Études et conception d'algorithmes de reconstruction 3D sur tablettes : génération automatique de modèles 3D éditables de bâtiments existants

Adrien Arnaud

université PARIS-SACLAY

NNT : 2018SACLS512

Thèse de doctorat

UNIVERSITÉ PARIS SUD
Comprendre le monde, construire l'avenir

Étude et Conception d'Algorithmes de Reconstruction 3D sur Tablettes : Génération Automatique de Modèles 3D Éditables de Bâtiments Existants

Thèse de doctorat de l'Université Paris-Saclay
préparée à l'Université Paris-Sud

École doctorale n°580 Sciences et Technologies de l'Information et de la Communication
Spécialité de doctorat : Informatique

Thèse présentée et soutenue à Orsay, le 3 Décembre 2018, par

Mr Adrien Arnaud

Composition du Jury :

Samir Otmane Professeur, Université Evry Val-d 'Essonne (– IBISC)	Président
Vincent Frémont Professeur, École Centrale Nantes (– LS2N-CNRS)	Rapporteur
Guillaume Moreau Professeur, École Centrale Nantes (– AAU-CNRS)	Rapporteur
Sylvie Treuillet Maître de conférences, Université d'Orléans (– PRISME)	Examinateur
Michèle Gouiffès Maître de conférences, Université Paris-Saclay (– LIMSI-CNRS)	Examinateur
Mehdi Ammi Maître de conférences, Université Paris-Saclay (– LIMSI-CNRS)	Directeur de thèse

université PARIS-SACLAY
ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Étude et Conception d'Algorithmes de Reconstruction 3D sur tablettes : Génération Automatique de Modèles 3D Éditables de Bâtiments Existants

Mots clés : Reconstruction 3D, systèmes mobiles, segmentation 3D

Résumé : L'objectif de ces travaux de thèse consiste à mettre en place des solutions algorithmiques permettant de reconstruire un modèle 3D éditable d'un environnement intérieur à l'aide d'une tablette équipée d'un capteur de profondeur.

Ces travaux s'inscrivent dans le contexte de la rénovation d'intérieur. Les normes Européennes poussent à la rénovation énergétique et à la modélisation 3D des bâtiments existants. Des outils professionnels utilisant des capteurs de type LIDAR permettent de reconstruire des nuages de points de très grande qualité, mais sont coûteux et longs à mettre en œuvre. De plus, il est très difficile d'identifier automatiquement les constituants d'un bâtiment pour en exporter un modèle 3D éditable complet.

Dans le cadre de la rénovation d'intérieur, il n'est pas nécessaire de disposer des informations sur l'ensemble du bâtiment, seules les principales dimensions et surfaces sont nécessaires. Nous pouvons alors envisager d'automatiser complètement le processus de modélisation 3D.

La mise sur le marché de capteurs de profondeur intégrables sur tablettes, et l'augmentation des capacités de calcul de ces dernières nous permet d'envisager l'adaptation d'algorithmes de reconstruction 3D classiques à ces supports.

Au cours de ces travaux, nous avons envisagé deux approches de reconstruction 3D différentes. La première approche s'appuie sur des méthodes de l'état de l'art. Elle consiste à générer un maillage 3D d'un environnement intérieur en temps réel sur tablette, puis d'utiliser ce maillage 3D pour identifier la structure globale du bâtiment (murs, portes et fenêtres). La deuxième approche envisagée consiste à générer un modèle 3D éditable en temps réel, sans passer par un maillage intermédiaire. De cette manière beaucoup plus d'informations sont disponibles pour pouvoir détecter les éléments structuraux. Nous avons en effet à chaque instant donné un nuage de points complet ainsi que l'image couleur correspondante. Nous avons dans un premier temps mis en place deux algorithmes de segmentation planaire en temps réel. Puis, nous avons mis en place un algorithme d'analyse de ces plans permettant d'identifier deux plans identiques sur plusieurs prises de vue différentes. Nous sommes alors capables d'identifier les différents murs contenus dans l'environnement capturé, et nous pouvons mettre à jour leurs informations géométriques en temps réel.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France

université PARIS-SACLAY

ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Title : Study and Conception of 3D Reconstruction Algorithms on Tablets: Automatic Generation of 3D Editable Models of Existing Buildings

Keywords : 3D reconstruction, mobile devices, 3D segmentation

Abstract: This thesis works consisted to implement algorithmic solutions to reconstruct an editable 3D model of an indoor environment using a tablet equipped with a depth sensor.

These works are part of the context of interior renovation. European standards push for energy renovation and 3D modeling of existing buildings. Professional tools using LIDAR-type sensors make it possible to reconstruct high-quality point clouds, but are costly and time-consuming to implement. In addition, it is very difficult to automatically identify the constituents of a building to export a complete editable 3D model.

As part of the interior renovation, it is not necessary to have information on the whole building, only the main dimensions and surfaces are necessary. We can then consider completely automating the 3D modeling process.

The recent development of depth sensors that can be integrated on tablets, and the improvement of the tablets computation capabilities allows us to consider the adaptation of classical 3D reconstruction algorithms to these supports.

During these works, we considered two different 3D reconstruction approaches. The first approach is based on state-of-the-art methods. It consists in generating a 3D mesh of an interior environment in real time on a tablet, then using this 3D mesh to identify the overall structure of the building (walls, doors and windows). The second approach envisaged is to generate a 3D editable model in real time, without passing through an intermediate mesh. In this way much more information is available to be able to detect the structural elements. We have in fact at each given time a complete point cloud as well as the corresponding color image. In a first time we have set up two planar segmentation algorithms in real time. Then, we set up an analysis algorithm of these plans to identify two identical planes on different captures. We are then able to identify the different walls contained in the captured environment, and we can update their geometric information in real-time.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France

Remerciements

Ces trois années de thèse ont été l'occasion pour moi d'acquérir des connaissances très précieuses, et je tiens à remercier tout d'abord mes encadrants Mehdi Ammi et Michèle Gouiffès. Merci de m'avoir fait confiance pour travailler sur ce sujet de thèse, et de m'avoir encadré tout au long de ces trois ans. Merci pour votre disponibilité, que ce soit pour des réunions de temps en temps, la relecture d'articles, ou pour échanger des idées, merci à Michèle pour son temps passé à relire ce manuscrit.

Merci à mes rapporteurs, Guillaume Moreau et Vincent Frémont d'avoir pris le temps de lire ce manuscrit à leur tour et de le commenter, et merci également à Sylvie Treuillet et Samir Otmane d'avoir accepté de faire partie du jury.

Ensuite, je tenais à remercier mes collègues et amis qui m'ont accompagné durant ces trois années. Tout d'abord, merci à Gibet et Maxence pour tous ces moments constructifs (ou non) passés ensemble, que ce soit au labo ou en dehors, et ce depuis le début, et une pensée émue pour Julien, Thomas et Iñaki qui nous ont quittés trop tôt... Merci à Sylvain, Maxime, Vincent, Nicolas, et à l'équipe des doctorants CPU, ainsi que les collègues du groupe AMI que j'ai pu côtoyer durant ces trois années.

Merci à Nédé pour tous ses bons cafés et sa bonne humeur, et pour nous avoir supportés tous les matins.

Merci à Franck pour toutes ces discussions sur les architectures ARM, et ses bons gâteaux.

Enfin, merci à toute ma famille, et en particulier mes parents, de m'avoir soutenu et d'avoir permis que je puisse aller au bout de ces trois ans, et merci à Ele pour son optimisme à toute épreuve qui compense mes périodes de doute !

Remerciements

Table des matières

1 Introduction **21**

1.1 Contexte . . . 22

1.2 Plan 3D Énergie . . . 24

1.3 Problématiques . . . 25

1.4 Objectifs . . . 26

1.4.1 Hypothèses . . . 26

1.4.2 Contraintes . . . 27

1.4.3 Matériel utilisé . . . 28

1.5 Contributions . . . 28

2 État de l'art sur la reconstruction 3D **35**

2.1 Introduction . . . 36

2.2 Capteurs de profondeur . . . 37

2.2.1 Capteurs temps de vol . . . 38

2.2.2 Capteurs stéréoscopiques . . . 39

2.2.3 Utilisation de capteurs de profondeur sur tablette . . . 40

2.3 Mise en correspondance de données RGB-D . . . 41

2.3.1 Alignement d'images . . . 41

2.3.2 Alignement de nuages de points . . . 44

2.3.3 Correction des dérives . . . 45

2.4 Reconstruction de maillages 3D en temps réel . . . 47

2.4.1 Nouveaux algorithmes de reconstruction 3D . . . 48

2.4.2 Optimisation de structures de données . . . 49

2.4.3 Reconstruction de maillages en temps réel sur tablettes . . . 49

2.5 Segmentation et classification de données 3D . . . 51

2.5.1 Segmentation 3D . . . 51

2.5.2 Classification d'objets . . . 53

2.6 Conclusion . . . 54

3 Reconstruction 3D d'un environnement intérieur et export de ses propriétés géométriques **57**

3.1 Introduction . . . 58

3.2 Travaux connexes . . . 59

3.2.1 Identification de la structure d'une scène 3D . . . 59

3.2.2 Conclusion . . . 61

3.3 Génération d'un maillage 3D en temps réel . . . 61
3.3.1 Vue d'ensemble de l'algorithme . . . 61
3.3.2 Données manipulées . . . 62
3.3.3 Alignement des points . . . 64
3.3.4 Intégration d'un nouveau nuage de points . . . 66
3.3.5 Mise à jour des maillages . . . 66
3.3.6 Paramètres . . . 67
3.4 Identification de la structure du bâtiment reconstruit . . . 67
3.4.1 Identification du sol et des murs . . . 68
3.4.2 Détection des ouvrants . . . 70
3.4.3 Paramètres . . . 71
3.5 Implémentation . . . 72
3.5.1 Répartition des tâches . . . 72
3.5.2 Acquisition des données . . . 73
3.5.3 Calculs . . . 73
3.6 Évaluations . . . 73
3.6.1 Performances . . . 74
3.6.2 Génération du modèle 3D . . . 75
3.6.3 Évaluation des performances énergétiques . . . 75
3.6.4 Évaluation à l'usage . . . 76
3.6.5 Discussion . . . 77
3.7 Conclusion . . . 78

4 Segmentation planaire en temps réel à l'aide d'un algorithme de croissance de régions **81**
4.1 Introduction . . . 82
4.2 Travaux connexes . . . 83
4.2.1 Calcul d'une moyenne pondérée . . . 83
4.2.2 Optimisation d'une fonction de coût . . . 84
4.2.3 Conclusion . . . 85
4.3 Description de l'algorithme . . . 85
4.3.1 Vue générale de l'algorithme mis en place . . . 85
4.3.2 Acquisition des données . . . 86
4.3.3 Segmentation planaire . . . 89
4.3.4 Paramètres . . . 92
4.4 Implémentation . . . 93
4.4.1 Répartition des tâches . . . 94
4.4.2 Mise en forme des données . . . 94
4.4.3 Segmentation . . . 95
4.4.4 Mémoires et partage des données . . . 95
4.5 Évaluations . . . 96

4.5.1 Évaluation de la précision et de la fiabilité 96
4.5.2 Discussion 97
4.6 Conclusion et travaux futurs 98

5 Segmentation planaire en temps réel à l'aide d'algorithmes de clustering **99**
5.1 Introduction 100
5.2 Description de l'algorithme 101
5.2.1 Vue générale de l'algorithme mis en place 101
5.2.2 Mise en forme des données 102
5.2.3 Clustering sur les normales 103
5.2.4 Classification finale 106
5.2.5 Paramètres 106
5.3 Implémentation 107
5.3.1 Tri des données 108
5.3.2 Clustering des normales 108
5.4 Évaluations 108
5.4.1 Évaluation de la précision et de la fiabilité 108
5.4.2 Discussion 109
5.4.3 Comparaison des deux algorithmes de segmentation planaire 109
5.5 Conclusion 110

6 Vers la reconstruction en temps réel d'un modèle 3D éditable **113**
6.1 Introduction 114
6.2 Identification de plans identiques lors de différentes prises de vue 115
6.2.1 Stockage en mémoire 115
6.2.2 Correction des dérives 117
6.3 Construction incrémentale des murs 120
6.3.1 Mise à jour de l'enveloppe de chaque plan 121
6.3.2 Classification finale 122
6.4 Finalisation de la maquette 123
6.4.1 Détection des ouvrants à l'aide des images RGB 123
6.4.2 Export de la maquette finale 124
6.5 Évaluations 124
6.5.1 Évaluation de la maquette générée 125
6.5.2 Discussion 126
6.6 Conclusion 126

7 Conclusion et travaux futurs **129**
7.1 Rappel des objectifs . . . 130
7.2 Contributions . . . 130
7.2.1 Reconstruction et segmentation de maillage 3D . . . 131
7.2.2 Mise en place d'algorithmes de segmentation planaire . . . 132
7.2.3 Construction d'un modèle 3D éditable à la volée . . . 133
7.3 Limitations . . . 133
7.4 Perspectives . . . 134

Annexes **134**

A Publications pendant la thèse **135**
A.1 Communications nationales . . . 136
A.2 Conférences internationales . . . 136
A.3 Journaux internationaux . . . 136

B Algorithmes utilisés **137**
B.1 Algorithme ICP . . . 138
B.2 Algorithme *Marching cubes* . . . 139

C Utilisation d'OpenMP pour paralléliser un algorithme **141**
C.1 Blocs parallèles . . . 142
C.2 Répartition du travail et synchronisation . . . 143

D Jeu d'instructions ARM NEON **145**
D.1 Description . . . 146
D.2 Exemple : calcul de l'intensité d'une image RGB . . . 147

E Étude expérimentale : navigation tangible dans un environnement virtuel 3D **149**
E.1 Introduction . . . 150
E.2 Travaux connexes . . . 151
E.2.1 Interaction tangible . . . 151
E.2.2 Navigation dans un environnement virtuel 3D . . . 152
E.2.3 Utilisation de facteurs d'échelle . . . 153
E.2.4 Conclusion . . . 153
E.3 Expérimentation . . . 153
E.3.1 Description . . . 154
E.3.2 Matériel . . . 154
E.3.3 Mesures . . . 155
E.3.4 Procédure . . . 155
E.4 Résultats . . . 156

E.4.1 Évaluation subjective 156
E.4.2 Évaluation des sketch maps 157
E.4.3 Analyse des trajectoires 158
E.5 Conclusion et travaux futurs 159

Bibliographie **160**

Table des figures

1.1 Différentes vues de l'application Plan 3D Énergie. 24
1.2 Représentation schématique des deux approches de reconstruction 3D envisagées au cours de ces travaux de thèse. 29

2.1 Les principaux domaines d'utilisation de la reconstruction 3D. Ce chapitre se focalise sur le domaine de la reconstruction en temps réel. 37
2.2 Carte de profondeur 38
2.3 Les trois tablettes et smartphones du projet Tango. 41
2.4 Reconstruction de maillages 3D avec CHISEL [71]. 50
2.5 Segmentation planaire en utilisant la méthode de Erdogan et al.[32]. 52
2.6 Exemple de segmentation avec *supervoxels* 53

3.1 Détection de la structure d'un bâtiment à partir d'images RGB [76]. 60
3.2 Vue d'ensemble de l'algorithme de génération du maillage 3D. . . . 62
3.3 Exemple de distances tronquées signées sur un espace à deux dimensions. Chaque case représente un voxel 2D, et la ligne rouge la surface pour laquelle ont été calculées les distances signées. . . . 63
3.4 Exemple de reconstruction de maillage avec notre algorithme. . . . 67
3.5 Les différentes étapes d'identification de la structure du bâtiment. . 68
3.6 Illustration du processus de détection des ouvrants. 70
3.7 Illustration du processus de reconstruction 72
3.8 Plan de la pièce avec les différentes mesures, complété par un participant. À gauche se trouve la vue de dessus de la pièce, et à droite les différents types d'ouvrants à mesurer. 74
3.9 Temps moyen d'intégration d'un nuage de points en fonction du nombre de threads de calcul utilisés, testé pour deux résolutions spatiale (2.5 cm et 3 cm). 74
3.10 Évaluation des performances énergétiques de la pièce testée. En rouge : la valeur obtenue avec l'application Plan 3D Énergie. 76
3.11 Scores SUS pour l'utilisation d'une tablette pour reconstituer une pièce. 77
3.12 Temps pris pour chacune des tâches : prise de mesures manuelle, puis à l'aide de la reconstruction 3D. 77

4.1 Calcul des normales à une surface en pondérant la somme par les angles . . . 84
4.2 Vue d'ensemble de l'algorithme de détection des plans . . . 86
4.3 Exemple de fenêtre glissante . . . 88
4.4 Exemple de données en entrée, avec le résultat de la segmentation planaire à la fin. Les données sont ordonnées en deux dimensions, ce qui évite l'utilisation d'algorithmes de recherche des plus proches voisins. . . . 89
4.5 Processus de génération des clusters . . . 91
4.6 Application de l'algorithme de détection des plans à plusieurs scènes différentes. . . . 92
4.7 Répartition des tâches et flux de données pour l'algorithme de segmentation. . . . 94
4.8 Montage expérimental pour évaluer la détection des plans : deux plans parallèles (le mur et la table) sont placés à la verticale devant la tablette. . . . 96
4.9 Exemple de plans parasites . . . 97

5.1 Vue générale de l'algorithme de segmentation planaire. . . . 102
5.2 Illustration 2D de l'algorithme de clustering des normales . . . 104
5.3 Exemples de détection des plans par clustering. . . . 107
5.4 Comparaison du pourcentage de frames justes n (en ordonnées) des deux algorithmes de segmentation planaire en fonction de la distance D (en abscisses) pour chaque configuration de x différente. 110

6.1 Mise à jour de la liste des plans . . . 116
6.2 Identification de plans identiques sur plusieurs prises de vue . . . 117
6.3 Recherche de correspondances . . . 119
6.4 Mise à jour de l'enveloppe de chaque plan . . . 121
6.5 Exemples de détection des murs. . . . 123
6.6 Exemple de fichier d'export. . . . 124
6.7 Les trois environnements réels utilisés pour l'évaluation. . . . 125

7.1 Résumé des deux approches de reconstruction 3D abordées et des briques algorithmiques mises en place. . . . 131

B.1 Les configurations de référence pour l'algorithme *Marching Cubes*[82].140

E.1 Boom chameleon [137]. . . . 151
E.2 Étude de l'utilisation de la marche réelle pour naviguer dans un EV 3D [148]. . . . 152

E.3 Techniques d'interaction considérées dans cette étude : mouvements de la tablette (en haut à droite), *multi-touch* "classique" (en haut à gauche), et *multi-touch* avec sticks (en bas). . 153
E.4 Vue du dessus de l'environnement virtuel visié par les participants lors de l'exploration. 154
E.5 Score SUS moyen pour chacune des modalités testées 156
E.6 Score d'acceptabilité moyen pour chacune des modalités testées. . . 156
E.7 Valeurs moyennes pour la justesse des proportions restituées, le score maximal est de 6. 157
E.8 Écarts moyens entre les ratios représentés et réels. 157
E.9 Distances angulaires moyennes. 158
E.10 Pourcentages moyens de temps d'exploration statique et à reculons. 158
E.11 Distance moyenne parcourue. 159
E.12 Nombre moyen de collisions. 159

Table des figures

Liste des tableaux

1.1 Résumé des différents chapitres. 34

3.1 Paramètres pour l'algorithme CHISEL 67
3.2 Paramètres pour la génération du maillage 3D et la segmentation planaire. 72
3.3 Comparaisons entre les mesures réelles et les dimensions calculées. 75

4.1 Paramètres de l'algorithme de segmentation planaire. 93
4.2 Résultats des évaluations de l'algorithme de segmentation par croissance de régions. 97

5.1 Paramètres pour la segmentation planaire basée sur le clustering. . 106
5.2 Résultats des évaluations de l'algorithme de segmentation avec *k-means*. 109

6.1 Évaluation de la détection des plans 125

7.1 Liste des hypothèses pour chacune des approches mises en place. Les quatre premières sont les hypothèses générales que nous avions formulées dès le début (voir §1.4.1). 132

E.1 Grandeurs mesurées à partir des trajectoires des participants 156

Liste des Algorithmes

1 Recherche de correspondances pour CHISEL[71]. 65
2 Alignement de correspondances en utilisant les SVD [3]. 65
3 Détection des zones rectangulaires vides. 71

4 Génération récursive d'un cluster. 90

5 Algorithme de tri des points. 103
6 Tri récursif. 103
7 Algorithme *k-means* pour le clustering des normales. 105
8 Séparation des plans parallèles dans un cluster. 106

9 Recherche de correspondances entre deux nuages de points ordonnés. 119
10 Alignement de correspondances en utilisant les normales. 120

11 Détail de l'algorithme ICP. 138
12 Détail de l'algorithme *Marching cubes*. 140

Liste des abréviations

BIM : *Building Information Modeling*

CAO : *Conception Assistée par Ordinateur*

DBSCAN : *Density-Based Spatial Clustering of applications with Noise*

DDL : *Degrés De Liberté*

CNN : *Convolutional Neural Network*

EGI : *Extended Gaussian Image*

EV : *Environnement virtuel*

FAST : *Features from Accelerated Segment Test*

HMD : *Head Mounted Display*

ICP : *Iterative Closest Point*

IFC : *Industry Fundation Class*

ISS : *Intrinsic Shapes Signature*

LIDAR : *LIght Detection And Ranging*

NARF : *Normal Aligned Radial Feature*

NDT : *Normal Distribution Transform*

SDF : *Signed Distances Field*

TOF : *Time Of Flight*

TSDF : *Truncated Signed Distances Field*

PCA-SIFT : *Principal Components Analysis SIFT*

SIFT : *Scale Invarient Feature Transform*

SLAM : *Simultaneous Localization And Mapping*

SURF : *Speed Up Robust Features*

SUSAN : *Smallest Univalue Segment Assimilating Nucleus*

SVD : *Singular Values Decomposition*

SVM : *Support Vector Machines*

VIO : *Visual Inertial Odometry*

Notations

Notations générales

x : Variable

$\boldsymbol{u}$: Vecteur de $\mathbb{R}^3$

$\boldsymbol{M}$: Matrice de $\mathcal{M}_{n,p}(\mathbb{R})$

E : Élément d'un ensemble

$\mathcal{E}$: Un ensemble

$\mathcal{E}(i,j)$: Élément de $\mathcal{E}$ indexé par (i,j)

$f(\cdot)$: Fonction mathématique

$\boldsymbol{\nabla}_f(.)$: Gradient de f

$Card(.)$: Cardinal d'un ensemble

$||.||$: Norme d'un vecteur

$\wedge$: Produit vectoriel

$\cdot$: Produit scalaire

$\oplus$: OU exclusif

$.^\top$: Transposée

Conventions pour ce document

θ_i, ϕ_i : Angles

ω_i : Coefficient de pondération

ϵ_i : Seuil

ρ_i : Résolution spatiale

$\boldsymbol{\lambda_i}$,$\boldsymbol{\mu_i}$: Centres de clusters

$\boldsymbol{p_i}$: Point dans $\mathbb{R}^3$

$\boldsymbol{n_i}$: Vecteur normal dans $\mathbb{R}^3$

$\boldsymbol{u_i}, \boldsymbol{v_i}$: Vecteurs unitaires de l'espace

$\boldsymbol{M_i}$: Transformation affine de $\mathbb{R}^3$

$\boldsymbol{T_i}$: Translation dans $\mathbb{R}^3$

$\boldsymbol{R_i}$ **:** Rotation dans $\mathbb{R}^3$

$\boldsymbol{O_p}(\mathbb{R})$ **:** Matrice nulle dans $\mathcal{M}_p(\mathbb{R})$

$\boldsymbol{I_p}(\mathbb{R})$ **:** Matrice identité dans $\mathcal{M}_p(\mathbb{R})$

V_i **:** Un voxel

P_i **:** Élément d'un nuage de points 3D

$\mathcal{V}_i$ **:** Volume de l'espace

$\mathcal{S}_i$ **:** Surface 3D

$\mathcal{C}_i$ **:** Couple de points correspondants

$\mathcal{P}_i$ **:** Nuage de points 3D

$\mathcal{M}_i$ **:** Carte 2D

Π_i **:** Plan 3D

$\mathcal{R}_i$ **:** Repère dans l'espace

$\phi(.,.)$ **:** Distance signée

$\phi_\tau(.,.)$ **:** Distance signée tronquée

$\delta(.,.)$ **:** Distance de $\mathbb{R}^3$

Chapitre 1

Introduction

1.1 Contexte 22
1.2 Plan 3D Énergie 24
1.3 Problématiques 25
1.4 Objectifs 26
1.4.1 Hypothèses 26
1.4.2 Contraintes 27
1.4.3 Matériel utilisé 28
1.5 Contributions 28

1.1 Contexte

Ces travaux de thèse s'inscrivent dans le cadre de la collaboration entre le LIMSI-CNRS, et l'entreprise Rénovation Plaisir Énergie (RPE). L'objectif de ces travaux est de concevoir un ensemble de solutions algorithmiques permettant de modéliser un environnement intérieur existant, à l'aide d'une tablette équipée d'un capteur de profondeur. Le modèle 3D généré devra être éditable et exportable sur d'autres applications d'édition 3D.

Ces outils seront intégrés à terme dans une application sur tablette permettant à un particulier ou un professionnel de construire une maquette 3D d'un bâtiment existant à moindre coût. À partir du modèle 3D obtenu, il sera possible d'avoir accès aux différentes dimensions et surfaces du bâtiment, ainsi qu'aux informations sur ses performances énergétiques. Ces travaux de thèse répondent à un besoin croissant de modéliser en 3D les bâtiment existants.

Ce besoin est en grande partie lié à la mise en place de la directive Européenne 2013/31/EU [1], qui encourage tous les états membres de l'UE à améliorer les performances énergétiques de l'ensemble de leur parc immobilier. Cela implique de mettre en place de nouvelles normes pour la construction, mais aussi de rénover les anciens bâtiments. En France par exemple, il a été fixé comme objectif de rénover 500000 bâtiments chaque année. La construction d'une maquette 3D d'un bâtiment à rénover permet de visualiser rapidement les futurs travaux, et d'en estimer les coûts.

De même, des diagnostics de performances énergétiques sont maintenant réalisés de manière systématique sur tous les bâtiments. Ils sont réalisés par des professionnels qui relèvent les différentes mesures et surfaces du bâtiment, ainsi que les matériaux de construction.

Le format des modèles numériques associés aux bâtiments a été standardisé par l'introduction du standard BIM (*Building Information Modeling*)[29], qui définit le format IFC (*Industry Fundation Class*) [2], qui décrit les différents composants du bâtiment (architecture, plomberie, matériaux, ...), et les relations entre ces éléments. On parle alors de maquette numérique pour désigner un modèle BIM associé à un bâtiment.

Cette maquette, en plus d'une modélisation 3D, contient les informations relatives à l'ensemble du cycle de vie du bâtiment (conception, construction, maintenance, etc...). Une maquette numérique au format IFC peut être importée dans tous les logiciels de Conception Assistée par Ordinateur (CAO).

Dès lors, ce standard tend à s'imposer pour modéliser les bâtiments, ce qui permet d'uniformiser leur représentation. Alors que la maquette numérique

1. https ://eur-lex.europa.eu/legal-content/EN/TXT/ ?uri=celex%3A32013L0031
2. https ://www.steptools.com/stds/ifc/html/

associée à un bâtiment est maintenant réalisée dès la phase de conception, il est en général plus complexe de modéliser les anciens bâtiments. De nombreux aspects, comme les installations électriques, ou la structure interne du bâtiment sont difficilement modélisables s'il n'existe plus de plan original du bâtiment. Nos travaux porteront sur la génération automatique du modèle 3D associé à cette maquette numérique, qui peut être reconstruite à l'aide d'un capteur de profondeur.

Des outils professionnels permettent déjà une reconstruction d'un nuage de points ou d'un maillage 3D de très bonne qualité d'un bâtiment. On peut citer par exemple la série de scanners 3D DPI de la société DOTProduct [3] ou la série de scanners 3D FARO focus [4]. Ces outils reposent sur l'utilisation de capteurs laser, ou de stéréoscopie pour générer des nuages de points 3D avec une très grande précision. Ils restent coûteux, et ne permettent pas de générer une maquette 3D en temps réel. Ils sont cependant très bien adaptés à la reconstruction d'un bâtiment de grande échelle.

Depuis peu, il est possible d'équiper des tablettes avec des capteurs de profondeur, qui possèdent en général une précision plus faible que des capteurs laser ou stéréoscopiques, mais qui s'avèrent beaucoup plus abordables. Ils ont été popularisés par l'émergence des technologies de réalité augmentée, comme le projet Tango de Google. Ce projet a consisté au développement de systèmes mobiles intégrant un capteur de profondeur, et une API de suivi de déplacements. Ces dispositifs ont été pensés afin de "percevoir" l'environnement physique dans lequel l'utilisateur évolue.

Le projet Tango a été abandonné au profit du projet ARCore [5], qui conserve uniquement les fonctionnalités de suivi des déplacements. Cependant, il existe des capteurs de profondeur à prix très abordable pouvant être connectés à une tablette, l'un des plus connus étant le StructureSensor [6].

Il est donc possible d'envisager l'utilisation d'une tablette pour reconstruire une maquette 3D d'un environnement intérieur réel. La portée de de ces capteurs étant souvent limitée à quelques mètres, la reconstruction ne peut se faire que dans des environnements intérieurs de faibles dimensions (typiquement, un logement). Dans les autres cas, les outils professionnels sont plus adaptés. De plus, il faut envisager une reconstruction de moins bonne qualité qu'à l'aide d'un capteur laser, mais qui peut se faire en temps réel. Une discussion plus détaillée des technologies des capteurs est présentée en section 2.2.

Nous nous sommes donc intéressés à l'utilisation d'une tablette équipée d'un capteur de profondeur pour générer une maquette numérique éditable d'un

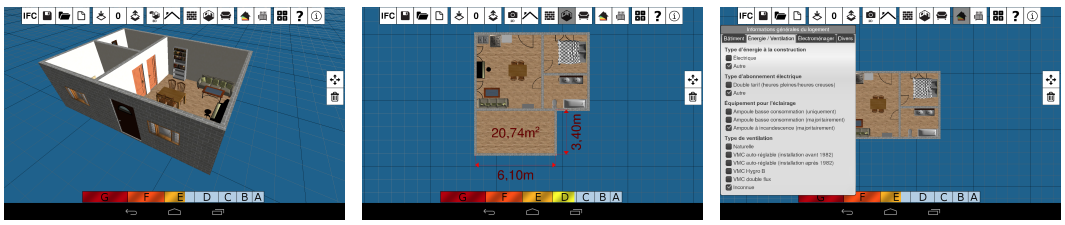
3. https ://www.dotproduct3d.com/
4. https ://www.faro.com/fr-fr/produits/construction-bim-cim/faro-focus/
5. https ://developers.google.com/ar/discover/
6. https ://structure.io/

environnement intérieur. À condition de faire quelques concessions sur la qualité de la reconstruction, l'utilisation d'une tablette présente deux principaux avantages. Tout d'abord, les capteurs disponibles sont bon marché par rapport à des capteurs de profondeur utilisés par des professionnels. Ensuite, dans le cas des tablettes développées pour le projet Tango de Google, il s'agit de dispositifs grand public, qui ne nécessitent pas de matériel particulier.

Le reste de ce chapitre est présenté comme suit : tout d'abord, nous présentons l'application Plan 3D Énergie (section 1.2), développée par le LIMSI-CNRS dans le cadre de la collaboration avec RPE. Nous voudrions l'enrichir avec un module de reconstruction 3D. Ensuite, nous décrivons les problématiques traitées au cours de cette thèse (section 1.3), et les objectifs des travaux menés (section 1.4). Enfin, nous détaillons les contributions apportées par nos travaux (section 1.5). La liste complète des publications scientifiques produites durant ces travaux est détaillée en annexe A.

1.2 Plan 3D Énergie

L'application Plan 3D Énergie a été développée par le LIMSI-CNRS dans le cadre de la collaboration avec l'entreprise RPE et est disponible sur Google Play[7]. Elle permet de réaliser et modifier un modèle 3D virtuel d'un logement sur une tablette tactile.



(a) *Vue 3D* **(b)** *Édition d'une pièce* **(c)** *Ajout d'informations*

Figure 1.1 – *Différentes vues de l'application Plan 3D Énergie. Le menu du haut contient les outils pour ajouter des pièces, du mobilier, ou éditer le modèle 3D. La barre du dessous affiche en temps réel les performances énergétiques estimées du logement. Des informations générales comme l'année de construction ou la nature des matériaux peuvent être ajoutées pour affiner l'estimation des performances énergétiques.*

En plus des outils d'édition classiques (ajout de pièces, murs, mobilier, mesures, etc...), l'application calcule en temps réel une estimation des performances énergétiques du bâtiment modélisé (voir fig. 1.1).

L'estimation des performances énergétiques est réalisée grâce une

7. https ://play.google.com/store/apps/details ?id=fr.limsi.rorqual

implémentation de l'algorithme 3CL[8]. Il s'agit d'un algorithme permettant d'estimer les performances énergétiques d'un bâtiment à partir de ses caractéristiques (surfaces, murs extérieurs, ponts thermiques, etc ...). Cette estimation est effectuée en temps réel, c'est-à-dire pendant que l'utilisateur réalise son modèle 3D. Le calcul des performances énergétiques se base dans un premier temps sur les caractéristiques géométriques du logement et la configuration la moins optimiste est prise par défaut pour les autres caractéristiques. L'utilisateur peut ensuite affiner cette estimation en renseignant des informations complémentaires pour chaque élément du modèle 3D, comme par exemple, la nature du vitrage, les matériaux, ou l'année de construction.

Grâce à cette application, un utilisateur peut réaliser un modèle 3D d'un environnement intérieur existant, et avoir des informations pratiques concernant d'éventuels travaux de rénovation à effectuer.

1.3 Problématiques

Comme nous le verrons dans le chapitre 2, les problématiques liées à la reconstruction 3D ont déjà été largement abordées par le passé. En effet, il s'agit d'un moyen de cartographier un environnement réel et ce domaine trouve ses premiers usages dans la robotique autonome. Il existe déjà de nombreux algorithmes permettant de manipuler des nuages de points 3D et d'en extraire des maillages 3D. Beaucoup de ces algorithmes sont regroupés dans la bibliothèque de manipulation de nuages de points *Point Cloud Library* (PCL)[9].

Ces recherches se sont intensifiées avec la démocratisation des capteurs de profondeur. Il existe désormais des algorithmes permettant de générer des maillages 3D en temps réel à grande échelle à l'aide d'un capteur de profondeur portable.

En parallèle, de nombreux travaux abordent la segmentation de données 3D et la classification automatique de ces données.

Cependant, certaines problématiques se posent lorsque l'on envisage de mettre en place un algorithme de génération de modèle 3D éditable sur une tablette :

8. http ://www.rt-batiment.fr/fileadmin/documents/RT_existant/DPE/DPE_outils/Nouvel_Algorithme_3CL-DPE_vf.pdf
9. http ://pointclouds.org/

Problématiques

- Les capacités de calcul des tablettes sont encore limitées.
- Il existe très peu d'outils de reconstruction 3D disponibles sur tablette.
- Il est difficile d'automatiser complètement le processus de modélisation.

En effet, un grand nombre d'algorithmes de reconstruction 3D existants reposent sur l'utilisation intensive des GPU, encore peu développés sur les tablettes. Il est alors souvent nécessaire repenser ces algorithmes pour les adapter à l'utilisation sur tablette.

De plus, les tablettes sont des dispositifs relativement récents. Les capteurs de profondeur qui peuvent y être intégrés sont encore peu nombreux, et les travaux abordant la reconstruction 3D sur tablette émergent tout juste.

Ensuite, beaucoup de travaux portent sur la segmentation et la classification de nuages de points 3D, mais il est presque impossible à l'heure actuelle de classifier l'ensemble des constituants d'un bâtiment. Cela est dû en partie à la complexité du problème, mais aussi parce qu'il est difficile d'effectuer une capture complète d'un bâtiment, car il existe toujours des zones inaccessibles au capteur de profondeur. Il est alors envisageable de formuler des hypothèses quant aux caractéristiques du bâtiment pour aider à la classification des éléments.

1.4 Objectifs

L'objectif des travaux de thèse est d'apporter des solutions à ces problématiques dans le contexte de la rénovation d'intérieur. Dans ce cas, nous n'avons besoin de modéliser que la structure globale du bâtiment (sol, plafond, murs et ouvrants). Cela suffit à obtenir les différentes dimensions et surfaces du bâtiment, ainsi que des informations sur ses performances énergétiques. Nous pouvons alors formuler des hypothèses sur la structure du bâtiment permettant de simplifier la modélisation.

1.4.1 Hypothèses

Par rapport au contexte de la modélisation d'un bâtiment pour la rénovation énergétique, nous avons émis un certain nombre d'hypothèses que nous avons utilisées pour l'ensemble des algorithmes que nous avons mis en place :

Hypothèses

1. Les murs du bâtiment à reconstruire sont plans.
2. La structure du bâtiment à reconstruire est alignée par rapport à un repère orthonormé $\mathcal{R}_0$.
3. La reconstruction se fait pièce par pièce.
4. La tablette utilisée intègre un algorithme d'odométrie permettant d'estimer sa position par rapport à un repère de base $\mathcal{R}_1$.

Une partie de ces hypothèses vient de l'affirmation que le bâtiment est construit selon une grille orthogonale. Ces hypothèses sont connues sous le nom de ***Manhattan World Assumption***. Ces hypothèses sont très utilisées lors de l'identification d'éléments structuraux pour des constructions (qu'il s'agisse d'environnement intérieurs ou de paysages urbains). La plupart des villes et des bâtiments sont construits selon une grille euclidienne : un axe colinéaire au vecteur gravité, et le deux autres définissant un plan orthogonal à cet axe. Ainsi, le sol est supposé être orthogonal à l'axe de gravité, tandis que les murs sont orthogonaux au sol et alignés sur cette grille.

Pour simplifier la reconstruction, nous pouvons l'effectuer pièce par pièce, et assembler les modèles *a posteriori*. La mise en place de *frameworks* de réalité augmentée, comme le projet Tango de Google permet de s'appuyer sur des technologies fiables pour le suivi des mouvements de la tablette.

1.4.2 Contraintes

Nous avons également dû prendre en compte un certain nombre de contraintes lors de nos travaux :

Contraintes

1. Les algorithmes mis en place doivent utiliser uniquement les capacités de calcul de la tablette.
2. La mise à jour du modèle 3D doit se faire dans un intervalle de temps inférieur à 200 ms.
3. La précision des mesures doit être inférieure à 5 cm localement.
4. L'erreur d'estimation des mesures à l'échelle d'une pièce doit être inférieure à 5% de la dimension réelle.
5. La modélisation doit demander le moins d'interventions possible.

Afin de viser un large public, les dépendances à des dispositifs externes doivent être réduites au maximum. Aussi, le seul dispositif externe que nous utilisons est un capteur de profondeur intégré à la tablette. Ces capteurs de profondeur sont très répandus sur le marché et très abordables. Pour le reste, les algorithmes mis en place utilisent uniquement les capacités de calcul de la tablette. De plus, utiliser dispositif externe pour effectuer les traitements algorithmiques revient à se placer dans un contexte de reconstruction 3D plus général, pour lequel de nombreuses solutions existent déjà.

Pour des raisons d'utilisabilité, la phase de capture 3D doit s'effectuer en temps réel. Cela permet à l'utilisateur d'avoir un retour visuel sur le modèle 3D créé, et permet de réaliser la capture dans un délai raisonnable. Nous avons fixé un intervalle de temps maximal de 200 ms pour effectuer les calculs lorsqu'un nouveau nuage de points est renvoyé par la capteur de profondeur. Cela permet d'en traiter 5 par seconde, ce qui est la fréquence d'acquisition du capteur de profondeur pour les dispositifs du projet Tango de Google.

Enfin, nous avons fixé comme contrainte que le processus de modélisation doit nécessiter le moins d'interventions possibles. L'utilisateur intervient seulement pour réaliser une capture complète de l'environnement intérieur à modéliser, en déplaçant la tablette dans l'environnement physique. Les algorithmes d'identification de la structure du bâtiment doivent fonctionner de manière autonome.

1.4.3 Matériel utilisé

Nous avons effectué tous nos développements et nos évaluations à l'aide d'une tablette Google Tango Yellowstone. Cette tablette intègre un capteur de profondeur et un algorithme d'odométrie basé sur l'algorithme *Visual Inertial Odometry* (VIO) [79].

Une API [10] permet d'accéder aux informations de ces capteurs. Le capteur de profondeur fournit des nuages de points ordonnés selon un espace 2D avec une résolution maximale de 640×780. La caméra couleur fournit une image avec une résolution de 1280×720 pixels, et il est possible d'estimer la position de la tablette à chaque instant grâce à l'algorithme d'odométrie.

1.5 Contributions

Nous avons tout d'abord effectué un état de l'art sur les techniques de reconstruction 3D. Ce problème étant très large, nous nous sommes concentrés

10. *Application Programming Interface*

sur les problématiques liées à la reconstruction de maillages 3D en temps réel, et la segmentation de données RGB-D. Ces travaux font l'objet du chapitre 2.

À partir de cet état de l'art, nous avons envisagé deux approches différentes pour reconstruire un modèle 3D éditable d'un bâtiment existant. La première approche consiste à générer un maillage 3D en temps réel de ce bâtiment à l'aide de la tablette, et d'en classifier les différents éléments structuraux (murs, portes et fenêtres) pour ensuite exporter un modèle 3D éditable. La seconde approche consiste à générer ce modèle éditable en temps réel. Ces deux approches sont synthétisées sur la figure 1.2.

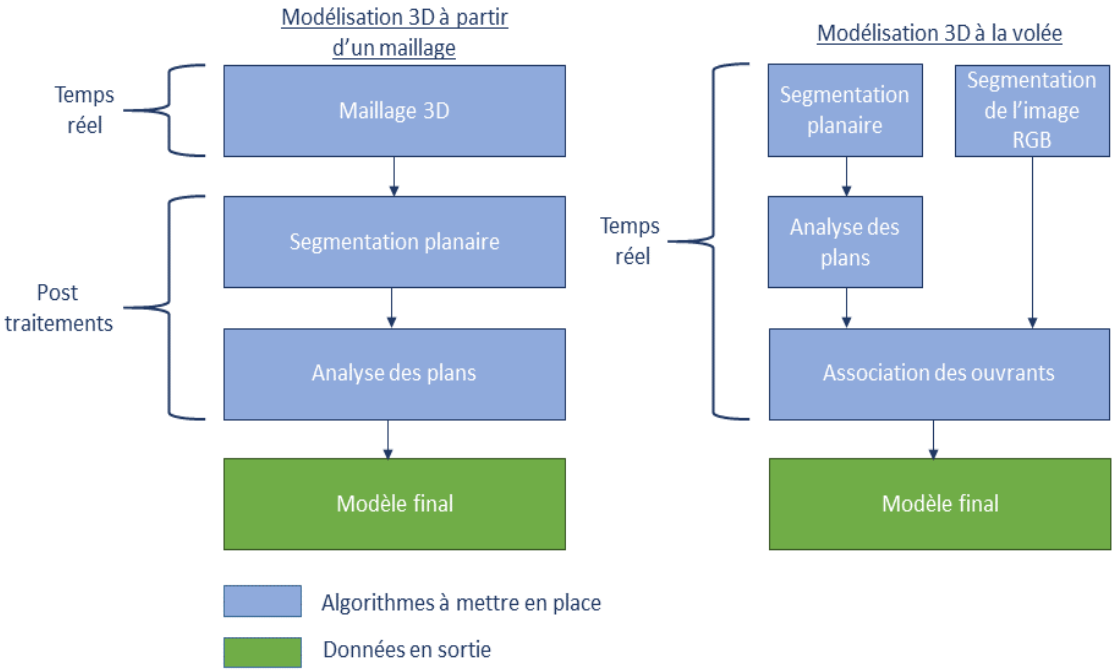


Figure 1.2 – *Représentation schématique des deux approches de reconstruction 3D envisagées au cours de ces travaux de thèse.*

Lors de la première approche (voir figure 1.2 à gauche), nous avons mis en place un algorithme de modélisation 3D qui s'effectue en deux étapes. Tout d'abord, un maillage 3D est généré. Pour cela, l'utilisateur effectue une capture de l'environnement intérieur à modéliser, et le maillage 3D associé se met à jour en temps réel. Une fois un maillage 3D reconstitué, la seconde phase consiste à le segmenter et à identifier la structure du bâtiment. Cette phase est effectuée en arrière plan et son temps d'exécution dépend de la taille du maillage 3D. Un algorithme de segmentation planaire permet d'isoler les plans du reste des points. Chaque plan est alors étiqueté comme étant un mur, le sol, le plafond ou du bruit. Un algorithme analyse ensuite chacun des plans pour identifier les ouvrants éventuels. Cette approche est décrite intégralement au chapitre 3.

La deuxième approche envisagée (voir figure 1.2 à froite) est plus originale. Plutôt que de construire d'abord un maillage 3D avant de classifier ses

constituants, nous avons choisi de générer un modèle 3D éditable directement. Cela évite de stocker ce maillage intermédiaire de taille conséquente, et permet de tirer parti de toutes les informations disponibles à chaque prise de vue des capteurs.

Nous avons tout d'abord proposé deux algorithmes de segmentation planaire, utilisant deux techniques différentes. Ces algorithmes permettent de segmenter un nuage de points d'une résolution maximal de 640×180 en temps réel.

Le premier de ces algorithmes, décrit au chapitre 4, repose sur le fait que les nuages de points en entrée sont ordonnés selon une grille 2D. Il consiste à segmenter les nuages de points en entrée en utilisant un algorithme de croissance de régions. Le deuxième algorithme utilise des algorithmes de clustering plus génériques pour permettre une segmentation plus efficace, mais nécessite de démarrer la capture face à un mur. Il est détaillé au chapitre 5.

Nous avons ensuite mis en place toute une chaîne de traitements permettant d'identifier et de construire de manière incrémentale les murs contenus dans une scène 3D, ces travaux sont détaillés dans le chapitre 6.

Nous détaillerons ensuite les travaux restants pour l'identification des ouvrants et la conclusion à ces travaux au chapitre 7.

Enfin, nous présentons en annexe E une étude sur l'utilisation des mouvements d'une tablette pour explorer un environnement 3D que nous avons réalisée en parallèle de ces travaux. L'objectif de cette étude était d'explorer de nouvelles techniques d'interaction dans l'objectif de pouvoir éditer le modèle 3D précédemment reconstruit.

Le résumé des différents chapitres est présenté dans le tableau 1.1.

