NNT : 2016SACLY018

Thèse de doctorat de l'Université Paris-Saclay préparée à l'ENSTA ParisTech

Ecole doctorale n°573
INTERFACES
Spécialité de doctorat : Informatique

par

M. Matthieu Garrigues

Accélération Algorithmique et Logicielle de l'Analyse Vidéo du Mouvement

Thèse présentée et soutenue à l'ENSTA-ParisTech, le 9 décembre 2016.

Composition du Jury :

M.	Guy Le Besnerais	Professeur ONERA	(Président du jury)
M.	Michel Devy	Directeur de recherche LAAS	(Rapporteur)
M.	Dominique Houzet	Professeur Grenoble INP	(Rapporteur)
M.	Emanuel Aldea	Maître de conférences Université Paris Sud	(Examinateur)
M.	Thierry Géraud	Professeur EPITA	(Examinateur)
M.	Antoine Manzanera	Professeur ENSTA-ParisTech	(Directeur de thèse)

université PARIS-SACLAY

ÉCOLE DOCTORALE
INTERFACES
Approches Interdisciplinaires : Fondements, Applications et Innovation

ENSTA ParisTech université PARIS-SACLAY

Accélération Algorithmique et Logicielle de l'Analyse Vidéo du Mouvement

Thèse préparée par Matthieu Garrigues sous la direction d'Antoine Manzanera à l'unité de Unité d'Informatique et d'Ingénierie des Systèmes de l'ENSTA-ParisTech.

Mots clés : traitement d'image, vision par ordinateur, calcul parallèle, estimation du mouvement, flux optique, suivi, génie logiciel.

Résumé : L'analyse du mouvement apparent dans une vidéo est un problème étudié depuis plusieurs décennies. Les algorithmes résultant de ces recherches ont déjà mené à un très large panel d'applications dans la vidéo surveillance, le domaine militaire, le cinéma, les jeux vidéos, etc. Cependant, malgré l'augmentation de la rapidité de calcul des processeurs et la progression de l'état de l'art, écrire un logiciel capable d'estimer à la fois précisément et rapidement le mouvement reste un problème largement ouvert.

La première difficulté, de nature algorithmique, est la plus souvent abordée par l'état de l'art. Elle concerne la qualité et la robustesse de l'estimation. En effet, suivre le mouvement d'un objet dans une vidéo est un problème difficile, surtout si l'objet en question subit des déformations, des changements d'illuminations, ou encore des mouvements rapides.

La deuxième difficulté concerne les fortes contraintes de temps de calcul qui impactent de nombreuses applications et limitent en général la fenêtre temporelle disponible à quelques millisecondes. Pour atteindre ces fréquences de traitement, il est nécessaire de prendre en compte la complexité algorithmique, mais aussi l'architecture du processeur cible. Son parallélisme, la nature et la taille de son cache mémoire, son jeu d'instructions et d'autres

spécificités fournissent des possibilités d'optimisation des programmes qu'il est opportun de prendre en compte dès la conception de l'algorithme.

Cette deuxième difficulté a malheureusement tendance à rendre les codes des implantations longs et complexes car seul des langages bas niveau permettent l'adéquation optimale d'un code à une architecture de processeur. Pour pallier ce problème, des bibliothèques de traitement d'image ont été proposées, mais leurs abstractions ont souvent un impact négatif sur le temps de calcul.

Ces constatations nous ont amenés à développer deux axes de recherches : **la conception d'outils logiciels pour le prototypage rapide d'algorithme de traitement d'image** et **la conception d'algorithmes d'analyse de mouvement rapide**.

Nous avons développé et mis à disposition de la communauté deux outils. Le premier est *Video++* [23], une bibliothèque de traitement d'image tirant parti des derniers standards C++11 et C++14 pour simplifier l'écriture de traitements d'image performants. Par exemple, l'extrait de code suivant additionne deux images en tirant parti des multiples cœurs et des extensions vectorielles des processeurs actuels :

Listing 1 – Addition de deux images avec `pixel_wise`

```
1 image2d<int> sum = vpp::pixel_wise(A, B) |
2                [] (int a, int b) { return a + b; };
```

Nous avons ensuite proposé un ensemble de primitives logicielles pour assister l'exploration de l'espace des paramètres d'un algorithme et la recherche de son paramétrage optimal [21]. Dans ce travail, nous avons accéléré la conception d'algorithmes en automatisant des tâches répétitives souvent réalisées manuellement par le développeur.

A l'aide de ces deux outils, nous avons pu développer et optimiser deux approches d'analyse de mouvement offrant des temps de calcul assez petits pour des applications embarquées sur des processeurs basse consommation. La première, nommée *Video Extruder*, transforme une vidéo en un flux semi-dense de plusieurs milliers de trajectoires mis à jour à chaque nouvelle image. La cadence de traitement de 150Hz a été atteinte entre autre grâce à la minimisation de l'espace mémoire utilisé et la parallélisation de l'algorithme

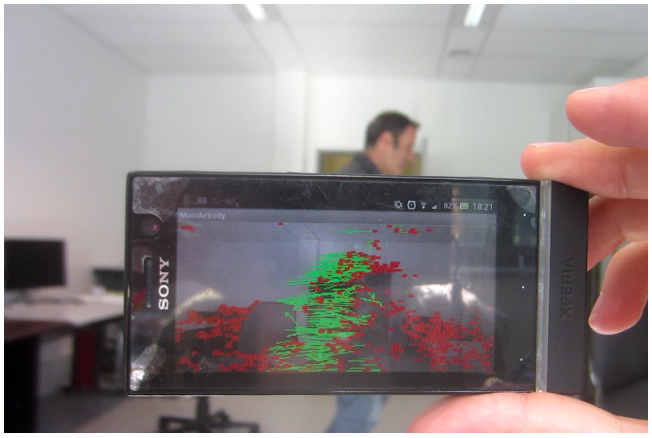

FIGURE 1 – *Video Extruder* fonctionne à 10Hz sur un smartphone d'entrée de gamme Xperia U embarquant un processeur ARM dual-core STE U8500. Vidéo disponible sur youtube : https://www.youtube.com/watch?v=jHNSshAsEJ4

d'estimation du flux optique. Sur l'image 1, l'algorithme traite le flux vidéo d'un téléphone Android en temps réel.

La deuxième approche est le suivi épars de points d'intérêt. Contrairement au suivi semi-dense, ce type d'algorithme ne peut pas bénéficier de la cohérence locale du champ de mouvement. Des descripteurs de points à haut pouvoir discriminant sont alors mis en correspondance par une recherche de plus proche voisin. Pour accélérer cette recherche, nous avons proposé une indexation jusqu'à deux ordres de grandeur plus rapide que celles de l'état de l'art (FLANN).

Dans le volet suivant, nous avons proposé une méthode rapide et robuste de l'estimation du flux optique sur une scène routière, observée depuis un véhicule en mouvement. Sur le challenge *KITTI Optical Flow 2012*, dans la catégorie « estimation partielle », nous avons obtenu le taux d'erreur le plus faible, pour un temps de calcul 1000 fois inférieur à l'algorithme classé deuxième dans la même catégorie. Les images de la figure 2 montrent les résultats obtenus sur une paire d'images de la base *KITTI*.

Durant toute la durée de cette thèse, nous avons pu appliquer nos algorithmes à plusieurs applications, développées pour nos besoins propres ou pour ceux de partenaires. Ces applications incluent la stabilisation vidéo, la

segmentation d'objets mobiles et la reconnaissance d'actions [49, 47, 48].

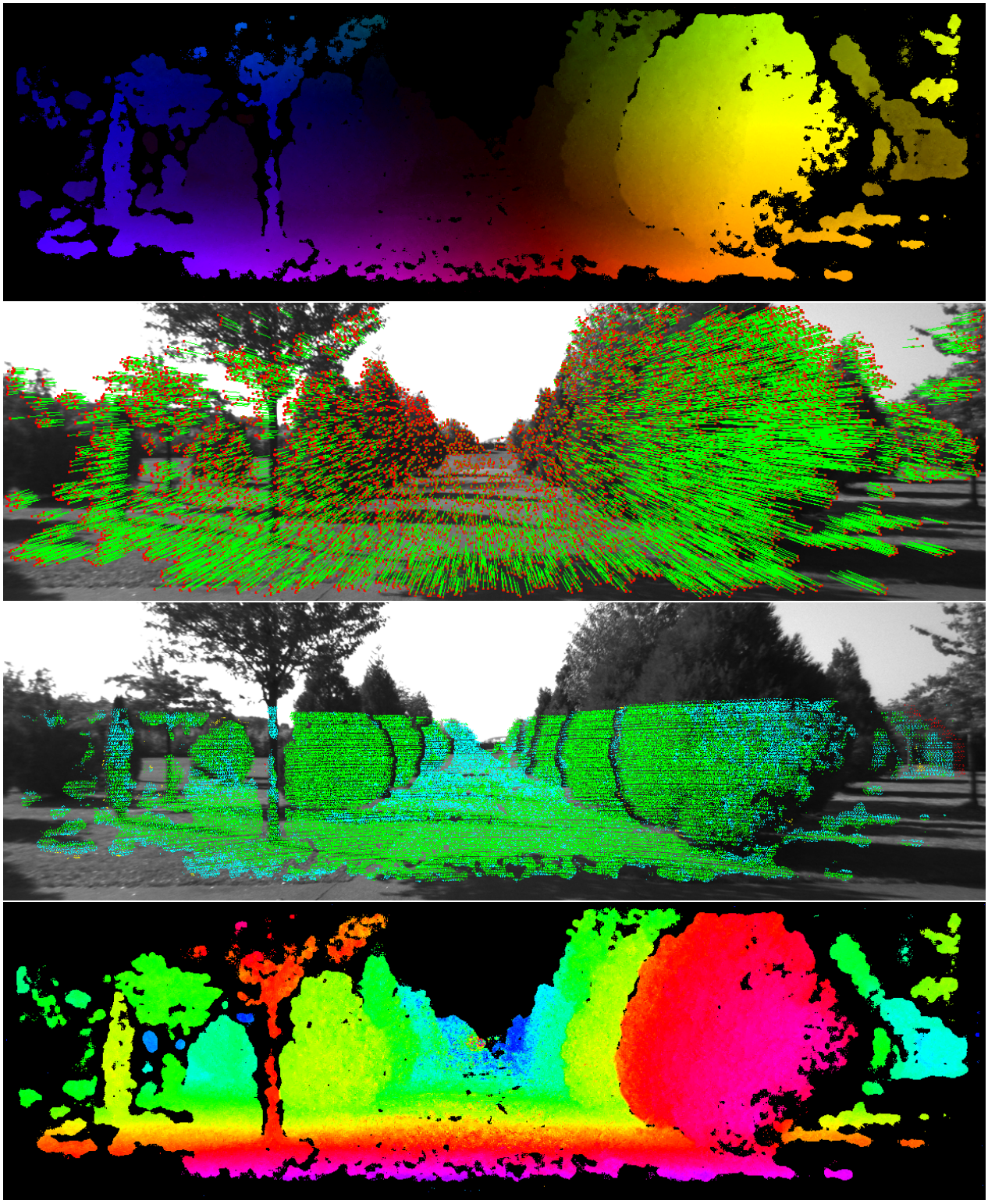

FIGURE 2 – De haut en bas. 1 - Flux optique semi-dense estimé sur une paire d'image de la base *KITTI*. 2 - Vecteurs de flux optique estimés. 3 - Précision du flux optique calculé sur une paire d'images de la base *KITTI*. Les erreurs inférieures à 1 pixel sont affichées en verts, entre 1 et 3 pixels en bleu, entre 3 et 10 pixels en jaune et supérieures à 10 pixels en rouge. 4 - Carte de profondeur obtenue à partir du flux optique et de l'odométrie visuelle.

Table des matières

Chapitre 1

Introduction

En 1965, Gordon Moore, l'un des cofondateurs d'Intel, a observé que le nombre de transistors placés sur un circuit électronique pour un prix par transistor optimal doublait tout les deux ans. Alors que les processeurs contenaient 50 transistors, il a prédit [42] que l'industrie allait tenir ce rythme pendant au moins une décennie, et probablement amener le nombre de transistors placés sur un circuit intégré à 65 000 en 1975. Cette tendance (voir figure 1.1) a justement été ré-estimée cette année-là à un doublement tout les 18 mois. En 2016, le PDG de NVidia Jen-Hsun Huang a présenté le Tesla P100, un processeur composé de 15 milliards de transistors, soit l'aboutissement de 34 itérations de la loi de Moore.

Bien que cette progression ait tracé une courbe exponentielle s'étalant sur 51 ans, les constructeurs de processeurs ont récemment éprouvé deux difficultés majeures dans l'augmentation de la puissance de calcul des processeurs. Le mur de la dissipation thermique (voir figure 1.2), rencontré dans les années 2000, a empêché les constructeurs d'augmenter la fréquence d'horloge à chaque nouvelle génération de transistor. Plus récemment, le prix et la difficulté du développement des nouvelles technologies de gravure a significativement augmenté pour des raisons physiques, comme l'effet tunnel, responsable d'une augmentation des fuites de courant quand la taille de grille diminue.

Ces difficultés rencontrées au niveau du transistor ont poussé les industriels à intensifier leurs efforts pour améliorer les architectures de processeur. Ces travaux ont abouti à deux types de transformation. Premièrement, la pa-

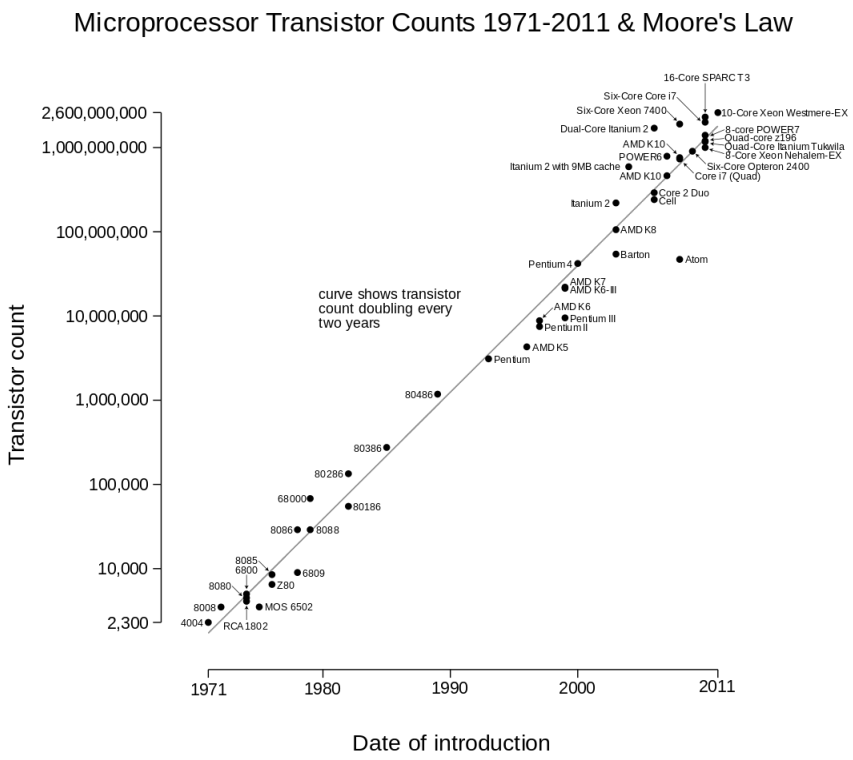

FIGURE 1.1 – Évolution du nombre de transistors placés sur un processeur de 1971 à 2011. *Source: Wikipedia*

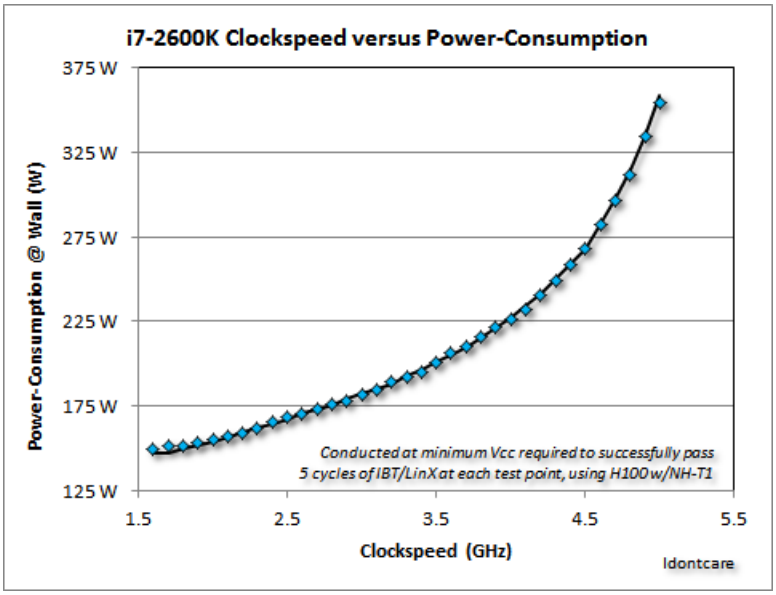

FIGURE 1.2 – Évolution de la consommation de courant en fonction de la fréquence d'horloge pour le processeur i7-2600k. *Source: Utilisateur idontcare du site Arnandtech*

rallélisation des calculs, car si la fréquence de calcul d'une unité est bridée par le mur thermique, rien n'empêche de multiplier la capacité d'un processeur en y plaçant plusieurs unités capables de travailler en parallèle. C'est pourquoi depuis 2006, les processeurs ont vu leur parallélisme augmenter au point de pouvoir exécuter jusqu'à plusieurs milliers d'opérations simultanément. Deux types de parallélisme ont alors émergé :

- les processeurs **multi-cœurs**, contenant plusieurs processeurs au sein d'une puce et capables d'exécuter différents programmes simultanément.
- les unités de calcul **vectoriel**, exécutant simultanément la même opération sur différentes données.

Ces deux paradigmes sont respectivement dénommés par Flynn [19] *Multiple Instruction stream Multiple Data* (**MIMD**) et *Single Instruction Multiple Data* (**SIMD**).

Le deuxième type d'optimisation architecturale a pour but de diminuer le temps d'accès aux données. En effet, si un accès à la mémoire RAM coûtait

autrefois un cycle processeur, il prend aujourd'hui 100 cycles. Pour contourner ce problème, des mémoires cache plus rapides ont été placées sur les processeurs pour limiter le nombre de transactions effectuées avec la mémoire RAM. Jusqu'à trois niveaux de cache sont utilisés dans les processeurs actuels. Ces trois niveaux sont appelés L1, L2 et L3, L1 étant le plus proche des unités de calcul, donc le plus rapide d'accès. Ainsi, un accès L1 est jusqu'à 2 ordres de grandeur plus rapide qu'un accès RAM. La figure 1.3 met en évidence les différences des ordres de grandeur des temps d'exécution/latence des différents accès mémoire et d'autres opérations CPU.

Contrairement à l'augmentation de la fréquence d'horloge qui accélère tous les programmes de la même manière, l'impact des mémoires cache et du parallélisme dépend de la nature du programme. En effet, tirer parti du parallélisme implique une modification du code, voire de l'algorithme s'il n'est pas de nature parallèle, et le cache n'est rentabilisé que si les accès mémoire se concentrent effectivement sur les zones qu'il a chargées. D'autres spécificités, comme la nature des opérations arithmétiques (voir figure 1.3) ont aussi un impact significatif sur le temps de calcul. Par exemple, une division peut consommer un ordre de grandeur de cycles CPU de plus qu'une addition.

Ces particularités architecturales augmentent considérablement l'influence des détails d'implémentation d'un algorithme et posent un nouveau challenge aux développeurs d'applications : **adapter les algorithmes aux architectures** et écrire des **implémentations logicielles rapides** tirant parti d'un maximum d'opportunités proposées par le processeur.

Cela est particulièrement vrai pour les applications de traitement d'image où les données manipulées sont souvent volumineuses. Dans certains cas, les images d'une vidéo doivent être traitées à une cadence de 20-30 images par seconde, laissant une fenêtre temporelle de seulement 30-50ms à l'exécution de l'algorithme. Les applications embarquées ont des contraintes encore plus fortes car les traitements sont exécutés sur des processeurs où la puissance de calcul a été réduite pour optimiser la consommation d'énergie.

En parallèle, l'explosion du marché des *smartphones* a poussé l'industrie des semi-conducteurs à augmenter les capacités de calcul des **processeurs basse consommation**, et les constructeurs de caméras à améliorer la qualité des **capteurs bas coût**. Ainsi, malgré les difficultés évoquées précédemment,

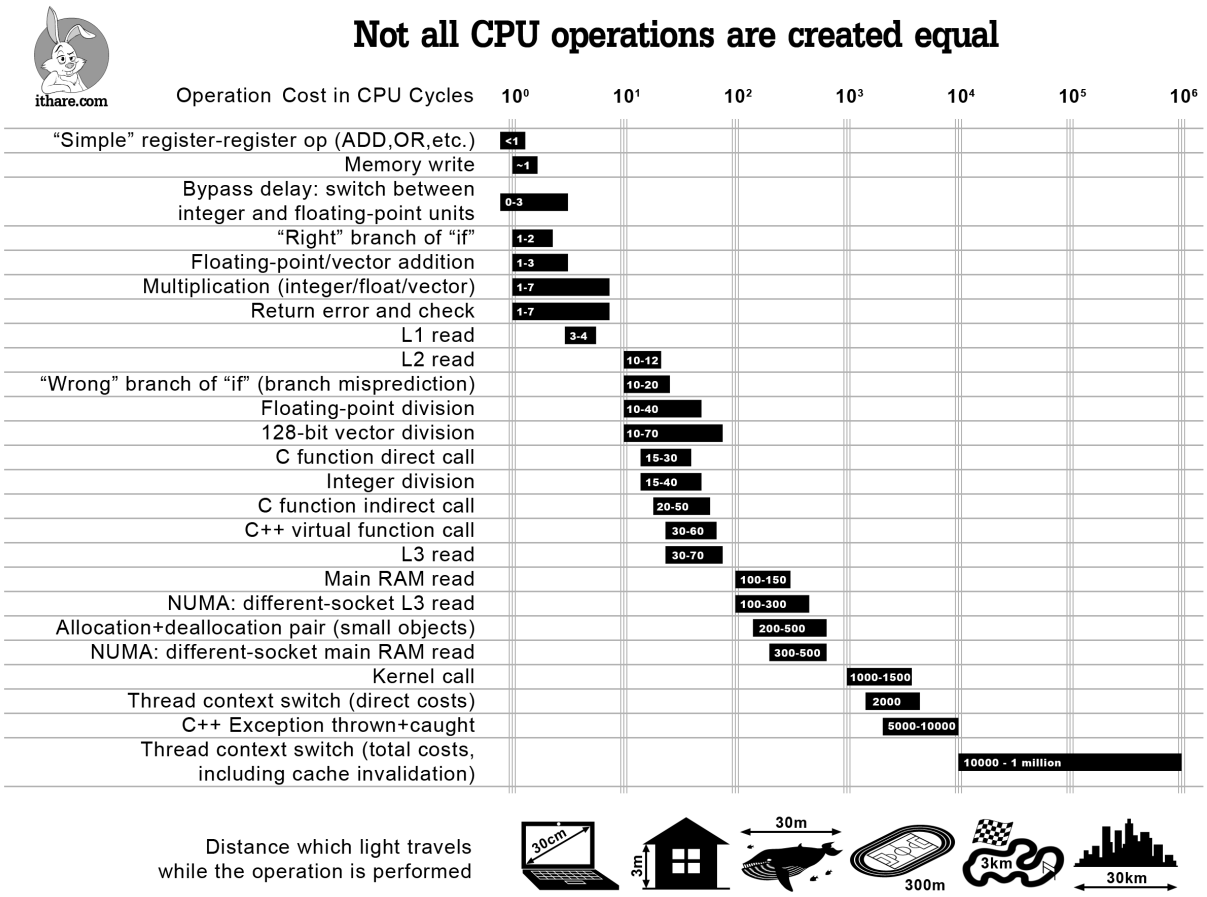

FIGURE 1.3 – Toutes les opérations CPU ne sont pas créées égales. Illustration du site IT Hare [50] mettant en évidence sur une échelle logarithmique les ordres de grandeur des temps d'exécution de différentes opérations CPU. Données collectées par Agner Fog [20].

nous avons observé ces dernières années un développement des applications commerciales reposant sur la vision par ordinateur. En effet, les avancées technologiques des processeurs basse consommation et des capteurs ont ouvert le champ à de nouveaux types d'applications comme la réalité augmentée ou la conduite autonome des voitures et des drones.

Toutefois, des difficultés persistent et de nombreux problèmes de vision par ordinateur n'ont pas encore été assez optimisés pour les applications temps réel et/ou embarquées sur des processeurs basse consommation. Nous nous intéressons dans ces travaux à l'un d'entre deux : **l'estimation du mouvement apparent dans une vidéo**.

Ce problème, couramment appelé calcul du flux optique, a captivé l'intérêt des chercheurs en traitement d'image depuis quelques dizaines d'années. Bien que les meilleurs algorithmes de l'état de l'art atteignent une grande précision, leur complexité calculatoire les rend inapplicables aux applications à fortes contraintes de temps de calcul. Sur les 70 solutions référencées par l'évaluation *KITTI optical flow 2012* [25] (voir figure 1.4), très peu fournissent un bon compromis entre qualité et temps de calcul.

Ce déséquilibre met en évidence la difficulté de combiner qualité de flux optique et rapidité d'exécution. Pourtant, une solution à la fois précise et rapide faciliterait significativement le calcul en temps réel d'un grand nombre d'algorithmes reposant sur le flux optique.

C'est pour cette raison que nous avons focalisé nos travaux sur le calcul du flux optique en donnant la priorité à la diminution du temps de calcul. De plus, comme nous l'avons vu précédemment, l'accélération de l'exécution d'un algorithme sur des processeurs modernes implique deux axes de travail : **l'adéquation de l'algorithme aux spécificités de l'architecture et le développement d'une implémentation logicielle rapide d'exécution.**

Le prochain chapitre de ce manuscrit est dédié à l'aspect logiciel. Le but de ces travaux est de fournir un ensemble d'outils facilitant le développement de nos algorithmes. Pour cela, nous sommes partis de l'observation suivante : la conception et l'implémentation d'algorithmes de traitement d'image suit en général un cycle de développement en spirale. En effet, les premières expériences étant rarement concluantes, il est courant d'alterner entre recherche d'améliorations de la méthode et modification de l'implémentation logicielle.

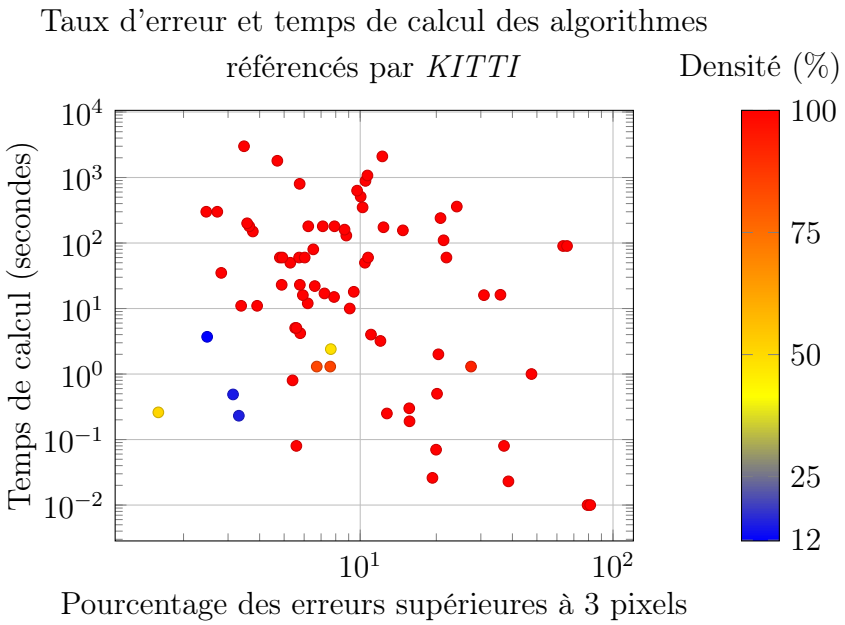

FIGURE 1.4 – Taux d'erreur et temps de calcul des implémentations d'algorithmes référencés par le benchmark *KITTI optical flow* section "estimation partielle", sur une échelle logarithmique. Les couleurs encodent la densité de couverture du flux optique calculé. Notons que les temps de calcul annoncés dépendent de plusieurs facteurs, variant d'une approche à l'autre : le langage de programmation, la plateforme matérielle et les détails d'implémentation.

Cependant, le temps nécessaire à une itération de ce cycle limite souvent le chercheur. Pour pallier ce problème, nous avons proposé deux outils. Le premier est *Vidéo++*, une bibliothèque tirant parti des nouveautés introduites par le langage C++14 [34] pour **faciliter l'écriture de traitements d'image rapides**. Ensuite, nous avons proposé *GPOF* (*Generic Parameter Optimization Framework*), une bibliothèque Python aidant à **l'analyse de l'espace des paramètres d'un algorithme**. Cet outil permet entre autres de rapidement visualiser le comportement d'un algorithme sur une plage de paramètres donnée, et ainsi valider ou invalider des hypothèses.

Le problème de l'analyse de mouvement est abordé dans le chapitre 3. Nous avons proposé deux estimateurs de mouvement se différenciant principalement par le nombre de points estimés. Le premier est **l'estimation d'un champ de plusieurs milliers de trajectoires**, appelé semi-dense. Dans un premier temps, nous avons repris certaines idées de l'état de l'art, comme la pyramide d'images, pour estimer de grands mouvements ou la descente de gradient pour maximiser une mesure de similarité. Ensuite, pour abaisser le temps de calcul à moins de **10 millisecondes par image** sur un CPU multi-coeur, nous avons utilisé des techniques comme la minimisation de l'espace mémoire consommé par l'algorithme, la réduction du nombre de calculs flottants, ou encore la parallélisation SIMD et multi-cœur. L'image 1.5 montre le flux de trajectoires obtenu sur une scène urbaine du jeu de données CamVid [9].

La deuxième catégorie d'algorithmes d'estimation de mouvement que nous avons optimisée est **éparse**, soit quelques centaines de points estimés par image. Le caractère épars d'une estimation permet en général d'utiliser des descripteurs à grande dimension pour identifier les points à mettre en correspondance. Dans ces travaux, nous avons proposé deux indexations de points respectivement basées sur leurs localisations dans l'espace image et sur leurs descripteurs. Comme pour le suivi semi-dense, nous avons tiré parti des paradigmes SIMD et MIMD des CPU modernes. Notre approche s'est révélée jusqu'à deux ordres de grandeur plus rapide que l'état de l'art (FLANN [44]) sur des ensembles de points de taille inférieure à 50 000 et quand le déplacement des points peut être grossièrement prédit.

Dans le chapitre 4, nous nous intéressons à l'estimation du mouvement

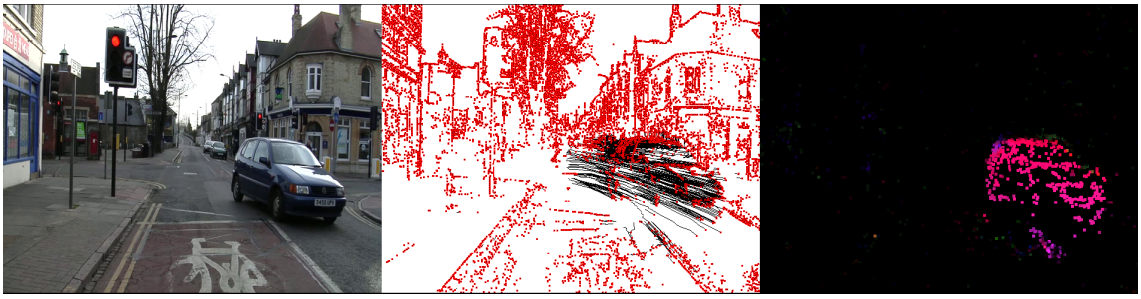

FIGURE 1.5 – Résultat de l'extraction semi-dense de trajectoires. De gauche à droite : la trame vidéo en entrée, les particules en rouge et leurs trajectoires en noir, le champ de mouvement semi-dense représentant les coordonnées polaires des vecteurs de mouvement avec un code de couleur $<\rho,\theta> \equiv <$intensité, teinte$>$.

d'une scène statique filmée par une caméra mobile. Ce scénario est typique des applications de reconstruction 3D. Dans ce cas précis, le mouvement apparent ne varie qu'en fonction du mouvement de la caméra et de la géométrie de la scène. Nous avons donc séparé et optimisé le calcul du flux optique en deux étapes : l'estimation du mouvement de la caméra, ou **odométrie visuelle** et l'estimation du champ de profondeur de la scène observée, ou **flux épipolaire**.

La précision de l'odométrie visuelle dépendant fortement de la répartition et de la qualité de l'ensemble des vecteurs de flux optique estimée au préalable, nous avons tenté d'en améliorer la qualité tout en diminuant le temps de calcul. Pour cela, à partir des travaux sur le flux optique épars présenté dans le chapitre 3 nous avons construit l'estimation rapide d'un champ de vecteurs ayant à la fois une bonne répartition spatiale et un nombre d'erreurs réduit. Cela a permis de diminuer le nombre d'itérations de l'odométrie visuelle avant convergence, tout en améliorant sa précision.

Nous avons ensuite développé une méthode d'estimation du flux épipolaire se différenciant de l'état de l'art par sa robustesse et son faible temps de calcul. Comme beaucoup d'autres approches, nous avons restreint l'espace de recherche aux lignes épipolaires. Cependant, afin de réduire davantage le temps de calcul, nous avons proposé un nouvel algorithme propageant les informations de mouvement déjà estimées de voisin en voisin. Les erreurs

ont ensuite été filtrées avec une méthode reposant en partie sur l'algorithme de flux optique Lucas-Kanade [7]. L'illustration 1.6 présente une carte de profondeur obtenue avec notre méthode.

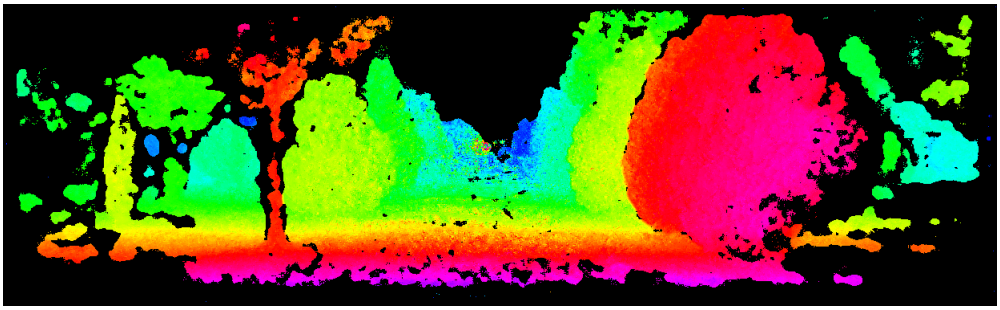

FIGURE 1.6 – Estimation de la profondeur construite à partir du flux épipolaire proposé.

Finalement, dans le chapitre 5, nous présentons quatre applications développées en parallèle de cette thèse et reposant sur les accélérations algorithmiques et logicielles que nous avons proposées.

Chapitre 2

Aide à la conception d'algorithmes de traitement d'image

Sommaire

2.1 Introduction

En traitement d'image hautes performances comme dans beaucoup d'autres domaines, la conception d'algorithme suit les trois étapes suivantes, illustrées par la figure 2.1.

1. **L'idée :** Après avoir identifié un problème et analysé l'état de l'art des solutions, un chercheur/développeur conçoit et élabore une idée dans le but d'améliorer la résolution de ce problème.
2. **Le prototype** : Cette idée est ensuite développée sous forme d'un algorithme et implantée dans un langage de programmation.
3. **L'évaluation et l'optimisation des paramètres** : la méthode développée est ensuite évaluée, et ses paramètres optimisés selon certains critères d'évaluation. En fonction des observations résultant de ces expériences, les résultats sont publiés/déployés et/ou des idées d'améliorations peuvent mener à un nouveau prototype, ramenant le chercheur/développeur à l'étape 2.

Malgré l'immense variété des idées et prototypes imaginables, la conception de prototypes et leur évaluation mobilisent un ensemble de tâches communes mises en œuvre par une grande majorité de chercheurs/développeurs lors de l'élaboration d'algorithmes de traitement d'images haute performance.

