Reconstruction 3D volumétrique par vision omnidirectionnelle sur architecture massivement parallèle

Romain Rossi

To cite this version:

Romain Rossi. Reconstruction 3D volumétrique par vision omnidirectionnelle sur architecture massivement parallèle. Vision par ordinateur et reconnaissance de formes [cs.CV]. Université de Rouen, 2011. Français. ⟨NNT : ⟩. ⟨tel-01112447⟩

HAL Id: tel-01112447
https://theses.hal.science/tel-01112447v1

Submitted on 3 Feb 2015

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

Distributed under a Creative Commons CC BY-NC-SA 4.0 - Attribution - Non-commercial use - ShareAlike - International License

IRSEEM - Institut de Recherche en Systèmes Électroniques EMbarqués - EA 4353

Reconstruction 3D volumétrique par vision omnidirectionnelle sur architecture massivement parallèle

THÈSE

présentée et soutenue publiquement le 24 Juin 2011

pour l'obtention du

Doctorat de l'Université de Rouen

Discipline : Sciences et technologies de l'information
Spécialité : Automatique et traitement du signal

par

Romain Rossi
Sous la direction de Belhacène Mazari et Anne Louis

Composition du jury

Président :	M. Pascal VASSEUR	Professeur à l'Université de Rouen
Rapporteurs :	M. David FOFI	Professeur à l'Université de Bourgogne
	M. Dominique HOUZET	Professeur à l'Université de Grenoble
Examinateurs :	M. Patrick HORAIN	Maître de Conférence à Télécom SudParis
	Mme Anne LOUIS	Enseignante-chercheuse à l'ESIGELEC, Directrice de thèse
	M. Xavier SAVATIER	Enseignant-chercheur à l'ESIGELEC, Encadrant
Invité :	M. Jean-Yves ERTAUD	Enseignant-chercheur à l'ESIGELEC, Encadrant

A Valérie, pour m'avoir soutenu et supporté.

Remerciements

Je remercie tout d'abord messieurs David Fofi et Dominique Houzet pour avoir accepté d'être rapporteurs sur le présent mémoire. Je remercie aussi Patrick Horain pour ses nombreuses remarques sur ce même document. Leurs analyses pertinentes et leurs suggestions ont permis d'en améliorer sensiblement la qualité finale. Je remercie aussi monsieur Pascal Vasseur pour avoir présidé le jury devant lequel cette thèse a été soutenue.

Je remercie chaleureusement Bélahcène Mazari, mon premier directeur de thèse, pour m'avoir permis de rejoindre le laboratoire et pour avoir encadré mes travaux jusqu'à son départ de l'IRSEEM. Je remercie également Anne Louis pour avoir pris sa succession dans cette tâche et m'avoir suivi sur les derniers mois.

Je remercie tout particulièrement Xavier Savatier, responsable du pôle Instrumentation Informatique et Systèmes, pour son engagement et la qualité de son encadrement durant ces longues années, ainsi que pour la confiance qu'il m'a accordé dès le début de cette aventure. Je remercie également Jean-Yves Ertaud pour son encadrement scientifique, son soutien durant toute cette thèse et ses idées originales pour faire progresser la science.

Je remercie amicalement Nicolas Ragot qui, à l'occasion d'une rencontre fortuite en terre étrangère m'a permis de découvrir les activités du pôle IIS. Je le remercie également pour sa collaboration sur des travaux communs, qui ont menés à une publication, ainsi que pour sa bonne humeur quotidienne et son aide en toutes circonstances. Je remercie tout aussi amicalement Jean-François Layerle, Rémi Boutteau, et Fengchun Dong. Cela a été un vrai plaisir de travailler tout ce temps à vos côtés. Je remercie également Yohan Dupuis, Pierre Merriaux et Yann Duchemin pour les échanges fructueux, ou à défaut distrayants et toujours passionnants.

Mes sincères remerciements également aux deux stagiaires qui ont activement collaborés à mes travaux : Michel Momboth et Mandava Aarthi.

Je remercie la région Haute-Normandie (ou du moins, son conseil général) pour m'avoir attribué l'allocation qui a financé ces travaux. Le personnel de l'ESIGELEC et de l'IRSEEM a aussi fortement contribué à la réussite de ce projet, je les en remercie donc collectivement.

Un grand merci aux nombreux contributeurs qui ont mis au point et font évoluer les nombreux logiciels libres utilisés pour ces recherches. Je pense notamment aux projets Linux, GNU, LaTeX, Lyx, Ubuntu, Debian, OpenCV et de nombreux autres.

Je remercie mes parents et toute ma famille pour leur soutien et leur aide.

Enfin, pour son soutien et ses encouragements pendant toutes ces années, ainsi que son aide précieuse pour la relecture du mémoire et la préparation de la soutenance, je remercie de tout coeur ma femme Valérie.

Table des matières

Introduction **1**

1 Reconstruction 3D par vision **3**

1.1 Généralités 4

1.2 Modalités d'acquisition 4

1.2.1 Acquisition active 5

1.2.2 Acquisition passive 6

1.2.3 Acquisition panoramique 7

1.2.4 Synthèse 11

1.2.5 Système retenu 11

1.3 Modèles 3D 14

1.3.1 Nuages de points 14

1.3.2 Modèles polygonaux 16

1.3.3 Modèles volumétriques 17

1.3.4 Construction géométrique solide 20

1.3.5 Synthèse 22

1.4 Reconstruction 3D 22

1.4.1 Reconstruction 3D par mise en correspondance de points d'intérêt 22

1.4.2 Reconstruction volumétrique 23

1.4.3 Synthèse 26

1.5 Conclusion 26

2 Architectures de traitement **29**

2.1 Vocabulaire 30

2.2 Historique rapide de l'évolution de l'informatique et des ordinateurs . . 31

2.3 Architecture des processeurs 34

2.3.1 Organisation de la mémoire 34

2.3.2 Partage de la mémoire 35
2.3.3 Jeux d'instructions 36
2.3.4 Taxinomie de Flynn 36
2.3.5 Optimisations 37
2.4 Caractéristiques des processeurs 38
2.5 Estimation des performances 38
2.6 État de l'art des processeurs 39
2.6.1 Processeurs électroniques 40
2.6.2 Processeurs non conventionnels 49
2.7 Synthèse 52
2.7.1 Processeurs non conventionnels 52
2.7.2 Microprocesseurs 52
2.7.3 DSP et FPGA 52
2.7.4 Architectures novatrices 52
2.7.5 CPU et GPU 53
2.7.6 Solution retenue 53
2.8 Présentation détaillée de l'architecture CUDA 53
2.8.1 Organisation des threads 54
2.8.2 Organisation de la mémoire 54
2.8.3 Vue d'ensemble de l'organisation des calculs 57
2.8.4 Évaluation des performances de CUDA 57
2.9 Conclusion 59

3 Dispositif et méthode proposés **61**
3.1 Objectifs 62
3.2 Vue d'ensemble 62
3.3 Matériel 63
3.3.1 Système de vision 63
3.3.2 Plateforme robotique mobile 64
3.3.3 Ordinateurs 64
3.4 Principes au coeur de la méthode 65
3.4.1 Modélisation volumétrique 65
3.4.2 Photoconsistance 66
3.4.3 Reconstruction incrémentale 66
3.5 Implémentation massivement parallèle 67

3.5.1 Accès parallèle aux données 67
3.5.2 Structure de données 68
3.6 Description détaillée 71
3.6.1 Calcul de l'indice du voxel 72
3.6.2 Calcul des coordonnées du voxel 74
3.6.3 Estimation de la visibilité 74
3.6.4 Projection 3D vers 2D 77
3.6.5 Projection d'un voxel 78
3.6.6 Mesure de la photoconsistance 81
3.6.7 Détermination de l'état d'un voxel 82
3.7 Conclusion 83

4 Résultats **85**
4.1 Validation de la photoconsistance 87
4.1.1 Reconstruction 3D 87
4.1.2 Discussion 88
4.1.3 Conclusion 91
4.2 Estimation de la visibilité et reconstruction incrémentale 91
4.2.1 Reconstruction 3D 92
4.2.2 Temps de calcul 93
4.2.3 Discussion 94
4.3 Influence des textures 94
4.3.1 Objets texturés 95
4.3.2 Arrière-plan uniforme 96
4.3.3 Arrière-plan texturé 98
4.3.4 Suppression du bruit 99
4.3.5 Conclusion sur l'influence des textures 102
4.4 Comparaison des méthodes de projection d'un voxel 102
4.4.1 Temps de calcul brut 103
4.4.2 Reconstruction 3D 103
4.4.3 Synthèse et discussion 104
4.5 Influence du positionnement des caméras 104
4.6 Comparaison avec la méthode « Generalized Voxel Coloring » 105
4.6.1 Reconstruction 3D 106
4.7 Scène complexe 107

4.7.1 Reconstruction 3D 108
4.8 Scène réelle 109
4.8.1 Reconstruction 3D 109
4.9 Répartition des calculs 111
4.10 Profilage du temps d'exécution 113
4.11 Discussion générale 114
4.12 Conclusion 115

5 Conclusion générale et perspectives **117**

Bibliography **121**

Nomenclature **125**

Annexes 127

A Modèle de la sphère d'équivalence **129**

B Algorithme de tracé de segment de Bresenham **131**

Introduction

La présente thèse s'inscrit dans une thématique de recherche développée à l'Institut de Recherche en Systèmes Électroniques Embarqués de l'ESIGELEC depuis 2005 portant sur l'étude des capteurs catadioptriques, leur conception, leur calibration géométrique et leur exploitation pour des traitements de haut niveau. Les travaux présentés ici portent plus particulièrement sur l'utilisation d'une paire de ces capteurs à bord d'un robot mobile pour la reconstruction 3D dense de l'environnement. Les applications pour un tel système sont multiples : télé-opération, évitement d'obstacles, suivit d'objets, visites virtuelles... La disponibilité du modèle en temps réel est même souhaitable dans certaines de ces applications et c'est précisément ce point qui a été étudié durant cette thèse.

Calculer un modèle 3D de la scène environnant le robot est une tâche lourde en traitements numériques et nécessite donc un calculateur performant. Heureusement, l'évolution continue des systèmes informatiques ouvre sans cesse de nouvelles opportunités en fournissant de nouveaux outils qu'il convient d'exploiter au mieux. La récente introduction sur le marché de processeurs multi-coeurs met en évidence l'évolution actuelle de l'industrie micro-électronique vers la fabrication de processeurs de plus en plus « parallèles ». Cette méthode de traitement des données diffère sensiblement des méthodes classiques séquentielles utilisées jusqu'alors et mérite que l'on y prête attention dans le but d'exploiter ces nouvelles architectures de manière optimale.

Ces travaux de thèse proposent une méthode de reconstruction 3D basée sur une discrétisation volumétrique de la scène à reconstruire, et l'adaptation de cette méthode à un capteur de stéréovision panoramique. Deux approches de reconstruction sont présentées : une méthode statique où seule une paire d'image est utilisée, et une méthode dynamique utilisant de déplacement du porteur des caméras pour acquérir de nouveaux points de vues de la scène. Dans ce dernier cas, une méthode de reconstruction incrémentale est proposée, permettant d'enrichir et de corriger le modèle de la scène au fur et à mesure des nouvelles données acquises.

Les algorithmes proposés sont conçus pour tirer parti d'un processeur massivement parallèle présent sur une carte graphique de PC, permettant ainsi un calcul rapide.

Présentation du mémoire

Le premier chapitre de ce mémoire présente un état de l'art des méthodes de reconstruction 3D utilisant des systèmes de vision. La constitution du capteur, les différents types de stockage du modèle ainsi que les méthodes de reconstruction 3D y seront abordées.

Les différentes familles de processeurs sont présentées dans le deuxième chapitre. Un aperçu de leur architecture est présenté, ainsi que leurs spécificités et quelques critères de comparaison. Ce chapitre présente aussi quelques résultats concernant l'évaluation d'une architecture massivement parallèle sur l'une des tâches de notre algorithme, argumentant ainsi le choix de ce processeur comme cible de nos recherche.

Le troisième chapitre présente en détail les différents aspects de la méthode de reconstruction 3D que nous proposons. Les principes au coeur de la méthode sont présentés avant d'entrer dans les détails de l'implémentation. L'adéquation entre la méthode de reconstruction 3D proposée et l'architecture du processeur sélectionné est mise en avant.

Le quatrième chapitre présentera en détail diverses expérimentations réalisées afin de valider chaque partie de notre méthode de reconstruction, mais aussi le système dans son ensemble. Des séquences d'images de synthèse et d'images réelles sont utilisées et des résultats de reconstruction 3D sont présentés. De nombreuses mesures du temps d'exécution du programme mettent en avant les performances de notre système.

Enfin, un dernier chapitre viendra conclure ce mémoire en effectuant une synthèse de ces travaux.

Note concernant les unités

Dans ce documents, les unités du système international d'unités (SI) sont utilisées. Les unités de tailles de données, en particulier, respectent les recommandations internationales de la norme CEI 600027-2 visant à éviter les ambiguïtés entre les préfixes binaires et décimaux. Ces préfixes sont rappelés dans le tableau 1.

Les unités utilisent les abréviations courantes en Français :

— b : bit

— o : octet = 8 bits

Un pixel sera abrégé pxl lorsqu'il est utilisé comme unité, de même que le voxel sera abrégé vxl. Les préfixes du système SI pourront être utilisés (3 Mpxl, 50 Mvxl).

Nom	Symb.	Valeur
kilooctet	ko	$10^3 = 1000$
mégaoctet	Mo	$10^6 = 1\,000\,000$
gigaoctet	Go	$10^9 = 1\,000\,000\,000$
téraoctet	To	$10^{12} = 1\,000\,000\,000\,000$

(a) Puissances de 10

Nom	Symb.	Valeur
kibioctet	Kio	$2^{10} = 1024$
mébioctet	Mio	$2^{20} = 1\,048\,576$
gibioctet	Gio	$2^{30} = 1\,073\,741\,824$
tébioctet	Tio	$2^{40} = 1\,099\,511\,627\,776$

(b) Puissances de 2

Table 1 – Multiples de l'octet, suivant la norme CEI 600027-2

Chapitre 1

Reconstruction 3D par vision

Sommaire

1.1 Généralités **4**
1.2 Modalités d'acquisition **4**
1.2.1 Acquisition active 5
1.2.2 Acquisition passive 6
1.2.3 Acquisition panoramique 7
1.2.4 Synthèse 11
1.2.5 Système retenu 11
1.3 Modèles 3D **14**
1.3.1 Nuages de points 14
1.3.2 Modèles polygonaux 16
1.3.3 Modèles volumétriques 17
1.3.4 Construction géométrique solide 20
1.3.5 Synthèse 22
1.4 Reconstruction 3D **22**
1.4.1 Reconstruction 3D par mise en correspondance de points d'intérêt 22
1.4.2 Reconstruction volumétrique 23
1.4.3 Synthèse 26
1.5 Conclusion **26**

Établir un modèle 3D d'une scène ou d'un objet *inconnu* est un problème classique de vision par ordinateur. Cette technique appelée *reconstruction 3D* ou *modélisation 3D* consiste à retrouver les caractéristiques tridimensionnelles de la scène à partir d'images acquises par un système de vision composé d'une ou plusieurs caméras. Étant donné qu'une caméra classique ne fournit qu'une projection 2D de la scène observée, une dimension est « perdue » lors de l'acquisition. Pour être en mesure de reconstituer cette information, il est ainsi nécessaire de posséder (au minimum) un second point de vue de la scène.

Un grand nombre de solutions ont été envisagées dans la littérature, selon le nombre et le type de capteur employés, leur disposition dans la scène, les méthodes de calcul du modèle 3D ou encore les caractéristiques du modèle 3D obtenu.

Ce premier chapitre sera consacré à dresser un état de l'art de la reconstruction 3D à partir de caméras. Après une section concernant les généralités, nous étudierons les différentes modalités d'acquisition couramment utilisées : nombre et type de caméras et leur disposition dans la scène. Nous étudierons ensuite les différents types de modèles 3D existants selon le type de représentation utilisée et les informations contenues. Nous verrons par la suite comment stocker ces informations dans la mémoire d'un ordinateur, les structures de données couramment utilisées et leurs caractéristiques. Les méthodes de reconstruction 3D classiques seront ensuite présentées, en particulier la classe de méthode qui nous intéresse particulièrement : la reconstruction volumétrique.

1.1 Généralités

Le procédé de reconstruction 3D, schématisé par la figure 1.1 utilise un système d'acquisition composé d'une ou de plusieurs caméras observant une scène. Les images capturées par ces caméras sont analysées par un programme qui reconstitue un modèle 3D de cette scène. Des informations sur la géométrie des caméras sont nécessaires et sont fournies par une calibration préliminaire du système de vision.

Les éléments principaux d'un système de reconstruction 3D sont :

- Un système d'acquisition (ensemble de caméras)
- Un logiciel de traitement (méthode de reconstruction 3D)
- Une méthode d'organisation des données pour stocker le modèle 3D
- Une méthode de calibration des caméras
- Un ordinateur exécutant le programme

Au cours de ce chapitre, nous allons étudier les différentes solutions existantes pour les éléments suivants : l'acquisition, le traitement et le modèle 3D. Les autres éléments feront l'objet d'une analyse dans les chapitres suivants.

1.2 Modalités d'acquisition

Le type, le nombre et la disposition des caméras utilisées pour effectuer l'acquisition des images de la scène conditionnent en grande partie l'algorithme de reconstruction 3D utilisable par la suite. Dans cette section, nous allons étudier différentes modalités d'acquisition d'image que l'on retrouve couramment dans les applications de vision.

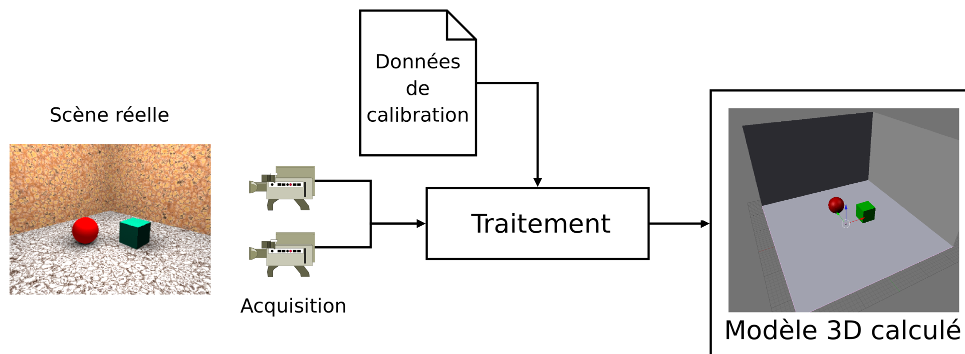


Figure 1.1 – Schéma de principe de la reconstruction 3D par vision

Nous pouvons tout d'abord séparer deux grandes catégories de méthodes : *actives* et *passives*, selon que le système d'acquisition émet ou non un signal vers la scène pour permettre la mesure. Nous étudierons brièvement quelques systèmes utilisant un système de vision active avant d'aborder plus en détail les systèmes passifs. Dans cette dernière catégorie, nous étudierons différentes dispositions classiques de caméras.

Nous aborderons ensuite l'utilisation de caméras panoramiques et nous montrerons leurs avantages et inconvénients dans le cas de notre application.

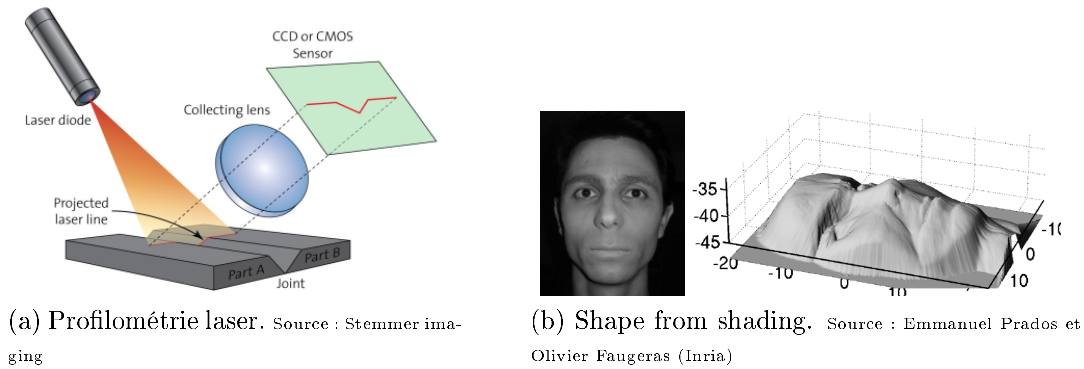
1.2.1 Acquisition active

Les solutions d'acquisition active émettent un signal connu dans la scène pour effectuer l'acquisition. Un projecteur de lumière structuré (raie ou mire laser) ou encore un vidéoprojecteur émettant un motif connu selon une direction connue et formant un angle avec la direction de visée de la caméra permet d'extraire l'information de profondeur de la scène observée.

Nous pouvons citer comme exemple de reconstruction 3D avec système actif la profilométrie laser, illustrée par la figure 1.2a, qui consiste à projeter une raie laser sur l'objet pour mettre en évidence sa structure 3D. L'utilisation d'une mire en 2D (figure 1.3a) est une autre solution, exploitée entre autre par le système Kinect de Microsoft. L'image obtenue ainsi que la position et l'angle de projection permettent de retrouver la géométrie de l'objet.

Il est aussi possible d'utiliser les informations concernant l'illumination de la scène, comme dans le cas du « shape from shading » [Hor89]. La structure 3D d'un objet convexe ou concave est alors retrouvée par le jeu des ombres grâce à un éclairage oblique. La figure 1.2b en montre un exemple. Bien que cette méthode ne soit pas à proprement parler « active », la réalisation d'un système de vision de ce type impose des contraintes sur l'éclairage similaires à celles que l'on trouve dans la méthode de profilométrie et notamment une lumière contrôlée ou modélisée.

Ces méthodes nécessitent donc un éclairage spécifique de la scène. Elles présentent un intérêt certain dans les applications où l'environnement est contrôlé ou connu (par exemple dans les applications industrielles d'inspection, la numérisation de livres, ...) mais sont difficilement applicables à un cas plus général où la structure de la scène est inconnue.



(a) Profilométrie laser. Source : Stemmer imaging

(b) Shape from shading. Source : Emmanuel Prados et Olivier Faugeras (Inria)

Figure 1.2 – Systèmes d'acquisition nécessitant une lumière contrôlée

(a) Mire infrarouge 2D projetée dans la scène

(b) Vue extérieure

Figure 1.3 – La caméra 3D Kinect de Microsoft Source : Wikipedia

Nous n'étudierons pas plus en détail ces méthodes d'acquisition et de reconstruction et ne parlerons par la suite que de la seconde catégorie : les méthodes passives.

1.2.2 Acquisition passive

Les méthodes passives utilisent la lumière naturellement présente dans la scène pour effectuer l'acquisition. Bien qu'il soit possible, dans certains cas, d'utiliser des caméras sensibles à une partie spécifique du spectre électromagnétique (caméras thermiques et infrarouges principalement) nous nous limiterons à l'étude des systèmes opérant en lumière visible.

Selon le nombre et la disposition des caméras, nous pouvons extraire plusieurs sous-catégories de méthodes d'acquisition.

Les méthodes *monoculaires* tout d'abord utilisent une seule caméra. Comme un seul point de vue de la scène ne permet pas de reconstruire un modèle 3D complet, il est nécessaire d'avoir un déplacement relatif de la caméra et de la scène pour effectuer l'acquisition de plusieurs points de vues. Cette méthode d'acquisition simple et peu coûteuse est faci-

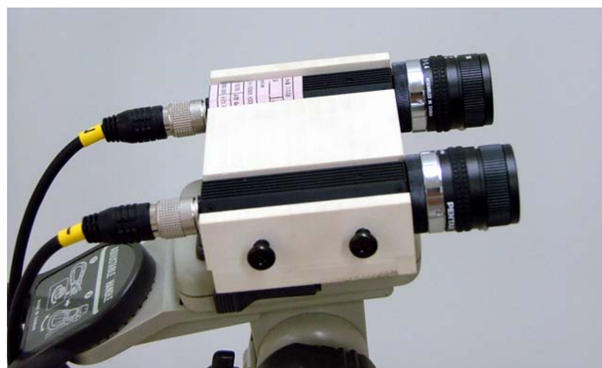


FIGURE 1.4 – Caméras en configuration binoculaire

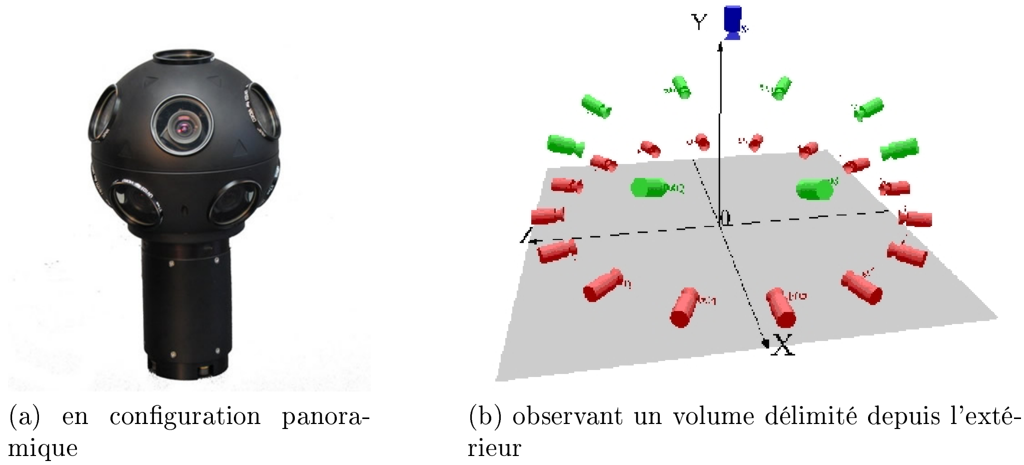
lement adaptable à toutes les situations. Cependant, les multiples points de vues acquis n'étant pas synchrones, cela peut poser problème dans le cas de scènes dynamiques. Les positions du capteur dans la scène peuvent alors être obtenues soit par un autre capteur (odométrie, télémétrie, GPS, ...) soit retrouvées grâce aux images. L'acquisition monoculaire est souvent utilisée par les algorithmes de SLAM (*Simultaneous Location And Mapping*) qui estiment de façon robuste les déplacements du capteur grâce aux images. Un état de l'art bien documenté sur ces techniques est présenté par [MFA09].

Inspirées par la vision humaine, les méthodes *binoculaires* utilisent deux caméras placées côte à côte, faiblement espacées et observant dans la même direction. La figure 1.4 présente une paire de caméras binoculaires. Les méthodes *trinoculaires*, par l'utilisation d'une caméra additionnelle observant la scène, permettent de lever certaines ambiguïtés géométriques lors de la reconstruction de la scène. Ces deux solutions proches permettent l'acquisition synchrone de multiples points de vue nécessaires au calcul de la géométrie 3D d'une scène statique ou dynamique. Leur champ de vue est cependant limité au champ de vue commun aux caméras et ne couvre qu'une petite partie de la scène.

Les systèmes multi-caméras utilisent un grand nombre de caméras (de 3 à plusieurs dizaines) qui peuvent observer la scène selon deux dispositions principales. Si les caméras sont rassemblées dans un volume restreint et observent vers l'extérieur de ce volume (en configuration « panoramique »), elle offrent ainsi un large champ de vue, couvrant jusqu'à l'intégralité de la sphère d'observation. La figure 1.5a donne un exemple d'une telle configuration. A l'inverse, si ces caméras sont disposées autour d'un volume déterminé et observent l'intérieur de ce volume comme sur la figure 1.5b, elles offrent ainsi une vue complète et redondante de la zone observée, utile pour la modélisation précise d'un objet.

1.2.3 Acquisition panoramique

L'augmentation du champ de vue du capteur est un but poursuivi par les technologies grand-angle ou panoramiques. Tout d'abord, il est possible de construire un système mécanique permettant de contrôler l'orientation d'une caméra classique. Appelées caméras cylindriques ou sphériques selon que la rotation s'effectue autour d'un axe vertical ou de deux axes perpendiculaires, ces systèmes permettent une acquisition en très haute résolution d'un environnement. Les deux problèmes majeurs de ces systèmes sont tout d'abord



(a) en configuration panoramique

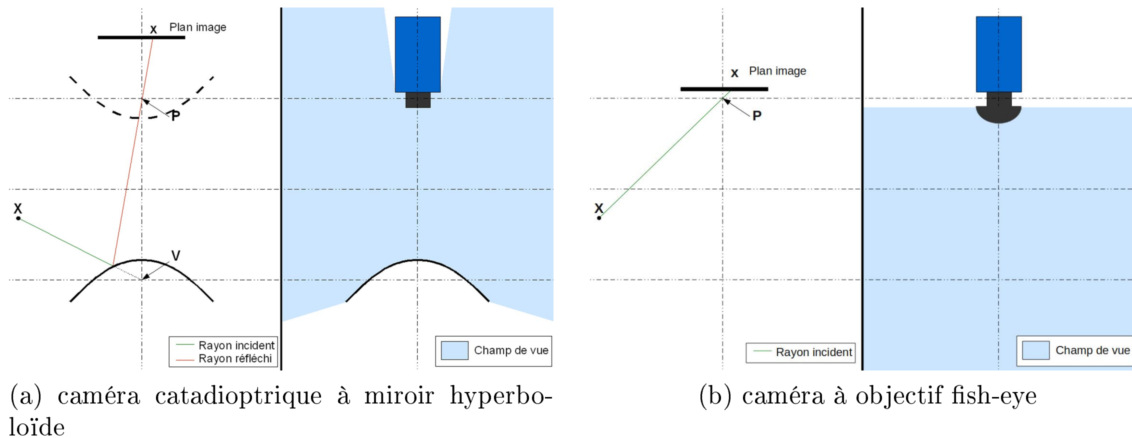
(b) observant un volume délimité depuis l'extérieur

FIGURE 1.5 – Systèmes multi-caméras...

FIGURE 1.6 – Caméra cylindrique Panoscan

la présence d'une structure mécanique mobile qui est encombrante et peut être gênée par les vibrations ou le déplacement du système l'embarquant ; ensuite le temps nécessaire à l'acquisition qui rend incompatible avec une application temps réel. Par exemple, la caméra cylindrique Panoscan MK3, représentée dans la figure 1.6, nécessite 8 secondes pour une acquisition complète [Pan09]. Ce type de capteur est aussi peu commode dans le cas d'une scène dynamique car l'acquisition n'est alors pas synchrone en tout point de l'image.

Une autre stratégie permettant d'augmenter le champ de vue consiste à associer plusieurs capteurs disposés en configuration panoramique, comme présenté dans la section précédente et illustré par la figure 1.5a. Les systèmes automobiles récents d'assistance aux manoeuvres utilisent de la même manière plusieurs caméras disposées autour du véhicule. Ces réseaux autorisent une acquisition haute résolution et rapide de l'environnement, les



(a) caméra catadioptrique à miroir hyperboloïde

(b) caméra à objectif fish-eye

FIGURE 1.7 – Champs de vues élargis obtenus grâce à un système optique

problèmes de synchronisation apparaissent cependant, surtout pour les réseaux intégrant un grand nombre de caméras. Le transfert de plusieurs images vers le dispositif de traitement peut aussi poser problème, avec l'apparition de goulot d'étranglement dans les bus de communication.

Enfin, une dernière stratégie consiste à étendre le champ de vue d'un capteur unique ou d'un faible nombre de capteurs (2 ou 3) par l'utilisation d'un système optique. Les caméras équipées d'un objectif fish-eye en sont un exemple d'utilisation courante en photographie. Les capteurs catadioptriques, formés par l'association d'une lentille et d'un miroir sont un autre exemple de cette stratégie. Les systèmes catadioptriques les plus répandus utilisent une caméra perspective dirigée verticalement, observant un miroir de révolution projetant une vue panoramique de la scène vers la caméra. Ce type de capteur fournit ainsi un large champ de vue, avec un système mécanique sans partie mobile et un seul capteur. La résolution de l'image panoramique résultante est cependant limitée par la résolution du capteur utilisé.

La figure 1.7 montre l'élargissement du champ de vue obtenu par l'utilisation de systèmes optiques.

1.2.3.1 Construction d'une caméra catadioptrique

Nayar et Baker [NB97] décrivent une grande variété d'associations objectif / miroir permettant de construire différents capteurs catadioptriques. De ces nombreuses combinaisons, seules quelques-unes présentent un champ de vue élargi, une projection centrale et sont réalisables en pratique. Au final, les constructions réellement opérationnelles de capteurs catadioptriques panoramiques centraux sont au nombre de 2 : miroir paraboloïde + objectif télécentrique ; miroir hyperboloïde + objectif perspectif.

L'association d'un objectif télécentrique avec un miroir paraboloïde permet le respect de la projection centrale sans nécessiter un positionnement rigoureux du miroir par rapport à la caméra. L'inconvénient majeur est le prix et l'encombrement de l'objectif télécentrique.

La deuxième solution consiste à associer un objectif perspectif avec un miroir hyper-

FIGURE 1.8 – Caméra catadioptrique à potence

FIGURE 1.9 – Caméra catadioptrique à cylindre de verre

boloïde. Bien que nécessitant un alignement précis du miroir avec la caméra, ce type de capteur est très compact et relativement bon marché grâce à l'utilisation d'un objectif classique.

Les figures 1.8 et 1.9 présentent deux modèles de caméra catadioptrique constitués d'un miroir hyperboloïde et d'un objectif perspectif. La figure 1.10 présente une image obtenue grâce à ce type de capteur.

1.2.3.2 Système stéréoscopique panoramique

Afin d'effectuer l'acquisition panoramique de la scène selon deux points de vues, l'utilisation d'une paire de caméras catadioptriques disposées en configuration stéréoscopique est envisageable. Dans l'article [GNT98], un système composé de deux caméras à miroir paraboloïde et objectif télécentrique est proposé. Les deux capteurs sont disposés l'un au dessus de l'autre, leurs axes optiques confondus, afin de maximiser le champ de vue commun aux deux capteurs. Les travaux présentés dans [RESM05] proposent d'utiliser une paire de caméras à miroir hyperboloïde et objectif perspectif, disposées de manière similaire. Ces deux solutions proches offrent une vue stéréoscopique et panoramique de l'environnement du capteur et permettent une acquisition rapide. Une autre solution pourrait consister à utiliser une paire de caméras équipées d'objectifs grand-angle de type fish-eye disposées elles aussi l'une au dessus de l'autre.

Le champ de vue de ces deux solutions est illustré dans la figure 1.11. Pour effectuer une acquisition des environs d'un robot mobile, le système catadioptrique fournit le champ

FIGURE 1.10 – Image obtenue par la caméra catadioptrique présentée figure 1.8

de vue le plus intéressant car le champ de vue commun aux deux capteurs couvre mieux les abords du porteur.

1.2.4 Synthèse

Selon le type de caméra utilisées (perspectives, catadioptriques, cylindriques,...), leur nombre et leur disposition dans la scène (extérieure, intérieure, distribuées, en déplacement,...) il existe un grand nombre de modalités d'acquisition d'une scène inconnue.

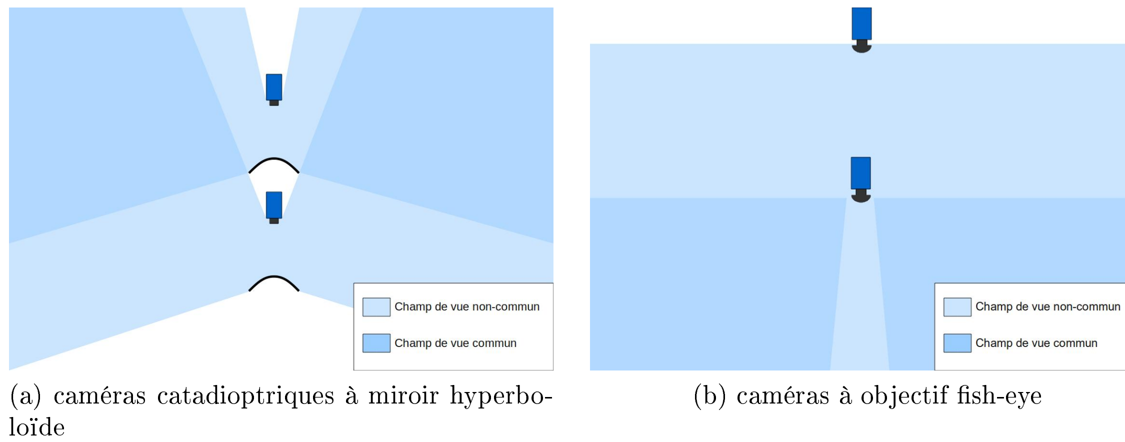
Notre objectif étant d'embarquer ce système optique à bord d'un robot mobile, les configurations possibles se limitent aux caméras adaptées. De plus, il est souhaitable que le système fournisse au minimum deux points de vues afin de pouvoir extraire des informations 3D de l'environnement même lorsque le robot est immobile relativement à la scène. Un système possédant un large champ de vue permet d'obtenir des informations sur l'ensemble du périmètre environnant le robot et nous semble être une solution intéressante.

Seules deux configurations semblent satisfaire ces contraintes : un réseau de caméras classiques disposées sur le pourtour du robot ou bien une paire de caméras panoramiques disposées en configuration stéréoscopique.

Pour limiter le nombre de caméras utilisées, nous avons choisi de développer un système stéréoscopique utilisant une paire de caméras catadioptriques. Ce système est présenté sur la figure 1.12.

1.2.5 Système retenu

Le système optique que nous avons utilisé durant ces travaux de thèse est le banc stéréoscopique présenté dans la figure 1.12. Il est composé de deux caméras couleur *Prosilica*



(a) caméras catadioptriques à miroir hyperboloïde

(b) caméras à objectif fish-eye

FIGURE 1.11 – Comparaison des champs de vue communs aux deux capteurs

FIGURE 1.12 – Capteur stéréo-catadioptrique utilisé pour nos travaux

TABLE 1.1 – Caractéristiques des caméras Prosilica GC1380C

Résolution	1360×1024, couleur (filtre de Bayer)
Capteur	2/3" CCD ExView HAD progressive scan Sony ICX285
Taille pixel	$6,45 \times 6,45\,\mu m$
Cadence maximale (pleine résolution)	20 images/secondes
Interface	IEEE 802.3 1000baseT (Gigabit Ethernet)
Temps d'exposition	$10\,\mu s$ à $60\,s$
Consommation	$3,3\,W; 12\,V$
Taille	$33 \times 46 \times 59\,mm$
Poids	$104\,g$

TABLE 1.2 – Caractéristiques des objectifs Fujinon HF9HA-1B

Focale	Fixe, $9\,mm$
Iris	$F\,1.4 \sim F\,16$
Angle de vue (2/3")	$52°06' \times 40°16'$
Focus	$0,1\,m \sim \infty$
Poids	$55\,g$

GC1380C, de deux optiques *Fujinon HF9HA-1B* et de deux miroirs *Néovision H3S*. Les caractéristiques de ces éléments sont respectivement données dans les tableaux 1.1,1.2 et 1.3. Ces éléments sont maintenus par une structure en aluminium massif spécialement réalisée pour cette application.

Chaque caméra possède un champ de vue, élargi grâce au miroir, de $-16°$ sous l'horizontale à $+80°$ au dessus.

TABLE 1.3 – Caractéristiques des miroirs Néovision H3S

Courbure	Hyperboloïde
Semi-axe a	$28,0950\,mm$
Semi-axe b	$23,4125\,mm$
Équation	$\frac{z^2}{789{,}3274} - \frac{x^2+y^2}{548{,}1440} = 1$
Matériau	Acier inoxydable
Poids	$212\,g$

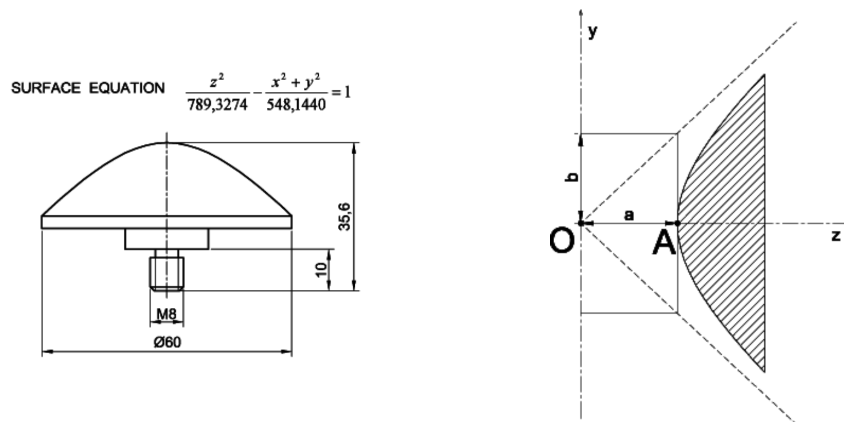


FIGURE 1.13 – Schéma du miroir Néovision H3S

1.3 Modèles 3D

Dans cette section, nous allons étudier les différentes méthodes pour définir un objet ou une scène en 3D de manière numérique et pour permettre de stocker ces informations efficacement dans la mémoire d'un ordinateur, tout en permettant leur exploitation rapide par un programme de traitement.

Ces modalités de stockage de données sont d'ores et déjà utilisées dans bon nombre de logiciels : outils de modélisation et de raytracing, jeux vidéos, CAO dans différents domaines (électronique, mécanique, hydraulique,...), imagerie médicale et bien d'autres. Nous allons ici présenter les principales méthodes de modélisation et de stockage d'objets 3D.

