *framework* unifié. (Figure reproduite de (Macklin et al., 2014)).

L'idée la plus proche de notre travail est de réaliser des simulations numériques autour de géométries réelles numérisées. Cette idée a été exploitée dans le contexte des jeux vidéo et de l'interaction utilisateur en réalité virtuelle ou augmentée. (R. A. Newcombe et al., 2011 ; Izadi et al., 2011) présentent une simulation interactive de corps rigides virtuels qui interagissent avec une scène numérisée en temps réel par une caméra de profondeur du commerce. Leurs résultats sont reproduits sur la figure 2.5. La partie intéressante de ce travail réside dans le fait qu'aucune représentation explicite de la géométrie de la scène n'est stockée en interne dans la mesure où celle-ci est uniquement représentée par une fonction de distance signée et tronquée qui est mise à jour en temps réel. La représentation de surface sous-jacente employée est (Curless et Marc Levoy, 1996). Une comparaison plus détaillée avec notre méthode est présentée dans la section 2.4.

Figure 2.5 – Simulation interactive sur une scène capturée en temps réel par l'algorithme *Kinect-Fusion*. Des milliers de particules interagissent avec la scène. (Figure reproduite de (Izadi et al., 2011)).

Dans le contexte de la préservation du patrimoine (Oreni et al., 2014 ; Barazzetti et al., 2015) proposent une méthode semi-automatique, basée sur des logiciels CAO du commerce, pour construire un modèle BIM [1] à partir de scan 3D laser de bâtiments historiques, comme illustré sur la figure 2.6. Ils utilisent ensuite le modèle CAO reconstruit pour réaliser une analyse par éléments finis des modes propres de vibration du bâtiment. Bien que leur méthode permet de gérer des jeux de don-

1. acronyme de *Building Information Model* qui signifie modélisation des données du bâtiment

nées massifs, elle n'est pas complètement automatique et requiert des compétences techniques pointues en modélisation CAO.

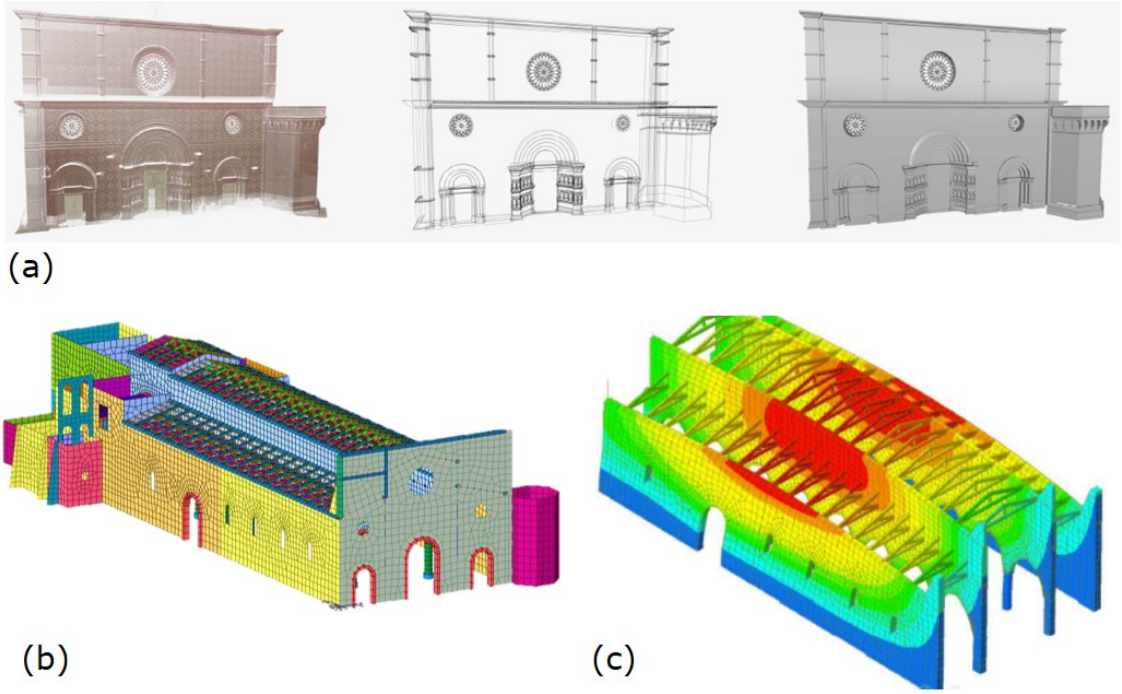


FIGURE 2.6 – (a) Méthode semi-automatique pour reconstruire un modèle CAO (à droite) à partir d'un nuage de points (à gauche). (b) Maillage volumique utilisé pour la simulation par éléments finis. (c) Mode propre de vibration du bâtiment obtenu par simulation. (Figure reproduite et modifiée de (ORENI et al., 2014)).

(CASTELLAZZI et al., 2015) proposent également une méthode semi-automatique pour réaliser des simulations éléments finis de bâtiments numérisées et représentés par des nuages de points. Ils sectionnent leur modèle 3D en tranches 2D horizontales qui sont traitées et nettoyées manuellement. Ces tranches sont ensuite empilées en 3D pour produire un modèle 2D voxelisé qui est utilisé comme support pour une simulation par éléments finis. La voxelisation est néanmoins une approche limitée dans la mesure où la résolution maximale du modèle est fixée par la taille des voxels, elle-même fixée par la taille des pixels dans les tranches 2D, ce qui est suffisant pour simuler le comportement mécanique structurel global mais qui n'est pas transposable à la simulation fine des interactions entre un fluide et une surface. Leur approche est illustrée sur la figure 2.7.

Des travaux récents dans le domaine de la simulation médicale ont démontré la faisabilité de la simulation de propriétés mécaniques ou d'écoulement de fluides à travers des organes à partir de scans tomographiques. (VENKATASUBRAMANIAM et al., 2004) présentent une méthode automatique pour générer des modèles éléments finis à partir de scans tomographiques dans le but d'étudier les anévrismes aortiques abdominaux. Plus récemment (CHNAFA et al., 2013) proposent également

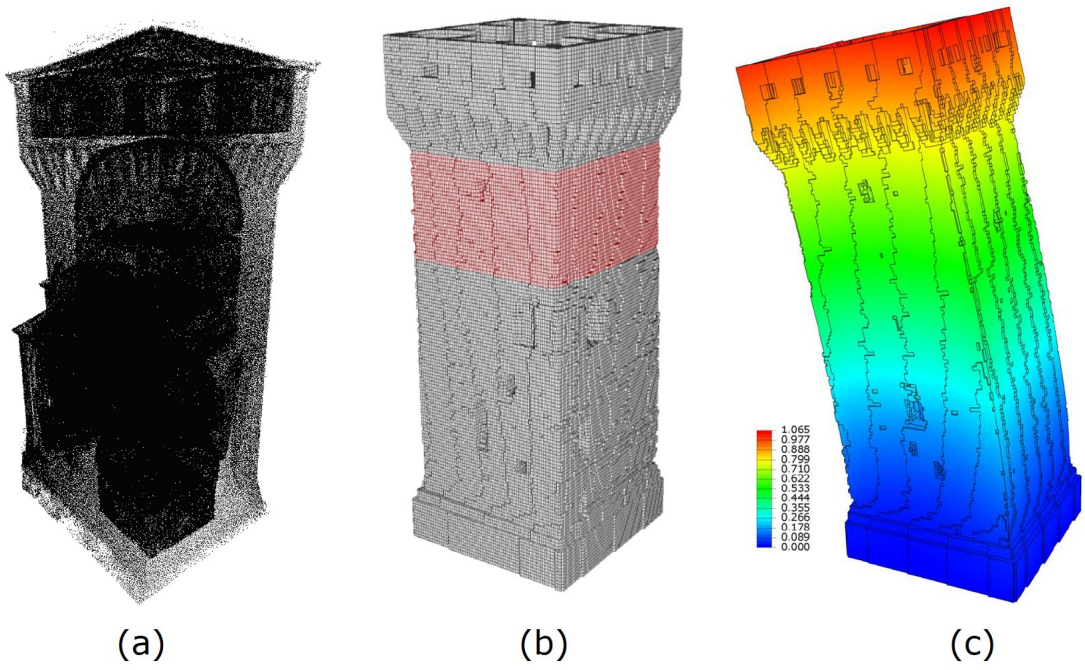


FIGURE 2.7 – Analyse par éléments finis des modes propres de la tour principale (Mastio) de la forteresse *San Felice sul Panaro* près de la ville de Modène en Italie. (a) Nuage de points de la tour. (b) Discrétisation en voxels utilisée comme maillage pour la simulation. (c) Norme du déplacement du mode propre en fléchissement de la structure à 1.9131 Hz. (Figure reproduite et modifiée de (CASTELLAZZI et al., 2015)).

une méthode pour simuler l'écoulement du sang durant le cycle cardiaque. Une simulation LES[2] est utilisée sur un maillage déformable, dont la déformation a été calculée par recalage sur une séquence dynamique de scans tomographiques. Comme on peut le voir sur la figure 2.8, la méthode parvient à capturer la nature turbulente de l'écoulement. Ainsi la simulation de la dynamique du cœur basée sur des scans réels permet à la simulation d'être spécifique à un patient donné ce qui laisse entrevoir de nombreuses et prometteuses applications. Néanmoins cette méthode est basée sur un maillage explicite qui n'est pas adapté pour la simulation CFD sur des géométries complexes. (MIHALEF et al., 2011) présentent aussi une méthode de simulation hémodynamique spécifique au patient à partir de scans volumétriques, leurs résultats sont reproduits sur la figure 2.9. Au lieu d'utiliser un maillage surfacique explicite en entrée de leur simulation, ils utilisent un solveur par volumes immergés et représentent le cœur par une fonction implicite calculée directement à partir des scans tomographiques. Cette approche est la plus proche de la nôtre, à la différence que le calcul de la fonction implicite est facilité ici par le fait que les données d'entrée sont structurées et volumiques, ce qui n'est pas le cas des nuages de points qui sont des données surfaciques et non structurées.

2. Acronyme de *Large-Eddy Simulation*, méthode de simulation à grande échelle de la turbulence

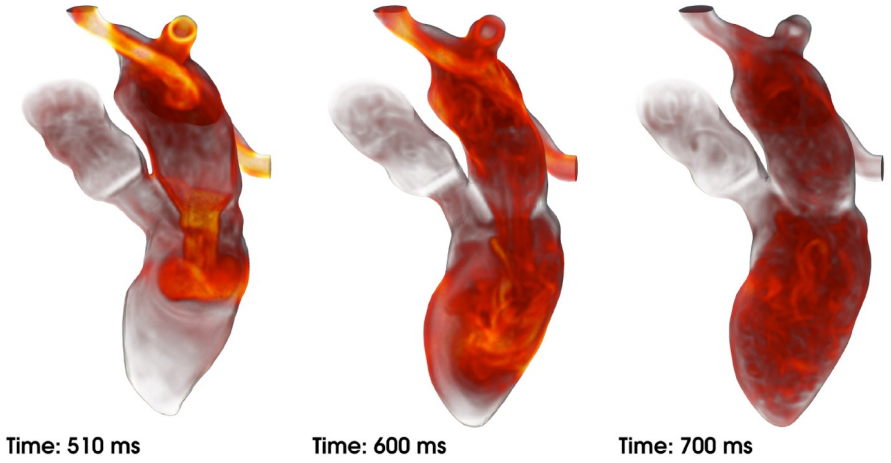


FIGURE 2.8 – Norme de la vorticité dans le ventricule gauche durant la diastole. (Figure reproduite de (CHNAFA et al., 2013)).

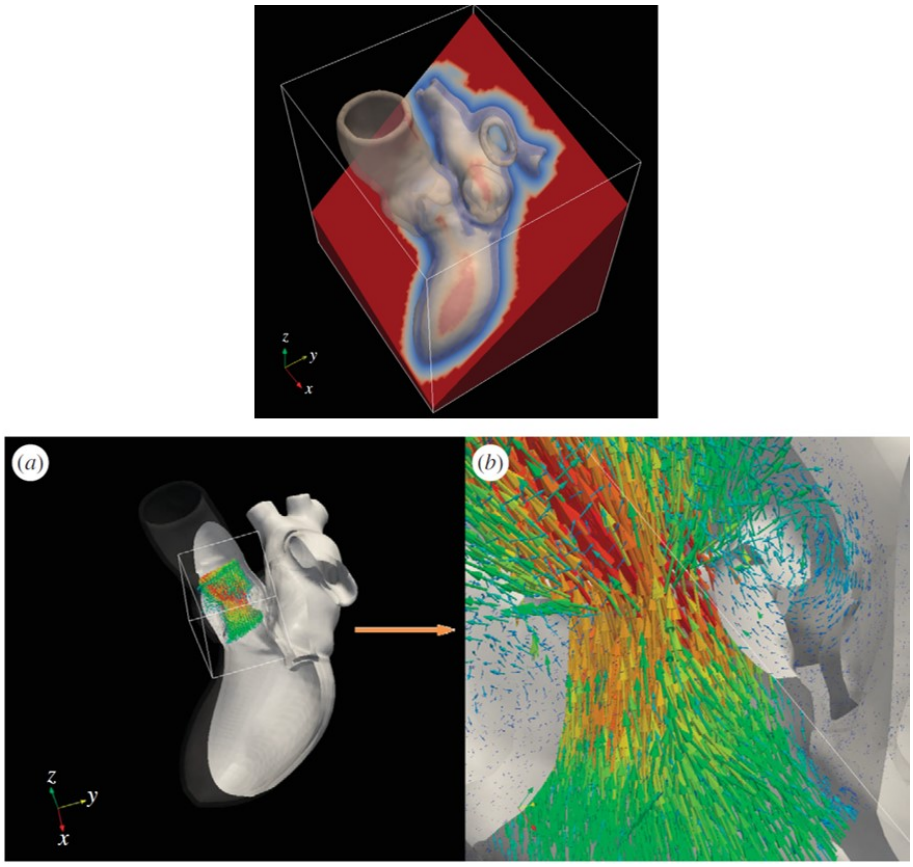


FIGURE 2.9 – (En haut) modèle *level set* du cœur immergé au sein du domaine rectangulaire. (En bas) Formation de vortex dans les sinus de l'aorte obtenus par simulation. (Figure reproduite et modifiée de (MIHALEF et al., 2011)).

2.3 Acquisitions et traitements de nuages de points 3D

2.3.1 Jeux de données

Les techniques de numérisation 3D permettent de produire des modèles 3D d'objets réels. De nos jours, ces techniques sont largement utilisées par les industriels mais également par les consommateurs. La mesure 3D peut être réalisée par le biais de différents phénomènes physiques : les scanners par contact utilisent la force de réaction d'une surface, les scanners lasers (ou lidars) utilisent la réflectance Lambertienne des surfaces pour mesurer le temps de vol d'un rayon laser plusieurs centaines de milliers de fois par seconde, les scans tomographiques et autres scanners volumétriques produisent des tranches 3D d'un objet. Dans cette section chapitre nous nous concentrons uniquement sur les techniques de numérisation de surface. (Zhao et al., 2016) présentent une méthode similaire à la méthode proposée adaptée à des données issues de scans volumétriques.

Dans le but d'illustrer la méthode proposée, nous avons choisi 3 différents jeux de données 3D représentés sur la figure 2.10. Ils ont été choisis, d'une part afin de couvrir une large variété de techniques de numérisation surfacique, et d'autre part afin de représenter un large panel d'applications industrielles de la CFD. Les cas de simulation associés à chacun sont détaillés dans la section 2.6.

(a) **lapin** : Déjà introduit précédemment, ce jeu de données a été acquis par le laboratoire d'informatique graphique de l'université de Stanford (*Stanford University Computer Graphics Laboratory*) en 1993-4. Il a été numérisé à l'aide d'un scanner Cyberware 3030 MS à partir d'une statue en argile. Il est composé de 10 images de profondeur alignées par un algorithme ICP [3] modifié (Turk et Marc Levoy, 1994). Le nuage de points mesure 15 cm de long et présente 362,270 points. Derrière son apparente simplicité géométrique, ce modèle présente de nombreux artefacts tels que des points aberrants et des trous dans sa partie inférieure. Sa surface présente également d'intéressants détails saillants.

(b) **salle** : La salle Saint Jacques est une salle de réunion qui se situe dans le bâtiment principal de l'école des Mines ParisTech à Paris. Elle a été acquise durant cette thèse, à l'aide d'un scanner laser fixe Faro Focus X130. 5 scans ont été acquis en différentes positions de la salle. Par commodité et dans la mesure où celle-ci n'a pas été entièrement numérisée, la table centrale visible sur la photo a été manuellement retirée du scan. La salle mesure 10 m de long, par 7 m de large, par 4 m de hauteur, elle est composée de 13 million de points et présente des murs en brique ainsi qu'un plafond en voute. La précision locale moyenne de chaque scan est de 2 mm, néanmoins le nuage de points présente

3. *Iterative Closest Point* : algorithme itératif qui permet de recaler deux nuages de points en trouvant la transformation rigide qui minimise la distance de l'un à l'autre.

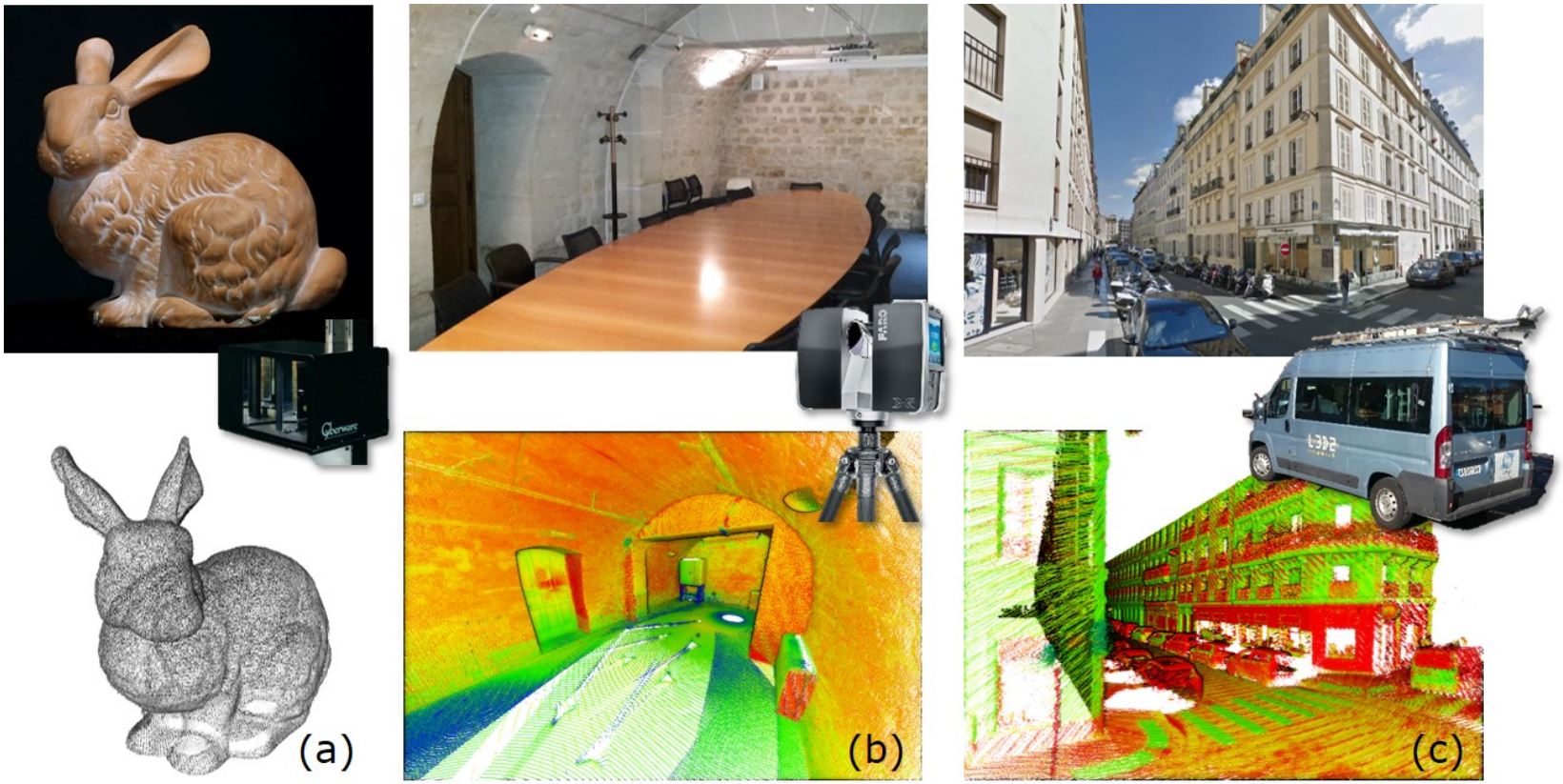


FIGURE 2.10 – Les trois jeux de données 3D utilisés dans ce chapitre. (a) **lapin** : scan d'un objet. (b) **salle** : Scan laser terrestre d'une salle de réunion. (c) **rue** : Scan laser mobile. (b) et (c) ont été colorisés par l'intensité de retour du lidar.

de nombreux points aberrants aux endroits où les rayons laser ont heurté des bords d'objets.

(c) **rue** : La rue Madame se situe dans le 6ème arrondissement de Paris. Il s'agit d'un jeu de données publique pour la segmentation et classification de nuages de points (Serna et al., 2014). Il a été acquis par le prototype L3D2 du laboratoire de robotique de l'école des Mines (CAOR). Il s'agit d'un système de cartographie mobile (F. Goulette et al., 2006) équipé d'un système de localisation GPS-IMU, et d'un lidar Velodyne HDL32E. Ce dernier présente une précision de 2 cm. Ses 32 fibres laser sont montées sur une tête rotative qui tourne à une fréquence de 10 Hz pour produire jusqu'à 700.000 points par seconde. Un système de cartographie mobile est conçu pour produire des données à grand rendement en même temps que le véhicule se déplace à vitesse normale dans la circulation. Le nuage de points utilisé ici est une portion de 13 millions de points et 106 m de long du jeu de données complet. Il s'agit du jeu de données le plus exigeant parmi les trois présentés dans la mesure où c'est le plus bruité car les erreurs du système de localisation s'ajoutent à celles du lidar (5 cm d'erreur moyenne totale). Comme on peut le voir sur la figure 2.10 il présente également de nombreux trous dus aux occlusions induites par les objets de la scène le long de la trajectoire d'acquisition.

2.3.2 Prétraitements de nuages de points

Dans la section 2.4 nous présentons une nouvelle représentation implicite de surface à partir de nuages de points nommée EIMLS. Afin de calculer cette fonction, le nuage de points doit être prétraité pour retirer les points aberrants, être sous-échantillonné et calculer les normales en chaque point. La section 2.4 donne un éclairage sur leur nécessité.

2.3.2.1 Retrait des points aberrants

Les nuages de points issus du monde réel contiennent toujours des points aberrants, dus à une mauvaise interprétation d'un écho laser par le logiciel embarqué du lidar. Une approche simple pour retirer ces points est de calculer la densité locale en chaque point du nuage et de supprimer ensuite les points avec une densité trop faible, en dessous d'un certain seuil. Cette densité peut être calculée de différentes manières. Pour le jeu de données **rue** nous avons choisi de calculer la distance au 3ème plus proche voisin avec un seuil à 30 cm.

Le jeu de données **salle** acquis par scan laser fixe présente des points aberrants structurés plus difficiles à retirer. Ils sont dus au fait que lorsque le laser rencontre un coin sur un objet, il produit une trainée de points fantômes dans l'axe de tir du laser. Pour supprimer ces points, on peut calculer l'angle entre la normale en chaque point et la direction du laser. Les points acquis avec un angle trop rasant (typiquement 88°) sont alors retirés.

2.3.2.2 Sous-échantillonnage

Les scans laser fixes et mobiles présentent une anisotropie au niveau de la répartition des points, due à la différence d'échantillonnage du capteur le long de ses axes principaux. Par exemple, le Velodyne HDL32E, utilisé pour acquérir le jeu de données **rue**, présente une résolution angulaire verticale [4] de 1.33° et une résolution horizontale de 0.1° à 10 Hz. Cette anisotropie doit être prise en compte dans la conception des algorithmes utilisés ensuite sur ce type de données, par exemple en utilisant des accumulateurs de Hough (Boulch et Marlet, 2012) pour le calcul de normales. Ici nous avons pris le parti d'utiliser une approche plus simple en sous-échantillonnant les jeux de données **salle** et **rue** sur un critère d'extension spatial. Pour ce faire, nous avons utilisé une structure d'octree (Elseberg et al., 2013) avec une taille de feuille fixée à 2 cm.

2.3.2.3 Estimation des normales

(Hoppe et al., 1992) ont introduit une méthode simple pour calculer des normales en chaque point d'un nuage de points. La direction du vecteur normal est définie comme la direction du vecteur propre associé à la plus petite valeur propre de la matrice de covariance du voisinage de chaque point du nuage. Cette formulation simple avec un voisinage de 100 points donne des résultats satisfaisants pour nos trois jeux de données. Nous utilisons ensuite l'information d'origine du scanner afin d'orienter correctement les normales. Néanmoins, le calcul de normales sur un nuage de points complexe représente un problème dur pour lequel des méthodes plus avancées peuvent fournir de meilleurs résultats sur des géométries complexes (Boulch et Marlet, 2012 ; Boulch et Marlet, 2016).

4. angle entre chaque fibre laser consécutive

2.4 Définition de surface par moindres carrés implicites étendus (EIMLS)

Notations 1	Definition
d	dimension de l'espace
$\mathcal{N} \subset \mathbb{N}$	ensemble d'indices de points
$\mathcal{P} = \{(p_i, n_i), i \in \mathcal{N}\}$	nuage de points orienté : points et normales
$p_i \in \mathbb{R}^d$	position d-dimensionnelle du $i^{\text{ème}}$ point de $\mathcal{P}$
$n_i \in \mathbb{S}^{d-1}$	normale unitaire du $i^{\text{ème}}$ point de $\mathcal{P}$ (sur la sphère unité)
$\mathcal{S}_\mathcal{P}$	surface lisse sous-jacente échantillonnée par $\mathcal{P}$
$\alpha : \mathbb{R}^d \to \mathbb{R}$	fonction implicite qui représente $\mathcal{S}_\mathcal{P}$

TABLE 2.1 – Notations et définitions pour la partie 2.4

Dans cette section, nous montrons comment construire une fonction scalaire $\alpha : \mathbb{R}^d \to \mathbb{R}$ à partir d'un nuage de points $\mathcal{P}$, qui définit implicitement la surface lisse $\mathcal{S}_\mathcal{P}$ échantillonnée par le nuage de points. Proche de la surface, cette fonction doit se comporter comme une fonction de distance à $\mathcal{S}_\mathcal{P}$. Le tableau 2.1 rassemble les notations utilisées dans cette partie.

