Suivi d'objets d'intérêt dans une séquence d'images : des points saillants aux mesures statistiques

Garcia Vincent

Université de Nice - Sophia Antipolis

ÉCOLE DOCTORALE STIC

Sciences et Technologies de l'Information et de la Communication

THÈSE

pour obtenir le titre de

Docteur en Sciences

de l'Université de Nice - Sophia Antipolis

Mention : Automatique, Traitement du Signal et des Images

présentée et soutenue par

Vincent GARCIA

Équipe d'accueil : CReATIVe - Laboratoire I3S

SUIVI D'OBJETS D'INTÉRÊT DANS UNE SÉQUENCE D'IMAGES : DES POINTS SAILLANTS AUX MESURES STATISTIQUES

Thèse dirigée par Michel BARLAUD et par Éric DEBREUVE

Soutenue publiquement le 11 décembre 2008 devant le jury composé de

Jenny BENOIS-PINEAU	Professeur des Universités (Bordeaux)	Présidente
Renaud KERIVEN	Professeur des Universités (Paris)	Rapporteur
Frank NIELSEN	Professeur à l'École Polytechnique (Paris)	Rapporteur
Luc ROBERT	CTO à Autodesk RealViz (Sophia Antipolis)	Examinateur
Michel BARLAUD	Professeur des Universités (Nice - Sophia Antipolis)	Directeur de thèse
Éric DEBREUVE	Chargé de recherche CNRS	Directeur de thèse

À ma mère bien trop tôt disparue

"With great power comes great responsibility."
Ben PARKER

TABLE DES MATIÈRES

Remerciements **xiii**

1 Introduction **1**
1.1 Contexte 1
1.2 Contributions 2

I Suivi d'objets dans une vidéo fondé sur les points saillants **7**

2 État de l'art des méthodes de suivi d'objets **9**
2.1 Généralités 9
2.2 Représentation et détection des objets 10
2.3 Suivi de points 11
2.4 Suivi de silhouettes 12
2.5 Suivi de noyaux 15

3 Description locale **17**
3.1 Introduction 17
3.2 Points et régions d'intérêt 18
3.2.1 Saillance visuelle 18
3.2.2 Moravec 20
3.2.3 Harris et Stephens 21
3.2.4 Lindeberg, LoG 24
3.2.5 Harris-Laplace 25
3.3 Vecteurs de description 27
3.3.1 Voisinage de pixels 29
3.3.2 Distribution 30
3.3.3 SIFT 31
3.4 Mesures de similarité 33
3.4.1 Corrélation 34
3.4.2 Distances 35
3.5 Conclusion 37

4 Suivi d'après les trajectoires de points d'intérêt 39
4.1 Introduction . . . 39
4.2 Mouvement global entre deux images . . . 40
4.2.1 Appariement de points d'intérêt . . . 40
4.2.2 Estimation du mouvement . . . 42
4.2.3 Résultats sur la séquence Carmap . . . 45
4.2.4 Résultats sur la séquence Arménie . . . 48
4.3 Mouvement global sur un GOP . . . 49
4.3.1 Construction et sélection des trajectoires . . . 50
4.3.2 Estimation du mouvement . . . 52
4.3.3 Résultats sur la séquence Carmap . . . 53
4.3.4 Résultats sur la séquence Arménie . . . 53
4.4 Estimation d'un mouvement local sur un GOP glissant . . . 54
4.4.1 Pondération spatio-temporelle . . . 58
4.4.2 Résultats sur la séquence Carmap . . . 63
4.4.3 Résultats sur la séquence Arménie . . . 65
4.5 Conclusion . . . 67

5 Suivi de la couronne de l'objet 69
5.1 Introduction . . . 69
5.2 Approche classique . . . 70
5.3 Masquage partiel du fond . . . 73
5.4 Estimation du mouvement . . . 77
5.5 Résultats . . . 80
5.6 Conclusion . . . 89

II Suivi d'objets dans une vidéo fondé sur des mesures statistiques 95

6 Suivi d'objets fondé sur des mesures statistiques 97
6.1 Vers des mesures statistiques pour le suivi . . . 97
6.2 Descripteur : couleur + géométrie . . . 98
6.3 Mesure de similarité : divergence de Kullback-Leibler . . . 100
6.3.1 Similarité entre distributions . . . 100
6.3.2 Définition . . . 100
6.3.3 Approximation de D_{KL} par la méthode des kPPV . . . 101
6.4 Algorithme . . . 102
6.5 Un frein : le coût calcul . . . 102

7 Recherche des k plus proches voisins 103
7.1 Problème . . . 103
7.2 Méthodes de recherche des kPPV . . . 104
7.2.1 Approche exhaustive . . . 104

7.2.2 Partitionnement de l'espace . . . 105
7.2.3 Locality sensitive hashing . . . 106
7.3 Résumé . . . 108

8 Implémentation GPU 111
8.1 Motivations . . . 111
8.2 NVIDIA CUDA . . . 112
8.2.1 Généralités sur l'API . . . 112
8.2.2 Vocabulaire . . . 113
8.2.3 Programmation sur GPU . . . 115
8.2.4 Mémoires . . . 117
8.2.4.1 Mémoire globale . . . 117
8.2.4.2 Mémoire partagée . . . 118
8.2.4.3 Mémoire texture . . . 119
8.3 Recherche des kPPV en CUDA . . . 120
8.3.1 Calcul des distances . . . 120
8.3.2 Tri des distances . . . 122
8.4 Résumé . . . 125

9 Expérimentations et application au suivi d'objets 127
9.1 Résultats sur données synthétiques . . . 127
9.1.1 Contexte théorique . . . 127
9.1.2 Spécifications techniques . . . 128
9.1.3 Temps de calcul . . . 128
9.1.3.1 Tableau général . . . 128
9.1.3.2 Influence de la dimension . . . 130
9.1.3.3 Influence du nombre de points . . . 130
9.1.3.4 Influence du paramètre k . . . 132
9.1.4 Répartition du temps de calcul . . . 132
9.1.4.1 Influence de l'implémentation . . . 135
9.1.4.2 Influence de la dimension . . . 135
9.1.4.3 Influence du nombre de points . . . 136
9.1.4.4 Influence du paramètre k . . . 136
9.2 Application au suivi d'objets . . . 137
9.2.1 Suivi d'objets . . . 137
9.2.2 Composantes testées . . . 138
9.2.3 Expérimentations sur la séquence Crew . . . 140
9.2.3.1 Vidéo et réglages . . . 140
9.2.3.2 Résultats qualitatifs . . . 140
9.2.3.3 GPU . . . 147
9.2.4 Expérimentations sur la séquence Poltergay . . . 147
9.2.4.1 Vidéo et réglages . . . 147
9.2.4.2 Séquence Poltergay . . . 148
9.2.4.3 GPU . . . 149

9.3 Conclusion 153

10 CUBLAS **157**
10.1 Introduction 157
10.2 Recherche des kPPV en CUBLAS 158
10.3 Experimentations 160
10.3.1 Influence de la dimension 160
10.3.2 Influence du nombre de points 161
10.4 Conclusion 162

III Conclusion générale et perspectives **165**

11 Conclusion générale **167**
11.1 Suivi d'après les trajectoires de points d'intérêt 167
11.1.1 Bilan des travaux présentés 167
11.1.2 Perspectives 168
11.2 Suivi de la couronne de l'objet 168
11.2.1 Bilan des travaux présentés 168
11.2.2 Perspectives 169
11.3 Suivi d'objets fondé sur des mesures statistiques 170
11.3.1 Bilan des travaux présentés 170
11.3.2 Perspectives 170

IV Annexes **173**

A Estimation de l'entropie **175**
A.1 Définition 175
A.2 Estimation 176
A.2.1 Histogramme 176
A.2.2 Méthode à noyau 177
A.2.3 k-ème plus proche voisin 178
A.3 Influence du nombre d'échantillons 180

B Influence du critère d'estimation de mouvement **183**

C Code source CUDA **189**

D La post-production cinématographique **191**
D.1 Cinématographie et production 191
D.2 Post-production visuelle 192
D.2.1 Matchmoving 193
D.2.2 Rotoscopie 194
D.2.3 Création et animation d'objets 3D 195

D.2.4 Compositing 199

E NVIDIA CUDA Zone **201**

Liste des figures **206**

Liste des tableaux **207**

Bibliographie **221**

Résumé **222**

REMERCIEMENTS

En premier lieu, je tiens à remercier les personnes qui m'ont fait l'honneur de participer à mon jury. Je remercie messieurs Renaud Keriven et Frank Nielsen pour le soin qu'ils ont porté à la lecture du manuscrit y apportant chacun un éclairage différent. Je remercie madame Jenny Benois-Pineau pour m'avoir fait l'honneur de présider ce jury. Je remercie monsieur Luc Robert pour avoir accepté de faire partie du jury et pour m'avoir accueilli au sein de la société RealViz pendant les trois premières années de ma thèse.

Je tiens également à remercier Messieurs Jean-Marc Fédou et Luc Pronzato, directeurs successifs du Laboratoire I3S de Sophia Antipolis, pour m'avoir accueilli au sein de leur laboratoire.

Je souhaite exprimer ma reconnaissance aux personnes qui m'ont dirigé au cours des ces années de thèse. Je remercie monsieur Michel Barlaud pour la qualité de son encadrement. Il m'a guidé sur le chemin de la thèse et j'ai pu bénéficier de ses conseils judicieux et de sa vision d'ensemble du domaine. Je remercie monsieur Éric Debreuve pour sa patience, ses conseils avisés, sa rigueur scientifique, et pour tout ce qu'il m'a appris. Je n'aurais jamais pu atteindre ce niveau d'excellence en programmation (Matlab) sans son aide, même si mes connaissances actuelles dans ce domaine sont à des années lumière des siennes ;-)

Je remercie Micheline Hagnéré pour toute l'aide qu'elle m'a offerte pendant ces quatre années. Je remercie également Guy et Lionel pour les mêmes raisons.

Je remercie ma famille pour m'avoir soutenu avant et tout au long de la thèse. Je remercie mes parents (Agnès et Rémy), mes nombreux frères et soeurs (Renaud, Paloma, Guillaume, Bertin et la petite Luna), ainsi qu'Anne-Marie et Louisette. Je remercie Céline (Ba)Nania pour m'avoir soutenu et supporté pendant ces longues années.

Ces quatre années de thèse n'auraient pas été les mêmes sans les doctorants, les stagiaires et les permanents de l'équipe CReATIVe et plus généralement du laboratoire. Les pauses café / thé ont réellement contribué à la bonne ambiance générale du laboratoire et, à titre personnel, m'ont aidé à faire face aux difficultés de la thèse. Ainsi, je remercie Leonardo [1], Marie-Andrée (alias Mago), Changy, V., G., Thomas, Muriel [2], les ritals (Marco, Vincenzo, Paolo, Sylvia [3]), Éric (alias Ewol), les Freds (Payan & Précioso [4]), Marie (alias Moger), Sandrine, Yasmine, Anis, Aymen, Akram, Sami, François, Wali, et Elena. Je remercie Marie, Lionel, André, Ronald, Laure, Cédric, Laurent, Benoît, Stéphane, David, Aline, Ariane, ainsi que tous ceux que j'ai oubliés.

Je finis cette section en remerciant le Docteur Sylvain Boltz, également connu sous le pseudonyme de Lasyl. Sylvain a été mon co-bureau tout au long de la thèse. Il m'a supporté et m'a initié aux joies du FPS ;-). Sylvain m'a tout appris par ses connaissances avancées dans les domaines du design, de la photographie, des techniques de survie (*e.g. Comment prendre une douche quand on a pas de serviette ?*), des sports extrêmes, et de la vie en général. Sylvain m'a accompagné lors de nombreux voyages (Graz, Boston, Santorini, etc.) et m'a également enseigné son art : *Comment manger et boire le plus possible sans se faire remarquer à un buffet qui ne nous est pas réservé*. Comme dirait un brésilien cher à notre coeur « Mex, t'es un mec bien ! ».

1. Beleza ?
2. Girl power
3. Linus Torvalds spiritual daughter
4. J'ai compris mais j'rigole pas...

- CHAPITRE 1 -

INTRODUCTION

§ 1.1 CONTEXTE

Le traitement d'images désigne une discipline des mathématiques appliquées qui étudie les images numériques et leur transformation dans le but d'améliorer leur qualité, de réduire leur coût de stockage ou d'en extraire une information sémantique et pertinente. Une image peut tout à fait être traitée comme un signal bidimensionnel dans le cas d'images en niveaux de gris, et comme un signal tridimensionnel dans le cas d'images couleur. C'est pourquoi le traitement d'images est souvent défini comme étant un sous-ensemble du traitement du signal. Les premiers travaux sur le traitement d'images portaient sur la compression et la transmission d'images numériques. Puis, d'autres problèmes sont apparus tels que la détection de primitives (détection de contours [Can86, Der87] et de points d'intérêt [Mor77, HS88, SB97, MS04]) particulièrement utilisées en vision par ordinateur, l'imagerie médicale [MdSR+07, PKPB07, PVDK08, CAI+08, CPMG+08, KCF+06], l'imagerie satellitaire [RJZ03, RJZ06] (voir figure 1.1), ou encore la post-production cinématographique. L'extension naturelle du traitement d'images est le traitement de vidéos, une vidéo étant simplement un ensemble ou une collection d'images. Depuis les années 1990, le traitement d'images est omniprésent. On le retrouve dans les appareils photographiques numériques, les téléphones portables, les télévisions numériques haute définition, les webcams, les lecteurs vidéo, etc.
Les algorithmes de traitement d'images et de vision par ordinateurs se classent en trois catégories : algorithmes de bas niveau, de niveau intermédiaire, et de haut niveau. Les algorithmes de bas niveau réalisent des opérations basiques sur les pixels (*e.g.* plusieurs pixels se déplacent dans une image). Les algorithmes de niveau intermédiaire incluent des calculs de plus haut niveau tel que le groupement de pixels (*e.g.* un ensemble de

pixels de même couleur se déplace selon un mouvement affine unique). Enfin, les algorithmes de haut niveau permettent de donner une information sémantique en analysant une image ou une vidéo (*e.g.* un véhicule est en mouvement). Dans cette thèse, nous étudions et proposons des algorithmes de bas niveau et de niveau intermédiaire. Les informations de haut niveau requises pour la mise en place de ces algorithmes (*e.g.* initialisation de la position d'un objet d'intérêt) sont en effet définies par un opérateur humain.
Dans ce manuscrit, nous nous attachons au problème du suivi d'objets dans une vidéo. Dans notre cas, cela consiste à localiser un objet d'intérêt en chaque image d'une vidéo donnée d'après une position initiale déterminée manuellement sur la première image de cette même vidéo. Les applications du suivi d'objets sont nombreuses : vidéo surveillance (analyse du trafic routier ou du mouvement d'une foule), compression vidéo, *inpainting*, *compositing*, préservation d'anonymat (voir figure 1.2), etc. En post-production cinématographique, le suivi d'objets est souvent appelé *rotoscopie* en mémoire du nom des premiers appareils permettant de suivre manuellement un objet en mouvement dans une vidéo (voir annexe D.2.2). La rotoscopie est l'une des techniques les plus utilisées pour les effets spéciaux et l'une des plus chères car elle est encore souvent effectuée image par image.

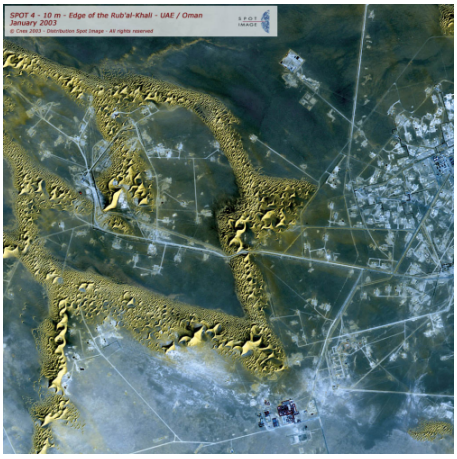
§ 1.2 CONTRIBUTIONS

Les principales contributions de cette thèse sont premièrement la proposition de nouvelles méthodes de suivi d'objets fondées sur l'utilisation de points saillants et/ou de mesures statistiques, et deuxièmement l'implémentation d'une méthode rapide d'estimation de mesures statistiques via la programmation GPU.

La première méthode de suivi d'objets proposée [GBDB06, GDB06, GDB07a, GDB07b, GDB07c] est fondée sur l'analyse de trajectoires de points d'intérêt. Un point d'intérêt est un point ayant un voisinage riche d'informations ce qui lui permet d'être détecté de manière précise et fiable dans une image ou un ensemble d'images (*e.g.* coins, jonctions, fins de ligne, etc. [Mor77, HS88, SB97, MS04]). Il est ainsi relativement simple de construire une trajectoire temporelle d'un point d'intérêt dans une vidéo. Pour cette méthode de suivi, notre contribution porte principalement sur l'analyse de trajectoires et l'utilisation de groupes d'images pour l'estimation robuste du mouvement d'un objet déterminé. La méthode proposée permet ainsi de suivre précisément un objet tout en gérant les occultations partielles et les déformations locales dudit objet.

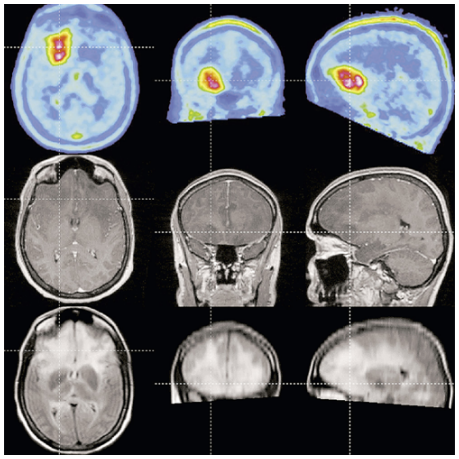
La seconde méthode de suivi d'objets proposée [GBDB07] repose sur le

(a) Compression d'images



(b) Imagerie satellitaire

(c) Imagerie spatiale



(d) Imagerie médicale

FIGURE 1.1 – *Traitement d'images et applications. La figure (a) illustre l'image Lena célèbre dans la communauté de traitement d'images car cette image a été utilisée à de très nombreuses reprises pour diverses applications telles que la détection de contours et la compression d'images. La figure (b) illustre la lisière du Rub'al-Khali (EAU / Oman) prise par le satellite SPOT 4 développé par le CNES. La figure (c) montre la constellation d'Orion où de jeunes étoiles sont en train de naître. Cette image a été prise par la NASA. Enfin, la figure (d) montre un exemple d'IRM (Imagerie par Résonance Magnétique) de cerveau humain particulièrement utilisée pour des applications médicales.*

FIGURE 1.2 – *Application du suivi d'objets à la préservation de l'anonymat dans une vidéo. Le visage de chaque personne à protéger est détouré manuellement sur la première image de la vidéo, puis ce visage est suivi sur les images suivantes. Photographie prise par Jean-Marc Luneau.*

suivi de la couronne de l'objet. Cette couronne est une bande de quelques pixels de large située sur le bord de l'objet d'intérêt. De par sa position, cette couronne contient des informations sur l'objet, des informations de contour, mais également des informations relatives au fond de l'image. Notre contribution porte premièrement sur l'utilisation de cette bande pour l'estimation du mouvement de l'objet. Cette méthode permet de suivre précisément des objets texturés ou des objets complètement uniformes. La seconde contribution porte sur l'utilisation de l'entropie de Shannon [Sha48, CT91] comme mesure de similarité. Cette mesure permet d'améliorer la robustesse du suivi aux données aberrantes (pixels du fond) et permet une plus grande souplesse dans le choix de la largeur de la couronne.

La troisième méthode présentée correspond aux travaux de Boltz *et al.* [BDB07, Bol08] sur l'utilisation conjointe d'une mesure statistique et de caractéristiques couleur-espace pour le suivi d'objets. Au cours du temps, un objet d'intérêt est susceptible de changer de forme, de taille, de couleur, etc. Ces variations peuvent rendre le suivi d'objets inefficace pour des approches locales. En revanche, les propriétés statistiques d'un objet (*e.g.* distribution des couleurs) varient généralement peu dans le temps. Boltz *et al.* présentent une méthode de suivi utilisant des composantes couleurs et des composantes géométriques et utilisant la divergence de Kullback-

Leibler [KL51, Kul59] comme mesure de similarité. Ces travaux furent précédemment proposés par Elgammal *et al.* [EDD03]. La contribution des auteurs porte sur l'utilisation de la recherche des k plus proches voisins (kPPV) pour l'estimation de la divergence de Kullback-Leibler en haute dimension. Le suivi proposé est précis et gère les occultations et les déformations géométriques.
Pour cette méthode, notre contribution porte sur l'implémentation rapide de la recherche des kPPV par la programmation parallèle sur GPU [1] [GDB08]. Nous montrons par différentes expérimentations que la programmation GPU peut grandement accélérer la recherche des kPPV en haute dimension. Les améliorations de notre travail se traduisent par une accélération du suivi d'objets, mais également par l'accélération d'autre algorithmes de traitement d'images (recherche d'images par le contenu).

1. GPU : *Graphics Processing Unit* signifiant *unité de calcul graphique*. Type de processeurs utilisés dans les cartes graphiques et spécialisés dans le calcul parallèle.

PREMIÈRE PARTIE

SUIVI D'OBJETS DANS UNE VIDÉO FONDÉ SUR LES POINTS SAILLANTS

- CHAPITRE 2 -

ÉTAT DE L'ART DES MÉTHODES DE SUIVI D'OBJETS

§ 2.1 GÉNÉRALITÉS

Le suivi d'objets est un problème fréquemment rencontré dans le domaine de la vision par ordinateur. L'augmentation constante de la puissance des ordinateurs, la diminution du coût des caméras et l'augmentation des besoins pour l'analyse de vidéos ont engendré un vif intérêt pour les algorithmes de suivi d'objets. L'analyse de vidéos se compose de 3 étapes : détection d'objets d'intérêt, suivi de ces objets, et analyse du mouvement de ces objets pour en déduire un comportement. Ainsi, le suivi d'objets permet de réaliser les tâches suivantes :

- Reconnaissance d'objets (ou de personnes) fondée sur l'analyse du mouvement (*e.g.* démarche d'une personne, analyse de trafic routier en temps réel)
- Surveillance automatique pour reconnaitre des activités suspectes ou inhabituelles (*e.g.* vidéo-surveillance [CBPG03, CBP05])
- Indexation de vidéo : annotation automatique et recherche de vidéos dans une base de données multimédia
- Interface homme-machine par l'analyse de posture ou le suivi du regard
- Navigation de véhicule : calcul de trajectoires et évitement d'obstacles

De manière très simple, le suivi d'objets peut se concevoir comme le problème d'estimation de la trajectoire d'un ou plusieurs objets dans le plan image. La difficulté du suivi d'objets dépend de plusieurs facteurs relatifs aux données ou à l'application :

- Perte d'informations causée par la projection d'un monde 3D sur une image 2D
- Présence de bruit dans les images
- Mouvements complexes
- Objets non rigides et/ou articulés
- Occultations partielles ou totales
- Objets de forme complexe
- Variation de l'illumination de la scène et des objets
- Nécessité d'un suivi en temps réel

De nombreuses approches de suivi d'objets ont été proposées. Outre l'algorithme lui-même, la différence entre ces méthodes réside en partie dans le choix de la représentation et de la forme des objets, des caractéristiques (composantes) de l'image utilisées, de la nature du mouvement estimé, etc. Ce choix dépend de l'application ainsi que de la vidéo traitée. Yilmaz *et al.* réalisent dans [YJS06] une étude des méthodes traitant du problème du suivi d'objets. Les auteurs y établissent une classification des méthodes de suivi dont les catégories sont :

1. Suivi de points
2. Suivi de silhouettes
3. Suivi de noyaux

§ 2.2 Représentation et détection des objets

La représentation des objets fait partie intégrante du processus de suivi. Un objet peut être représenté par un point (centre de l'objet [VRB01]) ou par un ensemble de points [SKMG04]. Cette représentation est adaptée aux objets occupant une petite partie de l'image.
Un objet peut également être représenté par une forme simple telle qu'un rectangle, une ellipse, etc. [CRM03]. Cette représentation est adaptée au suivi d'objets rigides (*e.g.* véhicules) mais peut également convenir pour des objets non rigides.
Un contour (ou une silhouette) peut permettre de représenter un objet. Cette représentation est adaptée au suivi d'objets non rigides ayant une forme complexe [YLS04].
Enfin, la représentation d'un objet par un squelette [AA01] ou par un modèle articulé (régions jointes par des articulations) est très adaptée au suivi d'objets articulés (*e.g.* suivi de personnes).

La première étape lors du processus de suivi d'objets consiste en la détection des objets à suivre. Cette détection peut intervenir au début de la vidéo ou à tout moment si des objets peuvent potentiellement entrer dans

le champ de la caméra. La méthode la plus simple pour cette détection est la mise à contribution d'un opérateur humain. Cet opérateur se charge de localiser le ou les objets à suivre.
La méthode de soustraction du fond calcule la différence entre deux images consécutives [JN79, WADP97] laissant apparaître les objets en mouvement. La détection d'objets peut également être réalisée en détectant les points d'intérêt de l'image [Mor77, HS88, SMB00]. Cette méthode est cependant peu adaptée si le fond de la vidéo est texturé. Dans un tel cas, il est probable que de nombreux points d'intérêt soient détectés faussant de fait la détection d'objets.

§ 2.3 SUIVI DE POINTS

Considérons qu'un objet est représenté par un ensemble de points [SKMG04]. Le problème du suivi d'objets peut être formulé comme un problème de mise en correspondance de points. La détection de points peut cependant être gênée par plusieurs facteurs tels que la présence de bruit dans les images, les occultations partielles ou totales, les erreurs de détection de points, etc.

Sethi et Jain [SJ87] résolvent le problème de mise en correspondance en utilisant une approche *gloutonne* (de l'anglais *greedy*) fondée sur l'ajout de contraintes de proximité et de rigidité des objets. Les appariements de points de deux images voisines, initialisés par une recherche des plus proches voisins, sont échangés en minimisant une fonctionnelle. Cette méthode ne permet pas de prendre en compte les occultations.

Suite aux travaux de Sethi et Jain [SJ87], Veenman *et al.* [VRB01] introduisent une contrainte de mouvement dans leur méthode de mise en correspondance. Cette contrainte est parfaitement adaptée au suivi de points appartenant au même objet et améliore la qualité du suivi. A contrario, cette contrainte n'est pas adaptée si les points appartiennent à différents objets dont le mouvement est différent. Le suivi est dans un premier temps réalisé sans contrainte sur les deux premières images de la vidéo. La contrainte de mouvement est utilisée dès la troisième image. Cette approche permet de gérer les occultations et les erreurs de détection de points.

Schmid *et al.* [Sch96, SM97, SMB98] introduisent une contrainte géométrique au processus d'appariement de points. Deux points voisins doivent être appariés à deux points également voisins dans l'image suivante. Les appariements ne respectant pas cette contrainte sont simplement ignorés pour la suite du processus. Cette contrainte géométrique a pour but de diminuer le nombre d'appariements incorrects. L'inconvénient majeur de

cette méthode est le fait que des appariements corrects sont également éliminés.

De manière similaire aux travaux de Schmid *et al.* [Sch96, SM97, SMB98], Trichet et Merialdo [TM07] ajoutent une contrainte géométrique (triangulation de Delaunay) au processus d'appariement. Les auteurs réalisent la détection des points dans le voisinage des objets détectés à l'image précédente. La triangulation de Delaunay de chaque ensemble de points doit être respectée entre deux images consécutives. Cette approche permet de diminuer le nombre d'appariements incorrects de points. Cependant, la contrainte étant plus forte que pour les travaux de Schmid *et al.*, un grand nombre d'appariements corrects sont éliminés.

§ 2.4 SUIVI DE SILHOUETTES

La représentation d'un objet par une forme simple telle qu'un rectangle ou une ellipse peut être mal adaptée si l'objet en question est de forme très complexe (*e.g.* main, corps humain, animal, arbre, etc.). La représentation d'un tel objet par une silhouette permet de tenir compte précisément de la forme de l'objet. Le but des méthodes de suivi fondées sur l'utilisation de silhouettes est d'estimer la silhouette des objets d'intérêt pour chaque image de la vidéo. Cette approche est également connue sous le nom de segmentation d'images. La segmentation d'images est une opération de traitement d'images qui a pour but de rassembler des pixels entre eux suivant des critères prédéfinis. Les pixels sont ainsi regroupés en régions qui constituent une partition de l'image. Il s'agit en général de séparer les objets du fond. Si le nombre de classes est égal à deux, elle est appelée aussi binarisation. Si l'homme sait naturellement séparer des objets dans une image, c'est grâce à des connaissances de haut niveau fondées sur la compréhension des objets et de la scène, compréhension liée à l'apprentissage. Mettre au point des algorithmes de segmentation utilisant une information sémantique pour chaque région de l'image est encore un des thèmes de recherche les plus courants en traitement d'images.

Les approches fondées sur l'utilisation de régions reposent sur une caractérisation de tout ou partie de l'image à segmenter. Dans [HS92, SSWC88], les auteurs proposent des méthodes qui effectuent un traitement global sur l'image comme par exemple un seuillage sur l'intensité de l'image afin de détecter les différents objets de la scène. Un seuil optimal peut être déterminé en étudiant l'histogramme de l'intensité pour repérer les différents modes de la distribution de l'intensité. Mais cette méthode ne fonctionne que si les objets ont une intensité homogène et, de plus, qui soit différente de celle du fond.

Les méthodes dites de division/fusion (*split and merge* en anglais) [HP77, NN04] consistent dans un premier temps à découper une image en petites régions. Ensuite, les régions adjacentes similaires au sens d'un critère d'homogénéité sont fusionnées. La méthode peut également consister à découper une image de façon itérative tant que les régions ne sont pas suffisamment homogènes. L'image segmentée est généralement représentée par un arbre [VM07].

Les méthodes de classification (plus souvent appelées *clustering* en anglais) consistent à regrouper en sous-ensembles les pixels qui possèdent des caractéristiques proches. Citons par exemple les méthodes de k-means [Har75, McQ67] qui consistent à séparer les pixels en k groupes par la minimisation de la distance entre un pixel donné et le représentant du groupe auquel il appartient. Ces méthodes fournissent une partition de l'image mais ne permettent pas de distinguer les classes appartenant à l'objet et les classes appartenant au fond.

Enfin, la dernière méthode de segmentation que nous présentons est celle des contours actifs [KWT87, DT79, OS88, Set99, CRD07]. Cette méthode consiste à faire évoluer un contour initial vers l'objet d'intérêt en calculant une vitesse d'évolution (voir figure 2.1), la direction d'évolution étant généralement donnée par la normale au contour. Nous devons pour cela déterminer une caractéristique qui permet de différencier l'objet du fond. La caractéristique, généralement appelée descripteur (de région ou de contour), peut être présente dans l'objet et dans le fond : par exemple si l'on cherche des régions de mouvement homogène (le mouvement du fond et de l'objet sont différents mais la caractéristique "mouvement homogène" est la même). Ces caractéristiques peuvent être multiples. Il peut s'agir de la couleur, de textures, d'une forme précise ou bien d'une combinaison de plusieurs caractéristiques. Un critère est élaboré à partir de ces caractéristiques. La minimisation de ce critère permet de faire converger le contour vers l'objet d'intérêt. Les méthodes de contours actifs peuvent globalement être classées en deux catégories distinctes : les méthodes fondées sur la détection de contours et les méthodes fondées la statistique des régions.
Les méthodes fondées sur la détection de contours consistent à faire évoluer un contour jusqu'à ce que ce dernier atteigne une forte valeur de gradient [CKS94, KKO$^+$95, XP98]. Bien que ces modèles aient de nombreux avantages (*e.g.* segmentation de régions non homogènes), ils ont un inconvénient important qui les rend inefficaces sur des images bruitées [CRD07], ou quand le contour n'est pas entièrement à l'intérieur ou à l'extérieur de la région à segmenter. De plus, ces modèles sont uniquement capables de segmenter des régions qui ont des contours nets ce qui peut conduire à une « fuite » du contour actif si les contours de l'objet sont flous. Dans [XP98],

les auteurs proposent de résoudre le problème d'initialisation en créant un champ de vecteurs attirant le contour actif vers les forts gradients de l'image. Malgré cette amélioration, la sensibilité au bruit n'est pas résolue.
Les approches régions consistent à faire évoluer le contour dans le but de séparer significativement les statistiques des régions qu'il délimite [CV01, VC02, TYW01, PD02, CRD07]. Par exemple, séparer les régions claires des régions sombres, ou alors extraire des régions qui ont une certaine distribution [KFY+02]. Ces modèles prennent en considération la région entière et sont ainsi plus robustes au bruit et à l'initialisation par rapport aux approches contours. Cependant, elles sont inefficaces pour la segmentation d'images dont les propriétés statistiques varient sur la région.
Jehan-Besson *et al.* [JBBA03, ABFJB03] proposent une méthode de segmentation d'images par contours actifs dans le cadre d'une approche variationnelle. L'équation d'évolution du contour actif est déduite de la dérivation d'un critère caractérisant l'objet d'intérêt, la dérivation effectuée à l'aide des gradients de forme. Gastaud *et al.* [GBA04] utilisent un a priori géométrique sans contrainte paramétrique qui minimise la distance du contour actif à un contour de référence. Ce descripteur a été appliqué à la déformation de courbes (*warping*).
Pour la segmentation vidéo, le fait de segmenter chaque image indépendamment donne l'impression d'un contour oscillant au cours du temps autour de la position réelle du contour de l'objet. Cet effet, appelé *flickering* en anglais, peut être diminué en considérant la segmentation comme un problème tridimensionnel (2D+t) plutôt que plusieurs problèmes bidimensionnels [DCM06]. Ceci a pour effet d'augmenter la cohérence spatio-temporelle de la segmentation. Boltz *et al.* [BHD+08] proposent une méthode de segmentation de vidéos où le critère à minimiser est une entropie jointe entre une information d'apparence (couleur) et de mouvement (résiduel de compensation).

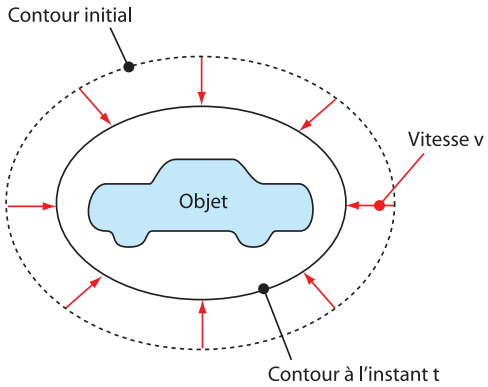


FIGURE 2.1 – *Évolution d'un contour actif vers l'objet d'intérêt*

§ 2.5 SUIVI DE NOYAUX

Le suivi de noyaux consiste à suivre un objet représenté par une forme géométrique basique telle qu'un rectangle ou une ellipse. Le mouvement estimé est généralement paramétrique (translation, rotation, affine, etc.).

L'approche la plus évidente pour suivre un objet repose sur l'utilisation d'un gabarit (en anglais *template*) [Bir98, SBW02]. En effet, si l'objet à suivre est de forme connue (*e.g.* une voiture), il est relativement simple de trouver la partie de l'image la plus similaire au gabarit considéré. Pour ce faire, la recherche est réalisée de manière exhaustive sur tout ou partie de l'image. Les informations utilisées sont l'intensité ou la couleur. Le gradient de l'image est également utilisé pour sa robustesse aux variations d'illumination. L'inconvénient majeur de cette méthode est la lenteur de la recherche exhaustive.

Une façon simple de suivre un objet est de minimiser une fonctionnelle dépendant de l'objet initial et d'un objet candidat. Cette fonctionnelle peut simplement être fondée sur la différence pixel à pixel entre deux objets (*e.g.* fonctions SAD pour *Sum of Absolute Differences* et SSD pour *Sum of Square Differences*). La minimisation de la fonctionnelle peut être assurée par une méthode classique dite de block-matching [KLH+81, ZM00, ZLC02]. Cependant, ce suivi est très sensible aux variations géométriques de l'objet survenant au cours du temps. Le suivi a tendance à devenir incorrect au bout de quelques images.

Comaniciu *et al.* [CM99a, CM99b, CRM00, CM02, CRM03] utilisent un histogramme de couleurs pondéré pour représenter l'objet à suivre. La maximisation du coefficient de Bhattacharyya permet de détecter dans chaque image et par un processus itératif l'objet d'intérêt. Plutôt que d'utiliser une approche exhaustive, les auteurs utilisent la méthode du *mean-shift* [FH75]. La pondération de l'histogramme donne plus d'importance aux pixels proches du centre et inversement. Les approches utilisant des distributions (ou des histogrammes) sont robustes aux variations géométriques. En effet, contrairement à la géométrie, les statistiques des objets varient peu dans le temps. Cette robustesse est cependant souvent obtenue en dépit de la précision du suivi. L'avantage de la méthode est sa rapidité liée à l'utilisation du mean-shift.

Elgammal *et al.* [EDD03] utilisent des histogrammes contenant des informations de couleurs et des informations géométriques. La similarité entre objets (*i.e.* entre histogrammes) est évaluée en utilisant la divergence de Kullback-Leibler [KL51, Kul59]. Le suivi résultant de cette méthode est pré-

cis tout en étant robuste aux variations géométriques. L'inconvénient de la méthode réside dans l'estimation de la divergence à proprement parler. En effet, l'estimation de mesures statistiques est complexe en haute dimension. Dans ce contexte, Boltz *et al.* [BDB07, Bol08] améliorent l'approche de Elgammal en proposant d'utiliser la recherche des plus proches voisins [Wol06] pour l'estimation de la divergence de Kullback-Leibler.

- CHAPITRE 3 -

DESCRIPTION LOCALE

§ 3.1 INTRODUCTION

Les méthodes de suivi proposées dans la première partie de ce manuscrit utilisent des outils classiques de description locale. Cette description permet de concentrer une étude à des régions de l'image où le contenu informatif est important. Dans ce chapitre, nous présentons les méthodes classiques de description locale ce qui permettra par la suite de justifier nos choix.
Le système visuel humain (SVH) permet de détecter des objets tridimensionnels et d'estimer leur mouvement dans l'espace en une fraction de seconde. Plusieurs travaux ont montré que le cerveau était capable de détecter des régions présentant un intérêt particulier en 200 ms dans le but de focaliser l'attention sur ces régions. Les primitives détectées sont par exemple des coins ou des contours. C'est à partir de ces primitives et des informations contenues dans leur voisinage que, selon certains spécialistes, le cerveau détecte les objets et en estime le mouvement. Ceci représente les fondements mêmes du concept de description locale.
La description d'une image consiste à décrire celle-ci par un certain nombre d'informations extraites de l'image (couleur dominante, structure globale de l'image, *etc.*). L'idée sous-jacente est de pouvoir comparer l'image à d'autres images dans le but d'estimer leur similarité. Les applications sont nombreuses et on peut citer par exemple la détection des copies [Ton06], l'indexation d'images et la recherche d'images et de vidéos par le contenu [CDBPD07, DDBP06, Sch96, ADPB08, PADB08, SARK08, SEAK08b, SEAK08a], le *Mosaicing* [NY06, Nie00], ou encore le suivi d'objets dans une vidéo [GDB06, GDB07b, GDB07c, GBDB07]. La description locale des images se fonde sur les mécanismes biologiques évoqués plus haut et se fait en deux étapes :

1. Détection de points ou de régions d'intérêt (primitives).

2. Construction d'un vecteur de description pour chaque point d'intérêt. Un vecteur de description est un ensemble de valeurs construites à partir des informations du voisinage et permettant de décrire ce point et de le différencier par rapport aux autres points.

La mesure de similarité permettant de comparer deux vecteurs de description devrait faire partie intégrante du système de description locale choisi. En effet, ces trois notions (détection des points/régions d'intérêt, vecteur de description, mesure de similarité) sont intimement liées. Une faiblesse de l'une d'entre elles rend la description locale de l'image inefficace car inapte à jouer son rôle discriminant.
La figure 3.1 présente un exemple d'appariement d'images utilisant la description locale. Les points d'intérêts sont détectés par la méthode DoG (pour *Difference of Gaussians*) [CP84, Lin93], les vecteurs de description sont construits par la méthode SIFT [Low99, Low04], et la mesure de similarité est la norme Euclidienne.

FIGURE 3.1 – *Appariement d'images par appariement de points d'intérêt. Les points d'intérêts sont détectés par la méthode DoG (Difference of Gaussians), les vecteurs de description sont construits avec SIFT, et la mesure de similarité est la norme 2. L'appariement des objets est correct malgré le changement d'orientation et d'échelle de l'objet à retrouver.*

§ 3.2 POINTS ET RÉGIONS D'INTÉRÊT

3.2.1 Saillance visuelle

Pendant de nombreuses années, les chercheurs ont tenté de comprendre le fonctionnement du système visuel humain dans l'analyse d'images. Un important résultat montre qu'un certain nombre de primitives visuelles sont détectées extrêmement rapidement et précisément par un système de

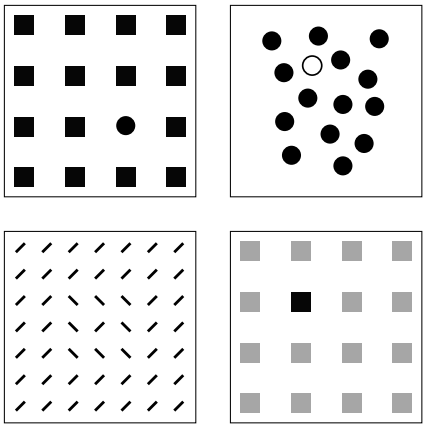


FIGURE 3.2 – *Exemples de saillance visuelle. Les primitives concernées sont (respectivement de gauche à droite et de haut en bas) la courbure, la couleur, l'orientation de lignes, et le changement d'intensité.*

bas niveau. Ces primitives étaient initialement appelées pré-attentives dans la mesure où l'attention est concentrée sur ces zones. Le but est en effet d'orienter le regard, l'attention, vers les régions pertinentes du champ perceptif dans le cas de l'apparition d'un danger, d'une menace. Le stade pré-attentif (période pendant laquelle les primitives sont extraites) a une durée inférieure à 200 ou 250 ms. Le mouvement des yeux met 200 ms à être initié. De nombreux travaux [Tre85] ont tenté de déterminer les primitives visuelles extraites pendant le stade pré-attentif. Nous pouvons par exemple citer l'orientation de lignes, la taille, la couleur, la courbure, l'intensité, la fermeture, la densité, les intersections, les coins, l'orientation 3D, la direction et la vitesse du mouvement, etc. La saillance visuelle correspond à la détection de ces primitives. La figure 3.2 illustre plusieurs exemples de saillance visuelle.
La saillance visuelle a été utilisée, au moins implicitement, dans de très nombreux travaux de vision par ordinateur. Les points d'intérêt en sont un parfait exemple. Ces points sont également appelés coins, points caractéristiques ou encore points saillants. Nous verrons par la suite que nous utiliserons également la notion de région ou de *blob* plutôt que celle de point selon la méthode de détection utilisée. Les points d'intérêt sont généralement utilisés pour mettre en correspondance deux images. Ces points contiennent l'information sur laquelle doit être portée l'attention. L'utilisation des points d'intérêt permet de simplifier le problème de mise en correspondance en se restreignant aux points d'intérêt. La mise en correspondance de deux images correspond simplement à l'appariement de deux nuages de points.
Historiquement, la détection de points d'intérêt a débuté avec la détection

de coins ce qui explique l'appellation précédente. Une définition simple possible serait de dire que les points d'intérêts correspondent à des doubles discontinuités de la fonction d'intensité dans une image. Celles-ci peuvent être provoquées, comme pour les contours, par des discontinuités de la fonction de réflectance ou des discontinuités de profondeur. Ce sont par exemple les coins, les jonctions en T ou les points de fortes variations de texture. La notion de point d'intérêt a évolué et a maintenant une définition plus générale. Un point d'intérêt est un point dans une image qui se caractérise par les propriétés suivantes :

- il est clairement défini (au sens mathématique),
- il a une position précisément définie dans l'espace image,
- la structure du voisinage local du point est riche en terme de contenu informatif,
- il est stable en présence de perturbations locales et globales dans l'espace image, incluant les déformations relatives au changement de perspective (transformation affine, changement d'échelle, rotation, et translation) et les variations d'illumination/luminosité,
- un point d'intérêt devrait également inclure un attribut d'échelle.

Dans la suite, nous présentons les principaux travaux sur la détection de points et de régions d'intérêt. Nous utiliserons l'ordre chronologique de manière à donner un aperçu des améliorations qui sont apparues au cours des années.

3.2.2 Moravec

Le détecteur développé par Hans P. Moravec [Mor77, Mor80] en 1977 est l'un des plus anciens algorithmes de détection de points d'intérêt. Moravec a défini le concept de points d'intérêt comme étant une région distinctive des images et a conclu que ces points pouvaient être utilisés pour apparier des régions dans des images consécutives. Ce traitement de bas niveau lui permettait de déterminer l'existence et la localisation d'objets dans l'environnement de déplacement de son véhicule.
Moravec définit un point d'intérêt comme étant un point où il y a une large variation de l'intensité dans certaines directions. Son détecteur est connu comme étant un détecteur de coins. L'auteur propose d'utiliser la fonction d'auto-corrélation (corrélation du signal avec lui-même décalé spatialement) pour détecter des variations de la fonction d'intensité dans un voisinage carré W (typiquement 3×3, 5×5, ou 7×7 pixels). Même si le terme auto-corrélation est clairement indiqué, la fonction utilisée pour détecter les coins est la somme du carré de la différence du signal avec lui même décalé spatialement. Ainsi, étant donné un pixel (x, y) de l'image et un décalage spatial (δ_x, δ_y), la fonction f calculée en (x, y) est définie par :

$$f(x, y; \delta_x, \delta_y) = \sum_W (I(x_k, y_k) - I(x_k + \delta_x, y_k + \delta_y))^2$$

où (x_k, y_k) sont les pixels du voisinage carré W centrée en (x, y) et I la fonction intensité. Le décalage spatial (δ_x, δ_y) est à valeur dans $[-1, 1]^2 \setminus \{0, 0\}$ (8-voisinage). L'auteur définit une fonction C à partir de laquelle les points d'intérêt vont être détectés. La valeur de la fonction en chaque pixel correspond à la valeur minimale de la fonction $f(x, y; \delta_x, \delta_y)$ pour les 8 décalages possibles :

$$C(x, y) = \min_{(\delta_x, \delta_y) \in [-1,1]^2 \setminus \{0,0\}} f(x, y; \delta_x, \delta_y)$$

La détection des points d'intérêt est simplement réalisée en exhibant les maxima locaux de C supérieurs à un seuil fixé. Le détecteur de Moravec a une faible complexité ce qui le rend intéressant pour des applications nécessitant une implémentation temps-réel. Cependant, du fait des problèmes exposés dans la section 3.2.3, ce détecteur est généralement considéré comme obsolète.

3.2.3 Harris et Stephens

Le détecteur de Moravec souffre de nombreuses limitations ce qui lui vaut de fonctionner dans un contexte limité. Harris et Stephen ont identifié certaines de ces limitations et, en les corrigeant, ont proposé en 1988 un détecteur de coins connu sous le nom de détecteur de Harris [HS88].
La réponse du détecteur de Moravec est anisotropique du fait du caractère discret de l'ensemble des décalages considérés (tous les 45 degrés). Cet aspect peut être résolu en considérant le développement de Taylor de la fonction intensité I au voisinage du pixel (x, y). Étant donné un décalage (δ_x, δ_y) et un pixel (x, y), la fonction $f(x, y; \delta_x, \delta_y)$ est définie par :

$$f(x, y; \delta_x, \delta_y) = \sum_W (I(x_k, y_k) - I(x_k + \delta_x, y_k + \delta_y))^2 \tag{3.1}$$

où (x_k, y_k) sont les points dans la fenêtre W centrée en (x, y) et I la fonction intensité. En considérant la fonction w définie par

$$w(x, y) = \begin{cases} 1 & \text{si} (x, y) \in W \\ 0 & \text{sinon} \end{cases}$$

la fonction $f(x, y; \delta_x, \delta_y)$ se récrit comme une somme d'un produit de convolution sur l'image entière :

$$f(x, y; \delta_x, \delta_y) = \sum_{x,y} w(x_k, y_k) * (I(x_k, y_k) - I(x_k + \delta_x, y_k + \delta_y))^2 \tag{3.2}$$

Considérons maintenant le développement de Taylor d'ordre 1 de l'image :

$$I(x + \delta_x, y + \delta_y) \approx I(x, y) + \begin{bmatrix} \frac{\partial I}{\partial x}(x, y) & \frac{\partial I}{\partial y}(x, y) \end{bmatrix} . \begin{bmatrix} \delta_x \\ \delta_y \end{bmatrix} \tag{3.3}$$

où $\frac{\partial I}{\partial x}$ et $\frac{\partial I}{\partial y}$ représentent les dérivées partielles de la fonction intensité relativement aux axes des abscisses et des ordonnées. La dérivation est approximée en utilisant une méthode par différence finie :

$$\frac{\partial I}{\partial x}(x,y) \approx \frac{I(x+1,y) - I(x-1,y)}{2} \tag{3.4}$$

$$\frac{\partial I}{\partial y}(x,y) \approx \frac{I(x,y+1) - I(x,y-1)}{2} \tag{3.5}$$

En substituant (3.3) à (3.1), nous obtenons l'expression suivante :

$$\begin{aligned} f(x,y;\delta_x,\delta_y) &\approx \sum_{x,y} w(x_k,y_k) * \left(\begin{bmatrix} \frac{\partial I}{\partial x}(x_k,y_k) & \frac{\partial I}{\partial y}(x_k,y_k) \end{bmatrix} . \begin{bmatrix} \delta_x \\ \delta_y \end{bmatrix} \right)^2 \\ &= \begin{bmatrix} \delta_x & \delta_y \end{bmatrix} A(x,y) \begin{bmatrix} \delta_x \\ \delta_y \end{bmatrix} \end{aligned}$$

où

$$A(x,y) = \begin{bmatrix} \left(w * \left(\frac{\partial I}{\partial x}\right)^2\right)(x,y) & \left(w * \left(\frac{\partial I}{\partial x}.\frac{\partial I}{\partial y}\right)\right)(x,y) \\ \left(w * \left(\frac{\partial I}{\partial x}.\frac{\partial I}{\partial y}\right)\right)(x,y) & \left(w * \left(\frac{\partial I}{\partial y}\right)^2\right)(x,y) \end{bmatrix} \tag{3.6}$$

La matrice A capture la structure du voisinage local des pixels de l'image d'où son nom de « matrice de structure ».
La seconde limitation provient de la fonction w. En effet, w étant une fonction binaire, la réponse de la fonction f est bruitée. Harris et Stephens ont alors proposé de remplacer w par une fonction Gaussienne

$$w(x,y) = G_\sigma(x,y) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3.7}$$

où σ (déviation standard = racine carrée de la variance) est un paramètre généralement fixé à $\sigma = 2$. La matrice A s'exprime alors comme suit :

$$A(x,y) = \begin{bmatrix} \left(G_\sigma * \left(\frac{\partial I}{\partial x}\right)^2\right)(x,y) & \left(G_\sigma * \left(\frac{\partial I}{\partial x}.\frac{\partial I}{\partial y}\right)\right)(x,y) \\ \left(G_\sigma * \left(\frac{\partial I}{\partial x}.\frac{\partial I}{\partial y}\right)\right)(x,y) & \left(G_\sigma * \left(\frac{\partial I}{\partial y}\right)^2\right)(x,y) \end{bmatrix} \tag{3.8}$$

Enfin, le détecteur de Moravec répond de manière trop forte aux contours car seul le minimum de f est pris en compte en chaque pixel. Les auteurs ont proposé d'étudier les valeurs propres (λ_1 et λ_2) de la matrice A. Ces deux valeurs propres sont proportionnelles aux courbures principales de la fonction f et forment une description de A invariante aux rotations. Il y a trois cas à considérer :

- Si les deux courbures (λ_1 et λ_2) sont faibles, la fonction f est alors approximativement constante ce qui indique que la région de l'image considérée soit homogène.
- Si une des courbures est forte alors que l'autre est faible, cela indique la présence d'un contour dans l'image.
- Enfin, si les deux courbures sont fortes, un coin est détecté.

Harris et Stephens ont proposé d'éviter le calcul fastidieux des valeurs propres en étudiant le déterminant et la trace de la matrice A. Les auteurs définissent alors une fonction C définie en chaque pixel (x, y) de l'image et à partir de laquelle les coins et les contours de l'image sont détectés :

$$C(x,y) = \lambda_1\lambda_2 - \kappa(\lambda_1+\lambda_2)^2 \quad (3.9)$$
$$= \det(A(x,y)) - \kappa.\text{trace}^2(A(x,y)) \quad (3.10)$$

où $\kappa \in [0.04, 0.15]$. Les coins sont détectés en exhibant les maxima locaux de C supérieurs à un seuil donné.

FIGURE 3.3 – *Détection de points d'intérêts par la méthode de Harris et Stephens sur l'image « vieille porte » prise par la photographe Aleksandra Radonic'.*

Le détecteur de Harris-Stephens, illustré sur la figure 3.3, est sans doute le détecteur de points d'intérêt le plus connu et le plus utilisé en traitement d'images. De nombreuses implémentations de ce détecteur existent car cinq paramètres doivent être choisis lors de son utilisation : le filtre dérivatif, le filtre Gaussien, le paramètre κ, le voisinage de l'extraction des maxima locaux, et le seuillage final. Son succès provient premièrement de sa facilité d'implémentation, et deuxièmement de la qualité des résultats obtenus. En effet, de récents travaux [SMB98, SMB00] prouvent que le détecteur de Harris donne les meilleurs résultats. De nombreux travaux sont venus amélio-

rer l'approche initiale. Ainsi, Schmid *et al.* proposent dans [SMB00] de calculer les dérivées partielles en convoluant l'image I par la dérivée d'une Gaussienne 1D. L'implémentation récursive du filtre Gaussien [Der93] permet une détection rapide. Montesinos *et al.* [MGD98] proposent de généraliser l'approche de Harris et Stephens en l'adaptant aux images couleur. La matrice de structure est alors constituée de la somme des dérivées partielles des composantes rouge, verte et bleue.

3.2.4 Lindeberg, LoG

Schmid *et al.* [SMB98, SMB00] ont montré que le détecteur de Harris a d'excellentes performances en comparaison d'autres détecteurs (au sens des critères de rappel et de précision [Sch96]). Cependant, comme tous les autres détecteurs coins, les performances du détecteur de Harris chutent lors de l'apparition d'un changement d'échelle entre deux images.
Lindeberg proposa dans [Lin98] en 1998 son détecteur de points d'intérêt multi-échelles nommé LoG (pour *Laplacian of the Gaussian*) ; les points détectés sont ainsi invariants aux changements d'échelle. On parle de détection de *blobs* plutôt que de détection de points car la détection exhibe des régions d'intérêt et non des points d'intérêt comme nous pouvons le voir sur la figure 3.4. Pour ce faire, Lindeberg décrit l'image dans l'espace-échelle Gaussien et utilise l'opérateur Laplacien.
Le Laplacien est un opérateur différentiel scalaire d'ordre 2 égal à la somme de toutes les deuxièmes dérivées partielles non mixtes d'une variable dépendante. En coordonnées cartésiennes dans $\mathbb{R}^n$, le Laplacien est défini par

$$\Delta f(x_1, ..., x_n) = \sum_{k=1}^{n} \frac{\partial^2 f}{\partial x_k^2}(x_1, ..., x_n). \tag{3.11}$$

Le Laplacien d'une image permet de mettre en évidence ses changements rapides d'intensité, ce qui en fait un opérateur particulièrement adapté à la détection de contours en étudiant les zéros de la fonction. La représentation de l'image I à l'échelle σ, notée L_σ, correspond à la convolution de I par une fonction Gaussienne bi-dimensionnelle G_σ d'écart type σ :

$$L_\sigma(x, y) = (G_\sigma * I)(x, y) \tag{3.12}$$

avec

$$G_\sigma(x, y) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2+y^2}{2\sigma^2}} \tag{3.13}$$

Par définition, le Laplacien de $\Delta L_\sigma(x, y)$ est égal à la somme de ses deux dérivées partielles secondes non mixtes. Cependant, il peut être directement calculé à partir de la fonction intensité I par convolution avec l'opérateur

LoG_σ :

$$\Delta L_\sigma(x,y) = \frac{\partial^2}{\partial x^2} L_\sigma(x,y) + \frac{\partial^2}{\partial y^2} L_\sigma(x,y) \tag{3.14}$$

$$= \left(\underbrace{\left(\frac{\partial^2}{\partial x^2} G_\sigma + \frac{\partial^2}{\partial y^2} G_\sigma \right)}_{LoG_\sigma} * I \right)(x,y) \tag{3.15}$$

Pour un facteur d'échelle σ donné, la détection de blobs est réalisée en extrayant les extrema locaux (minima et maxima) en espace de la fonction $\Delta L_\sigma(x,y)$. Cependant, l'efficacité de cette détection est intimement liée à la taille des blobs qui n'est pas supposée être connue. La détection des blobs de différentes tailles nécessite l'utilisation d'une approche multi-échelles. Une façon simple d'obtenir un détecteur de blobs multi-échelles est de considérer l'opérateur Laplacien normalisé :

$$\Delta_{\text{norm}} L_\sigma(x,y) = \sigma^2 \Delta L_\sigma(x,y) \tag{3.16}$$

Le Laplacien $\Delta_{\text{norm}} L_\sigma$ est calculé à plusieurs échelles, c'est-à-dire pour plusieurs valeurs de σ (*e.g.* $\sigma \in [0, 100]$). Les blobs sont les extrema locaux en espace et en échelle de $\Delta_{\text{norm}} L_\sigma$ sur un 26-voisinage (9+9+8) (voir figure 3.5) :

$$(\hat{x}, \hat{y}; \hat{\sigma}) = \arg \max_{(x,y;\sigma)} \min (x, y; \sigma) \text{local} \Delta_{norm} L_\sigma(x,y) \tag{3.17}$$

L'intérêt de la méthode est la détection simultanée de la position (x_i, y_i) des blobs et de leur échelle caractéristique σ_i. La probabilité de dévier de plus de $3\sigma_i$ est inférieure à 1%. Chaque blob correspond à un disque centré en (x_i, y_i) et de rayon $3\sigma_i$ (voir figure 3.4).

3.2.5 Harris-Laplace

Suite aux travaux de Harris et Stephens [HS88] et de Lindeberg [Lin98], Mikolajczyk et Schmid proposent dans [MS01] un détecteur de points d'intérêts multi-échelles nommé Harris-Laplace utilisant les avantages de ces deux méthodes, à savoir la performance du détecteur de Harris couplée à l'aspect multi-échelles utilisé par Lindeberg avec le détecteur LoG.
Soit L_{σ_D}, la représentation de la fonction d'intensité I à l'échelle σ_D (voir équation (3.12)). La matrice de structure A se définit par :

$$A(x,y;\sigma_D,\sigma_I) = {\sigma_D}^2 \cdot \begin{bmatrix} \left(G_{\sigma_I} * \left(\frac{\partial L_{\sigma_D}}{\partial x} \right)^2 \right)(x,y) & \left(G_{\sigma_I} * \left(\frac{\partial L_{\sigma_D}}{\partial x} . \frac{\partial L_{\sigma_D}}{\partial y} \right) \right)(x,y) \\ \left(G_{\sigma_I} * \left(\frac{\partial L_{\sigma_D}}{\partial x} . \frac{\partial L_{\sigma_D}}{\partial y} \right) \right)(x,y) & \left(G_{\sigma_I} * \left(\frac{\partial L_{\sigma_D}}{\partial y} \right)^2 \right)(x,y) \end{bmatrix} \tag{3.18}$$

FIGURE 3.4 – *Détection de blobs par la méthode LoG sur l'image « tournesol » prise par la photographe Candy Torres. Les cercles représentent les* 120 *maxima en échelle et en espace de la fonction* $\Delta_{norm}L$*. La taille des cercles indique l'échelle caractéristique de chaque blob.*

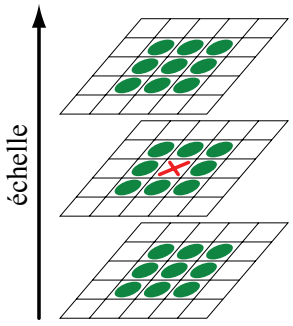
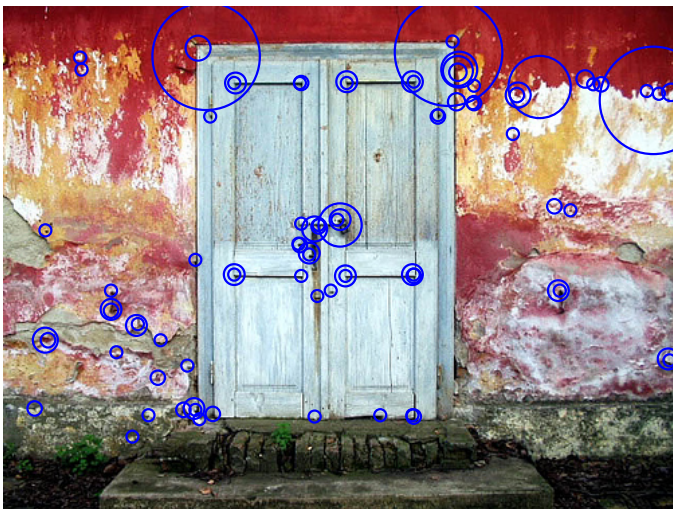


FIGURE 3.5 – *26-voisinage utilisé pour la détection des maxima locaux de l'opérateur LoG.*

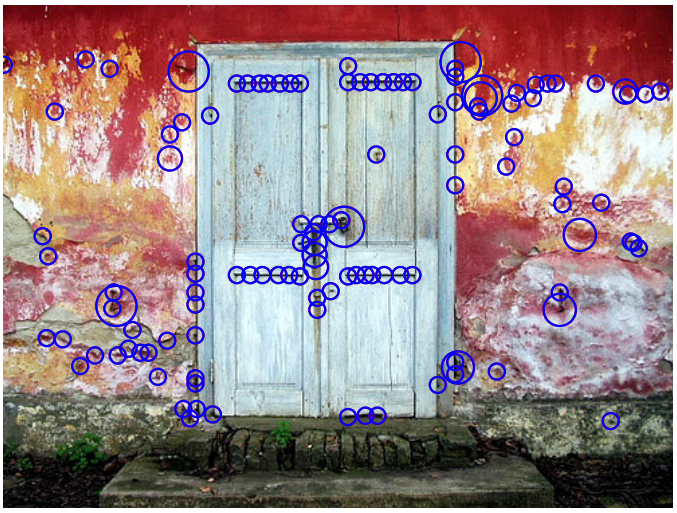
où σ_I est appelé échelle d'intégration et σ_D échelle de dérivation. L'approche proposée ne détecte pas les points 3D directement comme le fait l'approche de Lindeberg. Dans un premier temps, les auteurs proposent de détecter, pour chaque échelle, les points d'intérêt 2D au sens de Harris en utilisant la matrice de structure A définie dans (3.18) et en exhibant les maxima locaux en espace de la fonction définie dans (3.10). Parmi les points d'intérêt détectés, ceux qui correspondent à un extremum local en échelle au sens de Lindeberg sont retenus comme étant les points d'intérêt de Harris-Laplace.
La figure 3.6 montre un résultat de détection de points d'intérêt par la méthode de Harris-Laplace. Les points extraits correspondent effectivement pour la plupart à des coins dans l'image. Cette approche est différente de celle de Lindeberg pour laquelle seuls les blobs sont détectés. Nous pouvons également remarquer que le détecteur est beaucoup moins sensible au bruit de l'image que le détecteur de Harris. Cette robustesse est induite par la détection multi-échelles. Le détecteur de Harris-Laplace est considéré à l'heure actuelle comme étant le meilleur détecteur multi-échelles au sens des critères de répétabilité et de précision [MS01].
Les régions détectées sont, par construction, circulaires. Mikolajczyk et Schmid améliorent dans [MS04] le détecteur Harris-Laplace en utilisant des voisinages ellipsoïdaux. L'appariement d'images est alors plus robuste aux changements de point de vue.