Suivant les applications visées, différentes informations doivent être incluses dans le modèle 3D : informations géométriques (position, taille), couleur, texture, sources de lumières, informations volumétriques (matériaux, densité, indice de réfraction), etc.

Nous pouvons dans un premier temps dresser quatre catégories de modèles qui seront détaillés par la suite :
- nuages de points,
- modèles facettés ou polygonaux,
- modèles volumétriques,
- modèles de haut niveau.

1.3.1 Nuages de points

Cette première catégorie de modèles 3D est une représentation très bas niveau de l'objet. Ce type de modèle est constitué d'une liste de points 3D disjoints représentant la forme à modéliser. L'ensemble de ces points peut représenter n'importe quelle forme géométrique et éventuellement comporter d'autres informations comme, par exemple, la couleur. Ce type de représentation est souvent utilisé par les systèmes effectuant une acquisition 3D de points discrets, comme par exemple les systèmes à télémètres laser. Elle ne conserve pas la connectivité des points, et nécessite donc une étape de post-traitement pour recréer les facettes composant la surface des objets.

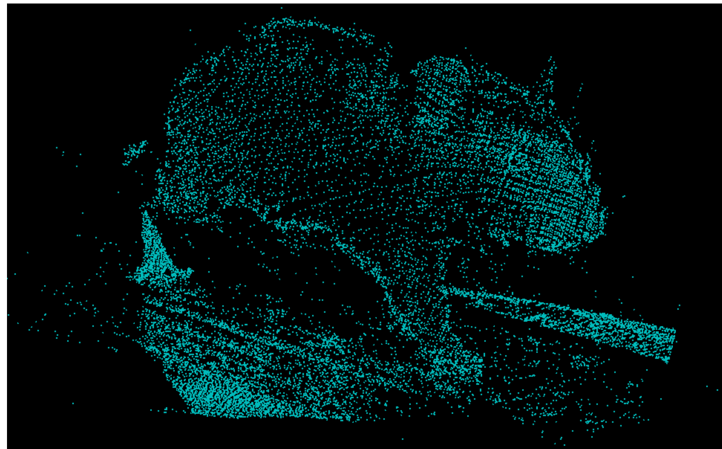


Figure 1.14 – Exemple de nuage de points 3D.

Source : http://www-evasion.imag.fr

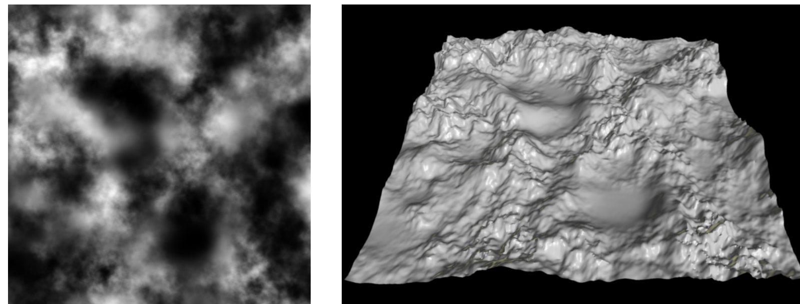


Figure 1.15 – Carte d'altitude (à gauche), les pixels clairs représentent les altitudes élevées. La même carte convertie en modèle polygonal et rendue (à droite). Source : Wikimedia Commons

Le stockage en mémoire de ce type de structure est souvent une simple liste non ordonnée de vecteurs indiquant les coordonnées de chaque point et éventuellement une couleur associée.

1.3.1.1 Carte d'altitude

Les cartes d'altitudes (*heightmap* en anglais) peuvent être vues comme un cas particulier des nuages de points. Cette représentation est principalement utilisée pour la modélisation du sol d'un environnement dans les jeux vidéos. Elle consiste à associer à chaque parcelle unitaire du terrain une altitude. Habituellement stockée sous la forme d'une image en niveaux de gris, où l'intensité de chaque pixel représente l'altitude du point associé, cette représentation est donc en 2.5D car certaines formes ne sont pas représentables (un tunnel dans une montagne, une falaise en surplomb). Le stockage en mémoire est très compact, car les coordonnées X et Y sont implicitement déduites de la position du point dans l'image ; la structure dans son ensemble pouvant de plus bénéficier des méthodes classiques de compression d'image.

Les cartes d'altitude sont souvent utilisées comme base pour la réalisation d'un modèle

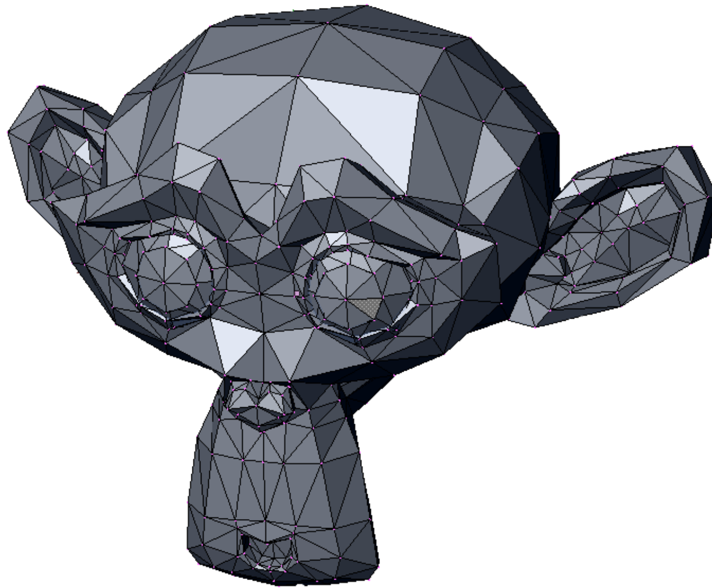


FIGURE 1.16 – Exemple de modèle polygonal.

Source : Modèle Suzanne, Blender Foundation

de scène 3D complété par des objets polygonaux dans les jeux vidéos réalisant un rendu 3D temps réel. Il est facile de colorier ce type de représentation en appliquant une autre image en couleur de même dimensions.

1.3.2 Modèles polygonaux

La modélisation polygonale approxime la surface d'un objet 3D par un polyèdre composé de multiples facettes généralement triangulaires. Ce type de représentation permet de représenter une surface fermée.

Le nombre de facettes conditionne directement la qualité du modèle obtenu : si l'on désire un modèle très précis, un grand nombre de facettes est nécessaire. A l'inverse, un modèle grossier pourra n'être constitué que de quelques polygones. La grande majorité des logiciels de modélisation 3D et des moteurs graphiques utilisent ce type de représentation pour effectuer des rendus temps réel, grâce à l'utilisation d'API graphiques (OpenGL, Direct3D) et d'une carte graphique.

Le stockage en mémoire d'une telle représentation s'effectue en général sous la forme d'une liste orientée de points 3D. Les informations de couleurs peuvent y être ajoutées directement. Il est ainsi possible d'associer une couleur à un sommet ou une arête d'un polygone, de même qu'une couleur ou une texture à une facette. Il est aussi très fréquent d'utiliser une texture sous forme d'image 2D appliquée tel un patron sur la forme géométrique 3D lors du rendu. Cette méthode permet un aspect très réaliste des objets tout en limitant la résolution géométrique de l'objet, limitant ainsi les temps de rendu.

Les représentations polygonales sont cependant limitées à la représentation d'une surface et ne contiennent aucune information concernant son volume. Ce type de représentation est très utilisée par les logiciels de CAO et les moteurs graphiques de jeux vidéos.

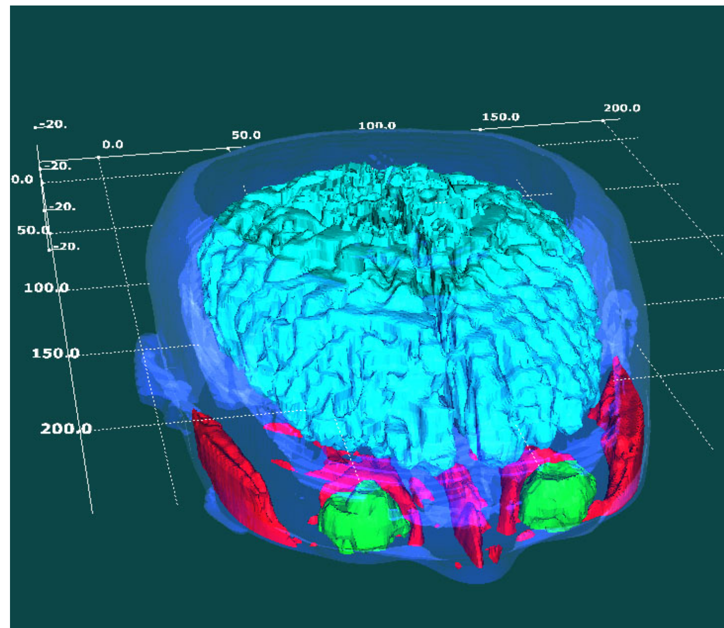


FIGURE 1.17 – Modèle volumétrique d'un cerveau humain.

Source : https://wiki.brown.edu

1.3.3 Modèles volumétriques

Ce type de modélisation défini un volume englobant, lui-même subdivisé en volumes élémentaires appelés voxels (contraction de volume et pixel) qui est donc un « pixel en 3D ». Généralement, le volume englobant ainsi que les voxels sont cubiques et de taille constante.

Pour une simple représentation géométrique d'une forme 3D, chaque voxel est marqué comme *plein* ou *vide* ; cependant une telle représentation volumétrique d'un espace 3D peut servir à stocker d'autres données : couleur (pour représenter un objet), densité et température (pour des simulations de fluides), opacité, texture, indice de réfraction, ...

Dans le cas de la modélisation d'un objet ou d'une scène, ce type de représentation permet l'approximation de n'importe quelle forme 3D. La précision de l'approximation dépend uniquement de la résolution, autrement dit du nombre de voxels par unité de volume ; comme dans le cas des pixels pour une image 2D.

1.3.3.1 Stockage d'un modèle volumétrique

Afin de stocker un modèle volumétrique en mémoire, il est tout simplement possible d'utiliser un tableau tridimensionnel statique ou dynamique.

L'inconvénient principal de ce type de structure est l'occupation mémoire conséquente car chaque voxel nécessite un emplacement, même s'il est inoccupé. Son principal avantage est de conserver un accès direct et rapide à chaque voxel, en particulier adapté aux accès concourants (dans le cas d'un programme multi-threads).

Pour stocker un modèle volumétrique, il est possible d'utiliser des structures hiérarchiques plus sophistiquées décrites par la suite : les octrees.

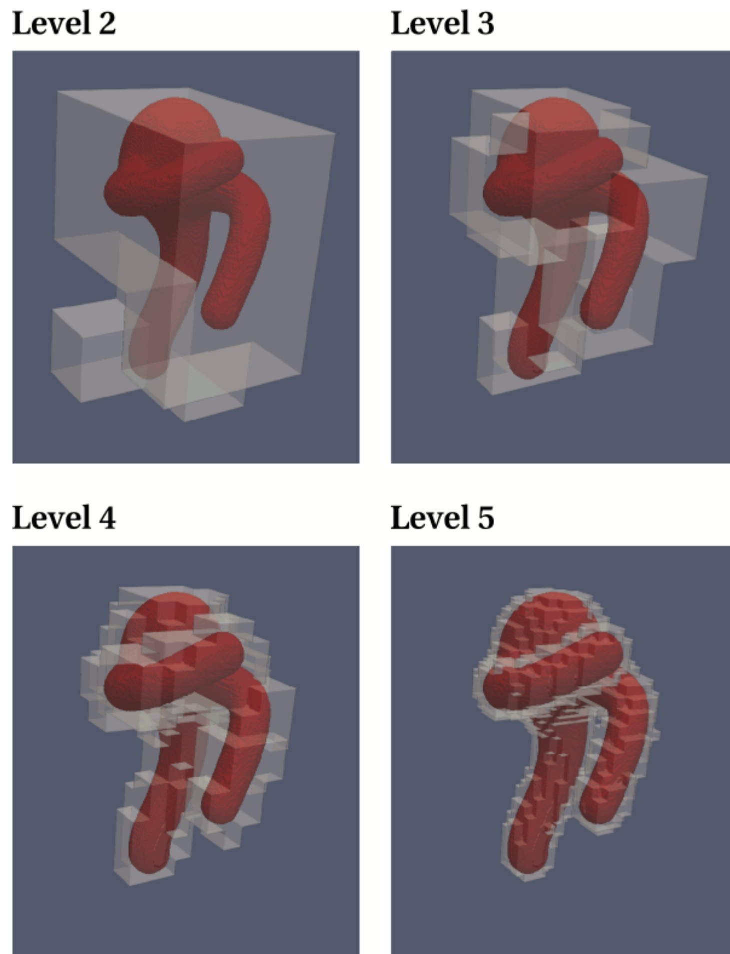


FIGURE 1.18 – Création d'un octree à différentes profondeurs.

Source : http://www.lbmethod.org

1.3.3.2 Octree

Un octree est une structure de données de type hiérarchique dans laquelle chaque noeud peut compter exactement 0 ou 8 descendants. Les octree sont le plus souvent utilisés pour partitionner un espace tridimensionnel en le subdivisant récursivement.

Chaque noeud d'un octree subdivise l'espace qu'il représente en huit sous espaces (les octants) de taille identique, le découpage étant effectué par trois plans orthogonaux entre eux dont l'intersection est le centre de l'espace représenté par le noeud. Le noeud de plus haut niveau, la *racine*, englobe tout l'espace à représenter.

Un noeud ne possédant pas de descendant est appelé une *feuille*. Suivant les implémentations, tous les noeuds ou seules les feuilles contiennent les données.

La *profondeur* d'un noeud indique le nombre de noeuds intermédiaires entre la racine et le noeud considéré. *L'ordre* d'un octree étant la profondeur maximale autorisée.

Pour construire un octree, on utilise en général une procédure récursive qui va, pour un noeud donné, déterminer si la grandeur physique que l'on veut représenter est homogène dans l'ensemble du volume représenté par ce noeud.

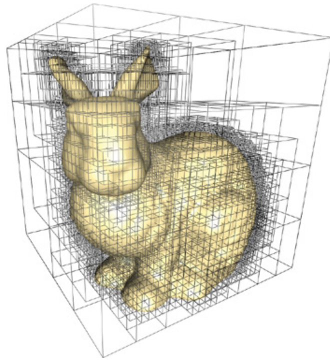


Figure 1.19 – Octree stockant un modèle 3D.

Source : GPU Gems 2

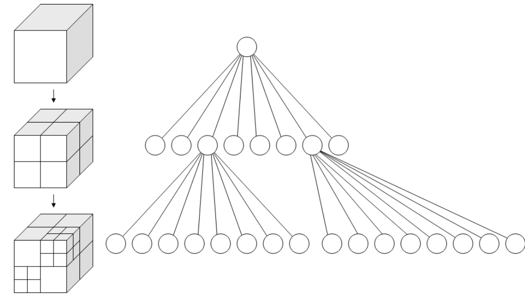


Figure 1.20 – Représentation du graphe d'un octree.

Source : Wikimedia Commons

Un exemple pratique concerne la représentation d'un objet 3D. La procédure commence par créer la racine de l'octree à partir d'une boite englobant l'objet. Ensuite, pour chaque noeud en commençant par la racine, la procédure teste si *tout* le volume représenté par ce noeud est occupé ou vide.

Si oui, le noeud est marqué comme étant une feuille, puis il est marqué entièrement occupé ou inoccupé et la procédure récursive s'arrête. Si non, le noeud est subdivisé en huit et la même procédure est appliquée aux descendants. La figure 1.18 présente les différents niveaux d'un octree en cours de construction.

Lorsque la profondeur courante est égale à l'ordre de l'octree, la procédure ne crée plus de descendants et tous les noeuds courants sont marqués comme étant des feuilles et traités en conséquence.

La figure 1.19 montre un exemple d'octree utilisé pour stocker un modèle 3D. Elle illustre bien l'intérêt principal de l'octree : les zones inoccupées du modèle étant représentées par une feuille de faible profondeur, la structure utilise donc peu de mémoire.

Variantes Il existe de multiples variantes autour de cette structure de base. Dans l'octree de type « point région », le noeud mémorise explicitement les coordonnées du point central utilisé pour effectuer la subdivision en huit. Cette structure permet notamment de représenter des espaces infinis.

Le *kd-tree* est une généralisation de l'octree pour un espace de dimension quelconque k. A chaque niveau, une seule dimension de l'espace est séparée en deux parties. Au niveau suivant, c'est l'une des autres dimensions qui sera partagée en deux.

Le quadtree enfin est une version 2D de l'octree. La racine est un carré et chaque noeud peut avoir 0 ou 4 descendants divisant en quatre parts égales la surface du parent.

Implémentation L'octree étant un arbre, comme le montre la figure 1.20, il est normalement implémenté à l'aide d'algorithmes récursifs parcourant le parcourant de haut en bas depuis la racine.

Suivant les applications, il est possible d'utiliser l'octree comme un contenant pour

diverses données. Par exemple, pour stocker une forme tridimensionnelle dans un octree, chaque feuille peut contenir l'équation du plan tangent à la forme au point localisé comme étant le centre du noeud. Ceci permet de stocker efficacement des formes facettées.

Bien que la position occupée par un noeud donné puisse être calculée à partir des coordonnées du noeud racine et de la profondeur du noeud considéré, il est fréquent de sauvegarder ces informations directement dans chacun des noeuds (ou au moins des feuilles). Cela permet notamment de gagner du temps lors de l'exploitation des données car ce calcul peut être long si l'ordre de l'octree est grand. De même, d'autres données annexes sont fréquemment stockées dans chaque noeud : profondeur, pointeur vers le noeud parent, ordre, ...

Autres utilisations courantes L'octree peut être utilisé pour toute une variété d'applications et de buts. Les exemples cités précédemment étaient surtout orientés vers le stockage d'une forme 3D en mémoire. Cependant, l'octree peut être utilisé pour partitionner d'autres espaces qu'un environnement 3D, comme par exemple, l'espace RGB des couleurs [GP88].

On peut aussi utiliser l'octree pour contenir l'ensemble des obstacles présents dans un espace donné, utilisé pour la navigation robotique. L'octree peut être utilisé à différents niveaux de profondeur pour les applications nécessitant un niveau de détail variable ; chaque noeud stockant par exemple la valeur moyenne de tous ses descendants.

1.3.4 Construction géométrique solide

Des solides 3D simples (cube, sphère, cylindre, cône, tore, ...) sont associés par des opérations booléennes (union, différence, intersection). Cette technique est appelée CSG (Constructive Solid Geometry) en Anglais.

Les solides de bases sont représentés par leur paramètres (position, orientation, taille). Ces solides et les opérations utilisés pour les composer entre eux sont stockées sous une forme arborescente (voir Fig.1.21).

Les avantages de ce type de représentation sont nombreux. Tout d'abord, les objets sont représentés de manière mathématique, les frontières sont donc « parfaites » et la précision obtenue peut être ajustée à volonté. Ce type de modélisation se prête bien aux opérations de rendu (ray-tracing) et de détection de collisions. Le stockage est aussi très efficace, car chaque primitive comporte peu d'informations (quelques coordonnées) et la structure en arbre peut être optimisée. La précision du modèle est aussi indépendante de sa taille, puisque les relations mathématiques sous-jacentes ne sont pas approchées ; cela permet de représenter des objets ou des scènes de grande dimensions sans augmentation dramatique de l'occupation mémoire. Les objets sont réellement « 3D » et le volume contenu dans ces objets est exploitable.

L'inconvénient principal est la difficulté à représenter des objets réels aux formes quelconques. Un simple ballon de baudruche gonflé d'air prend une forme de « poire » , complexe à modéliser par CSG. Bien qu'il soit facile pour un humain de construire un objet de forme simple grâce à ce type de modélisation, modéliser de façon automatique un objet réel reste complexe.

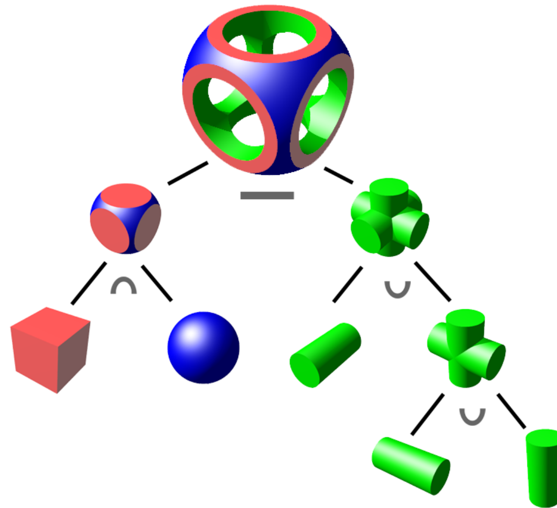


FIGURE 1.21 – Exemple de CSG mettant en évidence la structure hiérarchique de ce type de modélisation.

Source : Wikimedia Commons

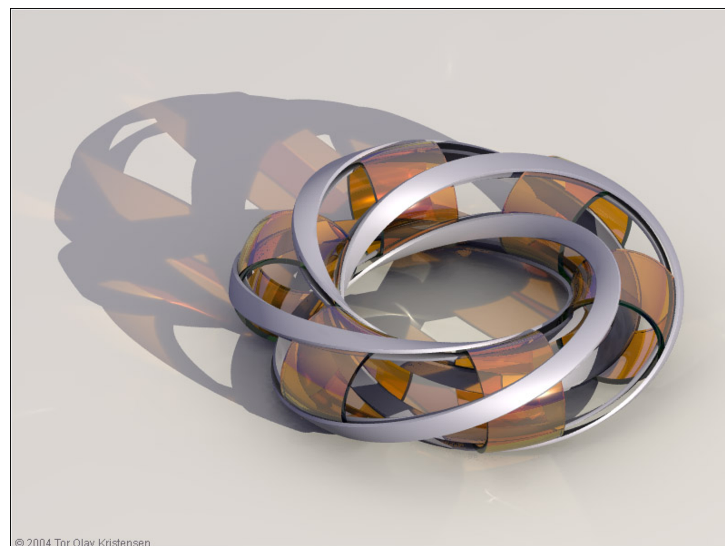


FIGURE 1.22 – Une construction utilisant la CSG.

Source : « Villarceau Circles », de Olav Kristensen

Dans les logiciels de modélisation 3D, cette forme de modélisation est rarement utilisée seule mais plutôt comme outil permettant la définition d'objets simples qui peuvent ensuite être convertis en modèles polygonaux ou combinés avec eux.

1.3.5 Synthèse

Les différents modèles présentés ici sont chacun fortement adaptés à une situation particulière. Dans le cadre de notre application, nous avons choisi de nous orienter vers un modèle volumétrique qui possède de nombreux avantages pour l'application envisagée.

Le modèle et la structure de données retenues, fortement liés au choix de l'architecture de traitement présenté dans le chapitre 2, seront développés en détail dans la section 3.5.2.

1.4 Reconstruction 3D

Dans cette section, nous présenterons quelques-unes des méthodes de reconstruction 3D courantes dans la littérature. Nous nous intéresserons tout particulièrement aux méthodes géométriques qui, d'une part, produisent un modèle 3D directement utilisable en tant que tel et, d'autre part, ne nécessitant pas de modèles a priori des objets présents dans la scène.

Nous commencerons par les méthodes les plus classiques et largement utilisées, notamment dans les applications utilisant un capteur unique ou bien une paire stéréoscopique. La seconde partie sera consacrée aux méthodes volumétriques qui présentent l'avantage de pouvoir facilement prendre en compte un nombre quelconque de points de vues et de caméras.

1.4.1 Reconstruction 3D par mise en correspondance de points d'intérêt

La première méthode de reconstruction 3D, et sans doute la plus fréquemment utilisée dans le cas de systèmes binoculaires, consiste à identifier des points remarquables sur l'une des images puis à chercher leurs correspondants sur l'autre image. La connaissance des paramètres des caméras, obtenus par une calibration préalable ou une étape d'autocalibration, permet ensuite de retrouver les coordonnées du point 3D ainsi identifié.

Cette stratégie peut être utilisée dans plusieurs cas : mise en correspondance entre images issues d'un système de vision bi- ou trinoculaire, calcul du déplacement d'une caméra unique, flot optique, SLAM (*Simultaneous Location and Mapping*)... Il est important dans ce cas que le champ de vue commun des caméras soit aussi grand que possible pour pouvoir effectuer la mise en correspondance d'un grand nombre de points.

Cette famille de méthodes présente deux points durs principaux : la recherche de points d'intérêt sur une image et l'appariement des points homologues entre plusieurs images.

La recherche de points d'intérêt utilise des opérateurs bas niveau sur l'image, comme le détecteur de Harris, afin de mettre en évidence des points remarquables : bordures, coins, maximums ou minimums locaux... Pour chaque point découvert, son correspondant sur l'autre image est ensuite recherché. La mise au point de descripteurs mathématiques

permettant l'appariement est un sujet très actif. Une très bonne synthèse des différentes méthodes existantes est donnée dans [CC04].

Le principal inconvénient de ces méthodes est le temps de calcul souvent long associé à l'appariement, ce qui limite en pratique le nombre de points appariés et donc la résolution du modèle 3D reconstruit.

Cette première famille de méthodes de reconstruction 3D est cependant intéressante dans le cas d'une configuration stéréoscopique simple avec des caméras calibrées. Dans ce cas la recherche de correspondants s'effectue sur une ligne épipolaire (1D) de l'image et non sur une région (2D), ce qui peut permettre une application temps réel par l'utilisation judicieuse d'un processeur adapté (FPGA ou DSP).

Lors de l'utilisation de caméras catadioptriques, les distorsions de l'image dues au miroir introduisent de nouvelles difficultés pour l'appariement. Deux groupes de méthodes existent : utiliser un espace projectif intermédiaire [DMB04, SC05] ou bien effectuer l'appariement directement sur l'image catadioptrique [AMWF08]. Les méthodes du premier groupe impliquent une interpolation de l'image réalisée lors de la projection intermédiaire qui peut entraîner une perte d'information. Les méthodes du deuxième groupe évitent ce problème en calculant un voisinage adapté directement sur l'image catadioptrique. La forme du voisinage étant adaptatif, il est différent pour chaque point considéré et est donc plus coûteux à calculer et parcourir.

1.4.2 Reconstruction volumétrique

Ces méthodes utilisent une représentation volumétrique de la scène, où un volume d'intérêt englobant la scène à reconstruire est subdivisé en voxels, équivalent 3D des pixels d'une image 2D. Cet échantillonnage permet de décrire n'importe quel type de scène, mais aussi d'inclure des informations sur les propriétés volumétriques des objets comme la densité, la pression, vitesse d'écoulement ... qui sont utiles dans les applications d'imageries médicale ou de simulations physiques.

Une revue des méthodes volumétriques basées sur une représentation échantillonnée de la scène 3D est disponible dans [SCMS01].

Nous allons ici présenter cette famille de méthodes de reconstruction 3D.

1.4.2.1 Voxel Coloring

Introduit par [SD97], la méthode de *Voxel Coloring* a pour but de créer une reconstruction 3D d'un objet ou d'une scène, consistante avec l'ensemble des images acquises de cette scène.

La scène est tout d'abord discrétisée suivant une grille régulière de voxels, la taille de la grille ainsi que sa résolution sont choisies en fonction de l'application. Cette scène est observée par un réseau de caméras disposées sur le pourtour de la scène et observant celle-ci selon plusieurs points de vues. L'application présentée dans cet article utilise un nombre relativement important de points de vues : 21 images représentant l'objet à reconstruire acquises en faisant tourner l'objet de 17° entre chaque acquisition face à une caméra unique fixe.

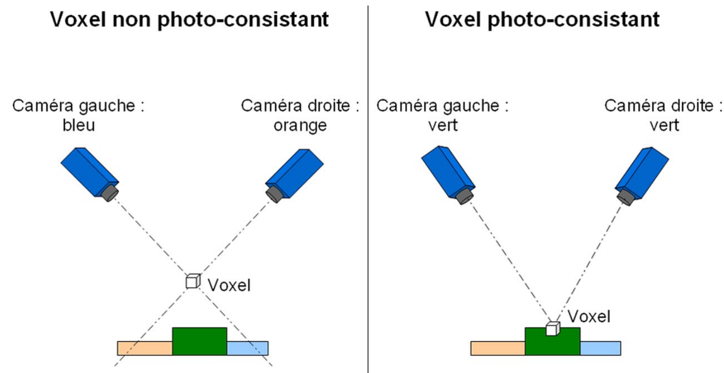


FIGURE 1.23 – Principe de la reconstruction 3D par photo-consistance

Connaissant les paramètres extrinsèques et intrinsèques de chaque caméra placée dans la scène, la projection de chaque voxel sur les images est ensuite calculée.

Un test de photoconsistance est ensuite appliqué pour toutes les projections de chaque voxel. Il consiste à vérifier que toutes les projections du voxel considéré sont bien « de la même couleur », indiquant ainsi que le voxel considéré appartient à la surface d'un objet. Une illustration de ce principe est donné par la figure 1.23.

Dans cette méthode, l'appariement n'est donc pas explicitement cherché sur l'image comme dans les méthodes présentées dans la section 1.4.1, mais effectué implicitement par la projection exhaustive de l'ensemble des voxels composant la scène.

Un élément clé du calcul est la « visibilité » du voxel par chaque caméra. Un voxel V_{ijk} est visible pour une caméra C_n s'il n'y a que des voxels transparents sur le segment $\overline{C_n V_{ijk}}$. Idéalement, seuls les voxels visibles doivent être reconstruits ; cependant le modèle 3D de la scène n'étant pas parfaitement connu, l'estimation de la visibilité est imparfaite.

L'article cité introduit plusieurs notions fondamentales propres aux reconstructions volumétriques, notamment l'«Ordinal Visibility Constraint » qui permet de simplifier le calcul de la visibilité d'un voxel en plaçant les caméras de manière à ce que le volume convexe entourant celles-ci ne contienne aucun point de la scène. Si cette contrainte est respectée, il est alors possible de définir un parcours ordonné de l'ensemble des voxels assurant que tous les voxels pouvant cacher un voxel donné sont évalués avant celui-ci ; par exemple en parcourant la grille de voxels par plan successifs en s'éloignant des caméras. Pour chaque voxel considéré photoconsistant (et donc opaque), un masque est appliqué sur l'ensemble des projections de ce voxel sur les images. Ces pixels ne pourront être utilisés par la suite pour calculer la photoconsistance d'un voxel, qui serait en fait masqué par le premier.

Cette méthode présente cependant quelques inconvénients. Tout d'abord, en présence d'objets possédant une texture périodique ou bien une couleur uniforme, le test de photoconsistance seul ne peut déterminer à coup sûr l'état d'un voxel. On utilise alors la silhouette des objets, calculée en éliminant l'arrière-plan sur chaque image d'entrée, afin de limiter la reconstruction au volume réellement occupé par l'objet dans la scène. Cette technique élimine une bonne partie des erreurs de reconstruction, mais la difficulté dans une application réelle est d'effectuer cette segmentation arrière-plan / avant-plan de manière automatique et non supervisée.

Dans le cas d'une application robotique autonome, la nécessité de posséder toutes les images de la scène avant d'effectuer la reconstruction est aussi une limitation importante. Il est alors nécessaire d'effectuer l'acquisition des multiples points de vues avant d'être en mesure de calculer le modèle 3D. Dans ce cas précis, l'acquisition des images n'étant pas synchrones puisque le robot doit se déplacer entre chaque prise de vues, les modifications dynamiques de la scène (déplacements d'objets mobiles, modification de l'éclairage,...) peuvent venir perturber le test de photoconsistance qui repose sur l'hypothèse d'une illumination lambertienne.

1.4.2.2 Generalized Voxel Coloring

La méthode *Generalized Voxel Coloring*, présentée dans [CMS00], propose de modéliser exactement la visibilité d'un voxel par chaque caméra. La scène initiale est considérée « pleine » (tous les voxels sont marqués opaques) ; puis pour chaque pixel de chaque image, on recherche le voxel opaque le plus proche de la caméra. Pour cela, deux valeurs sont mémorisées par pixel : l'indice du voxel s'y projetant et sa distance à la caméra. Ces valeurs sont mises à jour lorsque un voxel plus proche est trouvé. Ces voxels sont appelés « voxels de surface ». Chaque voxel de surface est ensuite testé pour la photoconsistance et s'il n'est pas consistant, il est marqué transparent et on met à jour les informations pour le pixel considéré (indice du voxel et distance). Une fois la visibilité de tous les voxels déterminée exactement, la couleur des voxels est calculée.

Cette méthode, proche du *Voxel Coloring* présenté précédemment, améliore cette dernière en permettant la prise en compte de caméras placées arbitrairement dans la scène, sans nécessité de respecter la contrainte de positionnement des caméras évoquée précédemment.

1.4.2.3 Mesure de photoconsistance

La fonction de mesure de la photoconsistance entre deux échantillons (pixels) est au coeur de la méthode. Seitz [SD97] utilise l'écart type de l'ensemble des couleurs des projections d'un voxel sur l'ensemble des images. Seules les images où le voxel est visible sont prises en compte par l'utilisation d'un masque, mis à jour à pour chaque voxel marqué opaque.

Ozun [OYA05] étudie et compare différentes mesures de la photoconsistance. La conclusion met en avant le rôle primordial du seuil de photoconsistance dans la qualité du modèle obtenu, sans réellement privilégier une méthode. Il met notamment en avant l'intérêt principal de l'utilisation de la distance de Minkowsky (norme L_1) comme fonction de photoconsistance : si une seule paire d'échantillons est inconsistant, le voxel ne peut pas être consistant et il n'y a donc pas lieu de tester toutes les paires d'échantillons.

Hornung [HK06] propose une méthode de suréchantillonage des voxels afin de rendre plus robuste l'estimation de la photoconsistance. Le résultat très intéressant nécessite cependant un grand nombre d'échantillons de couleur (et donc d'images sources) pour produire une estimation robuste.

1.4.2.4 Approches temps réel

L'utilisation de méthodes de reconstruction volumétrique en temps réel est un sujet bien documenté.

Prock [PD98] et Cheung [CKBH00] utilisent le voxel coloring pour reconstruire la silhouette d'un personnage en mouvement dans un volume entouré de caméras. La cadence vidéo peut être atteinte par ces deux systèmes, au détriment néanmoins de la résolution spatiale (inférieure à $64 \times 64 \times 64\,voxels$). [CKBH00] nécessite aussi un PC pour chacune des 5 caméras, chargé de calculer la silhouette de l'objet à reconstruire, plus un PC effectuant la reconstruction proprement dite (6 PC au total).

1.4.2.5 Conclusion sur les méthodes volumétriques

Ces méthodes de reconstruction 3D présentent l'avantage de créer un modèle dense de la scène : l'état de chaque voxel visible est calculé et non pas seulement quelques points d'intérêt épars. Le modèle obtenu est donc potentiellement plus riche que les modèles obtenus par d'autres méthodes.

Alors que les méthodes fonctionnant par mise en correspondance de points d'intérêt nécessitent que le champ de vue commun aux deux caméras soit aussi grand que possible, ce qui limite en pratique le nombre de caméra et de points de vues utilisables, les méthodes volumétriques permettent de prendre facilement en compte un nombre arbitraire de caméras et de points de vues.

1.4.3 Synthèse

Les méthodes de reconstruction 3D par vision reposent sur un principe commun : mettre en correspondance des points sur les différentes vues de la scène de manière à pouvoir en déduire la position dans la scène du point 3D correspondant.

Cette mise en correspondance peut être effectuée selon deux modalités principales :
- un point d'intérêt est cherché sur une image, ses correspondants sont recherchés sur les autres points de vues de la scène et le point 3D en est déduit (méthode $2D \rightarrow 3D$)
- un point 3D de la scène est projeté sur tous les points de vues de cette scène, et ces projections sont testées pour déterminer si elles sont consistantes (méthode $3D \rightarrow 2D$)

Tandis que la première de ces méthodes ne permet que l'appariement d'un faible nombre de points ; la seconde permet de calculer un modèle 3D régulièrement échantillonné, donnant ainsi une reconstruction dense de l'environnement.

1.5 Conclusion

Dans ce chapitre, nous avons présenté le canevas général d'une méthode de reconstruction 3D avant d'étudier plus en détail ses différents constituants, en particulier les différentes modalités d'acquisition ; les modèles 3D et leur stockage en mémoire ainsi que les algorithmes classiques de reconstruction 3D par vision.

Dans le cadre de notre application, nous avons sélectionné un système d'acquisition stéréoscopique composé d'une paire de caméras catadioptriques. Cette configuration permet tout d'abord d'obtenir des images panoramiques synchronisées représentant l'ensemble de la scène environnant le capteur, tout en permettant une acquisition rapide à cadence vidéo. Ce système est de plus relativement compact et de construction complètement statique, sans éléments mobiles ce qui le rend facilement embarquable.

Concernant le modèle 3D et son stockage en mémoire, une étude plus approfondie est donnée dans la section 3.5.2. La solution retenue, composée d'un modèle volumétrique discret stocké dans un tableau régulier permet un accès rapide et concourant, particulièrement adapté à l'architecture de traitement sélectionnée dans le chapitre 2.

La méthode de reconstruction 3D proposée, basée sur les méthodes volumétriques décrites ici, fait l'objet d'une étude en détail dans le chapitre 3.

Chapitre 2

Architectures de traitement des données pour la reconstruction 3D

Sommaire

2.1 Vocabulaire 30

2.2 Historique rapide de l'évolution de l'informatique et des ordinateurs 31

2.3 Architecture des processeurs 34

2.3.1 Organisation de la mémoire 34

2.3.2 Partage de la mémoire 35

2.3.3 Jeux d'instructions 36

2.3.4 Taxinomie de Flynn 36

2.3.5 Optimisations 37

2.4 Caractéristiques des processeurs 38

2.5 Estimation des performances 38

2.6 État de l'art des processeurs 39

2.6.1 Processeurs électroniques 40

2.6.2 Processeurs non conventionnels 49

2.7 Synthèse 52

2.7.1 Processeurs non conventionnels 52

2.7.2 Microprocesseurs 52

2.7.3 DSP et FPGA 52

2.7.4 Architectures novatrices 52

2.7.5 CPU et GPU 53

2.7.6 Solution retenue 53

2.8 Présentation détaillée de l'architecture CUDA 53

2.8.1 Organisation des threads 54

2.8.2 Organisation de la mémoire 54

2.8.3 Vue d'ensemble de l'organisation des calculs 57

2.8.4 Évaluation des performances de CUDA 57

2.9 Conclusion 59

Avec ces travaux, notre objectif est de définir un système de reconstruction 3D capable d'atteindre une cadence temps réel, il est donc primordial que l'architecture du processeur sélectionnée et l'algorithme s'exécutant sur ce processeur soient adaptés l'un à l'autre pour obtenir des performances optimales.

Après la présentation des méthodes d'acquisition et de reconstruction 3D dans le chapitre 1, ce chapitre est dédié à l'étude des différentes familles de processeurs actuellement disponibles sur le marché. Cette analyse a pour but de mettre en avant les caractéristiques propres à chaque famille de systèmes et qui leur procurent un avantage dans le domaine d'application visé : la reconstruction 3D par vision.

Ce chapitre commence dans la section 2.1 par préciser le vocabulaire spécifique utilisé par la suite. La liste des symboles située en fin de volume pourra servir d'aide-mémoire lors de la lecture de ce chapitre, les acronymes étant nombreux.

La section 2.2 présente un bref historique de l'évolution des processeurs depuis les débuts de l'informatique jusqu'aux composants intégrés modernes, ainsi que les grandes tendances actuelles qui permettent de mieux appréhender l'enjeu de ces travaux de thèse.

Les notions de base de l'architecture des microprocesseurs sont présentées dans la section 2.3 ; la section 2.4 présente les critères permettant de caractériser et évaluer les différents processeurs présentés par la suite, et la section 2.5 s'attardera sur les méthodes d'estimation et de comparaison des performances.

Les processeurs seront détaillés dans la section 2.6.1, regroupés par familles. Des architectures moins classiques seront détaillées dans la section 2.6.2.

Une synthèse permettant une vue d'ensemble des architectures de traitement est présentée section 2.7. Dans la section 2.8, l'architecture retenue est présentée en détail. Une comparaison des performances entre une architecture x86 classique et cette architecture est réalisée sur un des points clés de notre algorithme de reconstruction.

La section 2.9 conclura ce chapitre.

2.1 Vocabulaire

Avant de rentrer dans le vif du sujet et d'étudier les différents processeurs, nous allons préciser le vocabulaire utilisé par la suite. Les définitions communément admises pour les termes listés ci-dessous seront accompagnées de précisions concernant leur utilisation dans ce mémoire.

Ordinateur Un ordinateur est une machine capable de traiter des données en suivant un programme constitué d'une séquence d'opérations. Il comporte des interfaces d'entrées sorties (réseau de communication, interface homme-machine...), une mémoire de stockage et un processeur qui est responsable de l'exécution du programme. Dans ce mémoire, le terme ordinateur sera utilisé pour les considérations générales. Le terme PC (Personal Computer - Ordinateur Personnel) désignera plus spécifiquement un ordinateur de petite taille appartenant à la famille très largement diffusée des systèmes basés sur un processeur à architecture x86 ou équivalent. Un ordinateur est principalement constitué des éléments suivants :

Mémoire Vive c'est la mémoire de travail permettant de stocker les données en cours de traitement. La technologie la plus répandue est la RAM (Random

Access Memory), avec des tailles de quelques Kio à plusieurs Gio par processeur. Elle est en général volatile (son contenu disparaît lorsque le système est hors tension) ce qui oblige à utiliser un autre moyen de stockage à long terme pour les données à conserver.