C'est pourquoi en conservant la généricité nécessaire pour s'adapter à chaque contexte de développement, il est possible de concevoir des outils permettant de simplifier la réalisation de ces tâches. Dans le cadre de cette thèse, nous avons développé deux nouveaux outils de ce type, qui sont présentés dans le présent chapitre.

Le premier est ***Video++*** (section 2.2), une bibliothèque C++ aidant à l'écriture d'algorithmes de traitement d'image hautes performances. Le deuxième est ***GPOF*** (**G**eneric **P**arameter **O**ptimisation **F**ramework) (section 2.3), une boîte à outils aidant à l'exploration de l'espace des paramètres d'un algorithme et à la recherche de son paramétrage optimal.

Les deux outils ont été mis à disposition de la communauté par la publication de leur code source en licence libre.

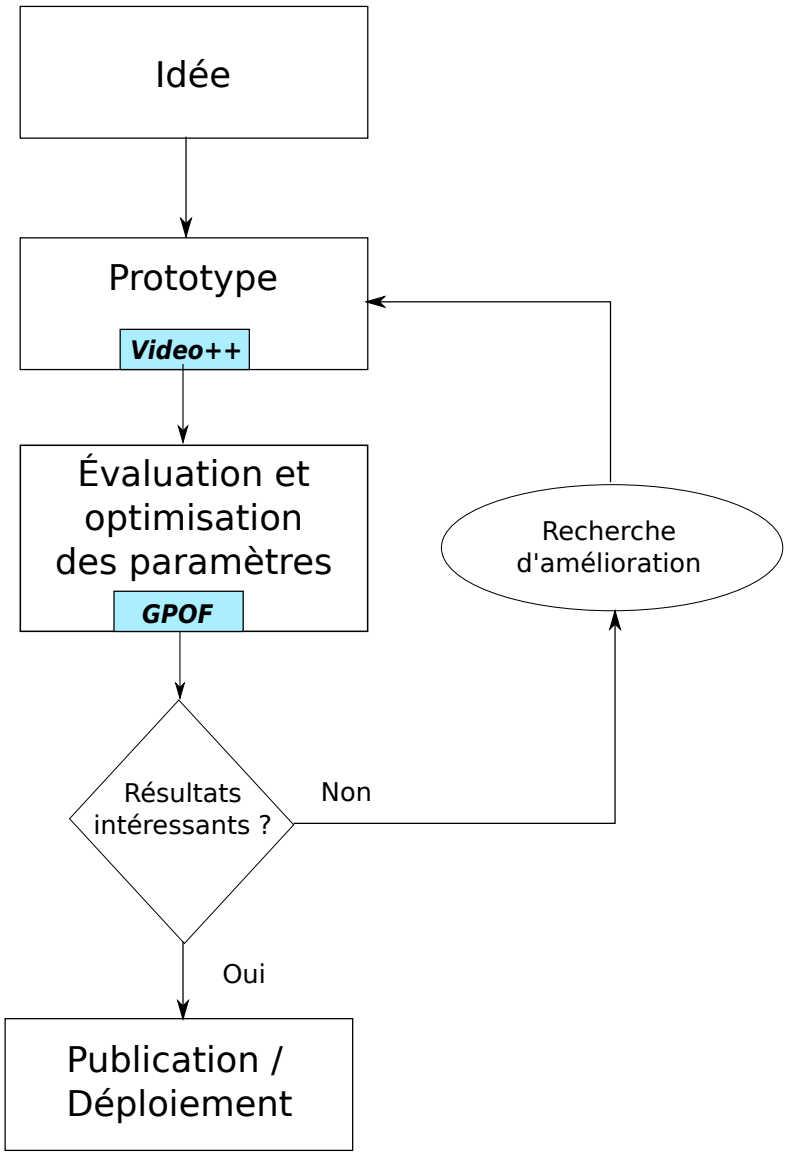

FIGURE 2.1 – Méthodologie suivie en général par un chercheur/développeur pour concevoir un algorithme innovant. Nous proposons dans ce chapitre *Video++* et *GPOF* pour simplifier respectivement la conception de prototype et l'évaluation d'un algorithme.

2.2 Création de Vidéo++, une bibliothèque C++ pour le prototypage rapide d'algorithmes sans compromis sur le temps de calcul

Depuis le siècle dernier, les scientifiques ont utilisé des ordinateurs pour automatiser le traitement des images numériques dans de nombreux domaines tels que l'imagerie médicale, l'imagerie par satellite, la surveillance vidéo, la photographie et d'autres. Les développeurs d'applications ont toujours fait face à un défi majeur : comment traiter une très grande quantité de données dans la plus courte période de temps, avec des ressources de calcul limitées ? Aujourd'hui, un processeur de $2cm^2$ exécute 10 000 fois plus d'instructions par seconde qu'ENIAC, le plus grand super ordinateur de l'année 1946. Cette miniaturisation des processeurs a permis aux scientifiques d'effectuer de plus en plus de calculs en un espace temps donné.

Jusqu'aux années 2000/2010, la principale progression entre deux générations de processeur était l'augmentation de la fréquence d'horloge. Elle permettait aux mêmes programmes de s'exécuter en moins de temps, sans besoin de faire évoluer le logiciel. Cela a fonctionné jusqu'au jour ou la chaleur dégagée par les commutations de transistors a empêché les fabricants de continuer ainsi.

C'est pourquoi, depuis une dizaine d'années, deux formes de parallélisme sont apparues dans les processeurs du commerce :

— **Le multi-cœur :** plusieurs cœurs sont placés sur une puce pour exécuter plusieurs instructions simultanément,
— **Le calcul vectoriel :** des unités de calcul permettent désormais d'exécuter simultanément la même opération sur plusieurs valeurs.

Contrairement à l'augmentation de fréquence d'horloge, tirer parti de ces évolutions nécessite une écriture différente des programmes : il faut désormais passer d'une écriture séquentielle à une écriture parallèle des codes applicatifs. Cela implique des efforts au niveau algorithmique : les problèmes doivent être séparés en sous-problèmes indépendants les uns des autres, et au niveau logiciel : l'utilisation de multiples threads pour le multi-cœurs et d'instructions SIMD pour le calcul vectoriel.

Pour contrer cette complexité à deux niveaux, plusieurs outils ont émergé. Des langages comme C++ ont intégré la notion de thread, les compilateurs ont introduit la vectorisation automatique et les directives OpenMP ont été créées pour permettre l'annotation de code parallélisable.

En plus des outils de parallélisation, le comité de standardisation de C++ a récemment publié deux nouveaux standards : C++11 et C++14. Ces deux documents sont un ensemble d'évolutions du langage et de la bibliothèque standard comblant les manques qui s'étaient avérés critiques depuis la publication du premier standard en 1998.

Ces avancées de langage et compilateurs vont nous permettre de construire des abstractions ayant pour but d'aider le programmeur à écrire des briques de traitement d'image exploitant toute la puissance de calcul des processeurs parallèles, sans pour autant avoir à écrire du code verbeux et source de bogues.

L'idée d'abstraire des concepts pour simplifier l'écriture de code n'est pas nouvelle. Dès la création de C++, les développeurs ont exploité les templates pour écrire du code à la fois générique et performant. Plus particulièrement en traitement d'image, où les algorithmes sont appliqués sur des types de données qui peuvent différer selon le capteur source ou le format de stockage.

Le framework de traitement d'image le plus utilisé actuellement est la bibliothèque OpenCV [8]. Initialement écrite en C, elle a été ensuite étendue vers C++ pour pouvoir tirer parti des templates. Son but est de fournir un ensemble très large de primitives de visions par ordinateur, couvrant plusieurs champs d'applications. En revanche, elle fournit peu d'outils aidant à l'écriture d'algorithmes et le fait qu'elle soit originaire du langage C l'empêche de tirer pleinement avantage de C++.

Olena [11] est une plateforme contenant Milena, une bibliothèque de traitement d'image C++. Elle repoussa les limites de la généricité en offrant des outils pour écrire simplement des algorithmes fonctionnant sur un large échantillon de types d'images comme les images 2D et 3D classiques, les graphes ou encore les images *run-lenght encoded* (RLE). Pour cela, Milena abstrait l'itération sur un ensemble (de pixels ou de coordonnées) via un objet itérateur. Bien que cette abstraction simplifie l'écriture d'algorithmes, elle est intrinsèquement séquentielle et ne permet pas l'exploitation du parallélisme des architectures de processeurs actuels.

D'autres frameworks C++ existent, comme VIGRA [35], CIMG [60] et ITK [32] fournissant des abstractions et des algorithmes. Cependant, étant tous conçus avec C/C++98, ils ne tirent pas ou peu parti des avancées récentes du langage et des compilateurs et cela rend leur utilisation verbeuse ou source de bogues.

Une approche plus récente, nommée Halide [52], a pour but de simplifier l'écriture et l'optimisation de chaînes de traitement d'image. En plus d'utiliser le parallélisme du processeur pour effectuer plusieurs calculs par cycle d'horloge, Halide fournit des outils pour optimiser facilement les schémas d'accès mémoire et maximiser l'utilisation du cache processeur. Les performances des codes générés surpassent les performances des chaînes optimisées à la main pour des temps de développements très inférieurs aux langages C/C++ purs. Toutefois, pour offrir de telles optimisations, ce modèle de programmation a été restreint aux fonctions transformant chaque coordonnée en une valeur. Il ne permet pas, par exemple, la transformation d'une image en une liste de points d'intérêt, ou encore les accès mémoire dépendant des données.

Tous ces frameworks ont été créés avec C++98 et ne tirent pas parti des nouveaux standards. C'est pourquoi nous avons proposé un nouveau framework, appelé *Vidéo++* tirant parti des nouveautés apportées par C++11/14, OpenMP et des versions récentes des compilateurs GCC et CLANG. Ce chapitre présente dans un premier temps les opportunités offertes par C++14 et les compilateurs, la nouvelle bibliothèque *Vidéo++* tout en évaluant ses performances.

2.2.1 Opportunités offertes par C++14 et les compilateurs

Les derniers standards C++11 et C++14 apportent un ensemble de nouveautés dans le langage C++ et offrent l'opportunité d'écrire des frameworks de traitement d'image plus génériques et plus performants. Cette section présente les nouveautés sur lesquelles nous nous sommes appuyés pour implanter ce nouveau framework et comment ces nouvelles fonctionnalités ont contribué à la simplicité et la performance de la bibliothèque *Video++*.

Les lambda fonctions (C++11) et les lambda fonctions génériques (C++14)

Le comité de normalisation du standard C++ a introduit avec C++11 le concept de lambda fonction [61]. Il permet de définir des fonctions anonymes traitées comme des variables non mutables. Une des premières applications des lambda fonctions est le passage simplifié de fonctions aux algorithmes de la bibliothèque standard (STL):

Listing 2.1 – Appel à la fonction std::sort utilisant une lambda fonction pour définir l'ordre de tri.

```
1
2 std::vector<int> v = {4,53,7,42,67,65};
3 std::sort(v.begin(), v.end(),
4          [] (int a, int b) { return a > b; });
```

Ces lambda fonctions sont un pur sucre syntaxique permettant de définir à la volée un objet fonction anonyme. En C++98, l'utilisateur était contraint à déclarer manuellement le type de cet objet en dehors de la fonction, ce qui résultait en un code verbeux et plus difficile à maintenir :

Listing 2.2 – Appel C++98 à la fonction std::sort utilisant un foncteur pour définir l'ordre de tri.

```
1 struct sup_ordering
2 {
3   bool operator()(int a, int b) const
4   {
5     return a > b;
6   }
7 };
8
9 std::vector<int> v = {4,53,7,42,67,65};
10 std::sort(v.begin(), v.end(), sup_ordering());
```

De plus, les instructions des lambda fonctions ont la possibilité de capturer des variables. Cette capture se fait soit par valeur (préfixe =), soit par référence (préfixe &) :

Listing 2.3 – Capture des variables de l'environnement par copie et par référence.

```
int X = 1;
int Y = 2;
auto fun = [&X, =Y] (int a, int b)
{
X = 4; Y = 4;
return X + Y + a + b;
};
int res = fun(1,2);
// res == 11
// X == 4
// Y == 2
```

C++14 étend les lambda fonctions en leur permettant de prendre en argument des paramètres à type générique :

Listing 2.4 – Opérateur d'addition générique.

```
auto plus = [](auto a, auto b){ return a + b; };
// plus(1, 2) == 3
// plus(std::string("a"),
// std::string("b")) == "ab"
```

Grâce aux lambda fonctions, il est désormais simple de créer des abstractions qui appliquent une fonction à tous les éléments d'un conteneur. Par exemple, dans *Video++*, cela permet d'appliquer une fonction à tous les pixels d'une image.

Listing 2.5 – Implantation de map, l'application d'une fonction sur les éléments d'un vecteur de la STL.

```
template <typename T, typename F>
void map(std::vector<T>& v, F f)
{
  for (int i = 0; i < v.size(); i++)
    f(v[i]);
}

```

```
8 std::vector<int> v = {1,2,3,4};
9 map(v, [](int& a) { a += 1; });
10 // v == 2,3,4,5;
```

Les templates variadiques

Introduits par C++11, les templates variadiques [14] prennent un nombre variable de paramètres. Ce concept s'applique aux fonctions templates et aux classes templates.

Listing 2.6 – Une fonction prenant une liste d'arguments de taille variable.

```
1 template <typename T, typename ...Tail>
2 void print() {} // Cas de la liste d'arguments vide.
3
4 // Affichage d'un nombre variable d'arguments.
5 template <typename T, typename ...Tail>
6 void print(T head, Tail... tail)
7 {
8   // Traitement de l'argument head.
9   std::cout << head;
10  // Traitement du reste de la liste d'arguments.
11  print(tail...);
12 }
```

Video++ utilise les templates variadiques pour construire des abstractions prenant un nombre variable d'arguments. Par exemple `pixel_wise` (sec. 2.2.2), le mapping d'une fonction sur les pixels d'une ou plusieurs images.

L'avantage de cette construction est que l'itération sur les paramètres d'une fonction variadique est réalisée à la compilation et n'a donc aucun impact sur le temps de calcul.

Les boucles *for* basées sur des plages

Le standard C++11 ajoute aussi la possibilité d'itérer sur des plages (*range-based for loops* [27]). C'est un pur raccourci de la boucle for de C++98 :

```
1 std::vector<int> v = {1,2,3,4};
2
```

```
// C++98
for(std::vector<int>::iterator& it = v.begin();
    it != v.end(); it++)
  *it += 1;

// C++11
for(auto& e: v)
  e += 1;
```

Comme les templates variadiques, cette fonctionnalité a pour but la simplification de l'écriture de code C++ et n'a aucun coût à l'exécution. Dans *Video++* nous utilisons ce nouveau type de boucle pour itérer sur des ensembles de valeurs comme les domaines des images. Il faut noter cependant que l'utilisation d'itérateurs sur plages peut masquer la contiguïté en mémoire des itérations successives, empêchant les compilateurs d'optimiser la parallélisation de l'exécution.

Vectorisation automatique, OpenMP et Cilk™Plus

Dans le but d'accélérer l'exécution de boucles parallélisables, les compilateurs actuels implantent une passe de vectorisation automatique de boucles. Cette passe d'optimisation détecte les boucles sans dépendances inter-itérations et les traduit en instructions vectorielles : SSE ou AVX pour x86, NEON pour ARM.

Listing 2.7 – Une boucle vectorisée automatiquement. Code extrait de la documentation du compilateur GCC [24]

```
int a[256], b[256], c[256];
foo () {
  int i;

  for (i=0; i<256; i++){
    a[i] = b[i] + c[i];
  }
}
```

Cette approche a l'avantage de ne pas être intrusive : le code n'a besoin

d'aucune modification pour être optimisé. Elle a cependant deux limitations : d'une part, elle n'exploite que les unités de calcul vectoriel, le premier niveau de parallélisation des processeurs actuels. Elle ne permet pas l'exploitation des multiples cœurs des processeurs actuels qui offre également un facteur d'accélération significatif. D'autre part, démontrer automatiquement l'absence de dépendances inter-itérations n'est possible que pour un ensemble de boucles restreint, limitant les possibilités de vectorisation.

Dans le but de pallier ces limitations, les outils OpenMP et Cilk™proposent au programmeur d'annoter le code. Ces annotations fournissent au compilateur des indications sur la stratégie de parallélisation à utiliser.

Listing 2.8 – OpenMP and Cilk™Plus

```
1 // Addition parallèle de deux tableaux avec OpenMP 4
2 #pragma omp parallel simd for
3 for(int i = 0; i < 10000; i++)
4 a[i] = b[i] + c[i];
5
6 // Avec Cilkplus
7 a[0:10000] = b[0:10000] + c[0:10000]
```

Video++ repose entre autres sur OpenMP4 pour accélérer l'application d'une fonction sur tous les pixels de l'image.

Veuillez noter que d'autres outils existent pour la parallélisation de code sur processeurs graphiques (GPU), mais nous avons limité le cadre de cette étude à l'accélération sur CPU.

Conclusion

Nous avons montré dans cette section comment le langage C++ a gagné en expressivité avec l'arrivée des standards C++11 et C++14, et comment ces nouveautés vont nous permettre de simplifier l'écriture et l'utilisation d'une bibliothèque de traitement d'image performante.

2.2.2 La bibliothèque *Video++*

Les images multi-dimensionnelles

Le type `imageNd` est l'élément central d'une bibliothèque de traitement d'image : c'est le premier type instancié par l'utilisateur final, et la grande majorité des algorithmes reposent sur son interface pour accéder simplement aux pixels d'une image à N dimensions. Comme dans la bibliothèque Milena [11], il a pour but de simplifier la manipulation d'images, mais aussi de simplifier l'écriture d'algorithmes accédant aux pixels. Notez que l'interface d'`imageNd` ne tire que peu parti du nouveau C++, et qu'elle est fortement inspirée de la bibliothèque Milena [11].

Le type `imageNd` fournit différentes fonctionnalités :

— **La gestion mémoire** : la classe `image` est responsable d'allouer et de désallouer le tableau de pixels, et cela de façon transparente pour l'utilisateur. Le comptage de références fourni par **std::shared_ptr** [12] est utilisé pour assurer la libération de la mémoire d'un tableau de pixels quand aucune instance du type image ne le référence.

— **L'alignement mémoire du début des lignes** : certaines opérations, comme les lectures et écritures dans la mémoire vive, sont plus rapides quand l'adresse de la zone mémoire en question est un multiple de 128 ou 256. Dans la suite, nous notons ces zones comme alignées. L'alignement optimal dépend du jeu d'instructions vectorielles ciblé. Pour tirer parti de cette opportunité, la classe image alloue le tableau de pixels de façon à ce que l'adresse du premier pixel de chaque ligne soit alignée de façon optimale.

— **L'accès aux pixels** : via la surcharge de l'opérateur parenthèse, une instance du type image se comporte comme une fonction transformant une coordonnée en la valeur du pixel correspondante.

— **Bordure en mémoire** : beaucoup d'algorithmes de traitement d'image utilisent des accès au voisinage d'un pixel. La classe image fournit une bordure en mémoire optionnelle qui permet à ces algorithmes de traiter les pixels de bordure de la même manière que les pixels centraux, sans provoquer de débordement de tableau ni générer des tests additionnels.

— **Copie légère** : pour éviter les copies de grands tableaux inopportunes,

l'opérateur égal et le constructeur par copie se limitent à la copie du pointeur de donnée de l'image. Seule la fonction `clone` permet de dupliquer une image en mémoire.

```
{
// Alloue une image de 100 (lignes) x 200 (colonnes) entiers.
image2d<int> img(100, 200);

// Initialise une image avec une valeur.
fill(img, 0);

// Accède à la valeur du premier pixel de l'image.
int pixel0 = img(0,0);

// Assigne les données de l'image img à img2.
image2d<int> img2 = img;

// img2 and img partagent maintenant le même tableau de pixels.
img2(0,0) = 42;
// img(0,0) == img2(0,0)

// Clone une image.
image2d<int> img3 = clone(img);
// img3 possède ses propres données.

// Alloue une image avec une bordure de 2 pixels.
image2d<int> img4(100, 200, _border = 2);
}
// Les trois images allouées sont libérées automatiquement à la fin
   du scope.
```

Mapping parallèle d'une fonction sur une image

En traitement d'image, beaucoup d'algorithmes consistent en l'application d'une fonction à l'ensemble de pixels contenu dans une image à 2 dimensions. Une des solutions est d'utiliser une double boucle *for*, annotée de directives OpenMP pour la parallélisation à la fois vectorielle et multi-thread.

```
// Parallélisation multi-thread / gros grain.
#pragma omp parallel for
for (int r = 0; r < image.nrows(); r++)
{
  // Parallélisation vectorielle / grain fin.
  #pragma omp simd
  for (int c = 0; c < image.ncols(); c++)
    A[r][c] = B[r][c] + C[r][c]
}
```

Bien que très performante, cette approche souffre de plusieurs problèmes :

— l'utilisation de deux variables, r et c, pour référencer un pixel est source de bugs : Un programmeur peut facilement intervertir les deux variables.

— la syntaxe de la double boucle for et ses annotations OpenMP est longue à écrire.

Avec C++98, certains frameworks comme Olena [11] résolvent ces problèmes en abstrayant l'itération sur un domaine 2D, et en passant les 2 coordonnées via un itérateur pour résoudre l'ambiguïté ligne/colonne.

```
mln_piter(A_type) p(A.domain());
for_all(p)
  A(p) = B(p) + C(p);
```

Cependant, le deuxième problème persiste : l'utilisation de l'abstraction itérateur a masqué la nature contiguë et massivement parallèle de l'addition pixel à pixel de deux images.

Grâce aux lambda fonctions et aux templates variadiques, la solution proposée dans la bibliothèque *Vidéo++* est à la fois parallèle et concise. Elle consiste en un patron d'algorithme générique sur lequel se branche la fonction à passer sur l'ensemble de pixels. L'exemple suivant montre une version simplifiée de cette abstraction, appelée `pixel_wise` et fonctionnant sur trois images :

Listing 2.9 – Abstraction de l'itération sur 3 images. Pour des raisons de simplicité, nous avons ignoré le fait que la bibliothèque ajoute un *padding*

d'alignement à la fin de chaque ligne

```
template <typename F>
void apply(image2d<int>& A, image2d<int>& B, image2d<int
    >& C, F f)
{
  int* a = &A(0,0);
  int* b = &B(0,0);
  int* c = &C(0,0);

  int* a_end = 1 + &A(A.nrows() - 1, A.ncols() - 1);

  while(a != a_end)
  {
    // Passage des trois pixels à la fonction de traitement.
    f(*a, *b, *c);

    // Passage au pixel suivant
    ++a; ++b; ++c;
  }
}

// Addition de deux images avec l'abstraction apply.
apply(A, B. C, [] (int& a, int b, int c) { a = b + c; });
```

`pixel_wise` est une version améliorée de `apply` car :

- en plus de générer des boucles vectorisables, elle peut tirer parti des multiples cœurs du processeur en distribuant le calcul des lignes de l'image sur plusieurs threads.
- grâce à l'utilisation les templates variadiques `pixel_wise` accepte un nombre arbitraire d'images.
- les algorithmes non parallèles sont supportés avec l'option `_no_thread`
- d'autres options définissent l'ordre de parcours de l'image.

L'extrait de code suivant montre l'utilisation de `pixel_wise` pour l'implantation d'une addition d'image.

Listing 2.10 – Addition de deux images avec `pixel_wise`

```
vpp::pixel_wise(A, B, C) | [] (int& a, int& b, int& c)
{
  a = b + c;
};
```

Les résultats présentés dans la section 2.2.3 démontre que le code généré via cette abstraction est équivalent à une double boucle optimisée avec OpenMP.

Notez que l'abstraction `pixel_wise` délègue aux compilateurs la parallélisation via les annotations OpenMP et la vectorisation automatique. Cela nous permet d'exploiter correctement des processeurs avec plus ou moins de cœurs et des jeux d'instructions vectorielles différents.

Simplification de l'accès au voisinage des pixels

L'abstraction `pixel_wise` présentée dans la section précédente permet l'implantation rapide d'opérations pixel à pixel. Cependant, elle n'est pas suffisante pour implanter des filtres nécessitant d'accéder au voisinage, tels que les convolutions.

C'est pourquoi nous avons proposé l'abstraction `relative_access` qui permet à une fonction passée à `pixel_wise` d'accéder aux pixels compris dans un voisinage rectangle. Et dans le but de permettre au compilateur de mieux optimiser le code, nous passons les dimensions du voisinage en paramètre template. Le listing 2.11 montre une implantation d'un filtre box de taille 5x5 tirant parti de l'abstraction `relative_access`.

Nous montrons dans la section 2.2.3 que cette abstraction équivaut à une version optimisée à la main.

Intéropérabilité avec OpenCV

Bien que les abstractions de *Video++* facilitent l'écriture d'algorithmes, la bibliothèque n'en contient que peu. Inversement, OpenCV contient beaucoup d'algorithmes mais peu d'outils facilitant l'écriture. Pour tirer parti de la complémentarité de ces deux bibliothèques, nous avons implanté un pont

Listing 2.11 – A 5x5 box filter

```
image2d<int> A(1000, 1000, _border = 2);
image2d<int> B(A.domain(), _border = 2);

vpp::pixel_wise(B, relative_access(Anbh)) |
[&] (int& b, auto a_nbh)
{
  int sum = 0;

  // Somme des pixels du voisinage 5x5.
  for (int i = -2; i <= 2; i++)
  for (int j = -2; j <= 2; j++)
    sum += a_nbh(i, j)

  // Écriture du résultat.
  b = sum / 25;
};
```

entre leurs types d'image : `from_opencv` convertit une image OpenCV en une image *Video++*, et inversement pour `to_opencv`.

Listing 2.12 – Chargement et écriture d'une image JPEG via un conteneur d'image *Video++*

```
image<vuchar3> img = from_opencv<vuchar3>
            (cv::imread("image.jpg"));

cv::imwrite("out.jpg", to_opencv(img));
```

Calcul Vectoriel

La bibliothèque Eigen3 fournit des types génériques de vecteur et matrice *type-safe* ainsi que des opérateurs d'algèbre linéaire. Pour permettre aux utilisateurs de représenter simplement des pixels à valeurs multidimensionnelles (par exemple, un vecteur RGB), nous avons défini des alias vers les types vecteurs les plus utilisés en traitement d'image.

Les types vecteurs sont notés `v{T}{N}` avec :

— T compris dans la liste: **char, short, int, float, double, uchar, ushort, uint**.

— N un entier compris entre 0 et 4.

Par exemple, `vint2`, `vuchar3`, `vfloat4` sont des types vecteurs valides et le type `image2d<vuchar4>` peut représenter une image RGBD 8-bit.

2.2.3 Evaluation des performances

En informatique, les abstractions permettent d'écrire du code de façon plus concise et moins sujette à erreurs. Cependant, le code résultant s'exécute souvent moins vite qu'une version équivalente optimisée à la main et n'utilisant aucune d'abstraction. Dans cette section, nous vérifions que les abstractions de *Video++* n'ont aucun impact sur le temps d'exécution. Pour cela, nous les comparons directement à leurs équivalents optimisés à la main. Nous comparons aussi les équivalents de la bibliothèque OpenCV. L'architecture utilisée est un processeur 4-cœurs, hyperthreadé et cadencé à 3.2GHz (Intel i7-4700HQ avec les extensions vectorielles AVX2). Tous les exécutables ont été compilés avec GCC 4.9.1 et les flags `-O3 -march=native -fopenmp`.

Addition de deux images

Pour vérifier que l'abstraction `pixel_wise` n'a vraiment aucun coût au niveau du temps de calcul, nous avons comparé trois implantations de l'addition de deux images : la première est la version naïve, itérant sur les lignes et les colonnes avec une double boucle *for* :

Listing 2.13 – Addition de deux images, version naïve.

```
1 for (int r = 0; r < A.nrows(); r++)
2 for (int c = 0; c < A.ncols(); c++)
3 {
4   A(r, c) = B(r, c) + C(r, c);
5 }
```

La deuxième (listing 2.14) repose sur `pixel_wise`.

Listing 2.14 – Addition de deux images, version `pixel_wise`.

```
pixel_wise(A, B, C) | [] (int& a, int b, int c)
{
  a = b + c;
};
```

La troisième, utilisée comme référence en termes de temps de calcul, récupère des pointeurs de début de ligne pour pouvoir écrire une boucle vectorisable automatiquement par le compilateur.

Listing 2.15 – Addition de deux images, version optimisée manuellement

```
int nr = A.nrows();
// Parallélisation du calcul des lignes en multi-thread.
#pragma omp parallel for
for (int r = 0; r < nr; r++)
{
// Pointeurs de début de ligne.
int* curA = &A(r, 0);
int* curB = &B(r, 0);
int* curC = &C(r, 0);
int* endA = curA + A.nrows();

int nc = A.ncols();

// Traitement d'une ligne SIMD
#pragma omp simd
for (int i = 0; i < nc; i++)
  curA[i] = curB[i] + curC[i];
}
```

Les graphiques 2.2 et 2.3 montrent trois résultats :

— l'abstraction `pixel_wise` est équivalente à la version optimisée manuellement en termes de temps de calcul.

— la version naïve est jusqu'à 26 fois plus lente.

— pour les images plus grandes que le cache L2, le gain offert par la parallélisation est limité par la bande passante mémoire.

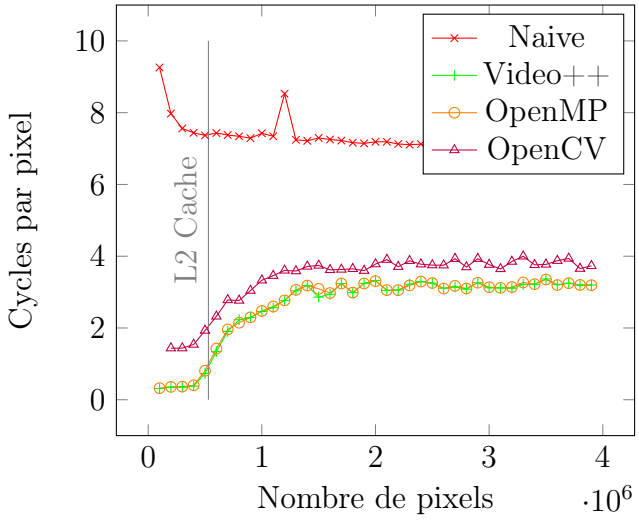

FIGURE 2.2 – Nombre de cycles par pixel pour les quatre additions d'images évaluées sur différentes tailles d'images. A gauche de la ligne verticale, le cache L2 de 6.2Mo contient entièrement les trois images utilisées.

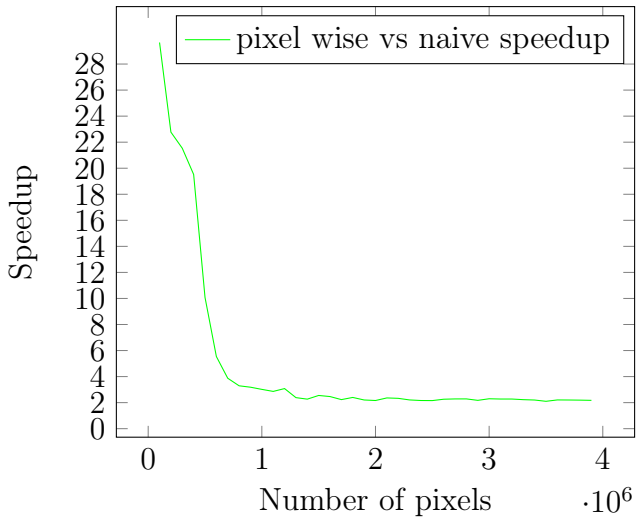

FIGURE 2.3 – Facteur d'accélération de `pixel_wise` par rapport à la version naïve de l'addition d'image.

Un filtre convolutionnel : le filtre moyenne

En utilisant le même protocole, nous avons évalué un second type de filtre : le filtre moyenne sur un voisinage 5×5. Le but de ce benchmark est de vérifier l'efficacité de l'abstraction voisinage permettant aux implantations basées sur `pixel_wise` d'accéder au voisinage. La version naïve accède aux pixels avec l'adressage 2D :

Listing 2.16 – Version naïve du filtre moyenne.

```
for (int r = 0; r < A.nrows(); r++)
for (int c = 0; c < A.ncols(); c++)
{
  int sum = 0;
  for (int d = -2; d <= 2; d++)
  for (int e = -2; e <= 2; e++)
    sum += B(r + d, c + e);
  A(r, c) = sum / 25;
}
```

La version la plus courte (listing 2.11) utilise les abstractions `relative_access` et `pixel_wise`. La version de référence (listing 2.17) optimisée à la main est basée sur OpenMP.

Comme dans le benchmark précédent, les versions OpenMP et Video++ parviennent aux mêmes temps d'exécution. La routine OpenCV `cv::boxFilter` est jusqu'à 5.6 fois plus lente car elle n'utilise qu'un seul cœur CPU. Le graphique 2.4 trace les temps de calcul des différentes versions pour plusieurs tailles d'images, et le graphique 2.5 montre le facteur d'accélération de la version `pixel_wise` par rapport à la version naïve.

Placement de *Video++* dans le domaine open source

Nous avons diffusé la bibliothèque *Video++* sous une licence permissive (MIT) via un dépôt GitHub [23]. Cela nous a permis de vérifier l'intérêt de la communauté pour ces abstractions : le dépôt Git reçoit une centaine de visiteurs par semaine et les sources sont téléchargées une fois tous les 4-5 jours.

Listing 2.17 – Version OpenMP manuelle du filtre moyenne

```
int nr = A.nrows();
#pragma omp parallel for
for (int r = 0; r < nr; r++)
{
  int* curA = &A(vint2(r, 0));
  int nc = A.ncols();

  int* rows[5];
  for (int i = -2; i <= 2; i++)
    rows[i + 2] = &B(vint2(r + i, 0));

  for (int i = 0; i < nc; i++)
  {
    int sum = 0;
    for (int d = -2; d <= 2; d++)
      for (int e = -2; e <= 2; e++)
        sum += rows[d + 2][i + e];
    curA[i] = sum / 25;
  }
}
```

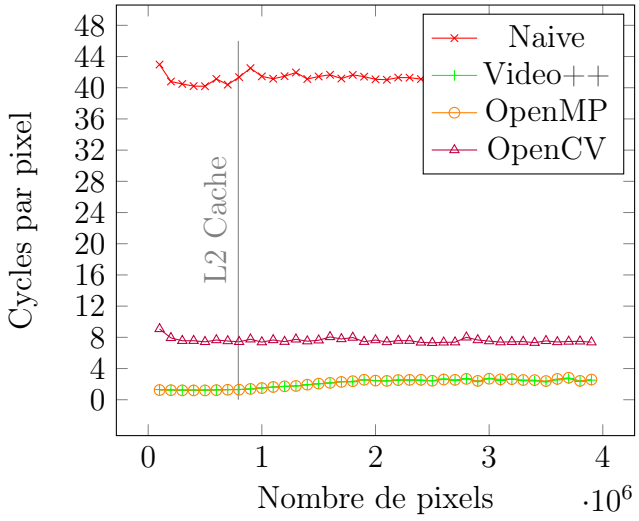

FIGURE 2.4 – Temps d'exécution des quatre implantations du filtre moyenne 5x5 en fonction de la taille de l'image en pixels. Les deux images sont entièrement contenues par le cache L2 (6.2Mo) à gauche du repère vertical.

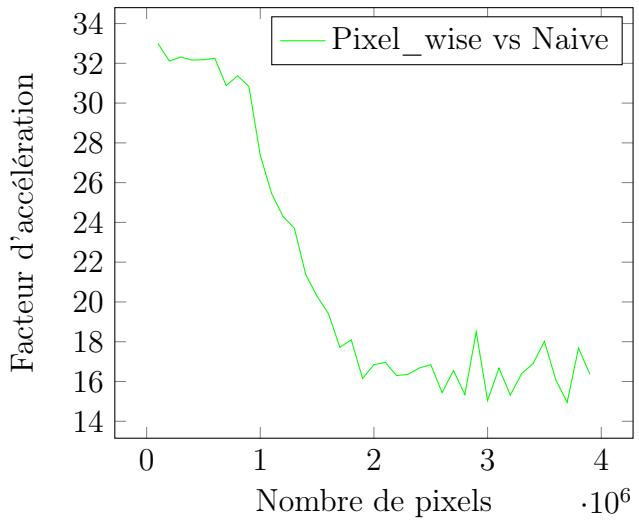

FIGURE 2.5 – Facteur d'accélération du filtre `pixel_wise` par rapport à la version naïve pour le filtre moyenne sur un voisinage 5x5.

