SIMON PERREAULT

Algorithmes pour la modélisation de l'apparence

Mémoire présenté
à la Faculté des études supérieures de l'Université Laval
dans le cadre du programme de maîtrise en génie électrique
pour l'obtention du grade de Maître ès sciences (M.Sc.)

FACULTÉ DES SCIENCES ET DE GÉNIE
UNIVERSITÉ LAVAL
QUÉBEC

2007

Résumé

Ce mémoire est divisé en deux parties indépendantes.

La première propose un nouvel algorithme de filtrage médian, lequel est une pierre angulaire dans plusieurs situations de traitement d'image. Cependant, son utilisation a souvent été entravée par sa complexité algorithmique de $O(r)$ selon le rayon de la fenêtre. Avec la tendance vers des images de meilleure résolution et des fenêtres de filtre proportionnellement plus larges, le besoin pour un algorithme de filtrage médian efficient devient pressant. Dans ce mémoire, un nouvel algorithme, simple mais bien plus rapide, affichant une complexité en temps d'exécution de $O(1)$, est décrit et analysé. Sa performance est mesurée et comparée à celle des algorithmes précédents. Des extensions vers des données de plus haute dimensionnalité ou précision, ainsi qu'une approximation de fenêtre circulaire, sont aussi présentées.

Dans la seconde partie, un système de modélisation de l'apparence d'objets est présenté. Il consiste en un robot à câbles déplaçant un appareil photo numérique autour du sujet en en prenant un grand nombre de photos. Le traitement informatique subséquent doit accomplir trois grandes tâches : segmentation, positionnement 3-D des photos et reconstruction de l'enveloppe visuelle de l'objet. Dans ce mémoire, les algorithmes impliqués sont analysés, critiqués et des améliorations sont proposées. En particulier, des algorithmes efficients pour le calcul d'enveloppe visuelle et pour la méthode des K plus proches voisins sont présentés. Le système dans son ensemble est finalement jugé selon les objectifs de conception préétablis et des recommandations sont faites afin d'en améliorer les performances.

Abstract

This dissertation is divided into two independent parts.

The first one proposes a new median filter algorithm. The median filter is one of the basic building blocks in many image processing situations. However, its use has long been hampered by its algorithmic complexity of $O(r)$ in the kernel radius. With the trend toward larger images and proportionally larger filter kernels, the need for a more efficient median filtering algorithm becomes pressing. In this dissertation, a new, simple yet much faster algorithm exhibiting $O(1)$ runtime complexity is described and analyzed. It is compared and benchmarked against previous algorithms. Extensions to higher-dimensional or higher-precision data and an approximation to a circular kernel are presented as well.

In the second part, a system for modeling object appearances is presented. It consists in a cable robot moving a digital camera around the subject while taking a high number of photos of it. The ensuing computer processing must accomplish three tasks : segmentation, 3-D positioning of the photos, and reconstruction of the object's visual hull. In this dissertation, the implied algorithms are analyzed, criticized, and improved. In particular, efficient algorithms for computing the visual hull and for performing the K nearest neighbors method are presented. The system as a whole is finally judged according to the preestablished design objectives and recommendations are made in order to improve its performances.

Table des matières

Première partie

Filtrage médian en temps constant

Chapitre 1

Filtrage médian en temps constant[1]

Le filtre médian est une pierre angulaire dans plusieurs situations de traitement d'image. Cependant, son utilisation a souvent été entravée par sa complexité algorithmique de $O(r)$ selon le rayon de la fenêtre. Avec la tendance vers des images de meilleure résolution et des fenêtres de filtre proportionnellement plus larges, le besoin pour un algorithme de filtrage médian efficient devient pressant. Dans ce chapitre, un nouvel algorithme, simple mais bien plus rapide, affichant une complexité en temps d'exécution de $O(1)$, est décrit et analysé. Sa performance est mesurée et mise en rapport aux algorithmes précédents. Des extensions vers des données de plus haute dimensionnalité ou précision, ainsi qu'une approximation de fenêtre circulaire, sont aussi présentées.

1.1 Introduction

Dans un filtre médian [42], chaque pixel de l'image de sortie est la médiane de la liste des pixels de l'image d'entrée situés à l'intérieur d'un rayon r du pixel d'entrée correspondant. Les canaux (par exemple, RGB) sont traités indépendamment. C'est une opération canonique de traitement d'image, bien connue pour son aptitude pour l'élimination de bruit poivre et sel. Ce filtre est aussi la fondation sur laquelle des filtres d'image plus avancés, tels le masquage flou, le traitement par ordre de rang et les opérations morphologiques sont bâtis [31]. Les applications de plus haut niveau incluent la segmentation d'objet, la reconnaissance de la parole et de l'écriture ainsi que l'imagerie médicale. La figure 1.5 montre un exemple de son application sur une

[1] Ce chapitre est l'objet d'un article accepté pour publication dans *IEEE Transactions on Image Processing*. [36]

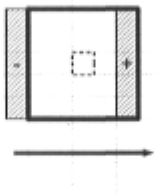


FIG. 1.1: Dans l'algorithme $O(n)$ de Huang, $2r + 1$ pixels doivent être ajoutés à et soustraits de l'histogramme de la fenêtre en se déplaçant d'un pixel au suivant. Dans cette figure, $r = 2$.

image à haute résolution.

Cependant, l'utilité du filtre médian a longtemps été limitée par le temps de traitement qu'il requiert. Le fait d'être non linéaire et non séparable le rend inadéquat pour les techniques d'optimisation communes. Une approche par force brute construit simplement une liste des valeurs des pixels situés dans la fenêtre du filtre et l'ordonne. La médiane est alors la valeur située au milieu de la liste. Dans le cas général, la complexité par pixel de cet algorithme est $O(r^2 \log r)$, où r est le rayon de la fenêtre. Quand le nombre de valeurs possibles pour un pixel est une constante, comme c'est le cas pour des images à 8 bits, on peut utiliser un tri par compartiments, ce qui ramène la complexité à $O(r^2)$. Ceci est encore impraticable sauf pour les plus petites fenêtres.

L'algorithme classique [21], utilisé dans virtuellement toutes les implémentations disponibles publiquement, affiche une complexité de $O(r)$ (voir l'algorithme 1.1). Il fait usage d'un histogramme pour accumuler les pixels de la fenêtre. Seulement une partie de celle-ci est modifiée en se déplaçant d'un pixel à un autre. Tel qu'illustré à la figure 1.1, $2r + 1$ additions et $2r + 1$ soustractions doivent être effectuées pour mettre à jour l'histogramme. Calculer la médiane d'un histogramme se fait en temps constant en sommant les valeurs à partir d'un des deux bouts et en arrêtant lorsque la somme atteint $(2r + 1)^2/2$. Pour des images à 8 bits, un histogramme est constitué de 256 compartiments et donc 128 comparaisons et 127 additions seront nécessaires en moyenne. Notez que n'importe quelle autre statistique de rang peut être calculée de la même manière en changeant la valeur d'arrêt.

Des efforts ont été faits pour réduire la complexité du filtre médian en deçà de linéaire. Une complexité de $O(\log^2 r)$ a été atteinte par Gil et al. [15] en utilisant un algorithme basé sur un arbre. Dans le même article, on démontre une borne inférieure de $O(\log r)$ pour n'importe quel algorithme de filtrage médian 2-D. La méthode proposée est similaire à celle de [7], où des listes ordonnées étaient utilisées au lieu d'his-

Algorithme 1.1 L'algorithme de filtrage médian $O(n)$ de Huang.

Entrée: Image X de taille $m \times n$, rayon r de la fênetre
Sortie: Image Y de même taille que X

```
Initialiser l'histogramme H de la fenêtre
pour i = 1 à m faire
  pour j = 1 à n faire
    pour k = −r à r faire
      Enlever X_{i+k,j−r−1} de H
      Ajouter X_{i+k,j+r} à H
    fin pour
    Y_{i,j} ← median(H)
  fin pour
fin pour
```

togrammes. Cette dernière démontrait une complexité de $O(r^2)$ et était relativement lente. Plus récemment, Weiss [43] a développé une méthode dont le temps de calcul est de $O(\log r)$ en utilisant une hiérarchie d'histogrammes. Dans son approche, même si la *complexité* a été diminuée, la *simplicité* a été perdue. Un algorithme simple et efficace, applicable aussi bien aux processeurs génériques qu'au matériel conçu sur mesure, serait désirable.

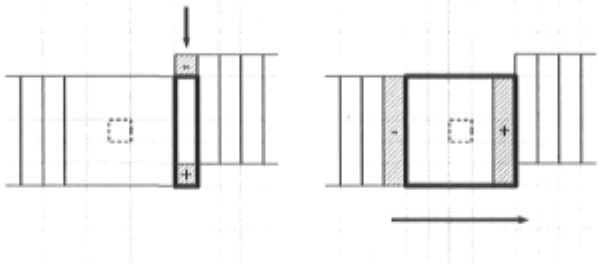
1.2 D'une complexité $O(n)$ à $O(1)$

Pour bien comprendre l'approche proposée, il est premièrement utile de premièrement faire ressortir l'inefficience dans l'algorithme de Huang [21] (Algorithme 1.1). Spécifiquement, remarquez qu'aucune information n'est conservée entre les rangées de l'image. Chaque pixel doit être ajouté à et enlevé de $2r + 1$ histogrammes au cours du traitement de toute l'image, ce qui cause la complexité $O(r)$. Intuitivement, on peut deviner qu'il faudra accumuler chaque pixel au plus un nombre constant de fois pour obtenir une complexité $O(1)$. Comme nous le verrons, ceci devient possible quand l'information est retenue entre les rangées.

Introduisons une propriété des histogrammes, celle de la distributivité [43]. Pour des régions disjointes A et B,

$$H(A \cup B) = H(A) + H(B).$$

Remarquez qu'additionner des histogrammes est une opération $O(1)$ par rapport au nombre de pixels accumulés. Elle ne dépend que de la taille de l'histogramme, laquelle



(a) (b)

FIG. 1.2: Les deux étapes de l'algorithme proposé. (a) L'histogramme colonne à la droite est déplacé vers le bas d'une rangée en lui ajoutant un pixel et en lui en retirant un autre. (b) L'histogramme de la fenêtre est mis à jour en lui ajoutant l'histogramme colonne modifié et en soustrayant celui le plus à gauche.

est elle-même fonction de la précision en bits de l'image. Cette observation faite, on peut passer à un nouvel algorithme $O(1)$.

L'algorithme proposé maintient à jour un histogramme pour chaque colonne de l'image, en plus de l'unique histogramme de la fenêtre. Cet ensemble d'histogrammes est préservé de rangée en rangée durant tout le processus. Chaque histogramme colonne accumule $2r + 1$ pixels adjacents et est initialement centré sur la première rangée de l'image. Quant à l'histogramme de la fenêtre, il est obtenu par la somme de $2r + 1$ histogrammes colonne adjacents. Ce que nous venons de faire est de séparer l'histogramme de la fenêtre en l'union de ses colonnes, chacune représentée par son propre histogramme. Lors du filtrage de l'image, tous les histogrammes peuvent être tenus à jour en temps constant suivant une approche en deux étapes.

Considérons le cas du déplacement vers la droite d'un pixel au prochain. Les histogrammes colonne à la droite de la fenêtre ne sont pas encore traités pour la rangée courante et sont donc centrés une rangée plus haut. La première étape consiste en la mise à jour de l'histogramme colonne à la droite de la fenêtre en lui soustrayant le pixel du haut et en lui ajoutant un nouveau pixel en-dessous (figure 1.2a). Ceci a pour effet d'abaisser l'histogramme colonne d'une rangée. Cette première étape est clairement $O(1)$ puisque seules une addition et une soustraction, indépendamment du rayon de la fenêtre du filtre, doivent être effectuées.

La deuxième étape consiste à déplacer l'histogramme de la fenêtre, qui est la somme de $2r + 1$ histogrammes colonne, d'un pixel vers la droite. Ceci est accompli en soustrayant son histogramme colonne le plus à gauche et en lui ajoutant l'histogramme colonne abaissé à la première étape (figure 1.2b). Cette deuxième étape est aussi $O(1)$. Tel qu'énoncé plus tôt, l'addition, la soustraction et le calcul de la médiane d'histogrammes

comportent un nombre d'opérations dépendant de la précision en bits de l'image, pas du rayon de la fenêtre du filtre.

L'effet net est que l'histogramme de la fenêtre se déplace vers la droite alors que les histogrammes colonne se déplacent vers le bas. On peut visualiser le tout comme une fermeture éclair se fermant vers le bas. Chaque pixel est visité une seule fois et est ajouté à un seul histogramme. La dernière étape pour chaque pixel est le calcul de la médiane. Tel qu'énoncé plus tôt, ceci est $O(1)$ grâce à l'histogramme.

Algorithme 1.2 L'algorithme de filtrage médian $O(1)$ proposé.

Entrée: Image X de taille $m \times n$, rayon r de la fenêtre
Sortie: Image Y de même taille

```
Initialiser l'histogramme H de la fenêtre et les histogrammes colonne h_{1...n}
pour i = 1 à m faire
  pour j = 1 à n faire
    Enlever X_{i-r-1,j+r} de h_{j+r}
    Ajouter X_{i+r,j+r} à h_{j+r}
    H ← H + h_{j+r} − h_{j−r−1}
    Y_{i,j} ← median(H)
  fin pour
fin pour
```

Toutes les opérations par-pixel (mettre à jour les histogrammes colonne et de fenêtre et calculer la médiane) sont $O(1)$. Maintenant nous abordons la question de l'initialisation, laquelle consiste en l'accumulation des r premières rangées dans des histogrammes colonne et le calcul de l'histogramme de la fenêtre d'après les r premiers histogrammes colonne.[2] Ceci résulte en une initialisation $O(r)$. De plus, il y a une surcharge lors du déplacement d'une rangée à l'autre qui compte pour un autre terme $O(r)$. Cependant, malgré ces effets de bord, la complexité par pixel peut être considérée comme $O(1)$ *amortie* sous la contrainte aisément respectée que les dimensions de l'image soient $> kr$ pour un k quelconque. En conséquence, l'algorithme proposé (voir l'algorihme 1.2) a une complexité globale de $O(1)$, ce qui résulte en un temps de calcul indépendant du rayon de la fenêtre du filtre.

En résumé, les opérations pour un pixel en tons de gris de précision 8 bits sont les suivantes :

- 1 addition pour ajouter le nouveau pixel à l'histogramme colonne à la droite de la fenêtre.
- 1 soustraction pour enlever le vieux pixel de ce même histogramme colonne.
- 256 additions pour additionner le nouvel histogramme colonne à l'histogramme de la fenêtre.

[2] On suppose qu'un remplissage avec des zéros ou par répétition de la bordure est utilisé.

- 256 soustractions pour soustraire l'ancien histogramme colonne de l'histogramme de la fenêtre.
- 128 comparaisons et 127 additions, en moyenne, pour trouver la médiane de l'histogramme de la fenêtre.

La constante obtenue peut sembler excessive. Cependant, la plupart de ces opérations sont naturellement vectorisables, ce qui abaisse la constante de temps considérablement. Plus important, plusieurs optimisations peuvent être appliquées pour réduire le nombre d'opérations. Elles sont discutées à la prochaine section.

1.3 Notes d'implémentation

Dans cette section sont décrites quelques optimisations qui peuvent être appliquées pour augmenter la performance de l'algorithme proposé. Elles dépendent toutes du type particulier d'architecture de processeur sur laquelle le filtre est exécuté. En tant que telles, leur effet peut varier énormément (voire même réduire l'efficience dans quelques cas) d'un type de processeur à un autre. Notez aussi que les optimisations des sections 1.3.3 et 1.3.4 sont dépendantes des données.

1.3.1 Vectorisation

Les processeurs modernes rendent disponibles des instructions SIMD[3] qui peuvent être exploitées pour accélérer l'algorithme. L'énumération des opérations montre que la plupart du temps est passé dans l'addition et la soustraction d'histogrammes. Ceci peut être accéléré considérablement avec les jeux d'instructions MMX, SSE2 ou Altivec en traitant plusieurs compartiments en parallèle. Pour maximiser le nombre de compartiments d'histogrammes que l'on peut additionner en une instruction, chaque compartiment est représenté avec seulement 16 bits. De ce fait, la taille de la fenêtre est limitée à 2^{16} pixels, ce qui est acceptable pour un usage typique. Cette limite n'est pas intrinsèque à l'algorithme proposé : elle est seulement introduite pour fins d'optimisation.

Un autre endroit où le parallélisme peut être exploité est la lecture des pixels de

[3]L'acronyme SIMD signifie *Single Instruction Multiple Data*. Employé comme qualificatif d'une instruction de processeur, il souligne le fait que l'instruction agit sur plusieurs variables simultanément, habituellement en parallèle. Les processeurs modernes offrent souvent un jeu d'instructions SIMD dont une implantation de l'algorithme proposé peut tirer profit. Parmi ceux-ci, on retrouve MMX et SSE2 sur les processeurs de type IA32 et Altivec sur le PowerPC.

l'image et leur accumulation dans les histogrammes colonne. Au lieu d'alterner entre la mise à jour d'un histogramme colonne et de l'histogramme de la fenêtre, tel que décrit à la section 1.2, on peut traiter les histogrammes colonne pour toute la rangée de pixels en premier. En utilisant des instructions SIMD, on peut mettre à jour de multiples histogrammes colonne en parallèle. On procède ensuite avec l'histogramme de la fenêtre comme à l'habitude.

1.3.2 Respect de la cache

L'algorithme de filtrage médian en temps constant doit tenir en mémoire un histogramme pour chaque colonne. Pour toute l'image, ceci peut facilement s'élever à des centaines de kilo-octets, surpassant souvent la taille de la cache des processeurs modernes. Ceci mène a un accès à la mémoire centrale répété et inefficient, ce qui annule l'utilité de la cache. Un moyen de contourner cette limitation est de séparer l'image en bandes verticales qui sont traitées indépendamment. La largeur de chaque bande est choisie telle que les histogrammes remplissent la cache mais n'excèdent pas sa capacité. Un désavantage de cette modification est qu'elle amplifie les effets de bord. En pratique, elle résulte habituellement en une diminution importante du temps de calcul. Notez que le traitement simultané de bandes sur différents processeurs est un moyen facile de paralléliser l'algorithme proposé.

1.3.3 Histogrammes multiniveaux

Les histogrammes multiniveaux ont été analysés dans [1] et se sont révélés être une optimisation très efficace. L'idée est de tenir en mémoire un ensemble parallèle d'histogrammes plus petits accumulant seulement les bits de poids fort des pixels. Par exemple, il est commun d'utiliser des histogrammes à deux niveaux pour des images à 8 bits, où le niveau du haut a une largeur de 4 bits tandis que celui du bas contient les données sur la pleine largeur de 8 bits. Il est coutumier de les nommer niveaux *grossier* et *fin*, respectivement. Le niveau grossier comprendrait 16 éléments (2^4) et chacun serait la somme d'un segment de 16 éléments de l'histogramme du niveau fin correspondants.

Il y a deux avantages aux histogrammes multiniveaux, le premier étant l'accélération du calcul de la médiane. Au lieu d'examiner les 256 compartiments au complet, on peut maintenant faire des bonds de 16 éléments en trouvant la médiane au niveau grossier. Ceci nous donne le segment du niveau fin contenant la médiane. Au lieu d'une moyenne de 128 additions et comparaisons, 16 sont maintenant nécessaires (8 à chaque niveau)

pour atteindre la médiane. Le deuxième avantage est relié à l'addition et la soustraction d'histogrammes. On peut sauter un segment de 16 éléments du niveau fin quand la valeur correspondante du niveau grossier est zéro. Lorsque r est petit, les histogrammes colonnes ont une forte proportion de zéros et donc les branchements supplémentaires peuvent devenir rentables.

1.3.4 Mise à jour conditionnelle de la fenêtre

La séparation des histogrammes en des niveaux grossier et fin rend possible une optimisation un peu moins évidente mais très efficace. Remarquez que jusqu'à ce point, la majorité du temps de traitement était passé dans l'addition et la soustraction d'histogrammes colonne à l'histogramme de la fenêtre. Avec la mise à jour conditionnelle, ce temps est diminué en ne tenant à jour que le niveau grossier de la fenêtre tandis que le niveau fin est mis à jour sur demande seulement.

Rappelons-nous que le calcul de la médiane se fait en parcourant premièrement le niveau grossier, ce qui indique le segment de 16 éléments du niveau fin qui contient la médiane. Puisqu'un histogramme colonne contribue à au plus $2r+1$ calculs de médiane, au plus $2r + 1$ de ses segments du niveau fin seront utiles. Lorsque les valeurs de pixel varient lentement dans l'image, le nombre est plus bas puisqu'on accède au même segment répétitivement. Mettre à jour le niveau fin de l'histogramme de la fenêtre avec des segments qui ne seront jamais utilisés peut et devrait être évité.

Pour ce faire, on doit maintenir une liste des indices des colonnes où les segments ont été mis à jour pour la dernière fois. Lors du déplacement d'un pixel au suivant, on met à jour les deux niveaux du nouvel histogramme colonne mais seulement le niveau grossier de l'histogramme de la fenêtre. Ensuite, on calcule la médiane de l'histogramme de la fenêtre au niveau grossier et on détermine dans quel segment du niveau fin la médiane se trouve. On met ce segment fin à jour en traitant les histogrammes colonne depuis la dernière colonne où il avait été mis à jour. Si cette colonne est séparée de la colonne courante par plus de $2r + 1$ pixels, alors il n'y a pas de recouvrement entre la fenêtre placée à l'endroit courant et celle placée à l'ancien endroit. On met donc à jour le segment d'histogramme fin à partir de zéro, sautant de ce fait des colonnes. C'est dans ce cas, en sautant des colonnes, que l'on bénéficie par un temps de traitement réduit de la logique supplémentaire.

Il est aussi avantageux d'entrelacer les histogrammes en mémoire de façon à ce que les segments de colonnes adjacentes soient aussi adjacents en mémoire. C'est-à-dire que les compartiments d'histogrammes devraient être ordonnés premièrement par indice de

segment, ensuite par indice de colonne et ensuite par indice de compartiment. De cette façon, la mise à jour d'un segment du niveau fin de la fenêtre correspond à la sommation d'un bloc de mémoire contigu.