Mémoire De Masse c'est la mémoire permettant de stocker de grandes quantités d'informations de façon pérenne. Non volatile, sa capacité peut atteindre plusieurs téraoctet. Les systèmes embarqués utilisent en général une mémoire Flash, les PC, serveurs et grands systèmes utilisent en général des disques durs.

Processeur C'est le composant principal de l'unité centrale d'un ordinateur. Il est chargé d'interpréter et d'exécuter les séquences d'instructions composant les programmes. Le terme processeur sera utilisé pour traiter des généralités et désignera ainsi tout système capable d'exécuter un ensemble d'instructions préétabli sur des données codées sous forme d'une grandeur ou d'un objet physique. Le terme microprocesseur (abrégé µP) désignera plus spécifiquement un processeur électronique de très petite dimensions constitué d'un seul circuit intégré, utilisant un codage binaire pour représenter l'information. Le terme CPU sera utilisé spécifiquement pour désigner un µP équipant un PC.

Interfaces d'entrées sorties Par exemple les interfaces réseaux, USB, SATA, interfaces homme-machine... Elles effectuent la communication avec les périphériques ce qui permet principalement d'insérer des données externes dans l'ordinateur afin de les traiter, puis d'extraire les résultats en vue de l'affichage, de la transmission à un autre système ou de la sauvegarde pour une utilisation future.

Architecture Externe (ou Architecture). Ce terme désigne la spécification fonctionnelle d'un processeur du point de vue du programmeur. Elle est constituée notamment du jeux d'instructions, des registres utilisables par le programmeur, ainsi que d'une organisation des différents niveaux de mémoire et des entrées sorties.

Architecture Interne (ou Micro-Architecture). Ce terme désigne l'implémentation de l'architecture externe dans le processeur. Il peut comporter par exemple des registres internes non accessibles par le programmeur. La microarchitecture peut être reprogrammable ce qui permet au processeur d'émuler le fonctionnement d'une autre architecture.

Outre ces définitions générales, les mots-clés et autres acronymes seront définis au fur et à mesure de leur utilisation dans le texte. La majorités de ces termes et leur définition sont repris dans la liste des symboles présente en fin de volume.

2.2 Historique rapide de l'évolution de l'informatique et des ordinateurs

L'informatique, science du traitement de l'information a vu le jour avec l'algorithme d'Euclide (-325 à -265) pour le calcul du PGCD de deux nombres qui est considéré comme étant le premier algorithme créé dans l'histoire.

La forme moderne de cette science, exploitant des machines pour réaliser ces traitements de manière automatique, date du 17e siècle. Ces systèmes sont alors mécaniques et ne permettent que des opérations simples, comme par exemple la Pascaline en 1642.

Cependant, l'informatique ne s'est réellement développée qu'avec la mise au point d'ordinateurs entièrement électroniques, comme l'ENIAC en 1947. Ce premier ordinateur électrique, monstrueuse machines de 267 tonnes occupant 63 mètres carrés de surface, nécessitait des jours de travail pour la programmation, alors réalisée à l'aide de câbles et de commutateurs manuels. Elle a notamment été utilisée pour effectuer les calculs nécessaires la mise au point de la bombe H. Les machines de cette génération étaient peu fiables, très gourmandes en énergie électrique et encore très limitées mais permirent néanmoins le développement et la théorisation de l'informatique. Les premiers ordinateurs à transistors sont apparus dans les années 1950 et marquent une avancée notable par rapport aux technologies précédentes : relais électromécaniques et lampes électroniques.

La réelle démocratisation des micro-ordinateurs est très récente : depuis les années 1980. Elle est liée à une invention majeure, le microprocesseur, qui a permis la réalisation d'ordinateurs compacts, puissants et surtout économiques. Largement diffusés, les PC ont bénéficié d'une compétition acharnée entre les différents acteurs industriels faisant régulièrement chuter les prix et augmenter les performances. Depuis le début des années 1990, le développement de jeux vidéos nécessitant des capacités graphiques évoluées, ainsi que la très large diffusion d'Internet a encore poussé en avant l'évolution de ces systèmes.

Les premiers microprocesseurs (Intel 4004 en 1970-71) exploitaient une gravure grossière (10 µm), leur intégration était donc limitée de même que leurs performances (horloge à 108 KHz). Avec l'évolution des procédés industriels de gravure sur silicium, il devint possible d'augmenter le niveau d'intégration, c'est à dire le nombre de composants élémentaires (transistors notamment) par unité de surface dans un circuit intégré. La miniaturisation permet de construire des circuits de plus en plus sophistiqués, mais aussi d'augmenter la fréquence en diminuant la taille des pistes (et donc le temps de propagation de l'électricité d'un endroit à l'autre de la puce). Cette rapide évolution est illustrée par la figure 2.1 sur laquelle nous remarquerons l'échelle verticale logarithmique.

Les années 1990-2005 virent une augmentation très rapide des fréquences de fonctionnement des processeurs, poussée par le développement des technologies de réalisation des circuits intégrés et attisée par la compétition féroce des fabricants.

Cette « course à la fréquence », bien souvent réalisée au détriment de l'efficacité énergétique, fut ainsi le principal moteur de l'évolution des performances durant cette période. Cependant, augmenter la fréquence de fonctionnement devint de plus en plus difficile à réaliser, notamment à cause de la relation entre consommation électrique, fréquence et tension d'alimentation : $P = K \times F \times V^2$ où P représente la puissance consommée (en Watts) et intégralement dissipée, F la fréquence de fonctionnement (Hz) et V la tension d'alimentation (Volts). K est un coefficient dépendant de l'architecture du processeur et de sa technologie.

Cette relation indique donc que, pour une architecture de processeur donnée, la consommation électrique croît de manière linéaire avec la fréquence, et avec elle la dissipation thermique du composant. Pour limiter cette augmentation de la consommation deux principales stratégies sont mises en oeuvre : limiter la tension d'alimentation (baisse de V) ou optimiser l'architecture et la technologie de réalisation du silicium (baisse de K).

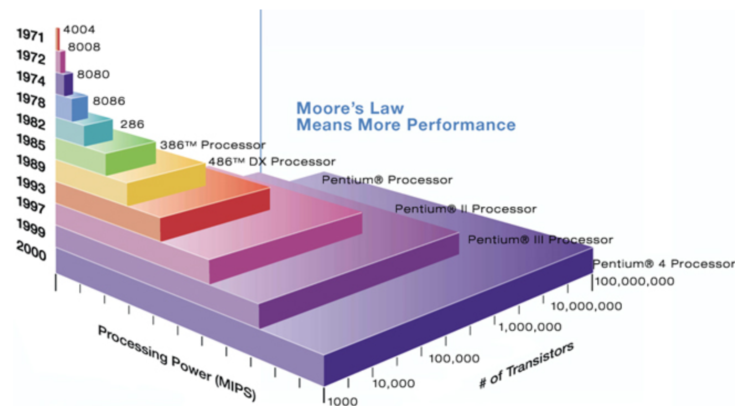


FIGURE 2.1 – Évolution de la puissance (MIPS) et de la complexité (nombre de transistors) des microprocesseurs de 1971 à 2000

Source : http://zeprince.ca

La course à la fréquence ralentit fortement avec l'arrivée des processeurs fonctionnant à environ 4 GHz, comme l'illustre la figure 2.2. Il devint très difficile de progresser au-delà par les techniques conventionnelles. Les processeurs les plus rapides actuellement sont sans doute les IBM POWER6 à 5.0 GHz ce qui semble pour le moment une limite difficilement franchissable, les futures évolutions de la famille POWER sont en effet annoncées à des fréquences inférieures (POWER7 : 4.4 GHz) principalement pour des raison d'efficacité énergétique. Cette réduction de la fréquence de fonctionnement n'est cependant pas synonyme de baisse de performances, un travail d'optimisation ayant été réalisé sur l'architecture.

En parallèle de l'évolution soutenue des fréquences, d'autres techniques permettant d'augmenter la puissance de calcul ont été mises au point. Nous pouvons ainsi citer : le cache prédictif dont le but est d'optimiser les accès à la mémoire RAM en anticipant la lecture des données, les unités arithmétiques superscalaires capable d'effectuer plusieurs traitements numériques en parallèle, l'exécution en « pipeline » des instructions pour augmenter le débit, le réordonnancement des instructions à la volée (Out of Order execution), les extensions vectorielles (MMX, SSE...), l'hyperthreading . . .

A l'heure actuelle, un changement profond est en cours dans l'industrie microélectronique. Pour les raisons physiques évoquées précédemment, il est devenu prohibitif d'augmenter la fréquence de fonctionnement des processeurs. Des travaux poussés sur l'architecture permettent d'en augmenter l'efficacité, mais les gains sont marginaux.

Cependant, la miniaturisation n'a pas atteint de limite infranchissable et poursuit donc son chemin en permettant d'intégrer de plus en plus de composants sur la même surface de silicium.

C'est ainsi qu'ont été développés les processeurs « multi-coeurs », cohabitation de plusieurs processeurs distincts sur la même puce. Ces processeurs, en permettant l'exécution en parallèle de plusieurs tâches, permettent ainsi d'obtenir un système plus réactif et plus puissant. Il est néanmoins nécessaire, pour en tirer partie, d'exécuter plusieurs tâches en même temps.

Les premières versions de ces processeurs possèdent 2, 3 ou 4 coeurs, les versions à 8,

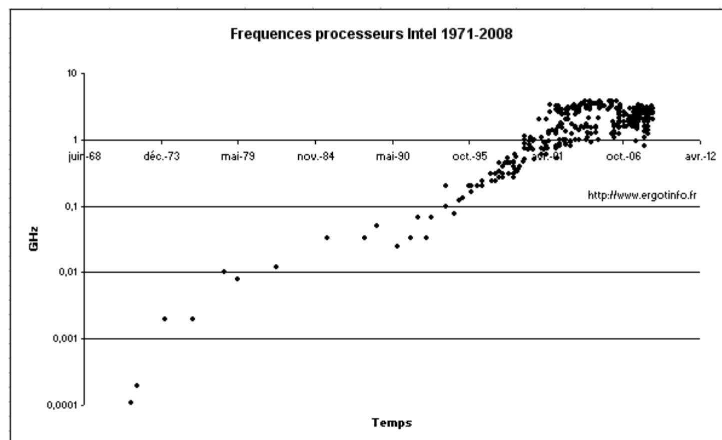


FIGURE 2.2 – Évolution de la fréquence des microprocesseurs Intel ; le seuil des 4 GHz semble pour l'instant difficile à franchir

Source : http://www.ergotinfo.fr

12 et 16 coeurs arrivent actuellement sur le marché. L'un des inconvénients principaux de ces processeurs résident dans le partage de la mémoire entre les différents coeurs. Comme il n'existe une seule mémoire centrale, la bande passante de celle-ci est partagée par l'ensemble des coeurs. Plus il y a de coeurs dans un processeur, plus il y a de chance que le bus mémoire soit saturé et entraîne des dégradation de performances.

Outre le marché des ordinateurs personnels, cette tendance au parallélisme semble se généraliser à toute l'industrie informatique, y compris dans le domaine des systèmes embarqués, où les gains en performances se traduisent par des gains en consommation électrique et donc en autonomie pour les systèmes alimentés par batteries.

L'évolution actuelle semble donc s'orienter vers des processeurs de plus en plus parallèles. De nombreux développements annoncés par les fabricants semblent d'ailleurs confirmer cette tendance. Par exemple, *larrabee* de Intel vise à développer un processeur possédant des centaines de coeurs ou le projet AMD *fusion* visant à réunir un processeur classique et un processeur graphique sur une même puce.

2.3 Architecture des processeurs

L'architecture des processeurs est un domaine complexe, en évolution rapide. Dans cette section, nous introduisons les éléments les plus courant de l'architecture des microprocesseurs électroniques afin de permettre une étude plus détaillée dans la section 2.6.1 présentant les différentes familles de ces processeurs.

2.3.1 Organisation de la mémoire

L'organisation de la mémoire d'un processeur peut être qualifiée suivant deux modèles défini ci-dessous.

von Neumann ou modèle à mémoire unifié. Un processeur de ce type comporte un bus

(adresses/données) unique pour accéder à la mémoire RAM qui contient à la fois les données et le programme. Cette structure permet notamment de diminuer la surface de silicium dédiée à la gestion des bus mémoires, mais aussi au programme de se modifier lui-même en cours de fonctionnement. Cette dernière possibilité introduit néanmoins des problèmes de sécurité et de fiabilité de fonctionnement. Ce modèle est utilisé par les processeurs pour PC et une grande majorité des processeurs à usage générique.

Harvard ou modèle à mémoires séparées. Deux mémoires distinctes (et en général physiquement séparées) stockent l'une le programme et l'autre les données. Cette configuration est plus coûteuse en surface de silicium que l'architecture de von Neumann, mais elle permet d'optimiser la bande passante et le type des deux mémoires ; par exemple en utilisant une mémoire morte pour le stockage du programme ce qui évite la dégradation involontaire du programme (bug, erreur d'écriture ou d'adresse, virus...). Enfin, l'utilisation de deux bus distincts permet d'adapter leur largeur à l'application (par exemple, bus 16 bits pour le programme et 32 bits pour les données). Beaucoup de DSP actuels sont sur ce modèle, de même que certains microcontrôleurs 8 bits à mémoire Flash (Atmel AVR).

2.3.2 Partage de la mémoire

Sur les systèmes possédant plusieurs processeurs destinés à travailler en parallèle, il existe plusieurs modèles de partage de la mémoire globale entre les différentes unités de calcul, selon l'organisation et la proximité des processeurs.

SMP (Symmetric Multi-Processing) Désigne un système où l'ensemble des processeurs accèdent de manière directe à un même espace de mémoire. Cette architecture simplifie la cohérence des données car la même mémoire est utilisée par tous les processeurs. Cette structure est cependant fortement pénalisée par le partage du bus mémoire ce qui limite les performances. De plus, elle n'est applicable que pour des systèmes où les processeurs sont physiquement proches les uns des autres. Les CPU multi coeurs actuels (Intel Core 2, AMD Phenom ...) sont basés sur cette organisation.

NUMA (Non-Uniform Memory Access) Désigne un système où plusieurs zones de l'espace de mémoire global sont accessibles avec des temps et des modalités d'accès différentes, et donc un espace d'adressage différent. On peut comme exemple citer un système composé de plusieurs PC connectés entre eux par un réseau. Chaque machine est composée d'un processeur ayant accès à un banc de mémoire. Le temps d'accès d'un processeur à sa propre mémoire RAM est court, mais l'accès à celle localisée sur un noeud distant est plus long car transitant par le réseau. La cohérence des données entre plusieurs noeuds est un des problèmes de ce modèle. Un autre problème majeur est la grande latence qui peut être nécessaire pour accéder aux données distantes. Ce modèle est cependant celui des plus grands systèmes informatiques actuels (supercalculateurs RoadRunner, grille de calcul distribuée Boinc, ...) car il permet de faire travailler ensemble des systèmes physiquement très éloignés. C'est aussi ce modèle qui passe le mieux à l'échelle.

2.3.3 Jeux d'instructions

Chaque modèle de processeur possède un jeu d'instructions unique, et souvent incompatible avec celui des autres processeurs. Il est même courant au sein d'une même famille de processeurs d'avoir de petites différences dans le jeu d'instructions, comme par exemple quelques instructions supplémentaires sur les modèles haut de gamme. Ces jeux d'instructions peuvent être classés selon leurs caractéristiques générales, comme présenté ci-dessous.

RISC (Reduced Instruction Set Computer) Désigne les processeurs supportant un nombre réduit d'instructions en langage assembleur. Cette limitation du nombre et de la complexité des instructions permet une optimisation de la réalisation physique du composant, ce qui diminue sa surface et donc son coût ; mais elle permet aussi une optimisation poussée du flot d'exécution des instructions (pipeline multi-étages) qui se traduit généralement par une meilleure performance. La gestion des opérations complexes est laissée au compilateur qui doit les décomposer en opérations plus simples.

CISC (Complex Instruction Set Computer) Désigne les processeurs supportant un jeu d'instructions étendus. Des instructions sophistiquées sont implémentées dans l'architecture, ce qui permet une diminution de la complexité du programme. Le nombre de commandes utilisables étant plus grand que sur les architectures RISC, il est moins aisé de le programmer directement en langage assembleur. Sur les machines modernes, même si l'architecture externe est de type CISC, l'architecture interne est souvent RISC et les instructions supplémentaires sont émulées par micro-programmation, ce qui permet de combiner les avantages des deux architectures.

VLIW (Very Long Instruction Word) Désigne une architecture à mot d'instruction très large (128, 256 bits ou plus), combine plusieurs instructions et plusieurs opérandes afin de tirer parti du parallélisme intrinsèque (comme par exemple les opérations mathématiques sur les vecteurs, les accès mémoires, ...).

Scalaire Désigne une architecture effectuant une opération sur une seule donnée à la fois. Opposé à Vectoriel.

Vectoriel Désigne une architecture capable d'effectuer une opération donnée sur un vecteur contenant plusieurs opérandes. Dans certaines architectures scalaires, une unité spécialisée a été ajoutée et des instructions vectorielles complémentaires sont créées pour en permettre l'utilisation, par exemple le MMX sur Intel Pentium.

2.3.4 Taxinomie de Flynn

La taxinomie de Flynn [Fly72] classe les architectures des processeurs suivant la nature du parallélisme exploitable. Flynn définit ainsi quatre types d'architectures de traitement de l'information suivant l'organisation des flux de données et d'instructions.

SISD Single Instruction, Single Data. Cela correspond à l'architecture classique de von Neumann. Un seul flot d'instructions opère de façon purement séquentielle sur un seul flot de données. Les premiers processeurs de PC sont de cette catégorie.

SIMD Single Instruction, Multiple Data. Le même flot d'instructions est exécuté sur plusieurs flux de données en même temps. Les processeurs vectoriels sont de cette catégorie, de même que les GPU.

MISD Multiple Instructions, Single Data. Plusieurs flots d'instructions opèrent en parallèle sur le même flot de données. Peu courant, les architectures de ce type sont en général conçues pour améliorer la sûreté de fonctionnement via le traitement indépendant et parallèle des données par plusieurs calculateurs distincts exécutant des séquences d'instruction différentes sensées aboutir au même résultat.

MIMD Multiple Instructions, Multiple Data. Plusieurs flots d'instructions différentes opèrent en parallèle sur plusieurs flots de données distincts. On peut distinguer deux sous catégories :

SPMD Single Process, Multiple Data. Plusieurs processeurs indépendants exécutent le même programme de façon indépendante mais non synchronisée (à l'opposé du SIMD), sur plusieurs flots de données.

MPMD Multiple Processes, Multiple Data. Au minimum deux programmes différents opèrent. L'exemple typique consiste à avoir un processeur maître exécutant un programme de gestion qui distribue les tâches aux autres processeurs exécutant un second programme qui traite les données. Les résultats sont par la suite centralisés sur le processeur maître.

2.3.5 Optimisations

Afin d'améliorer les performances des microprocesseurs, de nombreuses techniques d'optimisation ont été développées.

Super-scalaire Se dit d'un processeur possédant plusieurs unités arithmétiques et logiques capables de fonctionner en parallèle, par analogie avec un processeur scalaire exécutant une seule instruction à la fois.

Pipeline Les sous étapes (chargement, décodage, exécution...) de plusieurs instructions du programme sont exécutées en parallèle, ce qui augmente le débit du processeur, au détriment de la latence.

Out-Of-Order (OoO) Se dit d'un processeur capable de modifier l'ordre d'exécution des instructions du programme pour augmenter les performances, notamment en optimisant les accès mémoires (les instructions dont les données sont déjà connues sont exécutées avant les instructions nécessitant une lecture en mémoire).

Antémémoire (mémoire cache) L'accès à la RAM étant relativement lent par rapport à la vitesse de fonctionnement du processeur (400 MHz contre 4 GHz), les données souvent utilisées sont copiées dans une mémoire intermédiaire, très rapide, de petite taille et localisée très près du processeur. Ce cache peut être utilisé aussi bien en

lecture qu'en écriture afin d'accélérer les accès mémoires. Des circuits spécialisés (cache prédictif) essayent de deviner les prochaines adresses RAM accédées par le programme pour précharger ces données en cache lorsque le bus mémoire est sous-utilisé. La synchronisation entre la mémoire cache et la RAM est un problème complexe lorsqu'une machine est composée de plusieurs processeurs.

La plupart des processeurs actuels combinent plusieurs de ces technologies pour accroître la vitesse de traitement. Les processeurs pour PC, serveurs et stations de travail les utilisent tous intensivement.

2.4 Caractéristiques des processeurs

La classifications de processeurs aussi disparates que ceux présentés par la suite pose de nombreux problèmes. Il est en effet difficile de comparer ces différents systèmes sur une base commune, chacun ayant ses points forts et son application cible. Nous proposons ici une classification empirique, dans le but essentiellement d'estimer l'adéquation d'un système avec notre application de reconstruction 3D.

Lors de la présentation détaillée des familles de processeurs effectuées dans la suite de ce document, nous essayerons de les caractériser suivant les critères définis ci-dessous.

— Caractéristiques chiffrés

Puissance de calcul ordre de grandeur de la vitesse de traitement, exprimé en FLOPS ou en MIPS.

Largeur des données usuellement 8, 16, 32 ou 64 bits. Indique la taille des données directement manipulables par les circuits internes du processeur.

Bande passante RAM Indique la vitesse théorique maximale de transfert des données.

Consommation électrique besoins énergétiques pour le fonctionnement normal.

Encombrement taille physique du système.

Coût d'un système complet.

— Caractéristiques subjectives

Flexibilité facilité de programmation pour une tâche quelconque.

Disponibilité facilité à se procurer un système sur le marché.

Ces critères sont difficiles à évaluer de façon précise pour une famille de processeurs. Des ordres de grandeur seront données pour permettre une comparaison relative.

2.5 Estimation des performances

Mesurer la performance d'un processeur n'est pas trivial car celles-ci dépendent fortement de la tâche demandée. Chaque architecture possède une application de prédilection pour laquelle elle a été conçue, mais peut être relativement inefficace sur d'autres tâches.

La mesure la plus simple de la vitesse de traitement d'un processeur est le nombre d'instructions qu'il est capable d'accomplir en un temps donné. Généralement exprimée en MIPS (Million d'Instructions Par Seconde), elle permet de comparer efficacement deux processeurs ayant la même architecture interne. Cependant elle ne permet pas de comparer les architectures différentes.

Une autre mesure fréquemment utilisée est le nombre d'opérations de calcul en virgule flottante par seconde, ou FLOPS (Floating Point Operations Per Second). Ces opérations sont celles qui portent sur les chiffres réels stockés sur 32 bits (simple précision) ou 64 bits (double précision). Elle donne un ordre de grandeur de la puissance de calcul théorique d'un processeur mais est souvent optimiste.

Il existe un ensemble de tests synthétiques, créés pour comparer la performance de différents processeurs sur une tâche statistiquement représentative de programmes réels. Nous pouvons ainsi citer le test Whetstone [CW76] (calculs en virgule flottante), Dhrystone [Wei89] (calcul entier), CoreMark (utilise des algorithmes réels) et Linpack (classement Top500 des super-calculateurs). Ces tests ne fonctionnent cependant pas sur toutes les architectures et ne sont pas forcément représentatifs d'une tâche particulière.

La mesure des performances d'une architecture est donc délicate et forcément biaisée par le logiciel de test utilisé. Une solution évidente serait d'écrire l'application finale et de tester sa vitesse d'exécution sur différentes architecture. Bien que cette approche soit la seule qui puisse garantir qu'une architecture est vraiment plus rapide qu'une autre pour un travail donné, elle se heurte à plusieurs problèmes qui en limitent l'efficacité. Tout d'abord, très peu d'architectures différentes sont compatibles entre elles au niveau binaire, il est donc nécessaire de recompiler le programme de test pour chaque architecture ciblée. Dans la majorité des cas cependant cette comparaison serait purement artificielle, car une simple recompilation ne permet pas d'adapter le programme aux spécificités de l'architecture pour en tirer le meilleur parti. Pour comparer correctement chaque architecture, il est donc nécessaire de *réimplémenter* l'algorithme pour l'adapter à l'architecture. Cette nouvelle implémentation utilisera les fonctions spécifiques du processeur permettant d'atteindre les meilleurs performances. Un autre problème survient alors, car la comparaison de deux implémentations différentes s'exécutant sur deux machines différentes introduit un biais non négligeable. Le développeur pourra avoir un peu mieux optimisé l'une ou l'autre des implémentations, menant à un avantage lors de l'exécution.

Comparer la performance relative de deux architectures est donc relativement complexe, et les petites différences de performances ne sont pas significative la plupart du temps car une optimisation mineure peut faire basculer l'avantage.

2.6 État de l'art des processeurs

Après avoir présenté les différents constituants classiques d'un processeur dans les sections précédentes, nous allons ici présenter les différents processeurs existant et tenter de les classer et de les caractériser en fonction des critères présentés précédemment.

2.6.1 Processeurs électroniques

Les processeurs électroniques sont aujourd'hui les systèmes informatiques les plus performants, alliant de bonnes caractéristiques de puissances de calcul, un coût très faible et un encombrement minimal. La bonne maîtrise des différentes technologies nécessaires à leur réalisation (photolithogravure, CAO) et l'industrialisation possible a grande échelle explique le succès majeur de ces composants.

Dans cette section sont présentés les différentes familles de composants électroniques capable d'effectuer des calculs numériques. Un descriptif ainsi que les points clés de leur architecture interne est présentée.

2.6.1.1 Microprocesseurs

Il existe une variété énorme de microprocesseurs (abrégé µP), que l'on peut néanmoins classer par famille de largeur de bus : 4 bits, 8 bits, 16 bits, 32 bits, 64 bits et par jeu d'instruction : RISC ou CISC.

C'est le type de processeur le plus utilisé dans les PC, stations de travail et serveurs. Basé en général sur l'architecture de von Neumann, il est principalement constitué d'une unité de contrôle organisant l'exécution du programme, d'une unité arithmétique et logique (ALU) réalisant les opérations et d'une banque de registres, petites mémoires de travail accessibles très rapidement. Pour fonctionner, il est nécessaire d'adjoindre un ensemble de composants auxiliaires : un contrôleur de mémoire, de la mémoire, un circuit d'alimentation, un dispositif de réinitialisation (reset) et un générateur d'horloge. Des périphériques sont aussi indispensables pour permettre au µP d'accéder aux programmes qu'il doit exécuter : interface réseau, stockage de masse (disque dur, lecteurs de média amovibles...), périphériques d'interface utilisateur (clavier, écran, imprimante). Un µP est donc la partie centrale d'un système électronique complexe nécessaire à sa mise en oeuvre.

Courants dans les années 70 et 80, les µP 4, 8 et 16 bits sont aujourd'hui obsolètes comme composants isolés et sont maintenant intégrés dans des µC. Ils sont en général utilisés pour des applications qui requièrent peu de puissance de calcul, faible consommation électrique et compacité (systèmes embarqués divers, gestion d'essuie-glace de voiture, électroménager, petits périphériques informatiques, capteurs autonomes ...). Voir à ce sujet la section 2.6.1.2.

Les µP actuels peuvent se diviser en deux sous familles : d'une part les µP 32 et 64 bits utilisés dans les ordinateurs personnels (PC) et les serveurs ; d'autre part les µP optimisés pour une faible consommation électrique ou des caractéristiques spéciales (résistance aux radiations, aux hautes températures) et utilisés dans les systèmes embarqués nécessitant une grande puissance de calcul. Cette deuxième sous famille rejoint cependant peu à peu les microcontrôleurs en intégrant de plus en plus de périphériques.

Classification La plupart des µP actuels sont à architecture 32 ou 64 bits de type von Neumann. Ils sont la plupart du temps utilisés dans des machines SISD, bien que les systèmes MIMD se généralisent avec l'apparition des processeurs multicoeurs. L'accès mémoire est de type SMP sur un système multi-coeur et NUMA sur un système multiprocesseurs. Les CPU ayant une architecture compatible x86 ont un jeu d'instruction CISC,

il existe cependant toute une variété de processeurs CISC et RISC. La plupart des CPU actuels possèdent également des instructions vectorielles (MMX, SSE, AltiVec).

La famille de composant fabriqués par Transmetta (Crusoe) est particulière : elle possède une architecture interne VLIW, mais présente une architecture externe modifiable par programmation. Ceci permet au composant d'émuler l'architecture d'autres processeurs et notamment celle de la famille Intel x86.

Caractéristiques Les µP sont les plus flexibles et génériques des processeurs. Capable de traiter n'importe quel algorithme, on dit qu'ils sont turing-complets. Leur coût très faible et leur large diffusion en font aussi les composants les plus faciles à se procurer. Ils sont de plus très faciles à programmer et une grande variété de langages et d'outils de programmation existent.

Les processeurs de PC sont parmi les plus puissants des µP actuels, à l'exception des processeurs dédiés aux serveurs et aux grand systèmes (mainframe, HPC...). La puissance de calcul d'un Intel Corei7-965 Extreme Edition (actuellement le haut de gamme des processeurs pour PC, il comporte 4 coeurs) est de 69,23 GFlops (mesurée par le benchmark Sandra Sisoft). La bande passante RAM est de 18.4 Gio/s et la consommation électrique (système complet) de 368 W. En mai 2009, Fujitsu a annoncé un processeur à 128 GFlops contenant 8 coeurs.

2.6.1.2 Microcontrôleur

Un microcontrôleur (µC) est un circuit intégré comportant un µP ainsi que les périphériques indispensables à sa mise en oeuvre (alimentation, mémoire, reset, horloge) et des périphériques complémentaires (communication, convertisseurs analogiques...) dans le but d'obtenir un système complet dans un seul composant, facilitant ainsi la conception de systèmes.

La frontière entre microprocesseur et microcontrôleur est parfois difficile à définir. En effet, les microprocesseurs ont maintenant tendance à intégrer de plus en plus de périphériques pour augmenter les performances (par exemple, les processeurs Intel Core-i7 intègrent le contrôleur mémoire) ; tandis que les microcontrôleurs augmentent sans cesse leurs performances.

Les µC sont majoritairement utilisés dans les systèmes embarqués grâce à leur forte intégration qui diminue la complexité de conception, la consommation électrique et l'encombrement. Les capacités de calculs sont très similaires aux µP, mais ils sont en général moins rapides principalement à cause des contraintes de consommation.

Pour des applications de traitement d'image, il existe des µC comportant des périphériques spécialisés capable de réaliser de la compression/décompression vidéo ou encore des µC comportant un µP et un DSP complètement programmable pour les calculs numériques intensif (Texas Instrument OMAP). Ces systèmes hybrides, souvent appelés SoC (System On Chip) ont tendance à se généraliser dans certains domaines, comme la téléphonie mobile.

Classification Un µC étant basiquement un µP avec des périphériques intégrés, la classification et les caractéristiques sont similaires. Il existe néanmoins une bonne repré-

Table 2.1 – Caractéristiques des Micro-Processeurs

Organisation mémoire	von Neumann (majorité)
Partage mémoire	SMP (multicoeur), NUMA (multiprocesseurs)
Jeux d'instruction	CISC, Scalaire avec extensions Vectorielles (MMX...)
Taxinomie de Flynn	SISD (monoprocesseur), MIMD (multicoeur)
Optimisations	Super Scalaire, Pipeline (>20 étages), OoO, Cache très complexe
Puissance de calcul	10 à 100 GFlops
Largeur des données	32 ou 64 bits
Bande passante RAM	5 à 20 Gio/s
Consommation électrique	30 W (portable) à 200 W (station, serveur)
Encombrement	Faible (systèmes embarqués) à Moyen (PC, serveurs, mainframe)
Coût	Moyen (30 à 200 €)
Flexibilité	Excellente, (adaptable pour toutes les applications)
Disponibilité	Excellente (marché grand public / distributeurs spécialisés)

Table 2.2 – Caractéristiques des Micro-Contrôleurs

Organisation mémoire	von Neumann (haut de gamme), Harvard
Partage mémoire	SMP (multicoeur), NUMA (multiprocesseurs)
Jeux d'instruction	RISC, Scalaire
Taxinomie de Flynn	SISD (monoprocesseur), MIMD (multicoeur et SoC)
Optimisations	Cache, Pipeline
Puissance de calcul	1 à 100 MIPS
Largeur des données	8 à 32 bits
Bande passante RAM	1 à 1000 Mio/s
Consommation électrique	5 mW (8 bits) à 1 W
Encombrement	Très faible à faible (systèmes embarqués)
Coût	Très faible (1 à 10 €)
Flexibilité	Excellente (adaptable à toutes les applications)
Disponibilité	Très bonne (revendeurs spécialisés)

sentation de l'architecture Harvard, notamment dans les µC 8 bits (famille Atmel AVR, Microchip PIC), ce qui permet de placer le programme dans une mémoire Flash peu coûteuse car très dense et d'optimiser la taille de la RAM pour l'application.

Caractéristiques Les µC ont des caractéristiques très variées suivant leur cible d'application. Il existe des µC 4 bits très faibles consommation aux performances très faibles destinés à remplacer un ensemble de composants discrets (portes logiques, ADC) dans les applications simples. A l'opposé, les µC ARM 32 bits destinés aux applications de téléphonie mobile embarquent une grande quantité de périphériques et une puissance de calcul confortable. Certains embarquent même plusieurs coeurs.

Du point de vue des performances, ils restent cependant en retrait par rapport aux µP de PC présentés précédemment, tant que la consommation électrique n'entre pas en jeux.

2.6.1.3 *Digital signal processor*

Le *Digital signal processor* (DSP) est un type de microprocesseur dont l'architecture interne a été optimisée pour les calculs numériques. Adapté pour le traitement numérique du signal, comme le suggère son nom, il excelle dans les applications de filtrage et de calculs des FFT (Fast Fourier Transform).

Cependant, son architecture peu flexible le rend inapproprié pour l'exécution de pro-

grammes génériques comportant des instructions de contrôle (tests, branchements) et d'interruptions sur lesquels il est particulièrement inefficace. Ainsi, la plupart des DSP sont incapables de faire fonctionner un système d'exploitation.

Le coeur d'un DSP est essentiellement composé d'unités MAC (Multiply and ACcumulate) capables d'effectuer une multiplication et une addition en un seul cycle d'horloge. Souvent, plusieurs MAC sont présents dans un même DSP et sont capable d'opérer en parallèle pour des performances accrues. Bien que certains DSP soient capable d'opérer sur des nombres en virgule flottantes, la grande majorité ne fonctionne qu'en virgule fixe, mais sur des registres très larges (80 bits en interne) ce qui fournit une bonne précision sur les calculs.

Excepté pour les applications les plus simples, un DSP est utilisé conjointement à un CPU qui s'occupe de faire fonctionner le système d'exploitation et utilise le DSP comme un coprocesseur spécialisé pour les traitements audio et vidéo (compression, décompression, filtrage). Les DSP sont plutôt destinés à être utilisés dans des applications embarqués contraintes (téléphones cellulaires, lecteurs multimédia portatifs) grâce à leur excellent ratio performances/consommation.

Caractéristiques Les DSP sont souvent des architectures à mémoires séparées (Harvard) pour des raisons de performances. Ils ont une grande puissance de calcul (jusqu'à 500 MFlops environ) associée à une grande bande passante vers la mémoire grâce à plusieurs bus indépendants. Le coût est modéré, excepté pour les composants capables de traitement en virgule flottante habituellement plus cher.

Pour un exemple plus concret, le DSP TMS320C6457 de Texas Instruments est un DSP simple coeur, virgule fixe, hautes performances. Fonctionnant à 1.2 GHz il est capable d'exécuter 9600 millions d'opérations MAC (Multiply and ACcumulate) par secondes sur 16 bits, grâce à une architecture VLIW exécutant 8 instructions 32 bits par cycle. Malgré ses performances élevées, il ne consomme que 5 à 6 Watts et son coût est raisonnable (145$).

2.6.1.4 *Field-programmable gate array*

Ultime évolution des circuits logiques programmables (PLD : Programmable Logic Device), les *Field-Programmable Gate Array* (FPGA) sont à la pointe de la technologie de réalisation de circuits intégrés en exploitant gravure la plus fine disponible. Constitué de millions de blocs logiques programmables reliés par une matrice d'interconnexion, il est possible d'y implémenter n'importe quelle fonction logique combinatoire ou séquentielle. On utilise pour les programmer un langage spécifique de description matérielle : VHDL ou Verilog.

Sur les composants haut de gamme à grande densité, il est possible de synthétiser entièrement un coeur de µP, avec mémoire RAM et périphériques intégrés. Cette solution « SoftCore » permet notamment de modifier le coeur du µP pour étendre le jeux d'instructions en implémentant des fonctions haut niveau (multiplication de matrice par exemple)

Table 2.3 – Caractéristiques des DSP

Organisation mémoire	Harvard (majorité)
Partage mémoire	SMP
Jeux d'instruction	VLIW, Vectoriel
Taxinomie de Flynn	SIMD, MIMD (selon modèle)
Optimisations	Pipeline, Cache, Unités d'adressage spécifiques
Puissance de calcul	100 à 10000 MMACS
Largeur des données	16 à 80 bits
Consommation électrique	1 à 10 W
Encombrement	Faible (optimisé pour systèmes embarqués)
Coût	Faible à Moyen (5 à 200 €)
Flexibilité	Moyenne (optimisé pour traitement numérique)
Disponibilité	Bonne (revendeurs spécialisés)

directement utilisable lors de la programmation en assembleur du µP ; il est aussi possible d'adapter les périphériques (entrées sorties, bus de communications...) aux besoins de l'application. Les performances d'un softcore sont cependant inférieurs à celle d'un processeur réel équivalent. Afin d'améliorer les performances sur certaines applications, il existe des FPGA intégrant des fonctions spécifiques « en dur » : coeurs de processeurs génériques (le Xilinx Virtex Pro intègre 2 PowerPC), blocs DSP (MAC), mémoire RAM, blocs d'entrée sorties, bus communication spécialisés, fonctions analogiques, ...

La programmation d'un FPGA s'effectue à un niveau inférieur à celles des autres processeurs présentés ici, il est donc possible d'implémenter toute forme de processeur décrite par ailleurs dans un tel composant. Il est même envisageable de réaliser un système complet µP, RAM, Entrées/Sorties, réseau etc... sur un seul FPGA, permettant le prototypage du système avant sa réalisation définitive sur composant classiques.

L'une des applications majeures du FPGA est le prototypage de circuits dédiés (ASIC : Application-Specific Integrated Circuit) permettant de valider leur fonctionnement avant la mise en production à grande échelle utilisant une technologie de gravure sur silicium.

Les FPGA sont aussi souvent utilisés pour réaliser la logique de décodage entre plusieurs composants classiques, par exemple plusieurs µP et les banques de mémoires et autres périphériques d'entrée sortie. Il prendra ainsi en charge le décodage des adresses et la répartition des interruptions, tâches très consommatrices de logique combinatoire ce qui peut être très encombrant en utilisant des circuits logiques discrets.

TABLE 2.4 – Caractéristiques des FPGA

Puissance de calcul	Dépends du circuit synthétisé
Largeur des données	Au besoin (1 bit à plusieurs centaines)
Bande passante RAM	Dépends du circuit synthétisé / Fréquence M~GHz
Consommation électrique	1 à 20 W (dépend fortement du circuit synthétisé)
Encombrement	Faible à Moyen
Coût	Moyen à Élevé (5 à 3000 $)
Flexibilité	Mauvaise (mise au point du circuit délicate)
Disponibilité	Moyenne (marché professionnel)

2.6.1.5 Graphics Processing Unit

Le GPU(*Graphics Processing Unit*) est le processeur au coeur des cartes graphiques modernes pour PC, stations de travail, ordinateurs portables et consoles de jeux vidéos. Développé au début pour soulager le CPU des calculs lourds et répétitifs liés à l'affichage, notamment pour les jeux vidéos et les applications professionnelles (CAO), le GPU a évolué peu à peu vers un modèle de plus en plus programmable au point de pouvoir être dorénavant utilisé pour effectuer des calculs génériques. Cet usage connu sous le nom de GP-GPU (*General Purpose computation on GPU*) est apparu lorsque les fabricants ont introduit dans les GPU des fonctions de re-lecture des images générées. Il devint alors possible de considérer l'image comme une matrice 2D de valeurs numériques à faire traiter par le GPU.

Le GP-GPU ne s'est réellement démocratisé qu'avec la diffusion récente de processeurs graphiques entièrement programmables (nVidia 8800 et suivants) et des outils logiciels adaptés (nVidia CUDA, OpenCL) permettant une programmation aisée en langage de haut niveau (C/C++). L'abstraction présentée au programmeur est celle d'une architecture SIMD massivement parallèle exécutant plusieurs instances (*threads)* du même programme (*kernel*) sur un ensemble de données en parallèle. Avec cette nouvelle génération de processeur et ces outils logiciels, le GP-GPU sort des laboratoires pour investir toutes les applications de calculs intensifs. La quasi totalité des PC grands public vendus depuis 2007, portables ou fixes, étant équipés de GPU compatibles avec ces nouveaux outils, les logiciels nécessitant beaucoup de puissance de calcul (traitement photo et vidéo, rendu 3D, compression, outils de simulation, ...) peuvent ainsi profiter de ce coprocesseur pour accélérer les traitements. Cette tendance a aussi atteint le domaine du HPC (*High-Performance Computing*) : le système actuellement [1] classé deuxième au Top500, appelé Nebulae, est équipé (entre autre) de 4640 [2] GPU nVidia Tesla C2050.