2.2.4 Conclusions et perspectives

Dans cette étude, nous avons démontré comment les avancées récentes des compilateurs et du langage C++ ont permis de simplifier à la fois l'écriture et l'utilisation d'une bibliothèque de traitement d'image.

Nous avons conçu *Video++* pour fournir des abstractions permettant de faciliter et d'accélérer le développement sans impacter les temps de calcul : les applications écrites avec la bibliothèque sont moins complexes tout en étant aussi rapides que les versions optimisées, et jusqu'à 32 fois plus rapides que des versions naïves.

Bien que les processeurs graphiques (GPUs) ne soient pas gérés aujourd'hui, des abstractions pourront être ajoutées par la suite quand les compilateurs C++ implanteront des directives similaires à OpenMP ciblant les GPUs.

2.3 Aide à l'analyse de l'espace des paramètres d'un algorithme

2.3.1 Introduction

En traitement d'image, comme dans beaucoup d'autres domaines de l'informatique, les algorithmes prennent souvent un grand nombre de paramètres et leurs performances en dépendent significativement. Trouver un jeu de paramètres proche de l'optimal est souvent une tâche ardue car elle nécessite l'exploration et l'évaluation d'un grand nombre de jeux de paramètres ainsi que la génération de résultats, leur présentation, interprétation, etc.

Malgré l'existence de cette difficulté, et bien que de nombreux algorithmes aient été conçus pour minimiser une fonction de coût dans un espace multidimensionnel, aucun logiciel à notre connaissance ne permet d'automatiser ou de simplifier ce processus de recherche. En effet, l'automatisation de cette tâche quel que soit le contexte est très complexe, voire quasi-impossible car elle doit s'adapter à tous les facteurs pouvant varier d'un algorithme à l'autre :

— le langage de programmation
— la nature des paramètres et des résultats

— la manière dont l'espace de paramètres est exploré
— le protocole d'évaluation d'un jeu de paramètres
— le temps d'exécution d'une évaluation

L'absence d'un tel outil a deux principales conséquences : d'abord, une méthode prometteuse peut être sous-exploitée, faute d'avoir trouvé le jeu de paramètres optimal. Ensuite, l'exploration et l'évaluation des différentes variantes d'un algorithme peuvent être négligées car l'optimisation manuelle des paramètres est trop chronophage.

Pour pallier ce problème, nous avons proposé un ensemble d'outils automatisant les tâches élémentaires impliquées couramment dans les processus d'évaluation. Ces outils ont été implantés sous forme de bibliothèque Python. Ils sont sous licence libre et disponible sur GitHub [21]. Une première version a été publiée mais davantage de fonctionnalités pourront être ajoutées par la suite, comme le support d'autres langages, de nouvelles méthodes d'exploration, ou de nouveaux outils de visualisation.

La figure 2.6 offre une vue d'ensemble du système proposé. Son architecture s'articule autour des trois éléments suivants :

— **l'évaluation d'un jeu de paramètres :** Fournie par l'utilisateur, ce programme prend en entrée un jeu de paramètres et génère un ensemble de valeurs résultat.
— **l'exploration automatique :** Les fonctions d'exploration automatique appellent le programme d'évaluation sur un ensemble de jeu de paramètres. Ces primitives sont paramétrées et appelées par l'utilisateur via un script Python.
— **l'affichage :** Un ensemble de fonctions Python permet l'affichage des données générées par l'exploration automatique.

Nous parcourons dans cette section les primitives logicielles développées. D'abord, un format de fichier de configuration est présenté en section 2.3.2, ainsi que ses primitives C++ de lecture. L'enregistrement des résultats de chaque évaluation a aussi été simplifié via une routine C++ générique, présentée en section 2.3.3. Nous présentons ensuite l'automatisation de l'exploration de l'espace de paramètres avec en section 2.3.4 l'agrégation des résultats et en section 2.3.5 les deux modes d'exploration implémentés : l'exploration exhaustive d'une grille d'échantillonnage et la descente de gradient. Finale-

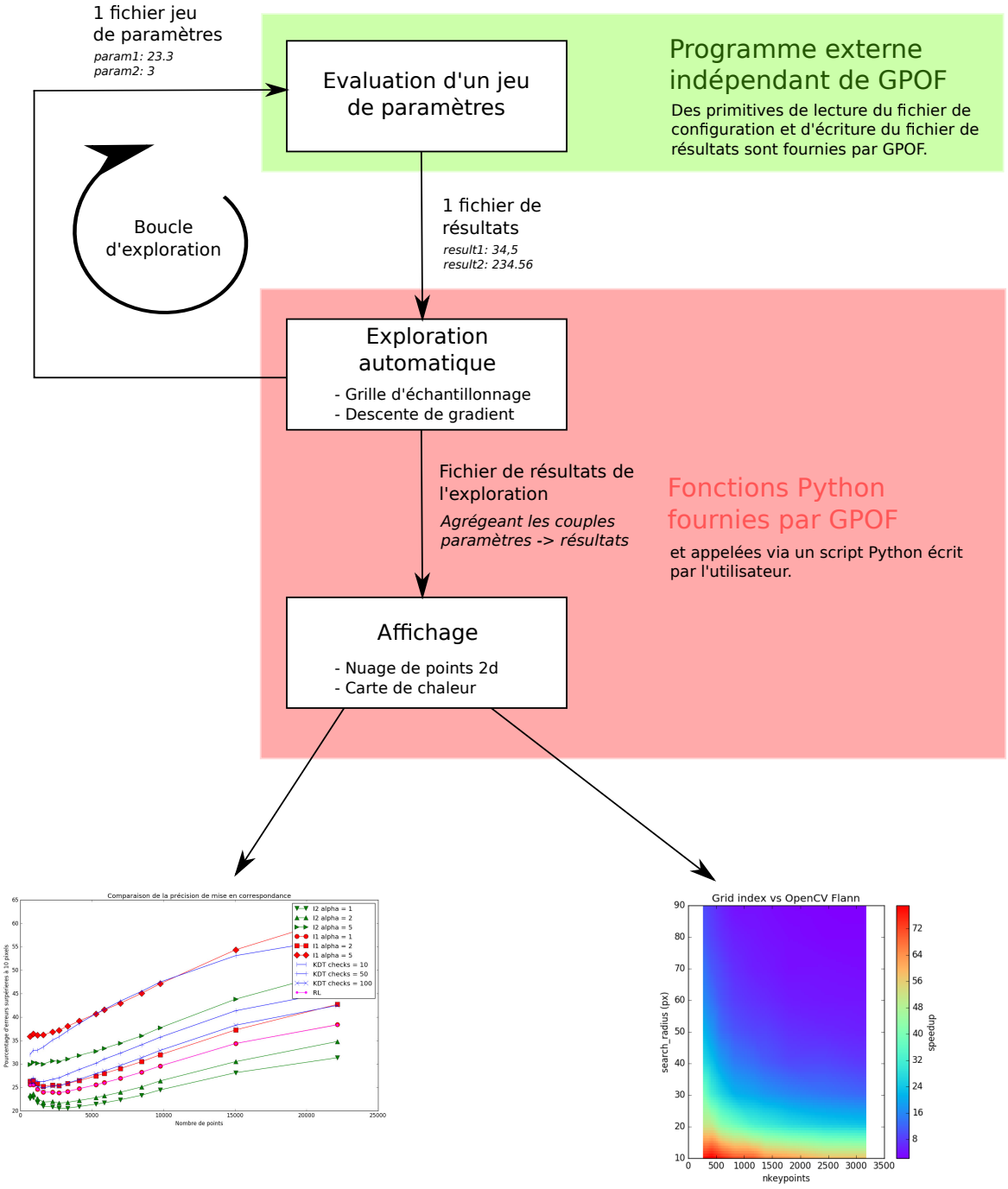

FIGURE 2.6 – Vue d'ensemble du projet GPOF.

ment, en section 2.3.6, nous présentons des primitives de visualisation basées sur Matplotlib [31] permettant l'affichage et l'interprétation des résultats.

2.3.2 Passage d'un ensemble de paramètres à un programme C++

Les paramètres d'un algorithme écrit en C++ sont passés via un fichier. La fonction `read_parameters` prend la liste des paramètres et de leurs types, comme l'illustre l'exemple suivant :

```
// Lecture du fichier de configuration.
// et spécification des valeurs par défaut de chaque paramètre.
auto conf = read_parameters("params.yml",
                            _param1 = int(4),
                            _param2 = float(0.5f));

// Passage des arguments à l'algorithme.
algorithm(conf.param1, conf.param2);
```

La fonction `read_parameters` repose sur les symboles de la bibliothèque Iod [22] que nous avons développée durant cette thèse. Les symboles permettent d'exprimer simplement les informations nécessaires à la lecture et écriture d'un fichier de paramètres comme le nom de chaque paramètre ainsi que son type.

Nous avons encodé le fichier de paramètres au format YAML [16] car c'est l'un des plus simples à éditer manuellement. L'exemple suivant illustre un fichier stockant deux paramètres :

```
param1: 23
param2: 34.23
```

Notons que le passage d'un fichier de configuration permet à l'exécutable contenant l'algorithme d'être utilisé de deux façons différentes : en l'éditant manuellement ou de façon automatique avec les routines d'exploration de l'espace des paramètres.

2.3.3 Enregistrement des résultats en C++

Chaque évaluation d'un jeu de paramètres génère un ensemble de résultats. Ils peuvent être un ensemble de scalaires, une courbe, ou une autre donnée représentable au format texte. Le format du fichier résultat est identique à celui du fichier de jeu de paramètres.

Ces résultats sont enregistrés dans un fichier texte au format YAML via la fonction `write_results` comme l'illustre l'exemple suivant.

```
write_results("results.yml",
              _result1 = 123,
              _result2 = "foo");
```

2.3.4 Spécification d'une évaluation et agrégation des résultats

Les primitives d'exploration présentées en section 2.3.5 évaluent un ensemble de jeu de paramètres. Pour pouvoir par la suite analyser les résultats de ces évaluations, il est nécessaire d'enregistrer chaque jeu de résultats obtenu et son jeu de paramètres associé.

Pour cela, nous proposons la fonction Python `command_runner` prenant en argument le chemin du fichier dans lequel les résultats seront agrégés et la ligne de commande à exécuter pour lancer une évaluation. Cette dernière devra contenir les marqueurs `%config_file` et `%result_file` qui seront remplacés respectivement par l'emplacement du fichier de paramètres (section 2.3.2) et le fichier de résultats à remplir (section 2.3.3).

L'objet retourné par la fonction `command_runner` est un objet de type `RunSet` stockant la ligne de commande et le chemin du fichier destinés à l'agrégation des résultats.

L'exemple suivant illustre un appel à `command_runner` :

```
runset = command_runner("results.rs",
            "./evaluation %config_file %result_file");
```

Notons que la classe `RunSet` écrit les résultats au cours de l'exploration, laquelle peut donc être interrompue sans perdre les résultats des jeux de

paramètres déjà évalués. De plus, si le fichier contient déjà les résultats d'évaluations précédentes, les jeux de paramètres déjà explorés seront ignorés par toute nouvelle exploration. Cela permet au développeur de visualiser les résultats avant la fin d'une exploration et éventuellement de l'interrompre pour la relancer avec une configuration d'exploration différente.

2.3.5 Exploration automatique de l'espace des paramètres

L'exploration de l'espace de paramètres est souvent une étape longue et hasardeuse du développement d'un algorithme. En effet, il est presque toujours trop long d'explorer l'intégralité de l'espace. Bien que le chercheur ait souvent une intuition sur un jeu de paramètres qui pourrait mener à de bons résultats, il est probable que ce jeu soit largement sous-optimal.

Pour automatiser la recherche d'un jeu de paramètres proche de l'optimalité, nous pourrions implémenter un des algorithmes de l'état de l'art permettant de minimiser une fonction de coût dans un espace multidimensionnel de façon complètement automatique. Cependant, la réalisation d'un logiciel capable de réaliser cette tâche dans un temps raisonnable est impossible pour les algorithmes ayant un nombre élevé de paramètres ou un temps d'évaluation très long.

C'est pourquoi nous avons proposé une solution semi-automatique, qui assiste le développeur dans sa recherche plutôt que de le remplacer. Pour cela, nous avons proposé des fonctions Python automatisant les tâches simples comme l'exploration d'une partie de l'espace et la descente de gradient à partir d'un jeu de paramètres initial.

Lors d'une exploration et pour chaque jeu de paramètres à évaluer, un fichier est généré et passé à l'évaluation via la ligne de commande passée à `command_runner` (section 2.3.4). Chaque évaluation génère un fichier de résultats qui est enregistré dans le fichier passé en premier argument de `command_runner` (section 2.3.4).

Exploration par échantillonnage

Ce mode d'exploration consiste à évaluer tous les jeux de paramètres présents dans un pavé discret de l'espace défini via la syntaxe illustrée par le listing 2.18.

Listing 2.18 – Définition d'un pavé discret à explorer

```
e={
  'param1': numpy.arange(0,5,1), # [0,1,2,3,4]
  'param2': 12,
  'param3': [2.3, 4.1, 5.6],
}
grid_sampling(runset, e);
```

Pour chaque paramètre, l'ensemble de valeurs à explorer est spécifié par une liste Python, ou un simple scalaire. Ainsi, le listing 2.18 définit un échantillonnage de 15 jeux de paramètres et la fonction `grid_sampling` permet son évaluation.

Exploration par descente de gradient

Bien que la méthode par échantillonnage présentée dans la section précédente permette de connaître le comportement d'un algorithme sur un ensemble de jeux de paramètres, elle est trop coûteuse quand le nombre de dimensions à explorer ou le temps pris par une évaluation sont grands. Dans ces cas problématiques, il est alors nécessaire de converger en minimisant le nombre de jeux de paramètres évalués.

Pour cela, nous avons proposé via la fonction `gradient_descent` une descente de gradient simple permettant de trouver un jeu de paramètres localement optimal à partir d'un jeu initial.

Le listing 2.19 illustre la définition de la configuration de la descente de gradient ainsi que son exécution.

2.3.6 Visualisation des résultats

Une fois un ensemble de jeux de paramètres évalué, il est souvent nécessaire d'interpréter les résultats, par exemple pour vérifier des intuitions ou analyser l'impact d'un paramètre.

Bien qu'en Python, la bibliothèque Matplotlib [31] permette la visualisation de données, les possibilités qu'elle propose sont très nombreuses et cela complexifie son utilisation. Nous avons donc fourni quelques fonctions

Listing 2.19 – Configuration de la descente de gradient.

```
conf=GradientDescentConfig(
    # Nombre maximum d'itérations
    max_iterations = 100,
    # Point de départ de la descente
    starting_point = {
        'param1': 1
        'param2': 12,
        'param3': 4.1,
    },
    # Echantillonnage de l'espace à analyser.
    parameters = {
        'param1': numpy.arange(0,5, 1), # [0,1,2,3,4]
        'param2': 12,
        'param3': [2.3, 4.1, 5.6],
    },
    # Fonction à minimiser. Le paramètre r expose
    # les résultats d'une évaluation.
    cost = lambda r: r['result2']
)

# exécution de la descente de gradient.
gradient_descent(runset, conf);
```

permettant le tracé de courbes en un seul appel de fonction.

Jusqu'à maintenant, seules deux fonctions ont été implémentées mais la bibliothèque est destinée à couvrir plus de cas d'utilisations. Elles prennent en argument des vues construites à partir du fichier *runset* résultant de l'exploration automatique.

Extraction d'une vue

Pour visualiser des données agrégées (section 2.3.4) lors de l'exploration, il est nécessaire d'en extraire des vues. Une vue est définie par l'ensemble des dimensions à analyser et par un filtre optionnel. Une dimension peut être soit un paramètre, soit un résultat.

L'exemple suivant construit une vue permettant l'affichage des variations de `result3` en fonction du paramètre `param2`. Les jeux de paramètres dont la composante `param1` est inférieure à deux on été ignorés grâce à un filtre. Les données de l'exploration sont récupérées en appelant la fonction `open_runset` avec le chemin du fichier *runset* généré lors de l'exploration.

```
runset = open_runset("exploration.rs")
view = runset.view(('param2', 'result3'),
                   filter = lambda x : x['param1'] <= 2)
```

Il est aussi possible d'exprimer des dimensions calculées via une lambda fonction python prenant en argument l'objet contenant les paramètres et résultats d'une évaluation :

```
view = runset.view((
        'param2',
        lambda x : x['result3'] / x['result2']
))
```

Une fois extraites, ces vues peuvent être affichées par les fonctions `display_2d_points` et `display_2d_heatmap` présentées dans la suite.

Affichage d'un nuage de points 2D

L'affichage le plus élémentaire, le nuage de points 2D, est implémenté par la fonction `display_2d_points`.

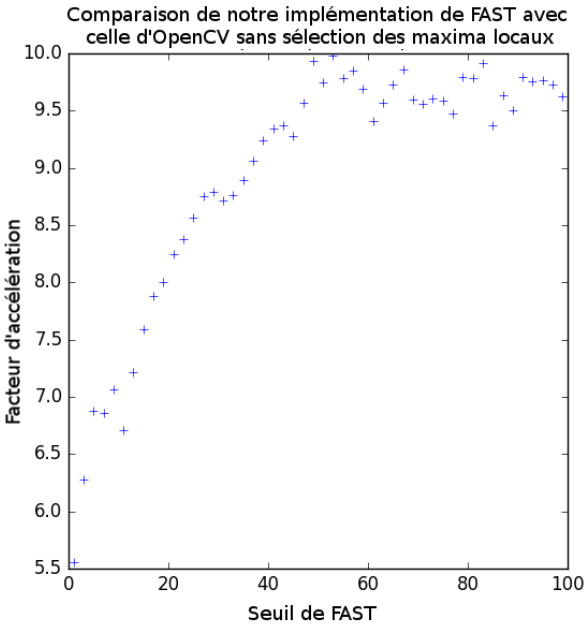

FIGURE 2.7 – Exemple de visualisation d'une vue 2D.

Le listing 2.20 donne l'exemple de la visualisation du temps de calcul d'un détecteur de points en fonction du nombre de points extraits. La figure 2.7 montre un graphique généré avec cette fonction lors d'une étude sur le détecteur de points d'intérêt FAST (voir section 3.3.2).

Listing 2.20 – Affichage d'un nuage de points 2D.

```
1 runset = open_runset("exploration.rs")
2 display_2d_points(runset.view(('result1', 'param1')))
```

Affichage de cartes de chaleur

L'affichage de trois dimensions a aussi été simplifié via la fonction `display_2d_heatmap`. Elle prend en argument une vue à trois dimensions et génère une carte de chaleur. La carte est colorée en fonction de la troisième dimension de la vue. Pour un affichage dense, les couleurs ont été interpolées entre les points.

L'exemple 2.21 illustre un appel à la fonction `display_2d_heatmap` et l'image 2.8 une carte de chaleur générée lors de l'étude d'une odométrie visuelle.

Listing 2.21 – Affichage d'une carte de chaleur.

```
runset = open_runset("exploration.rs")
display_2d_heatmap(
    runset.view(('param1', 'param2',
                 'result1'))
)
```

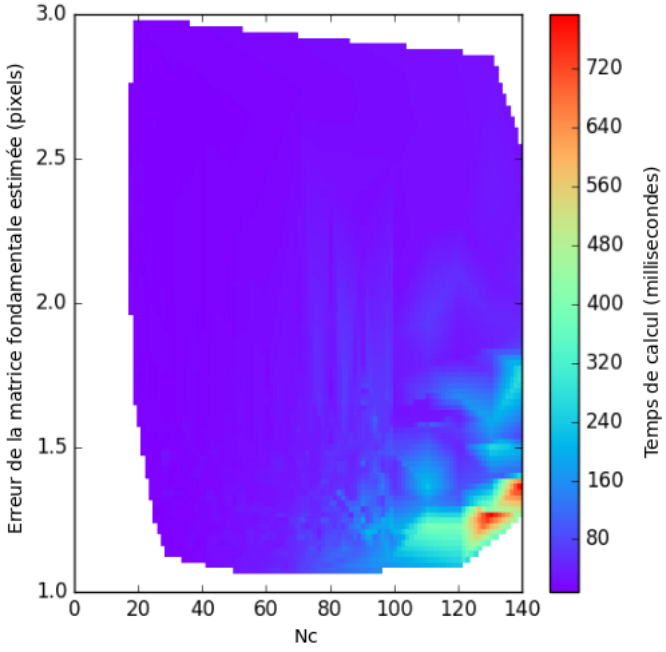

FIGURE 2.8 – Exemple de visualisation d'une vue 3D.

2.3.7 Conclusion et perspectives

Nous avons proposé dans ce travail un ensemble de briques logicielles facilitant la recherche de jeux de paramètres maximisant un objectif. Nous avons pu valider leur efficacité et leur applicabilité au domaine du traitement d'image et du langage C++ en les utilisant directement dans les travaux décrits dans la suite de cette thèse. Ils nous ont en effet permis non seulement d'optimiser les paramètres de nos algorithmes, mais aussi d'évaluer rapidement plusieurs implémentations d'un algorithme pour en sélectionner la meilleure.

Ce projet pourra à l'avenir être étendu par le support d'autres langages et de nouveaux algorithmes d'exploration ; le temps d'exploration pourra être diminué en parallélisant les évaluations sur les machines d'un cluster de calcul.

Pour cela, nous avons ouvert la collaboration à la communauté via la publication du code sur un dépot GitHub [21].

2.4 Conclusion du chapitre

Nous avons présenté dans ce chapitre deux nouveaux outils aidant à la conception d'algorithmes de traitement d'image haute performance.

Le premier, *Video++* a permis de simplifier l'écriture d'applications de traitement d'image sans compromettre le temps de calcul. Pour cela, nous avons utilisé les nouvelles fonctionnalités proposées par les derniers standards C++11 et C++14.

Le deuxième, *GPOF*, automatise partiellement l'exploration de l'espace de paramètres des algorithmes grâce à un ensemble de fonctions Python et C++. La généricité de cet outil nous a permis de l'appliquer à différents travaux : L'augmentation de la précision et la diminution du temps de calcul de l'analyse de mouvement, l'odométrie visuelle et la détection de cibles mobiles.

Bien que les bases de *Video++* et *GPOF* aient atteint une version stable, ces outils ont été mis en open source pour permettre à la communauté d'en étendre les fonctionnalités. Le premier pourra bénéficier par exemple d'implémentation de nouveaux algorithmes et le deuxième de la parallélisation sur plusieurs serveurs de calcul.

Chapitre 3

Estimation du mouvement pour les applications embarquées

Sommaire

3.1 Introduction

Durant ces dernières années, nous avons constaté une augmentation du nombre d'applications reposant sur l'estimation du mouvement apparent pour extraire des informations sur une scène pointée par une caméra. Cependant, la complexité de calcul des algorithmes et la grande quantité de données à traiter limitent leur fréquence de traitement. Leurs concepteurs se retrouvent donc souvent contraints soit à traiter les vidéos après acquisition, soit à utiliser des processeurs puissants et à haute consommation énergétique pour les traiter en temps réel.

Afin de permettre aux applications aux ressources de calcul limitées de bénéficier d'une estimation de mouvement précise, l'état de l'art propose souvent de réduire la complexité algorithmique, laquelle correspond au nombre d'opérations élémentaires, quelle que soit leur nature. Bien que pertinente, cette mesure ne prend pas en compte tous les facteurs impactant le temps de calcul : le nombre d'accès mémoire, la nature des calculs (flottants ou entiers), la localité des accès mémoire, ou encore le parallélisme peuvent le faire varier de plusieurs ordres de grandeur.

Dans des domaines comme la robotique et la conduite autonome, les vidéos doivent être traitées en temps réel sur des processeurs à faible puissance de calcul. Pour répondre à ces contraintes, plusieurs stratégies sont possibles : la diminution de la résolution de l'image, la réduction du nombre de pixels estimés, la dégradation de la méthode de recherche, etc. Bien que favorables au temps de calcul, ces stratégies détériorent généralement la qualité du flux optique et par conséquent aussi les résultats de l'application finale.

Dans ces travaux, afin d'accélérer l'estimation de mouvement sans forcément perdre en qualité, nous avons pris en compte tous les facteurs impactant la rapidité d'exécution. Deux types d'analyse de mouvement ont été considérés :

D'abord, en section 3.2, nous présentons une estimation rapide de mouvement semi-dense. Ce type d'approche est utile pour construire des algorithmes de reconstruction 3D semi-dense, ou des approches statistiques comme la reconnaissance d'actions à partir d'un faisceau de trajectoires. Pour estimer un champ de mouvement semi-dense, il est commun de faire l'hypothèse de

petits déplacements. Cela permet de restreindre le domaine de recherche au voisinage de chaque pixel estimé. De plus, le caractère semi-dense de l'estimation permet l'exploitation de la cohérence spatiale du flux optique pour améliorer sa qualité ou diminuer son temps de calcul. Avec la méthode que nous avons proposée, nous avons gagné un ordre de grandeur sur la fréquence de traitement de la version pyramidale de l'algorithme de Lucas Kanade [7] implémentée dans OpenCV [8].

Ensuite, en section 3.3, nous présentons une estimation rapide de mouvement épars. Ce type d'approche est particulièrement adapté à l'odométrie visuelle, au SLAM, et à d'autres applications nécessitant une information de mouvement éparse mais robuste aux grands déplacements ou aux changements d'apparence. Pour répondre à ces besoins, elle repose sur la répétabilité d'un détecteur de points et le pouvoir discriminant d'un descripteur pour mettre en correspondance les ensembles de points détectés dans deux images. Afin d'accélérer cette mise en correspondance, l'état de l'art propose des structures d'index permettant de diminuer sa complexité algorithmique. Bien qu'efficace d'un point de vue algorithmique, l'état de l'art (FLANN [44]) n'est avantageux qu'à partir d'un certain nombre de points d'intérêt. En effet, pour des ensembles de quelques milliers de points, notre implémentation de la recherche *brute force* est plus performante. A partir de ces observations et en prenant comme objectif la minimisation du temps de calcul, nous avons proposé une autre approche, gagnant jusqu'à deux ordres de grandeur sur le temps de calcul de FLANN, à qualité égale.

3.2 Estimation semi-dense du mouvement

3.2.1 Introduction

Estimer un champ de mouvement apparent dense ou semi-dense est un problème qui captive l'attention des chercheurs en traitement d'image depuis plusieurs dizaines d'années. Cela consiste à estimer un champ de déplacements couvrant la totalité ou une partie significative de la surface de l'image. Ce résultat, bien que bas niveau, peut fournir des informations sur la géométrie 3D de la scène et/ou la nature des actions observées. Cela lui confère

un grand intérêt pour plusieurs domaines incluant la vidéosurveillance, la défense/sécurité, la robotique, les jeux vidéos, etc.

Pour estimer un champ de mouvement, aussi appelé flux optique, il est courant de faire l'hypothèse qu'un objet observé ne change pas ou peu d'apparence entre deux images traitées. Bien que certains algorithmes proposent des méthodes pour estimer le mouvement malgré des changements d'illumination [13], la plupart des solutions proposées dans l'état de l'art reposent sur cette hypothèse. Étant donné deux images I_1 et I_2, une coordonnée de pixel $\mathbf{p} = \begin{bmatrix} x \\ y \end{bmatrix}$ et son vecteur de déplacement, inconnu, $\mathbf{v} = \begin{bmatrix} v_x \\ v_y \end{bmatrix}$, cette hypothèse est modélisée par la formule suivante :

$$I_1(\mathbf{p}) = I_2(\mathbf{p} + \mathbf{v}) \tag{3.1}$$

Nous pouvons ainsi formuler le flux optique $\vec{V}$ comme la minimisation d'une fonction D mesurant la dissemblance entre un point de l'image I_1 et son homologue dans l'image I_2 :

$$\vec{V}(\mathbf{p}) = \underset{\mathbf{v} \in \mathbb{R}^2}{\arg\min}\, D(\mathbf{p}, \mathbf{p} + \mathbf{v}) \tag{3.2}$$

Différentes mesures de dissemblance ont été évaluées mais la plus simple, la somme des différences absolues (SAD) des niveaux de gris, bénéficie d'un des meilleurs compromis entre qualité de résultat, temps de calcul rapide et simplicité d'implémentation.

Cette formulation est utilisée par Lucas Kanade [40]. Les approches de ce type sont qualifiées de locales car elles optimisent chaque vecteur de flux optique indépendamment des autres. Elles ont l'avantage d'être rapides à calculer et facilement parallélisable.

Pour calculer un vecteur de flux optique, ces approches proposent donc des algorithmes convergeant vers le minimum local d'une fonction de dissemblance.

En plus de l'hypothèse de constance de luminosité, la prise en compte de la régularité spatiale du flux optique permet d'augmenter la qualité de son estimation de manière significative. En effet, savoir qu'il y a une grande probabilité que deux pixels voisins aient des déplacements similaires permet

de détecter et/ou d'éviter un grand nombre d'erreurs.

C'est pourquoi la méthode Horn-Schunck [30] formule le flux optique comme le minimum global d'une énergie :

$$E(\vec{V}) = \iint_I \left[(\mathbf{v}.\nabla I + \frac{\partial I}{\partial t})^2 + \alpha^2 (||\nabla \mathbf{v}_x||^2 + ||\nabla \mathbf{v}_y||^2) \right] dxdy \tag{3.3}$$

où $\nabla \mathbf{v}_x$ et $\nabla \mathbf{v}_y$ représentent les gradients spatiaux des composantes $\mathbf{v}_x$ et $\mathbf{v}_y$. Cette énergie est minimisée en résolvant les équations d'Euler Lagrange, comme dans [10], et est composée de deux termes :

- **Le terme de données** reprenant l'hypothèse de constance de luminosité vu précédemment et correspondant au développement de Taylor à l'ordre 1 de l'équation 3.1.
- **Le terme de régularisation** pénalisant les variations spatiales du flux optique.

Favoriser la cohérence locale du flux optique permet de propager les informations de mouvement contenues sur les zones texturées dans les zones moins texturées, où l'hypothèse de constance de luminosité ne permet plus d'identifier le mouvement d'un pixel.

La non-convexité des fonctions à minimiser engendre la présence de minima locaux et empêche les algorithmes de converger vers la solution. Plus la norme des vecteurs de flux optique est grande, plus ces problèmes sont fréquents. Pour pallier ce problème, l'état de l'art propose plusieurs techniques :

- **La fenêtre d'intégration :** Au lieu d'utiliser une fonction de dissemblance entre deux pixels seuls, il est courant d'intégrer la fonction sur une fenêtre centrée sur le pixel. Plus la fenêtre est grande, et plus de grands déplacements pourront être estimés. Cependant, cela augmente aussi le temps de calcul et peut lisser exagérément le flux optique.
- **L'estimation multiéchelle :** Pour augmenter la portée de l'estimation du flux optique, un grand nombre de méthodes reposent sur une taille de fenêtre variable. Dans un premier temps, la fenêtre est agrandie pour capter les grands mouvements, puis successivement rétrécie pour augmenter la précision du flux optique. Il existe deux implantations du paradigme multiéchelle : la première, qui consiste à faire varier la taille de la fenêtre, est plus précise mais plus lente à calculer. Elle

est utilisée dans eFOLKI [51]. La deuxième, qualifiée de pyramidale et utilisée par la version multiéchelle de Lucas-Kanade [7], fait varier la résolution de l'image afin de conserver une taille de fenêtre constante. Cette dernière alternative a l'avantage d'avoir un impact limité sur le temps de calcul.

Au delà de la résolution analytique et numérique du flux optique, les chercheurs et les développeurs d'applications sont souvent confrontés à un deuxième challenge : comment l'implanter efficacement sur les processeurs actuels pour permettre **le traitement d'une vidéo en temps réel** ? Cette question soulève en fait deux problèmes, le premier étant la minimisation de la complexité algorithmique, et le deuxième l'implantation efficace sur l'architecture ciblée. Par exemple, dans eFOLKI [51], la version pyramidale de Lucas-Kanade [7] a été modifiée pour accélérer son exécution sur un **processeur graphique (GPU)**. En effet, le GPU a la particularité d'être une architecture massivement parallèle et les algorithmes doivent être adaptés à ce paradigme pour en tirer parti. Dans [36], les auteurs annoncent des fréquences de traitement allant jusqu'à **600Hz** sur des petites résolutions d'images. Ils proposent une implantation rapide de l'algorithme Lucas-Kanade en y ajoutant une minimisation d'énergie globale similaire à Horn-Schunck [30] (voir équation 3.3) prenant en compte les erreurs de matching et la régularité du flux optique.

Bien que le calcul de flux optique simple entre deux images suffise à un certain nombre d'applications, il est parfois nécessaire d'accumuler l'information de mouvement sur toute une séquence vidéo. Par exemple, les méthodes de reconnaissance d'actions dans les vidéos [49, 47, 48] reposent souvent sur **un ensemble de trajectoires de points** pouvant s'étendre sur plusieurs dizaines de trames vidéo. Un algorithme d'estimation de trajectoires à long terme a déjà été proposé dans *Particle Video* [57]. Il repose sur l'estimation d'un flux optique dense sur l'ensemble des trames d'une vidéo pour accumuler des trajectoires au cours de la vidéo. Les positions des points suivis, appelés particules, sont optimisées via la minimisation de l'énergie globale du graphe des particules avec une méthode variationnelle similaire à Brox et al. [10]. Cette méthode a l'avantage d'estimer avec une grande précision un flux de trajectoires, mais a l'inconvénient d'avoir un temps de calcul élevé : entre 15

et 70 secondes par image, sans compter le calcul du flux optique. Cela limite son intégration dans les applications à fortes contraintes de calcul.

Pour pallier ce problème, nous proposons dans cette section *Video Extruder* une méthode pour extraire un champ de trajectoires en moins de 10 millisecondes (3 ordres de grandeur de moins que *Particle Video*) par image pour une résolution de 640×480 pixels et se rapprochant de la précision du flux optique obtenu avec la version multiéchelle de Lucas Kanade [7]. Notre méthode est fortement inspirée de cette dernière, car elle repose aussi sur une descente de gradient pour apparier des points d'intérêt, et sur une pyramide pour estimer les grands mouvements.

Dans la suite, nous reprenons le terme "particule" de *Particle Video* [57] pour désigner un point de l'image suivi dans l'espace 2D+t. Notons que cette définition est différente de celle utilisée dans les filtres particulaires, où elle désigne un point d'un espace d'état qui peut éventuellement représenter la position d'un objet [33].

Nos contributions, axées sur l'accélération de l'extraction d'un champ de particules, sont les suivantes :

- **Un protocole d'évaluation de détecteur de points**, prenant en compte la précision d'un algorithme de mise en correspondance et non la répétabilité du détecteur. Cette étude a mené à la conception du **nouveau détecteur de points MIEL**.
- **Un algorithme multi-échelle de mise en correspondance** de particules reposant sur une prédiction à la fois temporelle et spatiale.
- **Un filtrage des trajectoires incohérentes** reposant sur la cohérence spatiale du flux optique.
- **Une indexation spatiale des particules** pour diminuer la complexité algorithmique de la construction des trajectoires.

Notre méthode comporte trois étapes : la mise en correspondance, le filtrage, et la détection de nouvelles particules. La figure 3.1 présente une vue d'ensemble du pipeline, dont les entrées sont la trame vidéo courante (indice t) et l'ensemble multiéchelle de particules à la trame précédente (indice $t-1$).