§ 3.3 VECTEURS DE DESCRIPTION

Dans la partie précédente, nous avons étudié la détection de points d'intérêt mono ou multi-échelles. Ces points ont été retenus car ils sont situés à des endroits où le signal présente un contenu informatif important. Soit $\{(\mathbf{x}_i, s_i)\}_{1 \leq i \leq n}$, un ensemble de n points d'intérêt dont la position est notée $\mathbf{x}_i$ et l'échelle caractéristique s_i. Dans le cas d'une détection mono-échelle (par exemple Harris), s_i est commune à tous les points détectés.
La mise en correspondance de points d'intérêt par la seule information de position et d'échelle n'est évidemment pas fiable. Pour réaliser la mise en correspondance, nous devons utiliser l'information contenue dans les images où les points sont détectés. Généralement, les informations utilisées sont l'intensité, la couleur, la norme du gradient, l'orientation du gradient, etc. Cependant, l'information présente au pixel $\mathbf{x}_i$ n'est pas suffisante pour une mise en correspondance fiable de points ; l'information y est trop peu discriminante. Le vecteur de description associé au point d'intérêt $(\mathbf{x}_i, s_i)$ est un ensemble de valeurs extraites à partir de l'image I dans le voisinage local de la position $\mathbf{x}_i$. La taille du voisinage est idéalement définie par s_i. L'espace des vecteurs de description est un espace vectoriel normé où une distance permet de mesurer la similarité entre les vecteurs (voir section 3.4).



(a) Détecteur Harris-Laplace



(b) Détecteur LoG

FIGURE 3.6 – *Détection de points d'intérêts par le détecteur de Harris-Laplace et le détecteur LoG sur l'image « vieille porte » prise par la photographe Aleksandra Radonic'. Les points d'intérêt sont différents pour les deux méthodes : il s'agit de coins pour Harris-Laplace et de blobs pour le détecteur LoG. La taille des cercles indique l'échelle caractéristique de chaque point d'intérêt.*

Plusieurs types de vecteurs de description existent. Dans cette partie, nous en présentons trois types classiques.

3.3.1 Voisinage de pixels

Le vecteur de description le plus simple est la matrice contenant les valeurs de l'image dans un voisinage (par exemple carré) de chaque point d'intérêt. La taille du voisinage utilisé est généralement de $6s_i \times 6s_i$ pixels. La figure 3.7 illustre un exemple de la construction de vecteurs de description. Pour chaque point, le voisinage considéré (encadré en rouge) est de taille 3×3 pixels. Ce voisinage est généralement exprimé sous forme de vecteur.
L'utilisation de plusieurs composantes permet d'augmenter le pouvoir dis-

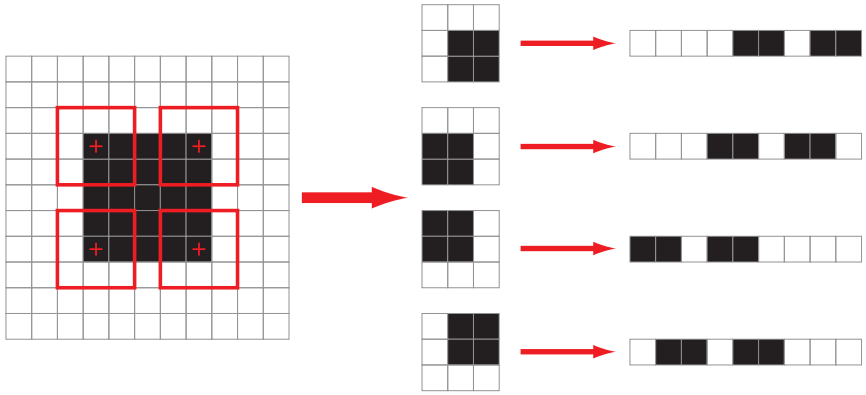


FIGURE 3.7 – *Construction de vecteurs de description pour 4 points d'intérêt indiqués par des croix rouges. Le voisinage de chaque point, délimité en rouge, est de taille* 3×3*. La matrice de pixel correspondant à chaque voisinage est extraite, puis est exprimée sous forme de vecteur.*

criminant des vecteurs de description. Ainsi, l'intensité de l'image est sensible aux changements d'illumination tandis que la norme et la direction du gradient ne le sont pas. Le choix des composantes est crucial et dépend de l'application.
L'utilisation d'un voisinage de pixels pour la mise en correspondance de points d'intérêt utilise une information géométrique stricte, c'est-à-dire que la position et la valeur de chaque pixel est importante. Ce type de vecteur de description est très discriminant spatialement mais il est sensible aux changements géométriques (zoom, rotation). Deux points seront identiques si et seulement si les voisinages de pixels sont identiques. Lors de la mise en correspondance de deux nuages de points d'intérêt détectés dans deux images consécutives d'une vidéo, les propriétés géométriques des objets changent peu, surtout si la fréquence d'acquisition est élevée. La mise en correspondance des points d'intérêt à partir de la matrice de voisinage

est par conséquent très fiable car très discriminante. En revanche, dans un cas plus général comme rencontré en recherche d'images dans une base de données, les images d'une même scène sont acquises avec des angles de vue très différents. L'orientation, l'échelle, la couleur et l'intensité des objets peuvent changer ce qui implique que l'utilisation d'une matrice de voisinage donne de mauvais résultats. Le but des méthodes présentées par la suite est de pallier ces problèmes.
Enfin, il faut noter un problème intrinsèque à l'utilisation de voisinages de pixels pour décrire un point d'intérêt. D'une part, la détection de points d'intérêt par une méthode multi-échelles implique que les voisinages extraits pour l'ensemble des points soient de taille différente. Souvent, une mesure de similarité, comme décrite dans la partie 3.4, va comparer les vecteurs de description élément par élément. Une telle comparaison est impossible si les voisinages utilisés sont de taille différente. D'autre part, la taille du vecteur de description (taille du voisinage) est généralement très grande. Par exemple, pour un point d'intérêt dont le voisinage est de taille 15×15, le vecteur de description est de taille 225. Cette taille importante a un impact sur le temps nécessaire pour comparer deux vecteurs de description. L'utilisation de tels vecteurs pose de gros problèmes en temps de calcul pour la recherche d'images par le contenu dans une grande base d'images. L'état de l'art nous apprend qu'un vecteur de description doit avoir une taille maximale de 150 pour de telles applications.

3.3.2 Distribution

L'utilisation de voisinage de pixels n'est pas un bon choix de descripteur dans un cadre général d'appariement d'images. Toute transformation autre qu'une simple translation modifie la structure géométrique globale et locale de l'image. En revanche, les transformations usuelles ont un impact limité sur les propriétés statistiques de l'image. La figure 3.8 illustre la construction d'un vecteur de description fondé sur la distribution de l'intensité. La première étape consiste à extraire la matrice de pixels dans le voisinage local du point d'intérêt. La seconde étape consiste à calculer la distribution des éléments de cette matrice. Plusieurs approches sont possibles (voir annexe A). Cette distribution est généralement approximée par un histogramme normalisé. Dans l'exemple présenté, l'histogramme a 8 classes de largeur 32, l'intensité d'un pixel s'exprimant par un nombre entier naturel compris dans l'intervalle $[0, 255]$. Swain et Ballard [SB91] ont utilisé des histogrammes couleurs pour leur méthode d'indexation. D'autres attributs que la couleur, comme l'intensité des réponses obtenues par des dérivées de Gaussienne ou par des filtres de Gabor sont utilisés dans [SC96].
La distribution est une information purement statistique. Ces histogrammes ne prennent pas en compte la répartition spatiale des attributs dans le voi-

sinage du point, ce qui leur confère l'invariance aux déformations géométriques usuelles. L'utilisation de distribution pour la description locale pose néanmoins un problème : deux matrices de pixels visuellement très différentes peuvent avoir la même distribution (voir figure 3.9).
L'utilisation d'une matrice de pixels comme vecteur de description pose problème si la détection des points est multi-échelles ; il est impossible de comparer point à point des vecteurs de description de taille différente. Ce problème ne se rencontre pas dans le cas de distributions. En effet, le nombre de pixels contenus dans le voisinage local d'un point d'intérêt ne modifie en rien le nombre de classes de l'histogramme. Les caractéristiques des histogrammes sont fixées et sont identiques pour tous les vecteurs de description construits. De plus, à la différence des vecteurs de description utilisant des matrices de pixels, l'utilisation de distribution permet de définir des vecteurs de taille réduite. Cette propriété est particulièrement intéressante pour la recherche d'images par le contenu dans une grande base de données.
Il est naturellement possible d'utiliser plusieurs composantes. Dans un tel cas, nous pouvons soit construire une distribution en dimension supérieure, ou, plus simplement, nous pouvons coller bout à bout les distributions calculées pour chacune des composantes. Parmi les composantes image existantes, l'utilisation de la norme et de la direction du gradient permet de rendre l'appariement robuste aux changements d'intensité.

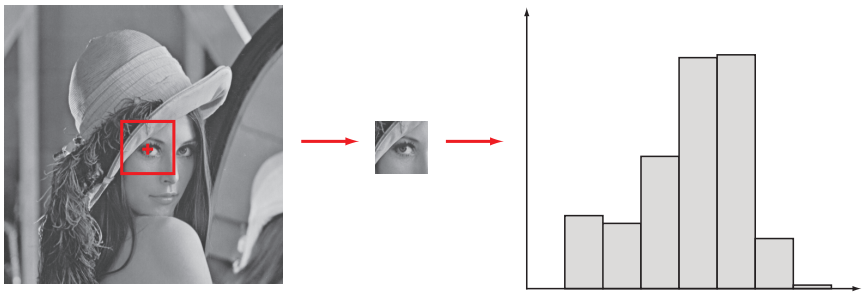


FIGURE 3.8 – *Construction de vecteurs de description fondé sur l'estimation de la distribution de l'intensité. Le point d'intérêt est localisé par une croix rouge et son voisinage, dépendant de son échelle caractéristique et délimité en rouge, est de taille* 97×97*. La distribution est approximée par un histogramme normalisé à 8 classes.*

3.3.3 SIFT

La méthode SIFT [Low99, Low04] (pour *Scale-Invariant Feature Transform*), est une méthode permettant de décrire un point à partir des orientations locales du gradient. Les vecteurs de description construits par cette méthode sont spatialement discriminants (contrairement aux distributions)

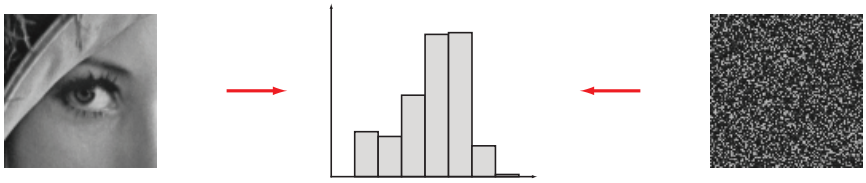


FIGURE 3.9 – *Illustration du problème intrinsèque à l'utilisation de distributions pour la description de points d'intérêt. Deux images visuellement très différentes ont exactement la même distribution.*

et robustes aux déformations géométriques usuelles et aux variations d'intensité.
Chaque point d'intérêt est décrit à partir de son voisinage de taille 16×16 à son échelle caractéristique. Dans ce voisinage est calculé l'histogramme des orientations locales. Leur contribution à l'histogramme est pondérée par la norme du gradient et par un facteur Gaussien d'éloignement au centre du voisinage (point d'intérêt). L'invariance à la rotation est assurée en tournant le voisinage et les orientations qu'il contient d'un angle égal à l'orientation dominante (orientation maximisant l'histogramme des orientations). Les figures 3.10 et 3.11 illustrent la construction du vecteur de description SIFT autour d'un point d'intérêt. Le voisinage de taille 16×16 tourné est divisé en 16 (4×4) sous-ensembles de dimension 4×4 pixels. Par commodité, le voisinage représenté sur la figure est de taille 8×8 et il est découpé en 2×2 carrés. Un histogramme d'orientations est construit pour les 16 sous-ensemble, chaque histogramme comporte 8 classes. Le vecteur de description est la concaténation des 16 histogrammes. Un vecteur de description SIFT comporte $16 \times 8 = 128$ éléments.
L'utilisation de l'orientation du gradient permet à la méthode d'être robuste aux variations d'intensité. L'intérêt de cette méthode réside dans le découpage du voisinage. Utiliser un histogramme pour chacun des 16 sous-voisinages permet d'ajouter une information spatiale qui est perdue si l'on utilise un histogramme sur tout le voisinage. L'information spatiale n'est cependant pas aussi contraignante que dans le cas d'une simple matrice de pixels. De plus, la rotation du voisinage (et des orientations qu'il contient) et l'extraction des orientations à l'échelle caractéristique rend le vecteur de description robuste aux rotations et aux changements d'échelle. Des vecteurs de description ainsi construits donnent de très bons résultats dans tous les cas. Le descripteur SIFT est l'un des descripteurs les plus robustes et les plus discriminants [MS05].

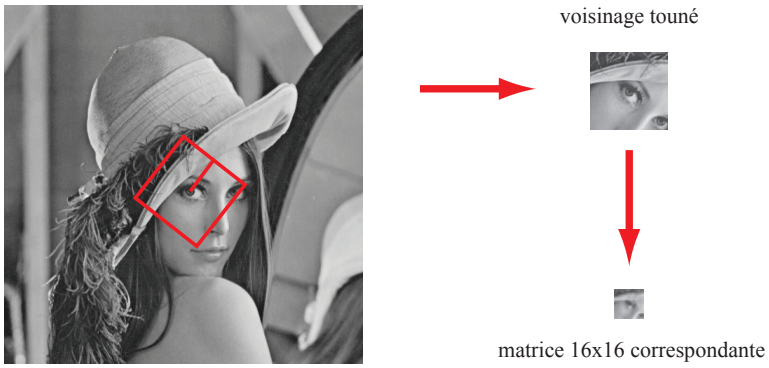


FIGURE 3.10 – *Construction de la matrice de pixels utilisée pour construire le vecteur de description SIFT. Le voisinage (encadré en rouge) est tourné relativement à la direction dominante du voisinage (segment rouge). La matrice utilisée pour construire le vecteur de description est de taille* 16×16*. Un simple interpolation de l'image d'origine permet d'extraire un voisinage tourné et de taille* 16×16*. Pour des raisons de compréhension, l'image utilisée ici représente la composante d'intensité. La méthode SIFT utilise l'orientation du gradient.*

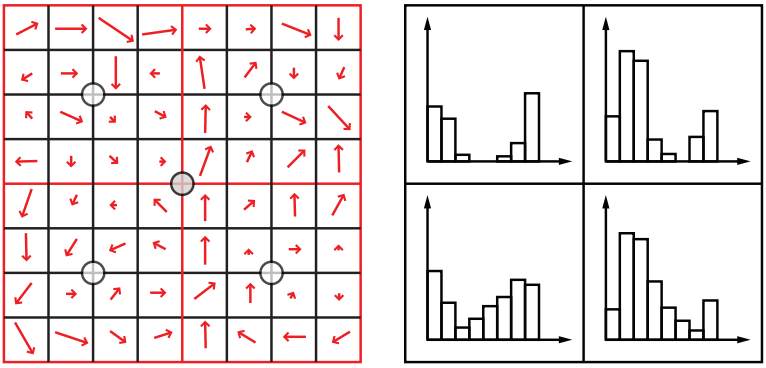


FIGURE 3.11 – *Construction du vecteur de description SIFT à partir de la matrice d'orientations de taille* 16×16 *tournée. Nous représentons ici une matrice de taille* 8×8 *pour clarifier l'illustration. De même, nous découpons la matrice en* 2×2 *parties au lieu des* 4×4 *habituelles.*

§ 3.4 MESURES DE SIMILARITÉ

Jusqu'à présent, nous avons introduit la détection de points d'intérêt et la construction de vecteurs de description. Dans la partie suivante, nous présentons deux mesures classiques permettant d'évaluer la similarité de deux vecteurs de description. Ces mesures sont utilisées pour apparier des ensembles de points d'intérêt.

3.4.1 Corrélation

En théorie des probabilités, la corrélation (ou coefficient de corrélation) indique la force et la direction de la relation linéaire existant entre deux variables aléatoires. Soient X et Y, deux variables aléatoires de moyenne μ_X et μ_Y et d'écart type σ_X et σ_Y. Le coefficient de corrélation $\rho_{X,Y}$ entre ces deux variables est donné par

$$\rho_{X,Y} = \frac{\text{cov}(X,Y)}{\sigma_X \sigma_Y} = \frac{E((X-\mu_X)(Y-\mu_Y))}{\sigma_X \sigma_Y}, \tag{3.19}$$

avec E l'espérance mathématique et cov la covariance. Par définition, nous avons $\mu_X = E(X)$, ${\sigma_X}^2 = E(X^2) - E^2(X)$ et idem pour Y. L'équation (3.19) se récrit comme suit :

$$\rho_{X,Y} = \frac{E(XY) - E(X)E(Y)}{\sqrt{E(X^2) - E^2(X)}\ \sqrt{E(Y^2) - E^2(Y)}}. \tag{3.20}$$

La corrélation n'est définie que si les valeurs σ_X et σ_Y sont finies et non nulles. Il s'agit d'un corollaire de l'inégalité de Cauchy-Schwarz indiquant que la corrélation ne peut excéder 1 en valeur absolue.
Soient $x = (x_1, \cdots, x_n)$ et $y = (y_1, \cdots, y_n)$, deux ensembles de n valeurs. Le coefficient de corrélation de Pearson peut être utilisé pour calculer la corrélation de x et y :

$$r_{xy} = \frac{\sum (x_i - \mu_x)(y_i - \mu_y)}{n \sigma_x \sigma_y}, \tag{3.21}$$

où la somme est réalisée entre 1 et n. Comme précédemment, nous pouvons récrire ce coefficient de la manière suivante :

$$r_{xy} = \frac{n \sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n \sum x_i^2 - \left(\sum x_i\right)^2} \sqrt{n \sum y_i^2 - \left(\sum y_i\right)^2}}. \tag{3.22}$$

Le coefficient de corrélation est égal à 1 dans le cas où l'une des variables est fonction affine croissante de l'autre variable, et est égal à -1 dans le cas où la fonction affine est décroissante. Les valeurs intermédiaires renseignent sur le degré de dépendance linéaire entre les deux variables. Plus le coefficient est proche des valeurs extrêmes -1 et 1, plus la corrélation entre les variables est forte. Une corrélation égale à 0 signifie que les variables sont linéairement indépendantes. Cohen [Coh88] suggère une interprétation de la valeur du coefficient de corrélation (voir tableau 3.1).
La corrélation est utilisée pour comparer des vecteurs de description, par exemple pour des vecteurs fondés sur la matrice des pixels (voir partie 3.3.1). Ce coefficient indique si les deux matrices se ressemblent ou non. Un intérêt important de la corrélation est la possibilité de fixer un seuil à partir duquel deux descripteurs sont considérés comme identiques. Cependant, la corrélation est sensible aux données aberrantes rendant le processus d'appariement instable dans le cas d'images bruitées.

Corrélation	Négative	Positive
Nulle	-0.10 à 0	0 à 0.10
Faible	-0.29 à -0.10	0.10 à 0.29
Moyenne	-0.49 à -0.30	0.30 à 0.49
Forte	-1.00 à -0.50	0.50 à 1.00

TABLE 3.1 – *Interprétation de la valeur du coefficient de corrélation proposée par Cohen [Coh88].*

3.4.2 Distances

La similarité entre deux vecteurs de description peut être évaluée par une distance usuelle. Une distance sur un ensemble E est une application $d : E \times E \rightarrow \mathbb{R}^+$ telle que

- $\forall x, y \in E, d(x, y) = d(y, x)$,
- $\forall x, y \in E, d(x, y) = 0 \Leftrightarrow x = y$,
- $\forall x, y, z \in E, d(x, z) \leq d(x, y) + d(y, z)$.

Un ensemble muni d'une distance s'appelle un espace métrique. Dans un espace vectoriel normé $(E, \|\cdot\|)$, il est toujours possible de définir de manière canonique une distance d à partir de la norme. En effet, il suffit de poser : $\forall (x, y) \in E \times E : d(x, y) = \|y - x\|$. En particulier, dans $\mathbb{R}^n$, on peut définir de plusieurs manières la distance entre 2 points, bien qu'elle soit généralement donnée par la distance Euclidienne.
Un vecteur de description est un point dans un espace vectoriel normé E. Parmi les distances usuelles, nous pouvons nommer la distance de Manhattan (ou 1-distance) et la distance Euclidienne (ou 2-distance). Ces distances sont dérivées de la distance de Minkowski d'ordre p (ou p-distance) (voir tableau 3.2).
La distance Euclidienne est très largement utilisée comme mesure de similarité entre vecteurs de description. En effet, son implémentation est simple et rapide. Lowe utilise cette distance pour comparer les vecteurs construits par la méthode SIFT [Low04]. Il utilise également un seuil à partir duquel l'appariement de deux points est considéré comme fiable. La distance Euclidienne est une mesure de similarité bien adaptée si la différence de deux vecteurs suit une loi normale. Dans le cas contraire, la distance Euclidienne est sensible aux données aberrantes. La 1-distance est connue pour être très robuste à ces données aberrantes. Cependant, la norme 1 (associée à la 1-distance) est une fonction non dérivable en 0. De nombreux travaux ont tenté d'approximer cette norme par une fonction dérivable (*e.g.* approximation de la variation totale). Dans le cas de la mise en correspondance entre vecteur de description, cette propriété de non dérivabilité en 0 ne pose pas de problème.

Nom de la distance	Formule		
p-distance	d(x,y)	=	$\sqrt[p]{\sum_{i=1}^{n} \lvert x_i - y_i \rvert^p}$
1-distance	d(x,y)	=	$\sum_{i=1}^{n} \lvert x_i - y_i \rvert$
2-distance	d(x,y)	=	$\sqrt{\sum_{i=1}^{n} \lvert x_i - y_i \rvert^2}$
∞-distance	d(x,y)	=	$\lim_{p \to \infty} \sqrt[p]{\sum_{i=1}^{n} \lvert x_i - y_i \rvert^p}$
		=	$\sup_i \lvert x_i - y_i \rvert$

TABLE 3.2 – *Distances classiques pour* $x = (x_1, x_2, \cdots, x_n)$ *et* $y = (y_1, y_2, \cdots, y_n)$*, deux vecteurs de E.*

En statistiques, la distance de Mahalanobis [Mah36] est une mesure fondée sur la corrélation de plusieurs variables aléatoires. Contrairement à la distance Euclidienne, la distance de Mahalanobis prend en compte la corrélation des éléments du vecteur considéré. Ainsi, les différentes composantes du vecteur sont pondérées de manière à minimiser l'impact des composantes bruitées. La distance de Mahalanobis est souvent utilisée pour la détection de données aberrantes.
Soit $x = (x_1, x_2, x_3, \ldots, x_p)$ un vecteur de p variables aléatoires ayant pour moyenne $\mu = (\mu_1, \mu_2, \mu_3, \ldots, \mu_p)$. Soit Σ la matrice de covariance de x. La distance de Mahalanobis est définie comme suit :

$$D_M(x) = \sqrt{(x-\mu)^T \Sigma^{-1} (x-\mu)}. \tag{3.23}$$

Dans notre cas, cette valeur correspond à la différence entre deux vecteurs de description. M est une matrice définie positive et donc inversible. La distance de Mahalanobis peut également être définie comme étant la mesure de dissimilarité entre deux vecteurs aléatoires x et y de même distribution et ayant pour matrice de covariance Σ :

$$D_M(x,y) = \sqrt{(x-y)^T \Sigma^{-1} (x-y)}. \tag{3.24}$$

Si la matrice de covariance est la matrice identité, la distance de Mahalanobis est égale à la distance Euclidienne. Si la matrice de covariance est diagonale, elle est appelée distance Euclidienne normalisée :

$$d(x,y) = \sqrt{\sum_{i=1}^{p} \frac{(x_i - y_i)^2}{\sigma_i^2}} \tag{3.25}$$

où σ_i est l'écart type de x_i.
Schmid *et al.* [SM96] utilisent la distance de Mahalanobis pour comparer des vecteurs de description fondés sur les invariants de niveau de gris (non présentés dans la partie 3.3). Les auteurs utilisent un seuil pour décider si deux vecteurs sont similaires. La qualité des résultats obtenus avec cette distance dépend de la représentativité de la matrice de covariance. Cette matrice doit tenir compte du bruit des images, des variations d'éclairage et de l'imprécision en position des points d'intérêt. Le changement de seulement un pixel perturbe de façon importante la valeur des invariants. Un calcul théorique de cette matrice est difficile puisque la forme du signal autour d'un point d'intérêt est quelconque. Un tel calcul est possible si l'on restreint les points utilisés à des coins. Cette matrice est donc estimée de façon empirique. Pour ce faire, un point d'intérêt est suivi manuellement sur toute la séquence vidéo. La matrice de covariance est calculée à partir de l'ensemble des vecteurs d'invariant calculés pour chacune des images. Pour obtenir une matrice représentative de la variété des points, ce calcul a été réalisé pour plusieurs points sur plusieurs séquences. Nous voyons ici que l'estimation de la matrice de covariance semble complexe et lourde à mettre en oeuvre. Les auteurs disent ainsi que la distance de Mahalanobis est fort peu pratique pour certaines applications nécessitant une exécution rapide. Un changement de base est possible, sous certaines conditions, ce qui permet d'utiliser une simple distance Euclidienne. Ce changement de base $x \mapsto Lx$ se fait grâce à la décomposition de Cholesky $M = L^T L$ avec $D_M(x, y) = \|Lx - Ly\|_2$.

§ 3.5 CONCLUSION

Dans ce chapitre, nous avons étudié la description locale d'une image. Cette description repose sur les étapes de détection de points d'intérêt et sur la construction de vecteurs de description. De manière générale, la description locale permet de mettre en correspondance des points dans deux images différentes. Ces correspondances sont ensuite utilisées pour de nombreuses applications telles que le recalage d'images ou la recherche d'images dans une base de données.
Considérons deux images d'une même scène mais prises à des angles de vue et à des moments différents. La position, l'orientation et l'échelle (la taille) des objets diffèrent d'une image à l'autre. De même l'intensité liée à l'éclairage de la scène peut varier. Dans un tel cadre, la littérature indique que le détecteur de points d'intérêt multi-échelles Harris-Laplace couplé à l'utilisation de la méthode SIFT pour construire les vecteurs de description donne les meilleurs résultats [MS05, MS04]. En effet, l'appariement des points est robuste aux changements d'échelles, aux rotations, aux changements d'intensité, et aux modifications géométriques locales des objets

dans une certaine mesure. La mesure de similarité généralement employée pour comparer des vecteurs de description fondés sur SIFT est la distance Euclidienne.

- CHAPITRE 4 -

SUIVI D'APRÈS LES TRAJECTOIRES DE POINTS D'INTÉRÊT

§ 4.1 INTRODUCTION

Dans ce chapitre, nous proposons trois méthodes de suivi d'objet fondées sur le suivi de points d'intérêt. Ces méthodes reposent toutes sur une hypothèse commune : le mouvement [1] de l'objet d'intérêt peut être déduit du mouvement des points d'intérêt définis dans l'objet. En d'autres termes, les points d'intérêt vont guider l'objet d'intérêt au cours du temps. La première méthode proposée consiste à calculer le mouvement de l'objet à partir d'informations extraites entre deux images. La deuxième introduit l'utilisation d'un groupe d'images (GOP pour « Group Of Pictures ») permettant d'améliorer l'estimation du mouvement. Enfin, la troisième ajoute l'utilisation d'un GOP glissant ainsi qu'une pondération spatio-temporelle améliorant une fois encore l'estimation du mouvement et permettant de tenir compte de déformations locales.
Un objet d'intérêt sur l'image I^t est déterminé par son contour C^t. Ce contour est un ensemble de N_C points d'échantillonnage $C^t = \{c_1^t, c_2^t, \cdots, c_{N_C}^t\}$ relié par une courbe. Le choix du type de courbe utilisé est fonction de l'objet d'intérêt. Un objet aux contours rectilignes (par exemple un véhicule) sera bien représenté par un polygone alors qu'un objet aux contours arrondis (corps humain) sera mieux représenté par une spline. Les méthodes que nous présentons dépendent des points d'échantillonnage des contours ainsi que de la région intérieure de chaque contour. La paramétrisation du contour ne sera plus évoquée par la suite. Seuls les points d'échantillonnage et la région intérieure aux contours le seront.

1. Le mouvement d'un objet dans une vidéo est généralement dû au mouvement de l'objet dans la scène et/ou au mouvement de la caméra (*e.g.* rotation de la caméra (panoramique), translation de la caméra (*travelling*), ou encore *travelling* optique (zoom)).

Le contour C^1 est défini manuellement ce qui permet d'assurer qu'il corresponde précisément au contour de l'objet. Pour expliquer nos méthodes, nous considérons par la suite que le contours $C^2, \cdots, C^t$ sont d'ores et déjà calculés. Pour V une vidéo de N_V images, le problème de suivi consiste à calculer les contours $C^{t+1}, \cdots, C^{N_V}$. Le contour C^t est appelé contour de référence car les contours suivants seront déduits de ce contour.

§ 4.2 MOUVEMENT GLOBAL ENTRE DEUX IMAGES

Nous proposons dans cette partie une méthode de suivi d'objet ou le mouvement global cet objet est estimé à partir d'informations extraites entre deux images consécutives. Le contours C^{t+1} est déduit en appliquant le mouvement ainsi estimé au contour de référence C^t.

4.2.1 Appariement de points d'intérêt

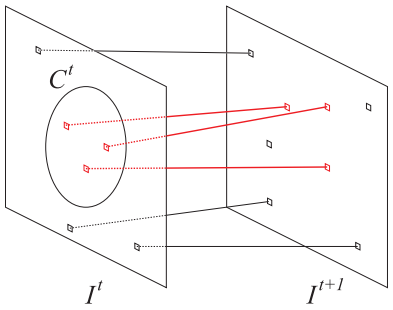


FIGURE 4.1 – *Construction de couples de points d'intérêt entre les image I^t et I^{t+1}. Les points sont détectés sur les image I^t et I^{t+1} et sont appariés par une méthode de validation croisée. L'appariement est fondé sur l'utilisation de vecteurs de description calculés pour chaque point d'intérêt extrait et relativement à la mesure de similarité entre vecteurs de description choisie. Les couples de points d'intérêt représentés en rouge sont les couples caractéristiques, c'est-à-dire ceux dont le point d'intérêt de l'image I^t sont définis dans le domaine formé par le contour C^t.*

La méthode proposée repose sur l'utilisation de couples de points d'intérêt créés entre les images I^t et I^{t+1}. L'appariement de points d'intérêt, utilisé pour la construction de ces couples, est une technique classique reposant sur la description locale présentée dans le chapitre 3. Cette construction, illustrée sur la figure 4.1, est réalisée en quelques étapes détaillées ci-après :

1. Détection des points d'intérêt p_i^t de l'image I^t puis extraction du vecteur de description local correspondant pour chacun des points.

2. Détection des points d'intérêt p_j^{t+1} de l'image I^{t+1} puis extraction du vecteur de description local correspondant pour chacun des points.
3. Mise en correspondance des deux ensembles de points par une validation croisée.

Pour un point d'intérêt p_i^t, l'appariement simple consiste à sélectionner le point p_j^{t+1} le plus proche de p_i^t au sens du vecteur de description et de la mesure de similarité utilisés. Les appariements créés par cette méthode peuvent être aberrants. La validation croisée est une méthode permettant de rejeter la plupart de ses appariements aberrants. La validation consiste à retenir les couples dont les points d'intérêt se sélectionnent mutuellement : Pour un point d'intérêt p_i^t, le point p_j^{t+1} le plus proche de p_i^t est sélectionné. Ensuite, le point p_k^t le plus proche de p_j^{t+1} est sélectionné. Si les points p_i^t et p_k^t correspondent au même point sur l'image I^t, cela signifie que les points p_i^t et p_j^{t+1} se sont choisis mutuellement, et l'appariement est retenu. Dans le cas contraire, l'appariement est rejeté. La figure 4.2 illustre le principe de la validation croisée. La méthode de validation croisée permet d'obtenir des appariements le plus souvent corrects. Cette qualité est obtenue par le fait qu'un point d'intérêt a un voisinage riche qui est alors exprimé par l'intermédiaire d'un vecteur de description. Le taux d'appariements incorrects peut être diminué en utilisant, par exemple, une contrainte de voisinage et une contrainte géométrique. Ces contraintes reposent sur une hypothèse de constance de la structure du voisinage. Ces contraintes sont souvent vérifiées pour deux images d'une même scène et prises au même moment. Dans le cas d'une vidéo et du fait du mouvement des objets, ces contraintes sont trop restrictives et risquent d'éliminer un nombre conséquent d'appariements corrects.

Parmi les couples de points d'intérêt construits, tous ne représentent pas

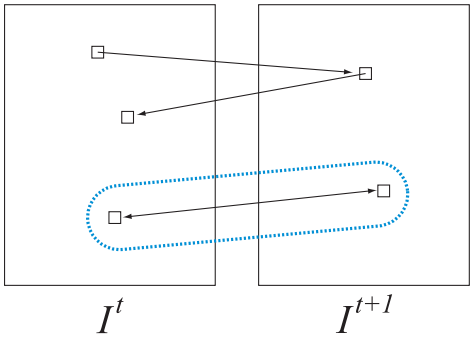


FIGURE 4.2 – *Illustration de la méthode de validation croisée. L'appariement entouré en bleu est validé car les points d'intérêts se choisissent mutuellement. En revanche, l'autre appariement est rejeté car il n'y a pas sélection mutuelle.*

le mouvement de l'objet d'intérêt. Nous définissons les couples caractéris-

tiques comme étant les couples de points d'intérêt $\{p_i^t, p_i^{t+1}\}$ tels que le point p_i^t est défini dans le domaine formé par le contour C^t relativement à sa paramétrisation. Nous supposons que ces points, définis dans l'objet d'intérêt, fournissent une information fiable du mouvement de l'objet d'intérêt. Dans la suite, les couples de points d'intérêt caractéristiques seront notés $\{p_i^t, p_i^{t+1}\}$, leur nombre étant noté N_p. Ces couples sont indiqués par des lignes rouges sur la figure 4.1.
Nous avons vu au chapitre précédant les principales méthodes utilisées en description locale (détection de points d'intérêt, construction de vecteurs de description et mesure de similarité entre vecteurs). Lors de la construction des couples de points d'intérêt décrite ci-avant, nous avons volontairement omis de préciser le choix des méthodes utilisées. La raison est que la plupart des méthodes existantes permettent de construire des couples corrects. Parmi les méthodes existantes, l'utilisation du trio Harris-Laplace + SIFT + norme 2 permettent de détecter les points d'intérêt à leur échelle caractéristique et d'adapter le vecteur de description à cette échelle. La mise en correspondance est alors robuste aux changements d'échelle, aux rotations, et aux transformations affines. Cette propriété est largement utilisée pour des applications d'indexation, de recherche d'image dans une base de données et de recherche de copies [Ton06, Sch96]. Dans notre cas, la mise en correspondance des points d'intérêt est réalisée entre deux images consécutives d'une vidéo. Pour une fréquence de prise de vue standard à 25 images par secondes, les variations entre deux images sont extrêmement faibles. Ainsi, le détecteur de Harris associé à des vecteurs de description contenant le voisinage de chaque point et associé à l'utilisation d'une norme 1 permettent de construire des couples de points fiables et précis. Nous les préférons ainsi pour ces raisons et pour leur simplicité d'utilisation et d'implémentation.

4.2.2 Estimation du mouvement

Soit M^t, la matrice affine à six paramètres décrivant le mouvement du contour C^t vers le contour C^{t+1} :

$$M^t = \begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix} \tag{4.1}$$

Rappelons que nous faisons l'hypothèse que le mouvement de l'objet et de son contour est identique au mouvement des points d'intérêt appartenant au domaine défini par le contour C^t. Pour un couple de points d'intérêt $\{p_i^t, p_i^{t+1}\}$, nous en déduisons l'égalité suivante :

$$p_i^{t+1} = M^t p_i^t \tag{4.2}$$

pour tout i dans $[1, N_t]$. Les points d'intérêt sont exprimés en coordonnées homogènes. Le mouvement M^t n'étant pas connu, le but est de l'estimer. Nous notons alors $\widehat{M}^t$ l'estimée du mouvement M^t. Ce mouvement n'étant à priori pas exact, il en résulte l'égalité suivante

$$p_i^{t+1} = \widehat{M}^t p_i^t + \epsilon_i^{t,\widehat{M}^t} \tag{4.3}$$
$$\tag{4.4}$$

d'où

$$\epsilon_i^{t,\widehat{M}^t} = p_i^{t+1} - \widehat{M^t} p_i^t \tag{4.5}$$

où $\epsilon_i^{t,\widehat{M}^t}$ représente l'erreur de localisation dépendant du couple de points $\{p_i^t, p_i^{t+1}\}$ et de la matrice estimée $\widehat{M^t}$. L'estimation de M^t consiste alors à minimiser une fonction de la norme de cette erreur :

$$\widehat{M}^t = \arg\min_{M} \sum_{i \in [1, N_p]} f(\|\epsilon_{i,M}^t\|) \tag{4.6}$$

où $\|.\|$ correspond à la norme Euclidienne également appelée norme 2, et f étant une fonction de coût. La solution fournie par l'équation (4.6) est usuellement nommée M-estimateur [Hub81, GM95] (« M » pour « maximum likelihood-type » ou « maximum de vraisemblance » en français).

De nombreux travaux [BA96, CBFAB97], portant sur des problèmes de régularisation, proposent des fonctions coût permettant d'augmenter la robustesse de l'estimateur aux données aberrantes. Dans notre cas, les données aberrantes sont généralement observées lors d'une mauvaise mise en correspondance de points d'intérêt. Nous présentons dans le tableau 4.1 et dans la figure 4.3 les principales fonctions utilisées en régularisation. Notez que le ϵ utilisé pour l'approximation de la variation totale est en aucun cas relié au ϵ utilisé dans la partie précédente. Sauf mention explicite, nous utiliserons dans la suite de ce manuscrit la fonction valeur absolue comme fonction de coût.

La minimisation présentée dans l'équation (4.6) consiste à parcourir un espace $\mathbb{R}^6$ (une dimension pour chaque paramètre de la matrice M^t). Le parcours exhaustif de cet espace n'est pas envisageable en terme de coup calcul. Pour effectuer une telle minimisation, des nombreux travaux existent dans la littérature. Pendant notre thèse, nous avons utilisé les deux méthodes d'optimisation suivantes : l'algorithme du simplex [NM65] et l'algorithme du recuit simulé [KGV83]. La méthode d'optimisation dite du "simplexe" est la méthode de Nelder et Mead [NM65] qui calcule rapidement et fiablement en pratique une approximation de la solution (minimum local). Il ne faut surtout pas la confondre avec le célèbre algorithme

Nom de la fonction	$f(x)$
Valeur absolue	$\lvert x \rvert$
Geman & Mc. Clure	$\frac{x^2}{1+x^2}$
Hebert & Leahy	$\log(1+x^2)$
Hypersurface	$\sqrt{x^2+1}-1$
Approximation de la variation totale	$\sqrt{x^2+\epsilon^2}-\epsilon$
Green	$2\log(\cosh(x))$

TABLE 4.1 – *Principales fonctions de régularisation.*

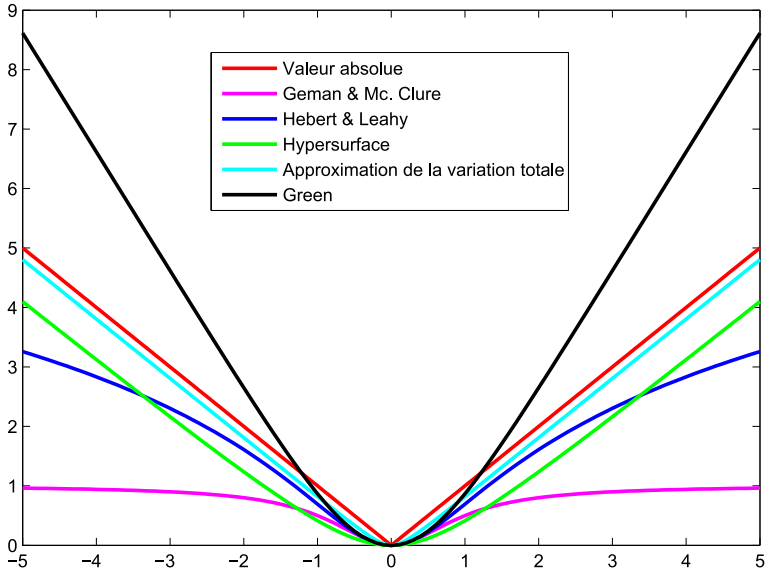


FIGURE 4.3 – *Tracé des principales fonctions de régularisation. Pour l'approximation de la variation totale, ϵ est fixé à $\epsilon = 0.2$.*

du simplexe de Dantzig [Dan51] pour résoudre des problèmes de programmation linéaire (l'optimum global est atteint).
Les résultats obtenus avec ces deux méthodes étant équivalents nous utiliserons dans la suite l'algorithme du simplexe. Enfin, notez que si nous choisissons une fonction de coût égale à la fonction carrée $f(x) = x^2$, il est possible d'obtenir la solution de l'équation 4.6 en une étape et sans passer par un algorithme de minimisation type simplexe ou recuit simulé. Cette solution est obtenue par la méthode des moindres carrés ordinaires. Cette méthode, particulièrement utilisée dans l'industrie pour sa rapidité, est présentée en détails dans l'annexe B. Cependant, la qualité d'estimation des paramètres est incertaine ce qui peut induire un suivi aberrant, raison pour laquelle nous préférons une minimisation classique par la méthode du simplexe.

À ce stade, nous disposons du contour de référence C^t et d'un mouvement estimé $\widehat{M}^t$. Le contour C^{t+1}

$$C^{t+1} = \{c_1^{t+1}, c_2^{t+1}, \cdots, c_{N_C}^{t+1}\} \tag{4.7}$$

s'obtient simplement en appliquant ce mouvement aux points d'échantillonnage du contour de référence :

$$c_i^{t+1} = \widehat{M}^t.c_i^t, \ \forall i \in [1, N_C]. \tag{4.8}$$

où les points d'échantillonnage sont exprimés en coordonnées homogènes.

4.2.3 Résultats sur la séquence Carmap

La figure 4.4 présente un résultat de suivi sur la séquence *Carmap* de dimension 240×320 et contenant 36 images. L'objet que nous voulons suivre est une pancarte rigide dont le mouvement apparent est dû au mouvement de la caméra. Ce mouvement est approximativement une translation. Cependant, du fait du changement de point de vue, la pancarte semble se déformer légèrement. Nous remarquons que le suivi est de très bonne qualité sur cette vidéo. 35 couples de points d'intérêt sont utilisés pour estimer le mouvement de l'objet. Comme le montre la figure 4.4, il est relativement aisé de se faire une idée de la qualité d'une méthode de suivi à partir de critères visuels. Il est en revanche beaucoup plus difficile de comparer deux résultats de suivi proches obtenus en modifiant quelques paramètres (par exemple, la sensibilité du détecteur de points d'intérêt). Nous devons définir un critère indiquant l'erreur commise par la méthode de suivi en comparant le contour estimé par rapport à la vérité terrain, c'est-à-dire le contour réel de l'objet édité manuellement.

Soient C^t et A^t respectivement le contour calculé et le contour réel (segmentation humaine manuelle) de l'objet pour l'image I^t. Bien évidemment,

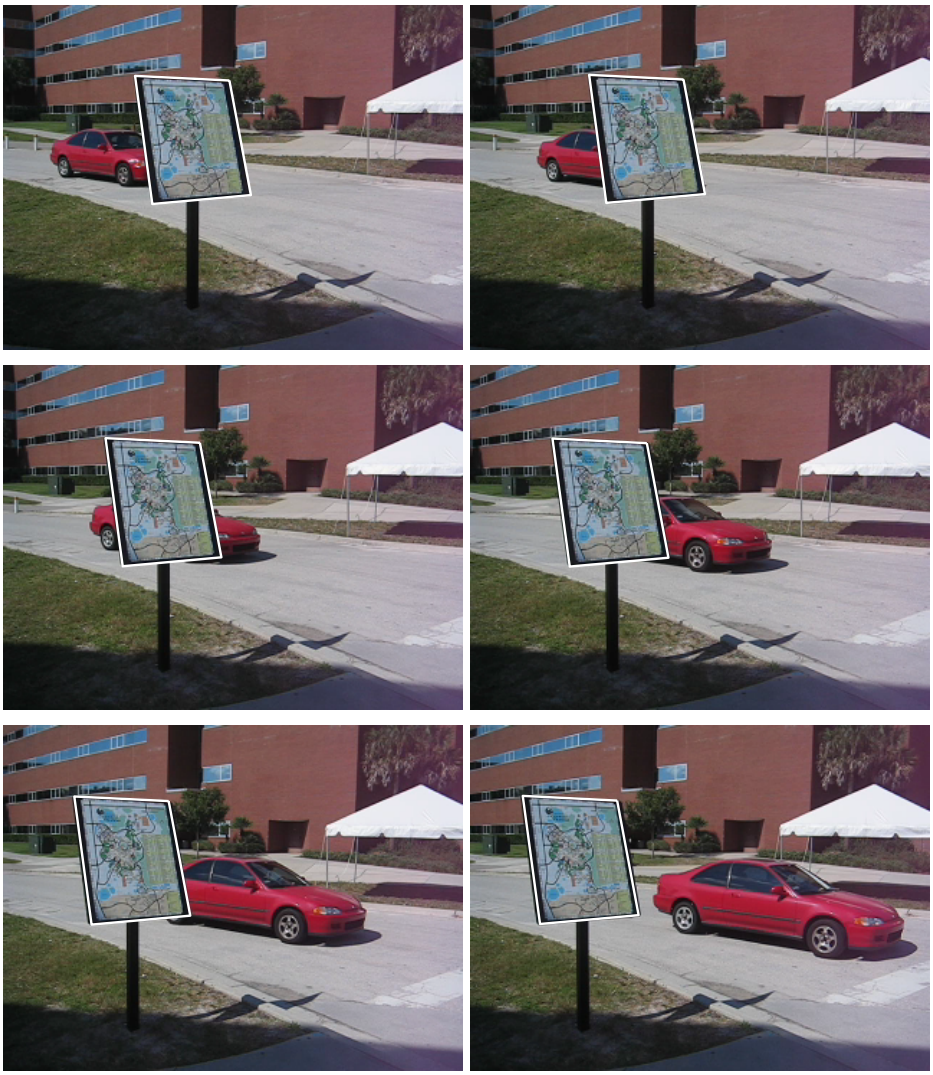


FIGURE 4.4 – *Résultat de suivi pour les images I^1, I^8, I^{15}, I^{22}, I^{29}, et I^{36} (respectivement de gauche à droite et de haut en bas) de la vidéo Carmap.*

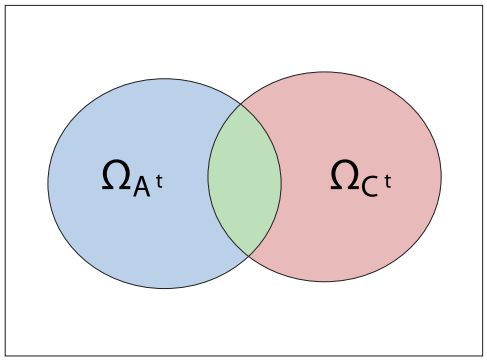


FIGURE 4.5 – *Illustration des domaines Ω_{C^t} et Ω_{A^t} définis à partir des contours C^t (contour estimé) et A^t (contour réel, vérité terrain). La partie verte représente les pixels bien classés. Les parties bleu et rouge représentent les pixels mal classés.*

un algorithme de suivi parfait provoquerait l'égalité $A^t = C^t$. Soient Ω_{C^t} et Ω_{A^t} les domaines définis à partir des contours C^t et A^t. Ces domaines sont un ensemble de pixels et la figure 4.5 en donne l'illustration. Un pixel appartenant à l'ensemble Ω_{A^t} est un pixel de l'objet. Un pixel appartenant à l'ensemble Ω_{C^t} appartient à l'objet si la méthode de suivi ne commet pas d'erreur. Sinon, il se peut que ce pixel n'appartienne pas à l'objet. Ainsi, un pixel est dit « bien classé » s'il appartient à l'intersection des ensembles Ω_{C^t} et Ω_{A^t}. A contrario, un pixel est « mal classé » s'il appartient à la différence symétrique des ensembles Ω_{C^t} et Ω_{A^t}, notée $\Omega_{A^t}\Delta\Omega_{C^t}$, c'est-à-dire s'il appartient à l'un des deux ensembles mais pas aux deux. Nous définissons l'erreur du contour C^t comme étant la proportion de pixels mal classés normalisée par rapport au nombre de pixels appartenant à Ω_{A^t} :

$$E(\Omega_{A^t}, \Omega_{C^t}) = 100\frac{\#(\Omega_{A^t}\Delta\Omega_{C^t})}{\#\Omega_{A^t}} \tag{4.9}$$

où $\#\Omega_{A^t}$ représente le nombre de pixels dans le domaine Ω_{A^t}. La différence symétrique se définit par

$$\Omega_{A^t}\Delta\Omega_{C^t} = \{x|(x \in \Omega_{A^t}) \oplus (x \in \Omega_{C^t})\} \tag{4.10}$$

où le symbole $\oplus$ représente l'opérateur « OU exclusif ». L'erreur $E(\Omega_{A^t}, \Omega_{C^t})$ s'exprime alors en pourcentage de pixels mal classés. Un erreur nulle signifie que les ensembles de pixels Ω_{C^t} et Ω_{A^t} sont identiques (ou vides). L'erreur peut dépasser 100% si $\#(\Omega_{A^t}\Delta\Omega_{C^t}) > \#\Omega_{A^t}$.
Nous définissons l'erreur de la méthode de suivi comme étant l'erreur moyenne des contours $C^2, \cdots, C^{N_V}$ par rapport aux contours réels de l'objet $A^2, \cdots, A^{N_V}$:

$$E(A, C) = \frac{1}{N_V}\sum_{t=2}^{N_V} E(A^t, C^t) \tag{4.11}$$

Nous ne considérons pas les contours A^1 et C^1 car, par définition, $A^1 = C^1$. Ce critère d'erreur sera utilisé dans tout le manuscrit pour évaluer la qualité d'une méthode de suivi.
Sur la vidéo *Carmap*, l'erreur de la méthode de suivi est de seulement 4% de pixels mal classés.

4.2.4 Résultats sur la séquence Arménie

La séquence *Arménie* est de dimension 480×270 et comporte 40 images. L'objet que nous voulons suivre est une voiture dont le mouvement apparent est approximativement une translation globale. L'intérêt de cette vidéo est que l'objet est partiellement occulté sur 11 images. Du fait de l'homogénéité de la voiture et de l'occultation, il y a peu de couples de points d'intérêt qui sont extraits pour estimer le mouvement (11 couples). Nous remarquons ainsi que le suivi est incorrect. En effet, l'erreur de la méthode est alors de 41.5% de pixels mal classés.

FIGURE 4.6 – *Résultat de suivi pour les images I^1, I^8, I^{16}, I^{24}, I^{32}, et I^{40} (respectivement de gauche à droite et de haut en bas) de la vidéo Arménie.*

§ 4.3 MOUVEMENT GLOBAL SUR UN GOP

La méthode proposée auparavant permet de suivre un objet à partir de l'information de mouvement contenue dans certains couples, dits caractéristiques, de points d'intérêt. Cependant, bien que relativement efficace, cette approche n'est pas sans problèmes.
Premièrement, l'estimation de paramètres nécessite en pratique un grand nombre d'observations (les couples caractéristiques) pour être fiable. Pour des objets d'intérêt de petite taille, le nombre de couples caractéristiques peut être de l'ordre d'une dizaine. Ce faible nombre n'est en pratique pas suffisant pour estimer correctement et précisément les paramètres de mouvement. Deuxièmement, il est courant d'observer des couples caractéristiques ne représentant pas le mouvement de l'objet d'intérêt mais représentant le mouvement du fond ou d'un autre objet. De tels couples sont observés lorsque le contour calculé ou manuellement édité ne colle pas parfaitement au contour réel de l'objet, ou encore lorsqu'un objet est partiellement occulté. Nous pouvons voir en effet sur la figure 4.8 qu'une trajectoire coupe sur l'image I^t la trajectoire de l'objet d'intérêt. Le couple extrait à partir de cette trajectoire est alors vu comme un couple aberrant par rapport au mouvement de l'objet. Nous devons donc gérer les problèmes de nombre et de fiabilité des couples caractéristiques.
Une première idée pour augmenter le nombre de couples serait de changer la sensibilité du détecteur de points d'intérêt. Cette augmentation du nombre de points détectés augmente la proportion de « faux » points d'intérêt (points détectés comme étant des points d'intérêt mais ne l'étant pas en réalité, comme par exemple les points détectés dans les zones de bruit présentes dans l'image) et augmente également les possibilités d'appariements défavorisant de fait la validation croisée. De plus, augmenter cette sensibilité est synonyme de diminuer la qualité des points d'intérêt détectés en terme de précision et de contenu informatif. Ainsi, il peut en résulter une augmentation de la proportion de mauvais appariements. Nous devons donc proposer une autre approche pour résoudre les deux problèmes.
Nous proposons de construire des trajectoires temporelles de points d'intérêt plutôt que simplement des couples entres les images I^t et I^{t+1}. De plus, nous proposons d'estimer le mouvement du contour C^t non plus entre les images I^t et I^{t+1} mais sur un groupe de N_{GOP} images $\{I^t, I^{t+1}, \cdots, I^{t+N_{GOP}-1}\}$. Rappelons que le contour est C^t, appelé contour de référence, est connu. Le mouvement de l'objet sur le GOP est estimé à partir de couples extraits des trajectoires, dites caractéristiques, sur l'ensemble des images du GOP. L'utilisation des trajectoires et de GOP permet alors d'augmenter considérablement le nombre de couples caractéristiques tout en réduisant la proportion de couples aberrants. L'estimation des paramètres du mouvement est par conséquent plus précise et plus robuste aux observations aberrantes.

L'hypothèse sous-jacente de l'utilisation d'un GOP est que le mouvement de l'objet d'intérêt est constant sur chaque GOP, ou à l'inverse, la taille du GOP peut être choisie pour que cette hypothèse soit vraie.

4.3.1 Construction et sélection des trajectoires

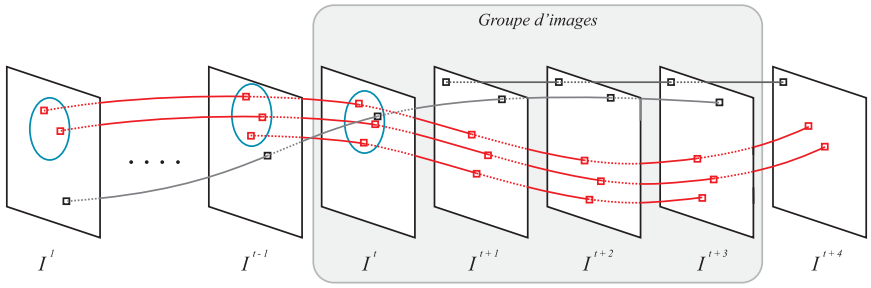


FIGURE 4.7 – *Illustration de trajectoires temporelles de points d'intérêt. Les contours $C^1, \cdots, C^t$ sont connus et représentés par une ellipse bleue. Les trajectoires caractéristiques sont indiquées en rouge. Nous pouvons voir qu'une des trajectoires est définie dans C^t mais que celle-ci ne représente pas le mouvement de l'objet. Cette trajectoire n'est pas caractéristique du fait de son passé. Enfin, est entouré en bleu l'ensemble des images appartenant au GOP. Le contour de la première image du GOP est connu. Un fois estimé, le mouvement global du GOP sera appliqué au contour C^t pour déduire l'ensemble des contours du GOP.*

La construction de trajectoires de points d'intérêt est intimement liée à celle des couples de points d'intérêt. Ces trajectoires sont totalement indépendantes des contours calculés et sont définies sur tout ou partie de la séquence vidéo. Leur construction s'effectue en 3 étapes :

1. Détecter les points d'intérêt pour chaque image de la séquence vidéo et construire les vecteurs de description correspondants.
2. Apparier les points d'intérêt détectés dans une image avec ceux détectés dans l'image suivante. Cet appariement est réalisé par une validation croisée relativement aux vecteurs de description et à la mesure de similarité choisie. Cette étape est en tout point similaire à l'appariement de points d'intérêt étudié dans la section 4.2.1.
3. Mettre bout-à-bout les couples ayant un point d'intérêt en commun. En effet, un point d'intérêt d'une image donnée peut être apparié à un point de l'image précédente et à un point de l'image suivante et ainsi de suite. Nous définissions l'ensemble de ces points appariés comme étant la trajectoire temporelle d'un point d'intérêt.

Dans la suite du manuscrit, le terme « trajectoire » est un raccourci pour exprimer l'idée de « trajectoire d'un point d'intérêt ». La figure 4.8 illustre les trajectoires construites et superposées à l'image 21 de la séquence vidéo

Driving. Nous pouvons y voir que le mouvement apparent des trajectoires représente fidèlement le mouvement de tous les objets visibles ce qui justifie l'utilisation de trajectoires de points d'intérêt pour estimer le mouvement des objets. Le nombre de trajectoires extraites dans la vidéo est noté N_T.

De par sa construction, une trajectoire est, en général, définie sur un sous-

FIGURE 4.8 – *Trajectoires temporelles des points d'intérêt extraits à l'image I^{21}. Les croix rouges représentent la position de chaque trajectoire sur l'image I^{21}. La ligne noire passant par croix rouge représente la trajectoire du point d'intérêt considéré. Pour clarifier l'affichage et la compréhension, nous affichons la trajectoire de chaque point sur une fenêtre temporelle de dix images centrée sur l'image I_{21}.*

ensemble d'images de la séquence vidéo. Une trajectoire T_i, $i \in [1, N_T]$ étant l'indice de la trajectoire, définie sur les images $[I^{\text{début}_{T_i}}, I^{\text{fin}_{T_i}}]$ est un ensemble de points d'intérêt, ou plutôt un ensemble de positions à différents instants d'un même point d'intérêt. Nous définissons la notation suivante pour représenter la trajectoire T_i :

$$T_i = \{\tau_i^{\text{début}_{T_i}}, \cdots, \tau_i^{\text{fin}_{T_i}}\} \tag{4.12}$$

Dans la suite, nous considérerons qu'une trajectoire est définie sur toute la séquence vidéo. Ce changement minime permet de grandement simplifier les notations tout en clarifiant l'explication de la méthode. Nous expliquerons au moment propice comment adapter la méthode proposée pour tenir compte des trajectoires définies sur un sous-ensemble d'images. Ainsi, une

trajectoire T_i est un ensemble de positions :

$$T_i = \{\tau_i^1, \cdots, \tau_i^{N_V}\} \tag{4.13}$$

Dans la section 4.2.1, nous avons vu qu'il fallait sélectionner les couples de points d'intérêt définis à l'intérieur du contour de référence C^t. En effet, tous les couples construits ne représentent pas le mouvement de l'objet d'intérêt. De même, toutes les trajectoires construites ne représentent pas le mouvement de l'objet (voir figure 4.8). Une trajectoires T_i est dite caractéristique si elle est définie dans les contours $C^1, \cdots, C^t$. De manière plus formelle, une trajectoires T_i est caractéristique si et seulement si

$$\forall j \in [1, t],\ \tau_i^j \in \Omega_{C^j}. \tag{4.14}$$

Dans le cas où une trajectoire est définie sur un sous-ensemble d'image de la séquence, nous ajoutons la contrainte que pour être caractéristique, une trajectoire doit être définie au moins sur les images I^t et I^{t+1}. Dans le cas contraire, cette caractéristique n'apporterait aucune information quand au mouvement de l'objet sur le GOP.
La figure 4.7 illustre la sélection des trajectoires caractéristiques. Sur cette figure, nous pouvons comprendre l'intérêt de l'utilisation de ces trajectoires pour estimer le mouvement de l'objet d'intérêt. En effet, nous pouvons y voir qu'une trajectoire coupe le domaine défini par le contour C^t. Cette trajectoire n'est alors pas considérée comme caractéristique du fait de son passé. Les trajectoires caractéristiques (au nombre de N_T) seront notées T_i.