1.4 Réfutation de la borne inférieure de Gil-Werman

Une borne inférieure de $\Omega(\log r)$ pour la complexité du filtre médian 2-D a été introduite dans [15]. On y affirme que "*n'importe quel algorithme pour le calcul du filtre médian de taille r pour une entrée de $n \times n$ avec $n \geq (3r-1)/2$ s'exécute en $\Omega(\log r)$ temps amorti par élément.*" Ceci semble être en contradiction directe avec les résultats présentés : on a prouvé que l'algorithme est dans $O(1)$ alors que $\Omega(\log r) \cap O(1) = \emptyset$ par définition.[4]

Même si leur raisonnement est correct, il est basé sur une réduction à partir du tri. Les auteurs montrent que le filtre médian 2-D a le pouvoir de trier une entrée arbitraire. Pour ce faire, ils construisent une image 2-D à partir des éléments à ordonner en suivant un ordre particulier. Les auteurs montrent ensuite que l'on peut extraire du résultat du filtrage médian de cette image la liste ordonnée des éléments d'entrée. Le problème de tri étant $\Omega(\log r)$, la complexité de l'opération globale ne peut être inférieure à cette borne. Cette complexité ne peut pas être attribuée qu'au filtre médian puisque la construction de l'image et l'extraction du résultat se font toutes les deux en un temps constant par élément. On en déduit que le filtrage médian est un problème de complexité $\Omega(\log r)$.

Ce raisonnement est vrai *tant que l'on considère un algorithme de tri par comparaison.* Ceci peut être évité lorsque le nombre de valeurs possibles est une constante. Il est bien connu que les algorithmes de tri ne fonctionnant pas à base de comparaisons, desquels le tri par compartiment et le tri digital (de l'anglais *bucket sort* et *radix sort*) sont des exemples, ne sont pas assujettis à cette borne inférieure [23]. Le traitement par histogramme dont l'algorithme proposé fait usage est analogue à un tri des données sans comparaison. Le contre-exemple d'un filtre médian montrant un temps d'exécution $O(1)$ par élément invalide la borne inférieure de $\Omega(\log r)$. Il est aussi immédiatement reconnu comme la vraie limite inférieure puisque le temps de traitement par élément ne peut pas être inférieur à une constante. Il devrait tout de même être possible de

[4] On dit qu'une fonction f est dans $O(g)$ s'il existe une fonction g telle que $\limsup_{n\to\infty} \frac{f(n)}{g(n)} < \infty$. Cette relation entre deux fonctions définit un ordre partiel que l'on appelle l'ordre *asymtotique* des fonctions. On peut alors écrire $f \preceq g$ et associer à g le sens de *borne supérieure* de la complexité de f. À l'inverse, $f \in \Omega(g) \equiv f \succeq g$. Dans ce cas, g est une *borne inférieure* de la complexité de f. Pour plus de détails sur cette notation, voir [22].

diminuer cette constante avec de nouvelles idées d'optimisation.

1.5 Extensions

L'algorithme proposé peut être étendu à de nouvelles situations. Quelques-unes des plus communes sont explorées dans cette section.

1.5.1 Plus grande précision

Les images d'une précision autre que 8 bits peuvent être traitées tout aussi facilement par l'algorithme proposé. Un seul changement doit y être apporté : le nombre de compartiments des histogrammes doit être mis à l'échelle conformément. Ceci implique que l'addition et la soustraction d'histogrammes, de même que la recherche de médiane prendront conséquemment plus de temps. Il serait utile à un certain point de faire usage d'histogrammes à trois niveaux (ou plus) alors que leur taille augmente.

Des histogrammes plus longs occuperont aussi plus d'espace en mémoire, ce qui résultera en des bandes plus minces (voir section 1.3.2). Si cela devient un problème, la transformée ordinale [43] pourrait être d'une certaine utilité. Cependant, puisque la taille des histogrammes augmente suivant $O(2^b)$, où b représente la précision en bits de l'image, les précisions élevées représentent une faiblesse fondamentale de n'importe quel algorithme basé sur un histogramme.

1.5.2 Plus haute dimensionalité

Le filtrage médian de données à plus de deux dimensions est commun dans des champs comme l'imagerie médicale [5, 34] et le traitement vidéo [3]. L'algorithme proposé peut accomoder des données N-dimensionnelles en une complexité $O(1)$ au coût d'une utilisation de mémoire plus importante. Par exemple, l'algorithme 1.3 montre comment le filtrage médian 3-D est accompli. L'étendre à des dimensions plus élevées est simple.

Chaque nouvelle dimension requiert un nouvel ensemble d'histogrammes dont la taille est le produit des tailles des dimensions d'ordre inférieur. Cet accroissement exponentiel rend ceci impraticable pour des dimensions plus élevées. Cependant, il serait

Algorithme 1.3 Filtrage médian en temps constant pour données 3-D.

Entrée: Image X de taille $m \times n \times o$, rayon de fenêtre r
Sortie: Image Y de même taille

Initialiser l'histogramme H de la fenêtre, les histogrammes plans $h^1_{1\ldots o}$ et les histogrammes colonne $h^2_{1\ldots n,1\ldots o}$
pour $i = 1$ à m **faire**
  pour $j = 1$ à n **faire**
    pour $k = 1$ à o **faire**
      Enlever $X_{i-r-1,j+r,k+r}$ de $h^2_{j+r,k+r}$
      Ajouter $X_{i+r,j+r,k+r}$ à $h^2_{j+r,k+r}$
      $h^1_{k+r} \leftarrow h^1_{k+r} + h^2_{j+r,k+r} - h^2_{j+r,k-r-1}$
      $H \leftarrow H + h^1_{k+r} - h^1_{k-r-1}$
      $Y_{i,j,k} \leftarrow \text{median}(H)$
    fin pour
  fin pour
fin pour

toujours possible de traiter les données en hyper bandes (voir la section 1.3.2), ce qui permettrait d'imposer une limite arbitraire sur l'utilisation de mémoire au coût de l'augmentation de l'importance des termes linéaires causés par les effets de bord. Quant au temps d'exécution, on peut voir que la boucle intérieure est toujours $O(1)$ selon la taille de la fenêtre. Puisque chaque dimension requiert une sommation supplémentaire d'histogrammes, elle est $O(N)$ selon le nombre de dimensions. Il est intéressant de remarquer qu'un seul nouveau pixel de l'image source est lu pour chaque pixel de l'image de sortie à être calculé.

1.5.3 Approximation de fenêtre circulaire

Une extension populaire pour plusieurs filtres est une fenêtre appromativement circulaire. Une telle forme de fenêtre est plus près de la fenêtre théorique parfaitement circulaire telle que définie sur des données continues et minimise les artéfacts causés par une fenêtre carrée. L'algorithme proposé peut être étendu à une fenêtre octogonale, tel que montré à la figure 1.3. Cinq histogrammes, chacun correspondant à un côté de l'octogone, sont utilisés au lieu d'un seul histogramme colonne dans le cas de la fenêtre carrée. Ils peuvent être préservés de rangée en rangée d'une façon très similaire. Au lieu d'un seul ensemble d'histogrammes colonne, cinq ensembles sont maintenant nécessaires : un pour les côtés verticaux de part et d'autre, et quatre pour les diagonales. Notez que les diagonales nécessitent des ensembles dinstincts pour les côtés droit et gauche parce qu'ils ont une orientation différente.

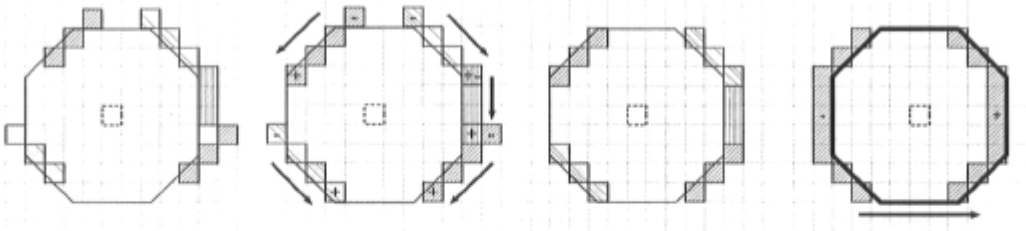


FIG. 1.3: Mouvement des histogrammes diagonaux et verticaux pour une fenêtre circulaire approximée par un octogone. (a) Disposition des 5 histogrammes une rangée au-dessus de la rangée courante. (b) Les histogrammes sont abaissés à la rangée courante. (c) Position des histogrammes lorsque centrés sur la rangée courante. (d) Fenêtre octogonale se déplaçant horizontalement.

L'algorithme est toujours $O(1)$, mais possède une constante plus élevée. Cinq ensembles d'histogrammes doivent être tenus à jour au lieu d'un seul dans le cas d'une fenêtre carrée. Déplacer l'histogramme de la fenêtre d'un pixel à l'autre requiert maintenant trois sommations et trois soustractions d'histogrammes au lieu d'une de chacune dans le cas d'une fenêtre carrée.

L'implantation réalisée du filtre octogonal n'est pas optimisée. On peut toutefois s'attendre à ce qu'une l'étant soit environ 5 fois plus lente que pour une fenêtre carrée. Il devrait être possible de concevoir de meilleures constructions géométriques qui diminueraient probablement la constante tout en maintenant l'algorithme $O(1)$. Comme autre exemple, les capteurs CCD tesselés hexagonalement, tels ceux produits par la compagnie Fuji, pourraient bénéficier d'une fenêtre hexagonale, laquelle serait construite en suivant un raisonnement similaire. L'important à retenir est que la complexité constante de l'algorithme proposé ne dépend pas d'une fenêtre rectangulaire.

1.6 Résultats

Le nouvel algorithme a été comparé avec Photoshop CS 2, lequel possède une implémentation de l'algorithme $O(r)$ classique de Huang, et avec Pixfoliate, un plugiciel pour Photoshop distribué par Weiss[5] et qui implémente son algorithme $O(\log r)$. Ce dernier est l'algorithme de filtrage médian 2-D le plus rapide à la connaissance de l'auteur. Le chronométrage a été conduit sur un PowerMac G5 1.6 GHz muni de Mac OS X 10.4. Une image RGB de 8 mégapixels (3504×2336) avec un contenu typique (montré à la

[5]http ://www.shellandslate.com/pixfoliatemacdemo.html

TAB. 1.1: Comparaison de l'efficience de la méthode proposée sur différents processeurs. ($r = 50$, image RGB de 8 mégapixels)

Processeur	Jeu d'instructions SIMD	Taille de la cache L2	Cycles d'horloge par élément traité
PowerMac G5 1.6 GHz	AltiVec	512 kB	102
Intel Core 2 Duo E6600	SSE2	4096 kB	153
AMD Sempron 2400+	MMX	256 kB	296
Intel Pentium 4 1.8 GHz	SSE2	256 kB	628

figure 1.5) a été filtrée avec des rayons de filtre de longueur variée. Les résultats sont affichés à la figure 1.4. On peut voir la trace plate générée par l'algorithme proposé, signalant sa complexité constante.

Malgré le fait que les optimisations des sections 1.3.3 et 1.3.4 sont dépendantes des données, le temps de traitement dans le pire des cas (pour du bruit uniforme) a été mesuré comme étant en moyenne seulement 3 % plus lent. D'autre part, le meilleur cas (pour une image complètement noire) a été traité deux fois plus rapidement. Le chronométrage pour un ensemble de photographies typiques a produit des résultats identiques à ceux de la figure 1.4. Le tableau 1.1 montre l'affinité de différents processeurs pour l'algorithme proposé. Les quatre optimisations de la section 1.3 étaient utilisées lors des tests sur ces processeurs.

Il peut sembler surprenant que l'algorithme $O(1)$ soit à ce point plus rapide que l'algorithme $O(r)$ pour les petites tailles de fenêtre. Par expérience, une réduction de complexité s'obtient souvent au prix d'une augmentation de la constante associée. Dans ce cas, l'algorithme proposé est à la fois moins complexe et plus efficient par une large marge à toutes les tailles de fenêtre lorsque comparé à l'algorithme classique. Il pourrait aussi être avancé qu'il soit plus simple en comparant les algorithmes 1.1 et 1.2 côte à côte.

Le nouvel algorithme performe mieux ou pire que celui $O(\log r)$ de Weiss selon la valeur de r. La différence relative maximale entre les deux est d'environ 33 % dans la plage testée de valeurs, avec un croisement à $r = 40$. Étant donné leurs chronométrages semblables, la différence repose en deux endroits. Premièrement, l'arbre d'histogrammes dans l'algorithme de Weiss rend son implémentation assez compliquée et généralement peu convenable pour du matériel fait sur mesure. De plus, puisqu'un arbre plus haut est requis pour un rayon plus grand, différentes implémentations doivent être générées, chacune étant chargée du traitement d'une portion de la plage de rayons possibles. En revanche, l'implémentation de l'algorithme proposé, incluant l'optimisation, fait 275 lignes

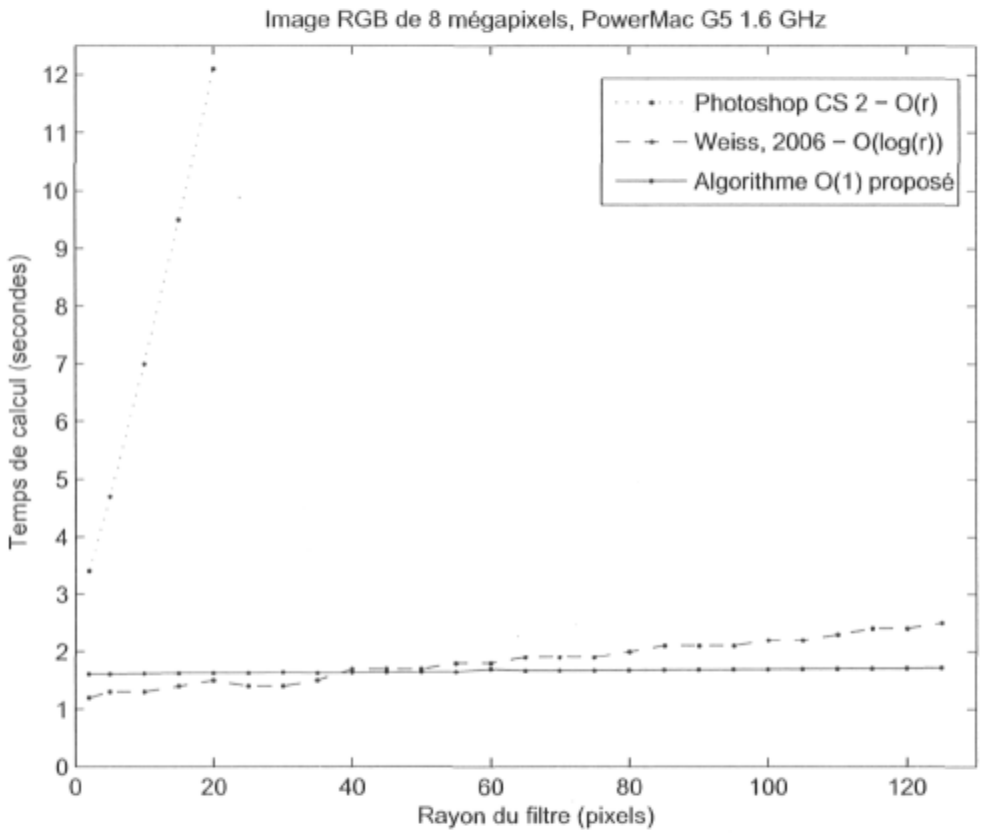


FIG. 1.4: Chronométrage de l'algorithme proposé.

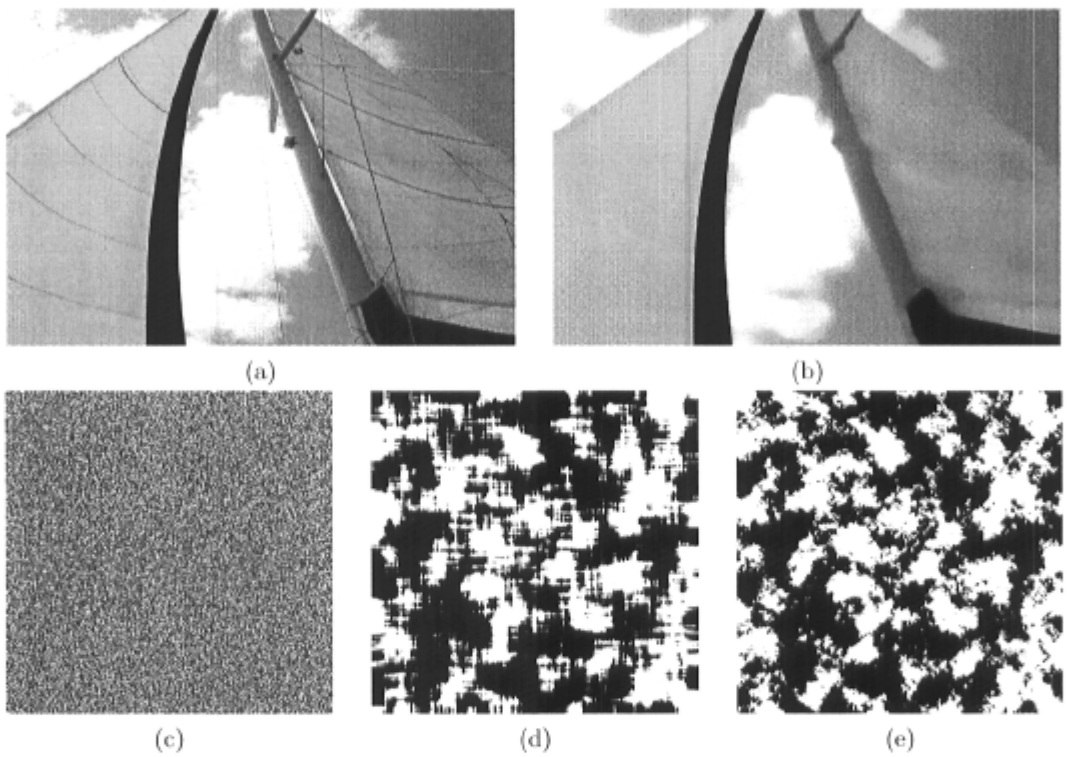


FIG. 1.5: Effet du filtrage médian. (a) Image RGB de 8 mégapixels originale. (b) La même image après application du filtre médian avec une fenêtre carrée de 50 pixels de rayon. Remarquez la netteté des arêtes alors que les petites structures ont été éliminées. (c) Image 512×512 de bruit binaire uniforme. (d) Bruit filtré avec une fenêtre carrée de 20 pixels de rayon. (e) Bruit filtré avec une fenêtre octogonale de 20 pixels de rayon. Remarquez la disparition des artéfacts horizontaux et verticaux de ligne.

de code C au total et accomode toute l'étendue de rayons. En second lieu, l'algorithme proposé a l'avantage d'une complexité constante. Ceci implique qu'il performera mieux qu'un autre d'une complexité plus élevée à mesure que le rayon de la fenêtre augmentera. Étant donné la tendance vers des images à résolution plus élevée, nécessitant également des filtres aux fenêtres plus grandes, ceci rend l'algorithme proposé paré au futur. Du matériel plus rapide avec de meilleures possibilités de vectorisation contribuera aussi à abaisser sa constante de temps.

1.7 Conclusion

Ce chapitre a présenté un algorithme de filtrage médian rapide, simple et dont le temps d'exécution et la quantité de mémoire sont $O(1)$ en fonction du rayon de la fenêtre. Quelques optimisations ont été proposées. Elles rendent cet algorithme aussi rapide que le plus rapide présentement disponible, à notre connaissance, tout en demeurant bien plus simple. En tant que tel, il convient à une implémentation basée sur un CPU ou sur du matériel fait sur mesure.

Des questions importantes concernant l'extensibilité de l'algorithme à de nouvelles situations ont trouvé réponse. En particulier, le filtrage de données de dimensionalité ou de précision plus élevée, de même qu'une approximation de fenêtre circulaire, ont été discutés. Il a été montré pourquoi la borne inférieure théorique de Gil-Werman de $\Omega(\log r)$ sur la complexité du filtre médian 2-D ne s'applique pas aux algorithmes traditionnels faisant usage d'histogrammes pour le tri des données.

Une implémentation en C de l'algorithme proposé est disponible gratuitement sur le web[6] et a été incluse dans la librairie de vision numérique populaire et gratuite OpenCV[7]. On peut être confiant que de nouvelles optimisations ingénieuses vont contribuer à abaisser sa constante de temps. On peut aussi espérer que la simplicité, la vitesse et l'adaptabilité de ce nouvel algorithme le rendront utile à travers une large gamme d'applications.

[6] http ://nomis80.org/ctmf.html

[7] http ://www.intel.com/technology/computing/opencv/

Deuxième partie

Modélisation de l'apparence

Chapitre 2

Introduction

La modélisation de l'apparence d'un objet a pour but d'en produire une représentation informatique dont la visualisation d'un point de vue choisi par l'observateur est possible. L'objectif est de maximiser la fidélité de la reproduction au niveau de la génération d'images synthétiques à partir de nouveaux points de vue. Celles-ci doivent ressembler le plus possible à ce qui serait observé du même point de vue pour l'objet réel.

Les travaux de recherche menant à la rédaction de cette deuxième partie on porté sur la réalisation d'un système de modélisation de l'apparence d'objets réels de tailles diverses et pouvant posséder des propriétés de réflectance quelconques. Pour accomplir cette tâche, l'approche par stéréoscopie multi-vue (voir, par exemple, [9, 18, 38]), basée sur l'explicitation de la géométrie de l'objet, s'appuie sur un modèle de réflectance posé a priori, souvent lambertien. Ceci limite son applicabilité à des objets réels en général. En contraste, l'approche basée sur les champs de lumière (de l'anglais *light field*) repose sur l'échantillonnage direct des rayons de lumière émis (et réfléchis) par l'objet. Par la suite, les nouvelles vues sont générées par interpolation de ces échantillons. Cette méthode requiert un grand nombre de photos prises de différents points de vue à cause des réflectances diverses et de la complexité de la géométrie. Pour cette raison, on doit automatiser l'acquisition.