Après un état de l'art sur la reconstruction de surface à partir de nuages de points (sous-section 2.4.2), nous présenterons une nouvelle méthode de représentation implicite étendue adaptée pour la définition de conditions aux limites d'une simulation numérique par volumes immergés (sous-section 2.4.2). En particulier, nous décrirons les différentes contraintes liées à ce cadre applicatif particulier.

2.4.1 État de l'art

Une surface est une $(d-1)$ variété de $\mathbb{R}^d$ qui peut être représentée de différentes manières. On différencie classiquement les représentations explicites des représentations implicites. Les représentations explicites décrivent une surface à l'aide d'un ensemble discret. Par exemple, en 3D une surface peut être décrite à partir d'un ensemble de triangles pour former un maillage, ou bien par un ensemble de points pour former un nuage de points. Les représentations implicites décrivent indirectement une surface, par exemple par une courbe de niveau d'une fonction scalaire ou bien par l'ensemble des points fixes d'un opérateur de projection. Ces exemples sont illustrés en 2D sur la figure 2.11.

Le fait de passer d'un nuage de point à un autre type de représentation est appelé *reconstruction de surface*. Il s'agit d'un large domaine de recherche en informatique graphique et en géométrie algorithmique. Pour un état de l'art exhaustif, le lecteur peut se référer à (BERGER et al., 2014).

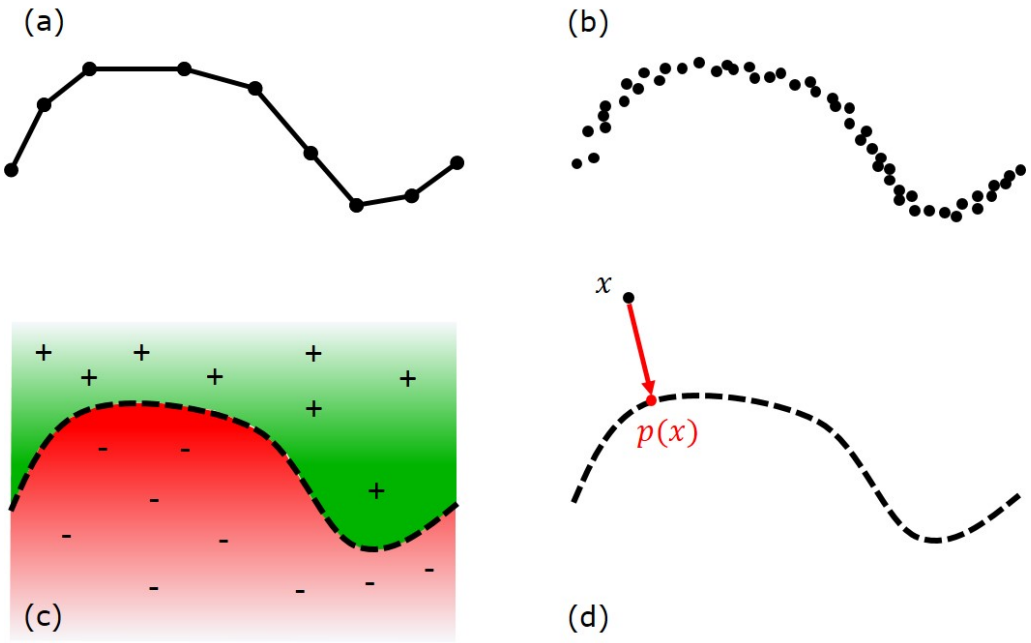


FIGURE 2.11 – Exemple de représentation de surface en 2D. (a) Maillage. (b) Nuage de points bruité. (c) Fonction implicite signée. (d) Opérateur de projection.

Reconstruction de surface explicite Il existe un ensemble de méthodes qui permettent de construire directement un maillage surfacique à partir d'un nuage de points. Ces méthodes sont traditionnellement qualifiées de *méthodes explicites*. L'algorithme BPA (pour *Ball Pivoting Algorithm*) (BERNARDINI et al., 1999) interpole les points du nuage en faisant rouler une boule virtuelle entre eux. À chaque fois que la boule touche 3 points, un nouveau triangle est créé. Cette méthode et ses variantes sont basées sur des heuristiques et échouent par exemple lorsque la courbure locale est plus grande que le rayon de la boule, ce qui est illustré sur la figure 2.12.

Un autre type de méthodes sont celles basées sur les diagrammes de Voronoï, telle que les méthodes *Crust* (AMENTA et BERN, 1999) et *Power Crust* (AMENTA, CHOI et al., 2001). Elles exploitent la propriété du diagramme de Voronoï du nuage de point qui assure que le maillage de la surface est un sous-ensemble de son dual, la triangulation de Delaunay. Cette propriété est illustrée sur la figure 2.13. Le principal avantage de ces dernières est qu'elle offrent des garanties théoriques sur les méthodes elles-même ainsi que sur la qualité du maillage obtenu. (CAZALS et GIESEN, 2006) donnent une revue des méthodes de reconstruction de surface basées Voronoï.

Généralement, les méthodes explicites sont interpolantes : elles essaient de relier tous les points du nuage afin de produire un maillage surfacique. Elle ne sont donc pas robustes au bruit et ne sont pas conçues pour traiter des nuages de points massifs, elles ne sont donc pas compatibles avec les nuages de points complexes du monde réel. De plus, comme cela a été précédemment évoqué, dans notre contexte, construire

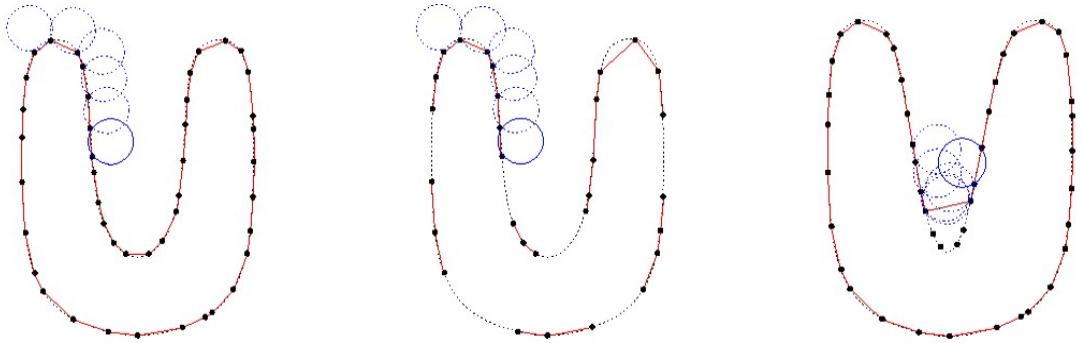


FIGURE 2.12 – *Ball Pivoting Algorithm.* (a) Un boule de rayon r pivote d'un point à l'autre en les connectant par des segments. (b) Lorsque la densité est trop faible certaines arrêtes ne sont pas crées. (c) Lorsque la courbure est plus grande que $\frac{1}{r}$, des points ne sont pas connectés et certains détails sont perdus. (Figure reproduite de (BERNARDINI et al., 1999))

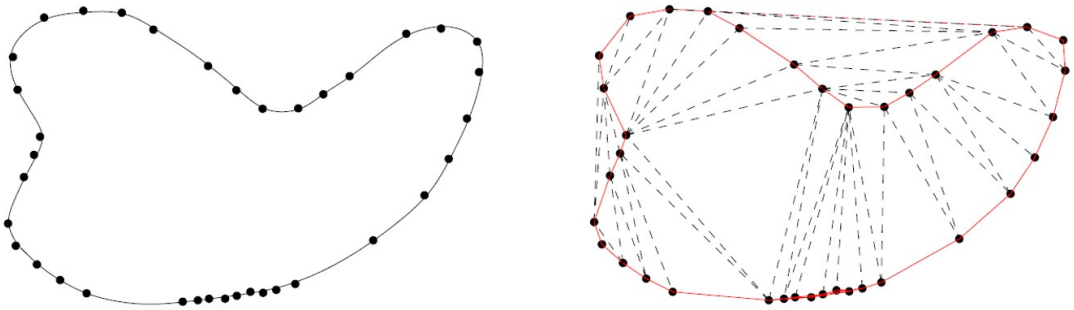


FIGURE 2.13 – (à gauche) Surface échantillonnée par points. (à droite) La triangulation de Delaunay du nuage de points. La surface reconstruite (en rouge) en est un sous-ensemble. (Figure reproduite de (CAZALS et GIESEN, 2006))

un maillage surfacique à partir d'un nuage de points consiste à faire une hypothèse forte sur les propriétés géométriques et topologiques de la surface. Finalement cela revient seulement à remplacer une représentation explicite par une autre. C'est pour cette raison que nous préférons construire une représentation implicite directement basée sur le nuage de points.

Reconstruction de surface implicite Les méthodes implicites permettent de construire une représentation implicite d'une surface à partir d'un nuage de points. La littérature sur les surfaces implicites est très abondante. (Curless et Marc Levoy, 1996) ont introduit une méthode pour construire une fonction de distance signée et tronquée à partir de mesures provenant de caméras de profondeur. Chaque image de profondeur est utilisée pour remplir une grille régulière dense de voxels. Chaque mesure permet d'étiqueter non seulement l'espace autour de la surface mais également l'espace vide le long de l'axe entre le capteur et le point mesuré, ce qui ajoute une information topologique additionnelle importante. Cette méthode de représentation de surface a connu récemment un regain d'intérêt avec la méthode *Kinect Fusion* (R. A. Newcombe et al., 2011), qui l'utilise pour assembler des images de profondeur issues d'une caméra de profondeur grand public, et reconstruire une surface. (Kazhdan, Bolitho et al., 2006 ; Kazhdan et Hoppe, 2013) ont récemment montré que le problème de reconstruction de surface pouvait être formulé par une équation de Poisson, qu'ils résolvent par éléments finis sur un octree. Comme illustré sur la figure 2.14, *La reconstruction de Poisson* est basée sur l'utilisation de normales orientées. Comme l'orientation des normales de manière consistante sur un nuage de points complexe est un problème difficile, certaines méthodes se sont concentrées sur l'utilisation de normales non orientées. Ainsi (Alliez et al., 2007) ont montré que la reconstruction de surface à partir d'un nuage de points avec des normales non orientées pouvait être exprimé comme un problème aux valeurs propres généralisé. (Mullen et al., 2010) apportent quant à eux une solution différente en calculant de manière robuste un champ de distance non signé et en le signant ensuite de manière statistique et discrète sur la grille qui a servi au calcul.

Reconstruction de surface explicite à partir d'une fonction implicite Néanmoins, il faut faire attention à la classification des méthodes de reconstruction de surface proposée ici. Le résultat obtenu et présenté dans les articles de recherche qui leur sont associés est souvent un maillage surfacique, même pour les méthodes de reconstruction implicites. Souvent, le maillage est seulement créé dans une étape finale indépendante de l'algorithme présenté, pour des besoins de visualisation par exemple. Il existe un ensemble de méthodes qui permettent de générer un maillage surfacique à partir d'une courbe de niveau d'une fonction implicite scalaire. (Lorensen et Cline, 1987) ont introduit l'algorithme dit de *Marching Cubes* qui permet d'extraire une maillage surfacique composé de triangles à partir d'une fonction scalaire échantillonnée sur une grille régulière. (Treece et al., 1998) ont étendu la

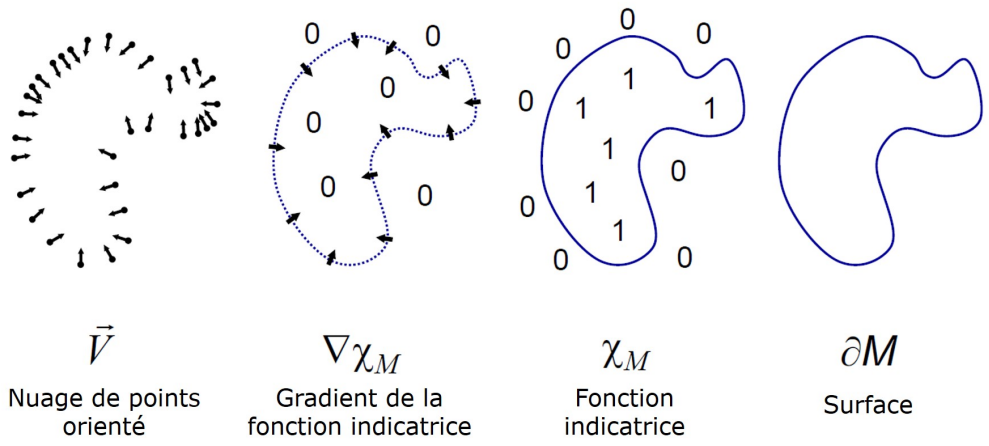


FIGURE 2.14 – Illustration de la reconstruction de surface par la méthode de Poisson. (Figure reproduite et modifié de (KAZHDAN, BOLITHO et al., 2006))

méthode pour mieux gérer les cas ambigus par un échantillonnage sur une grille tétrahédrique. (SCHAEFER et WARREN, 2004) ont également étendu la méthode initiale en proposant le *Dual Marching Cubes* qui extrait un maillage à partir d'un échantillonnage de la fonction implicite sur un octree. Cette méthode présente l'intérêt de produire un maillage avec des triangles de taille adaptative dans le but de limiter le problème de sur-échantillonnage de la méthode initiale.

Reconstruction de surface implicite par moindres carrés glissants Toutes les méthodes implicites précédemment citées sont *globales*, c'est à dire que la représentation implicite est calculée sur une grille discrète (grille régulière, octree ou grille tétrahédrique) à partir du nuage de points complet. Il existe un autre ensemble de méthodes qualifiées de *locales* dans la mesure où la représentation implicite est définie en tout point de l'espace et est calculée seulement sur un petit sous-ensemble localisé du nuage de points.

Parmi ces méthodes, les méthodes basées MLS (pour *Moving Least Squares*, qui signifie moindre carrés glissants) présentent de bonnes garanties théoriques avec un coût de calcul faible. Les travaux originaux de (HOPPE et al., 1992) sont les premiers à calculer une fonction scalaire signée à partir d'un nuage de points. Néanmoins, la première définition de surface par MLS est dérivée du travail original de (LEVIN, 2004). Il utilise un opérateur de projection calculé en deux temps : premièrement, étant donné un point de requête $q \in \mathbb{R}^d$, un plan est ajusté au voisinage de q dans le nuage de points. Une fonction polynomiale est alors ajustée localement et q est projeté sur la surface ainsi approximée. La surface est alors définie comme l'ensemble des points stationnaires de cet opérateur de projection. (ALEXA et al., 2003) ont alors utilisé cette définition pour le rendu de surfaces définies par points. Le principe de l'opérateur de projection par MLS qu'ils proposent est décrit sur la

figure 2.15. Les surface définies par un opérateur de projection calculé par MLS sont souvent qualifiées de PSS (pour *Point Set Surfaces*).

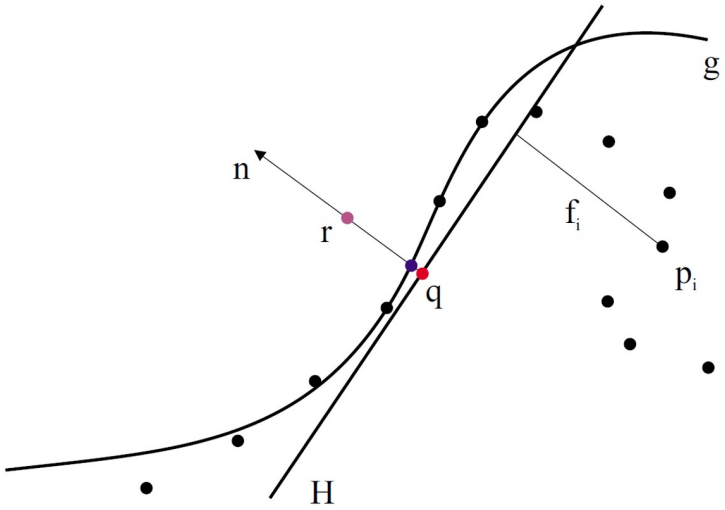


FIGURE 2.15 – Principe de l'opérateur de projection par MLS. Le point à projeter est le point r. Premièrement le plan local H est calculé par MLS à partir du voisinage du point r dans le nuage de point. Ensuite, un polynôme g est ajusté dans le repère défini par le plan H. Ce polynôme permet alors de calculer les distances f_i des points p_i du nuage de points local. (Figure reproduite de (ALEXA et al., 2003))

(SHEN et al., 2005) ont alors introduit une représentation par fonction scalaire implicite calculée par MLS dans le but d'estimer une surface à partir d'une *soupe de polygones*[5]. (KOLLURI, 2005) a alors appliqué cette définition aux nuages de points en démontrant également son bien-fondé théorique. Illustrée sur la figure 2.16, cette définition implicite porte le nom d'IMLS (pour *Implicit Moving Least Squares*, qui signifie moindres carrés glissants implicites).

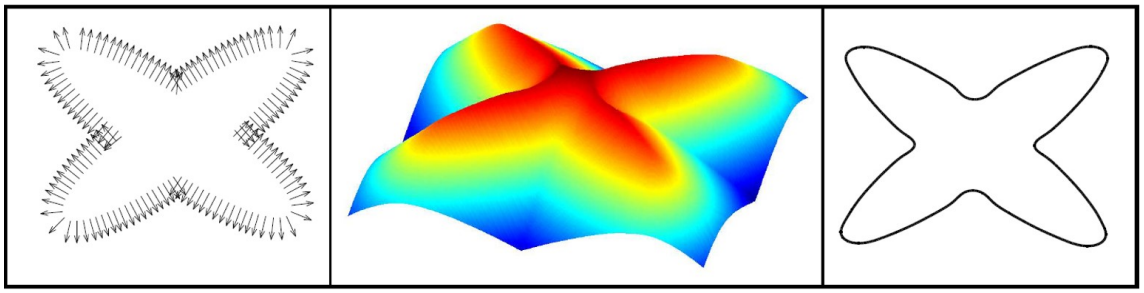


FIGURE 2.16 – (À gauche) Nuage de points orienté. (Au centre) Fonction implicite calculée par IMLS à partir des points. (À droite) Surface reconstruite en prenant l'iso-zéro de la fonction implicite. (Figure reproduite de (KOLLURI, 2005))

5. terme désignant un maillage surfacique sans connectivité entre éléments

Les formulations IMLS et PSS présentent toutes les deux des limitations intrinsèques lorsque la qualité du nuage de points est mauvaise. (Fleishman et al., 2005) sont les premiers à avoir introduit l'utilisation des statistiques robustes pour reconstruire une surface lisse par morceaux à partir d'un opérateur de projection modifié. (Öztireli et al., 2009) utilisent également le paradigme des statistiques robustes pour obtenir une version robuste de la méthode IMLS nommée RIMLS (pour *Robust IMLS*). L'intérêt principal de ces deux méthodes est de mieux reconstruire les parties saillantes dans les nuages de points bruités. (Guennebaud et Gross, 2007) ont également étendu les deux formulations MLS, implicite et par opérateur de projection, avec la méthode APSS (pour *Algebraic PSS*). Ils utilisent des sphères algébriques afin de mieux gérer les nuages de points sous-échantillonnés ainsi que ceux présentant des parties saillantes.

2.4.2 Concevoir une fonction scalaire implicite

Nous avons pris le parti de présenter dans cette section le cheminement qui nous conduit à définir notre nouvelle représentation de surface implicite étendue : EIMLS. Ce cheminement est présenté en deux temps. Premièrement, en partant des contraintes qu'imposent les données en entrée et des besoins du reste de la méthode, nous montrons que la représentation implicite qui convient le mieux à notre méthode est la représentation par IMLS. Dans un deuxième temps, après l'avoir présentée, nous montrons pourquoi elle nécessite d'être modifiée pour parfaitement répondre à notre besoin.

2.4.2.1 Contraintes de design

Artefacts des nuages de points réels Nous présentons les principaux artefacts que l'on peut retrouver dans les nuages de points réels (Berger et al., 2014) sur notre nuage de points 2D sur la figure 2.17 : bruit, trous, échantillonnage non uniforme, points aberrants. Nous avons proposé une solution pour traiter les deux derniers dans la section 2.3 avec des algorithmes de pré-traitement adaptés. Notre représentation implicite doit donc être capable de traiter les deux premiers : elles représentent nos contraintes de design liées aux données d'entrée.

Signée ou non signée ? Une autre question qui doit être considérée est quel type de représentation implicite nous devons choisir pour définir notre surface. Comme cela a été présenté dans l'état de l'art ci-dessus, une surface peut être définie implicitement par une fonction scalaire ou bien par un opérateur de projection. Les opérateurs de projection impliquent un processus itératif d'optimisation locale qui est plus coûteux en termes de temps de calcul que de calculer une simple fonction scalaire. C'est pourquoi nous choisissons ce dernier type de représentation.

Nous pouvons alors nous demander si cette fonction scalaire doit être signée ou non signée. Si elle l'est, alors la surface peut être facilement extraite par une

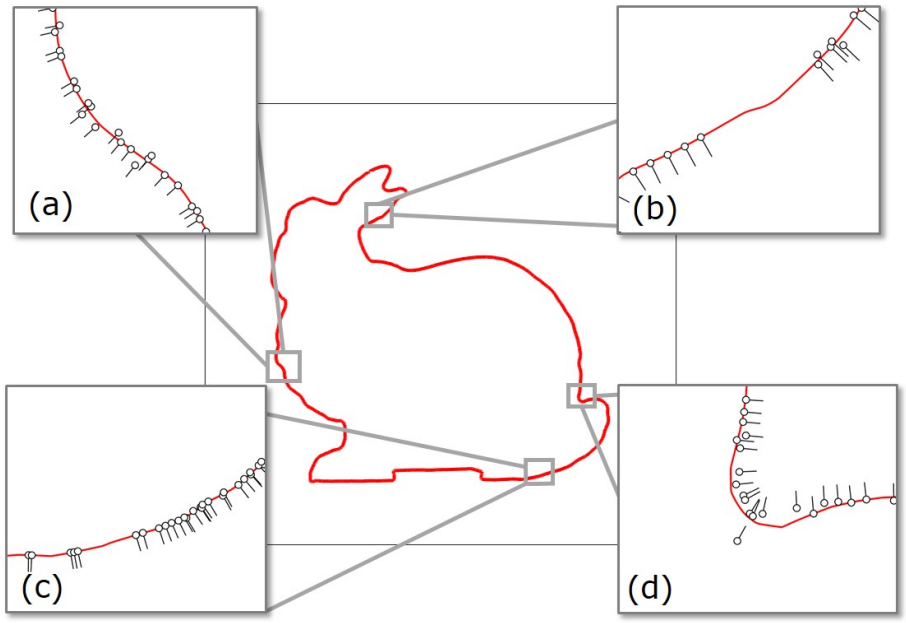


FIGURE 2.17 – Principaux artefacts que l'on peut trouver dans un nuage de points illustrés sur un modèle 2D. (a) Bruit. (b) Données manquantes (trous). (c) Échantillonnage non uniforme. (d) Points aberrants.

courbe de niveau (typiquement l'iso-zéro). Les fonctions scalaires non signées sont plus faciles à calculer étant donné qu'elles ne nécessitent pas que les normales soient orientées. L'extraction de surface passe alors par une recherche de minimum local. Néanmoins, contrairement à ce que l'on pourrait penser, le solveur par éléments finis, utilisé par la suite pour calculer les écoulements, ne nécessite pas que la fonction scalaire soit signée pour définir les conditions aux limites. Il ne s'agit donc pas d'une contrainte.

En revanche, les fonctions scalaires signées présentent un avantage majeur relativement à celles qui ne le sont pas : elle contiennent une information topologique supplémentaire sur l'intérieur et l'extérieur de la surface. Cette propriété supplémentaire permet traditionnellement de fermer les trous dans les modèles nuages de points [6], apportant ainsi une solution à la contrainte liée aux données manquantes. Par ailleurs, l'information d'intérieur/extérieur peut être utilisée par la suite si la simulation nécessite d'attribuer des propriétés physiques de l'objet étudié. Ainsi bien que les fonction scalaires non signées sont plus faciles à construire, nous choisissons d'utiliser une fonction scalaire signée.

6. À noter que la fermeture dont nous parlons ici ne crée pas d'information supplémentaire en *"devinant"* la partie occulté du nuage de points. La fermeture considérée ici garantie seulement que le modèle produit sera fermé.

Globale ou locale ? La fonction implicite doit-elle être calculée une seule fois sur une grille prédéfinie (cas d'une méthode globale) ou bien doit-elle être calculé à la demande au moment de l'exécution (méthode locale) ? Les méthodes globales sont plus robustes aux bruit dans les nuages de points mais elles sont plus coûteuses en temps de calcul et moins polyvalentes que les méthodes locales. En effet comme les méthodes locales sont définies sur un voisinage du nuage de point au point de requête auquel elles sont évaluées, elles sont plus adaptées à un contexte de maillage/remaillage adaptatif tel que le notre, où le maillage volumique est modifié à chaque itération de la simulation.