Nous disposons d'un ensemble de trajectoires caractéristiques T_i. La cohérence temporelle intrinsèque des trajectoires permet de supposer qu'une trajectoire caractéristique (définie à l'intérieur des contours $C^1, \cdots, C^t$) sera définie dans $C^{t+1}, \cdots, C^{N_V}$. Cette hypothèse permet d'extraire, à partir des trajectoires caractéristiques, des informations relatives au mouvement de l'objet sur l'ensemble des images du GOP $I^t, I^{t+1}, \cdots, I^{t+N_{GOP}-1}$. Ces informations sont formées d'un ensemble de couples de points d'intérêt. Pour T_i une trajectoire caractéristique donnée, les couples de points d'intérêt suivants sont extraits :

$$\{\tau_i^j, \tau_i^{j+1}\},\ \forall j \in [t, N_{GOP} - 1] \tag{4.15}$$

Naturellement, dans le cas où une trajectoire est définie sur un sous-ensemble d'image de la séquence, les couples sont extraits dans l'intersection temporelle des images du GOP et des images où la trajectoire est définie. Ces couples sont alors utilisés pour estimer le mouvement de l'objet sur le GOP.

4.3.2 Estimation du mouvement

Soit M^t, la matrice affine à six paramètres décrivant le mouvement du contour C^t sur l'ensemble des images $\{I^t, I^{t+1}, \cdots, I^{t+N_{GOP}-1}\}$. Chaque couple

de points extrait des trajectoires caractéristiques fournit une information relative au mouvement M^t de l'objet sur l'ensemble du GOP. Ce mouvement est estimé par la minimisation d'une fonctionnelle, comme vu dans la partie 4.2.2 :

$$\widehat{M}^t = \arg\min_M \sum_{i=1}^{N_T} \sum_{j=t}^{t+N_{GOP}-2} f(\|\tau_i^{j+1} - M\tau_i^j\|) \quad (4.16)$$

$$= \arg\min_M \sum_{i=1}^{N_T} \sum_{j=t}^{t+N_{GOP}-2} f(\|\epsilon_{i,M}^j\|) \quad (4.17)$$

où f est la fonction coût valeur absolue $f(x) = |x|$ et $\|.\|$ la norme Euclidienne. La minimisation de cette fonctionnelle est assurée par la méthode du simplexe. Insistons sur le fait que le mouvement estimé par l'équation (4.17) est un mouvement unique et global sur tout le GOP. Il va donc être appliqué successivement au contour C^t jusqu'au contour $C^{t+N_{GOP}-2}$:

$$c_i^{j+1} = \widehat{M}^t.c_i^j, \ \forall i \in [1, N_C] \ \forall j \in [t, t + N_{GOP} - 2] \quad (4.18)$$

La suite de la méthode de suivi consiste à répéter l'ensemble des opérations (sélection des trajectoires caractéristiques, estimation du mouvement et application du mouvement) sur un GOP composé des images $\{I^{t+N_{GOP}-1}, \cdots, I^{t+2N_{GOP}-2}\}$ de manière à ce que le premier contour du GOP, c'est-à-dire le contour $C^{t+N_{GOP}-1}$, soit connu.

4.3.3 Résultats sur la séquence Carmap

Avec l'approche utilisant seulement deux images, nous avons obtenu de très bons résultats du fait du nombre important de couples de points d'intérêt extraits. La figure 4.9 représente l'erreur de suivi en fonction de la taille du GOP. Nous pouvons y voir que cette erreur augmente avec la taille du GOP. Le suivi est de moins bonne qualité avec l'utilisation d'un GOP que sans (voir figure 4.10). L'utilisation d'un GOP fixe induit une saccade dans le mouvement (mouvement constant par morceaux). Pour cet exemple, l'utilisation d'un GOP fixe n'est pas une bonne solution et l'utilisation de seulement 2 images est particulièrement efficace. En effet, le mouvement réel de l'objet n'est pas constant sur un GOP.

4.3.4 Résultats sur la séquence Arménie

Nous avons vu à la section précédente que le suivi utilisant seulement deux images pour estimer le mouvement de la voiture sur la vidéo *Arménie* donnait de mauvais résultats. Ceci est dû au trop faible nombre de couples de points extraits. La présente approche, utilisant des GOP, offre un suivi de bien meilleure qualité. En effet, nous pouvons voir sur la figure 4.12 que la

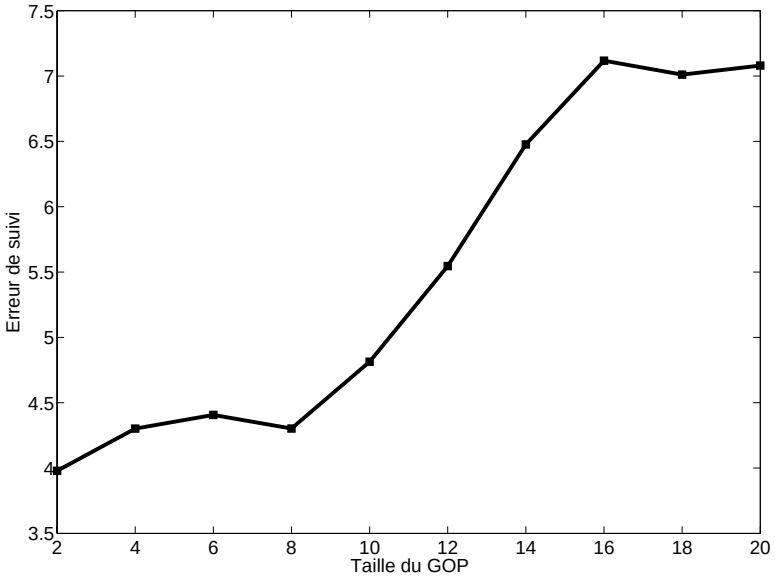


FIGURE 4.9 – *Évolution de l'erreur de suivi (pourcentage de pixels mal-classés) en fonction de la taille du GOP pour la séquence Carmap*

voiture est correctement suivie en dépit de l'occultation (GOP 14 images). Ce résultat se confirme sur la figure 4.11. Cette figure indique l'évolution de l'erreur de suivi en fonction de la taille du GOP utilisé. Un GOP de 2 images correspond à la première approche. Nous pouvons voir sur cette figure que l'erreur décroît fortement dès l'utilisation d'un GOP de 4 images. De manière générale, l'erreur décroît avec la taille du GOP, le mouvement étant quasiment constant sur toute la vidéo. Pour un GOP supérieur à 40 images, l'erreur est constante car la vidéo contient 40 images. Enfin, l'évolution de l'erreur est non régulière. Il est ainsi difficile de trouver la bonne taille de GOP à estimer car nous voyons qu'entre un GOP de 14 images et un GOP de 20 images, l'erreur est quasiment doublée. Nous voyons dans la section suivante une nouvelle approche qui permet de résoudre ce problème.

§ 4.4 ESTIMATION D'UN MOUVEMENT LOCAL SUR UN GOP GLISSANT

L'estimation du mouvement sur un GOP permet de sensiblement augmenter le nombre d'observations (couples de points d'intérêt) induisant de fait une estimation des paramètres plus précise et plus robuste aux données aberrantes. Cependant, le fait d'appliquer un mouvement identique sur un ensemble d'images a deux inconvénients.
Le premier inconvénient vient du fait que la taille du GOP doit être adap-

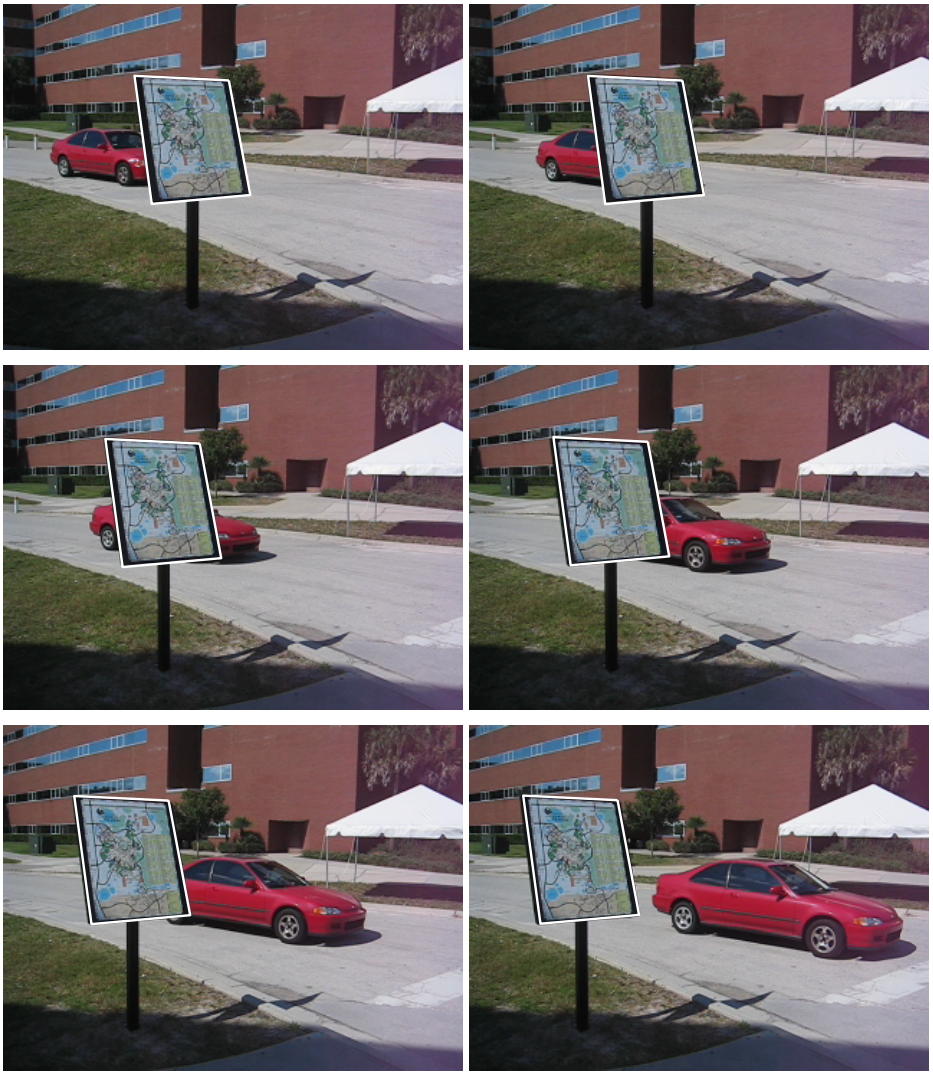


FIGURE 4.10 – *Résultat de suivi pour les images I^1, I^8, I^{15}, I^{22}, I^{29}, et I^{36} (respectivement de gauche à droite et de haut en bas) de la vidéo Carmap. La méthode utilise un GOP de 6 images. L'erreur de suivi est de 4.4% de pixels mal classés.*

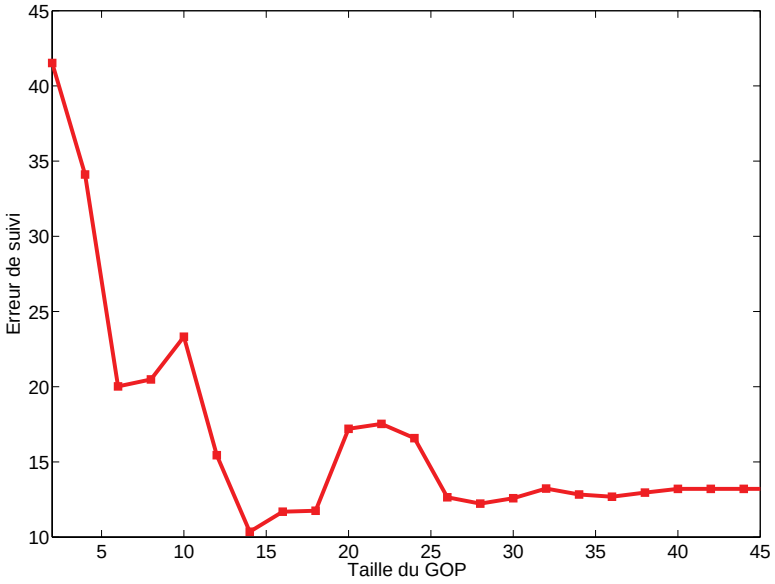


FIGURE 4.11 – *Évolution de l'erreur de suivi (pourcentage de pixels mal-classés) en fonction de la taille du GOP pour la séquence Arménie*

tée au mouvement de l'objet, chose que nous avons déjà évoqué. Certains objets, comme par exemple une balle de tennis, ont en effet un mouvement quasiment constant sur toute la séquence vidéo à l'exception de certaines images où le mouvement s'inverse brutalement. Pour de tels cas particuliers, à moins de ne considérer un GOP de seulement deux images (ce qui reviendrait à la première approche et donc non envisageable), il faudrait préalablement étudier les trajectoires caractéristiques pour déduire une taille de GOP adaptée.

Le second inconvénient de l'approche précédente est lié au système visuel humain. L'oeil humain est extrêmement sensible aux mouvements saccadés et aux mouvements périodiques. Le mouvement estimé $\widehat{M}^t$ par l'approche précédente est identique pour chaque image d'un GOP donné. Il en résulte que le mouvement est constant par morceau, c'est-à-dire qu'il est constant sur N_{GOP} images puis qu'il change brusquement entre deux GOP. Ce comportement, difficile à représenter sur le papier, est extrêmement gênant visuellement. Le suivi apparaît alors comme non naturel même si ce dernier est relativement correct.

En résumé, il apparaît que le mouvement doit être estimé sur un GOP pour des questions de précision et de robustesse aux données aberrantes. Cependant, il ne doit pas être appliqué à toutes les images du GOP tel que décrit par l'approche précédente. Nous proposons ici d'utiliser les avantages des deux premières approches. Le mouvement entre les images I^t et I^{t+1} est calculé à partir d'informations extraites de trajectoires caractéristiques et

FIGURE 4.12 – *Résultat de suivi pour les images I^1, I^8, I^{16}, I^{24}, I^{32}, et I^{40} (respectivement de gauche à droite et de haut en bas) de la vidéo Arménie. La méthode utilise un GOP de* 14 *images. L'erreur de suivi est de* 10% *de pixels mal classés.*

dans un GOP centré sur les images $\{I^t, I^{t+1}\}$ ce qui permet de considérer des informations relatives au mouvement dans le passé et dans l'avenir. Ainsi, le mouvement estimé est précis et robuste aux données aberrantes tout en garantissant une fluidité du suivi dû à l'estimation d'un mouvement par image. La figure 4.13 présente cette approche. Le mouvement est estimé sur un GOP entouré en bleu. Le mouvement est appliqué à C^t pour en déduire C^{t+1} (flèche rouge). Nous parlons alors de GOP glissant car, à chaque itération, le GOP est décalé d'une image.
Les deux approches précédentes étaient présentées sous forme de parties

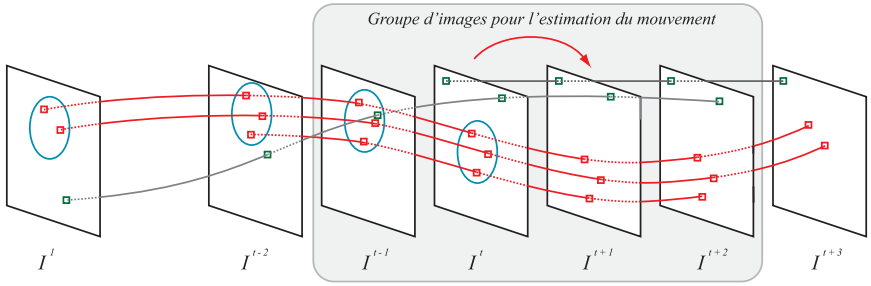


FIGURE 4.13 – *Illustration de trajectoires temporelles de points d'intérêt. Les contours $C^1, \cdots, C^t$ sont connus et représentés par une ellipse bleue. Les trajectoires caractéristiques sont indiquées en rouge. Le GOP utilisé pour estimer le mouvement est centré sur les images $\{I^t, I^{t+1}\}$ et est entouré en bleu sur la figure. Le mouvement estimé est alors appliqué au contour C^t pour en déduire le contour C^{t+1}. Á l'itération suivante, le GOP sera décalé d'une image et sera donc centré sur les images $\{I^{t+1}, I^{t+2}\}$ d'où l'appellation de « GOP glissant ».*

qui s'enchaînaient pour décrire deux méthodes de suivi dans leur globalité. Nous présentons ici une nouvelle approche sous une forme légèrement différente. En effet, les différentes étapes (construction et sélection des trajectoires caractéristiques, extraction de couples de points d'intérêt, estimation du mouvement, et application du mouvement au contour de référence) sont des étapes simples et déjà présentées. Nous allons donc les présenter succinctement dans le nouveau contexte. Nous présentons la nouvelle approche en deux parties : une méthode simple utilisant un GOP glissant et une méthode plus avancée utilisant un GOP glissant et des pondérations spatio-temporelles.

4.4.1 Pondération spatio-temporelle

La première étape consiste à créer les trajectoires de points d'intérêt et à sélectionner les trajectoires caractéristiques. Nous disposons de N_T trajectoires caractéristiques $\{T_i\}_{i\in[1,N_T]}$. L'hypothèse faite dans la partie 4.3.1 reste ici valable : une trajectoire définie à l'intérieur des contours $C^1, \cdots, C^t$, sera définie dans $C^{t+1}, \cdots, C^{N_V}$. Comme dans l'approche précédente, nous

allons extraire nos couples de points sur un GOP. La différence vient du fait que ce GOP de N_{GOP} images est centré sur les images $\{I^t, I^{t+1}\}$. Il est donc composé des images $I^{t-g+1}, \cdots, I^t, I^{t+1}, \cdots, I^{t+g}$ avec $g = \frac{N_{GOP}}{2}$ la demi taille du GOP. La taille du GOP N_{GOP} est nécessairement paire (d'où g entier) car le GOP est centré sur les images $\{I^t, I^{t+1}\}$. Pour T_i une trajectoire caractéristique donnée, les couples de points d'intérêt suivants sont extraits :

$$\{\tau_i^j, \tau_i^{j+1}\},\ \forall j \in [t-g-1, t+g-1] \tag{4.19}$$

Pour une trajectoire donnée, $N_{GOP} - 1$ couples de points d'intérêt seront extraits. Dans le cas où cette trajectoire est définie sur un sous-ensemble d'images de la séquence, les couples sont extraits dans l'intersection temporelle des images du GOP et des images où la trajectoire est définie. L'étape suivante est l'estimation du mouvement M^t à partir des couples de points d'intérêt. De manière semblable à ce que nous avons présenté auparavant, le mouvement estimé $\widehat{M}^t$ est solution d'un M-estimateur :

$$\widehat{M}^t = \arg\min_M \sum_{i=1}^{N_T} \sum_{j=t-g-1}^{t+g-2} f(\|\tau_i^{j+1} - M\tau_i^j\|) \tag{4.20}$$

$$= \arg\min_M \sum_{i=1}^{N_T} \sum_{j=t-g-1}^{t+g-2} f(\|\epsilon_{i,M}^j\|) \tag{4.21}$$

où f est la fonction coût valeur absolue $f(x) = |x|$ et $\|.\|$ la norme Euclidienne. La minimisation de cette fonctionnelle est assurée par la méthode du simplexe. Le mouvement estimé $\widehat{M}^t$ est alors appliqué aux points d'échantillonnage du contour de référence pour déduire le contour C^{t+1} :

$$C^{t+1} = \{c_1^{t+1}, c_2^{t+1}, \cdots, c_{N_C}^{t+1}\} \tag{4.22}$$

tel que

$$c_i^{t+1} = \widehat{M}^t.c_i^t,\ \forall i \in [1, N_C]. \tag{4.23}$$

Une fois que le contour C^{t+1} est obtenu, la méthode de suivi est recommencée en centrant maintenant le GOP sur les images $\{I^{t+1}, I^{t+2}\}$ d'où l'appellation de GOP glissant.
L'approche que nous venons de décrire permet de réaliser une estimation robuste et précise des paramètres du mouvement de l'objet entre les images I^t et I^{t+1}. Le suivi de l'objet est fluide en comparaison de l'approche présentée dans la partie 4.3 ce qui représente un réel gain visuel.

Le reproche que nous pouvons faire à l'approche que nous venons de décrire est relatif à l'utilisation des couples de points d'intérêt. En effet, tous les couples sont utilisés de la même manière lors de l'estimation des paramètres présentée dans l'équation (4.21). Ceci est une conséquence directe sur l'hypothèse faite à propos de la constance du mouvement sur un GOP. Nous savons que cette hypothèse est vraie sur des GOP de petite taille et

devient fausse sur des GOP de grande taille. Les couples de points d'intérêt $\{\tau_i^t, \tau_i^{t+1}\}$ vont bien représenter le mouvement présent entre les images I^t et I^{t+1} car ils sont effectivement extraits dans ce couple d'images. En revanche, les couples extraits à l'extrémité du GOP (et donc loin des images I^t et I^{t+1}) sur les images I^{t-g-1} et I^{t-g} vont beaucoup moins bien représenter le mouvement présent entre les images I^t et I^{t+1}. Nous proposons de pondérer l'influence des couples de points d'intérêt en fonction de leur proximité aux images I^t et I^{t+1}. Cette pondération intervient lors de l'estimation des paramètres :

$$\widehat{M}^t = \arg\min_M \sum_{i=1}^{N_T} \sum_{j=t-g-1}^{t+g-2} \alpha^{t,j} f(\|\epsilon_{i,M}^j\|) \tag{4.24}$$

où $\alpha^{t,j}$ est la pondération temporelle du couple de points d'intérêt $\{\tau_i^j, \tau_i^{j+1}\}$ par rapport à l'image I^t :

$$\alpha^{t,j} = \varphi(t-j) \tag{4.25}$$

La valeur $|t-j|$ représente la distance temporelle entre le couple de points d'intérêt $\{\tau_i^j, \tau_i^{j+1}\}$ et le couple d'images $\{I^t, I^{t+1}\}$. La fonction φ permet de donner une pondération égale à 1 pour un couple $\{\tau_i^t, \tau_i^{t+1}\}$ et une pondération proche de 0 pour un couple éloigné. La décroissance de la pondération avec l'éloignement temporel est assurée par la forme de la fonction φ. Nous considérons que la fonction φ a les propriétés suivantes :

- $\varphi : \mathbb{R} \mapsto \mathbb{R}^+$,
- φ est continûment dérivable,
- φ est une fonction paire,
- $\varphi(0) = 1$,
- $\lim_{t\to\infty} \varphi(t) = 0$.

Les propriétés de φ ne sont cependant pas fixées et une étude complète serait à envisager pour améliorer les résultats. Cependant, selon notre expérience, la fonction gaussienne centrée en zéro donne les meilleurs résultats :

$$\varphi(x) = e^{-\frac{x^2}{2\sigma_{\text{temp}}}} \tag{4.26}$$

avec σ_{temp} l'écart type de la gaussienne. Une gaussienne telle que nous l'avons écrite est considérée négligeable en dehors de l'intervalle $[-3\sigma_{\text{temp}}, 3\sigma_{\text{temp}}]$. L'écart type que nous devons définir dépend naturellement de la taille du GOP N_{GOP} ou plutôt de sa demi taille $g = \frac{N_{GOP}}{2}$. Sachant que $t-j$ est à valeur dans $[-(g-1), g-1]$, nous posons

$$\sigma_{\text{temp}} = \frac{g-1}{3}. \tag{4.27}$$

L'estimation ainsi réalisée est considérée comme spatialement globale et temporellement locale. En effet, l'ajout d'une pondération temporelle ajoute

un paramètre local dans l'estimation des paramètres. L'utilisation d'un GOP couplée à la pondération temporelle des couples de points d'intérêt permet de relâcher légèrement l'hypothèse de constance de mouvement sur tout le GOP. Le nombre de couples reste inchangé mais l'estimation des paramètres est plus précise car elle favorise les couples de points d'intérêt ayant une forte probabilité de bien représenter le mouvement du contour C^t entre les images I^t et I^{t+1}.

L'estimation du mouvement qui a été présentée jusqu'ici est une estimation de mouvement global pour tout l'objet, c'est-à-dire que le mouvement estimé $\widehat{M}^t$ est répercuté à l'ensemble des N_C points d'échantillonnage du contour de référence. Cette approche est très largement répandue dans la communauté étudiant les problèmes de « tracking ». En effet, pour certaines applications, le suivi d'un objet ne nécessite pas de définir avec précision le contour de l'objet d'intérêt, seule la position étant importante pour pouvoir identifier un objet (exemple : vidéo surveillance). Pour de telles applications, l'approche que nous venons de décrire fournit de très bons résultats. En revanche, pour d'autres applications où le contour doit être connu précisément, cette approche n'est pas suffisante sauf dans le cas où le mouvement de l'objet est effectivement un mouvement affine. Nous proposons d'améliorer l'approche précédente de manière à tenir compte des déformations locales du contour.
Nous utilisons la figure 4.14 pour présenter l'approche améliorée. Soit C^t

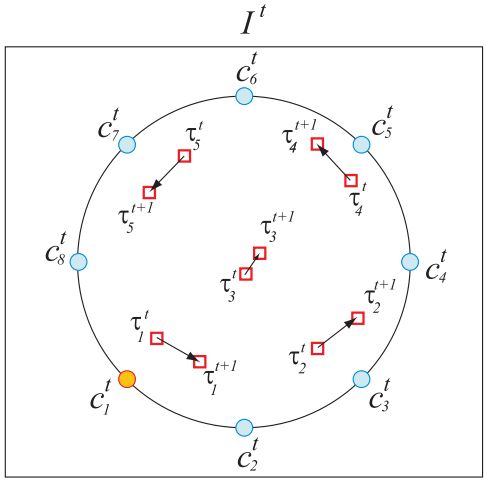


FIGURE 4.14 – *Illustration du contour C^t sur l'image I^t. Le mouvement des points d'intérêt (carrés rouges) permet d'estimer le mouvement du contour entre les images I^t et I^{t+1}.*

le contour défini sur l'image I^t et composé de six points d'échantillonnage $C^t = \{c_1^t, \cdots, c_6^t\}$. Pour des raisons de lisibilité, nous présentons sur la fi-

gure uniquement les couples de points d'intérêt entre les images I^t et I^{t+1}. Nous voulons calculer le mouvement local (en espace) du contour C^t pour en déduire le contour C^{t+1}. Ainsi, nous estimons un mouvement par point d'échantillonnage du contour. Nous voyons sur la figure cinq couples de points d'intérêt $\{\tau_i^t, \tau_i^{t+1}\}$. Si nous considérons le point d'échantillonnage c_1^t (indiqué en orange sur la figure), il semble évident que son mouvement sera mieux représenté par le couple $\{\tau_1^t, \tau_1^{t+1}\}$ (couple le plus proche spatialement de c_1^t) que par le couple $\{\tau_4^t, \tau_4^{t+1}\}$ (couple le plus éloigné spatialement de c_1^t). Il n'est pas possible de prouver cette propriété mathématiquement parlant si l'on considère que certains couples peuvent être aberrants (mauvais appariements). Cette propriété est cependant raisonnable pour la plupart des objets et des vidéos. L'idée de notre approche est de favoriser les couples de points d'intérêt spatialement proches du point d'échantillonnage pour lequel nous cherchons à estimer le mouvement. Nous utilisons une pondération spatiale dépendant de la position du point d'échantillonnage et du couple de points d'intérêt considérés.
Soit c_k^t le point d'échantillonnage dont on veut estimer le mouvement. Le mouvement estimé $\widehat{M_k^t}$ du point c_k^t est solution du M-estimateur suivant :

$$\widehat{M}_k^t = \arg\min_M \sum_{i=1}^{N_T} \sum_{j=t-g-1}^{t+g-1} \alpha^{t,j} \beta_{k,i}^t f(\|\epsilon_{i,M}^j\|) \tag{4.28}$$

où $\beta_{k,i}^t$ est la pondération spatiale du couple de points d'intérêt $\{\tau_i^j, \tau_i^{j+1}\}$ relativement au point d'échantillonnage c_k^t :

$$\beta_{k,i}^t \quad = \quad \phi(\mathcal{D}(c_k^t, \{\tau_i^j, \tau_i^j\})) \tag{4.29}$$

où ϕ est une fonction de pondération similaire à celle utilisée pour le calcul de $\alpha^{t,j}$. $\mathcal{D}(c_k^t, \{\tau_i^j, \tau_i^j\})$ représente la distance entre le point d'échantillonnage c_k^t et le couple de points d'intérêt $\{\tau_i^j, \tau_i^j\}$. Cette distance, identique pour tous les couples extraits d'une même trajectoire T_i, ne dépend que de la distance entre le point d'échantillonnage c_k^t et la position de la trajectoire sur l'image I^t :

$$\mathcal{D}(c_k^t, \{\tau_i^j, \tau_i^j\}) = d(c_k^t, \tau_i^t),\ \forall j \tag{4.30}$$

où d est une distance classique en deux dimensions, typiquement une distance euclidienne. Cette distance donne souvent de bons résultats car les objets sont généralement relativement convexes. Pour des objets non convexes dont les déformations locales sont importantes, une distance euclidienne n'est pas suffisante. Une distance géodésique serait préférable. Nous utilisons une fonction gaussienne en lieu et place de la fonction ϕ :

$$\phi(x) = e^{-\frac{x^2}{2\sigma_{\text{spat}}}} \tag{4.31}$$

L'écart-type σ_{spat} dépend de la taille de l'objet. Si la distance maximale possible entre un point d'échantillonnage et un couple de points d'intérêt est d_{MAX}, nous fixons

$$\sigma_{\text{spat}} = \frac{d_{MAX}}{3} \tag{4.32}$$

Le mouvement local (en temps et en espace) de chaque point d'échantillonnage du contour C^t est ainsi estimé. Le contour C^{t+1} est déduit de C^t en appliquant les N_C mouvements estimés respectivement aux points d'échantillonnage de C^t :

$$C^{t+1} = \{c_1^{t+1}, c_2^{t+1}, \cdots, c_{N_C}^{t+1}\} \tag{4.33}$$

tel que

$$c_i^{t+1} = \widehat{M}_i^t.c_i^t, \ \forall i \in [1, N_C]. \tag{4.34}$$

4.4.2 Résultats sur la séquence Carmap

La figure 4.15 présente l'évolution de l'erreur en fonction de la taille du GOP pour la méthode utilisant un GOP glissant avec pondération spatio-temporelle. L'erreur de suivi diminue avec la taille du GOP, le minimum étant atteint pour un GOP de 6 images. La diminution est assez faible du fait de la qualité obtenue avec un GOP de 2 images. Cependant, contrairement à la méthode utilisant un GOP fixe, l'utilisation d'un GOP glissant et d'une pondération spatio-temporelle améliore le suivi. La figure 4.16 présente le résultat de suivi avec cette méthode et pour un GOP de 6 images.

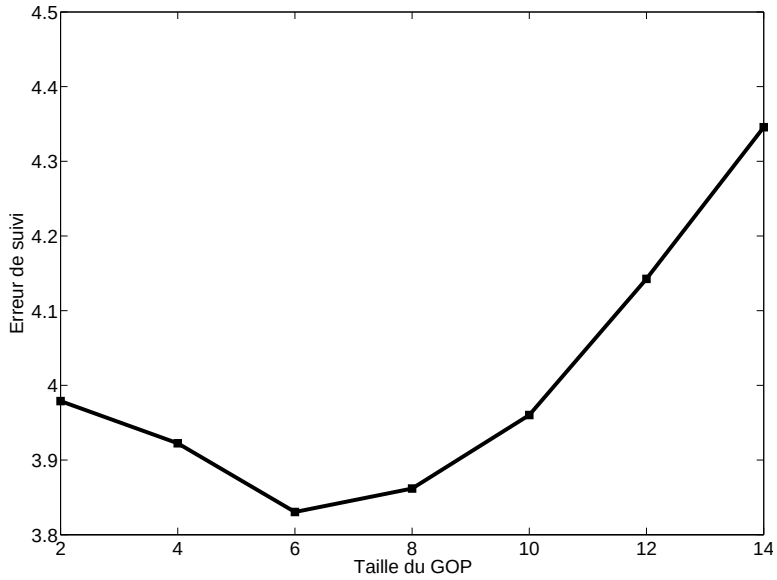


FIGURE 4.15 – *Évolution de l'erreur de suivi en fonction de la taille du GOP sur la séquence Carmap*

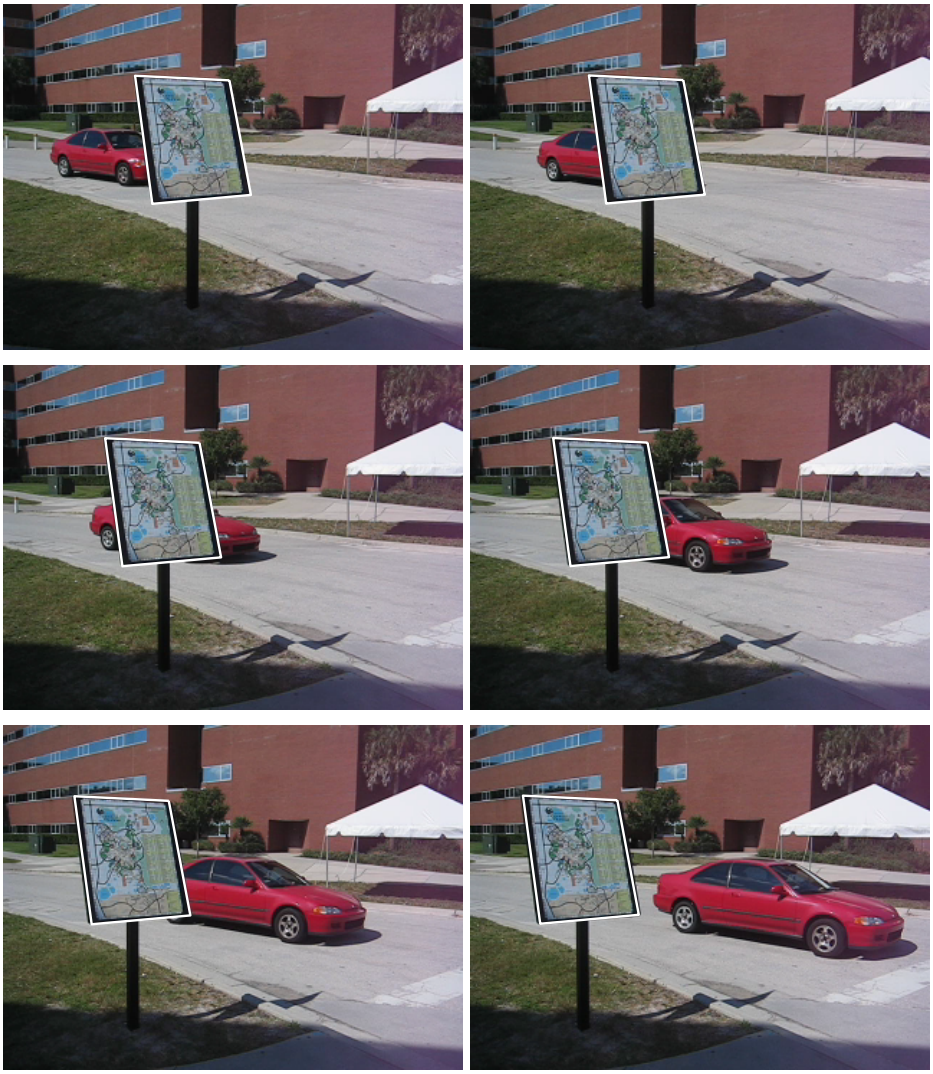


FIGURE 4.16 – *Résultat de suivi pour les images I^1, I^8, I^{15}, I^{22}, I^{29}, et I^{36} (respectivement de gauche à droite et de haut en bas) de la vidéo Carmap. La méthode utilise un GOP glissant de taille 6 images. L'erreur de suivi est de 3.8% de pixels mal classés.*

4.4.3 Résultats sur la séquence Arménie

La figure 4.17 présente l'évolution de l'erreur en fonction de la taille du GOP pour la méthode utilisant un GOP glissant avec pondération spatio-temporelle. L'erreur diminue également avec la taille du GOP et atteint un minimum global pour un GOP de 25 images. La courbe d'évolution de l'erreur est régulière et est constante pour une taille de GOP supérieur à 60 images. Cette taille dépasse celle de la vidéo car rappelons que les paires de points d'intérêt sont pondérées en utilisant une Gaussienne. La figure 4.18 présente le résultat de suivi avec cette méthode et pour un GOP de 25 images.
Nous avons vu que l'utilisation d'un GOP fixe dans le but d'augmenter le nombre de couples de points d'intérêt pose problème. En effet, soit nous remarquons une augmentation de l'erreur avec la taille du GOP, soit la courbe d'évolution de l'erreur est non régulière ce qui traduit que la méthode est très sensible à la taille du GOP. Nous remarquons que, pour les vidéos testées, l'utilisation d'un GOP glissant et d'une pondération spatio-temporelle permet d'améliorer la qualité du suivi. Cette méthode est plus fiable que celle fondée sur un GOP fixe. La taille de GOP optimale dépend naturellement du mouvement de l'objet. Il n'est pas évident de la calculer automatiquement et une étude plus approfondie serait à envisager. Cependant, nous avons remarqué au cours de nos différentes expérimentations qu'un GOP de taille 8 permettait de largement diminuer l'erreur de suivi.

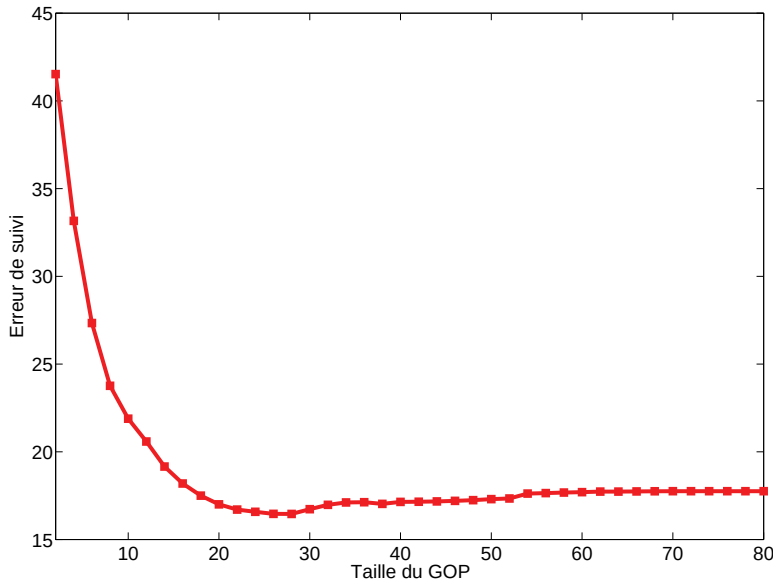


FIGURE 4.17 – *Évolution de l'erreur de suivi en fonction de la taille du GOP sur la séquence Arménie.*

FIGURE 4.18 – *Résultat de suivi pour les images I^1, I^8, I^{16}, I^{24}, I^{32}, et I^{40} (respectivement de gauche à droite et de haut en bas) de la vidéo Arménie. La méthode utilise un GOP glissant de taille* 25 *images. L'erreur de suivi est de* 17% *de pixels mal classés.*

§ 4.5 CONCLUSION

Le suivi d'objet dans une vidéo est une tâche complexe à automatiser. Dans ce chapitre, nous avons proposé une méthode de suivi fondée sur l'analyse de trajectoires de points d'intérêts dites caractéristiques. Ces trajectoires, représentant le mouvement de l'objet à suivre, sont utilisées pour extraire, sur un groupe d'images (GOP), un ensemble d'informations relatives au mouvement de l'objet. L'estimation du mouvement de cet objet est réalisée par la minimisation d'une fonctionnelle pondérée. Le contour initial de l'objet est édité manuellement sur la première image de la vidéo.
L'utilisation des points d'intérêt est très classique en vision par ordinateur comme par exemple en stéréovision. La fiabilité des points d'intérêt se retrouve dans les trajectoires temporelles de points d'intérêt. Ces trajectoires représentent fidèlement le mouvement de la plupart des objets contenus dans la vidéo. Les informations de mouvement extraites de celles-ci permettent une estimation précise du mouvement de l'objet d'intérêt.
Nous avons montré que l'utilisation d'un GOP permettait d'augmenter le nombre d'informations relatives au mouvement de l'objet ce qui permet d'améliorer la précision et la robustesse aux données aberrantes de l'estimation du mouvement. Cependant, nous avons évoqué le fait que l'estimation du mouvement par des GOP consécutifs engendrait une saccade visuelle gênante. De plus, la qualité du suivi est très sensible à la taille de GOP choisie. Nous avons alors proposé d'utiliser des GOP glissant. L'utilisation d'une pondération temporelle a pour effet d'adoucir l'hypothèse de constance de mouvement tandis que la pondération spatiale permet de tenir compte des déformations locales de l'objet.

D'après nos expérimentations, la méthode proposée permet de suivre précisément un objet d'intérêt en dépit d'occultations éventuelles. Cependant, la méthode étant fondée sur des trajectoires de points d'intérêt, elle est par conséquent totalement dépendante de la qualité de ces dernières. En effet, si l'objet est homogène ou très peu texturé, il est très possible qu'aucune ou très peu de trajectoires soient définies dans l'objet d'intérêt. L'estimation du mouvement est par conséquent non fiable ou tout simplement impossible. Considérons maintenant le cas où les trajectoires ne sont pas en accord avec le mouvement de l'objet. L'estimation du mouvement ne peut être correcte. Ce cas se présente par exemple si le contour de l'objet initialisé manuellement sur la première image de la vidéo est trop approximatif et qu'il contient de nombreux points d'intérêt correspondant au fond ou à un autre objet en mouvement. De même, ce cas se présente si l'objet est occulté dès la première image, les trajectoires alors considérées comme caractéristiques peuvent représenter le mouvement de l'objet ou alors un mouvement totalement différent. Enfin, la dernière limitation

vient du suivi d'objets articulés. La pondération spatiale permet de prendre en compte de petites déformations. Pour des objets articulés, il faudrait une densité très importante de trajectoires ce qui n'est pas le cas en pratique. Le suivi d'objet articulés est donc très approximatif et la qualité est souvent éloignée de ce que l'on pourrait obtenir avec une méthode de contours actifs mieux adaptée à ce type de problème.

Malgré ces quelques limitations, il apparaît que la grande majorité des applications de suivi d'objet ne nécessite qu'un suivi global. La méthode de suivi proposée permet donc de suivre précisément la plupart des objets.

- CHAPITRE 5 -

SUIVI DE LA COURONNE DE L'OBJET

§ 5.1 INTRODUCTION

Le chapitre précédant propose une méthode de suivi d'objet fondé sur l'utilisation de trajectoires de points d'intérêt. Nous y avons vu que l'utilisation d'un GOP couplé à l'estimation locale, aussi bien en espace qu'en temps, permettait de suivre précisément les objets tout en étant robuste aux occultations partielles. Cette méthode a toutefois deux points faibles.
Le premier est inhérent à l'utilisation de trajectoires. Si aucun point d'intérêt n'est défini dans l'objet, aucune des trajectoires définies dans la vidéo n'est une trajectoire caractéristique, c'est-à-dire une trajectoire définissant le mouvement de l'objet. Il en résulte que le mouvement de l'objet ne peut être calculé. Ce cas peut se rencontrer par exemple avec des objets de couleur homogène. De même, si les trajectoires construites sont totalement aberrantes pour quelques raisons que ce soit, bien qu'utilisant des méthodes d'estimation robustes aux donnée aberrantes, le suivi ne peut être garanti. Enfin, si l'objet est occulté dès la première image ou si l'occultation dure trop longtemps, les trajectoires sélectionnées pour être des trajectoires caractéristiques sont certainement aberrantes ce qui compromet de fait le suivi de l'objet.
Le second point faible de la méthode précédente est lié à l'hypothèse selon laquelle le mouvement de l'objet peut être estimé à partir des trajectoires de points d'intérêt. Le suivi global est très performant et nous avons vu que nous pouvions tenir compte de déformations locales de l'objet par l'estimation d'un mouvement par point d'échantillonnage du contour. Cependant, la densité des trajectoires ne permet pas de prendre en compte des déformations trop complexes telles que rencontrées avec les objets articulés. Pour des applications classiques (vidéo-surveillance, inpainting, colorisation, etc.), la précision du suivi est suffisante. En revanche, pour des

applications nécessitant une localisation très précise du contour (compositing), la méthode fondée sur les trajectoires n'est alors pas suffisamment précise (*e.g. Video Cutout* [WBC+05]).
Soit V une vidéo de N_V images $\{I_1, I_2, \cdots, I_{N_V}\}$. Comme dans le chapitre précédant, le contour C^1 est défini manuellement. Nous considérons que les contours $C^2, \cdots, C^t$ sont d'ores et déjà calculés, le contour C^t étant appelé contour de référence. Le problème de suivi consiste à calculer les contours $C^{t+1}, \cdots, C^{N_V}$. Le contour C^j à l'image I^j est un ensemble de N_C points d'échantillonnage $C^j = \{c_1^j, c_2^j, \cdots, c_{N_C}^j\}$ reliés par une courbe.
Dans ce chapitre, nous proposons une nouvelle approche très différente fondée sur l'estimation du mouvement des points d'échantillonnage du contour par une méthode dite de *block-matching* [1] (appariement de blocs). Cette approche n'est en rien fondée sur l'utilisation de points d'intérêt ni sur leur trajectoire. La nouvelle approche est présentée en deux étapes. Lors de la première, nous présentons une approche utilisant une méthode de block-matching pour laquelle les pixels du fond sont partiellement masqués. La seconde étape consiste à améliorer notre approche en utilisant un critère de mise en correspondance non paramétrique et fondé sur l'estimation de l'entropie du résiduel. L'ensemble de l'étude sera réalisé sur deux séquences d'images synthétiques présentées sur la figure 5.1.

§ 5.2 APPROCHE CLASSIQUE

Étant donné le contour de référence C^t, le problème est de calculer le contour C^{t+1}. Le mouvement du contour de référence est estimé à partir de l'information contenue dans le voisinage du contour. Pour prendre en compte les déformations complexes du contour, le mouvement de chaque point d'échantillonnage est séparément estimé. Les points d'échantillonnage, alors déplacés de leur mouvement respectif, forment le contour C^{t+1}. Le mouvement des points d'échantillonnage est supposé être une translation. Cette hypothèse, couramment utilisée en estimation du mouvement, est une hypothèse raisonnable entre deux images successives d'une vidéo. Si le contour est discrétisé suffisamment finement, cette dernière ne contraint en rien le mouvement global du contour de l'objet. En particulier, l'objet peut tout à fait être articulé.

Le mouvement de chaque point d'échantillonnage est estimé par une méthode dite de « block-matching » [KLH+81]. Le block-matching est une technique massivement utilisée dans le domaine de la compression vidéo. Le principe général est d'exploiter les redondances temporelles existantes entre des images consécutives. Pour cela, l'image dite de référence est dé-

1. Le block-matching peut être très rapide par corrélation de phase (*cross power spectrum*).

coupée en blocs de taille identique (typiquement 8×8 ou 16×16 pixels), chaque bloc étant considéré comme un objet indépendant des autres. Le mouvement des pixels d'un bloc est considéré comme étant identique. De manière informelle, étant donné un bloc de l'image de référence, l'algorithme consiste à choisir le bloc dans l'image suivante (ou précédente dans le cas d'estimation de mouvement « backward ») qui minimise un critère de comparaison calculé entre ces deux blocs. L'opération est réitérée en choisissant un autre bloc jusqu'à ce que tous les blocs d'une zone déterminée, appelée « fenêtre de recherche », de l'image suivante aient été testés ou jusqu'à un critère d'arrêt arbitraire. Le bloc le plus semblable au sens du critère de comparaison est ainsi identifié dans l'image suivante. En analysant la position initiale du bloc sur l'image de référence par rapport à la position du bloc « optimal » dans l'image suivante, nous obtenons un vecteur de mouvement. L'utilisation d'une fenêtre de recherche permet de limiter le nombre de blocs testés dans l'image suivante, la taille de la fenêtre dépendant du déplacement des objets dans la vidéo. Généralement, une fenêtre de dimension 15×15 est utilisée pour des vidéos CIF (CIF = 352×288 pixels). Pour des applications de compression vidéo, les vecteurs mouvement calculés sont alors utilisés pour compenser en mouvement l'image de référence.

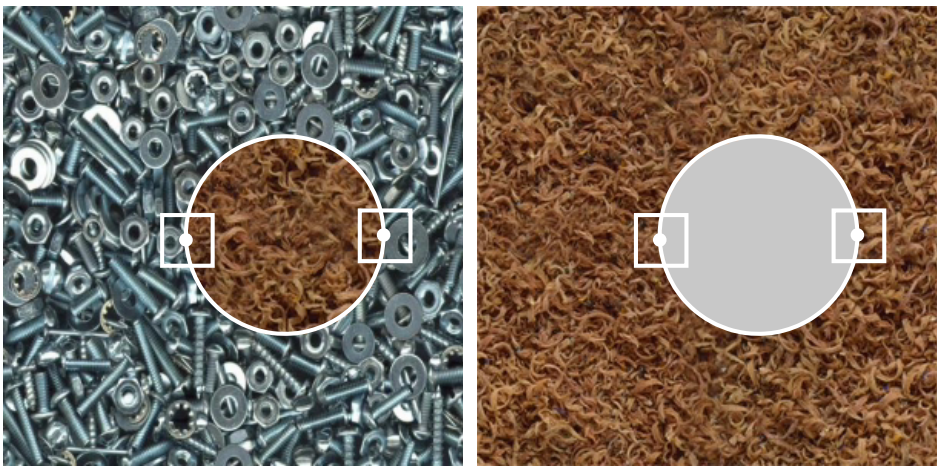
Nous adaptons alors la méthode de block-matching à notre application. Soit B_i le bloc carré centré sur le point d'échantillonnage c_i^t. Ce bloc contient les pixels de l'image C^t dans le voisinage carré de c_i^t. Le mouvement v_i du point c_i^t est déterminé par la position du bloc optimal dans l'image I^{t+1} relativement au critère de comparaison :

$$v_i = \arg\min_u \sum_{x \in B_i} \varphi(r(x,u)) \tag{5.1}$$

où x représente les coordonnées d'un pixel ($x \in \mathbb{R}^2$), φ une fonction positive à définir, et $r(x,u)$ est le résiduel défini par $r(x,u) = I^t(x) - I^{t+1}(x+u)$. La valeur u représente un vecteur de déplacement. La minimisation de l'équation (5.1) est réalisée dans une fenêtre de recherche exhaustive (tous les blocs de la fenêtre sont testés). Cette approche est lente du fait du nombre de blocs à tester. Il existe de nombreuses méthodes permettant d'accélérer la recherche du bloc optimal [LZL94, ZM00, ZLC02, LO04, Gha90]. Ces méthodes sont toutes fondées sur une hypothèse de décroissance monotone du critère de comparaison vers le bloc optimal. Parmi les méthodes existantes, nous utilisons le « diamond search » [ZM00] lorsque nous souhaitons accélérer le processus de minimisation.

De manière générale, les blocs peuvent contenir des pixels de différents objets. Il en résulte que le critère de comparaison utilisé doit être robuste en présence de données aberrantes. Cette condition exclut naturellement l'utilisation de la fonction carrée $\varphi(r) = r^2$. La fonction $\varphi(r)$ joue le même rôle que la fonction de coût vue au chapitre précédent. Nous utilisons donc la

fonction valeur absolue $\varphi(r) = |r|$ pour ses bonnes propriétés de robustesse aux données aberrantes. Notez toutefois que les fonctions définies dans le tableau 4.1 peuvent être utilisées à la place. Le critère de comparaison utilisant la valeur absolue sera nommé dans la suite SAD (pour « Sum of Absolute Differences », somme de la différence en valeur absolue en français). Enfin, notez que le critère de comparaison est minimal quand la similarité entre les deux blocs comparés est maximale.
Partant de la méthode classique de block-matching, nous allons dans cette partie améliorer pas à pas la méthode pour proposer au final notre méthode de suivi d'objet. Pour ce faire, l'évolution de la méthode sera justifiée sur deux vidéos synthétiques (voir figure 5.1). Chaque vidéo a pour dimension 300×300 pixels et représente un objet rond en translation horizontale de 4 pixels par dessus un fond texturé fixe. Les deux vidéos sont identiques sauf en ce qui concerne la texture de l'objet : l'objet est texturé sur la vidéo nommée V_{tex}, et homogène sur la vidéo V_{hom}. Notre étude se focalise sur deux blocs : B_g (bloc de gauche) and B_d (bloc de droite). Ces blocs, de taille 33×33 pixels et illustrés sur la figure 5.1, sont centrés sur deux points d'échantillonnage du contour. Notez que nous utilisons des objets ronds de manière à ce que la courbure du contour soit constante. Le but étant de garantir que les conditions de l'ensemble des blocs centrés sur les points d'échantillonnage soient identiques.



(a) Vidéo V_{tex} (b) Vidéo V_{hom}

FIGURE 5.1 – *Contour C^1 sur l'image I^1 pour les vidéos V_{tex} et V_{hom} (respectivement objet texturé et objet homogène). Les blocs B_g et B_d (respectivement bloc de gauche et bloc de droite) sont encadrés en blanc.*

§ 5.3 MASQUAGE PARTIEL DU FOND

Débutons notre étude avec la vidéo S_{tex} et utilisons la méthode de block-matching pour estimer le vecteur mouvement des blocs B_g et B_d. Étant donnée une fenêtre de recherche de 15×15 pixels, nous pouvons calculer la valeur du critère de comparaison SAD pour chacun des déplacements de la fenêtre. La figure 5.2 illustre la valeur de ce critère sur la fenêtre de recherche. Nous pouvons remarquer que ce critère admet un minimum local et un minimum global. Ces minima correspondent respectivement au mouvement de l'objet et au mouvement du fond. La figure 5.3 présente respectivement les profils des critères SAD pour les blocs B_g et B_d. Comme sur la figure 5.2, nous pouvons remarquer la présence de deux minima correspondant au mouvement de l'objet et au mouvement du fond. Dans la suite, nous utiliserons cette représentation pour illustrer notre raisonnement. La figure 5.3 nous apprend qu'en utilisant un algorithme de block-matching classique avec le critère SAD, le vecteur de mouvement associé au point d'échantillonnage du contour est égal au mouvement du fond, c'est-à-dire le vecteur nul sur cette séquence. Ceci n'est pas ce que nous désirons car nous voulons que le contour suive l'objet. En étudiant la figure 5.1, nous remarquons que les blocs B_g et B_d contiennent proportionnellement plus de pixels du fond que de pixels de l'objet. De manière plus générale, la proportion de pixels du fond et de pixels de l'objet dépend de la convexité locale de l'objet aux points d'échantillonnage du contour. Pour un objet convexe, ce qui est le cas sur les vidéos synthétiques, les pixels du fond sont toujours majoritaires en comparaison des pixels de l'objet. C'est de cette proportion que s'explique le résultat obtenu sur les figures 5.2 et 5.3. En effet, le but du block-matching est d'estimer un mouvement unique et global par bloc. Si les pixels du fond sont majoritaires, il est naturel de considérer que ce bloc appartient au fond plutôt qu'à l'objet. Attribuer à un tel bloc le mouvement du fond est par conséquent le comportement normal et attendu d'une méthode d'estimation du mouvement.

La présence d'un minimum local correspondant au mouvement de l'objet indique que l'utilisation d'une méthode de block-matching pour estimer le mouvement des points d'échantillonnage du contour est un choix cohérent. Cependant, cette méthode est en l'état inapte à estimer correctement le mouvement de l'objet. Il est nécessaire de diminuer l'influence des pixels du fond. Une méthode simple est de ne tenir compte que des pixels de l'objet. Pour ce faire, nous utilisons le domaine D^t dont le contour est égal à C^t. Lors de l'estimation de mouvement d'un bloc B_i par la méthode de block-matching, nous ne considérons que les pixels appartenant à $\Omega_i = B_i \cap D^t$. L'équation (5.1) se récrit alors :

$$v_i = \arg\min_u \sum_{x \in \Omega_i} \varphi(r(x,u)) \tag{5.2}$$

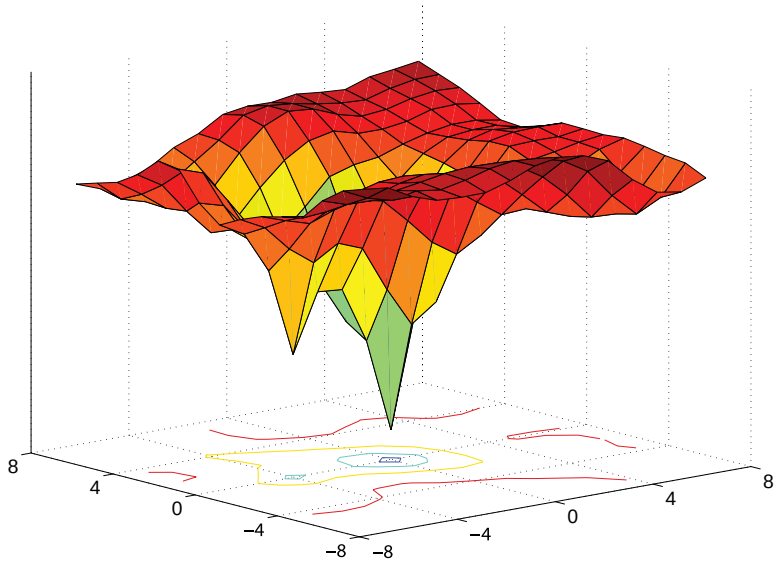


FIGURE 5.2 – *Critère de comparaison SAD sur la fenêtre de recherche de taille* 15×15 *pixels pour le bloc* B_g*. La vidéo utilisée est* V_{tex}*.*

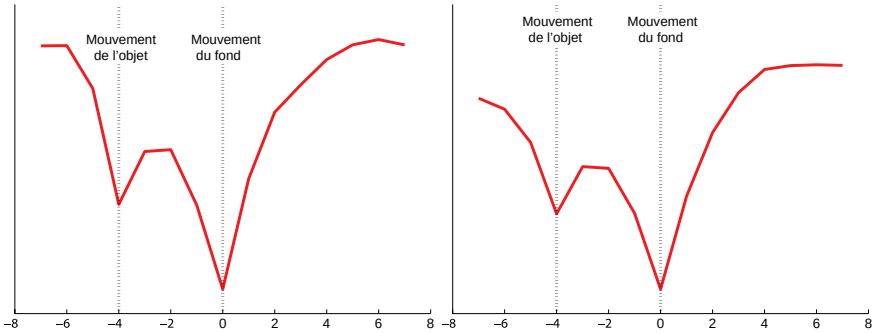


FIGURE 5.3 – *Profils 2D du critère SAD calculés pour les blocs* B_g *et* B_d *(respectivement à gauche et à droite) et avec la vidéo* V_{tex}*. Les profils passent par le minimum global et sont parallèles à l'axe horizontal.*

En appliquant cette équation pour estimer le mouvement des blocs B_g et B_d sur la vidéo V_{tex} (objet texturé), nous obtenons les critères (profils) présentés sur la figure 5.4. Nous pouvons y voir que le minimum du critère est obtenu pour un mouvement exactement égal à celui de l'objet. Étant donné qu'aucun pixel du fond n'est pris en compte dans le processus d'estimation de mouvement décrit dans l'équation (5.2), aucun minimum (local ou global) n'est détecté pour le mouvement du fond. Pour un tel objet, masquer les pixels du fond permet de résoudre le problème lié à la proportion des pixels du fond. Le suivi qui en résulte est très précis.

En appliquant l'équation (5.2) pour estimer le mouvement des blocs B_g et

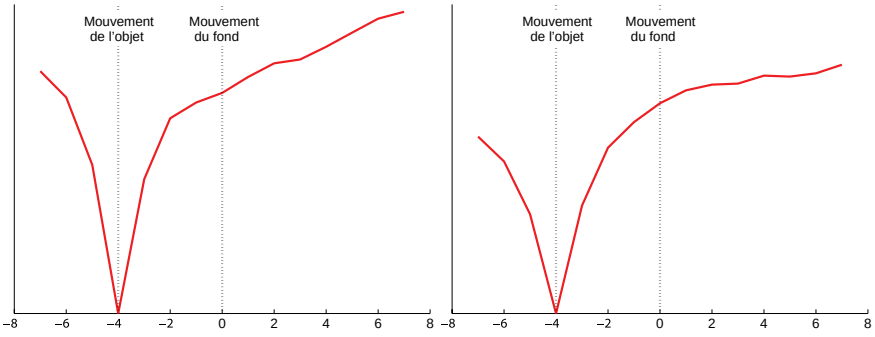


FIGURE 5.4 – *Profils 2D du critère calculés pour les blocs B_g et B_d (respectivement à gauche et à droite) et avec la vidéo V_{tex}. Les pixels du fond sont masqués en utilisant le domaine D^t. Les profils admettent un minimum global correspondant au mouvement de l'objet.*

B_d sur la vidéo V_{hom} (objet homogène), les critères obtenus sont très différents (voir figure 5.5). Le critère décroît de manière monotone vers un minimum localisé au niveau du mouvement de l'objet. Cependant, contrairement aux profils présentés dans la figure 5.4, le critère est plat et égal au minimum global pour les mouvements induisant un déplacement du bloc dans l'objet. Ce comportement est tout à fait logique. En effet, le bloc ne contenant que les pixels de l'objet, il est totalement homogène et est donc parfaitement similaire à tous les blocs homogènes contenus dans l'objet. La structure relative au bord de l'objet est perdue et la méthode d'estimation du mouvement ne permet pas d'estimer correctement le mouvement de l'objet.