Pour les deux approches, les photos obtenues doivent être *calibrées*. Par ceci, on entend que la pose de la caméra, c'est-à-dire sa rotation et sa translation par rapport à un référentiel global fixé, ainsi que ses paramètres internes, tels que la longueur focale et la taille du CCD, doivent être connus. Ceci est nécessaire afin de pouvoir donner une signification géométrique 3-D à une observation 2-D provenant de l'image. Pour une photo calibrée, on peut savoir quel rayon de lumière se projette sur un pixel, transfor-

mant ainsi la caméra en un outil de mesure. D'après le signal enregistré à ce pixel, on peut connaître la couleur et l'intensité de ce rayon de lumière. C'est par l'interprétation de plusieurs photos calibrées que l'on acquiert une connaissance suffisante de la lumière émise et réfléchie par un objet pour pouvoir en créer un modèle convaincant. Il est à noter que les conditions d'éclairage doivent être constantes pour toutes les photos, et que les objets non opaques ne pourront évidemment pas être modélisés par l'approche avec support géométrique basé sur l'enveloppe visuelle.

Le système réalisé adopte une approche hybride. La géométrie précise de l'objet n'est pas recherchée, contrairement à ce qu'implique l'approche basée sur l'appariement stéréo. On recherche plutôt une géométrie approximative qui servira de support aux échantillons (pour plus de détails, se référer à [6]). Comme point de départ, on utilise l'enveloppe visuelle de l'objet [26]. Celle-ci est obtenue d'après le *shape from silhouette* [33], une technique bien connue produisant une géométrie 3-D à partir de la silhouette segmentée de l'objet observé dans plusieurs photos calibrées.

2.1 Le système automatisé

Un robot a été conçu par Samuel Bouchard, étudiant au doctorat en génie mécanique, pour automatiser le processus de prise de photos [10]. La caméra, une Canon EOS-300D / Digital Rebel, est fixée à un effecteur suspendu par 6 câbles. Ceux-ci sont manipulés par des moteurs indépendants. En changeant la longueur des câbles, on peut déplacer et orienter la caméra dans l'espace. La figure 2.1 montre le robot photographiant un objet. On y remarque l'environnement vert facilitant la segmentation automatique. On peut aussi y voir des cibles de positionnement imprimées sur des cartons verts. Elles serviront à calculer la pose de la caméra pour chaque photo.

La communication avec la caméra est nécessaire afin de déclencher la prise de photo. Ceci est accompli à l'aide d'un système sans-fil fait sur mesure basé sur des modules RF RS-232 900 MHz de MaxStream (XC09-009PKC-R). Il a été développé par Nathaniel Zoso, stagiaire de premier cycle. Le signal est émis de la station de base et reçu par un circuit fixé à l'effecteur. Celui-ci active une diode électroluminescente placée devant le senseur de déclenchement à distance de la caméra. Ce dernier ne pouvait pas être utilisé tel quel puisqu'il est directionnel et qu'il ne sera pas toujours en position de recevoir le signal au cours d'une acquisition. Les photos sont stockées sur la carte mémoire de la caméra et traitées par la suite. La figure 2.2 montre un exemple de photo prise par le système.

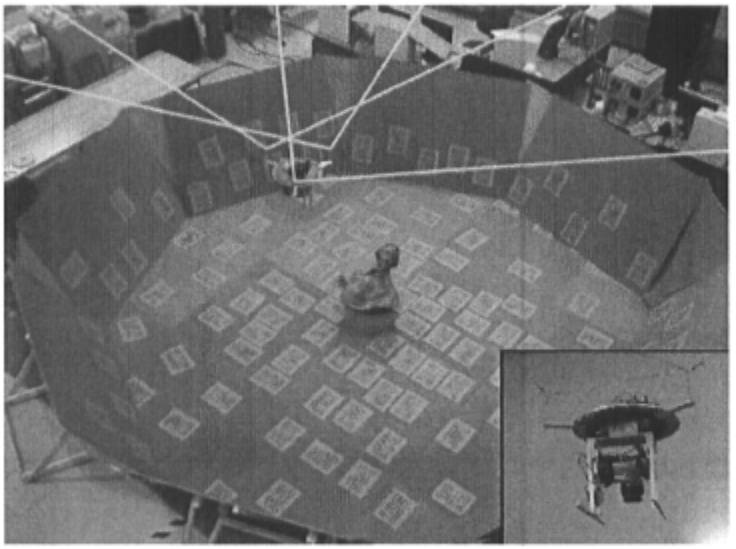


FIG. 2.1: Robot prenant une photo de l'objet à modéliser. Remarquez l'environnement vert et les cibles de positionnement. L'encadré montre l'effecteur, où la caméra est située.

Le mécanisme à câbles a l'avantage de pouvoir s'adapter à des objets de tailles diverses. L'objet est posé au sol et le robot effectue une trajectoire hémisphérique autour de lui, divisant l'espace en méridiens, et prend des photos à intervalle régulier. La taille et la position de l'hémisphère, ainsi que le nombre de photos prises, peuvent être changés facilement. Un autre avantage est que le robot génère peu d'ombre sur l'objet. En effet, une illumination constante est nécessaire au processus de modélisation.

On décrit dans [24] et [12] des systèmes similaires. L'enveloppe visuelle y est là aussi utilisée comme point de départ. La surface est raffinée par la suite suivant des techniques d'appariement stéréo. Dans [40], une méthode similaire est adaptée à la capture du mouvement de personnes. Des caméras vidéo sont disposées dans un studio et observent un acteur de différents points de vue. Là aussi, la surface initiale est l'enveloppe visuelle et est ajustée à l'aide de la stéréo multi-vues. Le système présenté ici n'utilise qu'une seule caméra et prend un plus grand nombre de photos (de l'ordre d'un millier) que ce qui est habituellement requis pour les méthodes photogrammétriques conventionnelles. Ceci est nécessaire afin d'explorer des techniques de raffinement de surface s'affranchissant des limites de la stéréo multi-vues en ne présupposant pas un modèle de réflectance spécifique. Quelques résultats de cette exploration sont présentés dans [25].

Le processus de modélisation étudié se divise en trois étapes principales. La première consiste en la segmentation des images afin d'y isoler l'objet et les cibles de positionne-

FIG. 2.2: Photo prise par la caméra montée sur le robot.

ment. Deuxièmement, on doit reconstruire un modèle des coordonnées 3-D des cibles, ce qui permet d'obtenir la position des caméras.[1] Troisièmement, on calcule l'enveloppe visuelle de l'objet par la fusion des silhouettes, ce qui fournit la géométrie approximative de départ.

2.2 Objectifs

L'objectif général du système est de calibrer un jeu d'images et d'en produire une géométrie approximative. Différentes techniques de raffinement de surface sont présentement à l'étude et utiliseront ces photos et cette géométrie approximative comme données d'entrée. Le système devra pouvoir accomoder des jeux de données allant jusqu'à 1000 images. Chacune des images occupe présentement 6.3 mégapixels, mais on doit pouvoir traiter des photos de plus grande taille, ce qui se produira dans la prochaine version du système alors qu'une caméra de meilleure résolution sera employée. Un jeu de données occupe actuellement 19 Go sans compression.

[1] Le robot lui-même ne connaît pas la position de l'effecteur avec une précision suffisante. Le positionnement obtenu par observation de cibles est plus précis.

Au point de vue informatique, le système doit être modulaire. Il est important de tenir compte du contexte dans lequel il est conçu : plusieurs étudiants à la maîtrise et au doctorat auront à l'utiliser, à le comprendre et à le modifier. Chacune des parties doit être simple et leur interaction doit être sans surprise. Ceci facilitera l'amélioration d'un module sans affecter les autres.

Le temps de traitement d'un jeu de données entier ne devrait pas dépasser 24 heures. Cette durée a été choisie afin que l'on puisse tester le système au complet au moins une fois par jour. Il est important de pouvoir effectuer un cycle complet rapidement afin d'accélérer le développement.

Le traitement d'un jeu de données ne devrait pas impliquer l'intervention d'un être humain. Il est sous-entendu qu'un traitement manuel pourra avoir lieu au début ou à la fin du processus, mais celui-ci ne devra pas être fastidieux. En particulier, le processus ne devrait pas être interrompu pour demander une interaction humaine une fois qu'il est lancé. Par contre, le système doit stocker les résultats intermédiaires afin de pouvoir les visualiser pour les valider ultérieurement.

Finalement, le système ne doit pas dépendre d'un système d'exploitation en particulier. Ce critère a pour but de favoriser son adaptabilité à des situations nouvelles ainsi qu'à le rendre plus abordable à de nouveaux étudiants. Il est attendu que les étudiants n'auront pas tous une connaissance approfondie de la programmation et qu'un système conçu à un haut niveau d'abstraction est préférable. En particulier, on s'attend à ce que le système fonctionne sous Linux et Windows.

Les travaux menant à la rédaction de ce mémoire on porté sur l'intégration de plusieurs algorithmes ainsi qu'à leur évaluation par rapport aux critères énoncés plus haut. Des améliorations leur ont été apportées afin d'en améliorer les performances.

Ce mémoire couvre le traitement informatique que subiront les photos afin d'en produire un modèle d'apparence. Le chapitre 3 décrit le système en détail ainsi que chacune de ses parties. Pour chacune, on retrouve une description de son fonctionnement, un commentaire sur son efficacité et sa complexité et, s'il y a lieu, une description des changements qui lui ont été apportés. Le chapitre 4 montre quelques résultats et discute de l'atteinte des objectifs fixés.

Chapitre 3

Analyse du système

Ce chapitre a pour but de décrire le système dans son ensemble et chacune de ses parties indépendamment, ainsi que d'apporter un jugement au niveau de la performance. Pour chaque partie, une critique sera formulée quant à la qualité des résultats produits et au temps de calcul requis. Le tests ont été menés sur un processeur AMD Athlon X2 4200+.

3.1 Choix technologiques

Quelques choix technologiques ont dû être faits. Les critères guidant ceux-ci sont décrits à la section 2.2.

3.1.1 Langage de programmation

Le langage de programmation choisi est Matlab[1]. Celui-ci a l'avantage d'être très répandu en ingénierie et d'être de très haut niveau. Il répond aux besoins de simplicité et de portabilité. Il est disponible sur plusieurs plateformes, dont Linux et Windows. Avec la *Image Processing Toolbox*, il fournit une gamme suffisante de fonctionnalités mathématiques et de traitement d'image de base.

Matlab a par contre la réputation d'être lent. Ceci a été confirmé au cours du

[1] http ://www.mathworks.com/

développement de plusieurs composantes. Lorsque nécessaire, on peut convertir une fonction Matlab en langage C portable. Ceci peut se faire de deux façons. La première consiste à écrire un *MEX file*, un fichier C ou C++ simple compilé dans une librairie dynamique appelée par Matlab. Le passage de paramètres s'y fait de façon simple en utilisant une interface fournie par Matlab. C'est une méthode très conviviale qui reste portable tant qu'on se limite à du C/C++ standard.

En revanche, elle n'offre pas toute la liberté parfois requise. Par exemple, l'allocation de mémoire doit toujours se faire en utilisant la fonction spéciale "mxMalloc". Ceci limite l'utilisation de librairies telles que OpenCV[2] lorsque les fonctionnalités utilisées font appel à une autre fonction d'allocation de mémoire. En particulier, tout appel à la fonction "malloc" et toute utilisation de l'opérateur "new" en C++ est interdite. Faire autrement cause de la corruption de mémoire qui peut faire terminer le programme prématurément. Lorsque c'est nécessaire, il faut créer un programme autonome qui sera appelé par Matlab. Les paramètres lui seront passés par des fichiers. Ceci offre toute la liberté voulue.

3.1.2 Gestion de projet

L'intégration du code source a été facilitée grâce au logiciel Subversion[3]. Il permet de suivre l'évolution du code et favorise la collaboration de plusieurs personnes qui peuvent travailler en parallèle sur le même code source et intégrer leurs changements au fur et à mesure. Il est disponible sous Linux et Windows.

Finalement, le logiciel de gestion de projet Trac[4] a été mis à contribution. Il intègre un *wiki* qui facilite l'écriture de documentation, un visualisateur de dépôt Subversion, et une base de données pour le suivi de tâches. Il est accessible à l'adresse http ://culbute.gel.ulaval.ca/trac/robot/.

3.2 Flot des données

Le flot des données dans le système analysé est illustré schématiquement à la figure 3.1. Les parallélogrammes représentent des données alors que les rectangles représentent

[2]http ://www.intel.com/technology/computing/opencv/
[3]http ://subversion.tigris.org/
[4]http ://trac.edgewall.org/

les traitements qu'elles subissent.

On peut séparer le système en deux parties : le positionnement des caméras et la création du modèle. Cette séparation est justifiée par le fait que les traitements de l'une sont en bonne partie indépendants de l'autre et vice-versa. Les données en entrée sont les photos prises par les robots et les paramètres intrinsèques de la caméra. On appellera l'ensemble de ces données un *jeu de données.*

La première tâche est la segmentation des cibles et de l'objet. On remarque à la figure 3.1 que la rectification, qui consiste à éliminer la distorsion et centrer le point principal dans l'image, se fait avant la segmentation de l'objet. En effet, la construction de l'enveloppe visuelle nécessite des masques rectifiés et on risque de perdre moins de précision lors de l'interpolation en rectifiant les photos originales plutôt que les masques binaires.

Pour le positionnement des caméras, les cibles segmentées sont ensuite détectées et leurs positions dans l'image estimées. Les points, mis en correspondance, sont utilisés pour la reconstruction d'un modèle 3-D des cibles. Pour des raisons techniques décrites en détail à la section 3.3.4, on doit construire ce modèle à partir d'un sous-ensemble des photos et estimer la pose des autres photos indépendamment d'après le modèle produit.

Les poses sont ensuite utilisées pour la construction de l'enveloppe visuelle à partir des masques de segmentation de l'objet. On raffine la surface obtenue afin d'en améliorer le potentiel de support géométrique pour l'apparence. Finalement, on synthétise des textures compressées variant selont l'angle de vision à l'aide du logiciel *Light Field Mapper* [8].

3.2.1 Format des données

Suivant sa fonction de support à la recherche, le système doit pouvoir faire constamment l'objet de développement. Il est à prévoir que des traitements soient remplacés par d'autres remplissant la même fonction de manière différente, ou même que plusieurs traitements soient enlevés ou fusionnés. Afin de faciliter cette effervescence, il est désirable d'établir des formats de stockage des données qui soient simples à lire et à écrire. La simplicité étant le critère principal, les formats établis pour le système dans son état présent sont les suivants :

- *Photos* : Ceci dépend de l'appareil photo. Présentement, l'appareil dont nous disposons est un Canon EOS-300D / Digital Rebel produisant des photos couleur

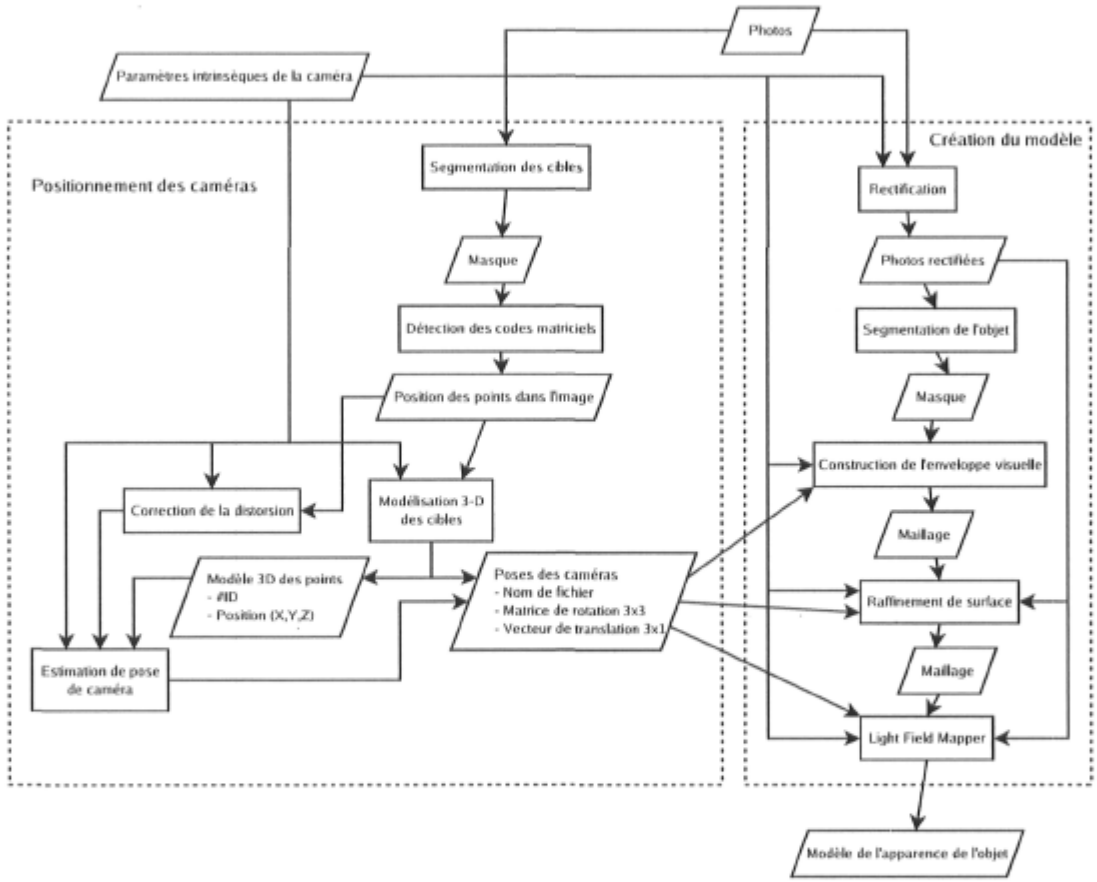


FIG. 3.1: Flot des données dans le système analysé.

de 6.3 mégapixels. Idéalement, on voudrait un format sans perte connu dans le domaine de la photographie comme format RAW. Cependant, il serait impossible pour le moment de faire tenir la totalité d'un jeu de données de 1000 photos, équivalant à environ $1000 \times 3 \times 6.3 \times 10^6 = 18.9$ Go, sur une seule carte mémoire. On opte donc pour le format JPEG.

- *Paramètres intrinsèques de la caméra* : Un fichier texte contenant les paramètres (voir section 3.3.1), convertis en ASCII avec la précision voulue, séparés par un ou des blancs. (Un blanc est un caractère d'espace, de tabulation ou de saut de ligne.) L'ordre des paramètres est celui du tableau 3.1.
- *Masque* : Fichier PNG binaire de la même taille que la photo d'entrée. Le blanc correspond à l'avant-plan et le noir à l'arrière-plan.
- *Position des points dans l'image* : Un fichier texte par image. Chaque ligne correspond à un point. Les valeurs sur chaque ligne sont la coordonnée horizontale suivie de la coordonnée verticale, toutes deux en pixels. Elles ne sont pas limitées à un nombre entier (résolution subpixel). Bien que peu intuitif pour l'indexage de pixels en mémoire, cet ordre a l'avantage de faciliter la lecture et l'affichage avec le logiciel Matlab.
- *Modèle 3-D des points* : Un fichier texte unique. Chaque ligne correspond à un point. Les valeurs sur chaque ligne sont le numéro d'identification suivi des coordonnées (x, y, z). Le repère global est déterminé au moment de la modélisation 3-D (voir section 3.3.4).
- *Poses des caméras* : Un fichier texte par pose. On y écrit simplement la matrice P intégralement (voir section 3.3.1).
- *Maillage* : Format OBJ rendu populaire par les logiciels de la compagnie Wavefront. Bien qu'aucune spécification formelle ne soit publiquement disponible, de l'information *ad hoc* le concernant est disponible sur Internet. La quasi totalité des logiciels manipulant des maillages disponibles sur le marché peuvent lire et écrire des fichiers de format OBJ. La description du format utilisé est disponible à l'annexe A.
- *Modèle d'apparence* : Ce format est pour le moment défini par le logiciel Light Field Mapper [8]. C'est un format compressé, destiné à l'affichage. Lors d'un éventuel remplacement de LFM on aurait avantage à définir un format de données répondant au besoin de simplicité.

3.2.2 Mode de communication

Afin de répondre au besoin de simplicité, les processus communiquent par lecture et écriture de fichier. Les données intermédiaires sont conservées pour visualisation et validation ultérieure ainsi que pour pouvoir sauter des étapes lors de la reprise du

traitement d'un jeu de données. À cette fin, l'ensemble du système est contrôlé par une fonction maîtresse à laquelle on fournit comme seul argument le nom d'un répertoire contenant les données. La fonction n'exécute chaque processus que si les données en sortie sont inexistantes. Ceci permet au programmeur de tester un nouvel algorithme simplement en supprimant un élément de données produit par cet algorithme et en exécutant la fonction maîtresse. Ce stratagème favorise le développement organique d'un système cohésif.

La fonction maîtresse établit un classement des données par répertoires. Toujours dans le but de simplifier le développement, les processus qui produisent un fichier de données par photo d'entrée placent l'ensemble des fichiers produits dans un répertoire séparé. Le nom de base de chaque fichier (c'est-à-dire le nom sans le chemin ni l'extension) est identique à celui de la photo correspondante. Par exemple, la pose de la photo `dataset/IMG_1234.JPG` se trouve dans le fichier `poses/IMG_1234.txt`. Pour démarrer le traitement, on appelle la fonction maîtresse sur le répertoire parent. Par exemple : `main('/datasets/chevreuil')`. Les détails des noms de fichiers et de répertoires produits se trouvent dans la documentation de la fonction maîtresse.

3.3 Étapes de traitement

Cette section décrit les traitements effectués à chaque étape de la production d'un modèle de l'apparence d'un objet à partir des photos d'entrée.

3.3.1 Calibrage de la caméra

Une caméra peut être modélisée comme un système agissant sur des points 3-D et produisant une image 2-D. La relation entre l'entrée et la sortie est exprimée par le *modèle de projection* de la caméra. Le calibrage d'une caméra consiste à mesurer les paramètres de ce modèle. Lorsque ceux-ci sont connus, on peut calculer l'image $\mathbf{x}$ d'un point 3-D $\mathbf{X}$ connu. Le modèle de projection linéaire utilisé [19, 39] est le suivant :

$$\underbrace{\begin{bmatrix} \lambda x \\ \lambda y \\ \lambda \end{bmatrix}}_{\mathtt{x}} = \underbrace{\underbrace{\begin{bmatrix} \frac{N}{w} f & 0 & \frac{N}{w} x_0 & 0 \\ 0 & \frac{M}{h} f & \frac{M}{h} y_0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix}}_{\mathtt{K}} \underbrace{\begin{bmatrix} \mathrm{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix}}_{\mathtt{P}}}_{\mathtt{M}} \underbrace{\begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix}}_{\mathbf{X}} \tag{3.1}$$

La matrice de projection `M` est le produit de la matrice des paramètres intrinsèques `K` et de la pose `P`. Les descriptions individuelles des paramètres figurent au tableau 3.1.