Combinant une grande puissance de calcul (env. 1 TFlops sur nVidia GT200), une

1. Classement de Juin 2010, http ://top500.org
2. selon http ://blog.zorinaq.com/ ?e=14

Table 2.5 – Caractéristiques des GPU

Organisation mémoire	von Neumann, mais segment programme protégé
Partage mémoire	NUMA (mémoire CPU et GPU dans 2 plans d'adresses différents)
Jeux d'instruction	Vectoriel (AMD) ou Scalaire (nVidia)
Taxinomie de Flynn	SIMD
Optimisations	Pipeline, Cache (unités textures)
Puissance de calcul	50 à 1000 GFlops
Largeur des données	32 bits
Bande passante RAM	60 Gio/s
Consommation électrique	30 W (portable) à 400 W (station)
Encombrement	Moyen (carte PCI-Express 16x, nécessite PC)
Coût	Moyen à élevé (30 à 500 €)
Flexibilité	Bonne (adapté à toute application)
Disponibilité	Excellente (marché grand public)

programmation relativement aisée, un coût modéré (50 à 2000 €) et une consommation électriques maîtrisée (100 à 400 W), les GPU offrent une alternative intéressante aux CPU pour des applications parallélisables très lourdes tout en restant sur une plateforme PC standard. Ce type de processeur ne peut néanmoins pas remplacer un CPU pour les applications séquentielles sur lesquelles il est beaucoup moins performant.

D'une architecture proche du GPU, nous pouvons aussi citer le PPU (*Physical Processing Unit*). Ce processeur est chargé d'assister le CPU d'un ordinateur pour les calculs liés à la physique (mécanique, dynamique. . .) en particulier pour la gestion des objets (collisions, inertie,...) dans les jeux vidéos. Inclus sur une carte fille PCI-express, sur une carte graphique ou directement dans la carte mère ; ce processeur permet d'accélérer le traitement de scènes complexes où des milliers d'objets interagissent. Le seul exemple de ce système, la carte PhysX de Ageira a eu une faible diffusion avant que nVidia ne fasse l'acquisition de la société et de la technologie pour fournir une implémentation software en GPU qui remplace la carte indépendante. A notre connaissance, aucun exemple d'utilisation d'un PPU pour des calculs génériques n'a été publié.

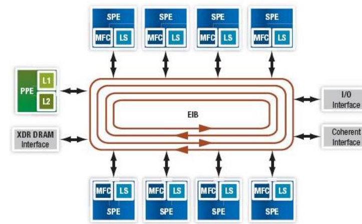


Figure 2.3 – Architecture du *Cell*

Source : http://ensiwiki.ensimag.fr

2.6.1.6 Architectures novatrices

Les grandes familles de composants présentées jusqu'à maintenant constituent les grands « classiques », largement distribuées et couramment utilisés. Des architectures plus exotiques existent néanmoins, certaines ayant même un grand succès commercial. Ces architectures, comportant souvent un seul représentant, sont présentées dans cette section.

Cell broadband engine architecture Ce composant particulier est constitué d'un µP standard (famille PowerPC) associé à des unités de traitement spécialisées, le tout sur le même puce. Pour l'instant, le seul représentant de cette famille est construit par STI, un groupement d'entreprises formé par Sony, Toshiba et IBM. Ce composant est actuellement au coeur de la console de jeu Playstation 3 de Sony et du troisième supercalculateur le plus performant [3] , le RoadRunner d'IBM.

Le « *Cell Broadband Engine* » ou CBE, ou *Cell* est constitué d'un PPE (*POWER Processing Element*) qui est une version allégée d'un µP PowerPC 64bits, auquel on a adjoint 8 SPE (Synergistic Processing Elements) qui sont des éléments de calculs spécialisés reliés par un bus interne en anneaux à très haut débit. Ce composant, à mi-chemin entre un µP et un GPU (de part sa capacité à effectuer des calculs parallèles) possède une puissance de calcul en virgule flottante conséquente (200 GFlops dans certaines applications où un CPU traditionnel de même fréquence atteint 25 GFlops).

Cette architecture est néanmoins difficile à programmer, les données devant être continuellement déplacées à travers le bus en anneau chaque SPE ayant des ressources internes (registres, RAM) très limitées et les entrées-sorties s'effectuant à travers ce bus commun.

KiloCore La société Rapport a développé, en partenariat avec IBM, deux processeurs massivement parallèles. Le premier (KC256) est constitué de 256 PE 8 bits, tandis que le second (KC1025) comporte 1024 PE associés à un PPE 64 bits de type PowerPC. La structure interne ressemble donc au CBEA, à la différence que les PE sont connectés sous forme de matrice à deux dimensions là où les PE du CBEA sont connectés en anneaux.

L'intérêt de cette architecture fortement parallèle est d'obtenir une puissance de calcul conséquente avec un composant fonctionnant à fréquence très modeste (100 MHz

3. Classement de Juin 2010, http ://top500.org

Table 2.6 – Caractéristiques du CBEA

Organisation Mémoire	von Neumann
Partage mémoire	NUMA
Jeux d'instruction	RISC, extensions Vectorielles
Taxinomie de Flynn	MIMD
Optimisations	Pipeline

pour le KC256), limitant ainsi la consommation électrique à un niveau très raisonnable (< 1 Watt).

La diffusion de ce composant intéressant semble pour l'instant confidentielle, et peu de documents sont disponibles.

2.6.2 Processeurs non conventionnels

L'information pouvant être codée sous de multiples formes : électronique, mais aussi optique, mécanique, magnétique… ; tout système capable de combiner de telles grandeurs physiques pour produire un résultat est donc un processeur. Dans cette section, nous présentons certains de ces processeurs non conventionnels traitant les données codées autrement que par l'électricité.

2.6.2.1 Processeur mécanique

Les premières machines à calculer, utilisaient des éléments mécaniques pour effectuer des opérations arithmétiques. Un ensemble de leviers et roues codeuses permettaient d'entrer les opérandes, et un système d'engrenages mues par la force humaine (ou par un moteur à vapeur ou électrique) calculait le résultat, restitué par un afficheur mécanique. Les plus simples de ces machines se limitaient aux additions et soustractions (par exemple la Pascaline de Blaise Pascal), mais d'autres beaucoup plus complexes ont été fabriqués, comme la machine à différences de Charles Babbages présentée Fig.2.4 qui permet le calcul de polynômes.

Ces machines ont cependant été assez vite remplacées par des systèmes électriques à relais, puis à lampes et enfin à transistors, beaucoup plus fiables et rapides. Ces systèmes mécaniques, de même que les processeurs à relais et lampes sont aujourd'hui obsolètes et largement inférieurs aux systèmes électroniques. Leur principal intérêt est maintenant historique.

Il faut néanmoins noter un regain d'intérêt récent envers ce type de traitement de l'information avec l'avancée des technologies MEMS (*Micro Electro-Mechanical Systems*) permettant la miniaturisation de composants mécaniques à l'échelle du micromètre. Des processeurs entièrement mécaniques étant insensibles aux effets des radiations et à la chaleur (si réalisé dans des matériaux à faible dilatation), ils pourraient être utilisés dans les environnements inhospitalier pour l'électronique : industrie nucléaire et aérospatiale.

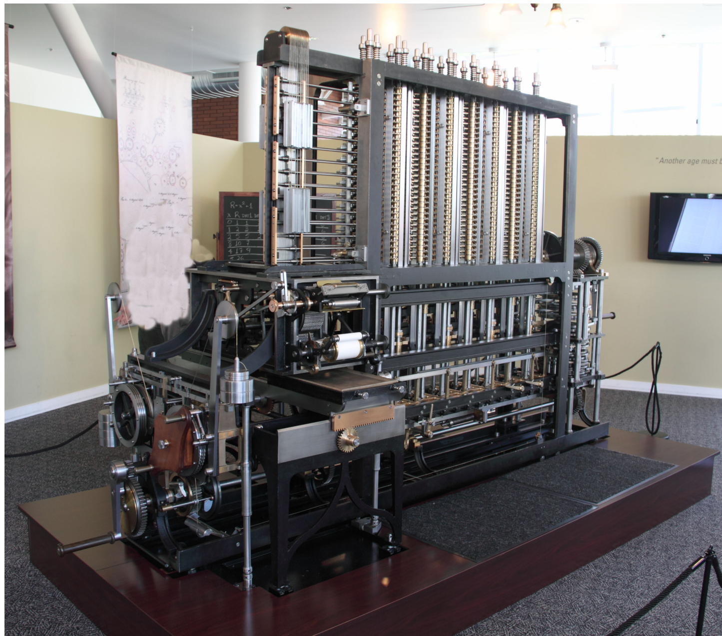


FIGURE 2.4 – La machine à différences de Charles Babbage (1834). Un exemple de processeur mécanique.

Source : Wikimedia Commons

2.6.2.2 Processeur à ADN

Pour continuer cette revue rapide des processeurs non conventionnels, nous pouvons citer le processeur a ADN qui fonctionne en laissant réagir ensemble des séquences d'ADN spécialement synthétisées pour représenter un problème. Utilisé uniquement pour résoudre des problèmes combinatoires, ce type de processeur peut examiner en même temps toutes les solutions possibles, assurant ainsi la découverte de la meilleure solution. Bien que fonctionnel, le processeur a ADN est relativement difficile à mettre en oeuvre car il nécessite des instruments de laboratoire sophistiqués et encombrants comme le montre la figure 2.5. De plus, son temps de réaction très lent (plusieurs minutes à plusieurs jours) le rend inadéquat pour toute autre application que la résolution de problèmes combinatoires ayant un nombre de solutions possibles très grand. Pour ces problèmes, le parallélisme dans la recherche de la solution permet de rentabiliser le long temps de « calcul ». Un exemple de ce type de problème est celui du voyageur de commerce où il est question de trouver le chemin le plus court pour une personne devant visiter un nombre donné de villes. Pour prouver qu'une solution à ce type de problème (dit NP-complet) est la meilleure, il est nécessaire d'explorer chaque solution possible, une tâche parfaite pour un processeur à ADN.

2.6.2.3 Corrélateur optique

Beaucoup plus adaptés au traitement du signal et des images, il existe des systèmes de traitement optiques de l'information. Ces processeurs utilisent des éléments optiques comme des modulateurs spatiaux de lumière (SLM, Spatial Light Modulator), des lentilles

FIGURE 2.5 – Processeur à ADN, prototype de développement de Olympus Optical Co., Ltd.

optiques, des filtres holographiques et sont capable de réaliser de multiples tâches, comme la reconnaissance d'un objet ou de son orientation [Tri98]. Un corrélateur optique est habituellement constitué d'un SLM générant l'image d'entrée à étudier, suivit d'une lentille permettant la génération du spectre de Fourier de cette image. Une matrice de filtres holographiques est placée avant une deuxième lentille effectuant la transformée inverse de Fourier. L'image résultante est acquise par une caméra numérique.

En utilisant une matrice de filtres représentant les spectres de Fourier d'un ensemble d'objets connus, et en plaçant en entrée une image d'un de ces objets, on obtiendra en sortie un point brillant à la position du filtre correspondant à l'objet présenté en entré, réalisant ainsi une reconnaissance. Le temps de calcul est égal au temps de propagation de la lumière à travers le corrélateur et n'est en pratique limité que par la bande passante du transducteur d'entrée (le SLM) et celui de sortie (la caméra numérique). De plus, la recherche de correspondance s'effectue en parallèle sur tous les filtres. Ces systèmes sont donc très rapides et plus performants pour effectuer ces opérations que leurs équivalents électroniques, notamment grâce à l'exploitation du parallélisme. Leur flexibilité est cependant limitée, habituellement un corrélateur optique ne peut effectuer qu'un seul type d'opération et nécessite d'être complètement reconfiguré pour effectuer une autre tâche. Les systèmes optique utilisés sont aussi très coûteux et la nécessité de travailler en lumière cohérente (laser) oblige à traiter spécialement l'image d'entrée.

2.6.2.4 Autres processeurs non conventionnels

Il existe un grand nombre d'autre processeurs exotiques ; certains ne sont que des ébauches, d'autres ont été conceptualisés et simulés (processeur à boules de billard) et certains sont en développement (processeur quantique), ils ont cependant la caractéristique commune de n'être pas encore prêt pour une utilisation et donc ne seront pas détaillés ici.

2.7 Synthèse

Nous avons présenté jusqu'à maintenant les différents processeurs disponibles et éventuellement utilisables pour notre application. Pour chacune des ces architectures, nous avons proposé une classification et une caractérisation en fonctions de critères communs.

Dans cette sections, nous allons maintenant proposer une synthèse de cet état de l'art afin d'opérer une comparaison qui nous permette de sélectionner l'architecture cible de nos développements.

2.7.1 Processeurs non conventionnels

Les processeurs non conventionnels présentés ici sont inutilisables actuellement. Il souffrent d'une trop grande spécialisation (corrélateur optique, processeur à ADN) ; sont très difficiles à mettre en oeuvre ; ou sont les vestiges d'une technologie obsolète et complètement dépassée (processeurs mécaniques).

Ces solutions ne peuvent donc pas représenter un choix pour notre application.

2.7.2 Microprocesseurs

Le principal intérêt de ces composants réside dans leur forte intégration, leur consommation réduite et leur faible coût. Cependant, ils ne sont pas destinés à être utilisé pour des applications de traitement intensif et présentent donc des performances de calcul limitées.

Seuls certains SoC possédant des fonctions dédiées au traitement d'image pourraient être envisageables pour notre application, ces composants sont cependant difficiles à mettre en oeuvre et à se procurer.

2.7.3 DSP et FPGA

Ces deux architectures présentent la même limite en ce qui concerne notre application : leur flexibilité limitée. Durant ces travaux de thèse, l'algorithme subira de fréquentes modifications et adaptations lors de la mise au point de la méthode de reconstruction 3D. Le manque de flexibilité de ces architectures imposera de longues et difficiles réimplémentations à chaque modification ce qui est fortement pénalisant.

Dans le cas d'une application où l'algorithme est connu et figé, ces architectures représentent cependant de bonnes solutions étant donné la puissance de calcul disponible sur DSP et la possibilité d'adapter finement l'architecture matérielle du FPGA au problème. Les méthodes d'AAA (Adéquation Algorithme-Architecture) classiques utilisent souvent un FPGA comme cible du développement pour cette raison.

2.7.4 Architectures novatrices

Le CBEA est facile à se procurer : il suffit d'acheter une Playstation 3. Sa programmation et son utilisation pour d'autres applications que les jeux vidéos est possible grâce au portage du système d'exploitation Linux sur cette machine (cette possibilité a cependant

été bloquée par Sony récemment[4]). Ce processeur est cependant difficile à programmer et présente donc les mêmes inconvénients que les DSP et FPGA dans sa mise en oeuvre.

Concernant le KiloCore, ce composant n'est pas encore disponible sur le marché.

2.7.5 CPU et GPU

Ces processeurs présentent d'excellentes caractéristiques de performance, coût, disponibilité et flexibilité. La consommation électrique est cependant leur point faible, à l'exception des systèmes optimisés pour les ordinateurs portables.

Le CPU, faiblement parallèle, est à son avantage dans les traitements séquentiels. Le GPU quand à lui est mieux adapté aux traitements massivement parallèles de type SIMD.

Une solution combinant ces deux processeurs semble la plus adaptée à notre problème.

2.7.6 Solution retenue

L'évolution actuelle des microprocesseur étant orientée vers la production de systèmes parallèles intégrant de plus en plus de coeurs de calcul, nous avons privilégié une solution basée sur une plateforme PC intégrant un GPU puissant et un CPU flexible.

Cette solution est facile à mettre en oeuvre, peu coûteuse et facilement disponible sur le marché. Étant une plateforme standard largement diffusée, les résultats des expérimentations présentés ici seront facilement reproductibles par d'autres équipes de recherche.

Dans le cadre de notre application visant à calculer un modèle 3D d'une scène inconnue en temps réel, la priorité doit être donnée à la puissance de calcul, la vitesse d'accès à la RAM et à la rapidité de transfert des images vers le processeur. Pour ces raisons, nous avons décidé d'utiliser un GPU nVidia comme cible principale de nos développements. Le CPU sera ici utilisé pour faire fonctionner le système d'exploitation et pour les tâches annexes. La disponibilité d'interfaces logicielles de haut niveau pour la programmation a aussi contribué à la sélection de cette architecture.

2.8 Présentation détaillée de l'architecture CUDA

CUDA est une interface logicielle permettant l'utilisation des processeurs graphiques de la marque nVidia pour le calcul générique. Cette interface, basée sur le langage C, se présente sous la forme d'un précompilateur spécifique et d'un ensemble de librairies. Elle définit une liste d'extensions au langage C permettant l'utilisation des capacités massivement parallèles du GPU qui sont interprétées par le précompilateur et séparées des instructions classiques traitées par le compilateur C normal. CUDA est présenté en détail dans [CUD10].

L'architecture CUDA présente le GPU sous la forme d'un processeur SIMD (un seul programme opère sur plusieurs flots de données différents en parallèle). Une fonction destinée à s'exécuter sur le GPU est appelée un *kernel*, et on spécifie le nombre de thread à exécuter en parallèle qui seront autant d'instances de ce *kernel* lors de l'appel à cette fonction. Bien que le programmeur spécifie le nombre global de threads à lancer, le nombre

4. http ://blog.us.playstation.com/2010/03/28/ps3-firmware-v3-21-update/

réel de threads s'exécutant en simultané et leur ordonnancement sont des paramètres gérés par le driver de la carte graphique en fonction des ressources matérielles disponibles sur la carte graphique. Ceci permet une compatibilité du code compilé avec toute la famille des processeurs nVidia compatibles avec CUDA.

2.8.1 Organisation des threads

L'organisation des différents thread instanciés par le lancement d'un *kernel* est hiérarchique. Un *kernel* est instancié sous la forme d'une *grille* de *threads* de taille variable. Cette grille est divisée en *blocks* de threads.

Par exemple, il est possible d'exécuter un *kernel* sous la forme d'une grille de 128 blocks contenant chacun 16 threads. On obtient ainsi $128 \times 16 = 2048$ threads qui fonctionneront en parallèle, si les ressources disponibles sur la carte graphique le permettent (dans le cas contraire, les threads non instanciables sont exécutés à la suite).

Les threads peuvent être organisés sous la forme d'un block à une, deux ou trois dimensions ; les blocks peuvent être organisés sous la forme d'une grille à une ou deux dimensions.

La taille de block et de la grille est un paramètre configurable lors du lancement du kernel. Comme le kernel est constitué d'un unique programme, chaque thread exécute exactement le même code en même temps. Pour permettre à chaque thread d'accéder à un flot de donnée différent des autres threads, une constante connue par chaque thread contient l'indice du thread en cours. Cette indice sera utilisé pour calculer les adresses mémoires à accéder. Cet indice comprend l'indice du thread dans le block courant et l'indice du block courant dans la grille. La taille de block et de grille sont aussi des paramètres connus du thread. Cette organisation est présentée Fig. 2.6.

2.8.2 Organisation de la mémoire

La mémoire accessible par chaque thread est organisée hiérarchiquement à la manière des thread eux-mêmes. Cette organisation est présentée dans la figure 2.7.

Chaque thread à tout d'abord accès à un ensemble de registres (*Registers*). Limités en taille, ces registres sont très rapides et accessibles en un seul cycle d'horloge. Ils sont idéalement utilisés pour les variables privées modifiées fréquemment.

Au niveau du block, une zone mémoire partagée (*Shared Memory*) est aussi accessible. Sa taille est allouée dynamiquement lors de l'exécution du kernel. Elle est accessible en lecture et en écriture par tous les threads du même block. Très rapide, elle est accessible en un seul cycle sous réserve que les différents threads du block organisent leurs accès selon un schéma particulier. En effet, des mécanismes câblés permettent par exemple une diffusion du contenu d'une adresse à l'ensemble des threads du block en un cycle. De même, si chaque thread accède à une l'adresse immédiatement supérieure à son voisin dans le block, l'accès est optimisé et s'effectue à la vitesse maximale. Dans le cas d'un accès mal structuré, les opérations sont séquentialisées et il peut en résulter une baisse de performances.

La mémoire interne au GPU, séparée entre registres et mémoire partagée, bien que rapide, est relativement petite (quelques Mio maximum). Une RAM externe (*Device Me-*

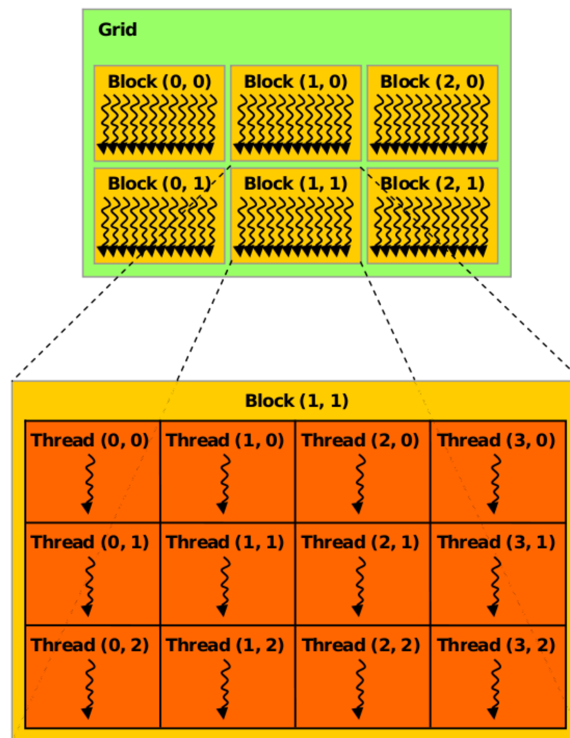


FIGURE 2.6 – Organisation hiérarchique des threads CUDA

Crédit : nVidia CUDA Programming Guide

mory) de plus grande taille (128 Mio à 1 Gio couramment, jusqu'à 4Gio sur les modèles haut de gamme) est présente sur la carte graphique pour stocker les grands volumes de données. Dans l'architecture CUDA, elle est partagée en 4 zones distinctes ayant chacune des méthodes d'accès différentes.

La *mémoire locale* est utilisée pour « délester » les bancs de registres internes, lorsque un kernel utilise plus de mémoire que disponible. Cette fonctionnalité est nécessaire pour assurer une compatibilité au niveau binaire entre les différents GPU. Elle présente néanmoins un grand danger : son utilisation est très coûteuse en temps d'accès (200 cycles), ce qui pénalise énormément un programme qui l'utilise. De plus, ce délestage est effectué automatiquement, il est donc nécessaire de s'assurer lors de la compilation qu'un kernel n'alloue pas plus de registres que disponibles sur le processeur ciblé. Cette mémoire est privée car chaque thread ne peut accéder qu'à sa propre zone.

La *mémoire de texture* est utilisée pour stocker des tableaux de données (1, 2 ou 3D). Un ensemble de circuits électroniques câblés fournissent des fonctionnalités additionnelles : les coordonnées peuvent être « wrappées » (dans un tableau 1D à n éléments, la lecture de l'élément $n+1$ renvoie l'élément 0) ou « saturées » (l'accès à $n+1$ renvoie l'élément n) et les valeurs peuvent être converties ou interpolées (dans ce cas, les coordonnées sont des nombres rationnels et la valeur retournée est interpolée entre les plus proches voisins). De plus, les accès concourants de plusieurs threads sont optimisés et un cache accélère le temps d'accès pour les éléments lus plusieurs fois. Cet ensemble de fonctionnalités est très utile lorsque l'on travaille avec des images (2D) ou des signaux échantillonnés (1D). La

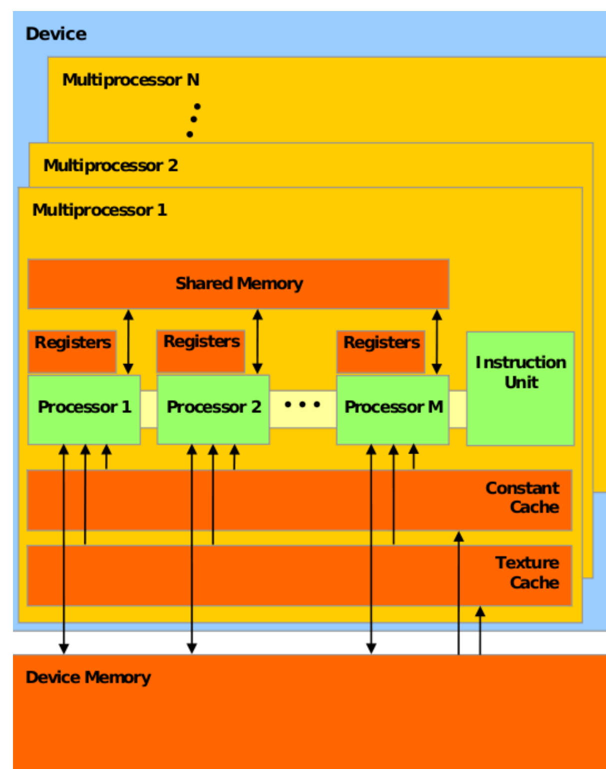


Figure 2.7 – Organisation de la mémoire CUDA

Crédit : nVidia CUDA Programming Guide

latence d'accès est constante, et l'interpolation linéaire ne coûte pas plus de cycles qu'une lecture normale. La mémoire de texture est accessible en lecture seule depuis un kernel, mais peut être écrite par le CPU.

La *mémoire de constantes* est une petite zone en RAM destinée à recevoir des données partagées par l'ensemble des threads. Comme la mémoire de texture, elle est accessible en lecture seule depuis un kernel mais est inscriptible par le CPU. Un système de cache permet l'optimisation des accès, elle est de ce fait très efficace.

Enfin, le reste de la RAM (la plus grande partie) est réservée à la *mémoire globale*. Cette zone de grande taille est accessible en lecture et écriture à la fois par le kernel et par le CPU, ce qui en fait un canal idéal pour l'échange bidirectionnel d'informations. Il n'y a pas de cache et il n'existe pas de système particulier pour l'optimisation des accès, son utilisation doit donc être réalisée avec prudence pour en tirer les meilleures performances possibles. Un point important est de respecter l'alignement des adresses accédées afin de maximiser la bande passante de la RAM. La latence de cette mémoire est grande (200 cycles), mais la large bande passante permet des transferts soutenus.

2.8.3 Vue d'ensemble de l'organisation des calculs

Un programme utilisant CUDA exploitera le GPU pour les séquences parallélisables du code. Il y a peu d'intérêt à l'utiliser pour les parties séquentielles car le CPU sera beaucoup plus performants et plus facile à programmer dans ce cas de figure. Un programme typique est donc constitué de passages de code séquentiel et de code parallèle entrelacés. Des transferts de données entre la mémoire du GPU et la mémoire du CPU interviendront probablement entre chaque séquence. La mémoire globale étant laissée en l'état entre chaque exécution de kernel, il est possible de séparer un traitement complexe en plusieurs kernels qui seront exécutés à la suite, éventuellement avec un nombre et une organisation différente des threads.

2.8.4 Évaluation des performances de CUDA

Afin de tester la performance d'un GPU par rapport à un CPU, et de faire en sorte que cette comparaison soit significative, nous avons décidé d'implémenter la partie centrale de notre algorithme (décrite au chapitre 3) sur les deux systèmes : le calcul de la projection d'un point 3D sur le plan image de la caméra. Ce calcul utilise le modèle de caméra dit « modèle de la sphère équivalente ». Les paramètres de ce modèle sont obtenus par une opération de calibration, puis un ensemble de points 3D est choisi (les centres des voxels d'une grille régulière). Les projections de tous ces points sont ensuite calculées par deux programmes. Le premier programme utilise un seul coeur du CPU de notre plate-forme de test, calculant les projections des points 3D de manière séquentielle. Le second utilise le GPU via CUDA pour effectuer les calculs, chaque thread effectue les calculs pour un point 3D et écrit les résultats dans la mémoire globale, et de nombreux threads s'exécutent en parallèle sur les 12 multiprocesseurs de la carte nVidia 8800GTS. Les données (paramètres de calibration, coordonnées des points 3D) sont placés en mémoire de constante.

La Fig.2.8 présente le temps d'exécution des deux variantes du programme, sur 2 CPU (Intel Core 2 Duo E8400 à 3GHz et Core 2 Quad Q6600 à 2,4GHz) et 2 GPU (nVidia

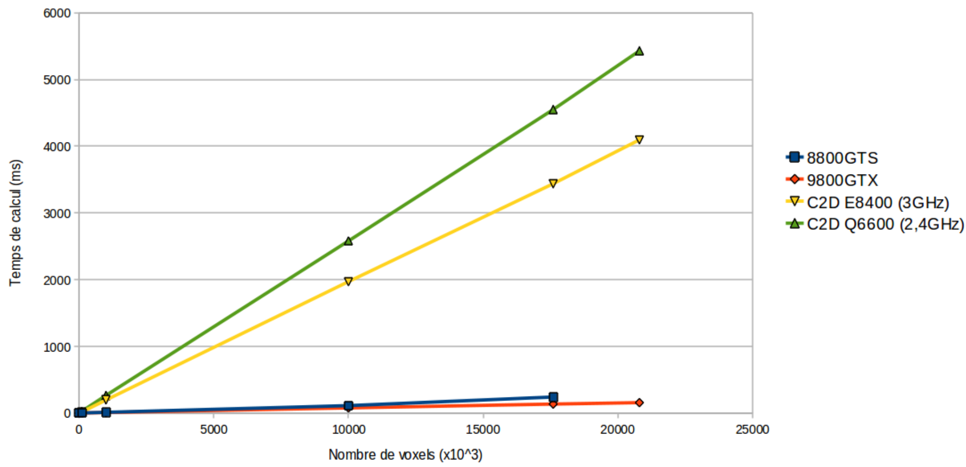


FIGURE 2.8 – Comparaison des performances CPU et GPU

TABLE 2.7 – Temps de traitement (ms)

Nombre de Voxels	8800GTS	9800GTX	Q6600	E8400
1 000	0,02	0,03	0,26	0,19
10 000	0,18	0,11	2,58	1,96
100 000	1,4	0,84	25,8	14,85
1 000 000	11	7,3	264	195,4
10 000 000	111	77	2580	1970
17 600 000	240	134,5	4550	3438
20 800 000	xxx	156	5430	4096

GeForce 8800GTS, 320 Mio RAM et 9800GTX+ 512 Mio RAM), constituant nos deux machines de test.

Les temps de calcul reportés sont mesurés par la fonction *clock* de la librairie standard C sur plusieurs répétitions (jusqu'à 1000) de l'algorithme pour augmenter la précision. Les temps de calculs reportés pour les GPU incluent aussi le temps de transfert des données depuis la RAM du GPU vers la RAM de l'hôte. Ces temps de transfert sont d'ailleurs largement dominants, le temps de calcul proprement dit restant en deçà du seuil mesurable sur ce kernel très simple.

La complexité algorithmique de la fonction de projection est en $O(N)$ avec N le nombre de points 3D à projeter (c'est une combinaison linéaire des paramètres de calibration avec les coordonnées des points 3D). Cette complexité est la même pour les deux implémentations, mais la version GPU bénéficie d'une exécution en parallèle sur plusieurs processeurs. La complexité en temps de l'implémentation CPU est donc en $O(N)$ (sur 1 seul coeur) tandis que la complexité en temps de l'implémentation GPU est en $O(N/P)$ avec P le nombre de actifs threads en parallèle (768 threads par multiprocesseur, 12 multiprocesseurs sur 8800GTS, 16 sur 9800GTX+ soit 9216 et 12288 threads actifs respectivement). La fréquence de fonctionnement des GPU est cependant inférieure à celle des CPU (1.35GHz pour le 8800GTS et 1.89GHz pour le 9800GTX+ comparé à 2,4 et

Table 2.8 – Bande-passante GPU vers CPU (Mio/s)

Nombre de Voxels	8800GTS	9800GTX
1 000	695,65	533,33
10 000	888,89	1454,55
100 000	1142,86	1904,76
1 000 000	1454,55	2191,78
10 000 000	1441,44	2077,92
17 600 000	1173,33	2093,68
20 800 000	pas possible (limite RAM)	2133,33
Benchmark « bandwidthTest »	1505,8	2137,7

3 GHz).

Pour le 9800GTX+, les benchmarks fournis avec CUDA donnent une bande passante maximale de 2,1 Gio/s de la RAM GPU vers l'hôte. Ces valeurs sont proches de celles mesurées par ces expérimentations comme indiqué par le tableau 2.8.

2.9 Conclusion

Cet état de l'art des principales architectures des processeurs présente les principales caractéristiques de ces composants et compare leurs caractéristiques.

Actuellement, les architectures les plus performantes (en terme de puissance de calcul maximale théorique) sont indéniablement celles permettant une exécution parallèle d'un grand nombre de tâches. Cette tendance se confirme à plusieurs niveaux, tant sur les très gros systèmes de calcul haute performance comme RoadRunner d'IBM, que les systèmes embarqués utilisant des DSP multicoeurs. Les PC ne sont pas en reste, avec un CPU intégrant de plus en plus de coeurs, eux-mêmes utilisant des techniques de parallélisation en interne (hyperthreading, superscalaire, pipeline).

Devant la variété des processeurs actuellement disponibles, il est difficile de réaliser une comparaison exhaustive des performances pour une application donnée, d'autant plus que la plupart des architectures ne sont pas compatibles entre elles, et donc une implémentation différente est nécessaire à chaque fois rendant la comparaison artificielle.

La solution retenue pour notre application exploite un PC dont le GPU constitue le moteur principal de notre programme de traitement. Cette solution peu coûteuse et performante a montré son intérêt face à l'utilisation du CPU seul lors de tests.

Chapitre 3

Dispositif et méthode proposés

Sommaire

3.1 Objectifs **62**
3.2 Vue d'ensemble **62**
3.3 Matériel **63**
3.3.1 Système de vision 63
3.3.2 Plateforme robotique mobile 64
3.3.3 Ordinateurs 64
3.4 Principes au coeur de la méthode **65**
3.4.1 Modélisation volumétrique 65
3.4.2 Photoconsistance 66
3.4.3 Reconstruction incrémentale 66
3.5 Implémentation massivement parallèle **67**
3.5.1 Accès parallèle aux données 67
3.5.2 Structure de données 68
3.6 Description détaillée **71**
3.6.1 Calcul de l'indice du voxel 72
3.6.2 Calcul des coordonnées du voxel 74
3.6.3 Estimation de la visibilité 74
3.6.4 Projection 3D vers 2D 77
3.6.5 Projection d'un voxel 78
3.6.6 Mesure de la photoconsistance 81
3.6.7 Détermination de l'état d'un voxel 82
3.7 Conclusion **83**

Le chapitre 1 a présenté différentes méthodes de reconstruction 3D par vision et le chapitre 2 a détaillé les processeurs actuellement sur le marché.

Ce chapitre présente la méthode de reconstruction 3D mise au point durant ces travaux de thèse. Nous commencerons par présenter les différents objectifs de notre système de reconstruction 3D ainsi qu'une vue d'ensemble de la méthode proposée. Nous exposerons aussi les principes fondamentaux au coeur de notre méthode de reconstruction 3D.

Nous présenterons ensuite le matériel utilisé comme support à notre méthode : le système optique d'acquisition d'images et le matériel informatique.

Les méthodes utilisées pour adapter cet algorithme à l'architecture massivement parallèle sélectionnée dans le chapitre 2 sont ensuite détaillées, notamment en ce qui concerne les structures de données.

Nous aborderons ensuite une description détaillée de la mise en oeuvre de notre méthode avant de conclure ce chapitre.

3.1 Objectifs

La méthode de reconstruction 3D développée ici a pour but de calculer en temps réel un modèle 3D dense d'une scène inconnue à partir d'images issues d'une paire de caméras catadioptriques. Cet outil doit fournir une représentation 3D « bas niveau » de la scène reconstruite, l'interprétation étant laissée aux couches supérieures de l'application finale.

La priorité est donnée à une reconstruction 3D rapide afin d'obtenir un modèle de la scène en quelques secondes au maximum. La chaîne de traitement dans son ensemble est orienté vers cet objectif, avec l'acquisition d'images par des caméras panoramiques permettant d'obtenir une vue à 360° de l'environnement en une seule prise de vue.

Principalement orientée vers des applications embarquées de type « robotique mobile », le programme de reconstruction 3D doit être capable de fonctionner dans un système informatique compact et monolithique. Deux aspects sont donc à mettre en relation pour obtenir des performances optimales : l'algorithme de reconstruction et l'architecture le faisant fonctionner. Dans le chapitre 2 nous avons montré que les GPU présentent des caractéristiques intéressantes en terme de performance, compacité et coût. Cette architecture présente la particularité d'être massivement parallèle et nécessite donc un algorithme adapté pour délivrer des performances maximales. La méthode de reconstruction proposée ici a été conçue pour tirer partie de ce type d'architecture.

3.2 Vue d'ensemble

Le processus de reconstruction 3D fait intervenir les éléments matériels suivants :

- Un système de vision composé de deux caméras catadioptriques
- Un PC chargé d'effectuer les traitements

Une vue d'ensemble de cette chaîne de traitement est présentée dans la figure 3.1.

D'un point de vue logiciel, la chaîne de traitement réalise les opérations suivantes :

- Capture des images (Acquisition)
- Calcul du modèle 3D de la scène (Traitement)

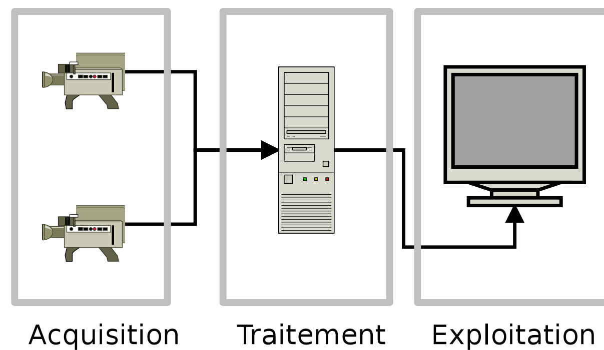


FIGURE 3.1 – Chaîne matérielle de traitement

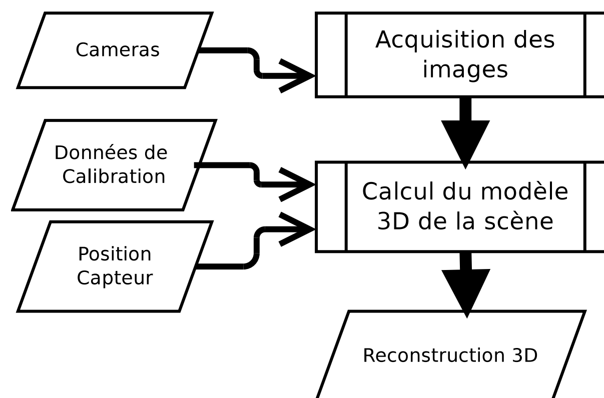


FIGURE 3.2 – Vue d'ensemble de l'algorithme de reconstruction 3D

— Archivage ou affichage du modèle 3D

L'opération d'acquisition consiste principalement à activer les caméras puis à télécharger les images vers l'ordinateur. Cette opération est effectuée simultanément sur les deux caméras de manière à obtenir une acquisition synchrone de la scène.

L'opération d'exploitation du modèle consiste, dans notre étude, à afficher une représentation du modèle sur un écran. Si nécessaire, le modèle peut aussi être enregistré sur le disque dur. Cet archivage du modèle 3D se rapproche ainsi d'une application de numérisation d'un environnement en 3D afin de permettre, par la suite, des visites virtuelles.

3.3 Matériel

Après cette vue d'ensemble du processus de reconstruction 3D, nous allons donner plus de détails sur les éléments matériels qui composent ce système.

3.3.1 Système de vision

Le système de vision innovant utilisé dans ces travaux résulte de la conception menée au laboratoire lors des travaux de Nicolas Ragot et de Rémi Boutteau [Rag09, BSEM08]. Le système est composé de deux caméras catadioptriques placées en configuration stéréo-

FIGURE 3.3 – Le robot mobile utilisé comme plateforme de test (vue de profil).
La potence supportant le système de vision est portée par un châssis équipé de quatre roues motrices.

scopique l'une au dessus de l'autre. Chaque caméra est capable d'observer un champ de vue de 360° autour du robot et de 106° en vertical.

Une description complète et détaillée de ce système a déjà été donnée dans la section 1.2.5.

3.3.2 Plateforme robotique mobile

Le robot présenté figure 3.3 est un modèle du commerce, le Pionner 3 fabriqué par *Mobile Robots* et utilisé comme plateforme roulante. Il est équipé d'un PC embarqué, de capteurs de proximité à ultra-son, d'un télémètre laser rotatif, d'un routeur réseau wifi et du système de vision. Dans les travaux présentés ici, le robot est téléopéré via le réseau sans fil et seules les caméras sont utilisées pour participer à la reconstruction 3D (les données issues des autres capteurs sont ignorées).

Le PC embarqué (de faible puissance) est utilisé uniquement pour commander les déplacements du robots. Le traitement d'image proprement dit est déporté grâce au réseau sans fil sur un PC plus puissant présenté dans la section suivante.

3.3.3 Ordinateurs

Pour les raisons expliquées dans le chapitre 2, nous avons utilisé une plateforme PC composée d'un CPU multi-coeurs et d'un GPU haut de gamme. Ce choix permet d'explorer les possibilités en matière de traitement parallèle et hybride.

La plateforme PC permet aussi d'utiliser un système d'exploitation et des outils de développement modernes et haut niveaux qui facilitent la mise au point des programmes. Enfin, la présence d'interfaces standard (USB, Ethernet, VGA, . . .) permettent de connecter directement les équipements externes, comme par exemple les caméras.

