Architectures massivement parallèles et vision artificielle bas-niveau

Aurélien Plyer

N° d'ordre: 1234

THÈSE

Présentée pour obtenir

LE GRADE DE DOCTEUR EN SCIENCES
DE L'UNIVERSITÉ PARIS XIII

Spécialité: Traitement du signal et image

par

Aurélien Plyer

Architectures Massivement Parallèles en Vision Artificielle Bas Niveau

Soutenue le 27/02/2013 devant la Commission d'examen:

M.	Guy Le Besnerais	
M.	Frédéric Champagnat	
Mme.	Françoise Dibos	(Directeur de thèse)
M.	Jacques Froment	
M.	Houzet Dominique	(Raporteur)
M.	Giovannelli Jean-François	(Raporteur)

Thèse

présentée pour obtenir
LE GRADE DE DOCTEUR EN SCIENCES
DE L'UNIVERSITE PARIS 13
par

Aurélien Plyer

Architectures massivement parallèles et vision artificielle bas-niveau

4 janvier 2013

Table des matières

1 Introduction

L'un des principaux problèmes de la vision par ordinateur est qu'une image (et plus encore une vidéo) représente une masse très importante de données à traiter qui dépassait, jusqu'à récemment, les capacités des ordinateurs grand public. Les avancées technologiques récentes en termes de puissance de calcul des processeurs constituent une opportunité pour permettre la réalisation de systèmes de perception fonctionnant à des cadences suffisamment élevées pour leur utilisation dans des chaînes de traitement complexes.

Une telle avancée nous ouvre la porte de nombreux domaines qui étaient jusqu'alors bridés par la gourmandise computationnelle des algorithmes de perception. Par exemple on peut considérer que la fermeture de la boucle perception/action sur un micro-drone est désormais réalisable, autrement dit que les architectures de calculs embarquables et les algorithmes de perception embarqués sont arrivés à des niveaux de maturité suffisants pour permettre un fonctionnement robuste à cadence vidéo.

Cependant, ces avancées remarquables ne se feront pas sans contre-partie, car comme le disait Sutter en 2005 : "*The free lunch is over*" [Sutter, 2005]. Cette affirmation concerne le fait qu'à partir de 2004, il est devenu nécessaire d'investir dans la programmation parallèle car les gains en performance des ordinateurs ne sont plus dus à la simple augmentation de leur fréquence d'horloge, mais à l'augmentation du nombre de coeurs de traitement, donc à l'introduction du parallélisme dans leur fonctionnement. Elle est d'autant plus vraie depuis 2007 avec la diffusion à grand échelle d'architectures massivement parallèles et facilement programmables : les GPUs (*Graphic Processing Unit*).

Notre travail concerne la prise en compte de cette évolution récente des architectures des moyens de calculs pour le développement de solutions de vision bas-niveau rapides. Nous insistons sur la vitesse de calcul, sachant que beaucoup d'équipes concurrentes ont plutôt utilisé les GPU pour accélérer des algorithmes complexes et obtenir des solutions plus rapidement, typiquement de la seconde à la minute. Au contraire nous cherchons à optimiser un compromis coût-complexité pour parvenir à des solutions à cadence vidéo, ou du moins des solutions en "*temps interactif*", c'est-à-dire permettant une interaction confortable avec un opérateur humain, tout en privilégiant systématiquement la robustesse et la simplicité de nos

algorithmes. Nous visons donc des temps de traitement de 10 à 100ms, soit au moins un ordre de grandeur en dessous de la plupart des travaux concurrents.

Synthèse des contributions et plan du document

Dans le chapitre 2, nous proposons un bref résumé historique de l'évolution des ordinateurs modernes, puis nous tentons de montrer que l'effort de développement rendu nécessaire par les nouvelles architectures massivement parallèles n'est pas si difficile. En particulier, nous esquissons un ensemble de schémas généraux de calculs qui permettent de couvrir la plupart des besoins de la vision bas-niveau. Nous proposons de plus un outil permettant d'évaluer et de représenter la complexité des algorithmes massivement parallèles.

L'objectif de la suite du travail est d'appliquer ces principes à la vision bas-niveau, pour développer des solutions rapides. Durant cette thèse, nous avons privilégié une approche *bottom-up* dans la construction des solutions algorithmiques. Notre première contribution, qui fait l'objet du chapitre 3 concerne donc une fonction de base, l'estimation du flot optique.

Partant d'un algorithme de recalage de fenêtres locales, de type Lucas-Kanade itératif, dénommé FOLKI [Le Besnerais and Champagnat, 2005], nous avons proposé en 2009 une implémentation parallèle extrêmement efficace sur GPU. Avec un temps de calcul de l'ordre de 20ms pour une séquence full HD, FOLKI est encore, à notre connaissance, la plus rapide des implémentations de flot optique dense disponibles sur une architecture grand public.

Cette implémentation a eu un grand impact au sein même de l'ONERA dans le domaine de la mesure de mouvement par imagerie, en particulier en imagerie par vélocimétrie de particules (PIV). Le logiciel FOLKI-SPIV, développé au département DAFE de l'ONERA par Yves Le Sant à partir de FOLKI, permet une cadence de traitement très supérieure aux logiciels commerciaux, tout en étant plus simple d'emploi et en offrant une précision équivalente. FOLKI-SPIV a fait l'objet de la publication de revue [Champagnat et al., 2011], qui est rappelée en Annexe A. C'est aujourd'hui l'outil de référence pour le traitement de données PIV à l'ONERA et il contribue aussi à l'offre des souffleries commerciales de l'ONERA.

Nous avons ensuite cherché à étendre le domaine d'emploi de FOLKI à des contextes moins contrôlés que ceux de la mesure physique, sur des séquences de test utilisées en vision par ordinateur. Partant des travaux menés en 2011 par Riadh Fezzani, nous avons étudié de nombreuses solutions de type "CLG", c'est-à-dire combinant le recalage de fenêtres locales avec un terme de régularisation globale. Ces travaux n'ont pas débouché sur une solution satisfaisante. En effet nous avons constaté que le nombre d'itérations nécessaires pour parvenir à exploiter le terme de régularisation globale était trop important, conduisant à des temps de traitements trop élevés (typiquement au-dessus de la seconde pour des formats 640×480).

Prenant exemple sur des articles très pragmatiques concernant le flot optique [Sun et al., 2010] et la stéréovision [Brown et al., 2003], nous avons proposé des améliorations de FOLKI, principalement un préfiltrage des images par un filtre de rang (Rank) et une adaptation itérative de la taille de fenêtre de corrélation. Ces modifications simples ne remettent pas en cause la structure algorithmique de FOLKI, les codes obtenus profitent donc toujours des architectures parallèles, ce qui donne des coût de calcul faibles, de l'ordre de 50ms. Elles permettent un gain très important sur les images réelles, et conduisent à une méthode suffisamment robuste pour atteindre la performance des 5 premières méthodes référencées sur le site d'évaluation de techniques de flot optique Kitti (`http://www.cvlibs.net/datasets/kitti/index.php`). La version de FOLKI obtenue reste largement plus rapide que les méthodes publiées sur ce site, même si les comparaisons de coût de calcul ne peuvent être menées avec précision, car les matériels et les langages utilisés diffèrent. Surtout, nous profitons de la très bonne précision des algorithmes de recalage de fenêtre, avec une erreur moyenne très inférieure aux techniques équivalentes.

Au chapitre 4, nous abordons la super-résolution sur vidéos. Nous avons choisi d'étendre les méthodes précédemment étudiées au DTIM [Létienne, 2010, Rochefort, 2005] à des méthodes basées sur des mouvements non paramétriques, et donc bien évidement d'exploiter ainsi nos travaux sur les estimateurs de flot optique.

Pour obtenir des solutions rapides, nous avons montré à nouveau l'importance du modèle de formation d'image approximatif "warpBR", consistant à intervertir l'opérateur de flou et l'opérateur de mouvement. Ce modèle avait déjà été étudié par Antoine Létienne [Létienne, 2010] au DTIM pour de la SR sur des zones réduites (petits objets). En se basant sur ce modèle et une reconstruction en deux temps par fusion/déconvolution avec une déconvolution par convolution par un noyau de dimension faible (small kernel), nous obtenons un algorithme de SR extrêmement rapide. Cette technique permet ainsi de faire, à cadence vidéo, de la SR à facteur 6 sur une séquence 350×250 en traitant des groupes de 21 images. A notre connaissance, il s'agit de nouveau de la solution SR la plus rapide disponible à l'heure actuelle sur des machines grand public.

Suivant le plan adopté pour le flot optique, nous avons ensuite cherché à proposer des solutions de meilleures qualité tout en conservant un temps de calcul faible ce qui nous à conduit à étudier des versions plus complexes d'algorithmes de super-résolution. En utilisant toujours le modèle "WarpBR" en conjonction cette fois-ci avec un terme de régularisation, nous avons pu développer toute une famille d'algorithmes itératifs intégrés à un framework de minimisation d'énergie.

Nous avons ainsi pu étudier l'intérêt de l'introduction de fonctions robustes, d'abord sur le

terme de régularisation de cette fonctionnelle, puis sur le terme d'attache aux données, afin de pouvoir minimiser les artefacts générés par l'utilisation d'un modèle de mouvement générique (le flot optique) dans le processus de super-résolution.

Pour finir, nous aurons bien évidement une conclusion présentant de manière générale nos perspectives futures ainsi que des travaux en cours à l'heure de la rédaction de ce manuscrit. Les perspectives propres à chacun des thèmes des chapitres de ce manuscrit sont présentés à la fin de ceux-ci. De plus, comme chaque chapitre porte sur une thématique différente, nous avons fait le choix de séparer la bibliographie par chapitre, cela nous a paru plus pertinent.

Sur ce je vous souhaite une bonne lecture tout en espérant qu'elle vous sera au moins agréable, voir instructive.

Bibliographie

[Brown et al., 2003] Brown, M., Burschka, D., and Hager, G. (2003). Advances in computational stereo. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(8) :993–1008.

[Champagnat et al., 2011] Champagnat, F., Plyer, A., Le Besnerais, G., Leclaire, B., Davoust, S., and Le Sant, Y. (2011). Fast and accurate piv computation using highly parallel iterative correlation maximization. *Experiments in Fluids*, 50 :1169–1182. 10.1007/s00348-011-1054-x.

[Le Besnerais and Champagnat, 2005] Le Besnerais, G. and Champagnat, F. (2005). Dense optical flow by iterative local window registration. In *IEEE International Conference on Image Processing 2005*, pages I–137. IEEE.

[Létienne, 2010] Létienne, A. (2010). *Super-résolution : Développement d'algorithmes rapides et évaluation de performance*. PhD thesis, Université Paris XIII – Institut Galilée, Villetaneuse, France.

[Rochefort, 2005] Rochefort, G. (2005). *Amélioration de la résolution de séquences d'images : application aux capteurs aéroportés*. PhD thesis, Université Paris-Sud, Centre d'Orsay.

[Sun et al., 2010] Sun, D., Roth, S., and Black, M. J. (2010). Secrets of optical flow estimation and their principles. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, pages 2432–2439.

[Sutter, 2005] Sutter, H. (2005). The free lunch is over : A fundamental turn toward concurrency in software. *Dr. Dobb's Journal*, 30(3) :202–210.

2 Architectures Massivement Parallèles

Dans ce chapitre nous introduisons les concepts liés aux architectures de calcul massivement parallèle. Nous débutons par une présentation historique de l'apparition des architectures massivement parallèles, puis nous présentons les modèles de calculs associés à ces architectures.

Dans la seconde partie de ce chapitre, nous présentons les outils sur lesquels nous nous appuyons pour développer les algorithmes des autres chapitres de ce manuscrit. Nous détaillons ainsi un ensemble de pattern (ou schémas) algorithmiques. Ces patterns décrivent de manière synthétique les principaux opérateurs que nous avons utilisés dans nos travaux en vision par ordinateur. Nous présentons aussi brièvement une méthode pour représenter de manière synthétique la structure algorithmique d'une fonction massivement parallèle.

2.1 Apparition et motivation du parallélisme

2.1.1 Bref historique des architectures parallèles

L'historique qui suit décrit l'arrivée des architectures massivement parallèles sur le marché grand public à partir du milieu des années 2000. Notons que les machines parallèles existent depuis bien plus longtemps que cela : dans les années 80 il y avait l'exemple fameux de la "*connexion machine*" qui était un gros ordinateur composé de plusieurs processeurs, coûtant jusqu'à un million de dollars. Dans ce qui suit, nous nous intéressons à l'apparition de cartes additionnelles s'installant dans un ordinateur standard (voir figure 2.5) et d'un prix de seulement quelques centaines de dollars [1] : les GPUs.

Acte I : "la fin d'une progression sans fin" Ces dernières années ont été le témoin d'une transformation majeure de l'industrie des processeurs et plus généralement de l'informatique (qui elle-même reste une industrie très jeune, que ce soit dans les aspects *hardware* et *software*).

1. La question du prix étant importante car c'est grâce à ce prix grand public qu'une large diffusion des GPU est possible.

Cette industrie est basée depuis son origine sur une constante progression des performances des processeurs. Cette progression s'est heurtée en 2004 à une première barrière physique : la limite de la dissipation thermique des puces en silicium, avec pour conséquence l'impossibilité de refroidir suffisamment le processeur. A cette occasion, le géant de cette industrie Intel s'est rendu compte qu'il ne pouvait continuer à augmenter la fréquence de ses processeurs pour améliorer leur puissance de calcul. Cette prise de conscience est intervenue lors de la conception du processeur Pentium 4 qui devait selon les *roadmap* d'Intel être cadencé à $4Ghz$, mais qui n'a jamais dépassé $3.8Ghz$ en refroidissement par air (qui est actuellement le seul type de refroidissement suffisamment peu contraignant pour une utilisation grand public).

Le Pentium 4 clôt donc en 2004 la course effrénée au gigahertz que menaient les différents constructeurs de processeurs depuis les années 1980. Le marché des microprocesseurs restant basé sur l'augmentation de la puissance de calcul d'une génération à une autre, il fallait trouver une solution. Cette solution est celle de l'utilisation d'architectures de processeurs parallèles, c'est-à-dire de processeurs dans lesquels on intègre plusieurs cœurs de calculs. C'est l'apparition des processeurs multi-cœurs (ou *multi-core* en anglais).

Pour bien comprendre l'intérêt du passage de l'architecture mono-coeur à multi-cœurs, reprenons l'étude de la référence [Chandrakasan et al., 1995]. Dans cette étude, la puissance P (qui est en relation directe avec la chaleur à dissiper) nécessaire à un processeur pour fonctionner est écrite :

$$P = cV^2 f, \tag{2.1}$$

en fonction de la fréquence f de fonctionnement, de la tension d'alimentation V et de la capacitance c. Ces grandeurs sont bien évidement liées les unes aux autres. Ainsi l'augmentation de la fréquence de fonctionnement du processeur entraîne la nécessité d'augmenter sa tension pour fonctionner correctement. Si on reprend l'analyse de [Chandrakasan et al., 1995] lorsqu'on passe d'une architecture mono-cœur fonctionnant à une fréquence f, à une architecture bi-cœur fonctionnant à une fréquence $\frac{f}{2}$, la capacitance passe à $2.2c$ et la tension électrique nécessaire passe à $0.6V$. Ainsi pour disposer du même nombre d'instructions par seconde, on passe d'une consommation électrique de P pour le processeur mono-cœur à une consommation de $0.396P$ pour le processeur bi-cœur. De cette simple analyse découle l'argument principal en faveur des architectures parallèles : atteindre une puissance de calcul supérieure à celle d'une architecture mono-cœur à consommation électrique (et donc à dissipation thermique) égale.

Cette évolution de l'architecture des processeurs n'est cependant pas sans poser des problèmes. Le plus gros problème, identifié dans le rapport d'un groupe de réflexion sur les perspectives du *computing* [Fuller and Millett, 2011], provient de la difficulté d'intégrer le paral-

lélisme dans les réflexions des développeurs et algorithmiciens. En effet, lors de la période de la course au GHz, ceux-ci avaient été habitués à bénéficier "gratuitement" de l'augmentation de la performance des produits disponibles. Autrement dit, jusqu'en 2004, il suffisait de changer le processeur d'un ordinateur pour une version plus récente pour profiter immédiatement d'un gain de performance. Comme le souligne l'article [Sutter, 2005], le passage à des architectures parallèles a changé les règles du jeu pour les développeurs.

Il s'agit selon nous d'une première rupture dans l'industrie informatique : le passage d'une période où l'augmentation de la performance des processeurs et le développement algorithmique des logiciels étaient indépendants, à une période où, pour pouvoir profiter des progrès apportés par l'évolution des processeurs, il est nécessaire de reprendre le développement et de rechercher les solutions algorithmes adaptées.

Acte II : "L'émergence d'un outsider"

Scène 1 : "Un climat morose" Bien que la nécessité de l'architecture parallèle était admise en 2004, on n'a pas observé le démarrage d'une course au nombre de cœurs entre les constructeurs, comme cela avait été le cas pour la course au gigahertz. Durant la période de 2004 à 2011, les processeurs sont passés d'en moyenne d'un cœur à quatre cœurs. Cette morosité est la résultante de plusieurs facteurs.

Tout d'abord comme cela avait été souligné dans [Fuller and Millett, 2011] le parallélisme est pour beaucoup d'acteurs du développement informatique un nouveau concept difficile à assimiler et à mettre en pratique.

Ensuite, la conception d'architecture multi-cœurs en utilisant comme cœur de base des processeurs "classiques" sur le modèle des CPU pose de réels problèmes d'architecture. Ceux-ci ont été révélés par exemple dans le projet Terascale de intel qui cherchait à mettre plusieurs dizaines de cœurs de processeurs de type Pentium en connections autour d'un unique bus de communication. Ce projet, après plusieurs rebondissements est toujours un projet interne chez Intel (actuellement sous le nom de Knigth Corner ou Knight Ferry) à l'heure de la rédaction de ces lignes (été 2012).

D'autres constructeurs ont subi des échecs dans la découverte de ce nouveau monde du parallélisme. On peut citer par exemple l'architecture PowerPc Cell de IBM Toshiba et Sony. Les Cell (les processeurs des Playstation 3) sont basés sur une architecture parallèle hétérogène. Celle-ci est composée d'un cœur principal complexe ayant un rôle de chef d'orchestre distribuant les traitements à huit autres cœurs simplifiés. Alors que l'idée des concepteurs du Cell était de simplifier l'architecture du processeur, il s'est avéré à l'usage que ce processeur est un réel enfer pour les développeurs logiciels.

En 2007, l'arrivée d'un nouvel acteur dans le monde du *computing* va petit à petit relancer la course. C'est par l'introduction sur le marché de son architecture unifiée pour processeurs graphiques que nVidia va mettre en route une nouvelle révolution, celle des architectures massivement parallèles.

Scène 2 : "Flash-back : sur l'origine des GPU moderne" Les GPU (*Graphical Processor Unit*) ont suivi leurs propres trajectoires de développement indépendamment de celles des CPU. Au début des années 1990, les GPU modernes n'existaient pas, à la place on avait des processeurs dédiés à l'affichage purement 2D. Jusqu'en 2000, le marché des cartes graphiques était très varié, les processeurs intégraient généralement des solutions d'affichage 3D basées sur des API[2] propriétaires et supportant peu ou pas du tout le standard OpenGL de programmation pour le rendu 3D — le consortium OpenGL s'était formé en 1992.

A la fin des années 90s les constructeurs adoptent le standard de programmation OpenGL mais les GPU restent encore des processeurs non programmables implémentant un *pipeline* de rendu graphique fixe. Le CPU envoie simplement les données au GPU et celui-ci fait le rendu. A cette période, le marché du jeu vidéo commence à croître et à exiger de la puissance de rendu graphique. L'industrie du jeu vidéo est le moteur qui va pousser les GPU à évoluer — pour mémoire, le marché du jeu devient plus gros que l'industrie du cinéma à partir de 2004. Pour répondre aux besoins sans cesse renouvelés du jeu vidéo, par exemple pour pouvoir effectuer des rendus de plus en plus réalistes de phénomènes physiques de plus en plus complexes et dynamiques (ombres, reflets, la simulation de fumée, ou encore flou de bougé qui sont une des marques caractéristiques des jeux vidéos modernes) il est vite apparu la nécessité de pouvoir réellement programmer le pipeline de rendu graphique inclus dans le GPU. Cette idée de rendre programmable ce pipeline vient en fait de l'industrie du cinéma, et en particulier de deux acteurs : Pixar et LucasFilm. Le premier a créé un moteur de rendu programmable appelé RenderMan pour les effets spéciaux et les films d'animation, RenderMan a en particulier été utilisé pour le film "Toy's Story". RenderMan peut être vu comme la préhistoire du calcul sur GPU (qui s'appellera bientôt GPGPU pour General Programming on GPU), il sera une source d'inspiration pour nVidia et Microsoft pour créer en 2001 les premiers langages permettant la programmation de moteurs de rendu programmable sur des GPU, langages dénommés respectivement Cg et HLSL. Ces langages de programmation, même s'ils sont principalement destinés au rendu graphique, commencent à pousser quelques pionniers du GPGPU ayant flairé l'intérêt de la puissance potentielle des processeurs graphiques dans des utilisations détournées [Mairal et al., 2005, Dixit et al., 2005, Sinha et al., 2006].

2. interface de programmation

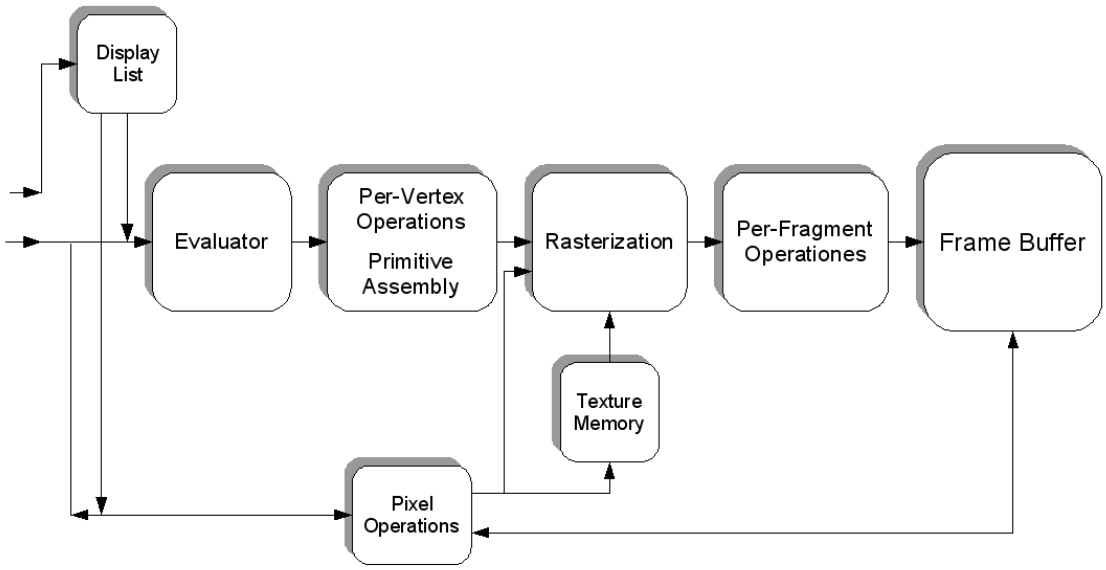


Figure 2.1 – Le pipeline de rendu graphique OpenGL. (from Wikipedia `http://en.wikipedia.org/wiki/OpenGL`)

Scène 3 : "La naissance d'un nouveau paradigme" La révolution arrive en 2007 quand nVidia crée CUDA (*Compute Unified Device Architecture*), une architecture de calcul unifiée. Le terme d'unification désigne l'idée de construire l'architecture des GPU autour d'un ensemble d'unité de calculs généralistes, a contrario d'unités spécialisées pour chacune des étapes du pipeline de rendu graphique (voir figure 2.1) comme c'était le cas jusque-là. Pour simplifier, on peut se représenter le pipeline de rendu graphique comme la succession de deux types de traitements (voir la Figure 2.1) :

- des traitements sur des maillages (Vertex operation),
- des traitements sur des textures 2D (Fragment operation).

Dans une architecture non unifiée, il peut avoir un déséquilibre entre l'utilisation des processeurs spécialisés maillages et les processeurs spécialisés texture. De manière à éviter cela et permettre une plus grande efficacité de l'architecture des GPU, nVidia proposa d'unifier ces deux types d'unités de calcul dans un unique petit processeur versatile.

Acte III : "La révolution des architectures massivement parallèles" Tout en proposant ces nouvelles architectures unifiées, nVidia met à disposition des développeurs une extension du langage C permettant d'exploiter avec une relative simplicité ses GPU pour effectuer des calculs généralistes. L'apport des GPU dans le domaine du calcul massif est depuis devenu indiscutable, la principale raison du succès résidant dans l'apparition d'un modèle de calcul

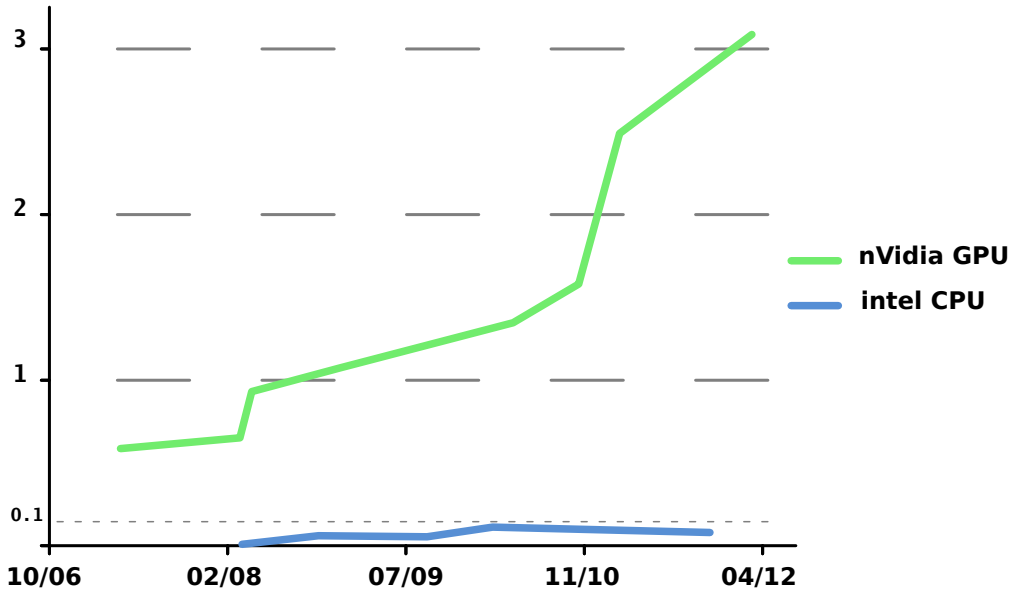


Figure 2.2 – Evolution de la puissance-crête (en téraFlops) des GPU (en vert) et des CPU (en bleu) ces dernières années. L'étoile bleue correspond au processeur Intel Xeon E7 8870, nous l'avons mis en hors catégorie car avec son coût de plus de 4000$ il ne correspond pas vraiment à un processeur grand public.

clair et versatile, modèle que nous détaillerons dans la section 2.2. Ce modèle de calcul est standardisé à l'été 2008 et donne naissance à l'API de programmation OpenCL, marquant ainsi le début de l'ère des architectures massivement parallèles grand public programmables. Parmi les domaines qui ont bénéficié de cette standardisation on peut citer le calcul sur processeur embarqué, tel qu'il se développe actuellement pour les besoins des smartphones.

A ce stade, on peut se demander où se situe la révolution annoncée dans le titre. Si on observe sur la figure 2.2 l'évolution de la puissance de calcul en teraflops (10^{12} opérations sur des nombres en virgule flottante par seconde) pour les CPU et les GPU ces dernières années, on constate que les GPU dominent les CPU de plusieurs ordres de grandeur. Il en est de même pour un autre indice très important : les gigaflops par watt consommé, représentés sur la figure 2.3. De nouveau les GPU offrent des performances bien plus intéressantes que les CPUs, ce qui est essentiel dans le domaine des calculs massifs, où les clusters de calcul peuvent consommer plusieurs dizaines de mégawatts [3], et aussi dans les applications embarquées où la ressource en énergie est très limitée.

3. A tel point que le coût principal d'un cluster tend à être sa consommation électrique

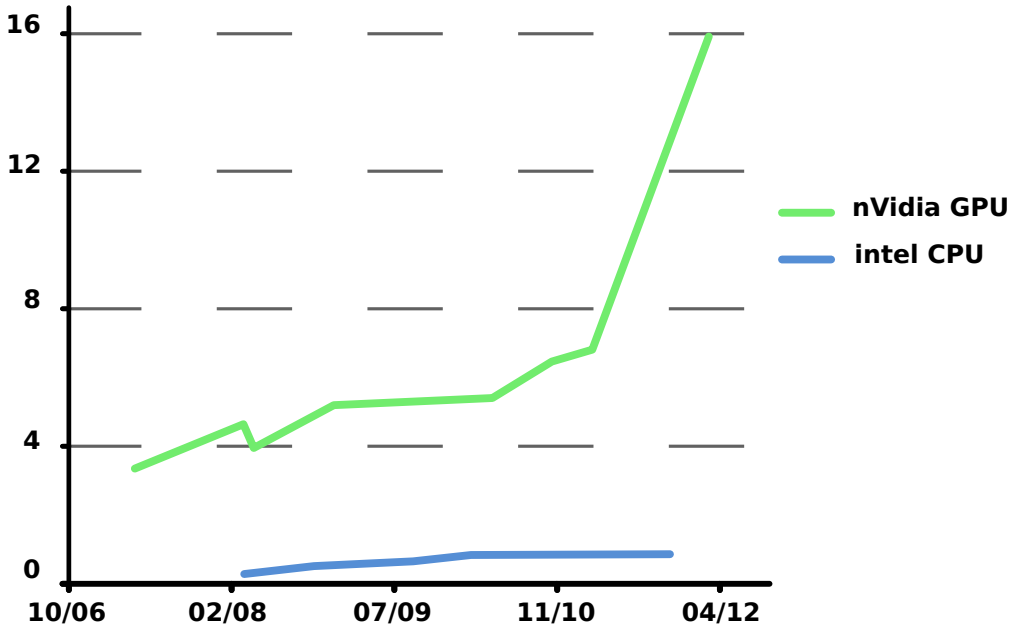


Figure 2.3 – Evolution de la puissance-crête par watt (en gigaFlops/Watt) des GPU (en vert) et des CPU (en bleu) ces dernières années. L'étoile bleue représente le processeur Intel Xeon E7 8870 — nous l'avons mis en hors catégorie car il ne correspond pas à un processeur grand public avec son coût de plus de 4000$.

2.1.2 CPU/GPU : pourquoi des performances aussi différentes ?

Deux types de parallélismes La réponse à cette question n'est pas simple, principalement parce que chacune des architectures (CPU et GPU) est efficace pour un ensemble d'algorithmes différents.

D'un côté, l'architecture multi-cœur des CPU est composée d'unités d'exécution complexes efficaces dans l'exécution de programmes génériques impliquant un grand nombre d'instructions conditionnelles et de sauts (ie. instructions type *if*). Pour ce faire, le CPU dispose d'algorithmes de *scheduling* (ordonnancement) complexes lui permettant d'exécuter des flux de traitements inhomogènes et de les répartir entre ses unités de calcul. De par cette structure, le CPU est adapté à la mise en œuvre efficace de fils d'exécution complexes et inhomogènes, selon un paradigme de parallélisme *orienté tâches* schématisé en figure 2.4).

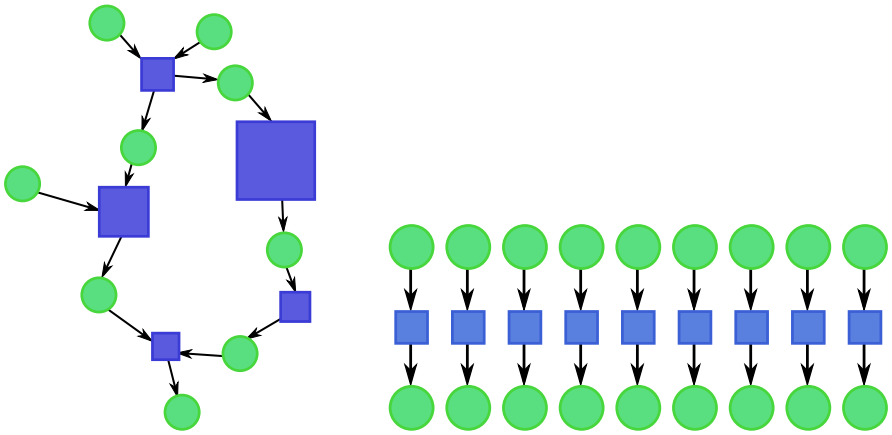


Figure 2.4 – Schémas de principe du parallélisme orienté tâche (gauche) et du parallélisme orienté données (droite). Les ronds verts symbolisent des éléments de stockage en mémoire, les carrés bleus des éléments de calculs, et les flèches indiquent le sens de parcours des données (écriture ou lecture en mémoire) : nous utiliserons ces symboles dans toutes les figures de la suite de ce chapitre.

D'un autre côté, l'architecture massivement parallèle des GPU est composée d'une multitude de petites unités de calculs beaucoup plus simples et d'un ordonnanceur adapté pour effectuer des traitements similaires sur un grand nombre de données. Cette structure est très bien adaptée au parallélisme orienté données, schématisé en figure 2.4, c'est-à-dire à un parallélisme très homogène, dans lequel des traitements simples sont *mappés* sur (appliqués à) une grosse

masse de données. C'est ce type parallélisme qui est prédominant en traitement d'image bas niveau par exemple, où les données sont de grande taille (les images sont aujourd'hui au minimum en mégapixel) et où, pour chaque pixel, les traitements à effectuer sont similaires.

Notons que pour pouvoir alimenter cette multitude de petits processeurs en données, les GPU possèdent une bande passante vers leurs mémoires bien plus importante que les CPU. Actuellement un GPU a une bande passante de l'ordre de 200 gigabytes par seconde vers sa mémoire alors qu'un CPU ne dispose que d'une bande passante de l'ordre de 20 gigabytes par seconde.

Conclusion Comme on le voit, ces deux architectures ne sont pas vraiment en concurrence du fait qu'à chacune d'elles est associé un type de parallélisme différent. Néanmoins, lorsqu'on s'intéresse au domaine du traitement vidéo, la grande majorité des fonctions bas niveau possèdent un parallélisme massif. Dans ce cas, le GPU est une plateforme de choix, offrant de meilleures performances brutes (Flops) et une meilleure efficacité (Flops par Watt) que des processeurs classiques (ou CPU).

Nous allons montrer dans la suite de ce manuscrit l'importance d'avoir un doublet *algorithme/architecture* cohérent, et ce principalement dans notre domaine qu'est la vision artificielle. On peut noter aussi que les architectures GPU ou CPU ne sont pas les seules solution envisageables à l'heure actuelle. Par exemple on peut utiliser des solutions plus intégrées comme des soft-processeurs dédiées au traitement d'images sur FPGA. Être capable d'obtenir des métriques afin de mesurer l'adéquation algorithme / architecture est devenu un sujet de recherche important, on peut par exemple citer le projet COTS du CEA qui cherche à obtenir une méthode de mesure de cette adéquation par une analyse empirique d'une grande base d'algorithmes implémentés pour différentes architectures.

2.2 Le modèle de calcul

Nous allons présenter dans cette section le modèle de calcul qui est utilisé par OpenCL ou Cuda pour permettre la programmation des GPU. C'est l'abstraction que doit assimiler le développeur qui veut profiter pleinement des architectures massivement parallèles. Nous terminerons cette section en pointant la relation entre l'architecture physique du GPU et le modèle de calcul, mais il est important de noter que seul le modèle de calcul est nécessaire pour comprendre l'impact sur l'algorithmie. Comprendre le lien entre le modèle de calcul et l'architecture physique n'est utile que pour mieux comprendre les limites actuelles du modèle et les évolutions possibles de celui-ci.

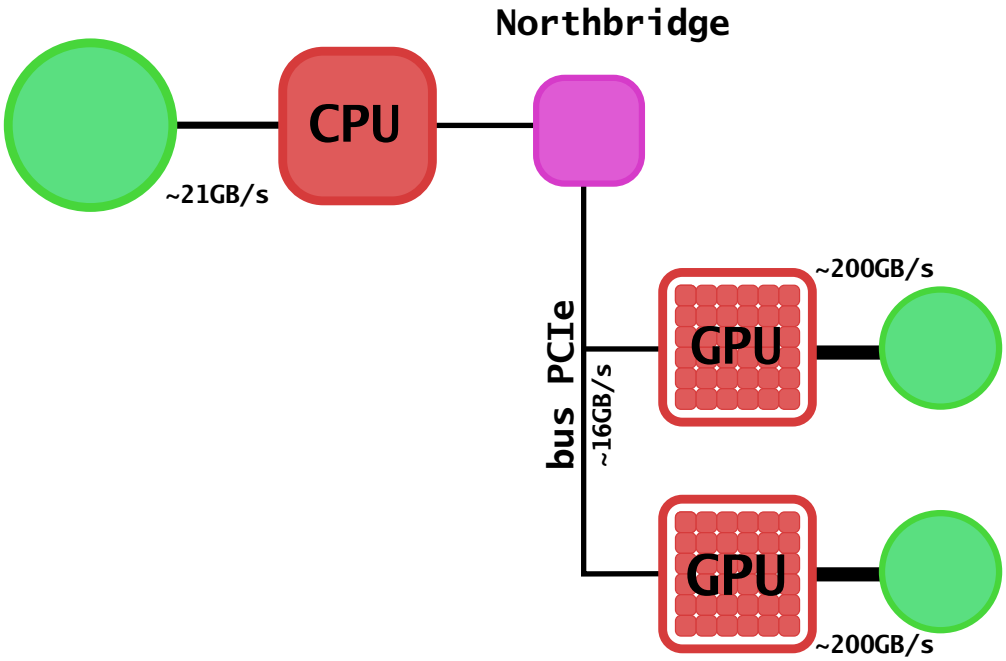


Figure 2.5 – Structure générale de l'architecture d'un PC moderne

Comme on le voit sur la figure 2.5, les GPU sont des processeurs possédant leurs propres mémoires et connectés au CPU central du PC via un bus standard : le bus PCIexpress. Ceci a plusieurs implications.

Tout d'abord, cela implique que le développeur doit gérer la localisation de ses données (une image par exemple) entre la mémoire du GPU ou celle du CPU, et au besoin effectuer des copies entre les deux mémoires. La bande passante du bus PCIexpress étant très limitée par rapport à celles qu'ont les processeurs (CPU ou GPU) vers leurs mémoires propres, il est préférable de limiter les échanges de données entre la mémoire GPU et la mémoire CPU. Le plus efficace est de chercher à maximiser la quantité de fonctions qui s'exécutent sur le GPU tout au long d'un algorithme, idéalement dans nos travaux, on ne copie vers le GPU que les données sources et on ne récupère que le résultat final — si besoin est. Il peut arriver en effet qu'on ne récupère même pas le résultat s'il suffit de l'afficher directement pour une visualisation, ce qui est la fonction d'origine d'un GPU.

La seconde implication est l'apparition d'un nouveau type de fonctions : les fonctions qui s'exécutent sur le GPU. Une telle fonction s'appelle un *kernel*, elle décrit les traitements effectués par une unité élémentaire de calcul. Les kernels sont associés à une grille d'exécution qui définit la structure du calcul. On peut voir sur la figure 2.6 un schéma représentant l'exécution d'un algorithme sur le GPU. Le kernel et sa grille d'exécution correspondent à la première partie du modèle de calcul, la seconde étant l'architecture de la mémoire. Nous allons étudier plus en

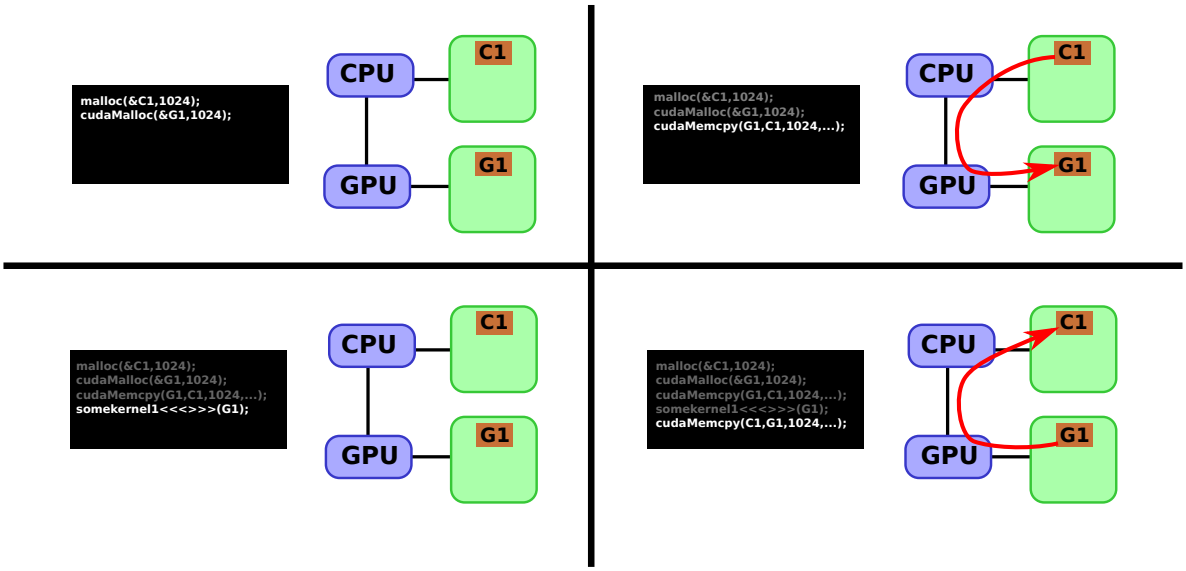


Figure 2.6 – Déroulement d'un traitement effectué par le GPU de haut en bas et de gauche à droite. On commence par allouer de la mémoire sur le GPU (G1) et le CPU (C1), on copie ensuite les données vers le GPU, on exécute la fonction sur le GPU, puis au besoin on rapatrie sur le CPU le résultat.

détail ces deux éléments.

2.2.1 Le modèle d'exécution

Le modèle d'exécution est construit autour d'un élément d'exécution atomique appelé *thread*. L'ensemble des threads est structuré en une grille à deux niveaux de granularité pour former ce que nous avons appelé une grille d'exécution, comme nous pouvons le voir sur la figure 2.7.

Une grille d'exécution est constituée d'une matrice multi-dimensionnelle (1D, 2D ou 3D, elle est 2D sur la figure 2.7) de groupes de threads $\mathbf{G}(i,j)$. Chacun de ces groupes ou blocs (*block* en anglais) est indicé par sa position sur la grille. Un bloc est composé d'une matrice multi-dimensionnelle de threads $\mathbf{T}(k,l)$ eux même indicés par leurs positions dans le bloc. Il est bon de noter que tous les blocs ont des dimensions (en nombre de threads) identiques.

Chacun des threads de la grille exécutent la même fonction, le seul élément qui les différencie est leur indice de position sur la grille de calcul, qui est donc double :

- indice du block dans lequel le thread se trouve (blockIdx en Cuda),
- indice du thread dans le block (threadIdx dans Cuda).

Ainsi, l'élément clef de la programmation massivement parallèle réside dans l'association, ou

mapping en anglais, entre les données et la grille de calcul.