Cette section est organisée de la façon suivante. La section 3.2.2 présente notre étude sur les détecteurs de points. La section 3.2.3 définit le descripteur utilisé par la mise en correspondance. La section 3.2.4 détaille notre

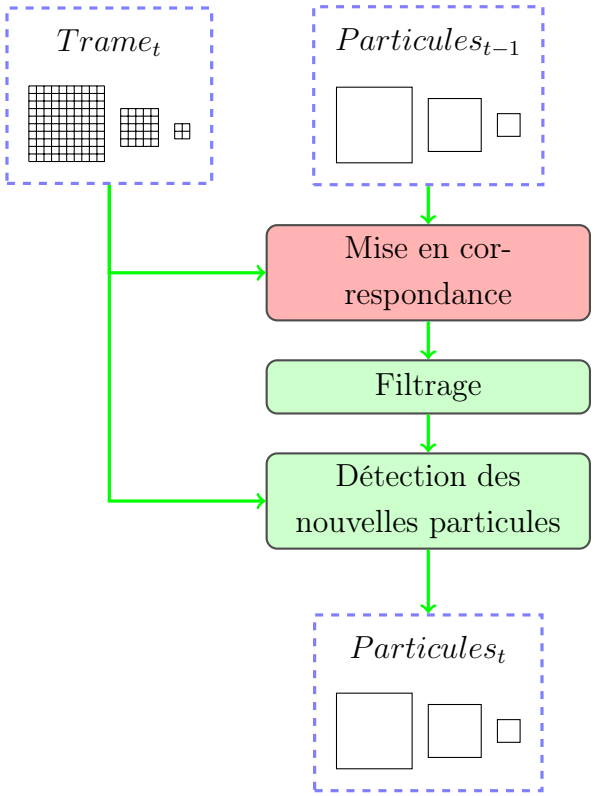

FIGURE 3.1 – Vue d'ensemble de *Video Extruder*. Seule l'étape de mise en correspondance (en rouge) doit être exécutée à l'arrivée de chaque nouvelle trame. Le filtrage et la détection peuvent être moins fréquents. Dans nos expériences, nous les avons exécutés toutes les 5 images, soit 6 fois par seconde dans une vidéo acquise à 30Hz.

algorithme multiéchelle de mise en correspondance, le filtrage des particules incohérentes, la fusion des particules proches, et compare les performances de notre algorithme à celles de Lucas Kanade [7] et à l'algorithme de flux optique dense de Farnebäck [17]. Finalement, nous expliquons dans la section 3.2.5 comment le pipeline a été implanté sur différentes architectures et présentons les temps de calcul.

3.2.2 Détection d'un champ semi-dense de particules

Dans la littérature, les concepteurs de détecteurs de points d'intérêt tels que FAST [55] évaluent la capacité du détecteur à détecter à nouveau les mêmes points après l'application d'une transformation géométrique à l'image. En effet, ce critère de **répétabilité** est à maximiser dans les applications reposant sur la mise en correspondance d'ensembles de points d'intérêt provenant de plusieurs images.

Cependant, le suivi semi-dense utilise une méthode différente : à chaque nouvelle trame vidéo, les nouvelles positions des particules sont recherchées via la **minimisation d'une mesure de similarité**. L'espace de recherche est une région de l'image et non l'ensemble des points d'intérêt détectés dans la nouvelle trame.

Finalement, les objectifs d'un détecteur de particules et d'un détecteur de points d'intérêt divergent : les particules sont adaptées au suivi à haute densité, quand leurs déplacements et déformations sont prédictibles, alors que les points d'intérêt plus adaptés au suivi épars ont pour but de reconnaître le même ensemble de points quel que soit leur déplacement.

Toutefois, la frontière entre particules et points d'intérêt est assez floue : un détecteur de points d'intérêt peut être paramétré pour extraire un grand nombre de points et atteindre une densité suffisante pour le suivi semi-dense.

L'aspect local de la recherche remet donc en question le besoin de points d'intérêt répétables pour le suivi semi-dense, car finalement, la qualité des trajectoires dépend davantage de la capacité de la descente de gradient de la mise en correspondance à atteindre les nouvelles positions des particules que de la répétabilité du détecteur.

Nous avons donc proposé dans cette étude le détecteur de particules

MIEL conçu spécialement pour les besoins du suivi semi-dense. Nous l'avons ensuite comparé au détecteur de points d'intérêt **FAST**, paramétré pour une extraction semi-dense. Pour mener cette évaluation, nous avons proposé un protocole prenant en compte directement la capacité de l'algorithme de mise en correspondance à suivre les points détectés avec précision et ignorant volontairement la répétabilité.

Le détecteur de points FAST

Le détecteur de points d'intérêt FAST [54] calcule des statistiques locales sur $B_3(\mathbf{p})$, le cercle de Bresenham de rayon 3 (Fig. 3.7, gauche) centré sur le pixel candidat $\mathbf{p}$. Soit I l'image de niveau de gris en entrée de l'algorithme. FAST calcule les deux ensembles S^- et S^+ :

— $S_c^-(\mathbf{p}) = \{\mathbf{q} \in B_3(\mathbf{p}); I(\mathbf{q}) \leq I(\mathbf{p}) - c\}$
— $S_c^+(\mathbf{p}) = \{\mathbf{q} \in B_3(\mathbf{p}); I(\mathbf{q}) \geq I(\mathbf{p}) + c\}$

Ensuite, sont sélectionnés les points p pour lesquels S_c^- ou S_c^+ contiennent un ensemble d'au moins n points contigus. c est un paramètre de contraste, alors que n est un paramètre géométrique.

Pour éviter de sélectionner des points d'intérêt adjacents, FAST restreint la sélection aux maxima locaux sur des voisinages 3×3 de la mesure de saillance suivante :

$$\Sigma_c^{\text{FAST}}(\mathbf{p}) = \max\left(\sum_{\mathbf{q}\in S_c^+(\mathbf{p})} \delta_c(\mathbf{p},\mathbf{q}), \sum_{\mathbf{q}\in S_c^-(\mathbf{p})} \delta_c(\mathbf{p},\mathbf{q})\right)$$

Avec $\delta_c(\mathbf{p},\mathbf{q}) = |I(\mathbf{q}) - I(\mathbf{p})| - c$. L'avantage de FAST est son bon compromis entre complexité calculatoire et taux de répétabilité. Nous montrons dans la suite qu'il peut être adapté à un suivi semi-dense, à condition d'adapter la stratégie de sélection.

Le détecteur de points MIEL

Le détecteur de points d'intérêt MIEL est calculé sur le même support $B_3(\mathbf{p})$ que FAST. Soit $\{\mathbf{q}_i\}_{0\leq i\leq 15}$ les 16 points de $B_3(\mathbf{p})$, numérotés dans le sens horaire. MIEL définit la fonction de saillance suivante :

$$\Sigma^{\text{MIEL}}(\mathbf{p}) = \min_{i=0}^{7} |2I(\mathbf{p}) - I(\mathbf{q}_i) - I(\mathbf{q}_{i+8})|$$

où « + » dénote l'addition modulo 8.

Cette formule est basée sur l'hypothèse que le suivi sera ambigu sur les points où il existe au moins une direction au long de laquelle les niveaux de gris varient linéairement. La fonction est équivalente à la valeur absolue minimum de la seconde dérivative dans les huit directions, approximée par un simple noyau de calcul discret pour minimiser le nombre d'accès mémoire.

Les points d'intérêt peuvent être sélectionnés en fonction d'un maximum local comme FAST, mais nous décrivons dans la suite une stratégie de sélection qui répond mieux aux besoins du suivi semi-dense.

Stratégie de sélection *blockwise* adaptée au suivi semi dense

Comme décrit dans la section 3.2.2, FAST sélectionne les maxima locaux de la fonction de saillance (LM). Cela limite la redondance des points d'intérêt en évitant que deux points adjacents soient détectés en même temps. Cependant, cette méthode n'est pas idéale pour le suivi d'un grand nombre de points, car un point avec une saillance élevée peut être suivable même s'il n'est pas un maximum local.

Pour atteindre notre objectif de haute densité, nous utilisons une méthode de sélection moins restrictive qui permet d'extraire plus de points dans les zones à forte saillance : pour chaque cellule de taille 3×3, le pixel avec la plus haute saillance est sélectionné si celle-ci est plus élevée que le seuil du détecteur. Dans la suite, nous appelons cette stratégie « blockwise maxima » (BM).

L'illustration 3.2 compare les deux stratégies.

Nous comparons dans la suite l'impact de cette stratégie sur la densité de particules extraites.

Optimisations de la détection de points d'intérêt

Dans le but de réduire l'impact du détecteur sur le temps de calcul du suivi semi-dense, nous avons utilisé deux types d'optimisation. Les premières, nommées optimisations intrinsèques, ont pour but de réduire le temps de calcul

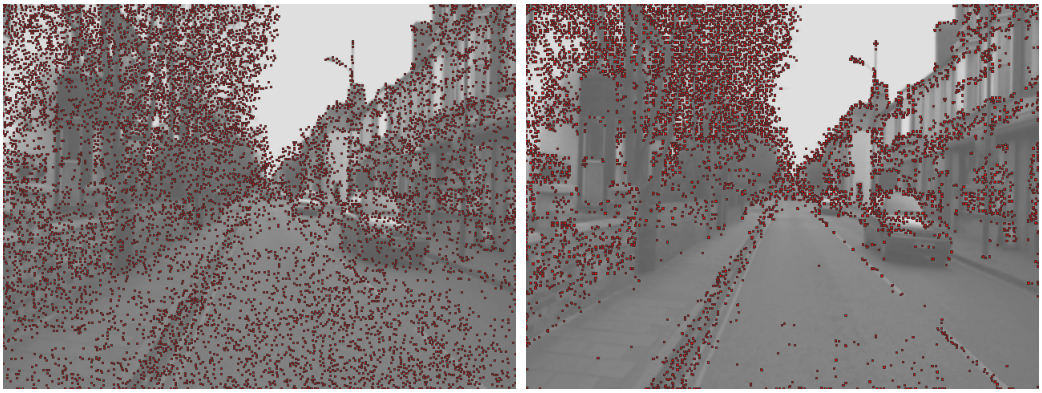

FIGURE 3.2 – Comparaison des maxima locaux classiques (LM) (image de gauche) et *blockwise* (BM) (image de droite) sur la saillance du détecteur MIEL (voir Sec. 3.2.2). Dans les deux cas, 9000 points ont été extraits. Finalement, pour atteindre un plus grand nombre de points, la stratégie LM sélectionne des points dans les zones peu texturées et difficiles à suivre, alors que BM fournit un meilleur recouvrement des zones saillantes.

du détecteur. Les deuxièmes, nommées optimisations extrinsèques, consistent à diminuer le nombre d'exécutions du détecteur dans le cadre du suivi semi-dense. Nous détaillons dans les deux sections suivantes les optimisations implantées.

Optimisations Intrinsèques

D'un point de vue algorithmique, FAST et MIEL ont la même complexité. Cependant, les auteurs de FAST [56] montrent que pour certaines valeurs du paramètre géométrique n, il n'est pas nécessaire de vérifier l'intégralité des 16 pixels de $B_3(\mathbf{p})$ pour prouver l'absence de points d'intérêt. Cela permet de diminuer significativement le nombre d'accès mémoire dans les zones sans texture comme le ciel. Toutefois, ces optimisations sont spécifiques à chaque valeur de n, et nous allons montrer dans la section 3.2.2 que pour nos critères, la valeur optimale de ce paramètre dépend du nombre de points d'intérêt visé.

Au contraire, MIEL n'a pas de paramètre géométrique et se prête à des optimisations plus simples : comme la fonction de saillance calcule un minimum, il est possible de rejeter un pixel candidat dès que la dérivée seconde le long de l'une des huit directions est détectée inférieure au seuil c. Comme

pour l'optimisation de FAST expliquée dans le paragraphe précédent, cette technique permet de diminuer le nombre d'accès mémoire dans les zones sans texture.

Optimisations Extrinsèques

Dans le pipeline du suivi semi-dense *Video extruder*, le détecteur se limite au traitement des zones venant d'apparaître. Ces apparitions sont provoquées par trois phénomènes :

— un changement d'angle de vue faisant apparaître une nouvelle partie de la scène sur le plan focal
— une désoccultation
— un échec du suivi laissant une zone texturée dépourvue de points d'intérêt
— un effet de zoom faisant apparaître de nouveaux détails

En fonction des types de scènes observées et de l'application, la fréquence d'exécution du détecteur doit être adaptée afin de consommer le moins de ressources possible. Dans nos expériences, nous avons fixé la fréquence de détection à 6 Hz.

Finalement, nous limitons l'exécution du détecteur aux zones encore non couvertes par des particules. Cela diminue significativement le nombre de pixels que le détecteur doit examiner.

Protocole d'évaluation de la sélection de particules

Le protocole d'évaluation de détecteur de points que nous avons proposé est décrit par la démarche suivante :

— extraire les points d'intérêt avec le détecteur sur une image I_1
— appliquer à l'image une translation $\mathbf{v}$.
— pour simuler les artefacts de caméra, ajouter à l'image translatée un bruit Gaussien ($\sigma = 5.0$). Cette image sera notée I_2
— avec l'algorithme de mise en correspondance, rechercher les nouvelles positions de particules dans I_2.
— calculer la distance moyenne entre les vecteurs de déplacement des particules et le déplacement de référence $\mathbf{v}$.

Nous moyennons les résultats de cette démarche sur toutes les rotations

FIGURE 3.3 – De gauche à droite et de haut en bas : 1 - Image utilisée pour l'évaluation des détecteurs. 2 - Points extraits avec le détecteur idéal (les 10 000 points avec la plus petite erreur de suivi). 3 - 10 000 points d'intérêt FAST. 4 - 10 000 points d'intérêt MIEL.

possibles de I comprises entre 0° et 360° avec un pas de 1°, et 100 translations $\mathbf{v}$ tirée aléatoirement dans l'espace des vecteurs de norme 5.

L'image 3.3 a été utilisée car elle présente plusieurs avantages pour l'évaluation d'un détecteur de points d'intérêt. D'abord, sa taille et la quantité d'information qu'elle contient permettent la détection de plusieurs milliers de points d'intérêt sur différents types de textures. En effet, cette scène urbaine inclut à la fois des zones fortement texturées et de différentes natures (immeubles, voitures, arbres, ...) ainsi que des zones à éviter comme le ciel et la route où l'algorithme de mise en correspondance a tendance à échouer. De plus, la vue d'une rue offre un large échantillon de niveaux de détails variant des très grands objets au premier plan aux très petits proches du point de fuite.

Résultats de l'évaluation

Grâce au protocole décrit précédemment, nous avons évalué et comparé différentes configurations des détecteurs FAST et MIEL ainsi que les stratégies de sélection LM et BM.

Pour mieux apprécier les performances de chaque détecteur, nous avons ajouté aux résultats les performances de deux détecteurs fictifs :

— **le détecteur aléatoire** sélectionne aléatoirement, selon une distribution uniforme, des points dans l'image.

— après avoir exécuté l'algorithme de suivi sur tous les pixels, **le détecteur idéal** sélectionne les points avec les vecteurs de flux optique les plus proches de la transformation de référence $\mathbf{v}$.

Les performances des détecteurs évalués seront forcément comprises entre celles de ces deux détecteurs de référence.

L'illustration 3.4 montre l'impact de la stratégie de sélection *blockwise* maxima (BM) (décrite dans la section 3.2.2). Nous observons qu'avec BM, il est possible d'extraire jusqu'à deux fois plus de points. De plus, avec des configurations permettant d'extraire plus de 8000 points FAST9 ou 5000 points MIEL, nous obtenons une erreur de suivi plus faible. Cela est principalement dû à une plus forte concentration de points d'intérêt sur les zones texturées.

Le graphique 3.5 compare différentes versions du détecteur FAST. Il montre que l'extraction est optimale avec $n = 9$ pour moins de 20 000 points, alors que $n = 8$ est meilleur pour plus de 20 000 points.

La figure 3.6 montre que les erreurs de suivi des particules détectées par MIEL sont jusqu'à 0.05px plus grandes que les points détectés par FAST, et environ égales pour plus de 25 000 particules extraites.

Les trois expériences que nous avons présentées nous ont permis de tirer plusieurs conclusions :

— Pour le suivi semi-dense, la simplification du détecteur FAST n'a pas détérioré la qualité du suivi. Il est même apparu dans nos expériences que plusieurs détecteurs simples basés sur le même voisinage $B_3(\mathbf{p})$ ont des performances similaires.

— Que ce soit avec FAST ou MIEL, changer la stratégie de sélection de LM à BM comme décrit dans la section 3.2.2 permet d'extraire plus

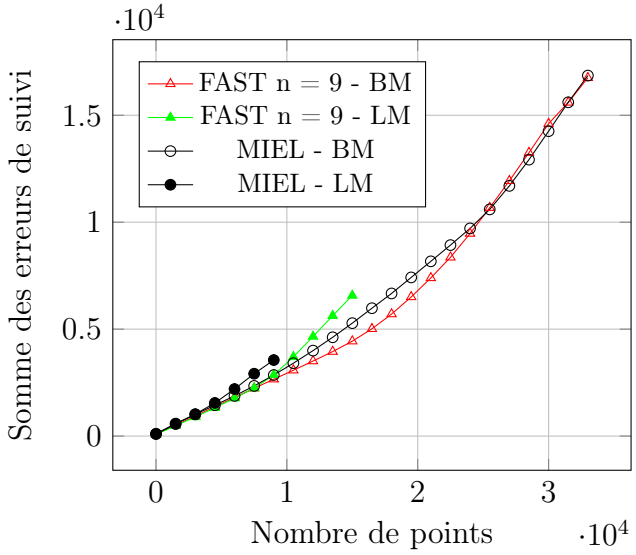

FIGURE 3.4 – Comparaison des stratégies de sélection LM et BM. Bien que plus simple à calculer, BM est proche ou meilleure que LM à partir de 5000 points MIEL or 8000 points FAST.

de points d'intérêt. Ou, pour un nombre de particules fixé, les points fournis par BM offrent une erreur de suivi plus faible.

— La comparaison avec le détecteur idéal montre qu'il existe encore une très large marge d'amélioration pour de meilleurs détecteurs. Dans des travaux futurs, nous pourrions utiliser cette évaluation en conjonction d'un algorithme d'apprentissage pour converger vers le détecteur idéal.

3.2.3 Descripteur de points d'intérêt à faible coût

Les descripteurs de points de l'état de l'art, comme SIFT [39] ou SURF [4], utilisés entre autres pour le suivi de points épars, sont en général conçus pour maximiser le compromis entre les deux qualités suivantes :

— **Le pouvoir de discrimination** : un descripteur est censé pouvoir reconnaître une région de l'image parmi d'autres, même similaires. Ce pouvoir est entre autres lié à la taille du descripteur : les descripteurs les plus grands ont en général un pouvoir de discrimination plus élevé.

— **Invariance** : les descripteurs doivent aussi être robustes aux change-

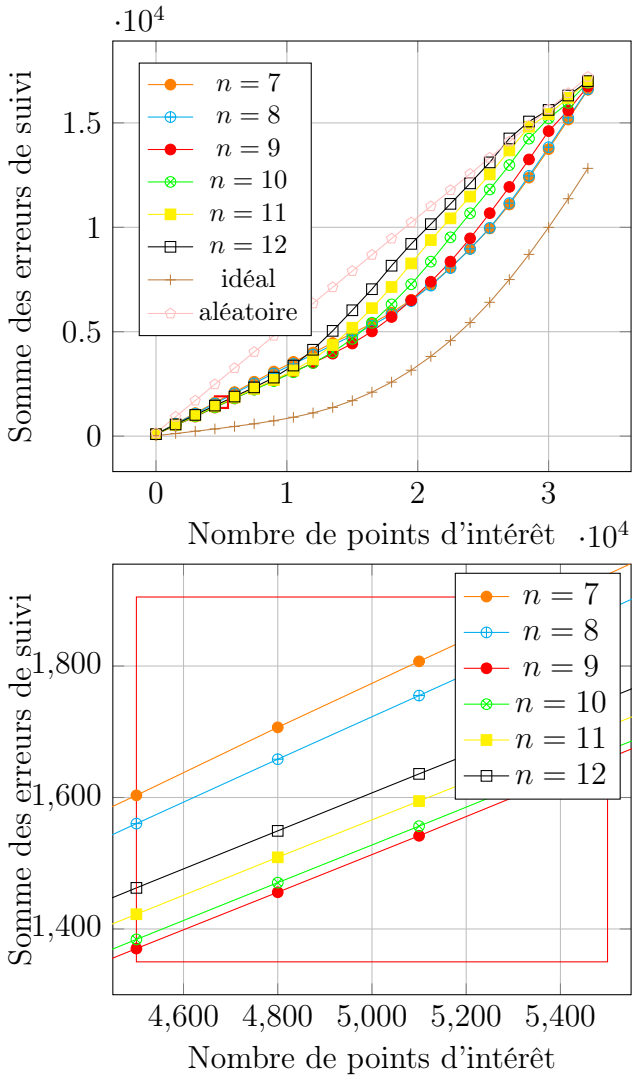

FIGURE 3.5 – Évaluation du détecteur FAST. Le meilleur suivi a été obtenu avec $n = 9$ pour quelques milliers de points (graphique du bas) et $n = 7$ ou 8 pour plus de 20 000 points (graphique du haut).

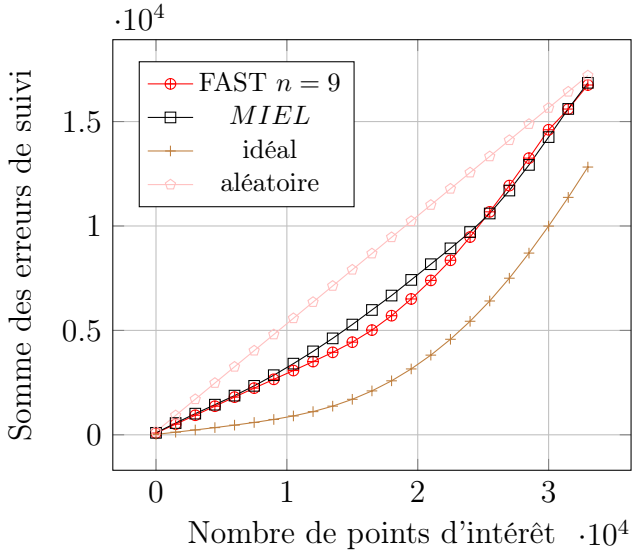

FIGURE 3.6 – Comparaison de FAST-9 et MIEL. MIEL est comparable au détecteur FAST, bien que légèrement moins bon en dessous de 25 000 particules.

ments géométriques et photométriques qui peuvent altérer l'apparence d'une région entre deux trames vidéos. Ces changements sont liés à des phénomènes comme le bruit du capteur, un changement de point de vue, d'illumination, etc...

Cependant, l'algorithme de recherche de points d'intérêt que nous avons présenté tire peu avantage de ces deux qualités, et cela pour deux raisons. D'abord, la prédiction de la prochaine position d'une particule et l'approche *coarse-to-fine* réduisent largement l'espace de recherche, ce qui limite le besoin d'un descripteur discriminant. De plus, comme notre suivi a été conçu pour traiter un flux vidéo à cadence élevée, le point de vue et l'apparence des objets ne doivent pas varier significativement entre deux trames consécutives, limitant le besoin d'une invariance de descripteur.

C'est pourquoi, dans le but de diminuer au maximum l'impact de la taille du point d'intérêt sur le temps de calcul, nous avons conçu un descripteur peu discriminant et peu invariant, stockant la quantité d'information minimum pour répondre aux besoins de notre approche.

Ce descripteur repose sur un vecteur de 32 valeurs scalaires, correspondant

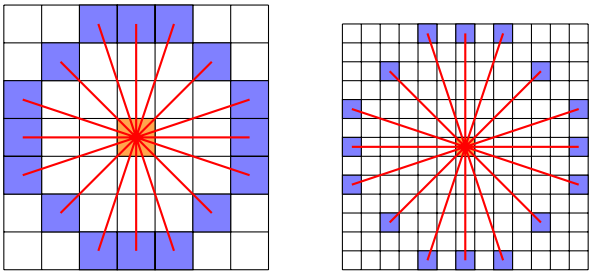

FIGURE 3.7 – Les deux voisinages B_3 et B_6 utilisés par le détecteur MIEL et le descripteur.

à deux ensembles de 16 valeurs échantillonnées uniformément sur les cercles de Bresenham de rayon 3 (première échelle) et 6 (deuxième échelle), comme le montre la figure 3.7.

Ces valeurs sont simplement stockées dans la trame vidéo courante, lissée par un flou gaussien d'écart type 1.0 pour la première échelle et d'écart type 2.0 pour la deuxième échelle. L'algorithme de recherche présenté dans la partie 3.2.4 utilise la distance $\mathcal{L}_1$ entre ces vecteurs de taille 32 comme mesure de dissemblance.

Finalement, par rapport aux descripteurs fortement discriminants et invariants de l'état de l'art, ce descripteur a deux avantages :

— **un temps de calcul réduit :** le calcul des descripteurs de tous les pixels se résume à l'application de deux flous gaussiens sur la trame courante.

— **une empreinte mémoire réduite :** en utilisant des images de niveau de gris 8 bits, la taille d'un descripteur est réduite à 256 bits. Cette petite taille accélère significativement le calcul de la distance et donc l'algorithme de mise en correspondance.

3.2.4 Prédiction et recherche des nouvelles positions des particules dans une nouvelle image

Cette section présente l'algorithme recherchant la position des particules courantes dans la nouvelle image du flux vidéo. Nous présentons d'abord une vue d'ensemble de la méthode puis détaillons chacune des parties.

Vue d'ensemble

Dans un premier temps, les nouvelles positions des particules sont prédites grâce à une approche *coarse-to-fine*. Des statistiques calculées à l'échelle supérieure et le mouvement estimé à la trame précédente sont utilisés pour prédire les positions des particules.

Ces prédictions sont ensuite raffinées avec une descente de gradient sur les deux échelles du descripteur.

Pour éviter les dérives causées par des occultations, chaque particule ayant convergé vers un emplacement non similaire (similarité calculée à partir du descripteur présenté en section 3.2.3) est éliminée. Ensuite, nous filtrons les particules ayant un mouvement anormal vis-à-vis du voisinage.

Finalement, pour éviter qu'une trop grande densité de particules n'augmente significativement le temps de calcul, nous fusionnons les particules situées sur des pixels adjacents.

Prédiction des nouvelles positions des particules

Pour gagner en robustesse aux grands déplacements, souvent dus aux mouvements de caméra soudains et aux objets à mouvement rapide, les positions des particules sont prédites avec une approche hybride : la prédiction repose sur les informations présentes à la fois dans les échelles supérieures, si elles existent, et dans le mouvement calculé dans la trame précédente.

Étant donné une particule résidant à un temps t et à une échelle s, P_t^s définit sa position dans la trame courante et $V_t^s = P_t^s - P_{t-1}^s$ son vecteur de déplacement. À la première échelle $s = s_{\max}$ (la plus grossière), les positions des particules sont prédites avec les informations de mouvements extraites à la trame précédente :

$$\widehat{P_t^{s_{\max}}} = P_{t-1}^{s_{\max}} + V_{t-1}^{s_{\max}}$$

Cette prédiction est ensuite raffinée avec une descente de gradient pour donner $P_t^{s_{\max}}$ la nouvelle position et $V_t^{s_{\max}}$ le nouveau vecteur de déplacement de la particule.

Ensuite, à partir du mouvement calculé à la première échelle, nous déduisons récursivement le mouvement des particules aux échelles suivantes ($s < s_{\max}$). La prédiction du mouvement à l'emplacement d'une particule

à l'échelle s hérite du mouvement estimé au même emplacement à l'échelle $s+1$ calculé précédemment, quand il est disponible. Cependant, malgré la densité du champ de particules, il est possible qu'une particule n'hérite d'aucun mouvement. Dans ce cas, elle est appelée particule orpheline, et il n'est pas possible de prédire sa nouvelle position par les échelles supérieures. Pour pallier ce problème, nous utilisons la stratégie de prédiction suivante à chaque particule à une échelle $s < s_{\max}$:

1. Pour densifier le champ de déplacements calculé à l'échelle supérieure et ainsi réduire le nombre de particules orphelines, nous remplaçons dans ce champ chaque bloc de taille 8×8 par le déplacement moyen des particules contenues dans ce même bloc. Ce champ plus dense est noté $\widetilde{V_t^{s+1}}$.
2. Ensuite, si $\widetilde{V_t^{s+1}}$ est défini, la position de chaque particule est prédite avec :
$$\widehat{P_t^s} = P_{t-1}^s + 2\widetilde{V_t^{s+1}}[P_{t-1}^s/2]$$
sinon la particule reste orpheline et sa position est prédite à partir de sa vitesse :
$$\widehat{P_t^s} = P_{t-1}^s + V_{t-1}^s$$

Un léger inconvénient du caractère multiéchelle de cette prédiction est qu'il introduit des erreurs aux frontières des objets mobiles, là où les blocs 8×8 recouvrent à la fois l'objet et le fond. Toutefois, le sous-échantillonnage de V_{t-1}^s utilise moins de mémoire et diminue l'impact des erreurs de suivi dans la prédiction.

Raffinement de la prédiction

Après avoir prédit la position d'une particule, nous recherchons dans une région locale l'emplacement minimisant la distance de son descripteur. Pour cela, nous utilisons une descente de gradient à deux étapes exploitant les deux échelles contenues dans le descripteur. L'algorithme prend en argument un descripteur `Fref`, une prédiction de nouvelle position `p`, la fonction `F` construisant le descripteur à un emplacement donné. Il retourne les coordonnées minimisant localement la distance `d`.

```
descent(reference_descriptor Fref,
        descriptor_function F,
        prediction_position p,
        distance d) {
  new_pred = p
  d_min = d(Fref,F(p))
  best_q = p
  repeat {
    local_min = 1
    for all q neighbour of new_pred {
      if (d(Fref,F(q)) < d_min) {
         d_min = d(Fref,F(q))
         best_q = q
         local_min = 0
        }
    }
    new_pred = best_q
  } until (local_min == 1)
  return best_q
}
```

Pour améliorer la robustesse et diminuer le temps de calcul de la descente de gradient, nous séparons la recherche sur les deux échelles stockées par le descripteur (voir la description du descripteur section 3.2.3).

Soit F_1 (resp. F_2) la partie du descripteur contenant les valeurs de pixels extraits à la petite (resp. grande) échelle. Soit d_1 (resp d_2) la distance $\mathcal{L}_1$ entre les sous-descripteurs F_1 (resp F_2). Pour une particule donnée, son descripteur `Fref` et sa position prédite `p`, nous estimons sa nouvelle position avec une succession de deux descentes, la première utilisant d_2 et la deuxième, plus fine, la distance $d_1 + d_2$:

```
descent(Fref, F, descent(Fref, F, p, d2), d1+d2)
```

Pour éviter de suivre les particules suivant une zone venant d'être occultée, nous rejetons les particules dont la distance d_1+d_2 dépasse un seuil θ. Ce seuil

permet d'équilibrer le suivi entre robustesse aux changements d'apparence et capacité à détecter correctement les occultations. Durant nos expériences, nous avons utilisé $\theta = 300$ pour obtenir les résultats du tableau 3.1. Cette valeur correspond à 7.5% de la valeur maximale théorique de $d_1 + d_2$.

Filtrage des incohérences spatiales

Être capable de détecter les erreurs est essentiel pour un algorithme de suivi. Ces erreurs sont en général dues à (i) le pouvoir de discrimination limité du descripteur et (ii) la réduction de l'espace de recherche empêchant la recherche de corriger une trop mauvaise prédiction de position.

Au lieu d'appliquer une régularisation spatiale coûteuse, nous avons choisi de limiter le filtrage à la suppression des particules ayant un déplacement incohérent avec leur voisinage. Pour limiter le temps de calcul, la détection des erreurs réutilise le champ de mouvement $\widetilde{V_t^{s+1}}$ et estime la divergence d'une particule par rapport aux autres se trouvant dans le même bloc de 8×8 pixels :

$$||V_t^s - \widetilde{V_t^s}|| > \lambda$$

Nous avons fixé $\lambda = 10$ pixels dans nos expériences.

Bien que ce filtrage et la sélection sur la distance des descripteurs présentée dans la section précédente peuvent parfois supprimer de bonnes particules, nous avons paramétré λ et θ pour obtenir un compromis entre taux d'erreur et densité. Finalement, grâce à la réutilisation de $\widetilde{V_t^{s+1}}$, ce filtrage a un impact négligeable sur le temps de calcul de la chaîne complète.

Fusion des particules proches

En calculant un champ de particules semi-dense, la probabilité que deux particules convergent sur la même trajectoire est élevée. Cela peut mener à des particules redondantes. Pour limiter ce problème qui peut impacter significativement le temps de calcul, les particules situées sur des pixels adjacents sont fusionnées en supprimant la particule la plus jeune. Grâce à l'index spatial présenté en section 3.2.5, tester si une particule est adjacente à une autre est de complexité $O(1)$.

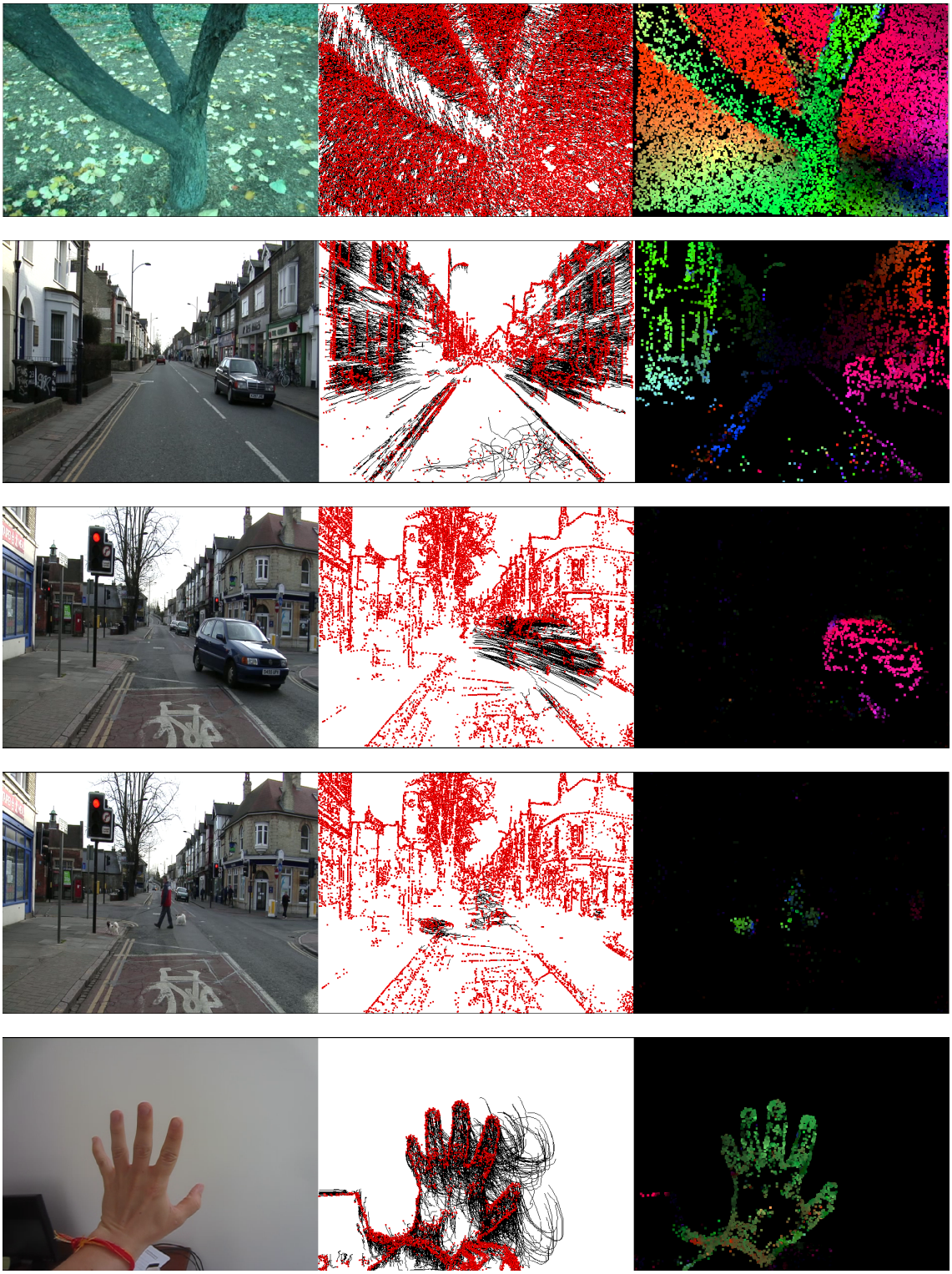

FIGURE 3.8 – De gauche à droite : la trame vidéo en entrée, les particules en rouge et leurs trajectoires en noir, le champ de mouvement semi-dense représentant les coordonnées polaires des vecteurs de mouvement avec un code de couleur $<\rho,\theta> \equiv$ <intensité, teinte>.