Durant ces travaux de thèse, nous avons utilisé principalement deux PC constitués des composants suivants :

- Premier PC
  - Processeur Intel Core2 Q6600 2,4 GHz (4 coeurs)
  - 4 Gio de RAM système (DDR2)
  - Carte graphique nVidia 8800GTS avec 320 Mio de RAM embarquée
  - Système d'exploitation GNU/Linux Ubuntu 8.04 Serveur
- Second PC
  - Processeur Intel Core2 E8400 3.0 GHz (2 coeurs)
  - 8 Gio de RAM système
  - Carte graphique nVidia 9800GTX+ avec 512 Mio de RAM embarquée
  - Système d'exploitation GNU/Linux Ubuntu 9.04

Les résultats présentés dans le chapitre 4 utilisent le second PC, mais quelques comparatifs ont été réalisés sur les deux machines.

Le développement logiciel a été réalisé en utilisant le langages C++ compilé par GCC, ainsi que CUDA pour programmer le GPU.

3.4 Principes au coeur de la méthode

L'opération de traitement, qui constitue la partie centrale du processus de reconstruction, s'inspire de la méthode « Voxel Coloring » proposée par [SD97] et présentée en détail dans la section 1.4.2.1. Cette méthode de reconstruction volumétrique multi-vues a été adaptée pour un système stéréoscopique panoramique en mouvement dans la scène à reconstruire.

Cette section présente les principes au coeur de notre méthode de reconstruction volumétrique, massivement parallèle et basée sur la photoconsistance.

3.4.1 Modélisation volumétrique

La méthode de reconstruction 3D que nous proposons modélise la scène sous la forme d'un volume parallélépipédique subdivisé en sous volumes de taille fixe appelés voxels. Cette subdivision est régulière et forme une grille en trois dimensions. Le modèle de la scène est construit en indiquant pour chaque voxel son état, transparent ou opaque, et sa couleur.

Cette stratégie de reconstruction a été sélectionnée car elle permet une répartition du travail de reconstruction entre plusieurs processeurs en affectant à chacun un ensemble de voxels à traiter. Elle offre ainsi l'opportunité de pouvoir utiliser une architecture massivement parallèle pour reconstruire rapidement la scène. L'utilisation conjointe de ce type de modélisation volumétrique et d'une architecture massivement parallèle peut se retrouver par exemple dans [GMDH08, LBN08, SS09]. Ce type de modélisation a été détaillé dans la section 1.3 et nous aborderons la description détaillée de la structure de données utilisée pour stocker ce modèle dans la section 3.5.2.

3.4.2 Photoconsistance

Le principe au coeur de la méthode de reconstruction est un test de photoconsistance. En formant l'hypothèse d'une illumination lambertienne de la scène, la surface d'un objet est vue d'une couleur indépendante de la direction d'observation. En conséquence, lorsque l'on observe une région d'une scène inconnue selon deux directions distinctes, si cette zone est perçue de la même couleur selon les deux directions alors elle représente probablement la surface d'un objet. Dans le cas contraire, si les couleurs perçues sont différentes, alors la zone observée est soit située dans un espace inoccupé de la scène, soit située derrière un obstacle. La méthode de reconstruction fonctionne en exploitant ce principe.

Le processus de reconstruction traite ensuite chaque voxel indépendemment et a pour tâche d'indiquer si le voxel et *opaque* ou *transparent* en se basant sur le test de photoconsistance. Les projections du voxel considéré sur les images sont calculées en utilisant les paramètres des caméras obtenus par une calibration géométrique.

La couleur de ces projections est ensuite comparée afin de déterminer si elles sont photoconsistante ou non. Comme expliqué précédemment, si un voxel est photoconsistant, il représente probablement un élément de surface d'un des objets de la scène. Dans ce cas, il est marqué *opaque* et sa couleur est stockée. Si le voxel testé n'est pas photoconsistant, il représente probablement un espace inoccupé de la scène et est marqué *transparent.*

Une fois cette opération effectuée pour l'ensemble des voxels constituant le volume d'intérêt, l'ensemble des voxels opaques et la couleur qui leur est affectée constitue le modèle 3D de la scène.

3.4.3 Reconstruction incrémentale

L'une des différences majeures de notre système par rapport aux méthodes de reconstruction 3D volumétrique classiques consiste en une méthode d'acquisition radicalement différente.

Alors que les méthodes classiques exploitent en général un grand nombre (4 à 16) de caméras perspectives positionnées de manière à encercler la scène ou l'objet à reconstruire, notre système comporte seulement deux caméras qui sont, de plus, positionnées à l'intérieur de la scène et évoluent dans celle-ci.

Notre système d'acquisition stéréoscopique fournit ainsi moins de points de vues que les systèmes multi-vues utilisés couramment. De plus, les acquisitions étant espacées dans le temps par le déplacement du robot d'une position à l'autre, ces points de vues ne sont pas synchrones. Cette modalité d'acquisition nécessite donc d'adapter la méthode de reconstruction à ces contraintes.

Pour s'adapter à ces contraintes, nous avons mis au point une méthode de reconstruction incrémentale qui vient enrichir, améliorer et corriger le modèle 3D avec de nouvelles informations à mesure que de nouvelles vues de la scène sont disponibles.

Cette méthode consiste à conserver le modèle 3D d'une prise de vue à l'autre, et à utiliser les données de ce modèle en plus des données issues des caméras pour calculer un nouveau modèle de la scène. Cette approche permet d'une part d'intégrer des informations issues de zones de la scène précédemment inconnues, mais aussi de tenir compte de la modification de la scène (déplacement d'objet la composant).

Cette reconstruction incrémentale possède aussi un autre avantage : un modèle 3D de la scène est connu à chaque instant (excepté à l'initialisation) et il n'est pas nécessaire d'attendre de posséder toutes les vues de la scène pour procéder à la reconstruction.

3.5 Implémentation massivement parallèle

La problématique principale étudiée par ces travaux de thèse concerne la mise au point d'une méthode de reconstruction 3D efficace sur une architecture massivement parallèle dans le but d'obtenir des performances temps réel sur un ordinateur unique et compact.

Pour exploiter au mieux l'architecture massivement parallèle retenue, il est indispensable d'exécuter simultanément un grand nombre de threads de calculs : 9216 pour une utilisation optimale de la carte graphique nVidia 8800GTS [5]. L'exécution parallèle d'un tel nombre de threads impose de limiter au maximum l'interdépendance entre ceux-ci, notamment au niveau des données.

Si chaque thread de traitement est complètement indépendant des autres, il peut effectuer et terminer ses traitements sans avoir à attendre le résultat des autres threads. Dans le cas contraire, par exemple si le calcul pour un voxel nécessite le résultat du calcul opéré sur un autre voxel, il se crée des dépendances sur les données qui limitent le degré de parallélisme possible, ce qui entraîne à son tour une sous-exploitation de la machine parallèle et donc limite les performances.

Ceci n'est bien sûr pas souhaitable et nous allons étudier dans cette section comment augmenter au maximum le degré de parallélisme de l'algorithme de reconstruction.

3.5.1 Accès parallèle aux données

Afin d'adapter l'algorithme à des machines massivement parallèles, il est préférable que les différents threads de calcul soient complètement indépendant les uns des autres. De cette manière, il est possible de lancer un nombre arbitraire de threads opérant simultanément sans rencontrer de problèmes d'interdépendances des données qui bloqueraient l'exécution des threads.

Nous avons ici choisi de répartir le travail de cette façon : **1 thread = 1 voxel**. Plusieurs milliers de threads sont donc lancés simultanément, ce qui assure une utilisation optimale de l'architecture et donc de bonnes performances.

Pour mettre en oeuvre cette répartition parallèle des calculs, il est nécessaire que la procédure effectuant le traitement d'un voxel opère indépendamment des autres. Nous avons mis au point la méthode de reconstruction en se basant sur cette contrainte.

La fonction calculant l'état d'un voxel a besoin des constantes suivantes :

5. 12 multiprocesseurs, 768 threads par multiprocesseur pour un taux d'occupation de 100%. Source : CUDA occupancy calculator

Donnée	Type de donnée
Indice du voxel traité	Constante locale
Taille de V	Constante globale
Taille d'un voxel	Constante globale
Données de calibration	Constantes globales
Seuil de photoconsistance	Constante globale

De plus, cette fonction a besoin d'accéder à deux zones mémoires distinctes :

Données	Type d'accès
Images acquises par les caméras	Lecture seule
Modèle 3D (état précédent du voxel, nouvel état)	Lecture / Écriture

Dans le cas d'une application multi-threads, les variables locales (utilisées par un seul thread) ne posent aucun problème d'interdépendance. Il en est de même des constantes globales accédées uniquement en lecture par tous les threads.

Les images sont un cas particulier : chaque thread accédera à une portion de l'image qui lui est propre. Comme cet accès est effectué en lecture seule, il n'y a pas de problèmes d'interdépendance. Étant donné que la quasi totalité de l'image est lue, il est néanmoins important d'optimiser ces accès pour ne pas saturer la bande passante de la mémoire.

Enfin, l'accès à la structure de données contenant les informations sur les voxels est celle qui pose le plus de problèmes. En effet, cette structure sera successivement lue et écrite par de nombreux threads, il est donc important d'éviter les conflits d'accès, par exemple lorsqu'un un threads veut lire une cellule en même temps qu'un autre essaye d'y écrire une donnée.

Les facteurs limitant le degré de parallélisme (nombre de threads) seront principalement les suivants :

- Nombre de voxels dans la scène
- Efficacité de la lecture des images
- Efficacité des accès à la structure de donnée contenant le modèle de la scène

Un point particulier qui mérite une étude plus approfondie concerne la structure de données utilisée pour stocker le modèle de la scène.

3.5.2 Structure de données

Différentes structures de données adaptées au stockage de modèles 3D ont été présentées dans la section 1.3. Nous nous intéressons ici aux solutions permettant plus précisément de stocker et manipuler des modèles 3D volumétriques et adaptées à un traitement parallèle.

La scène à reconstruire est inclue dans un parallélépipède rectangle et celui-ci est subdivisé en voxels cubiques de taille fixe, selon un pavage dense en 3D. Le parallélépipède rectangle englobant la scène est appelé VOI (*Volume Of Interest*) et sa taille est VX,VY,VZ selon les trois dimensions.

Le nombre de voxels constituant le modèle 3D de la scène est noté NX,NY,NZ. La taille d'un voxel notée vX,vY, vZ est donnée par la relation

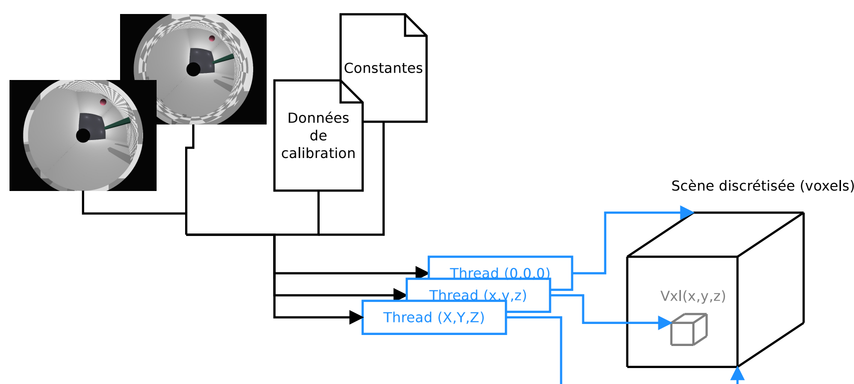


Figure 3.4 – Répartition des calculs et flux de données : un thread traite un voxel

$$\begin{cases} vX = VX/NX \\ vY = VY/NY \\ vZ = VZ/NZ \end{cases}$$

Le nombre total de voxels composant une scène est donc

$$N = NX \times NY \times NZ$$

Pour constituer le modèles 3D de la scène, il faut stocker dans chaque voxel son état (opaque ou transparent) ainsi que sa couleur. Si la couleur utilise l'espace RGB avec 8 bits par canal, il est nécessaire d'utiliser au minimum 25 bits par voxel (8×3 bits pour la couleur, 1 bit pour l'état). Il est également possible de coder l'état « transparent » par une couleur spéciale (par exemple $<0;0;0>$) ce qui limite les besoins à 24 bits par voxel. Cependant, l'architecture du processeur utilisé étant nativement 32 bits, l'adressage et l'alignement des données sont effectués sur 32 bits pour des raisons de performance. Les 8 bits additionnels sont donc disponibles pour ajouter des informations.

La « charge utile » d'un voxel est donc fixée à 32 bits, soit 4 octets, à laquelle il faut ajouter les données « de structure » parfois nécessaires. Par exemple, dans le cas d'une liste chaînée, chaque élément de la liste doit comporter, outre la donnée elle-même, un pointeur vers l'élément suivant de la liste. Un pointeur est généralement un entier de 32 bits (ou 64 bits sur les processeurs les plus récents). Avec notre donnée stockée sur 32 bits, chaque élément de la liste comporterais donc 64 bits dont la moitié seulement de charge utile.

Parmi les structures de données présentées dans la section 1.3, nous allons étudier plus précisément deux structures particulièrement bien adaptées à notre problème : le *tableau 3D* et l'*octree*.

3.5.2.1 Tableau 3D

La structure de donnée la plus simple consiste à stocker les éléments à la suite les uns des autres dans la mémoire, sans autre forme d'organisation. L'adressage des multiples dimensions est effectué en utilisant un simple décalage d'adresse mémoire.

Cette structure présente deux principaux avantages. Tout d'abord, il n'y a pas besoin de stocker de données de structure dans les éléments du tableau. Seule l'adresse du début du tableau et la taille de chacune des trois dimensions doivent être conservées à part pour pouvoir accéder à n'importe quel élément. Ensuite, l'accès à un élément est direct car son adresse est déterminée par son indice 3D et les paramètres de la structure (qui sont fixes et déterminées à l'avance). Cette dernière caractéristique est particulièrement intéressante dans le cas d'une application massivement parallèle. Un grand nombre de threads indépendant effectuent en effet des accès simultanés à la structure. Comme l'adresse d'un élément en particulier ne dépend pas des autres éléments, ces différents threads peuvent effectuer les accès en lecture et écriture sans occasionner de conflits avec les autres threads.

Le calcul de l'adresse en mémoire d'un voxel v d'indices $(vx; vy; vz)$ est donné par

$$Adresse(v) = (vx + vy \times (NX) + vz \times (NX \times NY)) \times sizeof(v) + origine$$

La taille du tableau, fixe, est donnée par

$$T = N \times sizeof(v)$$

Le principal inconvénient de cette structure de données est que les voxels transparents occupent autant de place en mémoire que les voxels opaques. Cette structure est donc peu compacte.

3.5.2.2 Octree

Cette structure de données hiérarchique a été présentée en détail dans la section 1.3.3.2. Son principal avantage est une occupation mémoire moindre, comparativement au tableau 3D présenté précédemment. En effet, un grand espace uniforme (opaque ou transparent) dans la scène sera représenté par une seule feuille et donc une seule valeur.

Ses principaux défauts sont le temps d'accès à un élément et l'interdépendance des données entre noeuds appartenant à une même branche. L'accès à une feuille nécessite en effet de parcourir les noeuds depuis la racine, puisque les données ne sont pas stockées de manière uniforme et ne sont pas directement accessibles. Cela nécessite donc beaucoup plus d'accès en mémoire que dans le cas du tableau 3D présenté précédemment. Par exemple, pour un octree d'ordre 8, il peut être nécessaire, dans le pire des cas, d'effectuer huit sauts d'adresse (et donc huit accès en lecture) pour atteindre une feuille.

L'autre problème majeur apparaît lorsque la structure est modifiée. La modification d'un élément peut nécessiter la mise à jour de l'ascendant ou la création des descendants. Les données d'une branche sont ainsi liées les unes des autres, et ce type de dépendance peut impacter l'exécution parallèle du programme.

La taille en mémoire d'un octree dépend du nombre de voxels subdivisés à chaque niveau et ne peut être connue à l'avance. Le pire cas peut être facilement calculé mais ne représente pas une situation réaliste.

TABLE 3.1 – Comparaison Octree et Tableau 3D

	Tableau 3D	**Octree**
Contenu d'une cellule	Couleur / État (32 bits)	Feuille : Couleur / État (32 bits) Noeud : 8 pointeurs (256 bits)
Accès à une cellule	Direct (indice 3D)	Parcours récursif depuis la racine
Temps d'accès	Constant	Proportionnel à la profondeur (différent pour chaque élément)
Modification d'une cellule	Directe	Peut nécessiter MàJ de tous les éléments de la branche
Compacité	Mauvaise	Bonne

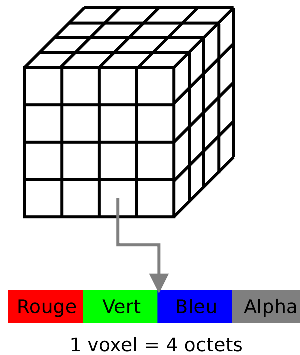


FIGURE 3.5 – Structure de données pour la représentation de la scène : un tableau 3D. Chaque voxel stocke une couleur (R,G,B) et un état (Alpha)

3.5.2.3 Synthèse

Les deux structures présentées précédemment présentent chacune des avantages pour notre application. Le tableau 3.1 offre une synthèse de ces caractéristiques.

Dans le cas de notre application, la donnée à stocker pour chaque voxel étant petite (4 octets), il semble préférable de privilégier l'efficacité d'accès et de manipulation de la structure au détriment de la compacité. Nous avons donc choisi d'utiliser un tableau régulier 3D comme structure de donnée.

3.6 Description détaillée

Dans cette section, nous allons maintenant présenter en détail la méthode de reconstruction 3D que nous proposons. Les différentes opérations du processus seront détaillées, et nous étudierons les différentes alternatives possibles pour certains points de l'algorithme

ainsi que les méthodes pour réaliser une implémentation massivement parallèle efficace.

Le programme que nous avons mis au point fonctionne en majeure partie sur un GPU. Ce processeur n'étant pas capable d'accéder directement aux données stockées sur le disque dur ainsi qu'aux interfaces réseaux, ces tâches seront confiées au CPU du système qui sera aussi chargé de gérer l'interface utilisateur. Le programme commence tout d'abord par exécuter les tâches d'initialisation :

- Initialisations globales (librairies graphiques, communication avec les caméras. . .)
- Chargement des données (paramètres de calibration, paramètres de reconstruction. . .)
- Transfert des données vers la mémoire de constantes du GPU

Une fois ces tâches effectuées, le programme entre dans la boucle principale chargée de reconstruire incrémentalement le modèle 3D de la scène. Cette boucle est effectuée jusqu'à ce que l'utilisateur y mette fin.

- Chargement d'une paire d'image depuis le disque dur (séquence stockée) ou depuis les caméras Ethernet (séquence temps-réel)
- Transfert des images dans la mémoire du GPU
- **Lancement du kernel (programme sur GPU) de reconstruction 3D**
- Si nécessaire, transfert d'une partie du modèle 3D depuis la mémoire du GPU pour affichage
- Recommencer avec la prochaine paire d'images

Le processus central de ce programme est le kernel de reconstruction 3D qui s'exécute sur le GPU. Il est instancié une fois par voxel composant le volume de reconstruction. Cette fonction réalise les opérations suivantes :

- Calcul de l'indice du voxel à traiter
- Calcul des coordonnées dans la scène du voxel à traiter
- Estimation de la visibilité du voxel traité
- Calcul des projections du voxel sur les images
- Lecture des pixels correspondant à la projection du voxel
- Calcul d'un indice de photoconsistance
- Mise à jour de l'état et de la couleur du voxel en fonction
  - de l'indice de visibilité
  - de l'indice de photoconsistance
  - de l'état précédent du voxel

Dans la suite de cette section, nous allons détailler les différentes étapes de ce kernel de reconstruction 3D.

3.6.1 Calcul de l'indice du voxel

Comme présenté dans la section 2.8, le GPU possède un modèle de programmation et d'exécution SIMD. Un programme unique appelé kernel est instancié de nombreuses fois en parallèle pour effectuer les traitements. Pour que chaque thread soit en mesure de traiter les données le concernant, il doit calculer l'adresse de ces données à partir de son numéro de thread.

La première tâche du thread est donc de lire son numéro de thread (threadIdx), son numéro de bloc (blockIdx) ainsi que la taille des blocs (blockDim) et la taille de la grille

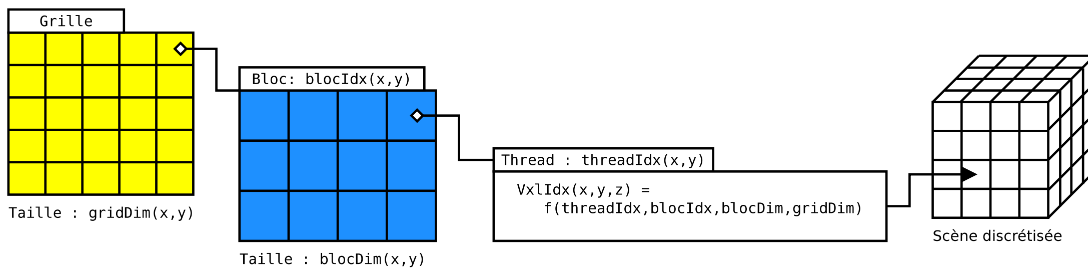


FIGURE 3.6 – Calcul de l'indice d'un voxel traité par un thread

(gridDim) et d'en déduire l'indice 3D du voxel à traiter. La variable threadIdx est un vecteur en 1,2 ou 3D indiquant l'indice du thread actuel dans le bloc courant. La variable blockIdx est un vecteur en 1 ou 2D indiquant l'indice du bloc courant dans la grille.

Dans notre implémentation, nous utilisons une grille de threads 2D constituée de blocs eux aussi en 2D. L'indice complet d'un thread donné est ainsi constitué de deux indices 2D et il faut en dériver l'indice 3D du voxel à traiter. Ces choix ont été effectués car les dimensions sont limitées : $65535 \times 65535 \times 1$ bloc maximum pour la grille et 512 threads par bloc.

L'opération est effectuée en deux étapes, tout d'abord une adresse 1D est calculée à partir des indices du thread, puis cet indice sert à calculer un indice 3D permettant d'adresser le voxel dans le tableau 3D.

Le listing 3.1 présente l'algorithme (en C) utilisé pour effectuer cette opération. Le type de donnée `uint3` est un vecteur de trois entiers non signés `x, y, z`.

Listing 3.1 – Algorithme de calcul de l'indice du voxel

```
// Calcule un indice 1D à partir de
// - l'indice du bloc de thread dans la grille (blockIdx, 2D)
// - l'indice du thread dans le bloc (threadIdx, 2D)
// - la taille de bloc (blockDim, 2D)
// - la taille de grille (gridDim, 2D)
uint vxl_idx1D = blockIdx.y * (blockDim.x*blockDim.y*gridDim.x)
               + blockIdx.x * (blockDim.x*blockDim.y)
               + threadIdx.y * blockDim.x
               + threadIdx.x;

// Calcule un indice 3D à partir de
// - l'indice 1D ci-dessus
// - la taille de la grille de voxels (vxlgridDim)
uint tmp = vxl_idx1D;
uint3 vxl_idx3D;

vxl_idx3D.z = tmp / (vxlgridDim.x*vxlgridDim.y);
       tmp -= vxl_idx3D.z * (vxlgridDim.x*vxlgridDim.y);
vxl_idx3D.y = tmp / (vxlgridDim.x);
       tmp -= vxl_idx3D * vxlgridDim.x;
vxl_idx3D.x = tmp;
```

Après avoir exécuté cette procédure, le thread courant connaît l'indice 3D du voxel qu'il doit traiter dans le modèle 3D de la scène.

3.6.2 Calcul des coordonnées du voxel

Après avoir calculé l'indice du voxel considéré, l'étape suivante consiste à déterminer les coordonnées du centre de ce voxel dans le référentiel monde attaché à la scène. Ce point sera ensuite projeté sur l'image et il est donc nécessaire de connaître sa position dans le référentiel caméra.

Les caractéristiques de la scène (taille du volume d'intérêt, nombre de voxels) sont nécessaires à cette opération, ainsi que l'indice du voxel considéré et la position de la caméra dans le référentiel monde.

La position du système de vision dans le référentiel monde est supposée connue et fournie à l'algorithme de calcul en même temps que les images acquises. Dans nos expérimentations, cette valeur était connue ou mesurée manuellement ; dans une application réelle, elle serait fournie par un système externe.

3.6.3 Estimation de la visibilité

L'étape suivante estime la visibilité du voxel considéré. Estimer la visibilité d'un voxel consiste à déterminer s'il est visible par les caméras depuis leur position actuelle. Pour calculer précisément cette visibilité, il est nécessaire de connaître le modèle 3D complet de la scène afin de pouvoir déterminer si un élément de la scène ne cache pas le voxel considéré.

Cependant, le modèle 3D de la scène étant inconnu, nous devons nous baser sur une approximation pour estimer la visibilité. Dans ce cas, estimer la visibilité d'un voxel revient à étudier si des voxels ont été marqués opaques sur la ligne de vue entre le voxel considéré et les caméras dans le modèle actuel de la scène.

Dans le cas où de tels voxels sont trouvés, le voxel considéré n'est pas visible, ce voxel ne sera donc pas mis à jour. Si, au contraire, aucun voxel opaque n'est trouvé sur la ligne de vue, le voxel n'est pas caché derrière un autre objet de la scène et peut donc être reconstruit.

En réalité, comme cette estimation de visibilité est elle-même basée sur une estimation de la géométrie de la scène, notre méthode n'effectue pas un choix binaire indiquant si le voxel est visible ou non. A la place, notre algorithme calcule un « indice de visibilité » indiquant si le voxel considéré est « plus ou moins » visible. Ceci permet de limiter les problèmes lorsque la visibilité est estimée à partir d'un modèle 3D imprécis.

Lors du calcul, l'indice de visibilité actuellement calculé pour le voxel considéré est comparé à l'indice de visibilité qu'avait ce même voxel lorsqu'il a été mis à jour pour la dernière fois. Si l'indice de visibilité actuel est supérieur à celui stocké dans la grille de voxel, alors le voxel est actuellement mieux vu que la dernière fois qu'il a été mis à jour. Il sera donc mis à jour et contiendra une nouvelle information de couleur et de visibilité. Si l'indice de visibilité est inférieur, le voxel est moins visible depuis la position actuelle des caméras et ne sera donc pas mis à jour. Cet indice de visibilité est stocké dans le modèle

3D de la scène, pour chaque voxel, dans l'octet surnuméraire évoqué dans la section 3.5.2 et inutilisé jusqu'à maintenant.

Cette section présente les différentes méthodes d'estimation de la visibilité que nous avons envisagées ou réalisées afin d'effectuer cette opération de manière optimale sur notre processeur massivement parallèle.

3.6.3.1 Ligne de vue 3D

Pour estimer la visibilité pour un voxel donné, le moyen le plus intuitif et direct dans notre cas est de chercher s'il y a un ou plusieurs voxels opaques sur le segment Caméra-Voxel.

Si tous les voxels situés sur ce segment sont transparents, alors le voxel considéré est visible. Dans le cas contraire (ou moins un voxel opaque), le voxel considéré est caché et ne peut être vu par les caméras. Ce processus doit être répété deux fois par voxel (une fois pour chaque caméra). Pour calculer l'indice de visibilité évoqué précédemment, le nombre de voxels transparents (N) sur le segment est utilisé avec la formule suivante

$$I_{visibilité} = 255 - N_{cam0} - N_{cam1}$$

Le parcours du segment 3D Caméra-Voxel est effectué en utilisant l'algorithme de Bresenham en 3D présenté dans l'annexe B et chaque voxel traversé est testé pour déterminer s'il est opaque ou transparent. Cette méthode présente donc le problème de nécessiter de nombreux accès en lecture sur un grand nombre de voxels qui ne sont pas forcément proches du voxel considéré. Ces accès mémoire sont donc aléatoires, ce qui pénalise fortement le temps d'exécution.

Cette méthode a été expérimentée mais le fort impact négatif sur les performances nous a incité à explorer d'autres voies.

3.6.3.2 Carte d'occupation

Dans le but d'améliorer le temps de calcul de notre estimateur de visibilité, nous avons développé une méthode alternative. Considérant le fait que, dans une scène réelle, la plupart des objets sont posés sur le sol, il est possible d'estimer les zones occupées ou libres de la scène 3D en utilisant une carte 2D (carte d'occupation), établie en projetant les voxels opaques du modèle 3D sur le sol. Ensuite, une carte de visibilité est établie en prenant en compte la position actuelle des caméras et permet d'estimer la visibilité d'un voxel pendant la phase de reconstruction.

La figure 3.7 illustre le processus de calcul de la carte de visibilité. Des exemples de cartes d'occupation et de visibilité sont donnés dans la figure 3.8.

Ce processus fonctionne ainsi en trois étapes. Tout d'abord, en utilisant le modèle 3D de l'étape précédente, la carte d'occupation est calculée. Pour cela, un kernel spécifique est implémenté en GPU et instancié pour chaque colonne de voxel dans la reconstruction. Ainsi, pour une reconstruction de $500 \times 500 \times 200$ voxels, le kernel est lancé 500×500 fois. Ce kernel est chargé de compter le nombre de voxels opaque dans une colonne verticale de voxels et d'écrire cette information dans la carte d'occupation.

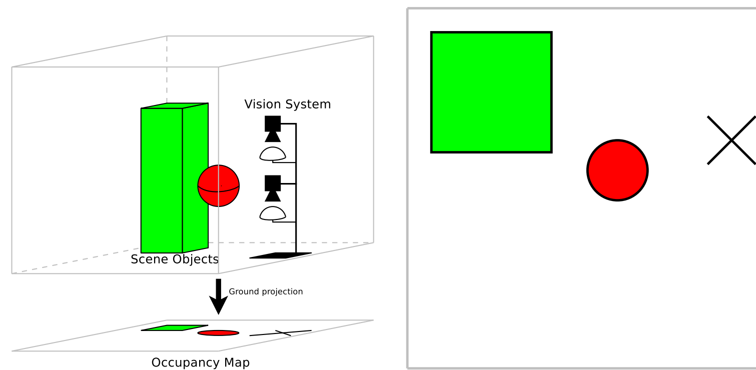


FIGURE 3.7 – Illustration du processus de calcul de la carte d'occupation.
A gauche : calcul de la carte d'occupation à partir du modèle 3D de la scène.
A droite : carte d'occupation résultante (la croix représente la position du capteur).

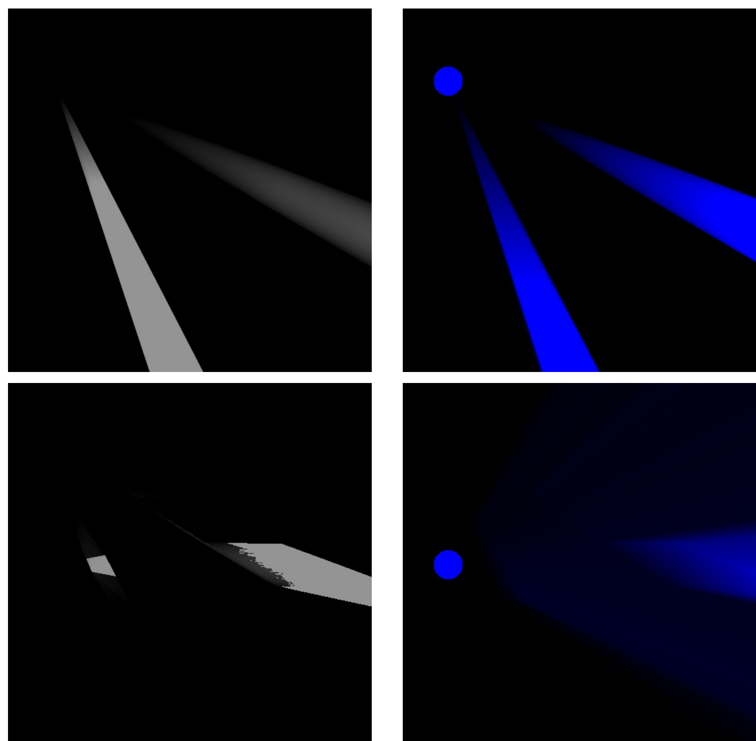


FIGURE 3.8 – Exemples de cartes d'occupation (à g.) et de visibilité (à d.)
A gauche : les zones claires représentent les zones occupées de la scène (plusieurs voxels opaques sur une colonne). Les zones sombres sont inoccupées.
A droite : grâce à la carte d'occupation, il est possible de déterminer si une zone de la scène est visible par la caméra (le cercle bleu représente sa position). Les zones noires sont visibles ; les zones bleu sombre sont partiellement visibles ; le bleu pur indique une zone occultée.

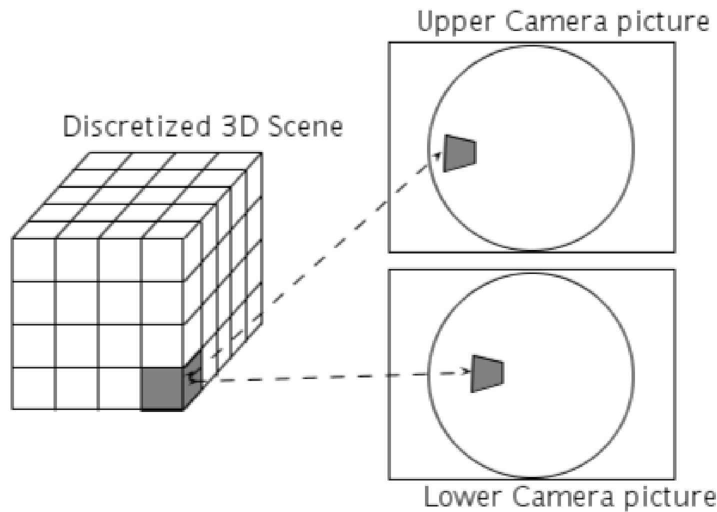


FIGURE 3.9 – Projection d'un voxel de l'espace 3D sur les images (principe).

La deuxième étape consiste, à partir de la carte d'occupation et de la nouvelle position des caméras, à calculer si chaque voxel est visible. Pour cela, un autre kernel est chargé de parcourir, sur la carte d'occupation, le segment entre le voxel considéré et la position actuelle des caméras, projetée sur cette carte. La recherche est bien entendue effectuée en 2D sur la carte, le résultat de visibilité sera ainsi identique pour tous les voxels appartenant à une même colonne verticale dans le modèle 3D. Le score de visibilité est obtenu en comptant le nombre de cellules de la carte d'occupation comportant une valeur supérieure à un seuil le long du segment parcouru. Ce score est inscrit dans la carte de visibilité.

Enfin, lors de la reconstruction, pour chaque voxel considéré, il suffit de lire la carte de visibilité aux coordonnées correspondant à ce voxel pour obtenir l'information de visibilité.

Cette méthode ne fournit bien sûr qu'une indication imprécise de la visibilité réelle du voxel, mais elle permet néanmoins d'améliorer grandement la qualité du modèle reconstruit tout en limitant le coût en calculs par rapport à un parcours complet en 3D du modèle de la scène.

Concernant l'implémentation, les cartes d'occupation et de visibilité sont stockées dans les voxels appartenant à l'une des tranches horizontales du modèle 3D, réservés à cette effet.

3.6.4 Projection 3D vers 2D

L'étape suivante consiste à calculer la projection du voxel considéré sur les images, connaissant les paramètres des caméras et la position de la caméra dans la scène. Nous utilisons un modèle central de la caméra, appelé modèle de la sphère d'équivalence développé par [MR07] et présenté dans l'annexe A. Les paramètres de ce modèle ayant été préalablement identifiés par une opération de calibration, nous sommes en mesure de calculer la projection sur l'image d'un point quelconque de la scène à partir de ses coordonnées 3D.

Le voxel à projeter est approximé par son centre de gravité. La projection de ce point sur les deux images issues des caméras est donc calculée en utilisant les paramètres

obtenus lors du calibrage des caméras. Les projections sont calculées avec une précision subpixellique qui sera exploitée lors de la lecture des images.

Ces calculs sont rapides, ne mettant en oeuvre que quelques équations simples. Les constantes nécessaires pour ces calculs sont stockées dans la « mémoire de constantes » du GPU et sont ainsi rapidement accédées par chaque thread. Les résultats sont temporairement conservés dans des registres et ne sont utilisés que pour lire le pixel correspondant sur les images.

Les images sont quant à elles conservées dans la « mémoire globale » du GPU et utilisent une « unité de texture » pour y accéder. Outre un cache des données lues, cette unité permet un accès subpixellique direct (les coordonnées sont interpolées) mais aussi un résultat interpolé bilinéairement. Ces opérations étant réalisées par une unité matérielle, elles sont sans impact négatif sur le temps d'accès.

Après lecture, les couleurs des projections du voxel considéré, constituées de vecteurs à 3 composantes R,G,B sont conservées dans des registres pour les opérations suivantes.

3.6.5 Projection d'un voxel

Un voxel n'est pas un point sans dimension, il est donc possible de prendre en compte différentes méthodes pour approcher au mieux sa projection sur l'image sans pour autant ralentir le calcul.

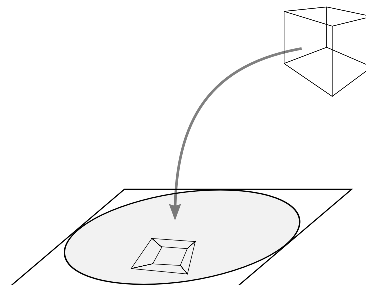


FIGURE 3.10 – Projection d'un voxel sur le plan image.
Quelle forme choisir pour approximer au mieux la surface couverte par la projection ?

Selon les paramètres de discrétisation du modèle 3D (taille de la scène et nombre de voxels), la taille d'un voxel peut varier. Si le voxel est suffisamment petit pour être considéré infinitésimal, la projection d'un des points du voxel sur l'image est suffisante pour estimer correctement sa projection. Cependant, si la résolution est faible ou le voxel proche de la caméra, celui-ci ne peut plus être assimilé à un point et il est nécessaire de calculer (ou au moins d'estimer) la surface de l'image sur laquelle il se projette.

Plusieurs méthodes de projection sont envisageables, selon le niveau de précision et de complexité désiré. Dans [SGEB00], l'auteur présente plusieurs méthodes de projections et leur impact sur la qualité d'un modèle reconstruit par *voxel coloring*.

3.6.5.1 Projection du centre

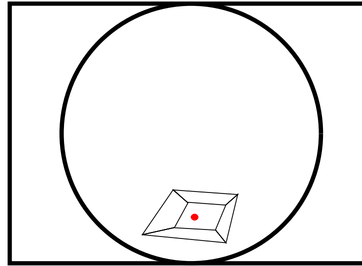


FIGURE 3.11 – Approximation de la projection d'un voxel par celle de son centre

La première approximation de la projection d'un voxel consiste simplement à calculer la projection de son centre. Cette technique est la plus facile à mettre en oeuvre. C'est aussi la plus rapide à l'exécution puisqu'elle ne nécessite qu'un seul calcul de projection et une seule lecture de l'image par voxel. Enfin, elle est précise si la taille apparente d'un voxel sur l'image est d'environ un pixel. Dans le cas contraire, il en résulte un sous-échantillonnage. Un point intéressant de cette méthode est que la projection est indépendante de l'orientation relative du voxel et de la caméra.

La figure 3.11 illustre la projection du centre d'un voxel sur une image panoramique. L'exemple présenté se place dans le cas où le voxel se projette sur une très grande surface de l'image, ce qui ne concerne que les voxels situés très près de la caméra. C'est cette méthode que nous avons majoritairement utilisée, en particulier pour sa vitesse d'exécution.

3.6.5.2 Projection exacte

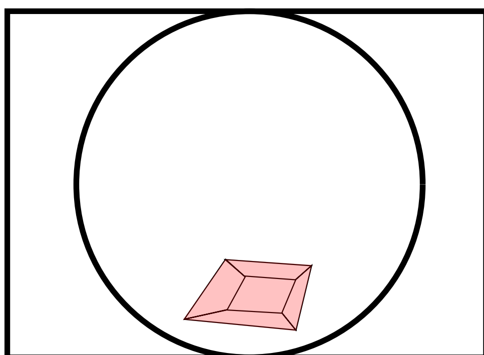


FIGURE 3.12 – Projection exacte du voxel

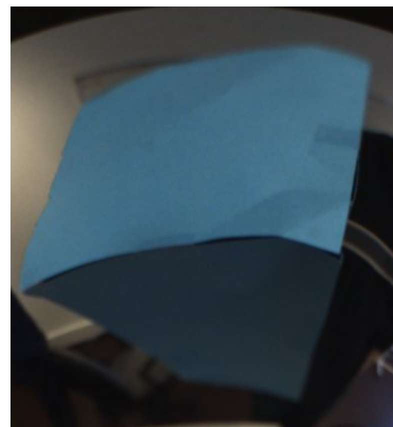


FIGURE 3.13 – Cube observé par une caméra catadioptrique

La deuxième solution évidente pour l'estimation de la projection d'un voxel est le calcul de sa projection exacte. Il est pour cela nécessaire de projeter ses 8 coins et de

déterminer le polygone convexe englobant ces 8 points sur le plan image. Cette solution est la plus précise car elle prend en compte tous les pixels de l'image appartenant à la projection du voxel. C'est aussi la plus coûteuse en calculs car, outre les 8 projections nécessaires, elle nécessite le calcul du polygone englobant. Ce calcul est relativement aisé lorsque une caméra perspective classique est utilisée, mais il devient complexe dans le cas de nos caméras catadioptriques à cause de la forme distordue de la projection.

La précision apportée par ce type de projection est nécessaire lorsque la taille de la projection d'un voxel est grande sur l'image (plusieurs dizaines de pixels). Pour des tailles inférieures, l'échantillonnage de l'image vient perturber le calcul du polygone et limite donc son intérêt.