Ceci est augmenté d'un modèle non linéaire de la distorsion de la lentille exprimant les coordonnées corrigées d'un point image $\mathbf{x}_c$ en fonction de celles observées $\mathbf{x}_d$:

$$\mathbf{x}_c = \mathbf{x}_d + \mathcal{F}_D(\mathbf{x}_d, \boldsymbol{\delta}) \tag{3.2}$$

où

$$\mathcal{F}_D(\mathbf{x}_d, \boldsymbol{\delta}) = \begin{bmatrix} \bar{x}_d(k_1 r_d^2 + k_2 r_d^4 + k_3 r_d^6 + \ldots) \\ \quad + (2p_1 \bar{x}_d \bar{y}_d + p_2(r_d^2 + 2\bar{x}_d^2))\,(1 + p_3 r_d^2 + \ldots) \\ \bar{y}_d(k_1 r_d^2 + k_2 r_d^4 + k_3 r_d^6 + \ldots) \\ \quad + (p_1(r_d^2 + 2\bar{y}_d^2) + 2p_2 \bar{x}_d \bar{y}_d)\,(1 + p_3 r_d^2 + \ldots) \end{bmatrix},$$

$$\bar{x}_d = x_d - x_0, \quad \bar{y}_d = y_d - y_0, \quad r_d = \sqrt{\bar{x}_d^2 + \bar{y}_d^2},$$

$$\text{et } \boldsymbol{\delta} = [k_1, k_2, \ldots, p_1, p_2, \ldots]^{\mathrm{T}}. \tag{3.3}$$

$(k_1, k_2, \ldots)$ sont les paramètres de distorsion radiale alors que $(p_1, p_2, \ldots)$ sont les paramètres de distorsion tangentielle. Dans le système réalisé, seuls k_1, k_2, p_1 et p_2 sont utilisés.

Il est montré dans [19] comment inverser le modèle de distorsion et obtenir $\mathbf{x}_d$ d'après $\mathbf{x}_c$. Alors, étant données les coordonnées dans le référentiel global[5] du point $\mathbf{X}$, la pose `P` de la caméra et les paramètres $(s, f, x_0, y_0, \boldsymbol{\delta})$, on peut calculer les coordonnées du point image `x`. Cette fonction de projection non linéaire combinée est identifiée $\mathcal{P}$.

Pour calibrer la caméra, on procède par observation d'un objet possédant des points de repère aux coordonnées 3-D connues. Pour le système analysé, on utilise une cible sur laquelle figurent des points circulaires noirs sur fond blanc. Quelques photos de la cible (environ une dizaine) sont prises sous plusieurs angles différents. Les réglages de la caméra doivent être les mêmes que lors de la prise de photos de l'objet par le robot.

[5] Dans le cadre du système analysé, le référentiel global est déterminé lors de la reconstruction du modèle 3-D des cibles de positionnement et de la pose des caméras.

TAB. 3.1: Paramètres intrinsèques de la caméra.

Paramètre	Symbole	Unités
Longueur focale	f	mm
Taille du CCD	w, h	mm
Taille de l'image	M, N	pixels
Coordonnées du point principal	x_0, y_0	mm
Paramètres de distorsion radiale	k_1, k_2, k_3	mm^{-2}, mm^{-4}, mm^{-6}
Paramètres de distorsion tangentielle	p_1, p_2	mm^{-2}

On peut cependant choisir à notre convenance de prendre les photos de calibrage avant ou après la prise de photos de l'objet.

Le processus informatique de calibrage peut être divisé en trois parties :

1. La détection. Il s'agit du traitement de bas niveau de l'image, le plus lourd en calculs. Il doit identifier les régions de l'image correspondant à la projection des points circulaires. Puisqu'il n'est pas requis que la cible occupe toute l'image, l'arrière-plan est supposé quelconque et peut contenir des objets ressemblant aux points recherchés. L'algorithme de détection se doit d'être assez discriminant mais doit favoriser l'exhaustivité au prix de l'augmentation du nombre de faux positifs. En effet, ces derniers peuvent être éliminés lors de l'étape suivante. On détermine les paramètres des ellipses observées à un niveau subpixel avec [35].
2. La mise en correspondance se doit d'associer à chaque point observé le point correspondant de la cible. La forme et la disposition des points de la cible y sont mises à contribution.
3. Le calcul des paramètres s'effectue suivant [19]. En particulier, on cherche à minimiser la distance entre les coordonnées image observées et prédites pour n points par rapport aux paramètres du modèle de projection.

$$\min_{s,f,x_0,y_0,\boldsymbol{\delta},\mathrm{R},\mathbf{t}} \sum_{i=1}^{n} \|\mathbf{x}_i - \mathcal{P}(\mathbf{X}_i, s, f, x_0, y_0, \boldsymbol{\delta}, \mathrm{R}, \mathbf{t})\|^2 \tag{3.4}$$

Ceci est résolu par un algorithme de moindres carrés non linéaires tel le Levenberg-Marquardt [27,32]. L'initialisation se fait en annulant la distorsion et en résolvant directement le modèle linéaire résultant.

On distingue les paramètres intrinsèques, qui ne relèvent que de la caméra, et extrinsèques, qui relèvent de la pose de la caméra dans le référentiel global. Les paramètres intrinsèques utilisés tout au long du traitement sont listés au tableau 3.1.

Critique

La détection et l'appariement sont très robustes. En plaçant la cible devant un arrière-plan peu texturé et en n'introduisant pas un arrière-plan lointain dans une image prise selon un angle trop prononcé, on peut être confiant que le traitement se fera sans intervention manuelle.

La détection est codée en C++ à l'aide de la librairie OpenCV et est rapide, avec un temps de calcul d'environ 8 secondes par image. On prend habituellement 6 images de la cible. Les autres étapes (appariement et minimisation de l'erreur de reprojection) prennent un temps de l'ordre de la seconde. Le temps de calcul total pour calibrer une caméra est d'environ une minute.

L'erreur RMS totale et l'erreur maximale sont affichées et on peut donc savoir rapidement si le calibrage est correct ou non. En pratique, l'erreur RMS est d'environ 0.25 pixels tandis que l'erreur maximale ne dépasse pas le pixel. Le code de détection, d'appariement et d'optimisation a été produit par Jean-Nicolas Ouellet, étudiant au doctorat.

3.3.2 Segmentation

Deux segmentations doivent être effectuées : celle de l'objet et celle des cibles. Les deux sont du même type : identifier une région, *l'avant-plan*, et rejeter le reste de l'image, *l'arrière-plan*. On utilise la même méthode simple dans les deux cas. Elle fonctionne par apprentissage manuel de la distribution F des valeurs des pixels de l'avant-plan. Pour un petit échantillon représentatif de photos (5 ou 6 points de vue variés suffisent), on identifie manuellement les régions d'avant-plan. On accumule les valeurs des pixels identifiés dans un histogramme 3-D correspondant au cube de couleur RGB. Ce cube est référé par la suite comme étant la distribution de l'avant-plan. Celle-ci a l'avantage d'être non paramétrique, ce qui lui confère une grande flexibilité. On peut conserver cette distribution et la réutiliser tant que les conditions d'éclairage ne changent pas.

La distribution B de l'arrière-plan n'est pas apprise. Pour signifier l'absence d'information à son égard, on la représente par une distribution de couleurs uniforme.

Dans le cas de la segmentation des cibles, on considère ces dernières comme l'avant-plan alors que le reste de l'image est l'arrière-plan. Ceci est justifié par le fait que les cibles ont été conçues de façon à ce que leur distribution occupe un espace restreint dans

le cube RGB (deux tons de vert rapprochés). Pour la segmentation de l'objet, on ne dispose pas de cette hypothèse simplificatrice. En effet, l'objet peut être quelconque. En revanche, l'environnement de l'objet, lui, a été choisi de façon à avoir une distribution restreinte. Les cibles en font déjà partie et comportent deux tons de vert. Il a donc été décidé de construire le reste de l'environnement à partir de panneaux de bois peinturés d'un troisième ton de vert. La distribution est alors restreinte à trois tons de vert. Elle est considérée être l'avant-plan alors que le reste est l'arrière-plan. Il est à noter que, d'un point de vue très incliné, les panneaux verts ne couvrent pas tout le champ de vue et il est possible d'apercevoir un arrière-plan quelconque qui n'est pas de l'objet mais qui sera identifié comme tel. Cette situation est acceptable parce que les parties superflues seront éliminées automatiquement lors du calcul de l'enveloppe visuelle décrit à la section 3.3.6.

Étant données ces deux distributions F et B, on détermine à laquelle sera associé un pixel observé par l'algorithme des K plus proches voisins. Celui-ci est conceptuellement simple : il s'agit de trouver les K pixels accumulés dans les histogrammes RGB qui sont les plus près dans l'espace RGB du pixel observé. Parmi ces K pixels, la proportion P de ceux faisant partie de la distribution de l'avant-plan sert de mesure pour associer ou non le pixel observé à cette distribution. On peut fixer un seuil sur P à partir duquel on classe le pixel comme de l'avant-plan. Dans l'implémentation réalisée, le seuil déterminé empiriquement est de 80 % pour la segmentation des cibles et de 8 % pour la segmentation de l'objet. Cette disparité importante est attribuable au fait que le calcul de l'enveloppe visuelle à partir des masques de segmentation de l'objet effectue une coupe sévère de l'espace (voir section 3.3.6), et donc la réduction de faux négatifs est plus importante.

Nicolas Martel-Brisson, étudiant au doctorat, a fourni l'idée ainsi qu'une implémentation préliminaire en Matlab. Cette dernière n'était qu'une preuve de concept et utilisait un algorithme naïf : en parcourant le cube RGB de la distribution d'avant-plan, on somme les valeurs dont la distance au pixel observé est inférieure ou égale à un seuil d. On commence en posant $d = 0$ et on l'incrémente jusqu'à ce que K pixels contribuent à la somme. Pour accélérer les calculs, on peut faire varier d selon une recherche dichotomique puisque l'espace est borné. J'ai développé un algorithme bien plus efficace.

Implémentation rapide

Dans une implémentation naïve, la majorité du temps sera passé dans des calculs de distance. Non seulement ceux-ci sont-ils fastidieux, on en répète la majeure partie lorsque l'on incrémente d et que l'on recommence le processus. La solution est de n'ef-

fectuer les calculs de distance qu'une seule fois par triplet (r, g, b) et d'enregistrer les résultats pour utilisation future. Ce faisant, on troque de l'espace mémoire pour du temps de calcul.

On crée un index inverse I associant une norme d à l'ensemble des triplets (r, g, b) pour lesquels $\sqrt{r^2 + g^2 + b^2} = d$. On dénote cet ensemble $I(d)$. L'index est précalculé pour toutes les coordonnées entières variant de $(-255, -255, -255)$ à $(255, 255, 255)$ dans le cas d'images 8 bits. L'index est implémenté de façon à ordonner les triplets (r, g, b) en ordre croissant de norme. En C++, la classe standard *multimap* répond à l'appel.

L'index est utilisé de la manière suivante : pour un pixel p à segmenter, on le parcourt à partir de la plus petite norme (qui sera associée à $(0, 0, 0)$). Pour chaque triplet qu'il contient, on l'additionne aux coordonnées RGB du pixel et on accumule les voisins de la distribution d'avant-plan à cet endroit. Le nombre de voisins de la distribution d'arrière-plan est obtenu analytiquement selon l'expression du volume d'une sphère puisque cette distribution est uniforme. On arrête lorsque le nombre de voisins total est égal ou supérieur à K. Cette méthode figure à l'algorithme 3.1.

On remarque que la redondance dans les calculs est disparue. Il est à noter que l'on peut éviter de recalculer l'index à chaque invocation de l'algorithme 3.1 puisqu'il ne dépend d'aucune valeur d'entrée. Comme autre optimisation, on peut enregistrer les résultats dans une cache lorsque les paramètres K, F et B sont les mêmes. On vérifie d'abord si le résultat P pour le pixel p est dans la cache avant de le calculer.

L'index peut être implémenté en C++ à l'aide de la classe standard *multimap*. On peut le parcourir en ordre de distance à l'aide des itérateurs fournis, ce qui est très efficient. Dans le cas d'images 8 bits, il devra contenir $(2 \times 2^8)^3 = 2^{27}$ triplets. Stockant chacun sur 8 octets (3×16 bits plus le remplissage), il nécessitera 1 Go de mémoire. On peut réduire sa taille en exploitant la redondance. En effet, pour chacune des 6 permutations des 3 coordonnées r, g et b, la distance est la même. De plus, que les coordonnées soient positives ou négatives, la distance est la même. À partir de r, g et b positifs, on peut former 8 triplets différents en assignant des signes négatifs. En ne stockant que les triplets dont les coordonnées sont positives et ordonnées, on réduit la taille de l'index d'un facteur $6 \times 8 = 48$, ce qui l'abaisse à 21.3 Mo. Pour chaque triplet de I visité, on n'a qu'à générer les 48 variétés à la volée. Les triplets contenant des coordonnées répétées ou des zéros doivent être traités spécialement.

Algorithme 3.1 K plus proches voisins rapide.

Entrée:

- Nombre K de voisins à considérer ;
- Distribution d'avant-plan F ;
- Densité uniforme d'arrière-plan B ;
- Pixel $p = (r, g, b)$.

Sortie: Proportion P d'avant-plan pour les K plus proches voisins du pixel p.

1: $I \leftarrow$ index inverse de distances
2: $K_F \leftarrow 0$
3: **pour** toute distance d figurant dans I, en ordre croissant, **faire**
4: $\quad K_B \leftarrow \frac{4}{3}\pi d^3 \times B$
5: $\quad$ **pour** chaque triplet $p' = (r', g', b') \in I(d)$ **faire**
6: $\quad\quad p'' = p + p'$
7: $\quad\quad$ **si** p'' fait partie du domaine de F **alors**
8: $\quad\quad\quad K_F \leftarrow K_F + F(p'')$
9: $\quad\quad$ **fin si**
10: $\quad$ **fin pour**
11: $\quad$ **si** $K_F + K_B \geq K$ **alors**
12: $\quad\quad$ Aller à la ligne 15.
13: $\quad$ **fin si**
14: **fin pour**
15: $P \leftarrow \frac{K_F}{K_B}$

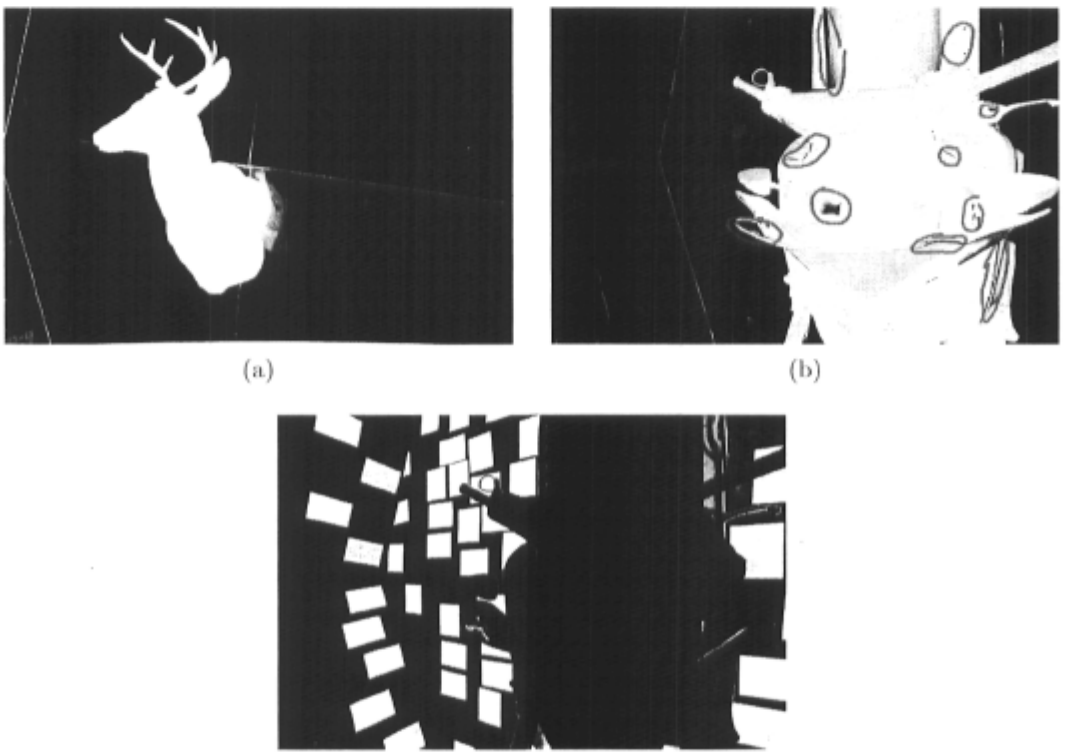
Critique

La segmentation est le maillon faible de la chaîne. Les résultats qu'elle produit affectent directement la qualité du modèle. Un exemple des problèmes importants que l'on rencontre est la quantité exagérée de faux positifs, c'est-à-dire des pixels classés comme de l'avant-plan alors qu'ils font en réalité partie de l'arrière-plan. Ils se présentent comme des trous dans la silhouette de l'objet. Ceux-ci se créent pour deux raisons : la spécularité de la surface et l'étalement de la couleur de l'environnement (*bleeding*). Dans le premier cas, la surface agit comme un miroir et reflète des parties vertes de l'environnement. Dans le deuxième cas, l'environnement crée une lumière ambiante verte qui est reflétée de manière diffuse par la surface, s'additionnant à sa couleur intrinsèque. À partir d'un certain seuil, la surface a une couleur faisant partie de la distribution apprise et est classée erronément.

Il est à noter que ce même algorithme de segmentation fonctionne suffisamment bien pour les cibles. La raison est que chaque composante connexe du masque des cibles est validé par la détection des codes matriciels. Si une cible n'est pas reconnue, la composante est rejetée. La segmentation ne sert qu'à identifier des régions potentielles de l'image où des cibles pourraient se trouver. De plus, si des pixels appartenant aux cibles sont associés à la mauvaise classe, la détection fonctionnera tant qu'il reste assez de pixels bien classés pour préserver la forme globale de la composante connexe.

C'est une autre histoire pour la segmentation de l'objet. Si ne serait-ce qu'un seul pixel appartenant à l'objet était mal classé, il se formerait un trou dans l'objet lors du calcul de l'enveloppe visuelle (voir section 3.3.6). En effet, l'algorithme utilisé effectue une coupe *sévère* du volume. Ce problème est très visible pour des objets spéculaires (voir figure 3.2 (b)).

La création de la distribution d'avant-plan prend environ 2 secondes par image segmentée manuellement. On utilise habituellement environ 5 images, ce qui donne un temps de l'ordre de la dizaine de secondes. La segmentation proprement dite prend un temps qui varie selon le paramètre K. Plus il est grand, plus la recherche doit se faire sur un grand nombre d'échantillons et plus le temps de calcul requis est grand. On pose habituellement $K = 400$, ce qui résulte en un temps de calcul d'environ 75 secondes par image. À titre d'exemple, $K = 5000$ résulte en un temps de 225 secondes. Pour un jeu de données de 1000 photos, les deux segmentations (celle de la cible et celle de l'objet) nécessiteraient $2 \times 75 \times 1000/3600 = 41.7$ heures au total, ce qui excède le budget maximum de 24 heures. Il est cependant très facile de paralléliser le traitement en assignant un sous-ensemble des images à chaque processeur. En exploitant les deux coeurs du Athlon X2, on atteint 20.8 heures. Cependant, ceci n'est pas automatisé



(a) (b)

(c)

FIG. 3.2: Résultats typiques de segmentation. (a) Segmentation adéquate d'un objet mat (un chevreuil empaillé). (b) Segmentation inadéquate d'un objet spéculaire (une motoneige). Les régions identifiées formeront des trous lors de la reconstruction de l'enveloppe visuelle. (c) Segmentation adéquate des cibles de positionnement.

puisque le langage de programmation Matlab ne permet pas de créer différents fils d'exécution (*threads*).

Afin d'accélérer l'implémentation, le parallélisme semble être la voie la plus simple. D'autres étapes du traitement pourraient aussi être parallélisées facilement. Les développements requis pour paralléliser la segmentation pourraient être adaptés directement aux autres traitements.

Des développements futurs devraient porter en premier sur l'amélioration de la qualité de la segmentation de l'objet. Les cibles sont déjà bien segmentées, ce qui permet d'obtenir la pose des photos. Cette information pourrait peut-être être mise à contribution au moment de la segmentation. Une approche de ce genre est décrite dans [14]. Les résultats qui y sont montrés sont encourageants.

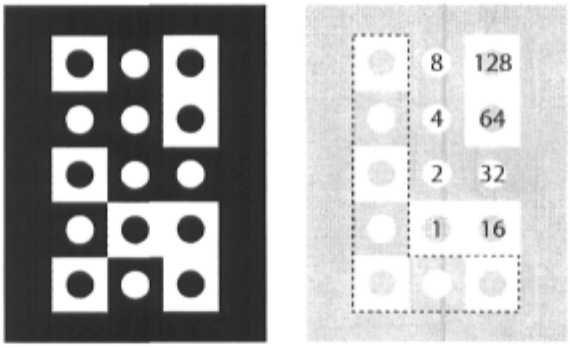
3.3.3 Codes matriciels

Afin de déterminer la pose de la caméra pour chaque photo, il est nécessaire d'utiliser des points de repère. Ceux-ci peuvent être naturels aussi bien qu'artificiels. Cependant, la deuxième alternative est à privilégier lorsque l'environnement le permet puisque la détection de cibles artificielles est habituellement beaucoup plus facile et fournit des poses plus précises.