2.4.2.2 Moindres Carrés Glissants Implicites (IMLS)

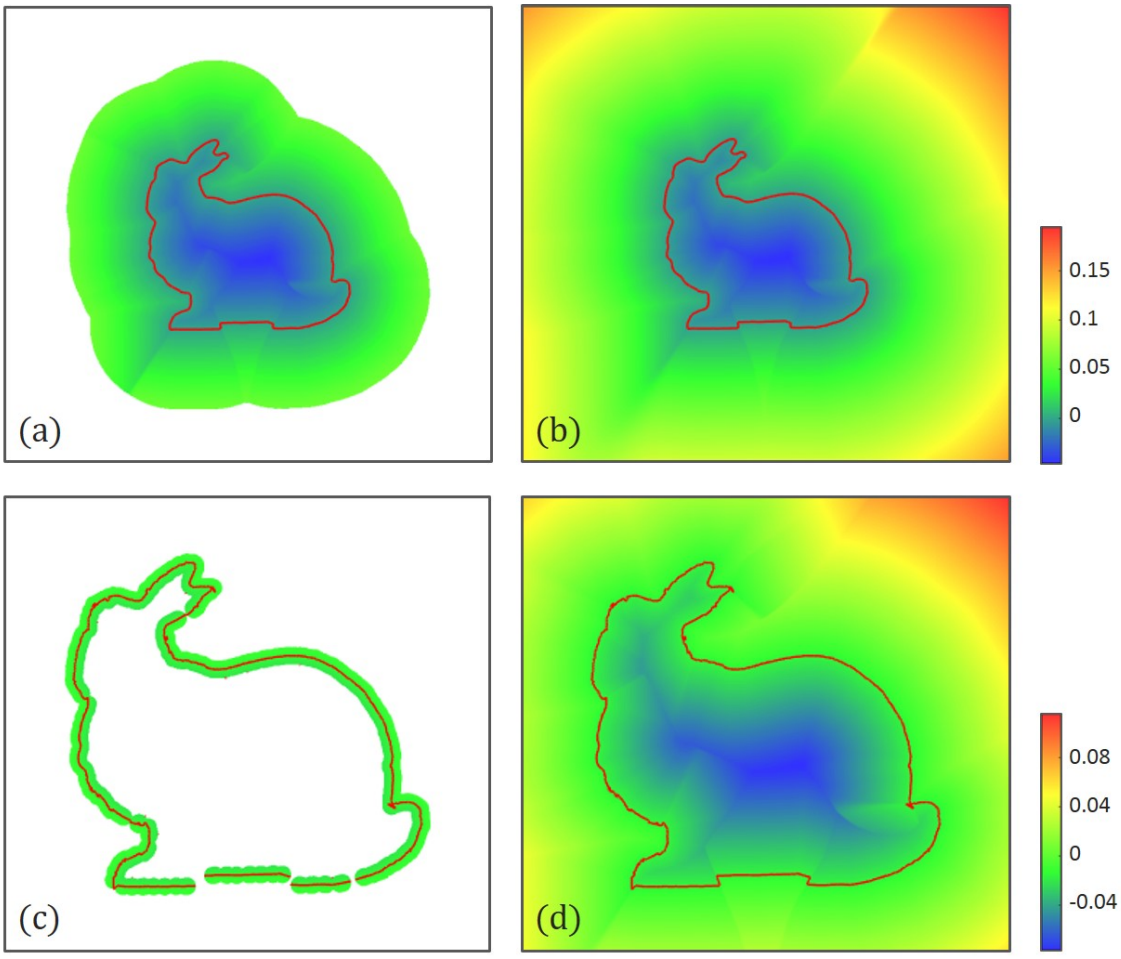


FIGURE 2.18 – Comparaison entre les méthodes IMLS avec un noyau Gaussien (à gauche) et EIMLS (à droite). La ligne supérieure est calculée avec $h_0 = 0.0015$ et la ligne inférieure est calculée avec $h_0 = 0.0001$. La formulation par IMLS n'est pas définie sur tout le domaine de calcul (parties blanches sur le figure) alors que la formulation par EIMLS l'est.

Dans ce paragraphe, nous présentons la représentation par IMLS et montrons pourquoi elle nécessite d'être étendue pour être adaptée à notre méthode. Les notations utilisées sont rassemblées dans le tableau 2.1

La représentation par IMLS a été introduite par (Kolluri, 2005). Il donne ainsi une formule analytique à l'opérateur de projection par MLS traditionnellement utilisé. Elle est définie par la formule suivante :

$$\alpha_{IMLS}(q) = \frac{\sum_{i\in\mathcal{N}} w_i(q)((q-p_i).n_i)}{\sum_{i\in\mathcal{N}} w_i(q)} \tag{2.1}$$

Comme un point p_i et sa normale n_i définissent un plan tangent local à la surface. Le terme $(q-p_i).n_i$ correspond à la distance signée de q à ce plan. L'équation 2.1 peut ainsi être vue comme la somme pondérée des distances signées aux plans tangents à $\mathcal{S}_\mathcal{P}$ aux points p_i.

w_i est la fonction de poids associée au point i. Elle peut être calculée à partir de fonction de poids génériques :

$$w_i(q) = \phi\left(\frac{\|p_i - q\|}{h_i(q)}\right) \tag{2.2}$$

h_i est un paramètre d'extension spatiale associé au point i. Il est souvent choisi constant pour tous les points. Ainsi si l'échantillonnage du nuage est uniforme, alors il peut être choisi constant est égal à l'amplitude du bruit dans le nuage $h_i = h_0$. Si le nuage de points est épars, alors il doit être choisi de l'ordre de grandeur de la densité locale de $\mathcal{P}$ au point i. En effet, ce paramètre détermine le rayon d'influence du noyau ϕ.

Le choix de la fonction de poids générique ϕ induit différents comportements de la surface sous-jacente. En voici quelques exemples :

- **Poids Gaussien :** Il s'agit là du poids classiquement utilisé par (Kolluri, 2005)

  $$\phi(x) = e^{-\frac{x^2}{2}} \tag{2.3}$$

  Avec un poids Gaussien, la surface est approximée en lissant le bruit dans le nuage.

  Dans ce cas, comme ϕ n'est pas à support compact, en théorie la fonction implicite α_{IMLS} devrait être définie sur tout $\mathbb{R}^d$. Nous montrons pas la suite pourquoi nous ne pouvons pas compter sur cette propriété en pratique.

- **Poids à support compact :** Comme dans (Guennebaud et Gross, 2007) les poids peuvent être définis par des fonctions polynômiales à support compact :

  $$\phi(x) = \begin{cases} (1-x^2)^4 & \text{if } x < 1 \\ 0 & \text{otherwise} \end{cases} \tag{2.4}$$

Ici le poids est à support compact, ce qui implique que la fonction implicite α_{IMLS} n'est définie que sur l'union de boules fermées de rayon h_i et centrées en chaque point p_i. Cette ensemble de définition ne garantit donc pas de couvrir l'ensemble du domaine d'étude (domaine de la simulation numérique).

- **Poids interpolants :** Les poids peuvent également être choisis pour interpoler les points. Pour ce faire, il suffit de les choisir tels qu'il comportent une singularité en zéro, comme par exemple :

$$\phi(x) = \frac{1}{x^2} \tag{2.5}$$

Comme cela a été précédemment évoqué, pour les nuages de points réels, une surface interpolante n'est pas désirable puisqu'elle induit un sur-ajustement du bruit dans le nuage.

Dans la suite nous ne considérons donc que des poids Gaussiens (donnés par l'équation 2.3). En théorie, ϕ est alors définie pour tout $x \in \mathbb{R}^+$ et donc α_{IMLS} devrait être définie sur tout $\mathbb{R}^d$. Cependant, en pratique, cela est vrai jusqu'à ce que $\phi(x)$ tombe en dessous de la précision machine.

Supposons alors $10^{-\gamma}$ notre limite numérique, en dessous de laquelle tout nombre est représenté par 0 en machine :

$$\phi(x) < 10^{-\gamma} \Rightarrow x > \sqrt{2\gamma \log(10)} = l_\gamma \tag{2.6}$$

Cela signifie que le poids Gaussien est en réalité à support compact. Ainsi pour un point de requête q suffisamment loin du nuage de point, tout les poids $w_i(q)$ sont nuls et donc α_{IMLS} n'est pas définie. Comme on peut le voir à gauche de la figure 2.18, le domaine de définition pour la méthode IMLS calculée avec des poids Gaussiens **(a)** ne couvre pas l'ensemble du domaine d'étude, et **(b)** peut ne même pas produire une surface fermée pour de très faibles valeurs de h_0.

Cela représente un problème pour notre méthode car celle-ci nécessite d'avoir une fonction implicite définie sur l'ensemble du domaine d'étude afin de pouvoir contrôler l'adaptation de maillage et définir les conditions aux limites pour le solveur CFD. De plus ce problème est mis en exergue dans la cas de l'étude CFD d'un objet (comme pour le jeux de données **lapin**) où le domaine d'étude est généralement choisi beaucoup plus grand que l'objet pour ne pas perturber l'écoulement autour de ce dernier, la fonction implicite doit donc être évaluable très loin de la surface. C'est pour toutes ces raisons que la représentation IMLS nécessite d'être étendue pour être utilisable dans le cadre de la simulation numérique d'écoulements. Dans la suite nous présentons notre nouvelle représentation de surfaces implicite étendue EIMLS.

2.4.2.3 Moindres Carrés Glissants Implicites Étendus (EIMLS)

Introduisons tout d'abord l'opérateur des k plus proches voisins (ou KNN en anglais, pour k *Nearest Neighbors*) sur un nuage de points :

Définition 2.4.1. *L'opérateur des k plus proches voisins sur le nuage de points*
Étant donné $q \in \mathbb{R}^d$ et k un entier positif, l'ensemble des k points les plus proches de q dans $\mathcal{P}$ est noté :

$$\pi_{\mathcal{P}}(k, q) \in P_k(\mathbb{N}) \tag{2.7}$$

Dans le cas spécial où $k = 1$, $\pi_{\mathcal{P}}(1, q)$ définit l'opérateur de projection du plus proche voisin sur $\mathcal{P}$ [7]. Pour simplifier les notations, il est noté :

$$\pi_{\mathcal{P}}(q) \in \mathbb{N} \tag{2.8}$$

Nous proposons alors l'expression étendue pour le paramètre d'extension spatiale :

$$h^E(q) = \max\left(\frac{\|p_{\pi_{\mathcal{P}}(q)} - q\|}{l_\gamma}, h_0\right) \tag{2.9}$$

Où h_0 est le paramètre d'extension spatiale précédemment utilisé qui peut être choisi de l'ordre de grandeur du bruit dans le nuage de points. l_γ est le coefficient de précision numérique pour le poids Gaussien donné par la formule 2.6. Ainsi la représentation étendue par moindres carrés glissants (EIMLS) peut être calculée à partir des équations 2.1, 2.2, 2.3 et 2.9. Avec cette formulation nous pouvons alors garantir qu'au moins un poids w_i ne tombe pas en dessous de la précision numérique $10^{-\gamma}$: celui associé au point $\pi_{\mathcal{P}}(q)$, qui est le point le plus proche de q dans le nuage de points.

Nous pouvons alors voir à droite de la figure 2.18 que EIMLS donne les mêmes résultats que IMLS là où cette dernière est définie. Nous pouvons aussi remarquer que **(b)** la représentation par EIMLS est bien définie sur l'ensemble du domaine d'étude. De plus **(d)** EIMLS est bien plus robuste qu'IMLS aux petites valeurs de h permettant dans tous les cas de définir une surface étendue à tout le domaine d'étude. On peut également remarquer que les valeurs de la fonction scalaire loin de la surface ne sont pas nécessairement exactes. Cela ne représente néanmoins pas un problème pour notre méthode dans la mesure où elle nécessite seulement de la précision proche de la surface. Ce point est éclairci dans la section 2.5.

2.4.3 Implémentation

Calculs de voisinages Si l'on utilise l'équation 2.1 avec un poids qui n'est pas à support compact, nous devrions sommer les contributions de tous les points de $\mathcal{P}$, ce qui n'est pas cohérent avec l'affirmation qu'IMLS est une représentation de surface locale. Néanmoins, comme les poids sont décroissants et tendent vers 0, il

7. À noter que ces deux opérateurs ne sont définis de manière unique que sur $\mathbb{R}^d$ privé d'un ensemble de mesure nulle qui correspond au squelette de $\mathcal{S}_{\mathcal{P}}$, sur lequel les points sont à égale distance d'au moins deux points de la surface. Le manque d'unicité ne représente néanmoins pas un problème pour la suite du raisonnement.

est commun d'utiliser l'approximation suivante. Au lieu de sommer sur $\mathcal{N}$, on ne somme que sur $\pi_{\mathcal{P}}(k, q)$:

$$\widehat{\alpha}_{EIMLS}(q) = \frac{\sum_{i \in \pi_{\mathcal{P}}(k,q)} w(q)((q - p_i).n_i)}{\sum_{i \in \pi_{\mathcal{P}}(k,q)} w(q)} \tag{2.10}$$

Ainsi, la représentation est locale et ne dépend que d'un nombre restreint et localisé de points. En revanche, il faut faire attention au choix du paramètre k : une valeur trop faible produit des résultats erronés et une valeur trop grande est trop coûteuse en temps de calcul. Il dépend également de l'échantillonnage du nuage de points : pour des nuages de points avec un échantillonnage anisotrope, une combinaison entre une recherche de voisinage par rayon et par k plus proches voisins peut fournir de meilleurs résultats. Nous nous affranchissons de ces problèmes par l'étape de sous-échantillonnage décrite dans la section 2.3.

Dans le but d'accélérer la recherche de plus proches voisins, nous utilisons un *k-d tree* (Moore, 1991). Pour rappel, cette structure est présentée dans la section 1.2. Nous utilisons l'implémentation *nanoflann* de la librairie *FLANN* (Muja et Lowe, 2009).

Utilisation de la mémoire L'utilisation de la mémoire pour des nuages de points de quelques dizaines de millions de points n'est pas un problème sur les ordinateurs d'aujourd'hui pour le stockage d'un nuage et de son k-d tree. Néanmoins, dans notre cas, l'algorithme doit s'exécuter dans un contexte massivement parallèle. La mémoire de chaque processus est donc indépendante de celle des autres, même s'ils sont présents sur le même nœud de calcul. Cela signifie que le nuage de points entier doit être chargé par chaque processus. Dans le cas des trois jeux données que nous utilisons, l'utilisation de la mémoire n'est cependant pas un problème compte tenu de leur faible empreinte mémoire.

Pour des nuages de points plus gros, nous pourrions avoir recours à des représentations plus compactes basées sur de la compression. Par exemple basée *octree* (Szymon Rusinkiewicz et Marc Levoy, 2000) ou bien basée sur des graphes orientés acycliques (ou *Directed Acyclic Graphs* (DAG) en anglais) (Kämpe et al., 2013) qui permettent d'atteindre des taux de compression très impressionnants. De plus, pour éviter d'avoir à charger l'ensemble du nuage de points, un mécanisme de chargement depuis le disque dur (qualifié d'*out-of-core*) pourrait être mis en place de manière conjointe. Il permettrait de ne charger que les parties utilisées du nuage de points permettant ainsi de réduire considérablement l'empreinte mémoire de l'algorithme total. Le chapitre 1 présente un exemple de ces méthodes, appliqué au rendu de nuage de points.

2.4.3.1 Visualisation de surface implicite par ray-tracing

Bien que la méthode proposée ne nécessite que peu de paramètres, il est intéressant de pouvoir visualiser la surface définie implicitement par la fonction scalaire signée ainsi construite. Pour cela, la méthode la plus ismple consiste à calculer les valeurs de la fonction sur une grille (régulière ou adaptative) et d'extraire un maillage à l'aide de l'algorithme de *marching cubes*. Cette approche est celle proposée par la majorité des logiciels de visualisation scientifique (comme par exemple Paraview, qui est basé sur la librairie Vtk).

Une autre approche consiste à calculer les valeurs de la fonction sur un octree et de rendre la surface implicite directement par lancer de rayon (ou *ray-tracing*). (Knoll et al., 2006) proposent une méthode sur GPU que nous avons implémenté sur CPU. L'intersection de chaque rayon avec la fonction implicite est calculée par une descente de l'arbre. La fonction implicite est interpolée trilinéairement au milieu de chaque cellule de l'octree par les valeurs calculées aux 8 sommets. Les résultats obtenus sont présentés sur la figure 2.19. Les fréquences d'affichage sont temps-réel pour de petites résolutions d'écran (400x600).

Cette méthode démontre l'intérêt de la représentation implicite de nuage de points pour le rendu de surfaces. Il serait intéressant d'appliquer au cas du nuage de points des méthodes plus complètes qui gèrent notamment le rendu multi-résolution et la gestion de volumes qui ne tiennent pas en mémoire principale (Laine et Karras, 2011 ; Miller et al., 2011 ; Crassin et Green, 2012 ; Crassin, Neyret, Sainz et al., 2011 ; Crassin, Neyret, Lefebvre et al., 2009 ; Lefebvre et al., 2005).

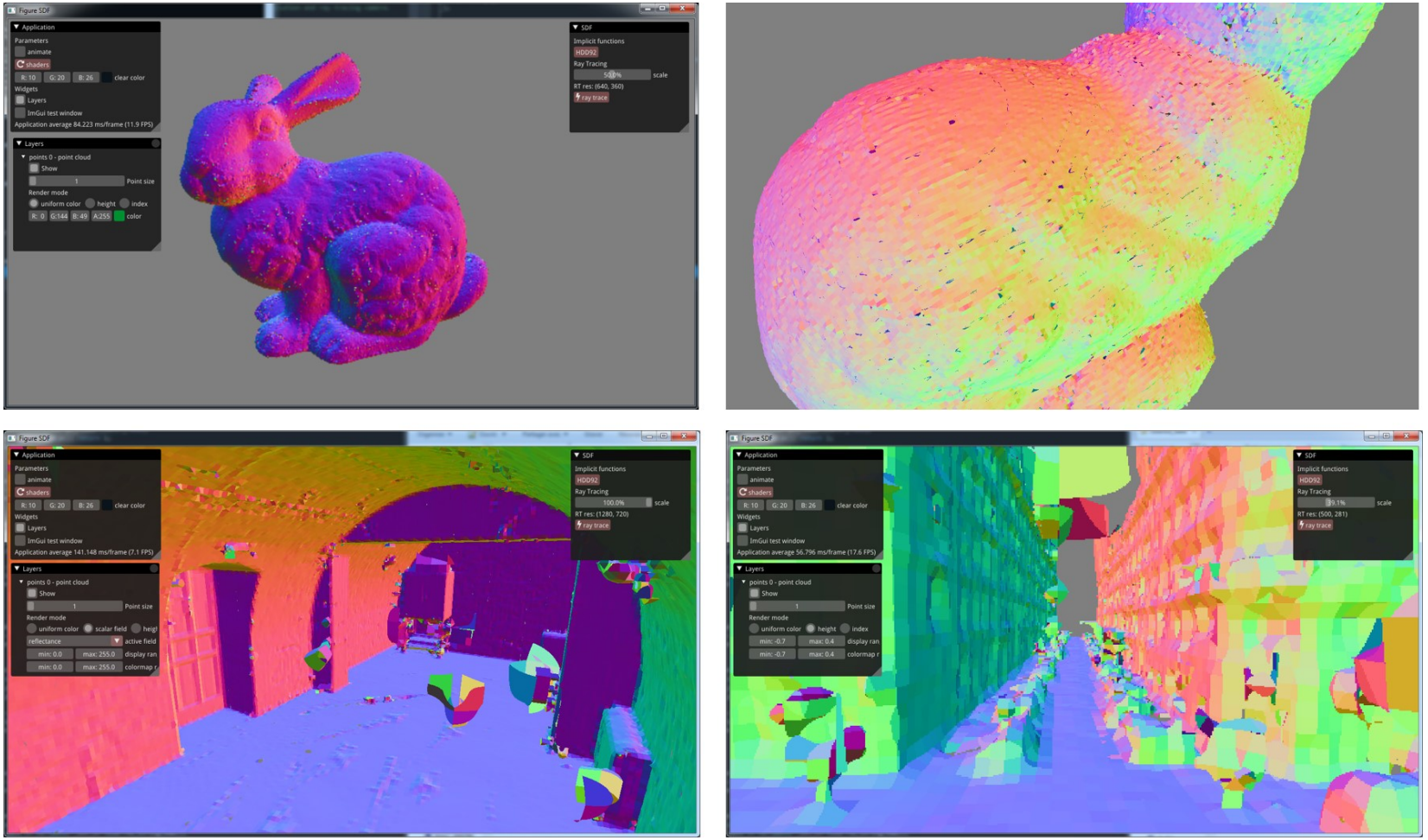


FIGURE 2.19 – Résultats obtenus sur les trois jeux de données.

2.5 Adaptation de maillage anisotrope

Notations	Définition
d	dimension de l'espace
Ω	domaine d'étude
$\mathcal{N} \subset \mathbb{N}$	ensemble d'indices de nœuds
$\mathcal{X} = \{\mathbf{X}^i, i \in \mathcal{N}\} \subset \mathbb{R}^d$	ensemble de nœuds
$\mathbf{X}^{ij} = \mathbf{X}^j - \mathbf{X}^i \in \mathbb{R}^d$	vecteur de l'arrête qui relie les deux nœuds i et j
$\Gamma(i) \subset \mathcal{N}$	ensemble de nœuds connectés au nœud i
$\mathbf{G}^i$	opérateur de reconstruction de gradient au nœud i
$\mathbb{M}^i$	métrique unitaire du maillage au nœud i

TABLE 2.2 – Notations et définitions pour la section 2.5

Dans cette partie nous présentons l'étape d'adaptation de maillage de la méthode proposée. Cette étape permet de construire un maillage volumique anisotrope et adapté autour de la surface définie par un nuage de points. L'adaptation est réalisée à l'aide de la fonction scalaire implicite calculée par EIMLS directement à partir du nuage de points. L'adaptation de maillage anisotrope est alors réalisée en minimisant une métrique d'erreur calculée *a posteriori* (Thierry COUPEZ, 2011). Les notations utilisées dans cette partie sont rassemblées dans le tableau 2.2.

2.5.1 État de l'art

Comme cela a été précédemment évoqué, la méthode est basée sur un solveur par éléments finis et en volumes immergés des équations de Navier-Stokes incompressibles et transitoires afin de calculer l'écoulement de fluides autour de géométries définies par des nuages de points. Comme cela est précisé dans (MITTAL et IACCARINO, 2005), la simulation par volumes immergés a été introduite par (PESKIN, 1972). Cette méthode permet de réaliser la simulation sur une grille qui occupe l'intégralité du domaine d'étude : aussi bien l'intérieur que l'extérieur des obstacles. A la différence des grilles ajustées qui n'occupent que la partie fluide du domaine (il n'y a pas de noeuds de calcul à l'intérieur des obstacles), dans le cas du paradigme classiquement utilisé pour la simulation CFD. Cela permet de simplifier le processus de création de la grille dans la mesure où sa complexité devient ainsi indépendante de celle du modèle 3D qui sert à définir les conditions aux limites. Cette méthode a été initialement développée pour des grilles Cartésiennes mais elle a récemment été adaptée aux grilles non structurées anisotropes. Un maillage volumique non structuré anisotrope permet d'obtenir une meilleure précision à nombre de nœuds de calcul fixé, il requiert néanmoins l'utilisation d'algorithme de maillage plus complexe.

(Thierry COUPEZ, 2000 ; Thierry COUPEZ et al., 2013) ont introduit un algorithme de maillage basé sur des optimisations locales. Ainsi comme il est basé uni-

quement sur des modifications locales d'un maillage existant, cet algorithme peut être utilisé à la fois pour créer un maillage, mais également pour le modifier, on parle alors de remaillage. Il peut être également utilisé dans un contexte massivement parallèle (T. Coupez et al., 2000). Comparé aux autres algorithmes de maillage plus traditionnels, il est bien moins coûteux en temps de calcul lorsqu'il s'agit de modifier un maillage existant, dans la mesure où le maillage n'est pas reconstruit à partir de zéro. Ainsi le maillage peut être adapté autour de la géométrie de l'objet immergé ainsi qu'autour de l'écoulement, entre chaque itération du solveur fluide. La théorie associée à cet algorithme est détaillée dans l'annexe B.

2.5.2 Une fonction de distance signée et tronquée pour l'adaptation de maillage anisotrope

Soit Ω le domaine de simulation et $\omega_{\mathcal{P}}$ le sous-domaine de Ω délimité par une surface définie par un nuage de points $\mathcal{S}_{\mathcal{P}} = \delta\omega_{\mathcal{P}}$. La fonction scalaire implicite α_{EIMLS} définie précédemment approxime la fonction de distance signée à la surface. Soit $D(q, \mathcal{S}_{\mathcal{P}})$ la fonction de distance non signée à $\mathcal{S}_{\mathcal{P}}$, la fonction α_{EIMLS} vérifie proche de la surface :

$$\alpha_{EIMLS}(q) \approx \begin{cases} -D(q, \mathcal{S}_{\mathcal{P}}) & \text{if } q \in \omega_{\mathcal{P}} \\ D(q, \mathcal{S}_{\mathcal{P}}) & \text{else} \end{cases} \tag{2.11}$$

Si $\mathcal{S}_{\mathcal{P}}$ est une surface lisse, alors α_{EIMLS} est différentiable presque partout et :

$$\|\nabla \alpha_{EIMLS}\| = 1$$

Dans la suite nous considérons une approximation P^1 sur un maillage volumique tétrahédrique de la fonction α_{EIMLS}. L'erreur d'interpolation commise sur le maillage est donc d'ordre 2. C'est pour cette raison qu'il est nécessaire d'introduire artificiellement une forte variation de la dérivée seconde de cette fonction proche de la surface si l'on désire adapter le maillage autour d'elle. Pour ce faire, nous utilisons une fonction auxiliaire tangente hyperbolique :

$$f_\epsilon(x) = \epsilon tanh\left(\frac{x}{\epsilon}\right) \tag{2.12}$$

Comme on peut le voir sur la figure 2.20 f_ϵ est conçue pour garder une dérivée égale à 1 autour de $x = 0$ quelle que soit la valeur de ϵ. Sa dérivée seconde augmente en valeur absolue lorsque ϵ diminue. En l'appliquant à α_{EIMLS} on obtient alors une fonction de distance signée tronquée et lisse :

$$\alpha_\epsilon(q) = f_\epsilon\left(\alpha_{EIMLS}(q)\right) \tag{2.13}$$

On peut observer l'influence du paramètre ϵ sur $\alpha_\epsilon(x)$ pour différentes valeurs de ϵ dans la dernière rangée d'images sur la figure 2.23. Cette figure montre également l'influence de ϵ sur l'algorithme d'adaptation de maillage.