Évaluation de la chaîne complète

Un aperçu des résultats du suivi semi-dense sur différents scénarios est présenté en figure 3.8. Dans cette section, nous proposons une méthode pour évaluer notre suivi semi-dense et le comparons avec d'autres approches :

— l'implantation pyramidale Lucas-Kanade (pyrLK) [7] fournie par OpenCV
— le flux optique de Farnebäck [17] aussi disponible dans OpenCV

Le suiveur de points pyrLK utilise un schéma de résolution d'Euler Lagrange itératif. La qualité des résultats obtenus avec cette approche est liée au caractère lisse de la fonction à optimiser, qui dépend principalement de la taille de la fenêtre utilisée pour intégrer les dérivées locales. Cette taille a aussi un impact majeur sur le temps de calcul. Nous avons donc choisi d'évaluer plusieurs configurations de pyrLK (par défaut, OpenCV fixe la taille de la fenêtre d'intégration à 21).

Le flux optique de Farnebäck [17] est fondé sur une expansion polynomiale, et estime les coefficients du polynôme à partir des moindres carrés pondérés sur l'image. La taille du noyau de convolution discret approximant une gaussienne et utilisé pour le calcul des coefficients a aussi un impact sur la qualité du flux optique estimé et la robustesse aux grands mouvements. Nous avons donc décidé d'évaluer cette approche avec plusieurs tailles. (par défaut, OpenCV fixe ce paramètre à 15).

Pour *Video Extruder*, le caractère lisse de la fonction de dissemblance à minimiser est lié à la taille et aux échelles utilisées dans le descripteur (voir section 3.2.3).

Génération de séquences synthétiques avec mouvement

Pour évaluer ces trois algorithmes d'estimation de mouvement, nous avons proposé une méthode simple générant des vidéos synthétiques avec leur champ de mouvement associé. Le générateur simule une scène à partir d'une large image panoramique en fond et trois vignettes d'objets. Deux mouvements y sont simulés : le mouvement pan-tilt de la caméra affectant la scène complète (fond et objets), et les mouvements propres à chaque objet.

Soit α_t le vecteur d'accélération aléatoire à l'image t, et β_t^i l'accélération aléatoire de chaque objet i à l'instant t. Les orientations de ces vecteurs sont changées aléatoirement toutes les 5 images, alors que leurs normes sont

gardées constantes.

Deux scénarios sont utilisés dans notre évaluation, le premier avec de petites accélérations est appelé S_A, avec $||\alpha_t|| = 1$ et $||\beta_t^i|| = 2$. Le second, S_B, contient de larges accélérations et a ses paramètres fixés à $||\alpha_t|| = 15$ et $||\beta_t^i|| = 5$ (unité en pixels par trame^2).

Évaluation qualitative

Nous avons mené une évaluation qualitative en comparant les trajectoires extraites par l'algorithme de suivi avec les vraies trajectoires obtenues sur les séquences générées.

Pour chaque scénario, les algorithmes sont évalués sur une séquence de 100 trames. Chaque trajectoire c calculée est comparée à la trajectoire de référence r commençant à la même position, sur la même trame. Plus précisément, soit $T_c = \{\mathbf{p}_s^c, \ldots, \mathbf{p}_e^c\}$ une trajectoire estimée, démarrant à la trame s et finissant à la trame e. La trajectoire correspondante $T_r = \{\mathbf{p}_s^r, \ldots, \mathbf{p}_f^r\}$, tel que $\mathbf{p}_s^r = \mathbf{p}_s^c$, et f la dernière trame de la trajectoire de référence.

1. L'erreur moyenne de T_c est définie par :

$$\sum_{s \leq t \leq \min(e,f)} \frac{||\mathbf{p}_t^c - \mathbf{p}_t^r||}{1 + \min(e,f) - s}$$

2. La particule est considérée perdue si $(f - e) > \eta$.
3. Il y a une occultation non détectée si $(e - f) > \eta$.

Nous fixons $\eta = 10$ trames pour permettre de petites erreurs de suivi, qui représentent 0.40 seconde dans un flux vidéo à 25 Hz. Pour chaque séquence et chaque algorithme, nous calculons donc :

1. l'erreur moyenne des trajectoires qui mesure la capacité de l'algorithme à suivre une particule sans dériver.
2. le pourcentage de particules perdues, qui montre la robustesse aux accélérations
3. le pourcentage d'occultations non détectées.

Dans ces trois statistiques toutes les trajectoires ont le même poids.

Le tableau 3.1 compare *Video Extruder* avec l'implantation de pyrLK d'OpenCV [7] configurée avec différentes tailles de fenêtre (WS) sur les scénarios S_A et S_B. Ces résultats montrent que, tout en étant significativement plus rapide, la qualité de *Video Extruder* est proche de celle de pyrLK en termes d'erreur de suivi. Les mesures 2 et 3 sont en faveur de pyrLK parce que, contrairement à *Video Extruder*, cette approche ne met pas à jour les descripteurs de points au cours du temps. Cela permet de détecter les occultations plus facilement, surtout quand la fenêtre du descripteur est occultée de manière très progressive. Cependant, en général, ne pas mettre à jour les descripteurs offre moins de robustesse aux changements d'apparence (non modélisés dans nos scénarios).

Une observation contre-intuitive est que le scénario S_A, bien que mettant en scène des mouvements plus petits, est en fait plus difficile car il contient des occultations lentes causant une dérive progressive des composantes du descripteur trop petite pour être détectée comme occultation.

Comparaison avec un calcul de flux optique dense

Notre approche étant un intermédiaire entre le suivi de points épars et le calcul d'un flux optique dense, il est aussi légitime de la comparer avec ce dernier. Cependant, la comparaison ne peut pas être faite directement comme avec pyrLK, parce que la sortie de la méthode est différente. Nous avons donc réalisé une comparaison limitée avec Farnebäck [17], une méthode de calcul de flux optique dense de l'état de l'art implantée dans la bibliothèque OpenCV.

Étant donné que cette méthode ne fournit pas de trajectoires de points mais un champ de vecteurs de déplacement dense, nous ne pouvons pas évaluer la qualité des trajectoires. À la place, nous évaluons la qualité des vecteurs de flux optique. Comme nos séquences synthétiques fournissent un flux optique de référence, l'erreur moyenne par pixel peut être facilement estimée. Le tableau 3.2 montre l'erreur moyenne des pixels pour les deux scénarios S_A et S_B.

L'intérêt de cette comparaison est de mieux évaluer les enjeux de la sparsité, densité, et semi-densité dans un contexte de temps réel. Les valeurs ne sont pas en faveur de Farnebäck [17] car elles intègrent l'erreur moyenne sur tous les pixels, incluant les zones sans texture où les erreurs sont les plus

importantes. Cependant, nos expériences ont montré qu'en moyenne, Farnebäck fournit une meilleure mise en correspondance que *Video Extruder* sur les points extraits par le détecteur. Cela justifierait donc l'utilisation d'un flux optique pour guider les particules comme le fait l'algorithme *Particle Video* [57]. Mais *Particle Video* n'est pas temps réel et le coût d'un algorithme comme Farnebäck est un ordre de magnitude plus élevé que *Video Extruder* (Voir tableau 3.3).

3.2.5 Implantation logicielle

Accélération sur CPU et GPU

Ces dernières années ont vu une accélération des ventes de *smartphones*, poussant les constructeurs de processeurs à accroître significativement les capacités de processeurs basse consommation. Et comme l'augmentation de la cadence d'horloge a atteint des limites physiques (avec les technologies de gravure actuelles) il y a déjà quelques années, le parallélisme a été la stratégie choisie par toute l'industrie pour augmenter les capacités de calcul sans augmenter la consommation d'énergie.

Pour profiter de cette tendance, nous avons conçu *Video Extruder* principalement sur deux types d'opérations massivement parallèles :

- **Des opérations locales aux pixels**, opérées au début du pipeline de suivi semi-dense : les deux flous gaussien du calcul du détecteur, et le détecteur de points d'intérêt. Ces opérations peuvent être séparées en autant de fils d'exécution indépendants qu'il y a de pixels dans l'image.
- **Des opérations locales aux particules**, incluant la mise en correspondance et le filtrage des erreurs. Pour maximiser l'utilisation du cache processeur, nous empaquetons l'ensemble des particules dans un tableau contigu en mémoire. Bien qu'il y ait moins de particules que de pixels dans l'image, leur nombre est assez grand pour occuper la totalité d'un GPU, nécessitant quelques milliers de threads pour fonctionner à plein régime.

Dans l'implantation GPU, nous utilisons un thread par pixel ou particules. Sur le CPU, capable d'exécuter seulement 4 threads en parallèle, nous séparons simplement l'image ou le tableau de particules en 4 sous-parties,

TABLE 3.1 – Comparaison de PyrLK [7], utilisant 3 différentes fenêtres d'intégration (WS), avec *Video Extruder*. Les particules perdues et les occultations non détectées sont exprimées en proportion du nombre de particules. Les erreurs de suivi sont exprimées en nombre de pixels par particule.

	PyrLK WS=5	PyrLK WS=11	PyrLK WS=21	Video Extruder
Scénario S_A - 5 000 particules				
Erreur de suivi	1.74	0.92	1.03	1.12
Particules perdues	8.27 %	8.26 %	7.91 %	8.82 %
Occultations non détectées	10.80 %	1.41%	3.39 %	12.82 %
Scénario S_B - 5 000 particules				
Erreur de suivi	3.62	1.12	1.18	0.94
Particules perdues	10.59 %	7.99 %	7.88 %	8.48 %
Occultations non détectées	9.93 %	1.01%	1.95 %	5.47 %
Scénario S_A - 15 000 particules				
Erreur de suivi	2.45	0.99	1.03	1.14
Particules perdues	8.92 %	8.39	8.54 %	8.33 %
Occultations non détectées	15.00 %	1.12 %	1.98 %	9.28 %
Scénario S_B - 15 000 particules				
Erreur de suivi	3.82	1.19	1.19	0.95
Particules perdues	12.80 %	8.13 %	8.56 %	8.74 %
Occultations non détectées	8.98 %	0.72 %	1.19 %	4.11 %

TABLE 3.2 – Erreur moyenne en pixels par vecteur de flux optique estimé avec la méthode de Farnebäck [17] avec différentes tailles de fenêtre d'intégration (WS).

	Farnebäck WS=5	Farnebäck WS=15	Farnebäck WS=25
Scénario S_A	3.25	2.14	1.73
Scénario S_B	5.35	3.46	2.86

chacune calculée sur un cœur différent.

Pour limiter le temps de développement, nous avons limité nos implantations à un parallélisme simple. Les techniques plus complexes, comme l'utilisation de la mémoire partagée GPU, ou l'utilisation explicite d'instructions (SIMD), n'ont pas été utilisées.

Les implantations CPU et GPU sont open source et disponibles en ligne : http://www.ensta-paristech.fr/~garrigues/video_extruder.html

Conteneur de particules

Pour accélérer le suivi semi-dense et les applications utilisant les faisceaux de trajectoires (voir chapitre 5), nous avons implanté un conteneur permettant :

— de fournir à chaque image un tableau de particules contigu en mémoire, afin d'itérer efficacement sur toutes les particules présentes dans l'image courante.

— l'accès en temps constant à la particule située à des coordonnées données.

Nous notons le tableau de particules P et l'accès aux particules est fait *via* l'index R tel que $R(\mathbf{p}) = i$ si $P(i)$, la $i$ème particule, a pour coordonnées $\mathbf{p}$.

Maintenir une telle structure en mémoire n'est pas un tâche triviale. Les particules apparaissent et disparaissent à chaque trame et P doit être mis à jours pour refléter ces changements. Un l'algorithme de compaction est utilisé pour conserver le caractère contigu du tableau de particules. Cette étape

déplace les particules en mémoire et complique l'attachement de données aux particules, sur lequel repose les applications du suivi semi-dense (cf chapitre 5). Pour pallier ce problème, nous fournissons une procédure permettant de synchroniser un tableau d'attributs avec le tableau de particules, tout en sauvegardant les données des particules venant de disparaître. Cette procédure est d'ailleurs utilisée par *Video Extruder* pour attacher un historique de position (trajectoire) à une particule.

Pour augmenter la localité mémoire des algorithmes de traitement d'image, nous trions les particules toutes les N trames, de telle sorte que les particules proches dans l'espace 2D soient aussi proches en mémoire. Pour cela, nous avons implanté un algorithme de compaction parallèle gros grains sur le CPU, et utilisé la compaction GPU de la bibliothèque Thrust [29].

Comparaison des temps de calcul

Finalement, la densité et la qualité du suivi de *Video Extruder* sont comparables à celles de pyrLK comme montré en section 3.2.4. Son principal avantage est son temps d'exécution plus faible. Nous montrons dans cette section qu'il surpasse pyrLK, atteignant une fréquence de traitement adaptée aux applications temps réel même sur des processeurs basse consommation tels que ceux embarqués dans des téléphones.

Le tableau 3.3 présente les temps de calcul des 3 méthodes sur la séquence urbaine du jeu de données Camvid [9]. *Video Extruder* se révèle 1.5 à 17 fois plus rapide que pyrLK selon les tailles de fenêtre.

Le graphique 3.10 montre la répartition du temps de calcul entre les différentes parties du suivi semi-dense. Les deux tâches les plus coûteuses sont la mise en correspondance et les deux flous gaussien estimant les deux échelles du descripteur. Les temps de calcul des autres parties sont négligeables car elles ne nécessitent que quelques accès mémoire par particule et quelques calculs arithmétiques simples.

Nous avons réalisé trois implantations de *Video Extruder*. Les deux premières fonctionnent sur des architectures de processeurs non embarqués, un CPU et un GPU, et atteignent respectivement une cadence de traitement de 152 et 166 trames par seconde. La troisième cible est le processeur basse

TABLE 3.3 – Comparaison des temps de calcul, moyennés sur les 1000 premières trames d'une vidéo de taille 640×480 tirée du jeu de données CamVid. PyrLK et *Video Extruder* ont été paramétrés pour suivre 8500 particules par trame. Pour *Video Extruder*, le détecteur a été exécuté toutes les 5 images. Comparaison faite sur un processeur 4-Cœurs Intel i5 2500k cadencé à 3.3 GHz.

	MPixels/s (Trames/s)	μs par particule
PyrLK $WS = 5$	31 (101.11)	1.21
PyrLK $WS = 11$	9.6 (31.25)	3.50
PyrLK $WS = 21$	2.7 (8.95)	14.44
Färneback (dense)	3.4 (11.1)	0.29 (par pixel)
Video Extruder	46.66 (151.9)	0.77

consommation à jeu d'instructions ARM, pour lequel on atteint une cadence de 10 trames par seconde sur des résolutions plus faibles. À notre connaissance, il s'agit des premières implantations capables de suivre un champ de particules aussi dense à des cadences aussi élevées. Il faut noter que, mis à part les convolutions, aucune des implantations ne reposent sur des optimisations logicielles avancées, ce qui laisse de la marge pour des optimisations futures. Les temps des calculs sur les différentes architectures testées sont présentés dans le tableau 3.4.

3.2.6 Conclusion et perspectives

Nous avons proposé un pipeline complet d'extraction d'un champ semi-dense de trajectoires de points d'intérêt appelé *Video Extruder*. Il a été conçu pour être utilisé comme primitive pour différentes applications embarquées. En effet, son parallélisme et son minimalisme lui confèrent une rapidité de calcul lui permettant d'être embarqué sur différentes architectures à haute et basse consommation. En plus de son temps de calcul avantageux, le champ de trajectoires calculé combine hautes résolutions spatiale (semi-densité), et

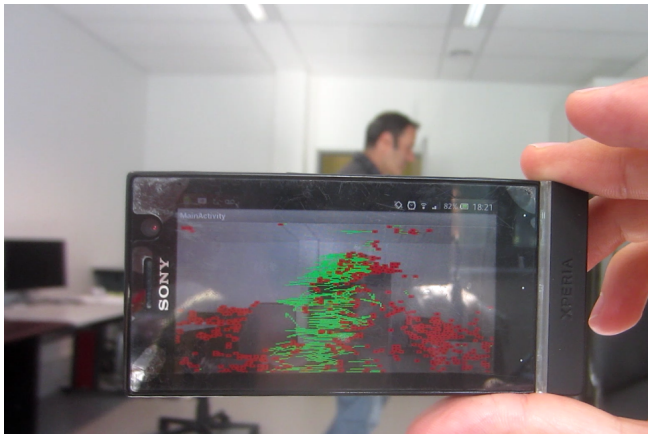

FIGURE 3.9 – *Video Extruder* fonctionne à 10Hz sur un smartphone d'entrée de gamme Xperia U embarquant un processeur ARM dual-core STE U8500. Vidéo disponible sur youtube : https://www.youtube.com/watch?v=jHNSshAsEJ4

TABLE 3.4 – Temps de calcul de *Video Extruder* sur différentes architectures (avec une fréquence de détecteur et de filtrage des erreurs fixée à 5Hz). Le nombre de cycles par particule (colonne notée cpp) correspond au temps de calcul (en cycles d'horloge) d'une image divisé par le nombre de particules présentes sur cette dernière.

Architecture	Résolution	Particules	Mpixels/s (Trames/s)	cpp
GPU Geforce GTX 460 1.35GHz	640×480	8500	50 (166)	957
CPU quad-core I5 2500k 3.3GHz	640×480	8500	46 (152)	2550
ARM dual-core STE U8500 1GHz	320×240	3000	0.84 (11)	30 300
ARM single-core IMX.53 1GHz	720×288	2000	2.07 (10)	50 000

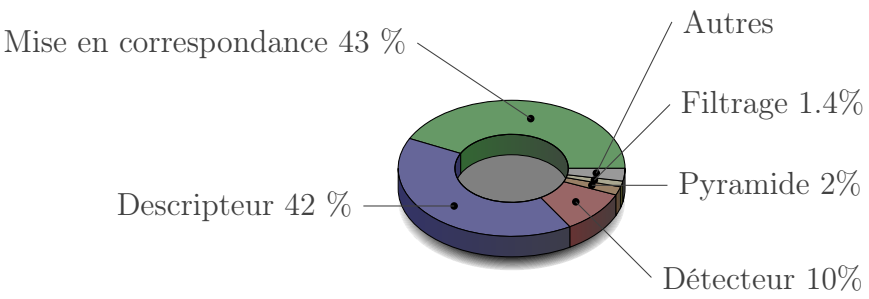

FIGURE 3.10 – Répartition du temps de calcul sur les différentes parties de l'algorithme fonctionnant sur le processeur i5-2500k.

temporelle (trajectoires long-terme). Cela rend l'approche utile pour beaucoup d'applications, entre autres le suivi d'objets non rigides, la reconstruction 3D semi-dense, la stabilisation vidéo, la segmentation par le mouvement ou encore la reconnaissance d'action. Nous l'avons utilisé pour construire des descripteurs d'actions [47, 46]. Dans [46], nous avons montré que les performances de la classification d'action augmentent avec le nombre de trajectoires jusqu'à une certaine limite : quand la qualité du suivi commence à décroître. Ces travaux ont confirmé l'intérêt du suivi semi-dense. Les applications auxquelles nous avons participé seront présentées dans le chapitre 5.

En plus du pipeline de suivi semi-dense ces travaux ont abouti aux contributions suivantes :

— une méthode d'évaluation de détection de points semi-dense, prenant en compte les performances de l'algorithme de mise en correspondance sur les ensembles de points détectés.
— un nouveau détecteur de points rapide pour la sélection de particules.
— une évaluation de la stratégie de sélection *blockwise* (BM), permettant d'extraire plus de points que la sélection des maxima locaux classique.
— une méthode à coût de calcul réduit permettant de prédire les futures positions de particules, prenant en compte à la fois l'information cinématique (prédiction temporelle) et spatiale (prédiction liée aux échelles supérieures).
— un descripteur de points à deux échelles avec une faible empreinte mémoire combiné à une mise en correspondance fondée sur une descente

de gradient à deux échelles.
— une méthode d'évaluation d'algorithmes de calcul de trajectoires (suivi long terme).

3.3 Estimation éparse du mouvement

3.3.1 Introduction

Pour certaines applications, comme l'odométrie visuelle, la densité de l'estimation du mouvement est moins importante que sa robustesse, car en théorie, moins d'une dizaine de vecteurs de flux optique sont nécessaires pour estimer les six degrés de liberté du mouvement de la caméra.

Pour atteindre le niveau de robustesse requis, on a souvent recours à des descripteurs de points de grande dimension, à la fois capables d'identifier un point parmi plusieurs milliers, et assez robustes pour conserver ce pouvoir d'identification malgré des changements d'apparence.

D'autre part, la méthode de calcul des vecteurs de l'estimation éparse de flux optique est différente de celle de l'estimation semi-dense : étant donné deux images I_1 et I_2, les points d'intérêt sont d'abord détectés et leurs descripteurs extraits. Puis, pour chaque point d'intérêt de I_1, le point avec le descripteur le plus similaire est recherché dans I_2. La similarité choisie est en général une distance euclidienne, de Manhattan, ou de Hamming entre les descripteurs. Contrairement à l'estimation semi-dense, les performances de cette méthode dépendent directement de la répétabilité du descripteur de point.

Le nombre de points d'intérêt et la taille de leurs descripteurs font que la recherche exhaustive du plus proche voisin (ou *brute force*) est trop longue à calculer pour des applications temps réel. Cela limite les applications à n'utiliser qu'un faible nombre de points pour limiter le temps de calcul ou à effectuer la mise en correspondance de façon différée.

Pour répondre à ce problème, la bibliothèque FLANN [44] indexe l'ensemble des points d'arrivée grâce à des k-d trees [5] aléatoires. À chaque nœud de l'arbre, l'ensemble de descripteurs est séparé en deux selon une dimension choisie aléatoirement parmi les 5 pour lesquelles il présente la plus

grande variance. Pour accélérer la recherche de plus proche voisin sur les processeurs multicœurs, plusieurs arbres sont construits et parcourus en parallèle. De plus, pour accélérer la recherche du plus proche voisin, le nombre de feuilles à examiner est borné par le paramètre *checks*. Nous considérons cette approche comme l'état de l'art. Même si elle est plus rapide que la recherche exhaustive, les temps de calcul restent trop élevés pour le suivi en temps réel de points dans une vidéo.

Dans cette étude, nous avons proposé des optimisations à la fois algorithmiques et logicielles pour la détection et la mise en correspondance de deux ensembles de points d'intérêt, dans le but de pouvoir en bénéficier dans des applications temps réel.

La démarche a consisté dans un premier temps (section 3.3.2) à accélérer le détecteur de points d'intérêt FAST [55] en distribuant le calcul sur les multiples cœurs des processeurs actuels et leurs unités de calcul vectoriel SIMD (Single Instruction Multiple Data).

Ensuite, sachant qu'il est souvent possible, dans une vidéo, de prédire les déplacements observés entre deux trames, nous avons limité la recherche de chaque point à un voisinage de taille restreinte autour de cette prédiction. Cette optimisation, présentée en section 3.3.4, bénéficie d'une indexation des points sur leur localisation dans l'image et diminue significativement le nombre de comparaisons nécessaires à chaque mise en correspondance.

De plus, nous avons proposé en section 3.3.5 une indexation des descripteurs de points permettant d'accélérer davantage la recherche. Cette indexation permet de diminuer l'espace de recherche, et d'accélérer la recherche en la rendant approximative.

Dans les deux indexations présentées ici, nous avons travaillé sur deux fronts : les **optimisations algorithmiques**, qui consistent à diminuer le nombre d'appels à la fonction distance, et les **optimisations logicielles** permettant de mieux rentabiliser le cache du processeur et son parallélisme.

Pour valider notre approche, nous avons comparé ses temps de calcul et son taux d'erreurs à ceux de FLANN sur la base de données *KITTI optical flow* [25]. Les temps de calcul se sont révélés jusqu'à 100 fois moindres pour des taux d'erreurs similaires. Une application a aussi été développée pour téléphone Android et traite plus de 10 images par seconde.

3.3.2 Accélération multi-cœur et vectorielle du détecteur FAST

Dans les chaînes d'estimation de mouvement, le détecteur de points intervient en général en premier. Il prend en entrée l'image brute et la réduit en un ensemble de points d'intérêt. Cette opération a pour but d'ignorer les zones difficiles à suivre et de réduire la quantité de données à traiter par la suite de la chaîne.

FAST est un détecteur de points connu pour sa simplicité et sa rapidité de calcul. Cependant, les implantations actuelles (libcvd et OpenCV) ne tirent que partiellement parti des processeurs actuels : elles n'exploitent que les instructions vectorielles SSE. Cela pose deux problèmes : le code ne tire parti que d'un seul cœur et n'est pas exécutable sur des processeurs utilisant des instructions vectorielles différentes (AVX, NEON).

Afin de réduire le temps de calcul de la chaîne de suivi de points épars, nous avons proposé une implantation de FAST reposant sur la parallélisation de l'exécution sur plusieurs cœurs et les trois jeux d'instructions vectoriels les plus courants : SSE, AVX et NEON (armv7 et armv8). Cette implantation a été intégrée à la bibliothèque open source Vidéo++ [23].

Pour simplifier le support de différents jeux d'instructions, nous avons séparé le cœur de l'algorithme FAST du jeu d'instructions utilisé. Cette séparation a été réalisée grâce à l'utilisation de templates C++ et n'a aucun impact sur le temps de calcul.

D'autre part, afin de minimiser l'utilisation de synchronisation inter-threads nécessaire pour la construction d'une liste de points d'intérêt unique, des tableaux temporaires sont alloués sur la pile et purgés dans la liste une fois remplis. En plus de minimiser l'attente entre threads, l'utilisation de tableaux sur la pile évite des allocations mémoire dynamiques coûteuses.

Pour vérifier l'efficacité de ces optimisations, nous avons comparé les temps de calcul de l'implantation d'OpenCV avec la nôtre, pour différents seuils de détection sur une image de 500×500 pixels. Les versions avec (graphique 3.11) et sans (graphique 3.12) suppression des non-maxima locaux ont été évaluées. Pour la version sans, nous avons pu observer un facteur d'accélération allant jusqu'à **10x** sur un processeur 4-cœurs. Les gains sont plus faibles, jusqu'à

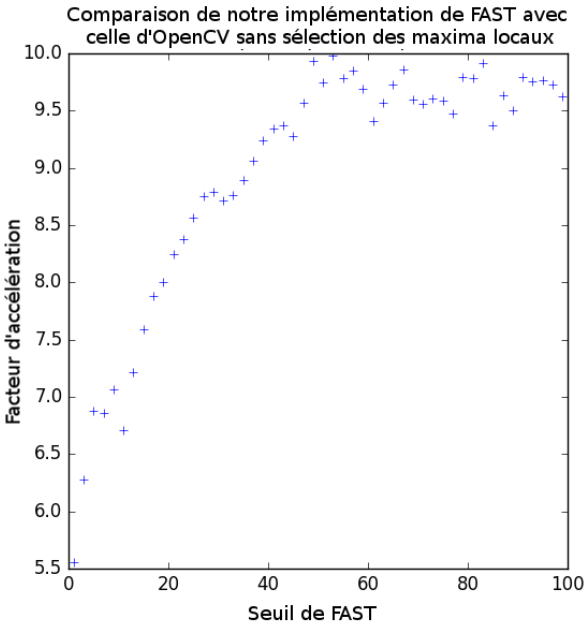

FIGURE 3.11 – Facteur d'accélération de l'implantation de FAST ($n = 9$) sans extraction des maxima locaux en fonction du seuil de contraste. Ces chiffres ont été obtenus avec un processeur Intel 4-cœurs i7-4700HQ.

5.5x, pour la version avec, car les scores FAST doivent être calculés pour détecter les maxima locaux.

Nous avons pu observer que plus le seuil de FAST est élevé, plus le champ de points est épars et moins le calcul des scores FAST tire parti du cache processeur. Cela a pour conséquence une réduction du facteur d'accélération de la version avec suppression des non-maxima locaux pour des faibles nombres de points extraits.

3.3.3 Description des points d'intérêt

Dès les points d'intérêt extraits, nous extrayons de l'image une description de chacun. Ces descripteurs ont pour unique but d'identifier de manière la plus discriminante possible un point d'intérêt, tout en étant robuste aux changements d'apparence possibles entre les deux images à mettre en correspondance.

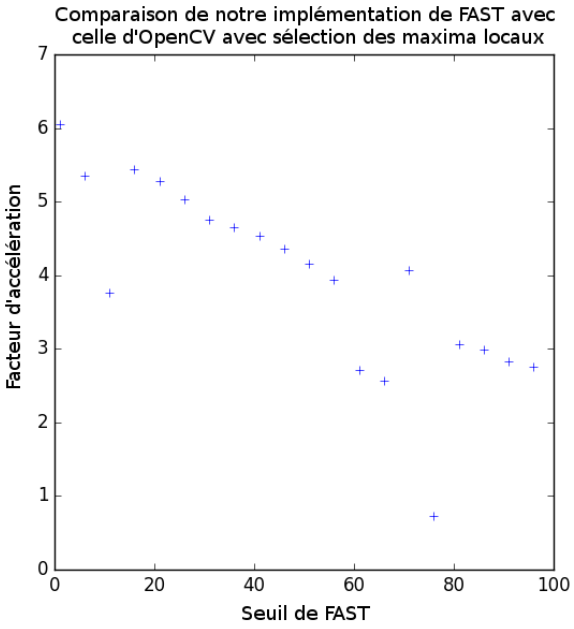

FIGURE 3.12 – Facteur d'accélération de l'implantation de FAST ($n = 9$) avec extraction des maxima locaux en fonction du seuil de contraste. Ces chiffres ont été obtenus avec un processeur Intel 4-cœurs i7-4700HQ.

Comme nous ciblons dans cette étude les vidéos avec peu de changements entre deux trames consécutives, et étant donné nos objectifs de réduction du temps de calcul, nous avons choisi un descripteur simple et rapide à extraire : le patch de pixels de taille 21×21 centré sur chaque point d'intérêt. Ce descripteur encode donc une coordonnée dans un espace à 441 dimensions.

Dans la suite, nous proposons une méthode pour accélérer la recherche de plus proche voisin (ou mise en correspondance) dans cet espace. Notons que la méthode présentée par la suite n'est pas dépendante du descripteur choisi et peut être appliquée à d'autres.

3.3.4 Indexation des points sur leur localisation

Sachant qu'entre deux trames vidéo consécutives les normes des déplacements de points sont en général prédictibles, nous avons restreint l'ensemble de recherche de chaque mise en correspondance à l'ensemble des points présents dans un cercle centré sur la prédiction de la position du point à mettre en correspondance.

Toutefois, sans structure de données adaptée, cette requête nécessite le parcours de la totalité des points de l'image pour en extraire les points inclus dans le cercle. Pour éviter ce parcours, nous avons indexé les points en fonction de leurs coordonnées avec la méthode suivante : étant donnée une grille de $Gs \times Gs$ cellules recouvrant le domaine de l'image, les coordonnées de tous les points situés dans la même cellule sont regroupées dans une liste. Ces listes sont indexées sur les lignes et les colonnes de l'image. Ainsi, l'accès à la liste de points de la cellule recouvrant un pixel donné se fait en temps constant. Dans nos expériences sur la base de données KITTI, la valeur $Gs = 6$ a fourni les meilleurs temps de calcul.

Cette structure a deux avantages pour la mise en correspondance de points : elle accélère la récupération des points inclus dans le cercle de recherche, et améliore la **localité des accès mémoire** d'une recherche. En effet, comme les points proches dans l'image sont aussi proches en mémoire, la recherche tire meilleur parti du cache processeur.

3.3.5 Réduction de l'espace de recherche par projection dans un espace 1D

L'index présenté précédemment permet de restreindre l'ensemble de recherche. Toutefois, quand la mise en correspondance repose sur une distance somme des différences au carré (SSD) ou somme des différences absolues (SAD), cet ensemble restreint peut être à nouveau réduit. Dans la suite, nous considérons la distance SAD, mais toutes les propriétés évoquées sont généralisables à toutes les distances de Minkowski, incluant la distance SSD. Aussi, pour des raisons de simplicité, nous présentons cette optimisation indépendamment de celle de la section précédente, même si les deux sont combinées dans l'algorithme final.

La réduction de l'ensemble de recherche repose sur l'inégalité suivante, vraie pour toute distance de Minkowski :

$$\sum_i |\mathbf{a}_i - \mathbf{b}_i|^p \geq \left| \sum_i \mathbf{a}_i - \sum_i \mathbf{b}_i \right|^p$$

et donc en particulier (pour $p = 1$) :

$$SAD(\mathbf{a}, \mathbf{b}) \geq \left| \sum_i \mathbf{a}_i - \sum_i \mathbf{b}_i \right| \tag{3.4}$$

Dans la suite nous notons $ADS(\mathbf{a}, \mathbf{b}) = |\sum_i \mathbf{a}_i - \sum_i \mathbf{b}_i|$.

Si nous connaissons une borne supérieure s de la distance entre un point $\mathbf{p}$ de l'ensemble de départ et son homologue dans l'ensemble d'arrivée $\mathcal{B}$, l'ensemble de recherche peut être restreint à l'ensemble de points $\mathcal{R}(\mathbf{p}, s)$ suivant :

$$\mathcal{R}(\mathbf{p}, s) = \{\mathbf{q} \in \mathcal{B} : ADS(\mathbf{p}, \mathbf{q}) < s\}$$

Pour accélérer la recherche, nous indexons et trions chaque descripteur $\mathbf{a}$ en fonction de sa somme $\sum_i \mathbf{a}_i$. Cela permet deux optimisations : quel que soit $\mathbf{p}$ et s, l'ensemble $\mathcal{R}(\mathbf{p}, s)$ est **contigu en mémoire** donc plus rapide à parcourir, et les indices de ses bornes peuvent être **résolus en temps constant**.

En plus de fournir un accès rapide à l'ensemble $\mathcal{R}(\mathbf{p}, s)$, l'index permet un accès rapide au point $\mathbf{q}_{init}$ le plus proche de $\mathbf{p}$ suivant la distance ADS :

$$\mathbf{q}_{init} = \arg\min_{\mathbf{q}\in\mathcal{B}} ADS(\mathbf{q}, \mathbf{p})$$

Étant donné que $\mathbf{q}_{init}$ sera probablement proche de $\mathbf{p}$ selon la distance SAD, initialiser la borne s avec $SAD(\mathbf{p}, \mathbf{q}_{init})$ restreint l'ensemble de recherche $\mathcal{R}(\mathbf{p}, s)$.

Si au cours de la recherche, un point s'avère plus proche de $\mathbf{p}$ que $\mathbf{q}_{init}$ selon la distance SAD, alors la borne s peut être mise à jour, réduisant davantage l'espace de recherche $\mathcal{R}$.

Finalement, pour accélérer la décroissance de s et donc de l'ensemble $\mathcal{R}(\mathbf{p}, s)$, nous parcourons les points du plus proche au plus éloigné de $\mathbf{p}$ selon la distance ADS.

3.3.6 Accélération par l'approximation

La réduction de l'espace de recherche présentée dans la section précédente permet de réduire le nombre de calculs de distance SAD sans affecter la qualité de la mise en correspondance. Nous présentons ici une modification de cette optimisation qui rend la recherche approximative, et permet un compromis entre le nombre d'erreurs en correspondance et le temps de calcul.

En effet, le problème de l'inégalité 3.4 est que la distance $|\sum_i \mathbf{a}_i - \sum_i \mathbf{b}_i|$ est inférieure à la distance SAD et donc l'ensemble des points $\mathbf{q}$ vérifiant $|\sum_i \mathbf{p}_i - \sum_i \mathbf{q}_i| < s$ est de bien plus grande taille que l'ensemble vérifiant $\sum_i |\mathbf{p}_i - \mathbf{q}_i| < s$.