Le problème qui se pose ici est l'estimation de mouvement d'objets homogènes, le cas des objets texturés étant résolu. Nous proposons d'utiliser l'information du bord de l'objet pour remédier à ce problème. Pour ce faire, l'utilisation du domaine D^t permettant de déterminer les pixels appartenant à l'objet est remplacée par l'utilisation du domaine $d_R(D^t)$, d_R étant l'opération de dilatation morphologique [Soi99] fondée sur l'utilisation d'un élément structurel circulaire de rayon R. Cette opération consiste

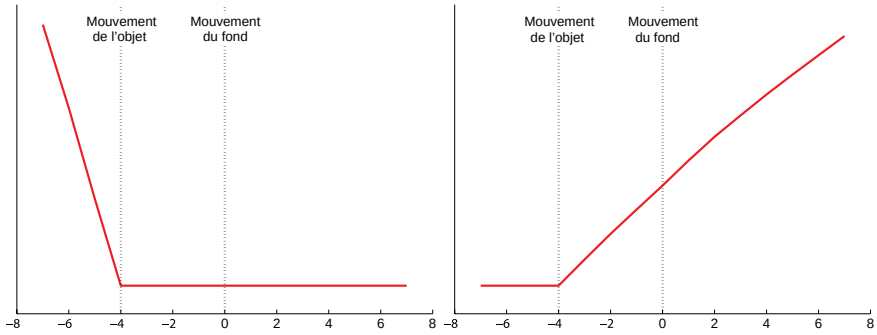


FIGURE 5.5 – *Profils 2D du critère calculés pour les blocs B_g et B_d (respectivement à gauche et à droite) et avec la vidéo V_{hom}. Les pixels du fond sont masqués en utilisant le domaine D^t.*

à ajouter une bande étroite de pixels du fond ajoutant ainsi la structure de bord manquant dans l'approche précédente. On parle alors de masquage partiel des pixels du fond. L'hypothèse sous-jacente est qu'une bande de pixels du fond est plus proche (en terme de similarité) de n'importe qu'elle partie du fond que de l'objet. Le mouvement est estimé comme suit :

$$v_i = \arg\min_u \sum_{x \in \tilde{\Omega}_i} \varphi(r(x,u)) \tag{5.3}$$

où $\tilde{\Omega}_i$ est la version dilatée de Ω_i donnée par la formule suivante :

$$\tilde{\Omega}_i = B_i \cap d_R(D^t). \tag{5.4}$$

La figure 5.6 présente les critères obtenus en utilisant un masquage partiel des pixels du fond. Différentes valeurs du rayon de dilatation sont utilisées. Notez qu'un rayon de 0 correspond à un masquage total des pixels du fond, et qu'un rayon supérieur à la longueur de la demi-diagonale (dans notre cas, la taille d'un bloc étant 33×33, $R > 23$ car $23 \approx 16\sqrt{2}$) correspond à ne masquer aucun pixel. La valeur du rayon de dilatation R permet de simuler les deux premières approches présentées (sans et avec masquage des pixels du fond). La figure 5.6 indique que l'utilisation d'une dilatation de rayon 2 ou 4 entraîne l'apparition d'un minimum global correspondant au mouvement de l'objet. Le choix du rayon de dilatation optimal est discuté en partie 5.4. En pratique, un rayon de 5 pixels permet de suivre précisément la plupart des objets. Au delà du minimum global, nous remarquons que le critère devient plat. La différence entre le mouvement de l'objet et le mouvement à partir duquel le critère devient plat est exactement égale au rayon de dilatation R utilisé. Ce phénomène est prévisible. En effet, en dilatant de R pixels, une bande de R de pixels du fond est ajouté au bloc. Dès lors que le centre du bloc a pénétré de R pixels dans l'objet, tous les pixels

considérés du bloc sont dans l'objet. Un plus ample mouvement dans l'objet ne pénalisera pas davantage le critère. Pour un rayon $R = 12$, le critère admet un maximum global correspondant au mouvement du fond. En effet, à partir de cette valeur, la proportion de pixels du fond devient importante sans dépasser celle des pixels de l'objet. La texture jouant un rôle important dans l'estimation du mouvement, le block-matching est faussé par la trop forte influence des pixels d'un fond très texturé. Nous remarquons toutefois que le minimum est plus clairement établit pour le bloc B_g. Cette propriété est la conséquence directe de l'occultation et de l'apparition de pixels du fond.

L'utilisation d'un masquage partiel pour un objet texturé ne pose évidem-

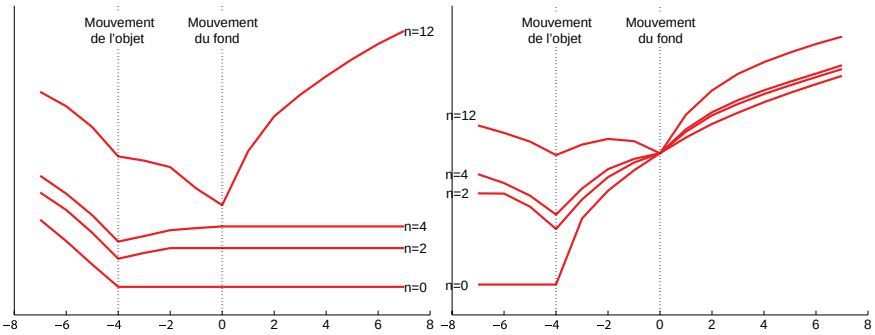


FIGURE 5.6 – *Profils 2D du critère calculés pour les blocs B_g et B_d (respectivement à gauche et à droite) et avec la vidéo V_{hom}. Les pixels du fond sont partiellement masqués. Pour chaque bloc, nous indiquons plusieurs profils correspondants aux différents rayons de dilatation utilisés. L'ajout d'une bande de pixels du fond permet d'estimer correctement le mouvement de l'objet bien qu'il soit homogène.*

ment aucun problème du fait de la majorité des pixels de l'objet par rapport à ceux du fond. Ainsi, nous avons maintenant la possibilité de suivre précisément un objet homogène ou texturé sur un fond texturé par l'utilisation d'une méthode de block-matching avec masquage partiel du fond. Le cas d'un objet (homogène ou texturé) sur un fond homogène est un cas plus simple que ceux que nous venons de tester. En effet, les pixels du fond n'ayant pas de correspondance exacte, seule l'information de bord (dans le cas d'un objet homogène) ou l'information de l'objet (dans le cas d'un objet texturé) peut permettre de suivre l'objet. Ces deux informations sont présentes lors du masquage partiel du fond.

§ 5.4 ESTIMATION DU MOUVEMENT

Nous avons présenté jusqu'ici une méthode de suivi fondée sur une méthode de block-matching. Les pixels du fond sont partiellement masqués

par l'utilisation du domaine associé au contour de référence et par l'emploi d'une dilatation morphologique. À ce stade, la question qui se pose est le choix du rayon de dilatation. Pour des blocs de tailles 33×33 pixels, nous savons que ce rayon doit être compris dans l'intervalle $]0, 23[$. Le choix de la dilatation dépend en réalité de la texture de l'objet, de la texture du fond, de la précision de la segmentation manuelle initiale, et de la netteté des contours. Il est en effet assez classique qu'un contour défini manuellement ne soit pas exactement sur le contour de l'objet et/ou que le contour de l'objet soit flou du fait du mouvement de l'objet (« motion blur »). Il peut en résulter qu'un objet détouré manuellement contienne déjà quelques pixels du fond auquel cas une dilatation moins importante est nécessaire. Pour un objet homogène, un rayon de dilatation trop faible risque de ne pas apporter l'information de bord suffisante pour estimer correctement le mouvement de l'objet. A contrario, un rayon trop grand risque d'ajouter un nombre trop important de pixels du fond. En conséquence, le rayon R doit être choisi dans un intervalle $[R_{\min}, R_{\max}]$ où, naturellement, $R_{\min} > 0$. Une première approche pour calculer un rayon de dilatation « optimal » serait d'analyser l'objet (courbure, texture) et la texture du fond. Un autre approche serait de choisir de manière heuristique ce rayon tout en élargissant au maximum possible l'intervalle $[R_{\min}, R_{\max}]$. Le critère de comparaison SAD, généralement utilisé en estimation robuste, permet d'assurer que le rayon maximal $R_{\max}$ ne soit pas trop petit. Cependant, nous proposons dans cette partie d'utiliser une approche non-paramétrique plus robuste aux données aberrantes que sont les pixels du fond.

La dilatation morphologique, utilisée pour pallier le possible manque de structure d'un bloc, ajoute au processus d'estimation du mouvement des pixels du fond alors considérés comme données aberrantes. La minimisation du critère de comparaison utilisé dans l'équation (5.3) fait implicitement une hypothèse paramétrique sur la distribution du résiduel $r(x, u)$ dans le sens où cette distribution est définie par un petit nombre de paramètres (par exemple la moyenne et la variance d'une distribution Gaussienne). Par exemple, la distribution est supposée Gaussienne pour $\varphi(x) = x^2$ et Laplacienne pour $\varphi(x) = |x|$. Cette hypothèse est fausse en général et, par conséquent, le mouvement du bloc peut être mal estimé. La figure 5.7 donne un agrandissement de la distribution estimée du résiduel obtenu dans le cas où le mouvement estimé correspond exactement au mouvement de l'objet. La vidéo utilisée pour calculer cette distribution est V_{tex}. Cette distribution n'est ni Gaussienne, ni Laplacienne. Notez que la distribution est estimée par un estimateur de densité à noyau présenté dans l'annexe A.2.2.

Nous proposons de relâcher l'hypothèse paramétrique en ajoutant lors de l'estimation du mouvement une dépendance à la distribution réelle du résiduel.Le critère de comparaison que nous proposons est l'estimation de

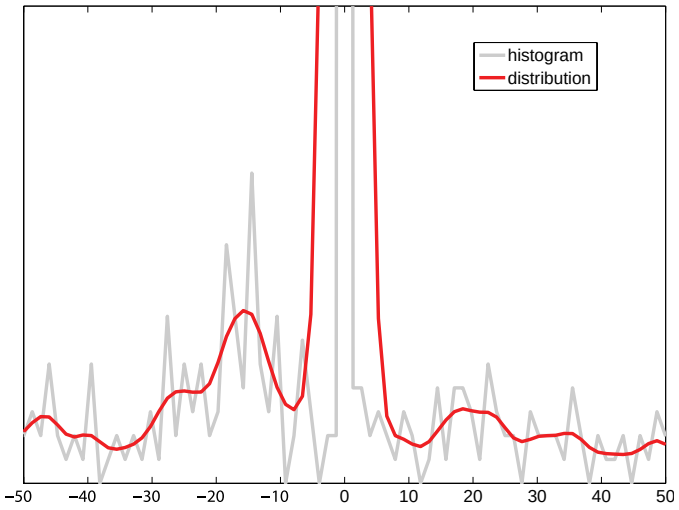


FIGURE 5.7 – *Agrandissement de la distribution du résiduel pour le bloc B_g et la vidéo V_{tex} obtenue dans le cas où le mouvement estimé correspond exactement au mouvement de l'objet. L'histogramme du résiduel est indiqué en gris tandis que la distribution estimée est tracée en rouge. La distribution n'est clairement pas paramétrique.*

l'entropie du résiduel calculé par l'approximation de Ahmad-Lin [AL76] :

$$v_i = \arg\min_u -\frac{1}{|\tilde{\Omega}_i|} \sum_{x \in \tilde{\Omega}_i} \log(\widehat{p}(r(x,u))) \tag{5.5}$$

où $\widehat{p}$ est la valeur de la distribution estimée par une méthode à noyau [Par62]. Cette méthode consiste à placer un noyau (par exemple un noyau gaussien) centré en chaque point de l'échantillon et à effectuer la somme de ces noyaux. Pour un noyau Gaussien, la distribution estimée $\widehat{p}$ est calculée comme suit :

$$\widehat{p}(x) = \frac{1}{n\sigma\sqrt{2\pi}} \sum_{j=1}^{n} e^{-\frac{(x-\mu)^2}{2\sigma^2}} \tag{5.6}$$

Le choix critique pour cette méthode est l'estimation du paramètre σ qui agit sur la largeur du noyau Gaussien. Plusieurs méthodes dans la littérature ont tenté de donner une valeur optimale de ce paramètre en fonction des observations. Ces méthodes reposent souvent sur des hypothèses paramétriques ce qui est incompatible avec notre approche. De plus, le choix de ce paramètre est encore plus complexe pour des images de dimension supérieure, comme par exemple avec plusieurs composantes couleur. Un autre façon d'estimer l'entropie est d'utiliser la méthode des k plus proches voisins. Cette méthode, présentée en détails dans les chapitres 7 et A, permet d'estimer l'entropie d'un échantillon d'observations en fonction de

la distance au k-ème plus proche voisin. Le seul paramètre à fixer est la valeur de k. En pratique, l'entropie estimée est peu sensible à la variation de ce paramètre. De même, la distance au k-ème plus proche voisin se calcule trivialement en dimension supérieure. Cette méthode est ainsi particulièrement adaptée au cas d'images couleurs ou hyper-spectrales. Nous proposons dans le chapitre 6 et dans l'annexe A une étude plus approfondie de l'estimation de densité et de mesures statistiques [Wol06].
L'équation (5.5) correspond à la méthode de block-matching pour laquelle les pixels du fond sont partiellement masqués et où le critère de comparaison est l'entropie du résiduel $r(x, u)$. La comparaison entre les critères SAD et entropie est illustrée sur la figure 5.8. Cette figure présente les profils des critères SAD et entropie pour deux rayons différents. Les critères étant de dynamique et de valeur différentes, nous les normalisons dans le but de pouvoir graphiquement les comparer. La vidéo utilisée pour cette expérimentation est V_{tex} et le bloc considéré est B_g. Pour chacun des rayons utilisés, la proportion de pixels de l'objet est supérieure à la proportion de pixels du fond. Pour un rayon de 10 pixels, chaque critère permet d'estimer correctement le mouvement de l'objet. De plus, un minimum local correspondant au mouvement du fond, dû à la présence de pixels du fond ajoutés par la morphologie mathématique, est apparent dans chacun des profils. Pour un rayon de 14 pixels, le comportement des deux critères diffère. En effet, le critère SAD est gêné par la trop forte proportion de pixels du fond et le mouvement estimé par le block-matching correspond au mouvement du fond. En revanche, le mouvement estimé pour le critère fondé sur l'entropie du résiduel correspond précisément au mouvement de l'objet. Le minimum local, déjà présent avec un rayon de 10 pixels, est renforcé mais l'augmentation du nombre de pixels du fond n'est pas suffisante pour fausser l'estimation du mouvement. Nous déduisons de cet exemple simple et synthétique que le critère fondé sur l'entropie du résiduel est plus robuste aux pixels aberrants (pixels du fond) que le critère SAD. Cela permet d'augmenter la valeur maximale $R_{\max}$ du rayon de dilatation permise par la méthode (l'intervalle $[R_{\min}, R_{\max}]$ est élargi), et, par conséquent, le choix du rayon de dilatation est moins critique.

§ 5.5 RÉSULTATS

Tout au long de ce chapitre, nous avons proposé une méthode de suivi fondée sur un algorithme de bock-matching pour lequel les pixels du fond étaient partiellement masqués. Nous y avons vu que l'utilisation d'une bande de pixels du fond était obligatoire pour suivre correctement les objets homogènes ne possédant pas la structure interne suffisante. De fait, nous n'allons pas démontrer dans cette section l'intérêt de cette bande. L'étude va porter ici sur le choix du rayon de dilatation morphologique R et sur le

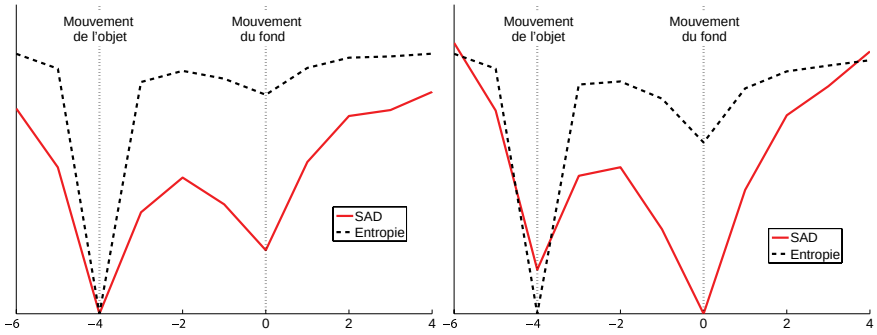


FIGURE 5.8 – *Comparaison des profils 2D des critères SAD et entropie pour le bloc B_g et la vidéo V_{tex}. Les rayons de dilatation testés sont* 10 *pixels (courbes de gauche) et* 14 *pixels (courbe de droite). L'utilisation d'un critère de comparaison fondé sur l'entropie du résiduel améliore la robustesse de l'estimation du mouvement aux données aberrantes (pixels du fond induit par la dilatation). Le choix du rayon de dilatation est par conséquent moins critique.*

choix du critère de comparaison. En effet, l'essentiel de l'étude tend à prouver que le critère fondé sur l'entropie du résiduel permet de mieux suivre les objets que le critère SAD dans la mesure ou l'entropie, étant un critère plus robuste aux données aberrantes que la SAD, permet de choisir plus librement le rayon de dilatation.

Étudions dans un premier temps la vidéo V_{tex} de dimension 300×300 pixels. Rappelons que le mouvement de l'objet est de 4 pixels vers la gauche et que le fond est fixe. Pour une dilatation de 5 pixels, la méthode permet de suivre exactement l'objet en utilisant le critère de comparaison SAD ou le critère fondé sur l'entropie du résiduel. Ce résultat est présenté sur la figure 5.9. L'erreur de suivi, comme défini dans le chapitre 4, est de 0% de pixels mal classés.

Dans la section 5.4, nous avons vu que, pour la vidéo V_{tex}, un rayon de dilatation de 10 pixels permettait de suivre précisément les blocs B_g et B_d. Si nous considérons maintenant le suivi de tous les blocs centrés sur tous les points d'échantillonnage du contour, nous remarquons que le comportement n'est pas similaire pour tous ces blocs. La proportion de pixels du fond n'est pas l'unique responsable d'une mauvaise estimation du mouvement. En effet, la texture joue aussi un rôle important. Ainsi, la figure 5.10 présente un résultat de suivi sur les 20 images de la vidéo V_{tex} utilisant le critère SAD et une dilatation de 10 pixels. Nous pouvons y voir que le mouvement des blocs B_g et B_d est correctement estimé sur les premières images. En revanche, le mouvement estimé de nombreux blocs correspond au mouvement du fond. Les blocs dont le mouvement est mal estimé sont situés sur la partie droite de l'objet car ils sont situés sur la zone où le fond

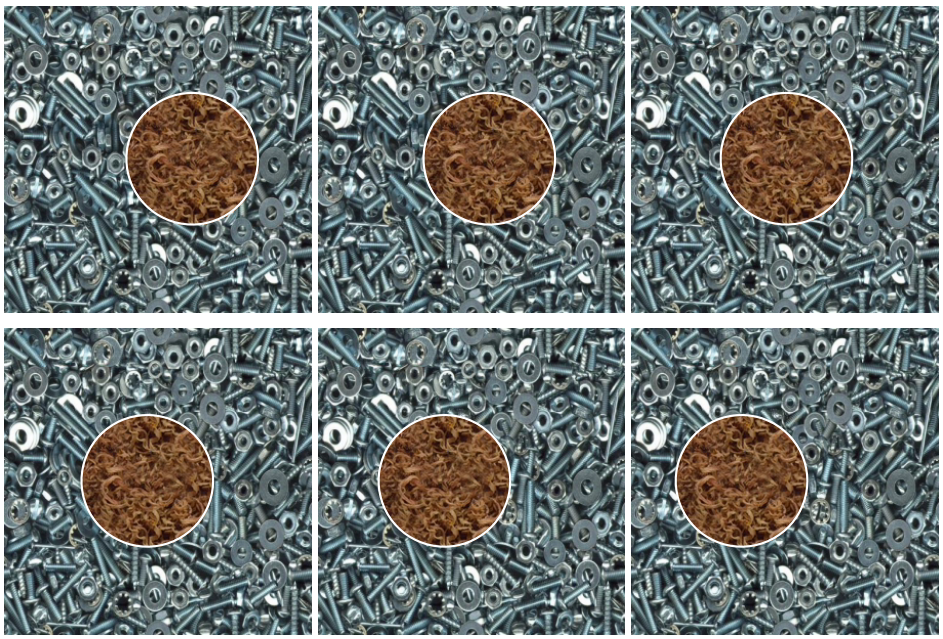


FIGURE 5.9 – *Résultat de suivi pour les images I^1, I^5, I^9, I^{12}, I^{16}, et I^{20} (respectivement de gauche à droite et de haut en bas) de la vidéo V_{tex}. La méthode utilise le critère de comparaison SAD ou le critère fondé sur l'entropie du résiduel et un rayon de dilatation de $R = 5$ pixels. L'erreur est de 0% de pixels mal classés avec ces deux critères.*

apparaît. Ces blocs sont « attirés » par le fond. Les blocs situés à gauche (zone d'occultation) sont poussés par l'objet. L'erreur commise par cette méthode est alors de 12% de pixels mal classés. Si nous utilisons maintenant un rayon de dilatation $R = 14$ pixels (voir figure 5.11), l'erreur de la méthode augmente jusqu'à une valeur de 37% de pixels mal classés. Cette augmentation est prévisible car la proportion de pixels du fond, et par conséquent leur influence, augmente avec le rayon de dilatation.

Étudions maintenant le suivi réalisé sur la même vidéo, avec les mêmes

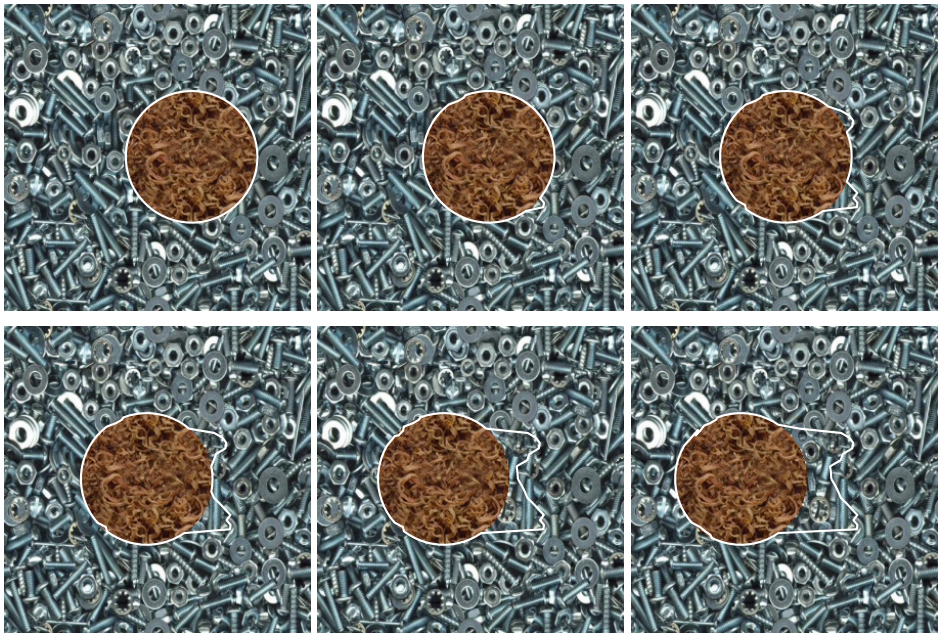


FIGURE 5.10 – *Résultat de suivi pour les images I^1, I^5, I^9, I^{12}, I^{16}, et I^{20} (respectivement de gauche à droite et de haut en bas) de la vidéo V_{tex}. La méthode utilise le critère de comparaison SAD et un rayon de dilatation de $R = 10$ pixels. L'erreur est de* 13% *de pixels mal classés.*

rayons de dilatation ($R = 10$ et $R = 14$), mais pour le critère de comparaison fondé sur l'entropie du résiduel. Les résultats obtenus avec ce critère sont présentés dans la figure 5.12. L'objet est parfaitement suivi tout au long de la vidéo que ce soit pour un rayon de 10 ou pour un rayon de 14 pixels. Dans les deux cas, l'erreur de la méthode est de 0% de pixels mal classés. Ces résultats prouvent et confirment que le critère fondé sur l'entropie du résiduel est plus robuste aux données aberrantes (pixels du fond) que le critère SAD.

La seconde vidéo utilisée pour réaliser des tests est la vidéo *Carmap* de dimension 320×240 pixels. Cette vidéo comporte 36 images. L'objet que nous voulons suivre est une pancarte rigide dont le mouvement apparent est dû au mouvement de la camera. Ce mouvement est approximativement

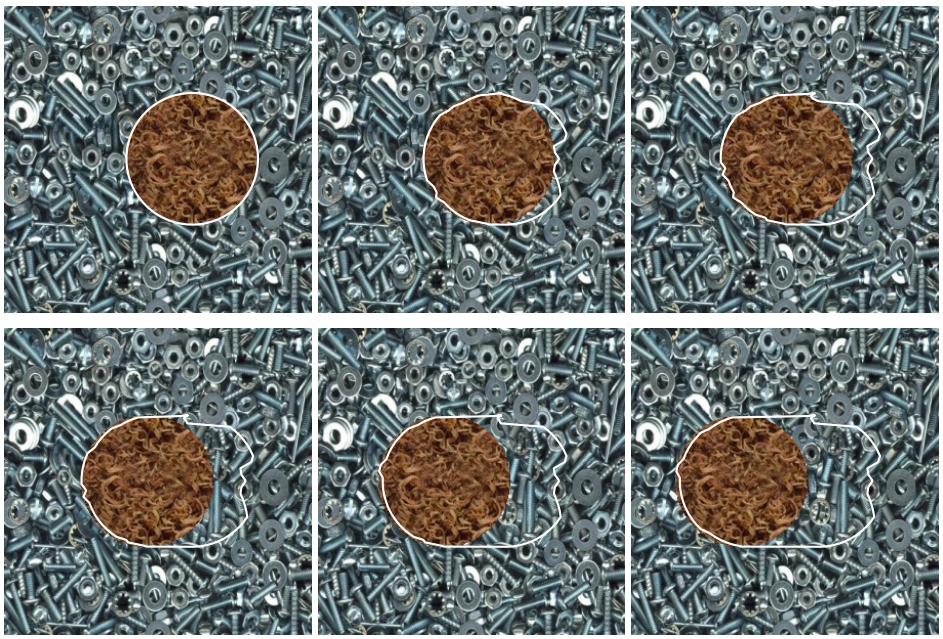


FIGURE 5.11 – *Résultat de suivi pour les images I^1, I^5, I^9, I^{12}, I^{16}, et I^{20} (respectivement de gauche à droite et de haut en bas) de la vidéo V_{tex}. La méthode utilise le critère de comparaison SAD et un rayon de dilatation de $R = 14$ pixels. L'erreur est de 37% de pixels mal classés.*

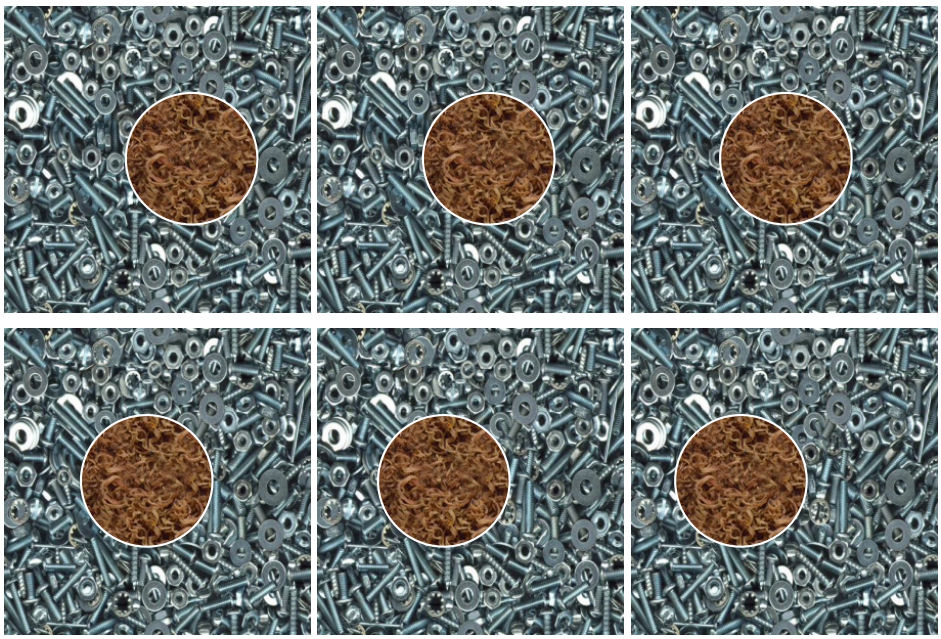


FIGURE 5.12 – *Résultat de suivi pour les images I^1, I^5, I^9, I^{12}, I^{16}, et I^{20} (respectivement de gauche à droite et de haut en bas) de la vidéo V_{tex}. La méthode utilise le critère de comparaison fondé sur l'entropie du résiduel et un rayon de dilatation de $R = 10$ ou $R = 14$ pixels. Dans les deux cas, l'erreur est de 0% de pixels mal classés.*

une translation. Cependant, du fait du changement de point de vue, la pancarte semble se déformer légèrement. L'objet et le fond sont tous deux texturés. Nous utilisons pour cette vidéo un rayon de dilatation de 5 pixels. L'utilisation du critère de comparaison SAD permet de suivre précisément 3 des 4 points d'échantillonnage. Le quatrième point est correctement suivi jusqu'à ce que la voiture apparaisse derrière la pancarte. Du fait de la dilatation trop importante, le mouvement estimé du point correspond au mouvement de la voiture et l'erreur de suivi est alors de 36% de pixels mal classés (voir figure 5.13). L'utilisation du critère fondé sur l'entropie du résiduel permet de diminuer l'impact des pixels aberrants. L'erreur est ainsi réduite à 4% de pixels mal classés ce qui correspond à un suivi de très bonne qualité comme nous pouvons le voir sur la figure 5.14.

La troisième vidéo utilisée pour réaliser des tests est la vidéo *Soccer* de dimension 704×576 pixels. Cette vidéo comporte 11 images. L'objet que nous voulons suivre est un joueur marchant sur un fond quasiment fixe. L'objet est par nature articulé lors de la marche. Le fond de la vidéo est texturé tandis que l'objet est homogène sur certaines parties du corps (jambes) et texturé sur d'autres (buste). Nous utilisons pour cette vidéo un rayon de dilatation de 10 pixels. En utilisant le critère SAD, nous pouvons voir que beaucoup de points ont un mouvement estimé égal au mouvement du fond. Le problème qui se pose ici est d'une part la présence de pixels du fond dans le bloc, et d'autre part la présence de flou lié au mouvement. En effet, nous pouvons voir sur la figure 5.15 que le fond est très net contrairement au personnage à suivre qui est légèrement flou. De même, le contour du personnage est très flou et par conséquent impossible à définir exactement. Ce flou, généralement appelé « motion blur » (flou de mouvement), est un problème connu en vision par ordinateur. Le critère SAD étant un critère strictement géométrique, il favorise donc les pixels du fond (nets) qui ont un correspondant exact au détriment des pixels de l'objet. L'erreur liée à la méthode est de 17% de pixels mal classés (voir figure 5.15). Le critère fondé sur l'entropie est géométriquement plus souple que le critère SAD. De même, comme nous l'avons déjà évoqué, ce critère est plus robustes aux données aberrantes. L'utilisation de ce critère permet de diminuer l'erreur de suivi à 8% de pixels mal classés. Le personnage est correctement suivi en dépit du mouvement articulé des jambes(voir figure 5.16). Cependant, nous pouvons également remarquer que, même si le suivi est meilleur qu'avec le critère SAD, certaines parties du personnage sur la dernière image sont segmentées avec une précision approximative. Ce comportement est lié à la netteté des contours. En effet, le motion blur rend le contour du personnage flou aux zones de mouvement (buste) et net là où il n'y a pas de mouvement (jambe droite). Du fait de ce flou irrégulier, l'ajout d'une bande de pixels du fond augmente la proportion de ces pixels de façon irrégulière. Il faudrait ainsi faire varier le rayon de dilatation pour chaque bloc en fonc-

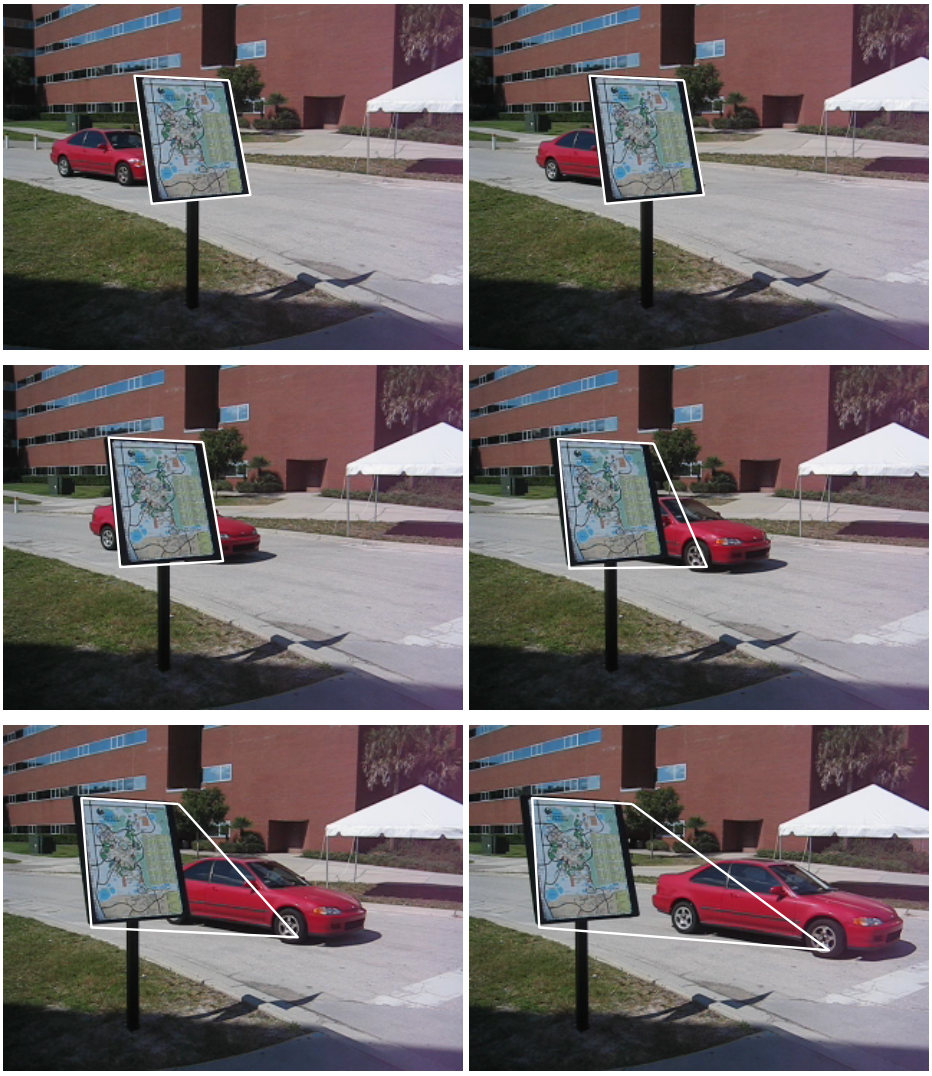


FIGURE 5.13 – *Résultat de suivi pour les images I^1, I^8, I^{15}, I^{22}, I^{29}, et I^{36} (respectivement de gauche à droite et de haut en bas) de la vidéo Carmap. La méthode utilise le critère de comparaison SAD et un rayon de dilatation de $R = 5$. L'erreur est de 36% de pixels mal classés.*

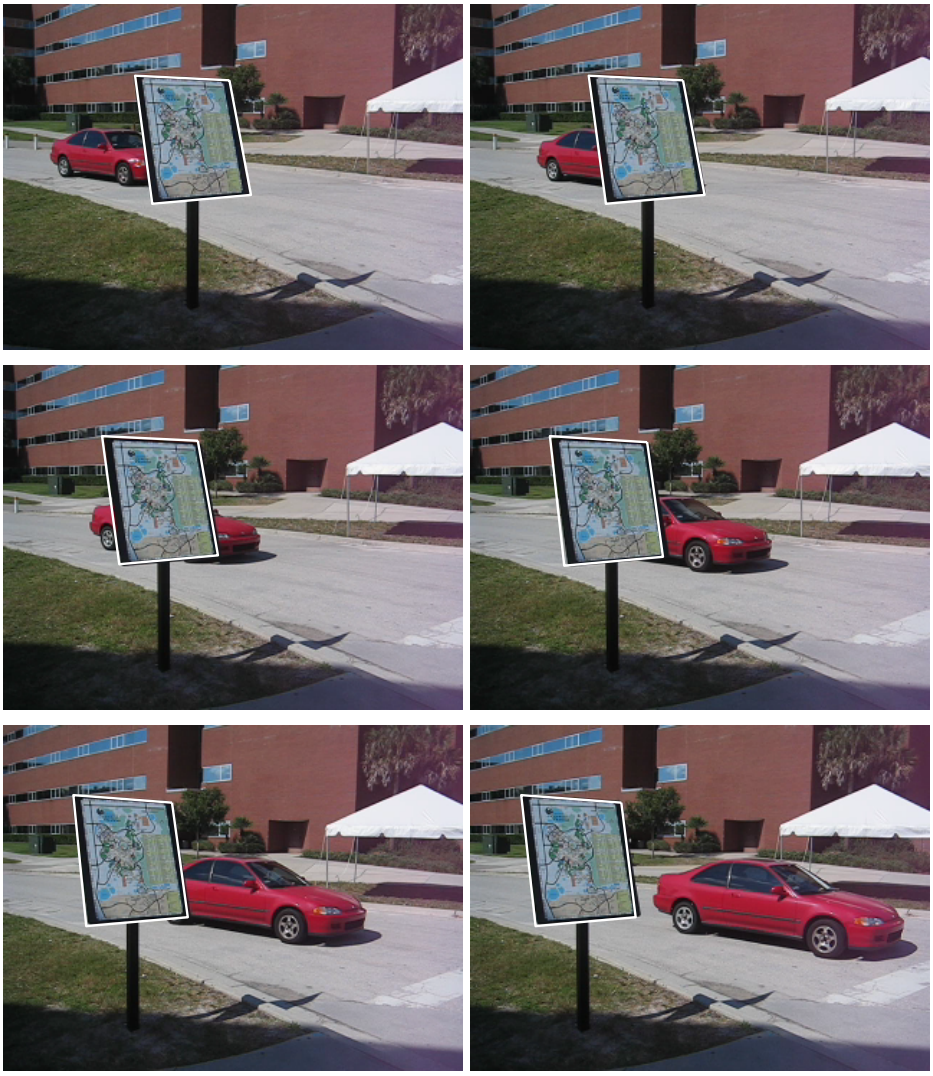


FIGURE 5.14 – *Résultat de suivi pour les images I^1, I^8, I^{15}, I^{22}, I^{29}, et I^{36} (respectivement de gauche à droite et de haut en bas) de la vidéo Carmap. La méthode utilise le critère de comparaison fondé sur l'entropie du résiduel et un rayon de dilatation de $R = 5$. L'erreur est de 4% de pixels mal classés.*

tion du mouvement de l'objet.

Enfin, la dernière vidéo utilisée est la vidéo *Ice* de dimension 704×576 pixels et comportant 6 images. Sur cette vidéo, l'objet à suivre est une patineuse dont les jambes et les bras sont en mouvement. Cette vidéo est intéressante car les critères SAD et entropie donnent des résultats similaires (voir figure 5.17). En effet, l'erreur de suivi est, dans les deux cas, de 4% de pixels mal classés en dépit du mouvement global du personnage et du mouvement complexe des jambes et des bras. Cette particularité est liée au fait que le fond est très homogène tandis que le personnage est plus texturé. Il est important de noter que le suivi est beacoup plus rapide en utilisant le critère SAD que pour le critère fondé sur l'entropie. Considérons une implémentation matlab d'un suivi de 6 images avec une recherche exhaustive sur une fenêtre de recherche de 15×15 pixels. L'ordinateur utilisé est un Pentium 4 avec 2 Go de mémoire vive DDR et fonctionnant sous Windows. Le suivi fondé sur la SAD est realisé en 1 minute contre 8 minutes pour le suivi utilisant l'entropie. Cette différence s'explique par le fait que l'entropie est beaucoup plus complexe à estimer qu'une simple somme des différences en valeur absolue. Ainsi, pour certaines vidéos telles que la vidéo *Ice*, l'utilisation du critère SAD permet de réduire les coûts calcul sans pour autant pénaliser la qualité du suivi.

§ 5.6 CONCLUSION

Dans ce chapitre, nous avons proposé une méthode de suivi d'objets. Le mouvement de chaque point d'échantillonnage du contour est suivi au cours du temps par une simple méthode de block-matching. Le problème principal auquel nous nous sommes attachés vient du fait que les blocs centrés sur les points d'échantillonnage du contour sont des blocs tangents contenant à la fois des pixels du fond et des pixels de l'objet d'intérêt. Les pixels du fond, alors considérés comme aberrants par rapport aux pixels de l'objet, perturbent l'estimation du mouvement. Nous avons alors proposé de les masquer partiellement en utilisant le masque de l'objet dilaté par une opération de morphologie mathématique. Ce masquage permet d'une part de diminuer la proportion de pixels du fond ce qui réduit leur impact lors du processus de block-matching, et d'autre part, cela permet de conserver la structure de bord essentielle lorsqu'un objet est localement homogène. Nous avons prouvé l'efficacité de notre approche sur deux séquences synthétiques.
La dilatation du masque de l'objet conserve une partie des pixels du fond. Selon le rayon de dilatation, la proportion de ces pixels (alors considérés comme aberrants) peut tout de même induire une mauvaise estimation du mouvement. Le choix du rayon de dilatation apparaît alors comme cri-

FIGURE 5.15 – *Résultat de suivi pour les images I^1, I^3, I^5, I^7, I^9, et I^{11} (respectivement de gauche à droite et de haut en bas) de la vidéo Soccer. La méthode utilise le critère de comparaison SAD et un rayon de dilatation de $R = 10$. L'erreur est de 17% de pixels mal classés.*

FIGURE 5.16 – *Résultat de suivi pour les images I^1, I^3, I^5, I^7, I^9, et I^{11} (respectivement de gauche à droite et de haut en bas) de la vidéo Soccer. La méthode utilise le critère de comparaison fondé sur l'entropie du résiduel et un rayon de dilatation de $R = 10$. L'erreur est de 8% de pixels mal classés.*

FIGURE 5.17 – *Résultat de suivi pour les images I^1, I^2, I^3, I^4, I^5, et I^6 (respectivement de gauche à droite et de haut en bas) de la vidéo Ice. La méthode utilise le critère SAD ou pour le critère fondé sur l'entropie du résiduel et un rayon de dilatation de $R = 10$. L'erreur est de 4% de pixels mal classés.*

tique : un rayon trop grand ou trop petit peut induire un suivi incorrect. Ce comportement est lié au fait que l'utilisation de critères de comparaison classiques (tels que la somme des différences en valeur absolue notée SAD) est généralement fondé sur une hypothèse de distribution paramétrique du résiduel (Laplacien dans le cas du critère SAD), hypothèse généralement fausse. Nous avons alors proposé de remplacer ce critère de comparaison par un critère semi-paramétrique fondé sur l'estimation de l'entropie du résiduel. Il apparaît alors que ce critère est plus robuste aux pixels aberrants. Le choix du rayon de dilatation est alors moins critique pour obtenir un suivi correct.
Le problème qui se pose est inhérent au mouvement des objets. Dans une vidéo acquise avec une vitesse d'obturation élevée, les objets et le fond sont généralement nets [2]. Cette netteté est cependant remise en cause si l'objet bouge et si la vitesse d'obturation n'est pas suffisamment élevée. Dans un tel cas, le contour de l'objet à suivre peut apparaître flou (on parle alors de flou de mouvement ou « motion blur »). Il est alors extrêmement difficile d'éditer manuellement la position exacte de ce contour. L'ajout d'une bande de pixels de fond peut alors s'avérer insuffisant dans le cas d'objets homogènes. Le choix du rayon de dilatation devient complexe. Le problème se complexifie davantage si le mouvement de l'objet n'est pas uniforme. En effet, dans le cas d'objets articulés, certaines parties peuvent être fixes et d'autres en mouvement. Il en résulte que le contour peut être net à un endroit et flou à un autre. Le choix du rayon de dilatation devrait alors tenir compte de tous ces paramètres ce qui est en pratique extrêmement complexe. Nous ne pouvons utiliser l'information de mouvement pour deviner si le contour risque d'être flou car notre problème est juste d'estimer ce mouvement. Une étude de la netteté serait à envisager pour déterminer un rayon de dilatation optimal pour chaque point d'échantillonnage du contour.

2. La netteté des objets est également liée à la profondeur de champ qui dépend de l'ouverture de l'objectif. Si un objectif a une faible profondeur de champ et que la mise au point est réalisée sur le fond, l'objet d'intérêt est souvent flou. Cependant, nous nous plaçons dans le cas où la profondeur de champ est suffisamment grande pour rendre nets les objets acquis.

DEUXIÈME PARTIE

SUIVI D'OBJETS DANS UNE VIDÉO FONDÉ SUR DES MESURES STATISTIQUES

- CHAPITRE 6 -

SUIVI D'OBJETS FONDÉ SUR DES MESURES STATISTIQUES

§ 6.1 VERS DES MESURES STATISTIQUES POUR LE SUIVI

Les méthodes de suivi fondées sur l'utilisation de points saillants et de leur voisinage local ont l'intérêt de pouvoir suivre un objet en s'adaptant aux déformations géométriques dudit objet. Le suivi résultant de ces méthodes est souvent « trop » précis pour des applications courantes. En effet, pour des applications telles que la surveillance de foules ou l'analyse de trafic routier, seule la position et parfois la taille des objets sont nécessaires ; les objets sont alors simplement représentés par des boîtes dites englobantes.

L'utilisation de boîtes englobantes induit une considérable augmentation de la quantité d'informations utilisées pour décrire l'objet d'intérêt. Malgré la présence de données aberrantes dans la boîte englobante (pixels du fond), la description de l'objet est généralement plus complète et plus précise que dans le cas de la description locale.

La méthode de suivi présentée ici consiste à trouver dans chaque image de la vidéo la boîte englobante la plus similaire à la boîte de référence éditée manuellement sur la première image de la vidéo. Les méthodes classiques de type block-matching [KLH+81] utilisent des mesures de similarité fondées sur le résiduel (*e.g.* SAD[1] ou SSD[2]). Ces mesures sont dites à géométrie stricte. Or, au cours du temps, un objet est susceptible de changer de forme, de taille, d'éclairage, etc. L'utilisation d'une mesure de similarité fondée sur le résiduel est donc exclue. En revanche, les propriétés statistiques d'un objet varient généralement peu dans le temps. L'utilisation

1. SAD : *Sum of Absolute Differences*
2. SSD : *Sum of Squared Differences*

d'une mesure statistique de similarité semble par conséquent être indiquée pour suivre un objet en dépit des éventuelles déformations rencontrées.

La description d'une boîte par une propriété statistique simple telle que la moyenne ou la variance de l'intensité ne permet pas de décrire efficacement les objets complexes. En revanche, la densité de probabilité (distribution) est suffisamment riche pour précisément décrire un objet. Le choix des composantes (*e.g.* couleur, luminance, etc.) utilisées pour la construction de cette distribution est crucial. En effet, l'utilisation de composantes inappropriées peut rendre le suivi d'objets inefficace.

Ces travaux on été réalisés dans le cadre de la thèse de Sylvain Boltz [BDB07, Bol08] sur le suivi d'objets fondé sur des mesures statistiques (voir figure 6.1) et ont été utilisés dans le projet *Wired Smart* de l'ANR RIAM [3].

FIGURE 6.1 – *Suivi de boîte englobante sur la vidéo « Water object ». La boîte de référence est définie sur la première image de la vidéo (figure de gauche). À l'image 60, la boîte correspondante est plus grande et est décalée spatialement (figure de droite).*

§ 6.2 DESCRIPTEUR : COULEUR + GÉOMÉTRIE

Dans le cas où la distribution est calculée sur la couleur, il est possible que plusieurs objets visuellement différents aient une même distribution (voir figure 6.2). Ces objets sont alors identiques au sens de la mesure statistique de similarité. L'enrichissement de l'espace des caractéristiques (espace dans lequel la distribution est construite) par l'adjonction de composantes permet d'augmenter le caractère discriminant des distributions. Dans cet esprit, Elgammal *et al.* [EDD03] ont proposé d'ajouter une information géométrique (*e.g.* coordonnées cartésiennes de chaque pixel dans la

3. RIAM : Réseau National de Recherche et d'Innovation en Audiovisuel et Multimédia

boîte) lors de l'estimation de la distribution d'un objet. La figure 6.3 illustre un exemple d'espace des caractéristiques contenant des informations relatives à la couleur et à la géométrie. Si la description d'une boîte utilise les composantes couleurs YUV et les coordonnées cartésiennes, un pixel p est décrit par le vecteur de description suivant :

$$p = \begin{pmatrix} Y(i,j) \\ U(i,j) \\ V(i,j) \\ i \\ j \end{pmatrix} \tag{6.1}$$

où (i,j) représentent les coordonnées du pixel p dans la boîte.
La contrainte géométrique est assouplie par le cadre statistique de la méthode. Boltz *et al.* [BDB07, Bol08] ont proposé d'enrichir l'espace des caractéristiques en utilisant le gradient de l'image et des patchs ce qui apporte une information géométrique locale. Dans ce manuscrit, nous étudions ce type d'espace (voir section 9.2).

Pour chaque boîte (référence et candidates), nous construisons la distribution des descripteurs. Il est alors nécessaire de définir une mesure de similarité entre distributions. Parmi les différentes mesures existantes, nous pouvons par exemple citer l'information mutuelle beaucoup utilisée en recalage d'images pour des applications médicales [SHH96, WVA+96]. La divergence de Kullback-Leibler [KL51, Kul59] est une mesure statistique qui a été utilisée avec succès pour le suivi d'objets [EDD03, BDB07, Bol08]. Nous utilisons donc cette divergence pour évaluer la similarité de deux distributions.

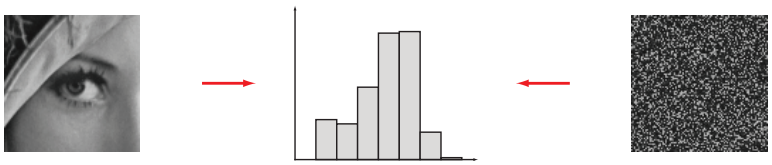


FIGURE 6.2 – *Deux images visuellement très différentes peuvent avoir une distribution identique.*

FIGURE 6.3 – *Utilisation d'informations couleur et d'informations géométriques pour la description d'une image. Chaque pixel de l'image est défini dans un espace de dimension 5 : 3 composantes couleurs RVB (rouge, vert, bleu) et 2 coordonnées cartésiennes.*

§ 6.3 MESURE DE SIMILARITÉ : DIVERGENCE DE KULLBACK-LEIBLER

6.3.1 Similarité entre distributions

Considérons une boîte de référence et une boîte candidate pour lesquelles nous voulons comparer les distributions. Les pixels de la boîte candidate et de la boîte de référence sont respectivement des réalisations des variables aléatoires P et Q. La divergence de Kullback-Leibler [KL51, Kul59], notée D_{KL}, permet d'évaluer la similarité des deux distributions :

$$D_{\mathrm{KL}}(P\|Q) = \begin{cases} 0 & \text{si et seulement si } P = Q \\ > 0 & \text{sinon.} \end{cases} \tag{6.2}$$

Pour notre application, le but est de trouver dans chaque image de la vidéo la boîte candidate P telle que $D_{\mathrm{KL}}(P\|Q)$ soit minimale.

6.3.2 Définition

Soient P et Q deux vecteurs aléatoires à valeurs dans $\mathbb{R}^d$. La D_{KL} est donnée par :

$$D_{\mathrm{KL}}(P\|Q) = \int_{x\in\mathbb{R}^d} f_P(x) \log \frac{f_P(x)}{f_Q(x)} dx \tag{6.3}$$

$$= E\left[\log \frac{f_P(P)}{f_Q(P)}\right] \tag{6.4}$$

où f_P et f_Q désignent respectivement les densités de probabilité de P et Q. Ces densités ne sont généralement pas connues et doivent être estimées à partir d'observations (*i.e.* réalisations de P et Q). L'estimation de densités est un problème particulièrement complexe en haute dimension. En particulier, les méthodes à noyau fixe [Par62, Par99] échouent. Les estimateurs Balloon et *sample-point* permettent de résoudre ce problème. L'estimateur de Balloon a l'avantage d'avoir une expression de la D_{KL} simple fondée sur la distance au k-ème plus proche voisin (voir suite).

6.3.3 Approximation de D_{KL} par la méthode des k plus proches voisins

Nous présentons ici l'estimation de D_{KL} d'après les échantillons contenus dans les boîtes de référence et candidates. Soient $\mathcal{P} = \{p_1, p_2, \cdots, p_{N_\mathcal{P}}\}$ un ensemble de $N_\mathcal{P}$ réalisations de P et $\mathcal{Q} = \{q_1, q_2, \cdots, q_{N_\mathcal{Q}}\}$ un ensemble de $N_\mathcal{Q}$ réalisations de Q. $\mathcal{P}$ et $\mathcal{Q}$ représentent respectivement les pixels de la boîte candidate et de la boîte de référence. La D_{KL} est approximée par la moyenne de $\log \frac{f_P(x)}{f_Q(x)}$:

$$D_{\mathrm{KL}}(P\|Q) \approx \frac{1}{N_\mathcal{P}} \sum_{i=1}^{N_\mathcal{P}} \log \frac{\widehat{f}_P(p_i)}{\widehat{f}_Q(p_i)} \tag{6.5}$$

où $\widehat{f}_P$ et $\widehat{f}_Q$ désignent respectivement les densités estimées de P et Q. Ces densités sont estimées à partir des réalisations $\mathcal{P}$ et $\mathcal{Q}$ par l'estimateur Balloon. Boltz *et al.* utilisent un noyau uniforme dont la taille (variable) dépend de la distance au k-ème plus proche voisin. Cette méthode permet de s'adapter à la densité locale de P et Q :

$$\widehat{f}_U(x) = \frac{k}{N_\mathcal{U}\, v_d\, \rho_k(x, \mathcal{U})^d} \tag{6.6}$$

où v_d représente le volume de la boule unité dans $\mathbb{R}^d$ et $\rho_k(x, \mathcal{U})$ désigne la distance entre le point x et son k-ème plus proche voisin dans l'ensemble $\mathcal{U}$ contenant $N_\mathcal{U}$ réalisations du vecteur aléatoire U. Si $x \in \mathcal{U}$, le calcul de la distance ne doit pas tenir compte du point x. En remplaçant 6.6 dans 6.5, nous obtenons l'expression de la D_{KL} suivante :

$$\begin{aligned} D_{\mathrm{KL}}(P\|Q) = &\log \frac{N_\mathcal{Q}}{N_\mathcal{P} - 1} \\ &+ \frac{d}{N_\mathcal{P}} \sum_{i=1}^{N_\mathcal{P}} \log(\rho_k(p_i, \mathcal{Q})) \\ &- \frac{d}{N_\mathcal{P}} \sum_{i=1}^{N_\mathcal{P}} \log(\rho_k(p_i, \mathcal{P} \setminus \{p_i\})) \end{aligned} \tag{6.7}$$

Il est important de remarquer que f_P et f_Q ne sont pas explicitement calculées. À la place, on construit simplement $\mathcal{P}$ et $\mathcal{Q}$. Pour plus de détails sur la méthode et sur l'estimation de la D_{KL}, le lecteur doit se référer à la thèse de Sylvain Boltz [Bol08].

§ 6.4 ALGORITHME

- La première étape de l'algorithme consiste à localiser l'objet sur la première image de la vidéo. La boîte de référence est éditée manuellement par un opérateur. Cette boîte nous permet de construire les échantillons $\mathcal{Q}$.

- Supposons que le suivi ait été réalisé jusqu'à l'image n. Ce suivi nous donne la position δ_n et l'échelle σ_n de la boîte optimale à l'image n.

- Considérons un ensemble de valeurs discrètes $(\delta_{n+1}, \sigma_{n+1})$ centrées sur (δ_n, σ_n). Chaque couple $(\delta_{n+1}, \sigma_{n+1})$ permet de construire les échantillons $\mathcal{P}$. Le couple $(\delta_{n+1}, \sigma_{n+1})$ minimisant D_{KL} définit la position et l'échelle de la boîte optimale pour l'image $n+1$.

§ 6.5 UN FREIN : LE COÛT CALCUL

La recherche des k plus proches voisins (kPPV) permettant l'évaluation de la divergence de Kullback-Leibler est un problème coûteux en temps de calcul. 95% du temps peut être attribué à cette recherche dans le cadre de la méthode de suivi d'objets présentée dans ce chapitre. Cette durée augmente avec la dimension des descripteurs. Deux approches permettent de rechercher les kPPV. La plus connue et la plus utilisée consiste premièrement à structurer les données sous forme d'arbre (*e.g.* kd-tree), et deuxièmement à rechercher les kPPV par une méthode itérative. L'approche que nous proposons dans la suite du manuscrit consiste à effectuer une recherche parallèle des kPPV sur les données brutes (pas de structuration des données). Nous utilisons pour cela la programmation GPU.

- CHAPITRE 7 -

RECHERCHE DES K PLUS PROCHES VOISINS

§ 7.1 PROBLÈME

La recherche des k plus proches voisins, notée kPPV dans la suite du manuscrit, est un problème rencontré dans de très nombreuses applications de traitement d'images, mais aussi dans des domaines bien différents tels que la biologie, la physique, la finance, la génétique, etc. Le problème est d'une très grande simplicité, au moins quant à sa définition. Soit $\mathcal{R}$, un ensemble de m objets dits de référence appartenant à un espace E donné (*e.g.* des points dans un espace Euclidien). Soit q, un objet, dit requête, appartenant à E (« q » pour *query* en anglais signifiant *requête* en français). Enfin, soient k un nombre positif généralement entier et ρ une mesure de distance. Le problème de la recherche des kPPV consiste à identifier dans E les k objets de référence les plus proches de q au sens de la distance ρ. La figure 7.1 illustre un exemple pour un espace Euclidien bidimensionnel et pour $k = 3$. Le problème de la recherche des kPPV se généralise à un ensemble noté $\mathcal{Q}$ de plusieurs requêtes. La distance la plus communément utilisée est la distance Euclidienne. À ce titre, cette distance sera utilisée par défaut dans la suite du manuscrit si la distance choisie n'est pas explicitement indiquée.
Nous verrons plus loin une méthode simple pour résoudre ce problème : la recherche exhaustive. Cependant, cette méthode a pour inconvénient d'être d'une extrême lenteur si le nombre de références et le nombre de requêtes sont grands ou si la distance utilisée est complexe. Il existe dans la littérature scientifique de nombreuses méthodes proposant d'accélérer cette recherche. En effet, pour des applications fondées sur la recherche des kPPV, le temps d'exécution lié à cette recherche représente souvent une part importante de la durée totale des calculs. Selon l'application, cette part peut représenter plus de 95% du temps de calcul total. L'accélération de cette recherche apparaît par conséquent cruciale.

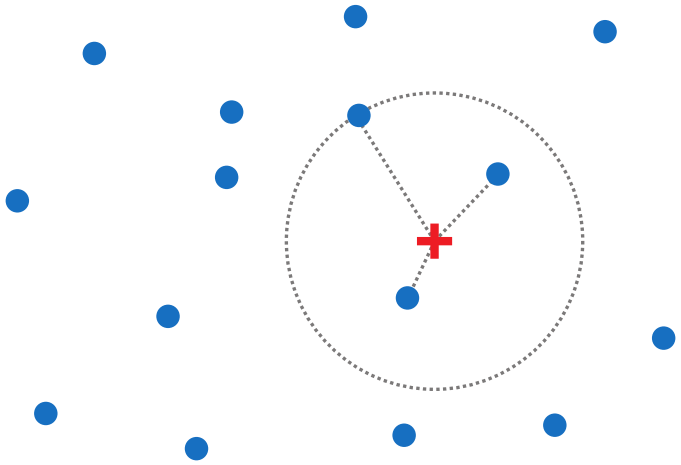


FIGURE 7.1 – *Illustration de la recherche des kPPV dans un espace Euclidien bidimensionnel. Les points bleus sont les points de référence. La croix rouge est le point requête. Dans cette exemple, $k = 3$. Si l'on considère une distance Euclidienne, les 3 plus proches voisins de la requête appartiennent au cercle en pointillés centré sur le point requête. Le rayon de ce cercle représente la distance au 3ème plus proche voisin. Cette distance est par exemple utilisée pour l'estimation de mesures statistiques telles que l'entropie de Shannon ou la divergence de Kullback-Leibler.*

§ 7.2 MÉTHODES DE RECHERCHE DES KPPV

7.2.1 Approche exhaustive

L'approche la plus simple pour rechercher les kPPV est la méthode consistant à tester toutes les possibilités. Cette approche exhaustive est connue en anglais sous le nom de *brute-force*. Par conséquent, elle sera désignée dans la suite du manuscrit par « BF », acronyme du terme anglais.
Pour expliquer en détails la méthode BF, donnons au préalable quelques notations. Considérons un espace Euclidien E. Soit $\mathcal{R} = \{r_1, r_2, \cdots, r_m\}$ un ensemble de m points de référence et soit q un point requête. $\mathcal{R}$ et q sont définis dans E. Enfin, soient ρ une mesure de distance définie sur E et k un entier naturel supérieur strictement à 0. La méthode BF consiste en les étapes suivantes :

1. Calculer les distances $\rho(q, r_i)$ pour $i \in [1, m]$.
2. Trouver les k points de référence (indices) engendrant les k plus petites distances.

La seconde étape est réalisée en triant les distances calculées par un tri quelconque. Il est important de conserver l'association entre une distance et l'indice du point de référence auquel elle correspond. Dans le cas où il n'y a qu'un point requête, la méthode BF est optimale.

7.2.2 Partitionnement de l'espace

Si nous considérons maintenant plusieurs points requête, la méthode BF devient très coûteuse en temps de calcul. Le calcul d'une distance étant souvent long en haute dimension, l'inconvénient de la méthode BF vient du fait que toutes les distances sont calculées. Ainsi, de nombreux travaux de recherche ont proposé des méthodes minimisant le nombre de distances calculées (*branch and bound*).

Un *kd-tree* [Ben75, Ind04] (pour *k-dimensional tree*) est une structure de données (arbre binaire) construite à partir d'un ensemble de points définis dans un espace Euclidien de dimension k. Par la suite, cette dimension sera notée d pour lever l'ambiguïté avec le paramètre k utilisé pour la recherche des kPPV. Un kd-tree est une partition de l'espace Euclidien, chaque noeud et chaque feuille étant un point de l'ensemble considéré. L'arbre est construit en début de programme en considérant l'ensemble des points de référence. Ensuite, chaque point requête parcourt l'arbre selon une procédure particulière jusqu'à trouver ses kPPV. L'arbre étant une partition de l'espace, le parcourt consiste grossièrement à descendre dans l'arbre en calculant à chaque étape la distance entre le point requête et le point de référence contenu dans le noeud courant. À la fin de la procédure, seule une partie de l'arbre a été parcourue.