Une cible artificielle doit remplir trois critères :

1. Détection robuste. Ceci est accompli par l'utilisation de formes et/ou de couleurs distinctes de l'environnement.
2. Estimation précise. La qualité du modèle 3-D reconstruit dépend directement de la précision de l'estimation dans l'image.
3. Identification possible. Il est nécessaire de pouvoir différencier une cible d'une autre et de pouvoir reconnaître les points de repère fournis par une cible.

Un modèle de cible répondant à ces critères a été conçu (voir la figure 3.3). Un cercle noir indique une valeur de zéro et un blanc une valeur de un. À chaque position est associée une puissance de deux, de façon à ce que chaque cercle ait un poids de un bit. Les couleurs alternantes des cercles dans le L pointillé sont constantes et on se sert de ce motif pour déterminer l'orientation de la cible. Ces cibles sont imprimés sur des feuilles ou des cartons 8.5×11" réguliers et placés dans la scène, autour de l'objet à être modélisé. Une imprimante à jet d'encre est utilisée afin d'éviter la spécularité de l'impression laser qui pourrait poser problème lors de la segmentation.



(a) (b)

FIG. 3.3: (a) Code matriciel représentant la valeur 46. (b) Motif utilisé pour déterminer l'orientation (L pointillé) et le poids de chaque bit.

La détection est rendue facile par le fait qu'un quadrilatère contenant une matrice 3-par-5 de cercles est recherché. En effet, une droite demeure une droite sous la projection, mais le parallélisme n'est pas préservé en général. Un rectangle se transforme donc en un quadrilatère convexe.

L'algorithme RANSAC classique [13] a été adapté à la tâche de recherche de quadrilatère de la manière suivante. Premièrement, une liste d'edgels[6] est obtenue par traitement d'image, habituellement avec un filtre de Sobel. De cette liste, RANSAC en choisit itérativement 2 au hasard et construit une droite. Les edgels se retrouvant à l'intérieur d'une distance prédéterminée de celle-ci sont utilisés pour calculer un nouvel ajustement de ligne. De cette façon, un ensemble de lignes dans l'image est construit. Par la suite, 4 droites sont choisies itérativement au hasard et un quadrilatère est déterminé. Finalement, le centre de chaque ellipse est détecté avec une précision subpixel au moyen de l'algorithme présenté dans [35], ce qui permet de vérifier la qualité de l'ajustement en calculant une homographie et en comparant les positions attendues des ellipses avec celles observées.

Une fois que la cible est détectée, l'identification a lieu. Les couleurs en alternance de la colonne de gauche et de la rangée du bas permettent de déterminer l'orientation. Les 8 cercles restant encodent un numéro d'identification de cible unique de par leur couleur. Puisque la cible ne doit pas être symétrique (sinon l'orientation ne pourrait pas être déterminée), les numéros de 88 à 95 ne peuvent pas être représentés. En tout, ce modèle fournit 248 possibilités. Ceci peut être étendu facilement en augmentant le nombre de cercles sur la cible.

Cinq points de référence correspondant aux centres des cercles des coins et du centre

[6]Un "edgel", par analogie avec "pixel", est un élément d'arête (*edge* en anglais).

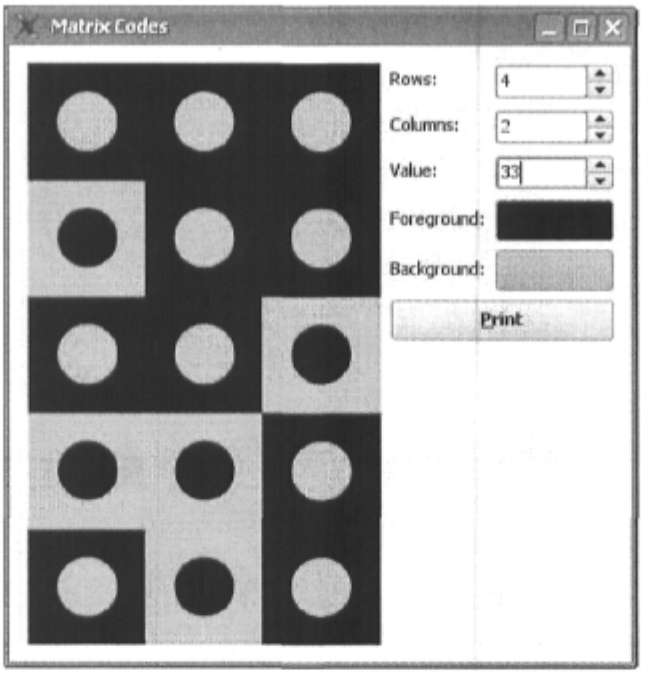


FIG. 3.4: Logiciel d'impression de codes matriciels.

sont extraits par cible. Ceux-ci servent de données d'entrée pour la construction du modèle 3-D à la section 3.3.4.

Les cibles sont confectionnées à l'aide d'un logiciel C++ avec interface visuelle qui permet de choisir la taille du papier, le nombre de colonnes et de rangées, ainsi que les couleurs d'impression de la cible (voir figure 3.4). Il permet aussi de générer un document PDF pouvant être utilisé pour impression ultérieure. On peut produire une seule cible ayant un numéro d'identification précis aussi bien qu'un grand nombre de cibles ayant des numéros consécutifs. On dispose ensuite les cibles le plus uniformément possible dans l'environnement de l'objet à modéliser, de façon à voir au moins 4 ou 5 cibles complètes par photo.

L'idée de base et le code de détection ont été proposés par Jean-Daniel Deschênes et Philippe Lambert, étudiants à la maîtrise et au doctorat. Le logiciel d'impression est de ma contribution.

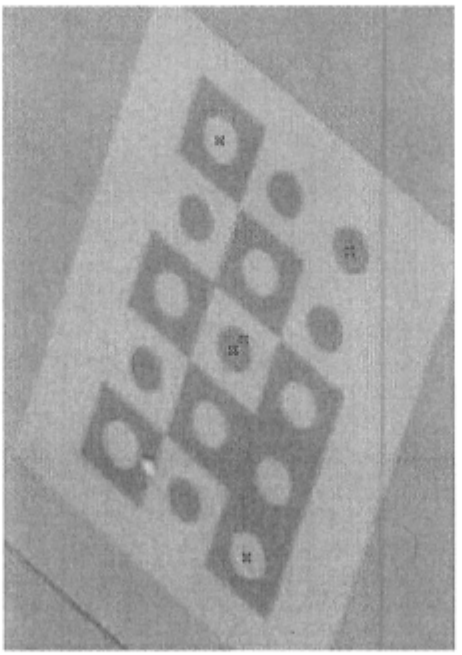


FIG. 3.5: Exemple de mauvaise détection des codes matriciels.

Critique

La détection des codes matriciels devrait être encore améliorée puisque les erreurs sont trop fréquentes. La figure 3.5 montre une telle erreur de détection. On remarque que le coin en bas à gauche n'est pas marqué comme le sont les autres, alors que la marque qui devrait s'y trouver se situe près du centre. Il est étonnant que ce genre de problème se produise puisqu'une validation avec une homographie est utilisée. Dans la plupart des cas, comme dans celui de la figure 3.5, une tache ou un objet inattendu crée un cas spécial déjouant la validation. Ces cas devraient être étudiés afin d'éliminer une plus grande proportion des erreurs.

Le temps de traitement est acceptable. Il est d'environ 18 secondes par image, ce qui donnerait 5 heures au total pour un jeu de 1000 images. S'il s'avère nécessaire d'accélérer ce traitement, il faudrait s'attarder à l'implémentation de l'algorithme RANSAC, laquelle est responsable pour la majeure partie du temps de calcul. Elle pourrait être traduite en C rapidement avec un grand gain potentiel. Finalement, comme pour la segmentation, le parallélisme pourrait être exploité.

3.3.4 Construction du modèle 3-D

Le problème de reconstruction 3-D consiste à estimer les m poses de la caméra simultanément avec les coordonnées 3-D des n points de repère (3.5).

$$\min_{\substack{\mathtt{R}_{1\ldots m} \\ \mathbf{t}_{1\ldots m} \\ \mathbf{X}_{1\ldots n}}} \sum_{j=1}^{m} \sum_{i=1}^{n} \|\mathbf{x}_{i,j} - \mathcal{P}(\mathbf{X}_i, s, f, x_0, y_0, \boldsymbol{\delta}, \mathtt{R}_j, \mathbf{t}_j)\|^2 \tag{3.5}$$

Ceci est accompli en utilisant un *bundle adjustment* itéré (se référer, par exemple, à [18]), un algorithme spécialisé basé sur la méthode des moindres carrés non linéaire minimisant la distance entre les coordonnées image observées et prédites. Il est à noter que l'initialisation de cet algorithme est un sujet particulièrement épineux. Sa qualité est essentielle puisque la fonction possède habituellement de nombreux minimums locaux piégeant l'algorithme de recherche. Elle est encore l'objet de nombreux travaux de recherche et il n'existe à l'heure actuelle que des heuristiques développés empiriquement.

Le logiciel PhotoModeler implémente l'algorithme de *bundle adjustment* ainsi que les heuristiques nécessaires à son initialisation. En entrée, on lui fournit les la position dans chaque image des points de repère détectés ainsi que les paramètres intrinsèques de la caméra. PhotoModeler calcule la position 3-D des points et la pose des caméras. On peut l'intégrer au système global et en automatiser l'usage grâce à son interface DDE[7]. Ceci permet de lui fournir programmatiquement les données d'entrée. On se sert aussi de DDE pour extraire les données de sortie, que l'on écrit ensuite dans des fichiers sous un format simple.

PhotoModeler éprouve des difficultés à partir d'environ 100 images.[8] Entre autres, il pourra faillir à trouver une solution ou tout simplement boguer. En particulier, de nombreuses discussions avec le support technique du fabricant ont amené ce dernier à fournir une version spécifique aux besoins du projet amenant une meilleure stabilité. De plus, la communication DDE a été emballée d'une couche épaisse de code de traitement d'erreur. Il en a résulté une stabilité satisfaisante sans toutefois dépasser à coup sûr la limite de 100 images.

Afin de dépasser la limite de 100 images imposée par PhotoModeler, il a été décidé de procéder en deux temps :

[7] DDE est un protocole de communication interprocessus utilisé sur le système d'exploitation Windows.

[8] La version testée était 5.2.3.

1. Utiliser jusqu'à 100 images pour construire le modèle des points de référence.
2. Estimer la pose des autres photos d'après les points du modèle observés dans chacune. Ceci est décrit à la section 3.3.5.

Le sous-ensemble des 100 images choisies doit fournir la meilleure qualité de modèle possible. De fait, il faut choisir les photos de façon à éviter tout cas pathologique. Comme exemple de tel cas on peut imaginer un ensemble de points de vue connexes, ne couvrant qu'une partie de l'objet. On choisit donc le sous-ensemble au hasard, de façon à maximiser la couverture.

Le code d'interaction par DDE avec PhotoModeler est de ma contribution, d'après une preuve de concept par Nathaniel Zoso.

Critique

La construction du modèle 3-D avec PhotoModeler se fait rarement au premier essai. La cause principale des erreurs est la mauvaise détection des codes matriciels. Il faut alors vérifier manuellement l'erreur résiduelle maximale afin d'identifier les mauvaises détections, supprimer ces points défectueux, et redémarrer le traitement. Une meilleure détection règlerait ce problème.

Parfois, PhotoModeler n'arrive toujours pas à converger pour une raison inconnue. Il affiche alors un message d'erreur et ne produit pas de résultat intermédiaire permettant de déceler la cause du problème. Régler ce problème impliquerait de programmer un nouvel algorithme de reconstruction 3-D, ce qui n'est pas une mince affaire.

Le temps de calcul varie selon le nombre d'images et le nombre de points. Puisqu'un sous-ensemble limité à 100 images est utilisé, le nombre de points est le facteur limitatif. Pour un jeu de données typique, il est de l'ordre de 10 minutes.

Il est à noter que le traitement est toujours interrompu après cette étape pour permettre à l'usager de placer le référentiel global à sa guise et de définir le facteur d'échelle. L'origine est habituellement placée au sol sous l'objet et l'axe z pointe vers le haut. Quant au facteur d'échelle, il est défini à l'aide de la distance mesurée entre deux points d'une cible. Cette interaction manuelle pourrait être retardée jusqu'à la fin du processus mais on le fait à ce moment pour deux raisons. Premièrement, PhotoModeler offre une interface graphique qui facilite la tâche. Deuxièmement, il faut déjà interrompre le processus pour éliminer manuellement les points de référence mal détectés.

3.3.5 Estimation de pose de caméra

L'estimation de pose consiste à déterminer la position et l'orientation d'une caméra à partir de l'observation de points dont les coordonnées dans un référentiel global sont connues. La pose est décrite par une rotation autour de l'origine suivie par une translation, le tout exprimé dans un référentiel cartésien global.

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \mathbf{R} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \mathbf{t} \tag{3.6}$$

On peut l'exprimer à l'aide de coordonnées homogènes sous forme matricielle :

$$\mathbf{x}' = \mathbf{P}\mathbf{x} \tag{3.7}$$

où

$$\mathbf{P} = \begin{bmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0} & 1 \end{bmatrix} = \begin{bmatrix} R_{11} & R_{12} & R_{13} & t_x \\ R_{21} & R_{22} & R_{23} & t_y \\ R_{31} & R_{32} & R_{33} & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

On forme une rotation décrite par les angles d'Euler (α, β, γ) de la façon suivante :

$$\begin{aligned}
R_{11} &= \cos(\beta)\cos(\gamma) \\
R_{12} &= \sin(\alpha)\sin(\beta)\cos(\gamma) - \cos(\alpha)\sin(\gamma) \\
R_{13} &= \cos(\alpha)\sin(\beta)\cos(\gamma) + \sin(\alpha)\sin(\gamma) \\
R_{21} &= \cos(\beta)\sin(\gamma) \\
R_{22} &= \sin(\alpha)\sin(\beta)\sin(\gamma) + \cos(\alpha)\cos(\gamma) \\
R_{23} &= \cos(\alpha)\sin(\beta)\sin(\gamma) - \sin(\alpha)\cos(\gamma) \\
R_{31} &= -\sin(\beta) \\
R_{32} &= \sin(\alpha)\cos(\beta) \\
R_{33} &= \cos(\alpha)\cos(\beta)
\end{aligned}$$

L'opération inverse, c'est-à-dire extraire les angles d'Euler d'une matrice de rotation,

se traduit par ces équations :

$$\begin{aligned} \alpha &= \arctan\left(\frac{R_{32}}{R_{33}}\right) \\ \beta &= \arcsin\left(-R_{31}\right) \\ \gamma &= \arctan\left(\frac{R_{21}}{R_{11}}\right) \end{aligned} \tag{3.8}$$

L'ambiguïté des signes est résolue par le fait que la tangente inverse est implantée de façon à retourner des valeurs entre $-\pi$ et $+\pi$.

Le modèle de caméra complet implique une projection des points connus dans un référentiel à trois dimensions vers les points observés dans le plan à deux dimensions de l'image.

$$\begin{bmatrix} \lambda x \\ \lambda y \\ \lambda \end{bmatrix} = [\mathbf{K}|\mathbf{0}]\,\mathbf{PX}$$

où

$$\mathbf{K} = \begin{bmatrix} \frac{N}{w}f & 0 & \frac{N}{w}x_0 \\ 0 & \frac{M}{h}f & \frac{M}{h}y_0 \\ 0 & 0 & 1 \end{bmatrix}$$

est la matrice des paramètres intrinsèques. Le problème d'estimation de pose se résume comme suit[9] :

> *Pour une image, connaissant* $\mathbf{K}$*, les coordonnées* (X, Y, Z) *des points réels et les coordonnées* (x, y) *des* n *points images, déterminer la pose* $\hat{\mathbf{P}}\,(\alpha, \beta, \gamma, t_X, t_Y, t_Z)$ *qui minimise l'erreur de reprojection*

$$\sum_{i=1}^{n} \left\| \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix} - \frac{[\mathbf{K}|\mathbf{0}]\,\hat{\mathbf{P}}\mathbf{X_i}}{\hat{\lambda}_i} \right\|^2 . \tag{3.9}$$

Algorithme

L'algorithme se sépare en deux parties : on applique une DLT pour produire un estimé de départ que l'on raffine par la suite avec un algorithme de minimisation non linéaire.

[9] On suppose que l'on a accès aux coordonnées corrigées (sans distorsion) des points 2-D ainsi qu'à leur correspondance avec les points 3-D.

La *direct linear transform (DLT)* est habituellement utilisée pour le calibrage d'une caméra. Elle permet de déterminer par la résolution d'un système linéaire homogène tous les paramètres du modèle de projection, c'est-à-dire les paramètres intrinsèques et extrinsèques.

La méthode consiste à réarranger chacune des équations correspondant aux points observés (deux équations par point) sous la forme d'un système linéaire homogène.

$$\underbrace{\begin{bmatrix} \mathbf{x}_1^{\mathrm{T}} & \mathbf{0} & -x_1\mathbf{x}_1^{\mathrm{T}} \\ \mathbf{0} & \mathbf{x}_1^{\mathrm{T}} & -y_1\mathbf{x}_1^{\mathrm{T}} \\ \mathbf{x}_2^{\mathrm{T}} & \mathbf{0} & -x_2\mathbf{x}_2^{\mathrm{T}} \\ \mathbf{0} & \mathbf{x}_2^{\mathrm{T}} & -y_2\mathbf{x}_2^{\mathrm{T}} \\ & \vdots & \\ \mathbf{x}_n^{\mathrm{T}} & \mathbf{0} & -x_n\mathbf{x}_n^{\mathrm{T}} \\ \mathbf{0} & \mathbf{x}_n^{\mathrm{T}} & -y_n\mathbf{x}_n^{\mathrm{T}} \end{bmatrix}}_{\mathbf{A}} \underbrace{\begin{bmatrix} m_{11} \\ m_{12} \\ \vdots \\ m_{34} \end{bmatrix}}_{\mathbf{m}} = \mathbf{0}$$

La matrice $\mathbf{A}$ est de dimensions $2n \times 12$ et le vecteur $\mathbf{m}$ des inconnues contient les entrées de la matrice de projection

$$\hat{\mathbf{M}} = \begin{bmatrix} m_{11} & m_{12} & m_{13} & m_{14} \\ m_{21} & m_{22} & m_{23} & m_{24} \\ m_{31} & m_{32} & m_{33} & m_{34} \end{bmatrix} = [\mathbf{K}|\mathbf{0}]\,\mathbf{P}.$$

La solution du système homogène est le vecteur propre correspondant à la plus petite valeur propre de $\mathbf{A}$. La décomposition en valeurs singulières (SVD) est utilisée pour la calculer. Elle produit toujours une solution de norme unitaire. On retrouve donc la matrice de projection $\hat{\mathbf{M}}$ à un facteur d'échelle près.

On connaît déjà les paramètres intrinsèques de la caméra et on peut obtenir la matrice de transformation des points du repère global au repère de la caméra (la pose) directement en inversant la matrice des intrinsèques :

$$\hat{\mathbf{P}} = \mathbf{K}^{-1}\hat{\mathbf{M}}$$

On doit ensuite appliquer des contraintes additionnelles pour s'assurer que la pose obtenue corresponde bien à une transformation rigide, c'est-à-dire que la sous-matrice $\hat{\mathbf{R}}$ doit être orthonormale et de déterminant $+1$. La façon d'obtenir la rotation qui approche le mieux une matrice 3×3 quelconque consiste à rendre unitaires les valeurs singulières de la matrice [20]. On décompose premièrement $\hat{\mathbf{R}}$ en valeurs singulières. On

la reconstruit ensuite en imposant la matrice diagonale $\mathbf{D}$ suivante.

$$\begin{aligned} \hat{\mathbf{R}} &= \mathbf{U}\mathbf{D}\mathbf{V}^{\mathrm{T}} \\ \mathbf{D} &= \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \det\left(\mathbf{U}\mathbf{V}^{\mathrm{T}}\right) \end{bmatrix} \end{aligned}$$

Le déterminant de $\mathbf{U}\mathbf{V}^{\mathrm{T}}$ sera toujours unitaire mais pourra être positif ou négatif. L'intention est que le déterminant de $\mathbf{D}$ fasse contrepoids à celui de $\mathbf{U}\mathbf{V}^{\mathrm{T}}$ pour que $\hat{\mathbf{R}}$ ait un déterminant de $+1$.

Il reste maintenant à mettre la translation à la même échelle que la rotation. Le facteur d'échelle correspond au ratio de la norme de Frobenius[10] de la rotation orthonormalisée par rapport à celle de la matrice initiale. Il a été vérifié expérimentalement que cette normalisation donnait des résultats acceptables.

Si les points appartiennent à un plan, il faut modifier la méthode parce que le système est sous-contraint. En effet, on doit estimer une homographie 3×3 plutôt qu'une projection 3×4. La matrice $\mathbf{A}$ a maintenant 9 colonnes plutôt que 12 parce que les coordonnées selon l'axe z sont toutes nulles. On peut tester si les points sont dans un plan d'après leur matrice de covariance. Si celle-ci possède deux valeurs propres fortes et une très faible, les points sont situés dans un plan. En pratique, on utilise un seuil de 1000 sur le rapport des deux plus petites valeurs propres. Si les points sont situés sur un plan autre que $z = 0$, il faut transformer les points pour les amener dans ce plan. La rotation à appliquer correspond à l'inverse de celle construite à partir des vecteurs propres de la matrice de covariance des points. Ceci annulera leur coordonnée z, rendant possible l'utilisation d'une homographie pour décrire la transformation.

Il est à noter qu'au moins 6 points en configuration générale ou 4 points appartenant à un plan doivent être observés pour que le système soit inversible. Puisqu'une pose est complètement décrite par 6 paramètres, on pourrait s'attendre à ce que 3 points observés soient suffisants. Il existe en effet des méthodes de résolution pour 3, 4 et 5 points en configuration générale [2, 17]. Il s'agit là d'une faiblesse de la DLT.

À partir de la solution donnée par la DLT, on extrait 6 paramètres décrivant la transformation rigide, c'est-à-dire les trois angles d'Euler (éq. (3.8)) et une translation en 3-D. On initialise l'algorithme de Levenberg-Marquardt avec ces paramètres. Pour minimiser (3.9), l'algorithme affectue des approximations linéaires successives en

[10] La norme de Frobenius est définie comme la somme du carré des entrées de la matrice. En équation : $\|\mathbf{A}\|_F^2 = \sum_{i=1}^{m} \sum_{j=1}^{n} a_{ij}^2$.