Pour pallier ce problème, nous proposons d'introduire un nouveau paramètre $\alpha \geq 1$ et de restreindre l'espace de recherche $\mathcal{R}$ de la façon suivante :

$$\mathcal{R}_\alpha(\mathbf{p}, s) = \{\mathbf{q} \in \mathcal{B} : \alpha \times ADS(\mathbf{p}, \mathbf{q}) < s\}$$

Les graphiques 3.13 et 3.14 montrent l'évolution du pourcentage d'erreurs de mise en correspondances supérieures à 10 pixels et le temps de calcul sur une paire d'images de la base KITTI. Plusieurs valeurs de α ont été évaluées. Ces chiffres montrent que ce paramètre permet de régler le compromis entre

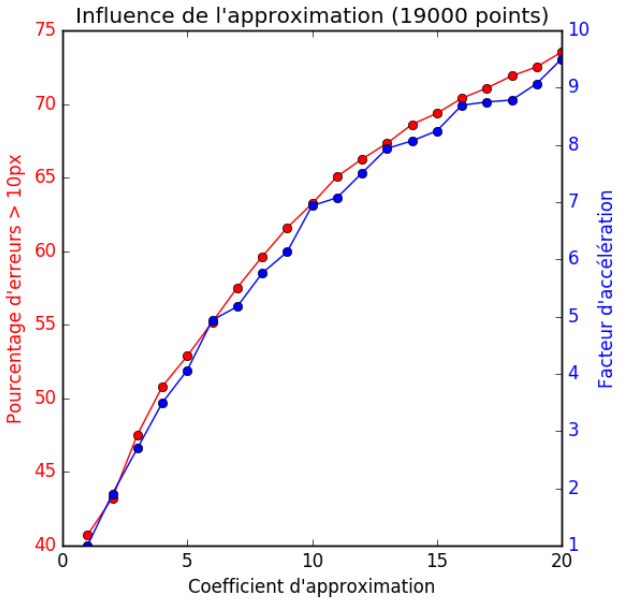

FIGURE 3.13 – Influence du facteur d'approximation α sur le temps de calcul et le pourcentage d'erreurs, pour 19000 points. Ces chiffres ont été obtenus avec un processeur Intel 4-cœurs i7-4700HQ.

rapidité de calcul et qualité de mise en correspondance. Nous avons aussi remarqué que plus le nombre de points est grand, plus le ratio accélération du calcul / augmentation des erreurs est intéressant.

3.3.7 Optimisations du calcul de la distance SAD

Pour tirer parti des instructions SIMD, nous avons découpé en blocs la boucle calculant la somme. Ces blocs sont écrits sous forme de boucle *for* avec des bornes connues à la compilation pour permettre au compilateur de les vectoriser automatiquement.

Ensuite, comme expliqué dans la section précédente, la mise en correspondance se fait via une recherche du point le plus proche. Étant donné que pendant le parcours, la distance minimum s est conservée au cours de la recherche, nous pouvons limiter le temps de calcul de la distance elle-même en l'arrêtant dès que le calcul de la distance dépasse s.

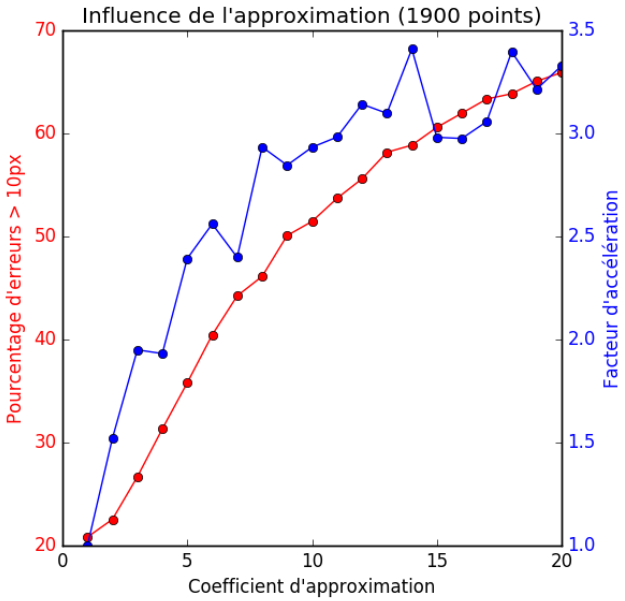

FIGURE 3.14 – Influence du facteur d'approximation α sur le temps de calcul et le pourcentage d'erreurs, pour 1900 points. Ces chiffres ont été obtenus avec un processeur Intel 4-cœurs i7-4700HQ.

Pendant nos expériences, l'utilisation de blocs de 128 éléments a fourni les temps de calcul les plus faibles.

3.3.8 Réduction de l'empreinte mémoire des descripteurs

Pour diminuer la quantité de données manipulée par l'algorithme, nous avons discrétisé les composantes des descripteurs sur des entiers 8 bits, soit quatre fois moins que les flottants utilisés par OpenCV. En plus de réduire l'empreinte mémoire, cela permet d'utiliser l'arithmétique entière pour le calcul de la distance, plus rapide que l'arithmétique flottante.

3.3.9 Parallélisation de la recherche

Les mises en correspondance de chaque point de l'espace de départ $\mathcal{A}$ étant indépendantes les unes des autres, nous les avons distribuées sur les multiples cœurs du CPU grâce aux directives OpenMP.

3.3.10 Évaluation et comparaison

Pour évaluer l'impact de nos propositions sur le temps de calcul et sur le taux d'erreurs, nous les avons testées sur les 193 paires d'images de la base de données « KITTI optical flow » [25] pour les comparer aux *k-d trees* aléatoires de FLANN.

Pour évaluer chaque algorithme de mise en correspondance, nous nous sommes appuyés sur le protocole d'évaluation suivant : dans un premier temps et pour chaque paire d'images, les points sont détectés et leurs descripteurs extraits. Ils sont ensuite mis en correspondance avec l'algorithme en question. Finalement, le temps de calcul et le taux d'erreurs supérieurs à 10 pixels sont moyennés sur toutes les paires d'images. Pour chaque algorithme, plusieurs nombres de points d'intérêt ont été testés.

Les différents algorithmes évalués sont les suivants :

— **la recherche exhaustive d'OpenCV :** la recherche exhaustive proposée par OpenCV. Pour chaque mise en correspondance, l'intégralité de l'ensemble de recherche est parcouru.

— **notre recherche exhaustive (BF) :** notre implantation de la recherche exhaustive, utilisant les optimisations de la distance SAD et la parallélisation sur des multicœurs.
— **FLANN (KDT):** implantation des k-d trees aléatoires [44]. Plusieurs compromis approximation / temps de calcul ont été testés grâce à plusieurs valeurs du paramètre *checks*. La version exacte (*unlimited checks*) et deux versions approximatives ($checks = 10$ et 50) ont été évaluées.
— **Index1 (I1):** indexation des points sur l'apparence proposée en 3.3.5 et 3.3.6 **sans restriction sur le domaine de recherche**. Plusieurs valeurs du paramètre α ont été évaluées. Cette approche bénéficie aussi de l'optimisation de la distance SSD et de la parallélisation multicœurs.
— **Index2 (I2):** indexation des points sur l'apparence proposée en 3.3.5 et 3.3.6 **avec un rayon de recherche fixé à 75 pixels** et l'indexation spatiale proposée en 3.3.4. Plusieurs valeurs du paramètre α ont été évaluées. Cette approche bénéficie de toutes les optimisations proposées.

Pour vérifier l'étendue de l'impact de ces travaux, nous avons mené notre évaluation sur deux descripteurs différents : le patch de pixels de taille 21x21 et le descripteur de points SIFT. Pour les deux, nous avons basé les algorithmes de mise en correspondance sur la distance de Manhattan. Notons que dans cette comparaison, seul le temps de calcul de la mise en correspondance a été mesuré : le temps pris par la détection de points et l'extraction des descripteurs n'a pas d'impact dans les chiffres présentés dans cette section.

Les résultats obtenus avec le descripteur PATCH (figures 3.16 et 3.15) montrent les impacts de nos différentes optimisations :

— la recherche exhaustive optimisée pour les processeurs multicœurs et utilisant les optimisations de la distance SAD est 10x plus rapide que celle d'OpenCV. Elle est aussi plus rapide que les versions approximatives (KDT) de FLANN pour des nombres de points inférieurs à 10 000.
— en y ajoutant la réduction de l'espace de recherche proposée en section 3.3.5 et 3.3.6 (I1), les temps de calcul obtenus sont entre 7 et 32 fois plus rapides que FLANN pour un taux d'erreur égal.

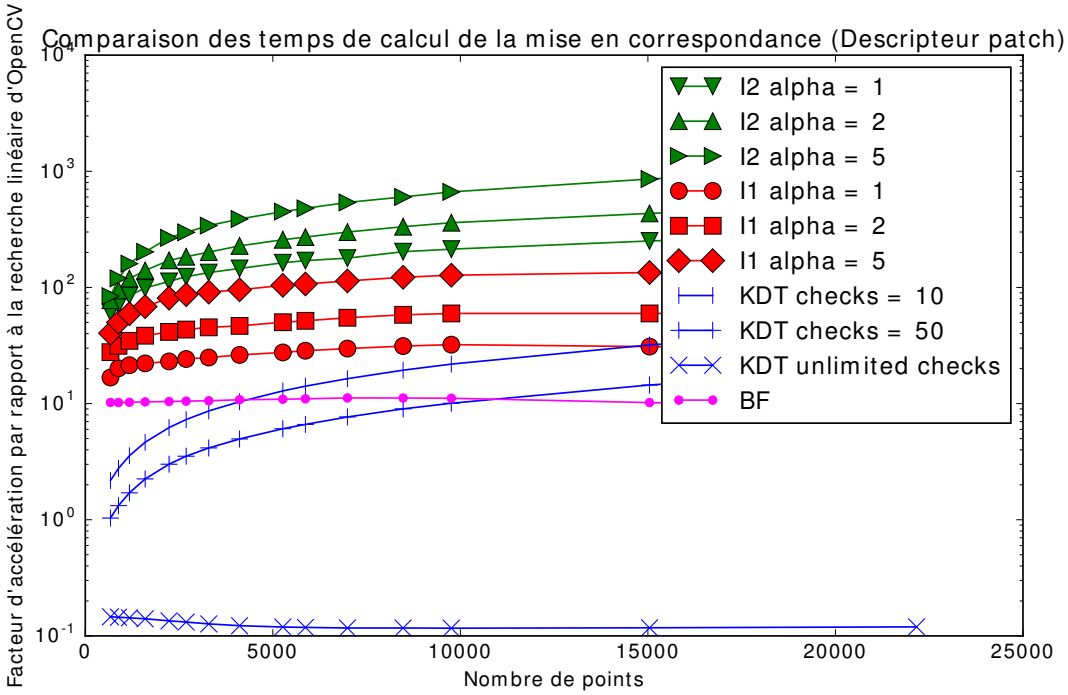

FIGURE 3.15 – Facteurs d'accélération des différents algorithmes de mise en correspondances de descripteurs PATCH par rapport à la recherche exhaustive d'OpenCV. Les optimisations proposées permettent de gagner jusqu'à deux ordres de grandeur sur les k-d trees aléatoires de la bibliothèque FLANN. Ces chiffres ont été obtenus avec un processeur Intel 4-cœurs i7-4700HQ.

— La combinaison de toutes nos propositions permettent des gains allant jusqu'à deux ordres de grandeur par rapport à FLANN pour un taux d'erreur égal.
— La réduction de l'espace de recherche autour de la prédiction de position permet de réduire le nombre d'erreurs de mise en correspondance.

Les résultats obtenus avec le descripteur SIFT à 128 dimensions (figures 3.18 et 3.17) montrent que ces gains, bien que plus faibles, subsistent pour des descripteurs à plus petite dimension.

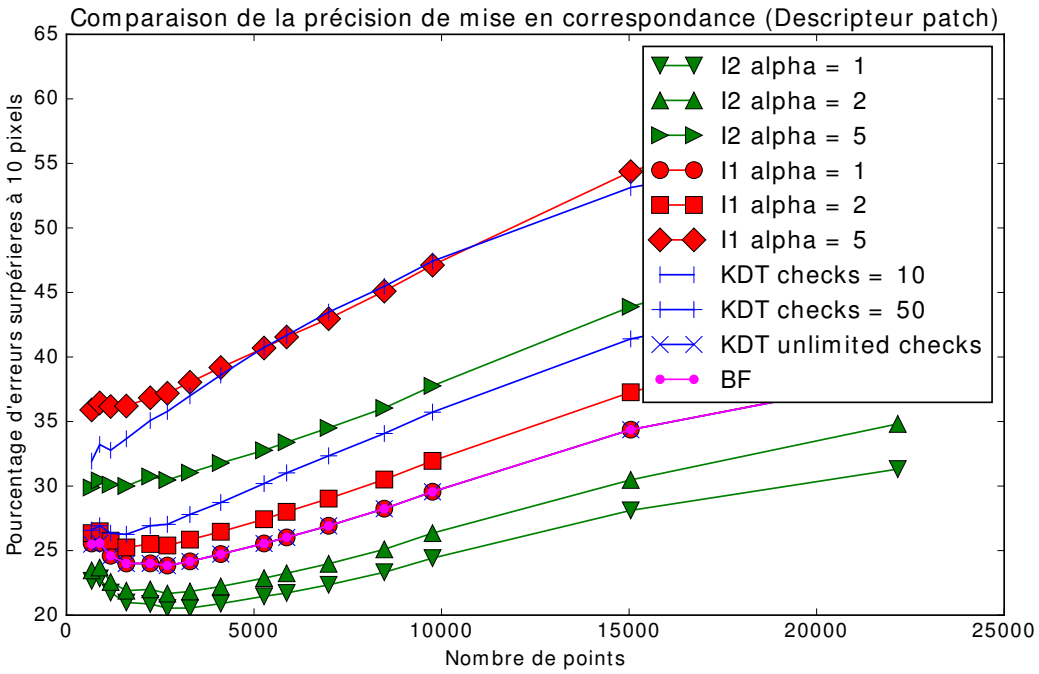

FIGURE 3.16 – Comparaison du nombre d'erreurs supérieures à 10 pixels entre notre approche et FLANN pour la mise en correspondance de descripteurs PATCH. La restriction du rayon de recherche, intégrée dans I2, permet de diminuer le nombre d'ambiguïtés lors de la mise en correspondance.

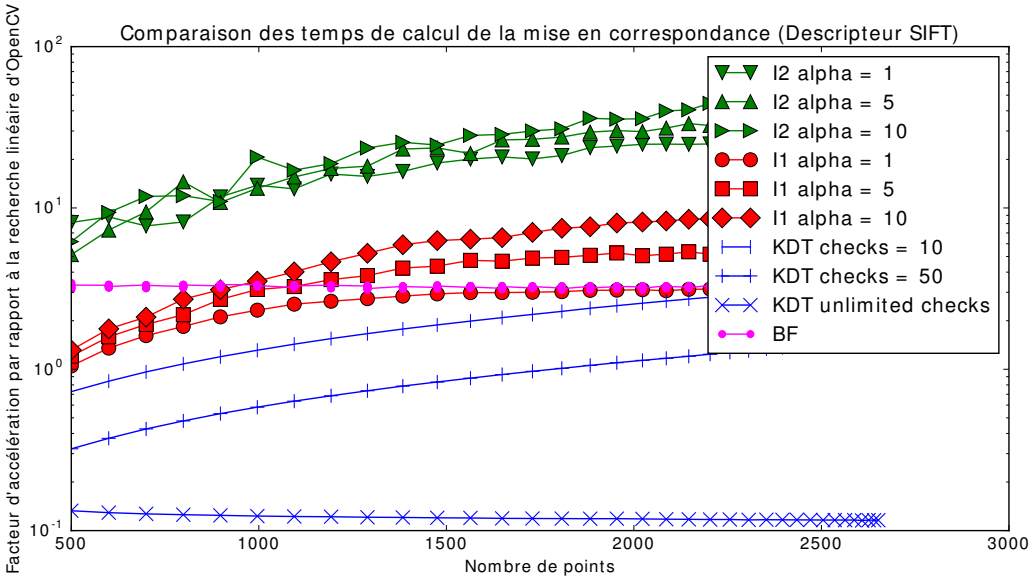

FIGURE 3.17 – Facteurs d'accélération des différents algorithmes de mise en correspondance de descripteurs SIFT par rapport à la recherche exhaustive d'OpenCV. Les optimisations proposées permettent de gagner plus d'un ordre de grandeur sur les k-d trees aléatoires de la bibliothèque FLANN. Ces chiffres ont été obtenus avec un processeur Intel 4-cœurs i7-4700HQ.

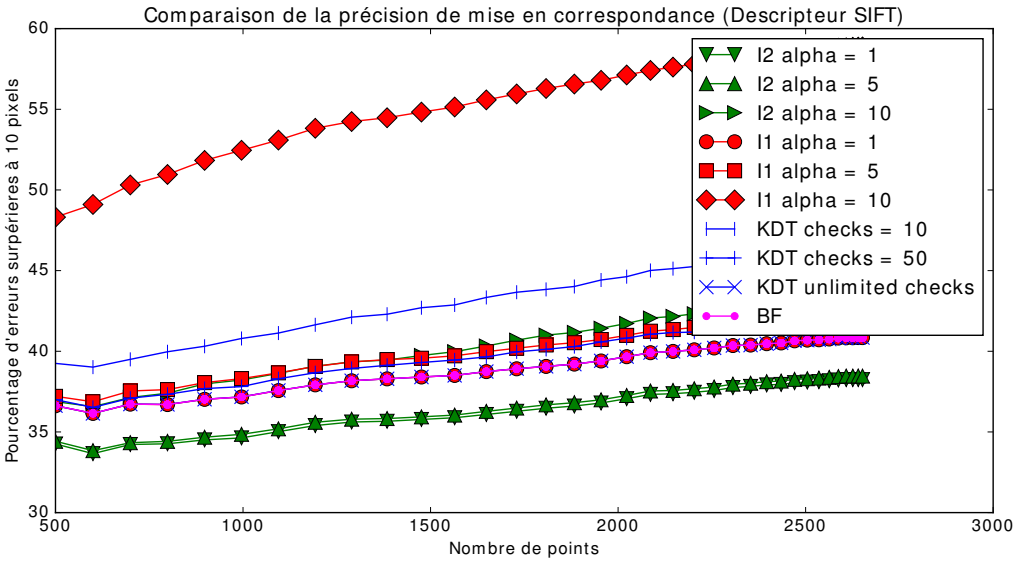

FIGURE 3.18 – Comparaison du nombre d'erreurs supérieures à 10 pixels entre notre approche et FLANN pour la mise en correspondance de descripteurs SIFT. Comme pour le descripteur PATCH, la restriction du rayon de recherche (I2), permet de diminuer le nombre d'ambiguïtés lors de la mise en correspondance.

3.3.11 Limitations

L'évaluation présentée dans la section précédente démontre l'intérêt de nos indexations pour mettre en correspondance des ensembles de cent à quelques dizaines de milliers de points. Toutefois, les KD-Tree de FLANN restent plus efficaces sur des ensembles plus grands. Les applications indexant un très grand nombre de descripteurs, par exemple un moteur de recherche d'images, ne sont pas impactées par nos travaux. D'autant plus que dans ces applications, il n'est en général pas possible de restreindre le domaine de recherche sur la position des points dans l'espace image.

3.3.12 Portage sur Android

La chaîne de détection et mise en correspondance de points d'intérêt développée dans cette étude a été portée sur Android pour y permettre le suivi de points temps réel depuis la caméra de l'appareil. Sur un processeur Snapdragon 810, l'application est capable de suivre 1100 points sur une vidéo de 640x480 pixels à 16 images par seconde.

Grâce au kit de développement natif d'Android (NDK), le code C++ de la chaîne de traitement a pu être repris sans modification et dans son intégralité, à l'exception du détecteur FAST [55] qui a dû être adapté aux instructions vectorielles NEON.

Des optimisations futures de cette application pourront reposer sur le GPU du téléphone pour optimiser davantage les opérations comme la détection de points d'intérêt.

L'application est disponible en accès libre sur le Google Play Store à l'adresse suivante : https://play.google.com/store/apps/details?id=cv.cvExperiments

3.3.13 Conclusion et Perspectives

Nous avons montré dans cette étude qu'en utilisant une série d'optimisations algorithmiques ou logicielles, il est possible d'effectuer l'extraction de points d'intérêt FAST [55] jusqu'à 5.5 fois plus rapidement que les implantations existantes, et la mise en correspondance de deux ensembles de points

en des temps jusqu'à 100 fois inférieurs à FLANN à taux d'erreur similaires.

Ces optimisations reposent sur l'exploitation autant que possible des parallélismes multicœur et SIMD, la localité des accès mémoire et la restriction de l'espace de recherche dans le domaine de l'image et dans l'espace des descripteurs.

Notons que même si le taux d'erreur a été réduit par l'algorithme de mise en correspondance proposé, cette étude a été focalisée sur la réduction du temps de calcul. Dans des travaux futurs, ce taux d'erreur pourra être diminué avec des méthodes de filtrage, l'utilisation de descripteurs plus discriminants, des détecteurs de points plus répétables et une prédiction améliorée des mouvements des points.

3.4 Conclusion du chapitre

Dans ce chapitre, nous avons présenté nos travaux sur deux types d'analyse de mouvement : l'extraction semi-dense de trajectoires de points à partir d'une vidéo et la mise en correspondance éparse de points d'intérêt.

Notre première contribution, *Video Extruder*, a permis d'accélérer l'extraction semi-dense de trajectoires d'un ordre de grandeur par rapport à la méthode classique Lucas-Kanade. Nos trois implémentations atteignent respectivement une fréquence de traitement de 165Hz sur un GPU, 150Hz sur un CPU x86 4-cœurs 3Ghz, et 10Hz sur un ARM mono-cœur 1Ghz, pour des flux de plusieurs milliers de trajectoires. Pour cela, nous avons optimisé chaque étape de la méthode, en proposant des algorithmes peu gourmands en instructions CPU et en bande passante mémoire. Nous avons aussi limité le nombre d'opérations traitant la totalité des pixels en préférant les opérations sur les trajectoires, qui sont au moins 10 fois moins nombreuses. Le fait que les étapes de détection, filtrage et fusion soient facultatives a aussi permis d'augmenter significativement la fréquence de traitement. Dans des travaux futurs, il sera possible d'améliorer la qualité et la précision de trajectoires en ajoutant par exemple une étape régularisation spatiale rapide, comme le proposent Kroeger et al. [36]. Cependant, pour limiter l'impact sur le temps de calcul, ces traitements devront être effectués à de basses résolutions.

Notre deuxième contribution a permis l'accélération de la mise en cor-

respondance de deux ensembles de points dans des espaces de descripteurs à grande dimension. Notre approche est particulièrement efficace sur des ensembles de petite taille (inférieure à 50 000 points) et/ou quand la position des points peut être grossièrement prédite. Pendant nos expériences, nous avons pu observer des facteurs d'accélération allant jusqu'à deux ordres de grandeur par rapport aux k-d trees aléatoires de FLANN.

Les accélérations présentées dans ce chapitre sont finalement une avancée vers l'embarquabilité à moindre coût de l'analyse de mouvement, et ouvre de nouvelles opportunités aux applications ayant à la fois besoin d'une information de mouvement précise et de fortes contraintes de temps de calcul.

Chapitre 4

Odométrie visuelle et flux épipolaire monoculaire

Sommaire

4.1 Introduction

Avec l'arrivée des applications de réalité virtuelle, augmentée et de la robotique, capturer la géométrie 3D de l'environnement a gagné l'intérêt des laboratoires de recherche en traitement d'images.

Une des problématiques de ces applications est l'estimation de la géométrie 3D de la scène à moindre coût. Aujourd'hui (2016), le LIDAR est la solution la plus précise mais aussi la plus chère. Ce capteur mesure la distance d'un objet projeté sur un pixel en chronométrant l'aller-retour d'une impulsion lumineuse émise dans sa direction.

Moins coûteux que le LIDAR, les capteurs reposant sur une diffusion de lumière structurée sont aussi répandus : la Kinect V1 de Microsoft [67] et d'autres appareils portables comme le Tango [26] de Google. Ils fonctionnent bien en intérieur et sur des courtes distances, mais leur précision chute en extérieur avec la lumière du soleil et sur des distances plus grandes.

Prometteuses, les caméras plénoptiques [45] et les pupilles codées [37] permettent l'estimation de la profondeur en interprétant le flou induit par une ouverture non ponctuelle. Ces capteurs ont l'avantage de fournir une information de profondeur tout en étant passifs. Leur consommation d'énergie est plus faible que celle les capteurs actifs comme le LIDAR ou les capteurs à lumière structurée mais dans leur état actuel, ils sont limités dans leur résolution et leur prix restent élevés.

Malgré la diffusion croissante en tant que produit sur étagères de ces technologies, le capteur le moins cher reste la caméra RGB. Toutefois, ce capteur ne fournit pas d'information de profondeur et il est nécessaire d'utiliser des algorithmes capables de l'estimer. Ces derniers reposent sur une caméra en mouvement (stéréo-vision monoculaire) ou plusieurs caméras (stéréo-vision multi-vues) et fonctionnent en deux étapes : d'abord, si elles ne sont pas connues au préalable, l'estimation des poses relatives aux deux points de vues. Ensuite, le calcul d'un flux optique et finalement l'estimation de la profondeur de chaque pixel. La consommation électrique de ces systèmes est plus faible que celle du LIDAR et des caméras à lumière structurée mais leur précision dépend hautement de la structure de la scène.

Dans ces travaux, nous nous sommes focalisés sur la stéréo-vision mono-

culaire. Ce choix a été guidé par le fait que l'utilisation d'une seule caméra diminue le coût de fabrication et la consommation électrique du système. De plus, elle peut offrir une meilleure triangulation de la scène car elle permet des écarts de points de vue de taille arbitraire, contrairement à la stéréo-vision binoculaire où les deux caméras sont en général fixées et alignées sur un appareil de taille limitée.

Malgré leur faible coût, les systèmes de stéréo-vision monoculaire sont en général plus lents et moins précis que les versions binoculaires car ils doivent estimer le déplacement relatif entre chaque paire d'images. Dans cette étude, nous nous sommes donc focalisés sur l'accélération et l'amélioration de la précision de la stéréo-vision monoculaire.

Durant ces dernières années, un grand nombre de travaux ont été menés pour améliorer la qualité et le temps de calcul d'une carte de profondeur à partir de deux points de vue, que ce soit en stéréo monoculaire ou binoculaire.

Dans les deux cas, les approches reposent sur les lignes épipolaires (voir section 4.3) pour restreindre le domaine de recherche du flux optique. Par rapport à un algorithme de flux optique classique, cela permet de diminuer le temps de calcul et le nombre d'erreurs.

Pour améliorer l'estimation dans les zones peu texturées, il est aussi courant de considérer la scène comme un ensemble de régions planaires [6, 64, 65, 66]. Le domaine de l'image est alors segmenté en un ensemble de régions (ou super pixels) contenant des pixels d'apparence et de mouvement similaires. Ensuite, les équations de plan de chaque région sont raffinées par la minimisation d'une fonction de coût. Des modèles de champs de Markov permettent finalement la réduction des incohérences aux frontières des objets. Dans [65] chaque frontière est catégorisée en occultation, liaison entre deux plans, ou prolongation de plan.

Bien que ces approches permettent d'estimer un flux optique précis contenant peu d'erreurs, leur complexité implique des temps de calcul élevés et limite leur intégration dans des applications embarquées.

Dans ce chapitre, nous proposons une méthode capable d'estimer un flux optique avec un taux d'erreur inférieur à l'état de l'art tout en gagnant plusieurs ordres de grandeur sur le temps de calcul.

Contrairement aux approches de l'état de l'art, nous avons conçu cette

chaîne de traitement en prenant en compte tous les facteurs impactant le temps de calcul : la complexité algorithmique, les schémas d'accès mémoire, la taille des données, leur type, et le parallélisme.

Notre approche s'articule en quatre étapes :

— **estimation de la matrice fondamentale** : dans un premier temps, nous estimons la rotation et la translation entre les deux points de vues pour en déduire la matrice fondamentale (voir section 4.3). Ici, la répartition et le nombre des points d'intérêt mis en correspondance ont été optimisés pour minimiser le temps de calcul tout en maximisant la précision de la matrice fondamentale.

— **estimation du flux optique** : nous utilisons ensuite les lignes épipolaires pour estimer un champ de mouvement semi-dense (section 4.4). Pour accélérer davantage cette étape, nous avons exploité la cohérence locale du flux optique en propageant les vecteurs de flux optique au cours de l'estimation.

— **filtrage des erreurs** : avec l'algorithme Lucas Kanade, nous vérifions que chaque vecteur de flux optique est aussi valide dans des directions autres que celle de la ligne épipolaire. Nous filtrons ensuite les incohérences spatiales (section 4.4.2).

— **propagation** : les zones encore non estimées sont finalement calculées en interpolant linéairement les vecteurs de flux optique d'un voisinage proche, s'ils existent (section 4.4.3).

Nous évaluons notre approche avec la base de données *KITTI optical flow* [3], un ensemble de 194 images avec flux optique de référence. Ses auteurs classent plus de 70 algorithmes selon deux critères :

1. le pourcentage d'erreurs sur tous les pixels de l'image, en interpolant linéairement les vecteurs des approches semi-dense.

2. le pourcentage d'erreurs sur les pixels estimés seulement.

Pour le deuxième critère, notre algorithme est classé premier tout en étant 1000 fois plus rapide que le deuxième. Les résultats sont présentés en section 4.5. L'image 4.1 montre une carte de profondeur obtenue à partir de l'odométrie visuelle et du flux optique estimés par notre approche.

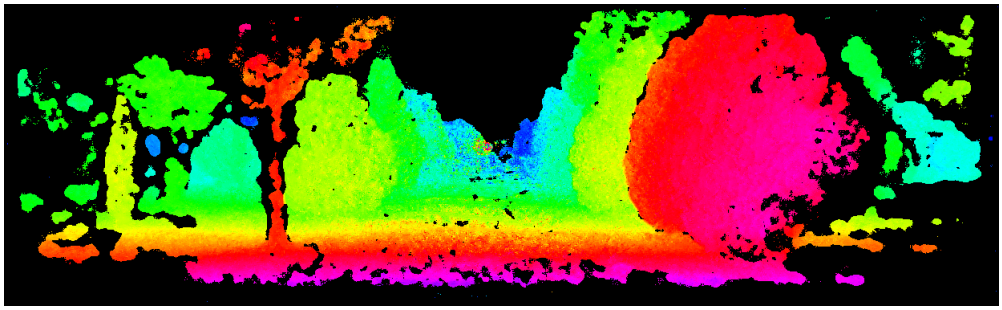

FIGURE 4.1 – Carte de profondeur obtenue à partir du flux optique et de l'odométrie visuelle.

4.2 Notations

Dans la suite nous utilisons les notations suivantes :

— I_1 et I_2 sont les deux images pour lesquelles nous calculons le flux optique.

— Un point 3D X est projeté sur les images I_1 et I_2 aux coordonnées respectives x_1 et x_2.

— $\tilde{x}$ représente les coordonnées homogènes 3D des coordonnées 2D x.

— F est la matrice 3×3 fondamentale estimée et établie la relation entre $\tilde{x}_1$ et $\tilde{x}_2$ (voir section 4.3).

4.3 Estimation de la matrice fondamentale

Les algorithmes de flux optique sont confrontés à deux enjeux majeurs : comment estimer avec précision le mouvement de zones peu texturées, ou avec des schémas répétitifs ? et comment le faire en un temps de calcul restreint étant donné le grand nombre de déplacements possibles ?

Toutefois, dans le cas d'une caméra mobile dans une scène statique, le mouvement est soumis à la contrainte épipolaire (voir figure 4.2) que nous pouvons exploiter pour restreindre l'espace de recherche du flux optique : si un point X est aux mêmes coordonnées 3D lors des acquisitions des deux images alors les cinq points suivants appartiennent à un unique plan épipolaire :

— les deux centres optiques O_1 et O_2

— les deux projections x_1 et x_2

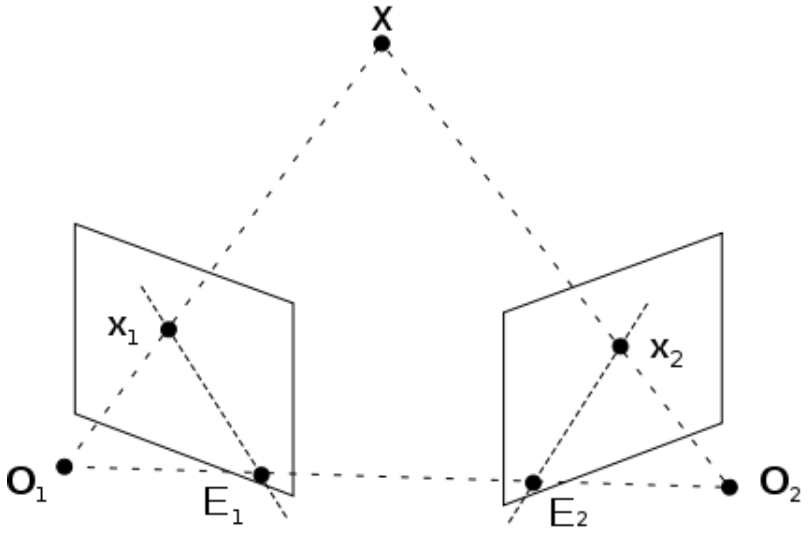

FIGURE 4.2 – Exploitation de la contrainte épipolaire pour améliorer l'appariement des points. Si un point X est aux mêmes coordonnées 3D lors de l'acquisition des deux images, les deux centres optiques O_1 et O_1, les deux projections x_1 et x_1 et X lui-même appartiennent à un unique plan 3D. Source: Wikipedia.

— X lui-même

Lors d'une l'estimation d'un vecteur de flux optique, les coordonnées de la projection x_2 sont recherchées sachant x_1. Et étant donné la contrainte épipolaire, x_2 appartient à la projection du plan épipolaire sur l'image I_2, appelée droite épipolaire. Il est alors possible de restreindre le domaine de recherche de x_2 à une droite. Cela permet de diminuer le temps de calcul du flux optique et d'en diminuer le nombre d'erreurs dues à des ambiguïtés de mise en correspondance.

Il est donc nécessaire de connaître trois points du plan épipolaire. x_1 et O_1 sont connus et O_2 peut être obtenu en estimant la rotation et translation séparant les deux points de vue.

En vision stéréoscopique, la géométrie intrinsèque de la caméra et la transformation liant les deux points de vue est modélisée par la matrice fondamentale F. Cette matrice permet d'exprimer simplement la contrainte épipolaire : si $\tilde{x}_1$ et $\tilde{x}_2$ sont les coordonnées homogènes des projections d'un point X de

la scène sur les deux images, nous avons :

$$\tilde{x}_2^T F \tilde{x}_1 = 0 \tag{4.1}$$

Cette équation est vérifiée si X, ses projections x_1 et x_2 et les deux centres optiques O_1 et O_2 appartiennent au même plan. Cette condition est réalisée par tous les points statiques de la scène.

La matrice F est utile à l'estimation du flux optique car, comme expliqué précédemment, elle nous permet de restreindre le domaine de recherche de x_2 à la ligne épipolaire $l_2(x_1)$, définie comme suit :

$$l_2(x_1) = \left\{ x_2 \in \mathbb{R}^2, \tilde{x}_2^T F \tilde{x}_1 = 0 \right\} \tag{4.2}$$

Pour estimer F, il est nécessaire de calculer un ensemble de vecteurs de flux optique entre les pixels de I_1 et I_2. En théorie, sept vecteurs peuvent suffire à estimer F mais en pratique, un plus grand ensemble est estimé pour pouvoir être robuste aux erreurs.

Plusieurs solutions existent pour l'estimation de la matrice fondamentale à partir d'un ensemble de mises en correspondance [38, 53]. Toutefois, la précision et le temps de calcul de ces algorithmes dépendent fortement du nombre d'erreurs et de la distribution spatiale des points mis en correspondance fournis en entrée. Pour augmenter la robustesse aux erreurs, ces algorithmes sont souvent couplés à RANSAC [18]. Cependant, cela augmente le temps de calcul et limite l'intégration dans des applications embarquées.

Nous proposons dans cette section l'extraction d'un flux optique épars conçu pour :

- minimiser le nombre d'itérations nécessaires à RANSAC [18] pour converger, en filtrant en amont les vecteurs de flux optique erronés.
- être robuste aux grands déplacements.
- assurer une bonne distribution spatiale des bipoints supports du flux optique pour améliorer la précision de F.
- obtenir le meilleur compromis entre temps de calcul et précision.

Nous donnons d'abord une vue d'ensemble de l'approche, puis détaillons chaque partie de l'algorithme pour finalement en démontrer son efficacité sur le jeu de données *KITTI optical flow*.