Chapitre	*Objectifs*
2. État de l'art sur la reconstruction 3D	**Patie 1 : capteurs de profondeur** • Description et comparaison des principales technologies utilisées pour les capteurs de profondeur. **Partie 2 : mise en correspondance de données RGB-D** • Présentation de techniques classiques de détection de points d'intérêt et de calcul de descripteurs dans une image 2D. • Extension de ces techniques à des nuages de points 3D. • Présentation de différentes techniques d'alignement de nuages de points 3D. **Partie 3 : reconstruction de maillages 3D en temps réel** • Aperçu des algorithmes de reconstruction de maillages 3D en temps réel depuis la popularisation des capteurs de profondeur. • Optimisation de ces algorithmes. • Revue des algorithmes de reconstruction de maillages 3D sur systèmes mobiles. **Partie 4 : segmentation et classification de données 3D** • Aperçu des principales techniques de segmentation et classification de données 3D.

Chapitre	*Objectifs*
3. Reconstruction 3D d'un environnement intérieur et export de ses propriétés géométriques	**Partie 1 : travaux connexes** • Études de travaux précédents sur l'identification de la structure d'un bâtiment à partir d'un nuage de points 3D. **Partie 2 : génération d'un maillage 3D en temps réel** • Implémentation de notre propre version de l'algorithme CHISEL [71]. **Partie 3 : identification de la structure du bâtiment reconstruit** • Segmentation planaire à l'aide de VCCS [102]. • Mise en place d'un algorithme de détection des ouvertures rectangulaires. **Partie 4 : implémentation** • Détail de l'implémentation pratique des parties critiques des algorithmes mis en place. **Partie 5 : évaluations** • Évaluation des performances des algorithmes mis en place. • Évaluation de la précision du modèle 3D généré. • Évaluation des algorithmes à l'usage.
4. Segmentation planaire en temps réel à l'aide d'un algorithme de croissance de régions	**Partie 1 : travaux connexes** • Aperçu des techniques de calcul des normales. **partie 2 : description de l'algorithme** • Mise en place d'un algorithme de segmentation planaire inspiré de VCCS [102]. **Partie 3 : implémentation** • Détails sur l'implémentation pratique de l'algorithme. **Partie 4 : évaluations** • Évaluation de la précision de l'algorithme mis en place. • Évaluation de la fiabilité de cet algorithme.

Chapitre	*Objectifs*
5. Segmentation planaire en temps réel à l'aide d'algorithmes de clustering	**Partie1 : description de l'algorithme** • Mise en place d'un algorithme de tri des nuages de points en fonction de leurs paramètres de plans. • Mise en place d'un algorithme de segmentation des normales basé sur l'algorithme *k-means*. • Mise en place d'un algorithme de clustering à une dimension pour séparer les plans parallèles. **Partie 2 : implémentation** • Détails de l'implémentation pratique des algorithmes mis en place. **Partie 3 : évaluations** • Évaluation de la fiabilité et de la précision de cet algorithme. • Comparaison des deux algorithmes de segmentation.
6. Vers la reconstruction en temps réel d'un modèle 3D éditable	**Partie 1 : Identification de plans identiques sur plusieurs prises de vue différentes** • Association d'un identifiant unique à chaque plan. • Mise en place d'un algorithme de correction des dérives sur des nuages de points ordonnés. **Partie 2 : construction incrémentale des murs** • Mise en place d'un algorithme de mise à jour de l'enveloppe concave d'un plan. **Partie 3 : finalisation de la maquette** • Détail de l'export final. **Partie 4 : évaluations** • Évaluation des dimensions du modèle final par rapport aux dimensions réelles.

Chapitre	*Objectifs*
7. Conclusion	**Partie 1 : rappel des objectifs** • Rappel du contexte et des problématiques. • Rappel des hypothèses et des contraintes. **Partie 2 : contributions** • Résumé des hypothèses pour chacun des algorithmes mis en place. • Résumé des contributions de ces travaux de thèse. **Partie 3 : limitations** • Résumé des limitations matérielles rencontrées. **Partie 4 : perspectives** • Perspectives à court terme. • Perspectives à long terme.

Tableau 1.1 – *Résumé des différents chapitres.*

Chapitre 2

État de l'art sur la reconstruction 3D

2.1 Introduction 36
2.2 Capteurs de profondeur 37
2.2.1 Capteurs temps de vol 38
2.2.2 Capteurs stéréoscopiques 39
2.2.3 Utilisation de capteurs de profondeur sur tablette 40
2.3 Mise en correspondance de données RGB-D 41
2.3.1 Alignement d'images 41
2.3.2 Alignement de nuages de points 44
2.3.3 Correction des dérives 45
2.4 Reconstruction de maillages 3D en temps réel 47
2.4.1 Nouveaux algorithmes de reconstruction 3D 48
2.4.2 Optimisation de structures de données 49
2.4.3 Reconstruction de maillages en temps réel sur tablettes . . . 49
2.5 Segmentation et classification de données 3D 51
2.5.1 Segmentation 3D 51
2.5.2 Classification d'objets 53
2.6 Conclusion 54

2.1 Introduction

La capture 3D d'un environnement existant se fait en déplaçant un capteur de profondeur dans l'environnement en physique, afin d'obtenir un ensemble d'images 3D couvrant l'intégralité de la surface à reconstruire. Pour mettre en correspondance les différentes images 3D entre elles il est nécessaire de connaître la position du capteur de profondeur à chaque capture.

Nous traitons ici le cas de l'utilisation d'un capteur RGB-D, fournissant une image couleur et une carte de profondeur. Dans le cas d'un système de capture autonome (sans dispositif de positionnement externe), connaître le vecteur de déplacement $\boldsymbol{u}(t)$ du capteur entre deux instants t et $t+1$ est en général assez complexe. On peut estimer de manière théorique le déplacement $\boldsymbol{v}(t)$ du capteur à l'aide d'une centrale inertielle par exemple. En pratique, $\mathbf{v}(t)$ est souvent différent du véritable déplacement $\boldsymbol{u}(t)$ en raison des différents bruits et erreurs de mesure. Cela entraîne alors une série de positions estimées du capteur $\{\boldsymbol{y_0}, \boldsymbol{y_1}, ..., \boldsymbol{y_T}\}$ différentes des positions réelles du capteur $\{\boldsymbol{x_0}, \boldsymbol{x_1}, ..., \boldsymbol{x_T}\}$, et une dérive croissante entre les $(\boldsymbol{x_t}, \boldsymbol{y_t})$. On utilise alors les observations $\boldsymbol{z_t}$ du capteur RGB-D pour corriger ces erreurs.

Puisqu'elle nécessite une navigation du capteur dans l'espace physique, la reconstruction 3D est en fait un sous problème du problème mathématique **SLAM** (*Simultaneous Localization And Mapping*) [28, 4, 135], qui concerne les robots autonomes. Le problème SLAM consiste pour un robot autonome évoluant dans un espace inconnu à mesurer sa position réelle, et à cartographier cet espace. Le robot ne connaît avec certitude que sa position initiale, et doit estimer sa position réelle à chaque instant t à partir de son vecteur de déplacement supposé, et des différentes observations qu'il peut faire de l'environnement physique.

De nombreuses solutions théoriques au SLAM [15, 92, 5] ont été proposées, permettant de suivre et de positionner un robot dans son environnement physique. Dans le cadre de nos travaux, nous nous intéressons principalement à la correction des dérives de position, qui peuvent affecter le résultat final de la reconstruction.

On peut facilement identifier quelques principaux domaines d'utilisation de la reconstruction 3D (voir la figure 2.1) :

- **SLAM pour la robotique :** l'une des premières approches de la reconstruction 3D [143, 94, 100, 31]. La reconstruction 3D sert à cartographier l'environnement physique dans lequel un robot autonome évolue.
- **Reconstruction à grande échelle :** la reconstruction 3D est très utilisée pour la modélisation à grande échelle [106, 65, 46], comme des monuments historiques ou des sites industriels. La modélisation est réalisée par des outils professionnels qui génèrent des maillages 3D de très grande précision. Pour

ce genre de modélisation, des capteurs de type LIDAR[1] sont très souvent utilisés.

- **Reconstruction de scènes en temps réel :** il s'agit d'une alternative plus abordable par rapport à l'utilisation d'outils professionnels qui s'applique à des environnements intérieurs [61, 96].
- **Reconstruction d'objets :** beaucoup de travaux étudient la reconstruction 3D d'objets, ou de personnes, très souvent en utilisant uniquement une image couleur [42, 56, 105], mais aussi à l'aide de capteurs de profondeur [2].
- **Imagerie médicale :** la reconstruction 3D sert ici à améliorer le visualisation des images médicales pour un diagnostic plus précis [147, 124].

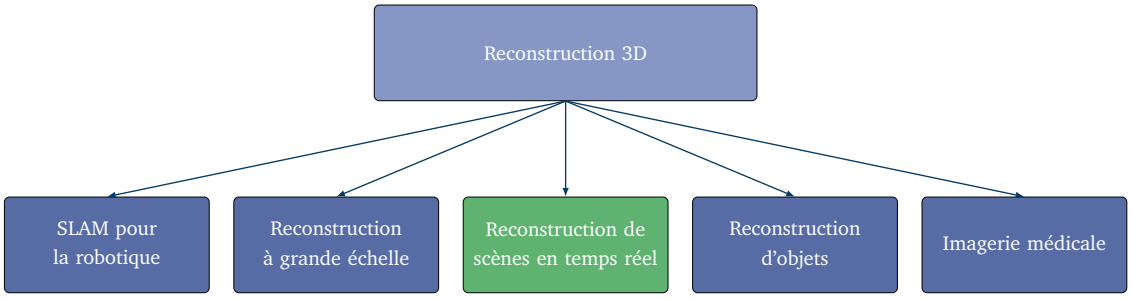


Figure 2.1 – *Les principaux domaines d'utilisation de la reconstruction 3D. Ce chapitre se focalise sur le domaine de la reconstruction en temps réel.*

Nous nous concentrons particulièrement sur les travaux portant sur la reconstruction de scènes en temps réel. Ce domaine s'est beaucoup développé ces dernières années grâce à la mise sur le marché de capteurs de profondeur très abordables et l'augmentation de la puissance de calcul des ordinateurs.

La suite de cette partie se déroule ainsi : tout d'abord, nous décrivons les principales technologies de capteurs de profondeur existantes (section 2.2). Ensuite, nous abordons différentes techniques de localisation dans l'espace (section 2.3), et de reconstruction de maillage 3D récentes (section 2.4). Enfin, nous passons en revue différentes techniques de segmentation et de classification des constituants d'un maillage 3D (section 2.5).

2.2 Capteurs de profondeur

Les capteurs de profondeur permettent de créer une image 3D de l'environnement physique dans lequel ils sont placés. Ces capteurs génèrent soit un nuage de points 3D, soit une carte de profondeur (de laquelle on peut extraire un ensemble de positions 3D). Une carte de profondeur (voir la figure 2.2) est

1. *LIght Detection And Ranging*

une image dont chaque pixel représente un point physique de l'espace. Cette position physique se retrouve pour chaque pixel à partir de sa position (u, v) dans la carte de profondeur et de son intensité lumineuse.

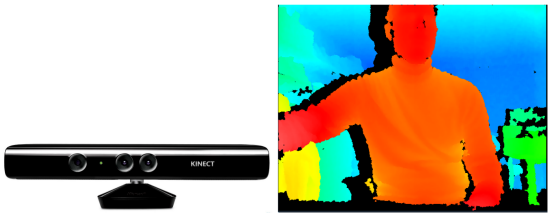


Figure 2.2 – *Capteur Kinect de Microsoft (à gauche), et exemple de carte de profondeur (à droite). Les couleurs chaudes correspondent à des pixels proches du capteur.*

Il existe deux principales technologies pour les capteurs de profondeur intégrés sur tablettes : les capteurs temps de vol (*Time Of Flight* (TOF)), et les capteurs stéréoscopiques. Les premiers se basent sur l'émission de rayons laser pour déterminer la position d'un point physique, tandis que les seconds utilisent les images couleur.

2.2.1 Capteurs temps de vol

Les capteurs temps de vol [153, 78, 49] utilisent des rayons infrarouges pour détecter la positions des objets dans un environnement physique. Ces rayons sont émis dans la scène et se réfléchissent sur les surfaces visibles. Un récepteur récupère les rayons réfléchis, et estime les distances des points d'impact à partir du temps de parcours de faisceaux.

Certains capteurs temps de vol n'envoient qu'un rayon lumineux à la fois. Dans ce cas, la génération d'une carte de profondeur ou d'un nuage de points se fait ligne par ligne. C'est le cas des capteurs de type LIDAR par exemple. Le nuage de points reconstitué est en général de très bonne qualité, et il n'y a pas de limite de distance. Ces capteurs sont donc très utilisés pour la reconstruction 3D à très grande échelle.

D'autres capteurs émettent plusieurs rayons lumineux à la fois. Cela permet de reconstituer très vite une carte de profondeur, offrant la possibilité d'une capture en temps réel. Cependant ces capteurs ont une portée et une précision beaucoup plus limitée par rapport aux capteurs de type LIDAR.

Pour résumer, les principaux avantages des capteurs temps de vol sont :

Avantages

- L'utilisation de plusieurs rayons à la fois permet de générer une carte de profondeur très rapidement.
- Les capteurs TOF sont utilisables sur des portées de quelques mètres avec une très bonne résolution.
- Ces capteurs sont accessibles au grand public et s'avèrent peu coûteux.

Il faut cependant relever certaines limitations à l'usage de capteurs temps de vol :

Limitations

- Les capteurs TOF à lumière structurée ne sont pas assez précis pour effectuer une capture d'une pièce de dimensions trop importantes.
- Certains matériaux sont absorbants ou transparents aux infrarouges. Ces matériaux ne sont pas détectés par un capteur TOF.
- Lors des déplacements, les rayons reçus à un instant t ne correspondent pas à la vraie surface en raison du temps mis par les rayons pour être réfléchis. Cela entraîne une forte dégradation de la qualité de la carte de profondeur. Ce phénomène est appelé *motion blur*.

Les capteurs temps de vol émettant des motifs de rayons lumineux représentent un choix intéressant pour la reconstruction 3D d'environnements intérieurs de faibles dimensions. Ils sont capables de fournir une carte de profondeur en temps réel, avec une précision suffisamment élevée pour permettre la génération de maillages de bonne qualité. Cependant, il faut prendre en compte le fait qu'il y ait des trous dans la carte de profondeur là où les rayons n'ont pas pu être réfléchis, et prendre en compte le *motion blur*.

2.2.2 Capteurs stéréoscopiques

Les capteurs stéréoscopiques [72, 153] utilisent plusieurs images d'un même objet, prises de plusieurs points de vue différents. Cela permet d'obtenir des informations sur la profondeur en appariant ces images entre elles. C'est de cette manière que fonctionne l'œil humain pour détecter du relief. Étant donné que ces capteurs ne discrétisent pas l'espace, les cartes de profondeur créées sont de bien meilleure résolution qu'avec un capteur TOF à lumière structurée. Cependant, la stéréoscopie nécessite d'effectuer plus de calculs pour générer une carte de profondeur par rapport à des capteurs à lumière structurée (calibrage, rectification, extraction de primitives visuelles dans chaque image, mise en correspondance).

Les principaux avantages des capteurs stéréoscopiques sont :

Avantages

- La résolution de la carte de profondeur est liée à la résolution des caméras. Cela permet des reconstructions 3D très denses par rapport aux capteurs temps de vol.
- Contrairement aux capteurs TOF, l'ensemble des composants d'une scène 3D peuvent être détectés.

Il y a cependant quelques limitations à l'utilisation de capteurs stéréoscopiques :

Limitations

- La stéréoscopie n'est efficace que pour des scènes texturées. En effet, il doit y avoir suffisamment de points remarquables porur pouvoir les mettre en correspondance avec ceux d'une deuxième image sans risque d'erreurs.
- Alors que le calcul d'une carte de profondeur est immédiat pour les capteurs temps de vol, il s'avère très lourd dns le cas de l'appariement stéréoscopique.

Les capteurs stéréoscopiques, comparés aux capteurs TOF, permettent une reconstruction de meilleure qualité, ont moins de limitation de portée. Ils peuvent représenter une très bonne option pour une reconstruction 3D en temps réel, si la fréquence de création des cartes de profondeur est suffisante. Leur principal inconvénient est qu'ils nécessitent d'être utilisés dans des scènes bien texturées pour être efficaces. Cela peut limiter leur usage à l'intérieur d'un bâtiment, dont les murs présentent souvent de grandes zones uniformes.

2.2.3 Utilisation de capteurs de profondeur sur tablette

Depuis peu, des capteurs de profondeur sont accessibles au grand public, le plus connu étant le capteur Microsoft Kinect. Des capteurs de profondeur sont également disponibles sur tablettes, comme par exemple le StructureSensor [2] ou le ZED sensor [3]. Ils fournissent une carte de profondeur en temps réel, et peuvent être utilisés pour la reconstruction 3D sur tablette. Récemment, le projet Tango de Google a aussi contribué à la mise sur le marché de tablettes intégrant des capteurs de profondeur (voir sur la figure 2.3).

2. https ://structure.io/
3. https ://www.stereolabs.com/zed/

Figure 2.3 – *Les trois tablettes et smartphones du projet Tango. Ces trois dispositifs sont équipés d'un capteur de profondeur, et intègrent le SDK du projet Tango, qui offre notamment une API pour l'odométrie.*

Bien que le projet Tango ait été abandonné, son successeur, le projet ARCore [4] permet l'utilisation de fonctionnalités avancées de suivi de mouvements, et peut être couplé avec d'autres capteurs de profondeur existants.

2.3 Mise en correspondance de données RGB-D

Puisque notre problème vise à réaliser une reconstruction 3D à partir du déplacement d'un capteur de profondeur dans l'espace physique, il peut être vu comme un sous problème du SLAM. Bien que des solutions théoriques au SLAM existent, il faut les adapter au contexte d'utilisation et aux données disponibles. Dans le cas de la reconstruction 3D, il est nécessaire de pouvoir aligner des données RGB-D successives afin de générer un modèle 3D.

Nous détaillons ici différentes techniques d'alignement de données RGB-D, ainsi que des techniques de correction des dérives classiques.

2.3.1 Alignement d'images

Les premiers algorithmes d'alignement concernent les images couleur uniquement. Ils s'étendent facilement au cas de l'utilisation d'une carte de profondeur en remplaçant l'intensité de l'image par l'intensité de la carte de profondeur.

Alors que les capteurs physiques sur un système autonome permettent d'estimer son positionnement théorique, l'odométrie visuelle permet d'estimer ce positionnement à partir d'observations de l'environnement réel, en comparant plusieurs prises de vue différentes. Dans le cas où le monde physique est connu à l'avance, l'odométrie peut se faire de manière globale. Il suffit de comparer des points remarquables dans une image prise à un instant t à l'ensemble des points remarquables qui constituent le monde physique. Dans le cas contraire, il faut comparer deux à deux les images capturées à chaque instant t et $t+1$ et identifier des zones identiques pour estimer le déplacement physique du capteur

4. https ://developers.google.com/ar/discover/

entre t et $t+1$. Nous nous concentrons ici sur les méthodes d'odométrie visuelle incrémentales, qui apparient des images couleur successives.

La plupart des algorithmes d'alignement utilisent des **descripteurs de points d'intérêt** pour estimer le déplacement. Ces descripteurs sont calculés à partir des données RGB et contiennent des informations sur le voisinage local de chaque point. Ils sont en général invariants par translation ou rotation, ce qui permet d'associer à un point physique un même descripteur sur deux prises de vue différentes. En général, on ne calcule pas de descripteurs sur l'image entière : une étape de détection permet d'identifier les points clés dans l'image pour lesquels il est pertinent d'en calculer. Ces points clés doivent présenter des singularités qui permettent de les identifier facilement, même après une rotation de ou une mise à l'échelle de l'image. Un descripteur est alors calculé pour ces points, il s'agit d'un ensemble de valeurs numériques encodant leurs particularités physiques.

2.3.1.1 Recherche de points d'intérêt

Les points d'intérêt sont des points de l'image présentant des caractéristiques particulières qui les rendent identifiables sur des prises de vues successives. Ainsi, les calculs sont restreints à quelques points saillants ayant une fiabilité d'appariement élevée, au lieu de réaliser une mise en correspondance dense et sujette à ambiguïtés.

Il existe plusieurs manières de calculer des points d'intérêt. La plupart du temps, ces points se situent sur les contours de l'image, c'est-à-dire les zones où le gradient de couleur ou de profondeur présente un extremum local. Ces contours peuvent être calculés à l'aide d'un filtre de type Canny [13], ou grâce à l'opérateur de Sobel [123].

Il est également possible de détecter les coins localisés sur les contours de l'image, réduisant le nombre de points clés et le risque d'ambiguïtés sur des zones homogènes. On peut citer par exemple le detecteur de Harris [50] ou le détecteur SUSAN [5] [122]. Rosten et al. [109] proposent le détecteur FAST [6], qui permet une détection de points clés significativement plus rapide que les détecteurs Harris et SUSAN.

D'autres méthodes permettent de détecter des points d'intérêt de manière plus générale, comme le laplacien des Gaussiennes, la différence des Gaussiennes ou le calcul du déterminant de la matrice Hessienne [88, 80]. Ces points clés s'avèrent invariants aux facteurs d'échelle, et au niveau de flou dans l'image.

5. *Smallest Univalue Segment Assimilating Nucleus*
6. *Features from Accelerated Segmentation Test*

2.3.1.2 *Descripteurs d'images*

Parmi les descripteurs les plus utilisés on peut citer le descripteur SIFT [7] [83]. Ce descripteur est calculé en deux étapes. Tout d'abord, des points clés sont détectés dans l'image à partir des différences de gaussiennes calculées sur l'image originale pour des écarts types différents. Une fois ces points clés trouvés, on calcule l'histogramme des gradients de cette image au voisinage de ces points. Ce descripteur est invariant par rotation et mise à l'échelle d'image.

Ke et al. [67] proposent une amélioration des descripteurs SIFT, qu'ils nomment PCA-SIFT [8]. Pour cela, l'image est subdivisée en régions pour lesquelles une analyse des composantes principales est appliquée pour trouver les deux vecteurs gradients dominants. Pour chaque point d'intérêt trouvé, le gradient associé est projeté dans cet espace pour donner un vecteur unique, qui correspond à un descripteur de dimension moins élevée qu'un descripteur SIFT classique. Cette méthode permet des correspondances plus précises entre points clés, et les descripteurs PCA-SIFT sont plus rapides à calculer.

Plusieurs travaux proposent des optimisations et approximations permettant d'optimiser le calcul des descripteurs SIFT [150, 39], faisant de ce descripteur l'un des plus utilisés.

Une amélioration en termes de performances des descripteurs SIFT est le descripteur SURF [9] [7]. Les points clés sont déterminés en calculant d'abord la matrice hessienne pour chacun des points de l'image originale. Les points dont le déterminant de la matrice hessienne dépasse un certain seuil sont retenus. Cette étape de calcul est très rapide grâce à l'utilisation d'images intégrales [21]. Ensuite, le descripteur est calculé à partir de la somme des réponses aux caractéristiques pseudo-Haar [141] calculées sur le voisinage de chaque point d'intérêt. Cette somme est pondérée avec des coefficients gaussiens. Cette étape fait également appel aux images intégrales, ce qui réduit fortement les temps de calcul.

D'autres travaux [86, 77] se basent sur le détecteur FAST et proposent des descripteurs pouvant parfois égaler le descripteur SURF en termes rapidité.

Cette liste de descripteurs de points d'intérêt est loin d'être exhaustive, et l'objectif des travaux de thèse ne nécessite pas d'en faire une liste complète. Néanmoins, le lecteur pourra se référer à d'autres travaux comme [90] ou [91] pour plus d'informations.

7. *Scale Invariant Feature transform*
8. *Principal Components Analysis SIFT*
9. *Speed up Robuste Features*

2.3.1.3 Estimation d'une transformation

Nister et al. [98], présentent des algorithmes génériques permettant de calculer une matrice de transformation à partir de deux images prises à deux endroits différents. Tout d'abord, il faut identifier des descripteurs dans chacune des images. Ensuite, il existe des algorithmes permettant de calculer une transformation 3D en fonction du type des descripteurs. Makadia et al. [87] proposent une méthode d'alignement d'objets 3D sans besoin de calculer de descripteurs particuliers, ou d'avoir un chevauchement quelconque des images 3D. Aussi, les prises de vue peuvent être complètement distinctes. Cette méthode d'alignement repose sur l'utilisation des images gaussiennes étendues *Extended Gaussian Images (EGI)* [57]. Les EGI consistent à placer les vecteurs normaux d'un objet 3D à l'intérieur d'une sphère. L'origine des vecteurs normaux se trouve au centre de cette sphère, et leurs sommets pointent sur un point de la sphère dépendant de l'orientation de la surface. C'est cette sphère que l'on nomme image gaussienne. Dans le cas d'un solide à faces, on affecte aux normales de chaque face un poids correspondant à la surface de cette face, cela constitue l'image gaussienne étendue. Dans le cas d'un objet 3D plus général, on peut diviser la sphère en régions et créer des histogrammes pour les vecteurs normaux.

Pour aligner les points, les auteurs utilisent ces histogrammes et cherchent une matrice de rotation qui permet de les aligner. Une fois cette matrice de rotation obtenue, les auteurs cherchent une translation qui maximise le chevauchement entre les deux objets.

2.3.2 Alignement de nuages de points

Nous nous intéressons ici à l'alignement de nuages de points 3D, supposant que les points sont issus d'un capteur de profondeur. Ils représentent donc une surface 3D, et il est alors possible de calculer les vecteurs normaux ou d'estimer la courbure de la surface en chaque point. Le couplage avec des informations RGB permet le calcul de descripteurs plus précis. Le processus d'alignement reste alors similaire à celui d'images 2D, à savoir que des points d'intérêt sont identifiés et des descripteurs 3D sont calculés pour chacun de ces points d'intérêt. Une transformation rigide est alors calculée pour aligner ces nuages de points.

Une grande partie des algorithmes présentés ici est implémentée dans la bibliothèque de manipulation de nuages de points PCL [111].

Certains détecteurs de points clés, comme le détecteur de Harris [121] ont été adaptés pour être utilisés sur des données nuages de points. De la même manière, la plupart des descripteurs 2D ont été étendus à la 3D [117, 103].

D'autres descripteurs [44] sont plus spécifiques aux nuages de points représentant une surface 3D. Ils décrivent la géométrie locale en chaque point.

Flint et al. s'inspirent des descripteurs SIFT et SURF pour créer les descripteurs THIRFT [35] qui utilisent les vecteurs normaux à la carte de profondeur, calculée à plusieurs niveaux d'échelle différents. En effet, étant orthogonaux au gradient, les vecteurs normaux dépendent directement de celui-ci. Dans le cas d'images RGB-D venant d'un capteur de profondeur, la densité spatiale des points dépend de leur position par rapport au centre optique du capteur. L'utilisation des vecteurs normaux fournit donc une estimation plus robuste des descripteurs par rapport au calcul d'un gradient dépendant des positions spatiales des points.

Zhong et al. [151] proposent les descripteurs ISS [10]. Ces descripteurs sont calculés en deux étapes. Tout d'abord, une base de vecteurs propres est calculée au voisinage de chaque point $\boldsymbol{p}$ à partir de la répartition du voisinage. Ensuite, dans cette base, centrée sur $\boldsymbol{p}$, un histogramme des orientations des vecteurs $\boldsymbol{p} - \boldsymbol{p_i}$, pour $\boldsymbol{p_i}$ appartenant au voisinage de $\boldsymbol{p}$ est créé. Les évaluations montrent que ces descripteurs sont très discriminants, et sont très efficaces pour aligner deux nuages de points.

Le descripteur 3D NARF [11] [125] est proposé par Steder et al. Il consiste à construire pour chaque point d'intérêt une carte de profondeur locale, obtenue en se plaçant dans la direction de la normale à ce point. À partir de cette carte locale, les auteurs calculent une orientation dominante dépendant de la distribution de l'intensité de cette carte de profondeur.

Rusu et al.[113, 110] proposent les descripteurs FPFH [12] pour caractériser un point 3D. Il s'agit de descripteurs encodant la géométrie du voisinage d'un point à l'aide d'histogrammes calculés à partir des positions géométriques et des normales des points du voisinage. Une liste plus complète de descripteurs pour des données 3D est présentée par Guo et al. [44].

2.3.3 Correction des dérives

Le couplage entre des capteurs physiques (rotation de roue, centrale inertielle, etc...) et l'utilisation de caméras RGB-D permet d'estimer de manière fiable les déplacements du capteur. Cependant, il subsiste souvent des erreurs de positions, qui peuvent être dues au bruit des différents capteurs, ou à des scènes peu texturées, pour lesquelles il est difficile d'extraire des points clés. Aussi, il est souvent nécessaire de mettre en place des algorithmes de correction des dérives. Contrairement aux algorithmes d'alignement précédents, les algorithmes de correction des dérives sont des algorithmes d'**alignement dense** qui fonctionnent uniquement entre deux prises de vue successives. Ils calculent une matrice de transformation qui minimise l'erreur de position entre ces deux ensembles de

10. *Intrinsic Shapes Signatures*
11. *Normal Aligned Radial Feature*
12. *Fast Point Feature Histograms*

données RGB-D.

Il existe deux principales approches pour l'alignement dense [85] : les algorithmes de type *Iterative Closest Point* (ICP) (décrites au §2.3.3.1), et l'utilisation de NDT *Normal Distribution Transform* (décrites au §2.3.3.2). Les algorithmes de type ICP ont un rayon de convergence plus faible que les algorithmes de type NDT, mais ils fournissent des résultats plus précis lors de la correction de faibles erreurs de positionnement.

2.3.3.1 Algorithmes de type ICP

L'algorithme ICP est originalement décrit par Besl et McKay [8]. Il consiste à aligner deux nuages de points 3D de manière itérative. Pour cela, à chaque itération, un ensemble de correspondances entre le nuage de points source et le nuage de points cible est identifié. Il s'agit de points supposés être identiques sur les deux nuages de points. Ensuite, une transformation minimisant la distance globale entre chacune des paires de correspondances est calculée. Le processus est ainsi répété jusqu'à convergence de l'algorithme. L'algorithme ICP est un algorithme qui converge localement, il n'est efficace lorsque l'amplitude de la transformation à estimer est faible (sur des données déjà presque alignées).

Le calcul des matrices de transformation est un problème d'optimisation mathématique, et peut être résolu en utilisant une méthode comme la décomposition en valeurs singulières [3]. En revanche, la phase de recherche de correspondances entre deux nuages de points $\mathcal{P}$ et $\mathcal{P}'$ est critique pour les performances de l'algorithme. En effet, pour chaque point $\boldsymbol{p} \in \mathcal{P}$, on va chercher un éventuel point correspondant parmi les points $\boldsymbol{p'} \in \mathcal{P}'$ situés au voisinage de $\boldsymbol{p}$. Cette recherche du voisinage peut être coûteuse en termes de temps puisqu'une recherche naïve a une complexité temporelle en $O(n^2)$.

Très souvent, on utilise des structures de données de type ***kd-tree*** [40, 99, 104] afin de réduire la complexité temporelle de la recherche de correspondances. Ces structures de données permettent de ranger les positions spatiales sous forme d'arbre binaire, réduisant la complexité moyenne de recherche de correspondances à $O(nlog(n))$ pour un arbre équilibré.

L'algorithme ICP peut se faire en minimisant les distances point à point entre les correspondances ou en minimisant les distances point à plan [16, 118]. Alors que l'utilisation de distances point à point est plus facile à mettre en œuvre, cette méthode peut produire des erreurs s'il n'y a pas de correspondance exacte entre les points, comme cela est le cas avec l'utilisation de surfaces discrétisées. Dans ce cas, l'utilisation des distances point à plan peut aboutir à de meilleurs résultats. Au lieu de calculer une distance entre un point source $\boldsymbol{p}$ et son point cible correspondant $\boldsymbol{p'}$, on calcule la distance entre $\boldsymbol{p}$ et le plan passant par $\boldsymbol{p'}$. Segal et al. [118] proposent une généralisation de l'algorithme ICP prenant en compte ces

deux aspects, tout en conservant la structure globale de l'algorithme. Pour cela, ils caractérisent la distribution du voisinage à chaque point comme étant une loi normale centrée sur les paramètres d'un plan, en supposant que la surface est plane localement.

Dryanovski et al.[26] proposent un algorithme d'alignement de nuages de points utilisant un algorithme ICP pour les deux étapes : alignement initial et correction des dérives. La correction des dérives se fait grâce à l'algorithme ICP généralisé. L'alignement initial se fait en calculant des descripteurs à partir des contours d'une carte de profondeur, calculés à l'aide d'un filtre de Sobel. Ensuite, l'angle θ du vecteur gradient en chaque point est calculé. Cet angle sert à rejeter des points correspondants si leurs gradients ne sont pas suffisamment alignés.

Henry et al. [51] présentent le *framework RGB-D Mapping*, dont le but est d'effectuer une reconstruction 3D à partir d'un capteur de profondeur. Dans ce papier, les auteurs présentent d'abord une variante de l'algorithme ICP qui combine les informations RGB et les informations de profondeur.

2.3.3.2 Algorithmes de type NDT

L'utilisation de *Normal Distribution Transforms* (NDT) a été introduite par Biber et al. [9] dans le cadre d'une application SLAM. Cette transformation consiste à projeter les points 3D sur un plan parallèle au sol, et à subdiviser ce plan selon une grille 2D. Pour chaque grille, la distribution des points 2D est calculée. L'alignement est ensuite effectué de manière itérative en minimisant la dissimilarité globale entre les points du nuage source et les NDT calculées pour le nuage cible. Cette approche permet d'éviter de passer par une recherche des correspondances, comme avec l'algorithme ICP, qui est une grande source d'erreur. L'utilisation de NDT a aussi montré une plus grande vallée de convergence que pour l'ICP [108, 127].

En raison de sa simplicité à mettre en œuvre et les différentes optimisations qu'il est possible de mettre en place, l'algorithme ICP est souvent l'algorithme de référence pour l'alignement final de nuages de points, une fois qu'une estimation initiale de la transformation et a été calculée. De ce fait, il est déjà implémenté dans plusieurs bibliothèques de calcul [104, 112, 54].

2.4 Reconstruction de maillages 3D en temps réel

L'arrivée sur le marché des capteurs de profondeur grand public, notamment le capteur Kinect, est à l'origine de nombreux travaux sur la reconstruction 3D, et en particulier la reconstruction de maillages en temps réel avec l'arrivée des GPU.

Ces nouveaux algorithmes reposent sur l'utilisation d'**approches volumétriques** pour fusionner les données des capteurs dans un modèle 3D global.

Les approches volumétriques se basent sur les travaux de Curless et al. [22], et d'Elfes et al. [30]. Dans une approche volumétrique, l'espace est subdivisé en voxels avec une résolution spatiale fixe. Pour chaque voxel, on calcule la distance de son centre à la surface la plus proche. Si le centre du voxel est placé avant la surface, cette distance est prise positive, sinon, elle est prise négative. On appelle cela une **distance signée**.

L'utilisation de **champs de distances signées** (*Signed Distance Fields* (SDF)) permet de représenter une surface de manière implicite, et permet une intégration plus facile d'un nouveau nuage de points à un modèle global en cours de construction. De plus, la taille mémoire du modèle 3D reconstruit dépend directement de son volume. Ces approches volumétriques on permis le développement d'algorithmes de reconstruction 3D en temps réel et à grande échelle.

2.4.1 Nouveaux algorithmes de reconstruction 3D

Newcombe et al. [61, 96] mettent en place l'algorithme *K*inect Fusion, qui a servi de base à la plupart des algorithmes de génération de maillages en temps réel. Cet algorithme permet de reconstruire un maillage 3D en temps réel d'un environnement intérieur à l'aide d'une caméra Microsoft Kinect [69, 144, 24, 68].

Kinect Fusion utilise un *pipeline* GPU afin de traiter l'intégralité des données des cartes de profondeur générées par la Kinect. L'algorithme met à jour un champ de distances signées tronquées après avoir déterminé le déplacement de la caméra en comparant les cartes de profondeur de deux états successifs. Si Kinect Fusion produit de très bons résultats, le volume maximal d'une pièce pouvant être reconstruit est limité à 7 m^3 seulement, ce volume est initialisé autour du point d'origine de la caméra.

Whelan et al. [145], proposent une extension de Kinect Fusion qui permet d'effectuer une reconstruction sans limitation de volume. Pour cela, un champ de distances signées est alloué autour de la position actuelle de la caméra. Ces distances signées sont mises à jour avec le même pipeline que Kinect Fusion. Lorsque le déplacement du centre de la caméra dépasse un certain seuil, le champ de distances signées est ré-alloué autour de cette nouvelle position. Le volume de reconstruction se déplace avec la caméra et le modèle global est stocké dans un maillage 3D qui s'incrémente au fur et à mesure.

Choi et al. [17] proposent un algorithme de génération de maillages 3D se basant sur une méthode volumétrique, mais avec une méthode d'alignement des points permettant d'obtenir un maillage final plus précis qu'avec d'autres outils de reconstruction basés sur Kinect Fusion.

2.4.2 Optimisation de structures de données

Afin de pallier aux limitations en termes de volume ou améliorer la rapidité de la reconstruction, un certain nombre de travaux ont été menés sur l'utilisation de structures de données adaptées. L'utilisation de structures de données de type **octree** est très courante lors de la manipulation de données 3D, car elle permet de les ordonner. De plus, ces structures s'implémentent très bien sur GPU [152, 149].

Steinbrucker et al. [126] proposent une méthode de reconstruction 3D à grande échelle fonctionnant en temps réel sur un ordinateur équipé d'un GPU. La reconstruction repose sur l'utilisation de distances signées. Les auteurs proposent une structure d'*octree* pour stocker les distances signées, permettant une gestion de la mémoire plus optimale qu'un algorithme de type Kinect Fusion.

Niessner et al. [97] introduisent le concept de **hachage spatial**. Plutôt que d'allouer de la mémoire de manière statique au début de l'exécution du programme, elle est allouée dynamiquement. Pour cela, l'espace est divisé en régions, pour lesquelles un identifiant unique est associé, en fonction de leurs coordonnées spatiales. Cet identifiant est alors utilisé comme clé de stockage dans une table de hachage. Cela permet à la fois d'économiser de la mémoire, et d'accéder à n'importe quelle région spatiale avec une complexité de $O(1)$, ce qui présente un net avantage par rapport à des structures de données statiques, ou de type *kd-tree* ou *octree*.

2.4.3 Reconstruction de maillages en temps réel sur tablettes

De nombreux de travaux portent sur la reconstruction 3D d'objets en utilisant un système mobile à l'aide de la caméra [131, 105, 101]. Ondruvska et al.[101] sont les premiers à introduire la reconstruction 3D à l'aide d'une approche volumétrique sur un dispositif mobile. L'arrivée récente de capteurs temps de vol portables permet à présent d'envisager une reconstruction 3D à plus grande échelle à l'aide d'une tablette.

CHISEL [71] est un algorithme de reconstruction 3D conçu pour fonctionner sur une tablette équipée d'un capteur de profondeur. Il suppose que la tablette dispose d'un algorithme d'odométrie permettant d'estimer les déplacements de la tablette. Il génère en temps réel un maillage 3D d'un environnement réel, comme l'illustre la figure 2.4.

L'algorithme CHISEL utilise des champs de distances signées tronquées (TSDF) [22] pour stocker les données 3D. L'utilisation de TSDF offre notamment une plus grande résistance au bruit du capteur. Ces données sont réparties en blocs 3D pouvant être alloués dynamiquement si nécessaire grâce à un mécanisme de hachage spatial [97] (voir §2.4.2).

CHISEL se démarque ainsi d'un algorithme comme Kinect Fusion dans le sens

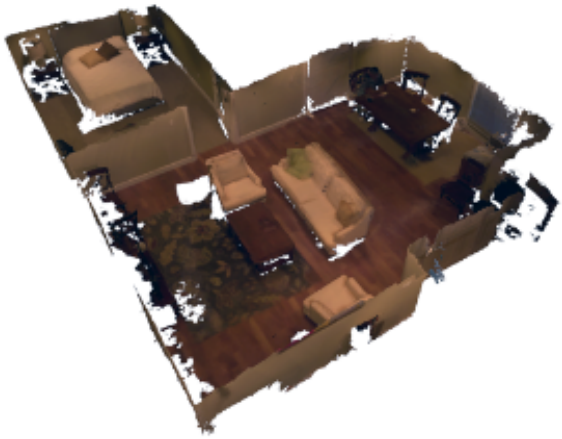


Figure 2.4 – *Reconstruction de maillages 3D avec CHISEL [71].*

où il a été conçu pour utiliser uniquement les capacités de calcul de la tablette, et où il permet d'effectuer une reconstruction 3D à grande échelle. L'inconvénient de l'algorithme CHISEL est qu'il est nécessaire de limiter la résolution spatiale, et donc la qualité de la reconstruction pour obtenir des performances satisfaisantes sur une tablette.

Lorsqu'un nouveau nuage de points est disponible, les TSDF associées sont mises à jour. Les blocs mémoire sont alloués ou supprimés dynamiquement si nécessaire. Le maillage final est réalisé grâce à l'algorithme *marching cubes*, détaillé dans l'annexe B [82].

Kahler et al. [66] proposent une amélioration de l'algorithme CHISEL en mettant en place un mécanisme de fermeture de boucle. La fermeture de boucle en SLAM consiste à reconnaître une région ayant déjà été observée au cours de la capture, et à se servir de cette information pour corriger l'ensemble de la trajectoire parcourue. Pour cela, des sous-volumes sont créés lors de la reconstruction, ce qui permet de les ré-aligner *a posteriori*.

Shops et al. [116] ont mis en place un algorithme de reconstruction 3D à large échelle, utilisable en intérieur comme en extérieur. L'algorithme a été implémenté sur une tablette de type Google Tango, et utilise le flux vidéo de la tablette pour générer des cartes de profondeur. L'algorithme utilise le GPU de la tablette pour faire les calculs. Han et al. [47] de basent sur CHISEL pour proposer un algorithme de reconstruction à large échelle utilisant uniquement les capacités de calcul d'un CPU, et optimisée de manière à pouvoir fonctionner sur une grande gamme de systèmes mobiles.

2.5 Segmentation et classification de données 3D

Nous nous intéressons ici à différentes techniques de segmentation et de classification de données 3D. Nous séparerons ces travaux en deux catégories [41] : la segmentation 3D et la classification d'objets. La segmentation 3D consiste à diviser un ensemble de données 3D en régions disposant de caractéristiques communes, tandis que la classification consiste à associer des classes à différents éléments d'un maillage 3D. Ces approches peuvent être complémentaires, puisque la segmentation 3D est souvent la première étape pour une classification des éléments.

2.5.1 Segmentation 3D

Les algorithmes de type *RANdom SAmple Consensus* (RANSAC) [115] permettent de détecter des formes géométriques simples dans un modèle 3D. Ils sont assez simples à mettre en œuvre et peuvent facilement traiter des nuages de points possédant jusqu'à 50% de valeurs aberrantes. Cependant ce sont des algorithmes statistiques qui peuvent fournir des résultats erronés s'ils sont mal paramétrés.

De nombreux travaux portent sur la segmentation planaire de données 3D [133]. En effet, les zones planes sont très présentes dans les environnements construits par l'homme, et de même, toute surface 3D peut s'approximer localement par un plan. La segmentation planaire permet de faire une première subdivision des données est sert de base à un grand nombre d'algorithmes de classification. En particulier, on retrouve des travaux portant sur l'utilisation d'algorithmes de type RANSAC [132] appliqués à la détection de surfaces planes. Pour ce faire, l'algorithme recherche itérativement le plus grand plan parmi les points qui n'ont pas encore été étiquetés.

Borrmann et al. [11] généralisent la transformée de Hough [27, 139] afin de détecter des plans 3D. Cependant la transformée de Hough est très coûteuse en ressources de calcul, et est limitée en précision. Bien que pouvant être source d'erreurs, les algorithmes de type RANSAC restent en général plus précis et plus rapides que la généralisation 3D des transformées de Hough [132].

Holz et al. [53] effectuent une segmentation planaire d'un nuage de points en effectuant d'abord une segmentation 3D de l'espace des normales, avant de séparer les plans parallèles. Cette technique est utilisée pour détecter des obstacles et des objets préhensibles dans une scène 3D, avec une très grande fiabilité (100% des obstacles, et jusqu'à 90% des objets préhensibles détectés). Cependant, les temps de calcul deviennent très vite rédhibitoires pour des grandes scènes.

Pour accélérer les algorithmes de segmentation, plusieurs méthodes réutilisent l'algorithme *superpixels* [36]. L'algorithme *superpixels* est un

algorithme de segmentation 2D qui consiste à diviser une image en sous-espaces et à effectuer la segmentation localement avant de fusionner les clusters obtenus.

Parmi ces méthodes, on peut citer les travaux d'Erdogan et al.[32]. Dans ces travaux, les auteurs segmentent les cartes de profondeur générées par un capteur Kinect, en utilisant l'algorithme *superpixels*. Les clusters obtenus sont ensuite fusionnés en utilisant une version généralisée de l'échantillonneur de Swendsen-Wang [6, 130]. Le résultat est visible sur la figure 2.5.

Figure 2.5 – *Segmentation planaire en utilisant la méthode de Erdogan et al.[32].*

Papon et al. ont mis en place l'algorithme *Voxel Cloud Connectivity Segmentation* (VCCS) [102], qui représente une extension des algorithmes de type *superpixels* [36] à des données RGB-D. Il s'agit d'un algorithme de segmentation 3D générique qui consiste à partitionner un nuage de points en une multitude de petites régions construites autours de points de référence. Chaque région contient des points supposés être similaires au point de référence qui a servi à construire cette région. La similarité Δ entre deux points 3D est déterminée par :

$$\Delta = \sqrt{\omega_c \delta_c^2 + \frac{\omega_p \delta_p^2}{3R_{seed}^2} + \omega_n \delta_n^2} \tag{2.1}$$

δ_c, δ_p et δ_n représentent les distances euclidiennes respectivement pour les couleurs, les positions, et les normales des deux points considérés. R_{seed} représente le rayon maximum d'un cluster, et ω_c, ω_p, ω_n sont des pondérations associées à chacune de ces distances. L'ajustement de ces pondérations permet de déterminer le type de segmentation à effectuer. Un exemple de l'utilisation de l'algorithme VCCS est présenté sur la figure 2.6.