TAB. 3.2: Erreur de reprojection en pixels pour une scène réelle.

Algorithme	Moyenne	Maximale
DLT+LM	0.33754	1.9144
Orthogonal Iterations	3.0712	157.81

évaluant le jacobien localement. On l'estime habituellement par différences finies. Cependant, on peut accélérer les calculs en le calculant analytiquement. Ceci est possible dans le cas présent puisque l'on a accès à l'expression analytique de la fonction d'erreur à minimiser. C'est donc ce qui est fait dans l'implémentation réalisée. Finalement, puisque l'erreur minimisée est directement (3.9), l'estimateur est sans biais.

Résultats pour une scène réelle

Avec une caméra Canon PowerShot A80 de 3.7 mégapixels, 79 photos d'une scène réelle on été prises. Des points de la scène, 405 ont été vus dans au moins une photo. Un modèle de leurs coordonnées 3-D a été construit avec PhotoModeler. La précision du modèle est telle que l'erreur de reprojection maximale dans chaque image est inférieure à un pixel. Les 79 poses ont par la suite été déterminées à l'aide de l'algorithme décrit ci-dessus. L'erreur de reprojection est utilisée pour mesurer la performance de l'algorithme (voir tableau 3.2). Pour fins de comparaison, l'algorithme *Orthogonal Iterations* [30] a été testé dans les mêmes conditions.

Ce test permet seulement de comparer les deux algorithmes afin de choisir le plus performant. Pour caractériser l'erreur de reprojection réelle, il faut estimer la pose de photos n'ayant pas servi à la construction du modèle. Ceci a été fait à plusieurs reprises lors du traitement de jeux de données réels. Pour la caméra Canon EOS-300D, l'erreur de reprojection moyenne est habituellement inférieure à un pixel, mais varie selon l'angle de la caméra par rapport à la cible de positionnement (erreur plus grande lorsqu'il est rasant) et selon le sous-ensemble de 100 photos ayant servi à construire le modèle. Dans le processus de traitement complet, on rejette les photos dont l'erreur quadratique moyenne est supérieure à 0.01 millimètre, ce qui correspond à environ 1.4 pixels pour la EOS-300D. Sur un jeu de données de 599 photos, 9 photos ont ainsi été rejetées.

Le code d'estimation de pose a été écrit par moi-même.

Critique

L'algorithme DLT+LM a été sélectionné pour accomplir la tâche d'estimation de pose dans le système de modélisation de l'apparence. Il a été testé sur plusieurs jeux de données de plusieurs centaines d'images chacun. Jamais de cas pathologique (rotation erronée par près de 180 degrés) n'a été rencontré. Un tel cas aurait été détecté automatiquement puisqu'il aurait été la cause d'une cascade d'erreurs menant à un modèle invalide lors de la reconstruction de l'enveloppe visuelle, celle-ci étant sensible à des erreurs de pose.

Le temps de calcul requis est inférieur à une seconde. L'estimation de toutes les poses est limité principalement par la vitesse d'accès du disque rigide.

3.3.6 Construction de l'enveloppe visuelle

L'enveloppe visuelle a été introduite dans le monde de la vision numérique par Laurentini [26]. Conceptuellement simple, elle est décrite comme l'intersection des cônes généraux formés par la reprojection de la silhouette de l'objet dans chaque image. Pour la modélisation de l'apparence, l'enveloppe visuelle sert de géométrie approximative initiale. L'enveloppe visuelle a l'avantage d'être calculable sans présupposer un modèle de réflectance spécifique. Sa qualité en tant que support géométrique de l'apparence peut être améliorée par raffinement de surface, ce qui sera le sujet de la section 3.3.7.

On reconstruit des modèles géométriques à partir de silhouettes depuis longtemps. En 1987, Potmesil [37] décrivit un système affecté à une telle tâche. L'enveloppe y était stockée volumétriquement, chaque voxel encodant une valeur binaire indiquant s'il était plein ou vide. Szeliski [41] améliora la méthode sans en changer la représentation. Déjà, on avait reconnu la nécessité d'utiliser un octree pour stocker l'information de manière efficace.

Suivant cette représentation, le calcul de l'enveloppe visuelle peut être effectué de manière naïve suivant l'algorithme 3.2. On y représente l'enveloppe visuelle comme un ensemble de voxels contenant chacun une valeur booléenne vraie lorsque le voxel est plein et fausse lorsque le voxel est vide. L'opération d'intersection effectuée est équivalente à un ET logique, ce qui se traduit par un sculpture *sévère* de l'espace, puisqu'un seul pixel classé comme de l'arrière-plan est suffisant pour déclarer tout un ensemble de voxels vides. On peut par la suite convertir cette représentation volumétrique en un maillage en produisant l'isosurface de niveau 0.5 avec, par exemple, l'algorithme des *marching*

cubes [28].

Algorithme 3.2 Calcul d'enveloppe visuelle, représentation volumétrique binaire.

Entrée:
- Silhouettes $S_{1\ldots n}$.
- Poses $(\mathtt{R}_{1\ldots n}, \mathtt{t}_{1\ldots n})$.
- Paramètres intrinsèques $(s, f, x_0, y_0, \boldsymbol{\delta})$ de la caméra.

Sortie: Enveloppe visuelle V (volumétrique).

$V \leftarrow 1$
pour chaque silhouette S_i **faire**
  pour chaque voxel $V_{x,y,z}$ **faire**
    $\mathbf{X} \leftarrow$ centre de $V_{x,y,z}$
    si $\mathcal{P}(\mathbf{X}, s, f, x_0, y_0, \boldsymbol{\delta}, \mathtt{R}_i, \mathtt{t}_i) \notin S_i$ **alors**
      $V_{x,y,z} \leftarrow 0$
    fin si
  fin pour
fin pour

Cet algorithme souffre de défauts importants. Le premier est que, pour une bonne précision, le volume V requiert une grande quantité de mémoire. Ceci limite la précision atteignable sur des ordinateurs conventionnels. Le deuxième défaut est que de très nombreuses évaluations de la fonction de projection $\mathcal{P}$ rendent l'algorithme lent. Ce défaut est exacerbé lors du traitement de photos à haute résolution.

La résolution de ces problèmes requiert une nouvelle représentation du volume comme une fonction implicite de distance. Au lieu de déterminer si un voxel est plein ou vide, on calcule plutôt la distance de son centre à la surface. Puisque la position de la surface 3-D n'est pas connue (c'est ce qu'on cherche), on l'approxime par la distance 2-D dans les images. Pour chacune des images, on calcule la distance entre la reprojection du centre du voxel et la silhouette segmentée. On inscrit dans le voxel la distance maximale parmi tous les points de vue. Alors que l'opération d'intersection correspondait à un ET logique dans le cas de la représentation volumétrique binaire, elle correspond, dans le cas d'une fonction implicite de distance signée, au calcul du maximum élément-par-élément.

Afin d'accélérer les calculs de distance d'un point à la silhouette, on effectue un prétraitement sur ces dernières consistant à calculer leur transformée de distance 2-D. Ceci s'effectue à l'aide d'un algorithme rapide [4]. La distance à la silhouette pour un point projeté peut être obtenue par interpolation dans cette transformée. L'algorithme 3.3 résume les opérations effectuées.

Algorithme 3.3 Calcul d'enveloppe visuelle, représentation par fonction implicite.

Entrée:
- Silhouettes $S_{1...n}$.
- Poses $(\mathtt{R}_{1...n}, \mathtt{t}_{1...n})$.
- Paramètres intrinsèques $(s, f, x_0, y_0, \boldsymbol{\delta})$ de la caméra.

Sortie: Enveloppe visuelle V (fonction implicite).

$V \leftarrow -\infty$
pour chaque silhouette S_i **faire**
  $D \leftarrow$ transformée de distance 2-D de S_i
  pour chaque voxel $V_{x,y,z}$ **faire**
    $\mathbf{X} \leftarrow$ centre de $V_{x,y,z}$
    $\mathbf{x} \leftarrow \mathcal{P}(\mathbf{X}, s, f, x_0, y_0, \boldsymbol{\delta}, \mathtt{R}_i, \mathtt{t}_i)$
    si $V_{x,y,z} < D(\mathbf{x})$ **alors**
      $V_{x,y,z} \leftarrow D(\mathbf{x})$
    fin si
  fin pour
fin pour

Il est à noter que la représentation calculée par l'algorithme 3.3 n'est pas strictement une fonction implicite de distance signée mais bien une approximation de celle-ci, et ce pour deux raisons. Premièrement, le nombre fini de points de vue limite la précision atteignable. Deuxièmement, l'utilisation d'une distance dans l'image implique un effet de projection qui varie selon la profondeur de l'objet. Par exemple, la distance mesurée dans l'image sera plus grande si la caméra est loin de l'objet. Si la caméra est toujours à la même distance de l'objet, cette distorsion n'aura pas lieu. Même si cette hypothèse n'est pas strictement respectée, l'isosurface $d = 0$ ne sera pas affectée.[11] Puisque c'est celle-là qui est d'intérêt lors de la construction de l'enveloppe visuelle, l'approximation est acceptable.

La représentation par fonction implicite de distance signée permet d'obtenir une meilleure reconstruction que pour la représentation binaire à résolution égale. En fait, cette dernière peut être considérée comme un cas particulier d'une représentation par fonction implicite de distance signée où chaque valeur est représentée avec une précision de 1 bit. L'avantage d'une plus grande précision est simplement l'amélioration de l'interpolation effectuée par l'algorithme des *marching cubes*.

L'idée de cette méthode et son implémentation ont été fournis par Jean-Daniel Deschênes et Philippe Lambert. Il en résulte de bons résultats. La figure 3.6 montre 4

[11] Elle le sera très faiblement dû à l'interpolation puisque les valeurs voisines, non nulles, sont biaisées. Pour un espace continu (sans discrétisation), l'erreur est nulle.

modèles produits avec celle-ci. On remarque que la précision limitée de la grille affecte les modèles. Ceci est particulièrement visible dans les régions fines comme les skis avants de la motoneige. Notons que les erreurs de segmentation avaient été corrigées manuellement, ce qui permet d'attribuer les défauts des modèles à la méthode de contruction de l'enveloppe visuelle.

Afin d'obtenir de meilleurs modèles, on désire augmenter la résolution de la grille. Cependant, ceci devient rapidement trop gourmand en mémoire. Par exemple, on désirerait travailler avec une résolution de $1024 \times 1024 \times 1024$. Ceci nécessiterait $1024^3 \times 8 = 8$ Go de mémoire, en supposant que chaque voxel est stocké sur 64 bits (un *double*). Même si c'était possible d'allouer une telle quantité de mémoire, le temps de calcul deviendrait alors un facteur limitatif puisqu'il faudrait parcourir tout le cube à chaque image et faire un grand nombre d'appels à la fonction de projection. Afin de contourner cette limitation, il faut encore une fois changer la représentation des données.

Calcul par octree

Un octree est une structure de données couramment utilisée afin de compression de l'information volumétrique. La stockage en mémoire se fait à la manière traditionnelle des arbres en informatique, à l'aide de noeuds reliés par des pointeurs. L'entrée de la structure est le pointeur vers le noeud racine. Trois types de noeuds sont utilisés : les noeuds branche, qui contiennent 8 pointeurs vers des noeuds enfant, les noeuds feuille, qui contiennent des données, et les noeuds nuls, qui sont représentés par un pointeur nul dans le noeud branche parent. Les noeuds feuille et nuls n'ont pas d'enfant : ils sont les terminaisons de l'arbre.

La méthode proposée est inspirée de celle décrite dans [37]. Dans celle-ci, un octree est utilisé pour stocker efficacement une représentation volumétrique binaire. Sa construction repose sur un critère de subdivision simple : si un cube d'un niveau n contient la surface, on le subdivise en 8 cubes de niveau $n+1$. On détermine à l'avance un niveau maximal de subdivision. À ce niveau maximal, on procède comme pour l'algorithme sans octree : on calcule une valeur binaire vraie si la projection du cube est comprise dans toutes les silhouettes.

Cet algorithme est reproduit ici avec quelques modifications (algorithme 3.4) pour fins de comparaison. Il est important de faire ressortir quelques-unes de ses caractéristiques. En particulier, notons qu'un noeud vide (valeur binaire zéro) est représenté par un noeud nul (équivalent à un pointeur nul) alors qu'un noeud plein (valeur binaire un) est représenté par un noeud feuille. L'octree doit être initialisé avec un noeud feuille

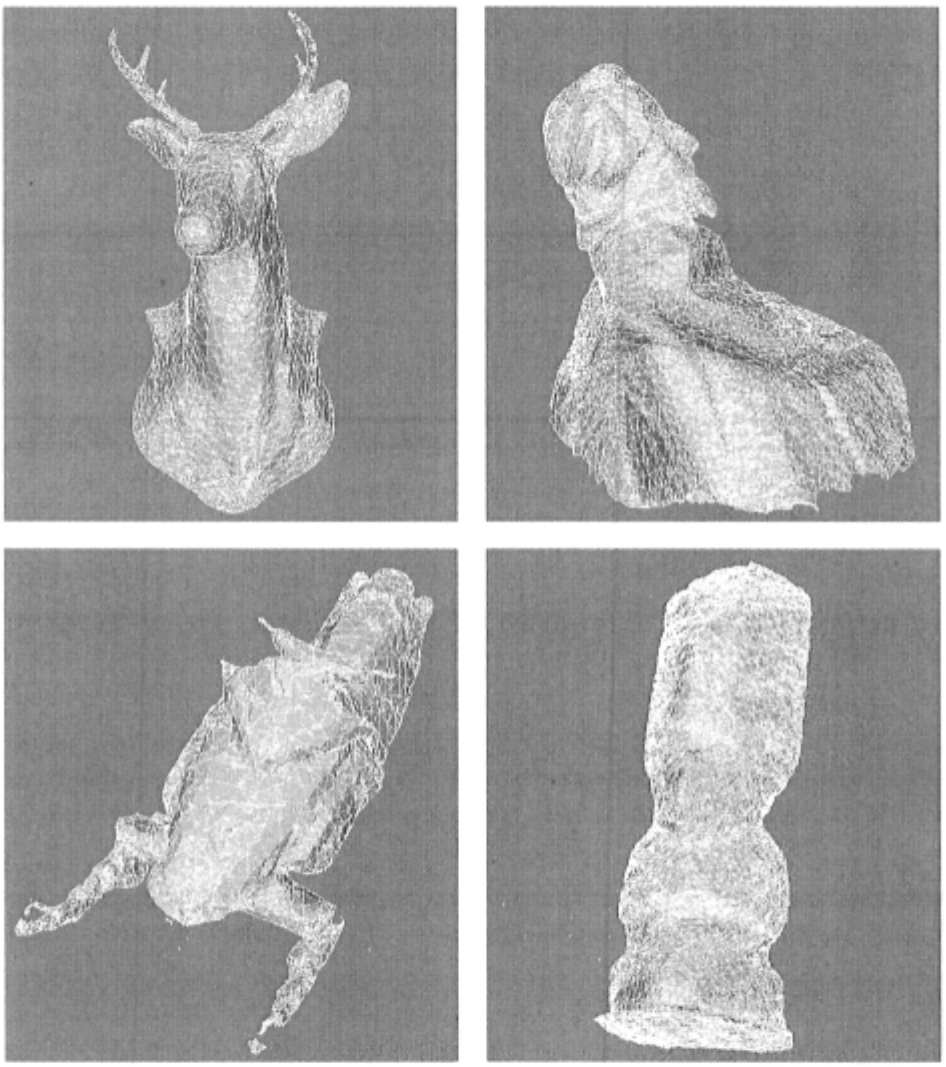


FIG. 3.6: Maillages produits suivant l'algorithme 3.3. La résolution de la grille est de $150 \times 150 \times 150$.

pour signifier que le volume est plein en l'absense d'information. Le traitement successif des silhouettes enlève des parties à ce volume et les remplace par des noeuds nuls. Il est aussi important de mentionner que l'extérieur de l'image est équivalent à la silhouette. En d'autres termes, si la projection d'un noeud tombe à l'extérieur de l'image, on n'obtient aucune information. On ne peut donc pas transformer un noeud feuille en un noeud nul, ce qui est équivalent au cas où la projection du noeud était tombée dans la silhouette.

On remarque aussi que l'on doit effectuer une nouvelle opération, le calcul de la silhouette d'un noeud. Ceci peut s'effectuer en trois étapes. Premièrement, on calcule un maillage à base de 12 triangles représentant le cube. Deuxièmement, on projète les sommets du cube. Troisièmement, on rastérise chaque triangle, c'est-à-dire que l'on détermine la liste des pixels formant sa silhouette. Les calculs subséquents d'intersection de silhouettes se font par comparaison de listes de pixels. En pratique, la rastérisation est étonamment rapide. Le goulot d'étranglement se situe plutôt au niveau de la projection de points dans l'image.

Afin d'obtenir une meilleure précision, on veut stocker dans l'octree une représentation par fonction implicite de distance signée plutôt que de simples valeurs binaires. Pour ce faire, on s'inspirera fortement de l'algorithme 3.4.

On propose dans [11] une méthode similaire. Cependant, l'algorithme est complexifié grandement parce que la représentation par fonction implicite de distance n'est pas constante par morceaux, contrairement à la représentation volumétrique binaire. En effet, l'avantage d'un octree est qu'il permet de remplacer plusieurs petits cubes connexes possédant la même valeur par un seul grand cube. Cependant, dans une fonction implicite de distance, la valeur change toujours d'un cube à l'autre même si on se situe complètement à l'extérieur ou à l'intérieur de l'objet.

Ceci amène les auteurs à utiliser un critère de subdivision différent, basé sur la complexité de la forme de la silhouette. On y fait usage d'un octree multirésolution, ce qui introduit des problèmes de continuité de surface lors du passage subséquent à une représentation analytique. Pour combler cette lacune, les auteurs proposent un heuristique appliqué en post-traitement visant à "boucher les trous" de la surface reconstruite.

L'approche proposée est beaucoup plus simple (algorithme 3.5). On impose simplement deux valeurs maximale (d_{max}) et minimale (d_{min}) à la fonction de distance. Ceci créera une mince couche de part et d'autre de la surface qui sera représentée à la résolution maximale de l'octree. Le reste de l'espace pourra être compressé de la manière habituelle. Comme dans l'algorithme 3.4, les noeuds passent de plein à vide,

Algorithme 3.4 Construction de l'octree pour calcul d'enveloppe visuelle avec représentation volumétrique binaire. Adapté de [37].

Entrée: Un noeud de l'octree, initialement la racine, et une silhouette calibrée.

Sortie: Le noeud et sa descendance sont modifiés pour intégrer l'information fournie par la silhouette.

si le noeud courant est un noeud *vide* **alors**
  Terminer.
sinon si le noeud courant est un noeud *branche* **alors**
  Visiter (récursivement) tous ses noeuds successeurs.
  si tous les noeuds successeurs sont vides **alors**
    Remplacer le noeud courant par un noeud vide (le supprimer).
  sinon si tous les noeuds successeurs sont pleins **alors**
    Supprimer les successeurs.
    Remplacer le noeud courant par un noeud plein (*feuille*).
  fin si
sinon
  Le noeud courant est un noeud *feuille*
  si la résolution maximale de l'octree est atteinte **alors**
    si le centre du noeud courant est devant la caméra **et** la projection du centre du noeud courant n'est pas dans la silhouette de l'objet **alors**
      Remplacer le noeud courant par un noeud vide (le supprimer) et terminer.
    fin si
  sinon si le noeud courant est entièrement devant la caméra **alors**
    Calculer la silhouette du noeud courant telle que vue du centre de projection de l'image.
    si la silhouette du noeud courant n'intersecte pas la silhouette de l'objet **alors**
      Remplacer le noeud courant par un noeud vide (le supprimer) et terminer.
    sinon si la silhouette du noeud courant est comprise dans la silhouette de l'objet **alors**
      Terminer.
    sinon
      Remplacer le noeud courant par un noeud branche.
      Générer les 8 successeurs comme des noeuds feuille.
      Visiter (récursivement) le noeud branche.
    fin si
  fin si
fin si

mais ceux-ci ne sont pas représentés de la même manière. Un noeud plein, dont la valeur est d_{min}, est représenté par un noeud nul. Les autres noeuds, dont la valeur est comprise dans $]d_{min}, d_{max}]$, sont représentés par une feuille dans laquelle on inscrit la valeur numérique. C'est par choix que l'on associe un noeud nul à d_{min} plutôt qu'à d_{max}. Ceci permet d'initialiser plus rapidement une branche puisque l'on n'a pas besoin d'allouer ses huit successeurs (ils sont représentés par des pointeurs nuls).

Optimisations

Afin d'économiser de la mémoire, on peut appliquer quelques optimisations. La plus simple est d'exprimer les distances dans les feuilles sur un nombre restreint de bits. Par exemple, en posant $d_{min} = -10$ et $d_{max} = 10$, on peut obtenir une précision du dixième de pixel avec des distances signées de seulement 8 bits. C'est ce qui est utilisé dans l'implémentation réalisée.

Ensuite intervient une optimisation non triviale. Chaque feuille ne contient qu'une donnée de distance exprimée sur 8 bits alors que les branches contiennent 8 pointeurs, chacun stocké sur 4 octets (sur un ordinateur 32 bits). La taille des branches est alors démesurée par rapport à celle des feuilles. Puisque chaque feuille nécessite un pointeur pour pointer vers elle-même à partir du noeud branche parent, chaque donnée de distance nécessite en fait 5 octets plutôt qu'un seul. La façon de régler ce problème est d'augmenter la taille relative des feuilles par rapport aux branches. Ceci est accompli par l'aggrégation de plusieurs feuilles. Par exemple, dans l'implémentation réalisée, les aggrégats ont une taille de $4 \times 4 \times 4$. Un pointeur de 4 octets est alors nécessaire pour un seul aggrégat de 64 octets. La taille des aggrégats représente un compromis et sera appelée à varier selon le nombre d'octets alloués pour représenter les nombres, la taille des pointeurs, et même la distribution des données. En effet, si les données étaient très éparpillées, les aggrégats deviendraient moins attrayants puisqu'une faible proportion de chacun serait utile. Dans le cas de données peu éparpillées, comme celui de la représentation d'une surface, les aggrégats sont plus efficients.