4.3.1 Vue d'ensemble

Le pipeline d'estimation de la matrice fondamentale proposé commence par la mise en correspondance de deux ensembles de points FAST, extraits des deux images. Le détecteur FAST a été modifié pour assurer une bonne répartition des points dans l'image. L'approche décrite en section 3.3 nous a permis de réduire significativement le temps de calcul de cette première étape. Nous raffinons ensuite les vecteurs de flux optique obtenus avec une descente de gradient pyramidale de type Lucas-Kanade [40]. Nous les filtrons par la suite sur un critère de cohérence spatiale. Finalement, ces vecteurs sont transmis à un algorithme d'estimation de matrice fondamentale de l'état de l'art.

4.3.2 Protocole d'évaluation

Pour évaluer l'erreur E_F d'une matrice fondamentale sur une paire d'images, nous estimons la distance maximale entre les lignes épipolaires (voir équation 4.2) et le flux optique de référence GT fourni par le jeu de données *KITTI optical flow* :

$$E_F = \max_{(x_1,x_2)\in GT} \frac{|\tilde{x}_2^T l|}{\sqrt{l_a^2 + l_b^2}}$$

Avec :

$$l = F\tilde{x}_1 = [l_a, l_b, l_c]$$

Dans la suite, nous évaluons chaque proposition par l'analyse de son impact sur E_F et le temps de calcul.

4.3.3 Détection et mise en correspondance de points d'intérêt

Pour calculer un ensemble de vecteurs de flux optique, beaucoup de méthodes reposent sur le fait que les déplacements sont assez petits pour être estimés par l'optimisation locale d'une mesure de similarité, par exemple une descente de gradient [40]. La qualité de ces méthodes dépend hautement de

la grandeur des déplacements, même si elle sont intégrées dans des raisonnements multi-échelles [7]. C'est pourquoi dans cette étude, nous avons conçu un algorithme de mise en correspondance ne faisant aucun *a priori* sur la taille des déplacements.

Pour chaque point FAST [55] x_1 détecté dans I_1, nous recherchons le point FAST x_2 détecté dans I_2 ayant la plus petite distance SAD sur un patch de taille 11×11. La méthode proposée en section 3.3 a permis d'accélérer significativement cette mise en correspondance. Grâce à la répétabilité du détecteur FAST, une grande partie des points détectés dans I_1 seront aussi détectés dans I_2.

Bien que cette méthode permette l'estimation d'un flux optique épars, sa contribution à la précision de l'estimation d'une matrice fondamentale est limitée. En effet, la distribution des points FAST dépend hautement du contenu de l'image, et le fait que ce détecteur puisse laisser des zones moins contrastées sans points d'intérêt impacte la précision de l'estimation de F.

Pour améliorer la distribution des points d'intérêt, nous utilisons la stratégie de sélection blockwise présentée en section 3.2.2. Cette stratégie sélectionne le point FAST ayant le meilleur score dans chaque cellule d'une grille de taille $Nc \times Nc$ recouvrant le domaine de l'image. Dans la suite, le seuil du détecteur sera nommé $keypointTh$.

Bien que cette stratégie altère la répétabilité du détecteur de point, nous avons pu observer son effet positif sur la précision de la matrice fondamentale F.

Pour évaluer la pertinence du champ de vecteurs de flux optique pour l'estimation de la matrice fondamentale, nous avons testé notre approche sur un ensemble de couples de paramètres Nc et $keypointTh$. Les résultats ont confirmé nos hypothèses :

— la figure 4.3 montre que la distribution engendrée par une plus petite valeur de Nc réduit le temps de calcul de l'estimation d'un ordre de magnitude sans augmenter l'erreur de F.
— la figure 4.4 met en évidence le fait que le nombre de points d'intérêt impacte directement le temps de calcul.

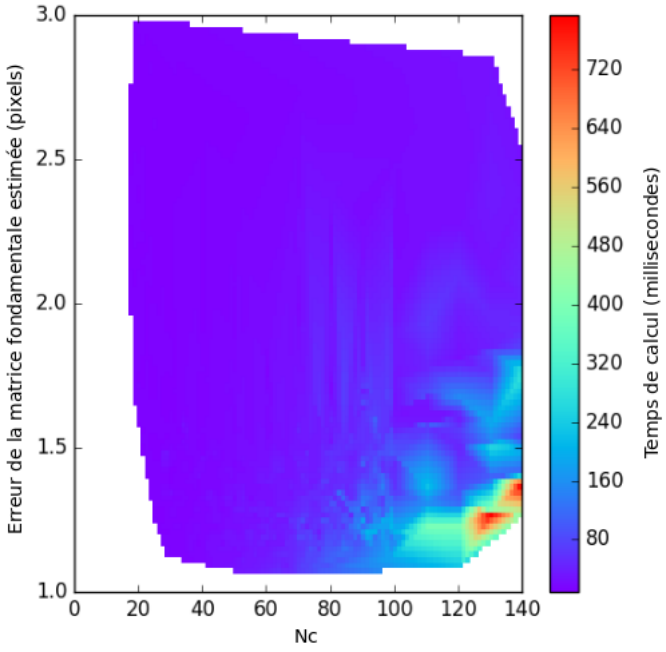

FIGURE 4.3 – Impact de la résolution de la grille Nc sur le temps de calcul et l'erreur associée à l'estimation de F. Dans la région en bas à droite correspondant à une faible erreur et une grande valeur de Nc, le temps de calcul est très élevé. Cela montre que sans contraindre la répartition des points d'intérêt, il est nécessaire d'extraire plus de points pour estimer une matrice fondamentale fiable, et cela augmente le temps de calcul d'un ordre de grandeur.

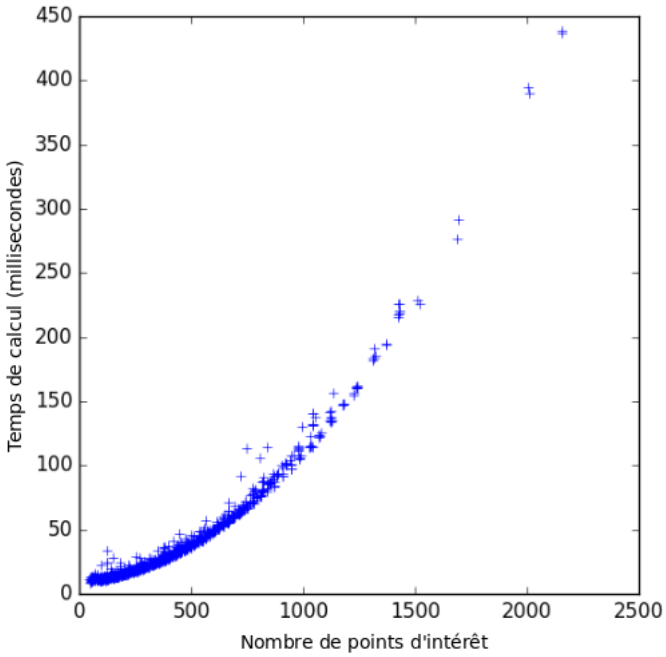

FIGURE 4.4 – Impact du nombre de points d'intérêt sur le temps de calcul de l'estimation de F.

4.3.4 Filtrage des vecteurs de flux optique incohérents

Comme la stratégie de sélection *blockwise* altère la répétabilité du détecteur de points, les points d'intérêt de la première image ne sont pas toujours détectés dans la deuxième image et peuvent donc être mis en correspondance avec d'autre points similaires. Ce phénomène génère des vecteurs de flux optique erronés.

Ces erreurs sont en général filtrées par une estimation robuste de type RANSAC [43]. Toutefois, cette méthode est coûteuse en temps de calcul car chacune de ses itérations réalise une estimation complète de la matrice fondamentale. Pour accélérer la convergence de RANSAC [18], nous proposons de filtrer en amont les vecteurs de flux optique divergeant trop fortement du mouvement local moyen.

Pour cela, nous supprimons tous les vecteurs de flux optique ne respectant pas le critère suivant :

$$||v - med|| < medTh$$

Avec med le vecteur de flux optique correspondant au déplacement médian local le long des deux dimensions de l'image. Le déplacement médian est estimé pour chaque cellule d'une grille de $Rf \times Rf$ recouvrant le domaine de l'image.

La figure 4.5 montre que ce filtre permet de diminuer le temps de calcul de 30% sans augmenter le nombre d'erreurs.

4.3.5 Raffinement de la mise en correspondance avec Lucas Kanade

Pour augmenter la précision des vecteurs de flux optique obtenue par la mise en correspondance de points d'intérêt, nous effectuons une descente de gradient pyramidale de type Lucas Kanade [7]. Même si l'amplitude de cette correction est faible, nous avons obtenu de meilleurs résultats avec deux échelles qu'avec une seule.

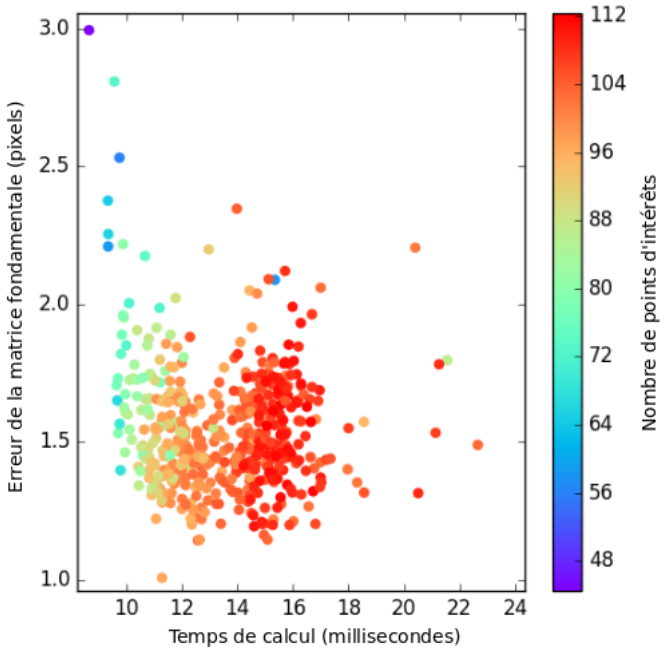

FIGURE 4.5 – Impact du filtrage des vecteurs de flux optique erronés sur le temps de calcul et l'erreur de l'estimation de F.

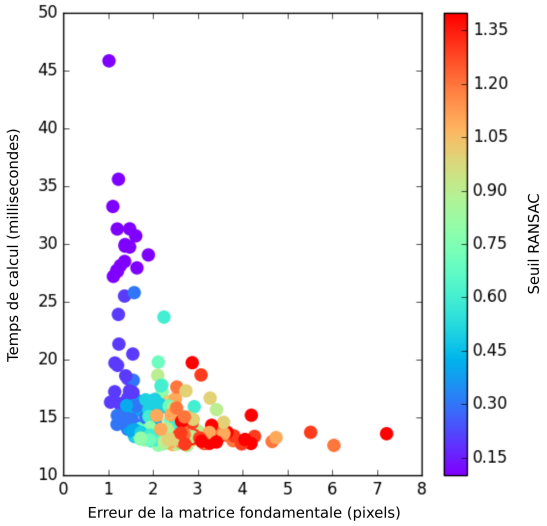

FIGURE 4.6 – Impact du seuil RANSAC sur l'erreur E_F et le temps de calcul.

4.3.6 Estimation de la matrice fondamentale

À partir de l'ensemble de vecteurs de flux optique raffinés, nous estimons la matrice fondamentale avec les approches de l'état de l'art combinées avec RANSAC [18].

La figure 4.6 montre le temps de calcul et l'erreur E_F obtenus avec l'algorithme des 8 points [38] implanté dans OpenCV [8], pour différentes valeurs de seuil RANSAC. Ces résultats montrent que même si nous avons accéléré l'estimation de la matrice fondamentale, le temps de calcul et l'erreur E_F dépendent hautement du seuil de RANSAC.

Il existe des méthodes pour estimer *a priori* une bonne valeur de seuil RANSAC [43], mais nous ne les avons pas étudiées ici.

Dans l'illustration 4.7 nous pouvons observer l'ensemble épars des vecteurs de flux optique calculé sur une image de test, ainsi que quatre lignes épipolaires.

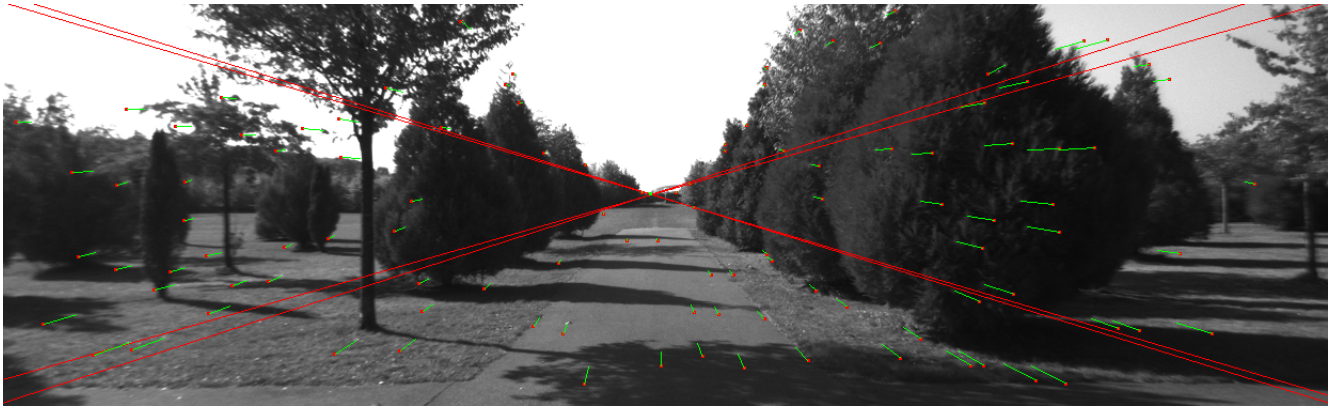

FIGURE 4.7 – Ensemble épars des vecteurs de flux optique estimés sur une image de la base KITTI. Deux lignes épipolaires ont aussi été tracées pour mettre en évidence le foyer d'expansion. Cette illustration témoigne de la bonne répartition des vecteurs, et de leur faible taux d'erreurs.

4.4 Estimation du flux optique

4.4.1 Recherche guidée par les lignes épipolaires

Une fois la matrice fondamentale estimée, nous l'utilisons pour guider l'estimation du flux optique entre les images I_1 et I_2. Comme expliqué en section 4.3, cela permet de diminuer le temps de calcul et les ambiguïtés de mise en correspondance.

Le calcul du flux optique guidé par la matrice fondamentale F peut être formulé comme suit : étant donné un point 3D X de la scène projeté sur l'image I_1 aux coordonnées x_1 connues, nous cherchons les coordonnées x_2 de la projection de X dans l'image I_2. Cette recherche peut être restreinte à la ligne épipolaire l_2 (voir équation 4.2). Dans la suite, nous modélisons cette ligne par $l_2(x_1, d) = e_2 + dv, \forall d \in \mathbb{R}$ avec d la disparité (scalaire), v le vecteur 2D unitaire support de l_2 et e_2 les coordonnées de l'épipole de I_2, obtenues en résolvant l'équation $e_2^T F = \vec{0}$

Pour trouver x_2 sur la ligne épipolaire l_2, nous recherchons la disparité d_{x_1} minimisant la distance SSD suivante :

$$d_{x_1} = \arg\min_{d} \sum_{u \in [-3,3]^2} \Big[I_1(x_1 + +u) - I_2(l_2(x_1, d) + u) \Big]^2 \tag{4.3}$$

Une approche naïve serait de calculer les distances entre x_1 et tous les pixels de la ligne l_2. Cependant, en termes de temps de calcul, cette opération est sous-optimale car elle ne tire pas parti du fait que les pixels voisins ont en général des disparités similaires. Cela implique des redondances de calcul sur les surfaces planes, très courantes dans les environnements urbains.

C'est pourquoi dans cette étude, nous avons conçu un algorithme tirant parti de la cohérence spatiale de la disparité pour réduire le temps de calcul d'un flux optique dense. Il est composé des deux étapes suivantes :

1 - Initialisation. Pour initialiser le champ de disparités, nous calculons les disparités des vecteurs de flux optique calculés pour l'estimation de la matrice fondamentale (voir section 4.3). Dans la suite, ces disparités sont appelées graines.

2 - Propagation. Une fois les graines placées sur le champ de disparité initial et dans la queue de traitement Q, nous appliquons l'algorithme glouton suivant : pour chaque point x_1 à traiter, nous recherchons via une descente de gradient la disparité minimisant localement la distance SSD (équation 4.3) puis ajoutons chacun de ses voisins $\{y_1^i\}_{i=1..8}$ à Q si toutes les conditions suivantes sont validées :

- La norme du gradient local au point y_1^i est supérieure au seuil $minGradient$. Cela évite la propagation sur des zones contenant trop peu d'information pour l'estimation de flux optique.
- La disparité au point y_1^i n'a pas encore été calculée ou est trop différente de la disparité à propager (valeur absolue supérieure à $dispPropTh$).

Notons que cet algorithme permet la propagation de plusieurs graines sur le même pixel. Cela aide à améliorer le flux aux bordures des objets.

Accélération sur processeurs multicœurs. L'avantage de cet algorithme glouton est que la propagation de chaque graine peut être faite indépendamment des autres et ainsi être exécutée dans un thread séparé. Il est toutefois

possible que deux threads convergent sur le même pixel au même moment, mais cela est peu probable et n'a pas nui aux résultats de nos expériences.

4.4.2 Détection et filtrage des erreurs

L'algorithme de recherche du flux optique présenté dans la section précédente se restreint aux lignes épipolaires. Pour vérifier la validité de chaque vecteur calculé, nous proposons de vérifier qu'ils minimisent localement la distance SSD (équation 4.3) dans toutes les directions.

Pour cela, nous effectuons une itération de l'algorithme Lucas-Kanade sur chaque vecteur estimé précédemment. En effet, cette descente de gradient divergera de la ligne épipolaire si le vecteur ne minimise pas localement la distance SSD dans toutes les directions.

Les vecteurs sont ensuite marqués comme erronés si au moins une des conditions suivantes est satisfaite :

— si la distance entre la destination du vecteur de flux optique et la ligne épipolaire est supérieure au seuil eth.

— si la distance entre un vecteur v et ses voisins dans un voisinage de taille $Nv \times Nv$ est inférieure à dth pour moins de $dp\%$ de ses voisins, il est considéré comme incohérent. Les meilleurs résultats ont été obtenus avec un voisinage de taille $Nv = 15$ et $dp = 60\%$.

Le graphique 4.8 montre l'évolution des taux de vrais positifs (vecteurs avec une distance supérieure à 3 pixels de la vérité terrain KITTI détecté comme erreur) et de faux positifs (vecteurs avec une erreur inférieure à 3 pixels détectés comme erreur) en fonction des seuils de détection eth et dth. Pour restreindre l'espace des paramètres de l'expérience, nous avons fait varier dth et fixé $eth = dth/4.3$.

Finalement, en plus d'aider au filtrage des erreurs, cette itération de Lucas Kanade a l'avantage d'affiner la précision du flux optique.

4.4.3 Propagation

Le filtrage des erreurs laisse des trous dans le champ de flux optique calculé et impacte la couverture du flux optique résultat. Pour limiter ce phénomène, nous remplaçons les pixels dont le vecteur de flux optique n'a pas été estimé ou

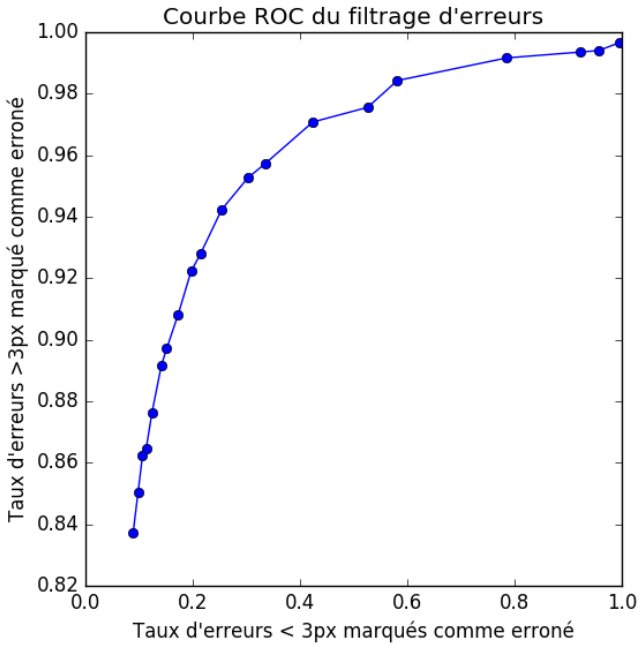

FIGURE 4.8 – Courbe ROC du filtrage d'erreurs. Nous considérons comme erreurs les vecteurs de flux optique distants de plus de 3pixels de la vérité terrain KITTI.

marqué comme erreur par la moyenne des vecteurs présents dans un voisinage de taille 7×7. Si ce voisinage ne contient aucun vecteur, le pixel est laissé découvert. Lors de nos expériences sur la base KITTI, cette méthode a permis de couvrir 15% des pixels de l'image en plus, avec une augmentation du nombre d'erreurs négligeable.

4.5 Évaluation et comparaison avec l'état de l'art

Pour évaluer notre approche nous avons exploité la base de tests *KITTI Optical Flow*. Elle contient un ensemble de paires d'images acquises par une caméra embarquée dans un véhicule en circulation et leur flux optique de référence déduit des données 3D du capteur LIDAR. La base contient plusieurs types de déplacements de caméra, comme des translations parallèles à l'axe optique ou des rotations mettant en jeu de grands déplacements.

Le tableau 4.1 présente le classement du site *KITTI* des algorithmes de flux optique, en fonction de leur taux d'erreur sur les pixels estimés. Notre algorithme est classé premier du classement *KITTI optical flow 2012* pour l'estimation non dense, avec un taux d'erreur et un temps de calcul respectivement 1.5× et 1000× inférieurs à ceux du deuxième du classement [63]. Ce dernier couvre toutefois 100% des pixels des l'image, pour 50% avec notre approche. La meilleure méthode [58] ayant un temps de calcul similaire à la nôtre exploite le GPU, a un taux d'erreur 2x plus élevé et ne couvre que 15% des pixels.

Finalement, notre approche est à la fois une des plus rapides du classement et la seule à offrir des taux d'erreurs aussi faibles.

Les images 4.9 présentent le flux optique calculé par notre approche, et l'image 4.10 en montre les erreurs.

4.6 Conclusion et perspectives

La méthode que nous avons développée a permis d'accélérer le calcul du flux optique tout en diminuant le taux d'erreurs. En évitant d'estimer le flux

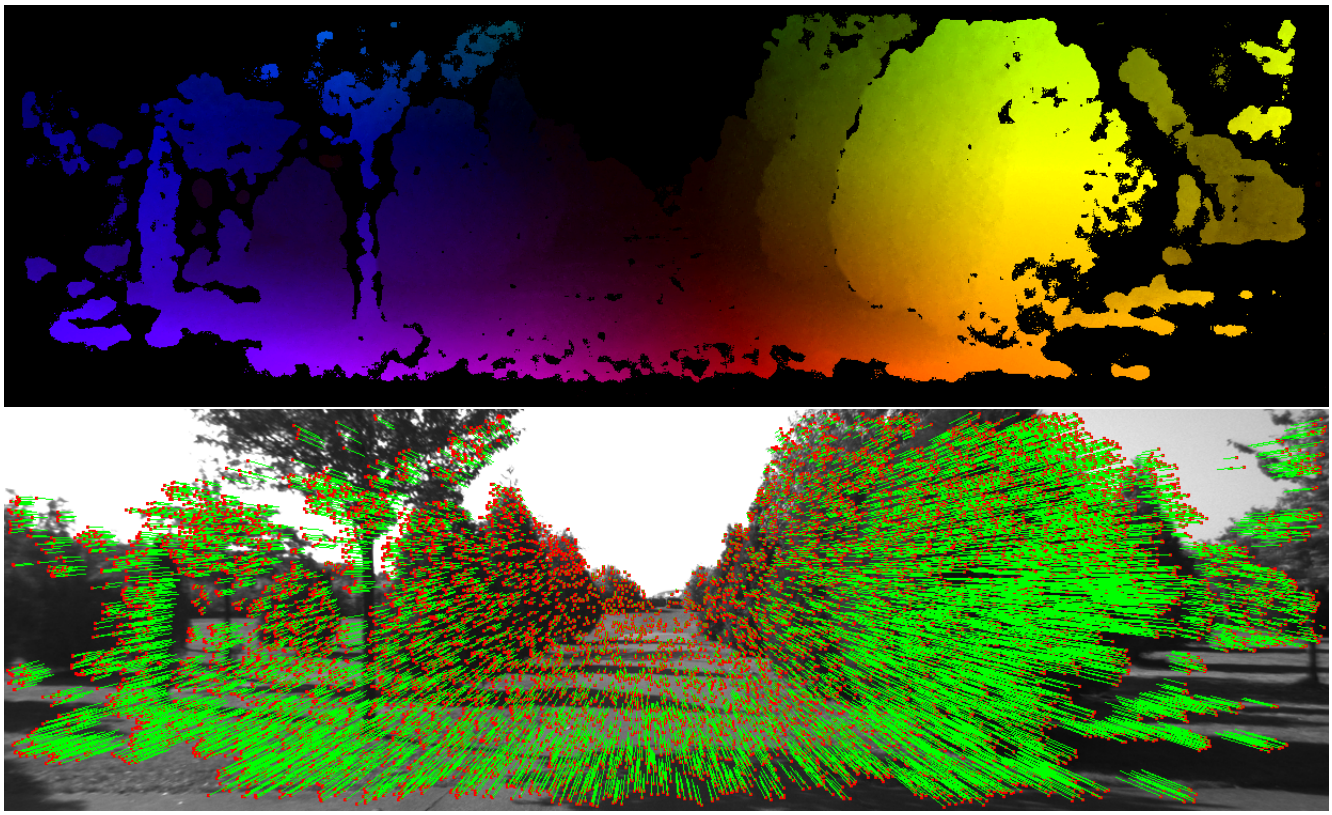

FIGURE 4.9 – Flux optique estimé sur une paire d'images de la base *KITTI* et présenté sous deux formes. En haut, les coordonnées polaires des vecteurs de mouvement encodées avec un code de couleur $<\rho,\theta> \equiv$ <intensité, teinte>, et en bas, un échantillonnage partiel des vecteurs mouvement.

TABLE 4.1 – Classement KITTI des algorithmes de flux optique, en fonction de leur taux d'erreur supérieur à 3 pixels sur les pixels estimés. Notre approche (notée FSDEF) calcule un flux optique avec moins d'erreurs en un temps de calcul moindre. La seule méthode ayant un temps de calcul inférieur, BERLOF, exploite le GPU et ne couvre que 15% des pixels. Pour chacune des approches plusieurs statistiques ont été comparées. *Out-Noc* et *Out-All* sont respectivement les taux d'erreurs moyens par image sur les pixels non occultés et tous les pixels. *Avg-Noc* et *Avg-All* sont respectivement l'erreur moyenne par image moyennée sur toutes les images de la base pour les pixels non occultés et pour tous les pixels. *Density* représente le taux de couverture moyen de l'algorithme et *Runtime* le temps de calcul moyen pour une paire d'images.

	Method	**Out-Noc**	**Out-All**	**Avg-Noc**	**Avg-All**	**Density**	**Runtime**
1	**FSDEF**	**1.59 %**	**1.77 %**	0.8 px	**0.9 px**	50.57 %	0.26 s
2	PRSM	2.46 %	4.23 %	**0.7 px**	1.0 px	100.00 %	300 s
3	GME-IM-RLOF	2.48 %	2.64 %	0.8 px	1.0 px	11.84 %	3.7 s
4	VC-SF	2.72 %	4.84 %	0.8 px	1.3 px	100.00 %	300 s
5	SPS-StFl	2.82 %	5.61 %	0.8 px	1.3 px	100.00 %	35 s
6	RLOF	3.14 %	3.39 %	1.0 px	1.2 px	14.76 %	0.488 s
7	BERLOF	3.31 %	3.60 %	1.0 px	1.2 px	15.26 %	**0.231 s**
8	SPS-Fl	3.38 %	10.06 %	0.9 px	2.9 px	100.00 %	11 s
9	OSF	3.47 %	6.34 %	1.0 px	1.5 px	100.00 %	50 min
10	PR-Sf+E	3.57 %	7.07 %	0.9 px	1.6 px	100.00 %	200 s
11	PCBP-Flow	3.64 %	8.28 %	0.9 px	2.2 px	100.00 %	3 min
12	PR-Sceneflow	3.76 %	7.39 %	1.2 px	2.8 px	100.00 %	150 sec
13	SDF	3.80 %	7.69 %	1.0 px	2.3 px	100.00 %	TBA s
14	MotionSLIC	3.91 %	10.56 %	0.9 px	2.7 px	100.00 %	11 s
15	PatchBatch_s	4.81 %	13.64 %	1.1 px	3.0 px	100.00 %	60 s

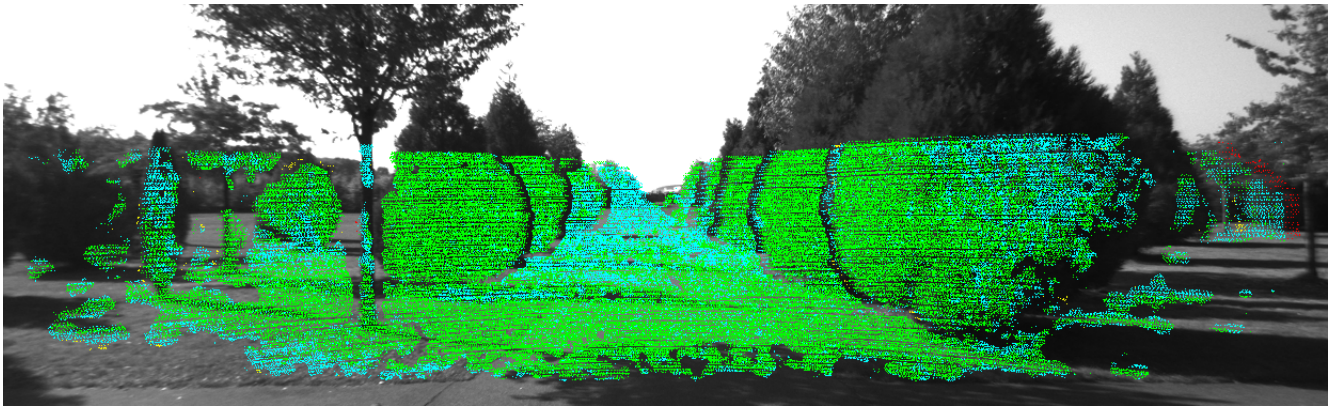

FIGURE 4.10 – Précision du flux optique calculé sur une paire d'images de la base *KITTI*. Les erreurs inférieures à 1 pixel sont affichées en vert, entre 1 et 3 pixels en bleu, entre 3 et 10 pixels en jaune et supérieures à 10 pixels en rouge.

optique dans les zones peu texturées ou ambiguës, nous avons pu réduire la complexité et le temps de calcul de notre approche. Ainsi, sur 50% du domaine de l'image, nous avons pu estimer un flux optique avec un taux d'erreur 1.5x inférieur à la meilleure solution [63] référencée par le classement *KITTI*, tout en étant 1000x plus rapide.

Ces résultats montrent qu'il est possible de construire une estimation du flux optique non dense combinant un faible coût de calcul avec un faible taux d'erreur.

Dans des travaux futurs, nous pourrons diminuer davantage le temps de calcul et le taux d'erreur en reprenant l'hypothèse que la scène est composée de facettes planes, ou importer une connaissance de la structure de la scène *a priori* pour guider l'estimation du flux optique. Cependant, bien que ces techniques permettent d'augmenter la couverture et/ou la qualité, combiner une accélération avec une amélioration de l'estimation du flux optique reste un enjeu de taille.

Chapitre 5

Applications développées

Sommaire

5.1 Introduction

Au cours de cette thèse, plusieurs occasions se sont présentées pour mettre en pratique nos recherches en analyse de mouvement. Nous présentons dans ce chapitre chacune d'elles en expliquant comment nous avons tiré parti des algorithmes présentés précédemment.

Nous détaillons d'abord en section 5.2 nos travaux sur la détection d'événements embarqués menés pendant le projet européen SPY. Ensuite, en section 5.3, nous présentons une implémentation de la stabilisation vidéo accélérée grâce à *Vidéo Extruder*. En section 5.4, une segmentation d'objets mobiles développée pour un partenaire privé est présentée et finalement, en section 5.5, nous présentons nos travaux en reconnaissance d'action.

5.2 Détection d'événements sur processeur embarqué

Un des premiers partenariats public/privé auquel nous avons participé est le projet européen SPY [15] (voir figure 5.1). Ce projet a rassemblé 14 partenaires industriels et académiques autrichiens, finlandais, français et turcs.

Notre rôle au sein du consortium a été la conception d'algorithmes de détection d'événements s'exécutant à 10 images par seconde sur les deux plateformes de calcul embarquées, fabriquées respectivement par l'entreprise française Eolane [2] et l'entreprise turque Aselsan [1]. Leurs processeurs étant conçus pour des applications embarquées dans des véhicules de transport public et militaire, leurs puissances de calcul sont limitées : ils contiennent respectivement un cœur ARM IMX 53 et un coeur ARM Cortex A8, les deux étant cadencés à 1GHz.

La première partie du projet a été dédiée à la finalisation de *Video Extruder* et à son optimisation pour atteindre la fréquence de traitement désirée sur les processeurs ciblés. Pour cela, il a été nécessaire de traiter les images à une petite résolution. L'image 5.2 montre les flux de trajectoires extraits sur des vidéos basse résolution. Dans la deuxième partie du projet, nous avons utilisé des mesures statistiques sur l'ensemble de trajectoires pour y détecter

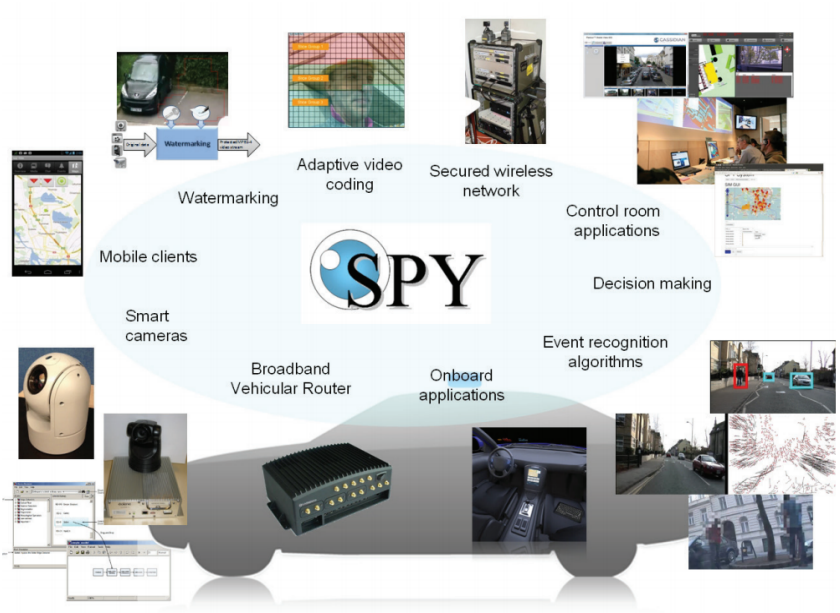

FIGURE 5.1 – Vue d'ensemble du projet européen SPY.

des chutes et des mouvements de foule.

Nos travaux ont finalement abouti à l'intégration de *Video Extruder* sur les deux plateformes de calcul. La détection d'événements a aussi pu être embarquée sur la plateforme Eolane, mais pas sur celle d'Aselsan à cause de ses capacités de calcul moindres.

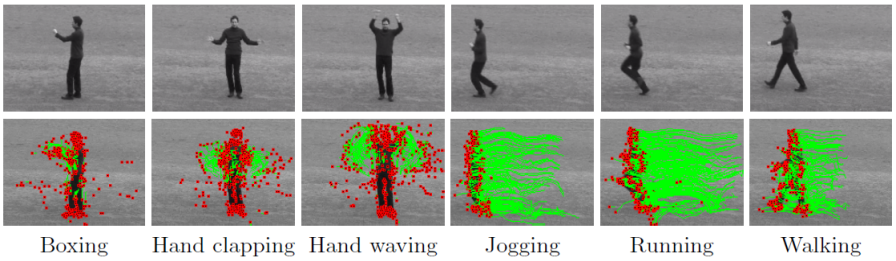

FIGURE 5.2 – Trajectoires extraites par *Video Extruder* sur les séquences KTH.