Un petit exemple permet de mieux comprendre comment tout cela fonctionne. Supposons que l'on dispose d'une matrice carrée A de 1024 éléments de côté, on souhaite exécuter une fonction **kerneltoto** ayant pour objectif d'ajouter 1 ou de soustraire 1 selon un schéma en damier sur la matrice. On décide alors que la grille de calcul est composée au total du même nombre de threads que d'éléments dans la matrice. On va choisir ici plus ou moins arbitrairement d'utiliser des blocs 1D de 256 threads. Ainsi la dimension de notre grille de blocs est de 1024 par 4. Le code CPU allouant une matrice sur le GPU, copiant la matrice A dessus, exécutant le kernel GPU et récupérant le résultat s'écrit :

```
float *d_A;
cudaMalloc(&d_A,1024*1024*sizeof(float));
cudaMemcpy(d_A,A,1024*1024*sizeof(float));
dim3 grid(1024,4);
dim3 bloc(256);
kerneltoto<<<grid,bloc>>>(d_A);
cudaMemcpy(A,d_A,1024*1024*sizeof(float);
```

Le code du kernel GPU est alors :

```
__global__ void kerneltoto(float *A)
{
  int x = blocIdx.x*blocDim.x+threadIdx.x;
int y = blocIdx.y*blocDim.y+threadIdx.y;
int idx = x + y*1024;

float val=A[idx];

if((x+y)%2 == 0)
    val--
  else
    val++

  A[idx]=val;
}
```

Dans ce kernel, on ne décrit que le travail que doit exécuter un thread de la grille. C'est par

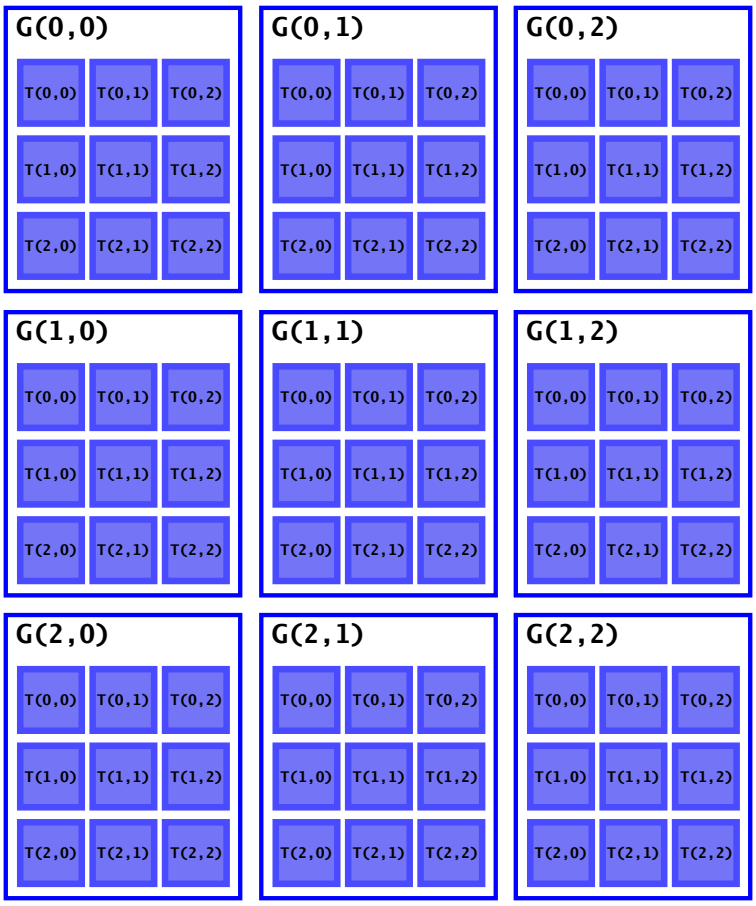


Figure 2.7 – Exemple d'une grille de calcul 2D de dimension 3 sur 3 blocs eux-mêmes composés d'une grille de threads 2D de dimension 3 sur 3.

l'association de ce kernel et du mapping des données sur la grille que l'on obtient la fonction globale voulue.

On voit bien ici que l'aspect fondamental de ce modèle de calcul est la structure d'accès aux données qui est spécifiée dans le thread. Nous présentons donc dans la section 2.3 un ensemble de schémas (ou *pattern*) définissant les principales structures d'accès que nous avons rencontrées tout au long de nos travaux.

On peut se demander la raison du découpage à deux niveaux d'exécution (blocs de threads puis threads). La raison de ce découpage réside dans le fait que les threads d'un même bloc partagent certaines ressources. Celles-ci sont essentiellement :

– des outils permettant la synchronisation de l'exécution ;
– un espace mémoire permettant l'échange de données.

En effet, il est très complexe (voire impossible) de construire une architecture matérielle où

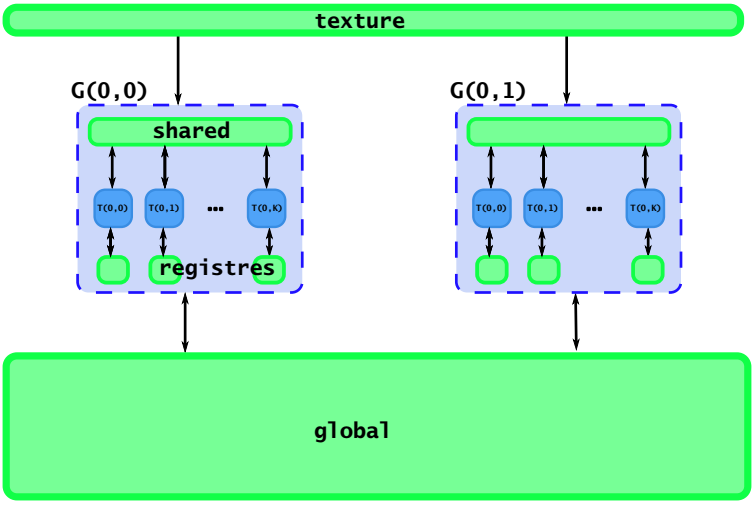


Figure 2.8 – Schéma du modèle mémoire

des ressources de ce type sont partagées entre un très grand nombre d'unités d'exécution. Le bloc représente donc la bonne échelle (en termes de nombre d'unités de calcul) pour mettre à disposition ces ressources. Au-delà, ces ressources n'existent pas, autrement dit il est impossible dans ce modèle de faire communiquer directement des threads qui appartiennent à différents blocs d'une grille d'exécution. Il n'y a de plus aucun contrôle sur l'ordre dans lequel les différents blocs de la grille sont exécutés.

2.2.2 Le modèle mémoire

Nous venons de voir la structure du modèle d'exécution d'un GPU. Nous allons maintenant étudier la hiérarchie mémoire de ce modèle. En effet, alors que sur un CPU les différents espaces mémoires (L1, L2, Ram) sont gérés automatiquement par des systèmes complexes de pagination, la gestion du transfert des données entre les différents espaces mémoire du GPU fait partie du travail laissé au programmeur.

Le plus vaste de ces espaces mémoires est la mémoire RAM propre au GPU (de l'ordre du gigaoctet), cette mémoire, dite mémoire globale, est persistante (à l'échelle du processus) et peut être copiée à partir de fonction CPU (cudaMemcpy) mais on ne peut pas accéder à ses éléments à partir d'une fonction CPU, cela est réservé aux kernels GPU.

La figure 2.8 présente la hiérarchie des différents espaces mémoires accessibles dans une grille de calcul. Un thread possède tout d'abord un espace mémoire qui lui est propre appelé registre. Toutes les variables définies dans un kernel (qui se trouveraient allouées sur la pile en C) se trouvent par défaut dans cet espace — c'est le cas par exemple des variables x,y et idx

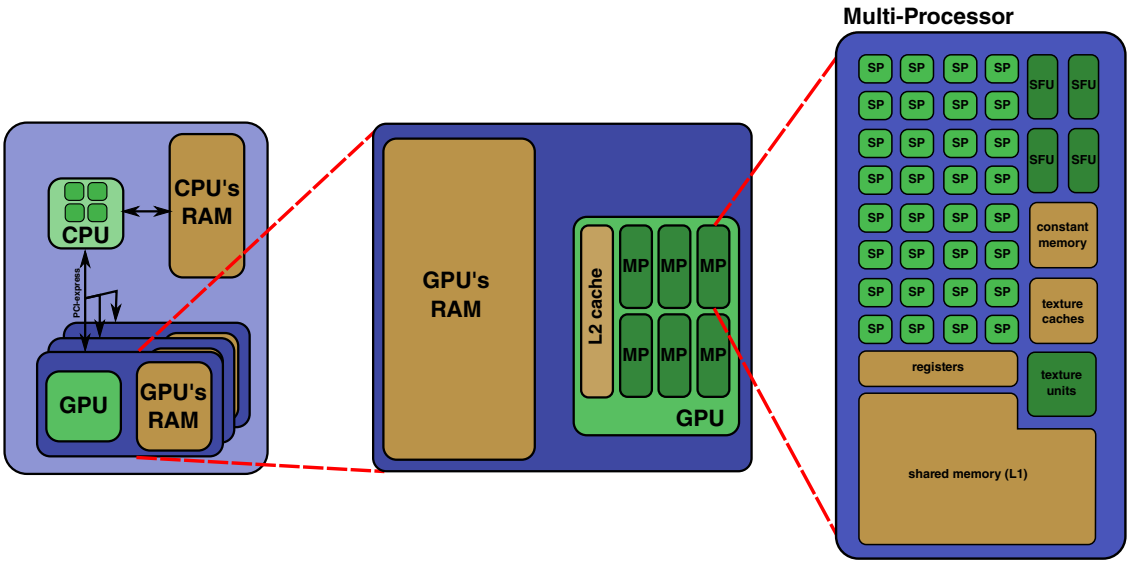


Figure 2.9 – Schéma de la structure physique d'un GPU Nvidia Fermi.

de l'exemple présenté dans la section précédente.

Ensuite vient la mémoire partagée ou *shared memory*, un espace mémoire partagé entre tous les threads d'un même bloc. La *shared memory* peut servir de mémoire cache car les accès à cette mémoire sont beaucoup plus rapides que les accès à la mémoire globale. Cela permet par exemple dans le pattern FIR présenté dans la section 2.3.2 que chaque thread d'un bloc puisse aller lire des éléments mémoires chargés par les autres threads du bloc.

Pour finir, la dernière mémoire utilisable est la mémoire de texture, celle-ci est très intéressante car elle constitue un moyen rapide et efficace d'effectuer une interpolation linéaire ou au plus proche voisin. Une texture est un tableau dans lequel on peut accéder via des indices de position flottants, l'interpolation étant directement implémentée via des unités spécifiques du GPU, ce qui la rend extrêmement rapide.

2.2.3 Structure physique et modèle de calcul

Bien que le modèle de calcul soit intimement lié à une structure physique du GPU et aux contraintes qui lui sont associées, il est important de ne pas confondre ces deux aspects.

Le modèle de calcul des GPU est une abstraction de la structure physique destinée à faciliter le travail du développeur, construite de manière à pouvoir dépasser la durée de vie d'une architecture de calcul. Par exemple durant cette thèse, quatre architectures matérielles de GPU se sont succédées chez nVidia (G90, GT200, Fermi, Kepler). Or, malgré cela, le modèle de calcul Cuda (ou OpenCL, qui sémantiquement est identique) n'a pas évolué.

On peut cependant voir que l'architecture du Fermi (figure 2.9) est très proche de la structure du modèle de calcul, avec une répartition des cœurs de calculs sur une structure à deux niveaux de granularités. Les GPU Fermi sont composés de plusieurs multi-processeurs qui sont eux-même composés d'un ensemble de cœurs de calculs. Cette architecture physique permet de mieux comprendre l'impact de certains choix de mapping sur l'exécution d'un kernel. Par exemple la taille maximum de mémoire partagée disponible dans chaque bloc où le nombre maximum de threads par blocs sont des contraintes qui sont dépendantes des différentes architectures et évoluent suivant celles-ci.

2.3 Les pattern (schémas) de calculs

Durant nos travaux de thèse, nous avons remarqué que la grande majorité (pour ne pas dire la totalité) de nos développements sur GPU s'articulaient autour de quelques schémas algorithmiques génériques. Nous allons présenter ces schémas dans les pages qui suivent. L'objectif de ces différents schémas est de mettre en valeur l'élément que nous considérons comme essentiel lors de la programmation d'architectures massivement parallèles : l'association ou *mapping* qui relie les éléments de calculs (les threads) aux données d'entrée (et éventuellement de sortie) qui sont localisées en mémoire globale.

Pour faciliter la compréhension du lecteur, nous présentons tout d'abord les schémas de calcul sur des signaux mono-dimensionnels. Nous présenterons ensuite les problématiques et spécificités de certains schémas dans le cas multi-dimensionnel en section 2.3.9.

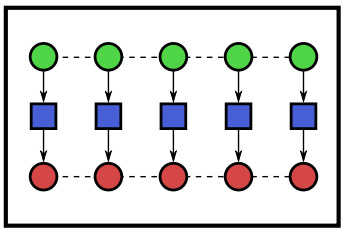


Figure 2.10 – Pattern pixel-à-pixel

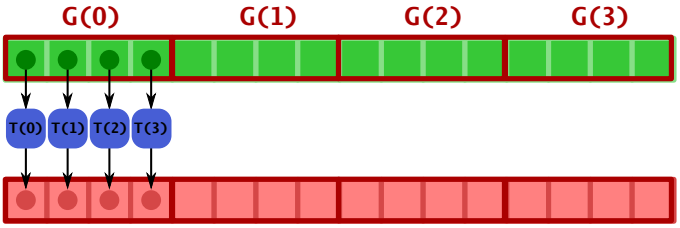


Figure 2.11 – Définition de la grille de calcul sur les données.

2.3.1 Le pattern Pixel-à-Pixel

Le schéma d'opération pixel-à-pixel est le plus simple qui existe. Il correspond à une bijection entre les éléments du signal (ou pour une image : les pixels) en mémoire et les threads de la grille de calcul. On retrouve ce schéma par exemple dans les kernels de résolution du système local du flot optique dans l'algorithme FOLKI, voir section 3.3.2.

Pour mettre en œuvre ce schéma, il suffit de construire une grille de calcul définissant une partition de l'espace de départ comme on peut le voir sur la figure 2.11.

Le traitement effectué par un thread est assez simple et se décompose en trois étapes :

1. récupération des données (dont, en général, l'intensité du pixel qui lui est associé) en mémoire globale,
2. calcul du résultat
3. écriture du résultat en mémoire globale à la même position qu'en entrée.

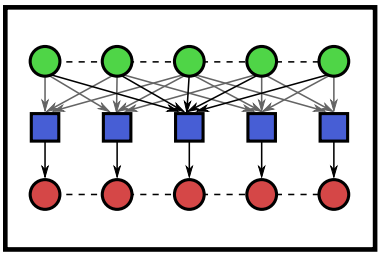


Figure 2.12 – Schéma de principe d'un filtre FIR

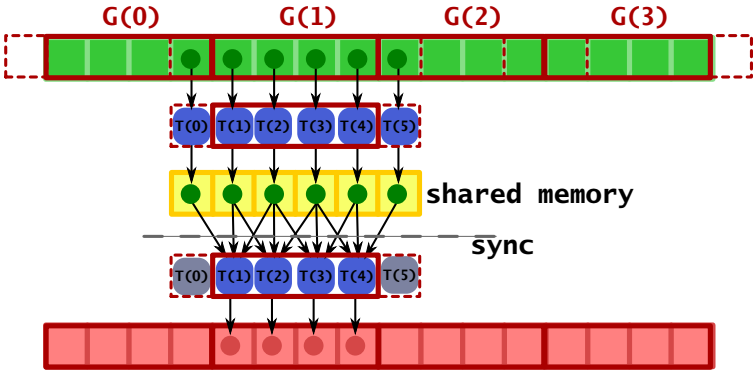


Figure 2.13 – Détail du pattern FIR, les threads d'un bloc se divisent en deux catégories, suivant qu'il correspondent à des pixels dont la valeur est modifiée par le filtrage ou à leurs voisins.

2.3.2 Le pattern FIR

Les opérations de type filtrage FIR (*Finite Impulsional Response* ou convolution par un noyau à support fini) sont largement utilisées en traitement du signal et des images. Le schéma de calcul de ce type d'opération est caractérisé par le fait que pour calculer la valeur d'un pixel du résultat, on a besoin de sommer la valeur des pixels autour de sa position. On a donc une relation de voisinage local, qu'il faut traiter au niveau de la mémoire partagée pour être efficace.

Étant donné que la mémoire partagée n'est accessible que par les threads d'un même bloc, chaque bloc va devoir charger en mémoire les pixels traités (ceux dont la valeur sera remplacée par le résultat du filtrage) et aussi les pixels correspondant aux bords du bloc. Pour ce faire, on effectue un tuilage des données avec des tuiles ayant un recouvrement correspondant à la demi-taille du noyau FIR. Ainsi, comme on peut le voir sur la figure 2.13, la zone correspondant au bord étant représentée en pointillé, l'ensemble des threads du bloc se divise en deux catégories. Les premiers ne sont actifs que lors de la copie des données de la mémoire globale vers la mémoire partagée locale, ce sont ceux situés aux bords recouvrant de la tuile ($T(0)$ et $T(5)$ sur la figure). Les threads de la seconde catégorie, une fois la copie de leurs données effectuée, calculent le résultat du filtrage au pixel correspondant à leur position globale en utilisant les données nécessaires, présentes dans la mémoire partagée. Pour s'assurer que les données en mémoire partagée sont valides, une synchronisation de tout le bloc est effectuée avant le calcul du résultat. Pour finir, ces threads écrivent le résultat dans la mémoire globale à leur position.

Les exemples de traitements utilisant le pattern FIR sont très nombreux, on peut citer par

exemple :

- la convolution séparable ;
- le calcul de gradients spatiaux sur une image ;
- les opérations morphologiques avec un élément structurant séparable ;
- le calcul de minimum et maximum locaux (par exemple utile dans l'extraction de points de Harris) ;
- la convolution non-séparable et le filtre bilatéral.

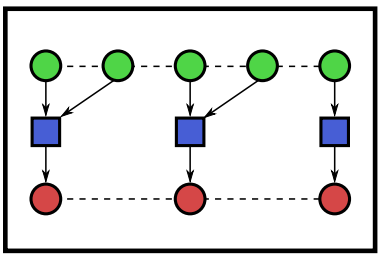


Figure 2.14 – Pattern down-sampling

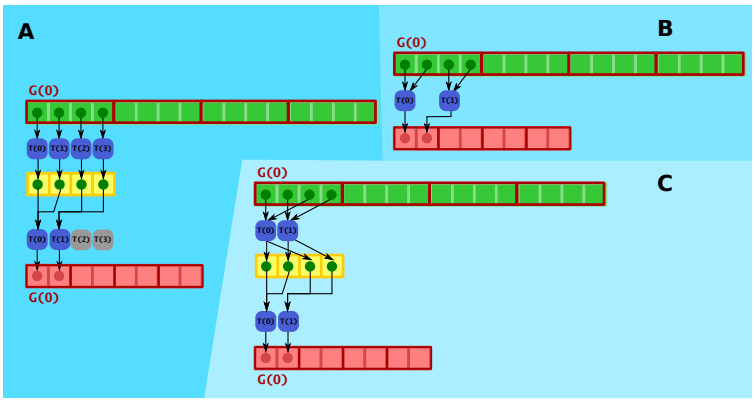


Figure 2.15 – Différentes architectures d'exécutions pour le pattern down-sampling

2.3.3 Le pattern down-sampling

Le pattern de down-sampling ou de décimation est utilisé principalement dans les pyramides d'échelles ou encore dans les transformées en ondelettes. Sa particularité est que la taille de la donnée de sortie est plus petite que celle d'entrée d'un facteur entier, souvent deux.

Il existe deux stratégies possibles pour sa mise en œuvre :

- attribuer un thread à chaque pixel de l'image d'entrée,
- attribuer un thread à chaque pixel de l'image de sortie.

Dans le premier cas, on utilisera la mémoire partagée du bloc pour partager localement les données puis la moitié des threads du bloc feront effectivement le calcul de réduction avant d'écrire le résultat en mémoire globale. Cette solution n'est pas optimale car la moitié de l'effectif des threads de la grille se retrouve inactive dans la seconde partie du kernel comme on peut le voir sur la figure 2.15 - illustration (A).

Nous préférons la seconde stratégie, où chaque thread est associé à une donnée de sortie et doit par conséquent lire deux données d'entrée, comme illustré sur la figure 2.15 - B.

On peut noter que sur certaines architectures exigeant un alignement de la lecture (ou une lecture dite *coalescente*, cf. section 2.3.9), il est plus efficace d'utiliser la mémoire partagée comme une mémoire cache afin de lire consécutivement deux segments continus de mémoire, puis d'utiliser les données en mémoire partagée pour effectuer le calcul comme illustré sur la figure 2.15 - C.

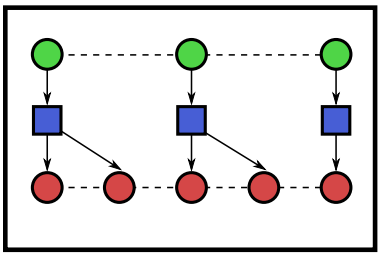


Figure 2.16 – Pattern up-sampling

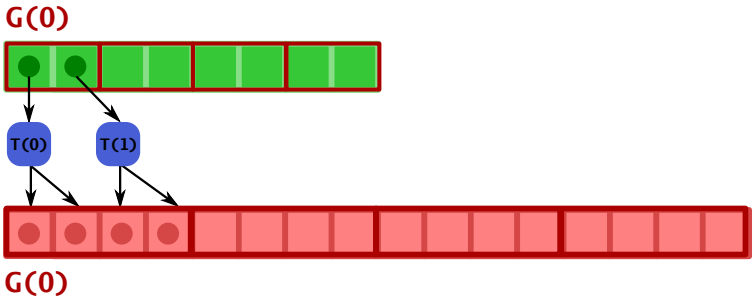


Figure 2.17 – Pattern up-sampling : définition de la grille de calcul sur les données.

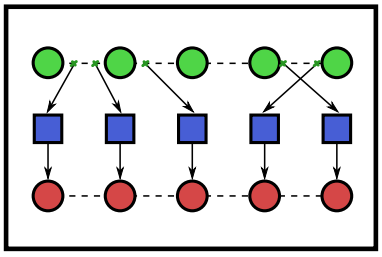


Figure 2.18 – Pattern Interpolation

2.3.4 Le pattern up-sampling

Le pattern up-sampling est l'opérateur transposé du pattern de down-sampling. Il consiste à augmenter la taille des données d'entrée en les dupliquant sur les données de sortie. Ce pattern s'exécute simplement en attribuant à chaque donnée d'entrée un thread comme on le voit sur la figure 2.17. Tout comme pour le pattern down-sampling, on peut avoir besoin d'effectuer les lectures/écritures de manière alignée : on utilise alors la stratégie vue en figure 2.15-C en exploitant la mémoire partagée.

2.3.5 Le Pattern Interpolation

En traitement du signal, et en traitement d'images pour notre cas, on a souvent besoin d'interpoler des valeurs sur une grille régulière, par exemple pour effectuer la déformation paramétrique d'une image. L'utilisation première des GPU étant le rendu graphique 3D où la déformation d'image est intensément utilisée lors de l'application des textures, les GPU sont équipés d'unités d'exécution spécifiques dédiées aux traitements d'interpolation 2D : les unités de textures. Ces unités peuvent être utilisées dans des programmes de GPGPU. Les tableaux alloués en mémoire globale deviennent accessibles en n'importe quelle coordonnée flottante

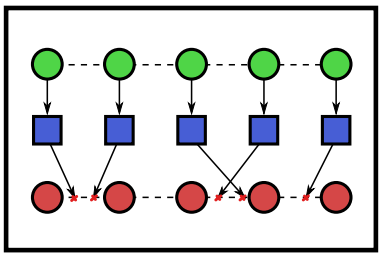


Figure 2.19 – Pattern Vote

à partir de l'intérieur d'un kernel dès lors qu'ils sont associés à une référence de texture. Cela se fait simplement par un appel d'API Cuda ou OpenCL dans le code CPU. Nous ne nous attarderons pas sur le mapping car il est ici trivial et correspond simplement à associer un thread à chaque pixel de la sortie.

Il est important de noter que les interpolations disponibles dans ces unités sont des interpolations linéaires (avec une interpolation en virgule fixe) ou des interpolations au plus proche voisin. On peut cependant noter qu'on utilise en traitement d'images une grande variété d'interpolations comme par exemple l'interpolation bicubique, ou spline cubique. Ces interpolations peuvent, elles aussi, se mettre en oeuvre de façon très efficace sur GPU, on peut citer par exemple [Champagnat and Le Sant, 2011] qui présente une interpolation spline cubique exploitant la troncature d'un filtre IIR pour obtenir un filtrage RIF plus adapté au fonctionnement d'un GPU.

2.3.6 Le pattern vote

Le pattern vote correspond au besoin d'accumuler des valeurs à des positions non déterminées dans le signal de sortie (les croix rouges dans la figure 2.19). Le principal problème est d'éviter les collisions entre threads qui voteraient sur la même case mémoire. Pour permettre cela, les GPU disposent de fonctions *atomic* qui permettent, moyennant un léger surcoût par rapport à une instruction classique, d'assurer que les écritures effectuées en parallèle n'entrent pas en concurrence.

Les procédures de vote sont très courantes en traitement d'image (transformée de Hough) et en vision (calcul d'histogrammes). Dans la suite de ce travail, un exemple de l'utilisation de ce schéma est l'opération de Shift-and-Mean qui est à la base des algorithmes de super-résolution que nous présenterons au chapitre 4.

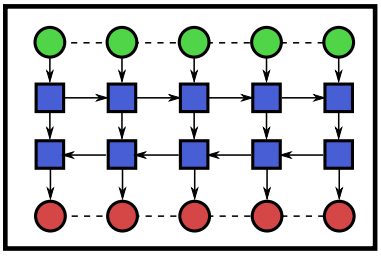


Figure 2.20 – Schéma IIR.

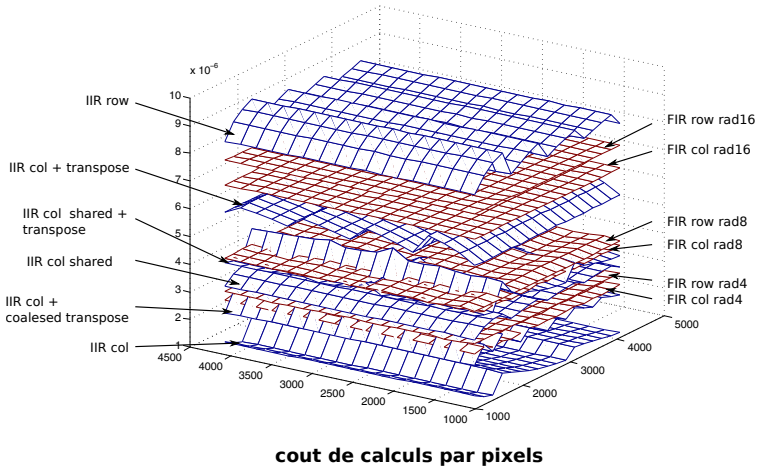


Figure 2.21 – Coût de calcul par pixel en fonction du nombre de colonnes et de lignes d'une image sur un GPU GT415 pour différents algorithmes de convolution IIR ou FIR.

2.3.7 Le pattern IIR

Le schéma de convolution IIR (*Infinite Impulse Reponse*) est un schéma qui n'est pas parallélisable en 1D. En effet, il est constitué d'une succession d'opérations non séparables. Cependant, ce filtre possède un intérêt lorsqu'il s'agit de l'appliquer à de plus grandes dimensions, et particulièrement lors de traitement sur des volumes. La principale raison est que le coût d'un filtrage gaussien écrit sous la forme IIR est indépendant de l'écart-type, ce qui permet de filtrer avec des grandes valeurs d'écart-types.

Pour pouvoir bénéficier de l'aspect parallèle de la mise en œuvre de ce schéma, il faut cependant que le ratio nombre de processeurs du GPU sur résolution des données (nombres de lignes ou nombre de colonnes traitable en parallèle) ne soit pas trop grand.

Une étude que nous avons menée montre que sur des images de 2048 pixels de côté et sur de petits GPU (typiquement un GPU de portable GT415 de 48 processeurs) il est plus avantageux d'utiliser un filtre IIR qu'un filtre FIR de rayon 8. Cependant cet avantage disparaît si on augmente la puissance du GPU en faisant le calcul sur une GTX480 par exemple qui possède 480 processeurs. On peut voir par exemple sur la figure 2.21 que l'utilisation d'une convolution de type IIR est plus rapide qu'une convolution de type FIR de rayon 16 sur un GPU GT415. Cependant, on n'observera plus cet avantage pour le schéma IIR sur un GPU plus puissant (tel qu'une GTX480). Ce phénomène s'explique facilement par le degré de parallélisme d'un schéma de calcul FIR qui est en nombre de pixels tandis que celui d'un schéma IIR n'est

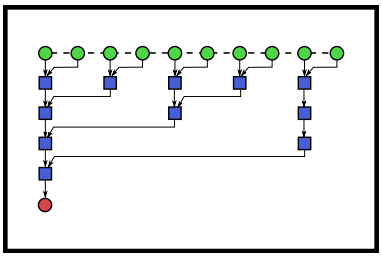


Figure 2.22 – Schémas de réduction.

qu'en nombre de lignes (ou de colonnes suivant la dimension).

2.3.8 Le pattern réduction

Une opération de réduction correspond à une opération où la quantité des données en sortie est inférieure à celle en entrée. Avec une définition aussi générale, le pattern de down-sampling est une réduction. Et en effet, en appliquant le pattern down-sampling de manière itérative sur les données on obtient le même type d'arbre logarithmique de réduction que ce que nous présentons ici.

Nous allons cependant nous intéresser ici à une situation plus extrême : le cas où on cherche à extraire un scalaire à partir d'un grand vecteur. Par exemple chercher le maximum d'une image, ou encore calculer la moyenne. Pour pouvoir mettre en œuvre une réduction il faut que l'opération $\mathbf{op}$ effectuée entre deux pixels pour les fusionner soit associative, c'est-à-dire que pour quatre pixels a, b, c, d on ait :

$$\mathbf{op}(a, b, c, d) = \mathbf{op}(\mathbf{op}(a, b), \mathbf{op}(c, d)) \tag{2.2}$$

Dès lors que l'opération de réduction est associative, on peut construire un arbre de réduction logarithmique. On connaît plusieurs stratégies d'implémentation pour ce schéma : la documentation du SDK de Cuda étudie un certain nombre de ces possibilités [4] . Sans entrer dans ces différentes possibilités, on sait que la réduction peut être implémentée en exécutant itérativement le pattern down-sampling. Il faudra donc $log_2(\frac{N}{2})$ noyaux consécutifs pour réduire un signal de longueur N.

Il est important de noter que, contrairement à tous les autres schémas que nous avons vus, le nombre d'itérations requis pour une réduction dépend de la taille du signal d'entrée.

4. www.nvidia.com/object/cuda_home_new.html

2.3.9 Généralisation : le Pattern FIR n dimensions

Nous allons présenter ici les éléments permettant de comprendre les contraintes qui interviennent lorsqu'on applique les schémas que nous venons de présenter dans un cadre multi-dimensionnel.

Pour pouvoir comprendre pourquoi un schéma mono-dimensionnel ne se généralise pas directement à un schéma multi-dimensionnel, il suffit de savoir que la mémoire d'un GPU est un espace d'adressage linéaire (ou mono-dimensionnel). Cela signifie que lorsqu'on cherche à stocker un signal multi-dimensionnel, il faut le découper en tranches linéaires qu'on stocke de manière consécutive dans la mémoire mono-dimensionnelle. Classiquement, pour les images, on stocke le tableau 2D d'une image en collant simplement bout-à-bout les lignes, cette méthode de stockage s'étend aisément à plus de deux dimensions.

La difficulté en calcul sur GPU vient du fait que ceux-ci possèdent une optimisation pour accéder rapidement aux données en mémoire globale, optimisation appelée *accès coalescent*. Lorsque les indices des threads accédant à la mémoire globale sont alignés avec la dimension d'adressage, le GPU peut multiplexer les accès mémoires de plusieurs threads et minimiser le nombre d'accès.

On a donc deux cas de figure qui se profilent, cf. Figure 2.23 :

- les accès en mémoire globale sont consécutifs (la dimension du signal est alignée avec la dimension d'adressage de la mémoire globale) ;
- les accès en mémoire globale ne sont pas consécutifs (nous dirons que la dimension du signal est *orthogonale* à la dimension d'adressage de la mémoire globale).

Le premier cas ne pose aucun problème étant donné que tout se déroule comme si on était dans le cadre mono-dimensionnel. Pour le second cas, il n'y a pas de problème à proprement parler, il faut seulement travailler avec des blocs de threads en deux dimensions. La première, alignée avec la dimension de la mémoire, sert à lire les données de manière coalescente ; et la seconde est alignée avec celle du schéma de calcul (la dimension du signal). Cela permet de décomposer la lecture et le traitement. Ainsi, lors de l'accès aux données, les indices des threads seront alignés avec les indices des traitements.

Dans les architectures actuelles (Fermi), les unités de mémoire traitent les accès mémoires de 16 threads simultanément [5], ainsi il suffira que la dimension des blocs qui est alignée avec la mémoire globale soit un multiple de 16 pour que tous les accès soient bien structurés.

5. Cette valeur ne fait pas partie d'un standard et peut, par conséquent, être amenée à évoluer.

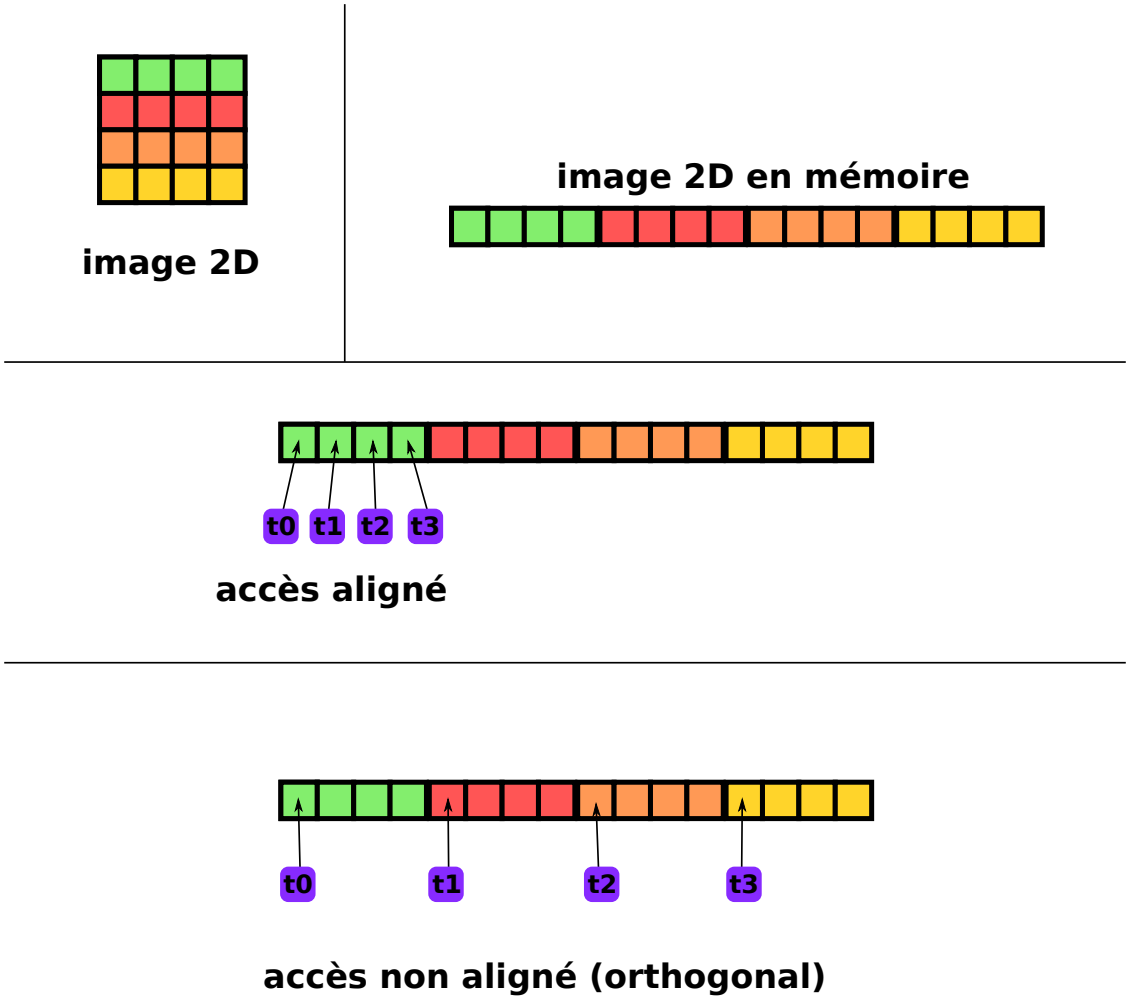


Figure 2.23 – Représentation graphique des deux cas possibles lors du traitement d'un signal multi-dimensionnel avec un schéma mono-dimensionnel (par exemple une convolution séparable).

2.4 Analyse de la complexité des algorithmes et schémas

Un élément important lors de la conception d'une chaîne algorithmique est de disposer d'un moyen d'évaluer rapidement la complexité d'un algorithme. Même si cette complexité n'est pas forcément une mesure absolue, mais plutôt une mesure relative permettant de comparer plusieurs solutions algorithmiques. Un tel outil permet aussi de mieux comprendre l'évolution des coûts de calcul en fonction de l'environnement de test, par exemple la taille des données manipulées ou le nombre de processeurs de la carte GPU utilisés.

Traditionnellement, en analyse algorithmique on utilise majoritairement les notations en grand *O* pour exprimer la complexité. Bien que cette notation soit utile, elle n'est pas adaptée aux architectures massivement parallèles car elle ne permet pas de prendre en compte le degré de parallélisme durant l'exécution de l'algorithme.

Lorsqu'on considère le modèle de calcul lié aux architectures massivement parallèles, il apparaît rapidement que la complexité (en termes de temps de calcul d'un algorithme) dépend de la complexité de la tâche confiée à chaque thread et du nombre de threads mis en œuvre, c'est-à-dire à la taille de la grille de calcul. Cependant, étant donné que la structure d'accès à la mémoire est un élément qui conditionne la manière dont un kernel GPU s'exécute, elle doit elle aussi intervenir dans la mesure de complexité. Aussi nous proposons de représenter la complexité d'un algorithme massivement parallèle par les trois valeurs suivantes :

- la complexité de traitement local effectué par un thread ;
- la structure du voisinage utilisé dans l'accès mémoire pour l'exécution du traitement local ;
- la cardinalité de la grille de calcul.

Pour représenter le voisinage, nous choisissons d'utiliser une liste de nombres. Par exemple $(1, 1)$ représente un voisinage de rayon 1 dans les deux premières dimensions du signal et $()$ signifie qu'il n'y a pas de notion de voisinage dans les traitements, comme c'est le cas dans le schéma de calcul pixel-à-pixel. Le voisinage a un impact important sur la construction de la grille de calcul à cause des problèmes de recouvrement entre blocs. En effet comme les données sont partagées uniquement entre threads d'un même bloc, les structures de voisinages de rayon non nul impliquent que les mêmes zones mémoires doivent être lues dans plusieurs blocs adjacents : c'est le recouvrement entre blocs. On peut voir sur la figure 2.24 différentes structures de blocs suivant le voisinage. Les carrés grisés représentant les threads inactifs qui servent uniquement à la lecture des données recouvrantes nécessaires aux calculs effectués par les threads actifs. Bien entendu le taux de threads actifs par bloc est une notion importante de l'efficacité globale du schéma.

Ainsi une convolution gaussienne séparable de rayon r sur une image de dimensions

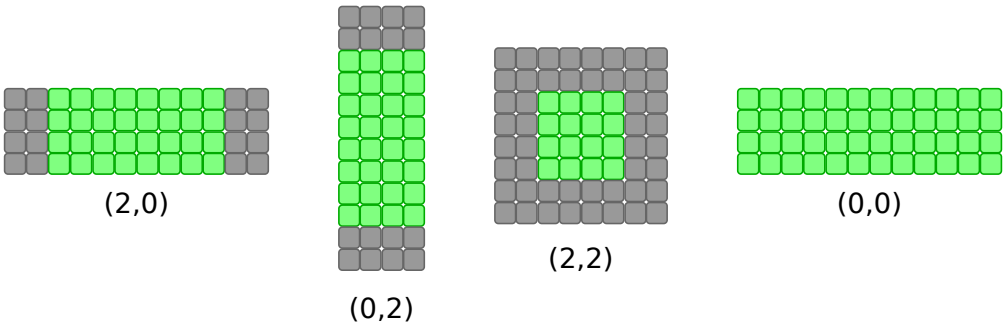


Figure 2.24 – La notion de voisinage dans la structure d'accès aux données en mémoire est très importante, elle définit en particulier le ratio de threads actifs par bloc.

(n_r, n_c) enchaîne deux schémas de convolution 1D, la complexité du premier (en ligne) se note $[4r+1, (r,0), n_r n_c]$ et celle du second (en colonne) $[4r+1, (0,r), n_r n_c]$. Pour un filtre non séparable, la complexité est quant à elle $[(4r+1)^2, (r,r), n_r n_c]$ ce qui montre non seulement une complexité de calcul plus élevée, mais aussi la nécessité de plus de mémoire locale partagée (*shared memory*).

2.4.1 La loi d'Amdhal

La loi d'Amdhal, du nom de l'architecte de processeur employé chez IBM qui l'a formulée, est donnée par :

$$a = \frac{1}{(1-s) + \frac{s}{N}} \tag{2.3}$$

où a est l'accélération d'un programme possédant une portion s de code non parallélisable obtenue en l'exécutant sur une machine à N processeurs. Selon cette loi, l'accélération de la parallélisation d'un programme est bridée par la portion de code séquentiel contenue dans ce code. Ainsi, si on s'en tient à la courbe de la figure 2.25, le gain en temps de calcul pour un programme contenant 10% de code non parallélisable est de seulement de 8.5 sur une architecture disposant de 50 processeurs. Mais même avec une infinité de cœurs, un programme possédant une telle portion séquentielle ne pourra pas dépasser une accélération de 10 de son exécution.