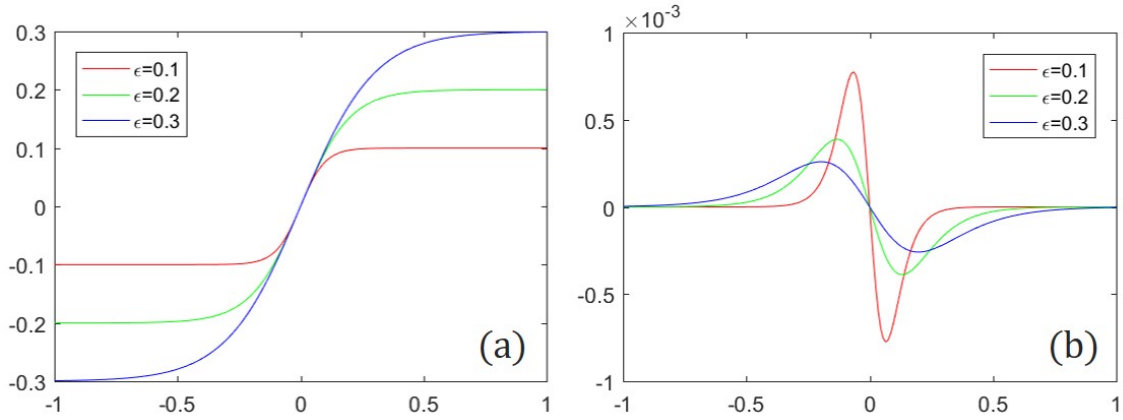


FIGURE 2.20 – Fonction f_ϵ (a) et sa dérivée seconde (b) pour différentes valeurs de ϵ, $\epsilon = 0.1$, $\epsilon = 0.2$, $\epsilon = 0.3$.

On peut également noter que comme on utilise ici une fonction de distance tronquée, celle-ci ne nécessite pas d'être numériquement précise loin de la surface. Cette remarque fait écho à celle précédemment formulée dans la section 2.4. La représentation EIMLS ne garantit pas d'être exacte loin de la surface cependant elle garde la cohérence de signe ce qui est parfaitement adapté à notre problème.

2.5.3 Estimation *a posteriori* de l'erreur d'interpolation

(Thierry COUPEZ, 2011) a introduit un estimateur d'erreur *a posteriori* basé sur le tenseur de distribution de longueurs calculé aux nœuds, et sur une analyse de l'erreur sur les arrêtes. (Thierry COUPEZ et al., 2013) ont ensuite étendu cette méthode afin de contrôler le nombre de nœuds total dans le maillage adapté résultant.

Soit $u \in \mathcal{C}^2(\Omega)$ un champ scalaire connu aux nœuds $\mathcal{X} = \{\mathbf{X}^i \in \mathbb{R}^d, i \in \mathcal{N}\}$ d'un maillage. Nous notons :

$$U^i = u(\mathbf{X}^i) \tag{2.14}$$

$$U^{ij} = U^j - U^i \tag{2.15}$$

(Thierry COUPEZ, 2011) montre que l'opérateur d'interpolation sur le gradient est donné par :

$$\mathbf{G}^i = \left(\sum_{j \in \Gamma(i)} \mathbf{X}^{ij} \otimes \mathbf{X}^{ij} \right)^{-1} \sum_{j \in \Gamma(i)} U^{ij} \mathbf{X}^{ij} \tag{2.16}$$

Cet opérateur peut être utilisé pour estimer l'erreur d'interpolation le long de chaque arrête :

$$e_{ij} = max(|\mathbf{G}^{ij}.\mathbf{X}^{ij}|, r) \tag{2.17}$$

Où r est un paramètre de régularisation qui permet d'assurer que e_{ij} n'est jamais nulle. En pratique, il peut être choisi de l'ordre de $\epsilon/20$.

Étant donné un maillage initial , on peut construire un champ métrique unitaire $\{\mathbb{M}^i\}_{i\in\mathcal{N}}$ qui représente la distribution statistique des arrêtes en chaque nœud.

$$\mathbb{M}_i = \frac{|\Gamma(i)|}{d}\left(\sum_{j\in\Gamma(i)} \mathbf{X}^{ij}\otimes\mathbf{X}^{ij}\right)^{-1} \tag{2.18}$$

Ce champ métrique peut être vu comme une représentation du maillage puisqu'il contient l'information des dimensions des éléments du maillage autour de chaque nœud. Inversement, à partir d'un champ métrique donné, on peut construire un maillage dont la dimension des éléments est conforme à celle décrite par le champ métrique. Utilisons donc cette dernière observation pour construire un nouveau champ métrique $\widetilde{\mathbb{M}}^i$ à partir d'un ensemble de coefficients d'étirement appliqués à toutes les arrêtes du maillage initial. Supposons alors que chaque arrête $\{i,j\}$ est étirée d'un coefficient $s_{ij}\in\mathbb{R}^+$:

$$\begin{cases} \widetilde{\mathbf{X}}^{ij} = s_{ij}\mathbf{X}^{ij} \\ \widetilde{e}_{ij} = s_{ij}^2 e_{ij} \end{cases} \tag{2.19}$$

Soit alors n_{ij} le nombre de nœuds créés le long de chaque arrête $\{i,j\}$:

$$n_{ij} = s_{ij}^{-1} = \sqrt{\frac{e_{ij}}{\widetilde{e}_{ij}}} \tag{2.20}$$

Supposons alors que l'erreur d'interpolation le long de chaque arrête est constante sur tout le maillage $\widetilde{e}_{ij} = e$. De cette manière, l'erreur est équitablement répartie sur le maillage de telle sorte à ce que les régions sur-détaillées sont rendues plus grossières en allongeant les éléments, et les région trop grossières sont raffinées, tout cela de manière anisotrope. Le problème reste cependant mal posé dans la mesure où on ne connait pas la valeur de cette erreur et qu'elle dépend du nombre de nœuds du maillage résultant. Supposons donc que le nombre de nœuds du maillage final est égal à N.

(Thierry Coupez et al., 2013) montrent que le nombre de nœuds créés au nœud i est donné par :

$$n^i(e) = e^{-d/2}\underbrace{det\left(\sum_{j\in\Gamma(i)}\sqrt{e_{ij}}\frac{\mathbf{X}^{ij}}{\|\mathbf{X}^{ij}\|}\otimes\frac{\mathbf{X}^{ij}}{\|\mathbf{X}^{ij}\|}\right)}_{n^i(1)} \tag{2.21}$$

De ce fait, le nombre total de nœuds dans le maillage résultant est donné par :

$$N = e^{-d/2} \sum_i n_i(1) \tag{2.22}$$

On peut alors exprimer l'erreur totale e en fonction de N :

$$e = \left(\frac{\sum_i n_i(1)}{N} \right)^{2/d} \tag{2.23}$$

Le nouveaux champ métrique, qui correspond à une erreur équirépartie sur l'ensemble du domaine, est alors donné par :

$$\widetilde{\mathbb{M}}_i(e) = \frac{1}{e} \frac{|\Gamma(i)|}{d} \left(\sum_{j \in \Gamma(i)} \frac{1}{e_{ij}} \mathbf{X}^{ij} \otimes \mathbf{X}^{ij} \right)^{-1} \tag{2.24}$$

2.5.4 Adaptation de maillage anisotrope autour de surfaces définies par points

La figure 2.21 montre les principales étapes de la construction d'un champ métrique adapté autour d'une fonction scalaire échantillonnée sur un maillage, présentées dans le paragraphe précédent. La rangée supérieure correspond à un maillage uniforme et la rangée inférieure à un maillage déjà adapté. La même fonction scalaire α_ϵ est échantillonnée sur les deux grilles, avec $h_0 = 0.003$ et $\epsilon = 0.005$. Le champ scalaire correspondant est représenté sur la première colonne.

Le champ métrique unitaire $\mathbb{M}^i$ calculé uniquement à partir du maillage est représenté sur la deuxième colonne à l'aide d'ellipsoïdes centrés en chaque nœud. Si on considère la distance associée à la métrique :

$$d_{\mathbb{M}}(x, y) = \sqrt{(y - x)^t \mathbb{M}(y - x)} \tag{2.25}$$

Chaque ellipsoïde représente l'ensemble des points à une distance $d_{\mathbb{M}}$ inférieure ou égale à 0.5 du nœud. On peut noter que, comme cela a été décrit plus haut, cette représentation montre bien que ce champ métrique unitaire décrit bien les dimensions des éléments du maillage.

La troisième colonne représente l'opérateur de reconstruction du gradient $\mathbf{G}^i$. On peut observer que sa variation perpendiculairement à la surface est plus brutale sur le maillage uniforme que sur le maillage adapté. Le nombre de nœuds créés par arrête n_{ij} (quatrième colonne) est anisotrope proche de la surface du maillage uniforme. Le champ métrique résultant $\widetilde{\mathbb{M}}^i$ (cinquième colonne) est donc également anisotrope. Le maillage qui en résulterait (après une étape de remaillage) serait alors grossi loin de la surface et raffiné de manière anisotrope, proche de la surface. La métrique résultant pour le maillage déjà adapté est très similaire à la métrique du maillage initial $\mathbb{M}^i$, ce qui signifie que le maillage est déjà correctement adapté sur l'ensemble du domaine.

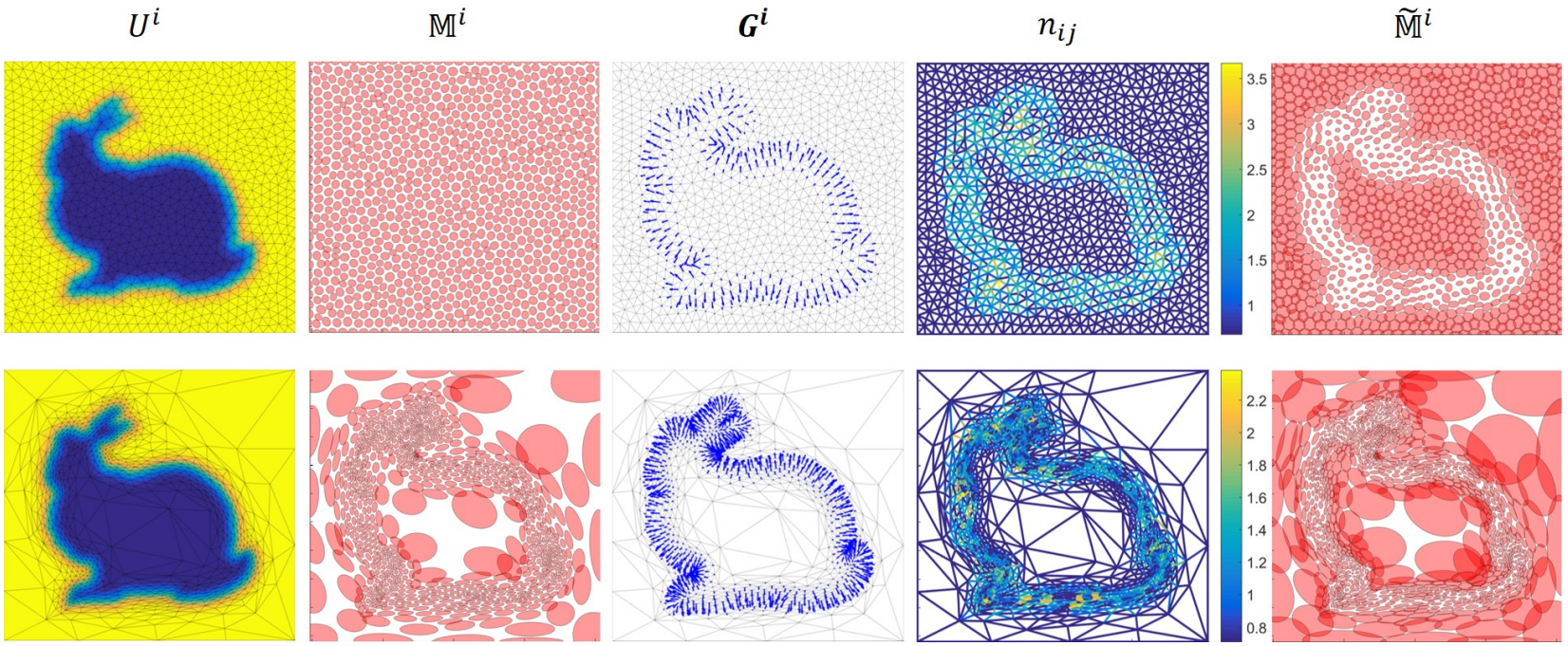


FIGURE 2.21 – Étapes de la construction d'un champ métrique adapté. (Rangée supérieure) en commençant avec un maillage uniforme. (Rangée inférieure) en commençant avec un maillage déjà adapté. Voir sous-section 2.5.4 pour plus de détails.

2.6 Résultats

Nous montrons dans cette section la capacité de la méthode proposée à simuler des écoulements autour de géométries définies par nuages de points. Ainsi, nous montrons dans un premier temps des résultats sur l'adaptation de maillage autour de surfaces en 2D et en 3D. Nous présentons ensuite des simulations d'écoulement en 2D et en 3D sur les trois jeux de données introduits dans la section 2.3. Tous les calculs ont été réalisés avec le code ICI-lib développé à l'Institut de Calcul Intensif (ICI) de l'école Centrale de Nantes. Ce code implémente un mailleur tétrahédrique par optimisation locale et un solveur Navier Stokes incompressible et transitoire par élément finis en volumes immergés.

2.6.1 Adaptation de maillage : reconstruction de géométrie

Nous présentons d'abord la capacité() de la méthode proposée à capturer une surface définie implicitement par EIMLS sur un nuage de points. Le fait de capturer correctement une surface représente un enjeu majeur pour la méthode. En effet, cela définit dans quelle mesure les détails géométriques sont représentés et donc pris en compte par le solveur.

2.6.1.1 Adaptation itérative de maillage

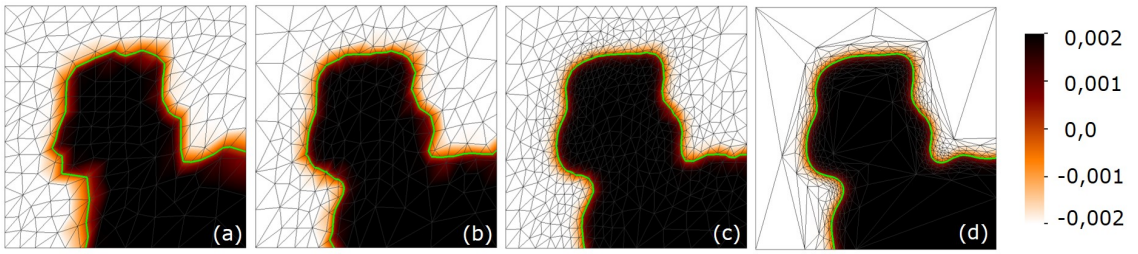


FIGURE 2.22 – Adaptation de maillage autour d'une surface définie par points en 2D. L'iso-zéro de la fonction implicite interpolée sur le maillage est représentée en vert. L'échelle sur la droite donne les valeurs de la fonction EIMLS tronquée α_ϵ ($h_0 = 0.003$ and $\epsilon = 0.002$). (a) Maillage initial isotrope. (b) Maillage après 10 étapes de remaillage. (c) Après 15 étapes de remaillage. (d) Après 30 étapes de remaillage.

La figure 2.22 montre la capacité de la méthode à itérativement capturer une surface 2D avec un nombre de nœuds constants. Le maillage initial est isotrope, l'erreur d'interpolation est donc élevée proche de la surface, à cause de la fonction implicite de distance tronquée. Ainsi la surface n'est visuellement pas correctement représentée. L'algorithme de maillage-remaillage est alors utilisé de manière itérative en alternant entre des phases de calcul de la métrique adaptée, et de remaillage. Ainsi la fonction implicite, de même que la métrique adaptée, ne sont calculées qu'une seule fois par itération de l'algorithme. Durant une itération, lorsque de nouveaux

nœuds sont créés, la fonction implicite et la métrique sont interpolés à la position du nouveau nœud. La figure montre que l'algorithme permet avec un budget fixé de nœuds de calcul de capturer la surface de manière précise après seulement 30 itérations de remaillage.

2.6.1.2 Influence d'ϵ

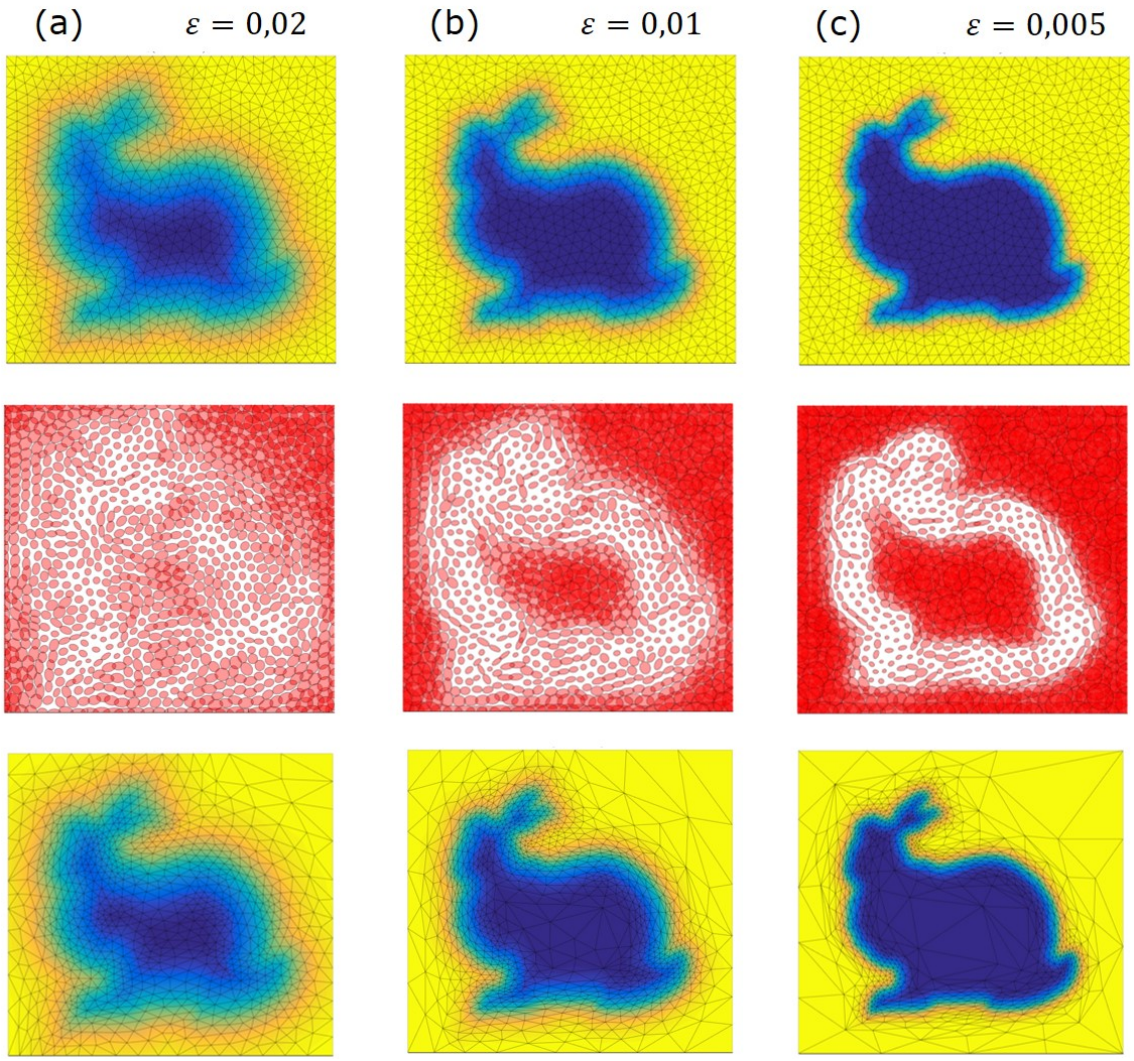


FIGURE 2.23 – Influence du paramètre ϵ de la fonction f_ϵ sur l'adaptation de maillage. (rangée supérieure) Fonction implicite α_ϵ échantillonnée et interpolée sur le même maillage isotrope (l'échelle de couleurs est ajustée sur l'intervalle $[-\epsilon, \epsilon]$ pour chaque colonne). (rangée centrale) Métrique adaptée pour répartir l'erreur d'interpolation commise sur α_ϵ sur l'ensemble du domaine. (rangée inférieure) maillage adapté conformément à la métrique.

Le paramètre ϵ introduit dans la section 2.5 permet de contrôler la variation du gradient de la fonction α_ϵ proche de la surface. Nous montrons en 2D sur la

figure 2.23 l'influence de ce paramètre. Nous pouvons observer qu'une valeur élevée d'ϵ, comparée à l'échelle caractéristique de l'objet (a) produit un champ métrique quasi-constant et isotrope. Le maillage résultant est donc très peu adapté comparativement à celui obtenu avec de plus petites valeurs d'ϵ (b) et (c). On peut également remarquer que plus ϵ est petit, plus la métrique est anisotrope et alongée dans la direction tangente à la surface.

2.6.1.3 Adaptation de maillage anisotrope 3D

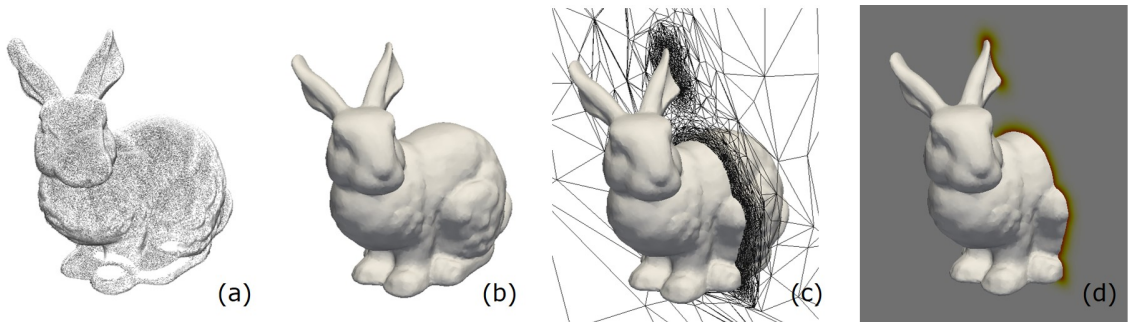


FIGURE 2.24 – Adaptation de maillage anisotrope autour d'une surface 3D définie par un nuage de points. (a) Nuage de points initial **lapin**. (b) Iso-zéro extraite de la fonction implicite α_ϵ calculée aux nœuds du maillage volumique. (c) Coupe du maillage volumique adapté. (d) Coupe de la fonction implicite α_ϵ.

La figure 2.24 montre les résultats de la méthode en adaptation de maillage 3D. L'iso-zéro extraite de la fonction implicite tronquée et interpolée sur le maillage est identique au nuage de points initial. Tous les détails sur la surface sont préservés de même que les trous sont fermés.

2.6.2 Simulations d'écoulements

Dans cette sous-section, nous donnons des résultats de simulation d'écoulements autour de géométries complexes définies par points en 2D et en 3D.

2.6.2.1 Adaptation de maillage multi critères dynamique

La figure 2.25 montre un exemple d'adaptation de maillage dynamique et multi critères en 2D. De l'air est soufflé à partir d'une petite buse située à gauche du domaine à 5 m/s. L'écoulement se sépare lorsqu'il atteint la surface.

Ici le maillage est adapté simultanément autour de la géométrie et autour de l'écoulement. Cette adaptation est réalisée en même temps que la simulation est effectuée et elle ne requiert aucune information apriori sur l'écoulement du fluide. Cela ne serait pas le cas si on avait utilisé le cadre classique des maillages ajustés, puisque le maillage aurait dû être préalablement raffiné aux endroits où le fluide

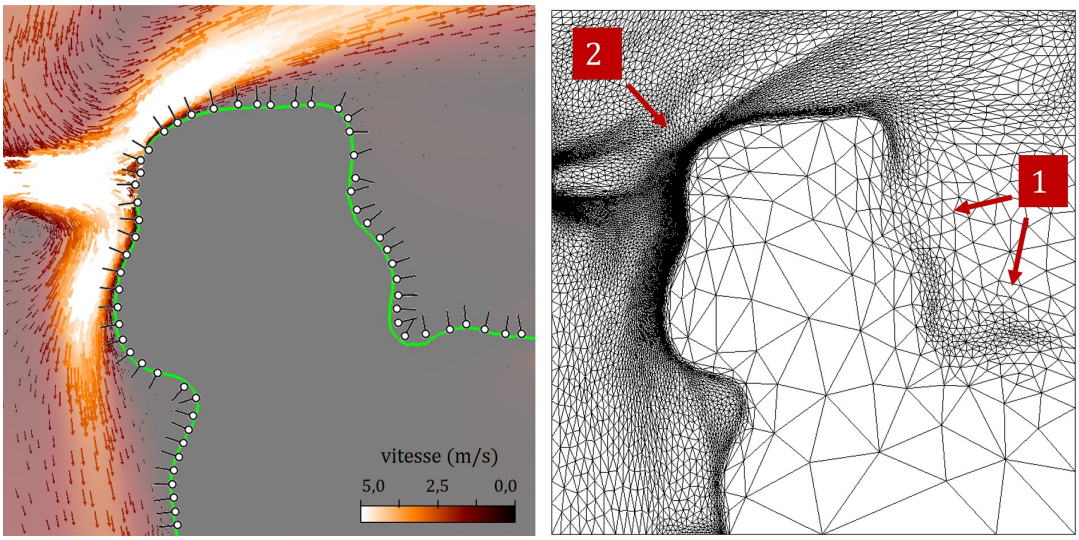


FIGURE 2.25 – (À gauche) Écoulement autour d'une partie du jeu de données 2D. (À droite) maillage volumique adapté simultanément autour de la surface de l'obstacle et autour de l'écoulement.

présente de fortes variations de gradient. Construire un tel maillage nécessiterait l'intervention d'un utilisateur très qualifié, et certainement un va et vient entre la simulation et la conception du maillage.

L'adaptation de maillage donne un meilleur contrôle sur le nombre de noeuds total de la simulation, puisque ceux-ci sont répartis dynamiquement durant la simulation de manière optimale pour minimiser l'erreur d'interpolation sur le maillage. Ici par exemple, dans la région (1) sur la figure, l'écoulement est calme et les variations de vitesse y sont faibles, le maillage est donc grossier. On peut néanmoins noter qu'il est légèrement raffiné au niveau de la surface afin d'en capturer la géométrie. Au contraire, dans la région (2), la variation de la vitesse du fluide est importante et donc le maillage est très raffiné afin de capturer au mieux le comportement du fluide.