La figure 3.12 illustre la projection « exacte » d'un voxel sur une image panoramique. Nous remarquons la forme complexe du polygone sur l'image, ce qui complique et ralentit les calculs. A titre de comparaison, la figure 3.13 présente l'image d'un cube en papier coloré vu par une caméra catadioptrique. Nous remarquons les fortes distorsions.

Nous n'avons pas utilisé cette méthode lors de nos expérimentations car, outre son coût élevé en calculs, elle offre un degré de précision bien supérieur à celle nécessaire dans notre application : les voxels ne sont qu'une approximation de la géométrie de la scène, les projeter de façon précise n'est pas indispensable.

3.6.5.3 Rectangle englobant

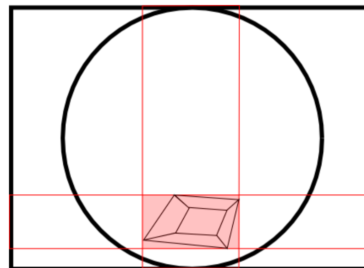


Figure 3.14 – Approximation de la projection d'un voxel par un rectangle englobant

Une solution intermédiaire entre les deux solutions précédentes consiste à calculer le rectangle englobant les 8 projections des 8 coins du voxel. Le rectangle étant aligné sur les lignes et colonnes de pixels de l'image, son calcul est rapide et le parcours de sa surface (lecture de tous les pixels inclus) est efficace. Cette solution reste plus coûteuse en calculs que la projection du centre, mais elle permet de tenir compte de la taille du voxel considéré. Sur la figure 3.14, nous observons que le rectangle englobant est toujours plus grand que la surface réelle du voxel, ce qui introduit dans la projection des pixels n'appartenant pas au voxel.

Cette méthode a été implémentée lors de nos premiers tests sur CPU. Le coût en calcul est cependant prohibitif et un trop grand nombre de pixels n'appartenant pas au voxel sont pris en compte, faussant ainsi le test de photoconsistance des voxels situés sur le bord d'un objet. C'est pourquoi cette méthode a été abandonnée par la suite, mais ses points forts ont été repris dans la méthode suivante.

3.6.5.4 Échantillonage de points

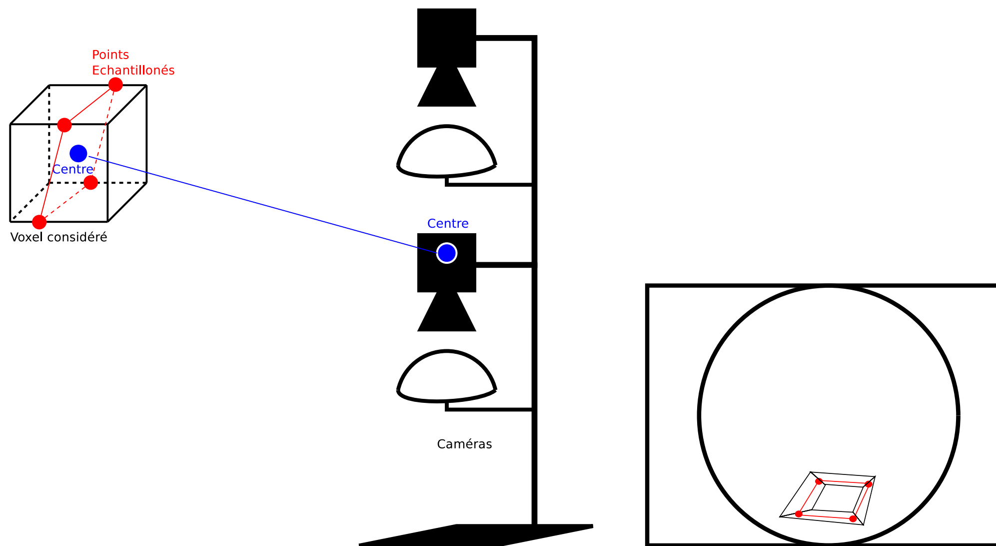


FIGURE 3.15 – Approximation de la projection d'un voxel par échantillonnage de quatre points

Pour pouvoir obtenir plusieurs échantillons de couleur par voxel, et offrir une certaine robustesse au bruit sur l'image par rapport à la méthode « projection du centre », il est envisageable de choisir plusieurs points intermédiaires appartenant au voxel et de calculer leur projections sur les images. Nous avons implémenté une méthode choisissant 4 points répartis sur une surface passant par le centre du voxel et orientée perpendiculairement à la ligne de vue des caméras.

Tout d'abord, le point situé à mi-distance des ceux caméras est utilisé comme référence pour la position des caméras. Le vecteur $\overrightarrow{Voxel\,Caméra}$ est calculé en utilisant cette référence. Un plan orthogonal à ce vecteur et passant par le centre de gravité du voxel est déterminé. Les quatre points projetés sont choisis sur ce plan, répartis autour du centre à une distance équivalente à la largeur du voxel. La figure 3.15 présente la méthode de sélection des points à échantillonner et leur position sur l'image. Ces quatre points 3D sont ensuite projetés sur chacune des images, et les tests de photoconsistance sont effectués sur les quatre paires de projections.

Ne nécessitant que quatre projections et quatre lectures de pixels par image, cette solution présente un coût en calculs similaire à la projection du centre du voxel. Elle permet cependant une plus grande fiabilité du test de photoconsistance. Une expérimentation comparant les résultats de cette méthode par rapport à la projection du centre de gravité du voxel est donnée dans la section 4.4.

3.6.6 Mesure de la photoconsistance

Pour comparer les couleurs des projections d'un voxel, il est possible de procéder de plusieurs manières. Quelques fonctions de photoconsistance ont déjà été présentées dans

la section 1.4.2.3, ici sont présentées les différentes méthodes que nous avons utilisées durant notre étude.

3.6.6.1 Distance seuillée

Le moyen le plus simple de comparer deux couleurs consiste tout simplement à calculer la distance entre ces deux couleurs. Comme une couleur est représentée par un vecteur à 3 dimensions (RGB, YUV, Lab, xyY... selon l'espace de couleur utilisé), une distance euclidienne peut être utilisée. Un voxel est ensuite déclaré photoconsistant si la distance entre la couleur de ses deux projections est inférieure à un seuil fixé à l'avance.

Cette méthode n'est bien sûr valable que pour 2 échantillons, obtenus en projetant un voxel (ou son centre) sur deux images. Elle est très sujette au bruits sur l'image et aux différences de colorimétrie entre les deux caméras. Le seuil est aussi un point faible de cette méthode et nécessite un ajustement pour chaque scène.

Un dernier point à considérer est la notion de distance entre les couleurs, qui peut être biaisé par l'espace de couleur utilisé. Par exemple, l'espace RGB n'est pas perceptuellement uniforme. Dans le cas d'un test de photoconsistance cependant, et comme seules les très faibles distances sont prises en compte, ceci n'est pas un obstacle majeur.

Notre méthode de reconstruction 3D utilise en majorité le calcul d'une distance en espace RGB pour déterminer si deux couleurs sont semblables ou non, comparé à un seuil fixé à l'avance.

3.6.6.2 Variance

Lorsqu'un plus grand nombre de caméras observe la scène (au minimum 3), il est possible d'utiliser la variance comme critère de photoconsistance. Cette mesure est beaucoup plus robuste face aux bruits car elle réalise un filtrage de type passe-bas en prenant en compte les multiples sources d'images.

C'est ce type de calcul qui est le plus souvent utilisé dans les méthodes volumétriques classiques. Il nécessite néanmoins un plus grand nombre d'échantillons de couleur pour produire un résultat significatif et robuste.

3.6.6.3 Conclusion

Notre méthode de reconstruction 3D n'exploitant que deux caméras, la mesure de photoconsistance utilisée est la distance seuillée. Elle présente l'avantage d'être rapide à calculer et de ne nécessiter qu'un échantillon de couleur par projection, et donc deux accès mémoire en lecture par voxel.

Le choix du seuil est effectué empiriquement pour chaque scène.

3.6.7 Détermination de l'état d'un voxel

Pour déterminer le nouvel état d'un voxel, les données suivantes sont utilisées :

- Indice de visibilité (détails section 3.6.3)
- Indice de photoconsistance (détails section 3.6.6)
- État précédent du voxel (couleur et indice de visibilité)

Ces données sont ensuite comparées afin de déterminer si le voxel considéré doit être mis à jour et dans ce cas, sa nouvelle valeur.

Le voxel doit être mis à jour si l'indice de visibilité actuel est supérieur à l'indice de visibilité conservé dans le voxel. Dans ce cas, il y a moins de voxels opaques pouvant potentiellement faire obstacle au voxel considéré et donc il est « mieux vu » selon la position actuelle des caméras. Dans ce cas, son nouvel état qui vient d'être calculé est plus pertinent que celui anciennement stocké et donc, le voxel est mis à jour.

L'indice de photoconsistance fournit la nouvelle valeur du voxel considéré : si le voxel est consistant, il sera marqué opaque et la couleur moyenne des deux projections lui sera attribuée. S'il n'est pas consistant, le voxel sera marqué transparent en utilisant une valeur spéciale pour la couleur : $(0, 0, 0)$. L'indice de visibilité du voxel sera quant à lui mis à jour avec la nouvelle valeur de visibilité.

Cette méthode pour la mise à jour des voxels permet de résoudre plusieurs problèmes. Tout d'abord, lors de l'initialisation, tous les voxels sont considérés comme ayant été marqués transparents avec un indice de visibilité de 0. Ainsi, tous les voxels du volume seront mis à jour en utilisant uniquement les informations de photoconsistance lors de la première acquisition. Lors de cette première itération cependant, les informations de visibilité ne sont pas encore disponibles (puisque calculées en fonction du modèle précédent de la scène qui n'existe pas encore). L'indice de visibilité est donc renseigné avec la valeur 1 qui permettra par la suite une mise à jour correcte, une fois que les informations de visibilité auront pu être calculées.

Ensuite, au fur et à mesure de la prise en compte des nouvelles vues de la scène, chaque voxel qui est mieux vus que sur les étapes précédentes est mis à jour avec les informations issues des dernières images de la scène. Le modèle 3D de la scène est ainsi progressivement amélioré au fur et à mesure de l'exploration.

3.7 Conclusion

La méthode de reconstruction 3D que nous proposons ici, inspirée par les méthodes volumétriques basées sur la photoconsistance, a été développée avec plusieurs objectifs en vue.

Tout d'abord, pour obtenir beaucoup d'informations sur la scène environnante en un temps très bref, une paire de caméras panoramiques est utilisée. Ces caméras catadioptriques composées d'un miroir de révolution observé par une caméra industrielle haute résolution permettent d'acquérir une image panoramique et stéréoscopique des environs du mobile en une seule prise de vue. Un modèle de projection central permet de calculer rapidement la projection d'un point quelconque de la scène sur le plan image.

Ensuite, pour être en mesure de calculer rapidement un modèle 3D de la scène, un algorithme de reconstruction adapté à l'architecture massivement parallèle d'un GPU est mis en oeuvre. La méthode a été pensée depuis le début pour tirer partie d'une répartition parallèle des calculs. Le modèle de la scène, basé sur une structure volumétrique constituée d'une grille régulière de voxels, permet une répartition de la charge de calcul à grande échelle : plusieurs milliers de thread sont instanciés en même temps, permettant ainsi une exploitation efficace de l'architecture de calcul utilisée.

Les différents éléments constituant ce processus de calcul ont été adaptés à l'architecture et optimisés pour une exécution rapide. Un soin particulier a été apporté aux accès aux différentes données, qui constitue souvent un point bloquant ou limitant pour atteindre des performances maximales dans les applications parallèles. Ici, notre méthode de calcul est en majeure partie indépendante pour chaque voxel, permettant ainsi une exécution sans interblocages dûs aux accès concourant sur les données.

Notre implémentation fait également usage des différents éléments matériels permettant une accélération de certaines portions du programme : la mémoire de constantes et les unités de textures. Alors que la première optimise les accès concourant en lecture seule des milliers de thread vers les données globales du programme, la deuxième permet d'effectuer les opérations d'interpolation bilinéaire sur les images « gratuitement » (en terme de temps de calcul).

Tous ces éléments réunis nous permettent de proposer une méthode de reconstruction efficace et rapide dont les résultats et les performances seront évaluées en détail au chapitre 4.

Chapitre 4

Résultats

Sommaire

4.1 Validation de la photoconsistance 87
4.1.1 Reconstruction 3D 87
4.1.2 Discussion 88
4.1.3 Conclusion 91
4.2 Estimation de la visibilité et reconstruction incrémentale 91
4.2.1 Reconstruction 3D 92
4.2.2 Temps de calcul 93
4.2.3 Discussion 94
4.3 Influence des textures 94
4.3.1 Objets texturés 95
4.3.2 Arrière-plan uniforme 96
4.3.3 Arrière-plan texturé 98
4.3.4 Suppression du bruit 99
4.3.5 Conclusion sur l'influence des textures 102
4.4 Comparaison des méthodes de projection d'un voxel . . . 102
4.4.1 Temps de calcul brut 103
4.4.2 Reconstruction 3D 103
4.4.3 Synthèse et discussion 104
4.5 Influence du positionnement des caméras 104
4.6 Comparaison avec la méthode « Generalized Voxel Coloring » 105
4.6.1 Reconstruction 3D 106
4.7 Scène complexe 107
4.7.1 Reconstruction 3D 108
4.8 Scène réelle 109
4.8.1 Reconstruction 3D 109
4.9 Répartition des calculs 111
4.10 Profilage du temps d'exécution 113
4.11 Discussion générale 114

4.12 Conclusion 115

Après la présentation détaillée de notre méthode de reconstruction dans le chapitre précédent, nous allons, dans ce chapitre entièrement consacré aux résultats, évaluer cette méthode sur différentes séquences d'images. Deux aspects majeurs sont étudiés ici : la qualité du modèle 3D obtenu et le temps de traitement. Différents types de scène valident les différents éléments composant le système de reconstruction mis au point et évaluent le comportement du système dans différentes situations.

Nous évaluerons tout d'abord le fonctionnement du test de photoconsistance tel que nous l'avons réalisé. La deuxième série de test viendra illustrer le fonctionnement de l'estimation de la visibilité à partir du modèle 3D partiellement reconstruit. Nous étudierons ensuite l'influence de la texture des objets et de l'arrière-plan. Un test montrera le comportement de notre méthode lorsque la position des caméras n'est pas connue de façon fiable. Nous réaliserons une comparaison avec un autre programme de reconstruction 3D implémentant une méthode plus classique.

Une expérimentation utilisant des images de synthèse représentant une scène complexe et une expérimentation sur des images réelles permettront de valider l'approche proposée avec des données représentatives d'une application réelle.

Une discussion générale effectuant une synthèse des éléments mis en avant par ces résultats ainsi qu'une conclusion viendront terminer ce chapitre.

Les résultats présentés ici ont été calculés sur le second PC présenté dans la section 3.3.3, utilisant principalement la carte graphique nVidia GeForce 9800GTX+ embarquant 512 Mio de RAM.

4.1 Validation de la photoconsistance

Cette scène synthétique, constituée d'une salle rectangulaire occupée par trois colonnes colorées à base carrée, a pour but de valider l'approche de reconstruction 3D basée sur la photoconsistance. La couleur des objets de la scène et l'éclairage ambiant permettent de simuler une illumination lambertienne de la scène.

La séquence complète comporte huit paires d'images catadioptriques acquises à huit emplacements connus dans la scène. La figure 4.1 présente quatre de ces paires d'images. Les murs rouges ont été supprimés lors de la reconstruction 3D par une opération de segmentation sur la couleur, afin de permettre une interprétation plus aisée des résultats et notamment de pouvoir observer facilement l'intérieur du modèle 3D reconstruit.

4.1.1 Reconstruction 3D

L'algorithme utilisé pour cette reconstruction est une version préliminaire qui n'intègre pas toutes les étapes présentées dans la section 3.6. Notamment, la notion de visibilité n'est pas prise en compte. De même, cet algorithme servant à valider les fondements de notre méthode de reconstruction, il n'a pas été implémenté sur GPU. Pour limiter les temps de traitement à une durée acceptable, la résolution du modèle 3D utilisée est très faible : $80 \times 80 \times 40\,voxels$. Malgré cela, la reconstruction complète nécessite plus de 20 minutes de calculs sur le premier PC détaillé dans la section 3.3.3.

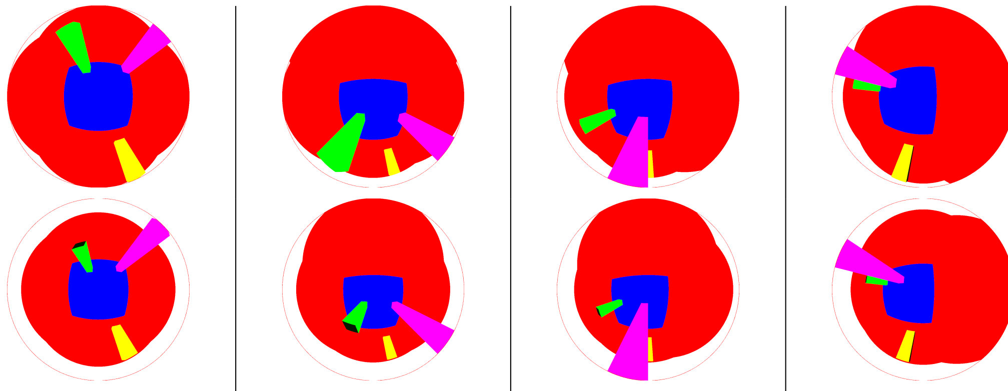


FIGURE 4.1 – Scène « Colonnes Colorées » : Images source
Chaque paire d'image correspond à une position du capteur (image caméra haute au dessus, caméra basse en dessous).

La reconstruction 3D de cette scène est effectuée en deux étapes. Tout d'abord, un modèle 3D est reconstruit pour chaque position du capteur dans la scène en utilisant uniquement l'information de photoconsistance entre les deux caméras. Le résultat est donc un jeu de huit modèles 3D « partiels » dans un référentiel commun.

Le modèle 3D « final » est ensuite calculé à partir de ces modèles partiels. Le processus consiste à étudier, pour chaque voxel composant le modèle final, quels sont les couleurs qui lui ont été affecté dans les différents modèles partiels. La variance de ces couleurs est calculée, et si elle est en deçà d'un seuil fixé à l'avance, le voxel est déclaré consistant et est donc marqué opaque dans le modèle final. Ainsi, un voxel possédant des couleurs très proches dans tous les modèles partiels sera marqué opaque, tandis qu'un voxel ayant une couleur très différente des autres dans au moins une des reconstructions partielles sera marqué transparent.

Cette méthode de reconstruction 3D diffère donc sensiblement de la méthode présentée dans le chapitre 3 et a pour but de valider le test de photoconsistance utilisé. Ce test est en effet utilisé deux fois : une première fois pour créer une reconstruction partielle à partir d'une paire d'images et une seconde fois pour calculer la reconstruction finale à partir des huit reconstructions partielles.

La figure 4.2 présente une vue de coté de la reconstruction 3D partielle obtenue grâce à la première paire d'images. La figure 4.3 quand à elle, présente deux vues du modèle 3D final.

4.1.2 Discussion

Sur la figure 4.2 représentant une reconstruction partielle, le plafond et le sol, de même que les colonnes, sont reconstruits sous forme d'objets coniques pointant vers les caméras. Le test de photoconsistance est en effet très peu discriminant sur les larges surfaces non texturées et parfaitement uniformes présentes dans la scène. Ainsi, un voxel normalement transparent localisé près de la surface d'un élément du décor sera vu d'une couleur similaire par les deux caméras et considéré consistant. Ce premier résultat illustre la limite principale du test de photoconsistance : il est facilement mis en défaut par

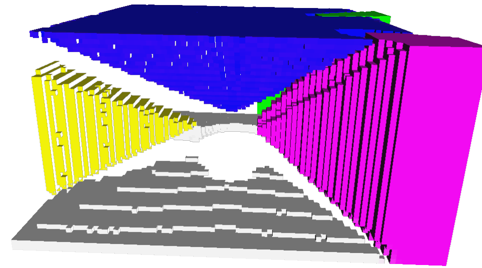


FIGURE 4.2 – Reconstruction partielle effectuée à partir des images de la figure précédente. Les surfaces de couleur homogènes induisent la reconstruction de pyramides pleines.

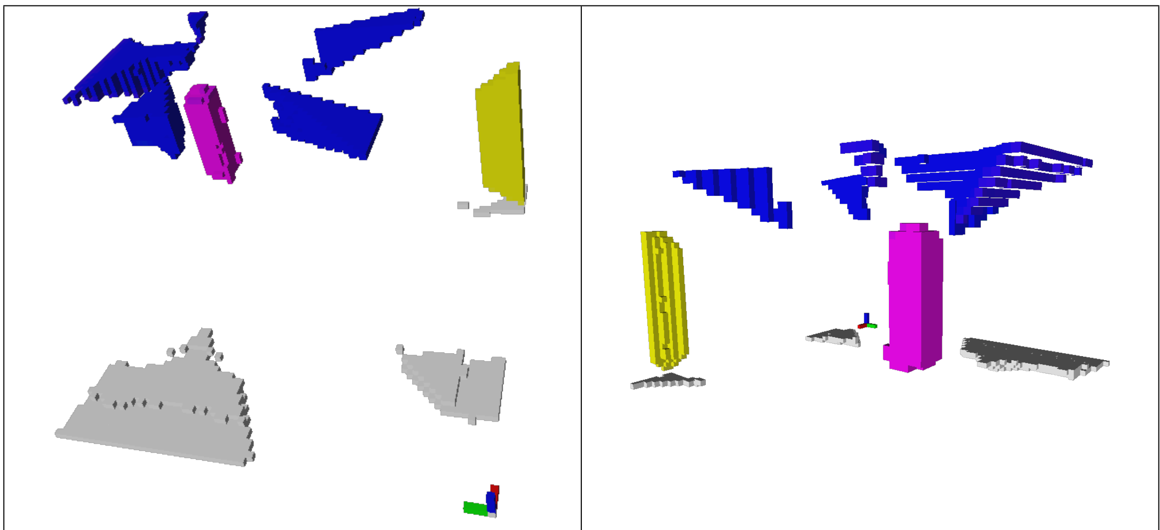


FIGURE 4.3 – Reconstruction complète (deux vues différentes).
Le déplacement du capteur permet la suppression des voxels faussement marqués consistants dans la reconstruction partielle. On remarque l'absence de la colonne verte.

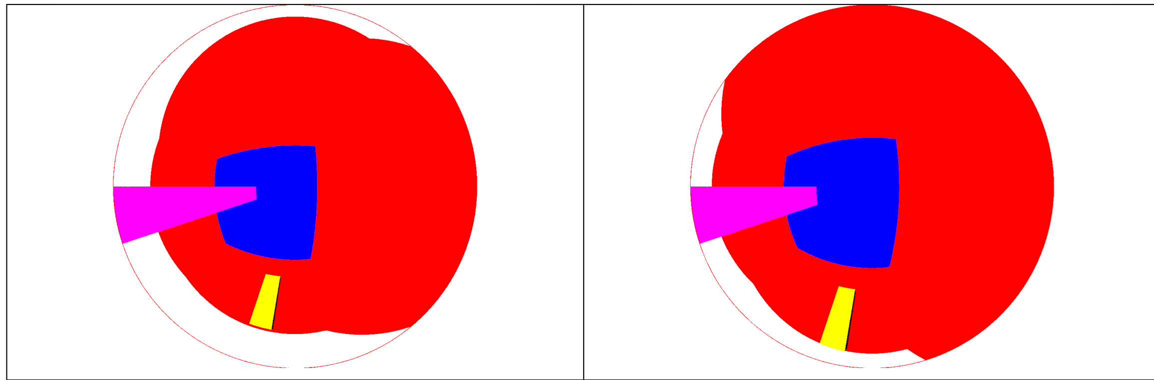


FIGURE 4.4 – Sur cette paire d'images sources, la colonne verte n'est pas visible.

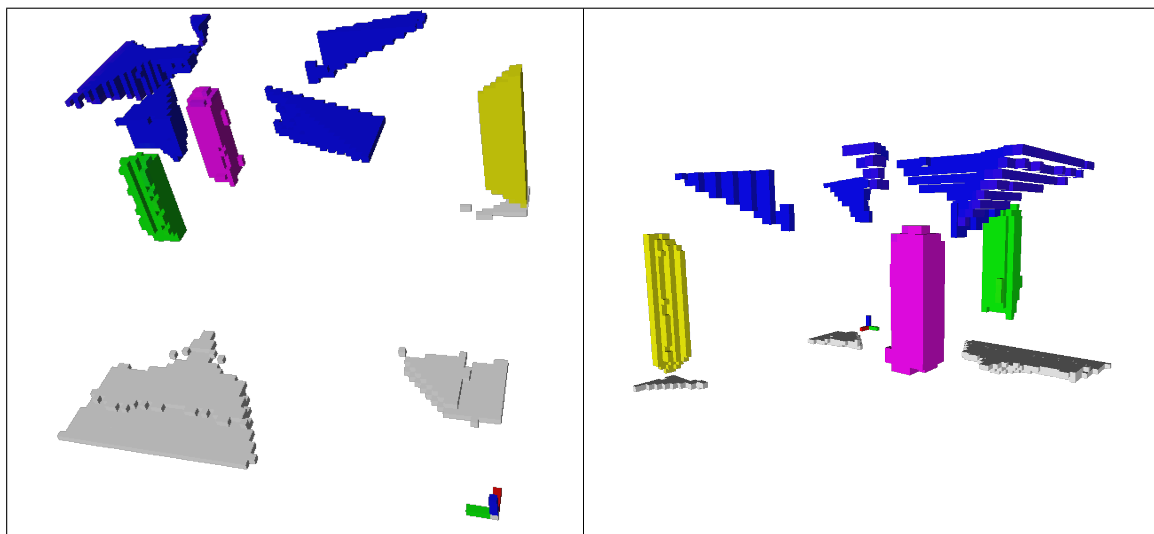


FIGURE 4.5 – Reconstruction complète calculée sans la paire d'image comportant l'occlusion de la colonne verte

des couleurs proches, des textures périodiques ou des objets de couleur uniformes. La détermination du seuil constitue l'autre point faible de cette méthode, mais ne pose pas de problème sur cette scène aux couleurs franches.

Sur les vues du modèle 3D présentées dans la figure 4.3, nous pouvons constater un gros défaut : la colonne verte n'apparaît pas dans le modèle final. Cette « disparition » est due à la méthode simpliste utilisée pour calculer le modèle final à partir des modèles partiels. En effet, comme la variance de la couleur d'un voxel est calculé à partir de tous les modèles partiels, il suffit qu'un seul des modèles soit erroné pour entraîner le marquage « transparent » d'un voxel dans le modèle final. Dans ce cas précis, sur une des positions du capteur, la colonne verte est cachée derrière la colonne rose, comme le montre la paire d'images de la figure 4.4. Le modèle partiel calculé à partir de cette paire d'image ne possède aucun voxel opaque vert et empêche donc cet objet d'être conservé dans le modèle final. Ce deuxième résultat illustre la nécessité de prendre en compte la visibilité d'un voxel avant de calculer sa couleur.

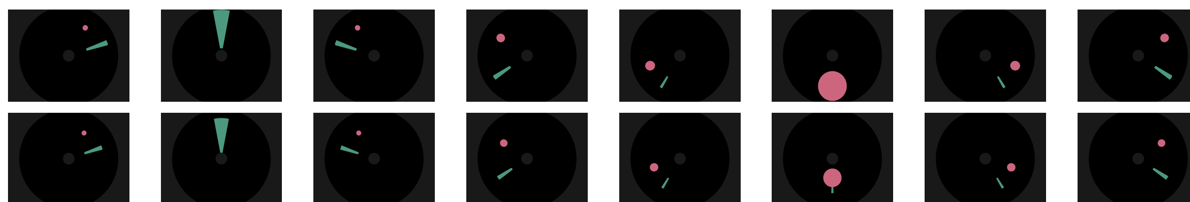


FIGURE 4.6 – Séquence « sphère et colonne » : Images sources

La figure 4.5 présente le modèle final reconstruit lorsque la paire d'images de la figure 4.4 n'est pas utilisée dans le processus de reconstruction.

Dans la reconstruction finale présentée sur la figure 4.5, la forme des colonnes est relativement bien estimée, compte tenu de la faible résolution de la grille de voxels ($80 \times 80 \times 40$ voxels). Le plafond conserve une forme « conique » car il est à chaque fois observé par en dessous.

4.1.3 Conclusion

Cette première scène synthétique, bien que simpliste, nous a permis de valider l'utilisation de la photoconsistance pour opérer une reconstruction 3D d'une scène inconnue à partir de caméras catadioptriques calibrées.

Deux principaux défauts ont été relevés. Tout d'abord, le test de photoconsistance est peu discriminant et échoue facilement à marquer un voxel transparent lorsqu'il est localisé entre les caméras et une surface de couleur uniforme. Ce résultat rejoint les études déjà menées sur ce type d'algorithme dans la littérature. Le test de photoconsistance ne peut donc suffire seul à déterminer avec précision la géométrie de la scène.

Le deuxième défaut principal apparaissant dans cette reconstruction concerne la disparition d'un objet dans la reconstruction finale. Cet objet est en effet marqué transparent car il est complètement caché par un autre objet sur une des paires d'images. Ceci doit être corrigé en prenant en compte la visibilité des voxels pendant le calcul de la reconstruction.

Dans les résultats présentés par la suite, une méthode d'estimation de la visibilité a été utilisée, contrairement à ce résultat.

4.2 Estimation de la visibilité et reconstruction incrémentale

Cette scène en images de synthèse comporte une sphère rouge et une colonne verte dans une scène complètement vide. Ici aussi, l'éclairage simule une illumination lambertienne et l'arrière-plan est ignoré par l'algorithme de reconstruction. La scène complète est constituée de 40 paires d'images panoramiques acquises à mesure que le capteur se déplace dans la scène. La trajectoire du capteur forme un carré qui suit les bords extérieurs de la scène. La figure 4.6 présente 8 paires d'images parmi les 40 qui composent la séquence complète. La séquence présente deux occlusions lorsque les objets se trouvent alignés avec les caméras. La figure 4.7 présente une vue du dessus de la scène. De ce point

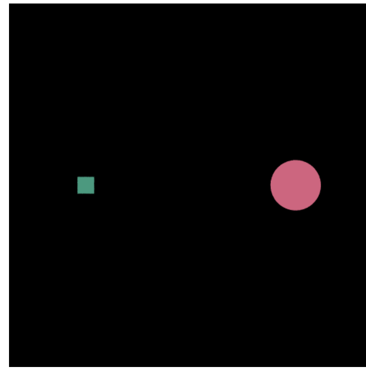


FIGURE 4.7 – Séquence « sphère et colonne » : Vue du dessus

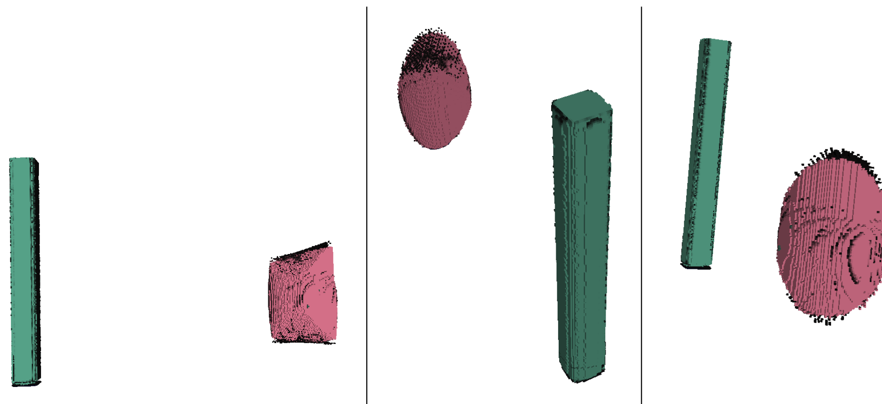


FIGURE 4.8 – Séquence « sphère et colonne » : vues du modèle 3D reconstruit
Le déplacement du capteur ainsi que le bon contraste avant/arrière-plan permettent une très bonne reconstruction.

de vue, la trajectoire des caméras forme un carré fermé suivant le bord extérieur de la scène, commençant en haut à gauche et se dirigeant d'abord vers le bas.

Cette scène permet de valider deux points essentiels du processus de reconstruction : l'estimation de la visibilité et la reconstruction incrémentale. De plus, l'ensemble de l'algorithme est implémenté en GPU ce qui permet une mesure des temps de calcul tout en autorisant une résolution beaucoup plus fine du modèle 3D.

4.2.1 Reconstruction 3D

Pour cette reconstruction, le programme utilisé tient compte de la visibilité grâce à la méthode de la « carte d'occupation » décrite en détail dans la section 3.6.3.2. Ce programme fonctionne sur GPU et la rapidité de traitement a permis d'augmenter la résolution de la grille de voxels à $500 \times 500 \times 200$ voxels. Le volume de reconstruction couvre un espace de $5 \times 5 \times 2\,m$, on obtient donc une taille de voxels de $1 \times 1 \times 1\,cm$.

La qualité globale du modèle 3D obtenu est correcte au terme du parcours dans la scène. La colonne verte est bien dessinée, mais la sphère rouge n'est pas parfaite comme le montre la figure 4.8 qui représente des vues recalculées à partir du modèle 3D.

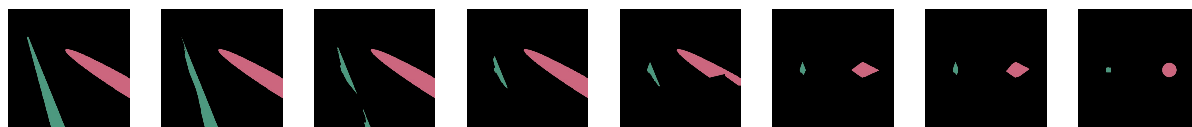


FIGURE 4.9 – Séquence « sphère et colonne » : vues en coupe du modèle 3D
Des coupes horizontales du modèle 3D sont présentées à différentes étapes lors de la reconstruction incrémentale.
Au fur et à mesure du déplacement du capteur, la forme réelle des objet est de mieux en mieux estimée.

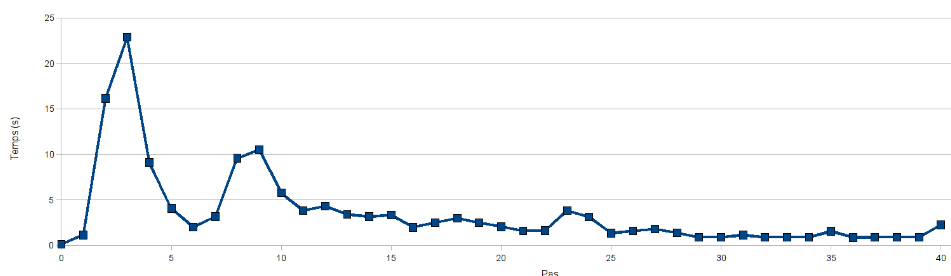


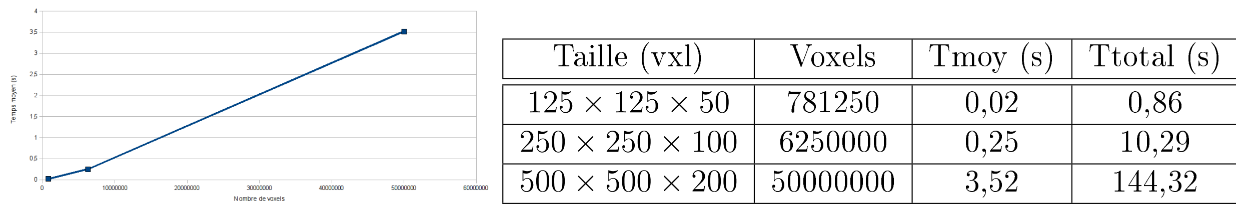
FIGURE 4.10 – Temps de reconstruction pour chaque étape.
Après les premières étapes relativement longues, le calcul s'accélère (moins de voxels à mettre à jour).

La figure 4.9 montre une vue en coupe du modèle 3D à différentes étapes de la reconstruction incrémentale. Au fur et à mesure du déplacement du capteur dans la scène, les voxel marqués opaques par erreur sont éliminés du modèle et les objets sont reconstruits de plus en plus précisément.

4.2.2 Temps de calcul

Le temps de calcul est représenté sur la figure 4.10. La reconstruction 3D présentée ici étant un processus itératif utilisant à chaque étape une paire d'images, ce graphique présente le temps de calcul du modèle pour chacune de ces étapes. Ce temps varie entre 140 ms et 22.87 s sur cette scène avec une moyenne à 3.52 s. En incluant toutes les étapes, le temps total de reconstruction est de 144.32 s.

Le temps de calcul varie pour chaque étape car la méthode d'estimation de la visibilité utilisée ici est itérative. A chaque itération, une nouvelle carte de visibilité est établie et la reconstruction est recalculée pour les voxels nouvellement visibles. Le processus est ainsi itéré jusqu'à ce que les informations de visibilité ne soient plus modifiées, ce qui peut nécessiter un grand nombre d'itérations. Couramment, entre 1 et 10 itérations sont nécessaires, mais des pics à plusieurs dizaines d'itérations sont possibles comme il apparaît sur les premières étapes de la figure 4.10. De tels pics indiquent qu'un grand nombre de voxels ont été mis à jour, ce qui est visible dans la figure 4.9 où de nombreux voxels sont devenus transparents entre la deuxième et la troisième image (correspondant respectivement aux étapes 2 et 4).



Taille (vxl)	Voxels	Tmoy (s)	Ttotal (s)
$125 \times 125 \times 50$	781250	0,02	0,86
$250 \times 250 \times 100$	6250000	0,25	10,29
$500 \times 500 \times 200$	50000000	3,52	144,32

FIGURE 4.11 – Temps moyen de reconstruction en fonction de la résolution du modèle 3D.
Le temps de calcul est directement proportionnel à la résolution du modèle reconstruit.

La figure 4.11 présente le temps moyen d'une étape de la reconstruction, ainsi que le temps total de reconstruction (pour 40 étapes) en fonction de la résolution du modèle 3D. La résolution la plus fine permet une reconstruction rapide en moins de 4 secondes par étape, mais elle ne permet pas la cadence vidéo. En abaissant la résolution à $125 \times 125 \times 50$ vxl, le temps de reconstruction chute radicalement et permet d'envisager des applications vidéos. Cette résolution correspond à une taille de voxel de $4 \times 4 \times 4\,cm$ dans le volume de reconstruction choisi.

4.2.3 Discussion

Lors de l'initialisation (première image de la figure 4.9), le modèle 3D est complètement inconnu et la reconstruction ne peut donc utiliser l'estimation de la visibilité. Comme seule la photoconsistance est utilisée, les objets faiblement texturés de la scène sont reconstruits avec les formes coniques déjà observées dans la section 4.1. Le temps de calcul est aussi très réduit sur cette première étape car l'estimation de la visibilité n'est pas utilisée.

Au fur et à mesure du déplacement du système et de l'utilisation de nouvelles vues pour améliorer le modèle 3D de la scène, les voxels faussement marqués opaques lors de l'initialisation sont peu à peu mis à jour avec un état correct. Malgré les occlusions présentes sur les images sources, les objets masqués ne sont pas supprimés du modèle 3D grâce à la prise en compte de la visibilité. Celle-ci est calculée en utilisant le modèle 3D de l'étape précédente pour fournir des informations approximatives.

Ce résultat illustre l'intérêt de la méthode de reconstruction incrémentale associée à l'estimation de la visibilité pour produire des résultats réalistes.

La précision finale du modèle est bonne, mais n'est qu'une approximation du modèle 3D réel de la scène et ne permet pas certaines applications très exigeantes comme la métrologie.

Le calcul est effectué rapidement, même avec la résolution très fine utilisée ici. Une diminution de la résolution est cependant nécessaire pour envisager des applications à cadence vidéo.

4.3 Influence des textures

Les méthodes de reconstruction basées sur la photoconsistance, sont habituellement fortement influencées par la texture (ou l'absence de texture) sur les objets présents dans

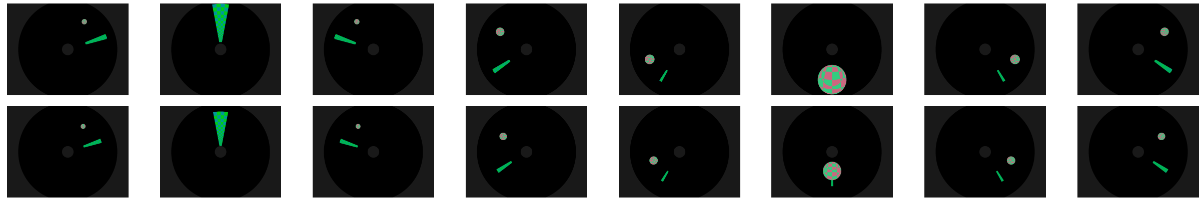


FIGURE 4.12 – Séquence « sphère et colonne texturées » : Images sources

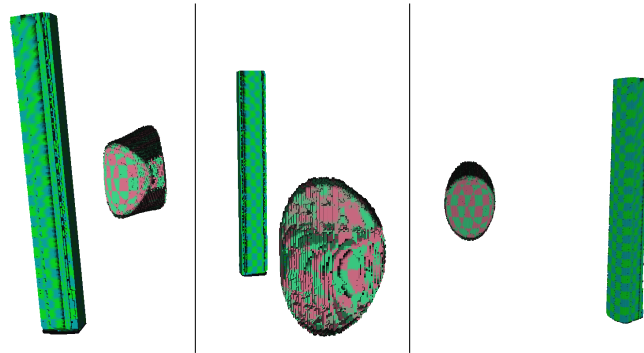


FIGURE 4.13 – Séquence « sphère et colonne texturées » : vues du modèle 3D reconstruit. La texture modifie peu le résultat.

la scène. Dans cette partie, nous nous efforcerons d'évaluer l'influence de telles textures sur la qualité du modèle 3D obtenu.