Cette loi est en général l'argument pessimiste qu'on oppose aux partisans du parallélisme. Le biais caché dans cette formule vient du fait qu'il existe une dépendance entre la taille des problèmes que l'on traite et la proportion de parallélisme qu'ils contiennent. Pour faire simple, plus on cherche à résoudre de gros problèmes (ce qui est possible grâce à l'augmentation de la puissance des processeurs), plus il y a de parallélisme dans ces problèmes. Pour les très

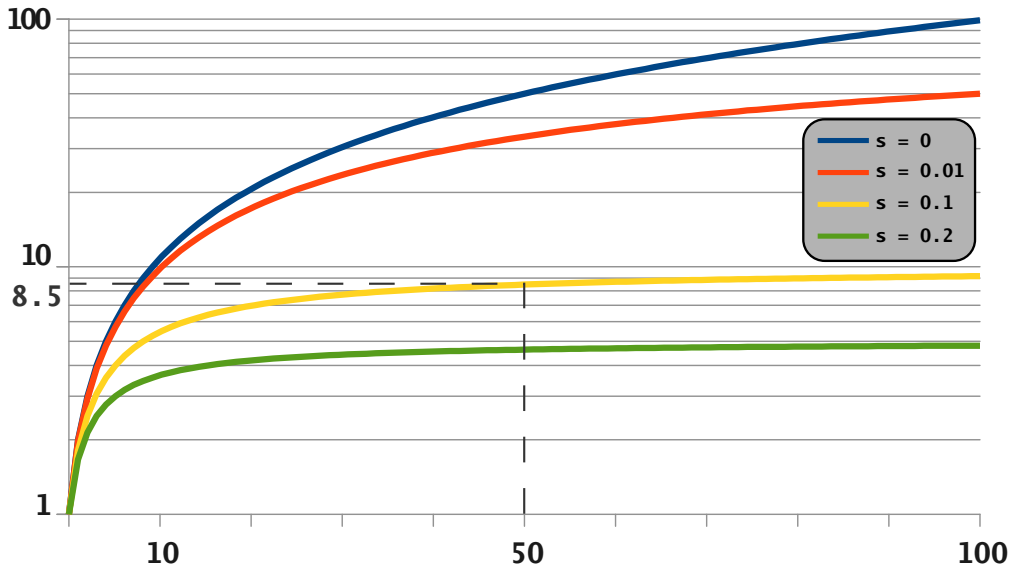


Figure 2.25 – Loi d'Amdhal : évolution de l'accélération de l'exécution d'un algorithme ayant une portion de code séquentiel s de 0%, 1%, 10% et 20% en fonction du nombre de processeurs.

gros problèmes on est donc en général proche de 100% de code parallèle.

Toute la difficulté de la conception de codes parallèles réside alors justement dans l'identification des schémas parallèles qui interviennent dans l'algorithme considéré. Un exemple simple qui illustre bien ce propos est l'exécution de la somme d'un très grand vecteur. Traditionnellement 90% des personnes ayant à écrire un programme réalisant une somme l'écrivent de la manière suivante (ou un de ses équivalents) :

```
sum = 0
for(i=0;i<N;i+=1)
  sum += A[i];
```

A première vue, cette portion de code est typiquement séquentielle. Or, en utilisant le schéma de la section 2.3.8, on peut effectuer une somme sur un grand vecteur de manière totalement parallèle. Ainsi, grâce à une transformation de la structure de l'algorithme, rendu possible ici par l'associativité de l'opération d'addition, on passe d'un algorithme séquentiel à un schéma parallèle.

Cette transformation d'algorithme, ici illustrée sur un exemple simplifié, peut être appliquée à des algorithmes plus complexes, mais elle devient elle-même très complexe à définir. On trouve par exemple dans les travaux d'Eva Burrows [Burrows, 2011] des schémas parallèles pour l'algorithme de tri.

2.5 Conclusion sur les architectures massivement parallèles

Dans ce chapitre nous avons rappelé brièvement comment les architectures massivement parallèles (et le parallélisme en général) ont transformé des sciences informatiques. Ces architectures procurent une large augmentation de la puissance de calcul disponible au plus grand nombre grâce à des produits bon marché. Cependant, ce surplus de puissance n'est exploitable qu'au prix d'une prise en compte du parallélisme au plus tôt dans le processus de conception algorithmique.

Nous avons présenté les principaux modèles de calcul entrant en jeu dans l'exploitation des architectures massivement parallèle. Nous avons aussi introduit un ensemble de schémas généraux de calculs, formant une sorte de taxonomie des opérations types que nous avons rencontrées durant nos travaux de thèse. Nous avons aussi introduit une notation permettant de représenter la complexité des algorithmes massivement parallèles afin de synthétiser leur structure.

Dans la suite de ce manuscrit, nous allons appliquer cette étude des architectures massivement parallèles aux problématiques de vision bas niveau. Nous allons ainsi montrer qu'il est aujourd'hui possible d'obtenir une chaîne de perception interactive sur un ordinateur grand public. Nous avons mené ces travaux durant la période de 2008 à 2012, en se focalisant que les GPU disponibles sur des stations de travail, qui disposaient d'API permettant leurs programmation massivement parallèle. A l'heure où nous écrivons ces lignes, ces fonctionnalités commencent à apparaître sur le marché des SoC destinés au marché des smartphones et tablettes tactiles. On peut citer par exemple le processeur exynos 5 de Samsung, offrant une puissance de 70Gflops grâce à son GPU. Avec l'arrivée de ces processeurs massivement parallèles à basse consommation, la porte s'ouvre pour de nouvelles applications de la perception embarquée.

Bibliographie

[Burrows, 2011] Burrows, E. (2011). *Programming with Explicit Dependencies : A Framework for Portable Parallel Programming*. PhD thesis, Research School in Information and Communication Technology, Department of Informatics, University of Bergen, Norway, PB 7803, 5020 Bergen, Norway.

[Champagnat and Le Sant, 2011] Champagnat, F. and Le Sant, Y. (2011). Efficient cubic b-spline image interpolation on a gpu. submitted to Journal of Graphics Tools.

[Chandrakasan et al., 1995] Chandrakasan, A., Potkonjak, M., Mehra, R., Rabaey, J., and Brodersen, R. (1995). Optimizing power using transformations. *Computer-Aided Design of Integrated Circuits and Systems, IEEE Transactions on*, 14(1) :12–31.

[Dixit et al., 2005] Dixit, N., Keriven, R., and Paragios, N. (2005). Gpu-cuts : Combinatorial optimisation, graphic processing units and adaptive object extraction.

[Fuller and Millett, 2011] Fuller, S. and Millett, L. (2011). Computing performance : Game over or next level ? *Computer*, 44(1) :31–38.

[Mairal et al., 2005] Mairal, J., Keriven, R., and Charior, A. (2005). A gpu implementation of variational stereo. *Rapport Technique, CERTIS-ENPC*.

[Sinha et al., 2006] Sinha, S., Frahm, J., Pollefeys, M., and Genc, Y. (2006). Gpu-based video feature tracking and matching. In *EDGE, Workshop on Edge Computing Using New Commodity Architectures*, volume 278. Citeseer.

[Sutter, 2005] Sutter, H. (2005). The free lunch is over : A fundamental turn toward concurrency in software. *Dr. Dobb's Journal*, 30(3) :202–210.

3 Estimation de mouvements

3.1 Introduction

L'estimation du champ des mouvements apparents entre deux images, ou flot optique, est à la base d'un très grand nombre de traitements vidéos. Cela permet par exemple de détecter les objets mobiles, ce qui peut s'avérer difficile lorsque l'observateur est lui-même en mouvement, comme nous le verrons à la fin de ce chapitre. Nous verrons dans le chapitre suivant comment exploiter le flot optique pour améliorer la qualité image par des techniques de super-résolution. Enfin l'estimation de mouvement image est aussi un préalable indispensable à la vision 3D : dans les perspectives situées dans la conclusion au chapitre 5, nous présenterons des résultats préliminaires sur l'utilisation d'un estimateur de flot optique conjointement avec un capteur de profondeur pour estimer les déplacements 3D de la scène observée. Dans toutes ces applications, il est nécessaire de disposer d'un outil d'estimation robuste et rapide du mouvement avec comme objectif la cadence vidéo, voir même au-delà pour certaines applications comme la super-résolution où plusieurs champs de mouvements sont nécessaires à chaque instant d'acquisition. Durant notre thèse, nous nous sommes donc intéressés dans un premier temps au développement d'une technique d'estimation de flot optique efficace et robuste tirant partie des avantages du calcul massivement parallèle sur GPU.

Notons qu'il existe un certain nombre travaux portant sur l'estimation rapide de champs de mouvements. Par exemple sur des architectures spécialisées comme des FPGA, on peut trouver des estimateurs de flot optique tournant à cadence vidéo [Barranco et al., 2012], mais on pourra remarquer que la résolution des vidéos traitées reste très faible. Depuis le milieu des années 2000, il existe aussi un certain nombre d'implémentations de flot optique utilisant des GPUs. Cependant, la plupart de ces travaux visent à l'accélération de traitements complexes et coûteux par le recours à une architecture massivement parallèle. Le résultat est en général un estimateur de bonne qualité (voire surdimensionné pour certaines applications) mais qui reste relativement lent sur des images de taille standard. Pour donner deux exemples dans les publications récentes, les codes décrits par Bruhn et al.[Bruhn et al., 2006] tournent en 16ms

à 400ms pour des images de dimension très réduite 160×120 ; les travaux de GPU4Vision [1] (T. Pock et al.) conduisent à des temps de calcul de l'ordre de la seconde sur des images de taille 640x480. Au contraire, nous nous intéressons à la recherche d'une technique qui puisse atteindre la cadence vidéo sur des images de grandes dimensions (au delà d'un megapixel) ou qui permette de recaler simultanément 20 images de dimensions réduites, ce qui nous sera utile pour la détection d'objets et la super-résolution. Dans le même temps, il faut que l'estimation soit robuste, assez précise et capable de gérer des mouvements importants entre les images.

Il se trouve que la base algorithmique d'une telle technique avait été proposée dans l'équipe EVF du DTIM quelques années avant le démarrage de nos travaux [Le Besnerais and Champagnat, 2005]. Cet algorithme, qui est nommé FOLKI pour Flot Optique par Lucas Kanade Itératif, relève des techniques de recalage de fenêtres locales présentées dans la section 3.2.2. En identifiant la structure extrêmement parallèle de cet algorithme et en y identifiant les différents schémas présentés au chapitre 2, nous avons pu proposer dès 2009 un code de flot optique, nommé FOLKI-GPU, remarquablement rapide et performant. La première section de ce chapitre présente cette solution et discute de ses performances et de ses paramètres.

Un autre domaine d'application de l'estimation de mouvements image est la mesure de champs de vitesse, de déplacement ou de déformation dans le cadre d'expérimentations et d'essais en mécanique des fluides et en science des matériaux. Ces applications, moins connues dans le domaine de la vision, ont donné lieu à des grands efforts de développements de méthodes robustes et surtout précises qui sont à la base de nombreux produits commerciaux. Les activités de mesure par imagerie sont devenues très importantes à l'ONERA dans les branches MFE (Mécaniques des Fluides et Energétique) et MAS (Matériaux et Structures). Partant du logiciel FOLKI-GPU, une collaboration avec des spécialistes de la mesure pour l'aérodynamique a permis d'obtenir FOLKI-SPIV, un nouvel outil de traitement dédié à la vélocimétrie par imagerie de particules (PIV) présenté en section 3.4 et qui a fait l'objet d'un article de revue [Champagnat et al., 2011] reproduit en Annexe A.1. Ce logiciel s'est révélé aussi précis et fiable que les outils commerciaux existants mais beaucoup plus rapide : en 2010, nous avons mesuré des gains d'un facteur 40 pour un résultat équivalent ou meilleur que celui donné par les logiciels commerciaux les plus récents de l'époque. Avec ce gain, le traitement des milliers d'images 4 megapixels qui forment une acquisition élémentaire dans ce domaine prend quelques minutes. C'est aussi la possibilité d'effectuer des réglages et des modifications des conditions expérimentales en observant les résultats de dépouillement des données en "temps interactif", c'est-à-dire à la cadence de visualisation de l'expérimentateur, ce qui ouvre la voie à

1. http://gpu4vision.icg.tugraz.at

des évolutions profondes de l'expérimentation dans ces domaines.

Dans la section 3.5, nous cherchons à améliorer l'estimation de flot optique que fournit FOLKI sans trop augmenter le temps de calcul. Une voie d'amélioration a été proposée dans l'équipe par Riadh Fezzani [Fezzani, 2011] sur le modèle CLG ("Combined Local-Global"), introduit à l'origine dans [Bruhn et al., 2005]. Nous rappelons cette approche, qui conduit à des temps de calculs nettement plus élevés que ceux de FOLKI. Nous proposons, sur le modèle de travaux récents concernant les techniques globales en flot optique [Sun et al., 2010], d'ajouter des traitements intermédiaires dans la structure FOLKI, sans changer cette structure pour améliorer les résultats. Ces améliorations sont illustrées sur les données du site Middlebury. Nous montrons ensuite l'effet de ces modifications sur des données réelles représentatives de divers domaines appliqués.

La dernière section 3.6 de ce chapitre aborde l'estimation de modèles de mouvements paramétriques à partir d'un flot optique. Nous montrons qu'avec des estimateurs très rapides comme FOLKI, il peut être avantageux de faire cette estimation par une régression robuste du flot plutôt que par les techniques habituelles de suivi de primitives. Ces modèles paramétriques correspondent à une décomposition du mouvement vidéo qui permet l'analyse de ce mouvement et en particulier la détection des objets mobiles.

En préliminaire, nous discutons brièvement en section 3.2 de la notion de flot optique, nous présentons les deux principales approches qui dominent la littérature depuis les années 80 et nous discutons des moyens d'évaluation des techniques d'estimation du flot optique.

3.2 Rappels sur le flot optique et son estimation

3.2.1 Notion de flot optique

Hypothèse fondamentale. Il est difficile de définir précisément la notion de flot optique. En effet, une vidéo n'est qu'une succession de tableaux de valeurs d'intensité ou de couleurs, les images, correspondant à la projection d'une scène 3D par un capteur optique. Les changements observables d'une image à l'autre résultent donc de la structure, du mouvement, et des propriétés d'illuminations de la scène, ainsi que des changements des paramètres de prise de vue. La définition du flot optique, comme son estimation, repose sur une modélisation de ces changements. La principale hypothèse de ce modèle consiste à attribuer ces changements à des déplacements dans la géométrie image, ce qu'on écrit :

$$\mathbf{I}_{t_1}(\mathbf{x}) = \mathbf{I}_{t_2}(\mathbf{x} + \mathbf{u}(\mathbf{x})), \tag{3.1}$$

autrement dit, l'intensité mesurée au pixel $\mathbf{x} = (x, y)^T$ à l'instant t_1 doit se retrouver à l'instant $t_2 = t_1 + \delta t$ à un autre pixel de position $\mathbf{x} + \mathbf{u}(\mathbf{x})$, avec $\mathbf{u} = (u, v)^T$. En particulier, cela suppose que les éléments observables conservent la même intensité aux temps t_1 et t_2, malgré le changement de point de vue éventuel de la caméra. L'équation (3.1) s'appelle l'équation du flot optique [Horn and Schunck, 1981], ou hypothèse de constance de l'intensité.

Exceptions à l'hypothèse de constance de l'intensité. Bien évidement, cette hypothèse n'est pas toujours valide pour les surfaces dont la réponse dépend des conditions d'éclairage, comme les surfaces spéculaires par exemple, ou bien lorsque les conditions d'illumination varient d'un instant à l'autre. Cependant, si on tient compte du fait qu'un capteur vidéo enregistre en général 10 à 30 images par seconde, ces variations de conditions d'illumination restent faibles entre deux images consécutives.

Un autre problème provient de la projection en 2D d'un monde 3D. Un objet peut en cacher un autre entre deux instants d'acquisition : on parle de zones d'occultation où le flot optique n'est pas défini. Ce phénomène d'occultation engendre des discontinuités locales dans le champ de mouvement observé, ce qui peut entraîner des artefacts dans le flot estimé. Une définition pratique de cette notion d'occultation est proposée par plusieurs auteurs, comme par exemple [Sand and Teller, 2008].

Enfin même en négligeant ces défauts du modèle, il reste une erreur inhérente à l'acquisition elle-même, ce qui fait que l'équation (3.1) ne doit être considérée exacte qu'à un bruit de mesure près. On recherchera donc le flot optique au sens d'un critère de recalage du type

$$\sum_{\mathbf{x}\in\Omega} \Phi\left(\mathbf{I}_{t_1}(\mathbf{x}) - \mathbf{I}_{t_2}(\mathbf{x} + \mathbf{u}(\mathbf{x}))\right) \tag{3.2}$$

où Φ est une fonction de pénalisation, souvent quadratique, et Ω désigne le support image, formé de N pixels indicés par $\mathbf{x}$.

Un problème non linéaire et sous-déterminé. Le problème de l'estimation de $\mathbf{u}$ est donc non linéaire (même avec une pénalisation quadratique, le critère (3.2) dépend de $\mathbf{u}$ par l'intermédiaire de la fonction d'intensité $\mathbf{I}_{t_2}$) et sous-déterminé. En effet, il est facile de voir qu'en linéarisant (3.2) et en cherchant à annuler le gradient, on trouve un système à N équations (le nombre de pixels) et $2N$ inconnues (le nombre de composantes de $\mathbf{u}$).

S'il fallait estimer réellement autant de vecteurs que de pixels, le problème serait sans solution. En fait, notre expérience des vidéos nous permet de dire que le flot optique est en général beaucoup plus contraint que cela. La plupart du temps, on peut considérer la

décomposition suivante :

$$\mathbf{u} = \mathbf{u}_{\text{ego}}(\theta, \mathbf{Z}) + \sum_{k=1}^{K} \mathbf{u}_{\text{objet}_k}(\theta_k), \tag{3.3}$$

c'est-à-dire que le flot optique $\mathbf{u}$ se décompose en la somme :

- d'un champ de déplacement lié au mouvement propre du capteur $\mathbf{u}_{\text{ego}}$ qui dépend d'un nombre réduit de paramètres de mouvements (typiquement 3 paramètres de translation et 3 paramètres de rotation, soit un vecteur θ à 6 paramètres) et éventuellement de la structure 3D de la scène, exprimée ici comme une image de profondeur $\mathbf{Z}$ qui est, en général, régulière par morceaux.
- et d'un ensemble de mouvements paramétriques $\mathbf{u}_{\text{objet}_k}$ liés aux objets mobiles (ou aux parties d'objets déformables) présents dans la scène.

Même si le nombre de degrés de liberté reste élevé, il est au final toujours bien inférieur à $2N$ (qui est de l'ordre de plusieurs millions sur un capteur moderne). Nous reviendrons sur la décomposition (3.3) à la fin de ce chapitre. Pour l'instant nous en déduisons que l'estimateur du flot optique $\mathbf{u}$ doit satisfaire à des conditions de régularité spatiale, en tous cas par morceaux, en plus du critère de recalage. Les techniques de régularisation spatiale sont introduites dans la section suivante.

3.2.2 Les principales approches pour l'estimation du flot optique

Bien que la littérature sur l'estimation de mouvement soit très riche, elle s'articule essentiellement autour de deux grandes méthodes de régularisation spatiale.

Méthodes globales. D'une part les *méthodes globales*, héritées de l'article pionnier de Horn et Shunk [Horn and Schunck, 1981], consistent à minimiser un critère régularisé du type :

$$J(\mathbf{u}) = \sum_{\mathbf{x}\in\Omega} D(\mathbf{x}, \mathbf{u}) + \lambda \mathcal{R}(\mathbf{x}, \mathbf{u}) \tag{3.4}$$

où le terme d'attache aux données est l'erreur de recalage :

$$D(\mathbf{x}, \mathbf{u}) = \Phi\left(\mathbf{I}_{t_1}(\mathbf{x}) - \mathbf{I}_{t_2}(\mathbf{x} + \mathbf{u}(\mathbf{x}))\right) \tag{3.5}$$

et le terme de régularisation dépend en général du gradient des composantes de $\mathbf{u}$:

$$R(\mathbf{x}, \mathbf{u}) = \Psi(\|\nabla u(\mathbf{x})\|) + \Psi(\|\nabla v(\mathbf{x})\|). \tag{3.6}$$

Φ et Ψ sont des fonctions de pénalisation. Ces formulations sont dites globales car l'intégration d'opérateurs de dérivation dans le terme de régularisation ajoute une contrainte globale sur l'ensemble du domaine image. L'optimisation de ce type de fonctionnelle conduit à des schémas numériques utilisant généralement des processus de type diffusion nécessitant un grand nombre d'itérations pour arriver à convergence.

Méthodes locales. De l'autre coté, on trouve les *méthodes locales* qui consistent à estimer un champ de mouvement paramétrique sur des voisinages de pixel (avec bien évidemment moins de paramètres que de pixels, sinon on retombe sur des systèmes sous déterminés), qui sont héritées de l'article de Lucas et Kanade [Lucas and Kanade, 1981].

Dans cette approche, pour déterminer le flot optique $\mathbf{u}(\mathbf{x})$ au pixel $\mathbf{x}$, on minimise le critère SSD (*Sum of Squared Differences*) :

$$L(\mathbf{x}, \boldsymbol{\theta}) = \sum_{\mathbf{x}' \in \Omega(\mathbf{x})} \left(\mathbf{I}_{t_1}(\mathbf{x}') - \mathbf{I}_{t_2}(\mathbf{x}' + \hat{\mathbf{u}}(\mathbf{x}', \boldsymbol{\theta}))\right)^2, \tag{3.7}$$

où $\Omega(\mathbf{x})$ désigne un voisinage spatial du pixel $\mathbf{x}$, typiquement une fenêtre carrée de rayon R : $\{\mathbf{x}' = (x', y') \mid |x' - x| \leq R \;\&\; |y' - y| \leq R\}$. Le champ de mouvement paramétrique $\hat{\mathbf{u}}(\mathbf{x}', \boldsymbol{\theta})$ estimé en minimisant (3.7) peut être affine (6 paramètres) ou tout simplement constant, auquel cas on l'associe au déplacement du pixel central : $\hat{\mathbf{u}}(\mathbf{x}', \boldsymbol{\theta}) = \hat{\mathbf{u}}(\mathbf{x}), \forall \mathbf{x}' \in \Omega(\mathbf{x})$. Notons qu'il existe de nombreuses variantes à cette approche utilisant d'autres critères que la SSD, par exemple une somme en valeur absolue (on parle souvent de SAD pour *Sum of Absolute Difference*) ou un coefficient de corrélation parfois appelé ZNCC, pour *Zero mean Normalized Cross-Correlation function*. Ces nombreuses techniques sont surtout populaires en stéréovision, et restent minoritaires dans le domaine de l'estimation de flot optique [Steinbruecker et al., 2009].

Approches hybrides. Il est impossible de rappeler ici toutes les approches qui ont été proposées depuis les années 80 pour l'estimation du flot optique. Le lecteur intéressé peut consulter le site de l'université de Middlebury, qui recense et compare plus de 70 méthodes [Baker et al., 2011]. On peut signaler cependant des méthodes hybrides qui utilisent le critère de recalage local (3.7) comme le terme d'attache aux données d'un critère global tel que (3.4). Ces méthodes ont été introduites dans [Bruhn et al., 2005] sous le nom de méthodes CLG pour *Combined Local-Global*. Elles ont été étudiées plus en détail par Riadh Fezzani au cours de sa thèse à l'ONERA [Fezzani, 2011, Fezzani et al., 2010]. Dans la section 3.5.1 nous reviendrons sur ces travaux concernant les méthodes CLG.

Evaluation du flot optique. Nous terminerons cette introduction rapide aux méthodes de flot optique en discutant brièvement de l'évaluation des performances. On peut remarquer que les

méthodes locales, qui sont simples, robustes et faciles à paramétrer, dominent les applications et les retombées commerciales du flot optique, par exemple pour la compression de vidéo ou la mesure par imagerie. Au contraire, les méthodes globales qui offrent de nombreuses voies d'amélioration et d'évolution, dominent les articles de recherche sur l'estimation du flot optique en vision par ordinateur et les comparatifs de performance, en particulier celui diffusé par l'Université de Middlebury [Baker et al., 2011].

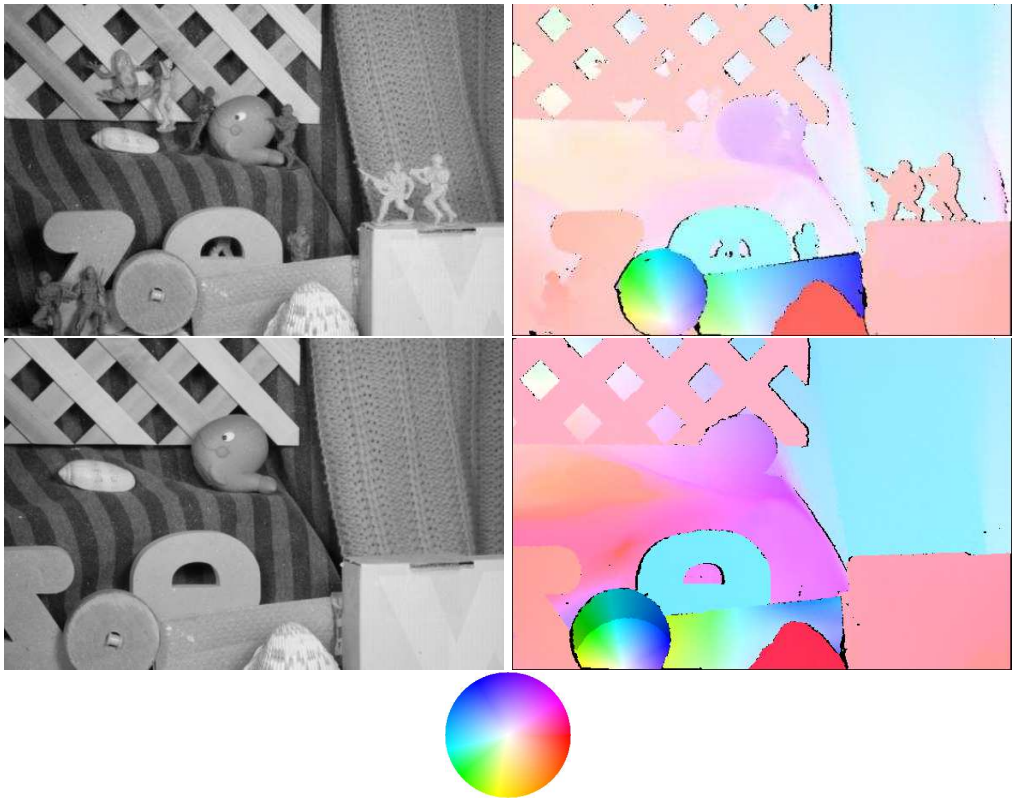


Figure 3.1 – En haut, exemple 'Army' de la base d'évaluation de flot optique de Middlebury. gauche, image convertie en niveaux de gris, au milieu vérité terrain (occultations en noir). En dessous, image et vérité terrain pour l'exemple 'RubberWhale' de la base d'apprentissage. Les couleurs de la représentation des flots codent l'orientation par la teinte et la norme par la saturation, suivant le code présenté au milieu en bas de la figure.

Le site de Middlebury propose un outil extrêmement pratique de comparaison objective sur des jeux de données avec vérité terrain. La figure 3.1 présente un exemple des données présentes sur ce site : il s'agit de couples d'images ou d'extraits de séquences, comprenant parfois une vérité terrain (le flot exact mesuré par un autre dispositif actif) qui permet d'évaluer des critères quantitatifs de performance. Deux jeux de données sont disponibles, un jeu d'ap-

prentissage, avec des vérité terrains acccessibles, pour effectuer le réglage de sa méthode et un jeu d'évaluation, dont la vérité terrain est connue uniquement des administrateurs du site, qui permet une évaluation objective des résultats. Les résultats sont mis en ligne dans un "wall of fame", qui se renouvelle régulièrement. La base d'apprentissage est conçue de sorte à présenter des exemples assez proches de certaines données de la base d'évaluation, comme on peut le voir sur la Figure 3.1. Il y a donc un risque de sur-apprentissage de ces données par la communauté : certains articles, comme celui de Sun et al. [Sun et al., 2010] ont mis à jour les *recettes* utiles pour monter dans le classement de ce comparatif — nous en rediscuterons en section 3.5.2.

On peut relever par ailleurs que ces données ont leurs limitations et peuvent être considérées comme peu représentatives des situations réelles d'emploi du flot optique. Dans certains domaines, la communauté est suffisamment développée pour organiser ses propres comparatifs sur des données sélectionnées. On peut donner en exemple les PIV Challenge [Stanislas et al., 2008], organisés au cours de certains congrès concernant la PIV.

Très récemment l'institut technologique de Karlsruhe (KIT) a mis en ligne sur le site *kitti* (http ://www.cvlibs.net/datasets/kitti/) des données urbaines issues d'un système d'acquisition embarqué dans une voiture, cf. Figure 3.2. Ce système comprend un capteur imageur stéréoscopique et un capteur actif 3D donnant une vérité terrain et des outils d'évaluation de performance pour le flot optique et pour d'autres problématiques de vision par ordinateur [Geiger et al., 2012]. Comme le site Middlebury, il fonctionne avec une base d'apprentissage téléchargeable et une base d'évaluation privée.

Dans l'introduction de la page consacrée au flot optique de ce site, les auteurs précisent leur motivation : "*Preliminary experiments show that methods ranking high on established benchmarks such as Middlebury perform below average when being moved outside the laboratory to the real world. Our goal is to reduce this bias and complement existing benchmarks by providing real-world benchmarks with novel difficulties to the community.*" Une première comparaison rapide indique que les principales différences entre les données de Middlebury et celles du site Kitti résident dans la valeur des déplacements maximaux (beaucoup plus grands dans le site Kitti) et dans les conditions d'illumination, qui sont très bien maîtrisées dans les données Middlebury, alors qu'il y a de très grosses variations dans les données réelles de Kitti.

Comme nos développements sur le flot optique visent plusieurs domaines d'applications, nous illustrerons nos résultats sur des données issues de plusieurs sites de comparaison et sur des images acquises à l'ONERA.

Figure 3.2 – En haut, Une image de la base d'apprentissage du site Kitti ; au milieu, la vérité terrain disponible sur un sous-ensemble de points mesurés par télémétrie laser ; en bas, le code de couleur utilisé.

3.3 FOLKI, un code de flot optique très rapide et performant

3.3.1 Algorithme

Nous allons reprendre rapidement ici les équations de l'algorithme FOLKI (Flot Optique Lucas-Kanade Itératif) initialement publié dans [Le Besnerais and Champagnat, 2005].

Cette méthode effectue la résolution locale de l'équation du flot optique (3.7) en supposant un modèle de mouvement localement constant sur la fenêtre et en intégrant une pondération :

$$L(\mathbf{x},\mathbf{u}) = \sum_{\mathbf{x}'\in\Omega(\mathbf{x})} \mathcal{W}_{\mathbf{x}}(\mathbf{x}') \left(\mathbf{I}_{t_1}(\mathbf{x}') - \mathbf{I}_{t_2}(\mathbf{x}'+\mathbf{u}(\mathbf{x}))\right)^2 \tag{3.8}$$

Cette équation doit être résolue simultanément sur tous les pixels $\mathbf{x}$ de l'image, pour fournir un flot optique estimé $\mathbf{u}(\mathbf{x})\forall\mathbf{x}\in\Omega$. L'objectif est d'obtenir une résolution itérative de ce problème de minimisation d'un critère des moindres carrés non linéaires. Pour cela, on suppose qu'on dispose à un instant donné d'une estimation initiale $\mathbf{u}_0$ du flot optique et on effectue un développement au premier ordre selon une stratégie de Gauss-Newton.

Si on fait un développement limité de (3.8) autour de $\mathbf{u}_0(\mathbf{x})$, au premier ordre en $\delta\mathbf{u}(\mathbf{x}) =$

$\mathbf{u}(\mathbf{x}) - \mathbf{u}_0(\mathbf{x})$, on écrit pour chaque terme de la somme

$$\begin{aligned}\mathbf{I}_{t_1}(\mathbf{x}') - \mathbf{I}_{t_2}(\mathbf{x}' + \mathbf{u}_0(\mathbf{x}) + \delta\mathbf{u}(\mathbf{x})) &\approx \mathbf{I}_{t_1}(\mathbf{x}' - \delta\mathbf{u}(\mathbf{x})) - \mathbf{I}_{t_2}(\mathbf{x}' + \mathbf{u}_0(\mathbf{x})) \\ &\approx \mathbf{I}_{t_1}(\mathbf{x}') - \nabla\mathbf{I}_{t_1}(\mathbf{x}')^T\delta\mathbf{u}(\mathbf{x}) - \mathbf{I}_{t_2}(\mathbf{x}' + \mathbf{u}_0(\mathbf{x})).\end{aligned}$$

Notons que nous avons reporté l'incrément de mouvement sur l'image à l'instant t_1, ce qui conduit au critère quadratique linéarisé suivant à minimiser :

$$L(\mathbf{x}, \mathbf{u}) = \sum_{\mathbf{x}' \in \Omega(\mathbf{x})} \mathcal{W}_{\mathbf{x}}(\mathbf{x}') \left(\mathbf{I}_{t_1}(\mathbf{x}') + \nabla\mathbf{I}_{t_1}(\mathbf{x}')^T(\mathbf{u}_0(\mathbf{x}) - \mathbf{u}(\mathbf{x})) - \mathbf{I}_{t_2}(\mathbf{x}' + \mathbf{u}_0(\mathbf{x}))\right)^2. \tag{3.9}$$

La minimisation se réduit alors à la construction et à l'inversion d'un système linéaire 2×2, dont la matrice normale s'écrit :

$$\mathbf{H}(\mathbf{x}) = \sum_{\mathbf{x}' \in \Omega(\mathbf{x})} \mathcal{W}_{\mathbf{x}}(\mathbf{x}') \nabla\mathbf{I}_{t_1}(\mathbf{x}') \nabla\mathbf{I}_{t_1}(\mathbf{x}')^T. \tag{3.10}$$

qui ne dépend pas du flot initial $\mathbf{u}_0$ grâce au report de l'incrément sur $\mathbf{I}_{t_1}$.

Cette méthode, utilisée par exemple dans [Bouguet, 2001], possède le désavantage que l'image décalée par le flot initial (qui intervient par l'intensité $\mathbf{I}_{t_2}(\mathbf{x}' + \mathbf{u}_0(\mathbf{x}))$) dépend de $\mathbf{x}$ et donc doit être recalculée d'un voisinage $\Omega(\mathbf{x})$ à l'autre. La complexité du calcul est alors de $O(NK)$ où N est le nombre de pixels et K le cardinal du voisinage. Ce coût peut vite devenir prohibitif lorsqu'on cherche à estimer le champ de mouvement sur l'ensemble d'une image.

Dans l'algorithme FOLKI, au contraire, on choisit de faire le développement limité autour de $\mathbf{u}_0(\mathbf{x}')$, ce qui donne :

$$L(\mathbf{u}) = \sum_{\mathbf{x}' \in \Omega(\mathbf{x})} \mathcal{W}_{\mathbf{x}}(\mathbf{x}') \left(\mathbf{I}_{t_1}(\mathbf{x}') + \nabla\mathbf{I}_{t_1}(\mathbf{x}')^T(\mathbf{u}_0(\mathbf{x}') - \mathbf{u}(\mathbf{x})) - \mathbf{I}_{t_2}(\mathbf{x}' + \mathbf{u}_0(\mathbf{x}'))\right)^2. \tag{3.11}$$

L'image $\mathbf{I}_{t_2}$ décalée par le flot initial ne dépend plus du pixel central de la fenêtre $\mathbf{x}$ mais uniquement du pixel courant $\mathbf{x}'$. Autrement dit, on peut se contenter de calculer et de stocker une seule image décalée $\mathbf{x}' \mapsto \mathbf{I}_{t_2}(\mathbf{x}' + \mathbf{u}_0(\mathbf{x}')), \forall \mathbf{x}' \in \Omega$ et de fournir les valeurs nécessaires de cette image au calculateur qui effectue la remise à jour locale au pixel $\mathbf{x}$.

La solution du problème linéarisé satisfait

$$\frac{\partial L}{\partial \mathbf{u}}(\mathbf{u}) = 0 \tag{3.12}$$

ce qui conduit au système 2×2 suivant :

$$\mathbf{H}(\mathbf{x})\mathbf{u}(\mathbf{x}) = \mathbf{c}(\mathbf{x}) \tag{3.13}$$

avec la matrice $\mathbf{H}$ vue en (3.10) et le second membre :

$$\mathbf{c}(\mathbf{x}) = \sum_{\mathbf{x}' \in \Omega(\mathbf{x})} \mathcal{W}_{\mathbf{x}}(\mathbf{x}') \nabla \mathbf{I}_{t_1}(\mathbf{x}') \left(\mathbf{I}_{t_2}(\mathbf{x}' + \mathbf{u}_0(\mathbf{x}')) - \mathbf{I}_{t_1}(\mathbf{x}') + \nabla \mathbf{I}_{t_1}(\mathbf{x}')^T \mathbf{u}_0(\mathbf{x}) \right) \tag{3.14}$$

La matrice normale du système local, $\mathbf{H}$, est la matrice de covariance empirique des vecteurs gradients de la fenêtre considérée. Son rang dépend de la texture locale. La matrice est singulière ou mal conditionnée lorsque la zone est homogène ou présente un contour unidirectionnel. Ce conditionnement est un indicateur de la précision de recalage qu'on peut attendre, donc de la qualité de la zone considérée pour l'observabilité des mouvements. Cette matrice est d'ailleurs à la base des extracteurs de points de Harris [Harris and Stephens, 1988] ou de Shi et Tomasi [Shi and Tomasi, 1994].

Lorsque la matrice $\mathbf{H}$ est mal conditionnée, donc par exemple dans les zones homogènes, on peut soit décider de ne pas estimer de vecteur flot à ce point (par un seuillage des valeurs propres) ; soit régulariser le problème en ajoutant une matrice diagonale à $\mathbf{H}$:

$$(\mathbf{H}(\mathbf{x}) + \beta \mathbf{Id}) \mathbf{u}(\mathbf{x}) = \mathbf{c}(\mathbf{x}), \tag{3.15}$$

ce qui, dans une structure itérative, correspond à ralentir l'algorithme en diminuant le pas de descente.

Au final, l'itération élémentaire de FOLKI consiste à partir d'un flot initial $\mathbf{u}_0$ et d'une image recalée $\mathbf{x}' \mapsto \mathbf{I}_{t_2}(\mathbf{x}' + \mathbf{u}_0(\mathbf{x}'))$ et à calculer une version remise à jour du flot $\mathbf{u}$ en inversant (3.13) ou bien (3.15) si une régularisation intervient. Dans la pratique, lorsque les mouvements sont importants, plusieurs itérations de ce type sont nécessaires pour aller à la convergence du critère de recalage local (3.8), ce qui conduit à une suite de flots $\mathbf{u}_k$. La section suivante décrit la structure de l'algorithme itératif obtenu.

3.3.2 Structure algorithmique

Comme beaucoup de méthodes d'estimation du flot optique, basées sur le développement limité de l'hypothèse de constance de l'intensité pour des petits mouvements, l'algorithme FOLKI s'inscrit dans un schéma de résolution multi-échelles "*coarse-to-fine*" afin de rendre possible l'estimation de plus grands mouvements. La structure externe est donc une boucle sur les niveaux d'une pyramide d'image, commençant au niveau le plus grossier jusqu'au niveau le plus fin, cf. Figure 3.3. En pratique nous utilisons une pyramide Gaussienne ou une pyramide de Haar, avec en général un facteur 2 entre les résolutions des images entre deux niveaux successifs, le nombre N_{lev} de niveaux étant pris maximal étant donnée la taille de l'image

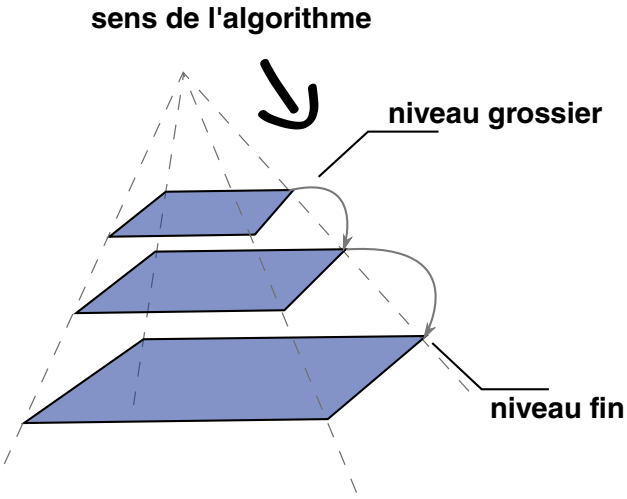


Figure 3.3 – Principe du calcul "coarse-to-fine" dans une pyramide d'images, utilisé par FOLKI.

d'origine. La structure interne effectue N_{iter} itérations sur le système (3.13) en réinterpolant l'image $\mathbf{I}_{t_2}$ à chaque fois en fonction du flot courant $\mathbf{u}_k$.

La constitution des systèmes locaux 2×2 et leur résolution *sur l'ensemble de l'image simultanément* suit le schéma présenté en Figure 3.4. On peut remarquer que les opérations nécessaires pour sa mise en œuvre sont très simples, et qu'elles peuvent être reliées aux motifs de calcul présentés au Chapitre 2 :

- des opérations *pixels à pixels* (voir section 2.3.1),
- une interpolations (voir section 2.3.5)
- des convolutions par des opérateurs à noyau limité et séparables (voir section 2.3.2) dont des gradients spatiaux.

Les coûts de calcul de l'algorithme FOLKI sont linéaires par rapport au nombre de pixels en entrée. Si on suppose qu'on effectue le calcul seulement sur un niveau de la pyramide pour lequel l'image contient p pixels et avec une fenêtre de rayon r, on obtient un coût de calcul en opérations élémentaires :

$$C(p,r,it) = 2[1,(1),p] + 3[1,(0),p] + 6[2r,(r),p] \quad (3.16)$$

$$+it\left([1,(0),p] + 2[1,(0),p] + 4[2r,(r),p] + [12,(0),p]\right), \quad (3.17)$$

où la partie 3.16 correspond à la complexité de la partie non itérative de FOLKI (la partie non-encadrée de la figure 3.4), et la partie 3.17 correspond aux opérations de la boucle d'itérations.

On remarquera que toutes les opérations sont au moins parallélisables en nombre de pixels. Les opérations les plus complexes de l'algorithme sont les convolutions séparables faisant intervenir un schémas FIR monodimensionnel. FOLKI a ainsi une structure algorithmique très

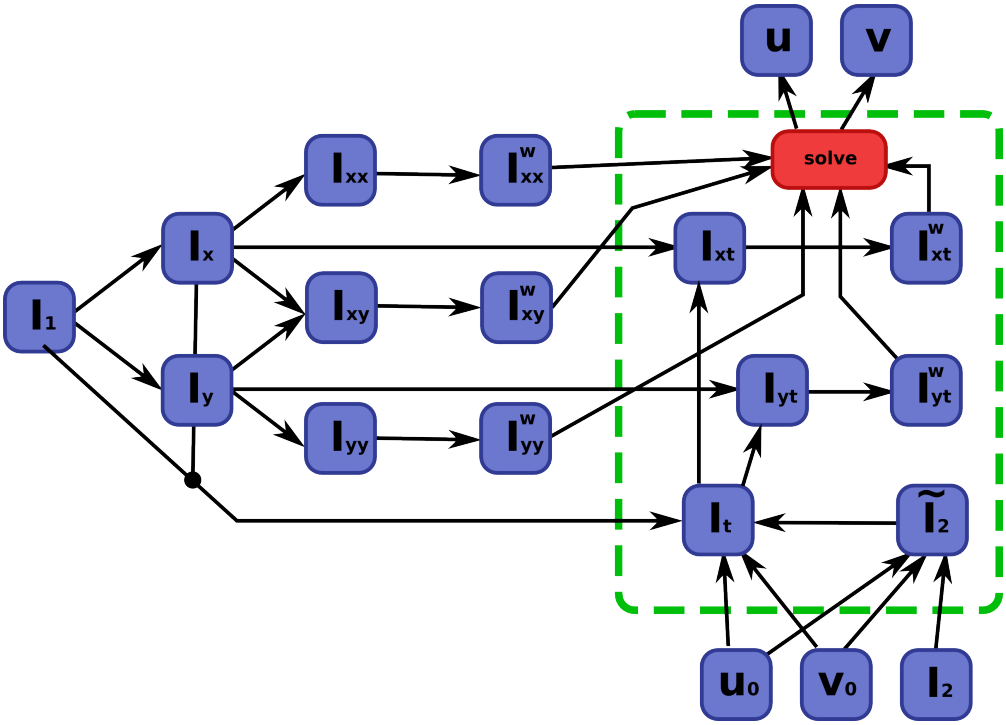


Figure 3.4 – Schéma de principe de FOLKI, le cadre vert représente la partie itérative de l'algorithme.

simple et profite par ailleurs (dans les zones qui sont correctement associées) de la vitesse de convergence du schéma de Gauss-Newton.