La construction d'un kd-tree est une procédure récursive simple à mettre en oeuvre. Cette procédure est fondée sur la recherche du point médian, c'est-à-dire la recherche du point appartenant à un ensemble et qui sépare cet ensemble en deux parties égales. Dans un espace Euclidien de dimension 1, la notion de médian est clairement définie. Pour un espace de dimension supérieure, cette notion n'est pas claire car il existe un grand nombre de partitions possibles d'un ensemble de points en deux sous-ensembles de taille égale. Pour la construction d'un kd-tree, la recherche du médian est toujours réalisée relativement à une dimension choisie. Cette recherche est donc semblable à celle en dimension 1.
Soit $\mathcal{P}$ un ensemble de m points définis dans un espace Euclidien de dimension d. La première étape consiste à déterminer le médian de l'ensemble $\mathcal{P}$ relativement à la première dimension. Ce médian constitue la racine de l'arbre. Les points inférieurs au médian (relativement à la dimension 1) constituent le sous-arbre gauche, et les points supérieurs constituent le sous-arbre droit. Le médian de chaque sous-ensemble est alors déterminé relativement à la seconde dimension. Ces médians correspondent aux racines des sous-arbres gauche et droit, mais aussi aux fils gauche et droit de la racine de l'arbre. Nous disposons à ce stade de 4 sous-ensembles dans lesquels la recherche du médian est réalisée relativement à la troisième dimension, et ainsi de suite. Cette procédure est itérée jusqu'à ce que l'en-

semble des dimensions ait été utilisé. La figure 7.2 illustre l'exemple de la construction d'un kd-tree.

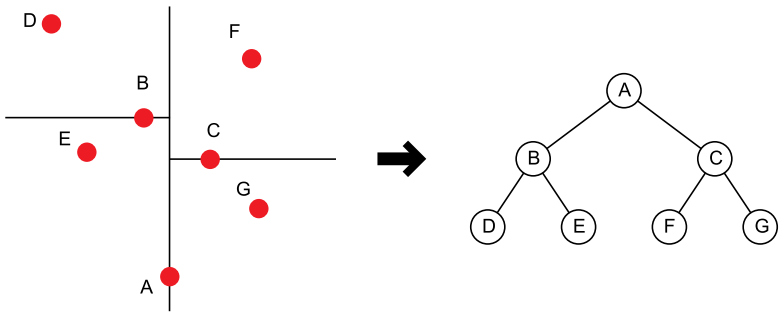


FIGURE 7.2 – *Construction d'un kd-tree à partir d'un ensemble de 7 points définis dans un espace Euclidien de dimension 2. Figure inspirée d'une illustration provenant du site Internet Wikipedia [Wik].*

7.2.3 Locality sensitive hashing

Pour des méthodes fondées sur le partitionnement de l'espace (*e.g.* kd-tree), il a été montré [WB98] que la recherche des kPPV en haute dimension n'était pas nécessairement plus rapide qu'une simple méthode exhaustive. Andoni *et al.* ont proposé [IM98, GIM99, DIIM04, AI06] une méthode de recherche des kPPV particulièrement intéressante pour de telles dimensions. Cette méthode, nommée LSH (pour *Locality Sensitive Hashing*), est fondée sur des fonctions de hachage.

Les fonctions de hachage sont des outils très utilisés en informatique et en cryptographie. Pour une information donnée, une fonction de hachage associe une valeur appelée valeur de hachage (ou empreinte). Le calcul d'une fonction de hachage doit être extrêmement rapide. Pour des applications classiques en informatique, la valeur de hachage est généralement un condensé ou tout simplement un indice servant à identifier très rapidement une donnée particulière. En règle générale, la valeur de hachage est plus courte que la donnée initiale. Le terme *collision* est employé pour indiquer que deux données différentes ont la même valeur de hachage. Une collision empêche de différencier des données ayant une même valeur de hachage. Ainsi, il est en général judicieux de choisir une fonction de hachage permettant d'éviter au maximum les collisions.

La méthode LSH repose globalement sur la remarque suivante. Pour une fonction de hachage donnée, la probabilité que deux points proches aient une même valeur de hachage est forte. Inversement, la probabilité que deux points éloignés aient une même valeur de hachage est faible. Soit

$\mathcal{F}$ une famille de fonctions de hachage $h : \mathcal{M} \rightarrow S$ définies sur l'espace métrique $\mathcal{M} = (M, \rho)$. Soient $R > 0$ un seuil, $c > 1$ un facteur d'approximation, $h \in \mathcal{F}$ une fonction choisie aléatoirement, et p et q deux points de $\mathcal{M}$. $\mathcal{F}$ est une famille LSH si elle vérifie les deux conditions suivantes :

- Si $\rho(p,q) \leq R$ alors $Pr[h(p) = h(q)] \geq P_1$,
- si $\rho(p,q) \geq cR$ alors $Pr[h(p) = h(q)] \leq P_2$.

où $Pr[h(p) = h(q)]$ désigne la probabilité que les points p et q aient la même valeur de hachage (collision). Une telle famille est dite (R,cR,P_1,P_2)-*sensitive* et est intéressante pour $P_1 > P_2$. Dans notre application, l'espace métrique considéré est l'espace Euclidien muni de la distance Euclidienne. Soit $h_{a,b} : \mathbb{R}^d \rightarrow \mathbb{Z}$ une fonction de hachage définie sur $\mathbb{R}^d$ et à valeur dans $\mathbb{Z}$. Pour $p \in \mathbb{R}^d$, $h_{a,b}$ est définie par :

$$h_{a,b}(p) = \left\lfloor \frac{a \cdot p + b}{w} \right\rfloor \tag{7.1}$$

où a est un vecteur de dimension d pour lequel chaque composante est tirée aléatoirement dans $\mathbb{R}$ selon une loi normale, et où b est un nombre réel tiré uniformément dans $[0, w]$. Le paramètre w correspond à la largeur de la projection. Dans [DIIM04], ce paramètre est fixé à $w = 4$. Soit $\mathcal{G}$ une famille de fonctions de hachage construites par la concaténation de v fonctions de hachage $h = h_{a,b}$:

$$g(p) = [h_1(p), h_2(p), \cdots, h_v(p)] \tag{7.2}$$

Pour $p \in \mathbb{R}^d$, $g(p)$ est vecteur à valeurs dans $\mathbb{Z}^v$. $g(p)$ est appelé *sceau* (*bucket* en anglais) car plusieurs points différents peuvent « tomber » dans le même sceau, c'est-à-dire avoir la même valeur de hachage relativement à la fonction g.

Nous disposons de deux ensembles de points : $\mathcal{R} = \{r_1, r_2, \cdots, r_m\}$ les points de référence et $\mathcal{Q} = \{q_1, q_2, \cdots, q_n\}$ les points requête. La première étape de l'algorithme consiste à construire L fonctions de hachage g_1, g_2, $\cdots$, g_L. Chaque point de référence r_i est ajouté aux sceaux $g_1(r_i)$, $g_2(r_i)$, $\cdots$, $g_L(r_i)$. Le nombre de sceaux possible étant trop important, les différents sceaux rencontrés sont stockés dans une table de hachage classique. Ensuite, pour chaque point requête q_j, l'algorithme collectionne les points de référence appartenant aux sceaux $g_1(q_j)$, $g_2(q_j)$, $\cdots$, $g_L(q_j)$. Afin d'optimiser le temps de calcul, l'algorithme parcourt ces différents sceaux jusqu'à obtenir $3L$ points de référence, répétitions incluses, où jusqu'à ce que les L fonctions g aient été parcourues. Dans cette version de l'algorithme, seuls les points collectionnés tels que $d(q_j, r_i) < R$ sont gardés. Cela permet de s'assurer que les kPPV sont à une distance inférieure de R des points requête. L'ultime étape consiste à calculer les distances entre q_j et les points de référence collectionnés. Les kPPV sont les points de référence (collectionnés) fournissant les k plus petites distances.

L'intérêt de l'utilisation de la méthode LSH réside dans le fait de ne devoir calculer les distances qu'avec un nombre très restreint de points de référence. Ceci permet à LSH d'être actuellement la méthode de recherche la plus rapide en haute dimension. Cependant, cette méthode présente quelques inconvénients pour des applications de traitement d'images. Premièrement, la méthode LSH est intéressante par rapport à une méthode de type kd-tree pour une très haute dimension et pour un grand nombre de points. Le gain observé est de l'ordre de 30 si 100000 points de dimension 500 sont utilisés. En traitement d'images, les dimensions rencontrées dépassent rarement 100 (généralement <20). Le gain est alors assez faible (de l'ordre de 5). Deuxièmement, l'étude réalisée par les auteurs ne tient compte que de la partie recherche, la création de l'ensemble des sceaux étant pré-calculée. Ce pré-calcul n'étant pas toujours possible en traitement d'images, le temps nécessaire au calcul des sceaux (souvent très long) s'ajoute au temps de recherche. Par conséquent, la méthode LSH peut être plus lente qu'une méthode de type kd-tree pour la recherche de kPPV.
Enfin, la méthode LSH ne garantit pas que les k points de référence sélectionnés soient effectivement les kPPV pour un point requête donné. Les paramètres v et L doivent être choisis de manière à minimiser le temps de calcul tout en garantissant une qualité minimale des kPPV sélectionnés. Dans [DIIM04], les auteurs utilisent $v = 10$ et $L = 30$ pour toutes leurs expérimentations.

§ 7.3 RÉSUMÉ

La recherche des k plus proches voisins, notée kPPV, est un problème couramment rencontré dans différents domaines du traitement d'images. En particulier, la recherche des kPPV permet d'estimer efficacement des mesures statistiques telles que l'entropie de Shannon ou la divergence de Kullback-Leibler en haute dimension. La méthode la plus simple est la recherche exhaustive consistant à tester l'ensemble des points de référence pour ne retenir que les k plus proches. Cette méthode a pour inconvénient majeur d'être très lente. De nombreuses méthodes sont proposées pour accélérer le processus de recherche. Par exemple, l'utilisation d'un kd-tree permet de partitionner l'espace. La recherche des kPPV consiste alors à parcourir une partie de l'arbre (de l'espace) évitant de fait un grand nombre de calculs. La méthode LSH, fondée sur l'utilisation de fonctions de hachage, est à l'heure actuelle la méthode de recherche la plus rapide en haute dimension. Cependant, cette méthode est peu adaptée à des problèmes de traitement d'images. Il en résulte que des méthodes utilisant des kd-trees sont souvent plus rapides pour de telles applications.
Toutes ces méthodes sont adaptées à l'architecture des ordinateurs actuels utilisant les processeurs (CPU) standard pour effectuer des calculs. Cepen-

dant, la récente apparition de la programmation parallèle sur les processeurs graphiques (GPU) permet d'envisager de nouveaux algorithmes bien plus performants.

- CHAPITRE 8 -

IMPLÉMENTATION GPU

§ 8.1 MOTIVATIONS

Pendant de nombreuses années, les acteurs principaux de la conception de microprocesseurs que sont Intel et AMD se sont livrés une guerre, chacun développant des processeurs de plus en plus puissants. Brusquement, cette guerre s'arrête aux alentours de 2003 : la fréquence des processeurs stagne. Cette stagnation est simplement due à des limitations physiques. Certains parlent alors de la fin de la loi (ou plutôt conjecture) de Moore. Cette loi, formulée par Gordon E. Moore [Moo65] prévoit le doublement annuel des performances des circuits intégrés (mémoires et processeurs).
Un processeur graphique (en anglais GPU pour *Graphics Processing Unit*) est un microprocesseur présent sur les cartes graphiques des ordinateurs ou des consoles de jeux vidéo. Une partie du travail habituellement exécutée par le processeur principal est déléguée au processeur graphique qui se charge des opérations d'affichage et de manipulation de données graphiques. En l'espace de quelques années, les GPUs ont évolué pour devenir de très puissants outils de calcul comme peut en témoigner la figure 8.1. Embarquant plusieurs microprocesseurs et une très importante bande passante mémoire, les GPUs actuels offrent une incroyable ressource pour des calculs aussi bien graphiques que non graphiques [RK05, DKP05, BK06, Nie08, Nie06b, Nie06a]. La principale raison se cachant derrière une telle évolution vient du fait que le GPU est spécialisé dans le calcul hautement parallèle : une même opération est effectuée en parallèle sur plusieurs données. Les transistors sont principalement dédiés au calcul des données plutôt que dans la mise en mémoire cache et dans le contrôle du flux des données (voir figure 8.2). De nombreuses applications de traitement d'images peuvent profiter de ce modèle de programmation pour grandement accélérer leur vitesse de calcul. Nous pouvons par exemple citer le rendu de

scènes 3D, l'encodage et le décodage vidéo, la stéréovision, etc. De même, de nombreux algorithmes appartenant à des domaines complètement extérieurs au traitement et au rendu d'images peuvent être accélérés en utilisant le calcul parallélisé. Citons par exemple le traitement du signal ou encore les simulations physiques appliquées à la finance ou à la biologie.

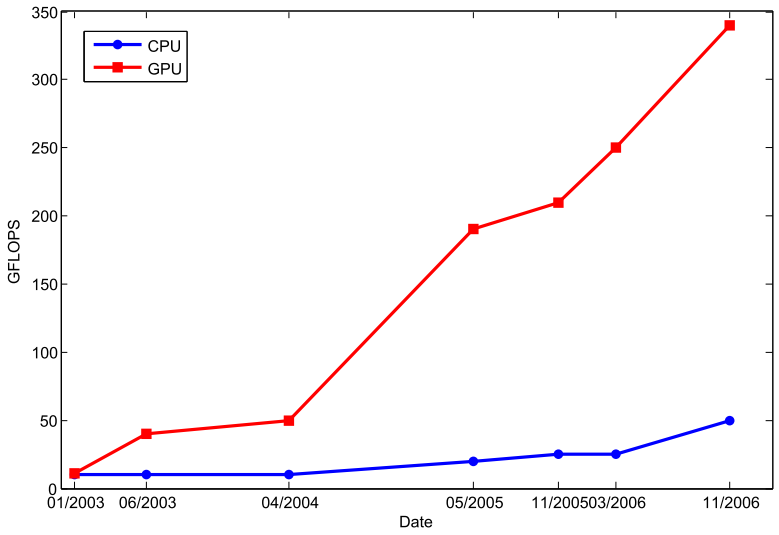


FIGURE 8.1 – *Comparaison du nombre de GFLOPS pour un CPU et un GPU. Le GFLOPS, pour « Giga FLoating point Operations Per Second », est une unité de mesure de la vitesse de calcul d'une machine, égale à un milliard d'opérations en virgule flottante par seconde. Figure inspirée du guide de programmation CUDA de NVIDIA.*

§ 8.2 NVIDIA CUDA

8.2.1 Généralités sur l'API

Jusqu'à maintenant, l'accès à la puissance du calcul parallèle par l'intermédiaire d'un GPU pour des applications non graphiques demandait d'être très astucieux. En effet, un GPU ne pouvait être programmé que par l'intermédiaire de l'API graphique (API pour *Application Programming Interface*, ou, en français, interface de programmation), cette API étant très peu adaptée à des applications non graphiques. De plus, la mémoire du GPU pouvait être lue mais ne pouvait pas être écrite ce qui réduisait considérablement la flexibilité de certains programmes. Fort de ce constat, NVIDIA a sorti le 15 février 2007 l'API CUDA.
CUDA, qui signifie *Compute Unified Device Architecture*, est l'architecture

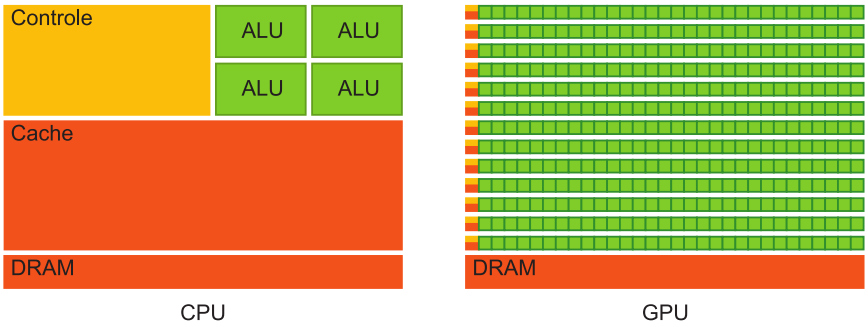


FIGURE 8.2 – *Le GPU alloue plus de transistors (représentés par ALU pour « Arithmetic Logic Unit » sur la figure) au calcul des données que le CPU. Figure inspirée du guide de programmation CUDA de NVIDIA.*

qui permet d'exploiter les capacités de calcul des GPUs en leur faisant traiter des kernels (programmes) sur un certain nombre de threads. Si CUDA englobe également en partie le GPU puisque celui-ci dispose de plus en plus d'optimisations destinées à faciliter le calcul non graphique, il s'agit en pratique principalement de la partie logicielle. CUDA est donc représenté par un driver, un runtime, des librairies (une implémentation de BLAS notamment), une API basée sur une extension du langage C et le compilateur qui va avec (qui redirige la partie non-exécutée sur le GPU vers le compilateur classique par défaut du système). L'API CUDA est de type haut niveau, c'est-à-dire qu'elle fait globalement abstraction du matériel bien que prendre en compte ses spécificités soit requis pour obtenir un rendement intéressant. En résumé, CUDA est une API de haut niveau qui permet de développer « facilement » des applications tirant partie de la puissance du GPU.
Dans la suite, nous expliquons les concepts que nous jugeons fondamentaux pour comprendre le travail et les choix qui ont été effectués. Ces explications n'ont pas pour but d'être exhaustives et ne permettent donc pas d'appréhender l'intégralité de l'API CUDA. Pour une description complète de CUDA, le lecteur doit se référer au guide de programmation CUDA fournit par NVIDIA [1].

8.2.2 Vocabulaire

L'utilisation de l'API CUDA fait intervenir un ensemble de concepts au vocabulaire spécifique qu'il est nécessaire de définir. Le vocabulaire est principalement anglophone. Il est souvent assez difficile de traduire certains de ces termes en français pour les mêmes raisons qu'il est impossible

1. http ://www.nvidia.com

de traduire le mot *bug* en français sans parler de son origine historique [2]. Dans la suite, nous traduirons les termes que nous considérons comme fidèlement traduisible en français. Les autres termes seront utilisés dans leur langue d'origine.

Une carte graphique de type NVIDIA (GeForce, Quadro, ou Tesla) supportant CUDA est un ensemble de multiprocesseurs, un multiprocesseur étant un ensemble de processeurs graphiques. Par exemple, la GeForce 8800GTX compte 16 multiprocesseurs, chaque multiprocesseur contenant 16 processeurs graphiques ce qui fait au total 128 processeurs graphiques. Chaque multiprocesseur a une architecture SIMD (de l'anglais *Single Instruction on Multiple Data*) : à chaque cycle d'horloge, l'ensemble des processeurs traite une même instruction pour des données différentes. Chaque multiprocesseur possède 4 catégories de mémoire différentes. Ces catégories seront détaillées dans la section 8.2.4. La composition matérielle d'une carte graphique est en réalité plus complexe. Cette complexité n'apportant que peu à l'étude de l'utilisation de CUDA pour le calcul scientifique, nous ne détaillerons pas plus cette composition matérielle.

La programmation parallèle consiste à exécuter un processus élémentaire simultanément sur plusieurs unités de calcul (par exemple des processeurs graphiques), chaque unité traitant des données différentes. Dans CUDA, ces processus sont appelés *threads*, terme que l'on peut traduire en français par *processus léger*, la traduction littérale étant *fil*. Chaque thread exécute un *kernel* (en français *noyau*), un kernel étant simplement la compilation d'un programme sous forme d'instructions.

Un *warp* est un ensemble de 32 threads. En CUDA, il s'agit du nombre de threads minimal traités de façon parallèle. Un multiprocesseur contient 16 processeurs. Les threads d'un warp sont exécutés en deux temps, par groupe de demi-warps (16 threads). Le terme de *warp* vient des machines à tisser, il désigne un ensemble de fils de cotons en analogie au terme *thread*.

L'exécution d'un programme écrit en CUDA fait intervenir une partie CPU et une partie GPU. Ces deux parties travaillant de concert, une nomenclature particulière est employée pour clairement identifier chacune d'entre elle. Le CPU est appelé *host* et le GPU *device*. Une variable stockée sur la mémoire de la carte mère ou une fonction C classique est accessible depuis l'*host*. De même, une variable stockée dans la mémoire de la carte graphique ou une fonction CUDA (kernel) est accessible depuis le *device*.

2. Le mot *bug*, signifiant *insecte* en français, a été introduit par Hopper en 1946 suite à une erreur dans le logiciel Harvard Mark III causée par un papillon de nuit pris dans un relais. Le terme désigne aujourd'hui une erreur dans un programme informatique.

8.2.3 Programmation sur GPU

En CUDA, le programmeur ne manipule pas directement les threads ou les warps. Les threads sont regroupées en ce qu'on appelle un bloc de threads (en anglais, *thread block*). Plus précisément, un bloc de threads est un ensemble de threads exécutées sur un même multiprocesseur et pouvant partager des données via l'utilisation d'une mémoire particulière et extrêmement rapide appelée *mémoire partagée*. Un bloc de threads a pour dimension maximum $512 \times 512 \times 64$ tout en ayant pour contrainte de contenir au maximum 512 threads. Chaque thread d'un bloc est identifié par son *thread ID*. Cet identifiant s'exprime par un ensemble de coordonnées 2D ou 3D selon la dimension du bloc employée. Par exemple, pour un bloc bidimensionnel de taille (B_x, B_y), l'identifiant du thread (t_x, t_y) est égal à $t_x + t_y B_x$.

Une carte graphique disposant de plusieurs multiprocesseurs, il parait judicieux d'exécuter plusieurs blocs de threads simultanément de manière à optimiser l'occupation des multiprocesseurs. Pour ce faire, une grille de blocs est définie. Cette grille est obligatoirement bidimensionnelle et a pour dimension maximale 65535×65535. Chaque élément de la grille est un bloc de threads pouvant être 1D, 2D, ou 3D. De la même manière que les threads dans le bloc sont identifiés par leur *thread ID*, chaque bloc est identifié par son *block ID*. Pour une grille de taille (G_x, G_y), l'identifiant du bloc (b_x, b_y) est égal à $b_x + b_y G_x$.

Un exemple clair valant mieux qu'un long discours, étudions l'utilisation de CUDA pour exécuter une opération basique sur une matrice 2D. Soit M, une matrice 2D de taille 128×128. En CUDA, la première dimension représente toujours les colonnes, et la seconde les lignes. L'opération que nous considérons est l'ajout du le nombre 2 à chaque élément de la matrice. Nous définissons une grille de taille 8×8 contenant des blocs de tailles 16×16. Comme $16 * 8 = 128$, chaque thread de la grille correspond à un élément de M. La parallélisation est donc totale. Nous définissons ensuite une fonction kernel qui va ajouter 2 à chaque élément de M. Pour identifier l'élément de M à traiter, le kernel a accès comme nous l'avons précédemment vu à l'indice du bloc dans la grille et à l'indice du thread dans le bloc :

- `blockIdx` : indice du bloc dans la grille
- `blockDim` : taille de chaque bloc
- `threadIdx` : indice du thread dans le bloc

Ce kernel est lié à la dimension de la grille et à celle des blocs définis précédemment. Sachant que la matrice M a pour taille 128×128, l'indice (x,y)

de l'élément considéré est calculé comme suit :

$$x = \texttt{blockIdx}.x * \texttt{blockDim}.x + \texttt{threadIdx}.x$$
$$y = \texttt{blockIdx}.y * \texttt{blockDim}.y + \texttt{threadIdx}.y$$

Ainsi, le kernel identifie simplement l'élément de M à traiter. Ce kernel lit en mémoire l'élément $M(x, y)$, ajoute 2 à cette valeur, et écrit la nouvelle valeur à la même position mémoire. La figure 8.3 montre l'organisation de plusieurs kernels exécutés sur le device. Une taille de grille et une taille de bloc sont définies pour chaque kernel à exécuter.
Le kernel est exécuté pour chaque thread défini. Dans le cas précédant, 64 blocs de 256 threads sont définis. Pour une carte graphique contenant 8 mutiprocesseurs, seuls 8 blocs pourront être traités simultanément. De même, si chaque multiprocesseur contient 16 processeurs graphiques, les 128 threads sont traitées séquentiellement par groupe de 16. La parallélisation, gérée par CUDA, dépend des ressources de la carte graphique ce qui rend la programmation très simple. L'intérêt de l'utilisation de CUDA réside dans la transparence de l'implémentation. Un code CUDA permet donc de cibler à la fois les GPUs d'entrée de gamme, les GPUs haut de gamme, et même les futurs GPUs. Il est toutefois important de noter que, pour certaines applications, la programmation doit prendre en compte les spécifications techniques de la carte pour espérer une optimisation maximale.

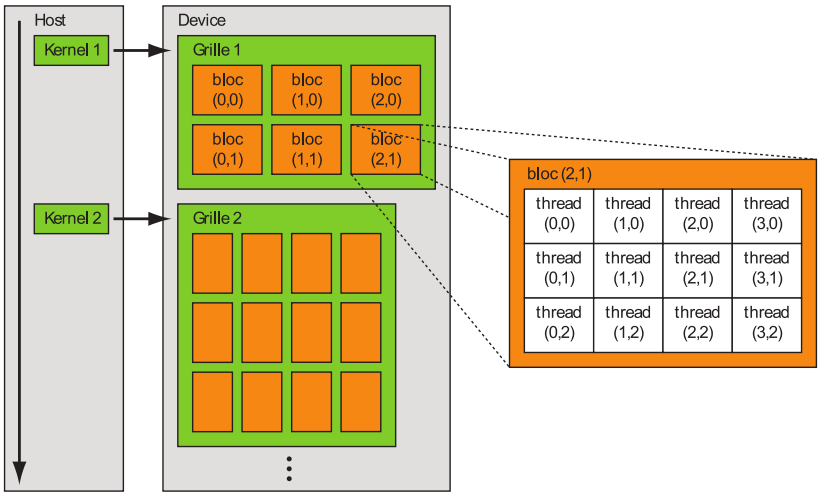


FIGURE 8.3 – *Pour une fonction kernel donnée, le programmeur définit une taille de grille et une taille de bloc. Le CPU (host) exécute le kernel pour chaque thread de la grille en répartissant les calculs sur les multiprocesseurs libres (device). Un kernel a accès à l'indice du thread considéré par l'intermédiaire de l'indice de bloc et de l'indice du thread. Figure inspirée du guide de programmation CUDA de NVIDIA.*

8.2.4 Mémoires

Un thread a accès à la DRAM (*Dynamic Random Access Memory*) et à la mémoire du processeur à travers un ensemble d'espaces mémoire de taille différente :

- Registres : lecture/écriture pour un thread
- Mémoire locale : lecture/écriture pour un thread
- Mémoire partagée : lecture/écriture pour un block
- Mémoire globale : lecture/écriture pour la grille
- Mémoire constante : lecture seule pour la grille
- Mémoire texture : lecture seule pour la grille

Les données écrites/lues par un thread sont propres à ce thread. La mémoire globale, la mémoire constante, et la mémoire texture peuvent être lues et écrites par le CPU. De plus, ces mémoires sont persistantes au travers des différents kernels appelés par une même application. Ces trois mémoires sont optimisées pour des usages différents.
Dans ce qui suit, nous donnons quelques détails sur la mémoire globale, la mémoire partagée et la mémoire texture. Une fois encore, ces explications ne sont pas exhaustives. Nous donnons dans la suite les clés qui vont permettre d'expliquer les choix que nous avons faits. La lecture complète du guide de programmation fourni par NVIDIA [3] doit être réalisée pour parfaitement comprendre l'utilisation des différentes mémoires disponibles.

8.2.4.1 Mémoire globale

En CUDA, une matrice bidimensionnelle de n colonnes et m lignes est toujours stockée dans la mémoire globale sous forme d'un tableau de taille mn. Pour profiter d'une bande-passante maximale, l'accès à la mémoire globale doit respecter certaines conditions. La notion de coalescence est ici primordiale. Un warp contient 32 threads et, par conséquent, un demi-warp contient 16 threads. Il est dit que la lecture ou l'écriture est coalescente si les threads du demi-warp accèdent à des zones de mémoire contiguës et alignées. Plus précisément, dans chaque demi-warp, si le premier thread d'un demi-warp a pour adresse `DemiWarpAdresse`, le thread numéro N devrait avoir pour adresse `DemiWarpAdresse` + N. La figure 8.4 illustre un exemple d'accès coalescents tandis que la figure 8.5 donne un exemple d'accès non coalescents.

3. http ://www.nvidia.com

L'alignement est une autre notion fondamentale. Considérons un tableau de flottants stocké dans la mémoire globale. L'adresse du premier élément de ce tableau est notée `AdresseTableau`. L'alignement est vérifié si et seulement si `DemiWarpAdresse - AdresseTableau` est un multiple de 16∗`tailleFlottant` où `tailleFlottant` est le nombre d'octets nécessaires pour stocker un nombre flottant (typiquement 4 octets). De manière générale, un tableau bidimensionnel de taille quelconque ne satisfait pas la condition d'alignement si la grille définie correspond effectivement à la taille de ce tableau. Il est souvent préférable de « gaspiller » de la mémoire en allouant plus de colonnes que nécessaire et en recopiant les données sur une partie de la mémoire allouée. CUDA fournit un ensemble de fonctions qui permettent de réaliser ces opérations de manière relativement transparente. Pour NVIDIA, un code devrait toujours utiliser ces fonctions pour manipuler des matrices 2D. En effet, le gain obtenu avec ces fonctions est de l'ordre de 10 pour un même kernel. Toutefois, l'utilisation de la fonction de copie des données sur une partie de la mémoire limite le nombre de colonnes des matrices que l'on peut traiter. Pour être plus exact, la limite se situe au niveau du nombre d'octets que peut comporter chaque ligne de la matrice. Cette limites est de $2^{18} = 262144$ octets. Si l'on considère des nombres flottants à simple précision codés sur 4 octets, les matrices sont alors limitées à 65536 colonnes.

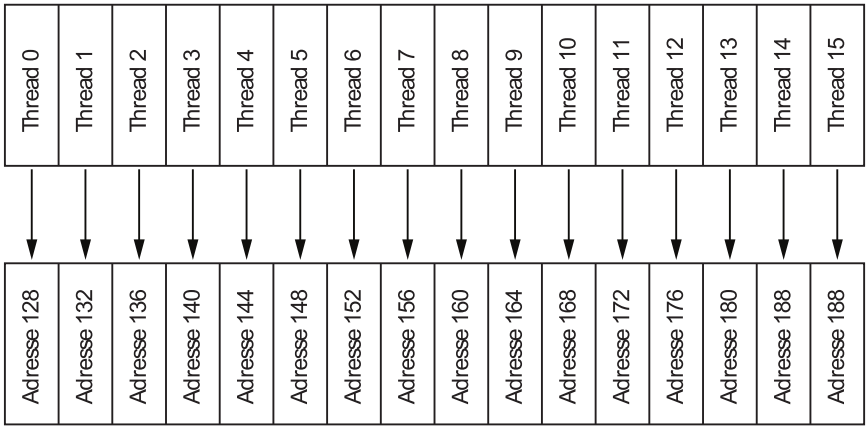


FIGURE 8.4 – *Exemple d'accès coalescents à la mémoire. Les données lues ici sont des nombres flottants codés sur* 4 *octets. Les* 16 *threads vont lire des zones mémoires contiguës.*

8.2.4.2 Mémoire partagée

La mémoire partagée est une mémoire d'une extrême rapidité, aussi rapide dans certains cas que la mémoire locale. La mémoire est partagée pour l'ensemble des threads d'un même bloc. Cette mémoire est particuliè-

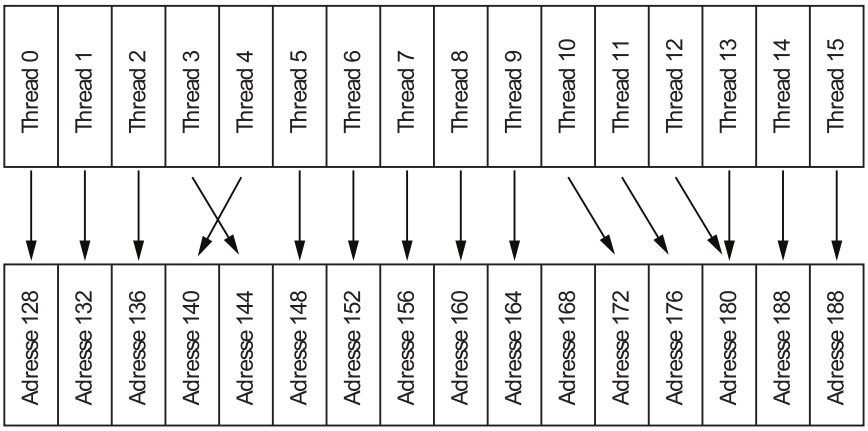


FIGURE 8.5 – *Exemple d'accès non coalescents à la mémoire. Les données lues ici sont des nombres flottants codés sur* 4 *octets. Les* 16 *threads vont lire des zones mémoires non contiguës. La bande-passante maximale ne sera pas atteinte.*

rement intéressante à utiliser dans les cas où les accès à la mémoire globale ne sont pas coalescents. En effet, il est quelquefois possible d'utiliser cette mémoire pour rendre les accès coalescents. Un exemple est la transposition de matrices. La lecture est coalescente alors que l'écriture ne l'est pas. En utilisant la mémoire partagée, la transposition de chaque bloc est réalisée dans la mémoire partagée qui n'est pas contrainte en lecture/écriture. L'écriture dans la mémoire globale est finalement réalisée de manière coalescente.

8.2.4.3 Mémoire texture

La texture est un objet essentiel pour une carte graphique. Une texture est généralement définie comme étant un ensemble de pixels 2D (matrice de pixels) que l'on va appliquer sur une surface ou un volume 3D. Les jeux vidéos utilisent massivement les textures pour créer des univers 3D réalistes. Les accès à la mémoire texture sont implémentés de manière matérielle dans le but d'être extrêmement rapides. L'API CUDA permet l'utilisation des textures. Les textures possèdent des avantages et des inconvénients par rapport à l'utilisation de la mémoire globale.
En CUDA, les textures peuvent être 1D ou 2D. Un pixel y est localisé par ses coordonnées x si 1D ou (x,y) si 2D. Les textures sont optimisées pour des localisations 2D. Une texture peut être lue et écrite depuis une fonction classique CPU (host). L'accès à une texture est en lecture seule depuis une fonction kernel (device). Le principal avantage de la texture par rapport à la mémoire globale est que les lectures depuis un kernel ne sont pas (peu) pénalisées si les accès ne sont pas coalescents. Une texture 2D est néanmoins limitée en taille : $2^{16} = 65536$ octets de larges et $2^{15} = 32768$ octets de

hauteur. Si nous considérons une texture contenant des nombres flottants à simple précision codés sur 4 octets, une texture est alors limitée à 16384 colonnes et 8192 lignes. Ces limites sont rarement atteintes pour des applications classiques ou les textures dépassent rarement le millier de pixels de large et de hauteur. En revanche, pour des applications non usuelles telles que nous cherchons à développer, ces limites peuvent être facilement atteintes.

§ 8.3 RECHERCHE DES KPPV EN CUDA

La méthode de recherche des plus proches voisins que nous implémentons en GPU est la méthode exhaustive notée BF (pour *brute-force*). Ce choix s'explique par la grande faculté de parallélisation de l'algorithme BF. Rappelons que les deux principales étapes sont premièrement le calcul des distances, et deuxièmement le tri des distances calculées. Nous expliquons dans cette partie l'implémentation GPU de ces deux étapes. Notez que le code CUDA est téléchargeable sur Internet (cf. annexe C).

8.3.1 Calcul des distances

Soit $\mathcal{R} = \{r_1, r_2, \cdots, r_m\}$ un ensemble de m points de référence de dimension d et soit $\mathcal{Q} = \{q_1, q_2, \cdots, q_n\}$ un ensemble de n points requête de dimension d (« $\mathcal{R}$ » pour *reference* et « $\mathcal{Q}$ » pour *query* respectivement signifiant en anglais *référence* et *requête*).
Au total, mn distances doivent être calculées. Nous créons une matrice de m lignes et n colonnes que nous appelons *matrice de distances* et que nous stockons dans la mémoire globale. Nous définissons la fonction kernel qui va calculer chaque distance en parallèle. Pour ce faire, nous définissons des blocs de threads et une grille 2D de manière à ce qu'il y ait exactement un thread par distance. Un bloc de threads de taille 16×16 va faire intervenir 16 points requête et 16 points référence. La figure 8.6 illustre la matrice de distances, un bloc de thread, ainsi que les points requête et points référence utilisés pour le calcul des distances du bloc considéré. Pour des raisons de compréhension visuelle, les points de référence sont transposés sur la figure 8.6. Les ensembles de points semblent y être de dimension 1. En réalité, ces points peuvent être de dimension quelconque à condition évidemment que cette dimension soit commune aux deux ensembles.

Considérons que $\mathcal{R}$ et Q soient stockés dans la mémoire globale sous forme de matrice 2D respectivement de taille $m \times d$ et $n \times d$ (colonnes $\times$ lignes). Dans la figure 8.6, le bloc considéré a pour taille 4×4 de manière à rendre la figure plus claire. Pour garantir l'accès coalescent, les blocs considérés doivent avoir au minimum 16 threads de large (demi-warp). Ainsi,

nous utilisons des blocs de taille 16×16. Un demi-warp correspond à une ligne d'un bloc. Le calcul des distances d'un demi-warp fait intervenir 16 points requêtes et 1 point référence. Il en résulte que la lecture des requêtes est coalescente tandis que la lecture des références ne l'est pas. L'utilisation de la mémoire partagée permet de résoudre ce problème. Nous définissons une fonction kernel qui va (1) lire de manière coalescente les 16 points référence et 16 points requête et les stocker dans la mémoire partagée, (2) calculer les distances à partir des informations stockées dans la mémoire partagée, et (3) écrire les distances dans la mémoire globale de manière coalescente. L'accès à la mémoire partagée n'étant pas pénalisée par des accès aléatoires, le gain que nous observons pour le temps imparti au calcul des distances et de l'ordre de 10.

Nous avons utilisé la mémoire partagée pour palier au fait que la lecture des points de référence était non coalescente. La mémoire texture semble également tout indiquée pour résoudre ce problème. Nous stockons ainsi les points de référence dans la mémoire texture et les points requête dans la mémoire globale. Nous définissons alors une nouvelle fonction kernel qui va (1) lire les points requête dans la mémoire globale de manière coalescente, (2) lire les points référence dans la mémoire texture, (3) calculer les distance, (4) écrire les distances dans la mémoire globale de manière coalescente.
Cette méthode possède deux avantages. Premièrement, le gain que nous observons en termes de temps de calcul se situe entre 2 et 10 selon le nombre de points et la dimension considérés par rapport à la précédente approche. Deuxièmement, le code est beaucoup plus simple car il n'est pas nécessaire d'utiliser la mémoire partagée. L'inconvénient majeur réside dans le fait que la limite du nombre de points passe, pour des nombres flottants codés sur 4 octets, de 65536 points en mémoire globale à 16384 en mémoire texture comme indiqué dans la section 8.2. Cette limite peut être facilement atteinte selon l'application visée. Il faut aussi noter que la dimension est, dans ce cas, limité à 8192. Cependant, une telle dimension n'est jamais rencontrée en pratique.

Pour palier à ce problème, nous testons en début de programme le nombre de points de référence et la dimension des points. Si ce nombre est inférieur à 16384 et si la dimension est inférieure à 8192, nous utilisons la méthode basée sur la mémoire texture, sinon, nous utilisons celle basée sur la mémoire globale et la mémoire partagée. Dans tous les cas, le nombre de points de référence et de points requête est limitée à 65536.
Il est possible de s'extraire des limites de la carte graphique et de n'être limité que par la mémoire de l'ordinateur. Cependant, le code deviendrait beaucoup plus complexe. Dans nos différentes applications, 65536 points sont suffisants. Il ne nous a pas semblé utile de complexifier d'avantage

notre code, le but initial étant d'explorer la puissance de calcul introduite par la programmation sur GPU.

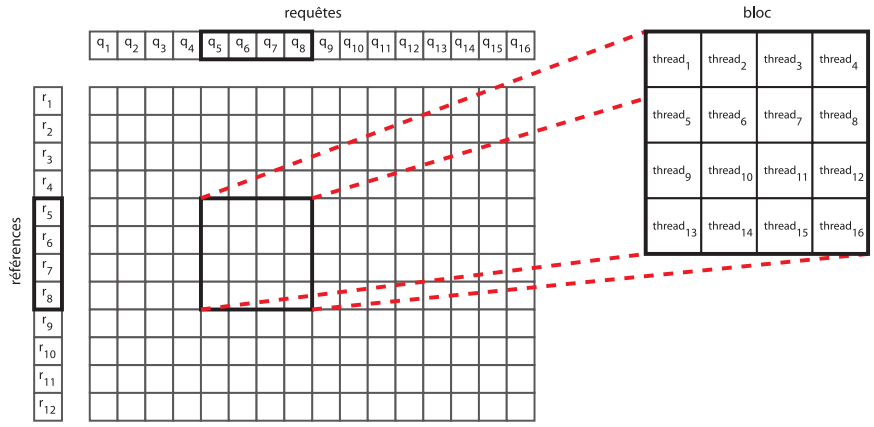


FIGURE 8.6 – *Organisation du calcul des distances. Pour un bloc de threads donné, le calcul des distances fait intervenir un sous-ensemble des points de référence et des points requête.*

8.3.2 Tri des distances

La seconde étape de la méthode BF est la tri des m distances calculées pour chaque point requête. Nous définissons une grille et des blocs de threads (comportant 1 ligne et 256 colonnes) de manière à ce que chaque thread traite une colonne de la matrice de distances. Ainsi, chaque thread trie l'ensemble des distances calculées pour un point requête donné par l'intermédiaire d'une fonction kernel. Les kPPV de chaque point requête sont les points référence qui donne les k plus petites distances. Si seule la distance des kPPV est nécessaire, il suffit de trier le tableau des distances. En revanche, si l'indice des kPPV est nécessaire, les opérations de lecture et d'écriture réalisées par le tri doivent également être réalisées sur une matrice d'indices, c'est-à-dire la matrice identifiant une distance calculée au point de référence correspondant.
Le choix du tri utilisé pour la fonction kernel est crucial. La littérature scientifique relative aux algorithmes de tri est très importante. Tout d'abord, précisons qu'une fonction kernel ne permet pas la récursivité. Ceci exclut par conséquent le tri rapide (*quick sort* [Hoa61]) et le tri fusion (*merge sort*). Le tri que nous avons implémenté en premier lieu au sein d'une fonction kernel est le tri de Dobosiewicz [Dob80] également appelé en anglais *comb sort*. Ce tri est une amélioration du tri de Shell [She58], lui même étant une amélioration du tri par insertion. Ce tri a la particularité d'être rapide et simple à implémenter.

Considérons un ensemble de 10000 points référence et une valeur de $k = 10$. Le tri de Dobosiewicz va trier l'ensemble du tableau alors que seules les 10 plus petites distances n'ont d'intérêt par rapport à notre problème. Si les 10 plus petites valeurs sont triées et regroupées dans la partie supérieure de la matrice de distances (10 premières lignes), le fait d'avoir la partie restante (lignes 11 à 10000) non triées n'a aucune incidence sur l'algorithme BF. Partant de ce constat, nous proposons une amélioration du tri par insertion. Pour une distance donnée, la comparer à la k-ème distance ; si elle est plus grande, ne rien faire ; si elle est plus petite, insérer la distance considérée dans les k-1 premières distances comme pour un tri par insertion classique. La différence majeure réside dans le fait qu'au maximum k-1 éléments sont décalés. La plupart des distances testées ne seront ainsi pas insérées ce qui évite de nombreuses étapes de lecture et d'écriture.
La figure 8.7 montre le temps attribué au tri en fonction du paramètre k pour le tri de Dobosiewicz et pour le tri par insertion amélioré. Pour cette expérimentation, 4800 points référence et requête de dimension 64 étaient utilisés. Le tri de Dobosiewicz est constant en temps de calcul dans la mesure où, quelle que soit la valeur de k, l'intégralité du tableau est trié. En ce qui concerne le tri par insertion amélioré, nous remarquons que sa durée augmente linéairement avec la valeur de k. En effet, le nombre de décalages augmente avec k.
Notons k_0 la valeur de k pour laquelle les deux tris sont équivalents en termes de durée (i.e. les deux courbes se coupent). Cela signifie que pour k inférieur à k_0, le tri par insertion amélioré est le plus rapide, et pour k supérieur à k_0, le tri de Dobosiewicz est le plus rapide. Dans cet exemple, $k_0 = 120$. La valeur de k_0 dépend principalement du nombre de points de référence et de la carte graphique utilisée. La figure 8.8 montre l'évolution de k_0 en fonction du nombre points de référence pour une carte graphique NVIDIA GeForce 8800 GTX. Il apparaît que k_0 augmente linéairement avec le nombre de points. Il n'est pas possible de déterminer quel sera le tri le plus rapide pour une configuration donnée (matériel informatique et données du problème). Nous avons toutefois estimé l'équation qui permet de déterminer dans notre cas la valeur de k_0 en fonction du nombre de points :

$$k_0(n) = 0.0247n + 1.3404 \tag{8.1}$$

où n désigne le nombre de points (référence et requête). Cette équation, estimée par régression linéaire, n'est valide que pour l'ordinateur sur lequel les tests ont été réalisés.
Dans nos applications, nous considérons de petites valeur de k (typiquement $k \leq 20$) et des ensembles qui contiennent plusieurs milliers de points. Dans ce cas, la figure 8.8 nous indique que le tri par insertion amélioré est toujours le choix de tri le plus judicieux. Ce tri est celui qui est intégré à notre méthode de recherche des kPPV implémentée en CUDA.

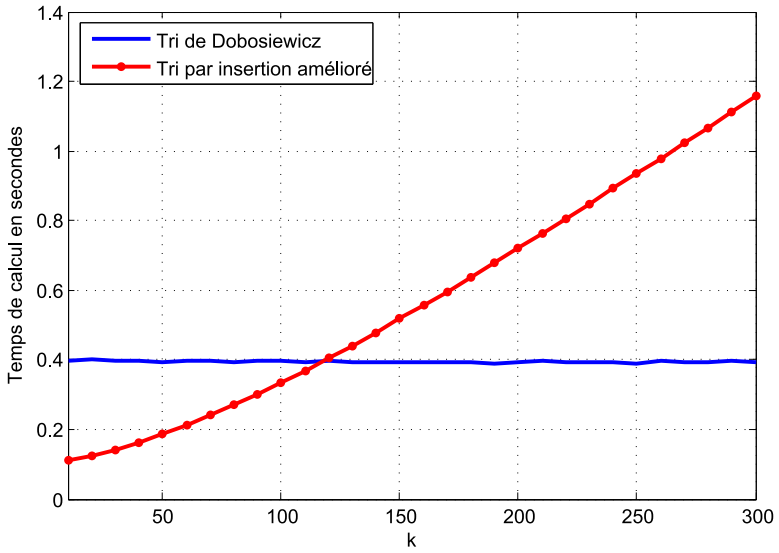


FIGURE 8.7 – *Évolution du temps de calcul en fonction du paramètre k pour le tri de Dobosiewicz (ligne bleue) et le tri par insertion amélioré (ligne rouge). 4800 points (référence et requête) de dimension 64 sont utilisés. Le temps de calcul est constant pour le tri de Dobosiewicz et augmente linéairement avec k pour le tri par insertion. Les deux algorithmes de tri sont équivalents en termes de temps de calcul pour* $k = k_0 = 120$.

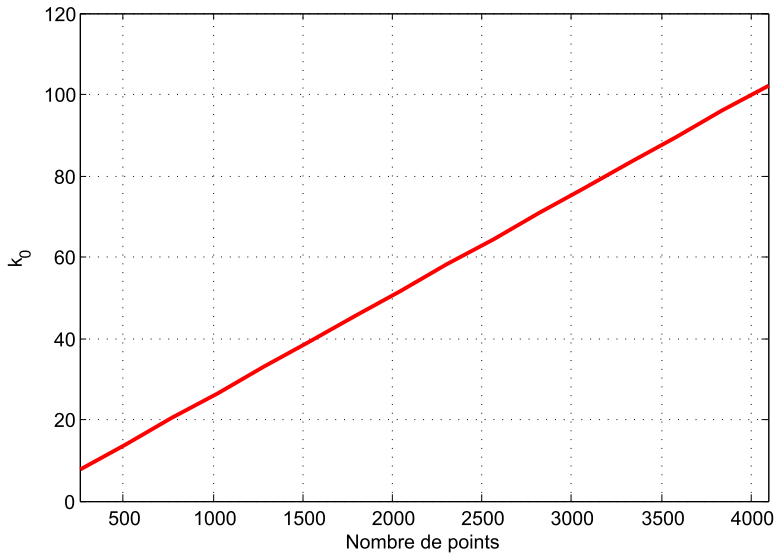


FIGURE 8.8 – *Évolution de* k_0 *en fonction du nombre de points (référence et requête). La dimension des points utilisée est 64. En dessous de cette ligne, le tri par insertion amélioré est le plus rapide. Au dessus de cette ligne, le tri de Dobosiewicz est le choix le plus intéressant.*

§ 8.4 RÉSUMÉ

La programmation sur GPU représente une (r)évolution très importante dans le domaine du traitement d'images et, plus généralement, dans le calcul scientifique. L'API NVIDIA CUPA permet de développer simplement des applications graphiques et non graphiques tirant profit de la puissance de calcul des GPUs. Dans ce chapitre, nous avons proposé une implémentation CUDA de la recherche des kPPV. La méthode implémentée est la recherche exhaustive (BF) qui a comme principal intérêt d'être entièrement parallélisable. Cette propriété est cruciale en programmation GPU ; l'implémentation d'un algorithme non parallélisable ou peu parallélisable n'a aucun intérêt en programmation GPU. L'implémentation de la méthode exhaustive fait intervenir deux fonctions CUDA : la première calculant toutes les distances, la seconde triant ces distances afin de trouver les kPPV. Selon le nombre de points de référence, le calcul des distances utilise la mémoire texture ou la mémoire partagée afin d'optimiser le temps de calcul. Enfin, il apparaît que le tri par insertion amélioré donne les meilleurs résultats en termes de temps de calcul pour des valeurs de k usuelles.

- CHAPITRE 9 -

EXPÉRIMENTATIONS ET APPLICATION AU SUIVI D'OBJETS

§ 9.1 RÉSULTATS SUR DONNÉES SYNTHÉTIQUES

9.1.1 Contexte théorique

Le but initial de l'étude était d'accélérer la recherche des kPPV au sein du logiciel Matlab pour différentes applications, applications que nous verrons dans la suite. Il est assez connu que certains algorithmes sont très lents sous Matlab. Pour remédier à ce défaut, il est possible d'utiliser depuis Matlab des fichiers C compilés pour Matlab. Ces fichiers sont connus sous le nom de *mex-files* (fichiers *mex*, abréviation de *MATLAB Executable*). La plupart des fonctions proposées par Matlab telles que la multiplication de matrices sont en réalité des mex-files. De la même manière, NVIDIA a récemment proposé un plug-in qui permet d'utiliser dans Matlab des fichiers CUDA. Ces fichiers sont compilés de manière très semblable à celle des mex-files.
Dans cette section, nous allons comparer le temps d'exécution de quatre méthodes de recherche des kPPV. Les données sur lesquelles la recherche est effectuée sont des données synthétiques tirées aléatoirement selon une loi uniforme $\mathcal{U}(0,1)$. La dimension et le nombre de points de chaque ensemble (références et requêtes) sont variables et seront spécifiés par la suite. Les méthodes de recherche des kPPV comparées sont les suivantes :

- BF implémentée en Matlab (notée BF-Matlab)
- BF implémentée en C (mex-file) (notée BF-C)
- BF implémentée en CUDA (notée BF-CUDA)
- Librairie ANN implémentée en C++ (notée ANN-C++)

Ainsi, nous comparons le temps d'exécution de la méthode BF implémentée en Matlab, C, et CUDA. Cependant, cette étude serait incomplète dans la mesure où la méthode BF est peu souvent utilisée (en C ou en Matlab) pour la recherche des kPPV du fait de sa lenteur. Par conséquent, nous ajoutons à cette étude l'une des méthodes les plus rapides connues à ce jour, à savoir la librairie ANN (pour *Approximate Nearest Neighbor*) implémentée en C++ [AMN+98, MA]. Cette librairie, extrêmement optimisée et fondée sur l'utilisation d'un kd-tree [Ben75, Ind04], supporte la recherche exacte et la recherche approchée des kPPV ; nous utilisons la version exacte.

9.1.2 Spécifications techniques

Pour cette étude, nous disposons d'un ordinateur de type PC dont la liste des principaux composants et logiciels utilisés est donnée ci-dessous :

- Processeur : Intel Pentium 4 cadencé à 3.4 GHz
- Mémoire vive : 2Go DDR2 PC2-5300 (4×512Mo dual-channel)
- Carte graphique : NVIDIA GeForce 8800 GTX (768Mo de mémoire GDDR3, 16 multiprocesseurs)
- Système d'exploitation : Microsoft Windows XP Professionnel
- Logiciel : Matlab 2007b
- Compilateur C : Microsoft Visual C++ Express Edition 2005
- Compilateur CUDA : NVIDIA CUDA 1.1

9.1.3 Temps de calcul

9.1.3.1 Tableau général

Le tableau 9.1 présente le temps de calcul, exprimé en secondes, de l'ensemble des méthodes et implémentations listées précédemment. Pour cette étude, les ensembles de points que sont les références et les requêtes sont de même taille et contiennent des données différentes. Dans le cas où les deux ensembles seraient identiques, le problème serait plus simple ; nous considérons le problème le plus dur.
Le temps de calcul dépend du nombre de points (noté n), de la dimension de ses points (notée d) et du paramètre k correspondant au nombre de voisins à considérer (kPPV). Boltz *et al.*[BDB07] ont montré que la valeur du paramètre k influait peu sur la qualité de leurs résultats. Ainsi, nous utilisons $k = 20$ pour la première partie de l'étude. Les valeurs de n et d utilisées sont des valeurs courantes que l'on retrouve dans plusieurs applications de traitement d'images.

Le principal résultat de cette étude est le fait que CUDA permette de grandement diminuer le temps de calcul nécessaire pour la recherche des kPPV. Selon le tableau 9.1, BF-CUDA est jusqu'à 407 fois plus rapide que

	Methodes	n=1200	n=2400	n=4800	n=9600	n=19200	n=38400
d=8	BF-Matlab	0.51	1.69	7.84	35.08	148.01	629.90
	BF-C	0.13	0.49	1.90	7.53	29.21	127.16
	ANN-C++	0.13	0.33	0.81	2.43	6.82	18.38
	BF-CUDA	**0.01**	**0.02**	**0.04**	**0.13**	**0.43**	**1.89**
d=16	BF-Matlab	0.74	2.98	12.60	51.64	210.90	893.61
	BF-C	0.22	0.87	3.45	13.82	56.29	233.88
	ANN-C++	0.26	1.06	5.04	23.97	91.33	319.01
	BF-CUDA	**0.01**	**0.02**	**0.06**	**0.17**	**0.60**	**2.51**
d=32	BF-Matlab	1.03	5.00	21.00	84.33	323.47	1400.61
	BF-C	0.45	1.79	7.51	30.23	116.35	568.53
	ANN-C++	0.39	1.78	9.21	39.37	166.98	688.55
	BF-CUDA	**0.01**	**0.03**	**0.08**	**0.24**	**0.94**	**3.89**
d=64	BF-Matlab	2.24	9.37	38.16	149.76	606.71	2353.40
	BF-C	1.71	7.28	26.11	111.91	455.49	1680.37
	ANN-C++	0.78	3.56	14.66	59.28	242.98	1008.84
	BF-CUDA	**0.02**	**0.04**	**0.11**	**0.40**	**1.57**	**6.65**
d=80	BF-Matlab	2.35	11.53	47.11	188.10	729.52	2852.68
	BF-C	2.13	8.43	33.40	145.07	530.44	2127.08
	ANN-C++	0.98	4.29	17.22	73.22	302.44	1176.39
	BF-CUDA	**0.02**	**0.04**	**0.13**	**0.48**	**1.98**	**8.17**
d=96	BF-Matlab	3.30	13.89	55.77	231.69	901.38	3390.45
	BF-C	2.54	10.56	39.26	168.58	674.88	2649.24
	ANN-C++	1.20	4.96	19.68	82.45	339.81	1334.35
	BF-CUDA	**0.02**	**0.05**	**0.15**	**0.57**	**2.29**	**9.61**

TABLE 9.1 – *Comparaison du temps de calcul, donné en secondes, pour les méthodes BF-Matlab, BF-C, ANN-C++ et BF-CUDA. BF-CUDA est jusqu'à* 407 *fois plus rapide que BF-Matlab,* 295 *fois plus rapide que BF-C et* 148 *fois plus rapide que ANN-C++.*

BF-Matlab, 295 fois plus rapide que BF-C et 148 fois plus rapide que ANN-C++. Ces différents gains sont peu révélateurs sous cette forme. À titre d'exemple, si nous considérons $n = 19200$ points en dimension $d = 96$, la recherche des kPPV prend approximativement 15 minutes avec BF-Matlab, 11 minutes avec BF-C, 5 minutes et 40 secondes avec ANN-C++ et 2.3 secondes avec BF-CUDA. Il est relativement complexe d'analyser simplement le tableau 9.1 dans son ensemble. Il parait plus simple de procéder par étape en étudiant séparément l'influence de la dimension, du nombre de points, et du paramètre k sur le temps de calcul.

9.1.3.2 Influence de la dimension

La figure 9.1 montre l'augmentation du temps de calcul en fonction de la dimension d pour des ensembles de $n = 4800$ points. Quelle que soit la dimension, BF-CUDA est la méthode la plus rapide. Le temps de calcul augmente linéairement avec la dimension d quelle que soit la méthode utilisée. Ce résultat semble logique dans la mesure où le nombre d'opérations nécessaires pour calculer la distance entre deux vecteurs dépend linéairement de la dimension. Bien que linéaire, l'influence de la dimension n'est pas égale entre les différentes méthodes et implémentations. Cette influence est exprimée par la pente de la fonction linéaire représentant le temps de calcul en fonction de la dimension. Cette pente, approximée par une méthode de régression linéaire, est calculée pour chacune des méthodes testées. Plus la pente est élevée, plus l'influence de la dimension est importante. Pour un ensemble de $n = 4800$ points et $k = 20$, cette pente est de 0.54 pour BF-Matlab, 0.45 pour BF-C, 0.20 pour ANN-C++, et est quasi nulle (0.001) pour BF-CUDA. ANN-C++ est moins sensible à la dimension que BF-Matlab et BF-C, et cette dimension a un impact quasiment négligeable sur BF-CUDA en comparaison des autres méthodes. En d'autres termes, plus la dimension est importante, plus l'utilisation de CUDA est intéressante. Cette particularité est très intéressante pour des applications où les dimensions utilisées sont élevées comme dans la recherche d'images par le contenu [ADPB08, PADB08]. La dimension y est généralement restreinte pour permettre une recherche rapide. L'utilisation de CUDA permettrait de lever cette restriction ; la description de chaque image et, par conséquent, la qualité de la recherche d'images seraient plus précises.

9.1.3.3 Influence du nombre de points

Nous connaissons l'influence de la dimension sur le temps de calcul. Attachons nous maintenant à l'influence du nombre de points n (références et requêtes). La figure 9.2 montre le temps de calcul en fonction de n et pour chacune des méthodes précédemment listées. Une fois encore, BF-CUDA

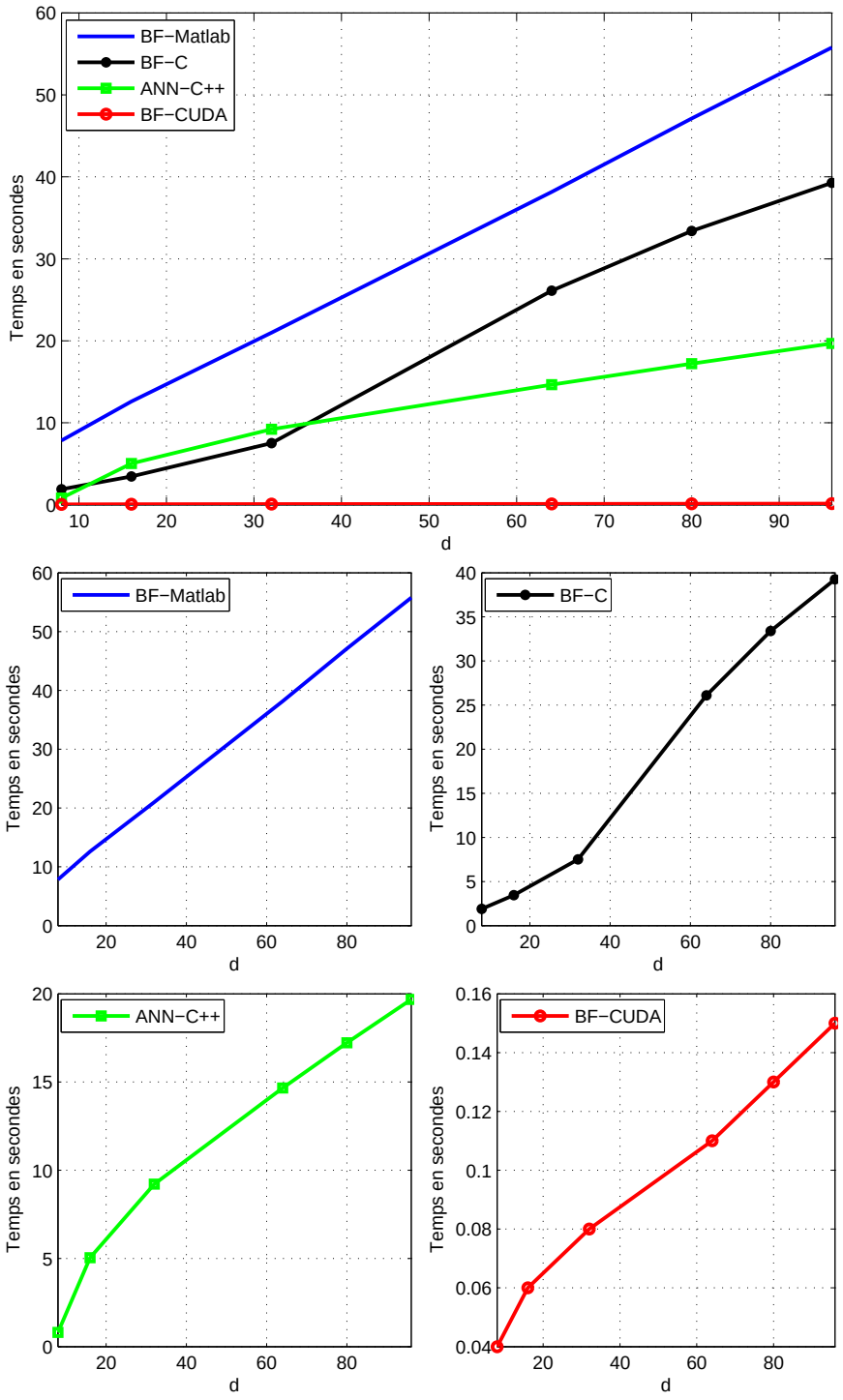


FIGURE 9.1 – *Évloution du temps de calcul en fonction de la dimension pour des ensembles contenant* 4800 *points et pour les méthodes BF-Matlab, BF-C, BF-CUDA et ANN-C++. Le paramètre k est fixé à 20. Quelle que soit la méthode utilisée, l'augmentation est linéaire. Cependant, en comparaison des méthodes BF-Matlab et BF-C, cette augmentation est moins importante avec ANN-C++, et est quasi-nulle avec BF-CUDA.*

est la méthode la plus rapide quel que soit le nombre de points. Nous remarquons que le temps de calcul augmente de manière polynomiale avec le nombre de points quelle que soit la méthode utilisée. Cette augmentation est un résultat attendu dans la mesure où nous avons n^2 distances à calculer, les deux ensembles de points étant de même taille dans nos expérimentations. En revanche, l'influence de n diffère selon la méthode. L'adéquation entre l'algorithme et l'architecture matérielle est telle qu'en comparaison des autres méthodes, l'augmentation du temps de calcul est quasiment négligeable pour BF-CUDA. Le nombre de points est souvent limité pour permettre une recherche *rapide* des kPPV. Une fois encore, l'utilisation de CUDA permet de lever cette restriction.