4.3.1 Objets texturés

Cette séquence en image de synthèses est identique à la séquence précédente, à l'exception de la texture des objets. Alors que dans la scène précédente la sphère et la colonne étaient colorées de manière unie, ils ont une texture en damiers dans cette séquence. Quelques paires d'images composant cette séquence sont présentées dans la figure 4.12.

La méthode de reconstruction utilisée est identique à celle utilisée pour le résultat précédent. Les paramètres étant identiques, cette scène permet d'apprécier la gestion d'objets texturés par notre algorithme.

4.3.1.1 Temps de calcul

Le temps de reconstruction varie de 0,14 s à 11,58 s avec une moyenne à 3,07 s. Le temps total de reconstruction est ainsi à 125,8 s. Ces résultats sont proches des valeurs du résultat précédent. Ici, une limite au nombre d'itérations possibles a été fixée afin de limiter le temps de reconstruction. Cette limite n'est cependant atteinte que pour 3 étapes.

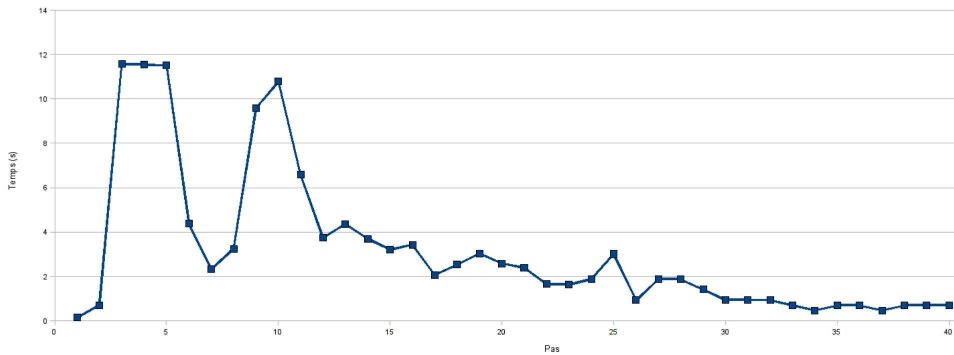


FIGURE 4.14 – Temps de reconstruction pour chaque étape.

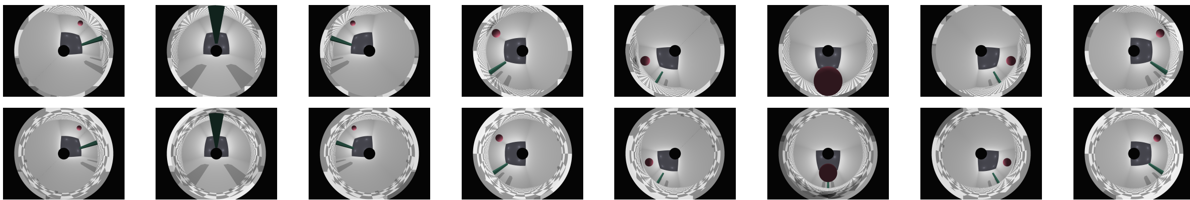


FIGURE 4.15 – Séquence « arrière-plan uniforme » : images sources.

4.3.1.2 Discussion

La figure 4.13 montre plusieurs vues du modèle 3D final. Le résultat est similaire à celui de la séquence précédente. La présence de textures sur les objets n'améliore pas sensiblement la qualité de la reconstruction. En effet, la discrimination entre voxels opaques et transparents est principalement effectuée grâce au déplacement du capteur, l'information de photoconsistance n'étant que peu discriminante. Il en résulte que la texture de l'objet, qui améliore habituellement le comportement du test de photoconsistance, n'apporte pas beaucoup dans notre application.

4.3.2 Arrière-plan uniforme

Cette troisième scène est identique celle présentée dans la section 4.1, a la différence que l'arrière-plan est maintenant présent. Les objets sont placés dans une pièce aux murs gris et au sol pavé. Les caméras suivent la même trajectoire que précédemment.

4.3.2.1 Reconstruction 3D

Le modèle 3D de la scène a été calculé suivant les mêmes modalités que les scènes précédentes. La résolution du modèle est de $500 \times 500 \times 200$ voxels et chaque voxel mesure $1 \times 1 \times 1$ cm.

La figure 4.16 présente différentes vues du modèle 3D reconstruit.

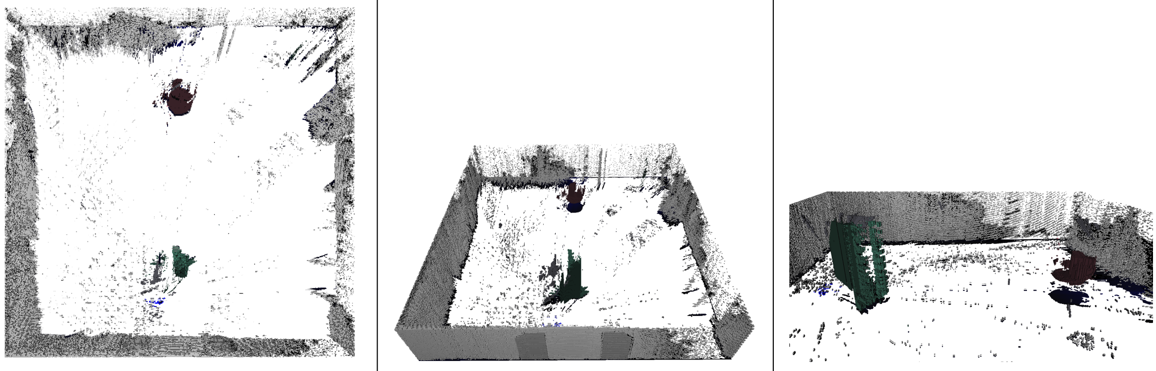


Figure 4.16 – Séquence « arrière-plan uniforme » : vues du modèle 3D reconstruit.
L'arrière-plan pénalise la qualité du modèle reconstruit, la scène est cependant reconnaissable.

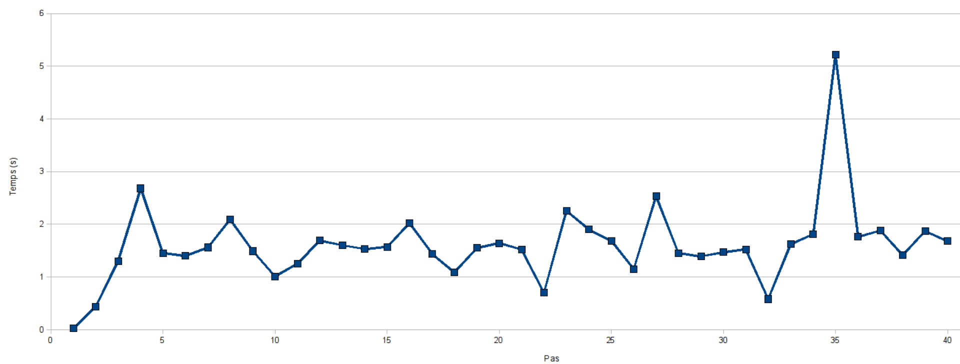


Figure 4.17 – Temps de reconstruction pour chaque étape.
L'arrière-plan impose des mises à jour de voxels à chaque déplacement.

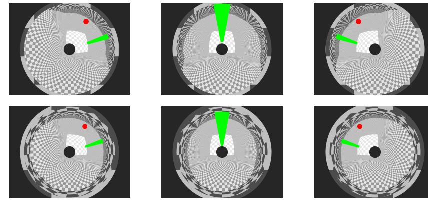


FIGURE 4.18 – Séquence « arrière-plan texturé » : images sources.

4.3.2.2 Temps de calcul

Le temps de calcul pour chaque étape de la reconstruction est présenté dans la figure 4.17. Le temps par étape varie de 20 ms à 5,22 s avec une moyenne à 1,6 s. Le temps total de reconstruction pour l'ensemble des étapes se monte à 65,56 s.

4.3.2.3 Discussion

La reconstruction 3D de cette scène est beaucoup moins précise. La présence d'un arrière-plan fait échouer le test de photoconsistance. Il en résulte un grand nombre de voxels marqués opaques à tort dans le volume central de la scène. La présence de ces voxels opaques gêne à son tour l'estimation de la visibilité ce qui occasionne d'autres erreurs de reconstruction sur les objets.

La reconstruction est plus rapide que pour les scènes présentées précédemment, grâce à un nombre de voxels mis à jour inférieurs. Il est aussi plus constant aux différentes étapes comme le montre le graphique 4.17 qui présente des pics de temps moins importants.

Le modèle reconstruit est bruité et peu précis, malgré la prise en compte de l'ensemble des images de la séquence.

4.3.3 Arrière-plan texturé

La présence d'un arrière-plan uniforme dégrade beaucoup la qualité du modèle 3D obtenu, comme nous avons pu le voir dans la section précédente. Dans cette section, nous allons étudier l'impact d'un arrière-plan texturé sur la qualité de la reconstruction.

La scène utilisée est identique à la précédente, mais les murs et le plafond ont été garnis d'une texture en damier. Quelques vues de cette scène sont présentés dans la figure 4.18.

4.3.3.1 Reconstruction 3D

La figure 4.19 présente des coupes horizontales du modèle 3D généré après utilisation de l'ensemble des vues de la scène. La présence d'un arrière-plan influence négativement le processus de reconstruction et limite la qualité du modèle obtenu.

Le caractère périodique de la texture murale se retrouve sous forme de couches successives de voxels faussement marqués opaques le long des murs. Les textures périodiques sont donc particulièrement difficiles pour le test de photoconsistance qui aura ainsi du mal à distinguer une réelle correspondance sur une surface texturée d'une fausse correspondance établie en avant de cette texture, les couleurs se répétant à intervalle régulier.

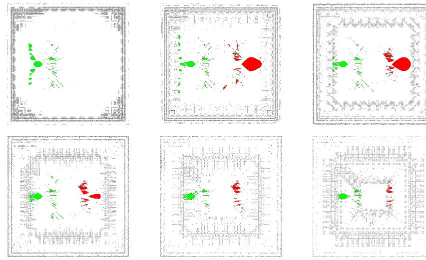


FIGURE 4.19 – Séquence « arrière-plan texturé » : Coupes horizontales du modèle 3D. La texture très régulière de l'arrière-plan vient bruiter le modèle 3D reconstruit.

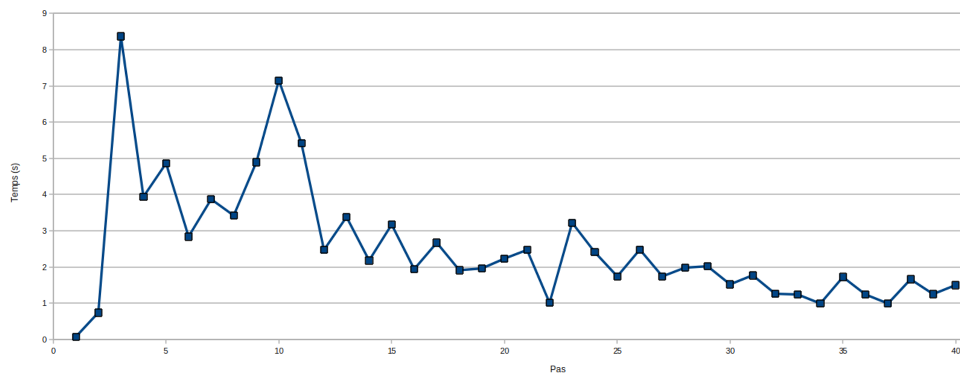


FIGURE 4.20 – Temps de reconstruction pour chaque étape.

Cet effet est une conséquence directe des inconvénients de la méthode de voxel coloring basée sur la photoconsistance telle que présentée dans la section 1.4.2.1.

4.3.3.2 Temps de calcul

Le temps de calcul pour chaque étape de la reconstruction est présenté sur la figure 4.20. On observe globalement des caractéristiques similaires aux autres reconstructions présentées précédemment. Le temps moyen de reconstruction s'établit ici à 2,51 s (minimum 7 ms ; maximum 8,37 s). Le temps total pour l'ensemble des étapes atteint 102,95 s.

Le graphique de la figure 4.20 montre les mêmes pics de temps dans les premières étapes, où un grand nombre de voxels sont creusés, puis converge peu à peu vers une durée de reconstruction inférieur à 2 s par étape.

4.3.4 Suppression du bruit

Lors de la reconstruction, quelques voxels peuvent être faussement détectés comme opaque, notamment à l'avant d'une surface possédant texture périodique comme vu précédemment.

Comme l'état des voxels (opaque ou transparent) est utilisé pour estimer la visibilité pour l'étape suivante, de telles erreurs de classification entraînent à leur tour des erreurs de reconstruction.

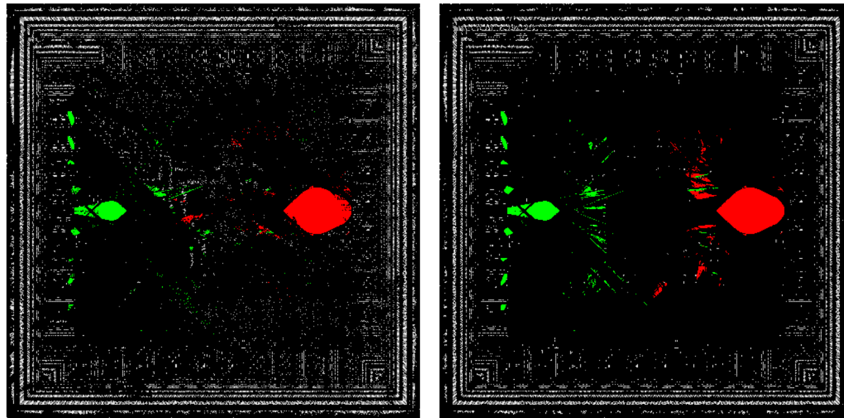


FIGURE 4.21 – Efficacité du filtre sur une reconstruction 3D. Les images représentent une coupe horizontale de la reconstruction 3D vue de dessus. La reconstruction de droite a été calculée avec le filtre activé.
Le filtre limite fortement le bruit dû à l'arrière-plan. Les objets de la scène sont aussi mieux reconstruits.

Pour limiter ces erreurs, nous utilisons un système de filtrage basique mais néanmoins efficace chargé de supprimer les voxels opaques isolés. En effet, les objets d'une scène sont plus grands que la résolution volumétrique de la grille de voxel. Un voxel isolé peut donc être considéré comme étant une erreur de mesure.

L'efficacité de cette méthode est illustrée par la figure 4.21. Dans les deux cas, la reconstruction 3D d'une scène a été effectuée selon notre méthode. Dans le cas de l'image de gauche, le filtre supprimant les voxels isolés a été activé ce qui conduit à une reconstruction moins bruitée et plus réaliste.

Le filtre utilisé teste en parallèle tous les voxels de la scène et compte, pour chacun d'entre eux, le nombre de voxels voisins possédant l'état « transparent ». Les six voxels voisins ayant une face commune sont testés. Si le nombre de voisins transparents est égal à 6, alors le voxel est isolé et peut être supprimé.

Pour respecter les contraintes de programmation massivement parallèle, les voxels ne sont pas immédiatement supprimés mais sont marqués pour être mis à jour à l'étape de reconstruction suivante. Cela est nécessaire car si un voxel est immédiatement supprimé, cela peut fausser le processus de filtrage d'un voisin immédiat qui est calculé en même temps.

Dans la pratique, ce filtre est utilisé sur toutes les scènes possédant un arrière-plan afin d'améliorer le résultat obtenu.

L'impact, en terme de temps de calcul, de l'utilisation de ce filtre a été évalué sur deux scènes. La première ne possède pas d'arrière-plan et permet donc d'évaluer le temps de calcul additionnel lié à l'utilisation du filtre, sans que cela ne modifie le modèle 3D obtenu. La deuxième possède un arrière-plan texturé générant un grand nombre de voxels bruités et permet donc d'évaluer le temps de calcul alors que le filtre améliore le modèle 3D obtenu.

La figure 4.22 présente le temps de reconstruction par étape pour chaque étape lors du calcul du modèle 3D d'une scène ne possédant pas d'arrière-plan. Le temps de calcul varie

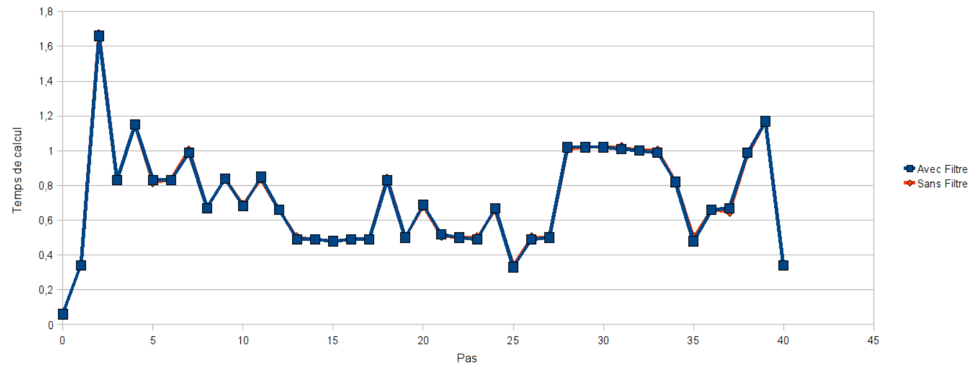


Figure 4.22 – Impact du filtre sur le temps de calcul par étape. Cette scène ne possède pas d'arrière-plan et le modèle 3D obtenu n'est pas modifié par l'utilisation du filtre. L'usage du filtre n'impacte pas négativement le temps de calcul dans ce cas.

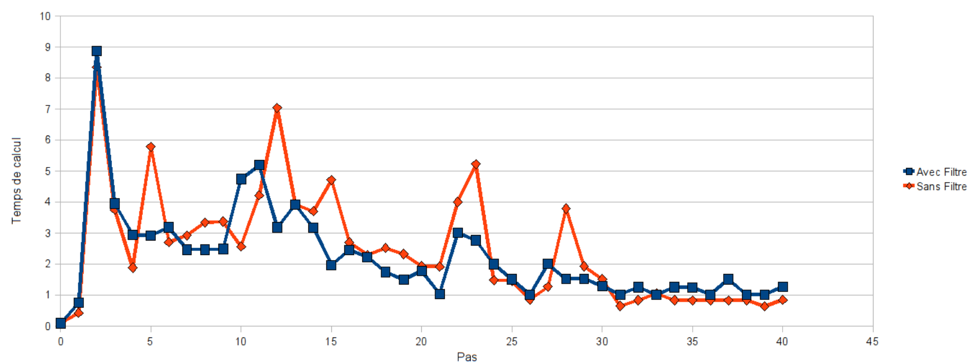


Figure 4.23 – Impact du filtre sur le temps de calcul par étape. Cette scène possède un arrière-plan texturé et le modèle 3D obtenu est amélioré par l'utilisation du filtre. L'usage du filtre réduit sensiblement le temps de calcul moyen.

entre 60 ms et 1,66 s avec une moyenne à 0,72 s. Le temps de calcul ajouté par l'utilisation du filtre est au pire de 10ms pour quelques étapes. Le temps total de reconstruction passe de 29,54 s à 29,57 s lors de l'utilisation de ce filtre.

La figure 4.23 présente l'impact du filtre sur le temps de calcul d'une scène possédant un arrière-plan texturé générant beaucoup de bruit. Le modèle 3D reconstruit est globalement amélioré par l'utilisation de ce filtre car il permet une plus grande précision de l'estimation de la visibilité. Sur cette scène, le temps de calcul varie entre 8 ms et 8,36 s, avec une moyenne à 2,49 s sans le filtre. Lorsque le filtre est activé, le temps de traitement diminue légèrement et varie entre 8 ms et 8,88 s, avec une moyenne à 2,22 s. Le temps total de traitement passe ainsi de 101,98 s à 91,04 s grâce à l'activation du filtre. Ce résultat s'explique par le fait que la fonction de filtrage s'exécute efficacement sur l'architecture choisie et impacte donc très peu le temps de calcul. Cependant, en améliorant la précision du modèle et notamment sa correspondance avec la scène réelle, cela limite le nombre d'itérations de reconstructions qui doivent sans cela prendre en compte et mettre à jours les voxels bruités.

En conclusion, l'utilisation de ce filtre simple et peu coûteux en temps de traitement permet d'améliorer le modèle 3D obtenu tout en réduisant légèrement le temps de traitement des scènes fortement bruitées. L'implémentation massivement parallèle de cette fonction est efficace et performante.

4.3.5 Conclusion sur l'influence des textures

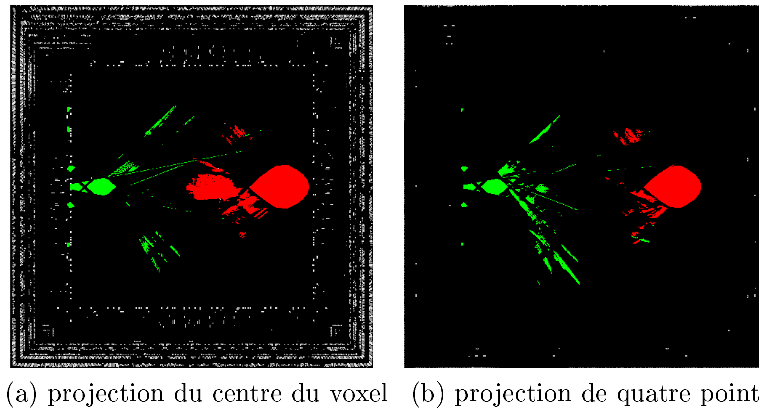
Dans la littérature, les méthodes de reconstructions volumétriques basées sur la photoconsistance fonctionnent mieux en présence d'objets fortement texturés. Ce constat reste valable pour les applications multi-vues exploitant plusieurs caméras, comme le montre [SD99].

Dans le cas de notre application cependant, la photoconsistance n'est pas le facteur dominant le calcul de la reconstruction 3D car notre système de vision, composé de seulement deux caméras, ne le permet pas. Le déplacement du capteur joue ici un rôle plus important en permettant la construction de l'enveloppe visuelle des objets. Dans ce cas, la texture des objets ou de l'arrière-plan de la scène n'améliore pas sensiblement la qualité de la reconstruction 3D calculée.

4.4 Comparaison des méthodes de projection d'un voxel

Comme présenté dans la section 3.6.5, nous avons utilisé principalement deux méthodes pour tester la photoconsistance d'un voxel. La première utilise, pour chaque voxel, une seule projection par image : celle du centre de gravité du voxel considéré. La seconde utilise la projection de quatre points échantillonnés dans le voxel et projetés sur chaque image.

Dans cette partie, nous allons comparer ces deux méthodes et analyser leur impact sur le modèle 3D reconstruit ainsi que sur le temps de calcul.



(a) projection du centre du voxel (b) projection de quatre points

FIGURE 4.24 – Modèle 3D reconstruit avec deux méthodes de projection des voxels. La projection utilisant quatre point permet une meilleure réjection des fausses correspondances et améliore ainsi sensiblement la reconstruction : les murs sont mieux reconstruit (moins de bruit) et la sphère mieux définie.

4.4.1 Temps de calcul brut

Nous allons comparer le temps de calcul de ces deux méthodes lorsque aucun autre facteur ne vient intervenir dans la reconstruction. Les autres étapes de l'algorithme sont désactivées (reconstruction incrémentale, filtres, estimation de la visibilité, ...) il ne reste que la fonction de projection des voxels, le test de photoconsistance et la coloration des voxels résultants. Bien entendu, le modèle reconstruit par cette version amputée de notre algorithme est peu réaliste, et le seul intérêt est de pouvoir comparer les temps d'exécution des deux fonctions de photoconsistance.

Avec la projection du centre du voxel uniquement, le temps d'exécution moyen d'une étape est de 0,04 secondes, avec un total de 1,62 secondes pour le calcul de l'ensemble des vues de la scène. La seconde méthode de projection utilisant quatre échantillons par voxel nécessite quand à elle 0,10 secondes en moyenne, pour un total de 4,06 secondes ; soit 2,52 fois plus long.

La seconde fonction de photoconsistance est donc plus lente, car elle nécessite quatre projections par voxel quand la première n'en effectue qu'un seul. Ce résultat attendu est néanmoins intéressant : le temps de calcul n'est pas multiplié par 4 comme on aurait pu le penser. L'impact négatif sur le temps de calcul étant limité notamment grâce au cache de texture qui permet une relecture rapide des pixels précédemment accédé.

4.4.2 Reconstruction 3D

Nous avons évalué les deux méthodes de projection sur une scène simple possédant un arrière-plan texturé déjà utilisée précédemment et dont les images sources sont visibles dans la figure 4.18. Des vues du dessus des deux modèles reconstruits sont visibles dans la figure 4.24.

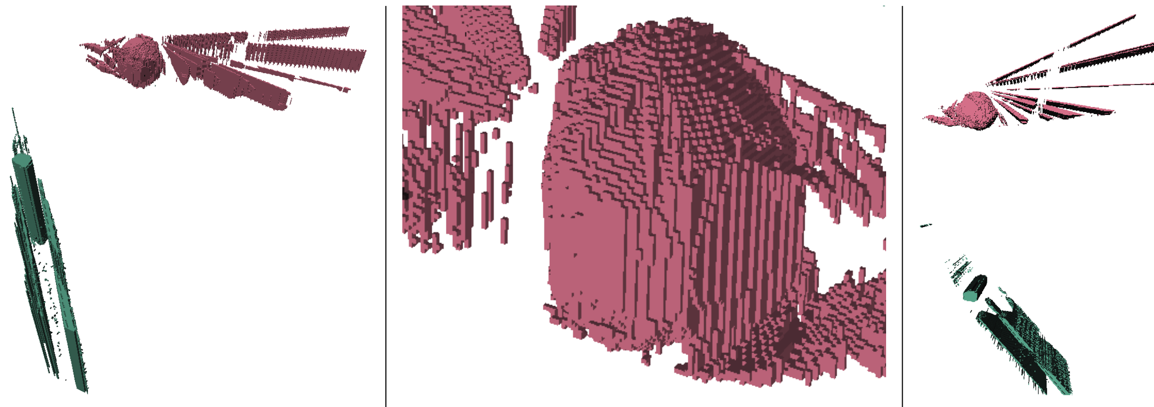


FIGURE 4.25 – Séquence « arrière-plan uniforme » : vues du modèle 3D reconstruit.

4.4.3 Synthèse et discussion

Les deux méthodes de projections sont très performantes et produisent de bons résultats. La projection du centre du voxel est légèrement plus rapide, mais le modèle 3D présente un bruit important en présence de scènes aux textures périodiques. La projection de quatre points permet de limiter fortement ces phénomènes indésirables, au prix d'un léger allongement du temps de calcul.

4.5 Influence du positionnement des caméras

Pour toutes les reconstructions présentées jusqu'ici, la position des caméras dans la scène pour chaque prise de vue est parfaitement connue. Dans cette section, la position des caméras est volontairement faussée afin d'estimer l'impact d'une erreur de positionnement sur la qualité du modèle 3D reconstruit.

Nous utilisons ici les séquences déjà utilisées dans les sections précédentes, mais une erreur aléatoire a été ajoutée à l'information de position du système de vision. Cette erreur aléatoire est comprise entre -25 mm et +25 mm sur chacune des dimensions X et Y de la position du capteur.

La figure 4.25 présente des vues 3D reconstruites à partir des images de la scène visible sur la figure 4.6 mais en ajoutant cette erreur de positionnement. Le modèle 3D calculé est impacté par cette erreur, notamment à cause de problèmes dans l'estimation de la visibilité. En effet, la visibilité étant estimée en fonction de la position supposée des caméras, une erreur de positionnement à ce niveau entraîne une information erronée de visibilité. Ceci a pour effet de fausser la classification des voxels placés entre un objets réel de la scène et la caméra, sur les bords de la silhouette de l'objet. Il en résulte les formes étirées visibles sur la figure 4.25, constituées de voxels marqués à tords comme opaques.

La connaissance précise de la position des caméras dans la scène est l'une des contraintes majeures de notre système. Lors de nos expérimentations, cette donnée était connue ou mesurée ; dans un système opérant en situation réelles, il serait nécessaire d'obtenir cette information d'une source externe fiable et précise.

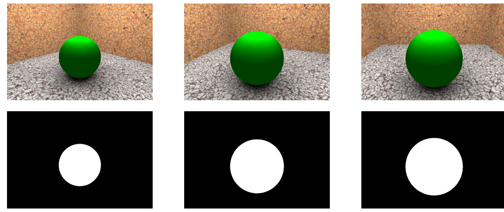


FIGURE 4.26 – Séquence « perspective » avec les silhouettes associées.

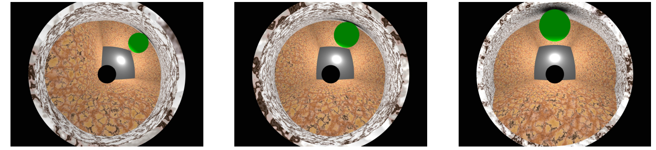


FIGURE 4.27 – Séquence « catadioptrique ».

4.6 Comparaison avec la méthode « Generalized Voxel Coloring »

Afin de comparer les performances de notre méthode de reconstruction par rapport à une méthode plus classique, nous avons mis en place le dispositif de test suivant.

Une scène virtuelle, créée avec PovRay, sert de référence. Les images de cette scène sont rendues de deux manières différentes : soit en utilisant une caméra perspective unique, soit en utilisant une paire de caméras catadioptriques. Dans les deux cas, la caméra est déplacée suivant un parcours prédéfini afin de fournir les multiples vues.

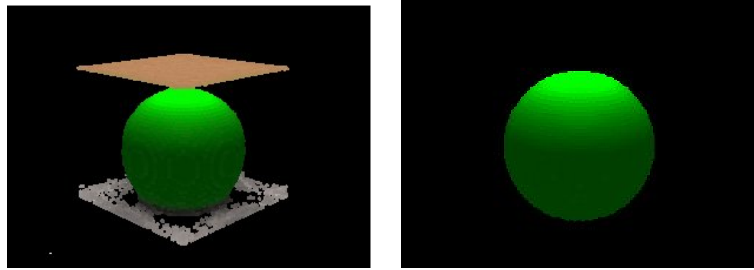
Des images binaires « silhouettes », représentant la segmentation avant-plan / arrière-plan sont aussi générées. Un fichier contenant la position et l'orientation des caméras pour chaque position dans la scène est aussi produit.

Un fois l'opération de rendu effectuée, nous obtenons donc deux séquences distinctes : une séquence « catadioptrique » et une séquence « perspective » ainsi que les silhouettes de segmentation d'arrière-plan correspondant et le fichier des positions de caméras.

Ces séquences sont ensuite utilisées pour calculer des modèles 3D. La séquence « catadioptrique » est utilisée par notre reconstructeur et un modèle 3D est généré. La séquence « perspective » ainsi que les silhouettes sont utilisées par le programme Archimedes [Lop02] qui est un programme librement disponible implémentant la méthode « Generalized Voxel Coloring » détaillée dans la section 1.4.2.2.

La figure 4.26 présente quelques-unes des images de la séquence « perspective » de la scène, tandis que la figure 4.27 présente les images de la séquence « catadioptrique ». L'arrière-plan est texturé et l'objet central présente un gradient de couleur dû à l'illumination de la scène.

Cette comparaison tâche d'évaluer la qualité du modèle reconstruit et non le temps de calcul qui est très différent entre les deux applications et peu représentatif. En effet, le programme Archimedes utilise un seul processeur alors que notre reconstructeur exploite la puissance de calcul parallèle d'une carte graphique haut de gamme. De plus, le temps de calcul prohibitif de Archimedes nous a obligé à limiter la résolution du modèle à $200 \times 200 \times 200$ voxels. Les temps de reconstruction n'ont donc pas été strictement mesurés. A titre d'information cependant, la reconstruction effectuée par le programme Archimedes nécessite plusieurs dizaines de minutes à une heure, tandis que notre algorithme offre des performances similaires à ce qui a été annoncé sur les scènes présentées précédemment (quelques secondes par étape, une à deux minutes pour l'ensemble de la séquence) et avec



(a) Calculé sans les silhouettes. L'arrière-plan est partiellement reconstruit (sol et murs).

(b) Calculé avec les silhouettes. Seul l'objet est reconstruit.

FIGURE 4.28 – Vues du modèle 3D calculé par Archimedes

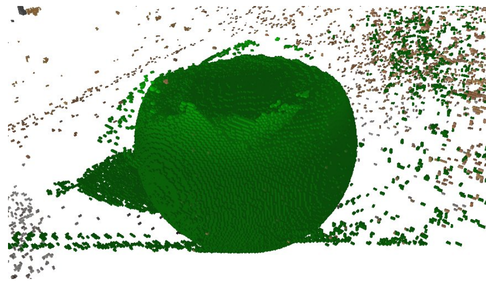


FIGURE 4.29 – Vues du modèle 3D calculé par notre reconstructeur (sans silhouettes). L'arrière-plan est partiellement reconstruit (voxels bruités entourant la sphère). Dû à la position du capteur, le sommet de la sphère n'est pas visible et est donc coupé sur le modèle reconstruit.

une résolution beaucoup plus fine.

4.6.1 Reconstruction 3D

Le programme Archimedes ne permet pas d'accéder directement au modèle 3D généré, mais permet en revanche d'exporter des vues de ce modèle. Archimedes permet d'utiliser des silhouettes de l'objet lors du processus de reconstruction afin de limiter les problèmes inhérents à une reconstruction basée sur la photoconsistance. Utiliser ces silhouettes revient à limiter l'étendue du domaine reconstruit au volume effectivement occupé par l'enveloppe visuelle de l'objet. Un modèle calculé avec et sans les silhouettes est présenté dans la figure 4.28.

La figure 4.29, quand à elle, présente une vue du modèle 3D obtenu par notre méthode. la forme de la sphère est globalement bien retrouvée, mais de nombreux voxels bruités apparaissent à cause de la présence d'un arrière-plan. La partie supérieure de la sphère n'est pas reconstruite car elle n'est pas visible depuis la position des caméras : l'acquisition est effectuée juste au dessus du niveau du sol dans la séquence panoramique, alors que dans la séquence perspective, les prises de vues sont réalisées en hauteur.

Il faut cependant souligner le fait que ce modèle est obtenu beaucoup plus rapidement que le modèle calculé par Archimedes et qu'il n'utilise pas de silhouettes réalisant une

FIGURE 4.30 – Séquence « bureau » : Images sources
Cette scène très riche a été retouchée manuellement : le plafond a été colorié en noir sur l'une des deux images pour ne pas apparaître dans la reconstruction.
Scène adaptée du modèle The average office (Jaime Vives Piqueres)

segmentation arrière-plan / avant-plan avant le calcul de la reconstruction. Une autre différence majeure dans la méthodologie de reconstruction mérite d'être soulignée : Archimedes a besoin de toutes les images au début du processus de reconstruction alors que notre reconstructeur effectue son travail incrémentalement.

4.7 Scène complexe

Notre méthode de reconstruction a été évaluée sur une scène virtuelle riche, comportant un grand nombre d'objets et simulant d'une façon réaliste un environnement réel. Quelques

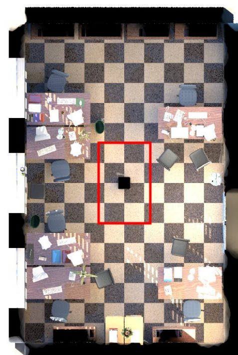
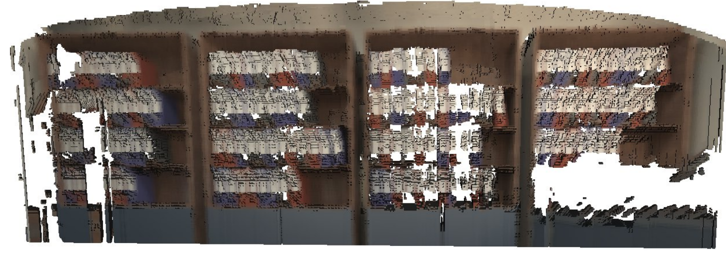


FIGURE 4.31 – Séquence « bureau » : vue du dessus. La trajectoire des caméras est indiquée en rouge

(a) Extrait d'une image-source.



(b) Vue de face du modèle 3D reconstruit.

FIGURE 4.32 – L'armoire aux classeurs.
La reconstruction 3D est proche de l'original. Les faibles disparités en profondeur entre les classeurs et le fond des étagères ne sont pas visibles sur la reconstruction.

images sources de cette séquence sont présentées dans la figure 4.30. La séquence complète comporte 40 paires d'images acquises au cours d'un déplacement du système de caméras dans le bureau virtuel. Le plafond de la scène a été segmenté manuellement (colorié en noir) sur l'une des images de chaque paire et ne sera ainsi pas reconstruit, cette large surface à la couleur uniforme provoque en effet de nombreuses erreurs de reconstruction. La trajectoire décrite par les caméras est présentée par un trait rouge sur la figure 4.31 présentant une vue du dessus de la scène complète.

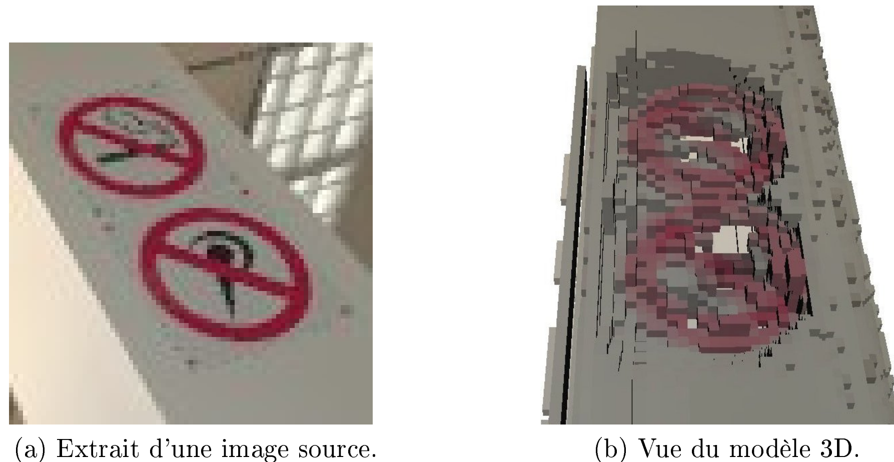
4.7.1 Reconstruction 3D

Le modèle 3D calculé possède ces caractéristiques : la taille du volume reconstruit est de $5\times 8\,m$, taille des voxels de $1\times 1\times 1\,cm$ soit une résolution du modèle de $500\times 800\times 200\,vxl$. Le modèle est calculé incrémentalement en utilisant la méthode de projection du voxel par échantillonage de quatre points présentée dans la section 3.6.5.4 et la méthode d'estimation de la visibilité par carte d'occupation présentée dans la section 3.6.3.2. Le filtre évoqué dans la section 4.3.4 est utilisé.

Conséquence directe de la haute résolution du modèle 3D reconstruit et du grand nombre d'objets présents dans la scène, le modèle 3D est très volumineux et il est pour cette raison difficile à visualiser dans son ensemble. Nous avons ici extrait certaines parties de ce modèle afin de présenter les résultats obtenus.

La figure 4.32 présente un premier objet extrait du modèle reconstruit. Il s'agit d'un ensemble de quatre armoires longeant le mur et garnies de classeurs. Le modèle reconstruit est aisément identifiable à l'image d'origine.

Le deuxième objet illustrant cette reconstruction est présenté dans la figure 4.33. Il représente deux panneaux signalétiques accrochés au poteau central présent dans la scène. Le modèle reconstruit est relativement fidèle à l'objet d'origine, malgré la petite taille de ce dernier (environ 20 cm de largeur). Cet objet est particulièrement bien reconstruit car la caméra, au cours de son déplacement dans la scène, en fait le tour.



(a) Extrait d'une image source. (b) Vue du modèle 3D.

FIGURE 4.33 – Signalétique sur le poteau central.
Les panneaux sont reconnaissables malgré leur petite taille : ils sont composés de quelques voxel seulement.

4.8 Scène réelle

Notre méthode de reconstruction 3D a été expérimentée sur des images réelles acquises grâce au système de vision équipant le robot mobile présenté dans la section 3.3.2. Pour effectuer l'acquisition, le robot est déplacé dans la scène et sa position est mesurée par rapport à un point fixe. A chaque position, les deux images issues des deux caméras sont archivées, la reconstruction 3D étant effectué par la suite.

La scène utilisée comme lieu d'expérimentation est l'une des salles de notre laboratoire, carrée et mesurant 6,5 m de côté. Le robot suit une trajectoire rectiligne dans l'allée centrale de la pièce, avançant de 50 cm entre chaque acquisition. La scène complète comporte 10 paires d'images, totalisant ainsi un déplacement linéaire de 5 m.

La figure 4.34 présente quelques vues de la séquence.

Comme pour le résultat précédent, le plafond a été segmenté sur l'une des images de chaque paire : les éclairages au néon et l'arrière plan blanc uniforme et saturé induisant de nombreuses erreurs dans le modèle et notamment la reconstruction d'un gros bloc de voxel blancs remplissant la partie supérieure de la scène.

4.8.1 Reconstruction 3D

Notre méthode de reconstruction 3D est sensible à la précision du positionnement des caméras dans la scène, comme nous l'avons montré dans la section 4.5. Dans une scène réelle, la mesure de cette position n'étant pas suffisamment précise, nous n'avons pas été en mesure d'utiliser l'ensemble de la séquence, les erreurs cumulées étant trop importantes. Pour calculer le modèle 3D présenté ici, nous n'avons utilisé que les quatre premières paires d'images de cette séquence.