La segmentation d'une scène 3D peut aussi se faire grâce à des méthodes de clustering [41]. Les méthodes de clustering permettent d'identifier des régions de différentes densités et de regrouper ensemble les éléments de ces régions. L'un

13. http ://pointclouds.org/

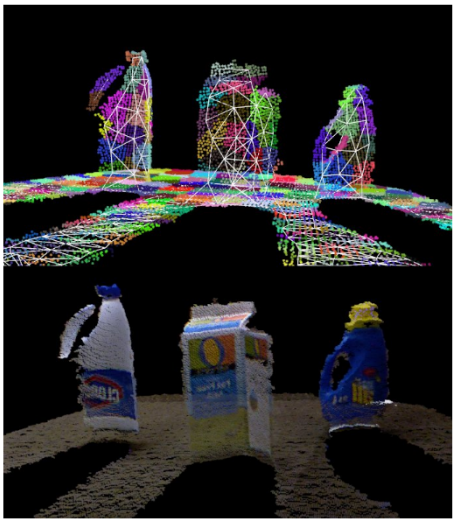


Figure 2.6 – *Exemple de segmentation d'un nuage de points en utilisant l'algorithme VCCS [102] (image issue de la documentation de la bibliothèque PCL)* [13].

des algorithmes les plus populaires pour le clustering est l'algorithme *k-means* [62, 14].

Le fonctionnement de l'algorithme *k-means* est assez simple. Tout d'abord, pour k classes à déterminer, on choisit k points du jeu de données, qui servent de centres pour chacun des k clusters. Le reste des points du jeu de données sont étiquetés comme appartenant au cluster dont le centre est le plus proche. Les centres des clusters sont alors mis à jour en prenant la moyenne de tous les points les constituants. L'algorithme se répète jusqu'à convergence. Les algorithmes de type *k-means* nécessitent de connaître à l'avance le nombre de classes à associer aux données à segmenter.

D'autres algorithmes comme DBSCAN [14] sont plus génériques, et permettent de regrouper des régions similaires par rapport à un critère de densité [33]. L'algorithme DBSCAN regroupe ensemble plusieurs point d'une région spatiale si leur densité est supérieure à un seuil. Cependant, ce seuil peut varier selon les données.

2.5.2 Classification d'objets

Les algorithmes de segmentation servent de base pour ensuite classifier les différents éléments d'un ensemble de données 3D. La segmentation planaire en particulier est très utilisée puisqu'elle permet d'identifier rapidement des éléments structuraux dans un nuage de points comprenant des bâtiments. Elle peut être la base d'un algorithme de classification, comme pour les travaux de Verma et al. [140], ou alors, permet de soustraire ces éléments afin de d'identifier les constituants d'une scène 3D d'environnement intérieur [25].

14. *Density-Based Spatial Clustering of Applications with Noise*

Beaucoup d'algorithmes entraînent des *classifiers* pour étiqueter les clusters obtenus [107, 45, 73]. Un premier algorithme de segmentation 3D sépare les données en sous-clusters, des descripteurs sont calculés classifiés. Sliberman et al. [120] par exemple, proposent un algorithme de segmentation de données RGB-D de prises de vues intérieures en se basant sur une première segmentation planaire de la scène avec une algorithme de type RANSAC. Ensuite, cette segmentation est utilisée comme entrée d'un algorithme de classification qui va regrouper les sous-régions entre elles en fonction de leur appartenance à une certaine classe d'objets.

2.6 Conclusion

La reconstruction 3D est un domaine très large, abordé dès les débuts de la robotique comme solution au SLAM. On trouve dès lors beaucoup de travaux proposant des solutions de cartographie d'un environnement réel à l'aide d'un robot autonome, ainsi que des solutions robustes au SLAM. En particulier, de nombreux algorithmes d'odométrie ont été mis en place et sont intégrés dans des bibliothèques, ou dans les nouveaux *frameworks* de réalité augmentée. Nous nous baserons sur des algorithmes existants et traiterons cette problématique comme une boîte noire.

Depuis quelques années, des capteurs de profondeur très abordables sont disponibles sur le marché, et ont popularisé l'accès aux données RGB-D. Avec l'augmentation des capacités de calcul des ordinateurs, de nombreux travaux récents se sont penchés sur la reconstruction de maillages 3D denses en temps réel en utilisant ces capteurs de profondeur bon marché. L'optimisation de ces algorithmes permet leur intégration sur des dispositifs mobiles comme des tablettes.

La popularisation des capteurs de profondeur a aussi permis d'importants travaux dans le cas de la segmentation 3D et de la classification. Ils est possible d'effectuer une segmentation planaire d'un ensemble de données RGB-D en des temps de calcul très courts, et d'identifier les différents constituants d'une scène 3D.

Nos travaux se focalisent sur l'utilisation de ces outils pour automatiser la génération de modèles 3D éditables à partir d'une capture 3D d'environnements intérieurs sur tablette. Lors de ces travaux de thèse, nous avons étudié deux approches de reconstruction 3D. La première approche exploite les récentes avancées en termes de génération de maillages 3D pour proposer un outil permettant de reconstruire un maillage d'un environnement intérieur en temps réel à l'aide d'une tablette, puis d'identifier la structure du bâtiment (sol, plafond, murs, et ouvrants). Nous avons ensuite envisagé une seconde approche

proposant une reconstruction d'un modèle 3D éditable d'un bâtiment à la volée, en se basant uniquement sur des algorithmes de segmentation 3D.

Chapitre 3

Reconstruction 3D d'un environnement intérieur et export de ses propriétés géométriques

3.1 Introduction . . . 58
3.2 Travaux connexes . . . 59
3.2.1 Identification de la structure d'une scène 3D . . . 59
3.2.2 Conclusion . . . 61
3.3 Génération d'un maillage 3D en temps réel . . . 61
3.3.1 Vue d'ensemble de l'algorithme . . . 61
3.3.2 Données manipulées . . . 62
3.3.3 Alignement des points . . . 64
3.3.4 Intégration d'un nouveau nuage de points . . . 66
3.3.5 Mise à jour des maillages . . . 66
3.3.6 Paramètres . . . 67
3.4 Identification de la structure du bâtiment reconstruit . . . 67
3.4.1 Identification du sol et des murs . . . 68
3.4.2 Détection des ouvrants . . . 70
3.4.3 Paramètres . . . 71
3.5 Implémentation . . . 72
3.5.1 Répartition des tâches . . . 72
3.5.2 Acquisition des données . . . 73
3.5.3 Calculs . . . 73
3.6 Évaluations . . . 73
3.6.1 Performances . . . 74
3.6.2 Génération du modèle 3D . . . 75
3.6.3 Évaluation des performances énergétiques . . . 75
3.6.4 Évaluation à l'usage . . . 76
3.6.5 Discussion . . . 77
3.7 Conclusion . . . 78

3.1 Introduction

Dans ce chapitre, nous présentons la première approche de reconstruction 3D abordée pendant la thèse. Ces travaux ont consisté à la mise en place d'un ensemble d'algorithmes permettant la reconstruction en temps réel d'un maillage 3D d'un environnement intérieur, et l'identification de la structure du bâtiment reconstruit. L'identification de la structure du bâtiment concerne les murs, le sol et les ouvrants. Ces informations permettent d'obtenir les différentes dimensions et surfaces du bâtiment, d'en estimer ses performances énergétiques, et d'exporter une maquette numérique du bâtiment brut.

Alors que l'identification de la structure de l'environnement reconstruit peut se faire *a posteriori*, la génération du maillage 3D se fait en temps réel afin de donner un retour à l'utilisateur pendant qu'il réalise la capture 3D. De nombreux travaux traitent déjà de la reconstruction de surfaces à partir des données d'un capteur de profondeur, et proposent déjà des résultats intéressants [61, 96]. La grande nouveauté dans l'approche envisagée est de permettre cette reconstruction en temps réel en utilisant uniquement les capacités de calcul de la tablette.

De même, l'identification des constituants d'un bâtiment a fait l'objet de nombreux travaux. Il est difficile d'automatiser complètement cette tâche en raison notamment de la grande diversité des classes d'éléments, des informations manquantes lors de la reconstruction de la surface, et de la précision des capteurs utilisés. En ne cherchant à identifier que la structure du bâtiment reconstruit, il est possible d'émettre plusieurs hypothèses (*Manhattan World Assumption* [20], murs plans, etc ...) afin d'automatiser au maximum cette détection (voir au §1.4.1).

Nous avons mis en place un ensemble d'algorithmes permettant de modéliser un environnement intérieur en 3D, puis d'identifier les éléments structuraux (murs, sol, et ouvrants) de ce bâtiment. Ces algorithmes sont utilisables sur toute tablette équipée d'un capteur de profondeur et disposant d'un algorithme d'odométrie.

La modélisation procède en deux temps. Tout d'abord, l'utilisateur effectue une capture de l'environnement intérieur en se déplaçant dans le bâtiment avec la tablette. Lors de cette étape un maillage 3D de la scène capturée est progressivement généré. Nous avons fixé comme objectif d'intégrer chaque nouveau nuage de points du capteur dans un intervalle de 200 ms, afin de se conformer aux contraintes énoncées au §1.4.2. Une fois le maillage 3D terminé, la deuxième partie du processus de modélisation consiste à effectuer une segmentation planaire du maillage 3D et d'identifier les différents éléments structuraux de la scène 3D reconstruite. Ce processus ne nécessite aucune intervention de l'utilisateur et peut être effectué en arrière plan.

Les algorithmes mis en place pendant ces travaux permettent de générer une maquette numérique avec une précision allant jusqu'à 95% pour les dimensions calculées. Des évaluations avec des utilisateurs montrent que la modélisation d'un bâtiment par reconstruction 3D est bien plus appréciée et efficace qu'avec une mesure manuelle des différentes dimensions du bâtiment. Cependant, afin de permettre une génération du maillage en temps réel, la qualité du maillage a été réduite, et la détection des ouvrants produit de nombreux faux positifs.

Dans le reste de ce chapitre, nous dressons d'abord un aperçu des solution d'identification de la structure d'un bâtiment à partir d'un maillage 3D (section 3.2). Ensuite, nous détaillons les différents volets algorithmiques mis en place : la génération en temps réel d'un maillage 3D (section 3.3), puis l'identification de la structure du bâtiment (section 3.4). Puis, nous détaillons l'implémentation de notre algorithme sur une tablette cible (section 3.5). Enfin, nous présentons les résultats obtenus lors des différentes évaluations menées (section 3.6).

3.2 Travaux connexes

Nous détaillons ici des travaux précédents traitant de la détection de la structure (murs et ouvrants) d'un maillage 3D d'un bâtiment.

En partant du principe que dans la plupart des cas les murs sont des surfaces planes, identifier la structure d'un bâtiment à partir d'un maillage 3D correspond tout d'abord à identifier les plans contenus dans ce maillage. Ensuite, il suffit de déterminer quels plans correspondent à des murs. Nous allons nous concentrer ici sur les travaux effectués dans le cadre de l'identification de la structure d'un bâtiment. Le lecteur pourra se référer aux sections 2.4 et 2.5 pour plus d'informations sur la génération de maillages 3D en temps réel ou la segmentation planaire.

3.2.1 Identification de la structure d'une scène 3D

Lee et al. [76] décrivent une méthode pour identifier la structure d'une scène (murs, sol et plafonds) en utilisant uniquement des images RGB.

L'algorithme détecte les différentes lignes contenues dans l'image, et déduit de l'agencement de ces lignes la géométrie de la pièce (voir la figure 3.1). Pour cela, plusieurs hypothèses sont faites : le plafond et le sol sont de même forme, les murs sont alignés selon une grille de Manhattan, et la configuration du bâtiment ne peut suivre que certaines configurations définies. Cette méthode permet d'obtenir un modèle 3D acceptable pour 70% des images testées, les principaux échecs étant dus à des scènes trop encombrées, ou à une configuration du bâtiment qui ne correspondait à aucune des configurations connues.

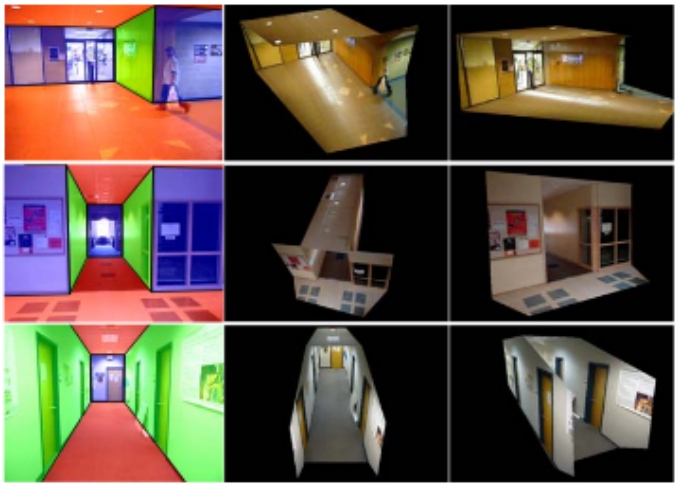


Figure 3.1 – *Détection de la structure d'un bâtiment à partir d'images RGB [76].*

Verma et al. [140] proposent un algorithme de segmentation 3D permettant de détecter des bâtiments à partir de données LIDAR, souvent obtenues à partir de prises de vues aériennes. Dans ce cas, seul le toit des bâtiments est visible. Tout d'abord, les zones planes sont identifiées, et les plans verticaux sont supprimés. En partant du principe que le sol constitue la zone plane restante la plus étendue, les auteurs en déduisent que les autres zones planes correspondent aux toits des bâtiments. Ensuite, la géométrie des différentes partie des toits est extraite en séparant les différents plans, et en trouvant leur forme à partir d'un modèle d'inférence de la forme d'un toit.

Adan et Huber [1] décrivent une méthode permettant d'extraire la structure d'un bâtiment dans un environnement encombré à partir d'un nuage de points. La détection des murs est facilitée par la projection du nuage de points sur un plan orthogonal à la verticale. Ainsi, les murs peuvent être simplement détectés à l'aide d'une transformée de Hough en 2D. De même, le sol et le plafond sont identifiés en projetant le nuage de points sur un plan vertical. Un algorithme de *raycasting* est ensuite appliqué sur chaque mur individuellement pour déterminer les zones vides, occupées ou masquées. Les ouvrants sont ensuite identifiés en utilisant des classificateurs SVM [1] [19]. Cette méthode permet de détecter des ouvrants avec une fiabilité allant jusqu'à 90%. Cependant, la précision de la détection peut être améliorée, la précision dans la mesure des dimensions des ouvrants était inférieure à 2.5 cm dans seulement 36% des cas.

De nombreux travaux portent aussi sur la génération semi-automatique de modèles BIM à partir de données issues de capteurs laser [95, 64, 84]. Jung et al. [64] par exemple, on développé un processus permettant de générer un modèle BIM à partir d'un nuage de points issu d'un capteur de type LIDAR. L'ensemble des plans contenus dans le nuage de points sont détectés en utilisant un

1. *Support Vector Machines*

algorithme de type RANSAC. La détection est ensuite affinée pour obtenir la meilleure précision possible pour les plans. Le BIM est généré en identifiant toutes les lignes d'intersection entres ces plans. Ces lignes sont ensuite exportées dans un logiciel de type CAO et servent de support pour générer un modèle BIM du bâtiment. Ce processus n'est dont pas complètement automatisé, mais accélère grandement la modélisation d'un bâtiment à large échelle.

3.2.2 Conclusion

Nous avons mis en évidence différents travaux portant sur la détection de la structure d'un bâtiment à partir de données d'un capteur temps de vol. Par rapport à ces travaux, il ressort qu'il est possible d'obtenir de bons résultats quant à la détection des principaux murs et ouvrants, à partir du moment où certaines hypothèses sont formulées. Notamment, il est souvent admis que les murs sont plans et orthogonaux entre eux. Une segmentation planaire du nuage de points 3D permet alors d'isoler les murs candidats, et d'envisager un export semi automatique d'un modèle BIM.

Les algorithmes de reconstruction 3D que nous présentons ici permettent de reconstruire en 3D un environnement intérieur existant en utilisant uniquement une tablette équipée d'un capteur de profondeur. Les informations récoltées permettent d'extraire suffisamment d'informations sur la géométrie du bâtiment pour en évaluer les performances énergétiques. Les parties suivantes décrivent en détail les algorithmes mis en place, qui s'appuient sur les travaux présentés précédemment.

3.3 Génération d'un maillage 3D en temps réel

Cette section décrit en détail l'algorithme de reconstruction 3D permettant de reconstruire un maillage en temps réel d'un environnement intérieur en utilisant uniquement une tablette équipée d'un capteur de profondeur.

3.3.1 Vue d'ensemble de l'algorithme

L'algorithme de génération de maillage utilisé est une ré-implémentation de l'algorithme CHISEL [71]. Plus précisément, il s'agit d'un portage d'une implémentation *open source* sur PC de cet algorithme, sur la tablette, avec l'ajout de nouvelles optimisations. L'algorithme CHISEL a été conçu pour pouvoir reconstruire un maillage 3D à grande échelle d'un bâtiment tout en en limitant l'utilisation des ressources matérielles. De plus, étant donné que l'objectif est d'identifier la structure du bâtiment, il n'est pas nécessaire d'avoir un très grand niveau de précision dans la qualité de la reconstruction.

La figure 3.2 montre les principales étapes de la reconstruction, qui seront détaillées par la suite. L'algorithme est composé de deux processus distincts : un processus de collecte et de mise en forme des données, et un processus de traitement de ces données. Ces deux processus s'exécutent en parallèle, et le processus de traitement des données est alimenté par les données issues du processus d'acquisition.

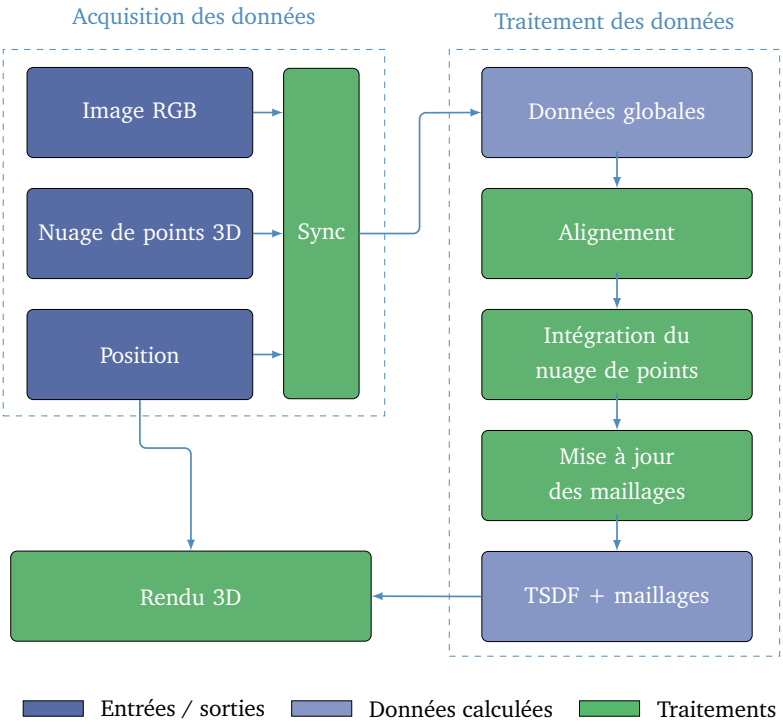


Figure 3.2 – *Vue d'ensemble de l'algorithme de génération du maillage 3D.*

3.3.2 Données manipulées

L'algorithme de maillage travaille avec des nuages de points 3D, qui sont calculés à partir de la carte de profondeur renvoyée par le capteur de profondeur. Afin de reconstituer un maillage 3D coloré, il faut également fournir une image vidéo associée à chaque carte de profondeur, pour associer à chaque point une couleur. Enfin, il est nécessaire de disposer d'un algorithme d'odométrie qui permet de donner à chaque instant une estimation précise de la position de la tablette (de faibles écarts peuvent être corrigés).

À partir de ces données, l'algorithme met à jour un champ de distances signées tronquées (TSDF). Soit $\mathcal{V}$ un volume de l'espace, délimité par la surface $\mathcal{S}$, et $\boldsymbol{p}$ un point de l'espace. On définit la distance signée de $\boldsymbol{p}$ à $\mathcal{S}$, $\phi(\boldsymbol{p}, \mathcal{S})$ par :

$$\phi(\boldsymbol{p}, \mathcal{S}) = \begin{cases} d(\boldsymbol{p}, \mathcal{S}) & \text{si } \mathbf{p} \in \mathcal{V} \\ -d(\boldsymbol{p}, \mathcal{S}) & \text{sinon} \end{cases} \tag{3.1}$$

Avec $d(\boldsymbol{p}, \mathcal{S}) = Min_{\boldsymbol{s}\in\mathcal{S}}(d(\boldsymbol{p}, \boldsymbol{s}))$, où d représente la distance euclidienne. Soit $\tau \in \mathbb{R}$, on peut alors définir la distance signée tronquée $\phi_\tau(\boldsymbol{p}, \mathcal{S})$ par :

$$\phi_\tau(\boldsymbol{p}, \mathcal{S}) = \begin{cases} \phi(\boldsymbol{p}, \mathcal{S}) & \text{si } \phi(\mathbf{p}, \mathcal{S}) \ \in [-\tau\ \tau] \\ \textit{Non défini} & \text{sinon} \end{cases} \tag{3.2}$$

La figure 3.3 illustre le concept de TSDF. Ce concept de TSDF a été évoqué par Curless et Levoy [22], et permet de représenter de manière implicite une surface, et son voisinage. La taille du voisinage τ est choisie en fonction du bruit du capteur de profondeur.

Dans le cas de la reconstruction 3D d'un bâtiment, la seule surface $\mathcal{S}$ considérée est le maillage à reconstruire. Nous simplifions donc la notation $\phi_\tau(\boldsymbol{p}, \mathcal{S})$ en $\phi_\tau(\boldsymbol{p})$, et pour tout voxel $V \in \mathcal{V}$, nous posons $\phi_\tau(V) = \phi_\tau(\boldsymbol{v})$, où $\boldsymbol{v}$ correspond aux coordonnées spatiales du centre de V.

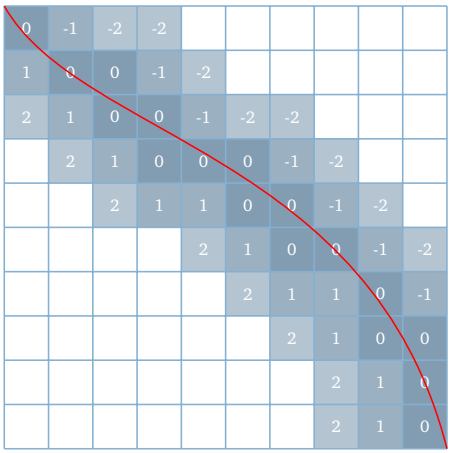


Figure 3.3 – *Exemple de distances tronquées signées sur un espace à deux dimensions. Chaque case représente un voxel 2D, et la ligne rouge la surface pour laquelle ont été calculées les distances signées.*

Afin de préserver la mémoire physique, l'allocation se fait dynamiquement. Pour cela, l'espace est divisé en régions 3D. Chaque région possède un identifiant unique dépendant des coordonnées géométriques de son centre. Cet identifiant est représenté sous la forme trois entiers, et est utilisé afin de calculer un indice de stockage dans une table de hachage [97]. Lors de l'intégration d'un nouveau nuage de points, on détermine l'ensemble des régions spatiales qu'il traverse. Il suffit ensuite d'ajouter une nouvelle entrée dans la table de hachage si une de ces régions est rencontrée pour la première fois.

L'utilisation d'une table de hachage permet une allocation dynamique de la mémoire, en utilisant uniquement pour les zones spatiales contenant des données intéressantes. Elle permet aussi de réduire la complexité temporelle d'accès en mémoire à une zone spatiale particulière.

Soient n_0, n_1 et n_2 trois grands nombres premiers, et $\boldsymbol{p} = [x, y, z]^\top$ un point de l'espace, à coordonnées entières. Les travaux de Niessner et al. [97] définissent une fonction de hachage spatial Ψ par :

$$\Psi(\boldsymbol{p}) = n_0 x \oplus n_1 y \oplus n_2 z \tag{3.3}$$

où $\oplus$ désigne l'opérateur ou exclusif.

Cette fonction de hachage permet d'associer à chaque région spatiale un index dans la table de hachage dépendant de son identifiant. L'usage des nombres premiers n_0, n_1 et n_2 permet de répartir uniformément les indices calculés, et remplir au maximum la table de hachage avec le moins de collisions possibles.

3.3.3 Alignement des points

L'estimation de la position globale de la tablette est faite grâce à un algorithme d'odométrie de type Visual Inertial Odometry [79]. Il subsiste cependant des erreurs de positionnement qui s'accumulent au fur et à mesure. Pour pallier à ce problème, nous avons mis en place un algorithme ICP [8], qui permet de corriger ces dérives entre deux captures.

L'algorithme ICP en lui-même est détaillé en annexe B.1, il s'agit d'un algorithme itératif dont l'objectif est d'aligner deux nuages de points supposés avoir des régions en commun. À chaque itération, un ensemble de points correspondants doit être déterminé, et une transformation affine minimisant la distance globale entre chaque couple de correspondances est estimée. Une fois le nuage de points à aligner modifié avec cette transformation, le processus est répété. Nous évoquons ici la recherche de points correspondants et de leur alignement propres à l'algorithme CHISEL.

3.3.3.1 Recherche de correspondances

Soit $\mathcal{P}$ un nuage de points à aligner avec notre modèle. Pour chaque point $\boldsymbol{p} \in \mathcal{P}$, nous allons chercher le point de la surface $\mathcal{S}$ définie par $\forall \boldsymbol{p'} \in \mathbb{R}^3, \phi_\tau(\boldsymbol{p'}) = 0$ qui est le plus proche de $\boldsymbol{p}$. Par définition, $|\phi_\tau(\boldsymbol{p})|$ correspond à la distance de ce point à $\mathcal{S}$. Pour connaître la direction dans laquelle se déplacer pour trouver le point **p'** associé sur $\mathcal{S}$, il suffit de calculer le gradient du TSDF au point $\boldsymbol{p}$. Si la distance entre $\boldsymbol{p}$ et $\boldsymbol{p'}$ est inférieure à un seuil ϵ, alors $\boldsymbol{p}$ et $\boldsymbol{p'}$ sont considérés comme étant correspondants.

Ce processus est détaillé dans l'algorithme 1 et permet de construire un ensemble $\mathcal{C}$ de points correspondants.

Algorithme 1 : *Recherche de correspondances pour CHISEL[71].*

$\mathcal{C} \leftarrow \emptyset$
for all $\boldsymbol{p} \in \mathcal{P}$ **do**
  $\boldsymbol{p_s} \leftarrow -\phi_\tau(\boldsymbol{p}).\boldsymbol{\nabla}_{\phi_\tau}(\boldsymbol{p}) + \boldsymbol{p}$
  if $d(\boldsymbol{p_s}, \boldsymbol{p}) \leq \epsilon$ **then**
    $\mathcal{C} \leftarrow \mathcal{C} + (\boldsymbol{p}, \boldsymbol{p_s})$
  end if
end for

3.3.3.2 *Alignement des correspondances*

Nous avons implémenté la méthode d'Arun et al.[3] (voir algorithme 2) pour aligner les correspondances. Cette méthode utilise la décomposition en valeurs singulières (SVD) pour trouver une transformation qui minimise la distance globale pour un ensemble de points correspondants donné. Pour cela, on calcule l'isobarycentre des points sources $\boldsymbol{\mu_s}$, et l'isobarycentre des points cibles $\boldsymbol{\mu_t}$. On soustrait $(\boldsymbol{\mu_s}, \boldsymbol{\mu_t})$ à chaque couple $(\boldsymbol{p_s}, \boldsymbol{p_t})$ de l'ensemble des correspondances $\mathcal{C}$, et on calcule la matrice de covariance $\boldsymbol{H}$ associée. On décompose la matrice $\boldsymbol{H}$ en valeurs singulières, et les matrices $\boldsymbol{U}$ et $\boldsymbol{V}$ associées à cette décomposition permettent de trouver la transformation $\boldsymbol{M}$ qui minimise la distance globale entre chaque couple $(\boldsymbol{p_s}, \boldsymbol{p_t})$ de correspondances.

Algorithme 2 : *Alignement de correspondances en utilisant les SVD [3].*

$\boldsymbol{H} \leftarrow \boldsymbol{O_3}(\mathbb{R})$
$\boldsymbol{\mu_s} \leftarrow \dfrac{\sum_{(\boldsymbol{p_s},\boldsymbol{p_t})\in\mathcal{C}} \boldsymbol{p_s}}{n}$
$\boldsymbol{\mu_t} \leftarrow \dfrac{\sum_{(\boldsymbol{p_s},\boldsymbol{p_t})\in\mathcal{C}} \boldsymbol{p_t}}{n}$
$\boldsymbol{H} \leftarrow \dfrac{1}{n}\sum_{(\boldsymbol{p_s},\boldsymbol{p_t})\in\mathcal{C}} (\boldsymbol{p_s} - \boldsymbol{\mu_s})(\boldsymbol{p_t} - \boldsymbol{\mu_t})^{\boldsymbol{T}}$
$(\boldsymbol{U}, \boldsymbol{V}) \leftarrow SVD(\boldsymbol{H})$
$\boldsymbol{R} \leftarrow \boldsymbol{V}\boldsymbol{U}^T$
if $det(\boldsymbol{R}) = -1$ **then**
  return $\boldsymbol{I_4}(\mathbb{R})$
else
  $\boldsymbol{T} \leftarrow \boldsymbol{\mu_t} - \boldsymbol{X}\boldsymbol{\mu_s}$
  $\boldsymbol{M} \leftarrow \begin{bmatrix} \boldsymbol{R}, \boldsymbol{T} \\ \boldsymbol{0_{1,3}}(\mathbb{R}), 1 \end{bmatrix}$
end if
return $\boldsymbol{M}$

3.3.4 Intégration d'un nouveau nuage de points

Tout d'abord, l'ensemble des régions spatiales qui seront modifiées est déterminé. Pour cela, un rayon virtuel est tracé entre le centre optique $\boldsymbol{O}$ du capteur de profondeur, et chacun des points $\boldsymbol{p}$ constituant le nuage de points $\mathcal{P}$ à intégrer. Par rapport à la distance de troncature τ, au delà de laquelle nous ne calculerons pas de distance signée, nous définissons le rayon $\zeta_{\boldsymbol{p}}$ par :

$$\zeta_{\boldsymbol{p}} = \left\{ \begin{array}{ll} \boldsymbol{p} + k\rho\boldsymbol{u}, & k \in \mathbb{Z} \\ & |k\rho| \leq \tau \end{array} \right\} \tag{3.4}$$

avec $\boldsymbol{u} = \dfrac{(\boldsymbol{p} - \boldsymbol{O})}{||\boldsymbol{p} - \boldsymbol{O}||}$, et où ρ représente la résolution spatiale de l'algorithme.

À partir des points de $\zeta_{\boldsymbol{p}}$ pour tout point $\boldsymbol{p}$ de $\mathcal{P}$, on retrouve l'ensemble des identifiants des régions spatiales qu'ils croisent en divisant leurs coordonnées spatiales par la largeur spatiale d'une de ces régions.

Une fois la liste des régions spatiales concernés par l'intégration de P construite, on peut leur allouer la mémoire nécessaire si c'est la première fois qu'on y fait référence.

Enfin, pour chaque point $\boldsymbol{p'}$ de $\zeta_{\boldsymbol{p}}$, $\boldsymbol{p} \in \mathcal{P}$, on met à jour la valeur de distance signée associée au voxel $V \in \mathcal{V}$ dont le centre est le plus proche de $\boldsymbol{p'}$. Cette mise à jour est réalisée de la manière suivante :

$$\Phi_\tau(V) = \frac{\omega_V . \Phi_\tau(V) + \omega_{\boldsymbol{p'}} . dist(\boldsymbol{p'}, \boldsymbol{p})}{\omega_V + \omega_{\boldsymbol{p'}}} \tag{3.5}$$

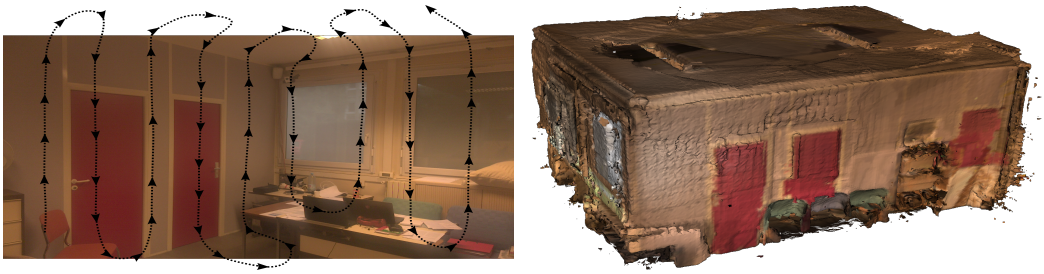
Bien que plusieurs fonctions de pondération peuvent être utilisées pour $\omega(\boldsymbol{p}')$, nous avons fait le choix d'utiliser une fonction de pondération constante égale à $\dfrac{1}{2\tau}$, comme pour l'algorithme original [71].

Afin d'améliorer les temps de calcul, on peut limiter la distance maximale D de l'origine $\boldsymbol{O}$ pour laquelle on observera un point $\boldsymbol{p}$. Cette distance étant de toute façon majorée par la portée maximale du capteur de profondeur (environ 5 m pour un capteur de type Kinect).

3.3.5 Mise à jour des maillages

Un maillage est construit pour chacune des différentes régions spatiales à l'aide de l'algorithme *Marching Cubes* [82]. Cet algorithme permet de générer un maillage 3D à partir d'un champ de scalaires, et notamment, à partir d'un champ de distances signées. Cet algorithme travaille avec un ensemble de cubes, dont les sommets sont les centres des voxels de la région spatiale considéré. Pour chaque configuration, un maillage partiel est déterminé en fonction de la configuration (signe des SDF) de chaque sommet.

Le détail de l'algorithme *Marching Cubes* est décrit en annexe B.2. Le résultat de la reconstruction 3D est visible sur la figure 3.4.



(a) *Capture de la pièce originale* **(b)** *Maillage 3D reconstruit*

Figure 3.4 – *Exemple de reconstruction de maillage avec notre algorithme.*

3.3.6 Paramètres

Le tableau 3.1 résume les paramètres utilisés pour l'algorithme de génération de maillages 3D, il s'agit des paramètres pour lesquels nous avons obtenus les meilleurs résultats lors des évaluations.

Étape	*Paramètre*	*Valeur*	*Commentaire*
CHISEL (§3.3.4)	τ	0.4	Distance de troncature (m)
	ω	1.25	Constante de pondération
	D	2.5	Portée max (m)
	ρ	0.3	Résolution spatiale (m)

Tableau 3.1 – *Paramètres pour l'algorithme CHISEL*

Pour l'algorithme CHISEL, nous avons utilisé une résolution spatiale de 3 cm, qui est largement suffisante pour la détection des plans contenus dans l'environnement intérieur dont on cherche à faire la capture. La portée maximale pour laquelle les calculs sont faits a été réduite à 2.5 m afin d'obtenir les meilleures performances de calcul.

3.4 Identification de la structure du bâtiment reconstruit

Afin d'automatiser la détection des murs et des ouvrants, nous avons fait les hypothèses suivantes :

- **Les murs sont plans et orthogonaux au sol :** nous supposons que la structure du bâtiment reconstruit est alignée sur une grille régulière (voir §1.4.1).
- **Les ouvrants sont rectangulaires :** afin de simplifier le processus de détection des ouvrants, nous supposons qu'ils sont de forme rectangulaire, ce qui est vrai dans la plupart des cas.

- **Les portes sont ouvertes :** la détection des ouvrants revient à chercher des ouvertues rectangulaires vides dans les maillages associés à des murs.

Grâce à ces hypothèses, nous sommes en mesure d'identifier la structure (sol, murs et ouvrants) du bâtiment reconstruit. Le processus d'identification de la structure, décrit à la figure 3.5, se fait en quatre étapes. Tout d'abord, un post-traitement est effectué sur le maillage 3D pour améliorer sa qualité. Ensuite, les plans principaux contenus dans le maillage sont détectés. Enfin, les murs sont identifiés à partir de ces plans, puis les ouvrants.

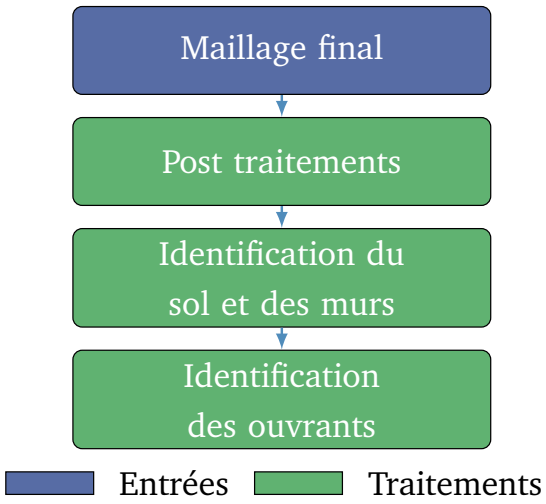


Figure 3.5 – *Les différentes étapes d'identification de la structure du bâtiment.*

À la différence de la reconstruction du maillage 3D, ce processus peut s'effectuer en arrière plan, et il n'y a pas de contrainte de performance en temps réel.

3.4.1 Identification du sol et des murs

Nous commençons par identifier les principaux plans contenus dans le maillage 3D. Ensuite, ces plans sont identifiés comme étant le sol, le plafond, des murs, ou du bruit.

3.4.1.1 Post traitements

Avant de procéder à la détection des plans, deux traitements sont effectués sur le maillage 3D. Tout d'abord, la densité spatiale du maillage est réduite à 5 cm, afin de diminuer le temps de traitement. Ensuite, un filtre statistique est appliqué sur le maillage 3D afin d'éliminer d'éventuels *outliers*. Pour chaque point $\boldsymbol{p}$, ce filtre calcule la position moyenne $\bar{\boldsymbol{p}}$ de ses k plus proches voisins (ensemble noté Ω). Si la distance entre $\boldsymbol{p}$ et $\bar{\boldsymbol{p}}$ est supérieur à l'écart type des distances entre chaque

point de Ω et $\bar{\boldsymbol{p}}$, alors $\boldsymbol{p}$ est considéré comme *outlier*. Nous avons pris $k = 20$ pour notre algorithme.

3.4.1.2 Détection des plans

Nous avons envisagé deux approches de segmentation 3D. Tout d'abord, nous avons envisagé d'utiliser un algorithme de type RANSAC de manière itérative sur les points constituant le maillage 3D, à la recherche du plan le plus important. Cependant, cette approche mène parfois à des estimations de plans trop erronées, et ne prend pas en considération les vecteurs normaux à chaque point. Nous nous sommes donc orientés vers une approche de segmentation basée sur l'algorithme VCCS [102]. Pour rappel (voir section 2.5), cet algorithme regroupe des points voisins selon un critère de similitude Δ défini par :

$$\Delta = \sqrt{\omega_c \delta_c^2 + \frac{\omega_p \delta_p^2}{3R_{seed}^2} + \omega_n \delta_n^2}$$

Où les δ_i représentent la distance euclidienne respectivement pour la couleur, la position et le vecteur normal associé à chaque point du maillage 3D. Les ω_i représentent eux l'importance accordée pour chacun des ces trois paramètres dans le calcul de Δ.

Afin d'effectuer une segmentation planaire, l'algorithme fusionne deux points si leurs vecteurs normaux sont suffisamment proches. Cela revient à utiliser un poids ω_n élevé. Les autres paramètres ne sont pas négligés, notamment, les murs sont souvent de couleur uniforme, aussi, la couleur est un critère supplémentaire pour associer deux points. Enfin, deux points appartenant à deux plans parallèles auront les mêmes vecteurs normaux, il faut donc aussi prendre en compte la distance de ces deux points dans le calcul de Δ.

Une fois l'algorithme VCCS appliqué, le maillage 3D est transformé en une liste de clusters de points coplanaires, et d'un graphe d'adjacence. Ce graphe d'adjacence permet de fusionner les clusters voisins similaires. Deux clusters sont fusionnés si l'angle entre leurs deux vecteurs normaux est inférieur à $20°$, et si les plans qu'ils représentent sont éloignés de moins de 10 cm.

3.4.1.3 Classification des plans

Une fois l'identification des plans principaux contenus dans le maillage 3D effectuée, chacun des plans est étiqueté comme étant le sol, le plafond, un mur, ou du bruit.

D'une manière générale, le sol sera le plan orthogonal à l'axe vertical situé le plus en bas, et de manière analogue, le plafond sera celui qui sera situé le plus en haut. Considérant que les murs sont verticaux et orthogonaux au sol [20], les murs formant un angle inférieur à $20°$ avec l'axe vertical sont considérés comme

des candidats potentiels. Parmi ces plans, ceux disposant d'une extension verticale d'au moins 80% de la distance sol-plafond sont considérés comme étant des murs.

3.4.2 Détection des ouvrants

Nous avons formulé l'hypothèse que les ouvrants (portes et fenêtres) que contiennent nos maillages reconstruits sont de forme rectangulaire. Étant donné que nous utilisons un capteur infrarouge, les surfaces en verre ne réfléchissent pas les rayons issus du capteur, les fenêtres laissent donc une surface rectangulaire vide. En effectuant une reconstruction d'un environnement intérieur dans lequel nous avons également laissé les portes ouvertes, tous les ouvrants laisseront une zone rectangulaire vide sur une des surfaces du maillage 3D.

La détection des ouvrants, illustrée sur la figure 3.6, se fait individuellement pour chaque mur préalablement détecté. Tout d'abord, nous détectons l'ensemble des ouvertures rectangulaires vides que contiennent chacun des murs. Ces ouvertures sont ensuite étiquetées comme portes, fenêtres, ou autres, en prenant en compte que d'autres objets, comme une armoire masquant une partie d'un mur, peuvent laisser une surface rectangulaire vide sur le maillage d'un mur.

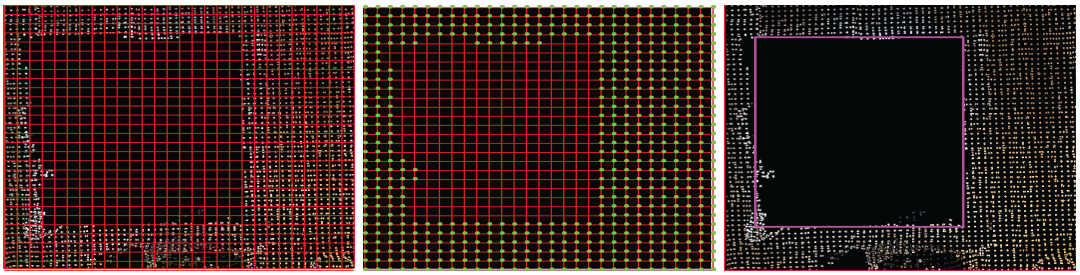


Figure 3.6 – *Illustration du processus de détection des ouvrants.*

3.4.2.1 Détection des ouvertures rectangulaires

L'algorithme 3 détaille le processus de détection des zones rectangulaires vides que contiennent les différents murs. Soit Π un plan détecté précédemment et étiqueté comme étant un mur. Soit $\boldsymbol{n}$ un vecteur normal à Π. Nous nous plaçons d'abord dans un repère $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{n})$ lié à ce mur. Pour cela, nous construisons deux vecteurs de base $\boldsymbol{u}$ et $\boldsymbol{v}$ tels que $\boldsymbol{u} \cdot \boldsymbol{v} = 0$, et $\boldsymbol{u} \wedge \boldsymbol{v} = \boldsymbol{n}$. Nous nous intéressons alors à la projection Π_{proj} de Π dans le repère $(\boldsymbol{u}, \boldsymbol{v})$.

Nous construisons une grille $\mathcal{G}$, qui associe à chaque position 2D (u, v) un booléen indiquant si cette zone est occupée ou non. Soit ρ la résolution spatiale du maillage 3D utilisé. Alors $\mathcal{G}$ aura comme largeur $w = \dfrac{L}{\delta}$ et comme hauteur

$h = \frac{H}{\delta}$. Où L désigne l'extension horizontale maximale de Π, et H son extension verticale maximale.

Ainsi, pour tout point $\boldsymbol{p} \in \Pi_{proj}$, nous lui associons l'élément de $\mathcal{G}$ situé en $\left(E(\frac{u}{\delta}), E(\frac{v}{\delta})\right)$, où $E()$ désigne la fonction partie entière. À l'inverse, nous associons à l'élément situé en (u, v) de $\mathcal{G}$ le point $(u \cdot \rho, v \cdot \rho) \in \Pi_{proj}$.

Algorithme 3 : *Détection des zones rectangulaires vides.*

$\Theta \leftarrow \emptyset$
$\Pi_{proj} \leftarrow proj(\Pi)$
for all $(u, v) \in \Pi_{proj}$ **do**
  $\mathcal{G}(E(\frac{u}{\rho}), E(\frac{v}{\rho})) \leftarrow 1$
end for
$\Omega \leftarrow cluster\_espaces\_vides(\mathcal{G})$
for all $\boldsymbol{p} = (u, v) \in \Omega$ **do**
  $\Theta_\Omega \leftarrow boite\_moyenne(\boldsymbol{p})$
  if $\frac{\Sigma_{(u,v)\in V} G(u,v)}{Card(\Theta_\Omega)} > \epsilon$ **then**
    $\Theta \leftarrow \Theta + \Theta_\Omega$
  end if
end for

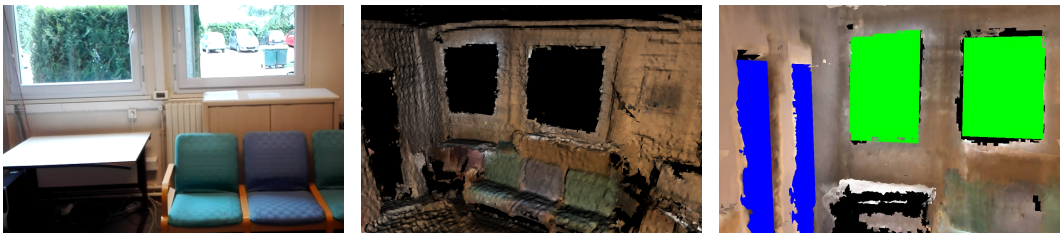
L'algorithme procède ainsi : pour tout point $\boldsymbol{p} \in \Pi_{proj}$, on met $\mathcal{G}\left(E(\frac{u}{\rho}), E(\frac{v}{\rho})\right)$ à 1. Ensuite, on regroupe les zones de $\mathcal{G}$ voisines ayant pour valeur 0, on notera Ω cet ensemble. Cela se fait facilement en ajoutant de manière récursive les voisins étant également à 0. Enfin, pour chaque région, on calcule sa boîte englobante moyenne Θ_Ω. Si le nombre de zones vides contenues dans Θ_Ω est suffisamment proche du nombre de points total dans Θ_Ω, alors on considère que cette zone est effectivement rectangulaire, et on l'ajoute à l'ensemble des zones rectangulaires Θ.

3.4.2.2 *Classification des ouvrants*

Les ouvrants sont séparés en deux catégories, les portes et les fenêtres. Les portes partent du sol et ont une hauteur d'au moins deux mètres, et les fenêtres sont plus proches du plafond que du sol. Chaque zone rectangulaire vide détectée remplissant l'une de ces deux hypothèses est étiquetée comme un ouvrant. Le résultat final peut être visualisé sur la figure 3.7.