5.3 Stabilisation vidéo

Une des premières applications basées sur les trajectoires de *Video Extruder* que nous avons développées est la stabilisation vidéo. Les mouvements *pan-tilt* de la caméra ayant un impact significatif sur le champ de trajectoires, nous avons pu les utiliser pour stabiliser une vidéo. Nous nous sommes appuyés sur le fait qu'en général, le plus grand ensemble de trajectoires ayant des mouvements similaires correspond au déplacement du fond statique, c'est-à-dire de la caméra.

Notre méthode comporte trois étapes. Dans un premier temps, la translation 2D de l'image (correspondant au mouvement *pan-tilt* de la caméra) est estimée. Pour cela, à chaque instant t, nous projetons dans un histogramme 2D les déplacements de toutes les particules, et sélectionnons la translation $\vec{v_t}$ correspondant à la classe ayant reçu le plus de particules. Nous intégrons ensuite temporellement $\vec{v_t}$ pour obtenir S_t, le décalage de l'image courante par rapport à la première image. S_t est ensuite lissé temporellement pour donner $\tilde{S}_t$, calculé avec les formules suivantes :

$$\begin{aligned} S_t &= S_{t-1} + \vec{v_t} \\ V_t &= V_{t-1} + k(S_t - \tilde{S}_{t-1}) \\ \tilde{S}_t &= \tilde{S}_{t-1} + V_t \end{aligned}$$

Avec $k \in [0, 1]$ le paramètre de régularisation de la stabilisation, $k = 1$ correspondant à l'absence de stabilisation et $k = 0$ à la stabilisation extrême. La dernière étape de notre stabilisation est la génération de l'image stabilisée I_t^s, simple translation de l'image courante I_t:

$$I_t^s(p) = I_t(p + \tilde{S}_t - S_t)$$

La figure 5.3 montre deux images capturées à deux instants de la vidéo avant et après stabilisation. Bien que les temps de calcul de cette stabilisation soient très bas, (inférieurs à une milliseconde sans compter l'extraction des trajectoires), ces résultats ne sont que préliminaires. En effet, il est possible d'obtenir une meilleure stabilisation en estimant les rotations, changements

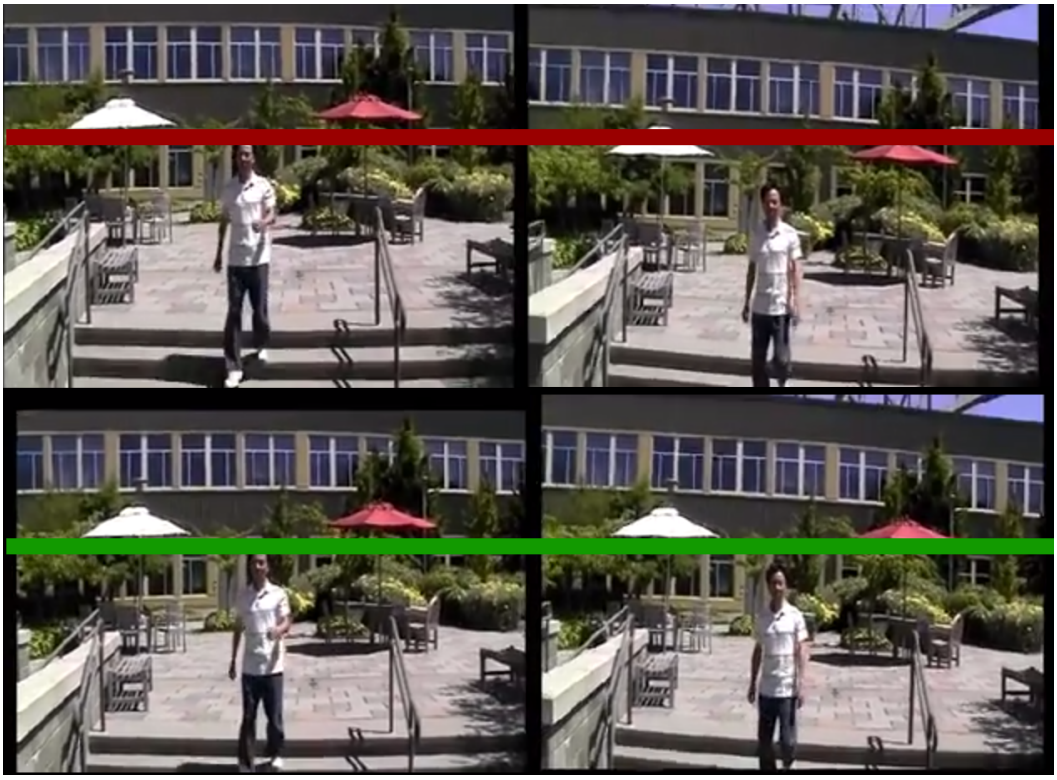

FIGURE 5.3 – Résultats de la stabilisation vidéo. Séquence de deux images avant stabilisation (Haut) et après stabilisation (Bas). La barre rouge (resp. verte) en surimpression sert de repère.

d'échelles, inclinaisons et changements de perspectives comme le font Grundman et al. [28] pour améliorer la qualité des vidéos YouTube. Toutefois, cela dépasse le cadre de cette application, qui avait simplement pour but de démontrer le potentiel de *Video Extruder*.

5.4 Segmentation d'objets mobiles

Le deuxième partenariat public/privé pour lequel nous avons appliqué nos travaux sur l'extraction de trajectoires est la segmentation d'objets mobiles observés par une caméra statique ou embarquée dans un aéronef.

Une fois les trajectoires extraites, nous avons modélisé le déplacement du fond par une homographie. Bien que cette transformation ne suffise pas à modéliser le mouvement d'un fond non planaire, elle s'est avérée suffisante pour le type de scènes qui intéressaient notre partenaire. La combinaison de RANSAC avec l'estimateur d'homographie nous a permis de gagner en robustesse aux trajectoires erronées. L'homographie estimée a ensuite servi à compenser les mouvements de caméra, ce qui a permis d'identifier les trajectoires appartenant à des objets mobiles. Finalement, les trajectoires ayant des mouvements similaires, mais aussi proches dans l'espace image, ont été regroupées et associées à un objet mobile par un algorithme de croissance de régions.

Bien que théoriquement le problème de la segmentation d'objets mobiles soit simple à formuler, la grande quantité de bruit et le manque de texture présent dans les vidéos de notre partenaire a significativement affecté la précision des mouvements estimés et la robustesse de l'estimateur d'homographie. Pour limiter cet impact, nous nous sommes appuyés sur *GPOF* (chap. 2) pour explorer l'espace des paramètres de l'algorithme et affiner ces derniers afin de se rapprocher au maximum de la vérité terrain fournie par notre partenaire. La figure 5.4 montre un objet mobile détecté dans une vidéo, ainsi que le faisceau de trajectoires.

À l'avenir, si le partenariat est reconduit, davantage de techniques pourront être explorées afin d'augmenter la robustesse de l'algorithme au bruit et au manque de texture. D'abord, l'utilisation de **flux optique dense avec régularisation spatiale** pourrait améliorer l'estimation de mouvement dans

FIGURE 5.4 – Illustration du fonctionnement de la segmentation d'objets mobiles. À gauche, l'objet détecté comme mobile est marqué par sa boîte englobante rose. À droite, nous avons affiché l'ensemble des particules suivies dans la vidéo, en marquant en rouge les trajectoires se détachant du mouvement fond.

les zones peu texturées. Ensuite, des algorithmes de **suppression de bruit vidéo** pourront être appliqués pour diminuer le nombre d'erreurs de l'estimation de mouvement. D'autres **modélisations de mouvement du fond** pourraient aussi fournir une meilleure précision que les homographies estimées actuellement. Finalement, des **méthodes de suivi d'objets** pourront améliorer la détection d'un objet mobile segmenté préalablement.

5.5 Reconnaissance d'action basée sur les faisceaux de trajectoires

Un certain nombre d'algorithmes de l'état de l'art dédiés à la reconnaissance d'action reposent sur un flux de trajectoires pour construire un modèle d'action. Très utile dans les applications de vidéosurveillance, ce problème a motivé un grand nombre de travaux ces dernières années. Plusieurs jeux de données fournissant des vidéos d'actions annotées ont été publiés, facilitant l'évaluation et la comparaison de telles approches.

En partant des trajectoires extraites par *Video Extruder*, Thanh Phuong Nguyen, un membre de notre équipe, s'est intéressé à la reconnaissance d'ac-

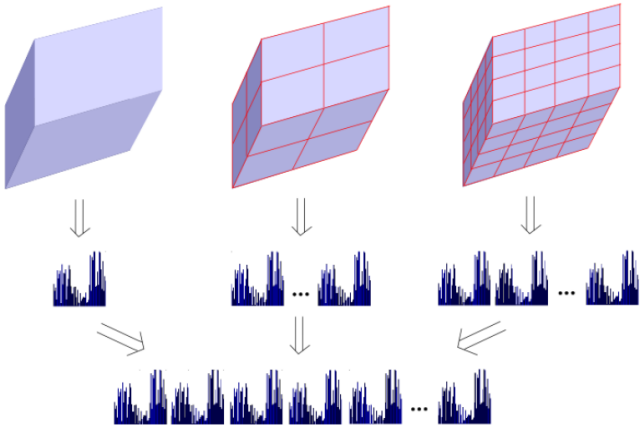

FIGURE 5.5 – Construction d'un descripteur basé sur la concaténation d'histogrammes calculés sur une partition de l'espace 2D+t. *Source:* [46]

tion en temps réel. Tirer parti de cette estimation de mouvement nous permet d'atteindre des temps de calcul significativement inférieurs à l'état de l'art, qui utilise généralement des algorithmes bien plus coûteux.

Toutes les méthodes présentées [46, 47, 48] ont deux points communs : d'abord la construction d'un descripteur de vidéo, basée sur la concaténation d'histogrammes calculés dans chaque sous-ensemble d'une partition hiérarchique de l'espace 2D+t (voir figure 5.5). Puis, l'utilisation de classifieurs SVM [62] (Support Vector Machine) multiclasses pour associer les descripteurs aux différentes actions.

Chaque méthode diffère par son approche pour calculer les histogrammes (appelés dans la suite descripteurs 2D+t), dont la concaténation donne le descripteur d'action transmis au classifieur final (voir figure 5.5).

Dans [47], les descripteurs 2D+t encodent la répartition des points dans un espace de descripteur discret, chaque point décrivant la tendance en une trajectoire à un instant t donné. Le descripteur utilisé encode la norme et l'orientation de la particule, et les deux signes des variations d'orientation entre l'instant t et les instants $t-1$ et $t+1$. Dans [48], cette information d'orientation a été remplacée par une information encodant les changements d'apparence de patches avoisinant la particule, ce qui a significativement amélioré (+7%) le taux de reconnaissance sur le jeu de données UCF Youtube.

	UCF Youtube	KTH
[48]	**72.07**	93.33
[47]	65.63	92.5
[46]	65.1	**95**

TABLE 5.1 – Taux de reconnaissance des trois méthodes proposées sur les jeux de données UCF Youtube et KTH.

Une méthode [46] fondée sur la technique des sacs de mots [59] a aussi été proposée pour construire des descripteurs 2D+t. Dans un premier temps, l'ensemble des points clefs (maxima locaux de l'accélération radiale calculés à plusieurs échelles, le long de chaque trajectoire) est extrait du faisceau de trajectoires. L'ensemble des caractéristiques locales calculées en chaque point clef (encodées dans un vecteur de dimension 14) est quantifié grâce à l'algorithme des K-moyennes pour former un dictionnaire de « gestes atomiques ». Les gestes atomiques (mots du dictionnaire) sont alors reprojetés dans l'espace 2D+t et le nombre d'occurrences de chaque mot est compté pour construire le descripteur 2D+t.

Le tableau 5.1 montre les scores des trois méthodes proposées sur les jeux de données UCF Youtube et KTH.

5.6 Conclusion

Nous avons brièvement présenté dans ce chapitre quatre applications développées au cours de nos travaux. La première a consisté en l'adaptation de *Video Extruder* sur des processeurs embarqués et en l'extraction de statistiques pour la détection d'événements en temps réel. Ensuite, une stabilisation vidéo compensant les translations 2D de l'image a été implémentée. Cela a confirmé les capacités de *Video Extruder* à capturer les mouvements *pan-tilt* de la caméra. Puis, lors d'un partenariat public/privé, nous avons développé une segmentation d'objets mobiles fonctionnant sur caméra statique et mobile aéroportée. Pour cela, le mouvement de la caméra a été approximé par une homographie, et les trajectoires appartenant à des objets mobiles détectées à partir de statistiques extraites sur les faisceaux de trajectoires.

Finalement, des descripteurs d'action haut niveau basés sur les trajectoires nous ont permis de modéliser et reconnaître les actions humaines et ont donné lieu à plusieurs publications scientifiques.

Ces applications ont ainsi valorisé les travaux présentés dans les chapitres précédents de ce manuscrit. Les optimisations algorithmiques et logicielles contenues dans *Video Extruder* ont permis d'atteindre les fréquences de traitements requises, même sur des processeurs embarqués. D'autre part, les outils *Video++* (section 2.2) et *GPOF* (section 2.3) nous ont facilité l'écriture du code et l'optimisation des performances des algorithmes.

À l'avenir, d'autres applications pourront être envisagées, entre autres le débruitage de vidéos, le SLAM (estimation simultanée du mouvement de caméra et de la géométrie 3D de la scène) ou encore la caractérisation de mouvements pathologiques [41].

Chapitre 6

Conclusion et perspectives

La problématique à la base de nos travaux est la suivante : comment accélérer l'estimation de mouvement pour la rendre accessible aux applications à fortes contraintes de temps de calcul ?

Deux niveaux d'optimisation sous-tendent cette problématique : la minimisation du temps de calcul par l'optimisation de l'implémentation logicielle, et celle de l'algorithme. Dans les deux cas, tous les facteurs impactant le temps de calcul sur le processeur ciblé sont à prendre en compte : la consommation mémoire, l'exploitation de la mémoire cache, la nature des opérations arithmétiques, l'utilisation des parallélismes SIMD et MIMD, etc.

Cette dualité nous a conduit à deux types de contributions : celles concernant le génie logiciel, et celles concernant l'algorithme d'analyse du mouvement.

Pour ce qui concerne le génie logiciel, nous avons montré que les avancées du langage C++, les progrès des compilateurs pour la vectorisation SIMD automatique et les nouveaux outils comme OpenMP facilitant la parallélisation permettent de **simplifier l'écriture de traitement d'image haute performance**. Ces travaux ont abouti à la publication d'une bibliothèque open source *Video++* contenant des abstractions permettant d'écrire facilement des algorithmes de traitement d'image performants.

Pour faciliter la tâche, souvent fastidieuse, de **l'évaluation du comportement de l'algorithme selon différents jeux de paramètres**, ou différentes configurations, nous avons proposé une bibliothèque Python auto-

matisant certaines parties du processus. Cet outil a aussi été publié dans le domaine open source, sous le nom de *GPOF* (*Generic Parameter Optimization Framework*).

Nous avons validé l'intérêt de *Video++* et *GPOF*, et nous les avons perfectionnés en les utilisant quotidiennement dans nos recherches pour atteindre de meilleurs compromis qualité/temps de calcul.

Nos travaux sur l'analyse de mouvement ont eux aussi abouti à plusieurs contributions. Dans un premier temps, nous avons montré qu'il est possible de calculer un champ de plusieurs milliers de trajectoires à une cadence de plus de 150 images par seconde sur un CPU multi-cœurs ou un GPU. Bien que la précision du mouvement estimé par cet algorithme soit légèrement inférieure à la version pyramidale de Lucas-Kanade [7], nous avons pu en tirer parti pour construire des méthodes rapides de reconnaissance d'actions [49, 47, 48], de stabilisation vidéo et de segmentation d'objets mobiles.

Puis, nous avons proposé une méthode pour accélérer la mise en correspondance de deux ensembles de points d'intérêt épars. Nous avons montré qu'une indexation unidimensionnelle permet une recherche jusqu'à un ordre de grandeur plus rapide que l'indexation par des arbres de type KD-tree sur des index de moins de 50 000 points. Nous avons aussi montré que cet index peut être combiné avec une indexation des points sur leurs positions spatiales pour gagner un ordre de grandeur supplémentaire sur le temps de calcul.

La dernière contribution de nos travaux sur l'analyse de mouvement est une estimation de flux optique assistée par l'odométrie visuelle, ou flux épipolaire. En moins d'un tiers de seconde, notre méthode permet d'estimer un flux optique couvrant en moyenne 50% des pixels avec le taux d'erreur le plus faible du classement *KITTI optical flow* (section non-dense). Dans ces travaux, nous avons tiré parti des lignes épipolaires et de la cohérence locale du flux optique pour diminuer le temps de calcul et le nombre d'erreurs. Nous avons aussi montré que l'utilisation d'un autre algorithme, ici Lucas-Kanade, permet d'éliminer une grande proportion d'erreurs.

Dans la lignée de nos avancées sur les axes génie logiciel et analyse de mouvement, plusieurs pistes restent à explorer et pourraient mener à de meilleurs compromis qualité / temps de calcul.

Par exemple, la bibliothèque *Video++* étant limitée aux architectures des CPU multi-cœurs, elle ne permet pas la parallélisation de code sur le GPU, une architecture massivement parallèle pourtant très adaptée au traitement d'image. Le choix de ne pas intégrer le GPU dans cette version de *Video++* est lié au fait que, bien que plusieurs frameworks de programmation GPU aient été annoncés (C++AMP, CUDA et SYCL), leur intégration est compliquée : CUDA n'est disponible que sur les GPU NVidia et ne gère pas le C++14, C++AMP ne fonctionne que sur Windows, et aucune implémentation de SYCL n'est accessible au grand public. Toutefois, il est possible que dans quelques années l'industrie converge vers une solution multi-plateformes. Il sera alors possible de simplifier l'écriture d'applications de traitement d'image GPU avec des abstractions similaires à celles que nous avons proposées pour le CPU.

Ensuite, bien que le projet *GPOF* accélère la mise au point d'algorithmes, l'ensemble de jeux de paramètres qu'il est capable d'explorer est très largement limité par la puissance de calcul de l'ordinateur utilisé. Il sera possible à l'avenir de contourner cette limitation en distribuant l'exploration de l'espace des paramètres sur un cluster de plusieurs machines. Cela permettra d'explorer plus rapidement plus de solutions et, entre autres, d'aboutir à de meilleurs compromis qualité / temps de calcul.

Sur le plan algorithmique, nos recherches sur l'analyse de mouvement pourront aussi être étendues. D'abord, il sera possible de combiner les idées de *Video Extruder* pour accélérer notre estimation du flux épipolaire. En effet, les temps de calcul de *Video Extruder* étant largement inférieurs à ceux du flux épipolaire, son intégration aurait un coût négligeable. Ensuite, les méthodes d'optimisation globale, que nous avons évitées par souci de temps de calcul, pourront en fait être implémentées sur de petites résolutions pour en limiter l'impact sur le temps de calcul. Cela permettra entre autres une meilleure couverture des zones faiblement texturées.

Aussi, dans la continuité de nos travaux sur la mise en correspondance des descripteurs de points, il sera possible de combiner nos deux indexations à celles de l'état de l'art comme les KD-tree. Cela pourrait permettre d'aboutir à une méthode hybride, efficace à la fois sur des petits et grands ensembles de points.

Pour favoriser la concrétisation de ces perspectives, nous avons publié sous licence libre nos codes sous forme de bibliothèques Python et C++ réutilisables. Les chercheurs intéressés par nos travaux pourront ainsi facilement reproduire nos résultats et les faire rapidement évoluer vers de futures améliorations.

Liste des productions scientifiques

Logiciels

— **Video++**: Bibliothèque de traitement d'image C++14. Disponible sur GitHub: https://github.com/matt-42/vpp

— **GPOF**: Outils Python pour l'aide à l'exploration de l'espace des paramètres d'un algorithme. Disponible sur GitHub: https://github.com/matt-42/GPOF

— **Iod**: Introspection statique en C++ et divers outils de méta-programmation. Disponible sur GitHub: https://github.com/matt-42/iod

— **CuImg**: Bibliothèque de traitement d'image C++ pour la programmation hybride CPU/GPU (avec CUDA). Dépréciée. Disponible sur GitHub: https://github.com/matt-42/cuimg

— **Détection temps réel d'évènements anormaux**, développée lors du projet ITEA2-SPY.

— **Segmentation d'objets mobiles**: code confidentiel, développé pour un partenaire privé.

— **Odométrie visuelle et implémentation du flux épipolaire**: code en cours d'intégration à la bibliothèque *Video++*.

— **Optimisation multicoeur-SIMD du détecteur FAST**: code intégré à la bibliothèque *Video++*.

— **Impp**: Bibliothèque de traitement d'image minimaliste compatible C++98. Développée pour un partenaire.

Publications dans des conférences internationales à comité de lecture

— Matthieu Garrigues and Antoine Manzanera. **Real time semi-dense**

point tracking. In A.J.C. Campilho and M.S. Kamel, editors, *Int. Conf. on Image Analysis and Recognition (ICIAR 2012)*, volume 7324 of Lecture Notes in Computer Science, pages 245–252, Aveiro, Portugal, june 2012. Springer.

— Matthieu Garrigues and Antoine Manzanera. **Video++, a modern image and video processing C++ framework**. In *Design and Architectures for Signal and Image Processing (DASIP)*, 2014 Conference on, pages 1–6. IEEE, 2014.

— Matthieu Garrigues and Antoine Manzanera. **Fast Semi Dense Epipolar Flow Estimation**. In *IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2017

— Thanh Phuong Nguyen, Antoine Manzanera, and Matthieu Garrigues. **Motion trend patterns for action modelling and recognition**. In *Interna- tional Conference on Computer Analysis of Images and Patterns*, pages 360–367. Springer Berlin Heidelberg, 2013.

— Thanh Phuong Nguyen, Antoine Manzanera, Ngoc-Son Vu, and Matthieu Garrigues. **Revisiting lbp-based texture models for human action recognition**. In *Iberoamerican Congress on Pattern Recognition*, pages 286–293. Springer Berlin Heidelberg, 2013.

Publications dans des revues internationales à comité de lecture

— Matthieu Garrigues, Antoine Manzanera, and Thierry M Bernard. **Video extruder: a semi-dense point tracker for extracting beams of trajectories in real time**. *Journal of Real-Time Image Processing*, 11(4):785–798, 2016.

— Thanh Phuong Nguyen, Antoine Manzanera, Matthieu Garrigues, and Ngoc-Son Vu. **Spatial motion patterns: action models from semi-dense trajectories**. *International Journal of Pattern Recognition and Artificial Intelligence*, 28(07):1460011, 2014.

Rapports techniques

— Contribution aux comptes rendus scientifiques des conventions DGA *vision basse consomation* et *Vision RGBD*.

— Contribution aux comptes rendus scientifiques du projet SPY et d'un partenariat classé confidentiel.

Bibliographie

[1] Aselsan. http://www.aselsan.com.tr.

[2] Eolane. http://corporate.eolane.com.

[3] Andreas Geiger and Philip Lenz and Raquel Urtasun. Optical flow evaluation 2012.

[4] Bay, H., Tuytelaars, T., and Van Gool, L. Surf: Speeded up robust features. *Lecture Notes in Computer Science* (2006), 404–417.

[5] Bentley, J. L. Multidimensional binary search trees used for associative searching. *Commun. ACM 18*, 9 (Sep 1975), 509–517.

[6] Bleyer, M., Rother, C., and Kohli, P. Surface stereo with soft segmentation. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on* (2010), IEEE, pp. 1570–1577.

[7] Bouguet, J.-Y. Pyramidal implementation of the affine Lucas Kanade feature tracker description of the algorithm. *Intel Corporation* (2001).

[8] Bradski, G. The OpenCV library. *Dr. Dobb's Journal of Software Tools* (2000).

[9] Brostow, G. J., Shotton, J., Fauqueur, J., and Cipolla, R. Segmentation and recognition using structure from motion point clouds. In *ECCV (1)* (2008), pp. 44–57.

[10] Brox, T., Bruhn, A., Papenberg, N., and Weickert, J. High accuracy optical flow estimation based on a theory for warping. In *European conference on computer vision* (2004), Springer, pp. 25–36.

[11] Burrus, N., Duret-Lutz, A., Géraud, T., Lesage, D., and Poss, R. A static C++ object-oriented programming (SCOOP) paradigm

mixing benefits of traditional OOP and generic programming. In *Proceedings of the Workshop on Multiple Paradigm with Object-Oriented Languages (MPOOL)* (Anaheim, CA, USA, Oct. 2003).

[12] Cpp Reference. Shared pointer. http://www.cplusplus.com/reference/memory/shared_ptr.

[13] Demetz, O., Stoll, M., Volz, S., Weickert, J., and Bruhn, A. Learning brightness transfer functions for the joint recovery of illumination changes and optical flow. In *European Conference on Computer Vision* (2014), Springer, pp. 455–471.

[14] Douglas, G., Jaakko, J., and Jens, M. Proposed wording for variadic templates (revision 2). http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2007/n2242.pdf.

[15] Eric Munier. Surveillance improved system, projet européen labélisé itea2. https://itea3.org/project/spy.html.

[16] Evans, C. C. YAML: YAML Ain't Markup Language. http://yaml.org/. [Online].

[17] Farnebäck, G. Two-frame motion estimation based on polynomial expansion. In *Image Analysis*. Springer, 2003, pp. 363–370.

[18] Fischler, M. A., and Bolles, R. C. Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. *Readings in Computer Vision* (1987), 726–740.

[19] Flynn, M. J. Some computer organizations and their effectiveness. *IEEE transactions on computers 100*, 9 (1972), 948–960.

[20] Fog, A. Software optimization resources. http://agner.org/optimize/.

[21] Garrigues, M. Generic Parameter Optimization Framework - Git Repository. https://github.com/matt-42/GPOF. [Online].

[22] Garrigues, M. Iod - Git Repository. https://github.com/matt-42/iod. [Online].

[23] Garrigues, M. Video++ - Git Repository. https://github.com/matt-42/vpp. [Online].

[24] GCC. Vectorizable loops. https://gcc.gnu.org/projects/tree-ssa/vectorization.html.

[25] Geiger, A., Lenz, P., and Urtasun, R. Are we ready for autonomous driving? the KITTI vision benchmark suite. In *Conference on Computer Vision and Pattern Recognition (CVPR)* (2012).

[26] Google. ATAP Project Tango - Google, Feb. 2014.

[27] Gregor, D. Range-based for loop wording.

[28] Grundmann, M., Kwatra, V., and Essa, I. Auto-directed video stabilization with robust l1 optimal camera paths. In *Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference* (2011), IEEE, pp. 225–232.

[29] Hoberock, J., and Bell, N. Thrust: A parallel template library, 2010. Version 1.7.0.

[30] Horn, B. K., and Schunck, B. G. Determining optical flow. *Artificial intelligence 17*, 1-3 (1981), 185–203.

[31] Hunter, J. D. Matplotlib: A 2d graphics environment. *Computing In Science & Engineering 9*, 3 (2007), 90–95.

[32] Ibáñez, L., Schroeder, W., Ng, L., and Cates, J. *The ITK Software Guide*. Kitware, 2003.

[33] Isard, M., and Blake, A. CONDENSATION - conditional density propagation for visual tracking. *International Journal of Computer Vision 29* (1998), 5–28.

[34] ISO. *ISO/IEC 14882:2014 Information technology – Programming languages – C++*.

[35] Köthe, U. *Reusable software in computer vision*. Academic Press, 1999.

[36] Kroeger, T., Timofte, R., Dai, D., and Van Gool, L. Fast optical flow using dense inverse search. *arXiv preprint arXiv:1603.03590* (2016).

[37] Levin, A., Fergus, R., Durand, F., and Freeman, W. T. Image and depth from a conventional camera with a coded aperture. *ACM transactions on graphics (TOG) 26*, 3 (2007), 70.

[38] Longuet-Higgins, H. A computer algorithm for reconstructing a scene from two projections. *Readings in Computer Vision* (1987), 61–62.

[39] Lowe, D. G. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision 60*, 2 (2004), 91–110.

[40] Lucas, B. D., Kanade, T., et al. An iterative image registration technique with an application to stereo vision. In *IJCAI* (1981), vol. 81, pp. 674–679.

[41] Martínez Carrillo, F., et al. *Characterization and modelling of complex motion patterns*. PhD thesis, Universidad Nacional de Colombia.

[42] Moore, G. Cramming more components onto integrated circuits. *Readings in computer architecture 56* (2000).

[43] Moulon, P., Monasse, P., and Marlet, R. Adaptive structure from motion with a contrario model estimation. In *Asian Conference on Computer Vision* (2012), Springer, pp. 257–270.

[44] Muja, M., and Lowe, D. G. Fast approximate nearest neighbors with automatic algorithm configuration. In *International Conference on Computer Vision Theory and Application VISSAPP'09)* (2009), INSTICC Press, pp. 331–340.

[45] Ng, R., Levoy, M., Brédif, M., Duval, G., Horowitz, M., and Hanrahan, P. Light field photography with a hand-held plenoptic camera. *Computer Science Technical Report CSTR* (2005).

[46] Nguyen, T. P., and Manzanera, A. Action recognition using bag of features extracted from a beam of trajectories. In *2013 IEEE International Conference on Image Processing* (2013), IEEE, pp. 4354–4357.

[47] Nguyen, T. P., Manzanera, A., and Garrigues, M. Motion trend patterns for action modelling and recognition. In *International Conference on Computer Analysis of Images and Patterns* (2013), Springer Berlin Heidelberg, pp. 360–367.

[48] Nguyen, T. P., Manzanera, A., Garrigues, M., and Vu, N.-S. Spatial motion patterns: action models from semi-dense trajectories. *International Journal of Pattern Recognition and Artificial Intelligence 28*, 07 (2014), 1460011.

[49] Nguyen, T. P., Manzanera, A., Vu, N.-S., and Garrigues, M. Revisiting lbp-based texture models for human action recognition. In

Iberoamerican Congress on Pattern Recognition (2013), Springer Berlin Heidelberg, pp. 286–293.

[50] No bugs hare. http://ithare.com/infographics-operation-costs-in-cpu-clock-cycles.

[51] Plyer, A., Le Besnerais, G., and Champagnat, F. Massively parallel Lucas Kanade optical flow for real-time video processing applications. *Journal of Real-Time Image Processing 11*, 4 (2016), 713–730.

[52] Ragan-Kelley, J., Barnes, C., Adams, A., Paris, S., Durand, F., and Amarasinghe, S. Halide: a language and compiler for optimizing parallelism, locality, and recomputation in image processing pipelines. In *Proceedings of the 34th ACM SIGPLAN conference on Programming language design and implementation* (2013), ACM, pp. 519–530.

[53] Richard, H. In defence of the 8-point algorithm. In *Proceedings of the 5th International Conference on Computer Vision, IEE Computer Science Press, Cambridge* (1995), pp. 1064–1070.

[54] Rosten, E., and Drummond, T. Fusing points and lines for high performance tracking. In *IEEE International Conference on Computer Vision* (October 2005), vol. 2, pp. 1508–1511.

[55] Rosten, E., and Drummond, T. Machine learning for high-speed corner detection. In *European Conference on Computer Vision (ECCV'06)* (May 2006), vol. 1, pp. 430–443.

[56] Rosten, E., Porter, R., and Drummond, T. Faster and better: A machine learning approach to corner detection. *IEEE Trans. Pattern Analysis and Machine Intelligence 32* (2010), 105–119.

[57] Sand, P., and Teller, S. Particle video: Long-range motion estimation using point trajectories. In *Computer Vision and Pattern Recognition (CVPR'06)* (New York, June 2006), pp. 2195–2202.

[58] Senst, T., Geistert, J., Keller, I., and Sikora, T. Robust local optical flow estimation using bilinear equations for sparse motion estimation. In *ICIP* (2013), pp. 2499–2503.

[59] Sivic, J., and Zisserman, A. Efficient visual search of videos cast as text retrieval. *IEEE transactions on pattern analysis and machine intelligence 31*, 4 (2009), 591–606.

[60] Tschumperlé, D. The CImg Library. http://cimg.sourceforge.net/.

[61] Vandevoorde, D. New wording for C++0x lambdas (rev. 2). http://www.open-std.org/JTC1/SC22/WG21/docs/papers/2009/n2927.pdf.

[62] Vedaldi, A., and Zisserman, A. Efficient additive kernels via explicit feature maps. *IEEE transactions on pattern analysis and machine intelligence 34*, 3 (2012), 480–492.

[63] Vogel, C., Schindler, K., and Roth, S. 3d scene flow estimation with a piecewise rigid scene model. *International Journal of Computer Vision 115*, 1 (2015), 1–28.

[64] Yamaguchi, K., Hazan, T., McAllester, D., and Urtasun, R. Continuous markov random fields for robust stereo estimation. In *European Conference on Computer Vision* (2012), Springer, pp. 45–58.

[65] Yamaguchi, K., McAllester, D., and Urtasun, R. Robust monocular epipolar flow estimation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition* (2013), pp. 1862–1869.

[66] Zhang, Y., Gong, M., and Yang, Y.-H. Local stereo matching with 3d adaptive cost aggregation for slanted surface modeling and sub-pixel accuracy. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on* (2008), IEEE, pp. 1–4.

[67] Zhang, Z. Microsoft kinect sensor and its effect. *IEEE MultiMedia 19*, 2 (Apr. 2012), 4–10.

université PARIS-SACLAY
ÉCOLE DOCTORALE
INTERFACES
Approches Interdisciplinaires : Fondements, Applications et Innovation

Titre : Accélération Algorithmique et Logicielle de l'Analyse Vidéo du Mouvement

Mots clefs : Traitement d'image, Vision par ordinateur, Calcul parallèle, Estimation du mouvement, Flux optique, Génie logiciel

Résumé :
L'analyse du mouvement dans une vidéo consiste à estimer, à partir d'une séquence d'images, le déplacement apparent des objets projetés sur le plan focal d'une caméra, statique ou mobile. Un grand nombre de domaines comme la robotique, la vidéo surveillance, le cinéma ou encore les applications militaires, reposent sur cette analyse pour interpréter le contenu d'une vidéo. Ce problème a été l'un des premiers à être approché par les chercheurs en traitement d'image. De nombreuses solutions ont été proposées et permettent une estimation suffisamment précise et robuste pour un grand nombre d'applications. Cependant, la complexité algorithmique de ces solutions et/ou le manque d'optimisations de leur implantations logicielles rendent leur utilisation dans les applications à forte contraintes de calculs difficile voire impossible.
Dans les travaux présentés dans cette thèse, nous avons optimisé trois types d'analyses de mouvement en prenant en compte, non seulement la complexité algorithmique, mais aussi tous les facteurs impactant le temps de calcul sur les processeurs actuels comme la parallélisation, la consommation mémoire, la régularité des accès mémoire ou encore le type des opérations arithmétiques. Cette diversité des problématiques nous a conduits à élaborer notre thèse à l'intersection des domaines du **génie logiciel** et du **traitement d'image**. Nos contributions ont permis le développement d'applications temps réel comme la reconnaissance d'actions, la stabilisation vidéo et la segmentation d'objets mobiles.

Title : Algorithmic and Software Acceleration of Video Motion Analysis

Keywords : Image processing, Computer Vision, Parallel computing, Motion estimation, Optical flow, Software engineering

Abstract :
Motion analysis in a video consists in estimating, from a sequence of images, the displacement of the objects projected on the focal plane of a camera, static or mobile. A large number of fields such as robotics, video surveillance, cinema or military applications rely on this analysis to interpret the content of a video.
This problem was one of the first to be approached by researchers in image processing. Numerous solutions have been proposed and allow a sufficiently accurate and robust estimate for a large number of applications. However, the algorithmic complexity of these solutions and/or the lack of optimizations of their software implementations make their use in applications with high computational constraints difficult or impossible.
In the work presented in this thesis, we optimized three types of motion analysis taking into account not only the algorithmic complexity, but also all the factors affecting computation time on current processors such as parallelization, memory consumption, the regularity of memory accesses, or the type of arithmetic operations. This led us to develop our thesis at the intersection of **software engineering** and **image processing**. Our contributions have enabled the development of real-time applications such as action recognition, video stabilization and segmentation of mobile objects.

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