9.1.3.4 Influence du paramètre k

La figure 9.3 montre l'influence du paramètre k sur le temps de calcul pour des ensembles de 4800 points en dimension 32. Selon cette figure, la méthode la plus rapide est BF-CUDA. Rappelons que nous utilisons le tri par insertion modifié défini dans la section 8.3.2. Pour la méthode BF, le paramètre k n'est utilisé, et par conséquent n'a d'influence, que dans l'étape de tri. Pour la méthode ANN-C++, ce paramètre modifie le parcours de l'arbre.
Quelle que soit la méthode utilisée, le temps de calcul augmente linéairement avec k. Cependant, cette augmentation, représentée par la pente de la courbe, est très faible. Cette pente est de 0.079 pour BF-Matlab, 0.007 pour BF-C, 0.017 pour ANN-C++, et 0.002 pour BF-CUDA. En ce qui concerne l'augmentation obtenue avec ANN-C++, il semble sur la figure que cette augmentation soit logarithmique. Cependant, ce comportant n'est observé que pour les premières valeurs de k ; dès lors que k est supérieur à 20, l'augmentation est linéaire.

9.1.4 Répartition du temps de calcul

L'étude de l'influence des différents paramètres sur le temps de calcul n'est pas suffisante pour comprendre en détails le fonctionnement réel d'un GPU. Nous savons à ce stade qu'aucune des méthodes testées (BF-Matlab, BF-C, ANN-C++) ne permet une recherche de kPPV aussi rapide qu'avec BF-CUDA. Dans la suite, nous allons principalement évaluer la part en termes de temps de calcul que prend chaque étape de l'algorithme BF, à savoir le calcul des distances et le tri. Pour ce faire, nous utilisons le *profiler* CUDA disponible sur le site de NVIDIA.

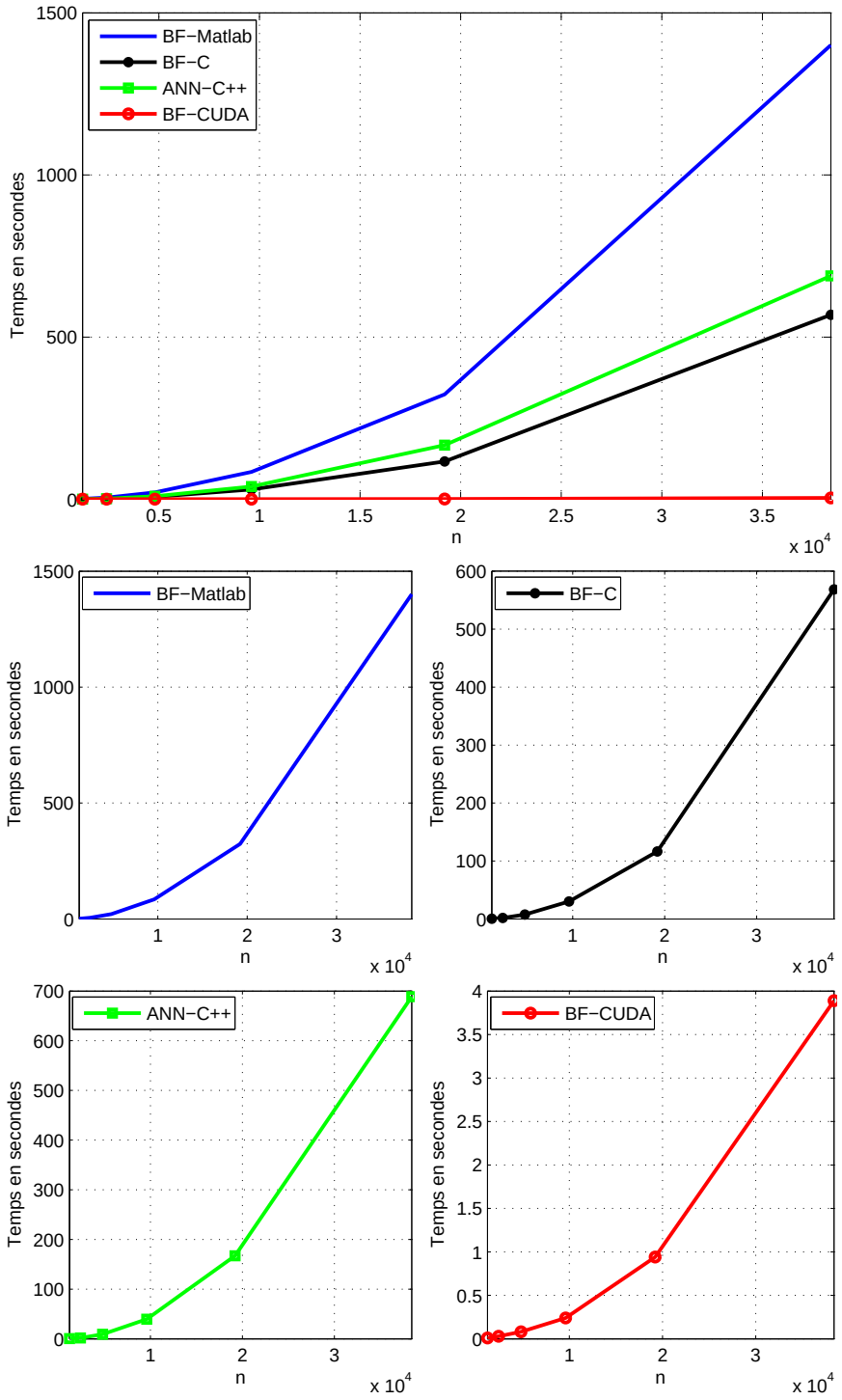


FIGURE 9.2 – *Évolution du temps de calcul en fonction du nombre de points et pour les méthodes BF-Matlab, BF-C, BF-CUDA et ANN-C++. La dimension est ici fixée à 32 et le paramètre k à 20. L'augmentation est polynomiale quelle que soit la méthode testée.*

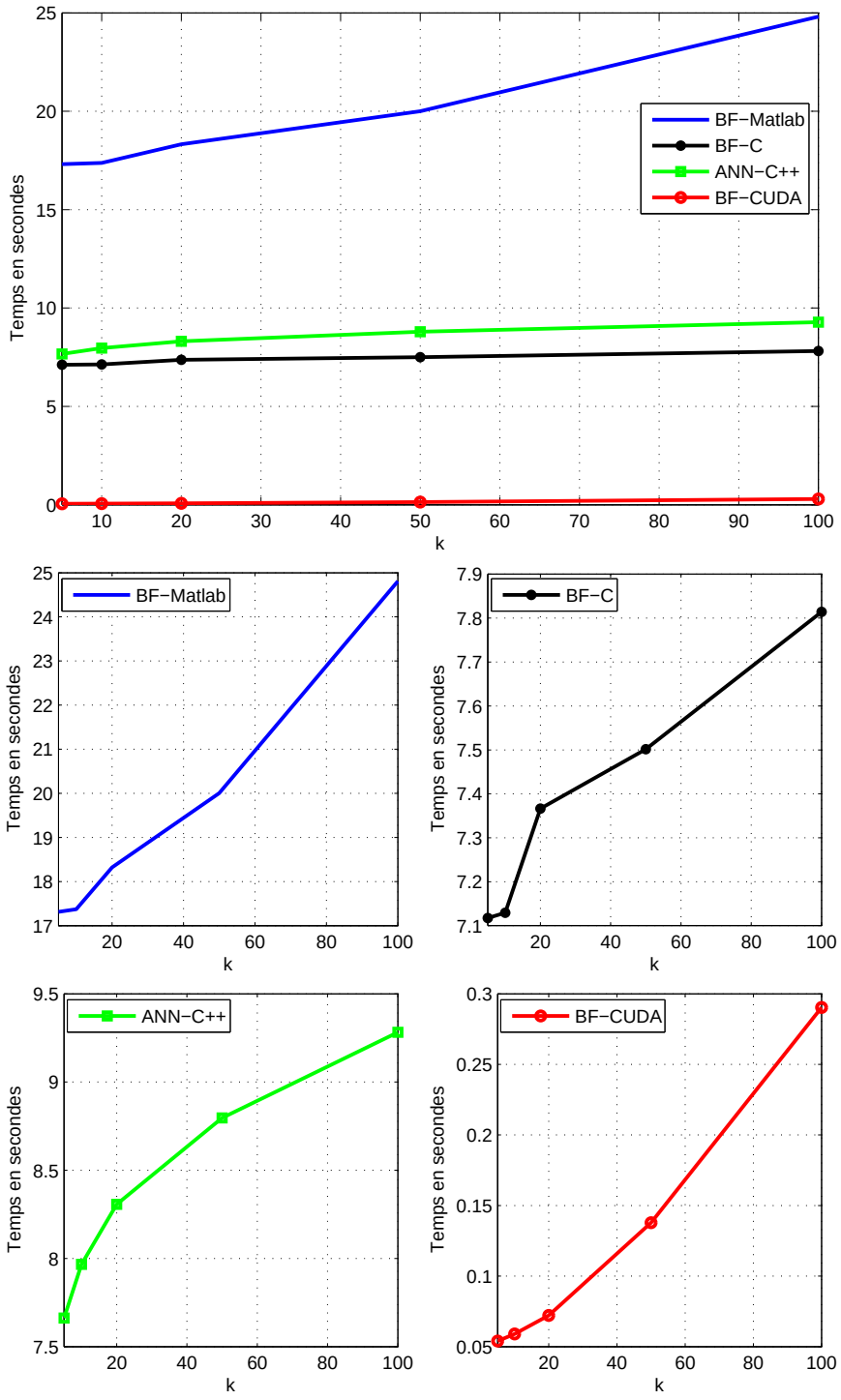


FIGURE 9.3 – *Évolution du temps de calcul en fonction du paramètre k et pour les méthodes BF-Matlab, BF-C, BF-CUDA et ANN-C++. Les ensembles de points considérés contiennent 4800 points de dimension 32. L'augmentation est linéaire quelle que soit la méthode testée. Cependant, cette augmentation est faible. La méthode la plus rapide et la moins sensible au paramètre k est BF-CUDA.*

9.1.4.1 Influence de l'implémentation

Le tableau 9.2 montre la part que prend chaque étape de l'algorithme BF selon qu'il soit implémenté en CPU (BF-C) ou en GPU (BF-CUDA). Les paramètres utilisés pour cette étude sont $n = 4800$, $d = 32$ et $k = 20$. Rappelons que n^2 distances sont calculées alors que seuls n tris sont effectués. Il apparaît que le calcul des distances sur CPU prend plus de 90% du temps de calcul contre moins 5% pour l'étape de tri. Le reste du temps est passé dans les allocations/desallocations, transferts, et autres diverses fonctions. Le calcul des distances est donc largement majoritaire. Etudions la répartition de ces mêmes étapes avec une implémentation GPU (BF-CUDA). Les deux tiers du temps de calcul sont passés à calculer les distances contre un tiers pour le tri. La part du tri est bien supérieure à celle observée pour la version CPU.
Le processus de tri est un processus plus complexe que le calcul d'une distance nécessitant, en général, plus d'opérations de lectures/écritures. La librairie CUDA donne des performances optimales si les accès mémoire sont minimisés et si ces accès sont coalescents. Dans le cas du calcul des distances, les accès mémoire sont optimisés. En revanche, les accès mémoire d'un tri sont, par nature, non coalescents car ils dépendent de la façon dont sont rangées les données. Le calcul des distances est par conséquent plus adapté que le tri à une implémentation en CUDA.

	Distances	Tri	Autre	Total
CPU	91%	4%	5%	7.51s
GPU	66%	32%	3%	0.076s

TABLE 9.2 – *Comparaison de la répartition du temps de calcul pour les implémentations CPU (BF-C) et GPU (BF-CUDA). Les étapes listées sont le calcul des distances et le tri. Dans ce tableau, nous utilisons* 4800 *points en dimension* 20 *et nous fixons* $k = 20$*. Dans cet exemple, BF-CUDA est 100 fois plus rapide que BF-C. De plus, la répartition des temps de calcul est plus équilibrée dans le cas de l'implémentation CUDA. Ceci est dû au fait que le calcul des distances est mieux adapté que le tri à une implémentation CUDA ; les lectures et écritures sont plus nombreuses et non coalescentes pour l'étape de tri.*

9.1.4.2 Influence de la dimension

Nous savons que la dimension a un impact sur le temps de calcul, et que cet impact est quasiment négligeable pour BF-CUDA en comparaison des autres méthodes testées. Il est néanmoins intéressant d'étudier l'influence de cette dimension sur la répartition du temps de calcul. Cette répartition est notée dans le tableau 9.3 pour 4800 points. Notez que la dernière case du tableau notée « autre » correspond aux fonctions annexes mais indispen-

sables telles que les allocations de mémoire, et, principalement, la copie des données sur la mémoire de la carte graphique.
Le temps d'exécution du tri ne dépend du nombre de points et du paramètre k. La dimension influence uniquement le temps nécessaire pour calculer l'ensemble des distances. En effet, le nombre d'opérations effectuées pour calculer une distance est proportionnel à la dimension. Le tri étant constant, il apparaît naturel que la part du calcul des distances augmente avec la dimension.
Comme nous l'avons déjà évoqué, le tri est moins adapté à une implémentation CUDA que le calcul des distances. Pour 4800 références et requêtes, le temps réservé au calcul des distances est majoritaire pour une dimension supérieure à 16.

d	8	16	32
Distances	37%	51%	66%
Tri	62%	47%	32%
Autre	1%	2%	2%
Temps total	0.040s	0.055s	0.076s

TABLE 9.3 – *Répartition du temps de calcul en fonction de la dimension pour chaque étape de la méthode BF. Dans ce tableau, nous utilisons 4800 points et nous fixons $k = 20$. La part du calcul des distances augmente avec la dimension.*

9.1.4.3 Influence du nombre de points

Le nombre de points influence le temps alloué au calcul des distances ainsi que celui alloué au tri. Le tableau 9.4 montre que la part du calcul des distances augmente avec le nombre de points n. Ceci s'explique par le fait que le nombre de threads augmente linéairement avec n pour l'étape de tri, et quadratiquement pour l'étape de calcul des distances. Dans ce tableau, la dimension est fixée à 16 et le paramètre k à 20.
Rappelons que le nombre de références et de requêtes est identique. Cependant, il est important de noter que le nombre de points dans ces deux ensembles influence différemment les deux étapes de l'algorithme BF. En effet, les requêtes jouent un rôle sur le nombre de tris à effectuer, quant aux références, elles influencent la durée de chaque tri. En ce qui concerne l'étape de calcul des distances, l'influence du nombre de points est identique quel que soit l'ensemble considéré.

9.1.4.4 Influence du paramètre k

Le calcul des distances est totalement indépendant du paramètre k dans la mesure où toutes les distances sont effectivement calculées. Dans le cas de l'utilisation d'un tri classique, tel que le tri de Dobosiewicz, qui va trier

n	2400	4800	9600
Distances	28%	51%	59%
Tri	70%	47%	40%
Autre	2%	2%	1%
Temps total	0.023s	0.055s	0.169s

TABLE 9.4 – *Répartition du temps de calcul en fonction du nombre de points pour chaque étape de la méthode BF. Dans ce tableau, les points ont pour dimension $d = 16$ et nous avons fixé $k = 20$. La part du calcul des distances augmente avec le nombre de points.*

l'ensemble des distances pour un point requête donné, le paramètre k n'a aucune influence sur le temps de calcul. Dans un tel cas, ce paramètre est utilisé pour indiquer le nombre de distances qu'il faut copier en mémoire et retourner en sortie d'exécution. Le paramètre k est uniquement utilisé lors de l'étape de tri si nous utilisons un tri par insertion modifié qui ne va regrouper que les k plus petites distances calculées. Nous avons déjà montré que ce tri est plus rapide qu'un tri classique pour de petites valeurs de k. Le tableau 9.5 montre que la part du tri augmente avec la valeur de k. Plus ce paramètre augmente, plus le nombre de distances à insérer est grand et le tri long.

k	5	10	20
Distances	82%	71%	51%
Tri	15%	26%	47%
Autre	3%	3%	2%
Temps total	0.033s	0.037s	0.055s

TABLE 9.5 – *Répartition du temps de calcul en fonction du paramètre k pour chaque étape de la méthode BF. Dans ce tableau, 4800 points en dimension 16 sont utilisés. La part du tri augmente avec le paramètre k.*

§ 9.2 APPLICATION AU SUIVI D'OBJETS

9.2.1 Suivi d'objets

Nous connaissons l'intérêt de l'utilisation de la programmation GPU pour la recherche des kPPV sur des données synthétiques. Nous évaluons dans cette section son intérêt dans un contexte applicatif réel, à savoir le suivi d'objets fondé sur l'utilisation de la divergence de Kullback-Leibler. En effet, rappelons que, initialement, nous nous sommes intéressés à la recherche des kPPV dans un contexte d'estimation de mesures statistiques

pour le suivi d'objets (voir chapitre 6). Nous revenons donc dans cette section à l'application initiale.
La méthode de suivi présentée dans le chapitre 6 correspond aux travaux de Boltz *et al.* [BDB07, Bol08]. Rappelons qu'un objet est décrit par un ensemble de composantes, typiquement 3 composantes couleur (YUV) et 2 composantes géométriques (coordonnées cartésiennes). La mesure de similarité utilisée pour comparer un objet de référence avec un objet candidat est la divergence de Kullback-Leibler estimée grâce à la distance au k-ème plus proche voisin. Boltz *et al.* ont montré que parmi les méthodes classiques, cette approche donne les meilleurs résultats (voir figure 9.4).
Dans cette partie, nous proposons d'ajouter d'autres composantes et nous étudions leur impact sur la qualité du suivi. L'unique obstacle à utiliser plus de composantes vient du fait que le calcul des kPPV en haute dimension est très long. La programmation GPU nous a montré qu'il est possible de calculer les kPPV en très haute dimension de manière très rapide. Nous évaluons donc également l'impact de l'ajout de ces composantes sur le temps de calcul.
Pour cette partie expérimentations, nous disposons de la vérité terrain (taille et position de l'objet) obtenue manuellement pour chacune des vidéos testées. Pour un suivi donné, cette vérité terrain nous permet de calculer l'erreur de position, l'erreur d'échelle, et la différence symétrique (voir section 4.2.3), et ce pour chaque image de la vidéo. Rappelons que la différence symétrique est exprimée en pourcentage de pixels mal classés.

9.2.2 Composantes testées

La description des objets utilise plusieurs composantes. Le choix des composantes employées influe sur la qualité du suivi. Le tableau 9.6 présente la nomenclature que nous utilisons dans la suite pour la description des objets. Nous précisons ci-dessous la nature de chaque composante :

- Y : luminance
- $\text{Patch}_{3\times3}$: voisinage 3×3 dans la composante Y
- $\text{Patch}_{9\times9}$: voisinage 9×9 dans la composante Y
- U : chrominance
- V : chrominance
- ∇_x : gradient selon l'axe des abscisses
- ∇_y : gradient selon l'axe des ordonnées
- x : coordonnée selon l'axe des abscisses
- y : coordonnée selon l'axe des ordonnées

Par exemple, un objet dont la représentation est notée CGP est représenté par les composante Y, U, V, ∇_x, ∇_y, x, et y. Par définition, $\text{Patch}_{3\times3}$ et $\text{Patch}_{9\times9}$ contiennent l'information Y ce qui explique que celle-ci soit absente des représentations C_3 et C_9.

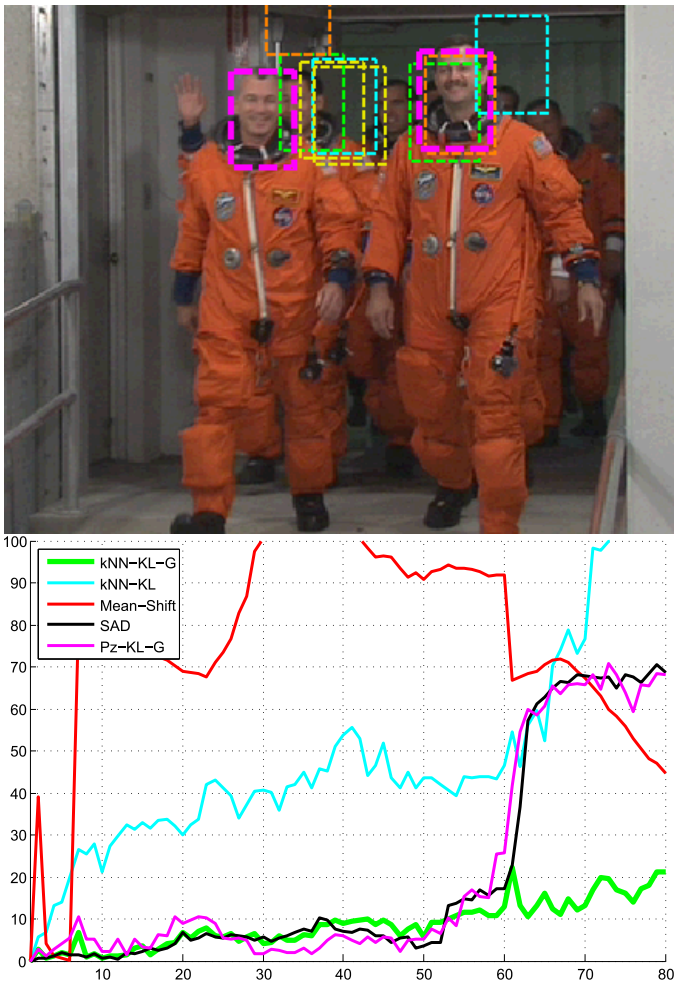


FIGURE 9.4 – *Images issues de la thèse de Sylvain Boltz. La figure du bas illustre l'erreur de position de la boite pour l'ensemble des méthodes testées. La figure du haut montre le suivi obtenu avec ces différentes méthodes (couleurs identiques). Parmi les méthodes testées, le descripteur KNN-KL-G composé de composantes couleur et de composantes géométriques associé à la divergence de Kullback-Leibler estimée par une méthode des kPPV donne le meilleur suivi. KNN-KL ne contient pas d'information géométrique, Pz-KL-G utilise la méthode de Parzen pour estimer la divergence de Kullback-Leibler, Mean-shift est la méthode de Comaniciu et al. fondée sur le mean-shift et SAD est un block-matching ayant pour critère SAD.*

Nom	Y	$\text{Patch}_{3\times3}$	$\text{Patch}_{9\times9}$	U	V	∇_x	∇_y	x	y
C	•			•	•				
C_3		•		•	•				
C_9			•	•	•				
G						•	•		
P								•	•

TABLE 9.6 – *Composantes utilisées pour la description d'un objet. C = couleur, G = gradient, et P = position.*

9.2.3 Expérimentations sur la séquence Crew

9.2.3.1 Vidéo et réglages

La vidéo *Crew* est composée de 100 images de taille 352×288. L'objet à suivre est le visage d'un des astronautes et est représenté par une boîte englobante. Cette boîte contient approximativement 900 pixels sur la première image et 2000 pixels sur la dernière image de la vidéo. La figure 9.5 présente l'évolution des paramètres de la vérité terrain pour cette séquence. Nous pouvons voir que l'échelle augmente linéairement au cours du temps. Cette augmentation est liée au fait que l'astronaute se rapproche de la caméra. La position (verticale et horizontale) suit un mouvement sinusoïdale induit par la marche. La mouvement de l'objet à suivre est complexe d'où l'intérêt de la séquence. De plus, il faut noter que la luminosité de l'objet change au cours du temps ce qui complexifie encore la séquence. Cette variation est due aux différentes sources lumineuses et aux flashs des appareils photographiques.

Pour cette séquence, nous utilisons les descripteurs suivants : C, CP, CG, CGP, C_3, C_3P. Pour cette expérimentation, nous éliminons les descripteurs C_9 et C_9P car la taille de patch utilisée est trop grande par rapport à la taille de l'objet dans la vidéo. Les composantes géométriques sont pondérées par un coefficient 0.6 pour atténuer l'influence de la géométrie. Ce paramètre, réglé manuellement, est celui qui a engendré le suivi le plus précis (obtenu avec le descripteur CGP). Pour des raisons de facilité, nous utilisons cette valeur pour l'ensemble des descripteurs testés. Ce paramètre devrait cependant être réglé indépendamment pour chacun des descripteurs. Pour des raisons similaires, le paramètre k est fixé 3.

9.2.3.2 Résultats qualitatifs

La figure 9.6 donne l'erreur sur les paramètres commise par la méthode de suivi en fonction du descripteur utilisé. La figure 9.7 donne l'erreur de suivi exprimée par différence symétrique. Enfin, la figure 9.8 présente

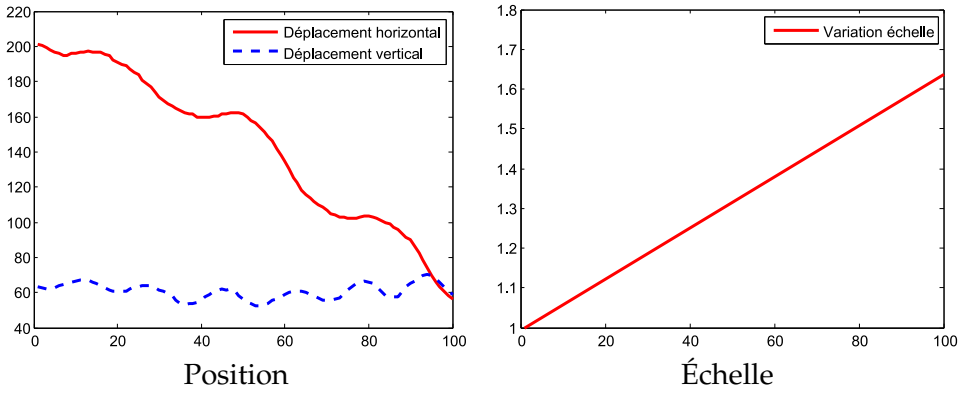


FIGURE 9.5 – *Évolution des paramètres de la vérité terrain en fonction de l'indice de l'image pour la séquence Crew*

le suivi sous forme de tube temporel. Chaque tube représente l'évolution spatio-temporelle de la boîte englobante. Cette illustration 3D confronte la vérité terrain au suivi calculé. Le tube vert correspond à la vérité terrain et le tube rouge correspond au suivi induit par l'utilisation d'un descripteur donné.

Il apparaît que l'ajout de l'information géométrique dans la description de l'objet améliore la qualité du suivi par rapport à une description équivalente sans information géométrique. Cette remarque confirme les travaux de Elgammal *et al.* [EDD03] et de Boltz *et al.* [BDB07, Bol08].

Considérons maintenant les descripteurs contenant l'information géométrique. L'ajout d'informations relatives au gradient de l'image (descripteur CGP) améliore la précision du suivi par rapport au critère sans gradient (CP). Le descripteur CGP donne en effet le suivi le plus précis parmi l'ensemble des descripteurs testés.

Un patch 3×3 contient grossièrement les mêmes informations que le gradient dans la mesure où ce gradient est calculé sur le voisinage 3×3. On s'attendrait par conséquent à obtenir des résultats similaires pour les descripteurs CGP et C_3P. Cependant, nous remarquons que le suivi est meilleur avec le descripteur CGP qu'avec le descripteur C_3P. Nous proposons deux explications à ce résultat. Premièrement, le gradient a une dynamique plus faible qu'une composante image. L'espace engendré par ces deux descripteurs est par conséquent différent ce qui implique que les distances entre points sont elles aussi différentes. La valeur de la divergence est par conséquent affectée par la répartition des points dans l'espace considéré. Deuxièmement, la dimension des points est de 7 pour le

descripteur CGP, et de 13 pour C_3P. L'influence des composantes géométriques est par conséquent plus faible pour C_3P que pour CGP. Pour palier à ce dernier point, la pondération des composantes géométriques pourrait être augmentée.

La figure 9.10 montre l'évolution des paramètres du suivi (taille et position de la boîte) pour le descripteur CGP superposée aux paramètres de la vérité terrain. La figure 9.9 montre le résultat du suivi sur les images de la vidéo pour ce même descripteur. Le suivi calculé est d'une grande précision. Seule l'échelle est légèrement sous-évaluée.

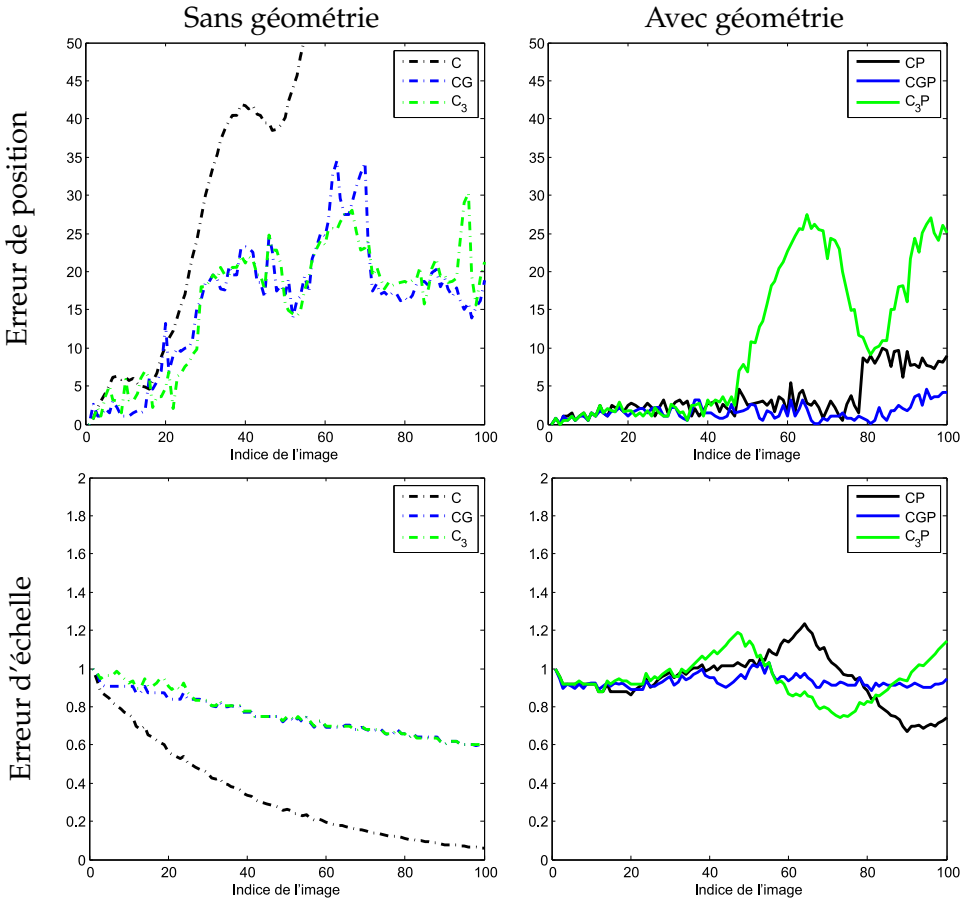


FIGURE 9.6 – *Erreur sur les paramètres (position et échelle) pour la séquence Crew en fonction de l'indice de l'image et du descripteur utilisé.*

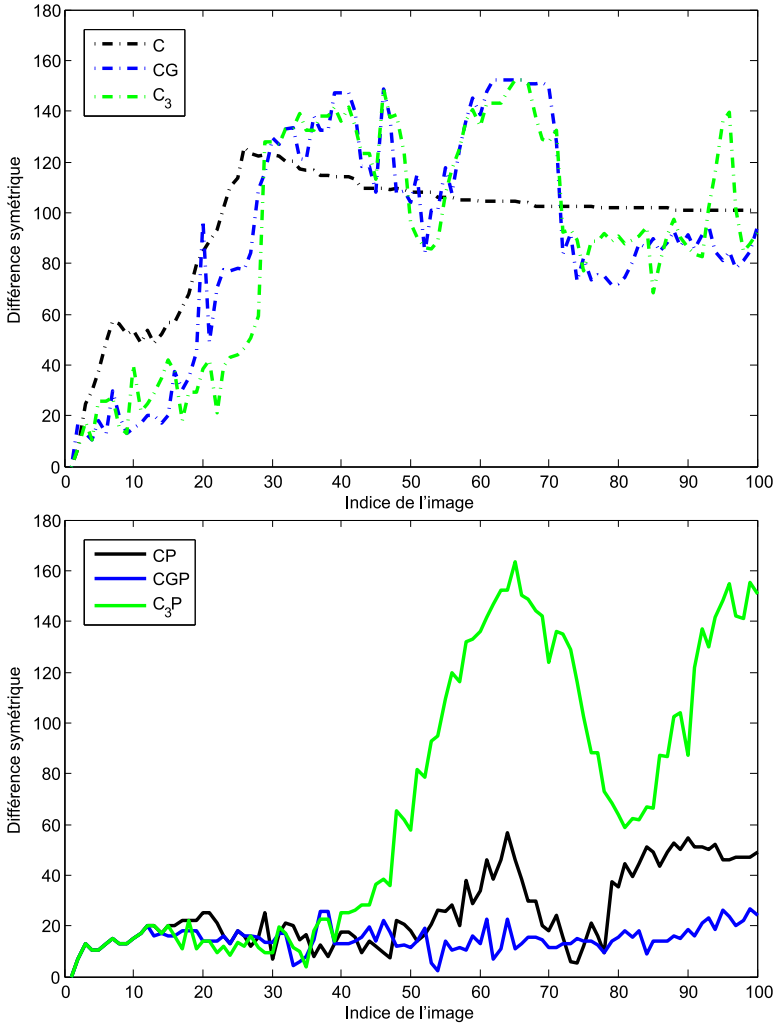


FIGURE 9.7 – *Erreur du suivi (différence symétrique) pour la séquence Crew en fonction de l'indice de l'image et du descripteur utilisé.*

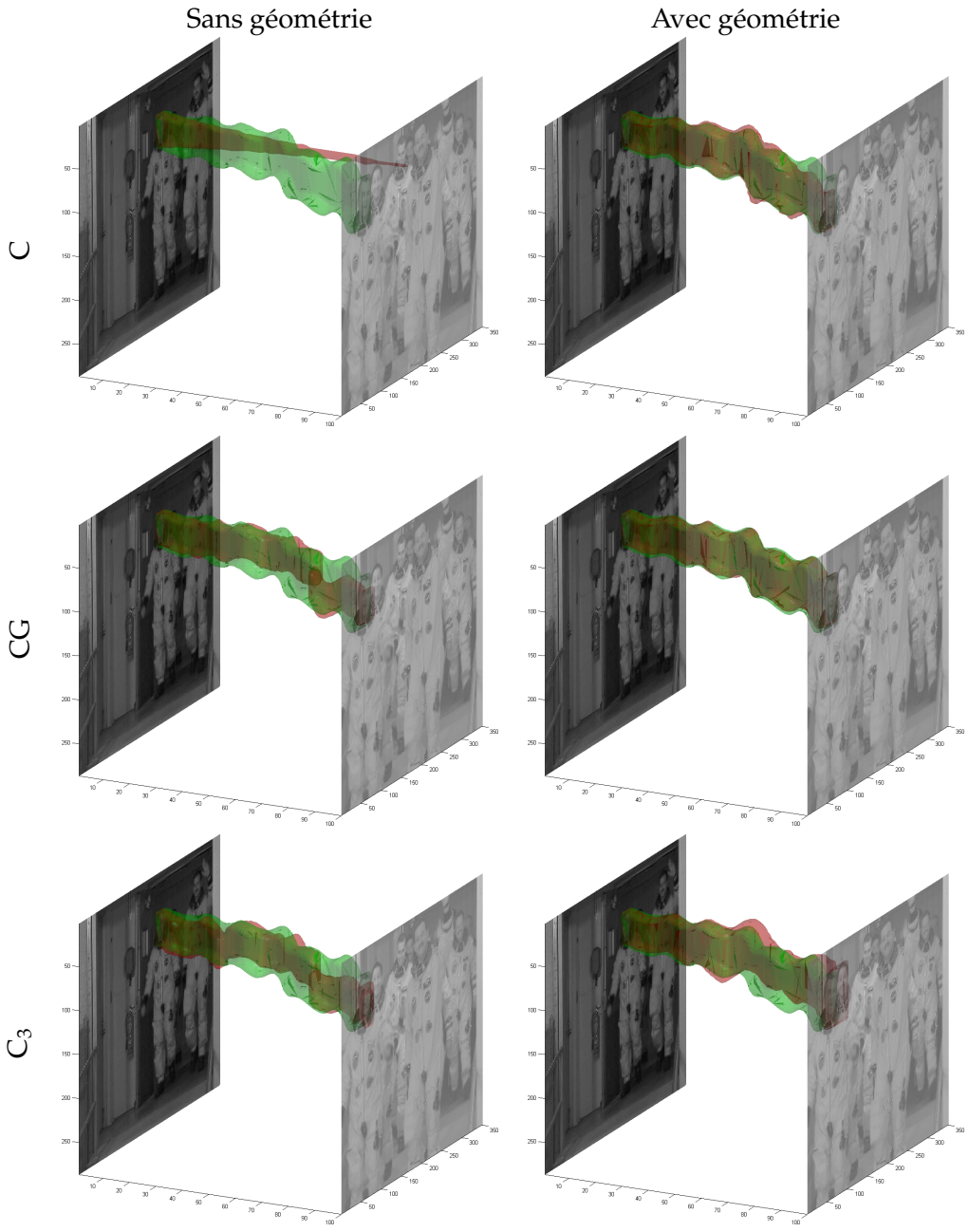


FIGURE 9.8 – *Évolution spatio-temporelle de la boîte englobante sur la séquence Crew. Le tube vert représente la vérité terrain et le tube rouge le suivi obtenu pour un descripteur considéré.*

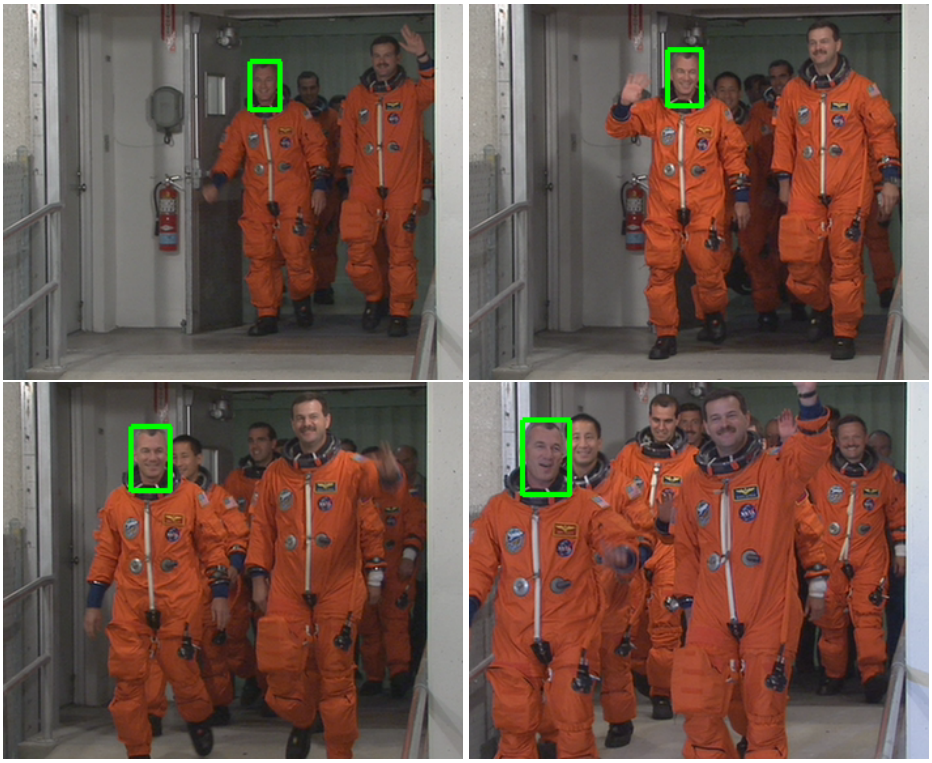


FIGURE 9.9 – *Résultat de suivi sur la séquence Crew. Les images présentées sont I_1, I_{33}, I_{50}, et I_{100} (resp. de gauche à droite et de haut en bas). Le descripteur employé pour ce résultat est CGP.*

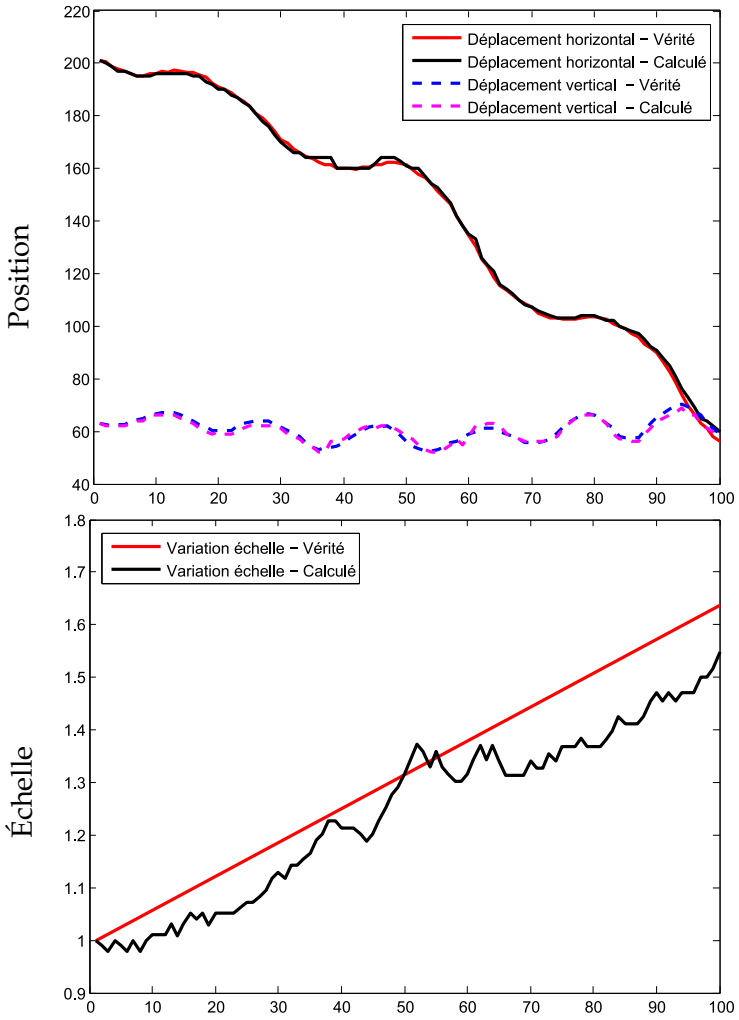


FIGURE 9.10 – *Évolution des paramètres de la vérité terrain et du suivi calculé (CGP) en fonction de l'indice de l'image pour la séquence Crew*

9.2.3.3 GPU

Le tableau 9.7 présente le temps nécessaire pour réaliser le suivi d'objet en fonction du descripteur et de la méthode de recherche des kPPV. Seules les méthodes ANN-C++ et BF-CUDA y sont comparées, BF-Matlab et BF-C étant trop lentes. Le choix du descripteur influe sur la dimension de chaque pixel de l'objet et par conséquent sur la durée du processus de suivi. Nous remarquons sur le tableau 9.7 que globalement le temps augmente avec la dimension des descripteurs. Toutefois, nous remarquons que le suivi utilisant le descripteur CGP (dimension 7) est plus lent que le suivi utilisant le descripteur C_3 (dimension 11). Le nombre de points (pixels) contenus dans les boîtes ainsi que le nombre de boîtes candidates considérées lors de la recherche de la boîte optimale influent sur la durée du suivi. La boîte calculée avec le descripteur C_3 devient, avec le temps, plus petite (moins de pixels) que celle calculée avec CGP ce qui explique en partie la différence de temps.
L'information importante à retenir dans tableau 9.7 est la confirmation du fait que, par rapport à des méthodes optimales implémentées en C++, l'utilisation de la programmation parallèle sur GPU accélère l'estimation de mesures statistiques pour des données réelles. Cette accélération a plusieurs avantages. Cela permet d'effectuer un suivi plus rapide, d'essayer des descripteurs de plus grande dimension, et de réaliser un plus grand nombre de tests. Ce dernier point est particulièrement intéressant pour des méthodes dans lesquelles le réglage de paramètres est essentiel. L'accélération observée est de 3 pour le descripteur CGP.

Descripteur	Dimension	ANN-C++	BF-CUDA	Gain
C	3	1m 33s	53s	1.8
CP	5	2m 05s	1m 05s	1.9
CG	5	2m 35s	1m 07s	2.3
CGP	7	4m 27s	1m 19s	3.3
C_3	11	6m 40s	1m 17s	5.2
C_3P	13	5m 43s	1m 12s	4.8

TABLE 9.7 – *Durée du processus de suivi sur la séquence Crew en fonction du descripteur et de l'implémentation (ANN-C++ ou BF-CUDA). Le temps est donné en minutes, secondes.*

9.2.4 Expérimentations sur la séquence Poltergay

9.2.4.1 Vidéo et réglages

La vidéo *Poltergay* [Ext06] est composée de 100 images de taille 720×576. L'objet à suivre est le visage de l'acteur Clovis Cornillac et est représenté

par une ellipse. Cette ellipse contient approximativement 9000 pixels sur la première image et 20000 pixels sur la dernière image de la vidéo. Cette séquence est particulièrement complexe à traiter du fait des couleurs sombres des images (voir figure 9.15). La figure 9.11 présente l'évolution des paramètres de la vérité terrain pour cette séquence. Ni l'échelle ni la position n'évolue linéairement sur l'ensemble de la séquence. Le mouvement de l'objet est complexe.

Pour cette séquence, nous utilisons les descripteurs suivants : C, CP, CG, CGP, C_3, C_3P, C_9, C_9P. Les composantes géométriques sont pondérées par un coefficient 0.8. Le paramètre k est fixé 10.

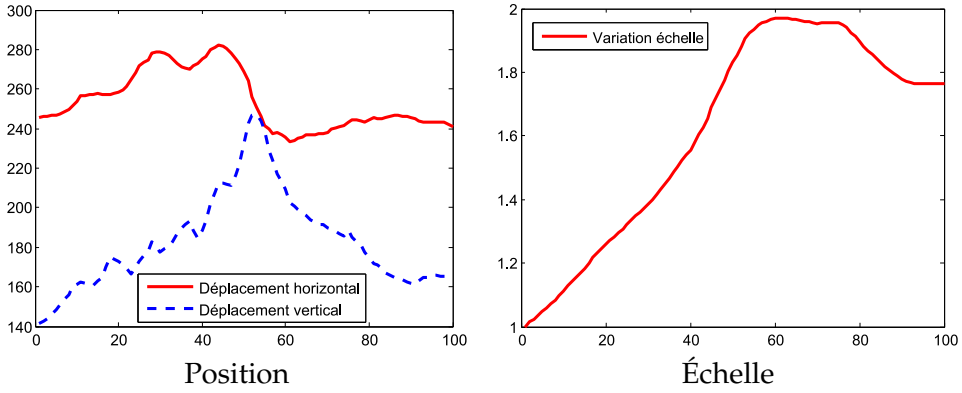


FIGURE 9.11 – *Évolution des paramètres de la vérité terrain en fonction de l'indice de l'image pour la séquence Poltergay*

9.2.4.2 Séquence Poltergay

La figure 9.12 donne l'erreur sur les paramètres commise par la méthode de suivi en fonction du descripteur utilisé. La figure 9.13 donne l'erreur de suivi exprimée par différence symétrique. Enfin, la figure 9.14 présente le suivi sous forme de tube temporel.

Comme précédemment, l'ajout de l'information géométrique dans la description de l'objet améliore la qualité du suivi par rapport à une description équivalente sans information géométrique. L'ajout du gradient (descripteur CGP) améliore la précision du suivi par rapport au critère sans gradient (CP). En effet, même si la différence est moins évidente que pour la séquence Crew, le meilleur suivi (globalement sur toute la séquence) est obtenu avec le descripteur CGP.

Nous remarquons également que, sans géométrie, la boîte diminue au cours du temps sans réellement suivre la trajectoire de l'objet. C_9 fait excep-

tion à la règle : malgré une diminution sensible de la taille de la boîte par rapport à la vérité terrain, il apparaît que le tube suit globalement la trajectoire de l'objet. En étudiant plus attentivement les résultats, nous remarquons que C_9 est le meilleur descripteur par rapport à l'ensemble des descripteurs sans géométrie. De même, le descripteur C_9P donne les meilleurs résultats jusqu'à l'image 80. À partir de là, le suivi obtenu avec C_9P est le plus mauvais (en comparaison des descripteurs avec géométrie). En effet, à partir de l'image 80, le personnage tourne la tête. L'utilisation d'un patch apporte une information relative au voisinage local. Cette contrainte de voisinage est respectée jusqu'à l'image 80 ce qui améliore le suivi. Ensuite, cette contrainte n'est plus respectée ce qui pénalise le suivi.

La figure 9.16 montre l'évolution des paramètres du suivi (taille et position de la boîte) pour le descripteur CGP superposée aux paramètres de la vérité terrain. La figure 9.15 montre le résultat du suivi sur les images de la vidéo pour ce même descripteur. Le suivi calculé est précis.

9.2.4.3 GPU

Le tableau 9.8 présente le temps nécessaire pour réaliser le suivi d'objet sur la séquence *Poltergay* en fonction du descripteur et de la méthode de recherche des kPPV. Comme précédemment, seules les méthodes ANN-C++ et BF-CUDA y sont comparées. Le temps de calcul augmente globalement avec la dimension des descripteurs. L'utilisation de CUDA permet d'accélérer le processus de suivi, et ce quelle que soit la description utilisée.

Représentation	Dimension	ANN-C++	BF-CUDA	Gain
C	3	7m 30s	5 m	1.5
CP	5	9m 45s	6m 10s	1.6
CG	5	11m 10s	5m 52s	1.9
CGP	7	1h 3m	42m	1.5
C_3	11	32m	8m	3.6
C_3P	13	1h 25m	1h 18m	1.1
C_9	83	6h 59m	42m	9.8
C_9P	85	27h 16m	5h 44m	5.3

TABLE 9.8 – *Durée du processus de suivi sur la séquence Poltergay en fonction du descripteur et de l'implémentation (ANN-C++ ou BF-CUDA). Le temps est donné en heures, minutes, secondes.*

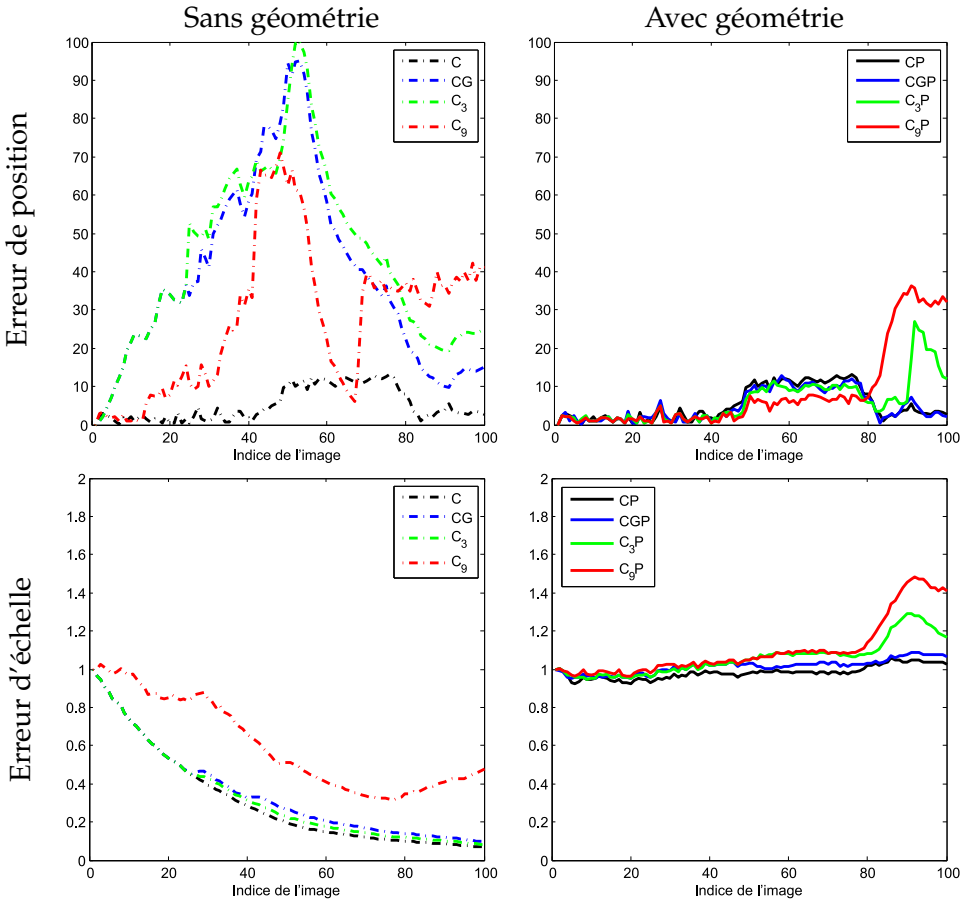


FIGURE 9.12 – *Erreur sur les paramètres (position et échelle) pour la séquence Poltergay en fonction de l'indice de l'image et du descripteur utilisé.*

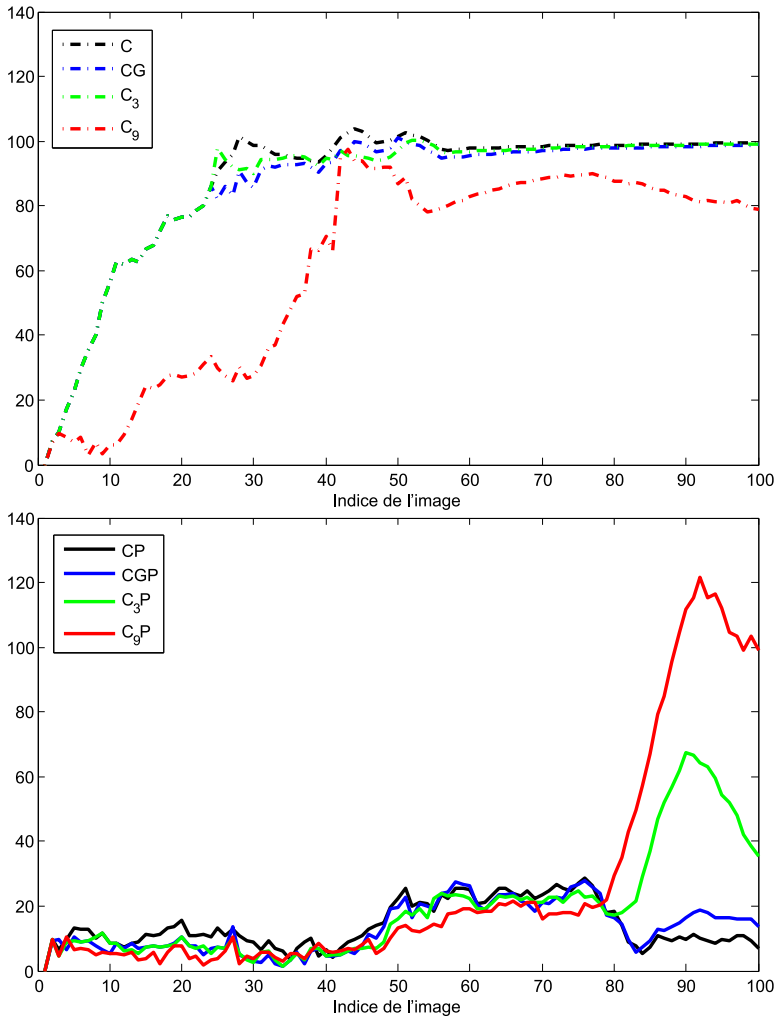


FIGURE 9.13 – *Erreur du suivi (différence symétrique) pour la séquence Poltergay en fonction de l'indice de l'image et du descripteur utilisé.*

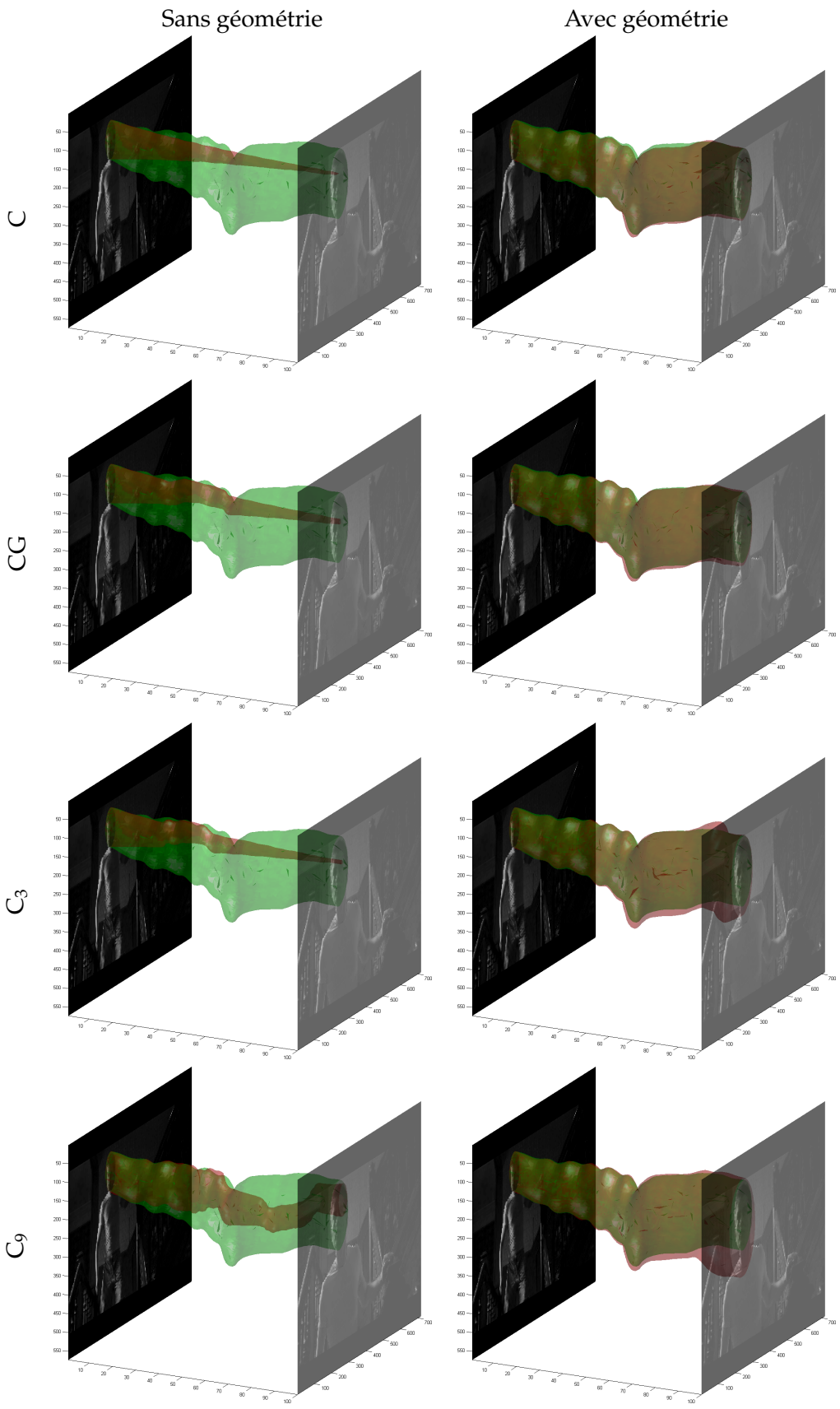


FIGURE 9.14 – *Évolution spatio-temporelle de la boîte englobante sur la séquence Crew. Le tube vert représente la vérité terrain et le tube rouge le suivi obtenu pour un descripteur considéré.*

FIGURE 9.15 – *Résultat de suivi sur la séquence Poltergay. Les images présentées sont I_1, I_{33}, I_{50}, et I_{100} (resp. de gauche à droite et de haut en bas). Le descripteur employé pour ce résultat est CGP.*

§ 9.3 CONCLUSION

La recherche des k plus proches voisins (kPPV) est un problème présent dans de nombreuses applications de traitement d'images, mais également dans des applications très différentes telles que la finance, la biologie, la physique, la génétique, etc. La recherche des kPPV est un problème long à résoudre, surtout quand le nombre de points et la dimension de ces mêmes points augmentent. Certaines méthodes récentes fondées sur des kd-trees ou des fonctions de hachage permettent d'accélérer la recherche. Toutefois, le processus reste lent. Cette lenteur est un inconvénient majeur généralement résolu en limitant la dimension et le nombre de requêtes et de références.