3.4.3 Paramètres

Le tableau 3.2 liste les paramètres utilisés pour l'algorithme de de segmentation planaire.



(a) *Original* **(b)** *Maillage 3D* **(c)** *Après la détection*

Figure 3.7 – *Illustration du processus de reconstruction et de détection de la structure d'un bâtiment, de la construction du maillage la détection des ouvrants.*

Étape	*Paramètre*	*Valeur*	*Commentaire*
VCCS (§3.4.1.2)	α_c	0.2	Pondération pour la couleur
	α_p	0.4	Pondération pour la position
	α_n	1.0	pondération pour vecteur normal
	R_{seed}	0.4	Rayon de recherche (m)

Tableau 3.2 – *Paramètres pour la génération du maillage 3D et la segmentation planaire.*

L'algorithme VCCS a été paramétré afin de privilégier des vecteurs normaux identiques pour générer les clusters. Pour cela, la pondération associée au vecteur normal α_n est plus important que les autres. Nous avons ensuite privilégié la position avec α_p, afin d'éviter de regrouper deux points appartenant à des plans parallèles.

3.5 Implémentation

Nous détaillons ici l'implémentation pratique de l'algorithme de maillage 3D implémenté sur la tablette cible (voir §1.4.3). Cet algorithme est le plus critique puisqu'il doit s'effectuer en temps réel.

3.5.1 Répartition des tâches

Les tâches de calcul sont réparties sur deux *threads* : un thread d'acquisition des données et un thread de calcul. L'algorithme est architecturé sur un modèle producteur - consommateur. Le *thread* d'acquisition des données prépare les données des capteurs afin de les mettre à disposition du thread de calcul qui mettra à jour un TSDF et le maillage 3D associé.

3.5.2 Acquisition des données

Le *thread* d'acquisition est chargé de synchroniser les données reçues par les capteurs (nuage de points, images vidéo et positions). À chaque nouveau nuage de points reçu à un instant t, le *thread* de synchronisation calcule la position de la tablette estimée à cet instant. La dernière image RGB obtenue est utilisée pour colorer ce nuage de points. Ce nuage de points et la position associée sont ensuite mis à disposition du thread de calcul.

3.5.3 Calculs

Le *thread* de calcul s'occupe de mettre à jour le TSDF à l'aide des nuages de points acquis par le capteur de profondeur. Les étapes du processus sont les suivantes :

1. **Alignement du nuage de points :** utilisation d'un algorithme ICP pour corriger les éventuels écarts de positionnement.
2. **Identification des régions spatiales à mettre à jour :** à partir du nuage de points en entrée, et de la position de la tablette, on détermine les différentes régions spatiales traversées par le champ de vision du capteur de profondeur.
3. **Mise à jour des régions spatiales :** mise à jour des valeurs des voxels des régions spatiales précédemment détectées à l'aide des points 3D reçus.
4. **Mise à jour des maillages 3D :** mise à jour des maillages avec *Marching Cubes* pour les régions spatiales mises à jour.

Ces quatre étapes doivent être réalisées de manière séquentielle, cependant, les étapes 2,3 et 4 agissent de manière indépendante sur chacune des régions spatiales. La réalisation de ces tâches a été divisée sur plusieurs threads, afin de permettre un traitement plus rapide des nuages de points entrant. Cette parallélisation a été effectuée à l'aide de la bibliothèque OpenMP (voir annexe C).

Quatre threads de calcul sont créés au début de l'étape 2, avec des barrières entre chaque étape, permettant d'améliorer les temps de calcul, comme nous allons voir à la section3.6.

3.6 Évaluations

Nous présentons ici, les évaluations de l'algorithme de reconstruction 3D mis en place. Toutes es évaluations portent sur un même environnement : une pièce contenant trois portes et deux fenêtres (voir sur la figure 3.8). Nous avons évalué plusieurs aspects de l'algorithme : performances, précision, et facilité d'usage, afin de montrer l'apport potentiel de l'automatisation du processus de modélisation sur une tablette.

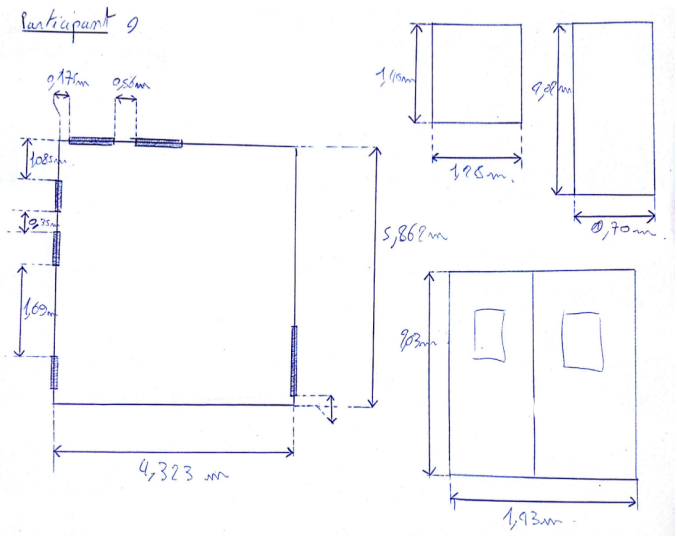


Figure 3.8 – *Plan de la pièce avec les différentes mesures, complété par un participant. À gauche se trouve la vue de dessus de la pièce, et à droite les différents types d'ouvrants à mesurer.*

3.6.1 Performances

Nous avons mesuré les gains de temps dus à la parallélisation de l'algorithme CHISEL. Ces mesures ont été faites sur un jeu de données de 500 nuages de points, nous avons mesuré le temps moyen d'intégration (mise à jour du TSDF et des maillages 3D associés) d'un nuage de points. Ces mesures sont présentées sur la figure 3.9.

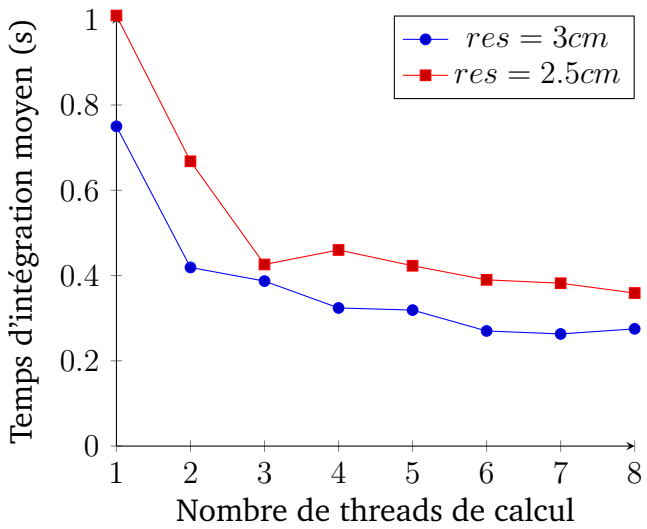


Figure 3.9 – *Temps moyen d'intégration d'un nuage de points en fonction du nombre de threads de calcul utilisés, testé pour deux résolutions spatiale (2.5 cm et 3 cm).*

Nous observons tout de suite une amélioration notable des temps de calcul lors de la parallélisation de l'algorithme d'intégration. Ces valeurs nous

permettent de choisir le meilleur compromis entre les performances de l'algorithme et la résolution spatiale de maillage 3D à utiliser. Compte tenu de notre objectif d'avoir un temps de traitement des données le plus proche possible de 200 ms, nous avons opté pour une résolution spatiale de 3 cm pour reconstituer l'environnement 3D virtuel. De même, nous avons opté pour la parallélisation de l'algorithme sur 4 *threads*. Cependant, nous n'avons pas réussi à obtenir un temps d'intégration d'un nuage de points inférieur à 200 ms.

3.6.2 Génération du modèle 3D

Nous avons évalué la précision de reconstruction du maillage 3D et d'identification de la structure d'un environnement réel. Pour cela, nous avons effectué plusieurs captures de notre environnement de référence. Nous avons comparé les mesures obtenues pour les dimensions (longueur, hauteur, largeur) de la pièce, et les dimensions (hauteur, largeur) des cinq ouvrants de cette pièce. Les résultats sont synthétisés dans le tableau 3.3.

Objet	*Dimensions réelles (m.)*			*Modlèle 3D (m.)*		
	L	H	W	L	H	W
Pièce	5.90	2.80	4.40	5.82	2.82	4.53
Porte 0	-	2.00	1.45	-	1.78	1.64
Porte 1	-	2.00	0.70	-	1.97	0.74
Porte 2	-	2.00	0.70	-	1.95	0.8
Fenêtre 1	-	1.24	1.12	-	1.35	1.10
Fenêtre 2	-	1.24	1.12	-	1.26	1.07

Tableau 3.3 – *Comparaisons entre les mesures réelles et les dimensions calculées.*

En comparant les mesures obtenues par reconstruction 3D et les mesures réelles, l'écart moyen mesuré est de 5%, et l'écart maximum est de 14%. Bien que l'écart est assez important dans certains cas, nous allons montrer que cela reste suffisant pour pouvoir évaluer les performances énergétiques du bâtiment reconstruit.

3.6.3 Évaluation des performances énergétiques

Nous avons évalué les résultats de l'évaluation des performances énergétiques de cinq captures différentes de l'environnement capturé à l'aide de la tablette puis importé dans l'application Plan 3D Énergie (voir section 1.2). Pour cela, nous avons comparé le score calculé par rapport au score du même modèle 3D réalisé à la main, à partir des dimensions réelles. Le score des évaluations énergétiques a été calculé en utilisant l'implémentation de l'algorithme 3CL de l'application Plan

3D Énergie. Le calcul des performances énergétiques se fait ici uniquement par rapport aux aspects géométriques (surface des murs et des ouvrants) de la pièce.

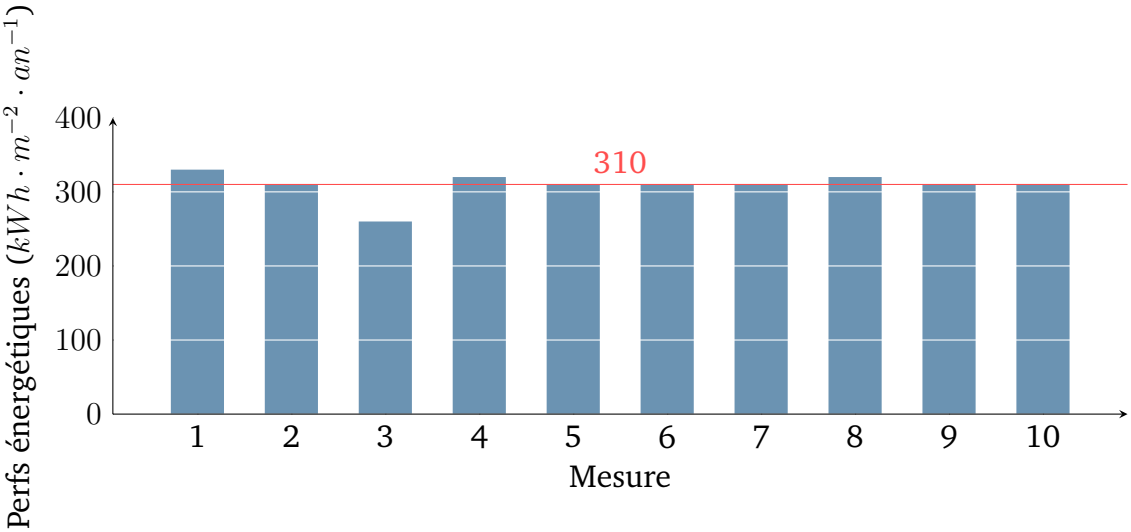


Figure 3.10 – *Évaluation des performances énergétiques de la pièce testée. En rouge : la valeur obtenue avec l'application Plan 3D Énergie.*

La figure 3.10 montre les résultats obtenus par rapport à la valeur de base de $310kWh \cdot m^{-2} \cdot an^{-1}$ calculée à l'aide des dimensions réelles. Le score moyen calculé à l'aide des modèles reconstruits est de $306kWh \cdot m^{-2} \cdot an^{-1}$. La plupart du temps, l'arrondi du score à la dizaine près est de 310, et dans tous les cas, la même note "E" est attribuée. Les différences de mesures entre les dimensions réelles et mesurées impactent peu le résultat du calcul des performances énergétiques.

3.6.4 Évaluation à l'usage

L'identification des constituants d'un bâtiment dont un maillage 3D a été construit se fait de manière automatique. Nous avons comparé le temps pris pour reconstruire ce maillage 3D par rapport à une prise de mesures manuelle, cela permet d'évaluer le gain de temps qu'il est possible d'obtenir en utilisant notre algorithme de reconstruction 3D par rapport à une prise de mesure manuelle, et une saisie dans un logiciel de CAO.

Nous avons comparé, pour 10 participants, les temps mis pour modéliser la pièce à l'aide de la tablette, puis le temps pris pour relever chacune des mesures intéressantes manuellement (les utilisateurs avaient à leur disposition un mètre et un télémètre laser). Les différentes mesures à prendre manuellement sont présentées sur la figure 3.8.

La figure 3.12 affiche les temps mis pour chacun des participants pour réaliser ces deux tâches. Les participants ont mis en moyenne $419 \pm 73sec$ pour relever toutes les mesures à la main, tandis qu'il leur a fallu en moyenne $247 \pm 44sec$

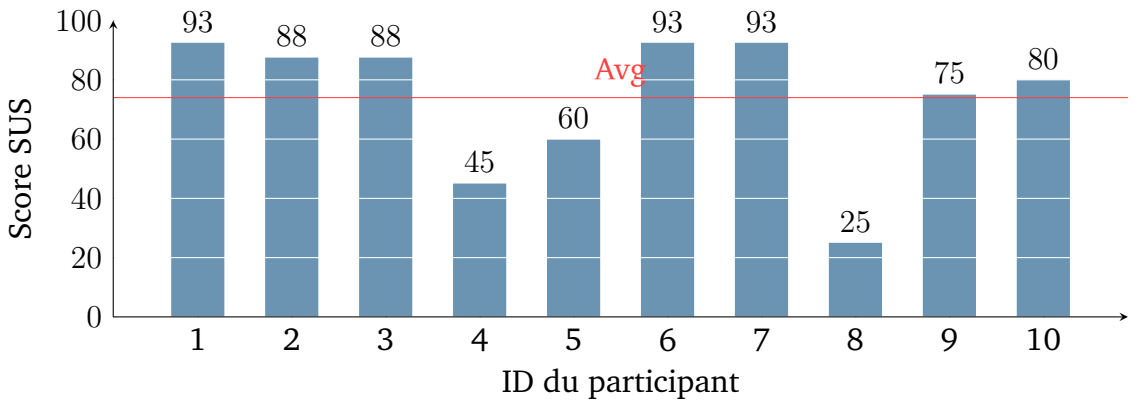


Figure 3.11 – *Scores SUS pour l'utilisation d'une tablette pour reconstituer une pièce.*

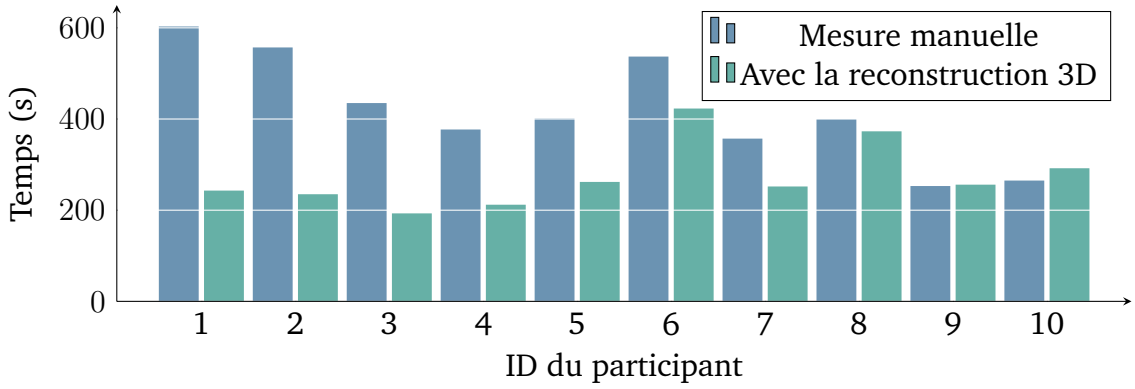


Figure 3.12 – *Temps pris pour chacune des tâches : prise de mesures manuelle, puis à l'aide de la reconstruction 3D.*

pour générer un maillage 3D avec notre algorithme de reconstruction 3D, pour un intervalle de confiance à 95%. L'utilisation de la tablette pour reconstruire un maillage 3D de la pièce semble dont plus rapide que de relever toutes les mesures à la main.

Nous avons également évalué l'utilisabilité de la tablette comme outil pour effectuer un scan 3D de la pièce. Nous avons calculé le score d'utilisabilité pour chaque participant à l'aide d'un questionnaire SUS [12], et reporté les résultats sur la figure 3.11. Bien que le score moyen soit de 74%, deux participants ont donné des scores très faibles, d'après leur retours, l'utilisation de la tablette ne représentait pas de réel avantage par rapport à une prise de mesures manuelle.

3.6.5 Discussion

L'utilisation d'une faible résolution spatiale (3 cm) mène à quelques imprécisions dans le modèle 3D final. Cependant, nos mesures montrent que nous pouvons estimer les principales dimensions du bâtiment reconstruit, et que les erreurs de mesure n'impactent pas l'estimation des performances énergétiques finale. De plus, nos évaluations suggèrent que l'utilisation de la tablette pour

effectuer une capture 3D d'un environnement intérieur (même simple) est plus efficace que qu'une prise de mesures manuelle.

Cependant, très peu d'informations dans le maillage 3D final permettent d'identifier les ouvrants de manière fiable. La seule information exploitable est la présence de zones rectangulaires vides sur les murs, mais ces ouvertures sont aussi dues à la présence de mobilier dans le bâtiment. Cela mène à de faux positifs et limite l'efficacité de nos algorithmes à un environnement intérieur peu meublé. Enfin, la génération en temps réel du maillage 3D reste très coûteuse en temps de calculs, même après optimisation.

3.7 Conclusion

Nous avons exploré une première approche de reconstruction 3D fonctionnant sur une tablette équipée d'un capteur de profondeur. Les algorithmes mis en place permettent à un utilisateur de reconstruire un maillage 3D en temps réel d'un environnement intérieur existant. Nous avons ensuite conçu un algorithme permettant d'identifier la structure du bâtiment (murs, portes et fenêtres) contenus dans un maillage 3D ainsi généré.

Grâce à ces outils, un utilisateur est capable de réaliser une maquette 3D éditable d'un environnement intérieur existant. Afin de générer un maillage 3D en temps réel, il a fallu réduire sa résolution spatiale. Il en résulte quelques imprécisions dans ces mesures. Cependant, nous avons montré que malgré ces imprécisions, il est tout de même possible d'utiliser ces outils de reconstruction pour une phase d'estimation des coûts de rénovation si un devis devait être fait. De plus, nous avons mis en évidence que la modélisation d'un environnement intérieur par reconstruction 3D est plus efficace qu'en relevant les mesures manuellement. Cette reconstruction 3D ne se substitue toutefois pas à l'élaboration d'un plan précis d'un bâtiment, mais est utile pour estimer les performances énergétiques globales du bâtiment, et les différentes surfaces et volumes.

Nous avons cependant mis en évidence certaines limitations dues à la construction d'un maillage 3D complet. En effet, la résolution spatiale étant limitée, les informations sont toutes fusionnées dans un seul et même modèle. Le maillage 3D généré ne donne aucune autre information que la géométrie globale du bâtiment, et il est très compliqué d'identifier les ouvrants de manière fiable dans ce maillage.

Dans le cas de la modélisation d'un bâtiment où nous ne cherchons qu'à identifier sa structure, les seules informations qu'il est nécessaire de conserver sont les positions et dimensions de chaque mur et de chaque ouvrant. Dès lors, l'utilité de construire un maillage 3D de basse résolution contenant une multitude

de points, mais finalement peu d'informations est discutable. Dans la suite de nos travaux, nous allons envisager une autre approche fonctionnant entièrement en temps réel, sans génération de maillage 3D intermédiaire. L'identification de la structure du bâtiment se fera à la volée. Cela permettra de ne sauvegarder que les informations intéressantes et de tirer parti de toutes les données des capteurs disponibles.

Chapitre 4

Segmentation planaire en temps réel à l'aide d'un algorithme de croissance de régions

4.1 Introduction . . . 82
4.2 Travaux connexes . . . 83
4.2.1 Calcul d'une moyenne pondérée . . . 83
4.2.2 Optimisation d'une fonction de coût . . . 84
4.2.3 Conclusion . . . 85
4.3 Description de l'algorithme . . . 85
4.3.1 Vue générale de l'algorithme mis en place . . . 85
4.3.2 Acquisition des données . . . 86
4.3.3 Segmentation planaire . . . 89
4.3.4 Paramètres . . . 92
4.4 Implémentation . . . 93
4.4.1 Répartition des tâches . . . 94
4.4.2 Mise en forme des données . . . 94
4.4.3 Segmentation . . . 95
4.4.4 Mémoires et partage des données . . . 95
4.5 Évaluations . . . 96
4.5.1 Évaluation de la précision et de la fiabilité . . . 96
4.5.2 Discussion . . . 97
4.6 Conclusion et travaux futurs . . . 98

4.1 Introduction

À partir de ce chapitre, nous abordons la deuxième approche de reconstruction 3D étudiée. Celle-ci consiste à générer un modèle 3D éditable d'un bâtiment à la volée, en traitant les données des capteurs au fur et à mesure plutôt que de traiter les données d'un maillage 3D *a posteriori*. Nous nous plaçons, comme lors du chapitre précédent, dans le cas où les murs de l'environnement intérieur à reconstituer sont plans et orthogonaux au sol (voir §1.4.1).

Parmi les limitations mises en avant lors du chapitre précédent, nous avons vu que pour pouvoir générer un maillage en temps réel sur une tablette, il est nécessaire d'en réduire sa précision. Cela permet d'effectuer tous les calculs dans le délai imparti, et d'économiser de la mémoire. De plus, les différentes données acquises par les capteurs (nuages de points, images, positions) sont fusionnées dans un même modèle global sans possibilité de conserver un historique des observations. Cela ne pose pas de problème majeur pour la détection des murs, mais limite les informations disponibles pour identifier les ouvrants.

Nous avons donc réorienté notre réflexion sur une nouvelle approche de reconstruction 3D. La maquette finale n'est plus reconstruite à partir d'un maillage 3D global, mais est mise à jour à chaque nouvelle acquisition des capteurs.

Cette solution présente deux avantages : tout d'abord, une plus grande quantité d'information est disponible pour identifier la structure de la scène (images RGB complètes, nuages de points, ...) ; ensuite, il n'est plus nécessaire de mémoriser un maillage 3D complexe, ce qui permet d'économiser du temps et des ressources de calcul.

Nos travaux reposent sur la capacité à pouvoir détecter l'ensemble des plans contenus dans chaque nuage de points 3D renvoyé par le capteur dans un intervalle de temps inférieur à 200 ms afin de satisfaire la contrainte temporelle énoncée au §1.4.2.

Nous présentons ici un premier algorithme de segmentation planaire en temps réel, qui constitue la première étape à la génération d'une maquette numérique. Cet algorithme fonctionne sur une tablette équipée d'un capteur de profondeur et permet de détecter à la volée l'ensemble des plans 3D contenus dans chaque nuage de points 3D renvoyé par le capteur de profondeur. Il s'agit d'un algorithme de croissance de régions, qui suppose que les nuages de points en entrée sont ordonnés (les points peuvent être rangés dans un tableau 2D), et leurs voisins sont donc accessibles immédiatement.

Grâce à cet algorithme il est possible de segmenter chaque nuage de points renvoyé par le capteur de profondeur en moins de 200 ms.

Le reste de ce chapitre s'organise ainsi, tout d'abord, nous dressons un aperçu

des techniques d'estimation des normales (section 4.2). Ensuite, nous décrivons en détail le fonctionnement de l'algorithme mis en place (section 4.3). Nous détaillons ensuite l'architecture de l'algorithme et son implémentation en pratique (section 4.4), puis nous présentons évaluons la précision et la fiabilité de cet algorithme (section 4.5).

4.2 Travaux connexes

Ces travaux de détection des plans à la volée reposent sur deux problématiques : la segmentation planaire en elle-même, et l'estimation des normales. Les techniques de segmentation planaire ont déjà été abordées à la section 2.5, nous nous concentrons donc ici sur l'estimation des vecteurs normaux dans un nuage de points.

Soit $\mathcal{P}$ un nuage de points. La détection des plans contenus dans $\mathcal{P}$ consiste à calculer pour chaque surface $\mathcal{S}_i \in \mathcal{P}$, l'ensemble de ses vecteurs normaux aux points $\{\boldsymbol{p_i} \in \mathcal{P} \cap \mathcal{S}_i\}$. Dans le cas d'une surface $\mathcal{S}_i$ plane, l'ensemble de ses vecteurs normaux sont colinéaires.

L'estimation du vecteur normal à une surface en un point $\boldsymbol{p}$ nécessite la connaissance de ses k plus proches voisins (KNN) afin de décrire localement la surface à laquelle il appartient. Pour un nuage de points ordonné (dont les éléments sont indexés selon un espace 2D), ce qui est typiquement le cas des capteurs à lumière structurée, la recherche des KNN est immédiate, il suffit de prendre les points d'indices voisins. Sinon, plusieurs méthodes permettent d'obtenir les KNN d'un nuage de points [37, 99, 40, 18, 114].

Il existe deux grandes méthodes pour estimer les vecteurs normaux dans un nuage de points : les algorithmes consistant à faire une moyenne par rapport aux KNN (détaillés au §4.2.1), et les algorithmes basés sur l'optimisation d'une fonction de coût [53, 70] (détaillés au §4.2.2).

4.2.1 Calcul d'une moyenne pondérée

Avec les méthodes basées sur la moyenne, le vecteur normal en un point est calculé en utilisant une moyenne pondérée des K plus proches voisins du point considéré.

Jin et al.[63] comparent six méthodes de calcul de normales à partir d'un maillage 3D. Ces six méthodes utilisent une moyenne des vecteurs normaux des faces voisines à chaque sommet. Elles vont du simple calcul des moyennes [38] à des sommes pondérées des normales de différentes faces voisines, pondérées par l'importance de ces faces. L'importance d'une face est déterminée soit par la taille de sa surface, soit par l'angle formé entre le sommet considéré, et les deux autres

sommets de cette face. D'après ces études, il ressort que la moyenne géométrique non pondérée est la solution la plus rapide.

La solution la plus précise est celle développée par Türrner et al. [136]. Les auteurs proposent de calculer le vecteur normal en chaque point en faisant une somme pondérée des normales des triangles ayant ce point en sommet. La somme est pondérée par les surfaces des triangles de manière à obtenir le même résultat quelle que soit la densité du maillage (voir la figure 4.1).

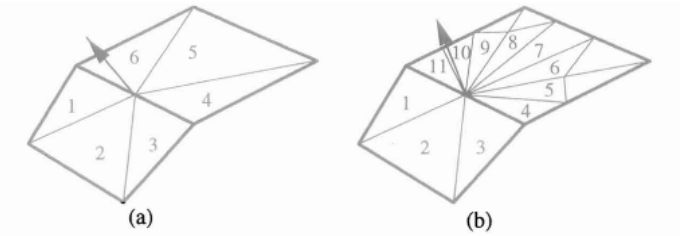


Figure 4.1 – *Calcul des normales à une surface en pondérant la somme par les angles [136]. La pondération du calcul par rapport à la surface des triangles permet d'obtenir le même résultat quelle que soit la densité des triangles.*

Dans le cas d'une carte de profondeur, Holzer et al. [55] utilisent une image intégrale de la carte de profondeur pour calculer le vecteur normal en un point Les images intégrales ont été introduites par Crow et al.[21]. Soit I une image, son image intégrale I' est définie par :

$$I'(u', v') = \sum_{u' \leqslant u, v' \leqslant v} I(u, v)$$

Une image intégrale permet de calculer la somme des éléments contenus dans un rectangle en seulement quatre accès mémoire, indépendamment de la taille de ce rectangle. Elles sont utilisées ici pour calculer la somme des voisins de chaque point dont on veut calculer le vecteur normal. La somme se fait selon quatre directions (haut, bas, gauche, droite). Le vecteur normal est ensuite calculé par produit vectoriel.

4.2.2 Optimisation d'une fonction de coût

Le second type d'approche consiste à déterminer le vecteur normal à chaque point en minimisant une fonction de coût [52, 58]. Une première méthode consiste à déterminer pour un point $\boldsymbol{p}$ l'équation du meilleur plan passant par $\boldsymbol{p}$ et ses k plus proches voisins [70, 55]. La fonction à minimiser peut être alors l'erreur globale de positionnement des points par rapport au plan par exemple. D'autres méthodes plus complexes consistent à approximer la surface de manière locale à l'aide d'une surface quadratique [146]. Ces méthodes peuvent conduire à des estimations très précises [70] des vecteurs normaux à une surface.

4.2.3 Conclusion

Nous avons conservé, comme lors des travaux précédents, l'idée de procéder à une segmentation planaire à l'aide d'un algorithme similaire à l'algorithme VCCS [102]. Il s'agit d'un algorithme de croissance de régions qui procède d'abord à plusieurs segmentations locales avant de fusionner les clusters générés. L'étape de segmentation locale est plus rapide à effectuer qu'une segmentation globale et peut facilement être parallélisée.

La détection des plans repose sur le calcul des vecteurs normaux à chaque point du nuage en entrée. Dans le cas où les nuages de points fournis en entrée sont ordonnés en 2D. Cela signifie que pour chaque point, il est possible d'accéder immédiatement à ses k plus proches voisins. Nous avons décidé d'utiliser un algorithme de détection des normales similaire à Holzer et al. [55].

L'algorithme calcule pour chaque point la position moyenne de ses plus proches voisins. À partir de ces points moyens, il est possible d'en déduire deux vecteurs de base du plan les contenant. Nous avons adapté cet algorithme afin de l'utiliser sur un nuage de points ordonné et non sur une carte de profondeur. En particulier, les nuages de points que nous avons utilisés ne sont pas continus (la carte de profondeur n'est pas dense), et nous n'avons pas pu utiliser d'image intégrale pour calculer les vecteurs normaux. Nous verrons cependant que cela ne pose pas de problème de performances.

4.3 Description de l'algorithme

Nous détaillons ici l'algorithme de détection des plans mis en place. En particulier, nous détaillons les différentes données en entrée, ainsi que les différentes étapes de la segmentation.

4.3.1 Vue générale de l'algorithme mis en place

L'algorithme de détection des plans est synthétisé sur la figure 4.2. Il fonctionne à partir d'une prise de vue à un instant t contenant une carte de profondeur, une image RGB, et une estimation de la position de la tablette. À partir de ces données, différentes cartes de données nécessaires à la segmentation sont construites. Par carte, nous entendons que ces données intermédiaires sont indexées selon un couple de coordonnées (u, v), et sont de la même dimension que la carte de profondeur en entrée. Ces cartes servent à stocker les données intermédiaires, telles que les couleurs, les positions, ou les normales. Ces différentes cartes sont décrites plus en détail au §4.3.2.

L'algorithme de segmentation mis en place utilise ces différentes cartes en entrée pour détecter les surfaces planes contenues dans le nuage de points

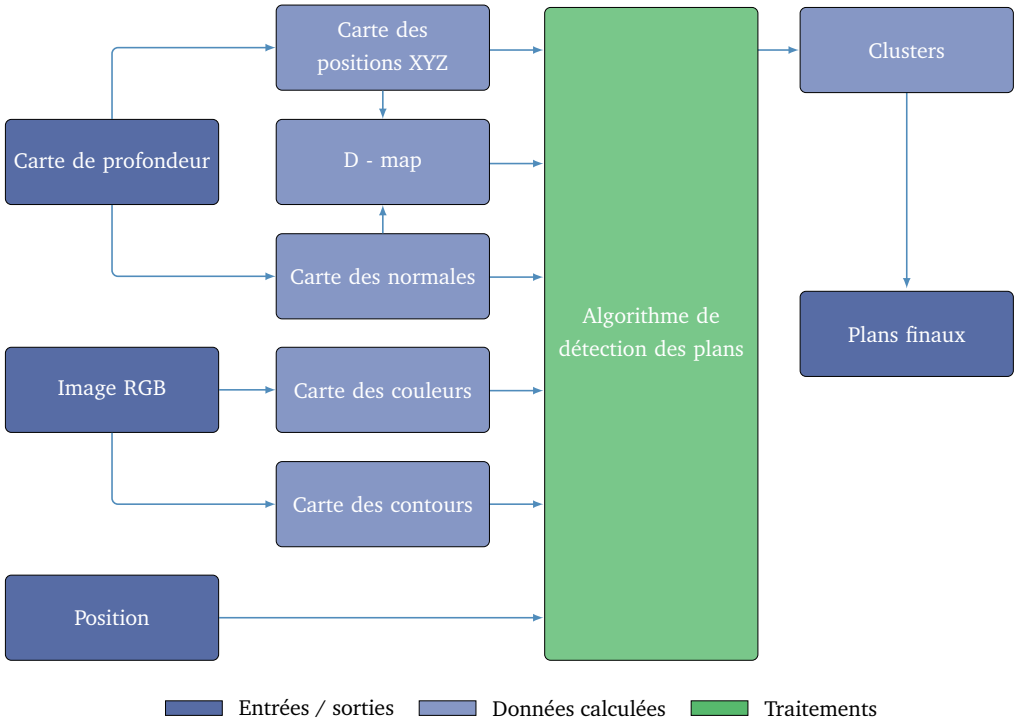


Figure 4.2 – *Vue d'ensemble de l'algorithme de détection des plans.*

d'entrée. Soit un point $\boldsymbol{p_0} = [x_0, y_0, z_0]^\top$ de vecteur normal $\boldsymbol{n} = [n_x, n_y, n_z]^\top$ unitaire. Une équation du plan Π passant par $\boldsymbol{p_0}$ et de vecteur normal $\boldsymbol{n}$ s'écrit :

$$(\Pi) : n_x.x + n_y.y + n_z.z + d = 0$$

avec

$$d = -(n_x.x_0 + n_y.y_0 + n_z.z_0)$$

Tous les points présentant des paramètres de plan (n_x, n_y, n_z, d) similaires sont regroupés en clusters.

Grâce à ces considérations, et des optimisations sur le pipeline d'exécution détaillées en section 4.4, l'algorithme est capable de détecter l'ensemble des plans contenus dans un nuage de points ordonnés en moins de $200ms$ avec une tablette de type Google Tango Yellowstone. Comme précédemment, l'algorithme a été conçu pour utiliser uniquement les capacités de calcul du CPU de la tablette. La figure 4.4 montre un exemple de données en entrée, et le résultat de la segmentation planaire de ces données. Nous allons à présent détailler les différentes données en entrée de l'algorithme ainsi que les étapes de la segmentation.

4.3.2 Acquisition des données

L'algorithme utilise les données des capteurs à un instant t, à savoir : une carte de profondeur, l'image RGB associée, et la position estimée de la tablette à cet

instant. Soient h la hauteur en pixels de la carte de profondeur et w sa largeur, on posera alors $\mathcal{A} = [\![0, h]\!] \times [\![0, w]\!]$. À partir de ces données, cinq cartes sont calculées :

1. **Carte des positions ($\mathcal{M}_{xyz}$) :** il s'agit des positions 3D (x, y, z) déduites de l'intensité des pixels de la carte de profondeur à chaque couple de coordonnées $(u, v) \in \mathcal{A}$.
2. **Carte des normales ($\mathcal{M}_n$) :** il s'agit des vecteurs normaux unitaires, orientés vers l'origine du capteur de profondeur associés à chaque point de la carte des positions.
3. **D-map ($\mathcal{M}_d$) :** cette carte est déduite des deux cartes précédentes. $\forall (u, v) \in \mathcal{A}$ soient
$$\boldsymbol{p} = [x, y, z]^T = \mathcal{M}_{xyz}(u, v)$$
et
$$\boldsymbol{n} = [n_x, n_y, n_z]^T = \mathcal{M}_n(u, v)$$
Alors
$$\mathcal{M}_d(u, v) = -(n_x.x + n_y.y + n_z.z)$$
La donnée de $\boldsymbol{n}$ et d à chaque point permet de caractériser entièrement le plan Π passant par $\boldsymbol{p}$ et de vecteur normal $\boldsymbol{n}$. Nous stockons ce paramètre d afin d'éviter de le recalculer lors de chaque itération de l'algorithme de segmentation.
4. **Carte des couleurs $\mathcal{M}_{rgb}$:** cette carte est obtenue en redimensionnant l'image RGB à la même échelle que la carte de profondeur, donnant ainsi la couleur associée à chaque point 3D.
5. **Carte des contours $\mathcal{M}_c$:** cette carte est obtenue en calculant les contours de l'image RGB, puis en redimensionnant cette image à la même échelle que la carte de profondeur.

Si les trois cartes $\mathcal{M}_{xyz}$, $\mathcal{M}_d$ et $\mathcal{M}_{rgb}$ se déduisent directement des autres données, les cartes des normales et des contours nécessitent un peu plus de calculs que nous allons détailler respectivement au §4.3.2.1 et au §4.3.2.2.

4.3.2.1 Estimation des normales

Les vecteurs normaux sont calculés en utilisant une méthode basée sur la moyenne (voir section 4.2). De cette manière, l'algorithme d'estimation des normales consiste à faire parcourir la carte des positions $\mathcal{M}_{xyz}$ à l'aide d'une fenêtre glissante de largeur L. Pour chaque point $\boldsymbol{p} \in \mathcal{M}_{xyz}$, son voisinage $\Omega_{xyz,\boldsymbol{p}}$ est accessible directement puisque $\mathcal{M}_{xyz}$ est ordonnée. De plus, l'algorithme est très facilement parallélisable et est donc rapide à s'exécuter.

Pour tout couple $(u_0, v_0) \in \mathcal{A}$, on définit une fenêtre $\mathcal{W}_{u_0,v_0}$ par :

$$\mathcal{W}_{u_0,v_0} = \left\{ (u,v) \in \mathcal{A}, \begin{array}{l} |u-u_0| \leq L \\ |v-v_0| \leq L \end{array} \right\} \tag{4.1}$$

Soit $\boldsymbol{p_0} = \mathcal{M}_{xyz}(u_0, v_0)$, son voisinage $\Omega_{xyz,\boldsymbol{p_0}}$ est donc défini par :

$$\Omega_{xyz,\boldsymbol{p_0}} = \{\mathcal{M}_{xyz}(u,v)/(u,v) \in \mathcal{W}_{u_0,v_0}\} \tag{4.2}$$

$\Omega_{xyz,\boldsymbol{p_0}}$ est divisé en quatre quadrants $Q_{no}, Q_{ne}, Q_{so}, Q_{se}$, comme illustré à la figure 4.3.

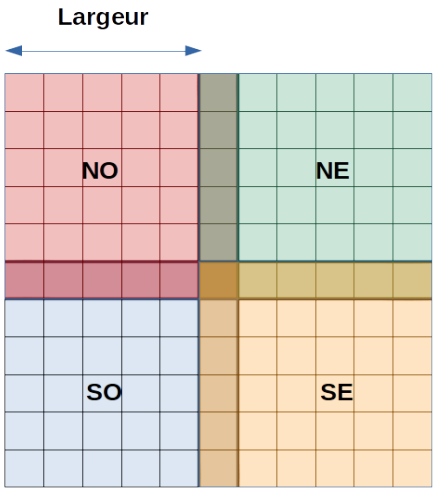


Figure 4.3 – *Exemple de fenêtre glissante pour une largeur de 5. Les quatre quadrans sont identifiés par des points par quatre points cardinaux : nord-ouest (NO), nord-est (NE), sud-ouest (SO) et sud-est (SE).*

Pour chaque tout $i \in \{no, ne, so, se\}$, on calcule le vecteur $\boldsymbol{v_i}$ défini par :

$$\boldsymbol{v_i} = \frac{\sum_{\boldsymbol{p} \in Q_i} \boldsymbol{p}}{Card(Q_i)} \tag{4.3}$$

On définit alors $\mathcal{M}_n(u_0, v_0)$ par :

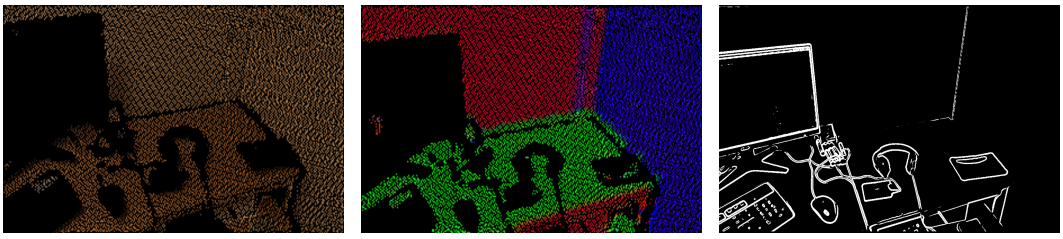
$$\mathcal{M}_n(u_0, v_0) = \frac{(\boldsymbol{v_{se}} - \boldsymbol{v_{no}}) \wedge (\boldsymbol{v_{ne}} - \boldsymbol{v_{so}})}{\| (\boldsymbol{v_{se}} - \boldsymbol{v_{no}}) \wedge (\boldsymbol{v_{ne}} - \boldsymbol{v_{so}}) \|} \tag{4.4}$$

L'utilisation d'une méthode basée sur la moyenne pour détecter les normales présente l'inconvénient de lisser les normales pour les zones non planes de la carte de profondeur. Plus la fenêtre de calcul est grande et plus la largeur de cette zone d'imprécision grandit. Cependant, la précision de la détection des normales sur les zones planes (celles qui nous intéressent) grandit avec la taille de la fenêtre.

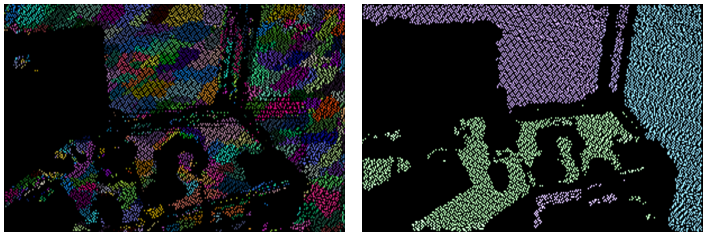
Afin de réduire au maximum la taille de la fenêtre de calcul, tout en gardant la meilleure précision possible pour l'estimation des normales sur les grandes zones planes, nous avons appliqué un filtre gaussien sur la carte des positions avant de calculer les normales. Le résultat de la détection des normales peut-être vu sur la figure 4.4.

4.3.2.2 Calcul des contours

Étant donné que l'algorithme de segmentation utilisé est un algorithme de croissance de régions, nous avons décidé de calculer les contours de l'image RGB. Ces contours serviront à borner la croissance des régions et éviter des "fuites" de régions sur des plans différents, mais assez proches en termes d'orientation. La détection des contours est faite en utilisant un filtre de Canny [13]. Les contours sont stockés dans une table de booléens, mis à zero si le point concerné n'est pas sur un bord, 1 sinon. Un exemple de carte de contours peut être vu sur la figure 4.4.



(a) *Nuage de points coloré* **(b)** *Normales calculées* **(c)** *Contours calculés*



(d) *Clusters intermédiaires* **(e)** *Plans finaux*

Figure 4.4 – *Exemple de données en entrée, avec le résultat de la segmentation planaire à la fin. Les données sont ordonnées en deux dimensions, ce qui évite l'utilisation d'algorithmes de recherche des plus proches voisins.*

4.3.3 Segmentation planaire

L'algorithme de segmentation est inspiré de l'algorithme VCCS [102], et a été adapté afin de prendre en compte l'organisation de nos données, et éviter l'utilisation d'un algorithme de recherche des plus proches voisins. La détection des plans s'effectue en deux étapes. Tout d'abord, un ensemble de sous-clusters est généré autour de points d'origine répartis selon une grille régulière, dont chaque élément est séparé d'une largeur R_{seed}. Ensuite, ces sous-clusters sont fusionnés s'ils représentent un même plan.

4.3.3.1 Génération des clusters

L'algorithme de génération des sous-clusters, illustré sur la figure 4.5, prend en entrée une carte de descripteurs $\mathcal{M}$, dans laquelle les données sont ordonnées en 2D. Il s'agit en fait d'un regroupement des cartes $\mathcal{M}_{xyz}, \mathcal{M}_n, \mathcal{M}_d$ et $\mathcal{M}_{rgb}$ calculées précédemment, avec en plus un poids ω, initialisé à 1 pour chaque descripteur. Un descripteur contient donc pour chaque point les informations de plan et de couleur calculées précédemment. Soit P un descripteur, caractérisé par sa position (x, y, z), ses paramètres de plan (n_x, n_y, n_z, d), et sa couleur (r, g, b). À cela s'ajoute la carte binaire des contours $\mathcal{M}_c$.

Algorithme 4 : *Génération récursive d'un cluster.*

```
P ← M(u, v)
if V(u, v) or stop(P) then
    return
end if
V(u, v) ← 1
if Card(Π) == 0 then
    P_ref ← P
    indices(Π)+ = (u, v)
else
    Δ ← δ(P, P_ref)
    if Δ ≤ ε then
        mettre_a_jour(P_ref, P)
        indices(Π)+ = (u, v)
    end if
end if
if Prof < R_search then
    for all (u', v') ∈ voisinage(u, v) do
        ajouter_point(u', v', cluster, prof + 1)
    end for
end if
```

Un cluster est représenté par un descripteur, et une liste d'indices, permettant de retrouver l'ensemble des points le constituant dans $\mathcal{M}$. L'algorithme 4 décrit le processus de génération d'un cluster. L'algorithme procède ainsi : une grille régulière $\mathcal{G}$ est appliquée sur l'ensemble $\mathcal{A}$ des indices 2D (u, v) permettant de parcourir $\mathcal{M}$. Pour chaque descripteur :

$$P_{ref} = \begin{bmatrix} x_{ref}, y_{ref}, z_{ref} \\ n_{xref}, n_{yref}, n_{zref}, d_{ref} \\ r_{ref}, g_{ref}, b_{ref} \\ \omega_{ref} \end{bmatrix} \in \mathcal{G},$$

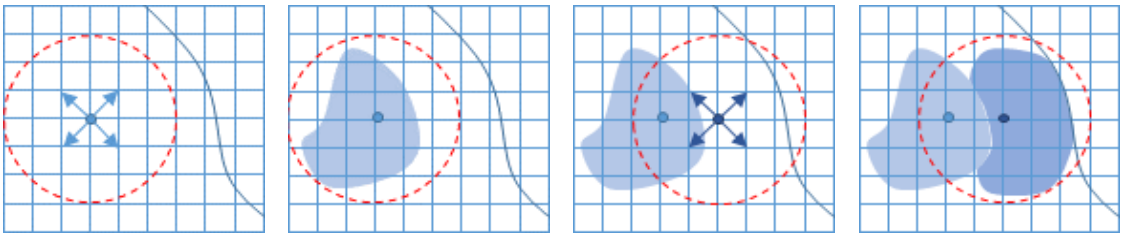


Figure 4.5 – *Processus de génération des clusters : un ensemble de points sources est choisi parmi la carte des descripteurs selon une grille régulière. Le prochain point source est sélectionné (a), et génère un cluster autour de ce point en ajoutant de manière récursive les points voisins (délimités par la zone rouge) ayant des descripteurs similaires (b). Puis, le prochain point source non déjà inclus dans un cluster est sélectionné (c), et un nouveau cluster est généré. L'agrandissement d'une région s'arrête lorsque les voisins observés sont considérés comme trop différents des points constituant le cluster, lorsque l'algorithme rencontre un contour de l'image, ou lorsque la largeur maximale R_{search} est atteinte (d).*

un cluster Π est initialisé avec P_{ref}. Ensuite, l'ensemble des descripteurs voisins à P_{ref} sont récursivement ajoutés s'ils sont considérés comme suffisamment similaires. La similitude entre deux descripteurs P, et P_{ref} est mesurée par la relation $\delta(P, P_{ref})$, définie par :

$$\delta(P, P_{ref}) = \sqrt{\omega_d \delta_d(P, P_{ref})^2 + \omega_n \delta_n(P, P_{ref})^2 + \delta_c(P, P_{ref})^2} \tag{4.5}$$

où $\delta_d = |d - d_{ref}|$, et δ_n et δ_c représentent les distances euclidiennes respectivement pour la normale, et la couleur, et ω_c, ω_p et ω_n l'importance relative de ces trois paramètres.