La précision géométrique du modèle 3D disponible reste néanmoins très élevée, puisqu'elle est portée par le nuage de points et non pas par le maillage. Ainsi en décorrélant les deux supports, la simulation peut à tout moment se servir de cette précision pour raffiner à la demande la surface.

2.6.2.2 Simulations d'écoulements 2D et 3D

La figure 2.26 donne un autre exemple de simulation sur un nuage de points 2D. On peut voir que le domaine de calcul utilisé est beaucoup plus grand que le modèle lui-même. En effet, la veine d'essai mesure 7 mètres de longueur pour 2 mètres de hauteur, contre 20 cm pour le modèle. La représentation de surface par EIMLS garanti que la fonction implicite est bien définie sur l'ensemble du domaine.

La figure 2.26 montre les allées de Bénard-Von Karman formése par l'écoulement derrière le modèle. Ce phénomène est cohérent avec le nombre de Reynolds de 200 de l'écoulement ($\mathcal{R}_e = \frac{UL}{\mu} = \frac{1*0.2}{0.001} = 200$) .

La figure 2.27 est un exemple d'étude aérodynamique d'un objet scanné à la main. Le nuage de points **lapin** est placé dans un écoulement à 1 m/s. La veine d'essais mesure 5m de long et sa section transversale est de 1m par 1m. La viscosité du fluide est choisie pour atteindre un nombre de Reynolds de 2000. L'écoulement observé est turbulent, ce qui est cohérent avec le régime d'écoulement choisi. On peut ainsi observer les recirculations autour des oreilles du lapin.La simulation a été réalisée avec un nombre de nœuds de 1,000,000 en parallèle sur 100 cœurs de calcul. Avec un pas de temps à 0.0001s, elle a duré 3 heures. Ce type de simulation peut être étendu aux objets industriels pour des applications de retro-ingénierie par exemple.

La figure 2.28 est un exemple de simulation de ventilation réalisée sur le jeu de données **salle** qui a été acquis par scan laser fixe. Un fluide est soufflé par une buse de 20cm par 20cm placée dans le plafond . On remarque que l'air se répartit dans la pièce. La pièce présente deux avancées transversales à un tiers de la longueur environ. Cette avancée semble partitionner l'écoulement du fluide à ce régime d'écoulement. Ce type de simulation pourrait permettre d'évaluer plusieurs solutions d'aménagement pour une pièce ou un ensemble de pièces dans un bâtiment existant sur un modèle 3D très détaillé qui représente fidèlement la réalité. Ce type de simulation ouvre ainsi de nombreuses applications pour l'étude d'aménagement des bâtiments historiques.

Concernant le jeu de données **rue**, la figure 2.29 montre des premiers résultats sur une portion de celui-ci. La simulation a été réalisée sur un maillage comportant 33,000 nœuds. La simulation a duré 35 minutes en parallèle sur 4 cœurs pour 120 pas de temps. Il s'agit d'un exemple de simulation de ventilation ; de l'air est soufflé d'un côté de la rue, dans l'axe de la route. Les résultats montrent que la rue induit un pincement de l'écoulement qui pour effet d'accélérer l'air au centre de la rue. Cet effet d'"entonnoir" est bien connu dans les villes.

Réaliser ce type de simulation à cette échelle est nouveau dans la mesure où il n'est traditionnellement réalisé qu'à l'échelle d'une ville, avec des géométries grossières "boite à chaussures". Simuler l'écoulement de l'air à l'échelle d'une rue ou d'un quartier pourrait être utilisé par exemple, pour étudier l'effet de la pollution sur les bâtiments, pour étudier l'effet de catastrophes naturelles, ou bien encore pour étudier la propagation d'une onde de choc comme dans le cas d'une explosion par exemple.

La figure 2.30 est un exemple de simulation de ventilation réalisée à l'échelle d'une rue sur le jeu de données **rue** complet. Les résultats démontrent les limitations actuelles de la méthode. En effet, le jeu de données **rue** est large et le mailleur éprouve des difficultés pour répartir équitablement les mailles sur l'ensemble du

domaine. Ce phénomène est exacerbé par la présence d'artefacts dans la fonction implicite, dûs à la mauvaise orientation de certaines normales.

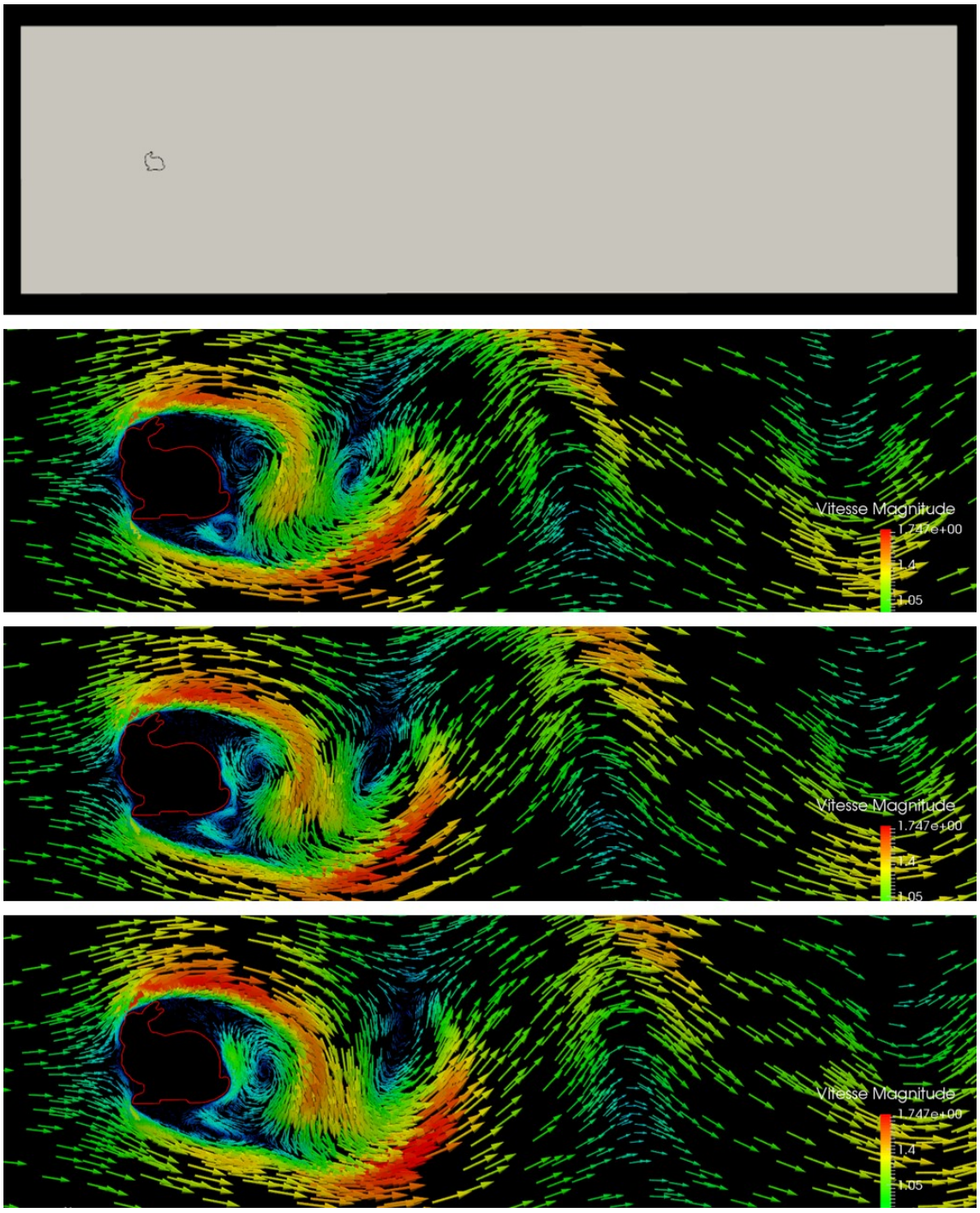


FIGURE 2.26 – (En haut) Domaine de calcul. (En dessous) Champ de vitesse à différents pas de temps de la simulation.

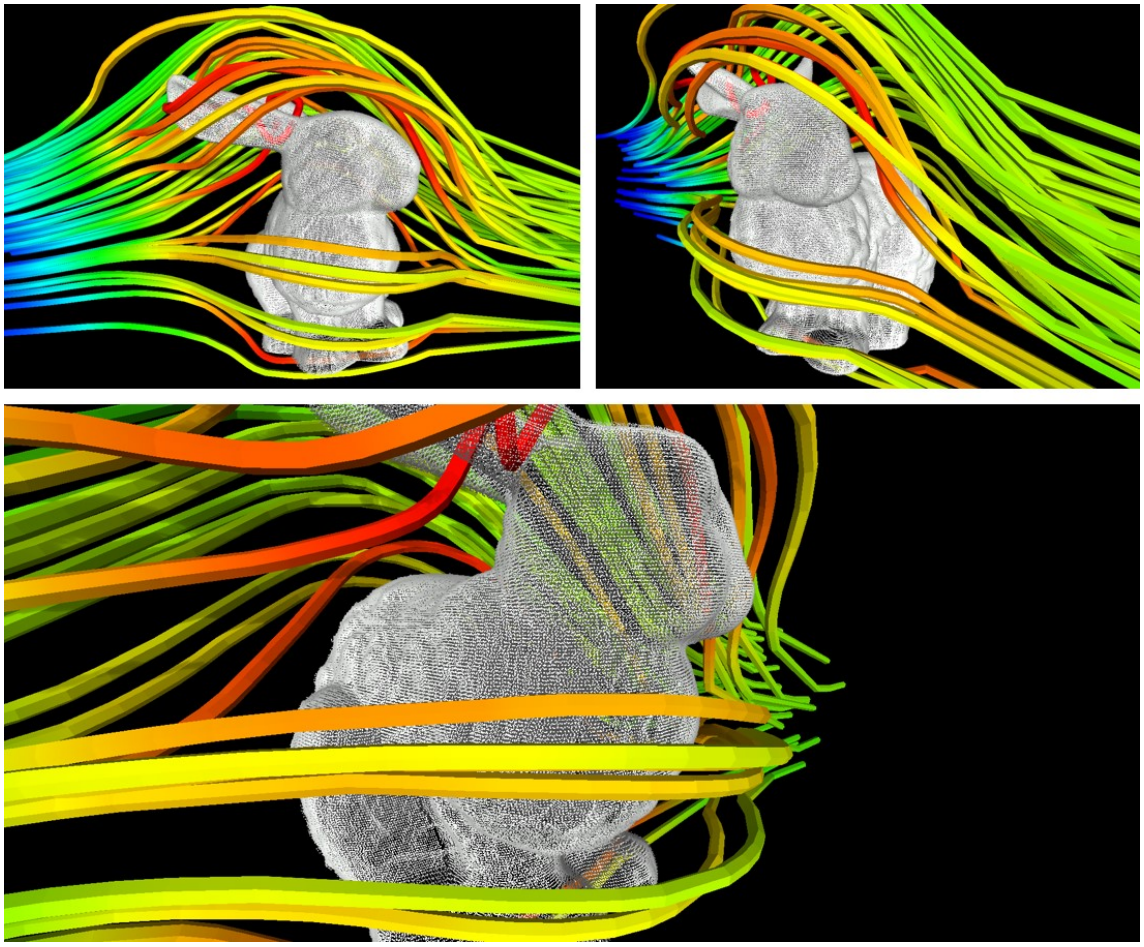


FIGURE 2.27 – Lignes de courant autour du jeu de données **lapin**.

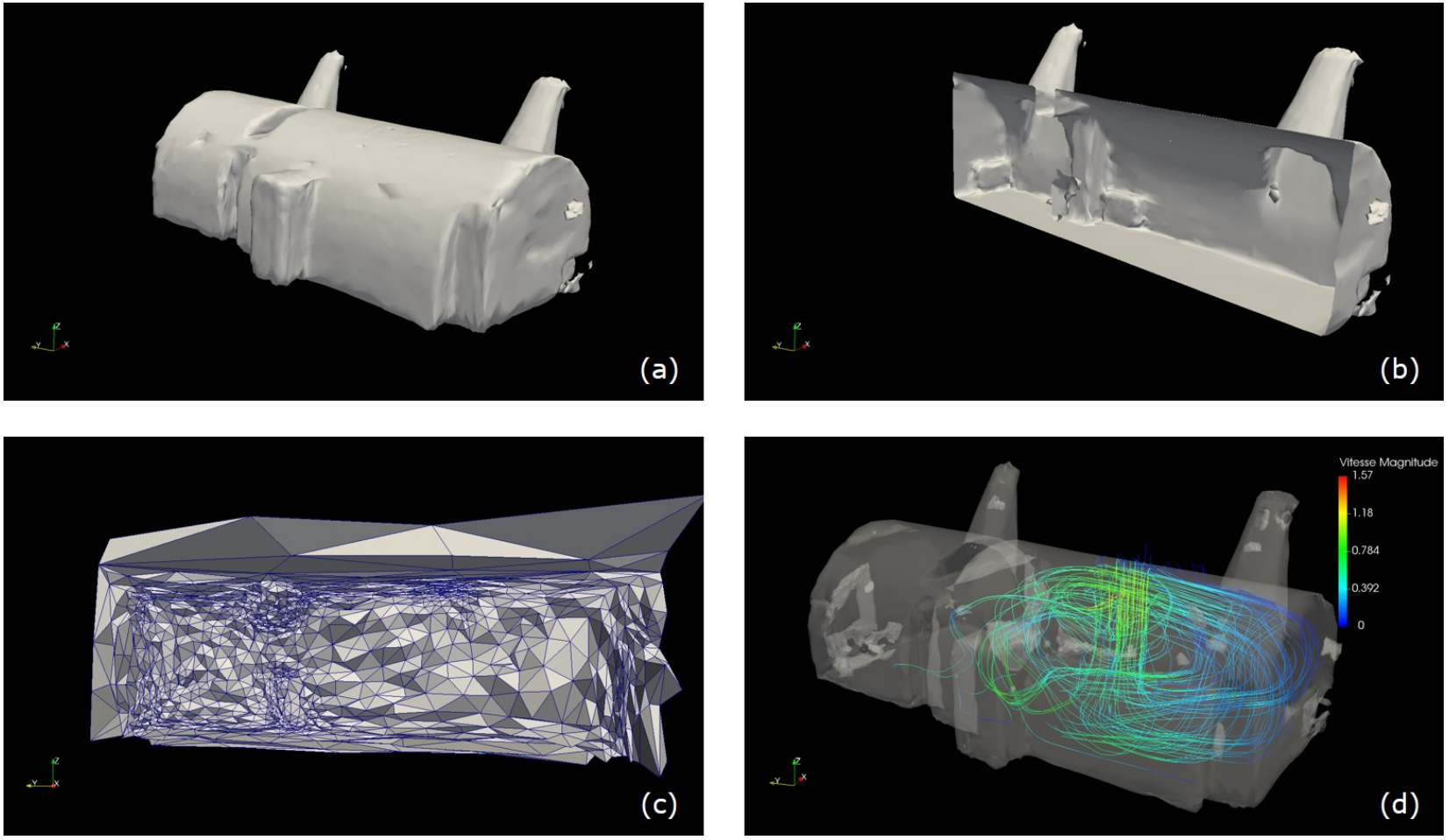


FIGURE 2.28 – Jeu de données **salle**. (a) isosurface extraite de la fonction EIMLS échantillonnée sur le maillage adapté. (b) Coupe de l'isosurface montrant les détails géométriques à l'intérieur de la pièce. (c) Coupe du maillage adapté. (d) Lignes de courant d'une simulation d'une bouche de ventilation dans le plafond de la pièce.

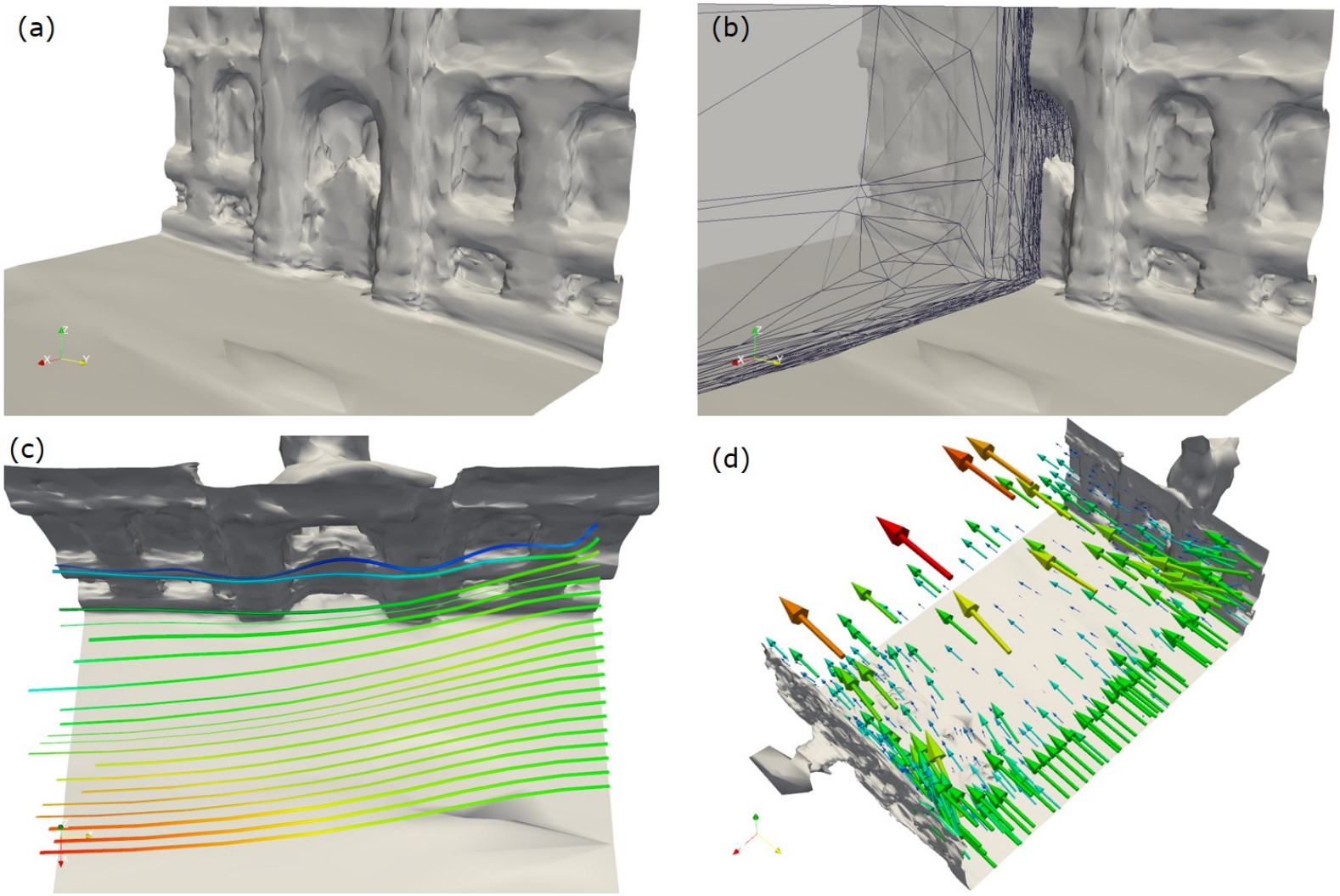


FIGURE 2.29 – Portion du jeu de données **rue**. (a) isosurface adaptée autour d'un portail de la rue. (b) Coupe par transparence du maillage volumique montrant l'anisotropie du maillage à cet endroit. (c) Lignes de courant autour du portail. (d) Champ de vitesse dans la scène.

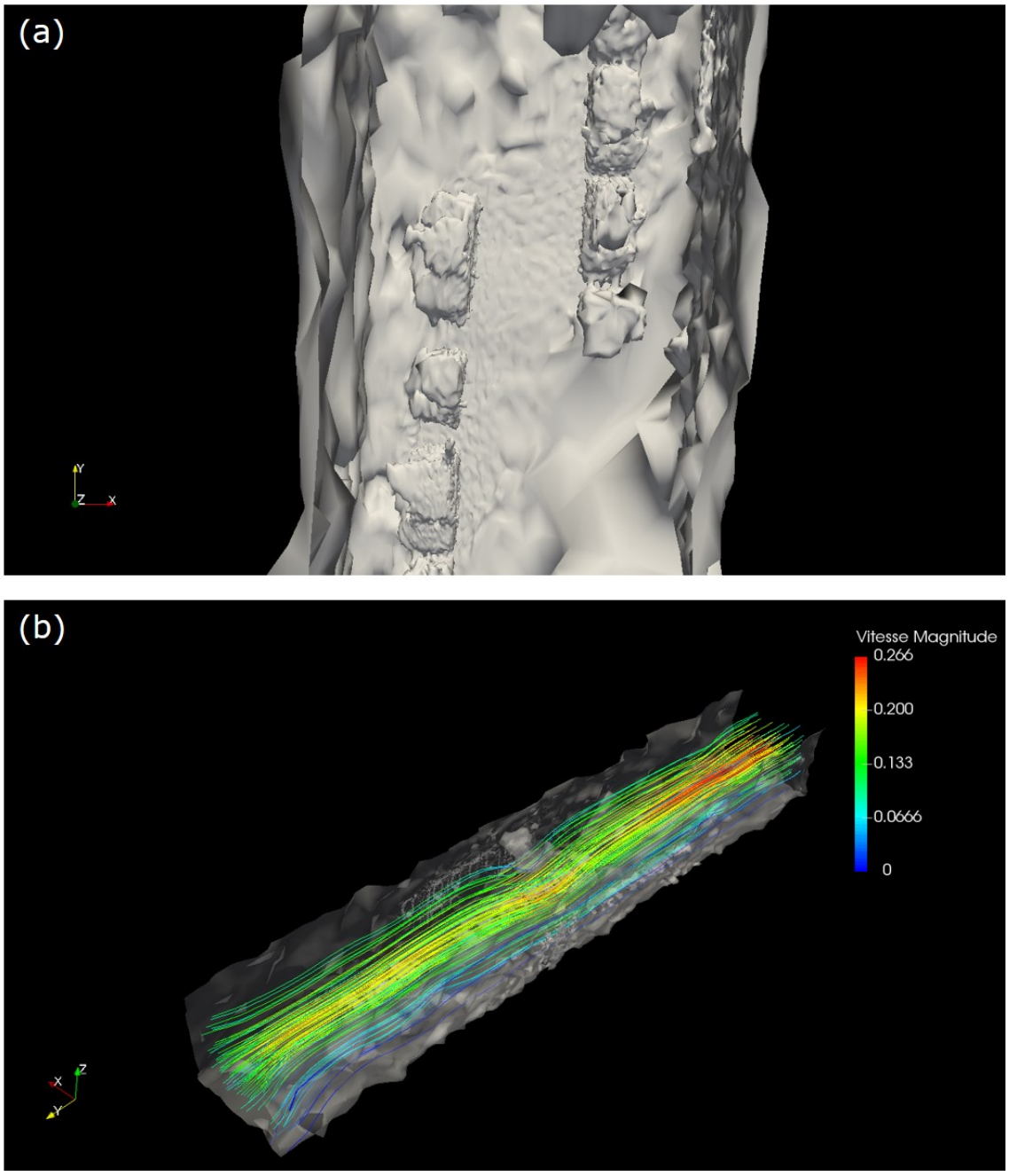


FIGURE 2.30 – Jeu de données **rue**. (a) Détail de la surface adaptée. (b) Lignes de courant d'une simulation de ventilation dans la rue.

2.7 Conclusion

Nous avons proposé dans ce chapitre une méthode qui permet de simuler des écoulements autour de nuages de points 3D sans passer par un maillage surfacique intermédiaire. À notre connaissance, il n'existe aucun travaux similaires dans la littérature. La méthode est basée sur une nouvelle représentation de surface locale et calculée à la volée, par moindres carrés glissants implicites étendus (EIMLS). Cette représentation est une extension de la méthode IMLS pour permettre sa définition sur l'ensemble de l'espace. L'iso-zéro de cette fonction est alors utilisée pour imposer les conditions aux limites d'un solveur éléments finis des équations de Navier-Stokes en volumes immergés. Ce solveur est couplé à un algorithme de maillage adaptatif qui raffine le maillage au cours de la simulation. La méthode proposée permet alors de simuler des écoulements autour de géométries complexes scannées.

Les limites actuelles de la méthode résident dans la sensibilité aux données aberrantes de la fonction implicite. Une manière de résoudre ce problème pourrait, par exemple, consister à utiliser une représentation de surface globale, comme la méthode de Poisson (Kazhdan, Bolitho et al., 2006). La structuration des données pourrait également être un problème si la méthode venait à être utilisée sur des nuages de points massifs, afin de contrôler précisément la quantité de mémoire RAM utilisée par chaque cœur de calcul.

Finalement la méthode proposée permet d'intégrer des géométries scannées, et donc basées sur le réel, directement dans des simulations éléments finis. Nous espérons que cette approche originale ouvrira de nouvelles applications dans le futur, comme par exemple l'utilisation de scans 3D pour les études d'aérologie dans les villes à l'échelle d'une rue ou d'un quartier.

Conclusion

Nous avons présenté dans cette thèse deux contributions en traitements basés points : une en rendu et une en simulation numérique. La première contribution est une méthode de rendu de nuages de points bruts. Elle consiste à rendre un nuage de points comme un solide sans avoir besoin de précalculer des attributs supplémentaires sur le nuage, comme les normales ou les densités locales. Cette méthode s'applique à des nuages de points de taille arbitraire et elle est indépendante de la complexité de la scène rendue. Elle est par ailleurs plus rapide d'au moins un ordre de grandeur que les méthodes similaires de l'état de l'art.

La deuxième contribution présentée est une méthode qui permet de calculer des écoulements autour de géométries scannées et représentées par des nuages de points, sans avoir besoin de calculer un maillage surfacique intermédiaire. Cette méthode originale est basée sur une nouvelle représentation implicite étendue de surface, calculée à la volée à partir d'un nuage de point. Cette méthode permet d'intégrer des modèles très détaillés basés sur le réel dans des simulations numériques en mécanique des fluides. L'approche proposée permet toutefois de garder un contrôle fin sur le nombre de nœuds de la grille de calcul, qui conditionne directement le temps de calcul nécessaire pour la simulation.