Nous avons proposé une implémentation GPU, via l'utilisation de l'API NVIDIA CUDA, de la recherche exhaustive des kPPV. Le but de ce chapitre était de montrer l'intérêt d'une implémentation GPU par rapport à des approches classiques et optimales en termes de programmation CPU. Nous avons montré qu'à algorithme équivalent (recherche exhaustive), l'utilisation de GPU permet d'accélérer la recherche jusqu'à un facteur 400 au sein de Matlab. De plus, nous avons montré que notre implémentation est jusqu'à 150 fois plus rapide qu'une implémentation en C++ d'un algorithme (ANN) fondé sur l'utilisation de kd-trees, algorithme reconnu

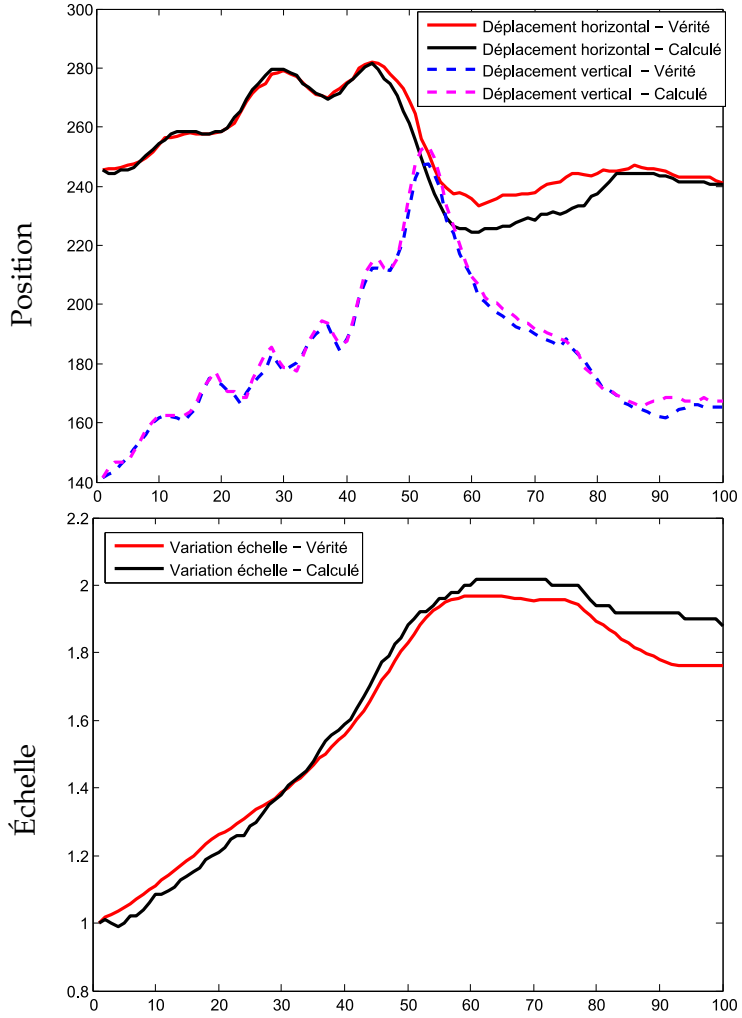


FIGURE 9.16 – *Évolution des paramètres de la vérité terrain et du suivi calculé (CGP) en fonction de l'indice de l'image pour la séquence Poltergay*

pour sa rapidité. Nous avons en particulier montré que l'utilisation de CUDA permet de diminuer les temps de calcul pour une application de suivi d'objets fondée sur l'estimation d'une mesure statistique, à savoir la divergence de Kullback-Leibler. Le gain obtenu s'échelonne entre 2 et 15 en fonction de la description et de la séquence considérée. Nous avons également utilisé notre approche pour accélérer une méthode d'indexation d'images [ADPB08, PADB08] d'un facteur 10 (non montré dans ce chapitre).

L'arrivée de la programmation sur GPU remet en cause beaucoup d'idées reçues sur le choix d'un algorithme pour la résolution d'un problème donné. En effet, l'hyper-parallélisation des processeurs graphiques permet de réouvrir des branches de la recherche fermées il y a des décennies pour cause d'impossibilité d'utilisation en un temps raisonnable. Un algorithme efficace en CPU n'est pas forcément efficace en GPU, et inversement. Nous avons vu que l'étape de calcul des distances est bien plus efficace en GPU qu'en CPU relativement au temps total de l'exécution. À l'inverse, le tri « pénalise » la recherche des kPPV en GPU. L'avenir des calculs scientifiques est indissociable de l'utilisation de processeurs type GPU. Toutefois, les CPUs ne sont pas condamnés à disparaître. En effet, certaines méthodes ne peuvent être parallélisées et sont donc plus efficaces sur CPU. Il est probable que, dans un proche avenir, les deux types de processeurs cohabitent dans la plupart des ordinateurs, ou alors que de nouveaux processeurs hybrides fassent leur apparition.

- CHAPITRE 10 -

CUBLAS

§ 10.1 INTRODUCTION

BLAS (acronyme de *Basic Linear Algebra Subprograms*) est, comme son nom l'indique, un ensemble de fonctions réalisant des opérations basiques d'algèbre linéaire. Les opérations sont par exemple l'addition ou la multiplication de vecteurs ou de matrices. Les fonctions BLAS sont très optimisées ce qui leur vaut d'être fréquemment utilisées. Par exemple, la librairie LAPACK (pour *Linear Algebra PACKage*) est une bibliothèque dédiée à la simulation numérique et utilisant massivement les fonctions BLAS. De même, Matlab est connu pour être un langage de programmation particulièrement adapté aux opérations matricielles. Ces opérations sont en effet assurées par des fonctions BLAS. Par conséquent, un programme écrit dans un fichier Mex et réalisant un simple produit de matrice est en général beaucoup plus lent que le produit matriciel de Matlab.
CUBLAS (pour CUDA-BLAS) est une implémentation de BLAS écrite en CUDA. Cette librairie permet d'accélérer les fonctions BLAS classiques en utilisant la puissance de calcul de la programmation GPU. Un avantage majeur de la librairie CUBLAS est le fait qu'elle puisse être utilisée sans aucune instruction CUDA. En effet, les instructions CUDA usuelles telles que la création de matrices sont réalisées par des fonctions implémentées dans CUBLAS de manière à simplifier l'utilisation de la librairie. Toutefois, il est également possible d'utiliser CUBLAS au sein d'un code CUDA classique. La librairie CUBLAS est bien documentée ce qui permet de l'utiliser facilement, même pour un néophyte de la programmation.
Le but de ce chapitre est de tester la librairie CUBLAS et d'évaluer son utilisation dans le contexte de la recherche des kPPV.

§ 10.2 RECHERCHE DES KPPV EN CUBLAS

Suite à un discussion avec Quaid Morris [Mor], il nous a été suggéré de modifier notre approche du calcul des distances entre deux points [Nie05, NN07]. Les points sont généralement représentés sous forme de vecteurs colonne. Dans la suite, on utilisera la notion de point ou de vecteur selon que l'on parle de l'objet ou de sa représentation. Soient X et Y deux points définis dans $\mathbb{R}^d$:

$$X = \begin{pmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_d \end{pmatrix} \qquad Y = \begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ y_d \end{pmatrix}$$

La distance Euclidienne entre ces points, notée ρ, est calculée comme suit :

$$\rho(X,Y) = \sqrt{\sum_{i=1}^{d}(x_i - y_i)^2} \tag{10.1}$$

Le carré de cette distance peut être simplement calculé par un produit scalaire :

$$\rho^2(X,Y) = {}^t(X-Y).(X-Y) \tag{10.2}$$

En développant le produit scalaire, nous obtenons l'expression suivante :

$$\rho^2(X,Y) = {}^t\mathrm{X}.X \ + \ {}^t\mathrm{Y}.Y \ - \ {}^t\mathrm{X}.Y \ - \ {}^t\mathrm{Y}.X \tag{10.3}$$

$$= \|X\| \ + \ \|Y\| - \ 2{}^t\mathrm{X}.Y \tag{10.4}$$

où $\|X\|$ et $\|Y\|$ représentent respectivement la norme des vecteurs X et Y.
Rappelons que nous calculons les distances entre un ensemble de points référence et un ensemble de points requête. L'opération racine carrée est une opération lourde à calculer. La fonction racine carrée étant une fonction croissante et bijective, l'ordre des distances ne change pas. Ainsi, pour un point requête donnée, le carré des distances est calculé, puis le tri est appliqué, et enfin la fonction racine carrée est appliquée aux seules k plus petites distances.
Le calcul des distances au carré tel qu'il est défini dans l'équation (10.4) présente de nombreux avantages :

- Tout d'abord, la norme de chaque vecteur (référence et requête) peut être pré-calculée et additionnée au moment du calcul d'une distance.
- Considérons que Y soit un point requête. La norme $\|Y\|$ est commune pour toute distance calculée entre Y et un point référence quelconque. Comme pour le calcul de la racine carrée, l'addition de cette norme n'est nécessaire qu'après l'étape de tri et uniquement pour les k plus petites distances, les autres étant par la suite ignorées.

– Enfin, le produit scalaire ${}^tX.Y$ représente la grande majorité des calculs qui doivent être réalisés. L'avantage est que ce produit est parfaitement adapté à l'utilisation d'une fonction BLAS (ou CUBLAS) hautement optimisée et spécialisée dans le produit matriciel.

La généralisation de cette méthode à des ensembles de points est triviale. Soient X et Y des matrices représentant respectivement m points de référence et n points requête de dimension d :

$$X = \begin{pmatrix} x_{1,1} & x_{1,2} & \cdots & x_{1,m} \\ x_{2,1} & x_{2,2} & \cdots & x_{2,m} \\ \vdots & \vdots & \ddots & \vdots \\ x_{d,1} & x_{d,2} & \cdots & x_{d,m} \end{pmatrix} \qquad Y = \begin{pmatrix} y_{1,1} & y_{1,2} & \cdots & y_{1,n} \\ y_{2,1} & y_{2,2} & \cdots & y_{2,n} \\ \vdots & \vdots & \ddots & \vdots \\ y_{d,1} & y_{d,2} & \cdots & y_{d,n} \end{pmatrix}$$

Le produit vectoriel devient un simple produit matriciel. Le calcul de la norme de X et de Y correspond ici à la norme de chaque colonne des matrices X et Y. Les remarques relatives à l'ajout de $\|Y\|$ après l'étape de tri restent valides.

Nous avons donc écrit un code CUDA recherchant les kPPV selon cette méthode de calcul des distances. Les fonctions kernel que nous avons développées sont les suivantes :

1. Calcul de la norme de chaque vecteur colonne de X. La grille et les blocs utilisés sont mono-dimensionnels de manière à ce que chaque thread calcule la norme d'un vecteur colonne. Cette fonction est extrêmement optimisée car la lecture et l'écriture en mémoire globale est coalescente.
2. Calcul de la norme de chaque vecteur colonne de Y. Cette fonction est identique à la précédente.
3. Calcul du produit matriciel ${}^tX.Y$. Ce calcul est assuré par la fonction CUBLAS `cublasSgemm` spécialisée dans la multiplication et l'addition de matrices. La transposition est assurée par cette même fonction. Ce calcul représente la majorité du temps de calcul.
4. Ajout de la norme des vecteurs de référence aux distances calculées. Nous définissons une grille et des blocs de threads de manière à ce qu'il y ait un thread par élément de la matrice. Cette fonction est pénalisée par le fait que la lecture dans la mémoire globale n'est pas coalescente. En effet, la norme d'un vecteur de référence est ajoutée à tous les éléments d'une même ligne de la matrice des distances. L'utilisation de la mémoire partagée permet d'optimiser les temps de calcul.
5. Tri par insertion. Cette fonction kernel est identique à celle utilisée dans le chapitre 8.

6. Ajout de la norme des vecteurs requête aux k plus petites distances. Nous définissons une grille et des blocs de threads de manière à ce qu'il y ait un thread par colonne de la matrice de distances. Chaque norme est ajoutée à tous les éléments de la même colonne dans la matrice de distances. Ainsi, la lecture et l'écriture dans la mémoire globale sont coalescentes.

Nous utilisons ainsi des fonctions CUDA et une fonction CUBLAS au sein d'un même programme prouvant de fait que cela est tout à fait possible. Le lecteur notera que le nombre de fonctions kernel pour le calcul des distances est de 6 pour cette implémentation contre 2 pour l'implémentation présentée dans le chapitre 8.

§ 10.3 EXPERIMENTATIONS

Dans cette section, nous comparons l'implémentation BF de la recherche des kPPV proposée dans le chapitre 8 à l'implémentation fondée sur l'utilisation de CUBLAS. L'expérimentation porte sur des données synthétiques tirées aléatoirement selon une loi uniforme $\mathcal{U}(0,1)$. La configuration informatique utilisée est identique à celle donnée dans la section 9.1.2.
Considérons que le nombre de points de référence et de point requête est identique. Rappelons que 3 paramètres entrent en jeu dans la recherche des kPPV : le nombre de points noté n, la dimension des points notée d, et le paramètre k. Le paramètre k n'est utilisé que lors de l'étape de tri. L'implémentation CUDA et l'implémentation CUBLAS partageant la même fonction kernel de tri, ce paramètre influence identiquement ces deux implémentations. L'étude de ce paramètre est par conséquent inutile. Seuls n et d ont une influence sur la durée du calcul des distances. Ainsi, nous comparons dans la suite l'influence de ces deux paramètres sur le temps total de la recherche. Il n'est pas possible de mesurer le temps du calcul des distances seul car, pour l'implémentation utilisant CUBLAS, ce calcul est réparti sur plusieurs fonctions kernel encadrant l'étape de tri.

10.3.1 Influence de la dimension

La figure 10.1 montre l'évolution du temps de calcul en fonction de la dimension pour l'implémentation CUDA (notée BF-CUDA) et pour l'implémentation CUBLAS (notée BF-CUBLAS). Nous utilisons pour ce faire 4800 points et nous fixons $k = 20$. Quelle que soit l'implémentation choisie, le temps de calcul augmente linéairement avec la dimension. Ceci s'explique par le fait que dans les deux cas le nombre d'opérations pour le calcul d'une distance augmente linéairement avec d. En revanche, malgré le fait qu'il y ait plus de fonctions kernel pour BF-CUBLAS, l'augmentation est plus importante avec BF-CUDA qu'avec BF-CUBLAS. L'explica-

tion vient du fait que le produit matriciel, produit correspondant à la majorité des calculs, est extrêmement optimisé pour BF-CUBLAS. Ainsi, pour des hautes dimensions, BF-CUBLAS est jusqu'à 2 fois plus rapide que BF-CUDA selon nos expérimentations. Pour des faibles dimensions, le calcul des distances (produit matriciel) est moins lourd. Dans ce cas, le nombre des fonctions kernel dans BF-CUBLAS ainsi que la fonction kernel effectuant des lectures non coalescentes pénalisent fortement les temps de calcul. Ceci explique que BF-CUDA soit plus rapide que BF-CUBLAS en faible dimension. Pour 4800 points, l'équilibre (c'est-à-dire la dimension pour laquelle les deux implémentations sont équivalentes en termes de temps de calcul) se situe approximativement pour $d = 16$. Au-delà de cette valeur, BF-CUBLAS est plus intéressante. En deçà, BF-CUDA est l'implémentation la plus rapide.

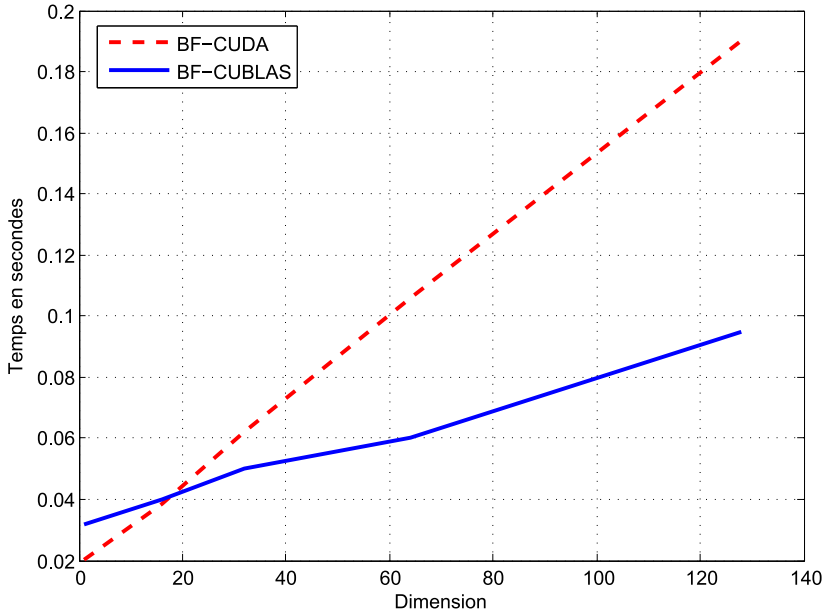


FIGURE 10.1 – *Évolution du temps de calcul en fonction de la dimension des points. Pour cette expérimentation, 4800 points (référence et requête) sont utilisés et k est fixé à 20. L'augmentation du temps de calcul est linéaire. En haute dimension, BF-CUBLAS (ligne bleue) est plus rapide que BF-CUDA (ligne en pointillés rouges).*

10.3.2 Influence du nombre de points

La figure 10.2 présente l'évolution du temps de calcul en fonction du nombre de points. Tout d'abord, l'augmentation du temps de calcul est polynomiale. Ce comportement est logique. En effet, si n désigne le nombre de points (référence et requête), le nombre de distances calculées est n^2.

Nous donnons dans la figure 10.2 l'évolution pour $d = 8$ et $d = 32$. BF-CUDA semble plus rapide que BF-CUBLAS pour $d = 8$ et moins rapide pour $d = 32$. Dans les deux cas, les temps de calcul sont identiques pour un faible nombre de points. La différence entre les deux implémentations croît avec le nombre de points. Le fait que BF-CUDA soit plus ou moins rapide que BF-CUBLAS a une explication très simple. Nous avons vu que la dimension influait sur le temps de calcul des deux implémentations et que BF-CUBLAS était plus efficace en haute dimension. Ici, nous avons délibérément choisi comme dimensions $d = 8$ et $d = 32$ car elles se situent de part et d'autre de la dimension pour laquelle BF-CUDA et BF-CUBLAS sont équivalentes. Pour $d = 8$, nous savons que BF-CUDA est plus rapide que BF-CUBLAS. Ainsi, plus le nombre de points augmente, plus le nombre de distances à calculer est grand, et plus le gain induit par BF-CUDA augmente. À l'inverse, pour $d = 32$, BF-CUBLAS étant plus rapide que BF-CUDA, le gain induit par BF-CUBLAS augmente avec le nombre de points.

§ 10.4 CONCLUSION

BLAS est un ensemble de fonctions basiques spécialisées dans l'algèbre linéaire. Ces fonctions ont la particularité d'être très optimisées ce qui leur vaut d'être utilisées par de nombreuses librairies (*e.g.* LAPACK) et de nombreux logiciels (*e.g.* MATLAB). CUBLAS (CUDA-BLAS) est une implémentation CUDA de BLAS. Il est ainsi possible de profiter de la puissance de la programmation GPU tout en utilisant des fonctions BLAS classiques. L'objectif de ce chapitre était d'évaluer l'utilisation de CUBLAS dans notre application de recherche des kPPV.
La recherche des kPPV consiste principalement dans le calcul des distances entre les points requête et les points de référence. Ce problème peut aisément se réécrire sous forme d'un produit matriciel, au moins en ce qui concerne la plus grande partie des calculs. Cette réécriture présente l'avantage de pouvoir utiliser la librairie CUBLAS. Selon nos expérimentations, il apparaît que l'utilisation de CUBLAS est particulièrement intéressante en haute dimension ($d > 16$). La recherche des kPPV est alors jusqu'à 2 fois plus rapide qu'avec une implémentation CUDA plus classique (voir chapitre 9). En revanche, les fonctions annexes nécessaires pour le calcul des distances pénalisent les temps de calcul pour de faibles dimensions. Pour $d < 16$, l'implémentation CUDA présentée est plus intéressante en termes de temps de calcul que l'implémentation CUBLAS.
Dans le chapitre 9, nous avons volontairement omis de parler de l'implémentation de la méthode BF en CUBLAS. En effet, le but ce chapitre était d'étudier l'intérêt de l'utilisation CUDA pour implémenter un problème simple. La comparaison était faite avec des implémentations similaires écrites

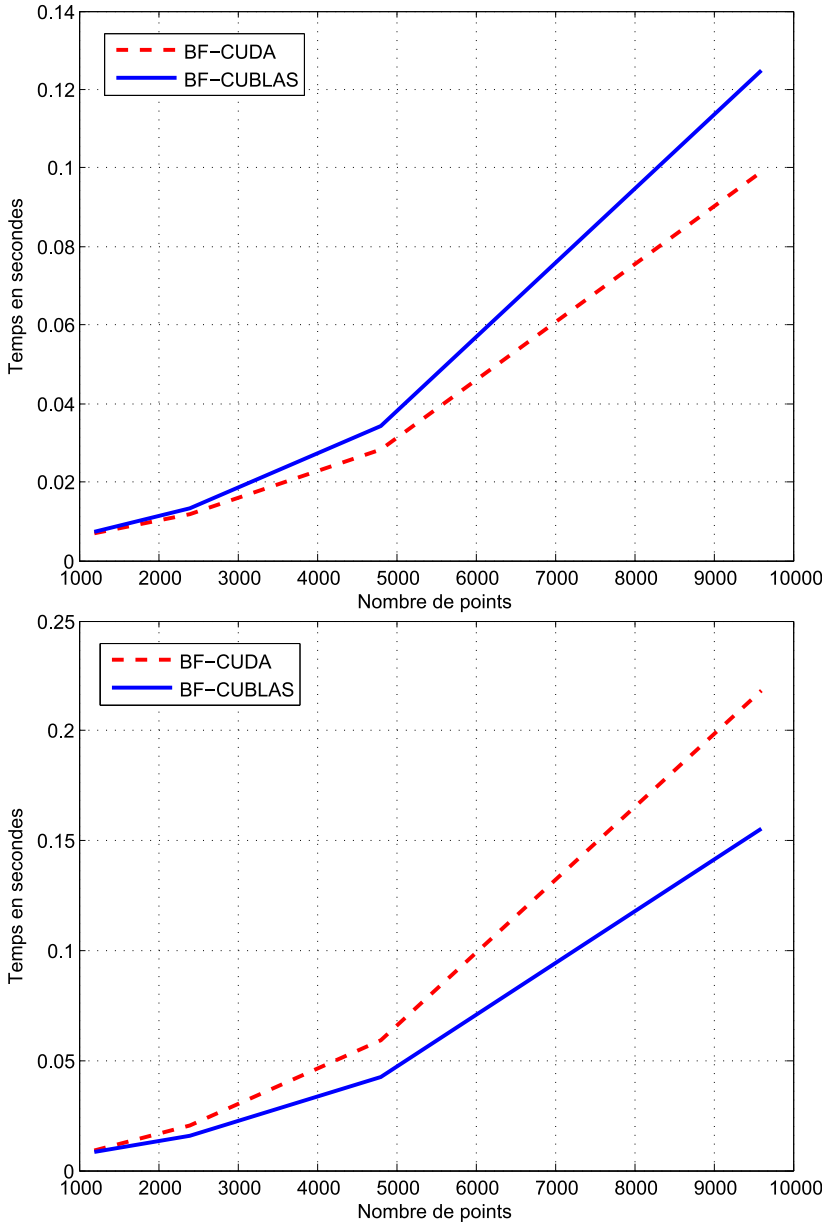


FIGURE 10.2 – *Évolution du temps de calcul en fonction du nombre de points. Pour ces expérimentations, les dimensions utilisées sont $d = 8$ (figure du haut) et $d = 32$ (figure du bas). Le paramètre k est fixé à 20. L'augmentation du temps de calcul est polynomiale. Pour $d = 8$ BF-CUDA (ligne en pointillés rouges) est plus rapide que BF-CUBLAS (ligne bleue). Pour $d = 32$, BF-CUBLAS est l'implémentation la plus rapide.*

en Matlab et C et avec une méthode écrite en C++ fondée sur l'utilisation de kd-trees. La plupart des problèmes n'étant pas représentables par des opérations d'algèbre linéaire simples, l'utilisation de CUBLAS dans ce contexte était inadaptée.

Troisième partie

Conclusion générale et perspectives

- CHAPITRE 11 -

CONCLUSION GÉNÉRALE

Dans cette thèse, nous avons abordé le problème du suivi d'objets dans une vidéo. En particulier, nous avons présenté plusieurs approches. La première est fondée sur le suivi de points d'intérêt, la seconde sur le suivi de la couronne des objets, et la dernière sur l'utilisation de mesures statistiques de similarité. Pour cette dernière, nous nous sommes particulièrement attachés à l'utilisation de la programmation GPU pour l'accélération de nos algorithmes.

§ 11.1 SUIVI D'APRÈS LES TRAJECTOIRES DE POINTS D'INTÉRÊT

11.1.1 Bilan des travaux présentés

Nous avons proposé une méthode de suivi fondée sur l'analyse temporelle de trajectoires de points d'intérêt. Ces trajectoires, représentant le mouvement de l'objet à suivre, sont utilisées pour extraire un ensemble d'informations relatives au mouvement de l'objet. L'estimation du mouvement de cet objet est réalisée par la minimisation d'une fonctionnelle pondérée.
L'utilisation des points d'intérêt permet de construire des trajectoires temporelles à la fois fiables et précises. Ces trajectoires représentent fidèlement le mouvement de la plupart des objets contenus dans la vidéo. Les informations extraites de celles-ci permettent une estimation précise du mouvement de l'objet à suivre.
L'utilisation d'un groupe d'images permet d'augmenter le nombre d'informations relatives au mouvement de l'objet, ce qui implique une amélioration de la précision et de la robustesse (aux données aberrantes) de l'estimation du mouvement. Nous proposons de plus d'utiliser une pon-

dération spatio-temporelle appliquée aux informations extraites des trajectoires. L'utilisation d'une pondération temporelle a pour effet d'adoucir l'hypothèse de constance de mouvement tandis que la pondération spatiale permet de tenir compte des déformations locales de l'objet.
D'après nos expérimentations, la méthode proposée permet de suivre précisément un objet d'intérêt en dépit du mouvement de l'objet et des occultations éventuelles.

11.1.2 Perspectives

Tout d'abord, le suivi proposé repose entièrement sur la précision des trajectoires de points d'intérêt. Selon la séquence utilisée, il est possible que le nombre de trajectoires soit trop faible pour permettre une estimation du mouvement fiable. Si l'objet est trop petit ou si la texture de l'objet n'est pas assez marquée, il est probable que peu de points d'intérêt soient détectés, impliquant de fait un faible nombre de trajectoires. L'utilisation d'algorithmes de super-résolution [PPK03] permettrait d'augmenter la taille des images et des objets tout en améliorant la qualité des images. Le nombre de points détectés (et par conséquent le nombre de trajectoires) augmenterait et leur précision s'améliorerait à condition évidemment que l'objet à suivre soit suffisamment texturé. L'inconvénient de cette méthode est le coût calcul induit premièrement par la super-résolution, et deuxièmement par l'augmentation du temps de détection des points d'intérêt.
La pondération spatiale que nous utilisons est fonction de la distance Euclidienne entre le point d'échantillonnage considéré et l'origine de la trajectoire. Les objets rencontrés sont pour la plupart globalement convexes. Cependant, pour des objets non convexes (*e.g.* objets articulés : corps humain), deux points proches au sens de la distance Euclidienne peuvent avoir des mouvements très différents (*e.g.* points situés sur les deux pieds). L'utilisation d'une distance géodésique ajouterait une notion de distance interne à l'objet ce qui améliorerait la prise en compte des mouvements locaux de l'objet.

§ 11.2 Suivi de la couronne de l'objet

11.2.1 Bilan des travaux présentés

La seconde méthode de suivi proposée repose sur l'utilisation de la couronne de l'objet. Le mouvement de chaque point d'échantillonnage du contour est suivi au cours du temps par une simple méthode de block-matching. Le problème principal auquel nous nous sommes attachés vient du fait que les blocs centrés sur les points d'échantillonnage du contour sont des blocs tangents contenant à la fois des pixels de l'objet et des pixels

du fond. Les pixels du fond, considérés comme aberrants, perturbent l'estimation du mouvement.
Nous avons proposé de masquer partiellement les pixels du fond en utilisant le masque de l'objet dilaté par une opération de morphologie mathématique. Ce masquage permet d'une part de diminuer la proportion de pixels du fond ce qui réduit leur impact lors du processus de block-matching, et d'autre part, de conserver la structure de bord essentielle lorsqu'un objet est localement homogène.
La dilatation du masque de l'objet conserve une partie des pixels du fond. Selon le rayon de dilatation, la proportion de ces pixels peut tout de même induire une mauvaise estimation du mouvement. Ce comportement est lié au fait que l'utilisation de critères de comparaison classiques est fondée sur une hypothèse de distribution paramétrique du résiduel, hypothèse généralement fausse. Nous avons alors proposé de remplacer ce critère de comparaison par un critère semi-paramétrique fondé sur l'estimation de l'entropie du résiduel. Il apparaît alors que ce critère est plus robuste aux pixels aberrants.

11.2.2 Perspectives

Lors de l'étape d'estimation de mouvement, les blocs ne contiennent que des informations relatives à la couleur (YUV ou RVB). L'utilisation d'autres composantes telles que les informations relatives au gradient de l'image permettrait d'améliorer la description des blocs en augmentant leur pouvoir discriminant. Cette amélioration se répercuterait alors automatiquement sur la qualité du suivi. Cependant, en considérant le fait que les blocs ne contiennent que peu de points, le nombre de composantes ajoutées devrait être faible pour éviter la malédiction de la dimension (de l'anglais *Curse of dimensionality*).
La taille de la bande de pixels considérée est un paramètre que nous fixons manuellement. Cependant, il serait judicieux de fixer localement cette taille en fonction de plusieurs critères. La proportion de pixels du fond par rapport aux pixels de l'objet dépend de la courbure locale de l'objet en chaque point d'échantillonnage. Plus la courbure de l'objet (convexe) est importante, plus la part des pixels du fond est importante et inversement. De même, l'influence des pixels de l'objet et du fond sur le critère de comparaison dépend de leur texture respective. Ainsi, la prise en compte de la courbure locale de l'objet, de la texture du fond, et de la texture de l'objet permettrait de déterminer une taille adéquate de la couronne ce qui améliorerait la qualité du suivi d'objet.

§ 11.3 Suivi d'objets fondé sur des mesures statistiques

11.3.1 Bilan des travaux présentés

L'ultime méthode de suivi d'objets présentée repose sur les travaux menés au cours de la thèse de Sylvain Boltz [BDB07, Bol08]. Cette méthode consiste à décrire un objet par un ensemble de composantes couleurs et de composantes géométriques. La mesure de similarité utilisée entre descripteurs est la divergence de Kullback-Leibler. Pour éviter la malédiction de la dimension, Boltz propose d'estimer cette divergence par la distance au k-ème plus proche voisin.
La recherche des k plus proches voisins (kPPV) est un problème présent dans de nombreuses applications de traitement d'images, mais également dans des applications très différentes telles que la finance, la biologie, la physique, la génétique, etc. La recherche des kPPV est un problème long à résoudre, surtout quand le nombre de points et la dimension de ces mêmes points augmentent. Certaines méthodes récentes fondées sur des kd-trees permettent d'accélérer la recherche. Toutefois, le processus reste lent. Cette lenteur est un inconvénient majeur généralement résolu en limitant la dimension et le nombre de requêtes et de références.
Nous avons proposé dans cette thèse une implémentation GPU de la méthode de recherche exhaustive des k plus proches voisins. Cette méthode est par nature hautement parallélisable ce qui lui vaut d'être adaptée à la programmation parallèle sur processeur graphique. Nous avons utilisé pour cela la récente API NVIDIA CUDA. Nous avons montré que notre implémentation accélérait grandement la recherche des k plus proches voisins en comparaison des approches classiques (ANN), et ce sur des données synthétiques mais également dans le cadre du suivi d'objets évoqué précédemment.

11.3.2 Perspectives

L'algorithme proposé de recherche des k plus proches voisins est bien adapté à la programmation parallèle. Il est cependant important de remarquer que toutes les distances sont calculées pour un point requête donné. La structuration des données pourrait permettre d'accélérer le temps de recherche en minimisant le nombre de distances calculées. Il est cependant essentiel de considérer un algorithme adapté à la programmation GPU. En effet, la création de la structure de données doit être parallélisable pour espérer apporter un gain significatif par rapport à la programmation CPU.
De manière plus générale, le développement de nouveaux algorithmes (quels qu'ils soient) devrait tenir compte de la puissance de calcul offerte par la

programmation GPU. Cette révolution permet d'envisager de nouvelles approches jusque là impossibles en programmation classique.
Enfin, dans la mesure où certaines méthodes ne peuvent être parallélisées, il serait intéressant de concevoir des algorithmes de recherche des k plus proches voisins hybrides alliant programmation CPU et programmation GPU. En effet, la programmation GPU ne tire pas un trait sur des décennies de recherches fondées sur la programmation CPU. L'avenir passe certainement par des méthodes sachant tirer profit des deux technologies.

Quatrième partie

Annexes

- ANNEXE A -

ESTIMATION DE L'ENTROPIE

L'entropie de Shannon, due à Claude Shannon [Sha48, CT91], est une fonction mathématique qui correspond à la quantité d'information contenue ou délivrée par une source d'information. Cette source peut être une langue, un signal électrique, ou un fichier informatique quelconque. La définition de l'entropie de Shannon d'une source est telle que plus la source est redondante, moins elle contient d'information au sens de Shannon. En l'absence de contraintes particulières, l'entropie est ainsi maximale pour une source dont tous les symboles sont équiprobables. Les deux motivations de l'entropie sont (1) l'incertitude d'une variable aléatoire et (2) l'information. Une étude approfondie de l'estimation d'entropie est présentée dans la thèse de E. Wolsztynski [Wol06].

§ A.1 DÉFINITION

Soit X une variable aléatoire continue. La probabilité de l'état x_i est notée $p(x_i)$:

$$p(x_i) = P_X(x_i) = P(X = x_i) \tag{A.1}$$

L'entropie différentielle de Shannon de la variable X notée $H(X)$, également appelée information propre moyenne, se définie par

$$\begin{aligned} H(X) &= E_X[I(x)] && \text{(A.2)} \\ &= \int p(x) I(x) dx && \text{(A.3)} \\ &= \int p(x) \log\left(\frac{1}{p(x)}\right) dx && \text{(A.4)} \\ &= -\int p(x) \log p(x) dx && \text{(A.5)} \end{aligned}$$

où $I(x)$ désigne l'information propre de l'état x. Pour X une variable aléatoire discrète prenant ses valeurs dans l'univers des possibles $\mathcal{X} = \{x_1, x_2, \cdots, x_m\}$, l'entropie de Shannon de X se définie par

$$H(X) = -\sum_{x \in \mathcal{X}} p(x) \log p(x) \tag{A.6}$$

Soit Y une variable aléatoire (continue ou discrète) et $\{y_1, y_2, \cdots, y_n\}$ un ensemble de n réalisations de la variable Y appelé « échantillon de Y ». L'approximation de Ahmad-Lin [AL76] permet d'estimer l'entropie différentielle de Shannon à partir de la probabilité des points de l'échantillon :

$$H(Y) \approx -\frac{1}{n} \sum_{i=1}^{n} \log p(y_i) \tag{A.7}$$

Dans la suite, nous utiliserons cette approximation pour calculer l'entropie car elle nécessite en général moins de calculs.

§ A.2 ESTIMATION

L'estimation de l'entropie d'une variable aléatoire Y est triviale si la loi de cette variable est connue. Dans le cas contraire, l'entropie doit être estimée à partir de l'échantillon de Y. Il est alors nécessaire de connaître la densité de probabilité des points de l'échantillon, c'est-à-dire $p(y_i)$ pour i dans $[1, n]$, si nous utilisons l'approximation de Ahmad-Lin. Nous présentons dans ce qui suit trois méthodes permettant d'estimer cette densité et d'en déduire la valeur de l'entropie.

A.2.1 Histogramme

La méthode la plus simple, la plus ancienne et la plus utilisée est l'utilisation d'un histogramme construit à partir de l'échantillon $\{y_1, \cdots, y_n\}$. Cette méthode est particulièrement adaptée aux variables discrètes, surtout si nous connaissons l'univers des possibles de la variable Y (par exemple les pixels d'une image en niveaux de gris prennent leur valeur dans $[0, 255]$). La fréquence d'apparition des points de l'échantillon dans chaque classe est alors calculée. L'essentiel de la méthode réside dans l'estimation des paramètres de l'histogramme, c'est-à-dire la largeur l des classes de l'histogramme. La figure A.1 illustre l'importance du choix de cette largeur : une largeur trop grande ou trop faible ne permet pas d'estimer précisément la densité de probabilité de la variable Y. L'ultime étape consiste à normaliser les valeurs de l'histogramme de manière à ce que l'aire totale des classes de

l'histogramme soit égale à 1. La densité de probabilité correspond trivialement à l'amplitude de la classe. L'entropie estimée $\widehat{H}_{n,l}$ de l'échantillon par la méthode des histogrammes s'écrit donc :

$$\widehat{H}_{n,l}(Y) = -\frac{1}{n}\sum_{i=1}^{n}\log(h(y_i)) \tag{A.8}$$

où $h(y_i)$ représente l'amplitude de la classe à laquelle le point y_i appartient. Cette méthode se généralise simplement aux dimensions supérieures, c'est-à-dire dans le cas où Y est vecteur aléatoire (dimension supérieure à 1). La méthode est en tout point similaire à ceci prêt que l'on utilise alors un histogramme à plusieurs dimensions et que la normalisation finale utilise le volume des classes et non leur aire. Enfin, si le choix de la largeur des classes est déjà complexe en dimension 1 pour estimer correctement la densité de Y, ce choix est encore plus difficile en dimension supérieure.

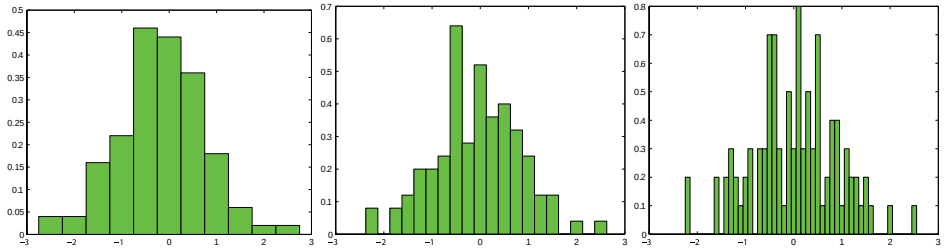


FIGURE A.1 – *Estimation de densité de probabilité par une méthode d'histogramme à partir d'un échantillon de* 100 *points pour une variable aléatoire de loi normale* $\mathcal{N}(0,1)$*. Les trois histogrammes présentés utilisent le même échantillon. Respectivement de gauche à droite, les classes sont de largeur* 0.5, 0.25, *et* 0.1. *En fonction de la largeur choisie, la densité est plus ou moins bien estimée.*

A.2.2 Méthode à noyau

L'estimateur à noyau a été défini par Parzen [Par62]. Cette méthode consiste à placer un noyau (par exemple un noyau Gaussien) centré en chaque point de l'échantillon et à effectuer la somme de ces noyaux. La figure A.2 illustre la méthode à noyau pour un échantillon de 100 points d'une variable aléatoire de loi uniforme $\mathcal{U}(0,1)$. L'entropie estimée $\widehat{H}_{n,\sigma^2}$ de l'échantillon par la méthode à noyau s'écrit donc :

$$\widehat{H}_{n,\sigma^2}(Y) = -\frac{1}{n}\sum_{i=1}^{n}\log(\widehat{p}_{\sigma^2}(y_i)) \tag{A.9}$$

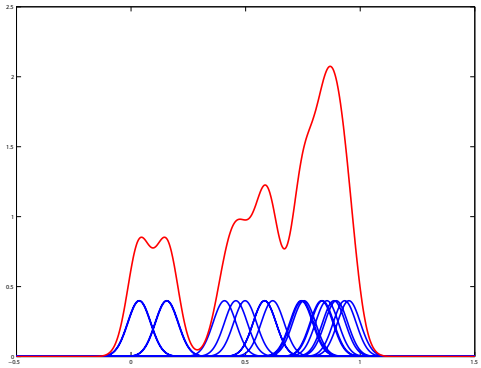


FIGURE A.2 – *Estimation de densité de probabilité par une méthode à noyau à partir d'un échantillon de* 20 *points d'une loi uniforme* $\mathcal{U}(0,1)$*. Un noyau Gaussien (en bleu) de variance* $\sigma^2 = 0.0025$ *est centré en chaque point. La densité de probabilité estimée (en rouge) est égale à la somme des noyaux. La précision de l'estimation de densité dépend du choix du noyau et de ses paramètres.*

La probabilité estimée du point y_i, notée $\widehat{p}_{\sigma^2}(y_i)$, dépend du choix du noyau f. Dans le cas d'un noyau Gaussien, la probabilité s'écrit :

$$\widehat{p}_{\sigma^2}(y_i) = \sum_{j=1}^{n} \frac{1}{n} f(y_i; y_j, \sigma^2) \tag{A.10}$$

où

$$f(x; \mu, \sigma^2) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}. \tag{A.11}$$

Les paramètres μ et σ^2 sont respectivement la position du centre et la variance du noyau Gaussien. Notez que le noyau est normalisé pour sommer à $1/n$ de manière à ce que la densité de probabilité somme à 1. La variance σ^2 du noyau est un paramètre à estimer. Il existe dans la littérature plusieurs approches permettant d'estimer sa valeur optimale en fonction des observations et d'un critère d'optimalité à définir/choisir (souvent de type MSE pour « Mean Square Error »). Le choix du noyau et de ses paramètres est déterminant pour estimer correctement la densité de probabilité de Y (voir figure A.3). Enfin, en dimension supérieure, l'utilisation de noyau multivarié permet de généraliser cette approche. Cependant, l'estimation de la matrice de covariance du noyau en dimension supérieure s'avère beaucoup plus complexe qu'en dimension 1.

A.2.3 k-ème plus proche voisin

La dernière méthode d'estimation repose sur la statistique des « k-ème plus proche voisin »(kPPV). Nous considérons en effet l'utilisation de cette

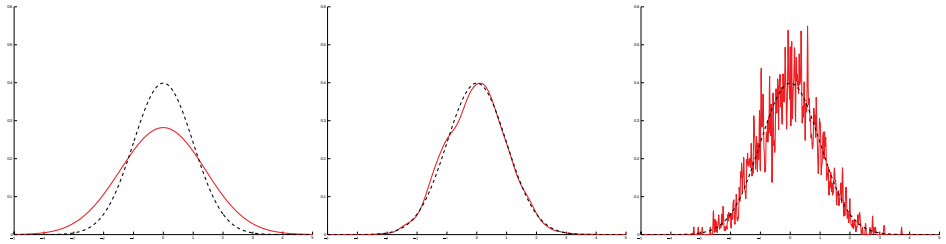


FIGURE A.3 – *Influence du choix des paramètres du noyau sur l'estimation de densité de probabilité. Les trois figures présentées utilisent un même échantillon de* 2000 *points pour une variable aléatoire de loi normale* $\mathcal{N}(0,1)$*. La densité de probabilité réelle est représentée, sur les trois figures, en tirets noirs. Respectivement de gauche à droite, les noyaux sont des Gaussiennes de variance* 1, 0.2, *et* 0.01. *La densité est correctement estimée pour une variance de* 0.2.

statistique pour chacune des observations selon une métrique donnée, par exemple la distance euclidienne, pour estimer l'entropie de l'échantillon. Pour $k = 1$, Kozachenko et Leonenko [KL87] introduisent un estimateur de l'entropie de Shannon d'une variable aléatoire observée en utilisant cette statistique. Cet estimateur est étendu à $k > 1$ dans [GLMI05].
Cette méthode est particulièrement adaptée à l'estimation d'entropie en dimension supérieure. Nous nous plaçons ici dans une telle configuration. Soit Y un vecteur aléatoire de dimension d. Nous disposons d'un échantillon de n points $\{y_1, y_2, \cdots, y_n\}$. Pour un point y_i, nous calculons sa distance $\rho_k(y_i)$ au k-ème plus proche voisin. La boule fermée centrée en y_i et de rayon $\rho_k(y_i)$ contient ainsi les k points les plus proches de y_i. La Figure A.4 présente un échantillon de 100 points pour un vecteur aléatoire de dimension 2 et de loi $\mathcal{N}(0, I)$. Une petite taille des disques, en principe accompagnée par des chevauchements de disques plus fréquents à cet endroit, indique une densité plus importante de points dans le nuage considéré ; c'est l'information apportée par la statistique du k-ème plus proche voisin sur l'échantillon considéré. L'estimation de l'entropie dépend de n (nombre de points dans l'échantillon) et de k (k-ème plus proche voisin) :

$$\widehat{H}_{n,k}(Y) = \frac{1}{n}\sum_{i=1}^{n} \log((n-1)\rho_k(y_i)) + \log(c_1(d)) - \Psi(k) \tag{A.12}$$

où $\Psi(k)$ est la fonction digamma

$$\Psi(k) = \frac{\Gamma'(k)}{\Gamma(k)} = \int_0^\infty \left[\frac{e^{-t}}{t} - \frac{e^{-kt}}{(1-e^{-t})}\right] dt \tag{A.13}$$

et

$$c_1(d) = \frac{2\pi^{\frac{d}{2}}}{d\Gamma(\frac{d}{2})} \tag{A.14}$$

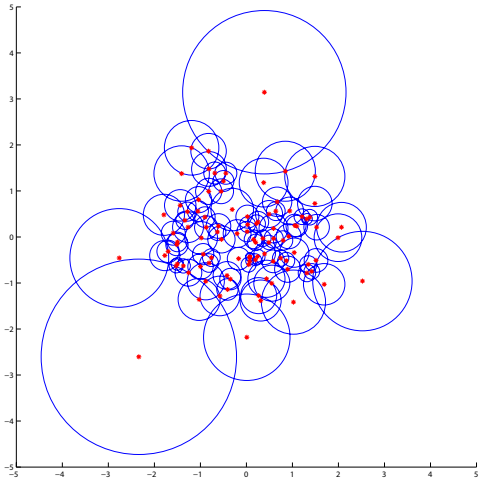


FIGURE A.4 – *Illustration de* 100 *réalisations d'un vecteur aléatoire de dimension* 2 *et de loi normale* $\mathcal{N}(0, I)$. *Un cercle centré en chaque réalisation représente la distance au k-ème plus proche voisin, avec* $k = 2$ *dans notre exemple.*

donne le volume de la boule unité dans $\mathbb{R}^d$. La fonction gamma Γ est l'extension de la fonction factorielle à l'ensemble des nombres complexes (excepté en certains points). La fonction digamma n'a pas de formule close et doit être calculée. Cette forme d'estimateur nécessite encore de choisir la valeur de k, mais il apparaît le plus souvent en pratique que la méthode est peu sensible à un écart de la valeur de k et le réglage du « paramètre technique »ne constitue plus une difficulté critique à la différence de la largeur des noyaux dans (A.10). Cet estimateur a d'ores et déjà prouvé son intérêt dans de nombreuses applications [WTP05, BWD+06]. Pour plus de détails sur cette méthode d'estimation d'entropie, il est conseillé de lire la thèse de Wolsztynski [Wol06].
Comme nous l'avons déjà mentionné, cette méthode permet d'estimer l'entropie d'un échantillon quelle que soit la dimension du vecteur aléatoire associé à l'échantillon. Nous utiliserons cet estimateur d'entropie dans la section A.3.

§ A.3 INFLUENCE DU NOMBRE D'ÉCHANTILLONS

Nous disposons d'outils puissants permettant de calculer l'entropie d'une variable aléatoire ou d'un vecteur aléatoire de dimension d. Dans cette section, nous nous posons le problème suivant. Quelle taille d'échantillon est nécessaire pour calculer précisément l'entropie de Y ? Pour cette étude, nous utilisons une variable aléatoire de loi Gaussienne $\mathcal{N}(0, 1)$. La valeur théorique de l'entropie d'une telle loi est $H(y) = \ln(\sigma\sqrt{2\pi e}) \approx 1.4189$.

Nous faisons varier la taille de l'échantillon utilisé de 100 à 4000 et nous répétons l'estimation d'entropie 1000 fois (d'où 1000 échantillons) pour chaque taille d'échantillon testée. Les figures A.5 et A.6 représentent respectivement la moyenne de l'entropie et la variance de l'entropie estimées pour 1000 échantillons. Nous pouvons y voir que la valeur de l'entropie estimée augmente avec la taille de l'échantillon considéré pour converger vers la valeur théorique de l'entropie. *A contrario*, la variance diminue vers 0. Au vu de ces courbes, un échantillon de 1000 points semble être le bon compromis entre nombre de points et précision de l'estimation. En effet, à partir de cette taille d'échantillon, l'évolution de la moyenne de l'entropie estimée ainsi que sa variance semblent quasi nulles. La précision de l'estimation est alors de l'ordre de 10^{-3}. Ainsi, nous considérons que, pour estimer précisément l'entropie, nous devons utiliser des échantillons de 1000 points.
Le nombre de points minimal dans un échantillon permettant de précisément estimer l'entropie dépend naturellement de l'estimateur d'entropie utilisé. Ici, nous avons utilisé l'estimateur utilisant la statistique des kPPV. Pour une méthode à noyau, il faudrait procéder à une importante étude sur la largeur des noyaux utilisés. Notez que l'utilisation de techniques de ré-échantillonnage (par exemple bootstrap [Efr79]) permet de fortement diminuer ce nombre.

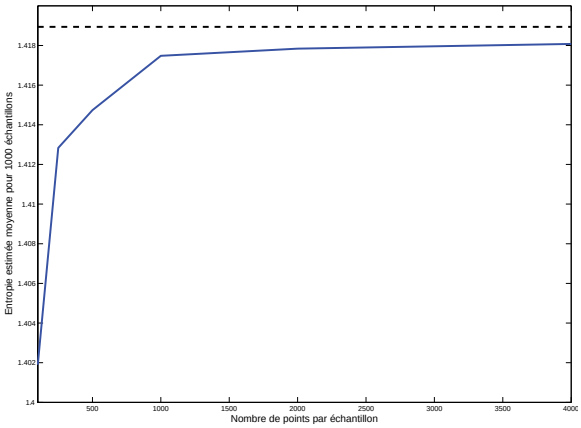


FIGURE A.5 – *Évolution de l'entropie moyenne estimée pour* 1000 *échantillons en fonction de la taille des échantillons. Chaque échantillon est généré à partir d'une loi normale* $\mathcal{N}(0,1)$*. L'entropie augmente avec la taille de l'échantillon considéré pour converger vers la valeur théorique de l'entropie (*$H(y) \approx 1.4189$*) représentée par une ligne discontinue noire. À partir d'un échantillon de* 1000 *points, la pente de la courbe est quasi nulle. L'entropie estimée approxime l'entropie théorique avec une précision de* 4.10^{-3}*.*

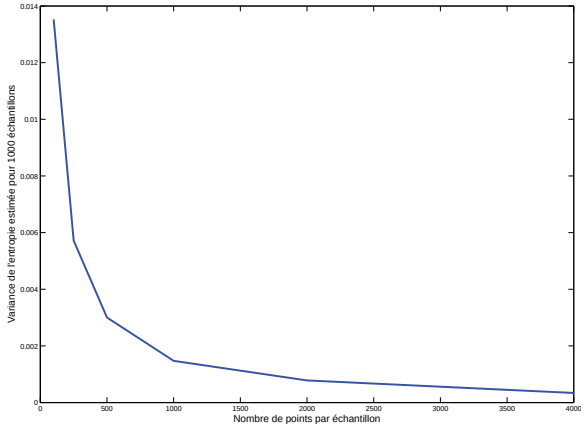


FIGURE A.6 – *Évolution de la variance de l'entropie estimée pour* 1000 *échantillons en fonction de la taille des échantillons. Chaque échantillon est généré à partir d'une loi normale* $\mathcal{N}(0,1)$*. La variance diminue avec la taille de l'échantillon considéré. À partir d'un échantillon de* 1000 *points, la pente de la courbe est très faible. La variance de l'entropie estimée vaut* 3.10^{-3}*.*

- ANNEXE B -

INFLUENCE DU CRITÈRE D'ESTIMATION DE MOUVEMENT

Nous avons étudié dans le chapitre 4 l'estimation de la matrice M^t décrivant le mouvement du contour C^t vers le contour C^{t+1}. Cette matrice est estimée à partir d'un ensemble de couples de points d'intérêt $\{p_i^t, p_i^{t+1}\}$ représentant respectivement la position des points p_i sur les images I^t et I^{t+1}. Nous y avons également étudié l'estimation du mouvement sur un groupe d'images (noté GOP pour « Group Of Pictures ») mais nous nous intéressons dans cette annexe au premier cas plus simple à étudier. Pour simplifier les notations, M représentera dans la suite le mouvement réel du contour C^t vers le contour C^{t+1}, et $\widehat{M}$ représentera le mouvement estimé de M. Nous avons les égalités suivantes

$$p_i^{t+1} = M.p_i^t \tag{B.1}$$

$$p_i^{t+1} = \widehat{M}.p_i^t + \epsilon_{i,\widehat{M}} \tag{B.2}$$

où ϵ_i est l'erreur de localisation dépendant du point p_i et de la matrice estimée $\widehat{M}$. La matrice de mouvement est estimée comme suit :

$$\widehat{M} = \arg\min_M \sum_{i=1}^{n} f(\|p_i^{t+1} - M.p_i^t\|) \tag{B.3}$$

$$= \arg\min_M \sum_{i=1}^{n} f(\|\epsilon_{i,M}\|) \tag{B.4}$$

où f est une fonction coût, $\|.\|$ la norme euclidienne, et M une matrice de mouvement affine à six paramètres

$$M = \begin{pmatrix} a & b & c \\ d & e & f \\ 0 & 0 & 1 \end{pmatrix}. \tag{B.5}$$

Nous utilisons dans le chapitre 4 une minimisation du M-estimateur (B.4) par la méthode du simplexe [Dan51, Nas00, LRWW98] couplée à l'utilisation de la fonction coût valeur absolue $f(x) = |x|$. Nous présentons dans cette annexe une autre approche particulièrement utilisée dans l'industrie et communément appelée « estimateur des moindres carrés ordinaires ». Cette méthode permet d'obtenir les paramètres optimaux en une étape ce qui en fait un algorithme très rapide. Nous montrons dans cette annexe comment formuler notre problème pour pouvoir utiliser cette méthode.
Le vecteur d'erreur $\epsilon_{i,M}$ s'écrit

$$\begin{aligned}
\epsilon_{i,M} &= p_i^{t+1} - \widehat{M}.p_i^t && \text{(B.6)}\\
&= \begin{pmatrix} p_{i,1}^{t+1} \\ p_{i,2}^{t+1} \\ 1 \end{pmatrix} - \begin{pmatrix} \widehat{a} & \widehat{b} & \widehat{c} \\ \widehat{d} & \widehat{e} & \widehat{f} \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} p_{i,1}^{t} \\ p_{i,2}^{t} \\ 1 \end{pmatrix} && \text{(B.7)}\\
&= \begin{pmatrix} p_{i,1}^{t+1} - \widehat{a}.p_{i,1}^t - \widehat{b}.p_{i,2}^t - \widehat{c} \\ p_{i,2}^{t+1} - \widehat{d}.p_{i,1}^t - \widehat{e}.p_{i,2}^t - \widehat{f} \\ 1 \end{pmatrix} && \text{(B.8)}\\
\epsilon_{i,M} &= \begin{pmatrix} \epsilon_{i,\{\widehat{a},\widehat{b},\widehat{c}\}} \\ \epsilon_{i,\{\widehat{d},\widehat{e},\widehat{f}\}} \\ 1 \end{pmatrix} && \text{(B.9)}
\end{aligned}$$

Les trois composantes du vecteur $\epsilon_{i,M}$ sont donc indépendantes. Dans le cas particulier où $f(x) = x^2$, l'équation (B.4) se récrit

$$\widehat{M} = \arg\min_M \sum_{i=1}^{n} \left[\epsilon_{i,\{\widehat{a},\widehat{b},\widehat{c}\}}\right]^2 + \left[\epsilon_{i,\{\widehat{d},\widehat{e},\widehat{f}\}}\right]^2 + 1 \qquad \text{(B.10)}$$

Chaque membre de la somme étant nécessairement positif ou nul, cette minimisation équivaut à minimiser chacun des termes séparément. Les paramètres $\{a, b, c\}$ et $\{d, e, f\}$ sont ainsi estimés séparément

$$\begin{aligned}
\{\widehat{a},\widehat{b},\widehat{c}\} &= \arg\min_{\{a,b,c\}} \sum_{i=1}^{n} \left[\epsilon_{i,\{a,b,c\}}\right]^2 && \text{(B.11)}\\
\{\widehat{d},\widehat{e},\widehat{f}\} &= \arg\min_{\{d,e,f\}} \sum_{i=1}^{n} \left[\epsilon_{i,\{d,e,f\}}\right]^2 && \text{(B.12)}
\end{aligned}$$

et la matrice de mouvement estimée $\widehat{M}^t$ s'écrit

$$\widehat{M} = \begin{pmatrix} \widehat{a} & \widehat{b} & \widehat{c} \\ \widehat{d} & \widehat{e} & \widehat{f} \\ 0 & 0 & 1 \end{pmatrix}. \qquad \text{(B.13)}$$

Les équations (B.11) et (B.12) sont des problèmes de régression linéaire classiques. Dans la suite, nous étudions la résolution de l'équation (B.11),

l'autre étant similaire en tout point. Nous cherchons alors à estimer les paramètres $\{a, b, c\}$. Ce problème correspond à la résolution du système de n équations linéaires à 3 inconnues suivant :

$$\begin{cases} p_{1,1}^{t+1} & = a.p_{1,1}^{t} - b.p_{1,2}^{t} - c \\ p_{2,1}^{t+1} & = a.p_{2,1}^{t} - b.p_{2,2}^{t} - c \\ & \vdots \\ p_{n,1}^{t+1} & = a.p_{n,1}^{t} - b.p_{n,2}^{t} - c \end{cases} \tag{B.14}$$

où, sous forme matricielle

$$\begin{pmatrix} p_{1,1}^{t+1} \\ p_{2,1}^{t+1} \\ \vdots \\ p_{n,1}^{t+1} \end{pmatrix} = \begin{pmatrix} p_{1,1}^{t} & p_{1,2}^{t} & 1 \\ p_{2,1}^{t} & p_{2,2}^{t} & 1 \\ \vdots & \vdots & \vdots \\ p_{n,1}^{t} & p_{n,2}^{t} & 1 \end{pmatrix} \begin{pmatrix} a \\ b \\ c \end{pmatrix} \tag{B.15}$$

$$Y = XA \tag{B.16}$$

L'équation (B.11) est un cas particulier de régression linéaire où nous cherchons à minimiser le carré de l'erreur. Cet estimateur est usuellement appelé « estimateur des moindres carrés ordinaires ». La solution de cet estimateur est très classique et est obtenue en une étape :

$$\widehat{A} = X^{+}Y = (X^{T}X)^{-1}X^{T}Y \tag{B.17}$$

où X^T représente la matrice transposée de X et X^{-1} l'inverse de la matrice X. X^+ est la pseudo-inverse de la matrice X. L'avantage de cette méthode est que les paramètres optimaux sont obtenus en une seule opération. En effet, il est ici nul besoin d'utiliser des algorithmes d'optimisation comme le simplexe [Dan51, Nas00, LRWW98] ou le recuit simulé [KGV83]. Pour cette raison, cette méthode est particulièrement utilisée dans l'industrie car extrêmement rapide en terme de coût calcul. Cependant, cette solution peut conduire à une estimation aberrante des paramètres si la matrice X^TX est mal conditionnée [1]. La figure B.1 illustre ce phénomène. La matrice de mouvement estimée entre les images I^1 et I^2 est aberrante car la matrice X^TX a un conditionnement de 2.10^8. Le suivi est par conséquent incorrect. Si maintenant la fonction de coût valeur absolue est minimisée par la méthode du simplexe (voir figure B.2), nous remarquons que le suivi est visuellement beaucoup plus précis. L'estimation des paramètres est cependant beaucoup plus lent à calculer. Ce problème de conditionnement est très classique. Il faut donc choisir entre précision et rapidité.

Dans le chapitre 4, nous avons montré que l'ajout d'une pondération pouvait améliorer l'estimation des paramètres du mouvement. L'estimateur

1. Le conditionnement d'un matrice A est donné par $K(A) = \|A\|\|A^{-1}\|$.

FIGURE B.1 – *Suivi d'un visage réalisé sur* 13 *images de la séquence Crew au format SD (SD=*704 $\times$ 576 *pixels). Les images présentées correspondent, de gauche à droite et de haut en bas, aux images* I^1, I^5, I^9, I^{13}. *Le contour* C^1 *édité manuellement et les contours* C^5, C^9, *et* C^{13} *sont superposés sur leur image respective. La minimisation du M-estimateur est réalisée par l'estimateur des moindres carrés ordinaires. La matrice de mouvement a clairement été mal estimée. Cette erreur est liée au mauvais conditionnement de la matrice* $X^T X$ (2.10^8 *entre les images* I^1 *et* I^5). *Le comportement de l'estimateur est donc incertain.*

des moindres carrés ordinaires peut prendre en compte ces pondérations. Considérons que nous disposons d'une pondération w_i pour chaque couple de point $\{p_i^t, p_i^{t+1}\}$ ce qui est équivalent à une pondération par ligne du système linéaire. Nous définissons la matrice W comme suit :

$$W = \begin{pmatrix} w_1 & 0 & \cdots & 0 \\ 0 & w_2 & \cdots & 0 \\ \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & \cdots & w_n \end{pmatrix} \tag{B.18}$$

Les paramètres estimés $\{\widehat{a}, \widehat{b}, \widehat{c}\}$ sont obtenus par la formule

$$\widehat{A} = \begin{pmatrix} \widehat{a} \\ \widehat{b} \\ \widehat{c} \end{pmatrix} = (X^T W^{-1} X)^{-1} X^T W^{-1} Y. \tag{B.19}$$

FIGURE B.2 – *Suivi d'un visage réalisé sur* 13 *images de la séquence Crew au format SD (SD=*704 $\times$ 576 *pixels). Les images présentées correspondent, de gauche à droite et de haut en bas, aux images* I^1, I^5, I^9, I^{13}. *Le contour* C^1 *édité manuellement et les contours* C^5, C^9, *et* C^{13} *sont superposés sur leur image respective. La minimisation du M-estimateur est réalisée par la méthode du simplexe et la fonction coût utilisée est la fonction valeur absolue. Le suivi est correct sur toute la séquence.*

Une fois encore, cette méthode d'estimation de paramètres est très rapide mais la qualité de l'estimation des paramètres obtenue est incertaine.