Si $\Delta = \delta(P, P_{ref})$ est inférieure à un seuil ϵ, alors P est considéré comme appartenant à Π. Dans ce cas, le couple de coordonnées (u, v) de P dans $\mathcal{M}$ vient compléter la liste des indices des descripteurs appartenant à Π.

P_{ref} est ensuite mis à jour avec P comme suit :

$$P_{ref} = \frac{\omega_{ref} P_{ref} + \omega_P P}{\omega_{ref} + \omega_P} \tag{4.6}$$

avec $\omega_P = 1$.

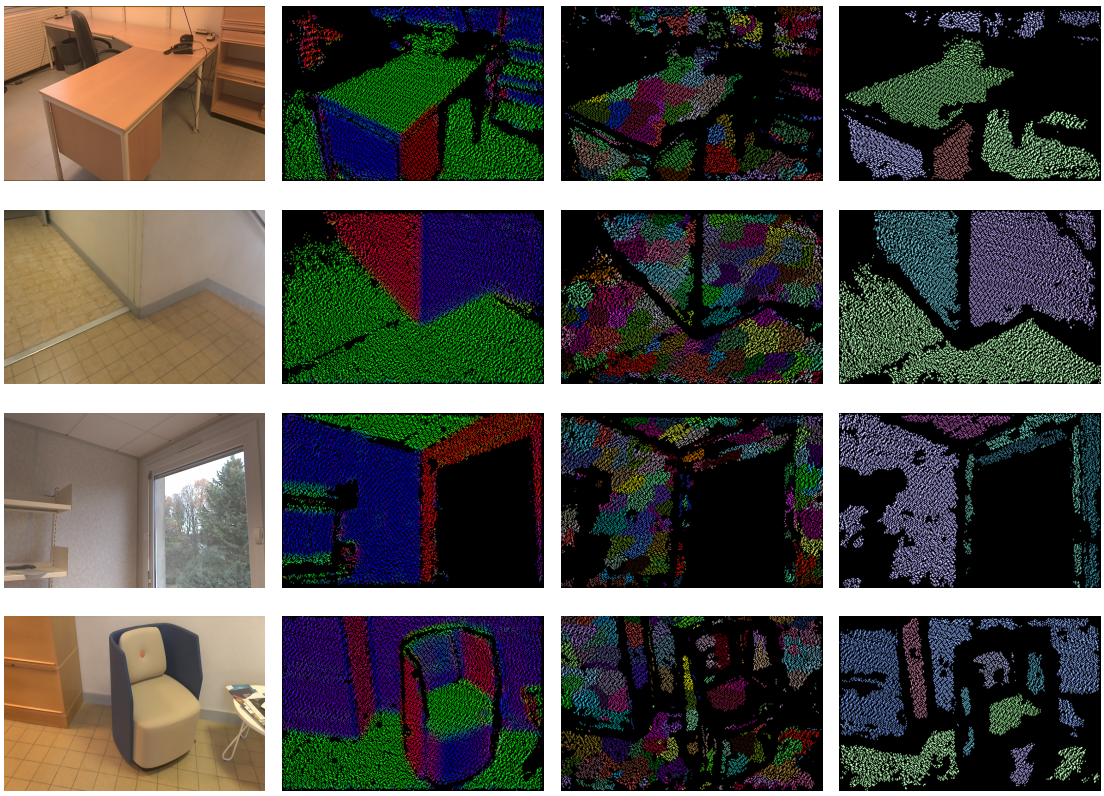
Chaque descripteur déjà visité est marqué afin d'éviter de repasser dessus lors d'une prochaine itération de l'algorithme. C'est le rôle de la carte de booléens V. L'algorithme s'arrête lorsque la profondeur R_{search} est atteinte, dans ce cas, Π a sa largeur maximale. $stop(P)$ est vrai quand P est sur un contour ou lorsque P est nul, dans ce cas, l'algorithme s'arrête aussi. L'extension maximale d'un cluster est contrôlée par la limitation de la profondeur de récursion de l'algorithme.

4.3.3.2 *Fusion des clusters*

Les clusters sont fusionnés pour obtenir le résultat présenté à la figure 4.6. Si deux clusters Π_0 et Π_1 sont considérés comme similaires, c'est-à-dire, si la distance entre leurs descripteurs est inférieure à un seuil θ, alors ces clusters sont fusionnés ensemble. Soient P_0 et P_1 les descripteurs associés à ces deux clusters, alors le nouveau cluster Π formé par la combinaison de Π_0 et Π_1 vaut :

$$\Pi = \frac{\omega_0 \Pi_0 + \omega_1 \Pi_1}{\omega_0 + \omega_1} \tag{4.7}$$

et leurs listes d'indices sont concaténées.



(a) *Original* **(b)** *Normales* **(c)** *Clusters* **(d)** *Plans*

Figure 4.6 – *Application de l'algorithme de détection des plans à plusieurs scènes différentes.*

À ce stade, chaque cluster Π représente un seul des plans contenus dans la carte de profondeur fournie en entrée. Ce cluster Π est constitué d'un descripteur de la même forme que P_{ref} qui contient les paramètres (n_x, n_y, n_z, d) du plan, et un poids ω représentatif de la taille du plan. De plus, Π possède une liste de couples (u, v) permettant de retrouver l'ensemble des points constituant ce plan.

4.3.4 Paramètres

Le tableau 4.1 synthétise l'ensemble des paramètres utilisés par les algorithmes de segmentation planaire (génération et fusion des clusters), et les valeurs qui leur

ont été attribuées.

Étape	*Paramètre*	*Valeur*	*Commentaire*
Génération des clusters (§4.3.3.1)	ω_p	0.20	Pondération pour la distance
	ω_n	0.70	Pondération pour la normale
	ω_c	0.10	Pondération pour la couleur
	R_{search}	30	Largeur max (en pixels) d'un cluster
	R_{seed}	5	Intervalle minimal entre deux clusters
	N_{thres}	0.20	Seuil (en %) pour distinguer deux descripteurs
Fusion des clusters (§4.3.3.2)	min_size	500	Taille minimale d'un cluster pour le conserver
	θ	0.10	Seuil pour distinguer deux plans

Tableau 4.1 – *Paramètres de l'algorithme de segmentation planaire.*

Comme précédemment, une plus grande importance est donnée au vecteur normal pour regrouper les points entre eux. Nous avons donc donné une valeur plus importante à ω_n par rapport aux autres coefficients de pondération dans l'algorithme.

N'ayant pas de problème de performances, nous avons utilisé un faible intervalle R_{seed} entre les différents clusters, afin de maximiser les chances de détecter tous les plans. En effet, en espaçant trop la recherche des clusters, on augmente le risque de rater un plan, ou de regrouper des points appartenant à deux plans différents. Le seuil angulaire N_{thres} entre deux normales a été fixé à 20%.

Pour distinguer deux plans identiques, nous avons mis un seuil θ sur le calcul de leur distance pour les considérer comme identiques. Une fois la fusion faite, seuls les plans disposant d'un minimum de points min_size sont conservés.

4.4 Implémentation

Nous détaillons ici l'implémentation pratique de notre algorithme de segmentation planaire. Comme précédemment, nous avons utilisé la tablette du projet Tango de Google (voir §1.4.3). Notre algorithme est capable de segmenter les nuages de points fournis par le capteur de profondeur dans un intervalle de temps de l'ordre de 100 ms, et dans tous les cas, cet intervalle de temps est inférieur à 200 ms.

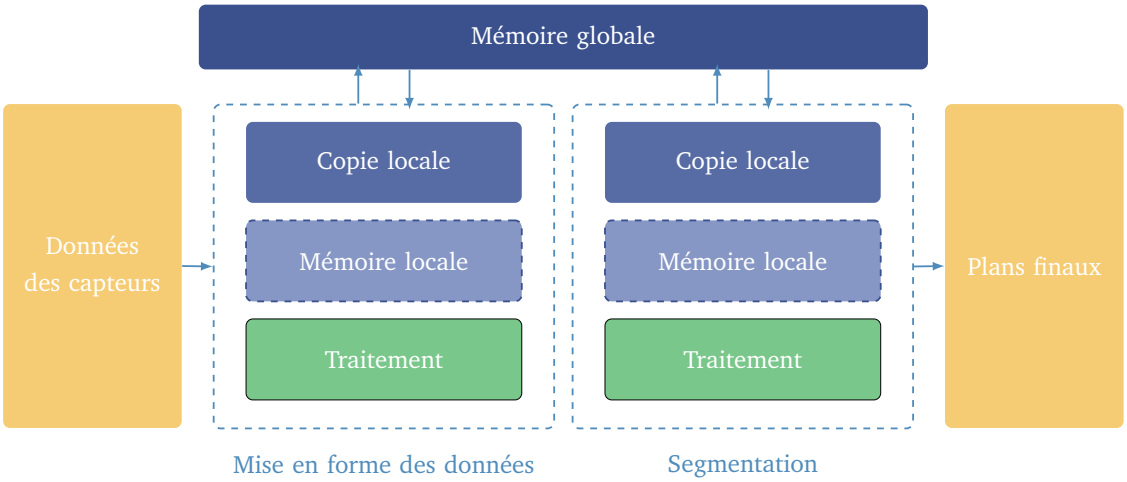


Figure 4.7 – *Répartition des tâches et flux de données pour l'algorithme de segmentation.*

4.4.1 Répartition des tâches

L'algorithme est réparti sur deux *threads* indépendants : un thread d'acquisition des données, un thread de traitement des données (présentés sur la figure 4.7). Ces deux *threads* fonctionnent en parallèle, et traitent chacun une étape du processus de détection des plans.

4.4.2 Mise en forme des données

Le *thread* d'acquisition des données contient tous les prétraitements effectués afin d'obtenir les différentes cartes décrites au §4.3.2. Comme indiqué à la figure 4.2, les données intermédiaires dépendent soit de la carte de profondeur, soit de l'image RGB. Aussi, les groupes $(\mathcal{M}_{xyz}, \mathcal{M}_n, \mathcal{M}_d)$ et $(\mathcal{M}_{rgb}, \mathcal{M}_c)$ peuvent être calculés indépendamment. Il a donc été décidé de créer un *thread* de calcul pour chacun de ces deux sous-ensembles, afin de les exécuter en parallèle.

De même, lors du calcul des vecteurs normaux, et donc de $\mathcal{M}_n$, l'algorithme calcule l'ensemble des vecteurs de $\mathcal{M}_n$ indépendamment les uns des autres à partir de la carte XYZ. Nous avons donc réparti ces calculs sur quatre threads différents. Nous n'avons pas pu tirer parti ici des instructions de calcul SIMD [1] du processeur. En effet, en projetant les points sur un espace 2D de taille fixe, il y a des zones dans la carte des positions $\mathcal{M}_{xyz}$ où il n'y a pas de point. Il est donc nécessaire, avant de calculer la normale à un indice (u, v) de vérifier si un point est bel et bien contenu ici. Dans le cas contraire, le calcul serait inutile. Cela fait que pour un nombre n d'éléments contenus dans $\mathcal{M}_{xyz}$, il y aura seulement k vecteurs normaux à calculer, et $k \leq n$. Sur une implémentation SIMD (voir annexe D), où le nombre de sorties dépend uniquement de la taille des entrées, cela reviendrait à calculer un vecteur normal pour chacune des n entrées considérées, puis de tester individuellement

1. Single Instruction Multiple Data

les entrées pour vérifier si on prend en compte le résultat. Il n'y a pas de gain de temps significatif.

Les *threads* de calcul ont été créés à l'aide de la bibliothèque OpenMP (voir annexe C). Étant donné que le calcul de $\mathcal{M}_n$ consiste à répéter les mêmes calculs pour tous les points de $\mathcal{M}_{xyz}$, et que la répartition des pixels sans points est uniforme, nous avons utilisé une répartition statique du travail dans chacun des threads.

4.4.3 Segmentation

Afin d'effectuer la détection des plans en un minimum de temps, l'algorithme de génération des clusters a également été parallélisé. Pour cela, l'espace des coordonnées (u, v) sur lequel itérer pour chacune des cartes en entrée a été divisé en quatre régions. Afin d'éviter les problèmes de chevauchement de régions dus à la taille variable des sous-clusters qui seront générés, et donc les accès concurrents, nous avons réduit les dimensions de ces régions. Ainsi, chaque région a une largeur égale à $\left[\frac{w}{2} - \frac{R_{search}}{2}\right]$ et une hauteur égale à $\left[\frac{h}{2} - \frac{R_{search}}{2}\right]$, où w correspond à la largeur de la carte de profondeur, h à sa hauteur, et R_{search} la largeur (en pixels) maximale d'un cluster.

Le calcul des clusters pour ces quatre régions est effectué en parallèle dans quatre *threads* de calcul. Les deux premiers *threads* ayant terminé leurs calculs s'occupent ensuite de générer les clusters pour la ligne et la colonne centrale.

Nous avons là aussi utilisé la bibliothèque OpenMP pour paralléliser la segmentation, et avons utilisé une répartition statique de la charge de travail pour chaque *thread*.

4.4.4 Mémoires et partage des données

L'algorithme est architecturé selon un principe producteur/consommateur. Le *thread* de mise en forme des données produit les différentes cartes, et ces cartes sont ensuite utilisées par le *thread* de segmentation.

La mémoire globale aux deux *threads* (voir la figure 4.7) conserve une copie des 5 cartes (positions, normales, d, couleurs et contours). Cette mémoire est alimentée par le *thread* de mise en forme des données, et consommée par le thread de segmentation.

Afin d'éviter l'utilisation de primitives de synchronisation, cette mémoire est dupliquée pour chaque *thread*. Chaque *thread* lit ou écrit sur sa copie locale de la mémoire. Un *thread* lit ou écrit dans la mémoire globale en échangeant son pointeur vers sa copie locale avec le pointeur vers la mémoire globale. Cette opération d'échange peut être réalisée de manière atomique, sans utilisation de mécanisme de synchronisation particulier.

Quand le *thread* de mise en forme des données a généré un nouveau set de données, il met aussi à jour un flag indiquant au thread de segmentation que de nouvelles données sont à traiter.

4.5 Évaluations

Nous avons évalué la fiabilité et la précision de l'algorithme mis en place. Pour cela, nous avons placé la tablette immobile face à un mur de couleur uniforme, à une certaine distance x. Nous avons placé une surface plane entre la tablette et le mur, parallèle au mur (voir la figure 4.8). Dans cette configuration, l'algorithme doit détecter uniquement deux plans, et il est possible de calculer la distance entre ces deux plans.

Figure 4.8 – *Montage expérimental pour évaluer la détection des plans : deux plans parallèles (le mur et la table) sont placés à la verticale devant la tablette.*

4.5.1 Évaluation de la précision et de la fiabilité

Nous avons mesuré pour 100 prises de vue successives la distance d entre les deux surfaces à partir des résultats de l'algorithme, ainsi que le nombre de prises de vue n où le nombre de plans détectés était égal à 2. Nous avons fait varier la distance réelle D entre les deux surfaces planes, et avons recommencé les mesures. Le tableau 4.2 détaille les résultats de cette évaluation. Nous avons fait varier x entre 1.0 m et 3.0 m, et avons placé le deuxième plan entre 10 cm et 1.5 m du mur, et avons mesuré la distance d moyenne, à partir des données exploitables. À partir de ces données, nous avons calculé l'écart moyen $\bar{\delta}$ (en cm) entre la distance véritable D et la mesure d mesurée. De même, nous avons calculé le pourcentage moyen $\bar{n}$ de frames "justes", pour lesquelles l'algorithme n'a détecté que deux plans.

x(m)	D(cm)	d(cm)	n	x(m)	D(cm)	d(cm)	n
1.0	10	-	-	1.5	10	12	47
	20	20	83		20	23	46
	30	28	88		30	31	28
	50	47	98		50	56	48
					100	101	60
2.0	10	10	22	3.0	10	14	25
	20	18	37		20	19	41
	30	36	15		30	34	11
	50	53	31		50	54	6
	100	102	45		100	102	30
	150	150	36		150	150	12
$\bar{\delta} = 2.1$ cm				$\bar{n} = 40.50$ %			

Tableau 4.2 – *Résultats des évaluations de l'algorithme de segmentation par croissance de régions pour 100 mesures. D représente la distance réelle entre les deux plans, d représente la distance mesurée entre les deux plans, n le nombre de frames justes, et x la distance de la tablette au plan le plus éloigné.*

4.5.2 Discussion

La précision moyenne de l'algorithme pour la restitution des mesures est de $\bar{\delta} = 2.1$ cm, et le pourcentage de prises de vues contenant exactement deux plans est en moyenne de $\bar{n} = 40.5\%$. L'algorithme est donc capable de replacer des plans à moins de 5 cm près. Cependant, on remarque dans plus de la moitié des cas, que nous avons des plans parasites (voir figure 4.9). Ces plans parasites sont le résultat de la génération de clusters sur les zones où les vecteurs normaux ne sont pas calculés de manière très précise, le plus souvent sur le bord des surfaces planes. Cela conduit à la génération de plans parasites le long de ces contours. Ces plans parasites ne sont souvent pas un problème car ils ne sont pas constants d'une prise de vue à l'autre, mais peuvent fausser les résultats.

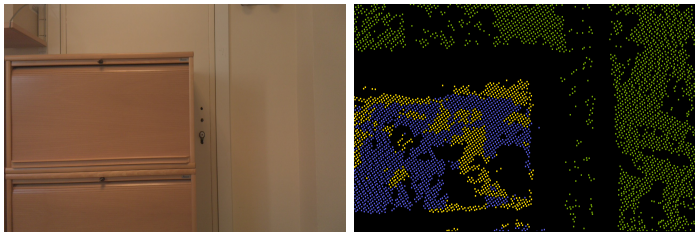


Figure 4.9 – *Des plans parasites (ici, en jaune) peuvent apparaître avec cette méthode de segmentation, cela est dû aux imprécisions de l'estimation des normales sur les zones à fort gradient de la carte de profondeur.*

4.6 Conclusion et travaux futurs

Nous avons mis en place un premier algorithme de détection des plans contenus dans un nuage de points 3D. La détection des plans se fait dans un intervalle de temps inférieur à 200 ms, ce qui nous permet de traiter les nuages de points renvoyés par un capteur de profondeur à la volée. Grâce à cet algorithme, nous sommes capables de positionner des plans avec une précision inférieure à 5 cm. Cependant, la détection des plans renvoie de nombreux plans parasites, qui peuvent avoir un impact lors de la reconstruction d'un modèle 3D global. Ces plans parasites sont dus aux imprécisions du capteur de profondeur lorsque la distance dépasse 2 m, et aux imprécisions du calcul des normales, surtout sur les bords des surfaces planes.

Pour palier ces problèmes, les prochains travaux porteront sur la mise en place d'un algorithme de segmentation 3D plus global, et l'utilisation d'une autre approche que par croissance de régions.

Chapitre 5

Segmentation planaire en temps réel à l'aide d'algorithmes de clustering

5.1 Introduction . . . 100
5.2 Description de l'algorithme . . . 101
5.2.1 Vue générale de l'algorithme mis en place . . . 101
5.2.2 Mise en forme des données . . . 102
5.2.3 Clustering sur les normales . . . 103
5.2.4 Classification finale . . . 106
5.2.5 Paramètres . . . 106
5.3 Implémentation . . . 107
5.3.1 Tri des données . . . 108
5.3.2 Clustering des normales . . . 108
5.4 Évaluations . . . 108
5.4.1 Évaluation de la précision et de la fiabilité . . . 108
5.4.2 Discussion . . . 109
5.4.3 Comparaison des deux algorithmes de segmentation planaire 109
5.5 Conclusion . . . 110

5.1 Introduction

Nous présentons ici une deuxième méthode de segmentation planaire en temps réel mise en place durant ces travaux de thèse. Cette méthode repose sur l'utilisation d'algorithmes de clustering génériques, sans faire d'hypothèse sur l'organisation des nuages de points en entrée.

Nous avons présenté au chapitre 4 un premier algorithme de détection des plans à la volée. Il s'agit d'un algorithme par croissance de régions qui exploite la structure ordonnée des nuages de points en entrée. Cet algorithme fonctionne de manière analogue à un algorithme de traitement d'image 2D, en remplaçant pour chaque pixel, l'intensité lumineuse par les quatre paramètres d'un plan 3D.

Cet algorithme permet une détection en temps réel de l'ensemble des plans contenus dans un nuage de points ordonné, et ne demande pas de précaution d'usage particulière. Cependant, cet algorithme peut produire des plans parasites au bord des zones où l'estimation des normales est de mauvaise qualité. Cela est dû à la construction de régions de manière incrémentale sans vision globale sur la structure générale du nuage de points.

Nous proposons donc une autre méthode de segmentation planaire, cette fois-ci en traitant le problème comme un problème de clustering, sans autre supposition sur la nature des données en entrée. Cela permet de considérer le nuage de points en entrée dans sa globalité, et d'obtenir une segmentation plus juste. En considérant que l'environnement intérieur à reconstruire est aligné sur une grille régulière (voir 1.4.1), les murs auront des orientations bien définies. Grâce à cela, nous savons que les vecteurs normaux des points constituant les murs appartiendront à un ensemble de classes connu à l'avance, ce qui permet de simplifier la segmentation.

L'algorithme de détection des plans mis en place est basé sur l'utilisation successive de deux algorithmes de clustering. Le premier sert à regrouper ensemble les points ayant des vecteurs normaux identiques à l'aide d'un algorithme de type *k-means*. Le second sépare les plans parallèles en effectuant un clustering sur une dimension.

Ce nouvel algorithme permet de segmenter une nuage de points 3D entre 5 à 10 fois plus rapidement, avec une meilleure précision qu'une approche par croissance de région. Cependant cet algorithme est moins générique et ne sera efficace que si le bâtiment à reconstruire est bâti selon une structure régulière. De plus, il faudra commencer la reconstruction face à un mur de manière à aligner le système de coordonnées du capteur de profondeur à la structure du bâtiment.

Dans ce chapitre, nous détaillons l'algorithme de segmentation mis en place (section 5.2), puis nous parlons de l'implémentation pratique de cet algorithme afin d'obtenir les performances requises (section 5.3). Enfin, nous présentons,

comme en 4.5, les résultats obtenus lors de l'évaluation de cet algorithme.

5.2 Description de l'algorithme

Nous détaillons ici le fonctionnement de l'algorithme de segmentation planaire mis en place. Cet algorithme repose sur des méthodes de clustering génériques et nécessite un tri des données en entrée. Afin de pouvoir s'exécuter correctement, la capture du bâtiment à reconstruire doit se faire en tenant la tablette face à un mur. De cette manière, un repère $\mathcal{R}_0$ lié à la position de la tablette à cet instant s'initialise et est aligné avec la structure du bâtiment. Ensuite, chaque nouvelle donnée 3D acquise par le capteur de profondeur devra être exprimée dans ce repère. Cela permet de mettre en place un algorithme de clustering semi-supervisé pour la détection des plans.

5.2.1 Vue générale de l'algorithme mis en place

Le fonctionnement de l'algorithme de segmentation 3D mis en place est synthétisé dans la figure 5.1. L'algorithme prend en entrée un nuage de points 3D et les vecteurs normaux qui y sont associés.

Pour toute cette partie, nous désignerons par $\mathcal{P}$ le nuage de points 3D à segmenter, dont les éléments ont leur coordonnées exprimées dans $\mathcal{R}_0$. Chaque élément P de $\mathcal{P}$ est constitué d'un point de l'espace $\boldsymbol{p} = [x, y, z]^\top$, du vecteur normal au plan local $\boldsymbol{n} = [n_x, n_y, n_z]^\top$ en ce point, et d'un réel d défini par

$$d = -(n_x.x + n_y.y + n_z.z).$$

Le quadruplet (n_x, n_y, n_z, d) constitue les paramètres du plan $\Pi_{\boldsymbol{p}}$ passant par $\boldsymbol{p}$ de vecteur normal $\boldsymbol{n}$.

La segmentation se fait en deux étapes. Tout d'abord, un premier algorithme de clustering est appliqué sur l'espace des normales, afin de regrouper les points ayant des vecteurs normaux identiques. Ce regroupement se fait à l'aide d'un algorithme de type *k-means*, qui converge rapidement vers une solution si l'on connaît d'avance les classes auxquelles les clusters à chercher sont susceptibles d'appartenir.

Dans le cas où le capteur de profondeur génère une carte de profondeur, le calcul des vecteurs normaux peut se faire d'une manière analogue à celle présentée lors du chapitre précédent (voir §4.3.2). Il suffit de calculer une carte $\mathcal{M}_{xyz}$ des positions indexées sur un espace 2D, et d'appliquer un algorithme de détection des normales.

Une fois ces premiers clusters générés, le deuxième algorithme de clustering va séparer les plans parallèles. Pour cela, il faut que les points contenus dans les

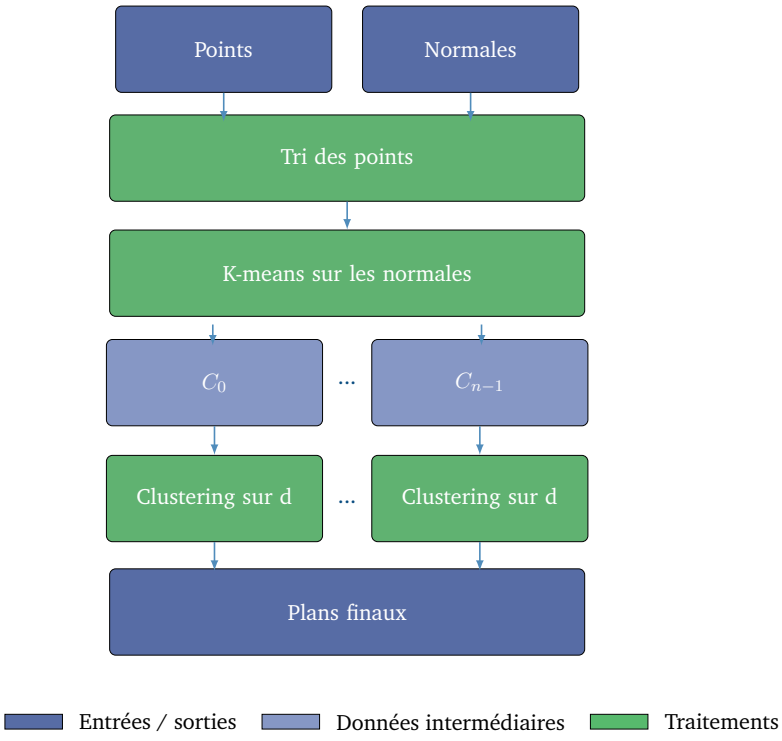


Figure 5.1 – *Vue générale de l'algorithme de segmentation planaire. La segmentation se fait en deux phases : tout d'abord, un algorithme de type k-means est employé pour séparer les points de normales identiques. Ensuite un algorithme de type DBSCAN à une dimension sur les clusters trouvés sépare les plans parallèles.*

différents clusters soient ordonné selon leur paramètre d croissant. Ce tri est en fait effectué avant de commencer la segmentation.

5.2.2 Mise en forme des données

5.2.2.1 Données en entrée

L'algorithme utilise un nuage de points 3D ainsi que les vecteur normaux associés pour effectuer la segmentation. Dans le cas d'un nuage de points ordonné, nous pouvons utiliser le même algorithme de détection des normales que précédemment (voir §4.3.2.1). En utilisant une estimation de la position de la tablette lors de la capture, les coordonnées des points et des vecteurs normaux sont converties dans le repère $\mathcal{R}_0$ défini au début de la capture. Une fois les points et les vecteurs normaux calculés, les paramètres de plans associés à chaque points sont calculés.

5.2.2.2 Tri des données

L'algorithme de clustering permettant de séparer les plans parallèles nécessite que les clusters de points de mêmes normales soient ordonnés par d croissant.

Pour cela, nous rangeons les différents descripteurs, caractérisés par leur position spatiale (x, y, z) et les paramètres du plan local associé (n_x, n_y, n_z, d), dans un arbre binaire. Le critère de comparaison de deux descripteurs étant la valeur de d. La liste triée est alors obtenue en parcourant cet arbre par la gauche de manière récursive. La complexité moyenne de cette méthode de tri est de $O(nlog(n))$.

Afin d'avoir le tri le plus efficace, il vaut mieux trier l'ensemble des données dès le départ, plutôt que de trier chacun des clusters séparément. Cela permet de diminuer les chances d'avoir un arbre binaire déséquilibré, qui conduirait à une complexité en $O(n^2)$ pour l'algorithme, et le tri peut alors plus facilement être réparti sur plusieurs threads qui auront une charge de travail équivalente.

L'algorithme de tri (voir algorithme 5) procède en deux étapes, tout d'abord, un arbre binaire $\mathcal{T}$ est construit en insérant l'ensemble des $P \in \mathcal{P}$, en utilisant d comme élément de comparaison.

Algorithme 5 : *Algorithme de tri des points.*

1: **for all** $P \in \mathcal{P}$ **do**
2: $\quad \mathcal{T}.insert(\boldsymbol{P})$
3: **end for**
4: $sort\_data(\mathcal{T}, \mathcal{P}')$

Le tri se fait ainsi par construction de l'arbre, il suffit donc ensuite de parcourir l'arbre avec la fonction $sort\_data()$ pour obtenir la liste $\mathcal{P}'$ des éléments triés.

L'algorithme 6 détaille la fonction $sort\_data()$, qui parcourt l'arbre $\mathcal{T}$ en profondeur en commençant par la gauche et qui effectue le tri final.

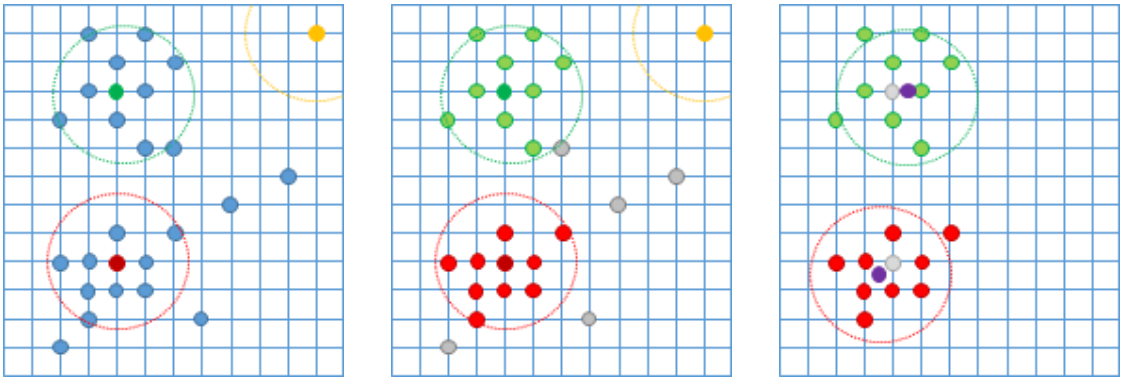
Algorithme 6 : *Tri récursif.*

1: $\mathcal{P}' \leftarrow \emptyset$
2: **if** $T = \emptyset$ **then**
3: $\quad$ **return**
4: **end if**
5: $sort\_data(\mathcal{T}.left, \mathcal{P}')$
6: $\mathcal{P}'.append(\mathcal{T}.value)$
7: $sort\_data(\mathcal{T}.right, \mathcal{P}')$

5.2.3 Clustering sur les normales

L'algorithme de clustering des vecteurs normaux est un algorithme de type *k-means* (illustré fig. 5.2). Les algorithmes de type *k-means* consistent à diviser un ensemble de données en entrée en k classes différentes. Il est donc nécessaire de

connaître à l'avance le nombre de classes que contiendra l'ensemble de données à classifier.



(a) (b) (c)

Figure 5.2 – *Illustration 2D de l'algorithme de clustering des normales mis en place. Cet exemple montre le cas où l'on attend au plus trois clusters. À la première itération (a), l'ensemble des points situés à une distance inférieure à ϵ (cercles colorés) d'un centre sont rattachés au centre le plus proche. Ensuite (b), les points qui ne sont rattachés à aucun cluster sont retirés (en gris), et les éventuels centres (ici en jaune) auxquels aucun point n'a été affecté sont supprimés pour la prochaine itération. Les centres des clusters sont recalculés (en violet), et une nouvelle itération est faite avec ces nouvelles données (c).*

Afin d'accélérer la convergence de l'algorithme, on peut sélectionner les k centres de manière à ce qu'ils soient déjà chacun dans un des k clusters à trouver. C'est ce que nous avons fait en considérant la structure du bâtiment dont nous souhaitons effectuer une capture. Dans le cas de la capture 3D d'un bâtiment, on peut considérer que les plans qui nous intéressent (les murs, le sol, et le plafond), n'ont que six orientations possibles si la structure du bâtiment est alignée selon un repère orthonormal (voir §1.4.1).

L'algorithme 7 décrit en détail la méthode de clustering que nous avons mise en place.

Cet algorithme regroupe les vecteurs normaux à partir de six classes $(k = 6)$ de vecteurs normaux $\boldsymbol{\mu_i}, i \in [\![0, k-1]\!]$, correspondant aux 6 orientations possibles selon les trois axes $\boldsymbol{u}_x, \boldsymbol{u}_y$ et $\boldsymbol{u}_z$. Afin d'être efficace, il faut alors s'assurer que la capture débute lorsque le repère du capteur de profondeur est aligné avec celui de l'environnement intérieur à capturer. Les centres des clusters seront ensuite mis à jour pour prendre en compte l'orientation réelle des plans, notamment pour corriger une erreur d'alignement au départ.

Nous avons adapté l'algorithme *k-means* original pour détecter au plus six classes de vecteurs normaux, mais éventuellement, en détecter moins.

Soit $\boldsymbol{\lambda_i}$ l'ensemble des centres des clusters finaux, et $\mathcal{P}_l$ l'ensemble des éléments de $\mathcal{P}$ étiquetés. Chaque élément $\boldsymbol{\lambda_i}$ est initialisé avec $\boldsymbol{\mu_i}$. $\mathcal{P}_l$ est initialisé avec les

Algorithme 7 : *Algorithme k-means pour le clustering des normales.*

for $i \in [\![0, k-1]\!]$ **do**
  $\boldsymbol{\lambda_i} \leftarrow \boldsymbol{\mu_i}$
end for
while $j < N$ **do**
  for all $(\boldsymbol{p}, \boldsymbol{n}, d, l) \in \mathcal{P}_l$ **do**
    $\Delta_{min} \leftarrow \epsilon_n, l \leftarrow +\infty$
    for $i \in [\![0, k-1]\!]$ **do**
      if $\boldsymbol{\lambda_i} \neq 0_{1,3}(\mathbb{R})$ **then**
        $\Delta \leftarrow \delta(\boldsymbol{p}, \boldsymbol{\lambda_i})$
        if $\Delta < \Delta_{min}$ **then**
          $l \leftarrow i, \Delta_{min} \leftarrow \Delta$
        end if
      end if
    end for
  end for
  for $i \in [\![0, k-1]\!]$ **do**
    $\mathcal{P}'_l \leftarrow \{(\boldsymbol{p}, \boldsymbol{,} d, n) \in \mathcal{P}_l, l = i\}$
    if $Card(\mathcal{P}'_l) \neq 0$ **then**
      $\boldsymbol{\lambda_i} \leftarrow \frac{\Sigma_{(\boldsymbol{p},\boldsymbol{,}d,n)\in\mathcal{P}'_l}\boldsymbol{n}}{Card(\mathcal{P}'_l)}$
    else
      $\boldsymbol{\lambda_i} \leftarrow 0_{1,3}(\mathbb{R})$
    end if
  end for
end while

éléments de $\mathcal{P}$ étiquetés par $+\infty$.

Pour chaque itération de l'algorithme une première boucle parcourt l'ensemble des éléments $(\boldsymbol{p}, \boldsymbol{n}, d, l)$ de $\mathcal{P}_l$ et va les comparer à chacun des centres $\boldsymbol{\lambda_i}$. Si la distance euclidienne $\Delta = \delta(\boldsymbol{n}, \boldsymbol{\lambda_i})$ entre $\boldsymbol{n}$ et $\boldsymbol{\lambda_i}$ est inférieure à un certain seuil ϵ_n, alors i est retenue comme valeur candidate pour l. La valeur i_0 qui minimise les labels candidats est retenue pour l, s'il en existe, sinon, l est mis à $+\infty$.

Une fois tous les éléments de $\mathcal{P}_l$ étiquetés, chaque $\boldsymbol{\lambda_i}$ est mis à jour avec la valeur moyenne des vecteurs normaux des éléments de $\mathcal{P}_l$ ayant pour label i. Si aucun élément de $\mathcal{P}_l$ n'est étiqueté par i, $\boldsymbol{\lambda_i}$ est mis à $O_{1,3}(\mathbb{R})$, et n'est plus pris en compte pour les prochaines itérations.

Ce processus se répète jusqu'à convergence de l'algorithme, ou lorsqu'un nombre maximal de répétitions N est atteint.

5.2.4 Classification finale

Une fois les points étiquetés, ils sont rangés dans différents clusters $\{\mathcal{K}_i, i \in [\![0, k-1]\!]\}$ en fonction de leurs classes d'appartenance. Puis, nous appliquons un deuxième algorithme de clustering sur chacun des clusters $\mathcal{K}_i$ pour séparer les plans parallèles. Ce processus se résume très bien sur l'algorithme 8.

Algorithme 8 : *Séparation des plans parallèles dans un cluster.*

$\Psi \leftarrow \emptyset$
$\Pi \leftarrow \emptyset$
$\Pi \leftarrow P_{i,0}$
for all $j \in [\![1, n-1]\!]$ **do**
  if $|d_{i,j} - d_{i,j-1}| \leq \epsilon_d$ **then**
    $\Pi \leftarrow \Pi + P_{i,j}$
  else
    $\Psi \leftarrow \Psi + \Pi$
    $\Pi \leftarrow P_{i,j}$
  end if
end for

L'algorithme parcourt l'ensemble des points $P_{i,j}(\boldsymbol{p_{i,j}}, \boldsymbol{n_{i,j}}, d_{i,j})$ de $\mathcal{K}_i$, et les compare avec le point précédent $P_{i,j-1}$. Si l'écart $|d_{i,j} - d_{i,j-1}|$ est inférieur au seuil ϵ, les points sont supposés appartenir à un même plan Π. Dès que cet écart est supérieur à un seuil ϵ_d, un nouveau plan est créé et ajouté à l'ensemble Ψ des plans détectés, et les nouveaux points seront maintenant placés dans celui-ci.

5.2.5 Paramètres

Le tableau 5.1 présente les différents paramètres utilisés pour l'algorithme de segmentation planaire.

Étape	*Paramètre*	*Valeur*	*Commentaire*
Clustering des normales (§5.2.3)	N	5	Max d'itérations
	ϵ_n	0.10	Distance maximale pour intégrer un point à un cluster
Clustering 1D (§5.2.4)	ϵ_d	0.03	Distance pour séparer deux plans parallèles

Tableau 5.1 – *Paramètres pour la segmentation planaire basée sur le clustering.*

Pour le clustering des normales, nous avons fixé le seuil de distance ϵ_n entre un vecteur normal et le centre d'un cluster, à 0.10. Cette valeur assez faible permet d'éviter d'ajouter des points dont le vecteur normal ne serait pas aligné

sur le repère initial. L'algorithme converge très rapidement puisqu'il est aidé dès le départ renseignant les centres des clusters que nous souhaitons trouver, aussi nous avons limité le nombre d'itérations maximum N à 5.

Pour l'algorithme de clustering 1D qui va séparer les plans parallèles, nous avons fixé un seul ϵ_d à 0.03, ce qui permet de distinguer des plans qui sont séparés d'au moins 3 cm. Ce paramètre est limité par la précision des nuages de points renvoyés par le capteur de profondeur et celle de l'algorithme. Le résultat final de l'algorithme de détection des plans est visible sur la figure 5.3.

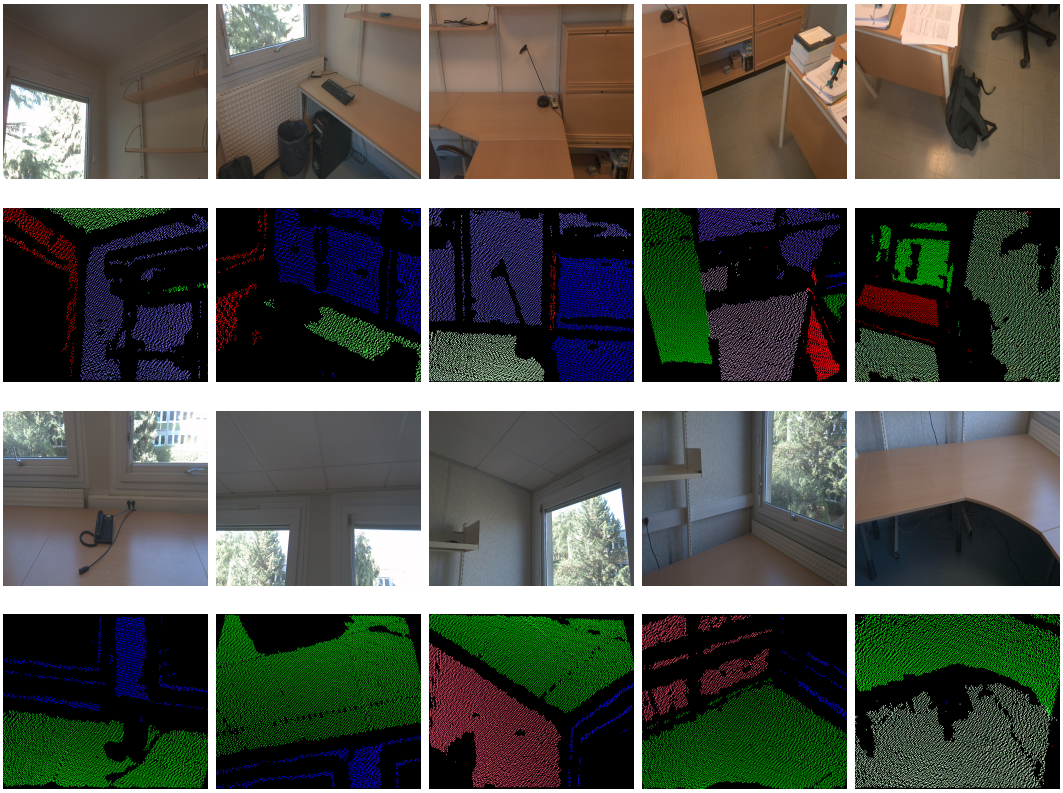


Figure 5.3 – *Exemples de détection des plans par clustering.*

5.3 Implémentation

Nous détaillons ici notre implémentation de l'algorithme de segmentation planaire mis en place sur la tablette cible (voir en 1.4.3). L'organisation des tâches de travail est similaire à celle utilisée pour le précédent algorithme de segmentation (voir section 4.4). Deux threads indépendants sont mis en place, l'un s'occupe de la mise en forme des données, et l'autre s'occupe de la segmentation à proprement parler. Le thread de mise en forme des données s'occupe de calculer les vecteurs normaux à la surface en chaque point $\boldsymbol{p}$ d'un nuage de points 3D entrant. Ensuite, la liste des points et des vecteurs normaux associés est mise à disposition du thread de segmentation.

Nous allons ici nous concentrer sur l'implémentation pratique des algorithmes de tri des données et de clustering des normales, qui sont les deux algorithmes nécessitant le plus de calculs.

5.3.1 Tri des données

Nous avons divisé les données d'entrée en deux ensembles, afin d'effectuer le tri sur deux threads différents. Pour cela, nous calculons $\bar{d}$, la valeur moyenne de d pour l'ensemble des points de $\mathcal{P}$. Les données sont donc séparés en deux groupes, celles dont d est inférieur à $\bar{d}$ et les autres. Deux arbres binaires indépendants sont construits et triés, chacun utilisant un des deux jeux de données. Ces deux tâches se réalisent alors en parallèle.

5.3.2 Clustering des normales

Le clustering des normales parcourt de manière indépendante l'ensemble des points de $\mathcal{P}$ pour les classifier, il est donc facile de diviser le travail sur plusieurs threads de calcul. En pratique, nous avons divisé cette tâche sur quatre threads, et avons pu utiliser les extensions SIMD du processeur (voir annexe D).

Nous n'avons pas rencontré de problème de performances particulier pour cet algorithme, qui est très rapide à s'exécuter. En pratique, cet algorithme s'exécute entre 5 et 10 fois plus rapidement que l'algorithme précédent.

5.4 Évaluations

Nous avons effectué les mêmes évaluations de fiabilité et de détection que pour l'algorithme de segmentation planaire précédent (voir section 4.5). Nous présentons tout d'abord les résultats de cette évaluation, puis comparons ces résultats par rapport au premier algorithme de segmentation planaire mis en place.