3.3.3 Un exemple de résultat

Pour une structure algorithmique aussi simple, les résultats de FOLKI sont remarquables. La Figure 3.5 présente le résultat du traitement d'une séquence aérienne issue d'une vidéo de la BBC au format HD (1920×1080). Sur cette séquence réelle où la caméra effectue une translation dans l'axe d'un canyon, le flot optique est un champ de zoom autour d'un point de fuite. La vitesse de défilement est liée à la distance à ce point de fuite et à la profondeur des reliefs relativement au centre optique. Le flot calculé est quasiment dense, les zones non renseignées correspondent aux zones d'ombre qui sont homogènes, ce qui implique que le système FOLKI est singulier.

En regardant plus attentivement la correspondance entre les ruptures de relief et les changements d'intensité de la carte de norme du flot (par exemple le rocher en saillie situé à gauche du champ), on constate l'effet classique des méthodes de recalage de fenêtres, qui est d'empâter les structures en proportion de la taille de fenêtre. Notons que dans une perspective d'évitement d'obstacle, ce biais n'est pas forcément très gênant puisqu'il conduit à surestimer l'emprise des obstacles et donc à adopter une attitude plus prudente.

Sur la figure 3.6 on peut voir les temps de calcul de l'algorithme pour une paramétrisation qu'on peut prendre comme standard. Le coût de calcul de FOLKI est presque exactement linéaire par rapport au nombre de pixels traités : la cadence vidéo est tenue pour des images jusqu'à 4 megapixels, une performance bien supérieure à celles de l'ensemble des algorithmes concurrents sur GPU. On remarque que cette loi linéaire s'aplatit légèrement pour les petites images : pour ces dimensions, la charge de calcul est trop faible pour profiter pleinement de toute la puissance de calcul du GPU. Cette évaluation du coût de calcul s'arrête ici à une résolution de 2048×4096 car on atteint alors les limites de la carte GTX480 utilisée en termes de mémoire (1Go). Pour les résolutions plus élevées, cette carte ne permet pas d'initialiser les structures de mémoire fixes nécessaires au calcul. La place mémoire nécessaire à FOLKI correspond approximativement à 20 fois la taille de l'image traitée, soit pour une vidéo HD à 166Mo de mémoire.

3.3.4 Paramètres

Un des avantages des méthodes locales est la simplicité de leur paramétrage. FOLKI a essentiellement 4 paramètres : la taille de fenêtre qui peut être spécifiée par un rayon en pixel

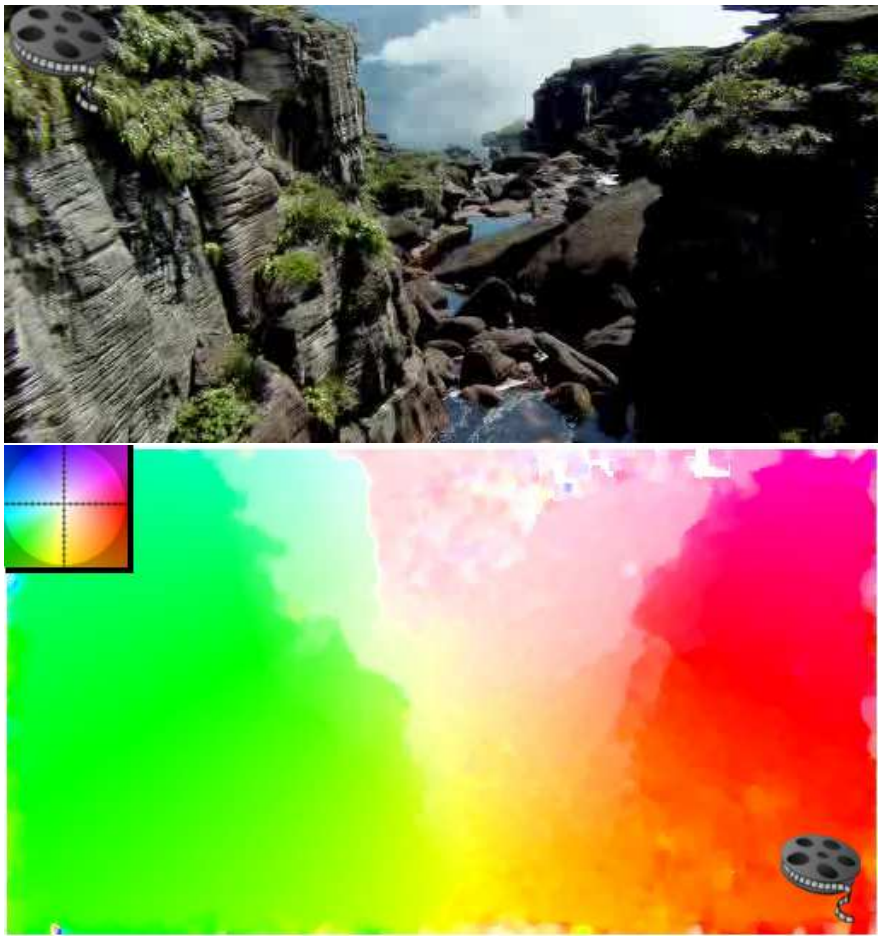


Figure 3.5 – Illustration de l'estimation du flot optique par FOLKI sur une séquence HD (1920x1080) issue du film "One day on Earth" de la BBC. Paramètres : fenêtre de rayon 10, deux itérations par niveaux et 6 niveaux de résolution. Le temps de calcul est de seulement... 17ms ! En haut, image d'origine ; en bas flot affiché avec une échelle de couleur codant la norme et l'orientation du champ de mouvement (cf. [Baker et al., 2011]). Les variations du flot suivent les contours des principaux reliefs : le champ de mouvement est en effet un champ de parallaxe dû à la structure 3D de la scène et au mouvement de l'observateur dans l'axe du canyon. Contenu vidéo : image supérieure, séquence d'origine ; image inférieure, norme du flot optique

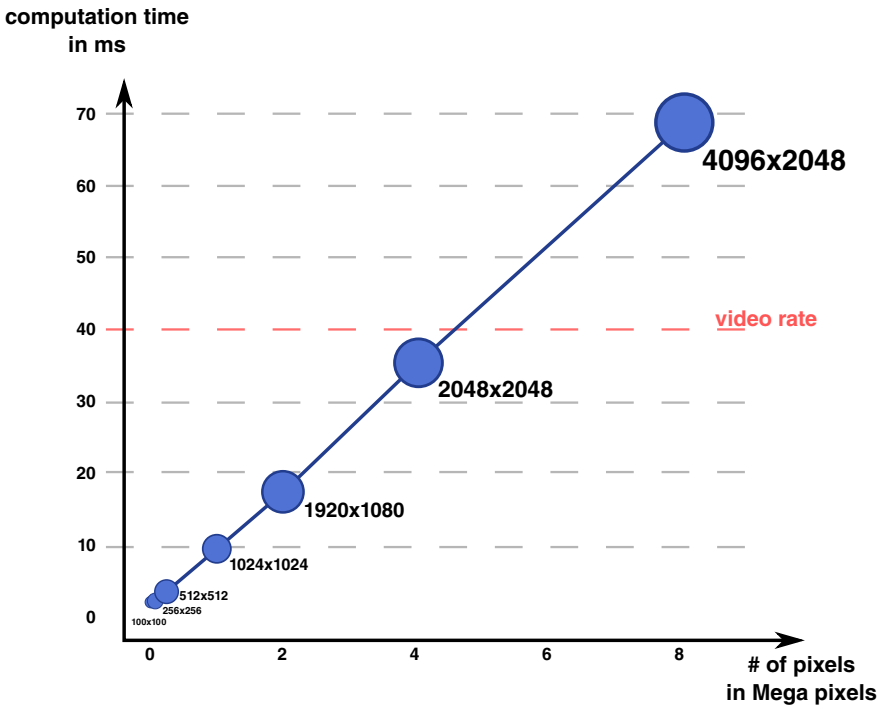


Figure 3.6 – Temps de calcul de l'algorithme FOLKI sur une GTX480 avec des fenêtres de rayon 8, deux itérations par niveaux et 6 niveaux.

pour une pondération uniforme ou un écart-type pour une pondération Gaussienne [2] ; le nombre de niveaux de résolution N_{lev} dans la pyramide d'images ; le nombre d'itérations par niveau de résolution N_{iter} ; enfin un seuil ou un coefficient de régularisation permettant d'éviter d'inverser des matrices singulières, voir l'équation (3.15). Il y a quelques paramètres auxiliaires concernant le type d'interpolation ou de gradient image utilisé et les options de masquage pour les bords.

Comme indiqué précédemment, le nombre de niveaux N_{lev} ne fait en général pas l'objet de débats : il est en fait déterminé en fonction de la taille des images de la vidéo d'entrée. Le nombre d'itérations N_{iter} joue un rôle plus important. Même si, de par sa structure de type Gauss-Newton, l'algorithme a, dans la plupart des cas, une vitesse de convergence rapide (2 à 3 itérations), il peut être intéressant de laisser itérer plus longuement (5 à 10 itérations) sur certaines zones. Ce comportement est visible sur les bords des personnages en mouvement de l'exemple en figure 3.7. Ces zones, rares, peuvent néanmoins être importantes pour la compréhension de la scène, ici pour la segmentation des personnages.

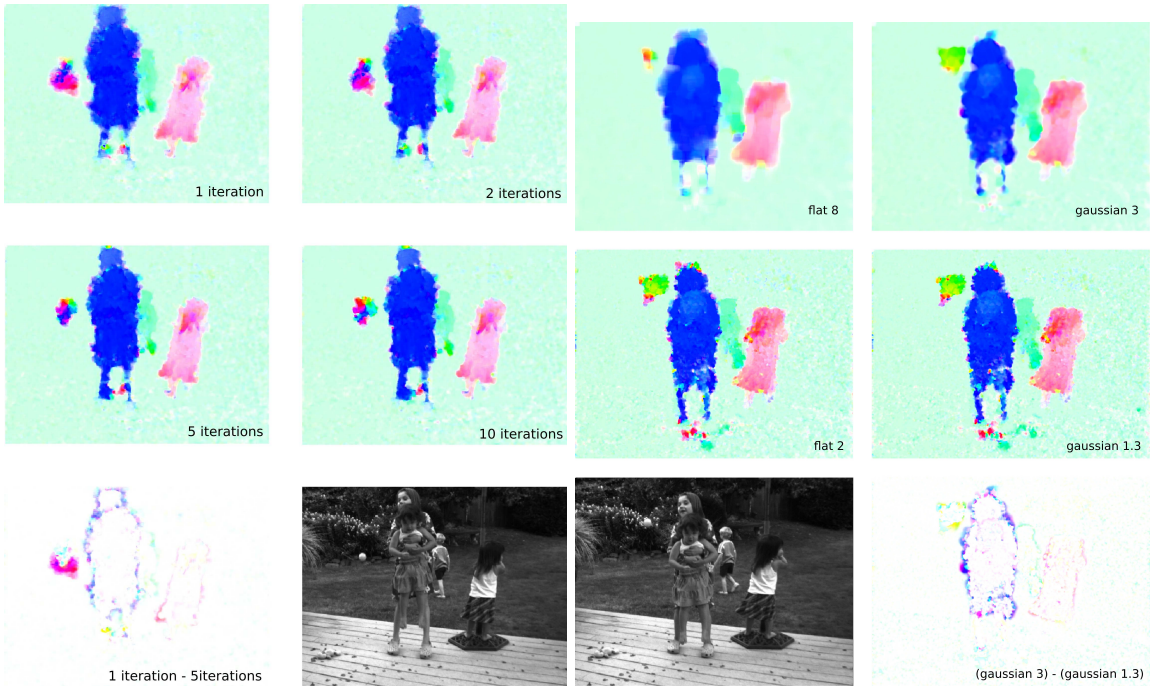


Figure 3.7 – Etudes paramétriques sur FOLKI. A gauche : Influence du nombre d'itérations. A droite : Variation de la largeur de la fenêtre et du type de pondération.

Le principal paramètre est bien entendu la dimension de la fenêtre de calcul locale qui fixe essentiellement la résolution du champ de mouvement estimé, comme cela est montré

2. Pour des raisons d'efficacité de calcul, nous n'utilisons que des pondérations conduisant à des convolutions séparables.

dans [Champagnat et al., 2011]. Il y a un compromis classique de type biais/variance sur ce paramètre : une grande fenêtre donne une estimation de mouvement moins bruitée mais lisse les discontinuités de mouvement, une fenêtre de petite taille limite le biais mais conduit à une grande variance d'estimation. Le choix de la pondération appliquée à la fenêtre (Gaussienne ou uniforme), illustré en Figure 3.7, est moins important, il s'agit essentiellement de donner une apparence différente au bruit résiduel de l'estimation. En pratique, on préfère la fenêtre uniforme qui fait économiser un peu de temps de calcul (pas de multiplication par la pondération).

3.4 Application de FOLKI à la PIV : FOLKI-SPIV

En 2009, nous avons présenté les premiers résultats d'estimation de champs de mouvement par FOLKI sur des images PIV (*Particle Image Velocimetry*) à Yves Le Sant, responsable de l'équipe MTRO au Département d'Aérodynamique Fondamentale et Appliquée (DAFE) de l'ONERA. Un exemple issu de la séquence que nous avons traitée à l'époque est présenté en Figure 3.8. Cette séquence, disponible sur le site du projet européen FLUID (http ://fluid.irisa.fr/), est composée d'images de dimension 1024×1024 et FOLKI permettait de les traiter à cadence vidéo, une performance absolument inédite à l'époque dans ce domaine.

Une collaboration s'est mise en place entre l'équipe du DTIM au sein de laquelle se déroule cette thèse et l'équipe DAFE/MTRO. Le résultat a été FOLKI-SPIV, une version de FOLKI dédiée aux données PIV qui a rapidement fait l'objet de plusieurs publications [Champagnat et al., 2009, Le Besnerais et al., 2009, Leclaire et al., 2010, Champagnat et al., 2011]. Le logiciel FOLKI-SPIV est maintenant développé et maintenu par l'équipe DAFE/MTRO et distribué en interne ONERA aux équipes d'expérimentateurs qui font des mesures PIV, en particulier depuis 2011 dans les grandes souffleries de l'ONERA, exploitées le département GMT (Grands Moyens Techniques).

La présente section décrit brièvement le contexte de la PIV et présente quelques résultats récents obtenus à l'ONERA à l'aide de ce logiciel. Pour plus de détails sur l'adaptation de FOLKI à la PIV et sur les validations menées par le DAFE et le DTIM, nous avons reproduit l'article de revue [Champagnat et al., 2011] en annexe de ce manuscrit.

3.4.1 Mesure PIV

La vélocimétrie par imagerie de particules (*Particle Image Velocimetry*, PIV) est une technique de mesure des mouvements de fluides apparue au milieu des années 80 [Raffel et al., 2007]. La PIV est devenue un outil essentiel de l'expérimentation en mécanique des fluides, et de ce

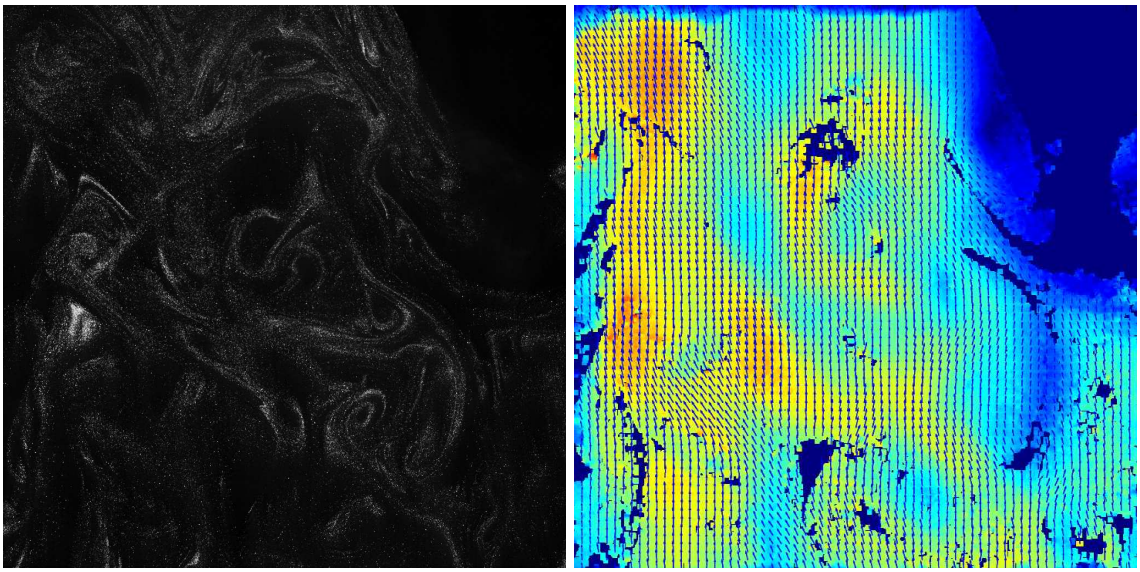


Figure 3.8 – Exemple de traitement d'image PIV par FOLKI. A gauche image issue du jeu de données PIV "time resolved" mis en ligne par B. Wienecke (site du projet FLUID, http ://fluid.irisa.fr/). A droite résultat de FOLKI, champ de mouvement (flèches) et rotationnel (couleurs)

fait, elle est mise en œuvre régulièrement par différentes équipes de l'ONERA : un aperçu de ces réalisations peut être trouvé dans la référence [Brossard et al., 2009].

Le principe de la PIV est de réaliser deux images d'un ensemble de particules en suspension dans un fluide puis à comparer les deux images pour estimer le champ de déplacement. L'hypothèse principale est que les particules sont suffisamment fines pour suivre "parfaitement" le mouvement du fluide dans lesquelles elles se trouvent. En général, les particules ne sont pas naturellement présentes, il faut donc les y introduire, une opération délicate appelée "ensemencement". L'illumination est assurée par un double flash laser et une caméra couplée avec le laser enregistre les images des particules. Ces images, dont un exemple est présenté sur la gauche de la Figure 3.8, sont constituées d'un nuage de points lumineux, plus ou moins séparés suivant les réglages. Le traitement numérique "traditionnel" en PIV consiste à associer non pas les particules individuelles (la densité est trop importante pour qu'on parvienne à "individualiser" les particules) mais des fenêtres comprenant un petit nombres de particules (typiquement 8) qui vont être recherchées comme des motifs rigides dans l'autre image.

Le traitement de PIV classique repose sur la maximisation d'un critère de corrélation entre des "fenêtres d'interrogation" : on retrouve là exactement le principe des approches locales du flot optique. Cependant les logiciels de PIV, dans leur grande majorité, effectuent une maximisation éventuellement itérative mais fondée sur le calcul de la corrélation par FFT (*Fast Fourier Transform*)

comme brique de base. De plus, le calcul est en général effectué en des points séparés d'une fraction (entre 0.75 et 0.5) de la taille de la fenêtre, et non en tous les pixels comme dans FOLKI.

Il est important de souligner que l'exploitation d'une expérimentation PIV ne se fonde pas sur l'examen d'un unique champ de déplacement calculé entre deux images, mais sur le résultat de post-traitements effectués sur des centaines, voire des milliers de champs calculés sur des acquisitions (de paires d'images) successives. Les post-traitements les plus simples sont typiquement des dérivations spatiales (calcul de rotationnel, de divergence) et des moyennes temporelles. Mais on fait aussi des opérations plus complexes, par exemple des analyses en composantes principales. Notons que suivant la cadence possible pour le système PIV et la dynamique du phénomène observé, les acquisitions sont considérées comme indépendantes ou corrélées. Dans ce dernier cas, on parle de PIV *Time-Resolved* (PIV-TR) — c'est le cas de la séquence de la Figure 3.8. Pour les applications aérodynamiques, qui intéressent l'ONERA, la PIV-TR exige une cadence du système PIV de l'ordre de 10kHZ. Ces cadences ne sont accessibles aux systèmes PIV commerciaux que depuis environ 5 ans : un exemple récent d'expérimentation de PIV-TR à l'ONERA est présenté en figure 3.12.

3.4.2 Adaptation de FOLKI pour la PIV

Le travail effectué sur FOLKI a consisté à améliorer sa précision et sa robustesse aux dégradations typiques des images PIV. Le développement d'une première version de FOLKI-SPIV, effectué par Yves Le Sant au DAFE/MTRO à partir d'éléments que nous lui avons transmis, a intégré trois modifications : l'utilisation d'un critère symétrique ; la prise en compte d'un masque sur l'estimation ; la normalisation locale des fenêtres.

Critère symétrique. Cette variante était déjà proposée dans l'article [Le Besnerais and Champagnat, 2005]. Il s'agit de minimiser un critère donnant un rôle symétrique aux deux images :

$$\sum_{\mathbf{x}' \in \Omega(\mathbf{x})} \left(\mathbf{I}_{t_1}(\mathbf{x}') - \mathbf{I}_{t_2}(\mathbf{x}' + \mathbf{u}(\mathbf{x})/2)\right)^2 + \left(\mathbf{I}_{t_1}(\mathbf{x}' - \mathbf{u}(\mathbf{x})/2) - \mathbf{I}_{t_2}(\mathbf{x}')\right)^2 . \tag{3.18}$$

Expérimentalement, il semble que ce critère permette une meilleure convergence (en particulier près des obstacles en lien avec la gestion des masques, voir plus bas) au prix d'un coût de calcul plus élevé. En effet, lors de la remise à jour du flot à partir de l'estimée précédente $\mathbf{u}_0$, la matrice à inverser devient :

$$\mathbf{H}(\mathbf{x}) = \sum_{\mathbf{x}' \in \Omega(\mathbf{x})} \nabla \mathbf{I}_{t_1}(\mathbf{x}' - \mathbf{u}_0(\mathbf{x}')/2) \nabla \mathbf{I}_{t_2}(\mathbf{x}' + \mathbf{u}_0(\mathbf{x}')/2)^T. \tag{3.19}$$

On voit que cette matrice dépend maintenant de l'estimée courante du flot, il faut donc la recalculer à chaque itération.

On peut noter que le flot qui apparaît dans (3.18) correspond à l'estimation du champ de vitesse à un instant $t = (t_1 + t_2)/2$ intermédiaire entre les deux instants d'acquisition. Cette caractéristique est gênante en général en vision par ordinateur, car on cherche à relier le flot optique aux mouvements de certains objets (cf. la décomposition 3.3) extraits de l'une des deux images. Or le flot intermédiaire n'est superposable à aucune des deux images, il doit être interpolé pour être remis dans une géométrie ou l'autre. Les calculs supplémentaires que cela entraîne font que nous n'utilisons pas la version symétrique dans ces applications. En PIV par contre, il est jugé intéressant d'avoir un champ de déplacement estimé à l'instant milieu.

Masques dynamiques. L'estimation du champ de déplacement se fait sur un domaine limité par le support des images sur le plan d'observation et par les bords des objets présents dans la scène (maquette, parois du conduit). Beaucoup d'informations intéressantes se trouvent à proximité de ces frontières, comme le phénomène de décollement près de la paroi inférieure dans l'exemple présenté en figure 3.11. Or l'approche par recalage de fenêtre pose des problèmes aux frontières lorsque la fenêtre de calcul contient des pixels en dehors du domaine. Le calcul exact du système linéaire (3.13) ne peut se faire que dans un sous-domaine de calcul "érodé" d'une demi-fenêtre à l'intérieur du support effectif. De plus lorsque, au cours des itérations, le vecteur déplacement estimé pointe vers l'extérieur du support, on est conduit à reculer encore la frontière du sous-domaine de calcul — et ce phénomène est encore amplifié par l'approche pyramidale. Pour gérer ce problème, FOLKI-PIV utilise un masque dynamique indiquant le sous-domaine de calcul, remis à jour à chaque itération. Les pixels exclus de ce masque sont ceux affectés d'un déplacement estimé courant conduisant à une position hors du support image et ceux dont la fenêtre contient plus de 80% de pixels déjà exclus. Notons que dans la version symétrique le déplacement courant utilisé est $\pm\mathbf{u}_0/2$, ce qui tend à limiter la réduction du masque lorsque le déplacement sort du support.

Normalisation locale. Dans sa forme de base, FOLKI minimise le carré de la différence d'intensité entre les images prises en t_1 et t_2, il est donc sensible à des changements d'illumination locaux ou même globaux d'un instant à l'autre. Un moyen classique (en stéréoscopie par exemple) pour améliorer la robustesse est de travailler sur une corrélation centrée normée ZNCC. Pour cela il suffit simplement de centrer et normer (diviser par l'écart-type empirique) les images à chaque itération de l'algorithme — en fait, la seule image pour laquelle il sera nécessaire de recalculer cette normalisation locale est l'image $\mathbf{I}_2$ qui est réinterpolée à chaque itération. Le schéma de principe de la version normalisée exacte de FOLKI est présenté en Figure 3.9.

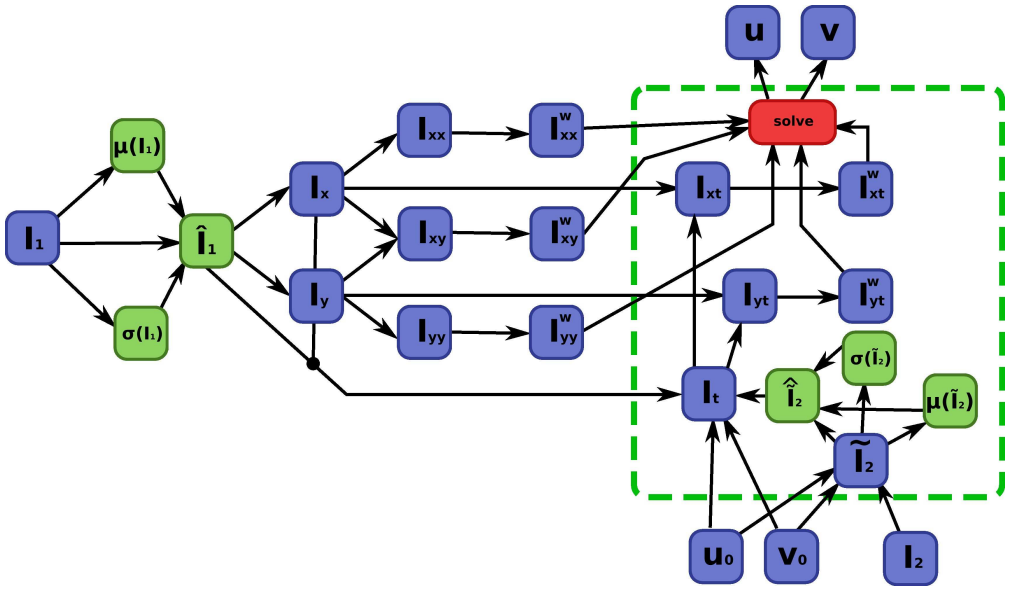


Figure 3.9 – Schéma de l'algorithme FOLKI avec normalisation des fenêtres pour la version exacte (lente). Les opérations supplémentaires par rapport à l'algorithme FOLKI original sont en vert.

On utilise, dans un souci d'efficacité de calcul, une simplification de cette version de FOLKI normalisé (cf. Figure 3.10) en précalculant simplement la moyenne et l'écart-type de $\mathbf{I}_2$ au début de l'algorithme, puis, à chaque itération, en interpolant les valeurs de $\mathbf{I}_2$, de l'image des moyennes locales et de l'image des écart-types locaux, pour effectuer la normalisation. Ainsi le coût supplémentaire est limité à deux interpolations par itération et quatre convolutions par niveau de résolution.

3.4.3 Résultats et extensions

FOLKI-PIV est une méthode d'estimation de champ de déplacements fondée sur un recalage de fenêtres locales comme les techniques de PIV traditionnelles. Les différences concernent l'utilisation d'un critère SSD et la technique d'optimisation itérative par Gauss-Newton. Le principal gain résultant de ces caractéristiques et de l'utilisation des GPU est le temps de calcul. A l'époque de la publication de l'article [Champagnat et al., 2011], le facteur d'accélération relatif aux logiciels commerciaux (sur CPU) était évalué entre 20 et 40. Même à l'heure actuelle, où les systèmes PIV incorporent tous des GPU, il reste un gain important du fait que la FFT est algorithmiquement moins efficace sur des architectures massivement parallèles que les structures de calcul de FOLKI.

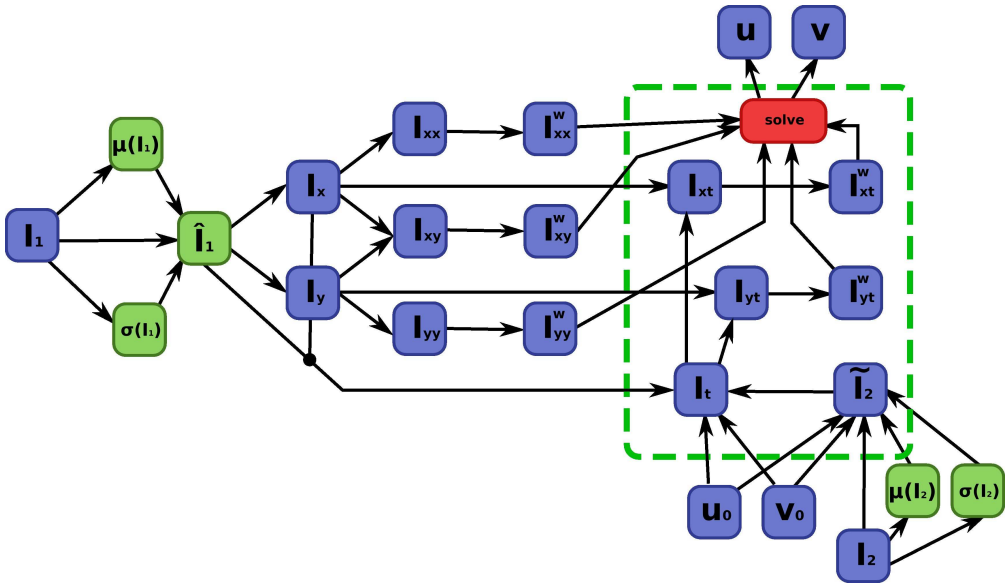


Figure 3.10 – Schémas de l'algorithme FOLKI avec normalisation des fenêtres pour la version approximée rapide, les opérations supplémentaires par rapport à l'algorithme FOLKI original sont en vert.

On peut remarquer que FOLKI fournit une estimation *dense* du champ de déplacement (ie. un vecteur par pixel) alors que, pour un temps plus élevé, les logiciels commerciaux donnent un champ sous-échantillonné d'un facteur au moins 4. Cependant la résolution effective du champ estimé par FOLKI est fixée par la taille de la fenêtre de calcul [Champagnat et al., 2011] et non par la densité des points de calcul. Il n'y a donc pas de gain en résolution par rapport aux méthodes traditionnelles. En revanche le champ de déplacement est disponible en tout point sans interpolation ce qui est utile pour mesurer des phénomènes localisés près des parois, comme des décollements. Sur ce type de phénomène, la figure 3.11 présente un résultat de FOLKI-PIV obtenue dans la soufflerie S19Ch de l'ONERA au cours d'une expérimentation menée par le DAFE, qui est décrite plus en détail dans la référence [Champagnat et al., 2011], cf. Annexe A. Le zoom présenté en bas de la figure montre l'intérêt du calcul dense pour fournir une estimation des déplacements qui peuvent aller en sens inverse par rapport à l'écoulement global dans la zone de décollement.

Ci-dessus nous rendons compte brièvement des travaux publiés récemment sur les extensions de FOLKI-PIV.

Précision. Pour la publication [Champagnat et al., 2011], un important travail de caractérisation de la performance de FOLKI-PIV a été mené, principalement par Samuel Davoust, doctorant

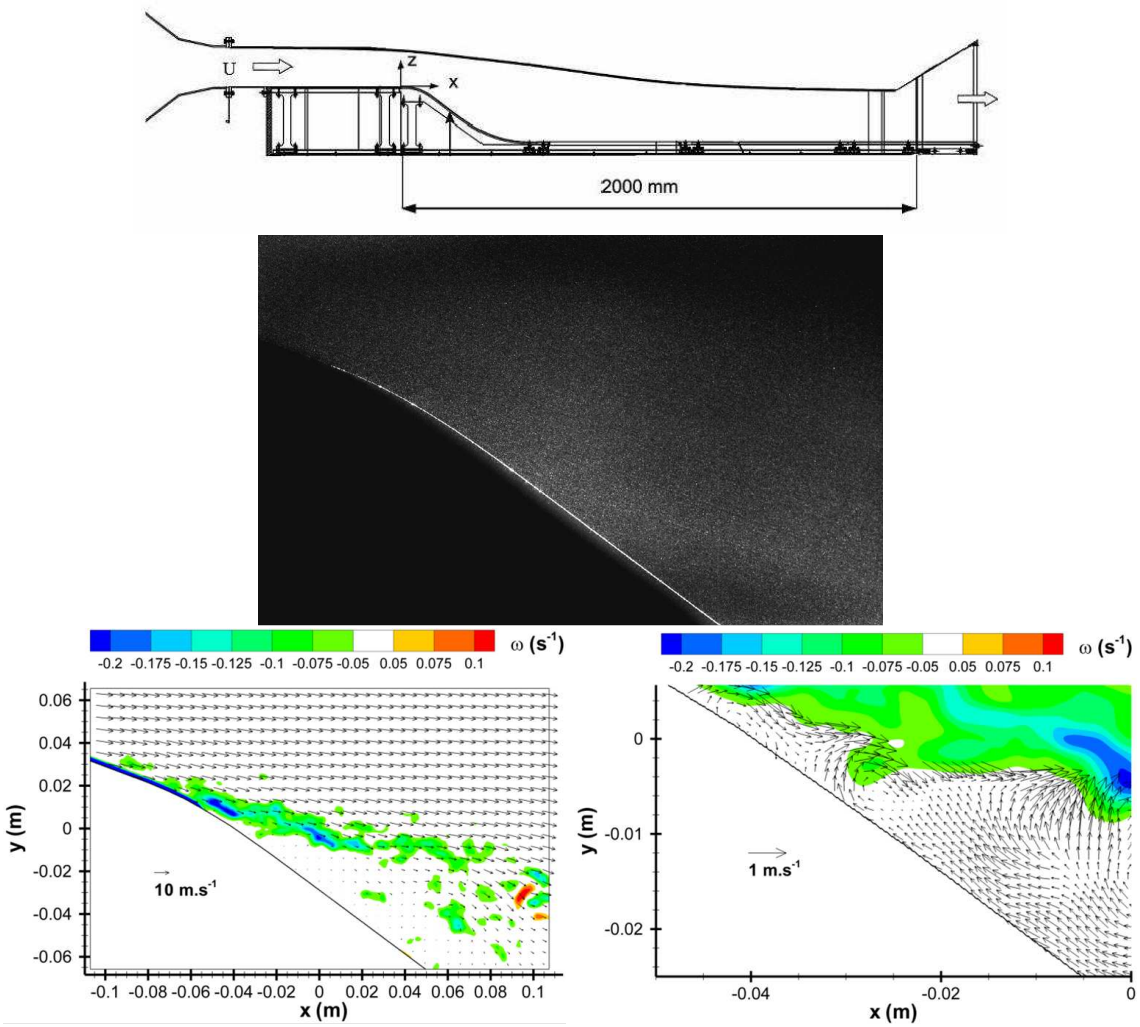


Figure 3.11 – Mesure PIV dans la soufflerie S19Ch de l'ONERA. Haut : plan de coupe du montage expérimental, l'écoulement va de gauche à droite et les observations concernent le décollement qui se produit sur la rampe descendante. Milieu : une image PIV, la rampe est clairement visible dans la partie inférieure. Bas : à gauche champ de déplacement estimé par FOLKI-PIV, à droite zoom du champ sur la zone de décollement. Expérimentation, image et traitements : ONERA/DAFE.

au DAFE [Davoust, 2011]. Les résultats, présentée en Annexe A montrent qu'avec une très bonne approximation, FOLKI-PIV fournit un champ de vitesse convolué par la fenêtre de calcul. Choisir une taille de fenêtre réduite permet de limiter ce biais, mais conduit à une amplification du bruit affectant le résultat (variance). L'article présente des courbes illustrant ce compromis biais/variance pour différent types de dégradation des images (bruit additif, disparition de particules à cause de mouvements en dehors du plan d'illumination, réduction de la densité de particules).

Convergence. Dans la Figure 3.7 nous avons illustré le fait que FOLKI converge très rapidement pour la grande majorité des pixels, mais qu'augmenter les itérations permet de corriger l'estimation en certaines zones difficiles. En 2011, Yves Le Sant a obtenu le même type de résultats sur FOLKI-PIV. Cependant, il a choisi de conserver un nombre d'itérations identique pour tous les pixels, car la gestion des tests ralentit le calcul en GPU sans que le gain ne soit vraiment important. Sur un plan théorique, Benjamin Leclaire et al. [Leclaire et al., 2011] ont proposé une interprétation de ces bonnes propriétés de convergence relativement à d'autre méthodes de déformations itératives de fenêtres précédemment publiées dans le domaine de la PIV.

Application à la PIV stéréoscopique. La PIV stéréoscopique consiste à acquérir simultanément deux images sous deux angles différents de la nappe laser. L'utilisation de deux couples d'images stéréoscopiques à deux instants différents permet d'estimer la composante hors plan du champ de déplacement, ce qui donne ce qu'on appelle un champ 2D-3C, c'est à dire un champ 2D de vecteurs à 3 composantes. Dans les logiciels de PIV commerciaux, l'estimation de la troisième composante se fait par triangulation à partir de deux estimations 2D-2C, une pour la caméra gauche et une pour la droite. En 2009, B. Leclaire et al. [Leclaire et al., 2010] ont proposé une approche en une seule étape en généralisant le critère (3.18). Le critère fait apparaître les termes correspondant à chaque caméra et dépend directement du champ de déplacement 2D-3C recherché, via des matrices de projection. Cette approche permet de conserver la structure de calcul très efficace de FOLKI et semble aussi améliorer l'estimation par rapport à l'approche traditionnelle dans certaines configurations [Leclaire et al., 2011]. La figure 3.12 présente une expérimentation de PIV TR 3C menée par Samuel Davoust à l'ONERA/DAFE en 2011.

Diffusion. Comme cela a déjà été mentionné, grâce à l'investissement de Yves Le Sant au DAFE, FOLKI-SPIV est devenu un outil de traitement de référence non seulement au département DAFE, mais dans l'ensemble de l'ONERA et en particulier dans les souffleries commerciales

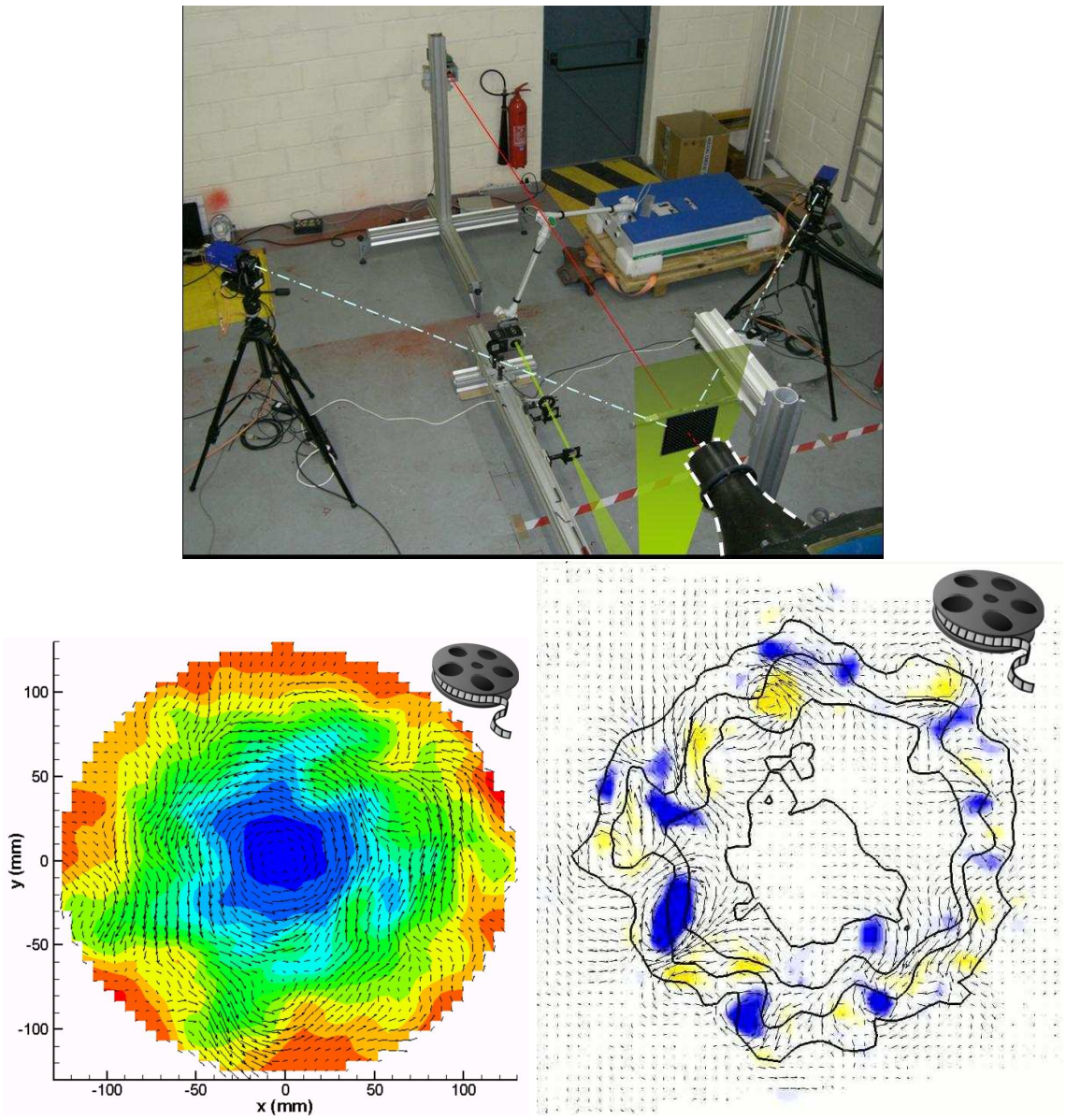


Figure 3.12 – Mesure PIV sur un jet tournant, menée par Samuel Davoust à l'ONERA/DAFE. Haut : vue du dispositif expérimental de PIV stéréoscopique. L'axe du jet est matérialisé en rouge, il s'écoule depuis la buse qu'on voit en bas qui est précédée d'un système qui donne au jet un mouvement de rotation. Le plan du lase PIV est présenté en jaune et on peut voir les angles de vue des deux caméras utilisées. Bas : images de résultat PIV obtenu par la version stéréoscopique FOLKI-SPIV, champ de vecteurs et, en couleurs, la norme à gauche et la vorticité à droite. Expérimentation, images et traitement, ONERA/DAFE.

du département GMT. Les outils de mesure et de visualisation contribuent à l'attractivité des souffleries ONERA, comme le démontre leur place importante dans le site `http://windtunnel.onera.fr`. La page 'flow-field-survey-and-visualization' mentionne ainsi les travaux menés au DTIM et au DAFE et renvoie sur les pages de description de FOLKI-SPIV au DAFE.