Calcul du maillage

L'algorithme *Marching Cubes* [28] peut être utilisé avec peu de modification pour produire le maillage. Lors de son exécution, il doit visiter tous les voxels. Lors du traitement d'un voxel, les données de 7 voxels voisins sont requises. Pour accomoder l'algorithme, on peut considérer l'octree comme un type de donnée abstrait et implémenter

Algorithme 3.5 Construction de l'octree pour calcul d'enveloppe visuelle avec représentation par fonction de distance signée.

Entrée: Un noeud de l'octree, initialement la racine, et une silhouette calibrée.

Sortie: Le noeud et sa descendance sont modifiés pour intégrer l'information fournie par la silhouette.

si le noeud courant est un noeud *vide* **alors**
  Terminer.
sinon si le noeud courant est un noeud *branche* **alors**
  Visiter (récursivement) tous ses noeuds successeurs.
  si tous les noeuds successeurs sont vides (feuilles à d_{max}) **alors**
    Supprimer les successeurs.
    Remplacer le noeud courant par un noeud vides (feuille à d_{max}).
  sinon si tous les noeuds successeurs sont pleins (noeuds nuls) **alors**
    Remplacer le noeud courant par un noeud vide (le supprimer).
  fin si
sinon
  Le noeud courant est un noeud *feuille*
  si la résolution maximale de l'octree est atteinte **alors**
    si le centre du noeud courant est devant la caméra **alors**
      Calculer la distance d à la surface, dans l'image, pour la projection du centre du noeud courant.
      si $d > d_{min}$ **alors**
        Remplacer le noeud courant par une feuille et y inscrire $\min(d, d_{max})$.
      fin si
    fin si
  sinon si le noeud courant est entièrement devant la caméra **alors**
    Calculer la silhouette du noeud courant telle que vue du centre de projection de l'image.
    si les pixels de la silhouette du noeud courant sont tous à une distance $\geq d_{max}$ de la silhouette de l'objet **alors**
      Remplacer le noeud courant par un noeud vide (feuille à d_{max}).
    sinon si les pixels de la silhouette du noeud courant sont tous à une distance $\leq d_{min}$ de la silhouette de l'objet **alors**
      Terminer.
    sinon
      Remplacer le noeud courant par un noeud branche.
      Générer les 8 successeurs comme des noeuds nuls.
      Visiter (récursivement) le noeud branche.
    fin si
  fin si
fin si

un opérateur d'indexation retournant la valeur associée à un triplet (x, y, z) donné. Pour accéder au noeud feuille associé à ce triplet, l'opérateur doit déréférencer un nombre de pointeurs souvent égal au nombre de niveau de l'octree. Ceci est peu efficient.

Un compromis efficient consiste à extraire un plan $z = k$ du volume et à le stocker dans une matrice 2-D. Ceci peut se faire à l'aide d'une fonction récursive. On voit facilement que le nombre de pointeurs déréférencés pour chaque voxel sera au maximum $\sum_{i=0}^{n} \frac{1}{4^i}$, où n est le nombre de niveaux de l'octree.

Les deux plans $z = 0$ et $z = 1$ sont suffisants pour procéder au calcul du maillage pour la première couche de voxels. Par la suite, on remplace le plan $z = 0$ par le plan $z = 2$ et on calcule le maillage pour la deuxième couche de voxels. On procède ainsi jusqu'à ce que tout le volume ait été couvert. Ce faisant, deux plans sont tenus en mémoire de façon dense, ce qui n'est pas excessif. En fait, l'article original [28] mentionnait l'astuce des deux plans pour économiser la mémoire : la quantité de mémoire que les auteurs avaient à leur disposition était tellement faible que le modèle ne pouvait pas y tenir au complet et seuls deux plans étaient lus à partir du disque à la fois.

Critique

La qualité de l'enveloppe visuelle reconstruite n'est limitée que par la qualité de la segmentation et du positionnement des caméras. Une résolution de $1024 \times 1024 \times 1024$ est présentement utilisée. La figure 3.7 montre l'évolution de l'utilisation de mémoire selon le nombre d'images. On remarque qu'elle diminue rapidement. Ceci est expliquable par le fait que chaque nouveau point de vue coupe une partie de l'espace et donc il reste une moins grande surface à stocker. Pour ce jeu de données de 100 images, 18 minutes sont requises pour le traitement. On peut extrapoler qu'un jeu de données de 1000 images serait traité en 3 heures.

Il est à noter que le traitement est interrompu avant de procéder au calcul de l'enveloppe visuelle. On demande alors à l'usager de spécifier la taille et les coordonnées dans le référentiel global du cube où l'enveloppe visuelle sera calculée. Une interface visuelle a été conçue pour faciliter la tâche (voir figure 3.8). Elle reprojette simplement les arêtes du cube dans une photo. L'utilisateur peut passer d'une photo à l'autre pour vérifier que le cube englobe bien l'objet à partir de tous les angles. Il serait avantageux d'automatiser cette étape. Ceci serait probablement possible en tenant compte de la disposition hémisphérique des caméras.

Les figures 3.9 à 3.12 montrent quelques résultats obtenus. Il est à noter que ceux-

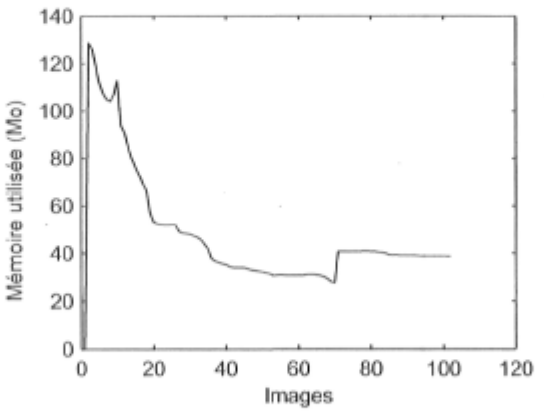


FIG. 3.7: Évolution de la quantité de mémoire utilisée par l'octree. Le jeu de données traité est celui du chevreuil et la résolution est de $1024 \times 1024 \times 1024$.

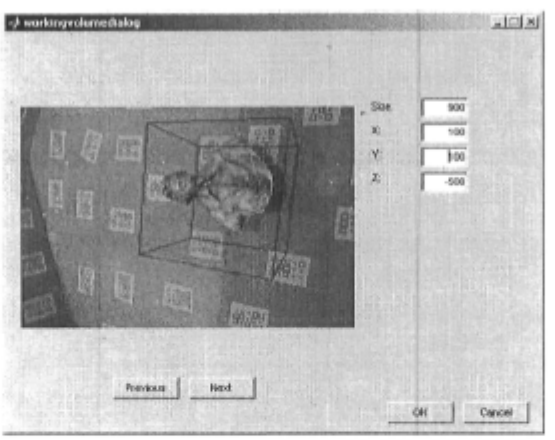


FIG. 3.8: Positionnement manuel du cube de calcul de l'enveloppe visuelle.

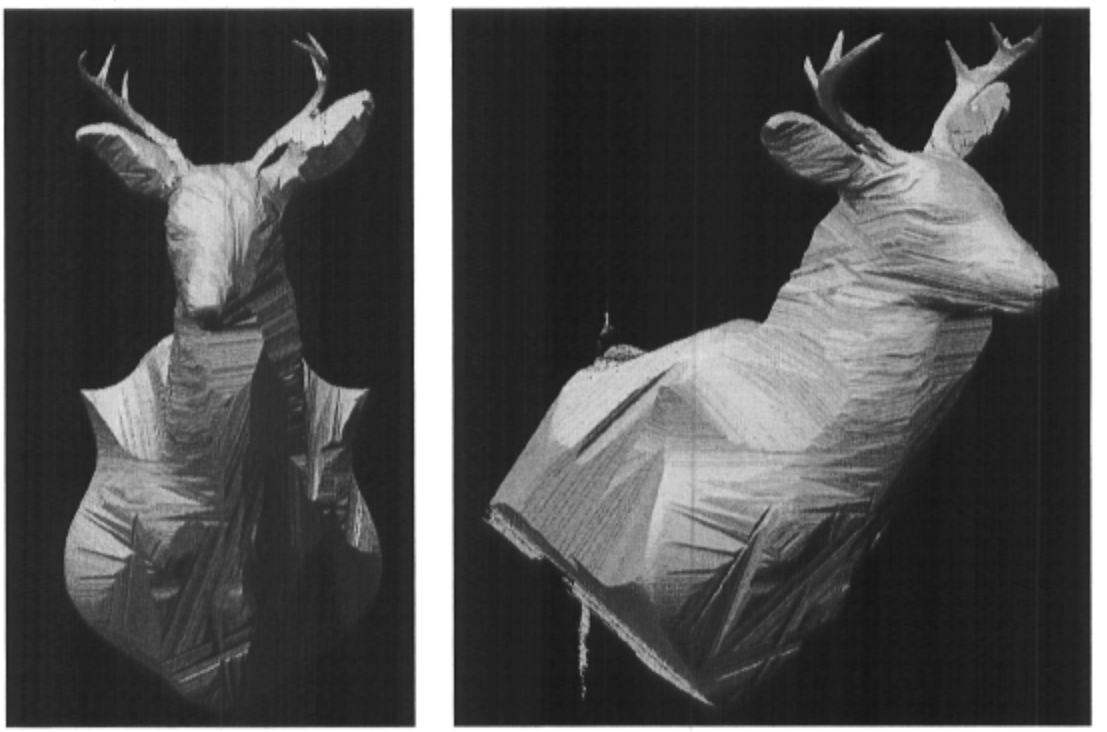


FIG. 3.9: Enveloppe visuelle du jeu de données *chevreuil* (100 photos) calculée sur une grille de $1024 \times 1024 \times 1024$ voxels en utilisant la méthode avec octree proposée.

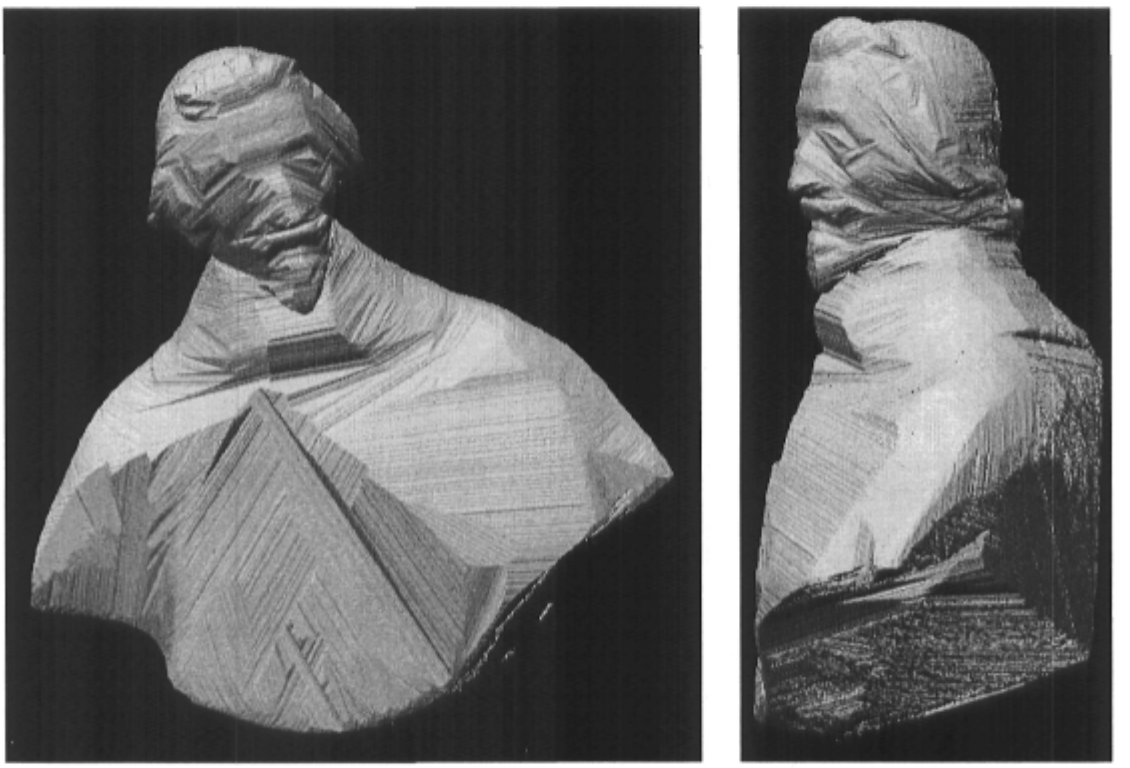


FIG. 3.10: Enveloppe visuelle du jeu de données *richelieu* (236 photos) calculée sur une grille de $1024 \times 1024 \times 1024$ voxels en utilisant la méthode avec octree proposée.

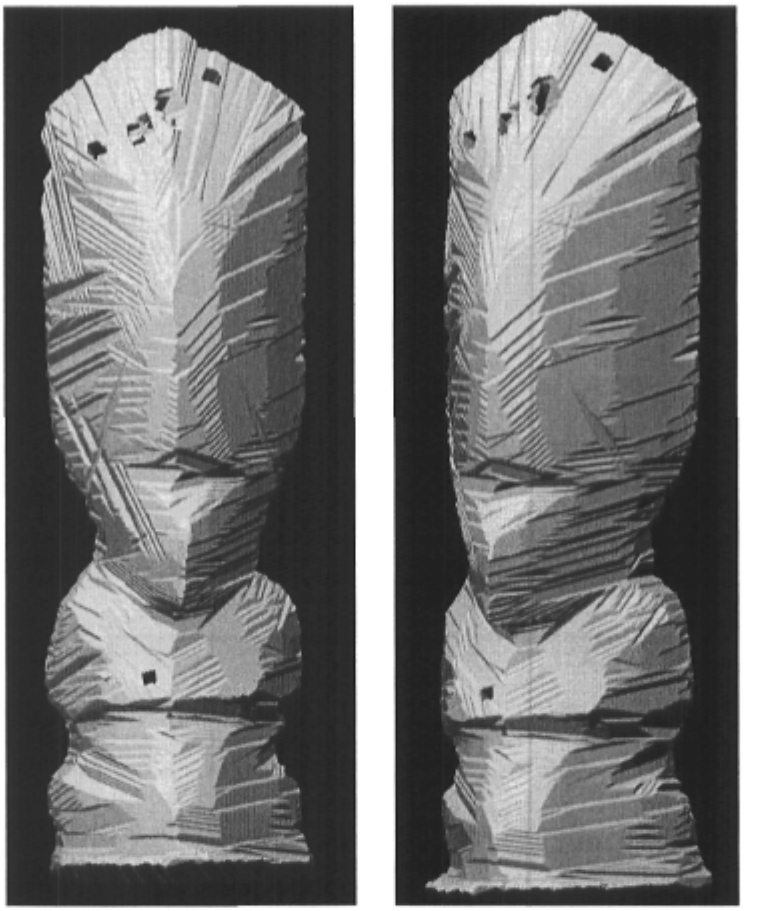


FIG. 3.11: Enveloppe visuelle du jeu de données *statue* (309 photos) calculée sur une grille de $1024 \times 1024 \times 1024$ voxels en utilisant la méthode avec octree proposée.

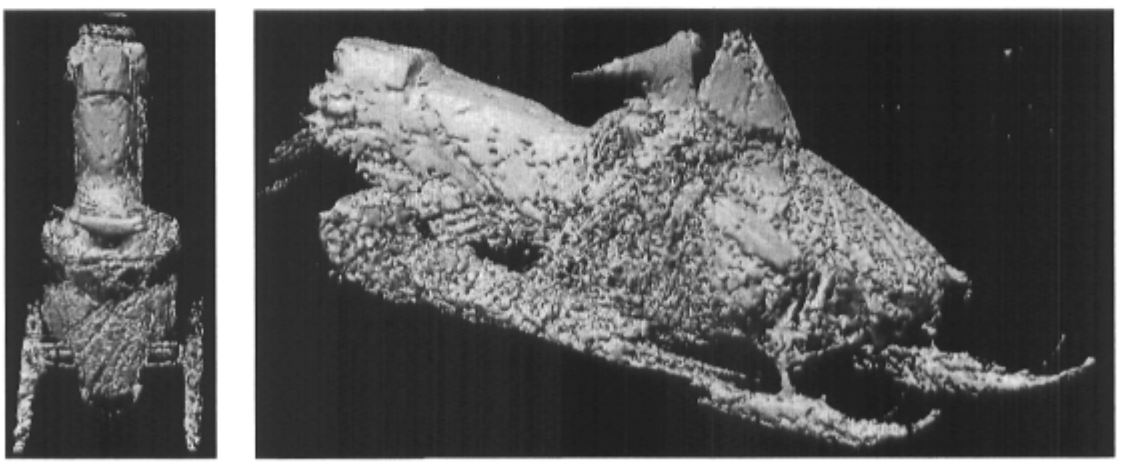


FIG. 3.12: Enveloppe visuelle du jeu de données *skidoo* (599 photos) calculée sur une grille de $1024 \times 1024 \times 1024$ voxels en utilisant la méthode avec octree proposée.

ci n'ont pas été travaillés manuellement, contrairement aux maillages montrés à la figure 3.6. La segmentation automatique a été utilisée telle quelle, sans boucher de trous. L'effet des trous est d'ailleurs particulièrement visible à la figure 3.11, au haut du modèle, ainsi qu'à la figure 3.12. On remarque sur cette dernière la différence de qualité entre le banc de la motoneige et la carosserie. Ce modèle est représentatif de la difficulté de segmenter un objet spéculaire par rapport à un objet mat. Un défaut de segmentation d'un type différent a produit l'artéfact visible à la figure 3.9. Il s'agit d'un interstice entre les panneaux de bois du plancher qui créait une ombre mal classée.

On remarque que la surface des objets n'est pas lisse mais plutôt rainurée. On propose l'explication suivante. Rappelons-nous que la surface est formée de l'intersection de cônes généraux. Chaque cône est tangent à l'objet en une région nommée *contour*. Ce contour est identifié par la segmentation de l'objet dans l'image associée au cône. Si le périmètre de la silhouette est lisse, alors le contour, et par extension le cône lui-même, le sera aussi. À l'inverse, si le périmètre de la silhouette est dentelé, alors le cône sera rainuré. L'intersection de plusieurs de ces cônes rainurés donnerait naissance à une surface telle que celles observées.

Mais comment justifier l'existence de silhouettes dentelées ? L'explication est fort simple. Rappelons-nous que le résultat de la segmentation, exprimé en tons de gris, doit être seuillé afin de produire un masque binaire. Ceci est nécessaire afin de calculer la transformée de distance 2-D. La silhouette en noir et blanc est naturellement dentelée puisqu'elle est formée de pixels carrés. Si on évitait le seuillage, les surfaces seraient sûrement plus lisses puisque l'on pourrait extraire le contour à un niveau subpixel.

3.3.7 Raffinement de surface

Le maillage obtenu à l'étape précédente peut être amélioré. Premièrement, il est souvent nécessaire de le *décimer*, c'est-à-dire de réduire son nombre de triangles en préservant le plus possible la forme globale. Ceci est nécessaire parce que le nombre de triangles dépend de la résolution de l'échantillonnage et non de la complexité de la forme, et excède habituellement 1 000 000 avec la résolution utilisée (grille de 1024^3 voxels). Le temps requis par le raffinement de surface est influencé directement par le nombre de triangles. Le logiciel PolyWorks[12] est affecté à cette tâche de décimation.

Deuxièmement, on peut modifier le maillage de façon à obtenir un meilleur rendu de l'apparence au moment de l'affichage. En effet, l'enveloppe visuelle n'est pas la surface

[12] http ://www.innovmetric.com/

la plus appropriée afin de représenter l'apparence d'un objet. Rappelons-nous que le but du projet est un modèle de l'apparence de l'objet photo-réaliste, et non une géométrie précise. On ne calcule une surface que pour avoir un support pour l'information d'apparence. Or, on montre dans [25] que l'on peut améliorer la surface, c'est-à-dire la rendre un meilleur support pour l'information d'apparence, en minimisant la fréquence angulaire moyenne de l'information d'apparence stockée à chaque sommet du maillage. L'information peut alors être mieux compressée puisque l'entropie à chaque sommet est plus faible.

L'algorithme proposé dans [25] permettant de minimiser le contenu fréquentiel à chaque sommet est le suivant. On déplace chaque sommet dans la direction de la normale jusqu'à ce que la position du minimum soit atteinte. Afin d'évaluer le contenu fréquentiel en un point, on reprojetette le point dans toutes les images où il est visible (c'est-à-dire non occulté par l'objet lui-même) et on interpole pour trouver la valeur des rayons de lumière. On effectue une transformée de Fourier 2-D sur ces données, où les dimensions du plan sont simplement les angles d'élévation et d'azimuth par rapport à la surface. Philippe Lambert travaille présentement sur cette partie du système. J'ai contribué à accéléré une bonne partie des calculs, sans en changer significativement les algorithmes.

Critique

Un désavantage de cet algorithme, et de nombreux autres similaires [38], est que la visibilité des sommets du maillage dans les photos peut changer lorsque l'un d'entre eux est déplacé. Un déplacement peut donc invalider le résultat pour un sommet précédent, c'est-à-dire que ce sommet ne se situe plus à la position de contenu fréquentiel minimal, puisque le pinceau lumineux y passant peut avoir été modifié. Le présent algorithme n'en tient pas compte. Il est de ce fait *glouton*. Une amélioration considérable serait d'utiliser une méthode de raffinement de surface globale, agissant sur toute la surface à chaque itération. On montre dans [16] une approche possible.

3.3.8 Compression et affichage

Il reste ensuite à compresser les données de façon à pouvoir les afficher en temps réel. Le logiciel *Light Field Mapper* [8] a été affecté à cette tâche. Quelques modifications mineures lui ont été apportées par Jean-Daniel Deschênes afin qu'il puisse traiter des jeux de données plus imposants.

3.4 Conclusion

Les traitements décrits constituent un système complet et fonctionnel de modélisation de l'apparence. Les trois grands problèmes abordés sont la segmentation d'image, le positionnement des photos et la reconstruction d'un modèle par silhouette. Le chapitre suivant montre quelques résultats et discute de l'atteinte des objectifs fixés.