5.4.1 Évaluation de la précision et de la fiabilité

Les résultats de l'évaluation de l'algorithme de segmentation planaire que nous venons de décrire sont présentés dans le tableau 5.2. Nous avons mené ces évaluations dans les mêmes conditions que pour l'algorithme présenté au chapitre 4. Nous avons placé deux plans parallèles devant la tablette, et avons fait varier la distance x de la tablette par rapport au plan le plus éloigné, et la distance inter plans D.

x(m)	D(cm)	d(cm)	n	x(m)	D(cm)	d(cm)	n
1.0	10	10	45	1.5	10	13	45
	20	22	100		20	23	100
	30	32	85		30	31	99
	50	50	100		50	55	99
					100	101	99
2.0	10	14	39	3.0	10	20	12
	20	22	21		20	-	-
	30	32	97		30	34	85
	50	51	99		50	51	98
	100	100	100		100	100	100
	150	150	99		150	150	99
$\bar{\delta} = 1.46$ cm				$\bar{n} = 77.15$ %			

Tableau 5.2 – *Résultats des évaluations de l'algorithme de segmentation avec k-means pour 100 mesures. D représente la distance réelle entre les deux plans, d représente la distance mesurée entre les deux plans, n le nombre de frames justes, et x la distance de la tablette au plan le plus éloigné.*

5.4.2 Discussion

Comme pour l'algorithme précédent, on observe de bons résultats pour la mesure de la distance inter-plans D, avec une erreur moyenne ($\bar{\delta} = 1.46$) bien inférieure à 5 cm. On constate également un très grand pourcentage d'observations justes (sans plans parasites), $\bar{n} = 77.15\%$ lorsque D augmente, mais ce pourcentage moyen reste faible lorsque les deux plans sont assez proches. Cette baisse de fiabilité augmente avec la distance x du capteur aux plans, et semble liée à des problèmes de précision du capteur de profondeur utilisé.

5.4.3 Comparaison des deux algorithmes de segmentation planaire

La figure 5.4 compare les valeurs de n pour les deux algorithmes de détection des plans (chapitre 4 et ce chapitre) mis en place, en fonction de x et de D.

On constate une nette amélioration du pourcentage de mesures non bruitées avec l'utilisation de clustering pour effectuer la segmentation planaire. Cela confirme que l'utilisation d'une approche de segmentation traitant les données dans leur intégralité mène à une meilleure segmentation. On constate cependant dans les deux cas que la segmentation produit de nombreux plans parasites lorsque D est faible. Cela est dû à l'algorithme de détection des normales qui peut produire des valeurs erronées si le gradient de position à la frontière des

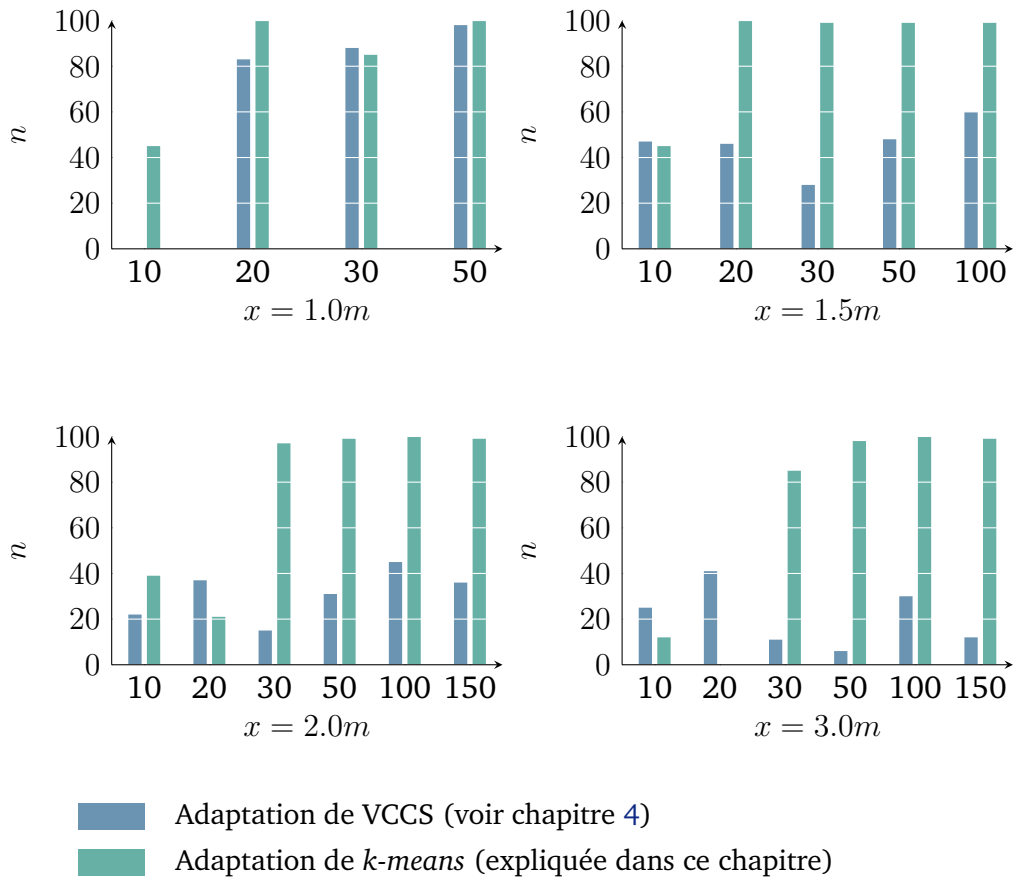


Figure 5.4 – *Comparaison du pourcentage de frames justes n (en ordonnées) des deux algorithmes de segmentation planaire en fonction de la distance D (en abscisses) pour chaque configuration de x différente.*

deux surfaces n'est pas assez prononcé. De meilleurs résultats devraient être possibles en utilisant une carte de profondeur de meilleure résolution, mais entraînera plus de calculs.

5.5 Conclusion

Nous avons mis en place un deuxième algorithme de détection des plans dans un nuage de points 3D. Il utilise une approche de type clustering et effectue une segmentation planaire sans faire de supposition sur la manière dont les données d'entrée sont organisées. La segmentation se fait en deux étapes : le clustering des normales, qui est adapté de l'algorithme *k-means*, et la séparation des plans parallèles. Afin d'accélérer la convergence de l'algorithme *k-means*, nous avons supposé que les vecteurs normaux intéressants sont alignés selon les trois axes d'un repère orthonormé global à l'ensemble des captures. Cela implique de commencer la capture en alignant le capteur de profondeur avec la structure du bâtiment dont on veut effectuer la capture. De plus, cela suppose que le bâtiment en question est

lui aussi bâti selon un repère orthonormal, mais cela correspond aux hypothèses que nous avons fixé en 1.4.1.

On obtient alors un algorithme de segmentation beaucoup plus rapide, et l'utilisation d'une approche de type clustering nous permet d'avoir une détection des plans beaucoup moins bruitée qu'avec l'algorithme de segmentation par croissance de région précédent. Les imperfections qui subsistent lorsque deux plans sont proches sont dues à l'algorithme de détection des normales que nous avons utilisé pour ces deux algorithmes et qui a tendance à lisser la carte de profondeur. Cependant, nous sommes capables de mesurer des distances avec une précision moyenne inférieure à 5 cm dans tous les cas, cet algorithme répond très bien aux contraintes énoncées au paragraphe 1.4.2.

Ce deuxième algorithme de segmentation représente un bon compromis entre la précision et les performances de calcul, et devrait se révéler efficace pour la reconstruction 3D d'un bâtiment vide.

Chapitre 6

Vers la reconstruction en temps réel d'un modèle 3D éditable

6.1 Introduction 114
6.2 Identification de plans identiques lors de différentes prises de vue 115
6.2.1 Stockage en mémoire 115
6.2.2 Correction des dérives 117
6.3 Construction incrémentale des murs 120
6.3.1 Mise à jour de l'enveloppe de chaque plan 121
6.3.2 Classification finale 122
6.4 Finalisation de la maquette 123
6.4.1 Détection des ouvrants à l'aide des images RGB 123
6.4.2 Export de la maquette finale 124
6.5 Évaluations 124
6.5.1 Évaluation de la maquette générée 125
6.5.2 Discussion 126
6.6 Conclusion 126

6.1 Introduction

Dans les chapitres précédents, nous avons montré qu'il était possible de détecter les plans contenus dans un nuage de points 3D renvoyés par un capteur de profondeur en temps réel sur une tablette. Ces algorithmes utilisent uniquement le CPU de la tablette. Nous avons en particulier mis en place un algorithme de segmentation planaire qui utilise des techniques de clustering pour effectuer cette segmentation. Cet algorithme est particulièrement rapide pour la détection des plans 3D disposés le long des axes d'un repère orthonormé. Par rapport aux évaluations que nous avons conduites précédemment, nous avons retenu cet algorithme de segmentation pour la suite de nos travaux.

Le couplage de cet algorithme de détection des plans et d'un algorithme d'odométrie pour suivre les déplacements de la tablette nous permet d'envisager de reconstruire un modèle 3D en temps réel sans générer de maillage 3D intermédiaire. En effet, le suivi des mouvements de la tablette permet d'identifier des plans identiques lors de prises de vues différentes. Il est possible de mettre à jour les dimensions de ces plans lors d'observations successives, il suffit ensuite d'identifier les plans qui sont associés à des éléments structuraux.

De cette manière, seules les informations géométriques des éléments structuraux sont sauvegardées et mises à jour. Le volume de données à enregistrer est donc considérablement réduit et le rendu 3D associé également. De plus, ce modèle 3D peut être directement exportable sans passer par une phase de segmentation et de classification.

La construction de ce modèle 3D en temps réel permet également d'exploiter les images renvoyées par une caméra couleur afin d'identifier les ouvrants contenus dans la scène 3D.

Nous présentons ici les premiers travaux effectués pour générer un modèle 3D éditable de manière incrémentale à partir des données d'un capteur de profondeur, d'un algorithme d'odométrie et d'une caméra couleur intégrés à une tablette. L'utilisateur a juste besoin de se déplacer dans l'environnement physique avec la tablette, et le modèle 3D est généré et mis à jour en temps réel.

Nous détaillons tout d'abord comment nous identifions des plans identiques sur des prises de vue différentes (section 6.2). Puis, nous détaillons le processus d'identification et de mise à jour des murs (section 6.3), et les travaux restants pour générer un modèle 3D complète (section 6.4). Enfin, nous présentons les résultats qu nous avons obtenus pour l'évaluation des mesures du modèle 3D exporté par rapport aux dimensions réelles (section 6.5).

6.2 Identification de plans identiques lors de différentes prises de vue

Nous nous plaçons ici après la phase de segmentation planaire, lorsqu'un nouveau nuage de points a été segmenté en clusters représentant chacun un plan. Grâce à la connaissance de la position de la tablette, tous les objets 3D sont exprimés par rapport à un même repère $\mathcal{R}_0$ initialisé au début de la capture. Nous sommes alors capables d'identifier des plans identiques sur des prises de vue différentes. Pour cela, nous associons un identifiant unique à chaque plan, calculé à partir d'histogrammes sur ses paramètres, et qui permet une estimation robuste des paramètres d'un plan. Il sert ensuite de clé pour référencer ce plan dans une table de hachage.

En identifiant des plans similaires, on peut calculer de manière incrémentale une enveloppe les englobant. Cela permet alors de connaître leur extension spatiale, et leurs dimensions. Une fois tous les plans de la scène détectés, les plans sont étiquetés comme étant le sol, le plafond, des murs, ou du bruit.

L'algorithme que nous présentons ici a été implémenté en utilisant l'algorithme de segmentation planaire présenté au chapitre 5.

6.2.1 Stockage en mémoire

Soit Π un cluster construit lors de la phase de segmentation planaire. Π est constitué de triplets P contenant un point $\boldsymbol{p} = [x, y, z]^\top$, le vecteur normal associé à ce point $\boldsymbol{n} = [n_x, n_y, n_z]^\top$, et un réel d de manière à ce que n_x, n_y, n_z, et d caractérisent le plan passant par $\boldsymbol{p}$ et de vecteur normal $\boldsymbol{n}$. Les coordonnées spatiales sont toutes converties dans un même repère défini au début de la capture. Dans une configuration idéale, chaque élément P de Π dispose exactement des mêmes paramètres de plan. En pratique, ces paramètres sont distribués selon une loi normale centrée autour des véritables paramètres du plan géométrique décrit par les éléments de Π.

Nous allons donc identifier un plan par rapport à des histogrammes de chacun des paramètres n_x, n_y, n_z, et d des éléments de Π. Ces histogrammes serviront à donner un identifiant unique au plan décrit par Π.

6.2.1.1 Histogrammes

Soit $\mathcal{E}$ un ensemble dénombrable de $\mathbb{R}$, et $\tau \in \mathbb{R}$. On définit l'histogramme $H_\tau(\mathcal{E})$ par $\forall i \in \mathbb{Z}$:

$$H_\tau(\mathcal{E})(i) = Card(\{x \in \mathcal{E} / x \in [i\tau, (i+1)\tau]\}), \tag{6.1}$$

Sa valeur moyenne $\mu_{i,\tau}(\mathcal{E})$, avec $i \in \mathbb{N}$ est calculée par :

$$\mu_{i,\tau}(\mathcal{E}) = \frac{\Sigma_{p=-i}^{i}(pH_\tau(\mathcal{E})(p))}{\Sigma_{p=-i}^{i}(H_\tau(\mathcal{E})(p))} \tag{6.2}$$

Soit Π un plan précédemment détecté, et n_x, n_y, n_z, d, l'ensemble des paramètres de plan associés à chacun des descripteurs qu'il contient. On calcule alors l'histogramme associé $H_\epsilon(D)$ ainsi qu'un identifiant (ID) $I_{i,\tau}(\Pi) \in \mathbb{Z}^4$ que l'on définit par :

$$I_{i,\tau}(\Pi) = (\mu_{i,\tau}(n_x), \mu_{i,\tau}(n_y), \mu_{i,\tau}(n_z), \mu_{i,\tau}(d)) \tag{6.3}$$

Cet identifiant est unique pour chaque plan, et dépend directement de ses paramètres.

Les histogrammes ont été calculés en utilisant les paramètres $\tau = 0.01$, et $i = 100$, on peut donc par la suite déduire d'un identifiant les paramètres de plan à 0.01 près. Ainsi, pour simplifier les notations, nous appellerons les différents histogrammes calculés H_x, H_y, H_z et H_d, et l'identifiant associé $I_p = I_{i,\tau}(P)$. Les plans sont alors identifiés par leur histogramme, qui permet de leur attribuer un identifiant unique et robuste au bruit, ainsi que par un poids ω, qui est représentatif de leur taille. Grâce à ces paramètres, nous sommes en mesure de les mettre en correspondance avec les autres plans détectés précédemment.

6.2.1.2 *Stockage et mise en correspondance des plans*

Les histogrammes calculés pour chaque cluster permettent de leur attribuer un identifiant unique comme illustré à la figure 6.1. Cet identifiant, qui dépend uniquement des paramètres du plan permet d'identifier deux plans identiques.

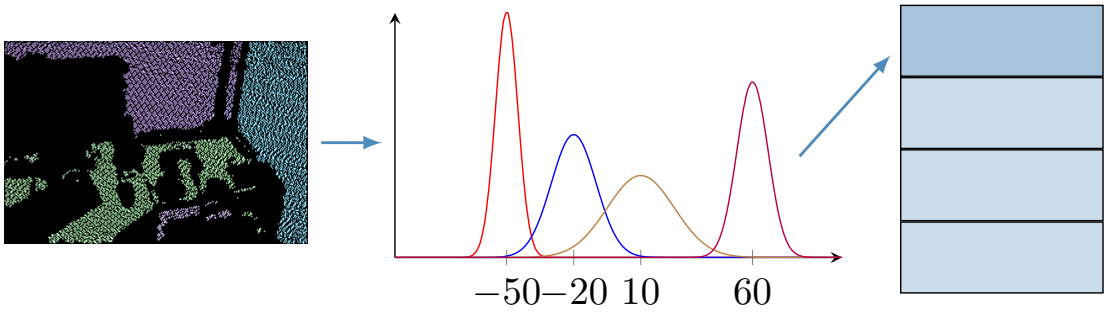


Figure 6.1 – *Mise à jour de la liste des plans. Pour chaque plan, on calcule un histogramme de ses quatre paramètres. Ces histogrammes permettent d'extraire un identifiant unique (ici :* $(-50, -20, 10, 60)$*) qui servira de clé pour stocker le plan dans une table de hachage.*

Afin de ranger efficacement les plans, et de vérifier si un plan similaire existe, nous utilisons une table de hachage. Nous avons en fait repris le principe de hachage spatial de Niessner et al.[97] (voir section 2.4.2). Nous définissons pour

cela une fonction de hachage de hachage prend comme entrée un identifiant $ID = (Id_{n_x}, Id_{n_y}, Id_{n_z}, Id_d)$, et renvoie un indice i défini par :

$$i = (p_0.Id_{n_x} \oplus p1.Id_{n_y} \oplus p_2.Id_{n_z} \oplus p_3.Id_d)[N] \tag{6.4}$$

où p_0, p_1, p_2, et p_3 sont trois nombres premiers, et N est la taille maximale de la table de hachage et $\oplus$ représente le ou exclusif. Chaque plan a donc une position précise dans cette table de hachage dépendant de ses paramètres.

L'algorithme est ainsi capable d'identifier l'ensemble des plans contenus dans un nuage de points 3D, et de les mettre en correspondance avec les plans précédemment détectés. Deux exemples d'application de l'algorithme sont présentés sur la figure 6.2. On voit que des plans similaires (sol, murs, bureau, etc ...) ont la même couleur (dépendant de l'identifiant calculé) sur les différentes prises de vues.

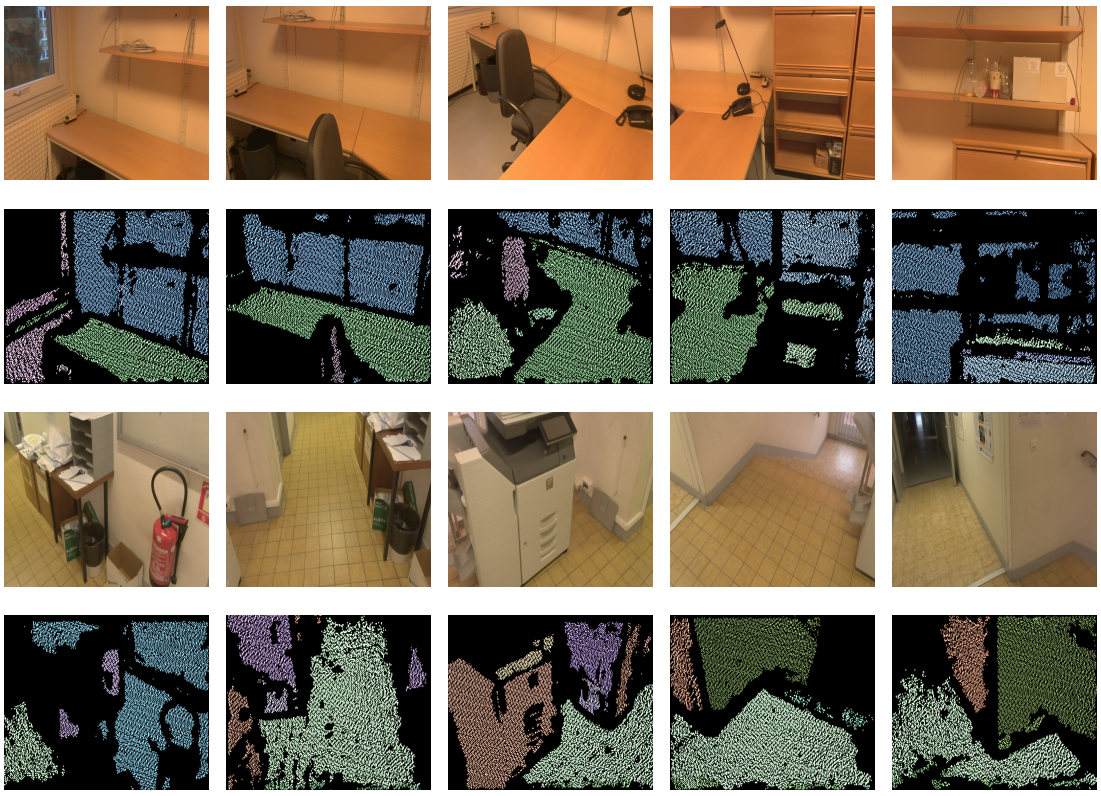


Figure 6.2 – *Identification de plans identiques sur plusieurs prises de vue différentes : deux mêmes plans apparaissent de la même couleur (la couleur dominante dépend de la valeur de la normale de chaque plan, et l'intensité distingue deux plans parallèles).*

6.2.2 Correction des dérives

Aussi précis que le dispositif de suivi des déplacements puisse être, il subsiste des erreurs de positionnement qui peuvent engendrer des imprécisions importantes pour la mise en correspondance des plans.

Nous avons donc implémenté une variante de l'algorithme ICP [8], afin de corriger les erreurs de positionnement entre deux acquisitions de données successives. L'algorithme ICP est présenté en section B.1. Il consiste à chercher de manière itérative un ensemble de points correspondants entre un nuage de points source et un nuage de points cible, puis de trouver une transformation qui minimise une fonction de coût entre ces correspondances. Ce processus est réitéré jusqu'à ce que l'algorithme converge, ou qu'un nombre maximal d'itérations a été effectué.

Ces deux étapes de recherche de correspondances, et d'alignement des correspondances peuvent être coûteuses en termes de ressources de calcul.

L'utilisation de nuages de points ordonnés et la connaissance des normales en ces points nous permet de simplifier les étapes de recherche de correspondances et d'alignement.

6.2.2.1 Recherche des correspondances

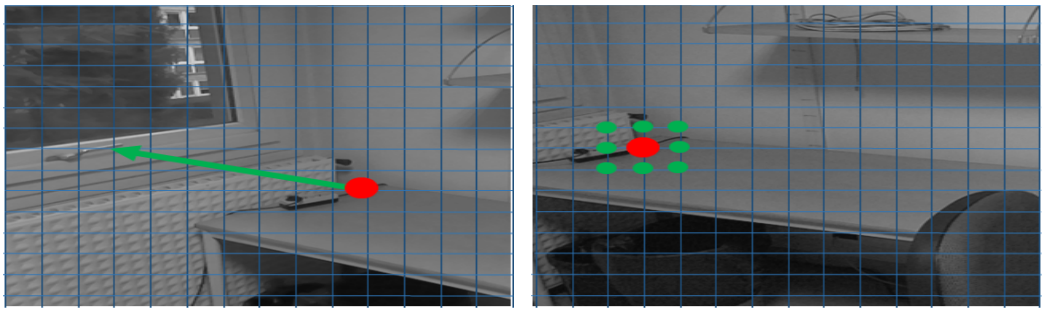
Notons $\mathcal{P}(t)$ le nuage de points acquis à chaque instant t, et $\mathcal{P}(t+1)$ le nuage de points acquis l'instant d'après. Dans notre exemple, chaque élément P de $\mathcal{P}$ est constitué d'une position $\boldsymbol{p} = [x, y, z]^\top$, d'une couleur $\boldsymbol{c} = [r, g, b]^\top$ et d'un vecteur normal $\boldsymbol{n} = [n_x, n_y, n_z]^\top$.

Considérant que deux nuages de points $\mathcal{P}(t+1)$ et $\mathcal{P}(t)$ sont déjà presque alignés, on peut supposer que les points correspondants entre ces deux nuages de points ne sont pas trop éloignés. La recherche de correspondances consiste donc à chercher pour chaque point 3D P appartenant à $\mathcal{P}(t+1)$, un point correspondant $P' \in \mathcal{P}(t)$, pour lequel une certaine distance $\delta(P, P')$ est inférieure à un seuil ϵ. Cette distance peut être une distance euclidienne entre les coordonnées spatiales ou toute autre distance permettant d'évaluer la similitude entre deux éléments de $\mathcal{P}$ et $\mathcal{P}'$. On recherche donc P' parmi les k plus proches voisins de $\mathcal{P}(t)$ à P.

En utilisant des nuages de points ordonnés, et en considérant de faibles déplacements de la tablette entre t et $t+1$, on peut considérer que pour chaque point $P' \in \mathcal{P}(t+1)$, d'indice (u, v), son point correspondant dans $\mathcal{P}(t)$ est situé dans le voisinage du point $P \in \mathcal{P}(t)$ d'indice (u, v). Afin d'affiner cette recherche, on peut aussi estimer le déplacement d'indices entre les deux nuages de points $\mathcal{P}(t+1)$ et $\mathcal{P}(t)$, de manière à définir un couple d'offsets (u_0, v_0).

Dans ce cas, soit $P \in \mathcal{P}(t)$ d'indice (u, v), on cherchera son correspondant dans le voisinage du point $P' \in \mathcal{P}(t+1)$ d'indice $(u - u_0, v - v_0)$, comme illustré à la figure 6.3.

L'algorithme 9 détaille le processus de construction d'un ensemble de correspondances $\mathcal{C}$ entre P et P'. Pour tout point P de $\mathcal{P}(t+1)$, on regarde l'ensemble des N points P' de $\mathcal{P}(t)$ situés au voisinage du point d'indice (u_0, v_0). On note cet ensemble $\mathcal{W}_{\mathcal{P},u_0,v_0}$.



(a) (b)

Figure 6.3 – *Recherche de correspondances. Pour un point situé en* $(10, 6)$ *sur a) (en rouge) on va déterminer la zone de recherche de son point correspondant dans b) compte tenu d'une estimation du déplacement d'un offset d'indices* $(-7, +12)$ *(flèche verte). On cherchera donc s'il y a un point correspondant dans b) au voisinage du point à l'indice* $(3, 8)$.

Algorithme 9 : *Recherche de correspondances entre deux nuages de points ordonnés.*

$\mathcal{C} \leftarrow \emptyset$
for all $P \in \mathcal{P}(t+1)$ **do**
  $\Delta_{min} \leftarrow \epsilon$
  for all $P' \in \mathcal{W}_{\mathcal{P},u_0,v_0}$ **do**
    $\Delta \leftarrow \delta(P, P')$
    $P_0 \leftarrow \emptyset$
    if $\Delta < \Delta_{min}$ **then**
      $\Delta_{min} = \Delta$
      $P_0 \leftarrow P'$
    end if
  end for
  if $\Delta_{min} < \epsilon$ **then**
    $\mathcal{C} \leftarrow \mathcal{C} + (P, P')$
  end if
end for

Deux points P et P' sont considérés comme correspondants s'ils ne sont pas trop éloignés spatialement, et si leurs caractéristiques sont les mêmes. Nous calculons donc une similitude $\delta(P, P')$, définie par :

$$\delta(P, P') = \frac{\omega_p \delta_p(\boldsymbol{p}, \boldsymbol{p'}) + \omega_c \delta_c(\boldsymbol{c}, \boldsymbol{c'}) + \omega_n \delta_n(\boldsymbol{n}, \boldsymbol{n'})}{\omega_p + \omega_c + \omega_n} \tag{6.5}$$

où $\delta_p, \delta_c, \delta_n$ représentent les distances euclidiennes dans l'espace pour les positions, les couleurs et les normales de P et P', et ω_p, ω_c, te ω_n un coefficient de pondération associé à chacune de ces distances. Cette distance prend en compte la proximité spatiale des deux points, mais aussi la similitude de couleur ainsi que la similitude des normales à ces deux points.

6.2.2.2 Alignement des correspondances

L'algorithme d'alignement des correspondances est présenté sur la figure 10. Une fois un ensemble de points correspondants $\mathcal{C}$ trouvé de deux nuages $\mathcal{P}(t)$ et $\mathcal{P}(t+1)$, l'algorithme recherche une matrice de transformation $\boldsymbol{M}$ à appliquer à chaque point P' de $\mathcal{P}(t+1)$ pour minimiser la distance globale entre ces correspondances. $\boldsymbol{M}$ se compose d'une translation $\boldsymbol{T}$ et d'une rotation $\boldsymbol{R}$. T est déterminée en calculant la différence entre les barycentres $\boldsymbol{\mu}$ et $\boldsymbol{\mu'}$ des positions spatiales des éléments de $\mathcal{P}(t)$ (respectivement $\mathcal{P}(t+1)$) contenus dans $\mathcal{C}$.

La matrice de rotation $\boldsymbol{R}$ est déterminée en calculant la rotation moyenne pour chaque couple de correspondances (P, P') permettant d'aligner leurs vecteurs normaux.

Algorithme 10 : *Alignement de correspondances en utilisant les normales.*

$\boldsymbol{M} \leftarrow \boldsymbol{I}_4(\mathbb{R})$
$\boldsymbol{\mu} \leftarrow [0,0,0]^\top$
$\boldsymbol{\mu'} \leftarrow [0,0,0]^\top$
$\boldsymbol{R} \leftarrow \boldsymbol{I}_3(\mathbb{R})$
for all $(P, P') \in \mathcal{C}$ **do**
  $\boldsymbol{\mu} = \boldsymbol{\mu} + \boldsymbol{p}$
  $\boldsymbol{\mu'} = \boldsymbol{\mu'} + \boldsymbol{p'}$
  $\boldsymbol{n_0} \leftarrow \frac{1}{2}(\boldsymbol{n} + \boldsymbol{n'})$
  $\boldsymbol{n_1} \leftarrow \boldsymbol{n'}$
  $\boldsymbol{u} \leftarrow \boldsymbol{n_0} \wedge \boldsymbol{n_1}$
  $\boldsymbol{R} \leftarrow \boldsymbol{R} + from\_quat(\boldsymbol{u})$
end for
$\boldsymbol{R} \leftarrow \frac{\boldsymbol{R}}{Card(\mathcal{C})}$
$\boldsymbol{\mu} \leftarrow \frac{\boldsymbol{\mu}}{Card(\mathcal{C})}$
$\boldsymbol{\mu'} \leftarrow \frac{\boldsymbol{\mu'}}{Card(\mathcal{C})}$
$\boldsymbol{T} \leftarrow \boldsymbol{\mu'} - \boldsymbol{\mu}$
$\boldsymbol{M} \leftarrow \begin{bmatrix} \boldsymbol{R}, \boldsymbol{T} \\ \boldsymbol{0_{1,3}}(\mathbb{R}), 1 \end{bmatrix}$

6.3 Construction incrémentale des murs

Pouvoir détecter et mettre en correspondance en temps réel l'ensemble des plans contenus dans les nuages de points renvoyés par le capteur de profondeur de la tablette permet d'envisager la construction de manière incrémentale d'un modèle numérique d'un bâtiment.

En effet, on peut distinguer les murs des autres plans car ils occupent la hauteur maximale de la pièce capturée, et ont les surfaces les plus importantes.

Nous détaillons ici le processus d'identification des murs mis en place lors de ces travaux.

6.3.1 Mise à jour de l'enveloppe de chaque plan

En plus des histogrammes et de leurs identifiants, nous stockons une enveloppe pour chaque plan (il s'agit de l'ensemble des points le délimitant). La mise en correspondance de deux plans identiques permet de mettre à jour l'enveloppe géométrique du plan concerné. À partir de cette enveloppe géométrique, il est possible de retrouver un maillage 3D du plan en question [142].

Deux approches ont été envisagées pour modéliser l'enveloppe d'un plan, elles sont présentées sur la figure 6.4.

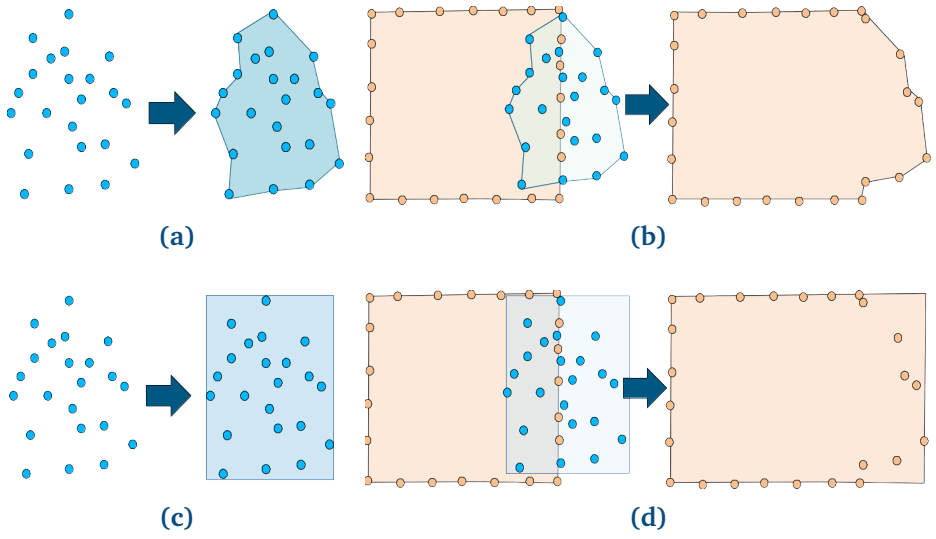


Figure 6.4 – *Mise à jour de l'enveloppe de chaque plan, deux approches ont été envisagées. La première approche consiste à calculer l'enveloppe concave de chaque plan lors de la détection (a), puis d'appliquer à nouveau cet algorithme avec l'enveloppe concave d'un plan précédemment détecté si besoin (b). L'autre option consiste à calculer les extensions verticales et horizontales des plans détectés (c), et de mettre à jour un plan précédemment détecté si besoin (d).*

La première approche consiste à calculer l'enveloppe concave de chaque plan détecté, à partir des nuages de points les constituant [93]. Ensuite, lors de l'appariement temporel, les enveloppes des plans mis en correspondance sont mises à jour. Cette approche permet d'obtenir des contours plus précis, et de modéliser des murs qui ne sont pas rectangulaires. Cette méthode présente cependant deux principales limitations. Tout d'abord, l'enveloppe concave de

chaque plan croît progressivement lors de la mise en correspondance avec de nouveaux plans. Il faut alors mettre en place un algorithme limitant la résolution spatiale des enveloppes concaves pour éviter une croissance trop rapide, et une saturation de la mémoire. Ensuite, l'algorithme de calcul de l'enveloppe concave, bien qu'il soit très performant est coûteux en temps de calcul.

Nous avons donc opté pour une deuxième solution, au moins dans un premier temps, où nous décrirons un mur avec les coordonnées de ses quatre coins. Soit Π un plan détecté à l'instant t. Π est caractérisé par l'équation $n_x.x+n_y.y+n_z.z+d = 0$, avec $\boldsymbol{n} = [n_x, n_y, n_z]^T$ un vecteur normal à Π, et $||\boldsymbol{n}|| = R = 1$.

On pose :

$$\boldsymbol{n} = \begin{bmatrix} R.sin(\phi).cos(\theta) \\ R.cos(\phi) \\ R.sin(\phi).sin(\theta) \end{bmatrix}$$

On détermine deux vecteurs $\boldsymbol{u}$ et $\boldsymbol{v}$, unitaires, orthogonaux entre eux, et vérifiant $\boldsymbol{u} \wedge \boldsymbol{v} = \boldsymbol{n}$.

En prenant $\theta \in [0, 2\pi[$, on a $\phi \in [0, \pi]$, on a donc $\phi = arccos(\frac{n_y}{R})$. On en déduit alors la valeur de $cos(\theta)$, et $sin(\theta)$, et on pose $\phi' = \frac{\pi}{2} + \phi$. On calcule alors :

$$\boldsymbol{u} = \begin{bmatrix} R.sin(\phi').cos(\theta) \\ R.cos(\phi') \\ R.sin(\phi').sin(\theta) \end{bmatrix}$$

On peut déjà vérifier que $||\boldsymbol{u}|| = R = 1$, et que $\boldsymbol{u} \cdot \boldsymbol{v} = 0$. On construit alors $\boldsymbol{v} = \boldsymbol{n} \wedge \boldsymbol{v}$.

Soit $\boldsymbol{O'} = \boldsymbol{0}_{3,1}(\mathbb{R}) - d.\boldsymbol{n}$. Alors le repère $\mathcal{R}'(\boldsymbol{O'}, \boldsymbol{u}, \boldsymbol{v}, \boldsymbol{n})$ forme un repère lié au plan Π, et dépendant uniquement de ses quatre paramètres n_x, n_y, n_z et d.

À la fin de l'étape de segmentation, nous calculons ce repère $\mathcal{R}'$ pour chacun des plans Π trouvés. Nous exprimons l'ensemble des points contenus dans Π dans le repère $\mathcal{R}'$, et nos calculons alors les extremums pour x, et y, pour tout pont $\boldsymbol{p}$ de Π dans $\mathcal{R}'$.

S'il s'avère que Π prolonge un autre plan Π' détecté précédemment, Π' est mis à jour en comparant les extremums de x et y qu'il contient avec ceux calculés pour Π.

Dans le cas où les plans sont alignés sur un repère $\mathcal{R}_0$, ces calculs peuvent être simplifiés : il suffit de prendre les extremums pour x et y pour la boite englobante d'un plan orthogonal à z, x et z si ce plan est orthogonal à y et y et z s'il est orthogonal à x.

6.3.2 Classification finale

Nous avons conservé le même raisonnement qu'en 3.4. Les plans verticaux possédant la plus grande extension verticale sont identifiés comme des murs. Le

résultat final est visible sur la figure 6.5.

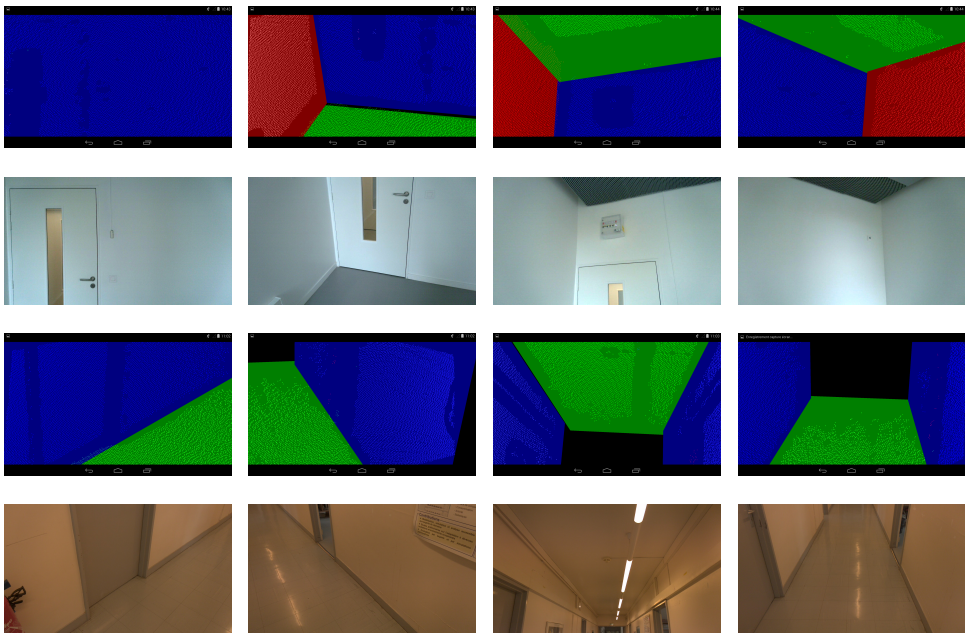


Figure 6.5 – *Exemples de détection des murs.*

6.4 Finalisation de la maquette

Une fois la capture effectuée, nous sommes en mesure d'exporter un modèle contenant les informations nécessaires à la création d'un modèle 3D éditable. Dans le cas de nos travaux, nous avons pu mettre en place l'export des murs détectés lors de la capture. Nous sommes capables à partir de nos informations de remonter à la position et aux dimensions de chaque mur détecté dans le bâtiment.

6.4.1 Détection des ouvrants à l'aide des images RGB

La détection des ouvrants n'a pas été traitée lors de ces travaux, et fera l'objet des futures études. Nous envisageons une première approche basée sur l'utilisation d'un réseau de neurones convolutif (*Convolutional Neural Network* (CNN)) pour effectuer une segmentation de l'image RGB et isoler les ouvrants s'il y en a.

Les CNN sont un cas particulier des réseaux de neurones faisant appel à une ou plusieurs couches de convolution. Ces couches consistent à appliquer un ou plusieurs filtres de convolution sur l'image d'entrée, et peuvent se succéder. La taille de l'image d'entrée est réduite d'une couche à l'autre, et une dernière couche de neurones effectue la classification finale. Pour plus de détails, le lecteur pourra se référer à [74, 75]. L'utilisation de couches de convolution permet de réduire considérablement la taille du réseau de neurones, et est particulièrement adaptée au traitement d'images.

Nous envisageons d'adapter un algorithme de segmentation générique développé par Long et al. [81], qui permet de segmenter une image selon la classe d'appartenance des différents éléments qui la constituent. Dans notre cas, il s'agit d'une segmentation selon deux classes : le pixel courant appartient à un ouvrant ou non.

Ces algorithme se prête très bien à la parallélisation, et les GPU des tablettes récentes nous permettent désormais de les utiliser pour faire du calcul. Nous envisageons donc une implémentation utilisant au maximum le GPU pour la détection des ouvrants. Cela offrira de meilleures performances de calcul tout en préservant le CPU pour l'identification des murs.

6.4.2 Export de la maquette finale

À partir des données obtenues après la capture nous pouvons exporter un modèle regroupant les informations nécessaires à la création d'un modèle 3D éditable. Nous avons choisi de faire un export au format .xml (voir figure 6.6).

```
<mur id="0" orientation="Z" poids="3927358.000000">
    <P0 X="-0.888551" Y="-1.373142" Z="-1.026195"/>
    <P1 X="2.600844" Y="-1.373142" Z="-1.026195"/>
    <P2 X="2.600844" Y="0.383837" Z="-1.026195"/>
    <P3 X="-0.888551" Y="0.383837" Z="-1.026195"/>
</mur>
<sol id="1" orientation="+Y" poids="938054.375000">
    <P0 X="-0.704738" Y="-1.383402" Z="-1.290271"/>
    <P2 X="2.563830" Y="-1.383402" Z="-1.290271"/>
    <p2 X="2.563830" Y="-1.383402" Z="0.000000"/>
    <p3 X="-0.704738" Y="-1.383402" Z="0.000000"/>
</sol>
```

Figure 6.6 – *Exemple de fichier d'export.*

Pour chaque mur, nous sauvegardons les positions spatiales de ses quatre coins, son orientation, et un identifiant unique. On pourra par la suite associer un ou plusieurs ouvrants aux murs grâce à l'algorithme de détection des ouvrants. Ces informations exportées permettent de calculer les différentes dimensions des murs, et leurs surfaces.

6.5 Évaluations

Nous présentons ici les résultats de l'évaluation de l'algorithme d'identification des murs. Une fois la capture effectuée, nous disposons d'un modèle 3D représentant un ensemble de murs et leurs dimensions géométriques. Nous pouvons alors vérifier que les dimensions calculées concordent avec les dimensions réelles.

6.5.1 Évaluation de la maquette générée

Nous avons évalué la fiabilité de la détection des murs en mesurant l'enveloppe externe de plusieurs pièces différentes. Nous avons utilisé la même tablette que lors des chapitres précédents (voir 1.4.3), et avons utilisé l'algorithme de segmentation décrit au chapitre 5, utilisant une approche de clustering. Nous avons choisi trois endroits différents, avec à chaque fois du mobilier placé le long de murs. Ces trois endroits sont présentés sur la figure 6.7.

Figure 6.7 – *Les trois environnements réels utilisés pour l'évaluation.*

Pour chacun de ces lieux, nous avons mesuré quand cela était possible la longueur max (L), la hauteur max (H), et la largeur max (l). Il n'y a pas de longueur l mesurée pour le couloir car il se terminait d'un côté par une porte vitrée, qui n'était pas détectée par le capteur de profondeur. Nous avons réalisé ces mesures en faisant une capture complète de ces environnements à l'aide de la tablette, afin de s'assurer d'avoir détecté tous les plans qu'ils contenaient. Nous avons ensuite mesuré les dimensions des murs identifiés pour en déduire les différentes mesures de la pièce. Nous avons répété les mesures 10 fois pour chacun des environnements. Les résultats sont reportés dans le tableau 6.1.

Mesure	Bureau			Couloir		Salle de réunion		
	L	H	l	L	H	L	H	l
Dim (.m)	5.80	2.70	3.30	2.70	1.64	6.80	2.70	5.60
Moy (.m)	5.82	2.73	3.35	2.70	1.62	6.76	2.89	5.64
Std dev. (.m)	0.08	0.13	0.04	0.04	0.04	0.12	0.25	0.21
Std dev. (%)	1	5	1	1	2	2	9	4
Erreur max (%)	3	11	4	3	5	4	25	11

Tableau 6.1 – *Comparaison entre les dimensions réelles et les dimensions déduites de la détection des murs.*

La première ligne de ce tableau montre les dimensions réelles, et le deuxième les dimensions moyennes mesurées. Nous avons ensuite calculé l'erreur standard en mètres et en pourcentages pour ces mesures (troisième et quatrième ligne). L'erreur maximale par rapport à la mesure réelle est reportée sur la dernière ligne.

Nos tests montrent que l'erreur de mesure est en moyenne inférieure à 5%, à l'exception d'un cas pour lequel les lumières du plafond ont faussé les mesures du capteur de profondeur.

6.5.2 Discussion

Les résultats montrent que nous sommes capables d'obtenir de bonnes précisions au niveau des mesures dans le modèle final par rapport au modèle réel. L'écart moyen entre les mesures étant inférieur à 5% en moyenne. Cependant, certaines imprécisions sont à prendre en compte, elles sont notamment dues aux erreurs de mesure pouvant résulter de l'algorithme de détection des plans, qui a du mal à distinguer deux plans parallèles s'ils sont trop proches.

6.6 Conclusion

À partir de l'algorithme de détection des plans basé sur des méthodes de type *k-means* (voir chapitre 5), et l'utilisation d'un algorithme d'odométrie, nous sommes en mesure de reconstruire un modèle 3D éditable d'un environnement intérieur à la volée. Pour cela, nous utilisons l'algorithme d'odométrie pour convertir l'ensemble des données spatiales dans un même repère initialisé au début de la capture. Nous calculons ensuite des histogrammes pour les paramètres de plans pour chacun des clusters issus de l'algorithme de segmentation, et utilisons ces histogrammes pour associer à chaque plan un identifiant unique dépendant de ses paramètres. Ces identifiants permettent d'identifier deux plans identiques même s'ils ont été détectés à des instants différents.

Grâce à cette mise en correspondance, nous pouvons calculer et mettre à jour une enveloppe pour chaque plan détecté, et étiqueter un plan comme étant un mur, sol, ou plafond en fonction de son orientation, sa position et ses dimensions.

Ces données nous permettent d'exporter un modèle sous une forme condensée, avec uniquement des éléments structuraux étiquetés, et les informations permettant de retrouver leur géométrie.

Nos évaluations ont montré que nous pouvons estimer les dimensions d'une pièce avec une erreur moyenne de 5%, mais avec des imprécisions lorsque plusieurs plans parallèles sont trop proches entre eux.

Ces travaux montrent des résultats prometteurs pour la génération d'une maquette 3D en temps réel, à partir d'une tablette équipée d'un capteur de profondeur. Les prochains travaux porteront sur l'utilisation d'un algorithme de classification pour détecter les ouvrants sur les images couleur d'une caméra et

de les intégrer au modèle final exporté.