A travers ces deux méthodes originales, nous avons couvert une large gamme de méthodes de traitements basés points. Nous avons également porté une attention particulière au traitement de données provenant de différentes sources d'acquisition : lidar mobile, lidar terrestre fixe, photogrammétrie. Cette variété de données d'entrée a permis de confronter les algorithmes proposés à différentes caractéristiques géométriques (bruit, répartition des données, occlusions ...). Par ailleurs, la notion de surface a été centrale dans cette thèse. Nous avons ainsi proposé deux approches pour définir une surface à partir d'un nuage de points : une temps réel basée sur des traitements en espace image, et une autre implicite calculée à la volée en n'importe quel point de l'espace.

Liste des publications

Conférences sans actes

- Hassan Bouchiba, Jean-Emmanuel Deschaud, François Goulette, Coupez Thierry, Luisa Rocha-Da-Silva et Simon Santoso (2016). "Du nuage de points au maillage volumique anisotrope pour la simulation en mécanique des fluides". Journées du Groupe de Travail en Modélisation Géométrique (GTMG). Dijon, France. (Bouchiba, Deschaud et al., 2016)

Conférences avec actes

- Hassan Bouchiba, François Goulette et Jean-Emmanuel Deschaud (2014). "Visualisation Temps Réel de Nuages de Points 3D Structurés Linéairement". Journées de l'Association Française d'Informatique Graphique (AFIG). Reims, France. (Bouchiba, Francois Goulette et al., 2014)
- Hassan Bouchiba, Raphaël Groscot, Jean-Emmanuel Deschaud et Francois Goulette (2017). "High quality and efficient direct rendering of massive real world point clouds" (Poster). Eurographics 2017, the 38th Annual Conference of the European Association for Computer Graphics. Lyon, France. (Bouchiba, Groscot et al., 2017)
- Hassan Bouchiba, Jean-Emmanuel Deschaud et François Goulette, "Raw point cloud deferred shading through screen space pyramidal operators" (Short paper). Eurographics 2018, the 39th Annual Conference of the European Association for Computer Graphics. Delft, Pays-Bas.

Journaux

- **(En cours de soumission)** Hassan Bouchiba, Simon Santoso, Jean-Emmanuel Deschaud, Luisa Rocha-Da-Silva, François Goulette et Thierry Coupez. "Computational Fluid Dynamics on 3D Point Set Surfaces". (Revue visée : Journal of Computational Physics).

Bibliographie

Adamson, Anders et Marc Alexa (2003). « Ray tracing point set surfaces ». In : *Shape Modeling International, 2003.* IEEE, p. 272-279 (cf. p. 26).

Alexa, Marc, Johannes Behr, Daniel Cohen-Or, Shachar Fleishman, David Levin et Claudio T. Silva (2003). « Computing and rendering point set surfaces ». In : *IEEE Transactions on visualization and computer graphics* 9.1, p. 3-15 (cf. p. 102, 103).

Alliez, P., D. Cohen-Steiner, Y. Tong et M. Desbrun (2007). « Voronoi-based Variational Reconstruction of Unoriented Point Sets ». In : *Proceedings of the Fifth Eurographics Symposium on Geometry Processing.* SGP '07. Aire-la-Ville, Switzerland, Switzerland : Eurographics Association, p. 39-48 (cf. p. 101).

Amenta, Nina et Marshall Bern (1999). « Surface reconstruction by Voronoi filtering ». In : *Discrete & Computational Geometry* 22.4, p. 481-504 (cf. p. 99).

Amenta, Nina, Sunghee Choi et Ravi Krishna Kolluri (2001). « The power crust ». In : *Proceedings of the sixth ACM symposium on Solid modeling and applications.* ACM, p. 249-266 (cf. p. 84, 99).

Barazzetti, L., F. Banfi, R. Brumana, G. Gusmeroli, D. Oreni, M. Previtali, F. Roncoroni et G. Schiantarelli (fév. 2015). « Bim from Laser Clouds and Finite Element Analysis : Combining Structural Analysis and Geometric Complexity ». In : *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 5, p. 345-350 (cf. p. 90).

Bavoil, Louis et Miguel Sainz (2008). « Screen space ambient occlusion ». In : *NVIDIA developer information : http ://developers. nvidia. com* 6 (cf. p. 59).

Berger, Matthew, Andrea Tagliasacchi, Lee Seversky, Pierre Alliez, Joshua Levine, Andrei Sharf et Claudio Silva (2014). « State of the art in surface reconstruction from point clouds ». In : *EUROGRAPHICS star reports.* T. 1, p. 161-185 (cf. p. 98, 104).

Bernardini, Fausto, Joshua Mittleman, Holly Rushmeier, Cláudio Silva et Gabriel Taubin (1999). « The ball-pivoting algorithm for surface reconstruction ». In : *IEEE transactions on visualization and computer graphics* 5.4, p. 349-359 (cf. p. 99, 100).

Botsch, M., A. Hornung, M. Zwicker et L. Kobbelt (juin 2005). « High-quality surface splatting on today's GPUs ». In : *Proceedings Eurographics/IEEE VGTC Symposium Point-Based Graphics, 2005.* p. 17-141 (cf. p. 29-31, 34, 56).

Boubekeur, Tamy, Florent Duguet et Christophe Schlick (2005). « Rapid visualization of large point-based surfaces ». In : *EUROGRAPHICS International Symposium on Virtual Reality, Archeology and Cultural Heritage (VAST)*. Eurographics (cf. p. 26).

Boucheny, Christian (2009). « Interactive scientific visualization of large datasets : towards a perceptive-based approach ». Thèse de doct. Ph. D. Thesis, Université Joseph Fourier, Grenoble, France (cf. p. 14).

Bouchiba, Hassan, Jean-Emmanuel Deschaud, François Goulette, Coupez Thierry, Luisa Silva et Santoso Simon (mar. 2016). « Du nuage de points au maillage volumique anisotrope pour la simulation en mécanique des fluides ». fr. In : (cf. p. 133).

Bouchiba, Hassan, Francois Goulette et Jean-Emmanuel Deschaud (oct. 2014). « Visualisation Temps Réel de Nuages de Points 3D Structurés Linéairement ». In : *Journées de l'Association Française d'Informatique Graphique (AFIG)*. Reims, France : AFIG (cf. p. 133).

Bouchiba, Hassan, Raphaël Groscot, Jean-Emmanuel Deschaud et Francois Goulette (avr. 2017). « High quality and efficient direct rendering of massive real-world point clouds ». en. In : (cf. p. 133).

Boulch, Alexandre et Renaud Marlet (août 2012). « Fast and Robust Normal Estimation for Point Clouds with Sharp Features ». en. In : *Computer Graphics Forum* 31.5, p. 1765-1774 (cf. p. 30, 97).

Boulch, Alexandre et Renaud Marlet (2016). « Deep learning for robust normal estimation in unstructured point clouds ». In : *Computer Graphics Forum*. T. 35, p. 281-290 (cf. p. 30, 97).

Calderon, Stéphane et Tamy Boubekeur (2014). « Point morphology ». In : *ACM Transactions on Graphics (TOG)* 33.4, p. 45 (cf. p. 8).

Castellazzi, Giovanni, Antonio Maria D'Altri, Gabriele Bitelli, Ilenia Selvaggi et Alessandro Lambertini (juil. 2015). « From Laser Scanning to Finite Element Analysis of Complex Buildings by Using a Semi-Automatic Procedure ». en. In : *Sensors* 15.8, p. 18360-18380 (cf. p. 91, 92).

Cazals, Frédéric et Joachim Giesen (2006). « Delaunay triangulation based surface reconstruction ». In : *Effective computational geometry for curves and surfaces*. Springer, p. 231-276 (cf. p. 99, 100).

Chnafa, Christophe, Simon Mendez et Franck Nicoud (oct. 2013). « Elucidating the turbulence nature of the intracardiac flow : from medical images to multi-cycle Large Eddy Simulations ». In : *arXiv :1310.3199 [physics]*. arXiv : 1310.3199 (cf. p. 91, 93).

Coupez, T., H. Digonnet et R. Ducloux (déc. 2000). « Parallel meshing and remeshing ». In : *Applied Mathematical Modelling*. Dynamic load balancing of mesh-based applications on parallel 25.2, p. 153-175 (cf. p. 114, 155).

COUPEZ, THIERRY (1991). *GRANDES TRANSFORMATIONS ET REMAILLAGE AUTOMATIQUE*. fr. Google-Books-ID : 0cvlMgEACAAJ (cf. p. 155).

Coupez, Thierry (2000). « Génération de maillage et adaptation de maillage par optimisation locale ». In : *Revue européenne des éléments finis* 9.4, p. 403-423 (cf. p. 113, 155).

Coupez, Thierry (2011). « Metric construction by length distribution tensor and edge based error for anisotropic adaptive meshing ». In : *Journal of Computational Physics* 230.7, p. 2391-2405 (cf. p. 113, 115).

Coupez, Thierry, Ghina Jannoun, Jeremy Veysset et Elie Hachem (2013). « Edge-based anisotropic mesh adaptation for CFD applications ». In : *Proceedings of the 21st International Meshing Roundtable*. Springer, p. 567-583 (cf. p. 113, 115, 116).

Crassin, Cyril et Simon Green (2012). « Octree-based sparse voxelization using the GPU hardware rasterizer ». In : *OpenGL Insights*, p. 303-318 (cf. p. 111).

Crassin, Cyril, Fabrice Neyret, Sylvain Lefebvre et Elmar Eisemann (2009). « Gigavoxels : Ray-guided streaming for efficient and detailed voxel rendering ». In : *Proceedings of the 2009 symposium on Interactive 3D graphics and games*. ACM, p. 15-22 (cf. p. 24, 111).

Crassin, Cyril, Fabrice Neyret, Miguel Sainz, Simon Green et Elmar Eisemann (2011). « Interactive indirect illumination using voxel cone tracing ». In : *Computer Graphics Forum*. T. 30. Wiley Online Library, p. 1921-1930 (cf. p. 111).

Csuri, Charles, Ron Hackathorn, Richard Parent, Wayne Carlson et Marc Howard (1979). « Towards an interactive high visual complexity animation system ». In : *Acm Siggraph Computer Graphics*. T. 13. ACM, p. 289-299 (cf. p. 24).

Curless, Brian et Marc Levoy (1996). « A volumetric method for building complex models from range images ». In : *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*. ACM, p. 303-312 (cf. p. 90, 101).

Dachsbacher, Carsten, Christian Vogelgsang et Marc Stamminger (2003). « Sequential Point Trees ». In : *ACM SIGGRAPH 2003 Papers*. SIGGRAPH '03. New York, NY, USA : ACM, p. 657-662 (cf. p. 22, 41).

Devaux, Alexandre et Mathieu Brédif (2016). « Realtime Projective Multi-texturing of Pointclouds and Meshes for a Realistic Street-view Web Navigation ». In : *Proceedings of the 21st International Conference on Web3D Technology*. Web3D '16. New York, NY, USA : ACM, p. 105-108 (cf. p. 28).

Elseberg, Jan, Dorit Borrmann et Andreas Nüchter (fév. 2013). « One billion points in the cloud – an octree for efficient processing of 3D laser scans ». In : *ISPRS Journal of Photogrammetry and Remote Sensing*. Terrestrial 3D modelling 76, p. 76-88 (cf. p. 20, 23, 45, 97).

Farias, Renato (2014). « Point Cloud Rendering Using Jump Flooding ». Thèse de doct. Citeseer (cf. p. 33).

Fleishman, Shachar, Daniel Cohen-Or et Cláudio T. Silva (2005). « Robust Moving Least-squares Fitting with Sharp Features ». In : *ACM SIGGRAPH 2005 Papers*. SIGGRAPH '05. New York, NY, USA : ACM, p. 544-552 (cf. p. 104).

Gobbetti, Enrico et Fabio Marton (2005). « Far Voxels : A Multiresolution Framework for Interactive Rendering of Huge Complex 3D Models on Commodity Graphics Platforms ». In : *ACM SIGGRAPH 2005 Papers*. SIGGRAPH '05. New York, NY, USA : ACM, p. 878-885 (cf. p. 24).

Gortler, Steven J., Radek Grzeszczuk, Richard Szeliski et Michael F. Cohen (1996). « The Lumigraph ». In : *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '96. New York, NY, USA : ACM, p. 43-54 (cf. p. 33, 64).

Goulette, F., F. Nashashibi, I. Abuhadrous, S. Ammoun et C. Laurgeau (2006). « An integrated on-board laser range sensing system for on-the-way city and road modelling ». In : *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* 34.A (cf. p. 7, 83, 96).

Greene, N. et P. S. Heckbert (juin 1986). « Creating Raster Omnimax Images from Multiple Perspective Views Using the Elliptical Weighted Average Filter ». In : *IEEE Computer Graphics and Applications* 6.6, p. 21-27 (cf. p. 30).

Gross, Markus et Hanspeter Pfister (2011). *Point-based graphics*. Morgan Kaufmann (cf. p. 89).

Grossman, J. P. et William J. Dally (1998). « Point sample rendering ». In : *Rendering techniques' 98*, p. 181-192 (cf. p. 33, 35, 64).

Guennebaud, Gaël et Markus Gross (2007). « Algebraic Point Set Surfaces ». In : *ACM SIGGRAPH 2007 Papers*. SIGGRAPH '07. New York, NY, USA : ACM (cf. p. 104, 107).

Günther, Christian, Thomas Kanzok, Lars Linsen et Paul Rosenthal (2013). « A GPGPU-based pipeline for accelerated rendering of point clouds ». In : (cf. p. 150).

Hoppe, Hugues, Tony DeRose, Tom Duchamp, John McDonald et Werner Stuetzle (1992). « Surface Reconstruction from Unorganized Points ». In : *Proceedings of the 19th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '92. New York, NY, USA : ACM, p. 71-78 (cf. p. 70, 84, 97, 102).

Izadi, Shahram, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison et al. (2011). « KinectFusion : real-time 3D reconstruction and interaction using a moving depth camera ». In : *Proceedings of the 24th annual ACM symposium on User interface software and technology*. ACM, p. 559-568 (cf. p. 90).

Kämpe, Viktor, Erik Sintorn et Ulf Assarsson (juil. 2013). « High Resolution Sparse Voxel DAGs ». In : *ACM Trans. Graph.* 32.4, 101 :1-101 :13 (cf. p. 24, 110).

Katz, Sagi, Ayellet Tal et Ronen Basri (2007). « Direct visibility of point sets ». In : *ACM Transactions on Graphics (TOG)*. T. 26. ACM, p. 24 (cf. p. 31, 32).

Kazhdan, Michael, Matthew Bolitho et Hugues Hoppe (2006). « Poisson surface reconstruction ». In : *Proceedings of the fourth Eurographics symposium on Geometry processing*. T. 7 (cf. p. 26, 84, 101, 102, 130).

Kazhdan, Michael et Hugues Hoppe (2013). « Screened poisson surface reconstruction ». In : *ACM Transactions on Graphics (TOG)* 32.3, p. 29 (cf. p. 26, 101).

Knoll, Aaron, Ingo Wald, Steven Parker et Charles Hansen (2006). « Interactive Isosurface Ray Tracing of Large Octree Volumes ». In : *In Proceedings of the 2006 IEEE Symposium on Interactive Ray Tracing*, p. 115-124 (cf. p. 26, 111).

Kobbelt, Leif et Mario Botsch (déc. 2004). « A survey of point-based techniques in computer graphics ». In : *Computers & Graphics* 28.6, p. 801-814 (cf. p. 29).

Kolluri, Ravikrishna (2005). « Provably Good Moving Least Squares ». In : *ACM SIGGRAPH 2005 Courses*. SIGGRAPH '05. New York, NY, USA : ACM (cf. p. 103, 107).

Kraus, Martin (2009). « The pull-push algorithm revisited ». In : *Proceedings GRAPP* 2009 (cf. p. 64, 66, 67).

Laine, Samuli et Tero Karras (2011). « Efficient sparse voxel octrees ». In : *IEEE Transactions on Visualization and Computer Graphics* 17.8, p. 1048-1059 (cf. p. 111).

Laurent, Gilles, Cyril Delalandre, Grégoire de La Rivière et Tamy Boubekeur (2016). « Forward Light Cuts : A Scalable Approach to Real-time Global Illumination ». In : *Proceedings of the Eurographics Symposium on Rendering*. EGSR '16. Goslar Germany, Germany : Eurographics Association, p. 79-88 (cf. p. 13).

Lefebvre, Sylvain, Samuel Hornus et Fabrice Neyret (2005). « Octree textures on the GPU ». In : *GPU gems* 2, p. 595-613 (cf. p. 111).

Levin, David (2004). « Mesh-independent surface interpolation ». In : *Geometric modeling for scientific visualization*. Springer, p. 37-49 (cf. p. 102).

Levoy, Marc, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg et al. (2000). « The digital Michelangelo project : 3D scanning of large statues ». In : *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co., p. 131-144 (cf. p. 11, 12).

Levoy, Marc et Turner Whitted (1985). *The use of points as a display primitive*. University of North Carolina, Department of Computer Science (cf. p. 24).

Lorensen, William E. et Harvey E. Cline (1987). « Marching cubes : A high resolution 3D surface construction algorithm ». In : *ACM siggraph computer graphics*. T. 21. ACM, p. 163-169 (cf. p. 101).

Machado e Silva, R., C. Esperança, R. Marroquim et A. A. F. Oliveira (fév. 2014). « Image Space Rendering of Point Clouds Using the HPR Operator ». In : *Comput. Graph. Forum* 33.1, p. 178-189 (cf. p. 32).

Macklin, Miles, Matthias Müller, Nuttapong Chentanez et Tae-Yong Kim (juil. 2014). « Unified Particle Physics for Real-time Applications ». In : *ACM Trans. Graph.* 33.4, 153 :1-153 :12 (cf. p. 89).

Marroquim, Ricardo, Martin Kraus et Paulo Roma Cavalcanti (2007). « Efficient Point-Based Rendering Using Image Reconstruction. » In : *SPBG*, p. 101-108 (cf. p. 33, 34, 36).

Marroquim, Ricardo, Martin Kraus et Paulo Roma Cavalcanti (avr. 2008). « Efficient image reconstruction for point-based and line-based rendering ». In : *Computers & Graphics* 32.2, p. 189-203 (cf. p. 64, 66).

Martinez-Rubi, O., Verhoeven, S., Van Meersbergen, M., Schütz, M., Van Oosterom, P., Gonçalves, R., Tijssen, T. et TU Delft (nov. 2015). « Taming the beast : Free and open-source massive point cloud web visualization ». en. In : The Servey Association (cf. p. 12).

Mehra, Ravish, Pushkar Tripathi, Alla Sheffer et Niloy J. Mitra (2010). « Visibility of noisy point cloud data ». In : *Computers & Graphics* 34.3, p. 219-230 (cf. p. 31).

Mihalef, Viorel, Razvan Ioan Ionasec, Puneet Sharma, Bogdan Georgescu, Ingmar Voigt, Michael Suehling et Dorin Comaniciu (juin 2011). « Patient-specific modelling of whole heart anatomy, dynamics and haemodynamics from four-dimensional cardiac CT images ». en. In : *Interface Focus* 1.3, p. 286-296 (cf. p. 92, 93).

Miller, Andrew, Vishal Jain et Joseph L. Mundy (2011). « Real-time Rendering and Dynamic Updating of 3-d Volumetric Data ». In : *Proceedings of the Fourth Workshop on General Purpose Processing on Graphics Processing Units*. GPGPU-4. New York, NY, USA : ACM, 8 :1-8 :8 (cf. p. 111).

Mittal, Rajat et Gianluca Iaccarino (2005). « Immersed Boundary Methods ». In : *Annual Review of Fluid Mechanics* 37.1, p. 239-261 (cf. p. 113).

Moore, Andrew W. (1991). *An Intoductory Tutorial on Kd-Trees* (cf. p. 20, 110).

Muja, Marius et David G. Lowe (2009). « Fast approximate nearest neighbors with automatic algorithm configuration. » In : *VISAPP (1)* 2.331-340, p. 2 (cf. p. 110).

Mullen, Patrick, Fernando De Goes, Mathieu Desbrun, David Cohen-Steiner et Pierre Alliez (juil. 2010). « Signing the Unsigned : Robust Surface Reconstruction from Raw Pointsets ». en. In : *Computer Graphics Forum* 29.5, p. 1733-1741 (cf. p. 101).

Müller, Matthias, Bruno Heidelberger, Marcus Hennix et John Ratcliff (avr. 2007). « Position Based Dynamics ». In : *J. Vis. Comun. Image Represent.* 18.2, p. 109-118 (cf. p. 89).

Newcombe, Richard A., Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J. Davison, Pushmeet Kohli, Jamie Shotton, Steve Hodges et Andrew Fitzgibbon (2011). « KinectFusion : Real-time Dense Surface Mapping and Tracking ». In : *Proceedings of the 2011 10th IEEE Interna-*

tional Symposium on Mixed and Augmented Reality. ISMAR '11. Washington, DC, USA : IEEE Computer Society, p. 127-136 (cf. p. 90, 101).

Oreni, Daniela, Raffaella Brumana, Fabrizio Banfi, Luca Bertola, Luigi Barazzetti, Branka Cuca, Mattia Previtali et Fabio Roncoroni (nov. 2014). « Beyond Crude 3D Models : From Point Clouds to Historical Building Information Modeling via NURBS ». en. In : *Digital Heritage. Progress in Cultural Heritage : Documentation, Preservation, and Protection*. Springer, Cham, p. 166-175 (cf. p. 90, 91).

Öztireli, A. C., G. Guennebaud et M. Gross (avr. 2009). « Feature Preserving Point Set Surfaces based on Non-Linear Kernel Regression ». en. In : *Computer Graphics Forum* 28.2, p. 493-501 (cf. p. 104).

Peskin, Charles S. (1972). « Flow patterns around heart valves : a digital computer method for solving the equations of motion ». Thèse de doct. Sue Golding Graduate Division of Medical Sciences, Albert Einstein College of Medicine, Yeshiva University (cf. p. 113).

Pfister, Hanspeter, Matthias Zwicker, Jeroen van Baar et Markus Gross (2000). « Surfels : Surface Elements As Rendering Primitives ». In : *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '00. New York, NY, USA : ACM Press/Addison-Wesley Publishing Co., p. 335-342 (cf. p. 29).

Pintus, Ruggero, Enrico Gobbetti et Marco Agus (2011). « Real-time Rendering of Massive Unstructured Raw Point Clouds Using Screen-space Operators ». In : *Proceedings of the 12th International Conference on Virtual Reality, Archaeology and Cultural Heritage*. VAST'11. Aire-la-Ville, Switzerland, Switzerland : Eurographics Association, p. 105-112 (cf. p. 35, 36, 56, 58, 59, 62, 65, 67, 72).

Preiner, Reinhold, Stefan Jeschke et Michael Wimmer (2012). « Auto Splats : Dynamic Point Cloud Visualization on the GPU. » In : *EGPGV*, p. 139-148 (cf. p. 33, 35, 69).

Rong, Guodong et Tiow-Seng Tan (2006). « Jump flooding in GPU with applications to Voronoi diagram and distance transform ». In : *Proceedings of the 2006 symposium on Interactive 3D graphics and games*. ACM, p. 109-116 (cf. p. 33).

Rosenthal, Paul et Lars Linsen (2008). « Image-space point cloud rendering ». In : *Proceedings of Computer Graphics International*, p. 136-143 (cf. p. 34, 36).

Roynard, X., J. E. Deschaud et Francois Goulette (2016). « Fast and Robust Segmentation and Classification for Change Detection in Urban Point Clouds ». In : *ISPRS 2016-XXIII ISPRS Congress* (cf. p. 53).

Rusinkiewicz, S. et M. Levoy (2001). « Efficient variants of the ICP algorithm ». In : *Third International Conference on 3-D Digital Imaging and Modeling, 2001. Proceedings*, p. 145-152 (cf. p. 48).

Rusinkiewicz, Szymon et Marc Levoy (2000). « QSplat : A Multiresolution Point Rendering System for Large Meshes ». In : *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '00. New

York, NY, USA : ACM Press/Addison-Wesley Publishing Co., p. 343-352 (cf. p. 12, 20, 22, 110).

Sainz, Miguel et Renato Pajarola (déc. 2004). « Point-based rendering techniques ». In : *Computers & Graphics* 28.6, p. 869-879 (cf. p. 29).

Schaefer, Scott et Joe Warren (2004). « Dual marching cubes : Primal contouring of dual grids ». In : *Computer Graphics and Applications, 2004. PG 2004. Proceedings. 12th Pacific Conference on*. IEEE, p. 70-76 (cf. p. 102).

Schuetz, Markus (2016). « Potree–Rendering Large Point Clouds in Web Browsers ». Thèse de doct. Technische Universität Wien (cf. p. 12).

Schütz, M. et M. Wimmer (sept. 2015). « High-quality point-based rendering using fast single-pass interpolation ». In : *2015 Digital Heritage*. T. 1, p. 369-372 (cf. p. 28, 29, 56).

Serna, Andrés, Beatriz Marcotegui, François Goulette et Jean-Emmanuel Deschaud (2014). « Paris-rue-Madame database : a 3D mobile laser scanner dataset for benchmarking urban detection, segmentation and classification methods ». In : *4th International Conference on Pattern Recognition, Applications and Methods ICPRAM 2014* (cf. p. 96).

Shen, Chen, James F. O'Brien et Jonathan R. Shewchuk (2005). « Interpolating and approximating implicit surfaces from polygon soup ». In : *ACM Siggraph 2005 Courses*. ACM, p. 204 (cf. p. 103).

Strengert, Magnus, Martin Kraus et Thomas Ertl (2006). « Pyramid methods in gpu-based image processing ». In : *Proceedings Vision, Modeling, and Visualization*. T. 2006, p. 169-176 (cf. p. 33).

Treece, G. M., R. W. Prager et A. H. Gee (1998). « Regularised Marching Tetrahedra : Improved Iso-Surface Extraction ». In : *Computers and Graphics* 23, p. 583-598 (cf. p. 101).

Turk, Greg et Marc Levoy (1994). « Zippered Polygon Meshes from Range Images ». In : *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*. SIGGRAPH '94. New York, NY, USA : ACM, p. 311-318 (cf. p. 94).

Venkatasubramaniam, A. K, M. J Fagan, T Mehta, K. J Mylankal, B Ray, G Kuhan, I. C Chetter et P. T McCollum (août 2004). « A Comparative Study of Aortic Wall Stress Using Finite Element Analysis for Ruptured and Non-ruptured Abdominal Aortic Aneurysms ». In : *European Journal of Vascular and Endovascular Surgery* 28.2, p. 168-176 (cf. p. 91).