3.5 Comment améliorer FOLKI ?

Dans cette section, nous cherchons à améliorer la qualité des résultats fournis par FOLKI tout en conservant un coût de calcul limité. Dans un premier temps, nous soulignons les limitations de cet algorithme et nous rappelons son positionnement en termes de coût de calcul par rapport à d'autres solutions. Ensuite nous présentons les voies d'améliorations choisies, dont nous illustrons les résultats sur des données réelles, issues de diverses bases de données de comparaison ou d'acquisitions faites à l'ONERA. L'enjeu de ce travail est de chercher à savoir si les techniques rapides de recalage de fenêtre peuvent avoir un domaine d'emploi au-delà des images spécifiques issues de la PIV ou d'autres domaines expérimentaux.

3.5.1 Limitations de FOLKI.

La figure 3.13 présente le résultat de FOLKI sur l'exemple 'Army' de Middlebury (qui a été présenté en Figure 3.1). Le paramétrage de FOLKI est celui utilisé sur la version qui a été soumise en 2005 sur le site de comparaison (fenêtre gaussienne, d'écart-type 2.5 pixels) et qui a été réglé pour optimiser la performance sur l'ensemble des images d'apprentissage mises à disposition par les auteurs du site. L'exemple Army est sans doute le plus facile de la base d'évaluation de Middlebury : les images sont presque intégralement texturées (même si cela n'est pas facile à voir sur la figure), les déplacements sont très limités (4 pixels au maximum) et les zones d'occultation sont rares. Il permet néanmoins d'observer immédiatement les limites de FOLKI. La plus évidente est l'empâtement des contours des objets au premier plan, du fait de la fenêtre de calcul. L'absence de gestion des occultations entraîne des valeurs aberrantes. On voit aussi que les textures image se retrouvent dans les résultats du flot (rayures de la couverture au centre, "W" sur la boîte en bas à droite) alors que la vérité est bien uniforme dans ces zones. Ce comportement est typique des techniques de corrélation de fenêtres : l'estimation est très dépendante des caractéristiques locales de l'image dans la fenêtre, textures et niveaux de gris. A la limite, dans les zones homogènes, le conditionnement du système (3.13) explose et cela se traduit par des valeurs aberrantes de l'estimation.

Dans l'exemple Army, qui est de très bonne qualité, ces divergences sont limitées, mais

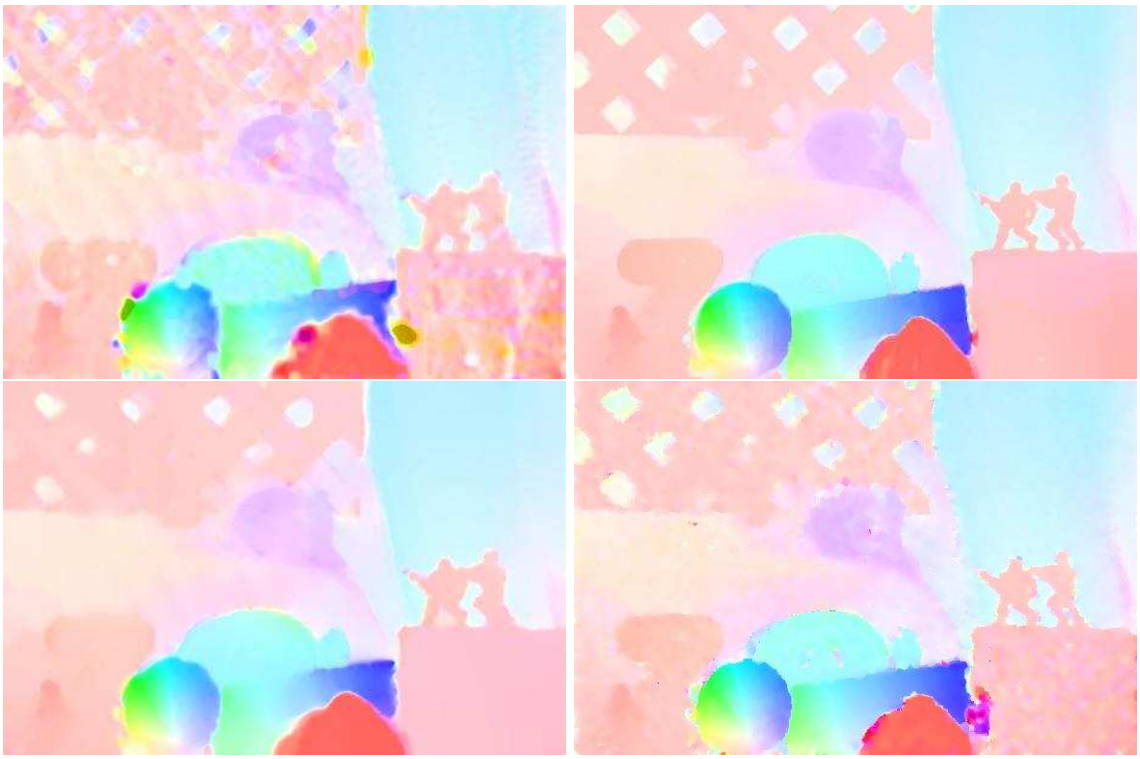


Figure 3.13 – Exemple Army du site Middlebury. L'image et la vérité terrain sont présentées en Figure 3.1. En haut, résultat FOLKI (fenêtre gaussienne, d'écart-type 2.5 pixels) et Anistropic Huber L1 (citeWerlberger10) ; en bas, résultats d'extensions de FOLKI : à gauche approche MCLG ([Fezzani, 2011]), à droite, FOLKI avec les modifications proposées dans la suite de cette section. Les images de flot utilisent le codage en couleur du site Middlebury avec un déplacement max de 5 pixels.

dans d'autres cas, elles rendent inexploitables le résultat, comme le montre l'exemple 'Mequon', tiré de la base Middlebury, et présenté en Figure 3.14. Ces cas sont malheureusement très fréquents lorsque l'on traite des données réelles, par exemple dans le contexte de la robotique terrestre de la base Kitti.

Technique globale et coût de calcul. La figure présente aussi le résultat d'une technique globale, que nous dénommerons HTVL1, publiée par Werlberger et al. [Werlberger et al., 2009], qui repose sur l'optimisation d'un critère associant un terme de recalage pixellique pénalisé en valeur absolue et un terme de régularisation anisotropique sur le gradient du flot pénalisé par une fonction de Huber [Zhang, 1997]. Cette technique a été sélectionnée parce qu'elle est disponible sur GPU. HTVL1 obtient un bon lissage adaptatif, avec des discontinuités précises et évite les explosions locales et la trop forte corrélation entre les résultats et les textures de l'image. Bien entendu, cette différence de qualité correspond aussi à une différence de temps de calcul. Sur une image de cette dimension, le temps de calcul de FOLKI est de l'ordre de 10ms. Le comparatif de Middlebury affiche un temps de calcul de l'ordre de 1.5 secondes pour HTVL1 (le temps affiché est de 2 secondes mais pour une image de taille un peu supérieure). HTVL1 a donc un coût de calcul de deux ordres de grandeur supérieur à celui de FOLKI. Notons en passant que le temps de calcul le plus faible affiché sur le comparatif Middlebury est de 120 ms, et que seules 4 références affichent des temps inférieurs à 1 seconde (pour une image 510x380) ce qui indique bien qu'encore actuellement, la question du temps réel est peu abordée.

Ajout d'un terme de régularisation. Une approche développée au DTIM par Riadh Fezzani est d'associer à un terme de recalage de fenêtre, sur le modèle de FOLKI, un terme de régularisation global. Le critère s'écrit alors :

$$J(\mathbf{u}) = \sum_{\mathbf{x}\in\Omega} D(\mathbf{x},\mathbf{u}) + \lambda \mathcal{R}(\mathbf{x},\mathbf{u}) \tag{3.20}$$

où le terme d'attache aux données est l'erreur de recalage intégrée sur une fenêtre :

$$D(\mathbf{x},\mathbf{u}) = \sum_{\mathbf{x}'\in\Omega(\mathbf{x})} \left(\mathbf{I}_{t_1}(\mathbf{x}') - \mathbf{I}_{t_2}(\mathbf{x}'+\mathbf{u}(\mathbf{x}',\theta))\right)^2 \tag{3.21}$$

et le terme de régularisation s'écrit comme nous l'avons déjà vu :

$$R(\mathbf{x},\mathbf{u}) = \Psi(\|\nabla u(\mathbf{x})\|) + \Psi(\|\nabla v(\mathbf{x})\|). \tag{3.22}$$

avec Ψ une fonction de pénalisation L1 ou L2-L1.

L'idée est d'utiliser un rayon minimal (sur des images de PIV où le rayon utilisé habituellement pour FOLKI est de 8, Riadh Fezzani utilise des rayons de 3) pour éviter le biais et de limiter la variance par l'intervention du terme de régularisation. Cette technique est appelée MCLG (*Modifed Combined Local-Global* sur le modèle de l'approche CLG originale proposée par Bruhn et al. [Bruhn et al., 2006]. Le résultat MCLG est présenté en Figure 3.13 (dernière ligne à gauche) : l'effet de la régularisation globale est net, avec un bon lissage des zones homogènes et des discontinuités bien marquées. La méthode MCLG se classe bien mieux dans le comparatif que FOLKI. Cela dit, le lissage des contours reste nettement supérieur à celui de HTVL1. Ayant choisi un critère MCLG, Riadh Fezzani a proposé plusieurs implémentations parallèles de l'optimisation, qu'il a comparé [Fezzani, 2011]. Nous avons poursuivi ces travaux en programmant sur GPU une version fondée sur un principe de "variable splitting" proche de celui décrit dans le chapitre 4 de son manuscrit [Fezzani, 2011], qu' il n'avait pu implémenter en GPU durant sa thèse. Les résultats de ce travail sont plutôt décevants. Malgré l'utilisation de diverses techniques de descente, il s'avère nécessaire d'itérer un grand nombre de fois pour obtenir un résultat satisfaisant. Le temps de calcul pour un exemple comme Army devient de l'ordre de la seconde. Ce coût de calcul nous paraît trop important pour justifier l'intérêt de cette approche, du moins dans le cadre des techniques rapides que nous souhaitons développer dans ce travail.

"Secrets of FOLKI". Dans un article remarquable titré "*Secrets of optical flow estimation and their principles*" [Sun et al., 2010], Sun et al. montraient qu'une technique d'estimation du flot optique "standard" pouvait être rendue nettement plus performante, au sens du comparatif de Middlebury, en effectuant divers filtrages souvent non linéaires, sur les données et sur le flot au cours des estimations. Dans la seconde partie de l'article ils reliaient un de ces traitements ad hoc à une modification "explicable" de leur critère de départ.

De fait, beaucoup des méthodes comparées sur le site de Middlebury ont leurs "secrets", qu'on ne découvre qu'après une lecture très attentive de la partie "mise en œuvre" de l'article et dont on peut parfois se demander si elles ne sont pas aussi importantes que la nouvelle technique d'optimisation mise en avant dans le titre... L'utilisation de ce genre d'amélioration sur FOLKI pour l'exemple Army donne le résultat présenté en Figure 3.13 en bas à droite. On constate que plusieurs défauts de l'estimation de base sont corrigés, sauf les valeurs aberrantes aux abords de l'occultation en bas à droite. De plus, la structure itérative de FOLKI reste inchangée et ces modifications aboutissent à une augmentation raisonnable du nombre d'itérations. Dans la suite de ce chapitre, nous présentons donc les améliorations du traitement FOLKI que nous avons sélectionné de manière empirique, en utilisant non seulement la base Middlebury mais aussi des images issues de contextes plus réalistes.

3.5.2 Améliorations de FOLKI

Dans la suite de cette section, nous présentons un certain nombre de traitements ajoutés à FOLKI pour améliorer les résultats. Les effets de ces améliorations sont d'abord illustrés sur des exemples issus de la base Middlebury, puis la dernière section présente l'application en pratique sur des cas plus réalistes.

Inhomogénité des images. Pour limiter la variabilité des résultats du fait des inhomogénéités de niveau et de texture des images, la plupart des techniques performantes présentées dans le comparatif de Middlebury ne calculent pas un recalage entre les images d'origine mais entre des transformées de ces images. Ces transformations visent à homogénéiser spatialement les images en conservant la texture utile pour guider l'association. La normalisation des fenêtres présentée en section 3.4.2 relève de ce genre de technique, elle est efficace pour les tailles de fenêtre suffisantes, sinon les statistiques locales sont mal estimées et le gain se réduit.

Une possibilité est d'utiliser une mesure dérivative, par exemple de recaler non seulement l'image mais le gradient de celle-ci [Brox et al., 2004, Zimmer et al., 2009, Zimmer et al., 2011]. Poussant plus loin cette logique de suppression du fond continu, la référence [Wedel et al., 2009] effectue le recalage sur des images auxquelles on retire leur version lissée par un algorithme de débruitage type Rudi-Osher-Fatemi [Rudin et al., 1992]. Cette transformation améliore en effet nettement les résultats de FOLKI sur les images Middlebury, mais son coût n'est pas compatible avec notre objectif de traitement rapide.

La transformation que nous avons sélectionnée est un simple filtrage de rang, proposé dans les années 90 [Zabih and Woodfill, 1994] et mentionné dans une étude comparative sur les mesures locales robustes en stéréovision [Brown et al., 2003]. Cette transformation, que nous appellerons Rank-n, est paramétrée par le rayon n d'un voisinage du pixel courant $\mathbf{x}$. Elle consiste à remplacer la valeur de l'intensité $I(\mathbf{x})$ par le nombre de pixels dans le voisinage de $\mathbf{x}$ ayant une intensité inférieure à $I(\mathbf{x})$. Très rapide à calculer, Rank diminue notablement la sensibilité de FOLKI à l'image, comme le montre l'exemple "Mequon" présenté à la Figure 3.14. Un autre apport, imprévu, de la transformation Rank est de réduire l'empâtement des fenêtres comme le montre la figure 3.15 ci-dessous.

Elimination des vecteurs aberrants. Une des contributions importantes de l'article de Sun et al [Sun et al., 2010] est de montrer l'efficacité (et de proposer a posteriori une justification) d'un filtrage médian, de support assez large, appliqué sur le flot optique (composante à composante) à l'issue d'une itération avant d'utiliser le flot pour réinterpoler l'image. Ce filtrage peut être rapproché des procédures de rejet de vecteurs aberrants qui sont courantes dans les logiciels

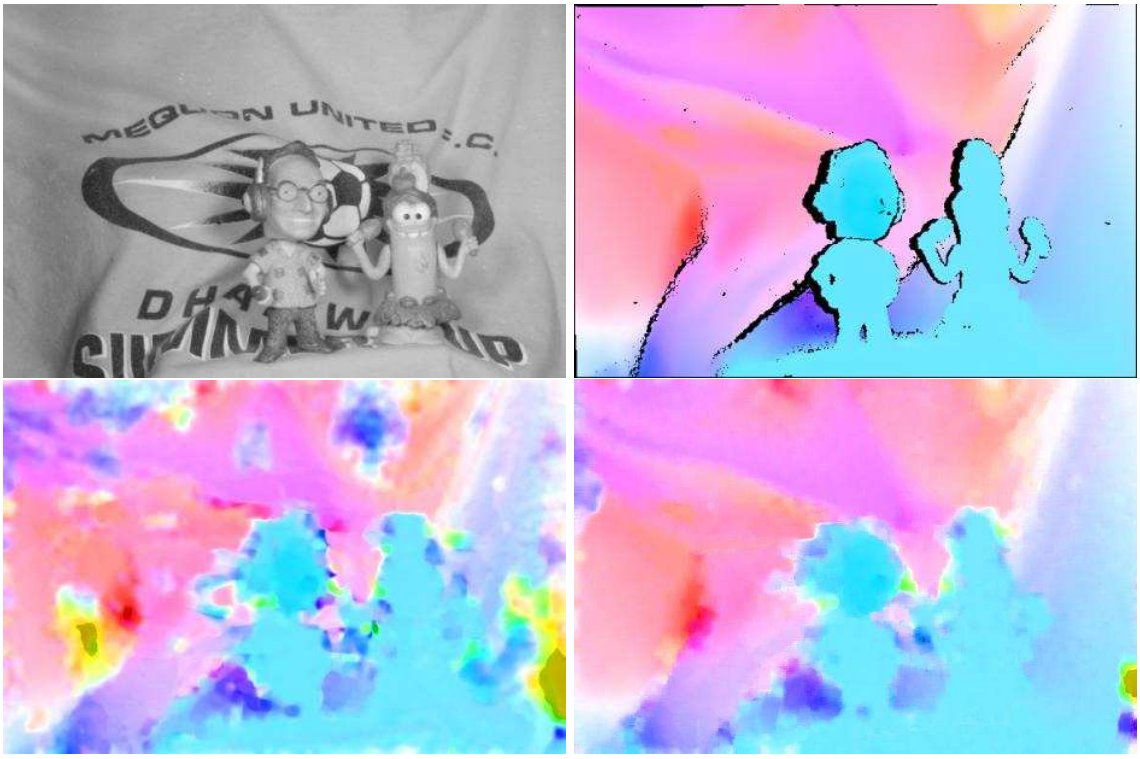


Figure 3.14 – Exemple Mequon du site Middlebury. En haut, image convertie en niveaux de gris et vérité terrain (occultations en noir) ; en bas à gauche, résultat FOLKI (fenêtre gaussienne, d'écart-type 2.5 pixels) ; en bas à droite, résultat FOLKI sur image transformée par un Rank-4 sur un voisinage de rayon 4. Les images de flot utilisent le codage en couleur du site Middlebury avec un déplacement max de 10 pixels.

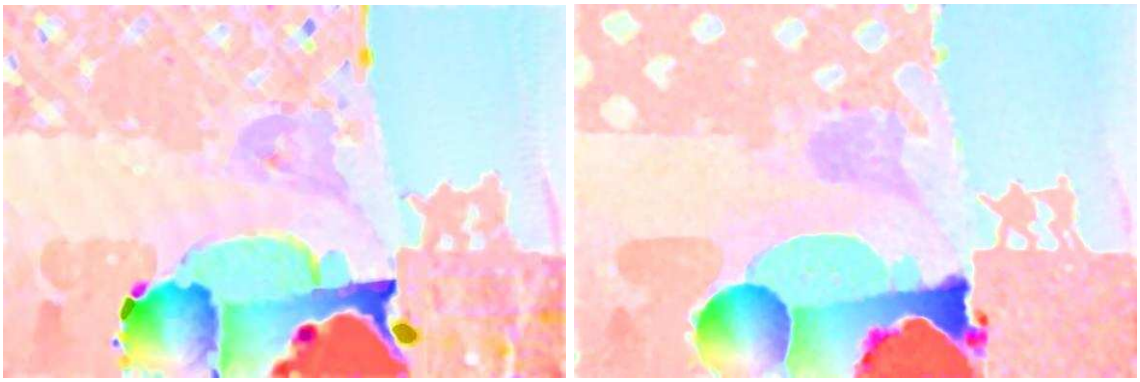


Figure 3.15 – Réduction de l'empâtement des objets par la transformée Rank. A gauche, résultat de FOLKI (fenêtre gaussienne d'écart-type 2.5) sur l'exemple Army ; à droite, même algorithme appliqué à l'image issue d'un Rank-4. Les images de flot utilisent le codage en couleur du site Middlebury avec un déplacement max de 5 pixels.

commerciaux de PIV (les vecteurs rejetés sont alors remplacés par une interpolation bilinéaire de leurs voisins). Nous avons testé de type de filtrage intermédiaire sur FOLKI. Nos résultats ne montrent pas d'apport significatif (et rejoignent des résultats similaires obtenus par Yves Le Sant sur FOLKI-SPIV). Il semble que pour un estimateur de flot relativement lissant, comme l'est FOLKI, les vecteurs aberrants isolés sont trop rares pour que leur élimination soit utile.

Adaptation de la taille de fenêtre. Nous avons observé des défauts de convergence de FOLKI dans certains exemples comprenant des mouvements importants. Ces problèmes peuvent être réglés en choisissant une taille de fenêtre plus importante, mais c'est au prix d'une perte de résolution du champ de déplacement estimé au final. L'adaptation de fenêtre consiste à ajouter une boucle de calcul à chaque niveau de pyramide pour réduire progressivement la taille de fenêtre. On profite ainsi du support étendu dans les premières itérations pour accrocher le bon déplacement, puis on affine l'estimation avec des fenêtres de petite taille. Les gains sur les exemples de Middlebury sont réduits car la plupart de ces exemples comprennent des mouvements faibles. Nous verrons que cette procédure améliore nettement les résultats sur les exemples réels de la section suivante.

Post-lissage par filtrage bilatéral. Si on utilise une adaptation de fenêtre finissant sur une valeur très petite du rayon (par exemple un rayon 2), on représente le flot à une résolution plus fine, mais on retrouve un bruit résiduel important. Ce type de procédure peut alors être suivie d'un lissage a posteriori du champ estimé. L'idée est d'utiliser un algorithme de lissage adaptatif pour éviter de relisser les discontinuités du champ : nous proposons ici un filtre bilatéral. Un filtre bilatéral [Tomasi and Manduchi, 1998] est un filtre défini sur un support relativement large, et dont la réponse (la pondération des pixels qui interviennent dans la remise à jour) dépend non seulement de la distance des pixels au pixel central, mais aussi de la différence de leurs intensités. Pour faire un filtre bilatéral sur un champ de vecteurs, on utilise une différence en norme ou en angle. Au pixel $\mathbf{x}$, le support du filtre bilatéral est noté $\mathcal{B}(\mathbf{x})$ et la pondération du pixel $\mathbf{x}' \in \mathcal{B}(\mathbf{x})$ prend la forme :

$$\omega(\mathbf{x}') \propto \exp(-||\mathbf{x}-\mathbf{x}'||^2/\sigma_{\text{dist}}) \exp(-||\mathbf{u}(\mathbf{x})-\mathbf{u}(\mathbf{x}')||^2/\sigma_{\text{norme}}) \exp(-(\text{Arg}\mathbf{u}(\mathbf{x})-\text{Arg}\mathbf{u}(\mathbf{x}'))^2/\sigma_{\text{angle}}) \tag{3.23}$$

Le filtre bilatéral est ensuite un simple filtre moyenneur, pondéré par ces coefficients, appliqué à chaque composante du champ :

$$u_{\text{out}}(\mathbf{x}) \propto \sum_{\mathbf{x}' \in \mathcal{B}(\mathbf{x})} \omega(\mathbf{x}') u(\mathbf{x}') \tag{3.24}$$

$$v_{\text{out}}(\mathbf{x}) \propto \sum_{\mathbf{x}' \in \mathcal{B}(\mathbf{x})} \omega(\mathbf{x}') v(\mathbf{x}') \tag{3.25}$$

On comprend l'intérêt de la pondération : elle permet de sélectionner dans le voisinage étendu considéré (dépendant de σ_{dist}) les pixels pour lesquels le déplacement est similaire à celui du pixel central au sens de la similarité des caractéristiques calculées dans les exponentielles. Un défaut est par contre que le filtre obtenu n'est plus séparable, ce qui le rend plus cher à appliquer. Notons que nous aurions pu appliquer un filtre médian, mais le filtre médian n'est pas vraiment une structure rentable sur un GPU, car les tris locaux sont coûteux et délicats à programmer.

Le résultat de la conjugaison d'une fenêtre adaptative et d'un post-filtrage bilatéral est présenté en Figure 3.16. Sur ces résultats, on voit que la fenêtre adaptative (en bas à gauche) permet d'obtenir une reconstruction sans divergences et avec une bonne résolution, un compromis qu'on n'atteint pas avec les solutions à fenêtre fixe (ligne du milieu). De plus, un post-lissage bilatéral permet de réduire significativement la variance du flot estimé, cf. résultat en bas à droite.

Critère aller/retour. En pratique, on possède rarement une vérité terrain pour pouvoir évaluer la fiabilité de l'estimation du flot optique. Le critère de fiabilité doit être si possible indépendant de la valeur du résidu de recalage que FOLKI minimise en chaque pixel. Une approche proposée par plusieurs auteurs [Sand and Teller, 2008, Zhao and Sawhney, 2002], consiste à utiliser un critère "aller/retour", consistant à comparer les flots estimés dans un sens et dans l'autre. La carte d'erreur aller/retour s'écrit :

$$ar(\mathbf{x}) = ||\mathbf{u}(\mathbf{x}) + \mathbf{u}_r(\mathbf{x} + \mathbf{u})||, \tag{3.26}$$

où $\mathbf{u}_r$ est le flot retour (calculé en sens inverse de $\mathbf{u}$). En utilisant un simple seuil (en pixels) sur cette carte on peut détecter des zones où l'estimation du mouvement a échoué. Le résultat de ce critère est présenté en Figure 3.17 pour l'exemple 'Schefflera' de Middlebury. Les zones détectée par le seuillage de la carte d'erreur aller/retour (pour un niveau de seuil de 0.5 pixel) recouvrent assez bien les zones où le flot est aberrant, qui sont très étendues pour l'algorithme FOLKI de base (ligne du milieu) : la densité du champ de vecteur estimé n'est plus que de 73.5%. Pour la version améliorée, ce pourcentage passe à 91.4% et on voit que le critère aller/retour commence à agir comme un détecteur des zones d'occultations (indiquées en noir dans la vérité terrain), un fait bien connu en stéréovision.

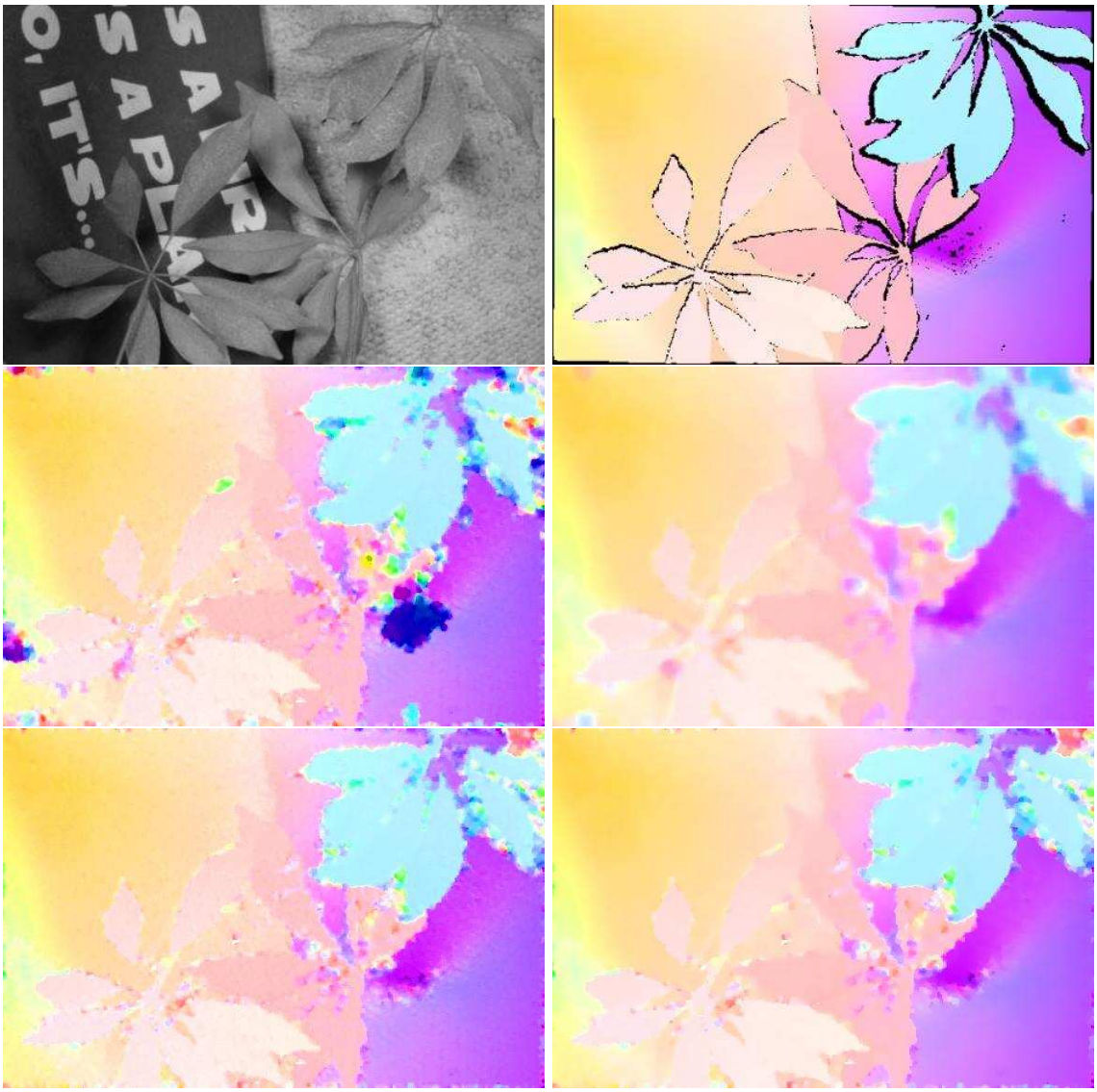


Figure 3.16 – Exemple Schefflera du site Middlebury. En haut, image convertie en niveaux de gris et vérité terrain (occultations en noir). Tous les traitements sont effectués après une transformation Rank-4. Au milieu, résultat FOLKI pour 2 valeurs du rayon : 2 (à gauche) et 5 (à droite). En bas, FOLKI à fenêtre adaptative de rayon décroissant de 8 à 2 par pas de 2. A gauche sortie brute, à droite après filtrage bilatéral. Les images de flot utilisent le codage en couleur du site Middlebury avec un déplacement max de 10 pixels.

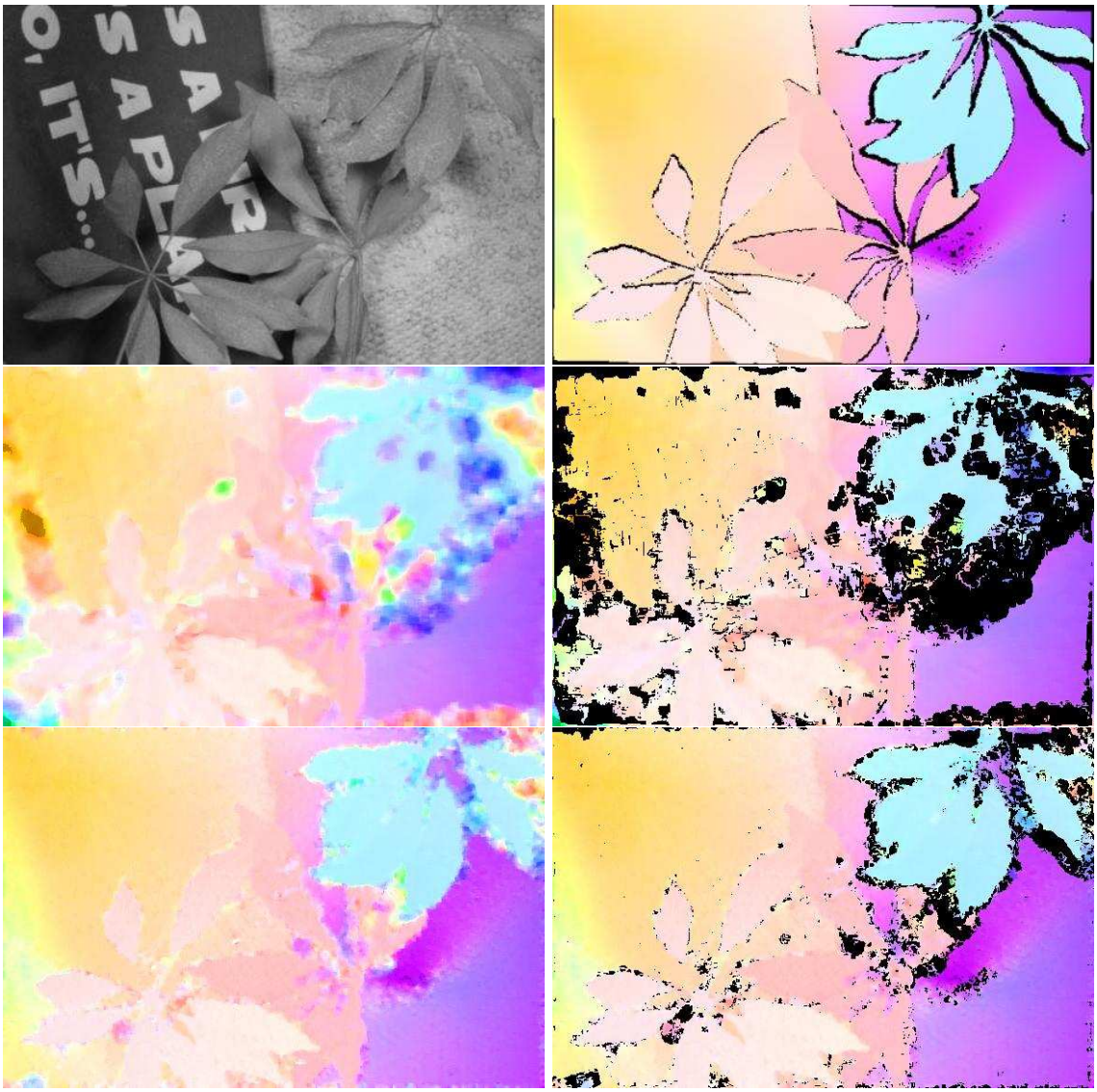


Figure 3.17 – Exemple Schefflera du site Middlebury. En haut, image convertie en niveaux de gris et vérité terrain (occultations en noir). Au milieu, à gauche, flot estimé par FOLKI (fenêtre de rayon 4), à droite flot estimé et zones détectées par le critère aller/retour, seuillé à 0.5 pixel. En bas, à gauche, flot estimé par FOLKI amélioré (fenêtre adaptative de rayon 8 à 2 par pas de 2 et préfiltrage Rank-4), à droite, flot estimé et zones détectées par le critère aller/retour, seuillé à 0.5 pixel. Les images de flot utilisent le codage en couleur du site Middlebury avec un déplacement max de 10 pixels.

3.5.3 Application à des données réelles

Nous vérifions ici que les améliorations illustrées sur Middlebury conduisent effectivement à des gains sur des images "opérationnelles". Nous commençons par montrer un gain quantitatif sur un sous-ensemble de la base de données Kitti [Geiger et al., 2012], puis nous évaluons une sélection d'algorithmes sur l'ensemble de la base. Dans la section suivante, nous présentons un gain qualitatif sur un certain nombre d'images réelles représentatives de différents cadres d'application.

Comparaison quantitative des versions de FOLKI sur images Kitti

Nous avons déjà présenté ce site d'évaluation d'outils de vision sur des données issues de campagnes d'acquisition en milieu urbain par une voiture équipées de divers capteurs (http ://www.cvlibs.net/datasets/kitti/). Le jeu de données pour le flot optique comprend plusieurs centaines de couples d'images successives et la vérité terrain du flot pour un sous-ensemble des pixels (environ 100000 pixels sont renseignés sur une image), ce qui permet d'apprendre le réglage de sa méthode avant de la soumettre pour une évaluation impartiale sur le site.

Dans cette section, nous avons d'abord utilisé un sous-ensemble de 16 images de cette base pour choisir les paramètres des améliorations de FOLKI que nous avons proposées en section précédente, puis nous avons calculé les scores d'erreur sur l'ensemble de la base Kitti pour juger quantitativement de l'apport de ces améliorations.

L'ensemble des paramètres des différentes versions testées sont présentés dans la table 3.1. Essentiellement, la version FOLKI1 utilise une fenêtre de rayon 4 sans préfiltrage, la version FOLKI2 utilise un préfiltrage 'Rank-4' et une fenêtre adaptative de rayons $\{12, 8, 4\}$, FOLKI3 descend jusqu'à un rayon minimal de 2 et FOLKI4 reprend le résultat de FOLKI3 et ajoute un filtrage bilatéral. Les temps de calcul moyens sur GPU sont présentés en dernière ligne sur une image 1241×376 (format de la base Kitti). On utilisera parfois une version aller/retour, avec un niveau de seuil de seuil qui sera précisé.

L'outil d'évaluation du site Kitti consiste, pour chaque pixel renseigné dans la vérité terrain, à calculer la distance en pixel entre les points d'arrivée des vecteurs flots estimés et exacts. Les histogrammes de ces distances sont calculés ainsi que des cartes de chaque classe. Dans le comparatif que nous effectuons, nous choisissons un score scalaire qui est le pourcentage d'erreurs supérieures à 3 pixels (c'est l'indice principal choisi sur le site Kitti). La table 3.2

	FOLKI1	FOLKI2	FOLKI3	FOLKI4
Rayon min	4	4	2	2
Rayon max	4	12	12	12
Pas rayon		4	2	2
Rank (0 sinon)	0	4	4	4
Nb itérations	2	10	10	10
Nb niveaux	4	4	4	4
Marge	0	0	10	10
Post-filtrage	Non	Non	Non	Bilatéral
coût (ms)	6	50	100	210

Table 3.1 – Paramètres des différentes versions de FOLKI comparées et coût de calcul moyen sur GPU (image 1241×376). Le paramétrage du filtrage bilatéral est le suivant : Rayon$= 4$; $\sigma_{\text{dist}} = 2$; $\sigma_{\text{norme}} = 0.2$; $\sigma_{\text{angle}} = 0.2$.

présente les scores obtenus par les 4 méthodes sur les 16 images d'apprentissage.

Les modifications proposées de FOLKI permettent d'améliorer très nettement les résultats par rapport à la version "classique" (FOLKI1), sur toutes les images, sauf pour FOLKI2 sur l'image numéro 20. Cette image, visible en Figure 3.18, présente à droite une zone mal éclairée et sur laquelle on a un fort effet de bougé. Sur cette zone le préfiltrage Rank-4 supprime trop d'information et dégrade l'association.

Cependant, en général, le préfiltrage Rank-4 permet d'améliorer nettement le résultat (par exemple dans la zone mieux contrastée à gauche de la même image). Si l'on revient au comparatif en Table 3.2, la version FOLKI3, qui utilise un rayon minimal de fenêtre de 2, présente un score un peu meilleur que celui de FOLKI2, mais un aspect plus bruité. Cet aspect peut être corrigé par le filtre bilatéral, mais le gain en termes de score est très faible.

En conclusion, la version FOLKI2, qui intègre donc simplement le filtre Rank-4 et une taille de fenêtre adaptative, semble le meilleur compromis qualité sur coût de calcul. Les figures 3.19 et 3.20 présentent en images le gain typique de FOLKI2 sur FOLKI1. Sur la Figure 3.19 on représente, de haut en bas, une des deux images du couple traité, le résultat obtenu par FOLKI1, la vérité terrain et l'erreur commise par FOLKI1. Les couleurs de la carte d'erreur correspondent aux intervalles de l'histogramme des erreurs présenté en dessous. Le pourcentage d'erreurs au-dessus de 3 pixels est de 52.2% sur cet exemple.

On constate que FOLKI1 produit des zones "d'explosion", dues à une texture trop faible, à des variations de l'illumination ou à du flou de bougé et aussi qu'il sous-estime les grands déplacements dans le bas et sur les bords du champ. Le même exemple est traité par FOLKI2

Numéro	FOLKI1	FOLKI2	FOLKI3	FOLKI4
172	58.65	47.48	46.62	46.17
74	89.64	56.72	59.47	59.08
11	53.73	32.87	32.45	31.76
12	36.32	17.68	15.80	15.27
13	19.49	11.37	11.41	10.98
14	46.59	31.93	31.65	31.16
17	52.23	19.38	18.57	17.83
20	45.05	48.87	45.74	45.20
34	34.55	18.41	18.00	17.54
36	13.86	8.46	8.71	8.21
39	64.10	18.12	19.59	19.04
99	37.79	18.72	13.94	13.84
100	27.54	7.58	7.24	6.44
101	70.88	53.14	51.86	51.63
106	22.10	7.61	5.88	5.40
108	37.11	21.44	17.42	17.14
	moyenne			
	44.35	26.24	25.27	24.79

Table 3.2 – Comparaison des différentes versions de FOLKI sur 16 images de la base Kitti. Le score est le pourcentage des erreurs supérieures à 3 pixels. La dernière ligne présente le score moyen.

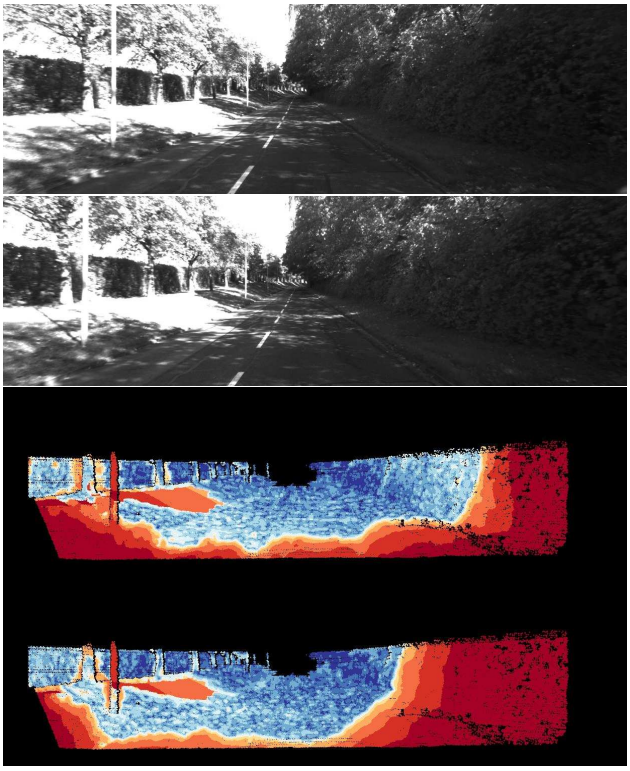


Figure 3.18 – Couple d'images numéro 20 de la base Kitti (en haut) et comparaison des cartes d'erreur obtenues avec FOLKI1 et FOLKI2 (en bas). Les cartes d'erreur utilisent une échelle de couleur associée aux cases de l'histogramme présenté en Fig. 3.19.