Malgré le faible nombre d'image utilisées, certains objets de la scène sont correctement reconstruits, comme par exemple l'étagère présentée dans la figure 4.35.

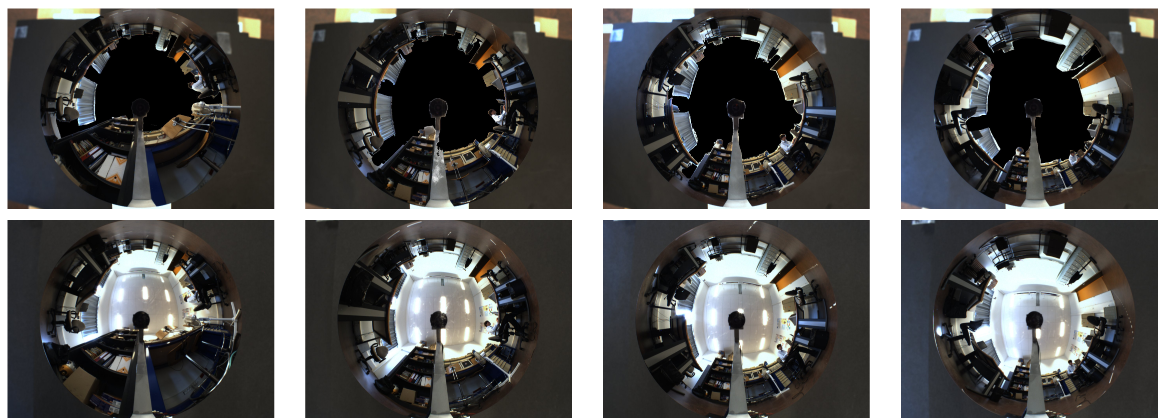
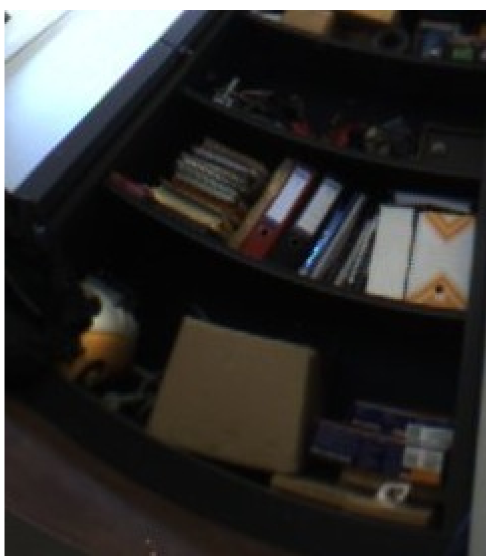


FIGURE 4.34 – Séquence « laboratoire » : Images sources.
Le plafond a ici aussi été segmenté.



(a) Extrait d'une image-source.

(b) Vue du modèle 3D.

FIGURE 4.35 – Image et vue du modèle 3D de l'armoire.
La forme générale de l'objet est restituée, les petits objets sont par contre éliminés.

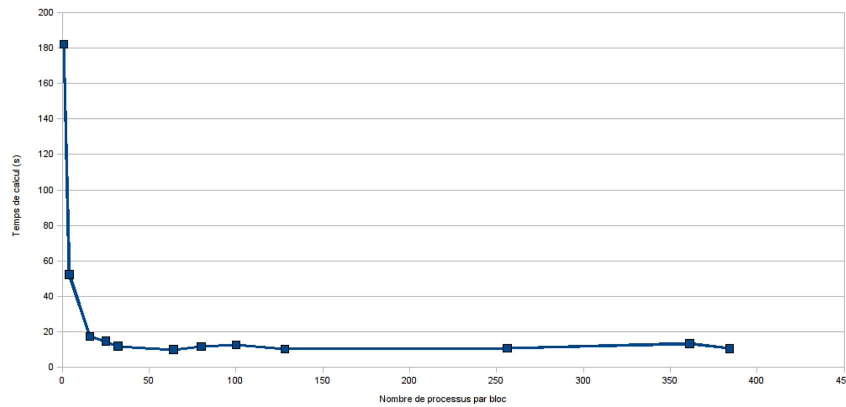


FIGURE 4.36 – Temps de calcul en fonction du nombre de threads par bloc.
Dès que suffisamment de threads sont instanciés par blocs, les disparités de temps de calcul sont relativement faibles.

4.9 Répartition des calculs

Un point important concernant l'exploitation optimale de l'architecture massivement parallèle du GPU concerne la répartition des calculs en « blocs ». Alors que le programme dans son ensemble doit traiter une grille d'environ $500 \times 500 \times 200$ voxels, le GPU ne peut prendre en compte que des blocs de 512 thread maximum. Il convient donc de découper la grille globale en autant de blocs que nécessaire.

Chaque thread nécessite cependant un certain nombre de ressources pour s'exécuter (registres et mémoire partagée principalement), qui sont partagées par tous les threads d'un bloc. Ainsi, plus un kernel nécessite de ressources, moins d'instances de ce kernel pourront être exécutées en parallèle dans un bloc.

Il convient donc d'étudier le point optimal entre la taille des blocs (plus il y a de threads par bloc, plus le traitement est rapide) et l'adéquation entre les ressources nécessaires pour un threads et celles disponibles sur l'architecture (si le GPU manque de ressources en registre, les données sont transférées vers la RAM globale, beaucoup plus lente, ce qui ralentit le traitement).

nVidia met à disposition un outil appelé « Cuda Occupancy Calculator » qui permet, à partir des ressources nécessaires pour un kernel, de calculer combien d'instances de ce kernel peuvent être appelées par bloc de manière à occuper au maximum le GPU. Avec les paramètres de notre application, cet outil outil nous indique qu'une taille de bloc de 64 threads est optimal.

Nous avons vérifié ce chiffre dans cette partie en exécutant notre programme selon différentes configuration de la taille de bloc. Le temps d'exécution de 6 étapes du processus de reconstruction est donné dans le tableau 4.1, selon différentes tailles et organisations du bloc de threads. Le graphique 4.36 présente quand à lui le temps d'exécution en fonction du nombre de threads par bloc.

Tout d'abord, cette expérimentation nous indique une tendance attendue : un trop faible nombre de threads par bloc impacte négativement les performances : le temps d'exécution pour des tailles de bloc inférieur à 16 threads est particulièrement significatif.

Table 4.1 – Temps de calcul pour différentes tailles du bloc de threads.
La géométrie du bloc a un impact très faible sur le temps de calcul.
La taille de bloc optimale est de 64 threads dans ce cas.

Nombre de threads par bloc	**Géométrie du bloc (taille $x \times y$)**	**Temps d'exécution (s)**
1	1 × 1	181.99
4	4 × 1	52.36
16	16 × 1	17.46
16	8 × 2	17.48
25	5 × 5	14.8
32	32 × 1	11.95
64	32 × 2	10.06
64	64 × 1	10.05
64	8 × 8	10.07
64	16 × 4	10.03
80	80 × 1	11.84
100	10 × 10	12.70
128	128 × 1	10.41
128	32 × 4	10.45
256	256 × 1	10.84
256	16 × 16	10.86
256	32 × 8	10.80
361	19 × 19	13.45
384	384 × 1	10.71

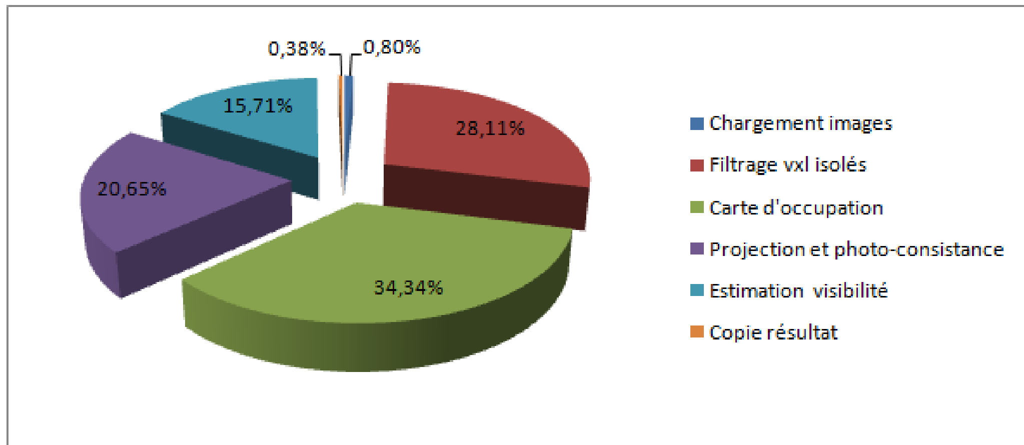


FIGURE 4.37 – Répartition du temps de calcul entre les différentes tâches

Ces mesures nous indiquent aussi que la géométrie du bloc n'a que peu d'impact sur les performances. Pour un bloc de 64 threads par exemple, les différences de temps d'exécution pour les géométries de 32×2, 64×1, 8×8 et 16×4 ne sont pas significatives ($< 0,5\%$).

Enfin, ces résultats confirment que la taille de bloc optimale se situe bien vers 64 blocs pour notre application, taille pour laquelle le temps minimum est relevé.

Pour des temps d'exécution très proches, il est préférable d'utiliser une configuration avec le moins de threads par bloc car cela libère un peu de ressource, que l'on peut mettre à profit pour optimiser l'implémentation du kernel.

4.10 Profilage du temps d'exécution

Nous allons maintenant étudier la répartition du temps de calcul entre les différentes tâches successives qui composent le programme de reconstruction 3D. Ces temps de calculs intermédiaires ont été mesurés lors des reconstructions 3D présentées précédemment. Pour effectuer ces relevés, le code est instrumenté afin de reporter le temps d'exécution de chaque partie du programme. Le résultat de ce profilage est présenté dans la figure 4.37.

Tout d'abord, les opérations de transfert depuis et vers le GPU représentent une faible part du temps total. Le chargement des images (ce temps inclus aussi les initialisations) et le transfert des données produites représentent à eux deux à peine plus de 1% du temps global d'exécution.

La partie centrale de l'algorithme, la projection des voxels et le test de photoconsistance (incluant les temps d'accès aux images), représente environ 20% du temps total. Cette performance est atteinte grâce à la forte adéquation entre l'algorithme utilisé et l'architecture massivement parallèle du GPU.

L'estimation de la visibilité représente presque 16% du temps global. Là encore, l'efficacité des méthodes proposées permet de limiter l'impact négatif de cette tâche plus difficile à paralléliser.

Les deux tâches les plus coûteuses sont le filtrage des voxels isolés (28%) et le calcul de

la carte d'occupation (34%). Bien que ces deux tâches soient bien parallélisées, elles ont en commun de devoir parcourir exhaustivement l'ensemble des voxels de la grille lors de leur traitement. La bande-passante de la mémoire de la carte graphique est alors l'élément qui limite le plus les performances.

4.11 Discussion générale

Les résultats présentés dans ce chapitre montrent plusieurs grandes tendances de notre méthode de reconstruction qu'il convient d'étudier.

Tout d'abord, concernant la qualité de la reconstruction, le modèle 3D généré à partir des images de la scène est globalement satisfaisant. La reconstruction incrémentale alliée à l'estimation de la visibilité permettent de limiter les problèmes causés par un test de photoconsistance simpliste et par conséquent souvent erroné. La présence d'un arrière-plan induit notamment de nombreuses erreurs, en particulier s'il présente une texture périodique ou une large surface uniforme.

Pour contrer ce problème, l'utilisation de critères beaucoup plus discriminants que la photoconsistance serait à envisager, comme les informations de texture par exemple. Cependant, de telles méthodes ajoutent un temps de calcul important et nécessitent l'analyse d'une surface plus étendue de l'image, ce qui peut pénaliser fortement les performances. Ce point précis est aussi un problème majeur dans la plupart des méthodes classiques de reconstruction volumétriques utilisant la photoconsistance. La solution courante étant d'utiliser des images prétraitées en entrée de l'algorithme sur lesquelles l'arrière-plan a été supprimé. Avec de telles images, notre méthode de reconstruction 3D obtient de meilleurs résultats comme le montre les séquences présentées section 4.2 et section 4.3.1. Cette solution n'est cependant pas applicable dans le cas d'un système automatique, car le processus de segmentation avant-plan / arrière-plan n'est pas facilement automatisable.

Concernant le temps de traitement, notre méthode fonctionne très rapidement grâce à l'adéquation de l'algorithme et du matériel le faisant fonctionner. Notre méthode est capable de fournir des modèles 3D d'une résolution très supérieure à ce que l'on trouve généralement dans la littérature, et cela avec des cadences rapides, la cadence vidéo pouvant même être atteinte pour une résolution un peu plus limitée du modèle. Nos résultats ont ainsi montré que les méthodes de reconstruction volumétriques sont de bonnes candidates pour une exploitation sur machine massivement parallèle. Les deux facteurs clés permettant de bonnes performances sur cette architecture sont un accès à des données proches en mémoire et une indépendance forte des traitements entre les différents threads.

La forte sensibilité à la précision de la position des caméras est l'une des principales limitation de notre méthode, commune avec la plupart des méthodes de reconstruction volumétriques existantes. Il est pour cela nécessaire de mesurer cette position avec une précision meilleure que la taille d'un voxel (1 cm ici), ou bien le modèle 3D reconstruit sera fortement dégradé. Une solution alternative pourrait consister à étendre notre méthode pour être en mesure d'estimer la pose des caméras dans la scène et leur déplacement d'une acquisition à l'autre.

La validation de notre méthode sur des scène riches ou réelles produit des modèles 3D très complexes, difficilement visualisables et très lourds à étudier et afficher. Dans le cas

d'une application réelle, il serait nécessaire de réaliser un logiciel capable d'effectuer cet affichage en temps réel, ce qui ne présente pas de difficulté technologique majeure mais nécessite une implémentation soignée et du matériel performant.

4.12 Conclusion

Dans ce chapitre, la méthode de reconstruction imaginée et détaillée dans le chapitre 3 a été confrontée à plusieurs scènes et les résultats de reconstruction obtenus ont été présentés.

L'utilisation d'une paire de caméras panoramiques permet l'observation stéréoscopique de l'environnement du robot sur 360°. La structure des capteurs, monolithique et statique, est bien adaptée à la robotique mobile où une monture motorisée générerait des perturbations et entraînerait une fragilité. La rapidité d'acquisition de ces caméras permet d'envisager des applications temps réel à cadence vidéo.

La méthode de reconstruction 3D présentée permet d'obtenir des modèles 3D approximatifs en quelques secondes en exploitant une architecture et un algorithme massivement parallèles. Les résultats présentés et la discussion ont mis en avant les avantages et les limites de cette méthode basée sur un test de photoconsistance rapide, mais nécessitant une mesure précise du déplacement des caméras dans la scène.

Les résultats obtenus sont encourageants et mettent en avant l'intérêt d'un algorithme de reconstruction volumétrique pour son adéquation avec les architectures massivement parallèles qui sont amenées à se généraliser dans les années à venir.

Chapitre 5

Conclusion générale et perspectives

Conclusion

Ce mémoire a présenté une synthèse des travaux de thèse mettant en oeuvre une architecture de traitement massivement parallèle dans le but d'obtenir en temps réel un modèle 3D d'une scène inconnue environnant un robot mobile grâce à un capteur de stéréovision panoramique.

Un état de l'art des méthodes existantes de reconstruction 3D à partir d'images ainsi qu'une étude des différentes familles de microprocesseurs potentiellement adaptées à cette tâche ont fait l'objet des deux premiers chapitres.

Les performances, proches du temps réel, sont atteintes tout d'abord par l'utilisation d'un capteur de vision innovant permettant d'acquérir des images panoramiques à cadence vidéo, puis par l'utilisation d'une méthode de reconstruction adaptée à une architecture de calcul massivement parallèle. L'étroit couplage entre l'algorithme et l'architecture permet une exécution rapide du traitement et les méthodes employées limitent au maximum les interdépendances de données aboutissant à des pertes de parallélisme et donc de performances.

La méthode proposée permet d'obtenir en quelques secondes un modèle volumétrique de la scène estimant la géométrie réelle de celle-ci. Au fur et à mesure du déplacement des caméras dans la scène, de nouvelles vues peuvent être utilisées pour affiner et améliorer le modèle 3D sans nécessiter une conservation de l'ensemble des images acquises. L'expérimentation sur un ensemble d'images de synthèses et d'image réelles ont permis de valider l'approche globale proposée ici.

Perspectives

La méthode implémentée lors de ces travaux de thèse atteint globalement les objectifs fixés, certains points restent cependant à améliorer.

Pour améliorer encore le temps de traitement il serait nécessaire de remplacer l'estimation de la visibilité des voxels, fonction qui introduit des interdépendances dans les calculs, par un processus local sur les données. Cette solution permettrait de paralléliser encore

plus le processus de reconstruction en supprimant les dernières portions séquentielles. Une telle solution reste cependant à développer.

La qualité du modèle 3D reconstruit, bien que suffisante pour les applications les moins exigeantes, est l'un des points à améliorer. L'utilisation de critères plus sophistiqués en complément ou à la place de la photoconsistance devrait permettre d'atteindre une qualité photoréaliste ; la difficulté étant de parvenir à les intégrer dans le processus massivement parallèle sans pénaliser de façon importante les performances.

Le dernier point nécessitant une amélioration notable concerne la sensibilité de l'algorithme aux erreurs de positionnement des caméras. L'utilisation d'une méthode complémentaire d'estimation de cette position grâce aux image pourrait être envisagée, comme par exemple une méthode de SLAM (*Simultaneous Location and Mapping*). Un tel système serait une amélioration majeure de notre méthode en permettant une réelle autonomie du robot mobile.

En complément de notre méthode de reconstruction 3D et pour être en mesure de fournir les informations de haut niveau nécessaire à la réalisation de tâches complexes de façon automatique, il est nécessaire de développer un programme de segmentation de l'espace 3D et de classification des objets s'y trouvant. Réaliser cette classification au fur et à mesure du processus incrémentale de reconstruction permettrait d'ouvrir de nouvelles perspectives en terme de reconnaissance et d'identification des objets présents dans la scène.

En dernier lieu, il convient de souligner la forte dynamique actuelle du marché des processeurs massivement parallèles dont la famille, déjà grande, ne cesse de se développer. Aujourd'hui présent dans tous les ordinateurs personnels, ces composant se généralisent actuellement dans l'informatique nomade (ordinateurs portables, téléphones portables...) et les systèmes embarqués (automobile, aéronautique, naval...). Ces nouvelles architectures amènent de nouvelles opportunités d'études et de développement afin d'améliorer encore la performance des systèmes de vision intelligents.

Communications

Les travaux réalisés durant cette thèse de doctorat ont donnés lieu à deux communications lors de conférences internationales :

- «3D volumetric reconstruction with a catadioptric stereovision sensor », *IEEE International Symposium on Industrial Electronics* (2008) [RRS+08]
- «Real-Time 3D Reconstruction for Mobile Robot Using Catadioptric Cameras », *IEEE International Workshop on Robotic and Sensors Environments* (2009) [RSEM09]

Bibliography

[AMWF08] S. Ainouz, O. Morel, N. Walter, et D. Fofi, "Mirror-adapted matching of catadioptric images," dans *IEEE International Conference on Image Processing (ICIP)*, 2008.

[Bre65] J. E. Bresenham, "Algorithm for computer control of a digital plotter," *IBM Systems Journal*, vol. 4, no. 1, p. 25–30, January 1965.

[BSEM08] R. Boutteau, X. Savatier, J.-Y. Ertaud, et B. Mazari, "An omnidirectional stereoscopic system for mobile robot navigation," *International Workshop on Robotic and Sensors Environments, 2008.*, pp. 138–143, 2008.

[CC04] S. Chambon et A. Crouzil, "Mesures de corrélation pour des images couleur," *Traitement du Signal*, vol. 21, no. 6, pp. 635–659, 2004.

[CKBH00] G. K. Cheung, T. Kanade, J.-Y. Bouguet, et M. Holler, "A real time system for robust 3d voxel reconstruction of human motions," *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, vol. 2, pp. 2714–2720, 2000.

[CMS00] W. Culbertson, T. Malzbender, et G. Slabaugh, "Generalized voxel coloring," dans *Vision Algorithms: Theory and Practice*, ser. Lecture Notes in Computer Science, B. Triggs, A. Zisserman, et R. Szeliski, Eds. Springer Berlin / Heidelberg, 2000, vol. 1883, pp. 100–115. [En ligne]. url: http://dx.doi.org/10.1007/3-540-44480-7_7

[CUD10] *NVIDIA CUDA C Programming Guide*, 2010.

[CW76] H. J. Curnow et B. A. Wichmann, "A synthetic benchmark," *The Computer Journal*, vol. 19, no. 1, pp. 43–49, January 1976. [En ligne]. url: http://dx.doi.org/10.1093/comjnl/19.1.43

[DMB04] K. Daniilidis, A. Makadia, et T. Bülow, "Image processing in catadioptric planes: Spatiotemporal derivatives and optical flow computation," *LFA'04, France*, pp. 3–10, 2004.

[Fly72] M. J. Flynn, "Some computer organizations and their effectiveness," *IEEE Transactions on Computers*, vol. 21, no. 9, pp. 948–960, September 1972.

[GD00] C. Geyer et K. Daniilidis, “A unifying theory for central panoramic systems and practical applications,” dans *ECCV ’00: Proceedings of the 6th European Conference on Computer Vision-Part II.* London, UK: Springer-Verlag, 2000, pp. 445–461. [En ligne]. url: http://portal.acm.org/citation.cfm?id=645314.649434

[GMDH08] N. GAC, S. Mancini, M. Desvignes, et D. Houzet, “High speed 3d tomography on cpu, gpu, and fpga,” *EURASIP Journal on Embedded Systems*, vol. 2008, pp. 1–12, 2008.

[GNT98] J. Gluckman, S. K. Nayar, et K. J. Thoresz, “Real-time omnidirectional and panoramic stereo,” dans *In Proceedings of the 1998 DARPA Image Understanding Workshop*, vol. 1, 1998, pp. 299–303. [En ligne]. url: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.29.6913

[GP88] M. Gervautz et W. Purgathofer, “A simple method for color quantization: Octree quantization,” dans *New Trends in Computer Graphics.* Springer Verlag, Berlin, 1988. [En ligne]. url: http://www.cg.tuwien.ac.at/research/publications/1988/gervautz-1988-simpl/

[HK06] A. Hornung et L. Kobbelt, “Robust and efficient photo-consistency estimation for volumetric 3d reconstruction,” dans *European Conf. on Computer Vision*, vol. 2, May 2006, pp. 179–190. [En ligne]. url: http://www-i8.informatik.rwth-aachen.de/index.php?id=30

[Hor89] B. K. P. Horn, “Obtaining shape from shading information,” dans *Shape from shading*, B. K. P. Horn, Ed. Cambridge, MA, USA: MIT Press, 1989, pp. 123–171. [En ligne]. url: http://dl.acm.org/citation.cfm?id=93871.93877

[LBN08] A. Ladikos, S. Benhimane, et N. Navab, “Efficient visual hull computation for real-time 3d reconstruction using cuda,” dans *Proceedings of the 2008 Conference on Computer Vision and Pattern Recognition Workshops*, Anchorage, AK, June 2008, pp. 1–8.

[Lop02] M. Loper. (2002) Archimedes - shape reconstruction from pictures; a generalized voxel coloring implementation. [En ligne]. url: http://gvc.sourceforge.net/

[MFA09] N. Muhammad, D. Fofi, et S. Ainouz, “Current state-of-the-art of vision-based slam,” dans *IS&T/SPIE Electronic Imaging - Image Processing: Machine Vision Applications II*, 2009.

[MR07] C. Mei et P. Rives, “Single view point omnidirectional camera calibration from planar grids,” dans *Int. Conf. on Robotics and Automation (ICRA’07)*, 2007, pp. 3945–3950.

[NB97] S. Nayar et S. Baker, “Catadioptric image formation,” dans *Proc. of the DARPA Image Understanding Workshop*, May 1997, pp. 1431–1437.

[OYA05] O. Ozun, U. Yilmaz, et V. Atalay, "Comparison of photoconsistency measures used in voxel coloring," dans *ISPRS Workshop in conjunction with 10th IEEE ICCV*, 2005.

[Pan09] *Panoscan*, 2009. [En ligne]. url: http://www.panoscan.com/MK3/

[PD98] A. C. Prock et C. Dyer, "Towards real-time voxel coloring," dans *DARPA Image Understanding Workshop*, 1998.

[Rag09] N. Ragot, "Conception d'un capteur de stéréovision omnidirectionnelle: architecture, étalonnage et applications à la reconstruction de scènes 3d," Thèse de doctorat, Université de Rouen, IRSEEM Institut de Recherche en Systèmes Electroniques Embarqués, 2009. [En ligne]. url: http://tel.archives-ouvertes.fr/tel-00417963

[RESM05] N. Ragot, J. Ertaud, X. Savatier, et B. Mazari, "Modelling of the environment of a mobile robot by catadioptric sensors. state of the art and proposal for an innovative architecture," dans *Best of Book AMSE'05 Conferences*, 2005.

[RRS+08] N. Ragot, R. Rossi, X. Savatier, J.-Y. Ertaud, et B. Mazari, "3d volumetric reconstruction with a catadioptric stereovision sensor," dans *IEEE International Symposium on Industrial Electronics*, 2008.

[RSEM09] R. Rossi, X. Savatier, J.-Y. Ertaud, et B. Mazari, "Real-time 3d reconstruction for mobile robot using catadioptric cameras," dans *IEEE International Workshop on Robotic and Sensors Environments*, 2009.

[SC05] O. Strauss et F. Comby, "Opérations morphologiques floues à noyaux variables pour images omnidirectionnelles à point de vue unique," *Traitement du Signal*, vol. 22, no. 5, pp. 1–22, 2005. [En ligne]. url: http://papyrus.lirmm.fr/Document.htm&numrec=031999735917150

[SCMS01] G. Slabaugh, B. Culbertson, T. Malzbender, et R. Schafer, "A survey of methods for volumetric scene reconstruction from photographs," dans *International Workshop on Volume Graphics*, 2001, pp. 81–100. [En ligne]. url: http://www.hpl.hp.com/personal/Tom_Malzbender/papers/VolReconSurvey.pdf

[SD97] S. Seitz et C. Dyer, "Photorealistic scene reconstruction by voxel coloring," dans *IEEE conference on Computer Vision and Pattern Recognition*, 1997, pp. 1067–1073.

[SD99] S. M. Seitz et C. R. Dyer, "Photorealistic scene reconstruction by voxel coloring," *International Journal of Computer Vision*, vol. 35, no. 2, pp. 151–173, November 1999.

[SGEB00] E. Steinbach, B. Girod, P. Eisert, et A. Betz, “3-d reconstruction of real-world objects using extended voxels,” dans *International Conference on Image Processing*, vol. 1, Vancouver, BC, Canada, September 2000, pp. 569–572.

[SS09] A. Schick et R. Stiefelhagen, “Real-time gpu-based voxel carving with systematic occlusion handling,” dans *Pattern Recognition*, ser. Lecture Notes in Computer Science, J. Denzler, G. Notni, et H. Süße, Eds. Springer Berlin / Heidelberg, 2009, vol. 5748, pp. 372–381.

[Tri98] J.-L. Tribillon, *Traitement optique de l’information & reconnaissance des formes par voie optique*, Tecknea, Ed. Tecknea, 1998.

[Wei89] R. P. Weicker, “Dhrystone benchmark (ada version 2): rationale and measurements rules,” *Ada Letters*, vol. IX, no. 5, pp. 60–62, 1989.

Nomenclature

µP Micro-Processeur

ASIC Application-Specific Integrated Circuit

CISC Complex Instruction Set Computer

DSP Digital Signal Processor

FPGA Field-Programmable Gate Array

GPU Graphics Processing Unit

MIMD Multiple instructions, Multiple Data

MISD Multiple Instructions, Single Data

MPMD Multiple Processes, Multiple Data

NUMA Non-Uniform Memory Access

PC Personnal Computer

RISC Reduced Instruction Set Computer

SIMD Single Instruction, Multiple Data

SISD Single Instruction, Single Data

SMP Symmetric Multi-Processing

SPMD Single Process, Multiple Data

VLIW Very Long Instruction Word

Annexes

Appendix A

Modèle de la sphère d'équivalence

Le modèle utilisé pour calibrer nos caméras catadioptriques repose sur l'équivalence entre la projection sur une surface quadrique et la projection sur une sphère. Geyer a démontré qu'il est possible de modéliser les capteurs centraux en utilisant une projection sur une sphère suivie d'une projection sur un plan à partir d'un point dépendant de la forme et des paramètres du miroir [GD00]. Ce théorème de l'équivalence projective, a donné son nom à la sphère unitaire correspondante: la sphère d'équivalence.

Le modèle initialement proposé par Geyer et Barreto a ensuite été repris et légèrement amélioré. Ainsi, le modèle proposé par Mei [MR07] (Figure A.1) est particulièrement intéressant puisqu'il ajoute au modèle initial des coefficients de distorsion pour corriger les imperfections des objectifs et de l'assemblage caméra/miroir. Dans ce modèle, le système catadioptrique est considéré comme un capteur unique et non pas comme l'association d'une caméra et d'un miroir, grâce à l'introduction de la focale généralisée qui dépend des paramètres de ces deux entités.

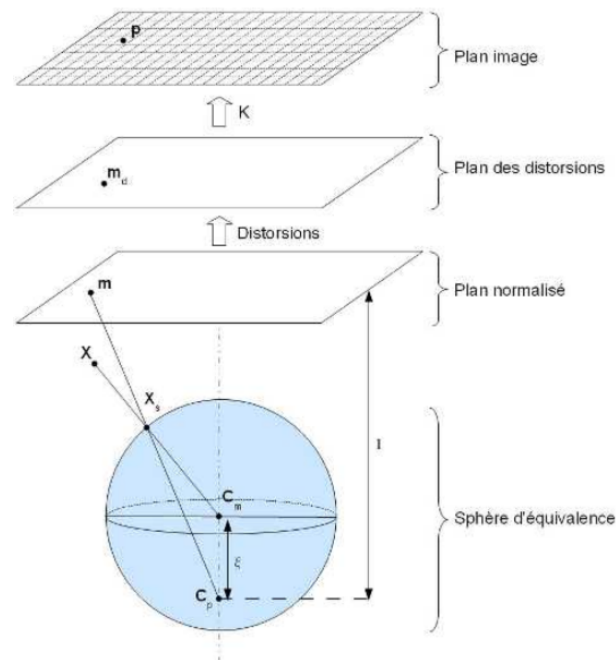


Figure A.1 – Modèle unifié proposé par Mei et utilisé dans nos travaux.

Les repères utilisés sont présentés sur la figure A.2. Pour le repère lié au capteur, l'axe x pointe vers la droite et l'axe y vers l'avant du robot. L'axe z quand à lui est vertical et pointe vers le haut. Concernant le repère-image, l'origine choisie est située en bas à gauche, l'axe u et horizontal, l'axe v vertical; l'origine étant située en bas et à gauche de l'image.

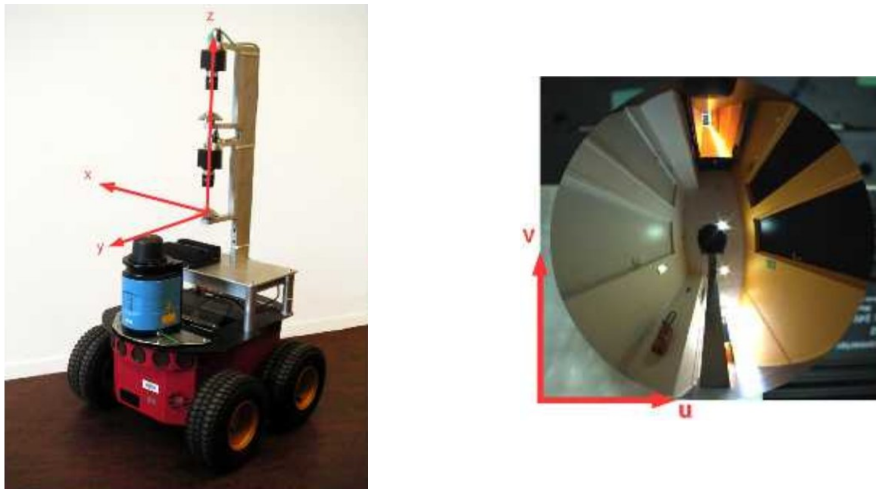


Figure A.2 – Repères utilisés pour le capteur (à gauche) et sur l'image (à droite).

Pour calibrer le système de vision, des images de mires sont acquises. Chaque mire possédant des caractéristiques connues (elles représentent un damier), ces informations sont ensuite utilisées pour estimer les paramètres du modèle présenté précédemment. Ce processus de calibration est présenté en détails dans [BSEM08].

Appendix B

Algorithme de tracé de segment de Bresenham

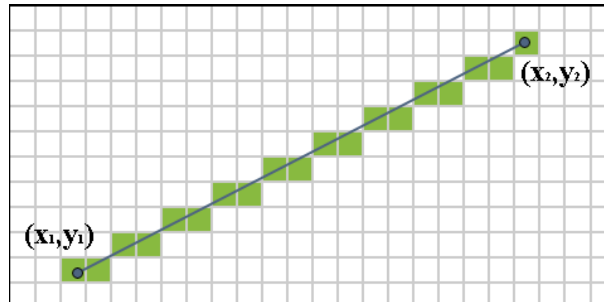


Figure B.1 – Exemple de tracé de segment sur une grille 2D

L'algorithme de Bresenham [Bre65] détermine quels sont les points d'un plan discret qui doivent être tracés afin de former une approximation de segment de droite entre deux points donnés. Cet algorithme est souvent utilisé pour dessiner des segments de droites sur l'écran d'un ordinateur ou une image calculée pour l'impression. Il est considéré comme l'un des premiers algorithmes inventé dans le domaine de la synthèse d'image.

La version classique de cet algorithme est utilisée pour tracer un segment entre deux points quelconques du plan, comme l'illustre la figure B.1. Une variante de cet algorithme permet de tracer des cercles.

Dans notre application nous avons utilisé une version capable de tracer des segments dans un espace discret à 3 dimensions. Le listing B.1 présente l'implémentation que nous avons utilisée.

Listing B.1 – Algorithme de tracé de segments en 3D

```
1  /** Structure stockant les informations utiles à l'itération
2   */
3  struct iterator3D {
4      uint3 v;        // Voxel en cours
```

```
5       int cumula;     // Erreur cumulée
6       int3 d;         // Delta (vecteur à parcourir)
7       int cumulb;     // Erreur cumulée
8       char3 inc;      // Incrément à ajouter à chaque pas (+/- 1)
9       int i;          // compteur de boucles (var temp)
10      char axis;      // Indique la coordonnée principale d'itération (x,y ou z)
11  };
12
13  /** Fonction d'initialisation de l'itérateur
14
15      L'itérateur est initialisé avec les coordonnées du point de départ
16      et d'arrivée.
17      @param start Indice 3D du point de départ
18      @param stop Indice 3D du point d'arrivée
19  */
20  void init_iterator3D(iterator3D & it, uint3 start, uint3 stop)
21  {
22      it.v = start;
23      it.d = make_int3(stop.x - start.x, stop.y - start.y, stop.z - start.z);
24      it.inc.x = (it.d.x > 0) ? 1 : -1;
25      it.inc.y = (it.d.y > 0) ? 1 : -1;
26      it.inc.z = (it.d.z > 0) ? 1 : -1;
27      it.d = abs(it.d);
28
29      if ((it.d.x >= it.d.y) && (it.d.x >= it.d.z)) {
30          it.cumula = it.d.x / 2;
31          it.cumulb = it.d.x / 2;
32          it.axis = 'x';
33      } else if (it.d.y >= it.d.z) {
34          it.cumula = it.d.y / 2;
35          it.cumulb = it.d.y / 2;
36          it.axis = 'y';
37      } else {
38          it.cumula = it.d.z / 2;
39          it.cumulb = it.d.z / 2;
40          it.axis = 'z';
41      }
42      it.i = 1;
43  }
44
45  /** Calcule l'indice 3D du point suivant sur la ligne 3D
46
47      Utilise les informations stockée dans la structure pour calculer
48      l'étape suivante du parcourt.
49
50      @param it Pointeur vers une structure de type iterator3D
51      initialisée
52      @param nextIdx Pointeur vers une variable prête à stocker les
53      coordonnées du point suivant
54      @return true si tout s'est bien passé, false si l'itération est
55      arrivée à la fin ou en cas d'erreur
56  */
```

```
57  bool next_vxl(iterator3D & it)
58  {
59      bool it_ok;
60      switch (it.axis) {
61      case 'x':
62          if (it.i < it.d.x) {
63              it.v.x += it.inc.x;
64              it.cumula += it.d.y;
65              it.cumulb += it.d.z;
66
67              if (it.cumula >= it.d.x) {
68                  it.cumula -= it.d.x;
69                  it.v.y += it.inc.y;
70              }
71
72              if (it.cumulb >= it.d.x) {
73                  it.cumulb -= it.d.x;
74                  it.v.z += it.inc.z;
75              }
76
77              it_ok = true;
78              it.i += 1;
79          } else {
80              it_ok = false;
81          }
82          break;
83
84      case 'y':
85          if (it.i < it.d.y) {
86              it.v.y += it.inc.y;
87              it.cumula += it.d.x;
88              it.cumulb += it.d.z;
89
90              if (it.cumula >= it.d.y) {
91                  it.cumula -= it.d.y;
92                  it.v.x += it.inc.x;
93              }
94
95              if (it.cumulb >= it.d.y) {
96                  it.cumulb -= it.d.y;
97                  it.v.z += it.inc.z;
98              }
99
100             it_ok = true;
101             it.i += 1;
102         } else {
103             it_ok = false;
104         }
105         break;
106
107     case 'z':
108         if (it.i < it.d.z) {
```

```
109             it.v.z += it.inc.z;
110             it.cumula += it.d.x;
111             it.cumulb += it.d.y;
112
113             if (it.cumula >= it.d.z) {
114                 it.cumula -= it.d.z;
115                 it.v.x += it.inc.x;
116             }
117
118             if (it.cumulb >= it.d.z) {
119                 it.cumulb -= it.d.z;
120                 it.v.y += it.inc.y;
121             }
122
123             it_ok = true;
124             it.i += 1;
125         } else {
126             it_ok = false;
127         }
128         break;
129     default:
130         return false;
131     }
132     return (it_ok);
133 }
```

Résumé

La reconstruction 3D d'une scène inconnue est un problème courant en vision par ordinateur. Les solutions classiques utilisant une paire de caméras en configuration stéréoscopique et un algorithme exploitant les disparités sur l'image ne permettent pas le calcul d'un modèle 3D dense régulièrement échantillonné. De plus, la réalisation de cette tâche en temps-réel est complexe et nécessite bien souvent une implémentation sur circuits spécialisés (FPGA ou DSP) puissants mais difficiles à mettre en oeuvre.

Dans ces travaux, nous proposons une méthode volumétrique ayant pour objectif une reconstruction en temps-réel d'un modèle 3D haute-résolution de l'environnement d'un robot mobile. Une paire de caméras catadioptriques permet l'acquisition panoramique de la scène. L'algorithme de reconstruction, adapté pour l'architecture massivement parallèle d'un processeur graphique (GPU), très puissant et peu coûteux, limite au maximum les dépendances de données menant à une séquentialisation du code et pénalisant les performances.

La méthode de reconstruction proposée ici permet en plus d'exploiter de nouvelles images obtenues au fur et à mesure du déplacement du robot dans la scène, améliorant ainsi incrémentalement le modèle volumétrique avec de nouvelles données.

Les résultats obtenus sont qualitativement proche de ceux obtenus par les méthodes classiques sur des scènes simples, mais notre approche permet une résolution du modèle 3D bien supérieure à l'état de l'art (500x500x200 voxels) tout en conservant une bonne rapidité d'exécution (environ 5 secondes par reconstruction). La cadence temps-réel (2 reconstructions par seconde) peut même être atteinte pour une résolution de modèle plus faible (150x150x150 voxels). Une expérimentation sur une scène réelle permet de valider l'approche développée avec une mise en situation réaliste.

Mots-clés: Vision par ordinateur, Caméra catadioptrique, Voxel Coloring, GPGPU, CUDA

Abstract

3D reconstruction of an unknown scene is a classical computer vision problem. Usual solutions, which use a pair of cameras in stereoscopic configuration and an algorithm relying on image disparities, don't allow to create a densely sampled 3D model. Moreover, processing this model in real-time is a complex task which often needs an implementation on dedicated hardware (FPGA or DSP), very powerful but hard to use.

In this thesis, we propose a volumetric reconstruction method aiming to produce a high-resolution 3D model of the scene surrounding a mobile robot. A pair of catadioptric cameras allows panoramic acquisition of the whole scene. The reconstruction algorithm, adapted for the massively-parallel architecture of a very powerful and inexpensive Graphical Processing Unit (GPU) tries to limit data-dependencies to improve performances.

This reconstruction method also benefit from additional pictures, taken as the robot moves in the scene, to incrementally improve the 3D model.

The final results are qualitatively equivalent to the ones obtained with classical methods, but our approach allows a 3D resolution far better (500x500x200 voxels) with a very short running time (about 5 seconds for each reconstruction). The real-time objective (2 reconstructions per second) can even be reached for a lower-resolution (150x150x150 voxels). Experimental results on a real image validate the proposed approach.

Keywords: Computer Vision, Catadioptric cameras, Voxel Coloring, GPGPU, CUDA