Chapitre 7

Conclusion et travaux futurs

7.1 Rappel des objectifs 130
7.2 Contributions 130
7.2.1 Reconstruction et segmentation de maillage 3D 131
7.2.2 Mise en place d'algorithmes de segmentation planaire 132
7.2.3 Construction d'un modèle 3D éditable à la volée 133
7.3 Limitations 133
7.4 Perspectives 134

7.1 Rappel des objectifs

L'objectif de ces travaux de thèse était de concevoir des algorithmes permettant de modéliser un environnement intérieur à l'aide d'une tablette équipée d'un capteur de profondeur. Ces algorithmes ont pour vocation à être intégrés dans une application de modélisation 3D destinée à des particuliers souhaitant faire des travaux de rénovation d'intérieur, simplifiant ainsi la tâche de réalisation du modèle. Le modèle 3D reconstruit à l'aide de la tablette doit donc pouvoir être édité, et cela implique d'identifier les différents constituants (structure, mobilier, etc...) qui le composent.

Nos travaux tirent parti de l'intégration récente de capteurs de profondeur sur les tablettes, et l'augmentation des capacités de calcul de ces dernières. Cela permet d'envisager de faire de la reconstruction 3D sur tablettes. Cependant, les capacités de calcul des tablettes restent largement inférieures à celles des ordinateurs de bureau, pour lesquels de nombreuses travaux sur la reconstruction 3D existent déjà. Une grande partie des travaux de thèse a consisté à repenser et adapter ces algorithmes pour pouvoir les exécuter sur une tablette. De plus, il est très difficile d'identifier l'intégralité des constituants d'un bâtiment, et il est dont peu envisageable de proposer un algorithme permettant l'export d'un modèle 3D éditable sans formuler d'hypothèses simplificatrices.

Dans le cas de travaux de rénovation, nous pouvons nous limiter à l'export d'un modèle 3D décrivant la structure du bâtiment (sol, plafond, murs et ouvrants). Cela nous permet de simplifier la classification des constituants du modèle 3D et ces informations sont suffisantes dans le cas de la rénovation d'intérieur.

Les algorithmes à mettre en place doivent dépendre le moins possible de dispositifs externes, et nous avons donc proposé des solutions n'utilisant que les capacités de calcul de la tablette. De même, nous avons proposé des algorithmes entièrement automatiques ne nécessitant l'intervention d'un utilisateur que pour effectuer une capture 3D de l'environnement à modéliser.

7.2 Contributions

Nous avons alors envisagé deux approches pour effectuer cette modélisation (voir la figure 7.1).

La première approche, assez classique, repose sur la construction d'un maillage 3D en temps réel, puis la segmentation et la classification des données qui le composent. La deuxième approche, qui est plus originale, consiste à effectuer cette modélisation à la volée, sans stocker de maillage 3D intermédiaire. Cette approche a donné lieu à la mise en place de deux algorithmes de segmentation planaire en temps réel, et à un algorithme de reconstruction des

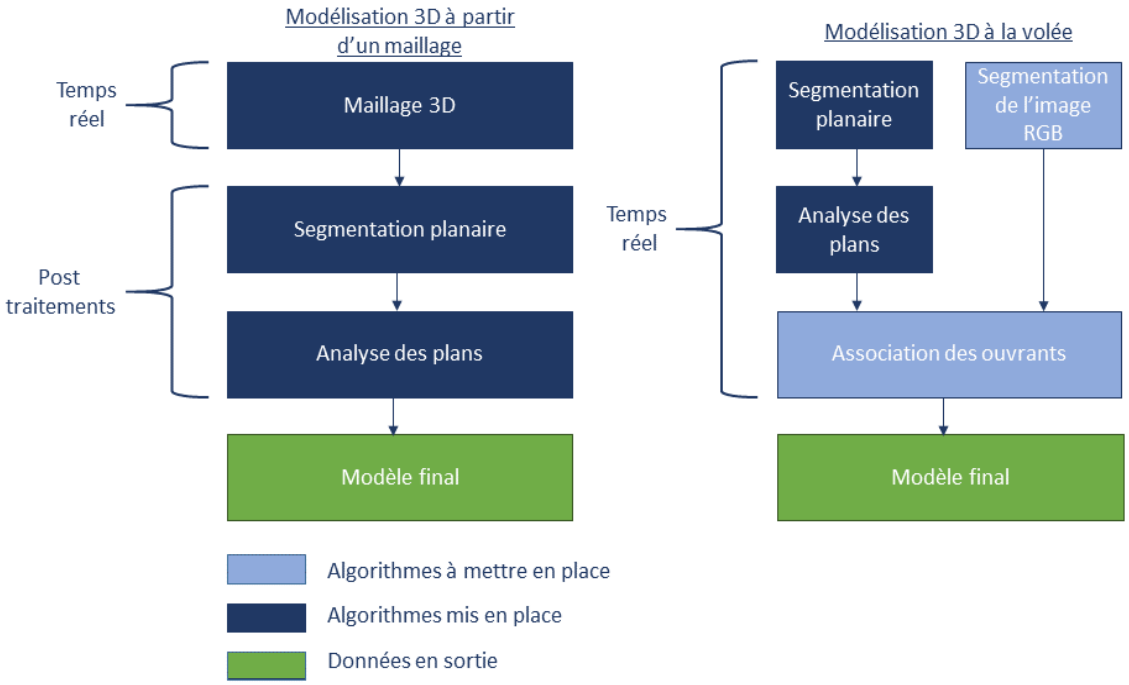


Figure 7.1 – *Résumé des deux approches de reconstruction 3D abordées et des briques algorithmiques mises en place.*

murs à la volée.

Chacune des solutions proposées repose sur des hypothèses présentées dans le tableau 1.1. Une partie de ces hypothèses correspond aux hypothèses formulées au §1.4.1, et d'autres sont spécifiques aux algorithmes utilisés.

7.2.1 Reconstruction et segmentation de maillage 3D

Dans le cadre de la première approche de reconstruction 3D développée, nous avons mis en place un algorithme permettant de reconstruire un maillage 3D en temps réel sur tablette. Une fois ce maillage 3D reconstruit, un second algorithme effectue une segmentation planaire dessus avant de classifier les plans détectés comme étant des éléments structuraux ou du bruit. Pour chacun des plans étiquetés comme étant des murs, un dernier algorithme va détecter les ouvertures rectangulaires et déterminer s'il s'agit d'ouvrants ou non.

Nous avons évalué l'utilisation de ces algorithmes en termes d'usages, et avons montré qu'utiliser la tablette pour effectuer la modélisation même d'un environnement intérieur simple était plus avantageux pour des utilisateurs que par une modélisation manuelle. Nous avons également montré que les algorithmes mis en place sont capables de restituer les différentes dimensions du bâtiment reconstruit avec une précision dans les mesures allant jusqu'à 90%. Cependant, l'algorithme de détection des ouvrants, n'est pas assez fiable et produit de nombreux faux positifs.

Hypothèse	Maillage 3D	Temps réel + VCCS	Temps réel + *k-means*
Murs plans	X	X	X
Murs sur une grille régulière	X	X	X
Reconstruction pièce par pièce	X	X	X
Utilisation d'un algorithme d'odométrie	X	X	X
Nuages de points ordonnés	-	X	X[1]
Début de la capture face à un mur	-	-	X
Ouvrants rectangulaires	X	-	-
Capture effectuée avec les portes ouvertes	X	-	-
Les portes font au moins 2 m de hauteur	X	-	-
Les fenêtres sont plus proches du plafond que du sol	X	-	

[1]Cela n'est pas obligatoire si l'on utilise d'autres algorithmes pour l'estimation des normales et la correction des dérives.

Tableau 7.1 – *Liste des hypothèses pour chacune des approches mises en place. Les quatre premières sont les hypothèses générales que nous avions formulées dès le début (voir §1.4.1).*

7.2.2 Mise en place d'algorithmes de segmentation planaire

Dans le cadre de la deuxième approche de modélisation envisagée, nous avons commencé par mettre en place deux algorithmes de segmentation planaire en temps réel. Le premier, basé sur un algorithme de croissance de régions, permet de détecter l'ensemble des plans contenus dans un nuage de points dans un ordre de temps de 100 ms. Il nécessite que les données en entrée soient ordonnées selon un espace 2D. Cet algorithme est par contre peu robuste à des erreurs d'estimation des vecteurs normaux, ce qui conduit à des détections de plans parasites. Une meilleure estimation des normales pourrait améliorer la qualité de cet algorithme.

Nous avons donc proposé un deuxième algorithme de segmentation planaire reposant sur des techniques de clustering génériques. Cela permet de considérer les données dans leur ensemble et ainsi, d'avoir une détection des plans plus robustes aux erreurs d'estimation des normales. L'algorithme mis en place utilise une méthode de type *k-means* pour segmenter les points selon la valeur de leurs vecteurs normaux, avant de séparer les plans parallèles. Pour fonctionner, cet algorithme nécessite que la capture s'effectue face à un mur. Il n'est en revanche pas nécessaire que les nuages de points en entrée soient ordonnés, à condition

d'utiliser un autre algorithme de détection des normales. Cet algorithme produit beaucoup moins de plans parasites et est entre 5 et 10 fois plus rapide à s'exécuter que l'algorithme par croissance de régions.

Nous avons évalué ces deux algorithmes pour mesurer la distance entre deux plans. Nous avons trouvé de très bons résultats au niveau de la précision des mesures pour chacun d'entre eux. Ils sont capables d'estimer une distance avec une erreur inférieur à 5 cm, mais avec une meilleure robustesse pour l'algorithme utilisant des techniques de clustering.

7.2.3 Construction d'un modèle 3D éditable à la volée

Nous avons ensuite mis en place un algorithme permettant de construire un modèle 3D contenant les murs, le sol et le plafond d'une pièce vide d'un bâtiment. Il s'appuie sur l'un des deux algorithmes de segmentation planaire décrit précédemment. Par rapport à nos évaluations, nous avons utilisé l'algorithme basé sur des techniques de clustering.

L'algorithme de reconstruction utilise le système d'odométrie de la tablette pour estimer se position repère global. Cette position permet d'exprimer les coordonnées de tous les objets dans un même repère. Lorsqu'un nouveau nuage de points est disponible, il est segmenté, et un histogramme est calculé pour chacun des paramètres géométriques des plans identifiés. Ces histogrammes permettent d'associer à chaque plan un identifiant unique qui permet de les ranger dans une table de hachage et de les mettre en correspondance avec les autres plans détectés précédemment. Nous sommes alors capable d'identifier que deux plans détectés sur deux prises de vues successives sont en fait identiques, et mettre à jour les caractéristiques géométriques du plan en question. Les plans dont les surfaces sont les plus importantes sont étiquetés comme étant des murs. Nos évaluations montrent que l'on est capable de mesurer les dimensions d'une pièce avec une erreur de mesure moyenne inférieur à 5% à l'aide de cet algorithme.

7.3 Limitations

Nous avons effectué tous nos développements sur une tablette du projet Tango de Google, qui intègre un capteur de profondeur et un algorithme d'odométrie visuelle.

Il n'est donc pas nécessaire de calibrer un capteur de profondeur à la caméra couleur, ni d'implémenter d'algorithme d'odométrie. De plus, une API permet d'accéder à ces différentes données très facilement. Cela nous a permis de travailler directement sur les algorithmes en eux-mêmes.

Cependant, nous avons dû faire face à un certain nombre de limitations matérielles, qui ont eu un impact sur la qualité des algorithmes mis en place. Tout d'abord, nous n'avions pas d'accès direct à la carte de profondeur renvoyée par le capteur. L'accès à la carte de profondeur aurait pu nous permettre de mettre en place un algorithme d'estimation des normales plus précis.

De plus, nous avons été confrontés à des imprécisions de données venant du capteur de profondeur. Nous avions évoqué le *motion blur*, qui survient lors de déplacements, mais la qualité de la carte de profondeur est également bruitée lorsque le capteur passe devant des zones très lumineuses, comme des lumières, des fenêtres ou des zones éclairées par le soleil.

7.4 Perspectives

Les travaux actuels portent sur la détection des ouvrants dans une image RGB à l'aide d'un réseau de neurones convolutif, pour compléter notre algorithme de construction de modèle à la volée. Les modèles de tablettes et de smartphones actuels proposent des processeurs graphiques plus performants, et il est envisageable d'implémenter ce réseau de neurones à l'aide du GPU. L'utilisation d'un *pipeline* GPU permet d'accélérer l'exécution de l'algorithme de détection des ouvrants, tout en garantissant qu'il n'y aura pas de perte de performances pour l'algorithme de détection des murs implémenté uniquement sur le CPU de la tablette.

Les travaux futurs porteront sur l'utilisation d'autres capteurs de profondeur pour réaliser la reconstruction 3D. En effet, d'une part, le projet Tango de Google n'existe plus, et d'autre part, l'utilisation d'autres capteurs de profondeur nous permettra de tester nos algorithmes sur d'autres types de données et d'utiliser des cartes de profondeur de meilleur qualité. Nous pourrons alors améliorer la robustesse de notre algorithme de reconstruction de modèle 3D à la volée et l'intégrer à l'application Plan 3D Énergie.

À plus long terme, nous pouvons envisager de réaliser une modélisation 3D plus complexe qu'un modèle 3D contenant des murs rectangulaires et des ouvrants. Pour cela, il faudra mettre en place un autre algorithme de calcul d'enveloppe concave plus générique, et envisager que deux plans identiques peuvent aussi désigner deux murs différents. Il serait également possible de classifier les points non étiquetés comme étant des murs et de tenter de reconstruire un modèle 3D plus riche.

Annexe A

Publications pendant la thèse

A.1 Communications nationales 136
A.2 Conférences internationales 136
A.3 Journaux internationaux 136

A.1 Communications nationales

- **Arnaud, A.**, Christophe, J., Gouiffès, M., & Ammi, M. Modélisation d'Environnements Intérieurs par Reconstruction 3D sur Tablette pour la Rénovation. Journée Nationale de l'Internet des Objets, 29 Novembre 2016.
- **Arnaud, A.**, Corrège, J. B., Clavel, C., Gouiffes, M., & Ammi, M. (2015, October). Exploration d'environnement virtuel sur tablette : Comparaison entre des modalités tactiles et tangibles. In 27ème conférence francophone sur l'Interaction Homme-Machine. (p. w18). ACM.

A.2 Conférences internationales

- **Arnaud, A.**, Correge, J. B., Clavel, C., Gouiffès, M., & Ammi, M. (2015, October). Exploration of a virtual environment : a comparison between tactile and tangible modalities. In Proceedings of the 27th Conference on l'Interaction Homme-Machine (p. 39). ACM.
- **Arnaud, A.**, Christophe, J., Gouiffès, M., & Ammi, M. (2016, November). 3D reconstruction of indoor building environments with new generation of tablets. In Proceedings of the 22nd ACM Conference on Virtual Reality Software and Technology (pp. 187-190). ACM.

A.3 Journaux internationaux

- **Arnaud, A.**, Gouiffès, M., & Ammi, M. (2018). On the Fly Plane Detection and Time Consistency for Indoor Building Wall Recognition Using a Tablet Equipped With a Depth Sensor. IEEE Access, 6, 17643-17652.

Annexe B

Algorithmes utilisés

B.1 Algorithme ICP 138
B.2 Algorithme *Marching cubes* 139

B.1 Algorithme ICP

L'algorithme ICP consiste à aligner deux nuages de points 3D $\mathcal{P}$ et $\mathcal{P}'$, en supposant qu'une partie de leurs points est commune. Cela permet de corriger d'éventuelles erreurs de positionnement. Il a été décrit pour la première fois par Besl et McKay [8], et consiste à identifier un ensemble $\mathcal{C}$ de couples points correspondants sur $\mathcal{P}$ et $\mathcal{P}'$ et à estimer une transformation $\boldsymbol{M}$ qui minimise la distance globale $\delta(\mathcal{C})$ entre ces points.

Deux points $\boldsymbol{p} \in \mathcal{P}$ et $\boldsymbol{p'} \in \mathcal{P}'$ sont considérés comme correspondants si leur distance est inférieure à un certain seuil. Cette recherche de correspondance est propre à chaque algorithme, car elle est critique pour les performances. En effet, il faut pouvoir déterminer les points voisins de $\boldsymbol{p}$ dans $\mathcal{P}'$ pour éventuellement trouver un point $\boldsymbol{p'}$ correspondant. La distance est le premier critère pour estimer si deux poins sont correspondants, mais il est possible d'affiner cette estimation à l'aide d'autres paramètres, comme la couleur ou la courbure locale de la surface en un point.

La recherche de $\boldsymbol{M}$ est en fait itérative, car en général il n'y a pas de correspondance exacte entre deux points de $\mathcal{P}$ et $\mathcal{P}'$. Tout simplement parce que $\mathcal{P}$ et $\mathcal{P}'$ sont issue de eux captures différentes et ont une résolution spatiale limitée, et parce que le plus proche voisin d'un point $\boldsymbol{p} \in \mathcal{P}$ dans $\mathcal{P}'$ n'est pas forcément son homologue. L'algorithme est donc itératif, et affine l'estimation de $\boldsymbol{M}$ à chaque itération.

Le fonctionnement de l'algorithme ICP est détaillé sur l'algorithme 11.

Algorithme 11 : *Détail de l'algorithme ICP.*

$\boldsymbol{M} \leftarrow \boldsymbol{I_4}(\mathbb{R})$
$\mathcal{C} \leftarrow trouver\_correspondances(\mathcal{P}, \mathcal{P}')$
$\Delta \leftarrow \delta(\mathcal{C})$
$\Delta_0 \leftarrow \Delta$
while $\Delta > \epsilon_\Delta$ **do**
    $\boldsymbol{\Delta_M} \leftarrow aligner(\mathcal{P}, \mathcal{P}')$
    $\mathcal{P} \leftarrow transformer(\mathcal{P}, \boldsymbol{\Delta_M})$
    $\mathcal{C} \leftarrow trouver\_correspondances(\mathcal{P}, \mathcal{P}')$
    $\Delta \leftarrow \delta(\mathcal{C})$
    $\boldsymbol{M} \leftarrow \boldsymbol{\Delta_M} \times \boldsymbol{M}$
    if $|\Delta - \Delta_0| < \epsilon_{diff}$ **then**
        retourner $\boldsymbol{M}$
    end if
    $\Delta_0 \leftarrow \Delta$
end while
retourner $\boldsymbol{M}$

L'algorithme estime d'abord un ensemble $\mathcal{C}$ de correspondances entre $\mathcal{P}$ et $\mathcal{P}'$, initialise $\boldsymbol{M}$ avec la matrice identité. Ensuite, il calcule la distance globale Δ entre ces correspondances, et sauvegarde cette valeur dans Δ_0. Si Δ est déjà inférieur à un seuil ϵ_Δ, alors les deux nuages de points sont déjà alignés.

Dans le cas contraire, l'algorithme calcule une transformation $\boldsymbol{\Delta_M}$ qui minimise la distance globale entre les couples de points de $\mathcal{C}$. Les points du nuage de points source, $\mathcal{P}$ sont transformés à l'aide de $\boldsymbol{\Delta_M}$, et $\mathcal{C}$ et Δ sont recalculés avec les nouvelles valeurs de $\mathcal{P}$.

$\boldsymbol{M}$ est multipliée par $\boldsymbol{\Delta_M}$, la convergence de l'algorithme est ensuite testée. Si la différence entre Δ et Δ_0 est inférieure à un certain seuil ϵ_{diff}, cela signifie que l'algorithme a convergé, il est donc inutile de continuer les itérations. Sinon, l'algorithme reboucle, tant que Δ est supérieur à ϵ_Δ, et tant que le nombre d'itérations maximal n'a pas été atteint.

L'algorithme ICP est un algorithme d'optimisation qui converge vers le premier minimum local. Pour être efficace, il faut que les nuages de points $\mathcal{P}$ et $\mathcal{P}'$ soient déjà presque alignés.

B.2 Algorithme *Marching cubes*

L'algorithme *Marching cubes* a été mis en place par Lorensen et al. [82]. Il permet de générer un maillage 3D à partir d'un champs de distances signées. Pour rappel, soit un point $\boldsymbol{p}$ de l'espace, et $\mathcal{S}$ une surface, on définit la distance signée $\phi(\boldsymbol{p}, \mathcal{S})$ de $\boldsymbol{p}$ à $\mathcal{S}$ par :

$$\phi(\boldsymbol{p}, \mathcal{S}) = \begin{cases} d(\boldsymbol{p}, \mathcal{S}) & \text{si p} \in \mathcal{V} \\ -d(\boldsymbol{p}, \mathcal{S}) & \text{sinon} \end{cases}$$

où $d(.)$ représente la distance d'un point à une surface. Dans le cas où il n'y a qu'une seule surface $\mathcal{S}$, on peut noter la distance signée à cette surface $\phi_\mathcal{S}(\boldsymbol{p})$. Si $\phi_\mathcal{S}$ entre deux points $\boldsymbol{p}$ et $\boldsymbol{p'}$ alors un point de $\mathcal{S}$ se situe sur le segment $[\boldsymbol{p}, \boldsymbol{p'}]$.

Dans le cadre d'une représentation volumétrique d'une surface, l'espace $\mathcal{V}$ est divisé en voxels d'une résolution spatiale ρ. Pour chaque voxel $V \in \mathcal{V}$, on calcule la distance signée de son centre $\boldsymbol{p_V}$ à $\mathcal{S}$, $\phi_\mathcal{S}(\boldsymbol{p_V})$.

L'algorithme *Marching cubes* consiste à regrouper les voxels de $\mathcal{V}$ par huit, de manière à ce que les huit centres forment un cube. Chaque sommet du cube est indexé de 0 à 7. En fonction des changements du signe de $\phi_\mathcal{S}$, il est possible de savoir si un point de $\mathcal{S}$ traverse chaque arête. Il y a au total $2^8 = 256$ configurations possibles pour les signes de $\phi_\mathcal{S}$ pour un cube. On peut en fait définir à l'avance les 256 configurations de référence pour chaque cube, que l'on peut réduire à 15 si l'on considère que certaines de ces configurations sont les mêmes à une rotation près. Ces configurations possibles sont présentées sur la figure B.1.

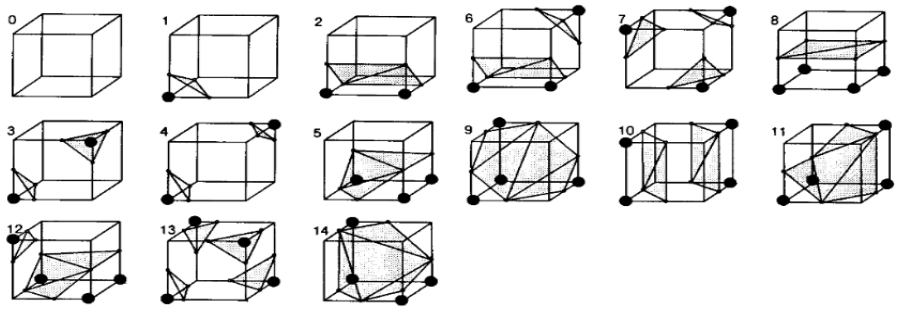


Figure B.1 – *Les configurations de référence pour l'algorithme Marching Cubes[82].*

L'algorithme 12 détaille le fonctionnement de *Marching Cubes*. Soit $\mathcal{A}$ l'ensemble des cubes formés précédemment. Chaque cube $\mathcal{A}_i$ de $\mathcal{A}$ construit à partir de $V_i \in \mathcal{V}$ est défini par :

$$\mathcal{A}_i = \{(x_i + j\rho, y_i + k\rho, z_i + l\rho)/(j,j,l) \in [\![0,1]\!]^3\} \tag{B.1}$$

où $\boldsymbol{p_i} = [x_i, y_i, z_i]^\top$ est le centre de V_i.

Chaque point de $\mathcal{A}_i$ est indexé de 0 à 7. On calcule un identifiant n codé sur 8 bits, pour lequel le bit à la position p est mis à 1 si le sommet de $\mathcal{A}_i$ associé a une distance signée à $\mathcal{S}$ négative, et 0 sinon. n est alors compris entre 0 et 255, et permet de charger la liste de triangles $\mathcal{T}_n$ associée à cette configuration dans $\mathcal{T}_{A_i}$.

Il suffit alors de translater l'ensemble des points $\boldsymbol{p}$ de $\mathcal{T}_{A_i}$ de $\boldsymbol{p_i}$ pour obtenir le maillage associé à $\mathcal{A}_i$.

Algorithme 12 : *Détail de l'algorithme Marching cubes.*

$\mathcal{T} \leftarrow \emptyset$
for all $\mathcal{A}_i \in \mathcal{A}$ **do**
  $n \leftarrow id(A_i)$
  $\mathcal{T}_{A_i} \leftarrow \mathcal{T}_n$
  for all $\boldsymbol{p} \in \mathcal{T}_{A_i}$ **do**
    $\boldsymbol{p} \leftarrow \boldsymbol{p} + \boldsymbol{p_i}$
  end for
  $\mathcal{T} \leftarrow \mathcal{T} + \mathcal{T}_{A_i}$
end for

Annexe C

Utilisation d'OpenMP pour paralléliser un algorithme

C.1 Blocs parallèles 142
C.2 Répartition du travail et synchronisation 143

La bibliothèque OpenMP [23] permet de paralléliser des portions de programme. OpenMP est intégrée par les compilateurs et est utilisable à travers des directives qui permettent gérer la répartition d'un travail sur plusieurs *threads*, et la synchronisation éventuelle entre plusieurs tâches. Nous présentons ici quelques concepts de base de la bibliothèque, des informations plus précises sont disponibles dans la documentation officielle [1].

C.1 Blocs parallèles

L'utilisation d'OpenMp repose sur la définition de blocs d'instruction qui seront exécutés en parallèle par chacun des threads du groupe de travail. On peut déclarer des variables à l'intérieur d'un bloc. Dans ce cas, chaque *thread* exécutant le bloc possède ses propres copies des variables déclarées.

Déclaration d'un bloc parallèle :

```
#pragma omp parallel num_threads(4)
{
        size_t i, j; // Variables propres à chaque bloc
        do_something();
} // End of parallel region
```

Ici, quatre *threads* sont créés, avec chacun leur propre copie de i et j, et chacun réalisera un appel à *do_something*. Cette directive sert de base à la parallélisation de code avec OpenMP. On peut par exemple paralléliser des boucles :

```
#pragma omp parallel num_threads(4)
{
        size_t i;
#pragma omp for
        for(i = 0; i < N; i++)
        {
                // Faire quelque chose ...
        }
} // End of parallel region
```

Les itérations de la boucle sont réparties sur quatre threads. Dans le cas où le bloc d'instructions ne consiste qu'à faire une boucle, et qu'il n'y a pas besoin de variables locale à un bloc, l'écriture précédente peut se synthétiser par :

1. https ://www.openmp.org/

```
size_t i;
#pragma omp parallel for num_threads(4)
for(i = 0; i < N; i++)
{
        // Faire quelque chose ...
}
```

De manière plus générale, on peut créer des tâches qui sont chacune exécutées par un *thread* différent :

```
#pragma omp parallel num_threads(2)
{
        size_t x;        // Une variable locale à chaque bloc
#pragma omp single
#pragma omp task
        {
                task1(); // Exécuté par l'un des deux threads
        }
#pragma omp single
#pragma omp task
        {
                task2(); // Exécuté par le deuxième thread
        }
} // End of parallel region
```

Dans cet exemple, la directive *single* indique que chacune des tâches doit être effectuée que par un seul *thread* du groupe de travail.

C.2 Répartition du travail et synchronisation

Lors de la parallélisation d'une boucle, il est possible de spécifier la manière dont le travail est réparti parmi les *threads*. Cette répartition peut être statique, et dans ce cas le nombre d'itérations est réparti équitablement parmi les différents *threads* du groupe de travail. Il s'agit de la répartition par défaut, elle est adaptée dans le cas où chaque itération se réalise dans un même intervalle de temps. Dans le cas contraire, il faut choisir une répartition du travail dynamique, où chaque *thread* se voit donner une nouvelle itération dès qu'il a fini la précédente. Les *threads* les plus rapides peuvent ainsi traiter plus de données. La spécification de la répartition du travail se fait de cette manière :

```
#pragma omp parallel
{
        size_t i;
#pragma omp parallel for schedule(static)
        for(i = 0; i < N; i++)
        {
                // Faire quelque chose ...
        }
} // End of parallel region
```

Il est également possible de synchroniser plusieurs threads avec les directives *barrier* ou *taskwait*.

```
#pragma omp parallel num_threads(2)
{
#pragma omp single
#pragma omp task
        task1();                // Exécuté par l'un des deux threads

#pragma omp single
#pragma omp task
        task2();                // Exécuté par le deuxième thread

#pragma omp taskwait    // Attente que task1()
                        // et task2() aient fini
} // End of parallel region
```

Annexe D

Jeu d'instructions ARM NEON

D.1 Description 146
D.2 Exemple : calcul de l'intensité d'une image RGB 147

D.1 Description

Les architectures ARM modernes disposent de l'extension d'architecture ARM NEON qui définit un certain nombre de registres pouvant contenir des vecteurs de données, et un jeu d'instruction permettant de manipuler ces registres. Cela permet d'exécuter une même instruction à un registre de n données.

Dans le cas de l'architecture arm7, cette extension définit 32 registres de 64 bits ($d_0, d_1, ..., d_{31}$), ou 16 registres de 128 bits ($q_0, q_1, ..., q_7$). Ces registres peuvent contenir des vecteurs de données de $8, 16$ ou 32 bits, dont la taille dépend de la taille maximale du registre. Un registre d_i pourra contenir 8 données de 8 bits, 4 données de 16 bits ou 2 données de 32 bits, et deux fois plus pour les registres q_i.

En programmation C, on peut définir des types associés à ces registres. Ils peuvent être associés à n'importe que type de base (entier signé ou non signé, flottant). On peut déclarer des vecteurs de q éléments de p octets. La seule limite étant que p doit valoir $8, 16$, ou 32, et que le produit $p \times q$ doit valoir 64 ou 128 en fonction de si l'on souhaite travailler avec un des vecteurs 64 ou 128 bits. Il est possible de déclarer des tableaux de vecteurs, allant de deux à quatre éléments.

Exemple de déclarations :

```
uint8x8_t v0;         // 8 entiers non signés de 8 bits
uint16x8_t v1;        // 8 entiers non signés de 16 bits
int32x4_t v2;         // 4 entiers signés de 32 bits
float32x4x3_t v3;     // 3 X 4 flottants de 32 bits
```

La syntaxe générale pour les déclarations est de la forme : <*type*>qxp_t <*identifiant*>.

Le jeu d'instruction ARM NEON permet de charger ou de ranger les données contenues dans ces vecteurs, et fournit les opérations arithmétiques et logiques de base (addition, multiplication, opérateurs logiques, etc...). Il est également possible de faire des opérations conditionnelles.

Exemples d'instructions :

```
float32x4_t v0, v1, v2;
uint16x8_t u0;
v0 = vaddq_f32(v1, v2);       // v0 <- v1 + v2
v0 = vmulq_f32(v1, v2);       // v0 <- v1 + v2
v0 = vmlaq_f32(v0, v1, v2);   // v0 <- v0 + v1 * v2
```

```
u0 = vshr_u16(u0, 8);          // u0 <- u0 >> 8
```

La syntaxe des instructions est de la forme : v<*nom*>_<*type*><*taille*>([*op*]). <*nom*> correspond au nom de l'instruction, <*type*> peut être u, s, ou f en fonction de si l'on considère qu'il s'agit d'opérations sur des vecteurs d'entiers non signés, signés, ou de flottants, et <*taille*> doit être 8, 16, ou 32. Une description plus détaillée du jeu d'instructions NEON est disponible sur la documentation officielle ARM [1].

D.2 Exemple : calcul de l'intensité d'une image RGB

Prenons l'exemple d'une fonction qui calcule l'intensité d'une image RGB *input_image* en entrée et écrit le résultat dans *output_image*. Cette fonction peut s'écrire de la manière suivante :

```
void img_intensity
        (
                uint8_t *__restrict input_img,
                uint8_t *__restrict output_img
        )
{
        size_t i;
        size_t index = 0;
        uint16_t acc;
        for(i = 0; i < IMAGE_RESOLUTION; i++)
        {
                acc = (uint16_t) input_img[index++];
                acc += (uint16_t) input_img[index++];
                acc += (uint16_t) input_img[index++];
                output_img[i] = (uint8_t)((acc * 0x55) >> 8);
        }
}
```

Ce code additionne les trois composantes, r, g, b pour un pixel de *input_img* à l'indice i et divise le résultat par trois. Ce résultat est stocké dans *output_img* à l'indice i.

1. https ://developer.arm.com/technologies/neon

En utilisant les instructions ARM NEON, ce code devient :

```
void img_intensity_opt
        (
                uint8_t *__restrict input_img,
                uint8_t *__restrict output_img
        )
{
        size_t i;
        uint8x8x3_t v_input;
        uint8x8_t v_output;
        uint8x8_t v_one = vdup_n_u8(1);
        uint16x8_t v_acc;
        uint8_t input = input_img;
        uint8_t output = output_img;
        for(i = 0; i < (IMAGE_RESOLUTION >> 8); i ++)
        {
                v_input = vld3_u8(input);
                v_acc = vaddl_u8(v_input.val[0], v_input.val[1]);
                v_acc = vmlal_u8(v_output, v_input.val[2], v_one);
                v_acc = vmulq_n_u16(v_output, 0x55);
                v_output = vshrn_n_u16(v_acc,8);
                vst1q_u8(output, v_output);
                input += 3 * 8;
                output += 8;
        }
}
```

On voit dans cet exemple, que nous traitons simultanément 8 données à la fois. Les données RGB en entrée sont stockées dans trois vecteurs de registres 8×8 bits différents : *v_input.val[0]*, *v_input.val[1]* et *v_input.val[2]*, chargés simultanément avec l'instruction vld3q_u8. Les données de sorties sont écrites dans le vecteur de registres 8×8 bits *v_output*.

Nous effectuons ensuite le même traitement que lors de la version séquentielle. L'instruction *vaddl_u8* additionne deux vecteurs de registres 8×8 bits et renvoie le résultat dans un vecteur de registres 8×16 bits. L'instruction *vmulq_n_u16* multiplie l'ensemble des données contenues dans v_acc par une constante (ici, $0x55$). L'instruction *vshrn_n_u16* effectue un shift de n bits vers la droite sur un vecteur de registres de 8×16 bits et tronque le résultat dans un vecteur de 8×8 bits.

Annexe E

Étude expérimentale : navigation tangible dans un environnement virtuel 3D

E.1 Introduction . . . 150
E.2 Travaux connexes . . . 151
E.2.1 Interaction tangible . . . 151
E.2.2 Navigation dans un environnement virtuel 3D . . . 152
E.2.3 Utilisation de facteurs d'échelle . . . 153
E.2.4 Conclusion . . . 153
E.3 Expérimentation . . . 153
E.3.1 Description . . . 154
E.3.2 Matériel . . . 154
E.3.3 Mesures . . . 155
E.3.4 Procédure . . . 155
E.4 Résultats . . . 156
E.4.1 Évaluation subjective . . . 156
E.4.2 Évaluation des sketch maps . . . 157
E.4.3 Analyse des trajectoires . . . 158
E.5 Conclusion et travaux futurs . . . 159

E.1 Introduction

Cette étude expérimentale a été réalisée au tout début de la thèse, son objectif était de comparer l'usage de modalités d'interaction tactiles et l'usage des seuls mouvements d'une tablette pour naviguer dans un environnement virtuel (EV) 3D.

En effet la navigation 3D est une tâche requérant souvent de manipuler plus de deux degrés de liberté (DDL) simultanément. Bien qu'il soit possible avec des dispositifs *multi-touch* de manipuler plus de deux DDL en simultané [48, 134], ces configurations peuvent se révéler contraignantes en termes d'occlusion de l'affichage sur des petits affichages comme les tablettes.

Au contraire, la mise sur le marché de nouvelles tablettes intégrant des capteurs permettant de suivre leurs déplacements de manière précise permet d'envisager les mouvements physiques de la tablette pour interagir. Il est ainsi possible de manipuler simultanément six DDL à l'aide des seuls mouvements de la tablette.

Cependant, les interactions tactiles sont actuellement majoritairement utilisées pour interagir sur des smartphones ou des tablettes. Certain gestes (comme le zoom à l'aide de deux doigts par exemple) finissent pas être considérés comme standards par les utilisateurs [60].

Ishii et Ullmer [138] définissent une interface tangible comme une interface donnant une forme physique à une information numérique en utilisant des artefacts physiques comme représentations et contrôles des données. Sharlin et al. [119] définissent trois heuristiques qui caractérisent une interface tangible :

- **Mapping spatial :** congruence entre les actions effectuées sur le dispositif tangible et l'action virtuelle effectuée.
- **Unification des entrées et sorties :** de la même manière que dans le monde réel, il faut éviter de faire de trop grandes distinctions entre les entrées et les sorties, ces distinctions peuvent être évitées avec un dispositif tangible car l'état de l'objet est directement connu de l'utilisateur qui le tient en main.
- **Support des essais erreurs :** une interface tangible peut permettre de corriger efficacement des erreurs, sans devoir annuler une séquence de manipulations effectuées après cette erreur.

Ces trois heuristiques justifient l'utilisation des mouvements physiques d'une tablette ou d'un smartphone pour naviguer en 3D. En utilisant ce dispositif comme élément tangible d'interface, on regroupe les entrées et les sorties sur ce même dispositif. En utilisant les translations et les rotations du dispositif mobile pour naviguer en 3D, on garantit un mapping spatial entre les actions physiques de l'utilisateur et les actions dans l'EV. Enfin, le support d'actions par essai-erreur est

également bien supporté : quelle que soit la position de l'utilisateur dans un EV, l'utilisateur peut revenir facilement à une position précédente car il sait où se situe ce point dans l'espace.

E.2 Travaux connexes

E.2.1 Interaction tangible

Marzo et al. [89] ont comparé la combinaison de modalités d'interaction tactiles et tangibles et tactiles ou tangibles seules pour des tâches de manipulation 3D. Cette étude a montré que la combinaison des deux modalités permettait globalement de meilleures performances en terme de temps d'accomplissement de la tâche. L'utilisation des seuls mouvements de rotation de la tablette pour effectuer une rotation d'un objet s'est cependant avérée moins efficace qu'avec l'utilisation du tactile.

Guéniat el al. [43] ont utilisé les mouvements d'une tablette afin d'effectuer de la navigation 3D. L'orientation de la caméra était contrôlée par les mouvements de la tablette. Il a été montré qu'en utilisant les mouvements physiques de la tablette les utilisateurs parcouraient plus de distance dans l'environnement virtuel. De plus, cette technique était plus appréciée par les utilisateurs. Fitzmaurice et al. [34] ont montré que l'utilisation des mouvements

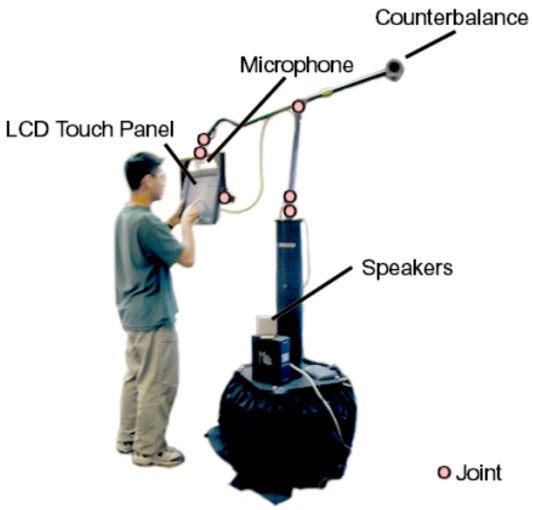


Figure E.1 – *Boom chameleon [137].*

d'un dispositif d'affichage portable pour naviguer en 3D donne à l'utilisateur le même perception de la profondeur qu'avec un large écran. De plus, ils ont aussi montré que l'utilisation de ces types d'interactions sont rapides à prendre en main [137], un exemple de dispositif développés lors de ces travaux est le *Boom chameleon* (voir fig.E.1).

E.2.2 Navigation dans un environnement virtuel 3D

Figure E.2 – *Étude de l'utilisation de la marche réelle pour naviguer dans un EV 3D [148].*

Zanbaka et al. [148] ont étudié l'utilisation d'un affichage de type *Head Mounted Display* (HMD) pour naviguer dans un EV. Quatre conditions différentes ont été analysées pour naviguer dans un EV avec un HMD incluant la marche réelle (voir fig.E.2). Les participants ont été invités à reproduire une vue de dessus de l'EV visité, puis on été invités à répondre à des questionnaires cognitif, de présence, et relatif au mal des simulateurs. Les trajectoires des participants ont été également enregistrées. L'étude n'a pas mis en évidence de différence significative entre les résultats sur l'évaluation des plans 2D réalisés, et sur les questionnaires cognitifs. Cependant, une différence a été observée sur les trajectoires suivies par les participants : lors de l'utilisation de la marche réelle, les participants évitaient plus les collisions qu'avec les autres conditions et concentraient leur trajectoire au centre de l'EV. Les participants se sont également sentis plus à l'aise avec la marche réelle.

Suma et al. [128] ont également étudié la navigation dans un EV avec un HMD. Ils ont testé trois conditions : la marche réelle, et deux conditions où les participants devaient pointer la direction de l'endroit où ils voulaient se déplacer (sens de la tête et pointage avec un stick). Les participants ont dû explorer un labyrinthe virtuel dans lequel des objets ont été placés et ont été invités à énumérer autant d'objets qu'ils pouvaient se rappeler. Ils ont également été soumis à une tâche de reconnaissance d'objets et ont été invités à dessiner une carte 2D de l'EV. Leur trajectoire a également été enregistrée. Les résultats ont montré que lors de la marche réelle, les participants avaient tendance à être plus performants dans les tâches qu'avec les deux autres conditions. La marche réelle a permis aux participants d'explorer plus rapidement et avec moins de collisions.

E.2.3 Utilisation de facteurs d'échelle

Dans la plupart des cas la géométrie de l'EV diffère de celle de l'environnement physique dans lequel évolue l'utilisateur. En particulier quand l'est très grand, la navigation peut se révéler fastidieuse à un facteur d'échelle de translation fixe. Plusieurs études ont exploré l'utilisation de facteurs d'échelle pour naviguer dans un grand EV [129]. Il ressort de ces études que sous certaines valeurs, les facteurs d'échelle sont imperceptibles.

E.2.4 Conclusion

L'utilisation des mouvements réels semble être une approche prometteuse pour naviguer dans un EV sur systèmes mobiles. Alors que plusieurs travaux ont étudié l'utilisation de dispositifs de type HMD pour naviguer dans un EV, peu d'études se sont concentrées sur la navigation 3D en utilisant les mouvements d'un smartphone ou d'une tablette. Et ces travaux se sont essentiellement concentrés sur l'usage de la rotation du dispositif pour orienter le point de vue de la caméra.

E.3 Expérimentation

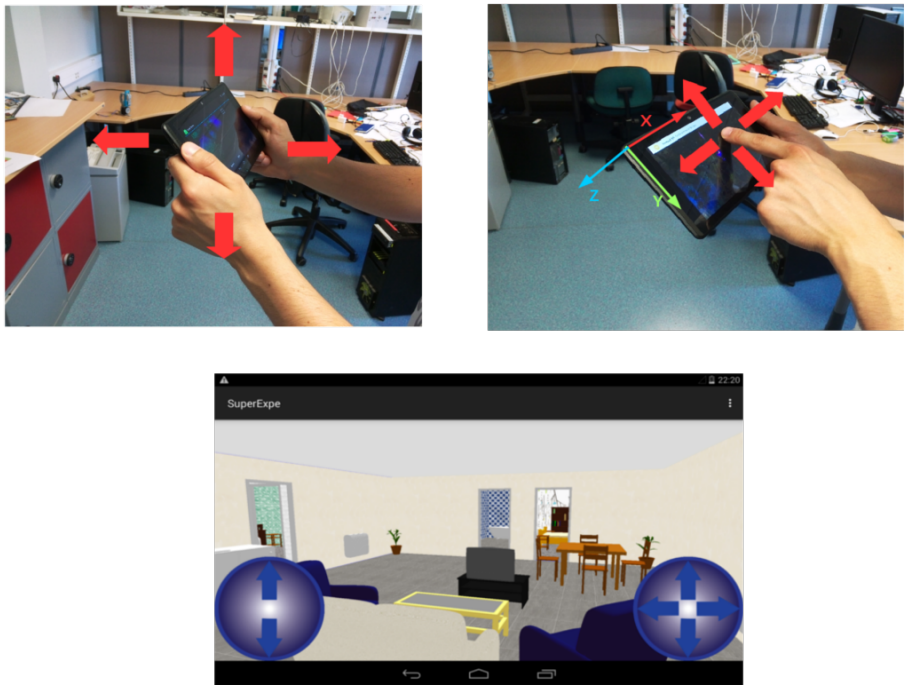


Figure E.3 – *Techniques d'interaction considérées dans cette étude : mouvements de la tablette (en haut à droite), multi-touch "classique" (en haut à gauche), et multi-touch avec sticks (en bas).*

E.3.1 Description

Nous avons mené une étude expérimentale visant à évaluer l'utilisation des mouvements d'une tablette pour la navigation dans un environnement 3D virtuel. L'objectif de cette étude était de vérifier que l'utilisation des mouvements de la tablette pour contrôler les translations et le rotations s'avère pertinent pour une tâche d'exploration 3D. Nous avons comparé cette navigation tangible avec des techniques de navigation tactile classiques.

L'évaluation des différentes techniques tactiles et tangibles s'est basée sur une évaluation subjective et sur la représentation mentale de l'EV visité par les participants.

La représentation mentale de l'EV par les participants a été évaluée à l'aide de sketch maps [10]. L'évaluation subjective a été réalisée à l'aide d'un questionnaire SUS [12] et d'un questionnaire d'acceptabilité [59].

E.3.2 Matériel

Figure E.4 – *Vue du dessus de l'environnement virtuel visié par les participants lors de l'exploration.*

L'environnement virtuel utilisé (voir fig.E.4) consistait en un appartement de cinq pièces. composé d'une entrée, d'un salon (incluant une cuisine), une salle de bain et deux chambres. Chaque pièce était connectée au salon, une fenêtre était placé dans chacune d'elle à l'exception de l'entrée.

Cet appartement virtuel a été dessiné à l'échelle réelle. Nous avons donc appliqué un facteur d'échelle afin de pouvoir naviguer dedans en utilisant les seuls mouvements de la tablette dans une pièce de taille raisonnable. Afin d'évaluer l'influence de ce facteur d'échelle, nous avons utilisé deux valeurs différentes pour la tâche d'exploration. L'environnement virtuel utilisé pour la

navigation tangible consistait en une pièce vide suffisamment grande pour que l'environnement virtuel puisse être exploré en se déplaçant dans cette pièce lors de l'utilisation des mouvements de la tablette pour naviguer.