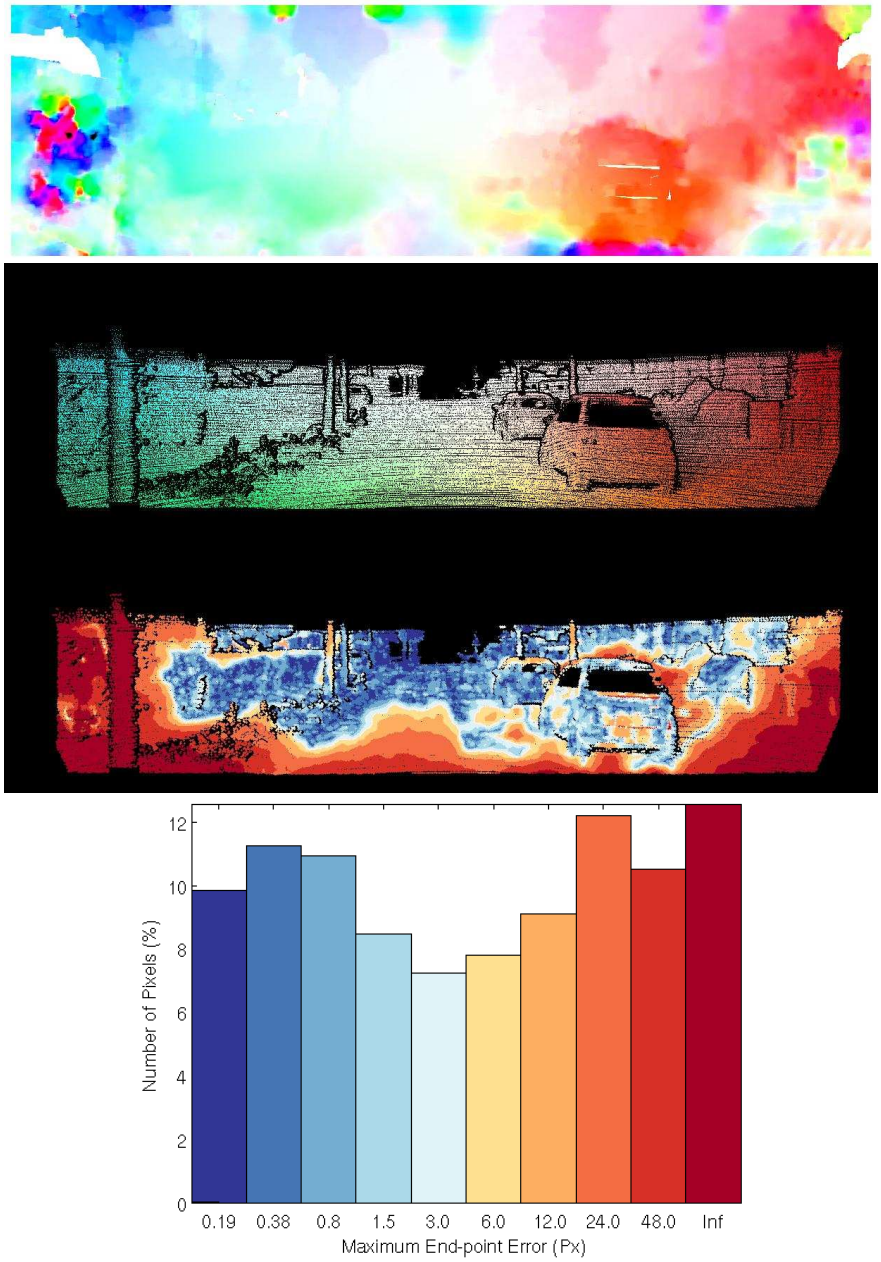


Figure 3.19 – Résultat FOLKI1 sur image 17 de la base d'apprentissage de Kitti. De haut en bas : une des deux images du couple traité ; resultat FOLKI1 et vérité terrain (même échelle de couleur) ; carte d'erreur avec une échelle de couleur associée aux case de l'histogramme présenté en dessous.

et le résultat est à la Figure 3.20. Le gain apporté par les améliorations (filtre de rang et taille de fenêtre adaptative) est spectaculaire et se traduit dans le score (pourcentage d'erreurs à plus de 3 pixels) qui n'est plus que de 19.4%. Les zones d'erreurs dans le résultat de FOLKI2 peuvent être attribuées à des occultations (bords des véhicules, poteaux) et aux effets de bords (sous-estimation du mouvement, fort effet de bougé sur les objets proches comme le poteau situé à gauche).

Evaluation sur l'ensemble de la base d'apprentissage Kitti

Finalement, nous avons lancé la comparaison entre FOLKI1 et FOLKI2 sur l'ensemble de la base Kitti, qui comprend 200 instants d'acquisitions (de 2 couples d'images stéréoscopiques). Les pourcentages à 3 pixels obtenus par les deux méthodes sont représentés sur la Figure 3.21. On constate que FOLKI2 améliore le score sur l'ensemble des images. Les scores moyens sont de 36.4% pour FOLKI1 et 18.8% pour FOLKI2, soit un gain moyen de près de 20%.

Ayant défini une version efficace de FOLKI, nous effectuons une évaluation systématique selon les critères du site Kitti. Les résultats sont présentés en table 3.3, en regard de la méthode TGV2CENSUS [Werlberger, 2012].

Version	data	Out-Noc	Out-All	Avg-Noc	Avg-All	density	runtime
FOLKI1	training	36.38	43.90	10.66	17.91	100.00	6ms
FOLKI2	training	18.82	27.62	4.80	9.68	100.00	50ms
FOLKI2 A/R	training	10.97	14.52	2.59	4.16	75.65	100ms
TGV2CENSUS	eval	11.14	18.42	2.9	6.6	100	4sec

Table 3.3 – Evaluation des scores d'erreur Kitti et des temps de calcul pour les méthodes FOLKI1, FOLKI2 FOLKI2 aller/retour (rejet à 3 pixel) et TGV2CENSUS [Werlberger, 2012] (données extraites du site Kitti). Out-Noc : pourcentage de vecteurs erronés (3 pixels) dans les zones non-occultées ; Out-All : pourcentage de vecteurs erronés sur le total ; Avg-Noc : erreur moyenne sur les pixels non occultés ; Avg-All : erreur moyenne.

Selon l'indice principal de classement du site Kitti (Out-Noc à 3 pixels), FOLKI2 se classe en 4ème position, derrière la méthode TGV2CENSUS (ce classement au 25/11/12 est rappelé en Annexe B). Attention cependant la comparaison n'est pas exacte car nous calculons les scores de FOLKI sur la base d'apprentissage ('data training' dans la table 3.3) alors que les scores affichés sur le site concernent la base d'évaluation ('eval'). En passant sur la base d'évaluation les scores de FOLKI seront différents, mais sans doute proches car la base d'apprentissage

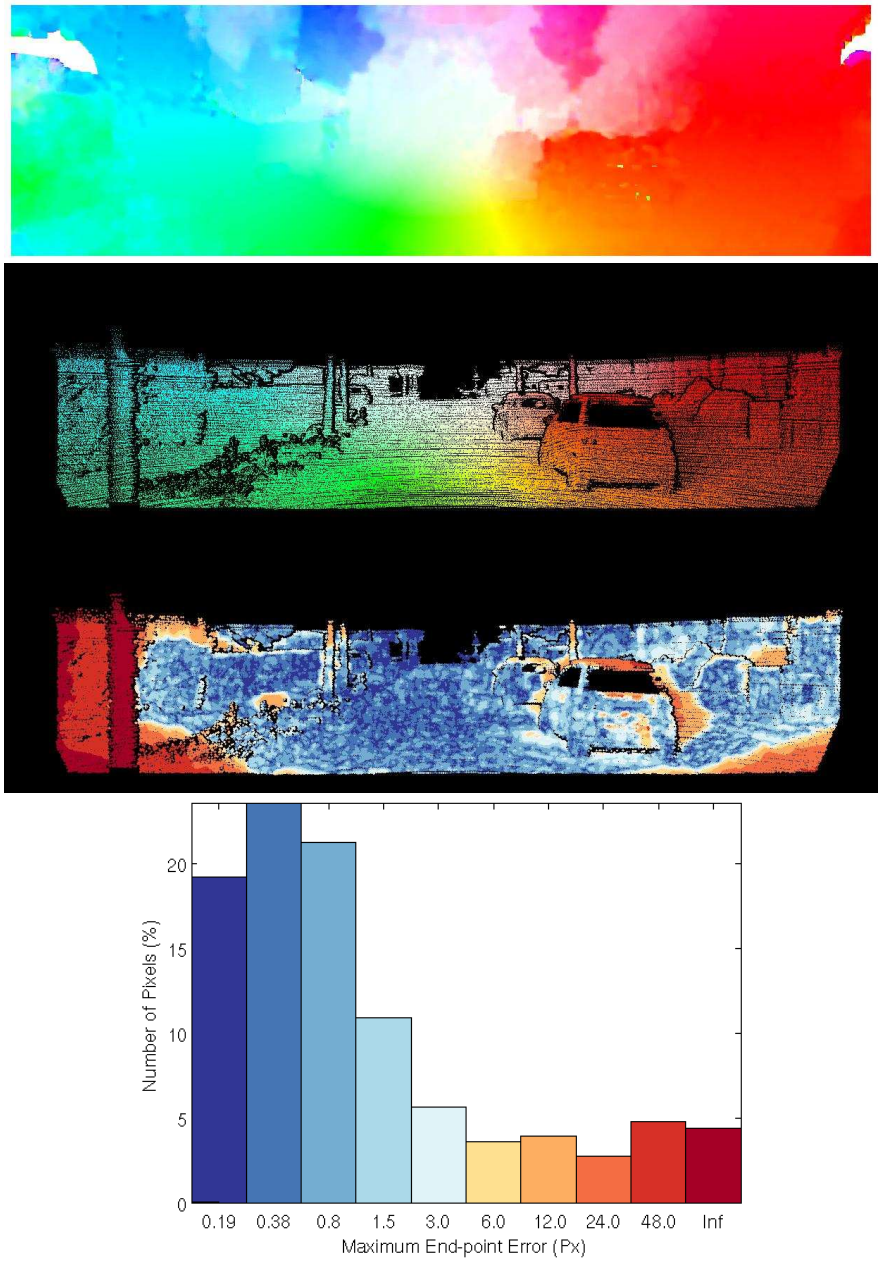


Figure 3.20 – Résultat FOLKI2 sur image 17 de la base d'apprentissage de Kitti. De haut en bas : une des deux images du couple traité ; resultat FOLKI1 et vérité terrain (même échelle de couleur) ; carte d'erreur avec une échelle de couleur associée aux cases de l'histogramme présenté en bas.

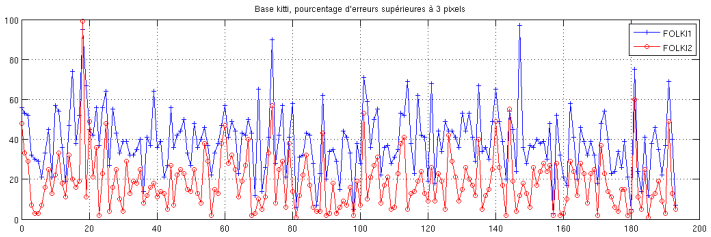


Figure 3.21 – Pourcentage de pixels pour lesquels l'erreur dépasse 3 pixels sur la base d'apprentissage Kitti (200 images). Bleu : score FOLKI1, rouge : score FOLKI2. L'erreur est la distance en pixel entre les points d'arrivés des vecteurs flot estimé et vérité-terrain

est sensée être représentative. On peut donc penser que FOLKI2 se classera dans le premier tiers du tableau.

On peut aussi remarquer que la version avec vérification par flot aller-retour de l'estimateur de mouvements FOLKI2 permet d'améliorer nettement la performance, avec cependant une densité de vecteurs plus faible. Le gain se situe surtout du côté de l'erreur sur tous les pixels, ce score étant même meilleur que celui de TGV2CENSUS, ce qui indique encore une fois que le critère aller/retour est un bon détecteur des zones d'occultations.

La comparaison des temps de calcul est difficile, car les temps affichés sur le site Kitti sont obtenus avec des machines et des langages (C++, Matlab) différents. Néanmoins la plupart des méthodes affichent des temps de calcul beaucoup plus élevés, de l'ordre de la minute, et seules trois méthodes ont des temps de calcul de l'ordre de la seconde (TGV2CENSUS est à 4 sec.), alors que FOLKI2 tourne en 50 ms. Même si les temps de calcul des méthodes concurrentes pourraient être réduits en passant sur GPU, au moins peut-on dire que pour FOLKI2 ce travail est déjà fait et qu'il s'agit donc actuellement de la seule méthode à cadence vidéo qui atteigne ce niveau de score.

Kitti stéréoscopique

Les capteurs embarqués de Kitti comprennent un capteur stéréoscopique et une vérité terrain donnée par un scanner laser. Le site propose aussi un outil d'évaluation de solution stéréoscopique. Nous proposons d'utiliser directement FOLKI (version 1 et 2) sur ces images, sans aucun changement paramétrique. Notons qu'en théorie on pourrait utiliser une version 1D de FOLKI dans laquelle la recherche se fait uniquement sur l'axe horizontal, car les images sont rectifiées. Cette contrainte épipolaire permet de restreindre l'espace de recherche et conduit en général à des temps de calcul plus faibles et à diminuer les ambiguités d'association. En utilisant

une simple projection 1D des estimations données par FOLKI, nous n'avons pas constaté de gain significatif. Notons que d'une part, la structure algorithmique par descente itérative ne profite pas énormément du passage à une recherche 1D, d'autre part, en pratique, la rectification est une opération numérique avec une précision limitée et la recherche doit se faire plutôt sur un "ruban épipolaire" plutôt que réellement sur une droite. Conserver une évolution libre de la recherche permet d'éviter de gérer ces problèmes d'imprécision de rectification.

La figure 3.22 présente un exemple assez difficile, où la disparité (ie. la composante horizontale du déplacement dans ce cadre rectifié) atteint 139 pixels sur la vérité terrain. L'estimation de FOLKI1 est quasiment inexploitable avec plus de 70% des pixels pour lesquels la disparité est à plus de 3 pixels de la disparité réelle. FOLKI2 améliore très nettement les choses, avec une reconstruction globale de la scène correcte, mais le taux d'erreur reste élevé à 55.78%.

La table 3.4 présente les scores évalués sur l'ensemble de la base d'apprentissage stéréoscopique Kitti avec les outils du site, performances comparées pour les codes FOLKI1 et FOLKI2. Le gain apporté par FOLKI2 est toujours très important. Cependant on noter que la performance obtenue est moins bonne comparée aux autres techniques comparées sur le site Kitti, puisqu'une bonne dizaine de méthodes ont des pourcentages d'erreur sous les 10%. Cela signifie que ces méthodes font un meilleur usage de la contrainte épipolaire que nous et qu'il nous faudrait réfléchir à une version plus élaborée de FOLKI pour le cas 1D. Notons par ailleurs qu'en moyenne les déplacements stéréoscopiques sont plus élevés que ceux observés dans les données de flot optique Kitti. D'autre part, il est possible que l'effet du flou de bougé soit aussi plus gênant en stéréoscopie qu'en flot optique. Dans le cas d'une translation, ce flou est en effet variable dans le champ en fonction de la distance des objets à la caméra : les déformations sont importantes et directionnelles dans le bas de l'image ce qui peut gêner le recalage de fenêtre dans ces zones. Le filtrage Rank améliore un peu ce problème (on le verra à nouveau dans les données Oxford en section suivante) mais il reste un taux d'associations fausses important dans ces zones.

Version	data	Out-Noc	Out-All	Avg-Noc	Avg-All	density	runtime
FOLKI1	training	55.45	56.43	19.85	21.00	100.00	6ms
FOLKI2	training	19.42	20.99	4.75	5.15	100.00	50ms
FOLKI2 A/R	training	12.12	12.67	2.45	2.56	82.50	100ms

Table 3.4 – Evaluation des scores d'erreur sur les données stéréoscopiques Kitti. Le seuil sur les vecteurs erronés est de 3 pixels. Out-Noc : pourcentage de vecteurs erronés dans les zones non-occultées ; Out-All : pourcentage de vecteurs erronés sur le total ; Avg-Noc : erreur moyenne sur les pixels non occultés ; Avg-All : erreur moyenne.

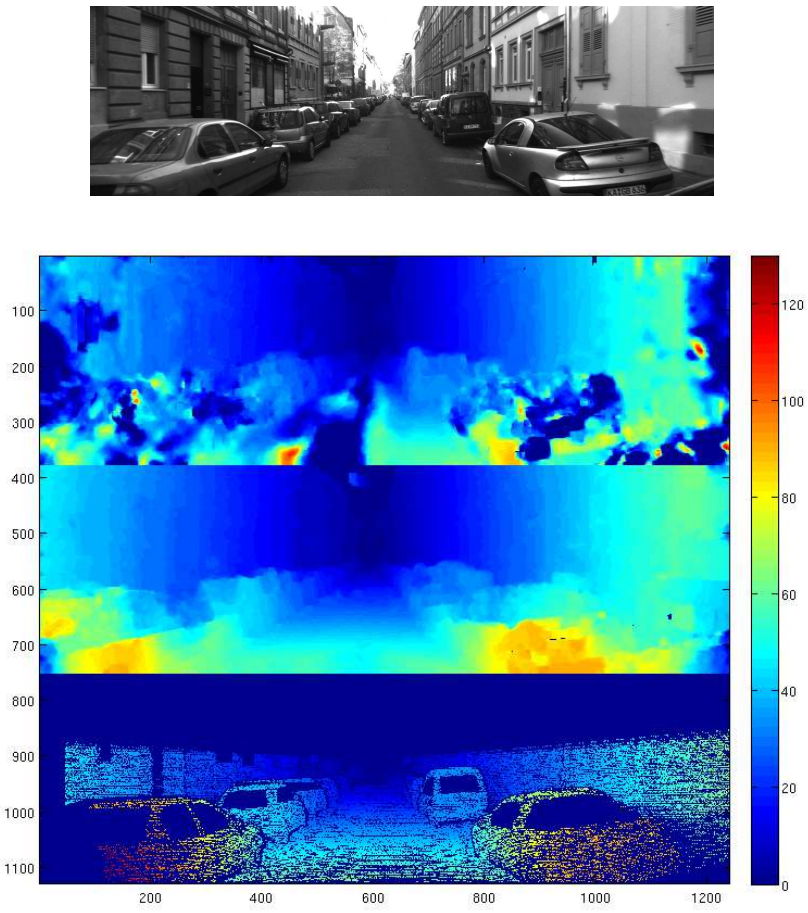


Figure 3.22 – Exemple stéréoscopique numéro 172 de la base Kitti. En haut : une image du couple stéréoscopique. En dessous, disparité estimée par FOLKI1 (haut), FOLKI2 (milieu) comparés à la vérité terrain (bas).

Quelques comparaisons qualitatives sur données réelles

Séquence Oxford L'université d'Oxford a mis en ligne des données de robotique terrestre (cf. le site http ://www.robots.ox.ac.uk/NewCollegeData/) et la publication [Smith et al., 2009]) issues de leur robot Lisa. Il s'agit d'un robot à roue dérivé d'une plate-forme Segway et équipé (entre autres) d'une tête stéréoscopique Point Grey Bumbleebee qui fournit des couples d'images 512x384 à 20Hz. La figure 3.23 présente au même instant les résultats du traitement stéréoscopique et du traitement par flot optique (avec l'image à l'instant suivant). Comme pour Kitti, les deux couples d'images (stéréoscopique ou temporel) sont traités avec exactement les mêmes algorithmes FOLKI1 et FOLKI2. FOLKI2 améliore nettement la cohérence de ces deux résultats. Comme pour Kitti on peut remarquer que le cadre stéréoscopique est plus difficile à cause de déplacements plus élevés et de l'effet du filé.

Séquence de vidéo-surveillance (PETS07) Le couple d'image en haut de la Figure 3.24 est issue d'une séquence video de la banque de données PETS 2007 (http ://pets2007.net/). Comme souvent dans le contexte de la video-surveillance, les images apparaissent d'assez mauvaise qualité avec un éclairement faible et un niveau de bruit important. Dans ce cadre FOLKI2 améliore l'estimation des grands mouvements et la segmentation des objet mais présente un bruit plus important, visible dans le fond fixe des images. Le filtrage Rank améliore la robustesse mais diminue le moyennage du bruit dans les zones de déplacement uniforme.

Séquences aériennes Le recalage de séquence aérienne est utile pour la raconstruction 3D, la détection d'objets mobiles, cf. section 3.6 et aussi pour la superrésolution au chapitre 4. La figure 3.25 présente un exemple sur une séquence aérienne 780×1280 sur la ville de Dubai, où les effets stéréoscopiques sont très importants du fait de la hauteur des immeubles. Le flot estimé par FOLKI1 est très dégradés par des effets de bord et par la saturation de l'intensité sur la tour à gauche. FOLKI2 corrige ces défauts et fournit un flot de bonne qualité.

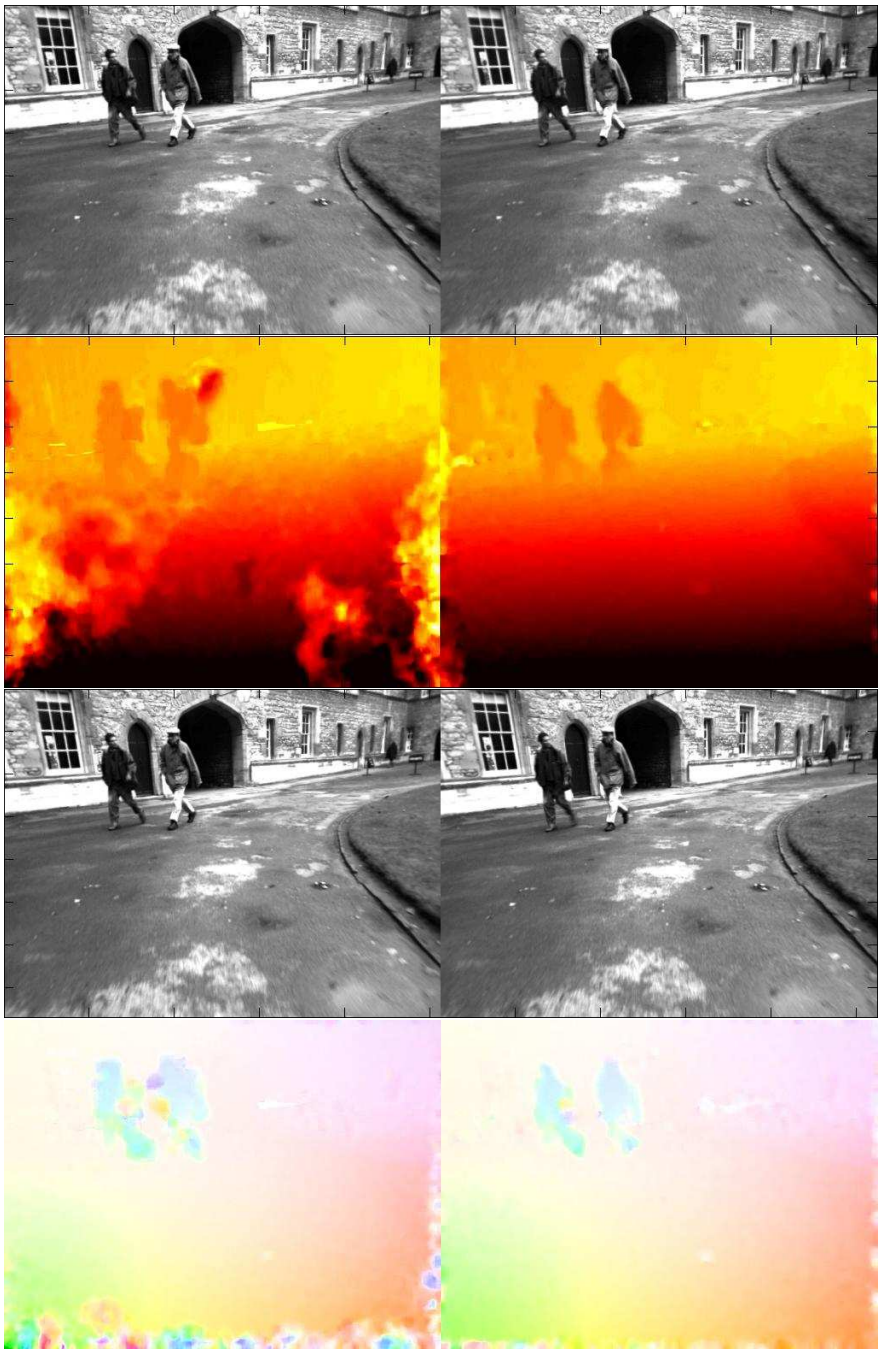


Figure 3.23 – Séquence de robotique terrestre "Oxford". De haut en bas : images stéréoscopiques et déplacement horizontal estimé par FOLKI1 (à gauche) et FOLKI2 (à droite), colormap 'hot' de Matlab linéaire entre les valeurs -10 et 30. Images successives de la caméra gauche et flots estimés par FOLKI1 (gauche) et FOLKI2 (droite), colormap de Middlebury, le flot maximal est de 18 pixels.

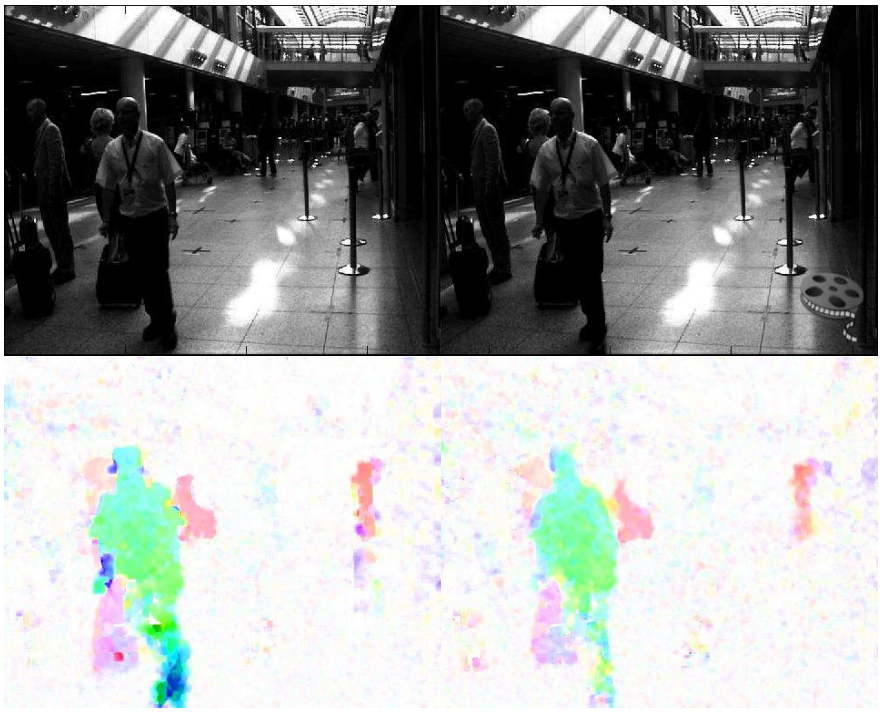


Figure 3.24 – Séquence de vidéo-surveillance (PETS07). Haut : images ; Bas : flots estimés par FOLKI1 (à gauche) et FOLKI2 (à droite). La table de couleur est celle de Middlebury, le flot maximal affiché est de 20 pixels.

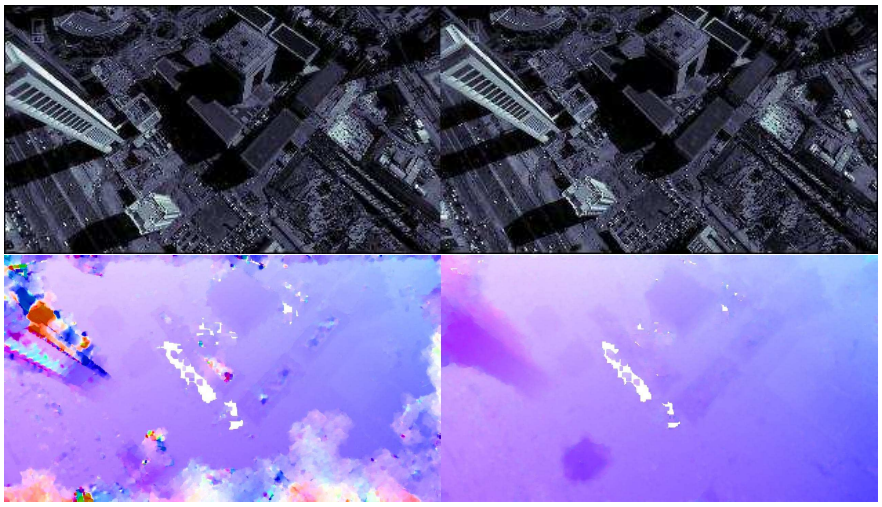


Figure 3.25 – Séquence aérienne "Dubai". Haut : images ; Bas : flots estimés par FOLKI1 (à gauche) et FOLKI2 (à droite). La table de couleur est celle de Middlebury, le flot maximal affiché est de 70 pixels.

3.6 Recalage paramétrique et détection

Nous avons vu dans la section introductive une décomposition du flot optique (3.3) en une somme de composantes associées aux mouvements des objets mobiles et une composante liée au mouvement propre de l'observateur et dépendant de la structure 3D de la scène. Dans le cas de la vidéo aérienne, sur lequel a porté une part importante de nos travaux, la partie du flot associée au mouvement propre de l'observateur est souvent rendue plus simple par l'hypothèse que la scène survolée est plane — on parlera alors de flot associé au mouvement apparent du sol. Cette hypothèse est souvent approximativement vraie, cela dépend du rapport de la hauteur des super-structures à l'altitude de vol et de la résolution du capteur : la Figure 3.25 présente un contre-exemple. Lorsque ces quantités sont telles que les mouvements de parallaxe sont de l'ordre d'un pixel ou moins, l'hypothèse plane peut être considérée comme valide. Dans ce cas, le mouvement propre de l'observateur génère un flot paramétrique qui suit un modèle d'homographie plane et qu'on peut approximer par un polynôme d'ordre inférieur ou égal à deux. Au sujet de ces modèles paramétriques de mouvement, on peut consulter le travail de Odobez et Bouthemy qui a donné le logiciel de recalage Motion2D [Odobez and Bouthemy, 1995], et la référence [Jonchery et al., 2008] qui étudie un paramétrage quadratique alternatif à celui de Odobez et Bouthemy.

Dans le cas des vidéos aériennes, même dans le cas où le modèle plan n'est pas valide sur tout le support image, il est souvent *dominant*, c'est-à-dire qu'il s'applique pour une majorité des pixels de l'image (c'est le cas de la Figure 3.25). Ce type de situation a conduit au développement de méthodes de recalage du mouvement dominant ou majoritaire entre deux images, dont le logiciel Motion2D [Odobez and Bouthemy, 1995] est un exemple important. Ces méthodes utilisent un modèle de mouvement paramétrique, une technique d'association ou de mise en correspondance des images et une méthode robuste permettant le rejet des régions qui ne suivent pas le modèle dominant (objets mobiles en particulier).

3.6.1 Approche par primitives ou approche dense ?

A l'heure actuelle, la plupart de ces méthodes utilisent une technique d'association de primitives : association de points de Harris [Harris and Stephens, 1988] par corrélation aller/retour, association de points SIFT [Lowe, 2004] par leurs descripteurs, etc. Le mouvement est ensuite estimé pour maximiser le nombre de couples associés par un algorithme robuste tel que Ransac [Fischler and Bolles, 1981]. Bien que ces chaînes algorithmiques puissent être implémentées sur des architectures massivement parallèles (il existe ainsi une implémentation GPU de SURF, une version rapide de SIFT) elles ne nous semblent pas forcément les plus efficaces

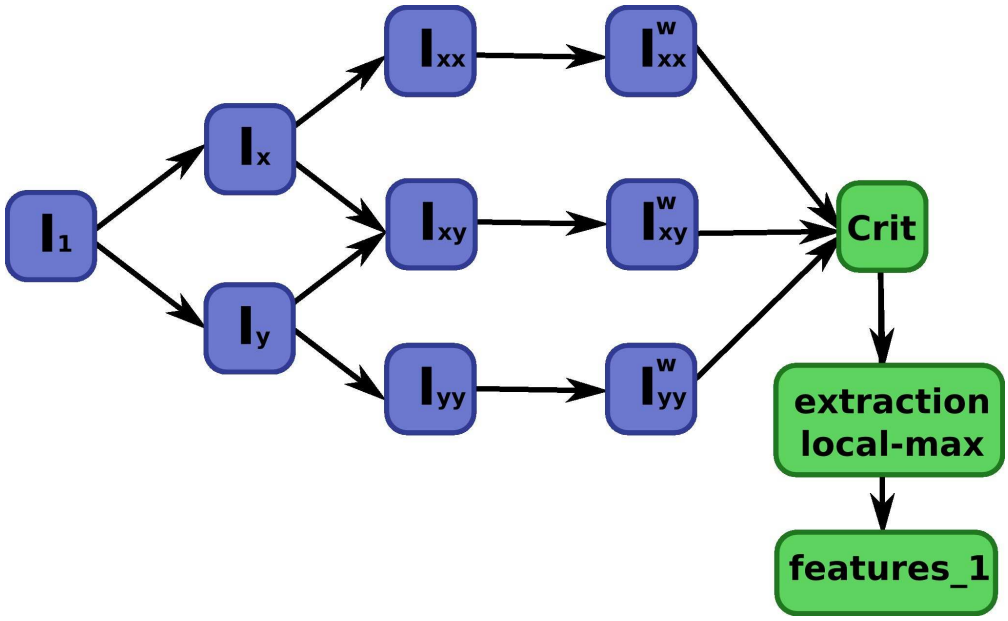


Figure 3.26 – Schémas de l'extracteur de points de Harris, en bleu les parties communes avec FOLKI.

pour résoudre le problème de calcul du mouvement paramétrique.

La première raison est que l'algorithme d'extraction de primitives de Harris (pour prendre des primitives rapides à calculer et très répandues) est formé d'opérations qui sont largement recouvrantes avec les opérations qui interviennent dans le calcul du champ de mouvements dense par FOLKI, comme illustré en Figure 3.26. Sachant que la mise en oeuvre de FOLKI sur GPU peut se faire en quelques millisecondes, il n'est pas forcément pertinent, d'un point de vue du coût de calcul, de se contenter d'une extraction de quelques centaines de primitives.

La seconde raison est que l'approche par extraction de primitive et algorithme robuste est sujette à des variations importantes des résultats, à la fois à cause du seuillage qui intervient dans l'extraction de primitives et à cause de l'aspect aléatoire de Ransac. Au final, il arrive souvent que le recalage ne soit pas d'une grande précision sur l'ensemble du champ comparé à un recalage dense, même si le modèle paramétrique est valide. Nous avons parfois constaté, en particulier lorsque l'on intègre le recalage sur plusieurs images pour faire apparâitre les objets mobiles, les effets de l'imprécision (cumulée) du recalage par primitives. Un exemple est présenté en Figure 3.27.

Nous allons donc présenter dans la sous-section 3.6.2, une méthode d'estimation dense de modèle paramétrique basée sur FOLKI qui s'est avérée très robuste et rapide à calculer. Ceci nous permet de proposer dans la sous-section 3.6.3 une méthode d'intégration temporelle des mouvements résiduels permettant d'augmenter le rapport signal-à-bruit des détections.

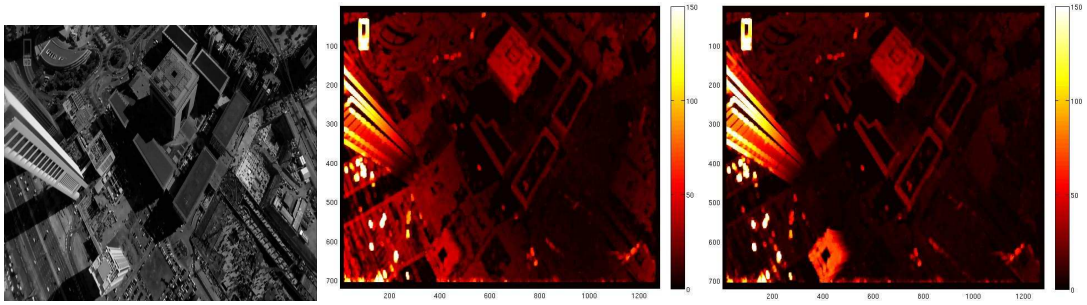


Figure 3.27 – Flot résiduel cumulé après recalage paramétrique d'une série d'images (séquence Dubai). A gauche, image d'origine, au milieu : recalage par primitive et estimation robuste, à droite : approche par recalage dense et régression robuste proposée ici.

3.6.2 Régression paramétrique robuste sur le flot optique

Pour effectuer un recalage paramétrique robuste de deux images I_1 et I_2, nous proposons simplement de faire une régression paramétrique robuste sur le flot optique estimé par FOLKI entre ces deux images, noté $\mathbf{u}$. Nous notons $\mathbf{f}(\mathbf{x};\boldsymbol{\theta})$ le modèle de mouvement paramétrique de paramètre θ, un cas particulier bien connu étant le mouvement affine :

$$\mathbf{f}(\mathbf{x};\boldsymbol{\theta}) = \mathbf{A}\mathbf{x} + \mathbf{t}.$$

qui dépend de 6 paramètres $\boldsymbol{\theta} = [a_{xx}, a_{xy}, a_{yx}, a_{yy}, t_x, t_y]$. La régression robuste consiste à minimiser une somme pénalisée des résidus :

$$\mathcal{J}(\boldsymbol{\theta}) = \sum_{\mathbf{x}\in\Omega} \phi(\|\mathbf{x} + \mathbf{u}(\mathbf{x}) - \mathbf{f}(\mathbf{x};\boldsymbol{\theta})\|),$$

où ϕ est une fonction de pénalisation croissant moins vite que la quadratique pour les grandes valeurs des écarts, telle que la fonction de Huber. Une liste complète de ces fonctions de pénalisation robustes peut être trouvée dans le rapport technique de Zhengyou Zhang [Zhang, 1997].

De manière classique, cette minimisation est effectuée par moindres carrés repondérés, c'est-à-dire qu'on résoud une succession d'optimisations quadratiques :

$$\mathcal{J}^{(k)}(\boldsymbol{\theta}) = \sum_{\mathbf{x}\in\Omega} w(r(\mathbf{x},\boldsymbol{\theta}^{(k)}))\|\mathbf{x} + \mathbf{u}(\mathbf{x}) - \mathbf{f}(\mathbf{x};\boldsymbol{\theta})\|^2, \tag{3.27}$$

où l'indice k désigne l'itération courante, $r(\mathbf{x},\boldsymbol{\theta}^{(k)})$ est la norme du résidu obtenu en $\mathbf{x}$ avec la valeur courante du paramètre :

$$r(\mathbf{x},\boldsymbol{\theta}^{(k)}) = \|\mathbf{x} + \mathbf{u}(\mathbf{x}) - \mathbf{f}(\mathbf{x};\boldsymbol{\theta}^{(k)})\|$$

et w est la fonction de sensibilité associée à la fonction de pénalisation ϕ [Zhang, 1997].

Le calcul des pondérations $w(r(\mathbf{x}, \boldsymbol{\theta}^{(k)}))$ est une opération pixel-à-pixel très rapide sur GPU, ensuite la consitution du système linéaire correspondant au minimum du critère 3.27 implique des opérations de réduction décrites en section 2.3.8. Enfin l'inversion du système (6×6 dans le cas du modèle affine) est effectuée sur le CPU. Notons que ce principe de calcul est plus complexe et plus instable pour un modèle non linéaire comme le modèle homographique : dans la mesure du possible, on évitera ce cadre en l'approximant par des modèles polynômiaux, comme le conseillent plusieurs auteurs [Odobez and Bouthemy, 1995, Jonchery et al., 2008].

Le résultat de ce processus de régression est illustré à la figure 3.28. Sur cette séquence, la caméra effectue un mouvement global important en survolant une route, sur laquelle circulent des véhicules. Le mouvement dominant est estimé par régression robuste sur le flot optique issu de FOLKI (version FOLKI1). Le flot résiduel fait apparaître les véhicules et un arbre situé au premier plan en bas, avec des orientations et des vitesses différentes. Dans ce cadre on peut détecter simplement en utilisant la norme du flot résiduel, présentée à la dernière ligne. On fait alors apparaître les véhicules, comme le montre la surimpression (image + flot résiduel seuillé) présentée en haut à droite.

La régression robuste prend environ 4ms par itération sur une séquence d'images 1280×720 et pour des modèles linéaires, la convergence est très rapide : 5 itérations sont en général suffisantes, ce qui fait un coût total de 20ms. Le coût de calcul de FOLKI sur ces résolutions étant inférieur à 10ms, on obtient donc au final un recalage paramétrique robuste fonctionnant à une fréquence proche de la fréquence vidéo.

3.6.3 Régularisation temporelle de la détection

Dans l'exemple de la Figure 3.28, le mouvement des véhicules est suffisant pour que leur détection soit possible en recalant uniquement deux images, mais la plupart du temps il est nécessaire d'intégrer le flot résiduel temporellement pour faire ressortir les objets mobiles du bruit qui affecte le flot résiduel estimé.

Ce processus d'intégration est très simple, il utilise le principe de l'oubli exponentiel. On suppose que l'on dispose d'une carte d'une image $\mathbf{D}_0^{t-1}$ d'un indice de détection cumulé jusqu'à l'instant $(t-1)$. A l'instant t, on dispose d'une nouvelle image I_t, qui permet le calcul par FOLKI du flot rétrograde $\mathbf{u}_t^{t-1}$. Ce flot permet de warper $\mathbf{D}_0^{t-1}$ en une carte de détection dans la géométrie image à l'instant t, qu'on note $\mathcal{W}_{t-1}^t(\mathbf{D}_0^{t-1})$. Par ailleurs, en effectuant une régression robuste sur le flot $\mathbf{u}_t^{t-1}$, on déduit l'indice de détection instantané à l'instant t, $\mathbf{D}^t$. On combine alors ces deux cartes selon un principe de filtrage temporel de paramètre α, pour

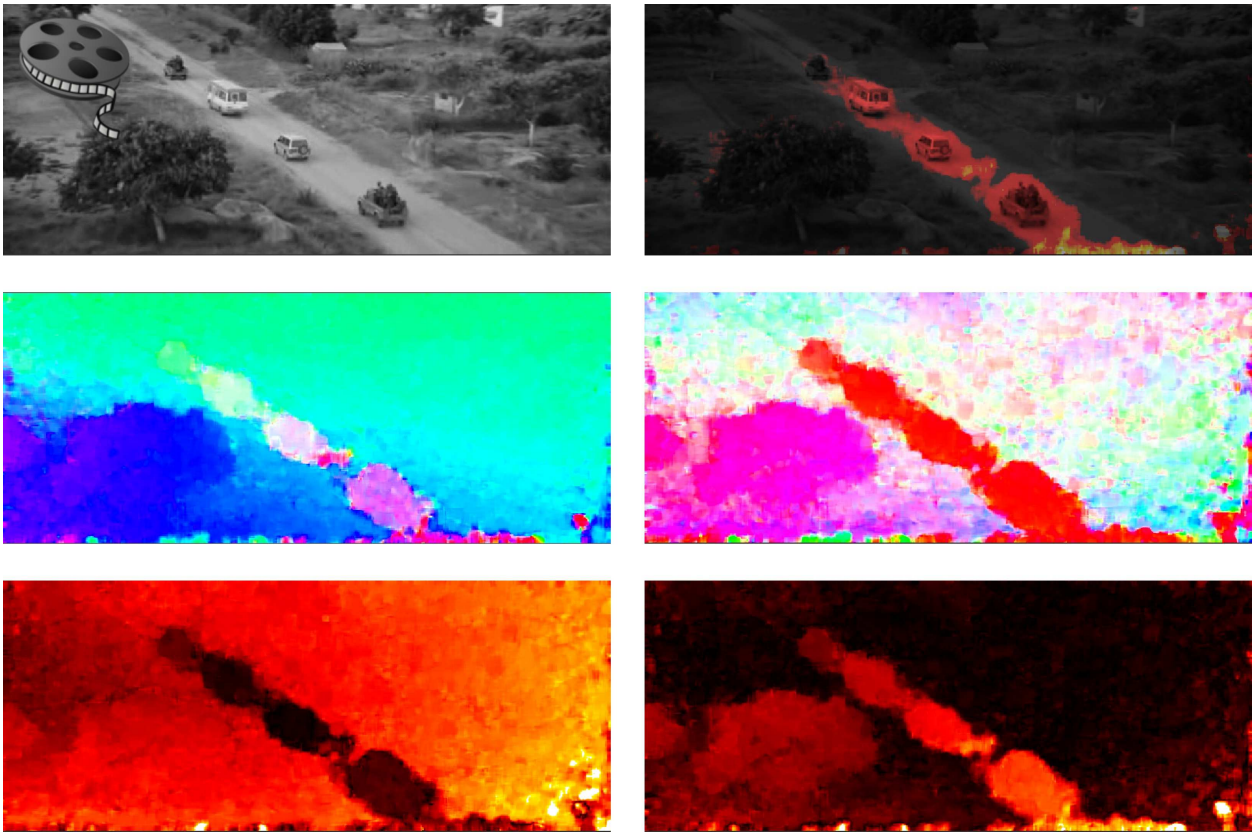


Figure 3.28 – Illustration du recalage robuste sur une séquence extraite du film "Blood Diamond". En haut, image d'origine couleur à gauche et une sur-impression de l'image convertie en niveaux de gris et du flot résiduel seuillé en rouge ; au milieu, à gauche, flot estimé avec FOLKI ; à droite, flot résiduel après régression robuste. En bas à gauche, norme du flot total ; à droite, norme du flot résiduel.

obtenir le nouvel indice de détection cumulé :

$$\mathbf{D}_0^t = \alpha \mathbf{D}^t + (1-\alpha)\mathcal{W}_{t-1}^t(\mathbf{D}_0^{t-1}) \tag{3.28}$$

Le procédé de détection obtenu s'est révélé très efficace et robuste sur des vidéos aériennes : les figures 3.29 et 3.30 suivantes en donnent plusieurs exemples.