Wand, Michael, Alexander Berner, Martin Bokeloh, Philipp Jenke, Arno Fleck, Mark Hoffmann, Benjamin Maier, Dirk Staneker, Andreas Schilling et Hans-Peter Seidel (avr. 2008). « Processing and interactive editing of huge point clouds from 3D scanners ». In : *Computers & Graphics* 32.2, p. 204-220 (cf. p. 23).

Wimmer, Michael et Claus Scheiblauer (2006). « Instant Points : Fast Rendering of Unprocessed Point Clouds. » In : *SPBG*. Citeseer, p. 129-136 (cf. p. 12, 22-24, 40, 41, 45, 48).

Xu, Hui, Minh X. Nguyen, Xiaoru Yuan et Baoquan Chen (2004). « Interactive silhouette rendering for point-based models ». In : *Proceedings of the First Eurographics conference on Point-Based Graphics*. Eurographics Association, p. 13-18 (cf. p. 28).

Zhao, Jia-Xin, Thierry Coupez, Etienne Decencière, Dominique Jeulin, David Cárdenas-Peña et Luisa Silva (sept. 2016). « Direct multiphase mesh generation from 3D images using anisotropic mesh adaptation and a redistancing equation ». In : *Computer Methods in Applied Mechanics and Engineering* 309, p. 288-306 (cf. p. 94).

Annexes

Annexe A

Rendu de nuages de points en vue perspective

Dans cette partie nous détaillons certains résultats liés au rendu en perspective projective. Nous présentons tout d'abord les calculs liés à la projection d'un point appliqués à la première étape du rendu de nuages de points que l'on qualifie de *"passe géométrique"*. Nous présentons ensuite la projection d'objets non ponctuels.

A.1 Projection et dé-projection d'un point et pipeline graphique

A.1.1 Projection

Étant donnée une configuration de caméra, définir l'opérateur qui permet de passer d'un point 3D à ses coordonnées dans l'espace image.

Soit $p = (x, y, z) \in \mathbb{R}^3$ un point 3D. On note $\widehat{p} = (x, y, z, w)^t \in \mathbb{R}^4$, avec $w = 1$, le vecteur associé en coordonnées homogènes. La première étape consiste à transformer $\widehat{p}$ par une transformation affine pour l'exprimer dans le repère de la caméra $\mathcal{R}_c$. On note alors $\widehat{p}_c = (x_c, y_c, z_c, w_c)$ ses coordonnées dans ce nouveau repère. La convention retenue pour l'orientation de ce repère est celle d'OpenGL, illustrée sur la figure A.1 ; l'axe z est orienté vers l'arrière x vers la droite et y vers le haut.

On note $\mathcal{T}_c : \mathbb{R}^3 \to \mathbb{R}^3$ la transformation affine associée au repère $\mathcal{R}_c$. Elle est linéaire sur l'espace affine des coordonnées homogènes. Elle est décrite par la matrice [1] $T_c = \begin{pmatrix} R_c & t_c \\ 0 & 1 \end{pmatrix}$, où R_c et t_c sont respectivement la matrice de rotation et le vecteur de translation du repère $\mathcal{R}_c$.

1. La matrice T_c^{-1} est traditionnellement appelé *"view matrix"*

$$p_c = \mathcal{T}_c^{-1}(p) \tag{A.1}$$
$$\widehat{p}_c = T_c^{-1}\widehat{p} \tag{A.2}$$

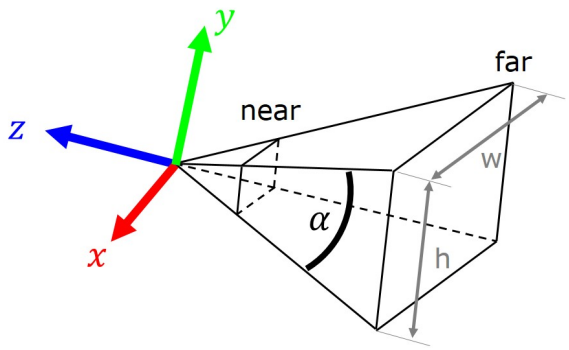


FIGURE A.1 – Convention d'orientation du repère caméra OpenGL. L'angle du champ de vision selon l'axe y est noté α. *near* et *far* désignent les plans par lesquels sont tronqués le cône de la caméra.

On définit alors Π l'opérateur de projection qui permet de passer d'un point 3D dans le repère de la caméra à ses coordonnées en espace image.

$$\Pi : p_c \in \mathbb{R}^3 \rightarrow p_v \in \mathbb{R}^3 \tag{A.3}$$

La transformation d'un point 3D dans le repère monde par cet opérateur est alors donnée par : $p_v = \Pi(\mathcal{T}_c^{-1}(p))$.

La deuxième étape consiste à transformer $\widehat{p}_c$ dans le repère projectif de la caméra $\mathcal{R}_p$. On notera ses coordonnées dans ce nouveau repère $\widehat{p}_p = (x_p, y_p, z_p, w_p)$. Pour ce faire il est nécessaire de définir la matrice de projection P_c associée à la caméra. Comme présenté sur la figure A.1 on note α l'angle du champ de vision [2] selon l'axe y. On introduit également par commodité l'angle $\theta = \frac{\alpha}{2}$. On notera $a_c = \frac{w}{h}$ le ratio entre la largeur et la hauteur du tronc de la caméra, et n_c et f_c la profondeur (en valeur absolue) selon l'axe z des plans *near* et *far* illustrés sur la figure A.1.

$$\widehat{p}_p = P_c \tilde{p}_c \tag{A.4}$$
$$P = \begin{pmatrix} \frac{1}{a_c \tan\theta} & 0 & 0 & 0 \\ 0 & \frac{1}{\tan\theta} & 0 & 0 \\ 0 & 0 & -\frac{f_c+n_c}{f_c-n_c} & -\frac{2f_c n_c}{f_c-n_c} \\ 0 & 0 & -1 & 0 \end{pmatrix} \tag{A.5}$$

2. C'est cet angle que l'on désigne par l'acronyme *FOVY* pour *Field Of View* dans la direction de l'axe y.

A ce stade, on pourrait tester si $\widehat{p}_p$ est compris dans le tronc de la caméra en comparant si les trois premières coordonnées sont comprises dans l'intervalle $[-w_p, w_p]$. C'est pourquoi ces coordonnées sont également appelées *clip coordinates*.

L'étape suivante consiste alors à passer $\widehat{p}_p$ en coordonnées normalisées[3] par une division par w_p, le résultat est alors noté $\widehat{p}_n = (x_n, y_n, z_n, w_n)$:

$$\widehat{p}_n = \frac{1}{w_p}\widehat{p}_p \tag{A.6}$$

La combinaison de ces deux étapes (repère projectif, puis repère normalisé) peut être vue comme une transformation du tronc de la caméra en cube centré en O et d'arête 2, ce qui permet facilement de supprimer les points hors champ en testant si leurs coordonnées sont comprises entre -1 et 1.

On notera que l'on a divisé par w_p qui est égal à $-z_c$. Par ailleurs la valeur z_n obtenue dépend non linéairement de z_c et est comprise entre -1 et 1 :

$$z_n = \frac{f_c + n_c + \frac{2 f_c n_c}{z_c}}{f_c - n_c} \tag{A.7}$$

z_n est la valeur classique de la profondeur utilisée en rendu car elle permet par un simple test de supprimer les points qui ne sont pas compris entre les plans *near* et *far*. Par ailleurs, la dépendance en $\frac{1}{z_c}$ permet de tenir compte de la perspective lorsque l'on interpole linéairement les valeurs de profondeur lors de la rastérisation d'un triangle.

Cette dépendance non linéaire pose néanmoins des problèmes de précision du tampon de profondeur car sa résolution n'est pas uniforme sur l'intervalle $z_c \in [-n_c, -f_c]$. Afin de s'affranchir de ces problèmes nous remplaçons cette valeur par une profondeur linéaire $z_l = -\frac{z_c}{f_c}$, ce qui s'écrit dans le *fragment shader* :

```
gl_FragDepth = -p_c.z / f_c;
```

On peut alors exprimer les coordonnées de p en espace image $p_v = (x_v, y_v, z_v) \in \mathbb{R}^3$. Les deux premières coordonnées sont exprimées en pixels et correspondent à la position du point sur la texture de rendu. La troisième coordonnée représente la profondeur linéaire introduite précédemment. On note (w, h) la résolution de la texture de rendu en pixels. On obtient alors :

$$\begin{aligned} x_v &= \frac{1}{2}(x_n + 1)w \\ y_v &= \frac{1}{2}(y_n + 1)h \\ z_v &= z_l \end{aligned} \tag{A.8}$$

3. Ou NDC pour *Normalized Device Coordinates*

Pour déterminer si le point apparait à l'écran, il est nécessaire de comparer sa profondeur z_l à la profondeur déjà présente sur le tampon de profondeur $z(\lfloor x_v \rfloor, \lfloor y_v \rfloor)$. Le point est alors rendu si sa profondeur est inférieure à cette valeur. Cette comparaison est réalisée par l'API graphique.

Le pipeline décrit ci-dessus est illustré sur un cas simple sur la figure A.2. Il est hautement parallélisable étant donné que chaque point peut être traité indépendamment [4] c'est pourquoi il est parfaitement adapté à une implémentation GPU.

Toutefois de récents travaux (Günther et al., 2013) ont noté que l'utilisation du pipeline classique n'est pas optimale lorsqu'il est appliqué au rendu de points. Cela est lié à la sous-utilisation des *rasterisation units* dans ce cas. Ainsi, l'utilisation d'un pipeline GPGPU (ou *pipeline compute*) permettrait d'augmenter la fréquence d'affichage d'un ordre de grandeur.

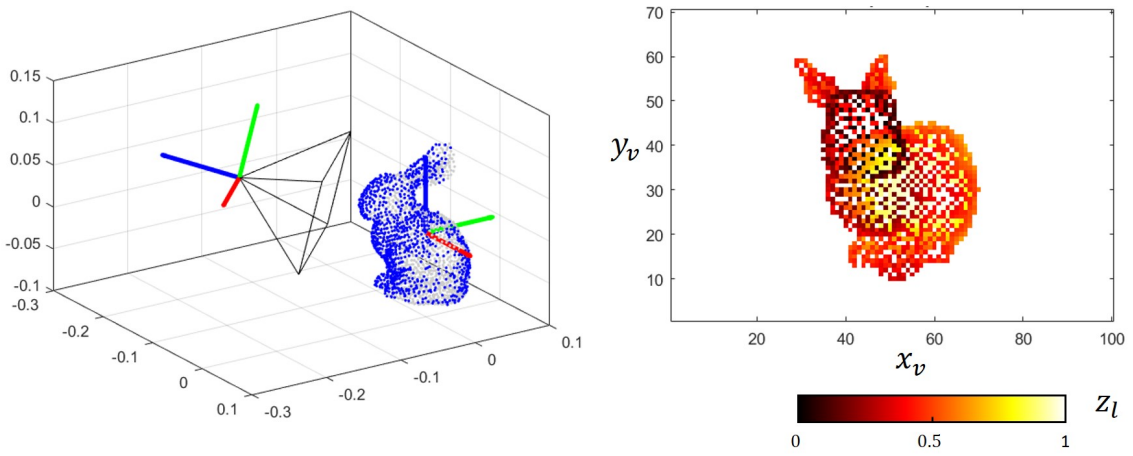


FIGURE A.2 – Exemple de rendu de nuage de points par projection perspective. (A gauche) nuage de points et cône de la caméra virtuelle ($\alpha = 60^\circ, a_c = \frac{10}{7}, w = 100, h = 70$). En bleu sont représentés les points qui servent au rendu, en gris ceux qui n'ont pas passé le test de profondeur. (A droite) la texture de profondeur résultante. Les pixels blancs correspondent à l'arrière-plan.

A.1.2 Dé-projection

Certains algorithmes décrits dans le chapitre 1 nécessitent de remonter aux coordonnées 3D à partir d'un point dans l'espace image. On note alors Π^{-1}, l'opérateur de dé-projection qui permet à partir d'un point dans l'espace image p_v d'obtenir ses coordonnées dans le repère 3D de la caméra p_c. Pour cela il suffit d'inverser les équations établies dans le paragraphe précédent :

4. À l'exception de l'étape de comparaison de profondeur qui nécessite d'avoir recours à une opération atomique au cas où deux fragment essayent d'accéder au même pixel.

$$r = \begin{pmatrix} a_c \tan(\theta)(\frac{2}{w}x_v - 1) \\ \tan(\theta)(\frac{2}{h}y_v - 1) \\ -1 \end{pmatrix} \tag{A.9}$$

$$\Pi^{-1}(p_v) = p_c = f_c z_v r \tag{A.10}$$

Le vecteur r peut être interprété comme le rayon qui part de la caméra et qui passe par le pixel de coordonnées (x_v, y_v). Les terme $a_c \tan(\theta)$ et $\tan(\theta)$ proviennent des termes diagonaux de la matrice de projection P_c. Ils correspondent respectivement à la demi largeur et à la demi hauteur du rectangle que l'on obtiendrait par une coupe du cône de la caméra par le plan $z_c = -1$.

On notera que lorsque l'on ne dispose que des coordonnées entières de pixels (r, c), on prendra par défaut le centre du pixel $(x_v, y_v) = (r + 0.5, c + 0.5)$ comme référence.

A.2 Projection de deux points et calcul de LOD

Dans cette section on s'intéresse à la projection en vue perspective de deux points séparés d'une distance r, schématisée en 2D sur la figure A.3. On notera D_Π la distance entre ces deux points projetée à l'écran.

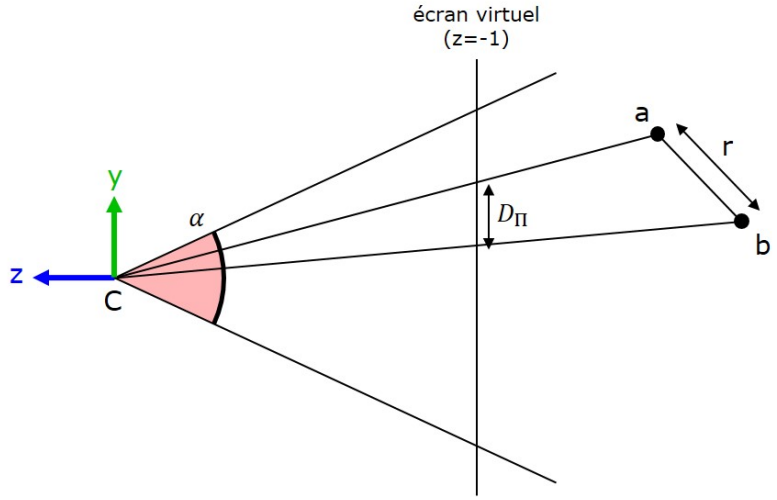


FIGURE A.3 – Projection en 2D de deux points séparés d'une distance r en vue perspective.

En première approximation, on peut supposer que les deux points se trouvent sur le même plan transversal par rapport à la caméra. Dans ce cas on peut montrer que la distance projetée entre les deux points est indépendante de leur position sur ce plan transversal. On note leurs coordonnées dans le repère de la caméra :

$$\begin{aligned} a &= \begin{pmatrix} x_a & y_a & -z \end{pmatrix}^t \\ b &= \begin{pmatrix} x_b & y_b & -z \end{pmatrix}^t \end{aligned} \tag{A.11}$$

Après projection dans l'espace image, on obtient :

$$\begin{aligned} a_v = \Pi(a) &= \begin{pmatrix} \frac{w}{2}(\frac{x_a}{a_c \tan(\theta) z} + 1) & \frac{h}{2}(\frac{y_a}{\tan(\theta) z} + 1) & -\frac{z}{f_n} \end{pmatrix}^t \\ b_v = \Pi(b) &= \begin{pmatrix} \frac{w}{2}(\frac{x_b}{a_c \tan(\theta) z} + 1) & \frac{h}{2}(\frac{y_b}{\tan(\theta) z} + 1) & -\frac{z}{f_n} \end{pmatrix}^t \end{aligned} \tag{A.12}$$

La distance projetée entre les deux points est alors donnée par :

$$D_\Pi = \sqrt{\left(\frac{w(x_b - x_a)}{2a_c \tan(\theta) z}\right)^2 + \left(\frac{h(y_b - y_a)}{2 \tan(\theta) z}\right)^2} \tag{A.13}$$

Or $\frac{w}{a_c} = h$, ainsi :

$$\begin{aligned} D_\Pi &= \frac{h}{2 \tan(\theta) z} \sqrt{(x_b - x_a)^2 + (y_b - y_a)^2} \\ D_\Pi &= \frac{hr}{2 \tan(\theta) z} \end{aligned} \tag{A.14}$$

Comme illustré sur la figure A.4, dans le cas d'un octree, à un niveau l donné exprimé depuis la racine de l'arbre (le niveau $l = 0$ correspond à la racine) la distance entre deux points voisins r_l est majorée par $2\sqrt{3}s_0 2^{-l}$. Où s_0 est la dimension du cube englobant la racine de l'arbre.

$$r_l \leqslant \frac{2\sqrt{3}s_0}{2^l} \tag{A.15}$$

Ainsi, en combinant les équations A.14 et A.15, on obtient l'inéquation suivante sur la projection de deux points voisins à un niveau l.

$$D_\Pi \leqslant \frac{\sqrt{3}hs_0}{\tan(\theta) z 2^l} \tag{A.16}$$

De ce fait, si on introduit un seuil t en pixels en dessous duquel doit être la distance projetée entre deux points voisins, il suffit d'avoir :

$$\frac{\sqrt{3}hs_0}{\tan(\theta) z 2^l} \leqslant t \tag{A.17}$$

Ce qui donne :

$$l \geqslant \frac{\log(\sqrt{3}hs_0) - \log(t \tan(\theta) z)}{\log(2)} \tag{A.18}$$

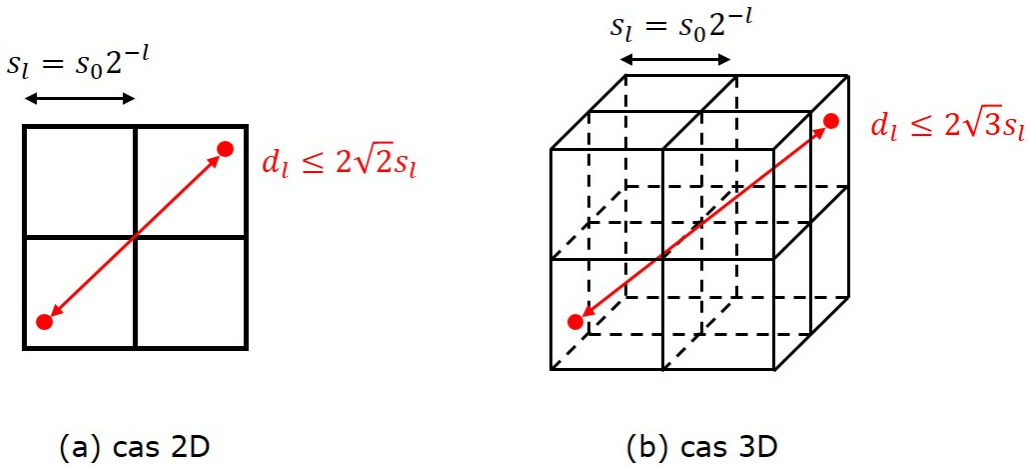


FIGURE A.4 – Distance maximale entre deux points voisins dans une grille (a) 2D ou 3D (b). Dans le cas d'une structure hiérarchique de type quadtree ou octree, la taille de chaque case de la grille est $s_l = s_0 2^{-l}$

Annexe B

Maillage par optimisation locale

Le maillage par optimisation locale développé par (Thierry Coupez, 2000) est une alternative aux méthodes classiques de maillage en mécanique numérique. Son intérêt réside dans le fait qu'elle présente un cadre théorique unifié pour la création et la modification de maillage.

Le lecteur pourra se référer à (COUPEZ, 1991 ; Thierry Coupez, 2000 ; T. Coupez et al., 2000) pour les preuves de théorèmes énoncés dans ce paragraphe.

B.1 Notations

Soit $\mathcal{N} \subset \mathbb{N}$ un ensemble de nœuds représentés par leurs indices.

Soit $\mathcal{P}_D(\mathcal{N})$ l'ensemble des parties de $\mathcal{N}$ composées de D éléments distincts. En dimension d, P_{d+1} est l'ensemble des simplexes constructibles avec les nœuds de $\mathcal{N}$.

Pour un ensemble de simplexes $\mathcal{T} \subset \mathcal{P}_D(\mathcal{N})$, on note l'ensemble de ses nœuds $\mathcal{N}(\mathcal{T}) = \bigcup_{T \in \mathcal{T}} T$.

Pour un simplexe $T \in \mathcal{P}_D(\mathcal{N})$, on note $\partial T = \mathcal{P}_{D-1}(T)$ l'ensemble des faces de T.

On peut alors noter $\mathcal{F}(\mathcal{T}) = \bigcup_{T \in \mathcal{T}} \partial T$ l'ensemble des faces de $\mathcal{T}$.

Soit $\eta \subset \mathcal{N}$ un ensemble de nœuds (typiquement : un nœud, une arête ou une face). On note $\mathcal{T}(\eta) = \{T \in \mathcal{T}, \eta \subset T\}$, l'ensemble des éléments de $\mathcal{T}$ qui possèdent le nœud η en commun.

B.2 Topologie de maillage

Nous pouvons maintenant définir le concept de topologie de maillage. Il s'agit d'une notation uniquement définie sur l'ensemble $\mathcal{N}$ et qui ne prend donc pas en compte la position des nœuds dans l'espace.

$\mathcal{T} \subset \mathcal{P}_D(\mathcal{N})$ est une topologie de maillage si ces éléments ne partagent qu'au maximum 2 faces, ce qui peut se noter :

$$\forall F \in \mathcal{F}(\mathcal{T}), card(\mathcal{T}(F)) \leq 2 \tag{B.1}$$

On note alors $\partial\mathcal{T} = \{F \in \mathcal{F}(\mathcal{T}), card(\mathcal{T}(F)) = 1\}$ sa frontière.

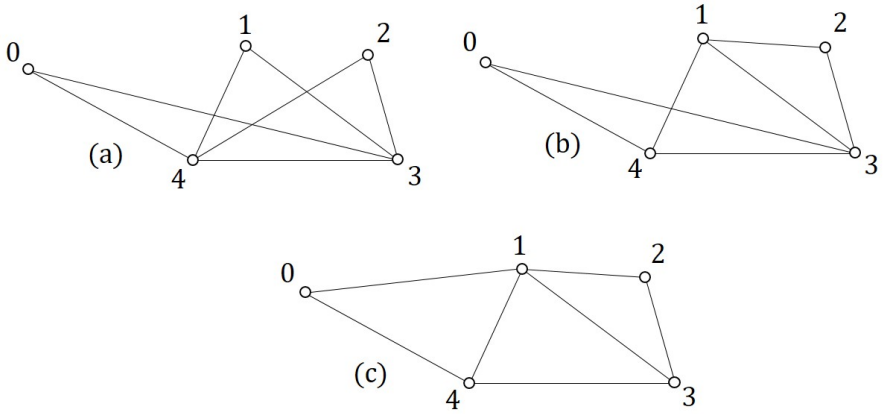


FIGURE B.1 – Ensemble de triangles $\mathcal{T}$ en 2D formés à partir d'un même ensemble de nœuds $\mathcal{N} = \{0, 1, 2, 3, 4\}$. (a) $\mathcal{T}$ n'est pas une topologie de maillage : les trois triangles partagent la même arête $\{4, 3\}$. (b) $\mathcal{T}$ est une topologie de maillage non conforme car deux triangles s'intersectent. (c) $\mathcal{T}$ est une topologie de maillage conforme.

B.3 Théorème de volume minimal

Si on définit l'application $X : \mathcal{N} \to \mathbb{R}^d$ qui permet d'obtenir les coordonnées d'un nœud, on peut établir un lien entre une topologie de maillage et le maillage associé par cette application.

On peut commencer par constater qu'un couple $\{\mathcal{T}, X\}$, n'engendre pas systématiquement un maillage conforme. Un maillage est conforme lorsqu'il ne contient pas d'éléments plats, qui se superposent ou qui sont inversés. La figure B.1 illustre ce constat.

Soit Ω un domaine à frontière polyédrique, et soit $\mathcal{T}$ une topologie de maillage telle que $\{\partial\mathcal{T}, X\}$ engendre une triangulation conforme du bord de Ω, $\partial\Omega$. Alors :

a $\sum_{T\in\mathcal{T}} |\Omega_T| \geq |\Omega|$

b $\{\mathcal{T}, X\}$ engendre un maillage de Ω ssi :
$\sum_{T\in\mathcal{T}} |\Omega_T| = |\Omega|$ et
$\forall T \in \mathcal{T}, |\Omega_T| > 0$

Où :

- Pour $E \subset \mathbb{R}^d$, $|E|$ est le volume de E
- $\Omega(\mathcal{T}, X)$ et $\Omega(T, X)$ noté Ω_T désignent respectivement le domaine occupé par $\mathcal{T}$ et le domaine occupé par un élément T.

Avec ce théorème on peut donc associer la création du maillage d'un domaine à partir de son bord à une minimisation de volume sur l'ensemble fini des topologies de maillages possibles pour ce domaine.