Nous avons considéré les modalités d'interaction suivantes pour cette étude (voir fig.E.3) :

- **Tactile classique (M) :** avec un seul doigt, l'utilisateur oriente le point de vue de la caméra. Avec deux doigts l'utilisateur avance dans l'EV.
- **Utilisation de sticks (S) :** les déplacements se font via des contrôles plus proches du jeu vidéo, un stick contrôle les déplacements selon z, et un stick contrôle la rotation du point de vue.
- **Tangible échelle 1 :1 (T1) :** les déplacements se font en utilisant l'interface comme fenêtre donnant sur l'EV, qui est reproduit à l'échelle 1 :1.
- **Tangible échelle 1 :2.5 (T2) :** les déplacements se font en utilisant l'interface comme fenêtre donnant sur l'EV, qui est reproduit à l'échelle 1 :2.5.

Les techniques de navigation ont été implémentées sur une tablette Google Tango, fonctionnant avec Android 4.4, et disposant d'un algorithme de motion tracking performant.

E.3.3 Mesures

La fidélité des sketch maps a été évaluée selon les critères suivants :

- Respect des proportions entre les différents éléments
- Écart entre les ratios réels et représentés : nous avons mesuré chaque ratio pertinent sur les sketch maps (ratio entre les longueurs et largeurs de chaque pièce par exemple) et les avons comparés aux ratios réels

Les trajectoires et orientations de participants ont aussi été enregistrées durant la phase d'exploration, le tableau E.1 décrit les différentes mesures effectuées à partir de ces trajectoires.

E.3.4 Procédure

53 participants ont été recrutés et répartis parmi les 4 conditions, nous avons assigné une modalité de navigation à chaque participant. Chaque participant a disposé de 20 minute pour explorer l'environnement virtuel.

Une fois l'exploration terminée, nous avons demandé aux participants de réaliser un plan 2D de l'appartement visité faisant apparaître les murs, les portes et les fenêtres. Une fois le plan 2D réalisé, chaque participant a répondu à un questionnaire d'utilisabilité et d'acceptabilité.

Valeur	Description
Total_distance	Distance totale parcourue
Back_distance	Distance totale parcourur à reculon
Theta_distance	Distance angulaire totale parcourue
Phi_distance	Distance angulaire verticale totale parcourue
Percentage_back	Pourcentage du temps d'exploration passé à reculer
Percentage_static	Pourcentage du temps d'exploration passé à l'arrêt
Total_collisions	Nombre total de collisions
Back_collisions	Nombre total de collisions effectuées en reculant

Tableau E.1 – *Grandeurs mesurées à partir des trajectoires des participants*

E.4 Résultats

E.4.1 Évaluation subjective

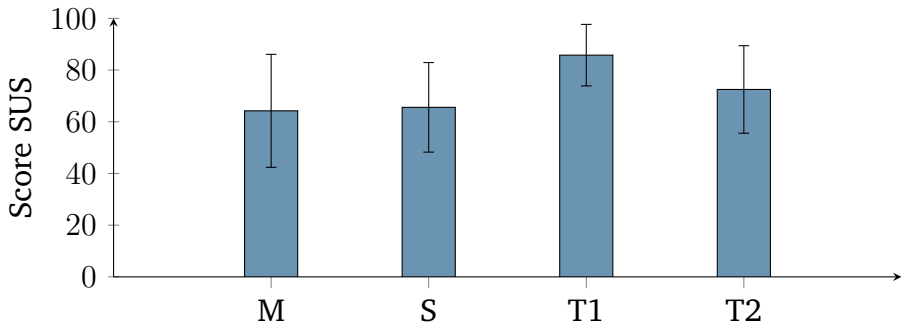


Figure E.5 – *Score SUS moyen pour chacune des modalités testées*

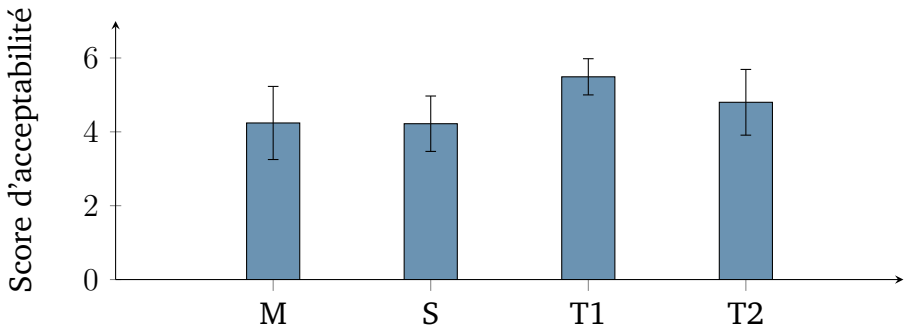


Figure E.6 – *Score d'acceptabilité moyen pour chacune des modalités testées.*

L'évaluation subjective concerne les questionnaires d'utilisabilité et d'acceptabilité. Les résultats sont présentés fig.E.5 et fig.E.6. Les T-tests suggèrent

que la modalité T1 a été la mieux appréciée en termes d'utilisabilité que les modalités M $p < 0.01$), S ($p < 0.01$), et T2 ($p = 0.03$). Il en va de même pour l'acceptabilité, où T1 a obtenu de meilleurs résultats que M ($p < 0.01$), S ($p < 0.01$) et T1 ($p = 0.022$).

Cependant, T2 n'a pas obtenu de bien meilleurs scores que M ou S en termes d'utilisabilité et d'acceptabilité. Ce qui nous amène à penser qu'un facteur d'échelle trop important pour la translation peut rendre l'utilisation des mouvements de la tablette pour naviguer moins efficace que l'utilisation du tactile.

E.4.2 Évaluation des sketch maps

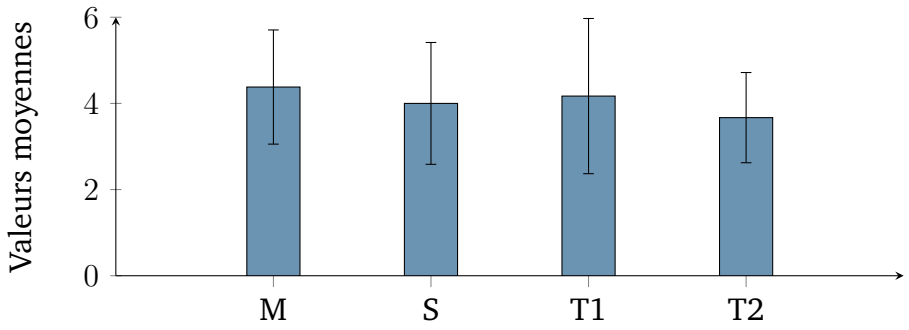


Figure E.7 – *Valeurs moyennes pour la justesse des proportions restituées, le score maximal est de 6.*

Pour chaque sketch map, nous avons évalué le ratio global (quel côté était le plus long) de chaque pièce reporté par les particpants, et l'avons comparé au modèle réel. Un point a été donné par rapport global juste. Fig.E.7 montre le score moyen pour cette tâche. Nous n'avons pas trouvé de différence significative pout ce score entre les différentes modalités.

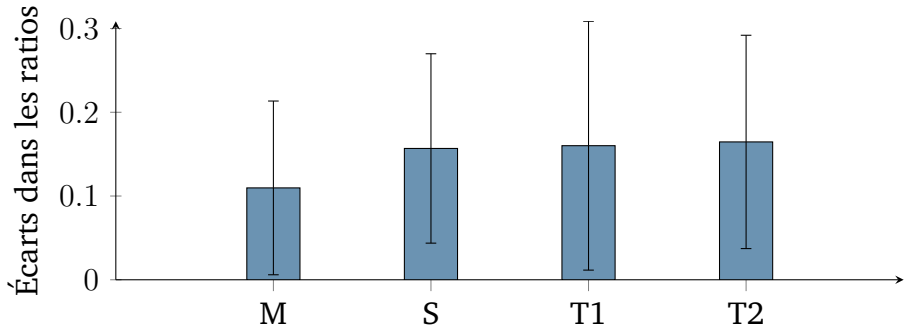


Figure E.8 – *Écarts moyens entre les ratios représentés et réels.*

De même pour les écarts entre les ratios représentés et réels, il semble qu'il n'y ait pas de différence notable, comme montré fig.E.8. Etant donné que ce score représente un écart entre ce qui est représenté et la réalité, plus il est faible et

plus le score est meilleur. Les participants ayant utilisé la modalité M semblent cependant avoir été un peu plus performants pour cette tâche étant donné que leur score moyen est le plus faible.

D'après les retours des participants, l'environnement virtuel utilisé n'était pas assez complexe, et les résultats obtenus dépendaient uniquement de leur capacité à mémoriser des objets.

E.4.3 Analyse des trajectoires

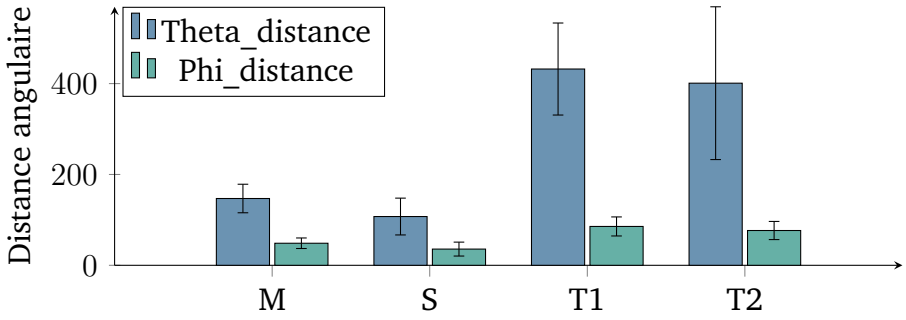


Figure E.9 – *Distances angulaires moyennes.*

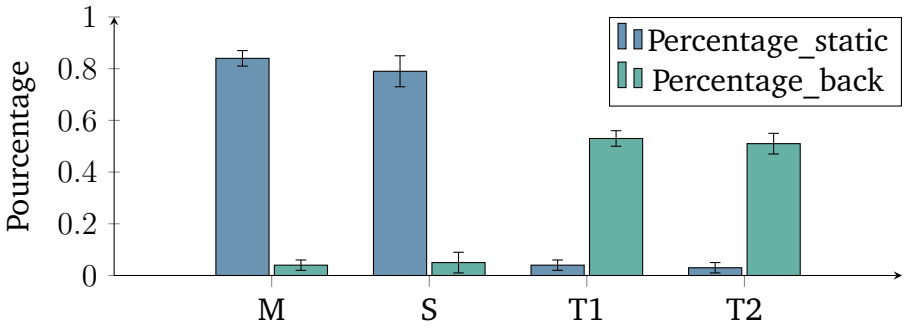


Figure E.10 – *Pourcentages moyens de temps d'exploration statique et à reculons.*

Les valeurs moyennes de balayages angulaires sont représentées fig.E.9, et les pourcentages moyens de temps passé à l'arrêt ou à reculons sont présentés fig.E.10. Les analyses ont également été conduites à l'aide de T-tests.

En termes de balayage angulaire horizontal, les participants utilisant T1 ont parcouru plus de distance que les participants avec M ($p < 0.01$) ou S ($p < 0.01$). De même, un balayage angulaire horizontal plus important a été effectué par les participants utilisant T2 par rapport à S ($p < 0.01$) et S ($p < 0.01$).

Les mêmes résultats apparaissent pour les balayages angulaires verticaux : les participants utilisant T1 ont parcouru une plus grande distance angulaire que M ($p < 0.01$) et que S ($p < 0.01$). Il en va de même avec T2, où les participants ont parcouru une plus grande distance verticale que M ($p < 0.01$) et S ($p < 0.01$).

Nous n'avons pas vu de différence significative en terme de balayage angulaire entre T1 et T2.

Les participants ont passé beaucoup moins de temps à l'arrêt lors de l'utilisation de T par rapport à M ($p < 0.01$) ou S ($p < 0.01$), de même pour T2 par rapport à M ($p < 0.01$) et S ($p < 0.01$).

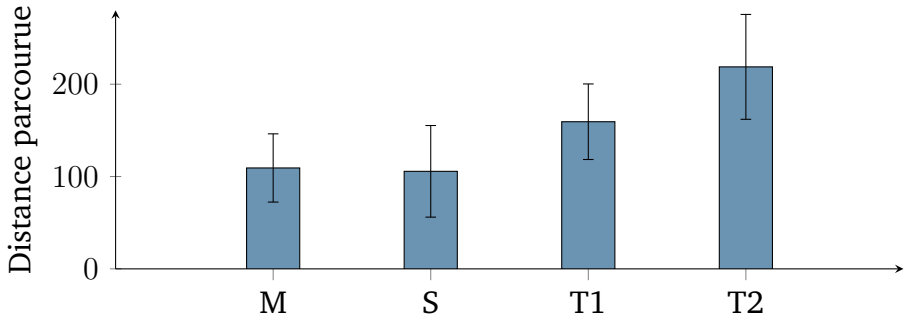


Figure E.11 – *Distance moyenne parcourue.*

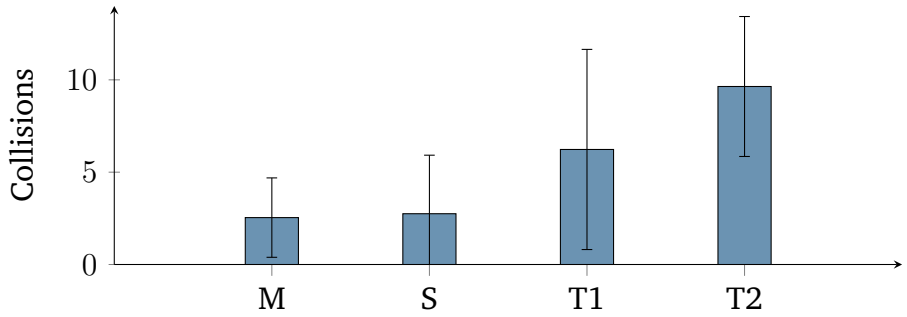


Figure E.12 – *Nombre moyen de collisions.*

Les distances totales moyennes parcourues pour chaque modalité sont représentées fig.E.11, et le nombre de collisions fig.E.12. Nous n'avons pas relevé de différence significative au niveau des collisions selon la modalité utilisée, ce nombre de collisions était plutôt corrélé à la distance totale parcourue ($r = 0.65, p < 0.01$).

E.5 Conclusion et travaux futurs

Nous avons mené une étude préliminaire ayant pour objectif d'évaluer la navigation tangible dans un environnement virtuel 3D avec une tablette. Nous avons comparé cette technique navigation tangible à d'autres techniques de navigation multi touch. Les résultats vont permettre de guider les designs de nos futures applications. Ces résultats suggèrent que l'utilisation des mouvements de la tablette pour explorer un environnement virtuel 3D semble plus naturel aux utilisateurs que l'utilisation du multi touch. Cependant, l'utilisation d'un facteur

d'échelle trop excessif pour la translation peut se montrer moins utilisable qu'une technique d'interaction tactile classique. De plus, les résultats suggèrent que même si les utilisateurs semblent occuper plus l'espace lors de l'utilisation des mouvements de la tablette (plus grandes distances angulaires parcourues, moins de temps immobile, plus de surface parcourue), ils n'ont pas été plus performants lors des tâches de sketch maps.

D'après nos études, nous ne pouvons affirmer que l'utilisation des mouvements de la tablette pour explorer un environnement virtuel 3D s'avère plus efficace que l'utilisation de techniques multi touch classiques. Cependant, l'utilisation des mouvements de la tablette semble plus intuitive pour les utilisateurs lorsque le facteur d'échelle est adapté. Lors de nos prochains travaux, nous voudrions nous concentrer sur l'adaptation du facteur d'échelle en translation par rapport aux dimensions de l'environnement virtuel, afin de tirer parti à la fois de l vitesse de navigation, et de l'aisance des utilisateurs à naviguer dans cet environnement virtuel. Enfin, nous voudrions évaluer l'usage des mouvements de la tablette pour naviguer dans un environnement plus complexe, et évaluer la mémorisation par les utilisateurs des objets et de l'environnement.

Bibliographie

[1] Antonio Adan and Daniel Huber. 3d reconstruction of interior wall surfaces under occlusion and clutter. In *2011 International Conference on 3D Imaging, Modeling, Processing, Visualization and Transmission*, pages 275–281. IEEE, 2011.

[2] Dimitrios S Alexiadis, Dimitrios Zarpalas, and Petros Daras. Real-time, full 3-d reconstruction of moving foreground objects from multiple consumer depth cameras. *IEEE Transactions on Multimedia*, 15(2) :339–358, 2013.

[3] K Somani Arun, Thomas S Huang, and Steven D Blostein. Least-squares fitting of two 3-d point sets. *IEEE Transactions on pattern analysis and machine intelligence*, (5) :698–700, 1987.

[4] Tim Bailey and Hugh Durrant-Whyte. Simultaneous localization and mapping (slam) : Part ii. *IEEE Robotics & Automation Magazine*, 13(3) :108–117, 2006.

[5] Tim Bailey, Juan Nieto, Jose Guivant, Michael Stevens, and Eduardo Nebot. Consistency of the ekf-slam algorithm. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 3562–3568. IEEE, 2006.

[6] Adrian Barbu and Song-Chun Zhu. Generalizing swendsen-wang to sampling arbitrary posterior probabilities. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(8) :1239–1253, 2005.

[7] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. Speeded-up robust features (surf). *Computer vision and image understanding*, 110(3) :346–359, 2008.

[8] Paul J. Besl and Neil D. McKay. A method for registration of 3-d shapes. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(2) :239–256, February 1992.

[9] Peter Biber and Wolfgang Straßer. The normal distributions transform : A new approach to laser scan matching. In *IROS*, volume 3, pages 2743–2748, 2003.

[10] Mark Billinghurst and Suzanne Weghorst. The use of sketch maps to measure cognitive maps of virtual environments. In *Virtual Reality Annual International Symposium, 1995. Proceedings.*, pages 40–47. IEEE, 1995.

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

[12] John Brooke. Sus-a quick and dirty usability scale. *Usability evaluation in industry*, 189(194) :4–7, 1996.

[13] John Canny. A computational approach to edge detection. *IEEE Transactions on pattern analysis and machine intelligence*, (6) :679–698, 1986.

[14] M Emre Celebi, Hassan A Kingravi, and Patricio A Vela. A comparative study of efficient initialization methods for the k-means clustering algorithm. *Expert systems with applications*, 40(1) :200–210, 2013.

[15] P Cheeseman, R Smith, and M Self. A stochastic map for uncertain spatial relationships. In *4th International Symposium on Robotic Research*, pages 467–474, 1987.

[16] Yang Chen and Gérard Medioni. Object modelling by registration of multiple range images. *Image and vision computing*, 10(3) :145–155, 1992.

[17] Sungjoon Choi, Qian-Yi Zhou, and Vladlen Koltun. Robust reconstruction of indoor scenes. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5556–5565, 2015.

[18] Michael Connor and Piyush Kumar. Fast construction of k-nearest neighbor graphs for point clouds. *IEEE transactions on visualization and computer graphics*, 16(4) :599–608, 2010.

[19] Corinna Cortes and Vladimir Vapnik. Support-vector networks. *Machine learning*, 20(3) :273–297, 1995.

[20] James M Coughlan and Alan L Yuille. Manhattan world : Compass direction from a single image by bayesian inference. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 941–947. IEEE, 1999.

[21] Franklin C Crow. Summed-area tables for texture mapping. In *ACM SIGGRAPH computer graphics*, volume 18, pages 207–212. ACM, 1984.

[22] Brian Curless and Marc Levoy. A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 303–312. ACM, 1996.

[23] Leonardo Dagum and Ramesh Menon. Openmp : an industry standard api for shared-memory programming. *IEEE computational science and engineering*, 5(1) :46–55, 1998.

[24] Angela Dai, Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Christian Theobalt. Bundlefusion : Real-time globally consistent 3d reconstruction using on-the-fly surface re-integration. *arXiv preprint arXiv :1604.01093*, 2016.

[25] Zhuo Deng, Sinisa Todorovic, and Longin Jan Latecki. Unsupervised object region proposals for rgb-d indoor scenes. *Computer Vision and Image Understanding*, 154 :127–136, 2017.

[26] Ivan Dryanovski, Carlos Jaramillo, and Jizhong Xiao. Incremental registration of rgb-d images. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 1685–1690. IEEE, 2012.

[27] Richard O Duda and Peter E Hart. Use of the hough transformation to detect lines and curves in pictures. *Communications of the ACM*, 15(1) :11–15, 1972.

[28] Hugh Durrant-Whyte and Tim Bailey. Simultaneous localization and mapping : part i. *IEEE robotics & automation magazine*, 13(2) :99–110, 2006.

[29] Chuck Eastman, Charles M Eastman, Paul Teicholz, and Rafael Sacks. *BIM handbook : A guide to building information modeling for owners, managers, designers, engineers and contractors*. John Wiley & Sons, 2011.

[30] Alberto Elfes and Larry Matthies. Sensor integration for robot navigation : combining sonar and stereo range data in a grid-based representataion. In *Decision and Control, 1987. 26th IEEE Conference on*, volume 26, pages 1802–1807. IEEE, 1987.

[31] Jakob Engel, Thomas Schöps, and Daniel Cremers. Lsd-slam : Large-scale direct monocular slam. In *European Conference on Computer Vision*, pages 834–849. Springer, 2014.

[32] Can Erdogan, Manohar Paluri, and Frank Dellaert. Planar segmentation of rgbd images using fast linear fitting and markov chain monte carlo. In *Computer and Robot Vision (CRV), 2012 Ninth Conference on*, pages 32–39. IEEE, 2012.

[33] Martin Ester, Hans-Peter Kriegel, Jörg Sander, Xiaowei Xu, et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Kdd*, volume 96, pages 226–231, 1996.

[34] George W Fitzmaurice, Shumin Zhai, and Mark H Chignell. Virtual reality for palmtop computers. *ACM Transactions on Information Systems (TOIS)*, 11(3) :197–218, 1993.

[35] Alex Flint, Anthony Dick, and Anton Van Den Hengel. Thrift : Local 3d structure recognition. In *Digital Image Computing Techniques and Applications, 9th Biennial Conference of the Australian Pattern Recognition Society on*, pages 182–188. IEEE, 2007.

[36] Brian Fulkerson, Andrea Vedaldi, and Stefano Soatto. Class segmentation and object localization with superpixel neighborhoods. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 670–677. IEEE, 2009.

[37] Vincent Garcia, Eric Debreuve, and Michel Barlaud. Fast k nearest neighbor search using gpu. In *Computer Vision and Pattern Recognition Workshops, 2008. CVPRW'08. IEEE Computer Society Conference on*, pages 1–6. IEEE, 2008.

[38] Henri Gouraud. Continuous shading of curved surfaces. *IEEE transactions on computers*, 100(6) :623–629, 1971.

[39] Michael Grabner, Helmut Grabner, and Horst Bischof. Fast approximated sift. In *Asian conference on computer vision*, pages 918–927. Springer, 2006.

[40] Michael Greenspan and Mike Yurick. Approximate kd tree search for efficient icp. In *3-D Digital Imaging and Modeling, 2003. 3DIM 2003. Proceedings. Fourth International Conference on*, pages 442–448. IEEE, 2003.

[41] E Grilli, F Menna, and F Remondino. A review of point clouds segmentation and classification algorithms. *The International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 42 :339, 2017.

[42] Enrico Grosso, Giulio Sandini, and Massimo Tistarelli. 3d object reconstruction using stereo and motion. *IEEE Transactions on Systems, Man, and Cybernetics*, 19(6) :1465–1476, 1989.

[43] Florimond Guéniat, Julien Christophe, Yoren Gaffary, Adrien Girard, and Mehdi Ammi. Tangible windows for a free exploration of wide 3d virtual environment. In *Proceedings of the 19th ACM Symposium on Virtual Reality Software and Technology*, VRST '13, pages 115–118, New York, NY, USA, 2013. ACM.

[44] Yulan Guo, Mohammed Bennamoun, Ferdous Sohel, Min Lu, and Jianwei Wan. 3d object recognition in cluttered scenes with local surface features : a survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 36(11) :2270–2287, 2014.

[45] Saurabh Gupta, Pablo Arbelaez, and Jitendra Malik. Perceptual organization and recognition of indoor scenes from rgb-d images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 564–571, 2013.

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

[47] Lei Han and Lu Fang. Flashfusion : Real-time globally consistent dense 3d reconstruction using cpu computing. Pittsburgh, Pennsylvania, June 2018.

[48] Mark Hancock, Sheelagh Carpendale, and Andy Cockburn. Shallow-depth 3d interaction : design and evaluation of one-, two-and three-touch

techniques. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 1147–1156. ACM, 2007.

[49] Miles Hansard, Seungkyu Lee, Ouk Choi, and Radu Patrice Horaud. *Time-of-flight cameras : principles, methods and applications*. Springer Science & Business Media, 2012.

[50] Chris Harris and Mike Stephens. A combined corner and edge detector. In *Alvey vision conference*, volume 15, pages 10–5244. Manchester, UK, 1988.

[51] Peter Henry, Michael Krainin, Evan Herbst, Xiaofeng Ren, and Dieter Fox. Rgb-d mapping : Using kinect-style depth cameras for dense 3d modeling of indoor environments. *The International Journal of Robotics Research*, 31(5) :647–663, 2012.

[52] Richard Hoffman and Anil K Jain. Segmentation and classification of range images. *IEEE transactions on pattern analysis and machine intelligence*, (5) :608–620, 1987.

[53] Dirk Holz, Stefan Holzer, Radu Bogdan Rusu, and Sven Behnke. Real-time plane segmentation using RGB-D cameras. In *RoboCup 2011 : robot soccer world cup XV*, pages 306–317. Springer, 2011.

[54] Dirk Holz, Alexandru E Ichim, Federico Tombari, Radu B Rusu, and Sven Behnke. Registration with the point cloud library : A modular framework for aligning in 3-d. *IEEE Robotics & Automation Magazine*, 22(4) :110–124, 2015.

[55] Stefan Holzer, Radu Bogdan Rusu, Michael Dixon, Suat Gedikli, and Nassir Navab. Adaptive neighborhood selection for real-time surface normal estimation from organized point cloud data using integral images. In *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*, pages 2684–2689. IEEE, 2012.

[56] Seung-Hyun Hong and Bahram Javidi. Improved resolution 3d object reconstruction using computational integral imaging with time multiplexing. *Optics Express*, 12(19) :4579–4588, 2004.

[57] Berthold Klaus Paul Horn. Extended gaussian images. *Proceedings of the IEEE*, 72(12) :1671–1686, 1984.

[58] Jianbing Huang and Chia-Hsiang Menq. Automatic data segmentation for geometric feature extraction from unorganized 3-d coordinate points. *IEEE Transactions on Robotics and Automation*, 17(3) :268–279, 2001.

[59] Gabriela Ibanescu. Facteurs d'acceptation et d'utilisation des technologies d'information : une étude empirique sur l'usage du logiciel" rational suite" par les employés d'une grande compagnie de services informatiques. 2011.

[60] Amy Ingram, Xiaoyu Wang, and William Ribarsky. Towards the establishment of a framework for intuitive multi-touch interaction design. In *Proceedings of the International Working Conference on Advanced Visual Interfaces*, pages 66–73. ACM, 2012.

[61] Shahram Izadi, David Kim, Otmar Hilliges, David Molyneaux, Richard Newcombe, Pushmeet Kohli, Jamie Shotton, Steve Hodges, Dustin Freeman, Andrew Davison, et al. 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*, pages 559–568. ACM, 2011.

[62] Anil K Jain. Data clustering : 50 years beyond k-means. *Pattern recognition letters*, 31(8) :651–666, 2010.

[63] Shuangshuang Jin, Robert R Lewis, and David West. A comparison of algorithms for vertex normal computation. *The Visual Computer*, 21(1) :71–82, 2005.

[64] Jaehoon Jung, Sungchul Hong, Seongsu Jeong, Sangmin Kim, Hyoungsig Cho, Seunghwan Hong, and Joon Heo. Productive modeling for development of as-built bim of existing indoor structures. *Automation in Construction*, 42 :68–77, 2014.

[65] Martin Kada and Laurence McKinley. 3d building reconstruction from lidar based on a cell decomposition approach. *International Archives of Photogrammetry, Remote Sensing and Spatial Information Sciences*, 38(Part 3) :W4, 2009.

[66] Olaf Kähler, Victor A Prisacariu, and David W Murray. Real-time large-scale dense 3d reconstruction with loop closure. In *European Conference on Computer Vision*, pages 500–516. Springer, 2016.

[67] Yan Ke and Rahul Sukthankar. Pca-sift : A more distinctive representation for local image descriptors. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 2, pages II–II. IEEE, 2004.

[68] Maik Keller, Damien Lefloch, Martin Lambers, Shahram Izadi, Tim Weyrich, and Andreas Kolb. Real-time 3d reconstruction in dynamic scenes using point-based fusion. In *3D Vision-3DV 2013, 2013 International Conference on*, pages 1–8. IEEE, 2013.

[69] Christian Kerl, Jurgen Sturm, and Daniel Cremers. Dense visual slam for rgb-d cameras. In *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*, pages 2100–2106. Citeseer, 2013.

[70] Klaas Klasing, Daniel Althoff, Dirk Wollherr, and Martin Buss. Comparison of surface normal estimation methods for range sensing applications. In

Robotics and Automation, 2009. ICRA'09. IEEE International Conference on, pages 3206–3211. IEEE, 2009.

[71] Matthew Klingensmith, Ivan Dryanovski, S Srinivasa, and Jizhong Xiao. Chisel : Real time large scale 3d reconstruction onboard a mobile device using spatially hashed signed distance fields. *Robotics : Science and Systems XI*, 2015.

[72] Mikko Kytö, Mikko Nuutinen, and Pirkko Oittinen. Method for measuring stereo camera depth accuracy based on stereoscopic vision. In *Three-Dimensional Imaging, Interaction, and Measurement*, volume 7864, page 78640I. International Society for Optics and Photonics, 2011.

[73] Kevin Lai, Liefeng Bo, and Dieter Fox. Unsupervised feature learning for 3d scene labeling. In *Robotics and Automation (ICRA), 2014 IEEE International Conference on*, pages 3050–3057. IEEE, 2014.

[74] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11) :2278–2324, 1998.

[75] Yann LeCun, Koray Kavukcuoglu, Clément Farabet, et al. Convolutional networks and applications in vision. In *ISCAS*, volume 2010, pages 253–256, 2010.

[76] David C Lee, Martial Hebert, and Takeo Kanade. Geometric reasoning for single image structure recovery. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 2136–2143. IEEE, 2009.

[77] Stefan Leutenegger, Margarita Chli, and Roland Y Siegwart. Brisk : Binary robust invariant scalable keypoints. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2548–2555. IEEE, 2011.

[78] Larry Li. Time-of-flight camera–an introduction. *Technical white paper*, (SLOA190B), 2014.

[79] Mingyang Li and Anastasios I Mourikis. Improving the accuracy of ekf-based visual-inertial odometry. In *IEEE Int. Conf. on Robotics and Automation (ICRA)*, pages 828–835. IEEE, 2012.

[80] Tony Lindeberg. Image matching using generalized scale-space interest points. *Journal of Mathematical Imaging and Vision*, 52(1) :3–36, 2015.

[81] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

[82] William E Lorensen and Harvey E Cline. Marching cubes : A high resolution 3D surface construction algorithm. In *ACM siggraph computer graphics*, volume 21, pages 163–169. ACM, 1987.

[83] David G Lowe. Object recognition from local scale-invariant features. In *Computer vision, 1999. The proceedings of the seventh IEEE international conference on*, volume 2, pages 1150–1157. Ieee, 1999.

[84] H Macher, T Landes, and P Grussenmeyer. Point clouds segmentation as base for as-built bim creation. *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, 2(5) :191, 2015.

[85] Martin Magnusson, Andreas Nuchter, Christopher Lorken, Achim J Lilienthal, and Joachim Hertzberg. Evaluation of 3d registration reliability and speed-a comparison of icp and ndt. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 3907–3912. IEEE, 2009.

[86] Elmar Mair, Gregory D Hager, Darius Burschka, Michael Suppa, and Gerhard Hirzinger. Adaptive and generic corner detection based on the accelerated segment test. In *European conference on Computer vision*, pages 183–196. Springer, 2010.

[87] Ameesh Makadia, Alexander Patterson, and Kostas Daniilidis. Fully automatic registration of 3d point clouds. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 1, pages 1297–1304. IEEE, 2006.

[88] David Marr and Ellen Hildreth. Theory of edge detection. *Proc. R. Soc. Lond. B*, 207(1167) :187–217, 1980.

[89] Asier Marzo, Benoît Bossavit, and Martin Hachet. Combining multi-touch input and device movement for 3d manipulations in mobile augmented reality environments. In *Proceedings of the 2Nd ACM Symposium on Spatial User Interaction*, SUI '14, pages 13–16, New York, NY, USA, 2014. ACM.

[90] Krystian Mikolajczyk and Cordelia Schmid. A performance evaluation of local descriptors. *IEEE transactions on pattern analysis and machine intelligence*, 27(10) :1615–1630, 2005.

[91] Ondrej Miksik and Krystian Mikolajczyk. Evaluation of local detectors and descriptors for fast feature matching. In *Pattern Recognition (ICPR), 2012 21st International Conference on*, pages 2681–2684. IEEE, 2012.

[92] Michael Montemerlo, Sebastian Thrun, Daphne Koller, Ben Wegbreit, et al. Fastslam : A factored solution to the simultaneous localization and mapping problem. In *Aaai/iaai*, pages 593–598, 2002.

[93] Adriano Moreira and Maribel Yasmina Santos. Concave hull : A k-nearest neighbours approach for the computation of the region occupied by a set of points. 2007.

[94] Etienne Mouragnon, Maxime Lhuillier, Michel Dhome, Fabien Dekeyser, and Patrick Sayd. Real time localization and 3d reconstruction. In *Computer*

Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on, volume 1, pages 363–370. IEEE, 2006.

[95] Claudio Mura, Oliver Mattausch, Alberto Jaspe Villanueva, Enrico Gobbetti, and Renato Pajarola. Automatic room detection and reconstruction in cluttered indoor environments with complex room layouts. *Computers & Graphics*, 44 :20–32, 2014.

[96] Richard A Newcombe, Shahram Izadi, Otmar Hilliges, David Molyneaux, David Kim, Andrew J Davison, Pushmeet Kohi, Jamie Shotton, Steve Hodges, and Andrew Fitzgibbon. Kinectfusion : Real-time dense surface mapping and tracking. In *Mixed and augmented reality (ISMAR), 2011 10th IEEE international symposium on*, pages 127–136. IEEE, 2011.

[97] Matthias Nießner, Michael Zollhöfer, Shahram Izadi, and Marc Stamminger. Real-time 3D reconstruction at scale using voxel hashing. *ACM Transactions on Graphics (TOG)*, 32(6) :169, 2013.

[98] David Nistér, Oleg Naroditsky, and James Bergen. Visual odometry. In *Computer Vision and Pattern Recognition, 2004. CVPR 2004. Proceedings of the 2004 IEEE Computer Society Conference on*, volume 1, pages I–I. Ieee, 2004.

[99] Andreas Nuchter, Kai Lingemann, and Joachim Hertzberg. Cached kd tree search for icp algorithms. In *3-D Digital Imaging and Modeling, 2007. 3DIM'07. Sixth International Conference on*, pages 419–426. IEEE, 2007.

[100] Andreas Nüchter, Kai Lingemann, Joachim Hertzberg, and Hartmut Surmann. 6d slam—3d mapping outdoor environments. *Journal of Field Robotics*, 24(8-9) :699–722, 2007.

[101] Peter Ondrúška, Pushmeet Kohli, and Shahram Izadi. Mobilefusion : Real-time volumetric surface reconstruction and dense tracking on mobile phones. *IEEE transactions on visualization and computer graphics*, 21(11) :1251–1258, 2015.

[102] Jeremie Papon, Alexey Abramov, Markus Schoeler, and Florentin Worgotter. Voxel cloud connectivity segmentation-supervoxels for point clouds. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2027–2034, 2013.

[103] Ernest PEREZ. Surf 3d : search engine, screensaver, software, smart agent. *Online*, 26(5) :33–37, 2002.

[104] François Pomerleau, Stéphane Magnenat, Francis Colas, Ming Liu, and Roland Siegwart. Tracking a depth camera : Parameter exploration for fast icp. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 3824–3829. IEEE, 2011.

[105] Victor Adrian Prisacariu, Olaf Kahler, David W Murray, and Ian D Reid. Simultaneous 3d tracking and reconstruction on a mobile phone. In *2013 IEEE International Symposium on Mixed and Augmented Reality (ISMAR)*, pages 89–98. IEEE, 2013.

[106] Tahir Rabbani and Frank Van Den Heuvel. 3d industrial reconstruction by fitting csg models to a combination of images and point clouds. *International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences (ISPRS)*, 35(B5) :2, 2004.

[107] Xiaofeng Ren, Liefeng Bo, and Dieter Fox. Rgb-(d) scene labeling : Features and algorithms. In *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*, pages 2759–2766. IEEE, 2012.

[108] Nora Ripperda and Claus Brenner. Marker-free registration of terrestrial laser scans using the normal distribution transform. *Proceedings of the ISPRS working group*, 4, 2005.

[109] Edward Rosten and Tom Drummond. Fusing points and lines for high performance tracking. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 2, pages 1508–1515. IEEE, 2005.

[110] Radu Bogdan Rusu, Nico Blodow, and Michael Beetz. Fast point feature histograms (fpfh) for 3d registration. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 3212–3217. IEEE, 2009.

[111] Radu Bogdan Rusu and Steve Cousins. 3d is here : Point cloud library (pcl). In *Robotics and automation (ICRA), 2011 IEEE International Conference on*, pages 1–4. IEEE, 2011.

[112] Radu Bogdan Rusu and Steve Cousins. 3D is here : Point Cloud Library (PCL). In *IEEE International Conference on Robotics and Automation (ICRA)*, Shanghai, China, May 9-13 2011.

[113] Radu Bogdan Rusu, Zoltan Csaba Marton, Nico Blodow, and Michael Beetz. Persistent point feature histograms for 3d point clouds. In *Proc 10th Int Conf Intel Autonomous Syst (IAS-10), Baden-Baden, Germany*, pages 119–128, 2008.

[114] Jagan Sankaranarayanan, Hanan Samet, and Amitabh Varshney. A fast all nearest neighbor algorithm for applications involving large point-clouds. *Computers & Graphics*, 31(2) :157–174, 2007.

[115] Ruwen Schnabel, Roland Wahl, and Reinhard Klein. Efficient ransac for point-cloud shape detection. In *Computer graphics forum*, volume 26, pages 214–226. Wiley Online Library, 2007.

[116] Thomas Schöps, Torsten Sattler, Christian Häne, and Marc Pollefeys. 3d modeling on the go : Interactive 3d reconstruction of large-scale scenes on

mobile devices. In *3D Vision (3DV), 2015 International Conference on*, pages 291–299. IEEE, 2015.

[117] Paul Scovanner, Saad Ali, and Mubarak Shah. A 3-dimensional sift descriptor and its application to action recognition. In *Proceedings of the 15th ACM international conference on Multimedia*, pages 357–360. ACM, 2007.

[118] Aleksandr Segal, Dirk Haehnel, and Sebastian Thrun. Generalized-icp. In *Robotics : science and systems*, volume 2, page 435, 2009.

[119] Ehud Sharlin, Benjamin Watson, Yoshifumi Kitamura, Fumio Kishino, and Yuichi Itoh. On tangible user interfaces, humans and spatiality. *Personal and Ubiquitous Computing*, 8(5) :338–346, 2004.

[120] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. Indoor segmentation and support inference from rgbd images. In *European Conference on Computer Vision*, pages 746–760. Springer, 2012.

[121] Ivan Sipiran and Benjamin Bustos. Harris 3d : a robust extension of the harris operator for interest point detection on 3d meshes. *The Visual Computer*, 27(11) :963–976, 2011.

[122] Stephen M Smith and J Michael Brady. Susan—a new approach to low level image processing. *International journal of computer vision*, 23(1) :45–78, 1997.

[123] Irwin Sobel and Gary Feldman. A 3x3 isotropic gradient operator for image processing. *a talk at the Stanford Artificial Project in*, pages 271–272, 1968.

[124] Luc Soler, Hervé Delingette, Grégoire Malandain, Nicholas Ayache, Christophe Koehl, Jean-Marie Clément, Olivier Dourthe, and Jacques Marescaux. An automatic virtual patient reconstruction from ct-scans for hepatic surgical planning. *Studies in health technology and informatics*, 70 :316–22, 2000.

[125] Bastian Steder, Radu Bogdan Rusu, Kurt Konolige, and Wolfram Burgard. Narf : 3d range image features for object recognition. In *Workshop on Defining and Solving Realistic Perception Problems in Personal Robotics at the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)*, volume 44, 2010.

[126] Frank Steinbrucker, Christian Kerl, and Daniel Cremers. Large-scale multi-resolution surface reconstruction from rgb-d sequences. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 3264–3271, 2013.

[127] Todor Stoyanov, Martin Magnusson, Henrik Andreasson, and Achim J Lilienthal. Fast and accurate scan registration through minimization of

the distance between compact 3d ndt representations. *The International Journal of Robotics Research*, 31(12) :1377–1393, 2012.

[128] Evan Suma, Sabarish Babu, Larry F Hodges, et al. Comparison of travel techniques in a complex, multi-level 3d environment. In *3D User Interfaces, 2007. 3DUI'07. IEEE Symposium on*. IEEE, 2007.

[129] Evan Suma, Gerd Bruder, Frank Steinicke, David M Krum, Mark Bolas, et al. A taxonomy for deploying redirection techniques in immersive virtual environments. In *Virtual Reality Short Papers and Posters (VRW), 2012 IEEE*, pages 43–46. IEEE, 2012.

[130] Robert H Swendsen and Jian-Sheng Wang. Nonuniversal critical dynamics in monte carlo simulations. *Physical review letters*, 58(2) :86, 1987.

[131] Petri Tanskanen, Kalin Kolev, Lorenz Meier, Federico Camposeco, Olivier Saurer, and Marc Pollefeys. Live metric 3d reconstruction on mobile phones. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 65–72, 2013.

[132] Fayez Tarsha-Kurdi, Tania Landes, Pierre Grussenmeyer, et al. Hough-transform and extended ransac algorithms for automatic detection of 3d building roof planes from lidar data. In *Proceedings of the ISPRS Workshop on Laser Scanning*, volume 36, pages 407–412, 2007.

[133] Aparna Tatavarti, John Papadakis, and Andrew R Willis. Towards real-time segmentation of 3d point cloud data into local planar regions. In *SoutheastCon, 2017*, pages 1–6. IEEE, 2017.

[134] Can Telkenaroglu and Tolga Capin. Dual-finger 3d interaction techniques for mobile devices. *Personal Ubiquitous Comput.*, 17(7) :1551–1572, October 2013.

[135] Sebastian Thrun. Simultaneous localization and mapping. In *Robotics and cognitive approaches to spatial mapping*, pages 13–41. Springer, 2007.

[136] Grit Thürrner and Charles A Wüthrich. Computing vertex normals from polygonal facets. *Journal of Graphics Tools*, 3(1) :43–46, 1998.

[137] Michael Tsang, George W. Fitzmzurice, Gordon Kurtenbach, Azam Khan, and Bill Buxton. Boom chameleon : Simultaneous capture of 3d viewpoint, voice and gesture annotations on a spatially-aware display. In *ACM SIGGRAPH 2003 Papers*, SIGGRAPH '03, pages 698–698, New York, NY, USA, 2003. ACM.

[138] Brygg Ullmer and Hiroshi Ishii. Emerging frameworks for tangible user interfaces. *IBM systems journal*, 39(3.4) :915–931, 2000.

[139] Hough Paul VC. Method and means for recognizing complex patterns, December 18 1962. US Patent 3,069,654.

[140] Vivek Verma, Rakesh Kumar, and Stephen Hsu. 3d building detection and modeling from aerial lidar data. In *Computer Vision and Pattern Recognition, 2006 IEEE Computer Society Conference on*, volume 2, pages 2213–2220. IEEE, 2006.

[141] Paul Viola, Michael J Jones, and Daniel Snow. Detecting pedestrians using patterns of motion and appearance. *International Journal of Computer Vision*, 63(2) :153–161, 2005.

[142] Desheng Wang, Oubay Hassan, Kenneth Morgan, and Nigel Weatherill. Efficient surface reconstruction from contours based on two-dimensional delaunay triangulation. *International journal for numerical methods in engineering*, 65(5) :734–751, 2006.

[143] Jan Weingarten and Roland Siegwart. Ekf-based 3d slam for structured environment reconstruction. In *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*, pages 3834–3839. IEEE, 2005.

[144] Thomas Whelan, Hordur Johannsson, Michael Kaess, John J Leonard, and John McDonald. Robust real-time visual odometry for dense rgb-d mapping. In *Robotics and Automation (ICRA), 2013 IEEE International Conference on*, pages 5724–5731. IEEE, 2013.

[145] Thomas Whelan, Michael Kaess, Maurice Fallon, Hordur Johannsson, John Leonard, and John McDonald. Kintinuous : Spatially extended kinectfusion. 2012.

[146] Min Yang and Eungki Lee. Segmentation of measured point data using a parametric quadric surface approximation. *Computer-aided design*, 31(7) :449–457, 1999.

[147] Stefan Zachow, Michael Zilske, and Hans-Christian Hege. 3d reconstruction of individual anatomy from medical image data : Segmentation and geometry processing. 2007.

[148] Catherine Zanbaka, Benjamin C Lok, Sabarish V Babu, Amy C Ulinski, Larry F Hodges, et al. Comparison of path visualizations and cognitive measures relative to travel technique in a virtual environment. *Visualization and Computer Graphics, IEEE Transactions on*, 11(6) :694–705, 2005.

[149] Ming Zeng, Fukai Zhao, Jiaxiang Zheng, and Xinguo Liu. Octree-based fusion for realtime 3d reconstruction. *Graphical Models*, 75(3) :126–136, 2013.

[150] Wei Zhang, Bing Yu, Gregory J Zelinsky, and Dimitris Samaras. Object class recognition using multiple layer boosting with heterogeneous features. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 323–330. IEEE, 2005.

[151] Yu Zhong. Intrinsic shape signatures : A shape descriptor for 3d object recognition. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pages 689–696. IEEE, 2009.

[152] Kun Zhou, Minmin Gong, Xin Huang, and Baining Guo. Data-parallel octrees for surface reconstruction. *IEEE Transactions on Visualization and Computer Graphics*, 17(5) :669–681, 2011.

[153] Jiejie Zhu, Liang Wang, Ruigang Yang, and James Davis. Fusion of time-of-flight depth and stereo for high accuracy depth maps. In *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*, pages 1–8. IEEE, 2008.