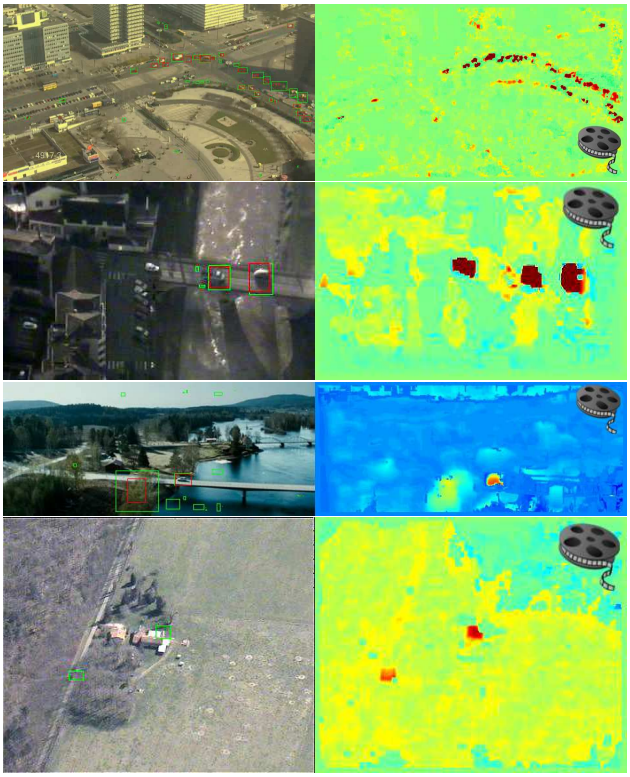


Figure 3.29 – Détection par flot résiduel intégré. De haut en bas : exemple simple avec un point de vue fixe — les exemples qui suivent comprennent tous un mouvement de l'observateur ; vidéo très compressée ; suivi d'un véhicule et végétation ; effet de zoom (la détection centrale est une mire)

A chaque fois on présente l'image du flot résiduel intégré et l'image d'origine avec des rectangles de couleur indiquant les zones détectées, le rectangle vert correspondant à un seuil bas et le rouge à un seuil haut (les niveaux absolus de seuil dépendent de la séquence). Dans la plupart des cas, quelque soit le mouvement propre de l'observateur et la nature de la scène, la détection est pertinente même s'il faut prévoir un traitement ultérieur de reconnaissance pour rejeter des fausses alarmes dues au relief, au mouvements de fumées, etc. Le seul cas d'échec est présenté au bas de la Figure 3.30, dans le cas de bâtiments très élevés et d'un point de vue oblique de l'observateur.

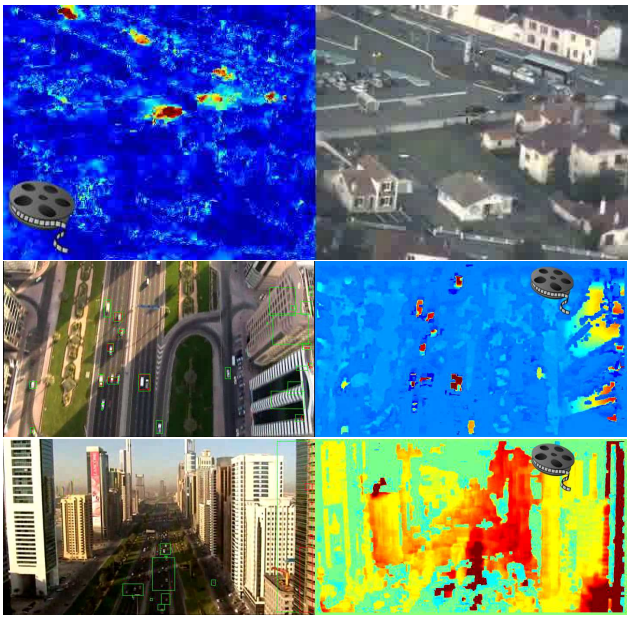


Figure 3.30 – Détection par flot résiduel intégré. De haut en bas : vidéo de très mauvaise qualité ; bâtiments de grande hauteur ; bâtiments élevés et prise de vue oblique : échec du recalage du mouvement dominant.

3.7 Conclusion

La première contribution de ce chapitre est un algorithme très rapide de flot optique, FOLKI. Un cadre d'application idéal a été trouvé au sein de l'ONERA avec la mesure PIV, ce qui a conduit au développement d'un produit, FOLKI-SPIV, désormais utilisé dans l'ensemble de l'ONERA. D'autres applications potentielles, que nous 'avons pas mentionnées, concernent les techniques de mesure de déformation (déformations de maquettes en soufflerie, expérimentations de résistance des matériaux) très utilisées aussi à l'ONERA.

L'utilisation de FOLKI sur des données vidéos "naturelles", type navigation urbaine ou séquences aériennes peut se faire avec quelques améliorations qui ne remettent pas en cause la structure et la vitesse de l'algorithme. Le résultat est une technique certes moins robuste que les méthodes les plus performantes des comparatifs de flot optique, mais beaucoup plus rapide.

On peut remarquer qu'il y a une tendance de la vision par ordinateur à privilégier les solutions robustes (ie. réduire les grosses erreurs) plutôt que les résultats précis (ie. minimiser la variance de l'erreur pour les inliers). Une orientation opposée à celle des domaines de la mesure (PIV, mesures de déformation de matériaux), où les grosses erreurs sont minimisées grâce à un

ajout de texture (ensemencement en PIV, peintures mouchetées, etc.) et les solutions sont comparées sur leur précision.

Le bon comportement de FOLKI amélioré sur les données Kitti devront être confirmées dans une optimisation paramétrique pour soumettre cet algorithme sur le site. Cette optimisation devrait porter sur l'ensemble des paramètres suivants : ordre du filtrage Rank, nombre de niveaux, nombre d'itérations à chaque niveau, rythme d'adaptation du pas, compromis entre stratégie aller/retour et densité du résultat, intérêt du post-filtrage de type débruitage non gaussien. L'objectif, comme dans le reste de nos travaux, sera d'optimiser non pas seulement la qualité du flot estimé mais un compromis qualité sur temps de calcul.

Parmi les extensions envisageables, on peut penser à combiner flot optique et stéréo de façon temporelle, en profitant de la vitesse de calcul de FOLKI. Cette extension est importante dans le contexte de navigation d'un micro-drone équipé d'un capteur stéréoscopique, nous y reviendrons en conclusion de ce manuscrit. Un autre aspect intéressant concerne la prise en compte du flou de bougé, qui affecte souvent les séquences de robotique terrestre. Il existe un couplage entre mouvement, flou de bougé et temps d'intégration qui nous semble une piste d'optimisation conjointe intéressante à étudier.

Bibliographie

[Baker et al., 2011] Baker, S., Scharstein, D., Lewis, J., Roth, S., Black, M., and Szeliski, R. (2011). A database and evaluation methodology for optical flow. *International Journal of Computer Vision*, 92(1) :1–31.

[Barranco et al., 2012] Barranco, F., Tomasi, M., Diaz, J., Vanegas, M., and Ros, E. (2012). Parallel architecture for hierarchical optical flow estimation based on fpga. *Very Large Scale Integration (VLSI) Systems, IEEE Transactions on*, 20(6) :1058–1067.

[Bouguet, 2001] Bouguet, J. (2001). Pyramidal implementation of the affine lucas kanade feature tracker—description of the algorithm. Technical report, Intel Corporation.

[Brossard et al., 2009] Brossard, C., Monnier, J., Barricau, P., Vandernoot, F., Le Sant, Y., Champagnat, F., and Le Besnerais, G. (2009). Principles and applications of particle image velocimetry. *Aerospace Lab J.*, (1).

[Brown et al., 2003] Brown, M., Burschka, D., and Hager, G. (2003). Advances in computational stereo. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 25(8) :993–1008.

[Brox et al., 2004] Brox, T., Bruhn, A., Papenberg, N., and Weickert, J. (2004). High accuracy optical flow estimation based on a theory for warping. *Computer Vision-ECCV 2004*, pages 25–36.

[Bruhn et al., 2006] Bruhn, A., Weickert, J., Kohlberger, T., and Schnörr, C. (2006). A multigrid platform for real-time motion computation with discontinuity-preserving variational methods. *Int. J. Computer Vision*, 70(3) :257–277.

[Bruhn et al., 2005] Bruhn, A., Weickert, J., and Schnörr, C. (2005). Lucas/kanade meets horn/schunck : Combining local and global optic flow methods. *International Journal of Computer Vision*, 61 :211–231.

[Champagnat et al., 2011] Champagnat, F., Plyer, A., Le Besnerais, G., Leclaire, B., Davoust, S., and Le Sant, Y. (2011). Fast and accurate piv computation using highly parallel iterative correlation maximization. *Experiments in Fluids*, 50 :1169–1182. 10.1007/s00348-011-1054-x.

[Champagnat et al., 2009] Champagnat, F., Plyer, A., Le Besnerais, G., Leclaire, B., and Le Sant, Y. (2009). How to calculate dense piv vector fields at video rates. In *Proceedings of 8th International Symposium on Particle Image Velocimetry-PIV09*.

[Davoust, 2011] Davoust, S. (2011). *Dynamique des grandes échelles dans les jets turbulents avec ou sans effets de rotation*. PhD thesis, Ecole Polytechnique X.

[Fezzani, 2011] Fezzani, R. (2011). *Approche parallèle pour l'estimation du flot optique par-méthode variationnelle*. PhD thesis, Univ. Paris 6 Jussieu.

[Fezzani et al., 2010] Fezzani, R., Champagnat, F., and Le Besnerais, G. (2010). Clarifying the Implementation of Warping in the Combined Local Global Method for Optical Flow Computation. In *EURASIP*.

[Fischler and Bolles, 1981] Fischler, M. and Bolles, R. (1981). Random sample consensus : a paradigm for model fitting with applications to image analysis and automated cartography. *Communications of the ACM*, 24(6) :381–395.

[Geiger et al., 2012] Geiger, A., Lenz, P., and Urtasun, R. (2012). Are we ready for autonomous driving ? the kitti vision benchmark suite. In *Computer Vision and Pattern Recognition (CVPR)*, Providence, USA.

[Harris and Stephens, 1988] Harris, C. and Stephens, M. (1988). A combined corner and edge detector. In *Alvey vision conference*, volume 15, page 50. Manchester, UK.

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

[Jonchery et al., 2008] Jonchery, C., Dibos, F., and Koepfler, G. (2008). Camera motion estimation through planar deformation determination. *Journal of Mathematical Imaging and Vision*, 32(1) :73–87.

[Le Besnerais and Champagnat, 2005] Le Besnerais, G. and Champagnat, F. (2005). Dense optical flow by iterative local window registration. In *IEEE International Conference on Image Processing 2005*, pages I–137. IEEE.

[Le Besnerais et al., 2009] Le Besnerais, G., Champagnat, F., Plyer, A., Fezzani, R., Leclaire, B., and Le Sant, Y. (2009). Advanced processing methods for image-based displacement field measurement. *Aerospace Lab Journal*, 1.

[Leclaire et al., 2010] Leclaire, B., Le Sant, Y., Davoust, S., Le Besnerais, G., and Champagnat, F. (2010). Folki-spiv : a new, ultrafast approach for stereo piv. *submitted to Experiments in Fluids*.

[Leclaire et al., 2011] Leclaire, B., Le Sant, Y., Davoust, S., Le Besnerais, G., and Champagnat, F. (2011). A propos de l'instabilité des algorithmes de piv itératifs. *20ème Congrès Français de Mécanique, 28 août/2 sept. 2011-25044 Besançon, France (FR)*.

[Lowe, 2004] Lowe, D. (2004). Distinctive image features from scale-invariant keypoints. *International journal of computer vision*, 60(2) :91–110.

[Lucas and Kanade, 1981] Lucas, B. D. and Kanade, T. (1981). An iterative image registration technique with an application to stereo vision. pages 674–679.

[Odobez and Bouthemy, 1995] Odobez, J. and Bouthemy, P. (1995). Robust multiresolution estimation of parametric motion models. *Journal of visual communication and image representation*, 6(4) :348–365.

[Raffel et al., 2007] Raffel, M., Willert, C., Wereley, S., and Kompenhans, J. (2007). *Particle Image Velocimetry. A practical guide.* Springer, 2nd edition.

[Rudin et al., 1992] Rudin, L., Osher, S., and Fatemi, E. (1992). Nonlinear total variation based noise removal algorithms. *Physica D : Nonlinear Phenomena*, 60(1) :259–268.

[Sand and Teller, 2008] Sand, P. and Teller, S. (2008). Particle video : Long-range motion estimation using point trajectories. *International Journal of Computer Vision*, 80 :72–91.

[Shi and Tomasi, 1994] Shi, J. and Tomasi, C. (1994). Good features to track. In *Computer Vision and Pattern Recognition (CVPR)*, pages 593–600.

[Smith et al., 2009] Smith, M., Baldwin, I., Churchill, W., Paul, R., and Newman, P. (2009). The new college vision and laser data set. *The International Journal of Robotics Research*, 28(5) :595–599.

[Stanislas et al., 2008] Stanislas, M., Okamoto, K., Kaler, C., Westerweel, J., and Scarano, F. (2008). Main results of the third international piv challenge. *Exp. Fluids*, 45 :27–71.

[Steinbruecker et al., 2009] Steinbruecker, F., Pock, T., and Cremers, D. (2009). Advanced data terms for variational optic flow estimation. In *Vision, Modeling, and Visualization Workshop*, Braunschweig, Germany.

[Sun et al., 2010] Sun, D., Roth, S., and Black, M. J. (2010). Secrets of optical flow estimation and their principles. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, pages 2432–2439.

[Tomasi and Manduchi, 1998] Tomasi, C. and Manduchi, R. (1998). Bilateral filtering for gray and color images. In *ICCV'98*, pages 839–846.

[Wedel et al., 2009] Wedel, A., Pock, T., Zach, C., Bischof, H., and Cremers, D. (2009). An improved algorithm for tv-l 1 optical flow. *Statistical and Geometrical Approaches to Visual Motion Analysis*, pages 23–45.

[Werlberger, 2012] Werlberger, M. (2012). *Convex Approaches for High Performance Video Processing*. phdthesis, Graz University of Technology. to appear.

[Werlberger et al., 2009] Werlberger, M., Trobin, W., Pock, T., Wedel, A., Cremers, D., and Bischof, H. (2009). Anisotropic Huber-L1 optical flow. In *Proceedings of the British Machine Vision Conference (BMVC)*, London, UK.

[Zabih and Woodfill, 1994] Zabih, R. and Woodfill, J. (1994). Non-parametric local transforms for computing visual correspondence. In Eklundh, J.-O., editor, *Computer Vision — ECCV '94*, volume 801 of *Lecture Notes in Computer Science*, pages 151–158. Springer Berlin / Heidelberg.

[Zhang, 1997] Zhang, Z. (1997). Parameter estimation techniques : A tutorial with application to conic fitting. *Image and vision Computing*, 15(1) :59–76.

[Zhao and Sawhney, 2002] Zhao, W. and Sawhney, H. (2002). Is super-resolution with optical flow feasible ? In Heyden, A., Sparr, G., Nielsen, M., and Johansen, P., editors, *Computer Vision — ECCV 2002*, volume 2350 of *Lecture Notes in Computer Science*, pages 599–613. Springer Berlin / Heidelberg.

[Zimmer et al., 2011] Zimmer, H., Bruhn, A., and Weickert, J. (2011). Optic flow in harmony. *International Journal of Computer Vision*, 93(3) :368–388.

[Zimmer et al., 2009] Zimmer, H., Bruhn, A., Weickert, J., Valgaerts, L., Salgado, A., Rosenhahn, B., and Seidel, H. (2009). Complementary optic flow. In *Energy minimization methods in computer vision and pattern recognition*, pages 207–220. Springer.

4 Super-Resolution

4.1 Introduction

L'extraction d'informations à partir de vidéos, que ce soit par un utilisateur humain ou par un module logiciel, conduit à rechercher en permanence la meilleure qualité d'image possible.

Il existe différents niveaux de la "chaîne image" (figure 4.1) sur lesquels on peut intervenir afin de produire une vidéo de meilleure qualité. Tout d'abord, on peut prendre en compte cette exigence au niveau de la conception de la caméra par exemple en spécifiant une caméra mieux résolue, c'est-à-dire dont le détecteur possède plus de pixels. En général cette solution passe par un compromis avec d'autres facteurs limitant la qualité, comme le bruit, par exemple, qui tend à augmenter lorsque la surface des détecteurs diminue. Par ailleurs, le principal problème est celui du coût, non seulement du détecteur, mais aussi du système optique, qu'il faudra choisir de meilleure qualité. Enfin il existe des limites liées aux technologies disponibles, comme celle de la finesse de gravure des capteurs CMOS.

A un autre niveau, on peut chercher à contrôler l'acquisition pour optimiser l'information qu'on cherche à extraire de la vidéo. On peut par exemple commander un zoom pour améliorer l'image d'un détail de la scène, tourner autour d'un objet pour capturer sa forme 3D, etc. Cette approche dite de "perception active" fait l'objet de nombreuses études (par exemple récemment à l'ONERA DTIM, la thèse de Joseph Defretin [Defretin, 2011]). Elle pose des problèmes théoriques pour définir et optimiser des stratégies d'acquisition en relation avec l'information recherchée.

Enfin on peut travailler sur une séquence vidéo acquise et chercher à améliorer sa qualité a posteriori. C'est sur ce type de solution, que nous appellerons globalement "super-résolution" (SR), que portent les travaux de ce chapitre.

Les effets de la SR. Les techniques de super-résolution (SR) ont pour objectif principal de reconstruire une image "haute résolution" à partir de plusieurs images dite "basse résolution" (BR) extraites du flux vidéo (en général ce sont des images proches temporellement). Plus précisément, le processus de SR a trois effets principaux :

– il effectue un *dépliement spectral*, c'est-à-dire qu'il fait apparaître des détails de hautes

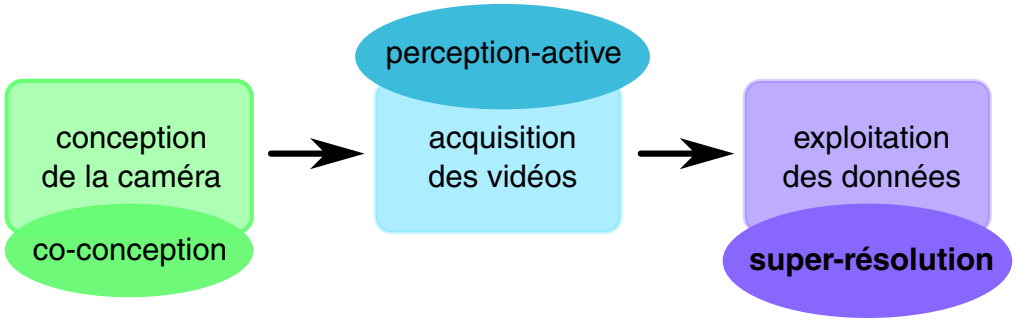


Figure 4.1 – La question de l'amélioration de la qualité des informations fournies par une vidéo peut être abordée à différentes étapes de la chaîne image, depuis la conception du capteur aux traitements des données enregistrées, en passant par le contrôle des conditions d'acquisition.

fréquences spatiales qui avaient été coupés ou altérés par la fréquence d'échantillonnage du capteur (ie. l'espacement entre pixels). Sur la partie gauche de la Figure (figure 4.2), cela se traduit par la possibilité de compter les motifs périodiques dans l'image restaurée, ce qui n'était pas possible dans l'une des images d'origine.

– il effectue un *débruitage*, autrement dit une augmentation du rapport signal à bruit. On le voit clairement sur la partie droite de la Figure (figure 4.2) en comparant l'image SR et la vidéo source issue d'un capteur infra-rouge très bruité (parce qu'utilisé avec une faible durée d'exposition).
– enfin, il améliore le contraste de l'image en compensant l'atténuation fréquentielle liée à la fonction instrument (ou PSF, pour Point Spread Function), selon un principe général de *déconvolution*. Cet effet intervient, de manière moins spectaculaire que le dépliement, sur la mire en partie gauche de la Figure (figure 4.2).

Il s'agit là des principaux effets de la SR, qui sont liés au modèle de formation d'image qui sera présenté en Section 4.2, mais la SR peut aussi permettre de revenir sur des phénomènes qui n'entrent pas dans ce modèle, comme d'améliorer des séquences détériorées par des algorithmes de compression vidéo, cf. section 4.5.3

Plusieurs types de SR. Pour atteindre ces objectifs, les algorithmes de super-résolution cherchent à exploiter les redondances propres à une séquence vidéo. Il existe plusieurs types de redondances auxquelles on peut associer plusieurs approches de la SR.

La solution la plus répandue, et sans doute la moins risquée au niveau des hypothèses, est d'exploiter la redondance temporelle liée à la présence du même objet dans plusieurs images consécutives d'une même vidéo. Il s'agit là du sens le plus commun du terme "super-résolution". La problématique est alors d'estimer les mouvements apparents de ces objets d'une image à

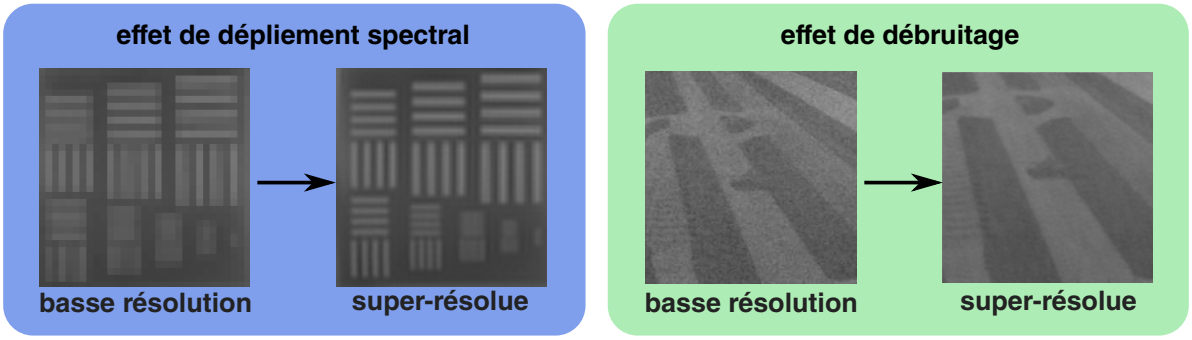


Figure 4.2 – Nous illustrons ici deux des principaux effets de la super-résolution que sont le dépliement spectral (à gauche) et le débruitage (à droite).

l'autre puis de combiner les intensités mesurées pour produire une image améliorée. On peut cependant remarquer que l'estimation d'un mouvement entre plusieurs images est un problème difficile (voir chapitre 3), certains auteurs proposent donc utiliser un ensemble concurrent de mouvements estimés, pondérés selon un critère de fiabilité. Dans [Takeda et al., 2009], cette méthode est poussée à l'extrême en évaluant tous les déplacements possibles dans un certain voisinage, ce qui conduit bien sûr à un coût de calcul très élevé.

Si on fait l'hypothèse qu'il existe dans chaque image de la séquence une redondance spatiale, on peut aussi chercher à l'exploiter. On peut par exemple associer les "patchs" (petites zones contiguës) d'une image qui sont similaires, pour les combiner et produire un patch amélioré. C'est le principe des filtres non locaux, comme le filtre "Non-Local Mean" proposé par Buades et al. [Buades et al., 2005] pour le débruitage et appliqué à la SR dans [Protter et al., 2009]. Ces travaux partent de l'idée que les patches composant une image sont hautement redondants, par exemple dans [Glasner et al., 2009] il est montré, sur la base d'images de Berkeley, que jusqu'à 80% des patches d'une image possèdent 9 patchs similaires dans cette image à l'échelle 1, et que jusqu'à 60% des patchs ont 9 patchs similaires à l'échelle 4. Forts de cette remarque, ces travaux exploitent la redondance intra-image pour effectuer de la super-résolution. On remarquera juste qu'en pratique cela exige des schémas algorithmiques beaucoup moins réguliers que ce qu'on peut faire pour l'exploitation de la redondance inter-images, donc que cela risque de conduire à des coûts de calcul bien plus élevés sur des architectures massivement parallèles.

Enfin des travaux récents [Shechtman et al., 2005] exploitent la redondance spatio-temporelle en recherchant des patchs 3D dans un bloc d'intensités 3D formé de plusieurs images successives d'une vidéo.

Dans ce travail, nous nous focaliserons sur la première famille de méthodes exploitant la

redondance temporelle, en nous appuyant sur nos travaux concernant l'estimation rapide du flot optique qui ont fait l'objet du Chapitre 3.

Contributions. Nous sommes partis des travaux précédents menés par Frédéric Champagnat dans l'équipe EVS de l'ONERA-DTIM. Il s'agit en particulier de travaux sur le choix du modèle de formation d'images, débutés durant la thèse de Gilles Rochefort [Rochefort, 2005] et poursuivis durant la thèse d'Antoine Létienne [Létienne, 2010] concernant les objets mobiles. Dans cette dernière thèse, on peut trouver un remarquable état de l'art des différentes méthodes de SR sur vidéos, ainsi qu'une analyse de méthodes approximatives rapides dites "*shift and mean*" dans le cadre de mouvements translationnels ou affines.

Par rapport à ces travaux, nos contributions sont les suivantes :

- Utilisation du flot optique, alors que tous les travaux précédents de l'équipe recalent l'image selon un modèle paramétrique, translationnel ou affine. Nous comparerons les résultats avec un mouvement paramétrique et le flot optique dans la section 4.5.4. En ce qui concerne l'estimation du flot optique nous nous restreignons ici à l'estimateur le plus rapide (FOLKI), quitte à tenter de corriger ses défauts par un traitement SR approprié. En effet, l'emploi d'un estimateur de flot plus sophistiqué (comme dans les références [Mitzel et al., 2009, Unger et al., 2010]) conduit à un temps de traitement trop important.
- Nous avons aussi cherché à exploiter au maximum la puissance de calcul d'architectures massivement parallèles de manière à pouvoir produire des chaînes de traitements temps réel (ie. à cadence vidéo) ;
- Utilisant ces architectures, nous avons développé une plate-forme de traitement SR qui permet de comparer différentes solutions en termes de coût et de qualité du résultat ;
- Cette étude nous permet de distinguer deux solutions, l'une utilisant une méthode directe très rapide mais laissant peu de place à l'évolutivité de l'algorithme, l'autre méthode, itérative, est moins rapide mais permet une grande souplesse dans l'intégration de critères variés.
- Enfin grâce à cette étude, nous avons montré l'intérêt de choisir un modèle de formation d'image (modèle direct) adapté pour obtenir des solutions rapides et robustes.

La super-résolution à partir de flot optique a été proposée il y a déjà 10 ans [Zhao and Sawhney, 2002], mais sa mise en pratique dans des conditions réalistes est très récente. Il s'agit essentiellement des travaux menés à l'université Technique de Graz (TU Graz) par l'équipe de H. Bischof [Mitzel et al., 2009, Unger et al., 2010]. Nos travaux ont été menés en parallèle de ceux de cette équipe, mais selon une orientation différente. Notre objectif est de proposer des techniques de SR *rapides* à base de flot optique,

en utilisant des algorithmes économes et adaptés aux architecture parallèles. Les travaux cités [Mitzel et al., 2009, Unger et al., 2010] utilisent des estimateurs (de mouvement et d'image SR) sophistiqués et chers et des implémentations sur GPU pour obtenir des temps de traitement raisonnables. Il n'y a pas d'étude de compromis temps de traitement / qualité de résultat dans leurs travaux et les méthodes qu'ils proposent sont de plusieurs ordres de grandeur plus lourdes (sur le même type d'architecture) que celles étudiées ici.

Le déroulement du chapitre et une remarque. Nous commencerons ce chapitre par une présentation des modèles de formation d'image qui font la base des algorithmes de super-résolution. Puis nous présenterons notre solution de SR rapide basée sur une résolution directe de l'inversion d'un de ces modèles. Nous présenterons ensuite un algorithme itératif, toujours basé sur le même modèle, mais possédant l'avantage de pouvoir évoluer vers divers types d'estimateur, alors que l'algorithme rapide n'en fournit qu'un. Nous finirons ce chapitre sur une étude de résultats de nos algorithmes appliqués à différents contextes plus ou moins difficiles.

Nous terminons cette introduction par une remarque : les résultats que nous présentons dans ce chapitre sont uniquement obtenus sur des données réelles non simulées. En effet, dans le cadre de la super-résolution, il n'est pas rare de voir des articles publiés présentant des reconstructions SR uniquement sur données simulées, c'est-à-dire en utilisant des images réelles dont la résolution est d'abord réduite avant d'être utilisée pour la SR. Ceci n'est qu'un modèle très simplificateur de l'acquisition réelle, cela ne simule pas des mouvements réalistes, la dégradation due à la compression, etc. Notre choix nous permet donc d'éviter de travailler sur des données non représentatives, en revanche l'utilisation de données réelles pose le problème de l'évaluation quantitative. Nous proposons une évaluation essentiellement qualitative sur des séquences présentant des motifs haute fréquence facilement identifiables, comme des mires de résolution, ou des caractères comme sur l'exemple du voilier présenté en Figure 4.3. Cet exemple, extrait d'une séquence IR, sert d'illustration dans les sections suivantes qui décrivent les algorithmes que nous avons développés.

L'origine de toutes les séquences servant aux illustrations de ce chapitre fait l'objet de la table B.2. Les paramètres de traitement de toutes les figures sont rappelés dans la table B.3. Ces deux tables sont présentées en section B.2.

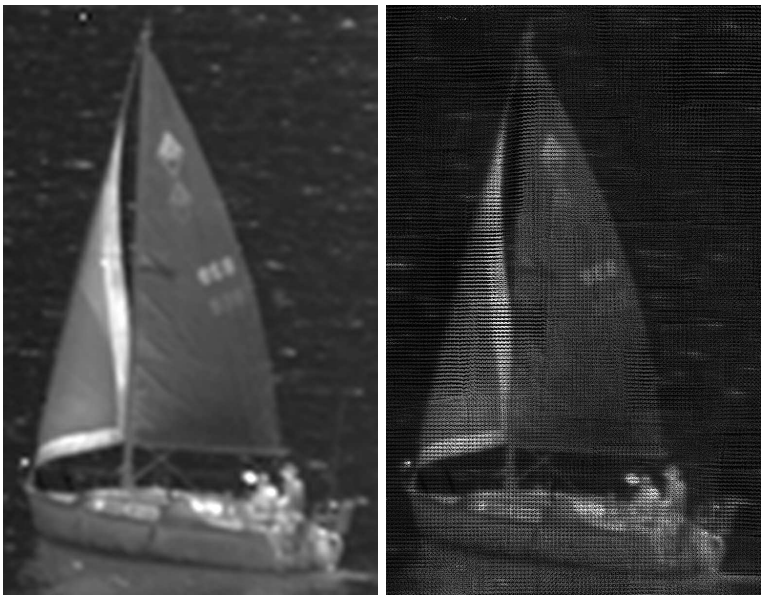


Figure 4.3 – A gauche, une image de la séquence Infra-Rouge BR "Voilier" (origine : voir table B.2) après zoom bilinéaire. La figure de droite présente le recalage dans une même géométrie haute résolution des intensités extraites de 21 images consécutives de la séquence (c'est le résultat du procédé de "Shift-and-Mean" qui sera décrit dans la section 4.3.1).

4.2 Modèles de formation d'image

Pour faire de la super-résolution, on commence par spécifier un modèle de formation des images dites Basses Résolution (BR) faisant apparaître l'image Haute Résolution (HR) que l'on cherche à reconstruire. Ce modèle, même s'il approche une réalité physique, n'en est pas une, mais essentiellement une abstraction numérique. Si l'on cherchait à retranscrire toute l'information de la formation physique d'une image numérique, le modèle serait extrêmement complexe et impossible à inverser. On fait donc toujours des hypothèses simplificatrices. Il arrive aussi qu'on sorte des hypothèses d'emploi d'un modèle simplifié : cela peut être apprécié comme une erreur mathématique, mais nous le ferons volontairement dans la mesure où cela peut améliorer l'efficacité de l'algorithme — et en vérifiant naturellement que la qualité des résultats n'est pas (trop) dégradée.

La SR suppose le choix d'une géométrie de référence, qui est en général définie par un changement d'échelle sur la géométrie d'une des images BR de la séquence (par exemple l'image centrale), qu'on appellera l'image BR de référence. Le facteur de changement d'échelle est appelé facteur de super-résolution, noté f_{SR}, il est typiquement de 1 à 10 dans les exemples que nous traitons. Ce facteur n'exprime pas le gain réel en résolution apporté par la SR, comme

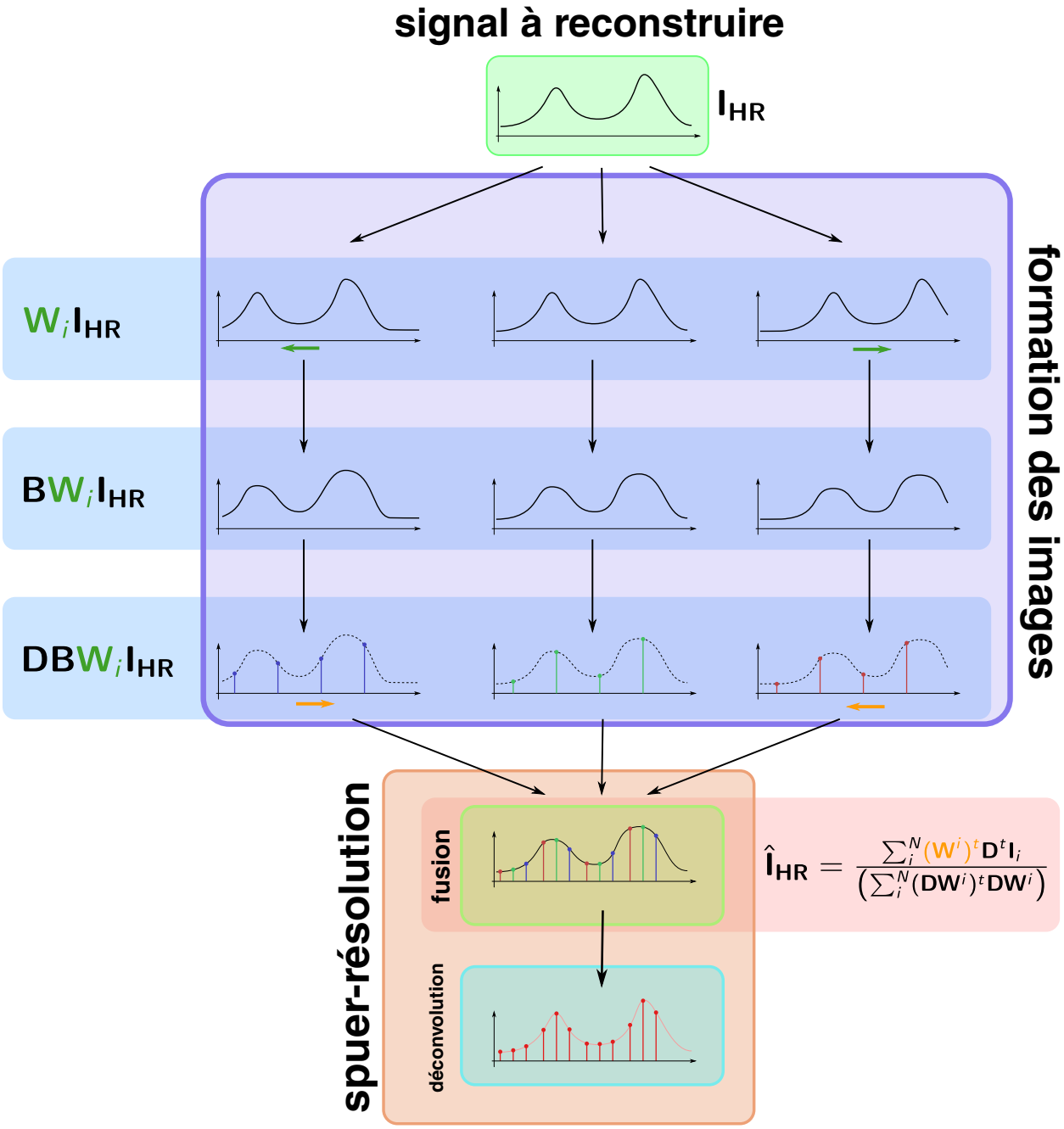


Figure 4.4 – Schémas de la formation des images et du processus de reconstruction par super-résolution en 1D (les notations sont définies dans le texte)

cela est rappelé dans la référence [Champagnat et al., 2009]. Dans les situations courantes, le gain réel en résolution est la plupart du temps inférieur à 2 : une évaluation empirique sur des images de mire est effectuée en section 4.5.1. Cependant, comme nous le verrons, il peut être intéressant de travailler avec un facteur f_{SR} plus important lors du traitement SR.

Modèle "WarpHR". Le modèle de formation d'une image BR à partir de l'image *idéale* HR le plus couramment employé est le suivant :

$$\mathbf{I}^i_{\mathrm{BR}} = \mathbf{DBW}_i \, \mathbf{I}_{\mathrm{HR}} + \mathbf{G}, \tag{4.1}$$

où $\mathbf{I}^i_{\mathrm{BR}}$ est un vecteur de dimension N_{BR} contenant les intensités d'une image basse résolution à un temps d'acquisition i, $\mathbf{I}_{\mathrm{HR}}$ est un vecteur de dimension $N_{\mathrm{HR}} = f_{\mathrm{SR}} N_{\mathrm{BR}}$ contenant l'image haute résolution recherchée dans la géométrie de référence. L'opérateur de déformation $\mathbf{W}_i$ correspond au champ de mouvement entre la géométrie de référence et le temps d'acquisition i. $\mathbf{B}$ est une matrice de convolution associée à la PSF modélisant le système optique et l'intégration sur la surface du détecteur, $\mathbf{D}$ est un opérateur de décimation permettant de passer de la haute résolution à la basse résolution et pour finir $\mathbf{G}$ est un bruit gaussien modélisant l'imperfection de la mesure capteur. Très souvent, la PSF du système est approximée par une tâche gaussienne isotrope, paramétrée par son écart-type.

Dans la partie supérieure de la Figure 4.4 nous avons représenté en une dimension le processus de formation d'image correspondant à (4.1). Ce modèle est employé dans les méthodes dites de *Hardie* dans [Létienne, 2010] ou plus récemment dans la référence [Mitzel et al., 2009, Unger et al., 2010].

Fondamentalement, ce modèle repose sur la même hypothèse que celle utilisée lors de l'estimation du flot optique, c'est-à-dire sur la constance des intensités (voir chapitre 3). On suppose donc que les différences entre les images s'expliquent exclusivement par l'opérateur de déformation $\mathbf{W}_i$ en géométrie 2D. Dans la pratique, beaucoup de phénomènes ne sont pas modélisés, comme la rotation 3D d'un objet se présentant sous différents points de vue au cours de l'acquisition, l'occultation d'une partie de la scène par un objet plus proche, les variations d'illumination globales (adaptation du gain de la caméra) ou locales (reflets spéculaires, etc.) ou enfin les variations de la PSF dues aux mouvements (flou de bougé) ou aux variations locales de profondeur (défocus). Néanmoins, nous travaillerons dans la suite en négligeant ces phénomènes, comme le font la plupart des travaux sur la SR (et sur le flot optique).

Dans le modèle (4.1), le mouvement est appliqué à la scène HR avant l'application des opérateurs d'imagerie proprement dits (convolution et décimation) : pour cette raison nous appellerons ce modèle WarpHR dans la suite. C'est un ordre qui peut paraître logique et qui conduit

à des approximations discrètes de bonne précision, même en présence de fortes rotations ou d'effets de zoom entre les images, comme l'a montré Gilles Rochefort [Rochefort et al., 2006].

En pratique, cependant, ce modèle a un gros défaut, lorsque l'on ne connaît pas les mouvements. Il faut alors les estimer et cette estimation se fait sur les images d'entrée, donc en géométrie BR. Dans ce travail, nous calculons les flots optiques entre toutes images de la séquence et l'image BR de référence. Ces recalages fournissent des champs de mouvements à l'échelle BR et non à l'échelle HR : il faudra donc les interpoler — dans [Unger et al., 2010], ce sont les images BR qui sont interpolées avant recalage — avant de les utiliser pour construire l'opérateur de warp $\mathbf{W}_i$. Mais le problème le plus grave, de notre point de vue, est que l'on manipule un opérateur $\mathbf{W}_i$ de très grandes dimensions, puisqu'il opère sur l'image HR $\mathbf{I}_{\mathrm{HR}}$: lors de l'inversion, cela entraîne des opérations essentiellement en HR, donc très coûteuses en temps et en place mémoire.

Modèle "WarpBR". Dans les techniques de SR rapides, on utilise souvent un modèle concurrent du précédent, dans lequel on commute les opérateurs de flou et de mouvement :

$$\mathbf{I}^i_{\mathrm{BR}} = \mathbf{D}\mathbf{W}_i\mathbf{B}\mathbf{I}_{\mathrm{HR}} + \mathbf{G}. \tag{4.2}$$

Ce modèle a été étudié dans les travaux précédents de l'ONERA DTIM, en particulier les travaux de Gilles Rochefort [Rochefort et al., 2006] et la thèse d'Antoine Létienne [Létienne, 2010]. La permutation ne se justifie rigoureusement que lorsque le mouvement entre les images est une translation globale ou pour des mouvements isométriques avec une PSF isotrope. Néanmoins Antoine Létienne a montré que ce modèle pouvait être utilisé dans le cas de mouvements localement affines avec une perte de performance limitée [Létienne, 2010].