Chapitre 4

Résultats, validation et conclusion

On présente dans ce chapitre quelques résultats du système de modélisation intégré. On procède aussi à la validation du système par rapport à l'atteinte des objectifs fixés.

Il est important de mentionner que les travaux menant à la rédaction de ce mémoire portaient sur l'intégration de plusieurs algorithmes, l'identification des problèmes de performance majeurs, et la proposition de solutions à ces derniers. Quant à la qualité des résultats produits par ces algorithmes, elle a été critiquée et des pistes de solutions ont été énoncées au chapitre 3.

4.1 Résultats

Les résultats présentés ici sont préliminaires puisque le système est toujours en développement. Il s'agit des meilleurs résultats produits jusqu'au moment de l'écriture de ce mémoire.

La méthode de reconstruction d'enveloppe visuelle par octree n'a pas été utilisée pour produire ces modèles. Cependant, puisque la même représentation par fonction implicite de distance est utilisée, l'enveloppe visuelle produite est la même. Elle est cependant calculée sur une grille moins précise ($150 \times 150 \times 150$) puisqu'elle demande plus de mémoire.

La figure 4.1 montre quelques modèles obtenus. Pour ceux-ci, le raffinement de surface n'a pas été utilisé puisqu'il est encore l'objet de nombreux développements. Il

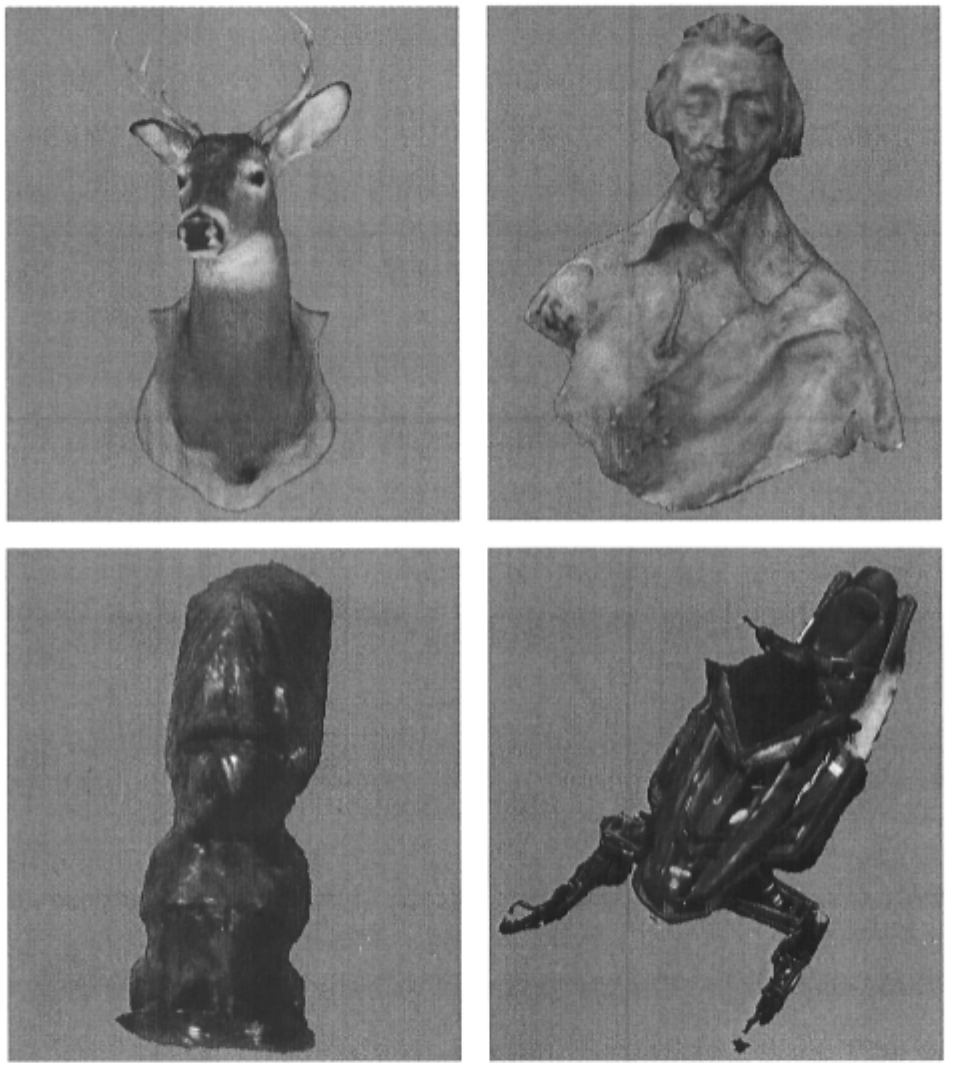


FIG. 4.1: Modèle finaux incluant les textures calculées par *Light Field Mapper*. Le raffinement de surface n'a pas été appliqué.

TAB. 4.1: Temps de calcul des étapes du traitement.

Étape de traitement	Temps mesuré	Extrapolation pour 1000 images
Calibrage	8 secondes par image	1 minute
Segmentation	2×75 secondes par image	41.7 heures
Détection des codes matriciels	18 secondes par image	5 heures
Rectification de la distorsion	6.5 secondes par image	108 minutes
Transformée de distance 2-D	3.3 secondes par image	55 minutes
Calcul du modèle 3-D des cibles	10 minutes pour 100 images	10 minutes
Estimation des poses	$<$ 1 seconde par image	1 minute
Reconstruction de l'enveloppe visuelle	18 minutes pour 100 images	3 heures
Light Field Mapper	1 heure pour 100 images	10 heures
Temps total		62.6 heures

est aussi important de noter que la segmentation a dû être ajustée manuellement afin de boucher les trous, surtout dans le cas de la motoneige.

4.2 Validation des algorithmes

Le tableau 4.1 résume les temps de calcul mesurés pour les étapes de traitement. De plus, son temps de calcul varie souvent au gré des développements. On remarque que la segmentation prend de loin le plus de temps. Le reste des traitements prend moins de 21 heures. L'objectif d'un temps total de 24 heures n'est pas encore atteint.

Dans sont état actuel, le système n'atteint pas l'objectif d'éliminer toute intervention humaine. Elle est encore nécessaire à deux endroits. Le premier est à la fin de la reconstruction du modèle 3-D des cibles et a deux objectifs : fixer le facteur d'échelle et le référentiel global, ainsi qu'éliminer les erreurs de détection des codes matriciels. La deuxième intervention humaine se produit avant de créer l'enveloppe visuelle afin de choisir la position du cube dans lequel elle sera calculée. Cependant, le système a été intégré de façon à ce que ces deux interventions ne soient séparées que par l'estimation des poses des caméras, opération qui prend environ une minute. Les deux interventions se résument donc à une seule interruption du processus de modélisation. Notons que l'on ne considère pas les interventions humaines effectuées au tout début ou à la toute fin, telle la segmentation manuelle de quelques images pour l'entraînement de l'algorithme de segmentation, comme des interruptions du processus.

Le système est simple et modulaire. Chaque étape est implémentée comme une fonction séparée. Les données intermédiaires sont stockées sur le disque dans des formats communs, simples d'interprétation de visualisation. Le langage de programmation Matlab est utilisé le plus possible. Les fonctionnalités implémentées en C/C++ sont : la détection de la cible de calibrage, la segmentation, la rectification de la distorsion et la reconstruction de l'enveloppe visuelle.

Finalement, le système n'est pas entièrement portable. Il fait usage de trois composantes qui ne sont présentes que sur le système d'exploitation Windows : les logiciels PhotoModeler, PolyWorks et Light Field Mapper. Le premier sert à la reconstruction 3-D, le deuxième à décimer le maillage produit par la reconstruction de l'enveloppe visuelle, et le dernier à compresser les données et produire un modèle affichable en temps réel.

Le reste du système est portable. Tout au long du développement, il a été utilisé autant avec Linux qu'avec Windows. Le code C/C++ a nécessité une attention particulière afin d'éviter des constructions spécifiques à un compilateur ou à un système d'exploitation donné. L'utilisation de la librairie portable OpenCV a facilité l'atteinte de cet objectif.

4.3 Recommandations

Trois objectifs n'ont pas été entièrement atteints : la portabilité du système, la limite de temps de traitement de 24 heures pour un jeu de données de 1000 photos, et l'élimination complète d'interruption du processus pour une intervention humaine. Quelques pistes de solutions ont été avancées, dont voici un résumé.

En vue d'atteindre l'objectif de portabilité, le remplacement de PhotoModeler par une solution portable nécessiterait le développement de nos propres heuristiques d'initialisation du *bundle adjustment*. Quelques tentatives exploratoires ont révélé que ce n'était pas une mince affaire. Quant au *bundle adjustment* lui-même, une implémentation en C++ de bonne réputation est disponible [29]. Pour remplacer PolyWorks, il faudrait utiliser une librairie de décimation de maillage. La librairie GTS[1] pourrait sûrement convenir. Quant à Light Field Mapper, on pourrait probablement le porter à un système d'exploitation différent puisqu'on en possède le code source.

Le temps de traitement pourrait être diminué facilement en tirant profit du pa-

[1]The GNU Triangulated Surface Library, http ://gts.sourceforge.net/.

rallélisme inhérent aux traitements. En particulier, la segmentation, à laquelle on attribue la plus grande partie du temps de traitement, pourrait en bénéficier. Avec l'omniprésence des processeurs à multiples coeurs, on peut déjà affirmer que les systèmes informatiques devront exploiter le parallélisme afin de tirer pleinement profit du matériel disponible. De plus, distribuer les calculs sur plusieurs ordinateurs reliés en réseau ne demanderait pas beaucoup plus d'efforts une fois que le mécanisme de partage des données fonctionne sur un même ordinateur.

Le deuxième traitement le plus long est l'exécution de *Light Field Mapper*. Il n'est malheureusement pas facilement parallélisable. Par contre, le code source est disponible et on devrait en faire le profilage afin d'identifier les possibilités d'optimisation. Advenant le cas où des modifications majeures seraient entreprises, on devrait en profiter pour changer le format de données afin qu'il réponde au besoin de simplicité.

Le troisième traitement le plus long est la détection des codes matriciels. Elle est présentement programmée entièrement en Matlab et la majorité du temps est passé dans l'implémentation de l'algorithme RANSAC. La conversion de cette fonction en langage C apporterait un gain significatif et ne nécessiterait pas beaucoup d'efforts.

Afin d'éliminer complètement l'intervention humaine, trois changements doivent être apportés. Premièrement, il faut éliminer le besoin de valider les résultats de la détection des codes matriciels en rendant celle-ci plus robuste ou en détectant et rejetant les points pour lesquels l'erreur de reprojection est anormalement élevée. Deuxièmement, il faut implémenter un algorithme permettant de calculer les coordonnées du cube de l'enveloppe visuelle d'après la position hémisphérique des caméras. Troisièmement, il faut déplacer le positionnement du référentiel à la toute fin du processus.

Il est aussi important de rappeler que la qualité de la segmentation est présentement le facteur ayant la plus grande influence sur la qualité du modèle final. Il est donc recommandé de développer un meilleur algorithme de segmentation, possiblement en exploitant la pose de la caméra.

4.4 Conclusion

Un système de modélisation de l'apparence d'objets réels a été analysé, critiqué, et amélioré. Les travaux de recherche menant à la rédaction de ce mémoire ont aussi porté sur l'intégration de différents algorithmes, l'identification de goulots d'étranglement et la mise en oeuvre de solutions.

Le premier modèle construit, le *chevreuil*, avait nécessité des efforts de la part de 4 personnes au cours de deux semaines intensives. Après l'intégration des différentes parties et l'optimisation de fonctions cruciales, le même jeu de données a été traité par une seule personne en une demi-journée. Une telle réduction en temps et en effectifs nécessaires résultera sans nul doute en un rythme de développement de beaucoup accéléré.

Le système est seulement à sa première version et produit déjà des résultats impressionnants. Au moment de l'écriture de ce mémoire, on commence la planification de la deuxième version. Bien que d'une construction mécanique complètement nouvelle, elle bénéficiera des progrès algorithmiques réalisés ici. Le même système informatique évoluera pour traiter des jeux de données encore plus imposants. Les algorithmes seront améliorés pour produire des modèles d'une qualité encore meilleure, plus rapidement et plus facilement.

Annexe A

Format OBJ

Le format OBJ sert à décrire un maillage 3-D constitué de triangles. C'est un format textuel basé sur l'encodage ASCII. Un sous-ensemble du format OBJ complet est utilisé dans le système et est décrit ici.

Le fichier est séparé en lignes par un caractère de saut de ligne. Chacune commence par une lettre identifiant le type de donnée représentée. Seuls deux types de données sont utilisés : les sommets et les faces.

Un sommet commence par la lettre "v" suivie des coordonnées XYZ. Il n'y a pas d'ordre à respecter. L'orientation de la normale figure dans une autre commande du format OBJ qui n'est pas utilisée ici puisque les normales ne sont pas calculées.

Une face commence par la lettre "f" suivie des 3 indices des lignes correspondant à des sommets. La face correspond au triangle formé par ces trois sommets. L'indice de la première ligne est 1.

Voici un exemple complet :

```
v 12.34 56.78 90.12
v 34.56 78.90 123
v 456 789 1011
f 1 2 3
```

Bibliographie

[1] L. Alparone, V. Cappellini & A. Garzelli – « A Coarse-to-Fine Algorithm for Fast Median Filtering of Image Data With a Huge Number of Levels », *Signal Processing* **39** (1994), no. 1-2, p. 33–41.

[2] A. Ansar & K. Daniilidis – « Linear Pose Estimation from Points or Lines », *IEEE Transactions on Pattern Analysis and Machine Intelligence* **25** (2003), no. 5, p. 578–589.

[3] G. Arce – « Multistage Order Statistic Filters for Image Sequence Processing », *IEEE Transactions on Signal Processing* **39** (1991), no. 5, p. 1146–1163.

[4] H. Breu, J. Gil, D. Kirkpatrick & M. Werman – « Linear Time Euclidean Distance Algorithms », *IEEE Transactions on Pattern Analysis and Machine Intelligence* **17** (1995), no. 5, p. 529–533.

[5] P. Carayon, M. Portier, D. Dussossoy, A. Bord, G. Petitpretre, X. Canat, G. Le Fur & P. Casellas – « Involvement of Peripheral Benzodiazepine Receptors in the Protection of Hematopoietic Cells Against Oxygen Radical Damage », *Blood* **87** (1996), no. 8, p. 3170–3178.

[6] J. Chai, X. Tong, S. Chan & H. Shum – « Plenoptic sampling », *Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (2000), p. 307–318.

[7] B. Chaudhuri – « An Efficient Algorithm for Running Window Pel Gray Level Ranking 2-D Images », *Pattern Recognition Letters* **11** (1990), no. 2, p. 77–80.

[8] W. Chen, J. Bouguet, M. Chu & R. Grzeszczuk – « Light Field Mapping : Efficient Representation and Hardware Rendering of Surface Light Fields », *ACM Transactions on Graphics* **21** (2002), no. 3, p. 447–456.

[9] P. E. Debevec, C. J. Taylor & J. Malik – « Modeling and Rendering Architecture from Photographs : a Hybrid Geometry- and Image-Based Approach », dans *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques* (New York, NY, USA), ACM Press, 1996, p. 11–20.

[10] J.-D. Deschênes, P. Lambert, S. Perreault, N. Martel-Brisson, N. Zoso & P. Hébert – « A Cable-driven Parallel Mechanism for Capturing

Object Appearance from Multiple Viewpoints », dans *Proceedings of the 6th International Conference on 3-D Digital Imaging and Modeling*, 2007.

[11] A. Erol, G. Bebis, R. D. Boyle & M. Nicolescu – « Visual Hull Construction Using Adaptive Sampling », dans *Proceedings of the Seventh IEEE Workshops on Application of Computer Vision* (Washington, DC, USA), vol. 1, IEEE Computer Society, 2005, p. 234–241.

[12] C. H. Esteban & F. Schmitt – « Multi-Stereo 3D Object Reconstruction », *First International Symposium on 3D Data Processing Visualization and Transmission* (2002).

[13] M. Fischler & R. Bolles – « Random Sample Consensus : a Paradigm for Model Fitting with Applications to Image Analysis and Automated Cartography », *Communications of the ACM* **24** (1981), no. 6, p. 381–395.

[14] J.-S. Franco & E. Boyer – « Fusion of Multi-View Silhouette Cues Using a Space Occupancy Grid », dans *Proceedings of the Tenth IEEE International Conference on Computer Vision*, vol. 2, 2005.

[15] J. Gil & M. Werman – « Computing 2-D Min, Median, and Max Filters », *IEEE Transactions on Pattern Analysis and Machine Intelligence* **15** (1993), no. 5, p. 504–507.

[16] M. Goesele, B. Curless & S. Seitz – « Multi-View Stereo Revisited », dans *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, vol. 2, IEEE Computer Society Washington, DC, USA, 2006, p. 2402–2409.

[17] R. M. Haralick, D. Lee, K. Ottenburg & M. Nolle – « Analysis and Solutions of the Three Point Perspective Pose Estimation Problem », dans *Proceedings of the 1991 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1991, p. 592–598.

[18] R. Hartley & A. Zisserman – *Multiple View Geometry*, Cambridge University Press, 2000.

[19] J. Heikkilä – « Geometric Camera Calibration Using Circular Control Points », *IEEE Transactions on Pattern Analysis and Machine Intelligence* **22** (2000), no. 10, p. 1066–1077.

[20] B. K. P. Horn, H. M. Hilden & S. Negahdaripour – « Closed-Form Solution of Absolute Orientation Using Orthonormal Matrices », *Journal of the Optical Society of America A* **5** (1988), no. 7, p. 1127–1135.

[21] T. Huang, G. Yang & G. Tang – « A Fast Two-Dimensional Median Filtering Algorithm », *IEEE Transactions on Acoustics, Speech, and Signal Processing* **27** (1979), no. 1, p. 13–18.

[22] D. KNUTH – « Big Omicron and Big Omega and Big Theta », *ACM SIGACT News* **8** (1976), no. 2, p. 18–24.

[23] — , *The Art of Computer Programming Volume 3 : Sorting and Searching*, deuxième éd., Addison Wesley Longman Publishing Co., Inc. Redwood City, CA, USA, 1998.

[24] H. KÜCK, W. HEIDRICH & C. VOGELGSANG – « Shape from Contours and Multiple Stereo : A Hierarchical, Mesh-Based Approach », dans *Proceedings of the 1st Canadian Conference on Computer and Robot Vision*, 2004.

[25] P. LAMBERT, J.-D. DESCHÊNES & P. HÉBERT – « A Sampling Criterion for Optimizing a Surface Light Field », dans *Proceedings of the 6th International Conference on 3-D Digital Imaging and Modeling*, 2007.

[26] A. LAURENTINI – « The Visual Hull Concept for Silhouette-Based Image Understanding », *IEEE Transactions on Pattern Analysis and Machine Intelligence* **16** (1994), no. 2, p. 150–162.

[27] K. LEVENBERG – « A Method for the Solution of Certain Nonlinear Problems in Least Squares », *Quarterly of Applied Mathematics* **2** (1944), p. 164–168.

[28] W. LORENSEN & H. CLINE – « Marching Cubes : a High Resolution 3D Surface Construction Algorithm », dans *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, ACM Press New York, NY, USA, 1987, p. 163–169.

[29] M. LOURAKIS & A. ARGYROS – « The Design and Implementation of a Generic Sparse Bundle Adjustment Software Package Based on the Levenberg-Marquardt Algorithm », Tech. Report 340, Institute of Computer Science - FORTH, Héraklion, Crète, Grèce, 2004.

[30] C. LU, G. HAGER & E. MJOLSNESS – « Fast and Globally Convergent Pose Estimation from Video Images », *IEEE Transactions on Pattern Analysis and Machine Intelligence* **22** (2000), no. 6, p. 610–622.

[31] P. MARAGOS & R. SCHAFER – « Morphological Filters–Part II : Their Relations to Median, Order-Statistic, and Stack Filters », *IEEE Transactions on Acoustics, Speech, and Signal Processing* **35** (1987), no. 8, p. 1170–1184.

[32] D. MARQUARDT – « An Algorithm for Least-Squares Estimation of Nonlinear Parameters », *Journal of the Society for Industrial and Applied Mathematics* **11** (1963), no. 2, p. 431–441.

[33] W. MARTIN & J. AGGARWAL – « Volumetric Descriptions of Objects from Multiple Views », *IEEE Transactions on Pattern Analysis and Machine Intelligence* **5** (1983), p. 150–158.

[34] T. NELSON & D. PRETORIUS – « Three-Dimensional Ultrasound of Fetal Surface Features », *Ultrasound in Obstetrics and Gynecology* **2** (1992), no. 3, p. 166–174.

[35] J.-N. Ouellet & P. Hébert – « A Simple Operator for Very Precise Estimation of Ellipses », dans *Proceedings of the Fourth Canadian Conference on Computer and Robot Vision*, 2007.

[36] S. Perreault & P. Hébert – « Median Filtering in Constant Time », *IEEE Transactions on Image Processing* (2007).

[37] M. Potmesil – « Generating Octree Models of 3D Objects from their Silhouettes in a Sequence of Images », *Computer Vision, Graphics, and Image Processing* **40** (1987), no. 1, p. 1–29.

[38] S. Seitz, B. Curless, J. Diebel, D. Scharstein & R. Szeliski – « A Comparison and Evaluation of Multi-View Stereo Reconstruction Algorithms », *Proceedings of the 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition-Volume 1* (2006), p. 519–528.

[39] C. C. Slama – *Manual of Photogrammetry*, quatrième éd., American Society of Photogrammetry, 1980.

[40] J. Starck & A. Hilton – « Surface Capture for Performance-Based Animation », *IEEE Computer Graphics and Applications* **27** (2007), no. 3, p. 21–31.

[41] R. Szeliski – « Rapid Octree Construction from Image Sequences », *Computer Vision, Graphics, and Image Processing : Image Understanding* **58** (1993), no. 1, p. 23–32.

[42] J. Tukey – *Exploratory Data Analysis*, Addison-Wesley Menlo Park, CA, 1977.

[43] B. Weiss – « Fast Median and Bilateral Filtering », *ACM Transactions on Graphics* **25** (2006), no. 3, p. 519–526.