- ANNEXE C -

CODE SOURCE CUDA

Les sources de l'implémentation CUDA de la recherche des kPPV par la méthode exhaustive sont ou seront disponibles en téléchargement sur Internet. Une requête avec les mots-clés « knn CUDA Garcia » dans un moteur de recherche devrait permettre de trouver la page consacrée à nos travaux. Le lecteur est invité à consulter régulièrement la page pour profiter des éventuelles mise à jour.

- ANNEXE D -

LA POST-PRODUCTION CINÉMATOGRAPHIQUE

§ D.1 CINÉMATOGRAPHIE ET PRODUCTION

Le cinéma (ou cinématographe) est un art du spectacle séculaire considéré aujourd'hui comme étant le septième art [1]. Pour comprendre ce qu'était le cinéma à sa création, il convient de donner l'origine du terme *cinématographe*. Ce terme vient du grec *kinêma* signifiant *mouvement* et de *graphein* signifiant *écrire*. Le cinéma consiste donc à raconter une histoire avec des images en mouvement.

Historiquement, le français Léon Bouly déposa le 12 février 1892 un brevet [2] pour sa machine « réversible de photographie et d'optique pour l'analyse et la synthèse des mouvements », machine qui sera plus simplement nommée « cinématographe » en 1893. En 1894, Léon Bouly perd le nom déposé « cinématographe » faute de paiement des redevances de ses brevets. Le nom de « cinématographe », alors devenu disponible, est breveté par les frères Lumière. Les historiens constatent aujourd'hui que, dans son brevet, Léon Bouly fut, avant les Lumière, le véritable inventeur du terme cinématographe. La première projection cinématographique publique et payante, organisée par Auguste et Louis Lumière, eut lieu le 28 décembre 1895 au Salon Indien dans les sous-sols du Grand Café à Paris. Quatre-vingt sept ans plus tard, les premiers mélanges d'images filmées et d'images numériques apparaissent avec la sortie en 1982 de « Tron » réalisé par Steven Lisberger. Ce fut alors le début des effets spéciaux numériques au cinéma.

La réalisation d'un film est un procédé long et compliqué. Les principales étapes qui la composent sont :

1. Les six arts sont : architecture, sculpture, peinture, littérature, musique et danse.
2. Le brevet déposé par Léon Bouly le 12 février 1892 porte le n°219 350

- Développement : écriture d'un scénario et préparation d'un projet de film.
- Pré-production : le film est imaginé et préparé, les acteurs sont choisis et engagés, les locaux sont loués. Des illustrateurs préparent un storyboard permettant de décrire les plans qu'il faudra tourner.
- Production : les différentes scènes sont filmées ce qui donnent les éléments bruts du film.
- Post-production : le film est édité. Les musiques sont composées et enregistrées. Les effets sonores sont conçus et enregistrés. Les effets spéciaux visuels sont ajoutés numériquement au film. Le film est alors complet.
- Distribution : le film est projeté à d'éventuels acheteurs (distributeurs), puis il est choisi par un distributeur. Ce dernier l'envoie alors aux salles de projection.

Le succès d'un film dépend d'un ensemble de critères tels que la qualité du scénario, le jeu de scène des acteurs, le réalisme des effets spéciaux, *etc.* Chaque partie d'un film doit être réalisée avec une grande attention, l'ensemble des choix appartenant au réalisateur. La partie qui nous intéresse dans ce manuscrit est la post-production. Cette étape utilise de nombreux outils apportés par les recherches menées dans les domaines du traitement d'images et de la vision par ordinateur. Dans la suite de ce chapitre, seule la partie post-production visuelle, c'est-à-dire l'ensemble des effets relatifs aux images et non au son, est détaillée.

§ D.2 POST-PRODUCTION VISUELLE

La production permet d'acquérir sur bobines l'ensemble des scènes qui vont composer le film. La première étape de la post-production consiste à numériser (ou *scanner*) ces bobines de film pour permettre un traitement numérique. L'étape suivante consiste à rectifier les images acquises. En effet, le matériel (caméra, objectif, bobine, *etc.*) utilisé pour filmer une scène donnée a un impact sur les images qu'il est nécessaire de corriger avant tout traitement. Ainsi, les images sont étalonnées (correction des couleurs) et la distorsion (déformation de la géométrie des images induite par les lentilles des objectifs) est corrigée ou, tout du moins, estimée pour être prise en compte dans la suite des étapes de post-production.
Débute alors la partie principale de la post-production. Un exemple très fréquent de travail effectué en post-production consiste en l'insertion d'objets virtuels, c'est-à-dire modélisés par ordinateur, dans une scène réelle filmée en studio ou en décor naturel. Les différentes étapes alors rencontrées sont les suivantes :

1. Matchmoving

2. Rotoscopie
3. Modélisation
4. Viewpainting
5. Animation
6. Illumination
7. Compositing

Certaines étapes peuvent être réalisées en parallèle, et d'autres sont nécessairement réalisées en série. La durée totale allouée à la post-production a un impact direct sur son coût. Dans la suite, nous présentons brièvement chacune de ces étapes et nous indiquons sa durée approximative. Le temps imparti à la post-production pour un film donné dépend du nombre et de la complexité des traitements et des effets spéciaux à effectuer. Il n'est donc pas possible d'évaluer une durée absolue. Par conséquent, nous indiquons des durées relatives mesurées en *unité de temps*. Ces durées sont bien entendu des moyennes. Elles nous ont été fournies par Ronald Mallet, ingénieur R&D à ILM [3].

D.2.1 Matchmoving

Le *matchmoving* est une technique utilisée en post-production permettant d'intégrer des objets virtuels dans une scène filmée. Pour que l'intégration soit réussie, les objets doivent avoir, à chaque image du film, la bonne taille, la bonne position, la bonne orientation et le bon éclairage. Pour parvenir à ce résultat, la scène filmée doit être analysée de manière à retrouver le mouvement de la caméra et des différents objets de la scène au cours du temps.
Plus simplement, le matchmoving permet de définir le monde 3D dans lequel toute la scène va se dérouler. 2 unités de temps sont généralement imparties à cette tâche.
Les étapes rencontrées lors du matchmoving sont les suivantes :

1. Tracking : Des points caractéristiques (typiquement des coins) sont identifiés et suivis sur toute la séquence vidéo à traiter. Ces points doivent permettre d'être localisés très précisément à chaque image, à condition bien évidemment qu'ils soient dans le champ de la caméra et non cachés par d'éventuels autres objets. Chaque point définit une trajectoire temporelle, trajectoire communément appelée *track*.
2. Calibration : La seconde étape est la résolution du mouvement 3D de la caméra à partir des différents tracks construits. Cette résolution est rendue possible par l'utilisation de la géométrie projective.

3. Industrial Light & Magic (ou ILM) est une société d'effets spéciaux de cinéma américaine (San Francisco, Californie), filiale de Lucasfilm Ltd. http ://www.ilm.com

3. Projection du nuage de points : La position 3D réelle de chaque point est alors identifiée en inversant la projection calculée à l'étape de calibration. Il résulte de cela un ensemble de points 3D appelé *nuage de points*.
4. Reconstruction : Le nuage de points laisse apparaître des structures et des objets de la scène réelle servant de base à la dernière étape appelée reconstruction. Grâce à différents points du nuage, le sol et différents objets de la scène réelle sont modélisés et placés. Pour le sol, il s'agit souvent d'un plan 2D. La modélisation est souvent assez grossière (sans détails) le but étant d'avoir des repères spatiaux précis.

À ce stade, nous disposons d'un univers 3D reconstruit correspondant à l'univers réel de la scène filmée. L'insertion d'un objet virtuel (personnage, animal, véhicule, *etc.*) dans l'univers 3D s'intègre alors parfaitement à la vidéo d'origine si l'on ne considère que la position, l'orientation, la taille, et le mouvement de l'objet considéré. L'animation de cet objet, la création de sa texture, et son éclairage sont des étapes réalisées ultérieurement.
Les premiers exemples de matchmoving sont apparus dans le film *Jurassic Park* (Steven Spielberg, 1993). Dans ce film, des acteurs réels et des dinosaures se côtoient dans des décors naturels. Le succès du film est, en partie, lié à la qualité des effets spéciaux réalisés créant une illusion inédite jusqu'alors au cinéma. Cet exploit technique a été réalisé de la manière suivante. Le suivi de points caractéristiques était réalisé à l'aide de balles de tennis. La couleur et la taille de ses balles permettait de les suivre très précisément au cours du temps. Les tracks ainsi créés ont servi de base au procédé de matchmoving. Les dinosaures modélisés ont alors pu être intégrés aux différentes scènes du film. Enfin, les balles de tennis sont effacées par ordinateur.

D.2.2 Rotoscopie

Le terme *rotoscopie* désigne aujourd'hui la technique permettant de tracer le contour d'un objet dans le but d'en extraire une silhouette numérique. Cette silhouette est alors utilisée pour diverses applications telles que supprimer un objet d'une scène (élément anachronique filmé involontairement), placer des personnages sur un autre fond, ou encore dupliquer des personnages pour créer une foule réaliste avec peu de figurants. La rotoscopie est particulièrement fréquente lors de l'incrustation d'objets virtuels. L'objet est plaqué sur la vidéo d'origine cachant souvent des parties situées en avant plan par rapport à la caméra. Les objets d'avant plan sont alors identifiés et détourés manuellement pour ne pas être cachés par l'objet virtuellement ajouté à la scène lors de l'étape d'incrustation.
La rotoscopie est vraisemblablement le plus ancien effet spécial utilisé dans

l'univers cinématographique. À ses débuts, la rotoscopie consistait à dessiner, image par image et à l'aide d'un rotoscope, un objet en mouvement en se basant sur une vidéo réelle originale. Le rotoscope était une machine qui projetait une vidéo sur un chevalet transparent dépoli. Un dessinateur déposait alors sa feuille de papier sur le chevalet et dessinait le personnage ou l'objet dont il voulait copier les contours (voir figure D.1). En dessinant image par image les contours d'un objet, le mouvement du personnage dessiné apparaissait alors très naturel. Cette technique, inventée par Max Fleischer aux alentours de 1915, fut utilisée en premier dans *Koko le Clown* en 1915 et, par la suite, dans de très nombreuses productions cinématographiques comme par exemple *Betty Boop*, *Superman*, *Blanche Neige et les sept nains*, etc. Une utilisation très connue de la rotoscopie peut être vue dans les films de la série *Star Wars* de Georges Lucas. Les sabres lasers sont en réalité de simples bâtons de bois suivis image par image et sur lesquels un effet de lumière ou d'incandescence est rajouté.
La rotoscopie est, aujourd'hui, utilisée dans les cas où des techniques telles que celles fondées sur l'utilisation du fond bleu ne sont pas assez précises. Cette précision n'est pourtant pas sans conséquence. Le caractère manuel de cette approche induit un temps de traitement considérable (8 unités de temps). Certains outils informatiques permettent de diminuer le temps de détourage des objets en permettant de dupliquer et de modifier facilement le contour des objets. Cependant, le temps utile à la rotoscopie est toujours très important. Un enjeu majeur pour la post-production est le développement d'outils automatisant au maximum ce travail. Une complète automatisation n'est cependant pas envisageable. Deux problèmes se posent : premièrement, la sélection des objets requiert une analyse sémantique de la scène. Deuxièmement, la segmentation d'objets fins, flous, ou transparents est, à l'heure actuelle, impossible informatiquement parlant.

D.2.3 Création et animation d'objets 3D

La création d'objets 3D représente une des parties les plus consommatrices en temps. Elle se compose des étapes de modélisation, de viewpainting, d'animation et d'illumination. Les objets créés sont généralement très variés. Ils peuvent être de simples chaises ou encore des personnages articulés extrêmement complexes tels que des robots. Du fait de sa complexité, cette étape est généralement répartie sur plusieurs équipes, chacune étant responsable d'un travail particulier. Les créateurs sont généralement des artistes. En effet, seule la vision du monde propre aux artistes permet de créer des objets réalistes, de par leur forme ou leur mouvement.
Les étapes de création des objets 3D sont les suivantes.

- Modélisation : 20 unités de temps. Cette partie est aussi simple à définir que complexe à réaliser. En infographie, la modélisation consiste à créer

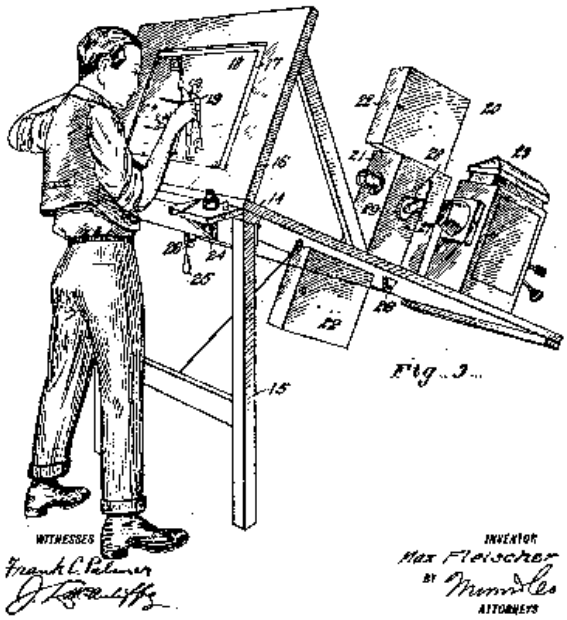


FIGURE D.1 – *Rotoscope original inventé par Max Fleischer. Une caméra projette sur un chevalet transparent dépoli une scène réelle filmée où un personnage réel (voir plusieurs) est en mouvement. Le dessinateur, ayant disposé sa feuille sur le chevalet, ajuste la position du personnage dessiné pour la calquer sur la position du personnage réel. Les mouvements du personnage dessiné semblent, de fait, totalement naturels.*

des modèles tridimensionnels dans un logiciel spécialisé. Plus exactement, lors de la modélisation, seule la surface (on parle également de « peau ») des objets va être créée. Le temps de création d'un modèle 3D dépend naturellement de sa complexité. La modélisation d'un objet même complexe débute en général à partir d'une forme simple, telle qu'un cube. Un peu à la manière d'un sculpteur, l'artiste façonne l'objet en ajoutant, au fur et à mesure, des détails jusqu'à obtenir la forme choisie. Cette façon de procéder permet, en général, de minimiser le nombre de triangles nécessaires pour arriver à la qualité souhaitée. Plus le nombre de triangles est important, plus le temps de calcul des images est long. L'optimisation des modèles est donc un facteur essentiel dans l'étape de modélisation.

- Viewpainting : 10 unités de temps. Un modèle créé lors de l'étape précédente, est, comme nous l'avons précisé, une surface, une peau. Plus importantes encore que la forme, la couleur et la texture de l'objet sur toute sa surface permettent de donner vie aux objets. Pour créer cette texture, il est assez courant de « déplier » le modèle 3D pour le plaquer sur une image 2D. L'artiste dessine, ou peint, sur cette image 2D. Cette image constitue la texture. Une fois la texture réalisée, celle-ci est plaquée sur le modèle 3D. Précisons qu'une texture peut également être définie de façon analytique par un algorithme. On parle alors de texture *procédurale*.
Enfin, il est important d'évoquer une autre catégorie de texture. Il est assez courant de plaquer sur le modèle 3D une texture dite de relief. On parle de *placage de rugosité*, de *texture par perturbation de la lumière*, ou encore de *bump mapping* en anglais. Nous verrons dans la suite qu'une autre étape importante est l'éclairage des modèles 3D. Imaginons que le modèle à créer est un visage humain. Un tel modèle est photo-réaliste si et seulement si des détails tels que les imperfections de la peau sont visibles. Créer un modèle si complexe est extrêmement long voir impossible. De même, le temps de rendu d'un tel modèle serait beaucoup trop long. La texture de relief permet de dévier localement les rayons lumineux sur le modèle 3D.

- Animation : 15 unités de temps. L'animation d'un modèle 3D consiste tout simplement à donner vie au modèle, à l'animer, à le faire bouger. L'animation doit être réalisée dans l'univers 3D créé lors du matchmoving. Pour des objets fixes, comme par exemple des bâtiments, aucune animation n'est requise. Pour des modèles articulés tels que des personnages humains virtuels, l'animation devient très complexe. Le réalisme d'un modèle dépend directement de son animation. L'animation du modèle est faite via l'animation d'un squelette. Comme pour un être vivant, ce squelette est un ensemble d'articulations reliées par des liaisons rigides. Ce squelette, généralement placé à l'intérieur du modèle à animer, est attaché à différentes parties de ce modèle. Il est très courant de recourir à la *capture du mouvement* (en anglais, *motion capture*) permettant d'enregistrer les mouvements d'un

élément réel comme par exemple les mouvements d'un corps humain. Ces mouvements réels enregistrés sont appliqués au squelette, puis répercutés sur le modèle 3D.

- Illumination : 25 unités de temps. L'ultime étape à l'intégration d'un modèle 3D dans la scène filmée est l'illumination. Sans illumination, le modèle ne paraît pas réel. Cette étape est la plus longue de toutes car il faut placer de très nombreuses sources lumineuses à des endroits très précis pour simuler le réel. Toutes ces sources se combinent et éclairent le modèle lors du rendu de la scène. Rappelons que la lumière est déviée en surface de l'objet par la texture de rugosité ajoutant visuellement de nombreux détails contribuant au photo-réalisme du modèle.

Un exemple de rendu photo-réaliste combinant l'ensemble des étapes précédemment citées est illustré sur la figure D.2.

Lors de la phase de rendu, les modèles créés sont intégrés à la vidéo réelle initialement filmée. Systématiquement, la scène ainsi réalisée est visionnée et corrigée : les modèles 3D sont affinés ; les mouvements sont ajustés ; des lumières sont modifiées, supprimées, ou ajoutées. Puis, le rendu de la scène est à nouveau réalisé, puis re-visionné, et ainsi de suite.

FIGURE D.2 – *Exemple d'un univers 3D photo-réaliste. Image créée par Gilles Tran.*

D.2.4 Compositing

Le compositing est l'ultime étape de la post-production. Chaque scène produite correspond à un bout de film. L'étape de compositing consiste à assembler ces différents bouts pour créer la version finale du film. L'organisation de l'enchaînement des différentes scènes est essentielle à la bonne compréhension du film par les spectateurs. La qualité et le succès d'un film dépendent grandement de choix réalisés lors du compositing. Lors de cette dernière étape, les bruitages, musiques et différentes voix sont ajoutés. Cela ne rentre pas dans la post-production visuelle mais est indissociable de l'étape de compositing.

- ANNEXE E -

NVIDIA CUDA ZONE

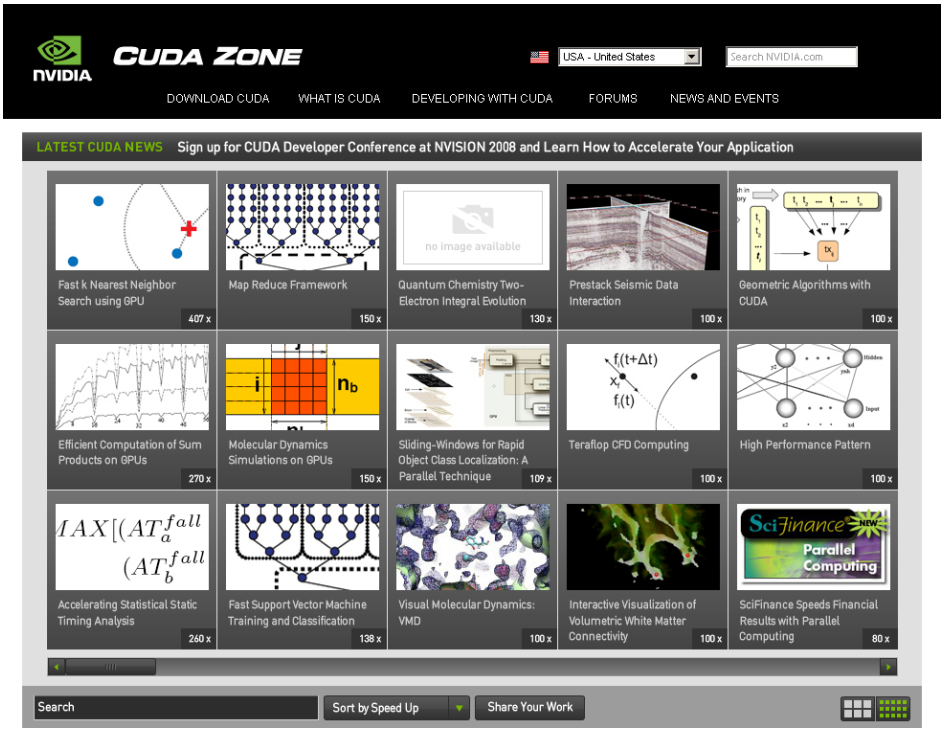


FIGURE E.1 – *Capture d'écran du site Internet NVIDIA CUDA Zone[NVI] réalisée le 24 juillet 2008. Ce site Internet référence les travaux utilisant l'API NVIDIA CUDA. Selon ce site, notre travail engendre le gain (en terme de temps de calcul) le plus important par rapport à l'ensemble des travaux référencés.*

TABLE DES FIGURES

1.1 Traitement d'images et applications 3
1.2 Application du suivi d'objets à la préservation de l'anonymat 4

2.1 Évolution d'un contour actif vers l'objet d'intérêt 14

3.1 Appariement d'images par appariement de points d'intérêt . 18
3.2 Exemples de saillance visuelle 19
3.3 Détecteur de Harris 23
3.4 Détecteur LoG 26
3.5 26-voisinage 26
3.6 Détecteur de Harris-Laplace et détecteur LoG 28
3.7 Vecteur de description : voisinage de taille 3×3 pixels . . . 29
3.8 Vecteur de description : distribution de l'intensité 31
3.9 Problème lié à l'utilisation de distributions pour la description locale 32
3.10 Vecteur de description : SIFT 33
3.11 Vecteur de description : SIFT 33

4.1 Appariement de points d'intérêt 40
4.2 Appariement de points d'intérêt par validation croisée . . . 41
4.3 Tracé des principales fonctions de régularisation. Pour l'approximation de la variation totale, ϵ est fixé à $\epsilon = 0.2$. 44
4.4 Suivi sur la vidéo *Carmap* 46
4.5 Pixels bien classés et pixels mal classés 47
4.6 Suivi sur la vidéo *Arménie* 48
4.7 Illustration de trajectoires temporelles de points d'intérêt . . 50
4.8 Trajectoires temporelles des points d'intérêt 51
4.9 Évolution de l'erreur de suivi (pourcentage de pixels mal-classés) en fonction de la taille du GOP pour la séquence Carmap 54
4.10 Résultat de suivi pour la vidéo *Carmap* 55

4.11 Évolution de l'erreur de suivi (pourcentage de pixels mal-classés) en fonction de la taille du GOP pour la séquence Arménie . . . 56
4.12 Résultat de suivi pour la vidéo *Arménie* . . . 57
4.13 Illustration de trajectoires temporelles de points d'intérêt . . 58
4.14 Estimation du mouvement du contour à partir du mouvement des points d'intérêt . . . 61
4.15 Évolution de l'erreur de suivi en fonction de la taille du GOP sur la séquence Carmap . . . 63
4.16 Résultat de suivi pour la vidéo Carmap . . . 64
4.17 Évolution de l'erreur de suivi en fonction de la taille du GOP sur la séquence Arménie. . . . 65
4.18 Résultat de suivi pour la vidéo Arménie . . . 66

5.1 Contour C^1 sur l'image I^1 pour les vidéos V_{tex} et V_{hom} . . . 72
5.2 Critère de comparaison SAD sur la fenêtre de recherche . . . 74
5.3 Profils 2D du critère SAD . . . 74
5.4 Profils 2D du critère SAD avec masquage des pixels du fond 75
5.5 Profils 2D du critère SAD avec masquage des pixels du fond 76
5.6 Profils 2D du critère SAD avec masquage partiel des pixels du fond . . . 77
5.7 Distribution du résiduel . . . 79
5.8 Comparaison des profils 2D des critères SAD et entropie . . 81
5.9 Résultat de suivi pour le critère SAD et pour la séquence V_{tex} 82
5.10 Résultat de suivi pour le critère SAD et pour la séquence V_{tex} 83
5.11 Résultat de suivi pour le critère SAD et pour la séquence V_{tex} 84
5.12 Résultat de suivi pour le critère entropie et pour la séquence V_{tex} . . . 85
5.13 Résultat de suivi pour le critère SAD et pour la séquence Carmap . . . 87
5.14 Résultat de suivi pour le critère entropie et pour la séquence Carmap . . . 88
5.15 Résultat de suivi pour le critère SAD et pour la séquence Soccer 90
5.16 Résultat de suivi pour le critère entropie et pour la séquence Soccer . . . 91
5.17 Résultat de suivi pour le critère entropie et pour la séquence Ice . . . 92

6.1 Suivi de boîte englobante . . . 98
6.2 Deux images différentes ayant même distribution . . . 99
6.3 Utilisation d'informations couleur et d'informations géométriques pour la description d'une image . . . 100

7.1 Recherche des kPPV dans un espace Euclidien 2D . . . 104

7.2 Construction d'un kd-tree 106

8.1 Évolution de la puissance des CPUs et GPUs 112
8.2 Architecture d'un GPU 113
8.3 Organisation d'un programme CUDA 116
8.4 Exemple d'accès coalescents à la mémoire 118
8.5 Exemple d'accès non coalescents à la mémoire 119
8.6 Calcul des distances en CUDA 122
8.7 Évolution du temps de calcul en fonction du paramètre k pour deux algorithmes de tri 124
8.8 Évolution de k_0 en fonction du nombre de points 124

9.1 Influence de la dimension sur la recherche des kPPV 131
9.2 Influence du nombre de points sur la recherche des kPPV . . 133
9.3 Influence du paramètre k sur la recherche des kPPV 134
9.4 Résultats de suivi sur la séquence Crew 139
9.5 Évolution des paramètres de la vérité terrain en fonction de l'indice de l'image pour la séquence Crew 141
9.6 Erreur sur les paramètres pour la séquence Crew 142
9.7 Erreur du suivi pour la séquence Crew 143
9.8 Évolution spatio-temporelle de la boîte englobante sur la séquence Crew 144
9.9 Résultat de suivi sur la séquence Crew 145
9.10 Évolution des paramètres sur la séquence Crew 146
9.11 Évolution des paramètres de la vérité terrain en fonction de l'indice de l'image pour la séquence Poltergay 148
9.12 Erreur sur les paramètres pour la séquence Poltergay 150
9.13 Erreur du suivi pour la séquence Poltergay 151
9.14 Évolution spatio-temporelle de la boîte englobante sur la séquence Crew 152
9.15 Résultat de suivi sur la séquence Poltergay 153
9.16 Évolution des paramètres sur la séquence Poltergay 154

10.1 CUBLAS : influence de la dimension 161
10.2 CUBLAS : influence du nombre de points 163

A.1 Estimation de densité de probabilité par une méthode d'histogramme 177
A.2 Estimation de densité de probabilité par une méthode à noyau Gaussien 178
A.3 Influence du choix du noyau sur l'estimation de densité de probabilité 179
A.4 Illustration de la recherche des k-plus proches voisins 180
A.5 Évolution de l'entropie en fonction de la taille des échantillons 181

A.6 Évolution de la variance de l'entropie en fonction de la taille des échantillons 182

B.1 Estimation du mouvement par une méthode des moindres carrés 186
B.2 Estimation du mouvement par la méthode du simplexe . . . 187

D.1 Rotoscope de Max Fleischer 196
D.2 Exemple d'un univers 3D photo-réaliste. 198

E.1 Site Internet *NVIDIA CUDA Zone* 201

LISTE DES TABLEAUX

3.1 Interprétation de la valeur du coefficient de corrélation . . . 35
3.2 Distances classiques 36

4.1 Principales fonctions de régularisation. 44

9.1 Tableau récapitulatif du temps de calcul pour différentes méthodes de recherche des kPPV 129
9.2 Influence de l'implémentation sur la répartition du temps de calcul 135
9.3 Influence de la dimension sur la répartition du temps de calcul 136
9.4 Influence du nombre de points sur la répartition du temps de calcul 137
9.5 Influence du paramètre k sur la répartition du temps de calcul 137
9.6 Composantes utilisées pour la description d'un objet 140
9.7 Durée du processus de suivi sur la séquence Crew en fonction du descripteur et de l'implémentation 147
9.8 Durée du processus de suivi sur la séquence Poltergay en fonction du descripteur et de l'implémentation 149

BIBLIOGRAPHIE

[AA01] A. Ali and J. K. Aggarwal. Segmentation and recognition of continuous human activity. In *IEEE Workshop on Detection and Recognition of Events in Video*, pages 28–35, 2001.

[ABFJB03] G. Aubert, M. Barlaud, O. Faugeras, and S. Jehan-Besson. Image segmentation using active contours : Calculus of variations or shape gradients ? *SIAM Applied Mathematics*, 63 :2128–2154, 2003.

[ADPB08] S. Anthoine, E. Debreuve, P. Piro, and M. Barlaud. Using neighborhood distributions of wavelet coefficients for on-the-fly, multiscale-based image retrieval. In *Workshop on Image Analysis for Multimedia Interactive Services*, Klagenfurt, Austria, May 2008.

[AI06] A. Andoni and P. Indyk. Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions. *IEEE Symposium on Foundations of Computer Science*, 51(1) :459–468, 2006.

[AL76] I. A. Ahmad and P. E. Lin. A nonparametric estimation of the entropy for absolutely continuous distributions. *IEEE Transactions on Information Theory*, 22 :372–375, May 1976.

[AMN+98] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *Journal of the ACM*, 45(6) :891–923, 1998.

[BA96] M. J. Black and P. Anandan. The robust estimation of multiple motions : parametric and piecewise-smooth flow fields. *Computer Vision and Image Understanding*, 63(1) :75–104, 1996.

[BDB07] S. Boltz, E. Debreuve, and M. Barlaud. High-dimensional statistical distance for region-of-interest tracking : Application to combining a soft geometric constraint with radiometry. In *IEEE International Conference on Computer Vision and Pattern Recognition*, Minneapolis, USA, 2007.

[Ben75] J. L. Bentley. Multidimensional binary search trees used for associative searching. *Communications of the ACM*, 18(9) :509–517, 1975.

[BHD+08] S. Boltz, A. Herbulot, E. Debreuve, M. Barlaud, and G. Aubert. Motion and appearance nonparametric joint entropy for video segmentation. *IEEE Transactions Pattern Analysis Machine Intelligence*, 80 :242–259, 2008.

[Bir98] S. Birchfield. Elliptical head tracking using intensity gradients and color histograms. In *IEEE International Conference on Computer Vision and Pattern Recognition*, Washington, DC, USA, 1998.

[BK06] F. Bernhard and R. Keriven. Spiking neurons on gpus. In *International Conference on Computational Science. Workshop General purpose computation on graphics hardware (GPGPU) : Methods, algorithms and applications*, Readings, UK, May 2006.

[Bol08] S. Boltz. *A statistical framework in variational methods of image and video processing problems with high dimensions*. PhD thesis, Université de Nice - Sophia Antipolis, 2008.

[BWD+06] S. Boltz, E. Wolsztynski, E. Debreuve, E. Thierry, M. Barlaud, and L. Pronzato. A minimum-entropy procedure for robust motion estimation. In *IEEE International Conference on Image Processing*, pages 1249–1252, Atlanta, USA, 2006.

[CAI+08] O. Commowick, V. Arsigny, A. Isambert, J. Costa, F. Dhermain, F. Bidault, P.-Y. Bondiau, N. Ayache, and G. Malandain. An efficient locally affine framework for the smooth registration of anatomical structures. *Medical Image Analysis*, 12(4) :427–441, 2008.

[Can86] J. Canny. A computational approach to edge detection. *IEEE Transactions Pattern Analysis Machine Intelligence*, 8(6) :679–698, 1986.

[CBFAB97] P. Charbonnier, L. Blanc-Feraud, G. Aubert, and M. Barlaud. Deterministic edge-preserving regularization in computed imaging. *IEEE Transactions on Image Processing*, 6(2) :298–311, February 1997.

[CBP05] L. Carminati and J. Benois-Pineau. Gaussian mixture classification for moving object detection in video surveillance environment. In *IEEE International Conference on Image Processing*, 2005.

[CBPG03] L. Carminati, J. Benois-Pineau, and M. Gelgon. Human detection and tracking for video surveillance applications in a low-density environment. In *VCIP*, pages 48–60, 2003.

[CDBPD07] F. Chevalier, J.-P. Domenger, J. Benois-Pineau, and M. Delest. Retrieval of objects in video by similarity based on graph matching. *Pattern Recognition Letter*, 28(8) :939–949, jun 2007.

[CKS94] V. Caselles, R. Kimmel, and G. Sapiro. Geodesic active contours. Technical report, HP Labs, September 1994.

[CM99a] D. Comaniciu and P. Meer. Distribution free decomposition of multivariate data. *Pattern Analysis and Applications*, 2 :22–30, 1999.

[CM99b] D. Comaniciu and P. Meer. Mean shift analysis and applications. In *IEEE International Conference on Computer Vision*, pages 1197–1203, Kerkyra, Greece, 1999.

[CM02] D. Comaniciu and P. Meer. Mean shift : A robust approach toward feature space analysis. *IEEE Transactions Pattern Analysis Machine Intelligence*, 24(5) :603–619, 2002.

[Coh88] J. Cohen. *Statistical power analysis for the behavioral sciences*. Lawrence Erlbaum, Hillsdale, NJ, 1988.

[CP84] J. L. Crowley and A. C. Parker. A representation for shape based on peaks and ridges in the difference of low-pass transform. *IEEE Transactions Pattern Analysis Machine Intelligence*, 1(2) :156–170, March 1984.

[CPMG+08] A. Cachia, M-L. Paillère-Martinot, A. Galinowski, D. Januel, R. De Beaurepaire, F. Bellivier, E. Artiges, T. Gallarda, J. Andoh, D. Bartrés-Faz, E. Duchesnay, D. Rivière, Y. Cointepas, M. Plaze, J.-F. Mangin, and J.-L. Martinot. Cortical folding abnormalities in schizophrenia patients with resistant auditory hallucinations. *Neuroimage*, 39(3) :927–935, 2008.

[CRD07] D. Cremers, M. Rousson, and R. Deriche. A review of statistical approaches to level set segmentation : Integrating color, texture, motion and shape. *IEEE Transactions Pattern Analysis Machine Intelligence*, 72(2) :195–215, 2007.

[CRM00] D. Comaniciu, V. Ramesh, and P. Meer. Real-time tracking of non-rigid objects using mean shift. In *IEEE International Conference on Computer Vision and Pattern Recognition*, pages 142–151, 2000.

[CRM03] D. Comaniciu, V. Ramesh, and P. Meer. Kernel-based object tracking. *IEEE Transactions Pattern Analysis Machine Intelligence*, 25(5) :564–575, 2003.

[CT91] T. Cover and J. Thomas. *Elements of Information Theory*. Wiley, New York, 1991.

[CV01] T. Chan and L. Vese. Active contours without edges. *IEEE Transactions on Image Processing*, 10(2) :266–277, February 2001.

[Dan51] G. B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. In T.C. Koopmans, editor, *Activity Analysis of Production and Allocation - Proceedings of a Conference*, volume 13 of *Cowles Commission Monograph*, pages 339–347. Wiley, New York, 1951.

[DCM06] C. Deroover, J Czyz, and B. Macq. Active contour attracted by a reference contour : a region-based approach. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, Toulouse, France, May 2006.

[DDBP06] M. Delests, A. Don, and J. Benois-Pineau. Dag based visual interfaces for navigation in index vidéo content. *Journal Multimedia Tools and Applications*, 31(1) :51–72, Oct 2006.

[Der87] R. Deriche. Using canny's criteria to derive an optimal edge detector recursively implemented. *IEEE Transactions Pattern Analysis Machine Intelligence*, 1 :167–187, 1987.

[Der93] R. Deriche. Recursively implementing the Gaussian and its derivatives. Technical Report 1893, INRIA, may 1993.

[DIIM04] M. Datar, N. Immorlica, P. Indyk, and V.S. Mirrokni. Locality-sensitive hashing scheme based on p-stable distributions. In *Symposium on Computational Geometry*, pages 253–262, New York, NY, USA, 2004. ACM Press.

[DKP05] N. Dixit, R. Keriven, and N. Paragios. Gpu-cuts and adaptive object extraction. Technical Report 05-07, CERTIS, March 2005.

[Dob80] W. Dobosiewicz. An efficient variation of bubble sort. *Information Processing Letters*, 11(1) :5–6, 1980.

[DT79] A. Dervieux and F. Thomasset. A finite element method for the simulation of Rayleigh-Taylor instability. *Lecture Notes in Mathematics*, 771 :145–159, 1979.

[EDD03] A. Elgammal, R. Duraiswami, and L.S. Davis. Probabilistic tracking in joint feature-spatial spaces. In *IEEE International Conference on Computer Vision and Pattern Recognition*, pages 781–788, 2003.

[Efr79] B. Efron. Bootstrap methods : Another look at the jackknife. *The Annals of Statistics*, 7 :1–26, 1979.

[Ext06] Extrait du film "Poltergay" dirigé par Eric Lavaine. Produit par Fabio Conversi, François Cornuau, et Vincent Roget, 2006.

[FH75] K. Fukunaga and L.D. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. *IEEE Transactions on Information Theory*, 21(1) :32–40, January 1975.

[GBA04] M. Gastaud, M. Barlaud, and G. Aubert. Combining shape prior and statistical features for active contour segmentation. *IEEE TCSVT*, 14 :726–734, 2004.

[GBDB06] V. Garcia, S. Boltz, E. Debreuve, and M. Barlaud. Contour tracking for rotoscoping based on trajectories of feature points. In *ECCV Workshop on Statistical Methods in Multi-Image and Video Processing (SMVP)*, Graz, Austria, May 2006.

[GBDB07] V. Garcia, Sylvain Boltz, E. Debreuve, and M. Barlaud. Outer-layer based tracking using entropy as a similarity measure. In *IEEE International Conference on Image Processing*, San Antonio, Texas, USA, September 2007.

[GDB06] V. Garcia, E. Debreuve, and M. Barlaud. A contour tracking algorithm for rotoscopy. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, Toulouse, France, May 2006.

[GDB07a] V. Garcia, E. Debreuve, and M. Barlaud. Méthode de suivi d'objets basée sur des trajectoires temporelles de points d'intérêt. In *Colloque GRETSI sur le Traitement du Signal et des Images*, Troyes, France, September 2007.

[GDB07b] V. Garcia, E. Debreuve, and M. Barlaud. Region-of-interest tracking based on keypoint trajectories on a group of pictures. In *IEEE International Workshop on Content-Based Multimedia Indexing*, Bordeaux, France, June 2007.

[GDB07c] V. Garcia, E. Debreuve, and M. Barlaud. Tracking based on local motion estimation of spatio-temporally weighted salient points. In *International Workshop on Image Analysis for Multimedia Interactive Services*, Santorini, Greece, June 2007.

[GDB08] V. Garcia, E. Debreuve, and M. Barlaud. Fast k nearest neighbor search using gpu. In *CVPR Workshop on Computer Vision on GPU*, Anchorage, Alaska, USA, June 2008.

[Gha90] M. Ghanbari. The cross-search algorithm for motion estimation. *IEEE Transactions on Communications*, 38 :950–953, 1990.

[GIM99] A. Gionis, P. Indyk, and R. Motwani. Similarity search in high dimensions via hashing. In *International Conference on Very Large Data Bases*, pages 518–529, 1999.

[GLMI05] M.N. Goria, N.N. Leonenko, V.V. Mergel, and P.L. Novi Inverardi. A new class of random vector entropy estimators and its applications in testing statistical hypotheses. *Journal of Nonparametric Statistics*, 17(3) :277–297, 2005.

[GM95] C. Gourieroux and A. Monfort. *Statistics and Econometric Models*, volume 1. Cambridge University Press, 1995.

[Har75] J. A. Hardigan. *Clustering Algorithms*. John Wiley & Sons, New York, 1975.

[Hoa61] C. A. R. Hoare. Algorithm 64 : Quicksort. *Communications of the ACM*, 4(7) :321, 1961.

[HP77] S.L. Horowitz and T. Pavlidis. Picture segmentation by a directed split and merge procedure. In *Computer Methods in Images Analysis*, 1977.

[HS88] C. Harris and M. Stephens. A combined corner and edge detector. In *Proceedings of The Fourth Alvey Vision Conference*, pages 147–151, Manchester, UK, 1988.

[HS92] R.M. Haralick and L.G. Shapiro. *Computer and Robot Vision*, volume 1. Addison-Wesley, 1992.

[Hub81] P. J. Huber. *Robust Statistics*. John Wiley and Sons, 1981.

[IM98] P. Indyk and R. Motwani. Approximate nearest neighbors : Towards removing the curse of dimensionality. In *Symposium on Theory of Computing*, pages 604–613, 1998.

[Ind04] P. Indyk. Nearest neighbors in high-dimensional spaces. In Jacob E. Goodman and Joseph O'Rourke, editors, *Handbook of Discrete and Computational Geometry, chapter 39*, chapter 39. CRC Press, 2004.

[JBBA03] S. Jehan-Besson, M. Barlaud, and G. Aubert. Dream2s : Deformable regions driven by an eulerian accurate minimization method for image and video segmentation. *IEEE Transactions Pattern Analysis Machine Intelligence*, 53 :45–70, June 2003.

[JN79] R. C. Jain and H. H. Nagel. On the analysis of accumulative difference pictures from image sequences of real world scenes. *IEEE Transactions Pattern Analysis Machine Intelligence*, 1(2) :206–213, April 1979.

[KCF+06] J. Kybic, M. Clerc, O. Faugeras, R. Keriven, and T. Papadopoulo. Generalized head models for meg/eeg : Bem beyond nested volumes. *Physics in Medecine and Biology*, 51 :1333–1346, Mar 2006.

[KFY+02] J. Kim, J. Fisher, A. Yezzi, M. Cetin, and A. Willsky. Nonparametric methods for image segmentation using information theory and curve evolution. In *IEEE International Conference on Image Processing*, pages 797–800, September 2002.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220(4598) :671–680, 1983.

[KKO+95] S. Kichenassamy, A. Kumar, P. Olver, A. Tannenbaum, and A. Yezzi. Gradient flows and geometric active contour models. In *IEEE International Conference on Computer Vision*, pages 810–815, 1995.

[KL51] S. Kullback and R. A. Leibler. On information and sufficiency. *The Annals of Mathematical Statistics*, 22(1) :79–86, 1951.

[KL87] L. Kozachenko and N. Leonenko. On statistical estimation of entropy of random vector. *Problems of Information Transmission*, 23(2) :95–101, 1987.

[KLH+81] T. Koga, K. Linuma, A. Hirano, Y. Iijima, and T. Ishiguro. Motion compensated interframe coding for video conferencing. In *IEEE National Telecommunication Conference*, New Orleans, LA, USA, 1981.

[Kul59] S. Kullback. *Information theory and statistics*. John Wiley and Sons, 1959.

[KWT87] M. Kass, A. Witkin, and D. Terzopoulos. Snakes : Active contour models. *IEEE Transactions Pattern Analysis Machine Intelligence*, 1(4) :321–331, 1987.

[Lin93] T. Lindeberg. Detecting salient blob-like image structures and their scales with a scale-space primal sketch : A method for focus-of-attention. *IEEE Transactions Pattern Analysis Machine Intelligence*, 11(3) :283–318, December 1993.

[Lin98] T. Lindeberg. Feature detection with automatic scale selection. *IEEE Transactions Pattern Analysis Machine Intelligence*, 30(2) :77–116, 1998.

[LO04] Y. Liu and S. Oraintara. Complexity comparison of fast block-matching motion estimation algorithms. *IEEE Transactions on Circuits and Systems for Video Technology*, 3 :341–344, 2004.

[Low99] D. Lowe. Object recognition from local scale-invariant features. In *IEEE International Conference on Computer Vision*, pages 1150–1157, 1999.

[Low04] D. Lowe. Distinctive image features from scale-invariant keypoints. *IEEE Transactions Pattern Analysis Machine Intelligence*, 20 :91–110, 2004.

[LRWW98] J. C. Lagarias, J. A. Reeds, M. H. Wright, and P. E. Wright. Convergence properties of the nelder-mead simplex algorithm in low dimensions. *SIAM Journal on Optimization*, 9 :112–147, 1998.

[LZL94] R. Li, B. Zeng, and M.L. Liou. A new three-step search algorithm for block motion estimation. *IEEE Transactions on Circuits and Systems for Video Technology*, 4(4) :438–442, August 1994.

[MA] D. M. Mount and S. Arya. Ann : A library for approximate nearest neighbor searching, http ://www.cs.umd.edu/~mount/ANN/.

[Mah36] P.C. Mahalanobis. On the generalized distance in statistics. *National Institute of Science of India*, 12 :49–55, 1936.

[McQ67] J. McQueen. Some methods for classification and analysis of multivariate observations. In *Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability*, pages 281–297, 1967.

[MdSR+07] G. Maclair, B. Denis de Senneville, M. Ries, B. Quesson, P. Desbarats, J. Benois-Pineau, and C. T. W. Moonen. Pca-based magnetic field modeling : Application for on-line mr temperature monitoring. In *MICCAI*, pages 411–419, 2007.

[MGD98] P. Montesinos, V. Gouet, and R. Deriche. Differential invariants for color images. In *International Conference on Pattern Recognition*, 1998.

[Moo65] G. E. Moore. Cramming more components onto integrated circuits. *Electronics*, 38(8) :114–117, April 1965.

[Mor] Quaid Morris. http ://morrislab.med.utoronto.ca/.

[Mor77] H. Moravec. Towards automatic visual obstacle avoidance. In *Proceedings of the 5th International Joint Conference on Artificial Intelligence*, page 584, August 1977.

[Mor80] H. Moravec. Obstacle avoidance and navigation in the real world by a seeing robot rover. Technical report, Robotics Institute, Carnegie Mellon University & doctoral dissertation, Stanford University, USA, September 1980.

[MS01] K. Mikolajczyk and C. Schmid. Indexing based on scale invariant interest points. In *IEEE International Conference on Computer Vision*, volume 1, pages 525–531, 2001.

[MS04] K. Mikolajczyk and C. Schmid. Scale & affine invariant interest point detectors. *IEEE Transactions Pattern Analysis Machine Intelligence*, 60(1) :63–86, 2004.

[MS05] K. Mikolajczyk and C. Schmid. A performance evaluation of local descriptors. *IEEE Transactions Pattern Analysis Machine Intelligence*, 27(10) :1615–1630, 2005.

[Nas00] John C. Nash. The (dantzig) simplex method for linear programming. *Computing in Science and Engineering*, 2(1) :29–31, 2000.

[Nie00] F. Nielsen. Adaptive randomized algorithms for mosaicing systems. *Transactions of the Institute of Electronics, Information, and Communication Engineers (IEICE), Information and Systems*, E83-D(7) :1386–1394, 2000.

[Nie05] Frank Nielsen. *Visual Computing : Geometry, Graphics, and Vision*. Charles River Media / Thomson Delmar Learning, 2005.

[Nie06a] F. Nielsen. A gpu panorama viewer for generic camera models. In *ShaderX*^{5} *: Advanced Rendering Techniques*. Charles River Media/Thomson Learning, 2006.

[Nie06b] F. Nielsen. Interactive gpu segmentation based on cellular automata. In *ShaderX*^{5} *: Advanced Rendering Techniques*. Charles River Media/Thomson Learning, 2006.

[Nie08] F. Nielsen. An interactive tour of voronoi diagrams on the gpu. In *ShaderX*^{6} *: Advanced Rendering Techniques*. Charles River Media/Thomson Learning, 2008.

[NM65] J. A. Nelder and R. Mead. A simplex method for function minimization. *The Computer Journal*, 7(4) :308–313, January 1965.

[NN04] R. Nock and F. Nielsen. Statistical region merging. *IEEE Transactions Pattern Analysis Machine Intelligence*, 26 :1452–1458, 2004.

[NN07] F. Nielsen and R. Nock. Approximating smallest enclosing balls with applications to machine learning. In *International Journal of Computational Geometry and its Applications (World-Scientific publisher)*, 2007. (invited paper).

[NVI] NVIDIA. Cuda zone,
http ://www.nvidia.com/object/cuda_home.html.

[NY06] F. Nielsen and N. Yamashita. Clairvoyance : A fast and robust precision mosaicing system for gigapixel images. In *Industrial Electronics Society*, CNAM, Paris, France, November 2006. IEEE SP.

[OS88] S. Osher and J.A. Sethian. Fronts propagating with curvature-dependent speed : Algorithms based on hamilton–jacobi formulations. *Journal of Computational Physics*, 79(1) :12–49, 1988.

[PADB08] P. Piro, S. Anthoine, E. Debreuve, and M. Barlaud. Image retrieval via kullback-leibler divergence of patches of multiscale coefficients in the knn framework. In *IEEE International Workshop on Content-Based Multimedia Indexing*, London, UK, June 2008. IEEE Computer Society.

[Par62] E. Parzen. On the estimation of a probability density function and mode. *Annals of Mathematical Statistics*, 33 :1065–1076, 1962.

[Par99] E. Parzen. *Stochastic processes*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1999.

[PD02] N. Paragios and R. Deriche. Geodesic active regions : a new paradigm to deal with frame partition problems in computer

vision. *Journal of Visual Communication and Image Representation, Special Issue on Partial Differential Equations in Image Processing, Computer Vision and Computer Graphics*, 13(1) :249–268, march/june 2002.

[PKPB07] M. Péchaud, R. Keriven, T. Papadopoulo, and J.-M. Badier. Combinatorial optimization for electrode labeling of eeg caps. In *MICCAI*, volume 4792, pages 793–800, 2007.

[PPK03] S. C. Park, M. K. Park, and M. G. Kang. Super-resolution image reconstruction : a technical overview. *Signal Processing Magazine*, 20(3) :21–36, 2003.

[PVDK08] M. Péchaud, I. Vanzetta, T. Deneux, and R. Keriven. Sift-based sequence registration and flow-based cortical vessel segmentation applied to high resolution optical imaging data. In *International Symposium on Biomedical Imaging*, Paris, May 2008.

[RJZ03] M. Rochery, I. Jermyn, and J. Zerubia. Higher order active contours and their application to the detection of line networks in satellite imagery. In *IEEE Workshop Variational, Geometric and Level Set Methods in Computer Vision*, 2003.

[RJZ06] M. Rochery, I. Jermyn, and J. Zerubia. Higher order active contours. *IEEE Transactions Pattern Analysis Machine Intelligence*, 69(1) :27–42, 2006.

[RK05] F. Bernhard R. and Keriven. Spiking neurons and gpus. Technical Report 05-15, CERTIS, November 2005.

[SARK08] H. Sahbi, J. Audibert, J. Rabarisoa, and R. Keriven. Object recognition and retrieval by context dependent similarity kernels. In *IEEE International Workshop on Content-Based Multimedia Indexing*, London, June 2008.

[SB91] M. J. Swain and D. H. Ballard. Color indexing. *IEEE Transactions Pattern Analysis Machine Intelligence*, 7(1) :11–32, 1991.

[SB97] S. M. Smith and J. M. Brady. Susan - a new approach to low level image processing. *IEEE Transactions Pattern Analysis Machine Intelligence*, 23(1) :45–78, 1997.

[SBW02] H. Schweitzer, J. W. Bell, and F. Wu. Very fast template matching. In *European Conference on Computer Vision*, pages 358–372, London, UK, 2002.

[SC96] B. Schiele and J. L. Crowley. Object recognition using multidimensional receptive field histograms. In *European Conference on Computer Vision*, pages 610–619, 1996.

[Sch96] C. Schmid. *Appariement d'images par invariants locaux de niveaux de gris*. Doctoral thesis, Institut National Polytechnique de Grenoble, GRAVIR, July 1996.

[SEAK08a] H. Sahbi, P. Etyngier, J.Y. Audibert, and R. Keriven. Interactive image retrieval. In *IEEE International Conference on Acoustics, Speech, and Signal Processing*, Las Vegas, Apr 2008.

[SEAK08b] H. Sahbi, P. Etyngier, J.Y. Audibert, and R. Keriven. Manifold learning using robust graph laplacian for interactive image retrieval. In *IEEE International Conference on Computer Vision and Pattern Recognition*, Anchorage, Alaska, June 2008.

[Set99] J.A. Sethian. *Level Set Methods and Fast Marching Methods : Evolving Interfaces in Computational Geometry, Fluid Mechanics, Computer Vision, and Materials Sciences*. Cambridge Monograph on Applied and Computational Mathematics. Cambridge University Press, 1999.

[Sha48] C. E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27 :379–423, 623–656, July, October 1948.

[She58] D. L. Shell. A high-speed sorting procedure. *Communications of the ACM*, 2(7) :30–33, July 1958.

[SHH96] C. Studholme, D. L. G. Hill, and D. J. Hawkes. Incorporating connected region labelling into automatic image registration using mutual information. In *Workshop on Mathematical Methods in Biomedical Image Analysis*, page 23, Washington, DC, USA, 1996.

[SJ87] I. K. Sethi and R. C. Jain. Finding trajectories of feature points in a monocular image sequence. *IEEE Transactions Pattern Analysis Machine Intelligence*, 9(1) :56–73, 1987.

[SKMG04] D. Serby, E. Koller-Meier, and L. Van Gool. Probabilistic object tracking using multiple features. In *International Conference on Pattern Recognition*, pages 184–187, Washington, DC, USA, 2004.

[SM96] C. Schmid and R. Mohr. Image retrieval using local characterization. In *IEEE International Conference on Image Processing*, volume 2, pages 781–784, September 1996.

[SM97] C. Schmid and R. Mohr. Local grayvalue invariants for image retrieval. *IEEE Transactions Pattern Analysis Machine Intelligence*, 19(5) :530–534, May 1997.

[SMB98] C. Schmid, R. Mohr, and C. Bauckhage. Comparing and evaluating interest points. In *IEEE International Conference on Image Processing*, Bombay, India, January 1998.

[SMB00] C. Schmid, R. Mohr, and C. Bauckhage. Evaluation of interest point detectors. *IEEE Transactions Pattern Analysis Machine Intelligence*, 37(2) :151–172, 2000.

[Soi99] P. Soille. *Morphological Image Analysis*. Springer-Verlag, 1999.

[SSWC88] P. K. Sahoo, S. Soltani, A. K.C. Wong, and Y. C. Chen. A survey of thresholding techniques. *Computer Vision, Graphics, and Image Processing*, 41(2) :233–260, 1988.

[TM07] R. Trichet and B. Mérialdo. Probabilistic matching algorithm for keypoint based object tracking using a delaunay triangulation. In *International Workshop on Image Analysis for Multimedia Interactive Services*, Santorini, Greece, June 2007.

[Ton06] F. Tonnin. *Description d'images fixes dans le domaine compressé*. PhD thesis, Université de Rennes 1, 2006.

[Tre85] A. Treisman. Preattentive processing in vision. *Computer Vision, Graphics, and Image Processing*, 31(2) :156–177, 1985.

[TYW01] A. Tsa, A.J. Yezzi, and A.S. Willsky. Curve evolution implementation of the mumford-shah functional for image segmentation, denoising, interpolation, and magnification. *IEEE Transactions on Image Processing*, 10(8) :1169–1186, 2001.

[VC02] L. Vese and T. Chan. A multiphase level set framework for image segmentation using the mumford and shah model. *IEEE Transactions Pattern Analysis Machine Intelligence*, 50(3) :271–293, 2002.

[VM07] V. Vilaplana and F. Marques. Region-based hierarchical representation for object detection. In *IEEE International Workshop on Content-Based Multimedia Indexing*, 2007.

[VRB01] C. J. Veenman, M. J. T. Reinders, and E. Backer. Resolving motion correspondence for densely moving points. *IEEE Transactions Pattern Analysis Machine Intelligence*, 23(1) :54–72, 2001.

[WADP97] C. Wren, A. Azarbayejani, T. Darrell, and A. Pentl. Pfinder : Real-time tracking of the human body. *IEEE Transactions Pattern Analysis Machine Intelligence*, 19 :780–785, 1997.

[WB98] R. Weber and S. Blott. A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces. In *International Conference on Very Large Data Bases*, pages 194–205, 1998.

[WBC+05] J. Wang, P. Bhat, R. A. Colburn, M. Agrawala, and M. F. Cohen. Interactive video cutout. In *SIGGRAPH*, 2005.

[Wik] Wikipedia. http ://www.wikipedia.org.

[Wol06] E. Wolsztynski. *Critère d'entropie pour l'estimation semi-paramétrique*. PhD thesis, Université de Nice - Sophia Antipolis, 2006.

[WTP05] E. Wolsztynski, E. Thierry, and L. Pronzato. Minimum entropy estimators in semi parametric regression models. In *International Conference on Applied Stochastic Models and Data Analysis*, pages 882–889, Brest, France, 2005.

[WVA+96] W. Wells, P. Viola, H. Atsumi, S. Nakajima, and R. Kikinis. Multi-modal volume registration by maximization of mutual information. *Medical Image Analysis*, 1 :35–51, 1996.

[XP98] C. Xu and J.L. Prince. Snakes, shapes, and gradient vector flow. *IEEE Transactions on Image Processing*, 7(3) :359–369, 1998.

[YJS06] A. Yilmaz, O. Javed, and M. Shah. Object tracking : A survey. *ACM Computing Surveys*, 38(4) :13, 2006.

[YLS04] A. Yilmaz, X. Li, and M. Shah. Contour-based object tracking with occlusion handling in video acquired using mobile cameras. *IEEE Transactions Pattern Analysis Machine Intelligence*, 26(11) :1531–1536, 2004.

[ZLC02] S. Zhu, X. Lin, and L. P. Chau. Hexagon-based search pattern for fast block motion estimation. *IEEE Transactions on Circuits and Systems for Video Technology*, 12(5) :349–355, May 2002.

[ZM00] S. Zhu and K.K. Ma. A new diamond search algorithm for fast block-matching motion estimation. *IEEE Transactions on Image Processing*, 9(2) :287–290, February 2000.

- RÉSUMÉ -

Le problème du suivi d'objets dans une vidéo se pose dans des domaines tels que la vision par ordinateur (vidéo-surveillance par exemple) et la post-production télévisuelle et cinématographique (effets spéciaux). Il se décline en deux variantes principales : le suivi d'une région d'intérêt, qui désigne un suivi grossier d'objet, et la segmentation spatio-temporelle, qui correspond à un suivi précis des contours de l'objet d'intérêt. Dans les deux cas, la région ou l'objet d'intérêt doivent avoir été préalablement détourés sur la première, et éventuellement la dernière, image de la séquence vidéo. Nous proposons dans cette thèse une méthode pour chacun de ces types de suivi ainsi qu'une implémentation rapide tirant partie du Graphics Processing Unit (GPU) d'une méthode de suivi de régions d'intérêt développée par ailleurs. La première méthode repose sur l'analyse de trajectoires temporelles de points saillants et réalise un suivi de régions d'intérêt. Des points saillants (typiquement des lieux de forte courbure des lignes isointensité) sont détectés dans toutes les images de la séquence. Les trajectoires sont construites en liant les points des images successives dont les voisinages sont cohérents. Notre contribution réside premièrement dans l'analyse des trajectoires sur un groupe d'images, ce qui améliore la qualité d'estimation du mouvement. De plus, nous utilisons une pondération spatio-temporelle pour chaque trajectoire qui permet d'ajouter une contrainte temporelle sur le mouvement tout en prenant en compte les déformations géométriques locales de l'objet ignorées par un modèle de mouvement global. La seconde méthode réalise une segmentation spatio-temporelle. Elle repose sur l'estimation du mouvement du contour de l'objet en s'appuyant sur l'information contenue dans une couronne qui s'étend de part et d'autre de ce contour. Cette couronne nous renseigne sur le contraste entre le fond et l'objet dans un contexte local. C'est là notre première contribution. De plus, la mise en correspondance par une mesure de similarité statistique, à savoir l'entropie du résiduel, d'une portion de la couronne et d'une zone de l'image suivante dans la séquence permet d'améliorer le suivi tout en facilitant le choix de la taille optimale de la couronne. Enfin, nous proposons une implémentation rapide d'une méthode de suivi de régions d'intérêt existante. Cette méthode repose sur l'utilisation d'une mesure de similarité statistique : la divergence de Kullback-Leibler. Cette divergence peut être estimée dans un espace de haute dimension à l'aide de multiples calculs de distances au k-ème plus proche voisin dans cet espace. Ces calculs étant très coûteux, nous proposons une implémentation parallèle sur GPU (grâce à l'interface logiciel CUDA de NVIDIA) de la recherche exhaustive des k plus proches voisins. Nous montrons que cette implémentation permet d'accélérer le suivi des objets, jusqu'à un facteur 15 par rapport à une implémentation de cette recherche nécessitant au préalable une structuration des données.

- ABSTRACT -

The problem of object tracking is a problem arising in domains such as computer vision (video surveillance for instance) and cinematographic post-production (special effects). There are two major classes of solution to this problem : region of interest tracking, which indicates a coarse tracking, and space-time segmentation, which corresponds to a precise tracking of the region of interest's contour. In both cases, the region of interest must be selected beforehand on the first, and possibly on the last image of the video sequence. In this thesis, we propose two tracking methods (one of each type). We propose also a fast implementation of an existing tracking method on Graphics Processing Unit (GPU). The first method is based on the analysis of temporal trajectories of salient points and provides a region of interest tracking. Salient points (typically of point of strong curvature of the isointensity lines) are detected in all the images of the sequence. The trajectories are built by matching salient points of consecutive images whose neighbourhoods are coherent. Our first contribution consists in the analysis of the trajectories on a group of pictures, which improves the motion estimation quality. Moreover, we use a space-time weighting for each trajectory which makes it possible to add a temporal constraint on the movement while taking into account the local geometrical deformations of the object ignored by a global motion model. The second method performs a space-time segmentation. The object contour motion is estimated using the information contained in an outer-layer centered on the object contour. Our first contribution is the use of this outer-layer which contains information about both the background and the object in a local context. Moreover, the matching using a statistical similarity measure (residual entropy) allows to improve the tracking while facilitating the choice of the optimal size of the crown. Finally, we propose a fast implementation of an existing tracking method of region of interest. This method relies on the use of a statistical similarity measure : the Kullback-Leibler divergence. This divergence can be estimated in a high dimension space using k-th nearest neighbor distance. These calculations being computationally very expensive, we propose a parallel implementation of the exhaustive search of the k-th nearest neighbors using GPU programming (via the programming interface NVIDIA CUDA). We show that this implementation speeds the tracking process up to a factor 15 compared to a classical implementation of this search using data structuring methods.