B.4 Opérateur d'étoilement

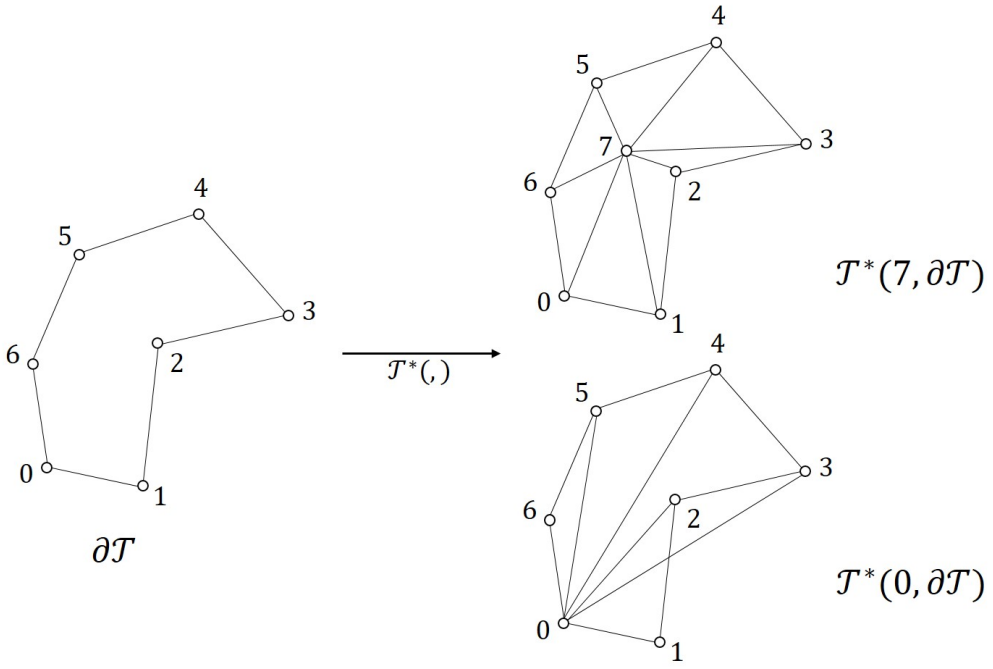


FIGURE B.2 – Opérateur d'étoilement appliqué sur une frontière $\partial\mathcal{T}$ composée de 7 arêtes. Au-dessus l'opérateur est appliqué au nœud 7 qui n'appartient pas à la frontière : on crée ainsi 7 triangles. Au-dessous il est appliqué au nœud 0 qui fait partie de $\partial\mathcal{T}$ on ne crée donc que 5 triangles.

On définit maintenant un opérateur de création de topologie de maillage à partir d'un nœud et d'un ensemble de faces. Cet opérateur, qualifié d'*opérateur d'étoilement* va nous permettre par la suite de définir les deux opérations élémentaires nécessaires pour décrire l'algorithme de maillage.

Cet opérateur consiste simplement à créer des éléments en reliant toutes les faces d'un bord $\partial\mathcal{T}$ à un nœud n donné. Il s'agit d'un opérateur purement topologique

puisqu'il ne fait pas intervenir la position des nœuds dans l'espace. Il peut être décrit de la manière suivante :

$$\mathcal{T}^*(n, \partial\mathcal{T}) = \{T = \{n\} \cup F | F \in \partial\mathcal{T}, n \notin F\} \tag{B.2}$$

Le fonctionnement de cet opérateur est illustré en 2D figure B.2.

B.5 Opérateur de fermeture

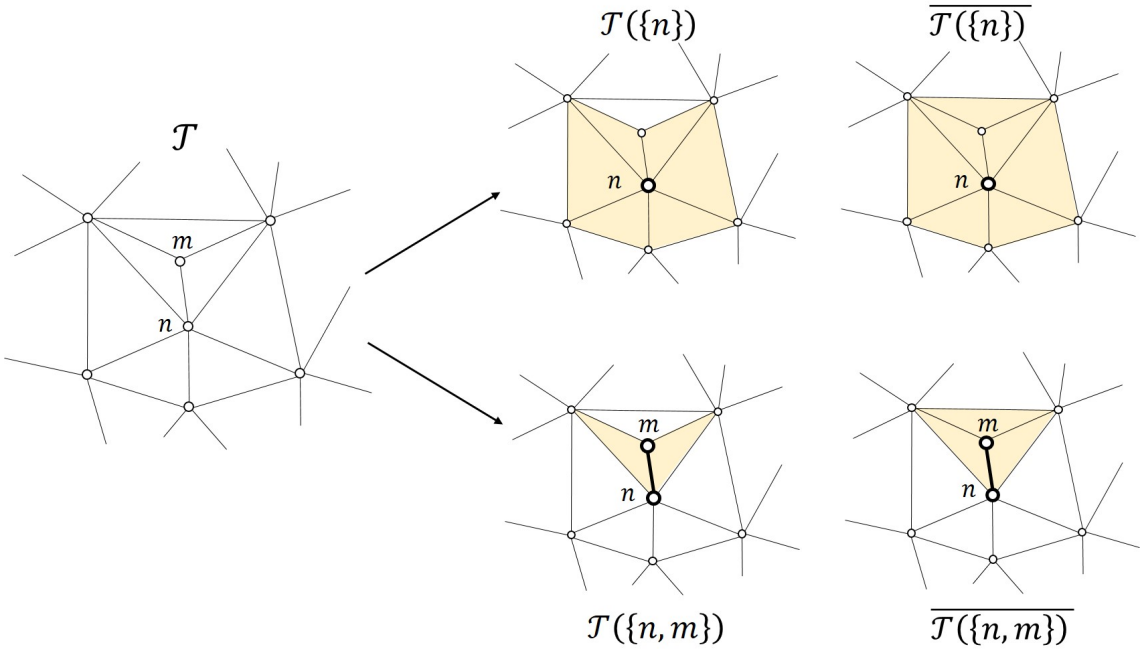


FIGURE B.3 – Voisinage fermé d'un nœud intérieur $\{n\}$ ou d'une arête intérieure $\{n, m\}$ d'une topologie de maillage $\mathcal{T}$.

Soit η un ensemble de nœuds, typiquement un nœud unique $\{n\}$ ou une arête $\{m, n\}$. $a = \mathcal{T}(\eta)$ est l'ensemble des éléments de $\mathcal{T}$ qui partagent les nœuds de η.

On note alors $\overline{a}$ la fermeture de a dans $\mathcal{T}$. Il s'agit de l'ensemble des éléments de $\mathcal{T}$ qui partagent les nœuds de a (l'ensemble des nœuds de a étant noté $\mathcal{N}(a)$) .

$$\overline{a} = \{T \in \mathcal{T}, T \subset \mathcal{N}(a)\} \tag{B.3}$$

Le fonctionnement de cet opérateur est illustré en 2D figure B.3.

B.6 Opérations élémentaires

Les deux opérateurs décrits précédemment permettent de décrire deux opérations élémentaires d'*extraction-remplacement*. Étant donné une topologie de maillage $\mathcal{T}$ ces opérations consistent à retirer un sous-ensemble $a \subset \mathcal{T}$ et de le remplacer localement par une nouvelle topologie de maillage b obtenue à l'aide de l'opérateur d'étoilement appliqué au bord ∂a de a :

$$\mathcal{T} \leftarrow \mathcal{T} - a + b \tag{B.4}$$

Si on note η un nœud $\{n\}$ ou une arête $\{n, m\}$ on peut alors décrire deux types d'opérations : les *opérations nodales* et les *opérations sur les arêtes* :

$$\theta_\eta^p(\mathcal{T}) = \mathcal{T} - \overline{\mathcal{T}(\eta)} + \mathcal{T}^*(p, \partial\overline{\mathcal{T}(\eta)}) \tag{B.5}$$

B.7 Algorithme

On suppose qu'on dispose en entrée d'une topologie de maillage initiale $\mathcal{T}_0$. On peut noter que si on dispose uniquement d'un bord $\partial\mathcal{T}_0$, on peut obtenir $\mathcal{T}_0$ à l'aide de l'opérateur d'étoilement en choisissant arbitrairement un nœud n_0 : $\mathcal{T}_0 = \mathcal{T}^*(\partial\mathcal{T}_0, n_0)$.

L'algorithme de maillage consiste alors à itérer successivement les opérations nodales et les opérations sur les arêtes. L'enjeu est alors de trouver à chaque étape les nœuds p^k pour *améliorer* le maillage à chaque itération de $\theta_\eta^{p^k}(\mathcal{T}^k)$. Le caractère itératif de l'algorithme est illustré sur la figure B.4.

Localement, étant donné un nœud $\{n\}$ ou un segment $\{n, m\}$ qu'on notera génériquement η. On va chercher parmi les nœuds de son voisinage fermé $\overline{\mathcal{T}(\eta)}$ lesquels minimisent le volume du maillage formé après l'opération élémentaire $\theta_\eta^p(\mathcal{T})$. Parmi ceux-ci nous ne retiendrons que le meilleur vis-à-vis d'une relation d'ordre que nous aurons définie au préalable.

Cette relation d'ordre peut être définie de différentes manières selon le maillage qu'on désire obtenir, par exemple avec un facteur de forme pour chaque triangle :

$$c(T) = \frac{|\Omega_T|}{h(T)^d} \tag{B.6}$$

$$h(T) = \sum_{i,j \in T} \|X(i) - X(j)\| \tag{B.7}$$

Ce facteur de forme permet d'ordonner les éléments d'une topologie de maillage. Il définit donc un critère pour comparer deux topologies. Ici le critère privilégie la création d'éléments réguliers. Cependant, il est possible de définir d'autres critères de comparaison des éléments en spécifiant par exemple la taille des arêtes souhaitée. Ceci peut également être spécifié de manière non homogène sur l'ensemble du

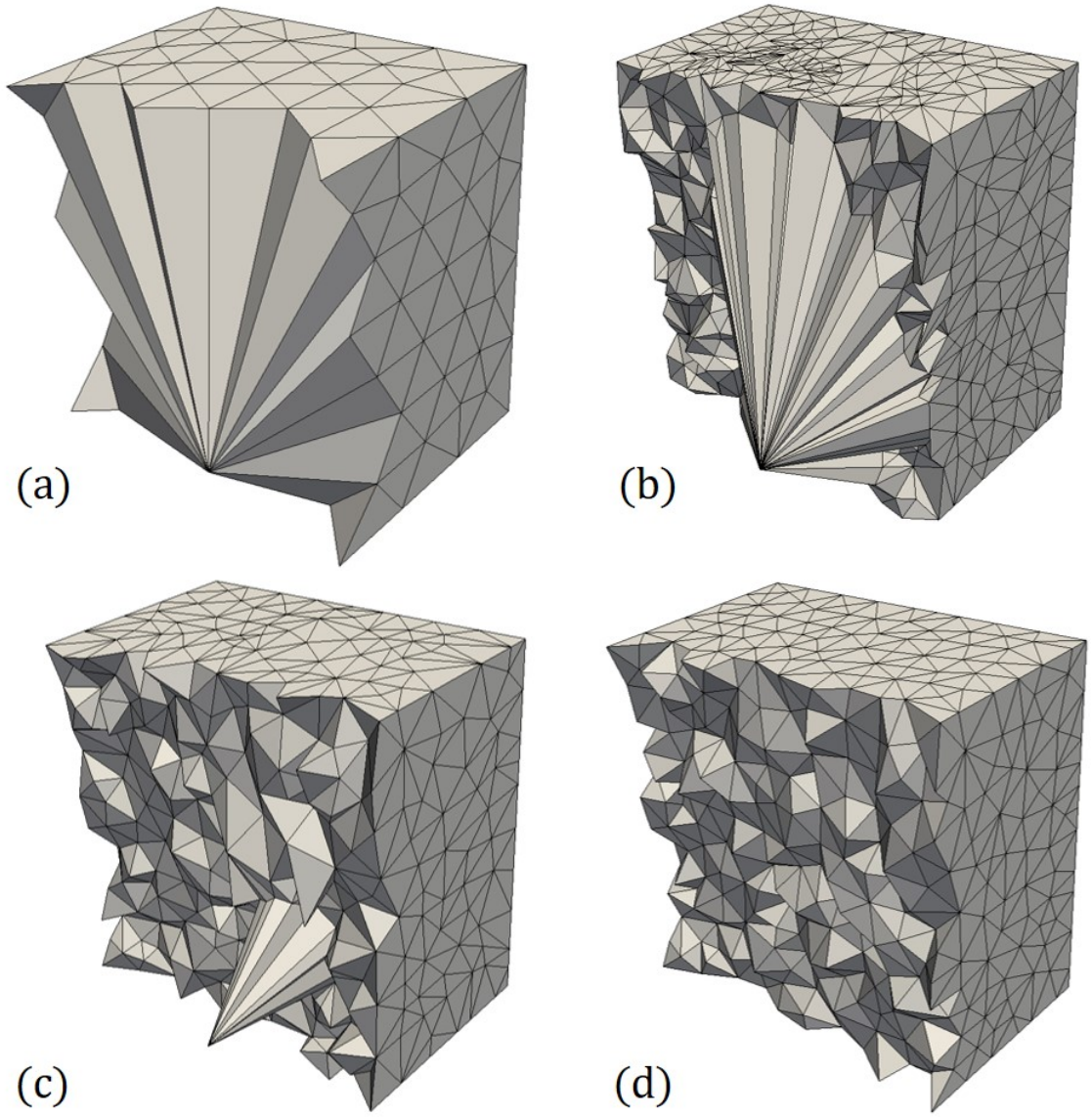


FIGURE B.4 – Vues en coupe d'un cube au cours de son maillage volumique par optimisation locale. (a) On part d'une discrétisation de la surface du cube à partir de laquelle on génère une première topologie de maillage en connectant toutes les faces à un nœud arbitraire. On remarquera que dans cette première configuration, certains tétraèdres sont aplatis voire retournés, ce que gère parfaitement cette méthode.

maillage mais également de manière à privilégier l'anisotropie des éléments dans une direction choisie.

B.8 Résultats

Un des intérêts de cet algorithme est qu'il s'applique aussi bien en 2D qu'en 3D. Les figures B.5 et B.6 montrent la robustesse de l'algorithme en 2D vis à vis de conditions initiales complexes (de nombreux triangles sont retournés ou d'aire nulle). Le maillage initial étant calculé en reliant toutes les arrêtes du bord à un nœud choisi arbitrairement. Les figures B.4 et B.7 montrent les résultats en 3D.

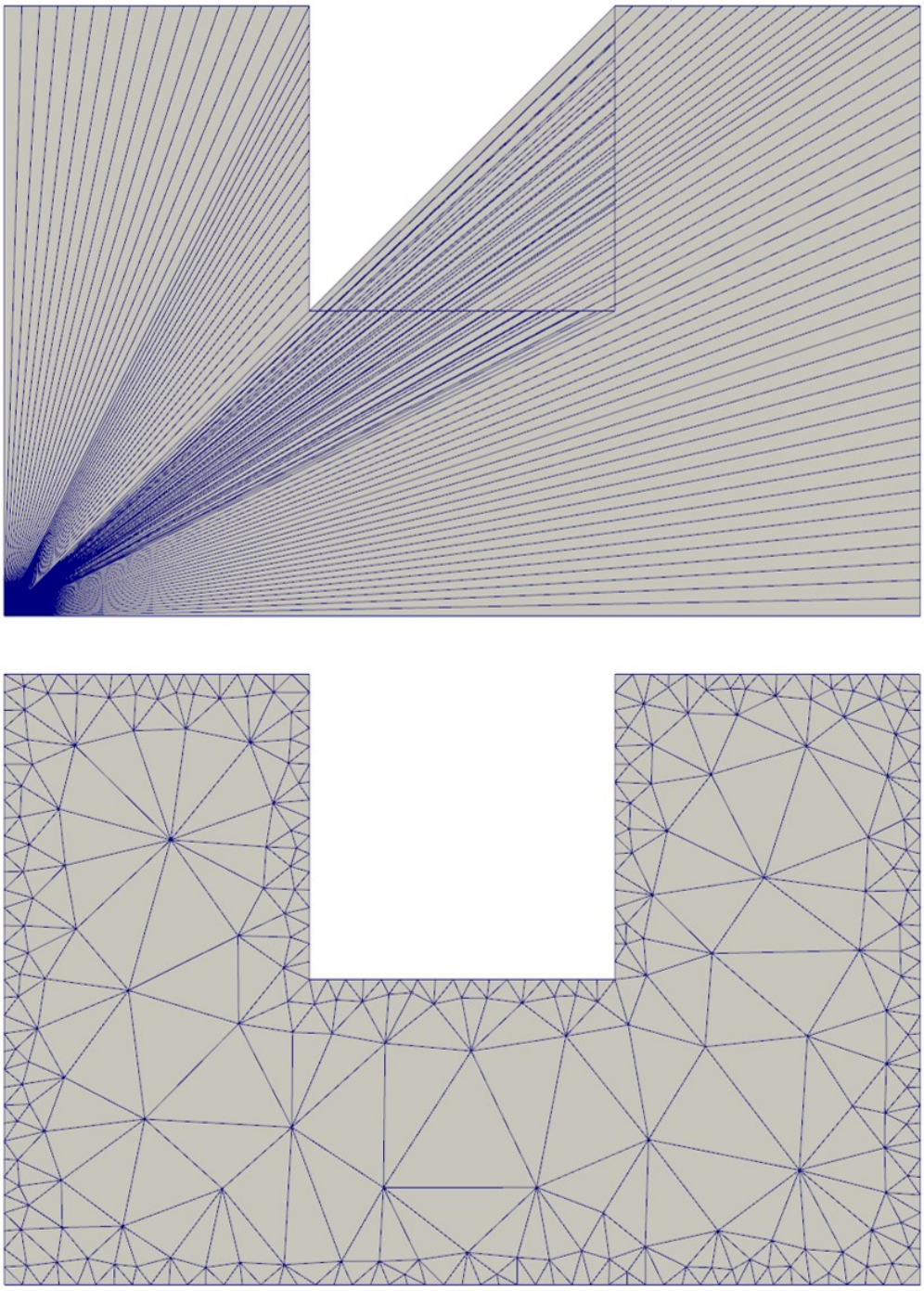


FIGURE B.5 – Résultats obtenus à partir d'un contour 2D non convexe qui comporte 203 nœuds. (En haut) état initial. (En bas) après 200 itérations le maillage comporte 571 triangles.

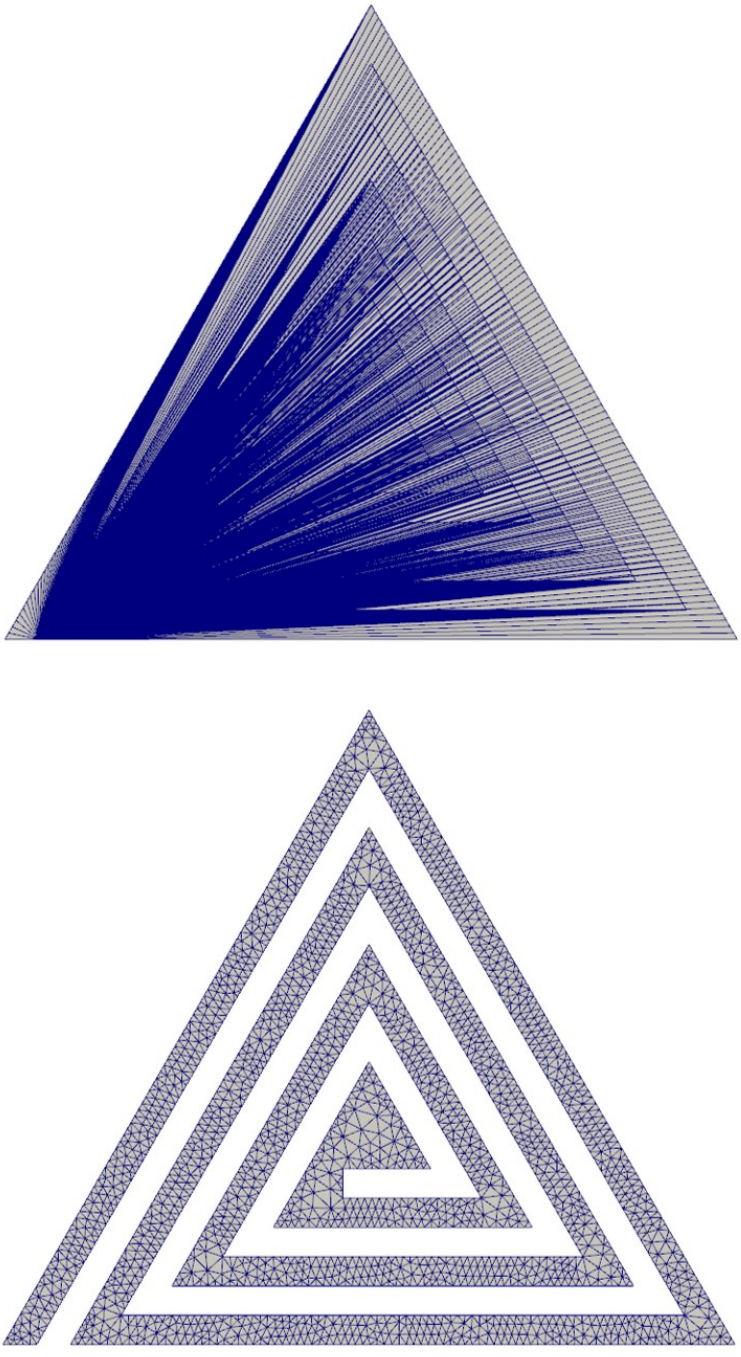


FIGURE B.6 – Résultats obtenus à partir d'un contour 2D en forme de spirale qui comporte 1232 nœuds. (En haut) état initial. (En bas) après 550 itérations le maillage comporte 3582 triangles.

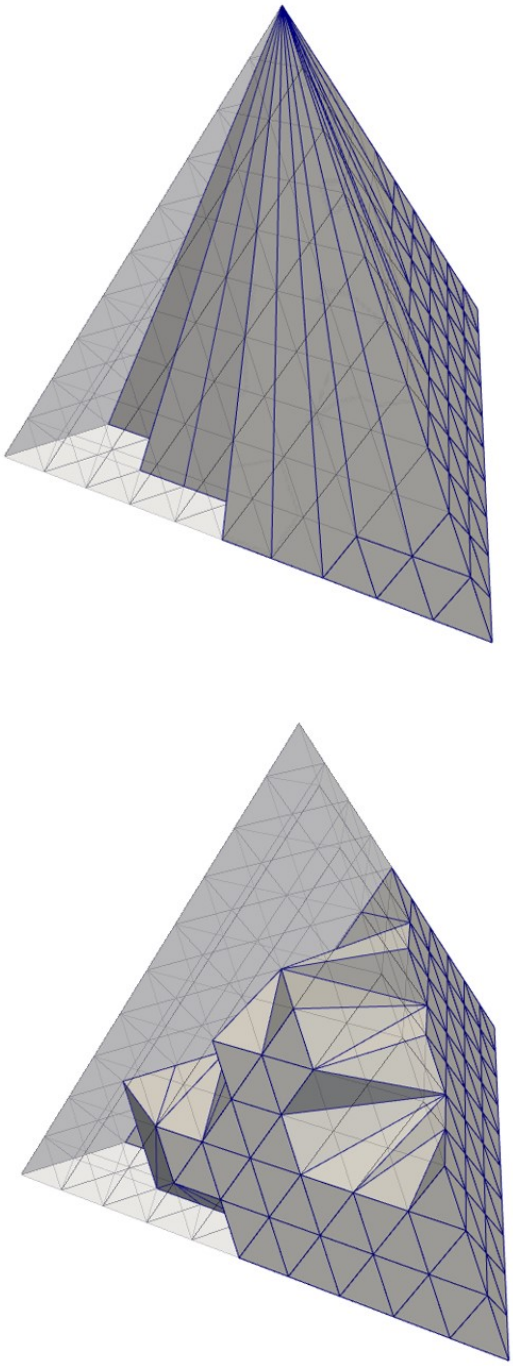


FIGURE B.7 – Résultats obtenus à partir d'un contour 3D de tétraèdre forme de spirale qui comporte 400 nœuds. (En haut) état initial. (En bas) après 380 itérations le maillage comporte 324 tétraèdres. La figure représente une coupe des cellules internes du maillage. L'enveloppe extérieure est affichée en transparence. La création de nœuds internes a été désactivée pour cet exemple.

Résumé

Le nuage de points 3D est la donnée obtenue par la majorité des méthodes de numérisation surfacique actuelles. Nous nous intéressons ainsi dans cette thèse à l'utilisation de nuages de points comme unique représentation explicite de surface. Cette thèse présente deux contributions en traitements basés points.

La première contribution proposée est une nouvelle méthode de rendu de nuages de points bruts et massifs par opérateurs pyramidaux en espace image. Cette nouvelle méthode s'applique aussi bien à des nuages de points d'objets scannés, que de scènes complexes. La succession d'opérateurs en espace image permet alors de reconstruire en temps réel une surface et d'en estimer des normales, ce qui permet par la suite d'en obtenir un rendu par ombrage. De plus, l'utilisation d'opérateurs pyramidaux en espace image permet d'atteindre des fréquences d'affichage plus élevées d'un ordre de grandeur que l'état de l'art.

La deuxième contribution présentée est une nouvelle méthode de simulation numérique en mécanique des fluides en volumes immergés par reconstruction implicite étendue. La méthode proposée se base sur une nouvelle définition de surface implicite par moindres carrés glissants étendue à partir d'un nuage de points. Cette surface est alors utilisée pour définir les conditions aux limites d'un solveur Navier-Stokes par éléments finis en volumes immergés, qui est utilisé pour simuler un écoulement fluide autour de l'objet représenté par le nuage de points. Le solveur est interfacé à un mailleur adaptatif anisotrope qui permet de capturer simultanément la géométrie du nuage de points et l'écoulement à chaque pas de temps de la simulation.

Mots Clés

Nuage de points, Rendu temps-réel, Espace-image, Méthodes pyramidales, Moindres carrés glissants, Adaptation de maillage anisotrope, Mécanique des fluides numérique, Simulation par volumes immergés

Abstract

Most surface 3D scanning techniques produce 3D point clouds. This thesis tackles the problem of using points as only explicit surface representation. It presents two contributions in point-based processing.

The first contribution is a new raw and massive point cloud screen-space rendering algorithm. This new method can be applied to a wide variety of data from small objects to complex scenes. A sequence of screen-space pyramidal operators is used to reconstruct in real-time a surface and estimate its normals, which are later used to perform deferred shading. In addition, the use of pyramidal operators allows to achieve framerate one order of magnitude higher than state of the art methods.

The second proposed contribution is a new immersed boundary computational fluid dynamics method by extended implicit surface reconstruction. The proposed method is based on a new implicit surface definition from a point cloud by extended moving least squares. This surface is then used to define the boundary conditions of a finite-elements immersed boundary transient Navier-Stokes solver, which is used to compute flows around the object sampled by the point cloud. The solver is interfaced with an anisotropic and adaptive meshing algorithm which refines the computational grid around both the geometry defined by point cloud and the flow at each timestep of the simulation.

Keywords

Point clouds, Real-time rendering, Screen space, Pyramidal image processing, Moving Least Squares, Anisotropic mesh adaptation, Computational fluid dynamics, Immersed boundary simulation