Par ailleurs le modèle (4.2) permet d'obtenir des simplifications très importantes lors de la résolution. La première est qu'en exprimant les composantes du produit $\mathbf{D}\mathbf{W}_i$ on constate que seuls les mouvements à l'échelle BR interviennent (et de même dans l'opérateur transposé $\mathbf{W}_i^t\mathbf{D}^t$ utilisé dans l'optimisation). On peut donc construire $\mathbf{D}\mathbf{W}_i$ directement à partir des flots optiques estimés entre les images BR et l'image BR de référence. Nous appellerons ce modèle "WarpBR", puisque les déformations d'image y sont effectuées en géométrie BR. Comme nous le verrons dans la suite, cette approximation est quasi-indispensable au développement de solutions SR temps réel — c'est à notre avis un défaut des techniques proposées par [Mitzel et al., 2009, Unger et al., 2010] de ne pas utiliser ce modèle mais le WarpHR.

4.3 Super-résolution rapide

Dans les années 90, plusieurs auteurs, dont M. Irani et M. Elad [Elad and Hel-Or, 2001], observent que le modèle WarpBR (4.2) permet de décomposer l'inversion en deux étapes, représentées dans la partie basse de la Figure 4.4. En notant $\hat{\mathbf{I}}_{\mathrm{HR}} = \mathbf{B}\ \mathbf{I}_{\mathrm{HR}}$, on effectue la SR par :

- estimation de $\hat{\mathbf{I}}_{\mathrm{HR}}$ par la fusion des M images basse résolution ;
- estimation de $\mathbf{I}_{\mathrm{HR}}$ par déconvolution de $\hat{\mathbf{I}}_{\mathrm{HR}}$.

En pratique, cela revient à optimiser successivement deux critères $E_{fus}(\hat{\mathbf{I}}_{\mathrm{HR}})$ et $E_{deconv}(\mathbf{I}_{\mathrm{HR}})$:

$$E_{fus}(\hat{\mathbf{I}}_{\mathrm{HR}}) = \sum_{i=1}^{M} \left\| \mathbf{I}^i_{\mathrm{BR}} - \mathbf{DW}_i\ \hat{\mathbf{I}}_{\mathrm{HR}} \right\|^2 \tag{4.3}$$

$$E_{deconv}(\mathbf{I}_{\mathrm{HR}}) = \left\| \hat{\mathbf{I}}_{\mathrm{HR}} - \mathbf{B}\ \mathbf{I}_{\mathrm{HR}} \right\|^2 + \mathcal{R}(\mathbf{I}_{\mathrm{HR}}) \tag{4.4}$$

Le terme $\mathcal{R}$ qui intervient dans le second critère est un terme de régularisation : il s'agit en effet d'un problème de déconvolution mal posé, pour lequel la minimisation du seul terme des moindres carrés est instable [Idier, 2001]. En revanche, le problème de fusion (4.3) est bien posé. La mise en œuvre de ces deux optimisations fait l'objet des deux sections suivantes.

4.3.1 Fusion des images BR

On va ici s'intéresser à l'estimation de $\hat{\mathbf{I}}_{\mathrm{HR}}$ grâce à la minimisation du critère E_{fus}, de l'équation (4.3). On va pour cela chercher le point stationnaire

$$\frac{\partial E_{fus}}{\partial \mathbf{I}_{hr}} = \sum_{i}^{M} (\mathbf{DW}_i)^t (\mathbf{DW}_i\ \hat{\mathbf{I}}_{hr} - \mathbf{I}_i) = 0 \tag{4.5}$$

soit :

$$\left(\sum_{i}^{N} (\mathbf{DW}_i)^t \mathbf{DW}_i \right) \hat{\mathbf{I}}_{hr} = \sum_{i}^{M} (\mathbf{DW}_i)^t \mathbf{I}_i \tag{4.6}$$

Cet estimateur est appelé "*Shift-and-Mean*" (SM). En effet l'opération au second membre consiste à décaler chaque pixel des images basse résolution selon le mouvement estimé pour les remettre en géométrie de référence et à accumuler l'intensité BR sur les pixels HR correspondants — c'est la partie "shift and add". Dans la suite on notera parfois cette image SA($\{\mathbf{I}_i\}$). La matrice normale du problème $\mathbf{P} = \left(\sum_i^M (\mathbf{DW}_i)^t \mathbf{DW}_i \right)$ est diagonale [Elad and Hel-Or, 2001], ses composantes sont des pondérations proportionnelles au nombre de pixels des images BR qui influent sur un pixel HR. En pratique on obtient ces pondérations en effectuant un "shift

and add" sur des images BR égales à 1, il sera parfois pratique dans la suite d'écrire ces pondérations sous la forme SA($\{1\}$). L'équation (4.6) peut alors s'écrire

$$\hat{\mathbf{I}}_{hr} = \text{SA}(\{\mathbf{I}_i\}) \oslash \text{SA}(\{1\})$$

c'est-à-dire comme la division point à point de l'image shift-and-add par les pondérations, ce qui réalise la moyenne des intensités des pixels BR — c'est donc l'étape "mean".

Mise en oeuvre. Il est important de remarquer que $(\mathbf{DW}_i)^t$ est une interpolation non régulière sur la grille haute résolution. Pour l'effectuer efficacement, nous utilisons simplement une troncature du mouvement au pixel HR le plus proche, ce qui est réalisé en pratique grâce au schéma "Vote" présenté dans la section 2.3.6 . Bien entendu, l'approximation correspondante est liée à la finesse de la grille HR choisie, donc au facteur de super-résolution, qui définit le pas de la grille HR relativement au pas pixel BR. C'est la raison pour laquelle nous utilisons en pratique des facteurs de SR plus élevés que le gain en résolution réellement attendu.

Remarquons que rien dans l'équation (4.6) ne garantit que tous les pixels HR reçoivent une valeur. En fait, dans la plupart des cas réels, même avec un nombre important d'images BR il reste des trous, dont la taille maximale est donnée par le facteur de super-résolution choisi. Pour les combler, on propage les intensités disponibles au plus proche voisin (en 4-connexité), en tenant compte des pondérations. Etant donné la taille maximale du trou, on sait combien d'itérations seront nécessaires pour obtenir une image estimée sur toute la grille HR.

Le facteur de super-résolution est le paramètre principal de cette étape de fusion d'images par SM. Il conditionne à la fois la taille mémoire, la précision de la troncature des mouvements et la taille des trous à combler (donc le nombre d'itérations nécessaires pour l'interpolation).

4.3.2 Déconvolution

Pour l'étape de déconvolution, nous avons plusieurs choix. On peut tout d'abord choisir d'optimiser un critère de déconvolution quadratique tel que :

$$E_{deconvQ}(\mathbf{I}_{\text{HR}}) = \left\| \hat{\mathbf{I}}_{\text{HR}} - \mathbf{B}\ \mathbf{I}_{\text{HR}} \right\|^2 + \lambda \|\nabla \mathbf{I}_{\text{HR}}\|^2. \tag{4.7}$$

Cette optimisation peut être menée itérativement par descente de gradient avec l'équation de remise à jour suivante :

$$\mathbf{I}_{\text{HR}}^{k+1} = \mathbf{I}_{\text{HR}}^{k} - \eta \left(-\mathbf{B}^t \hat{\mathbf{I}}_{\text{HR}} + \mathbf{B}^t \mathbf{B} \mathbf{I}_{\text{HR}}^{k} + \lambda \nabla^2 \mathbf{I}_{\text{HR}}^{k} \right), \tag{4.8}$$

où η est un pas de descente et λ le paramètre de régularisation. L'opérateur ∇^2 est une discrétisation du Laplacien par différences finies, obtenue par convolution

$$\nabla^2 \mathbf{I} = \mathbf{d}_x^{\mathrm{r}} * (\mathbf{d}_x * \mathbf{I}) + \mathbf{d}_y^{\mathrm{r}} * (\mathbf{d}_y * \mathbf{I})$$

où $\mathbf{d}_x$ (respectivement $\mathbf{d}_y$) désigne un noyau de dérivation en ligne (respectivement en colonne). La notation $\mathbf{d}^{\mathrm{r}}$ désigne le noyau retourné en ligne et en colonne, ce qui correspond à appliquer la transposée de l'opérateur. Comme la PSF de la caméra est modélisée par une fonction gaussienne isotrope, toutes les opérations de l'équation de remise à jour (4.8) s'obtiennent par des convolutions séparables. Pour les convolutions, les conditions de bord sont de supposer l'image nulle en dehors du support, sauf pour les dérivations qui sont faites avec une condition de bord libre.

Mise en oeuvre. Avec des convolutions sous hypothèse de bord périodique on sait que le critère précédent peut s'écrire avec des FFT et que le minimum peut s'obtenir directement (sans itérations) par un filtre de Wiener qui a une formulation explicite dans le domaine de Fourier [Idier, 2001] : cela conduit par exemple Antoine Létienne à proposer une mise en œuvre par FFT pour la SR rapide sur objets mobiles [Létienne, 2010]. Cependant la FFT est une opération qui reste coûteuse sur des architectures massivement parallèles, avec une résolution de l'algorithme "papillon" dont le coût est en log de la dimension de l'image.

En revanche, sur ce type d'architectures il est beaucoup plus efficace de mettre en œuvre un filtre FIR, si celui-ci est de support réduit. Une solution est d'approximer le filtre de Wiener de déconvolution par un filtrage avec un masque de taille réduite. Plutôt que de tronquer la réponse spatiale de ce filtre, nous reprenons l'approche "*small kernel*" (SK), introduite à l'origine dans [Reichenbach and Park, 1991]. Cette approche consiste à estimer un filtre de Wiener optimal, au sens d'un critère quadratique défini par un niveau de bruit et une fonction de covariance de la scène, sous une contrainte de support limité : un critère explicite plus simple que celui de [Reichenbach and Park, 1991] est présenté dans [Pereira and Goussard, 1997]. Nous utilisons une formulation utilisant moins de paramètres due à F. Champagnat [Champagnat, 2011]. Grâce à une extension du critère d'optimalité de Wiener aux processus intrinsèques définis par un variogramme, le filtre SK est uniquement défini par trois paramètres : l'écart-type de la PSF (supposée gaussienne), le rapport bruit-à-signal (ie. le paramètre de régularisation) et la puissance du modèle de variogramme, que nous choisirons en général égale à 1, soit un modèle de variogramme linéaire — ou processus brownien.

Deux exemples de réponse de filtres SK sont présentés en figure 4.5 ci-dessous. Les paramètres sont ceux utilisés pour traiter la séquence Camion (cf. Table B.3) avec un facteur de SR de 3. L'écart-type de la PSF est de 0.43 (pour un pixel BR de 1) il a été mesuré sur une

image BR par une technique d'identification utilisant un bord de plage, technique décrite par exemple dans [Champagnat et al., 2009], voir la Section 4.5.1. Le paramètre de régularisation est choisi à 0.01 et le modèle de variogramme est linéaire. Deux réalisations sont présentées pour des supports respectifs de 17×17 et 25×25. Avec ces paramètres, le support le plus étroit est suffisant. Le fait que le filtre le plus large est nul au bord du support illustre par ailleurs la décroissance du filtrage de Wiener dans cet exemple et justifie après coup l'approche SK.

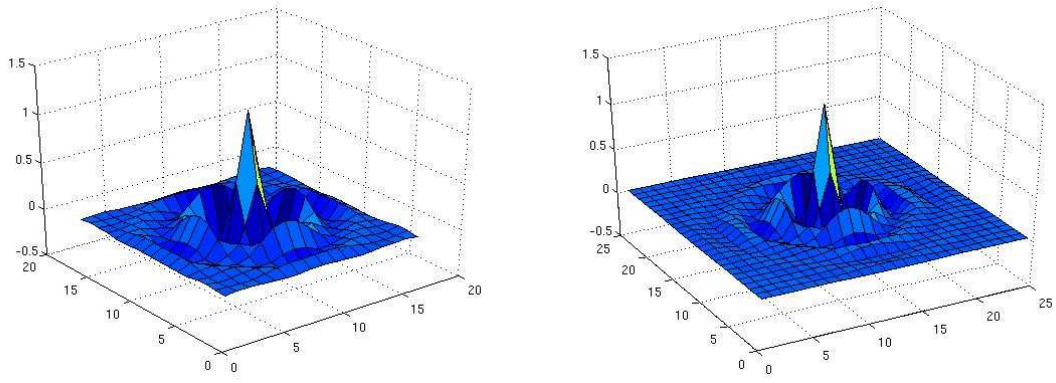


Figure 4.5 – Masques de convolution de filtre small kernel (SK) associés aux paramètres suivants : facteur de super-résolution 3, écart-type de la PSF 0.43 (en pixels BR), paramètre de régularisation 0.01, modèle de variogramme linéaire. A gauche, support 17×17 à droite, support 25×25. Pour ces paramètres, le filtre optimal est à support réduit et le filtre SK 17×17 est une réalisation correcte.

L'utilisation d'un small kernel au lieu d'un schéma itératif du type de (4.8) possède l'avantage d'être beaucoup plus efficace en temps de calcul et occupation mémoire. En termes de coût, le paramètre essentiel de cette étape est la taille du support du filtre SK. Celle-ci dépend de tous ses paramètres : dans tous les exemples traités dans la suite ce support est inférieur à 30×30.

Avant de présenter des résultats et des évaluations de performance, on peut souligner d'emblée une limitation de cette étape. L'image d'entrée $\hat{\mathbf{I}}_{\mathrm{HR}}$ résulte du Shift-and-Mean qui accumule un nombre de données BR différent suivant les pixels — ce nombre est donné par la composante correspondante de la matrice normale $\mathbf{P}$ — et interpole les pixels qui ne sont pas renseignés. Le rapport signal-à-bruit de ces données est donc in-homogène ce qui n'est pas pris en compte dans la technique de déconvolution rapide par filtre SK qui vient d'être décrite.

4.3.3 Etapes et coût de calcul

Nous détaillons ci-dessous la mise en œuvre de la SR rapide surtout pour les premières étapes (recalage et SM) qui seront utiles dans tous les autres algorithmes qui seront vus dans la suite.

Un algorithme en quatre étapes. La première étape (figure 4.6) consiste en l'estimation du mouvement entre toutes les images du groupement (on parlera souvent de GoP pour "Group of Pictures") utilisé et une image de référence. Pour ce faire nous avons le choix entre utiliser un algorithme d'estimation de mouvement global (présenté dans la section 3.6) ou bien un algorithme de flot optique. Au contraire de la plupart des travaux sur la SR, nous avons choisi ici d'utiliser le flot optique et plus précisément l'implémentation GPU de FOLKI présentée en section 3.3, qui présente l'intérêt d'être très rapide (même comparée au recalage d'image paramétrique, comme discuté en section 3.6) et capable, grâce à l'utilisation d'une pyramide d'images multi-résolution, d'estimer des grands déplacements. Ces deux caractéristiques sont utiles pour la SR, dans laquelle on doit recaler non seulement des images proches temporellement, mais toutes les images du GoP dans la géométrie de l'image BR de référence (cf. Figure 4.6), donc effectuer en un temps très court plusieurs recalages avec des déplacements éventuellement importants. Nous reviendrons en section 4.5.4 sur les avantages et inconvénients du recalage par flot optique, en termes de qualité du résultat SR et suivant la séquence utilisée.

Les seconde et troisième étapes constituent le Shift-and-Mean. Il faut d'une part (figure 4.7) utiliser les mouvements estimés pour faire voter chacun des pixels BR sur deux grilles HR. La première, $\tilde{\mathbf{I}}_{\mathrm{HR}}$, est l'image SA($\{\mathbf{I}_i\}$) qui accumule les intensités, la seconde $\mathbf{P}_{\mathrm{HR}} = \mathrm{SA}(\{1\})$ comptabilise le nombre de votes reçu par chaque pixel HR, ce qui correspond à la diagonale de la matrice normale $\mathbf{P}$ de l'opération de fusion (4.6). Cette étape utilise donc deux fois le schéma "vote" présenté en section 2.3.6. L'étape suivante interpole les valeurs manquantes de l'image SA et de la pondération et estime ensuite la valeur de l'image SM (4.6) en divisant point à point les intensités par la pondération. En termes de complexité, en utilisant les notations définies dans le premier chapitre (2) on a $f_{\mathrm{SR}} * [9, (1,1), N_{\mathrm{HR}}]$ c'est-à-dire qu'il faut lancer sur une grille ayant N_{HR} éléments f_{SR} kernels effectuant 9 opérations sur un voisinage 2D de rayon 1.

La dernière étape (figure 4.9) est la déconvolution par filtre Small Kernel (SK). Il s'agit donc d'une convolution non séparable (en général un rayon de 8 suffit) effectuée en géométrie HR qui fournit l'image SR résultat. Le coût est simplement $[4r+1, (r,r), N_{\mathrm{HR}}]$.

Interface SRviewer. L'algorithme ci-dessus a été implémenté dans un logiciel de traitement de séquences BR avec une interface permettant la visualisation de la séquence SR résultat et du

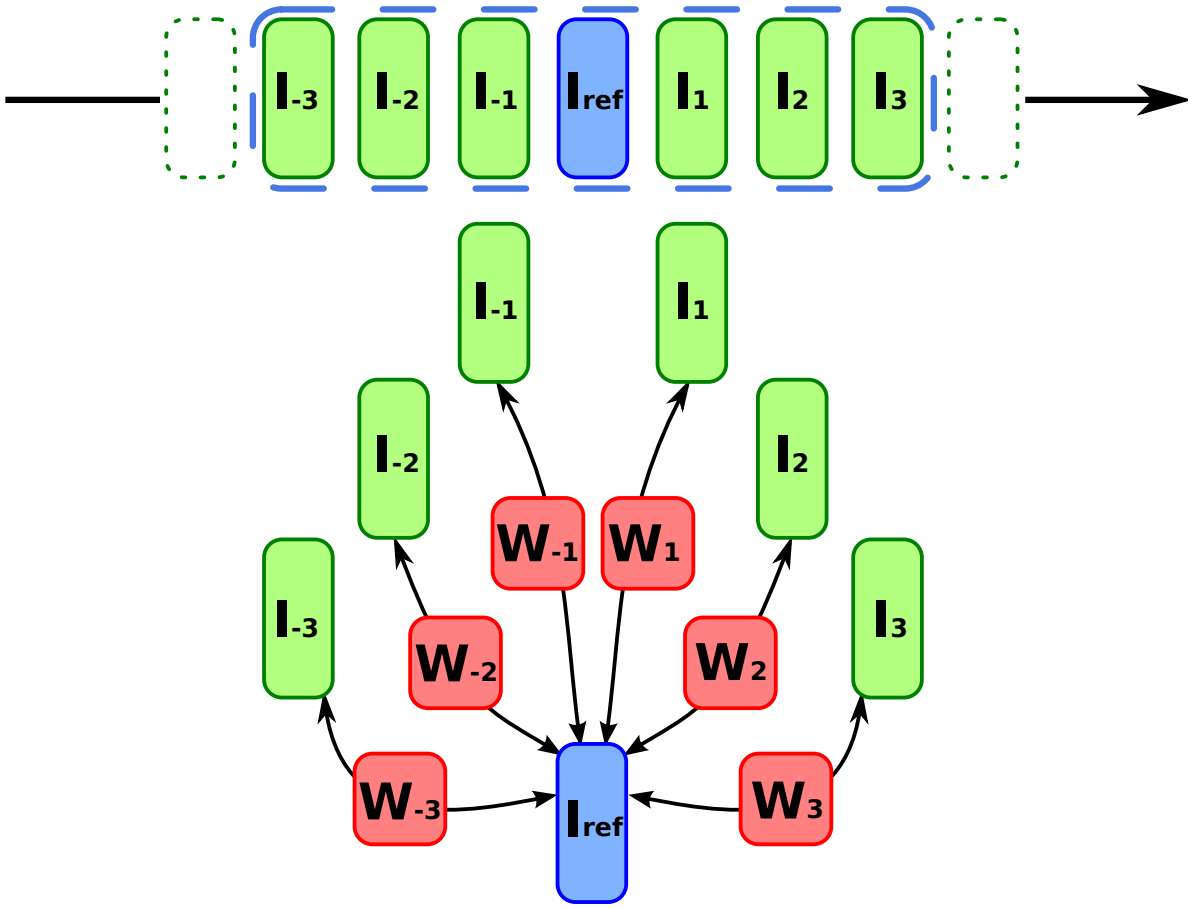


Figure 4.6 – Schéma de principe du recalage des images du GoP sur l'image BR de référence.

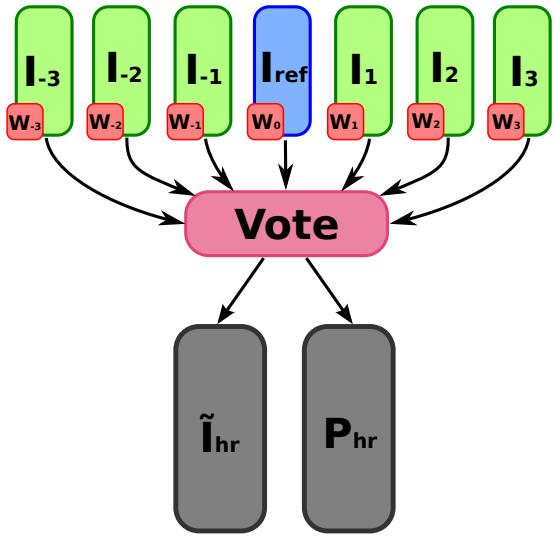


Figure 4.7 – Schéma de principe des opérations de vote donnant l'image SA et les pondérations.

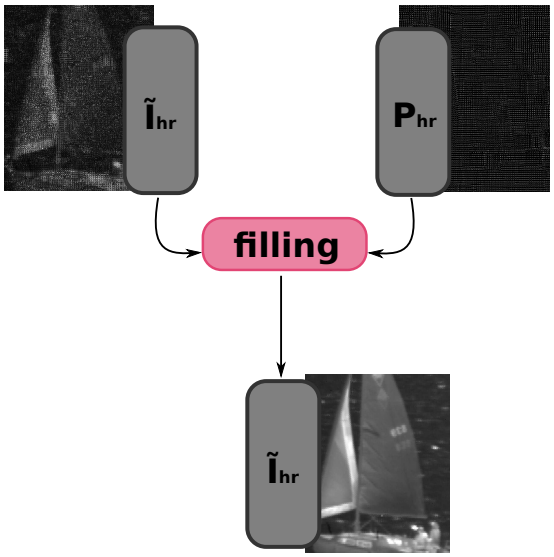


Figure 4.8 – Schéma de principe : bouchage de trous sur l'image SA et la pondération et division point à point, le résultat est l'image SM.

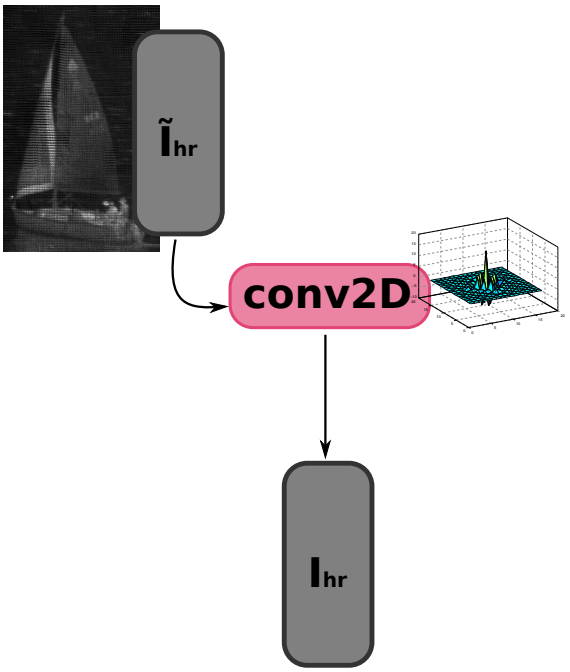


Figure 4.9 – Schéma de principe : Déconvolution par *small-kernel*.

coût de calcul, le choix des variantes algorithmiques et le réglage des paramètres en ligne. L'effet des choix effectués par l'utilisateur est directement observable sur la séquence SR résultat, ce qui permet la comparaison rapide de différentes solutions et une meilleure appréciation des différents réglages. Cette interface, nommée SRviewer, est brièvement présentée dans l'annexe C.

Un résultat. La Figure 4.10 présente des résultats de traitement sur la séquence infra-rouge Voilier, une séquence pour laquelle le recalage par flot optique est nécessaire pour traiter correctement l'objet mobile. On présente de gauche à droite, une image BR de la séquence d'origine zoomée bi-linéairement au facteur de SR choisi (ici 5.5), l'image SM, ("Shift and mean" avec bouchage de trous) issue des opérations décrites en section 4.3.1 et l'image après déconvolution par Small Kernel, dite SMSK. Les paramètres du noyau de déconvolution sont présentés en Table B.3.

Même avec un bouchage de trous très simple et une troncature des déplacements, l'image SM apporte un gain en résolution, les chiffres de l'immatriculation commencent à être lisibles. On peut remarquer que le fond présente des artefacts dûs à l'estimation de mouvement : en effet, l'hypothèse de constance des niveaux de gris n'est pas valide sur la surface de la mer. La déconvolution SK diminue nettement le flou, on peut lire clairement l'immatriculation et voir plus de détails sur les occupants et l'arrière du bateau. Ce gain se fait au prix de rebonds caractéristiques d'une technique quadratique avec un paramétrage à la limite de la sous-régularisation.

Coût de calcul. Passons maintenant à l'évaluation du coût de calcul. Les chronogrammes typiques présentés aux figures 4.11 et 4.12 donnent la durée totale de traitement et la répartition du temps passé entre les différentes étapes. Sur ces chronogrammes, fournis par le profileur CUDA, on visualise des boîtes correspondant aux différentes routines effectuées. On distingue par exemple la boîte "convolutionUnSepKer" qui correspond au filtre de déconvolution SK non séparable, dans la partie droite de la Figure 4.11. Les principales étapes de l'algorithme ont été soulignées et leur temps de calcul est indiqué.

Le chronogramme en figure 4.11 montre que l'estimation de mouvement est largement dominante. Notons qu'Antoine Létienne [Létienne, 2010], avec des outils de recalage différents et sur une architecture CPU classique, avait lui aussi observé que le coût de la partie mouvement occupe la majeur partie de la SR. Dans l'exemple de la figure 4.11, la charge du GPU est restée limitée (de l'ordre de 20 à 40%) au cours de l'estimation de mouvement. On pourrait donc facilement améliorer la performance de cette étape en utilisant les capacités d'exécutions asynchrone du GPU (les 'stream' en Cuda et 'queue' en OpenCL).

Figure 4.10 – SR rapide sur la séquence Voilier. De gauche à droite : image BR originale mise à l'échelle par interpolation bilinéaire (facteur de SR 5.5), Shift and mean avec bouchage de trous (SM), et Shift and mean et déconvolution par Small Kernel (SMSK). Les paramètres sont en table B.3.

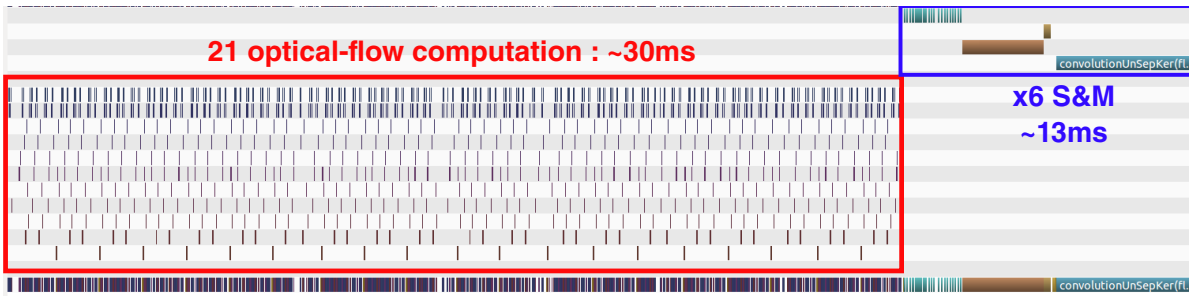


Figure 4.11 – Chronogramme de l'exécution de l'algorithme de SR rapide sur un groupe de 21 images de résolution 352×256, la partie estimation de mouvement est la plus coûteuse (30ms sur les 45ms), mais la charge du GPU sur cette partie de l'algorithme est faible (20 à 40%). La partie super-résolution par SMSK est extrêmement efficace malgré la grande taille de grille HR choisie ($\times 6$).

Le chronogramme de la partie reconstruction d'image de la SR est présenté en Figure 4.12. Notons d'abord que la durée totale est très faible (13ms) alors même que nous utilisons un facteur de SR très élevé ($\times 6$) afin de limiter l'erreur de troncature. Comme indiqué précédemment, l'algorithme SMSK associe des opérations qui sont extrêmement rapides sur GPU. Dans le détail, les opérations de vote sur la grille HR (cf. Figure 4.7) sont négligeables par rapport au bouchage de trou (Figure 4.7, boîte notée "fillK" dans le chronogramme) et à la convolution non séparable par le filtre SK.

Figure 4.12 – Zoom sur la partie super-résolution du chronogramme de la figure 4.11. On voit que la durée de la partie vote sur la grille HR est faible par rapport à celle de la convolution non séparable du Small-Kernel et au bouchage de trous.

4.4 SR itérative

La SR itérative consiste à rechercher une estimation de $\mathbf{I}_{\mathrm{HR}}$ par minimisation d'un critère régularisé incorporant les données disponibles, c'est-à-dire les M images BR $\mathbf{I}^i_{\mathrm{BR}}$ avec les champs de mouvements associés, et les informations a priori sur l'image SR recherchée, au travers d'un terme de régularisation. Dans la suite nous ne considérerons que l'optimisation par méthode de descente de gradient à pas fixe. Il existe de nombreuses méthodes plus sophistiquées qui peuvent améliorer la convergence, mais elles s'avèrent en général plus lourdes à mettre en œuvre sur GPU, ce qui n'est finalement pas forcément rentable. Le choix d'une unique méthode de minimisation, utilisée pour tous les critères que nous étudions, permet aussi de limiter le nombre de versions des algorithmes et le nombre de paramètres à régler.

Les différences entre méthodes proviennent alors du modèle de formation d'image utilisé (la section 4.2 en a présenté deux) et du choix des fonctions de pénalisation. Les pénalisations employées peuvent être quadratiques, ce qui simplifie l'optimisation, ou robustes, c'est-à-dire en pratique pénalisant moins les grands écarts que la quadratique, ce qui permet un comportement robuste de l'estimation : atténuation de l'influence des données aberrantes ou préservation des contours. Les techniques issues de ces différents critères sont présentées dans les sections

qui suivent.

4.4.1 Critère quadratique

En utilisant le modèle WarpHR on peut définir le critère régularisé :

$$E_{\text{WarpHR}}(\mathbf{I}_{\text{HR}}) = \frac{1}{N}\sum_{i=1}^{M} \left\|\mathbf{I}^i_{\text{BR}} - \mathbf{DBW}_i\, \mathbf{I}_{\text{HR}}\right\|^2 + \lambda\|\nabla \mathbf{I}_{\text{HR}}\|^2. \tag{4.9}$$

Ce critère est utilisé par exemple dans [Mitzel et al., 2009] pour développer une méthode de SR sur des séquences recalées par flot optique, donc dans un contexte très proche du nôtre. Le schéma de résolution par descente de gradient s'écrit :

$$\mathbf{I}^{k+1}_{\text{HR}} = \mathbf{I}^k_{\text{HR}} - \eta\left(\sum_{i=1}^{M}(\mathbf{W}_i)^t\mathbf{B}^t\mathbf{D}^t(\mathbf{DBW}_i\mathbf{I}^k_{\text{HR}} - \mathbf{I}^i_{\text{BR}}) + \lambda\nabla^2\mathbf{I}^k_{\text{HR}}\right). \tag{4.10}$$

Ou bien on peut utiliser le modèle WarpBR (4.2) pour écrire l'énergie suivante :

$$E_{\text{WarpBR}}(\mathbf{I}_{\text{HR}}) = \sum_{i=1}^{M} \left\|\mathbf{I}^i_{\text{BR}} - \mathbf{DW}_i\, \mathbf{BI}_{\text{HR}}\right\|^2 + \lambda\|\nabla \mathbf{I}_{\text{HR}}\|^2. \tag{4.11}$$

avec le schéma itératif

$$\mathbf{I}^{k+1}_{\text{HR}} = \mathbf{I}^k_{\text{HR}} - \eta\partial E_{\text{WarpBR}} \tag{4.12}$$

avec le gradient

$$\partial E_{\text{WarpBR}} = -\mathbf{B}^t\sum_{i=1}^{N}\mathbf{W}_i\mathbf{D}^t\mathbf{I}^i_{\text{BR}} + \mathbf{B}^t(\sum_{i=1}^{M}\mathbf{W}_i\mathbf{D}^t\mathbf{DW}_i)\mathbf{BI}^k_{\text{HR}} + \lambda\nabla^2\mathbf{I}^k_{\text{HR}} \tag{4.13}$$

Notons que le modèle WarpBR nous a servi, dans la section 4.3 à définir une technique non itérative en deux étapes : on peut donc penser que l'optimisation itérative de (4.11) n'apporte rien ici. En fait, comme remarqué précédemment, la résolution rapide en deux étapes ignore l'inhomogénéité des données re-projetées sur la grille HR, qui est prise en comte dans l'optimisation itérative de (4.11). En effet, la redondance des informations pour certains pixels HR se traduit par le fait que le pixel en question apparaît dans plusieurs résidus et réciproquement, les pixels qui ne sont pas renseignés n'apparaissent pas et leur valeur est interpolée par l'intermédiaire du terme de régularisation. Comme nous le verrons, cette différence peut conduire à une amélioration de la qualité des résultats.

Si l'on compare les deux schémas numériques fondés sur les modèles WarpHR ou WarpBR, il est facile de voir que le second est beaucoup plus efficace. En effet, on reconnaît dans le

gradient (4.13) des produits de l'opération de Shift-and-mean de la section 4.3.1. Le premier terme est l'image que nous avions appelé SA($\{\mathbf{I}^i_{\mathrm{BR}}\}$). Dans le second terme on reconnaît la matrice de pondération diagonale, dont les composantes sont SA($\{1\}$). Autrement dit ces deux éléments s'obtiennent rapidement par le procédé de vote décrit précédemment et peuvent être pré-calculés.

On entre ensuite dans la boucle d'optimisation, ce qui implique à chaque fois de :

- calculer le second terme de (4.13) soit deux convolutions séparables et une multiplication pixel-à-pixel sur la grille HR
- calculer le gradient du terme de régularisation $\nabla^2\mathbf{I}^k_{\mathrm{HR}}$ soit de nouveau deux convolutions séparables et une multiplication pixel-à-pixel sur la grille HR.
- Puis pour finir on effectue la remise à jour de l'estimée, ce qui correspond encore à une opération pixel à pixel sur la grille HR.

On constate qu'il n'y a que très peu d'opérations HR dans ce schéma (4 convolutions et 4 opérations pixels à pixels). Le schéma itératif déduit du modèle WarpHR implique beaucoup plus d'opérations HR, donc un coût supérieur. Ceci est confirmé par l'examen des chronogrammes typiques de ces deux solutions, présentés en section 4.4.4.

WarpBR vs. WarpHR. Le modèle WarpHR apporte-t-il un gain en qualité justifiant son surcoût ? Nos résultats montrent que ce n'est pas le cas sur les séquences réelles que nous avons traitées. Ces résultats ne sont pas contradictoires avec ceux de Gilles Rochefort [Rochefort et al., 2006], qui a observé des différences uniquement pour des facteurs de rotation ou de zoom importants, ce que nous n'avons pas dans nos séquences, et pour des régularisations très faibles. En pratique les différences sont donc la plupart du temps gommées par le niveau de régularisation nécessaire pour contenir le bruit (y compris le bruit de compression) sur la séquence. Nous ferons une comparaison plus précise sur un cas peu bruité en section 4.5.1. Enfin sur certaines séquences pour lesquelles le flot optique est difficile à estimer, le modèle WarpHR peut conduire à des instabilités, sans doute dues à l'interpolation du flot. C'est par exemple le cas sur la séquence Voilier, où le fond de mer ne satisfait pas à l'hypothèse du flot optique, cf. Figure 4.13.

La figure 4.14 présente, sur la séquence Voilier, les résultats comparés de l'approche de SR rapide (SMSK, à gauche) et de la SR quadratique itérative (à droite). Nous présentons aussi (image centrale) le résultat d'une déconvolution itérative appliquée sur l'image SM. Notons que toutes les méthodes sont paramétrées pour avoir le gain en résolution maximal ce qui conduit dans le cadre quadratique à des oscillations près des contours (légère sous-régularisation). Sur cette séquence on peut remarquer que la prise en compte des pondérations relatives des données est importante pour la qualité du résultat. Les deux images de gauche qui appliquent

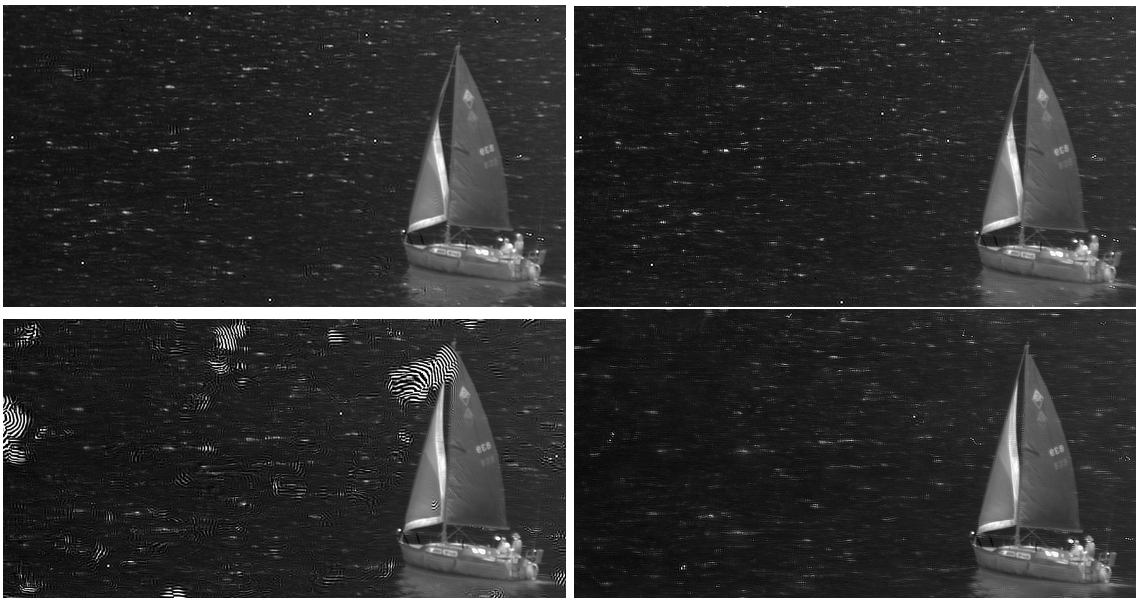


Figure 4.13 – SR quadratique avec modèle WarpHR (colonne de gauche) ou modèle WarpBR (colonne de droite). Sur la première ligne, traitement de 21 images, seconde ligne 61 images. Les résultats sur le voilier sont très proches pour les deux approches, mais la méthode WarpHR apparaît plus instable sur les zones où le recalage fonctionne mal, comme la zone de mer.

une déconvolution homogène, c'est-à-dire en considérant la sortie SM comme une image floue et bruitée par un bruit homogène, présentent des artefacts haute fréquence : bruit ponctuel autour des chiffres de l'immatriculation, effets de ligne sur l'arrière du bateau. La SR itérative, qui prend en compte les données BR d'origine et effectue la pondération correcte des données (et le bouchage de trous optimal au sens du critère régularisé choisi), fournit un meilleur résultat.

Ces différences sont cependant variables en fonction de la séquence traitée et souvent assez faibles, si le remplissage est très dense ou si d'autres problèmes viennent dégrader les reconstructions. En fait, l'intérêt des méthodes itératives réside surtout dans la possibilité d'obtenir des estimateurs plus sophistiqués comme ceux que nous présentons dans la suite.

4.4.2 Critère non quadratique sur la régularisation

Employés dans le terme de régularisation, les pénalisations robustes (norme L2-L1, norme L1, régularisation TV) permettent d'éviter le lissage excessif des zones de gradients d'une image (ou d'un flot optique comme nous l'avons vu dans le chapitre 3).

Forts des résultats de la section précédente, nous ne présentons dans la suite que des estimateurs fondés sur le modèle WarpBR, le seul qui permet de fournir des algorithmes rapides.

Figure 4.14 – SR quadratique sur la séquence Voilier. De gauche à droite : Shift and mean et déconvolution par Small Kernel (SMSK), image SM puis déconvolution L2 itérative, SR quadratique itérative. Les paramètres sont en table B.3.

La forme générale du critère est :

$$\mathbf{E}_{\phi(reg)}(\mathbf{I}_{\mathrm{HR}}) = \sum_{i=1}^{M} \left\| \mathbf{I}^i_{\mathrm{BR}} - \mathbf{D}\mathbf{W}_i\, \mathbf{B}\mathbf{I}_{\mathrm{HR}} \right\|^2 + \lambda\phi\left(\|\nabla \mathbf{I}_{\mathrm{HR}}\|^2 \right) \tag{4.14}$$

La différence réside donc dans la fonction de pénalisation ϕ. Nous utilisons ici les fonctions robustes telles qu'elles sont définie dans la référence [Zhang, 1995]). Avec une optimisation par un principe de moindres carrés repondérés, l'utilisation de ces fonctions ajoute simplement une repondération à l'équation de remise à jour vue dans le cas quadratique :

$$\mathbf{I}^{k+1}_{\mathrm{HR}} = \mathbf{I}^k_{\mathrm{HR}} - \eta \partial E_{\phi(reg)} \tag{4.15}$$

avec le gradient

$$\partial E_{\phi(reg)} = -\mathbf{B}^t \sum_{i=1}^{N} \mathbf{W}_i \mathbf{D}^t \mathbf{I}^i_{\mathrm{BR}} + \mathbf{B}^t \Big(\sum_{i=1}^{M} \mathbf{W}_i \mathbf{D}^t \mathbf{D} \mathbf{W}_i \Big) \mathbf{B}\mathbf{I}^k_{\mathrm{HR}} + \lambda \nabla \left(w(||\nabla \mathbf{I}^k_{\mathrm{HR}}||) \otimes \nabla \mathbf{I}^k_{\mathrm{HR}} \right) \tag{4.16}$$

Nous ne l'avons pas indiqué pour ne pas alourdir les notations, mais en pratique, la pénalisation non-quadratique ϕ et la fonction de repondération w dépendent d'un paramètre de seuil qui spécifie souvent la limite du régime quadratique (par exemple dans le cas de la fonction de Huber).

Le surcoût de l'utilisation d'une fonction de pénalisation non quadratique est donc négligeable à nombre d'itérations fixé, comparé au quadratique itératif. En revanche ce surcoût peut devenir

important pour un choix de paramètres rendant le critère plus long à optimiser (typiquement un choix rendant la pénalisation très proche de TV).

Dans une approche des moindres carrés repondérés, la fonction de repondération s'interprète comme un facteur permettant d'atténuer l'importance des régions de fort gradient dans le critère, ce qui permet de moins pénaliser ces gradients par rapport à la solution quadratique. Ce comportement est illustré par le résultat sur la séquence Voilier présenté en Figure 4.15, comparé à une solution SMSK et une solution quadratique itérative. Le travail de "nettoyage" de la pénalisation robuste apparaît par exemple sur les zones homogènes de la voile.

Figure 4.15 – SR sur la séquence Voilier. De gauche à droite : Shift and mean et déconvolution par Small Kernel (SMSK), SR quadratique itérative, SR itérative non quadratique avec une pénalisation robuste par fonction de Huber. Les paramètres sont en table B.3.

Parmi les effets indésirables liés à l'utilisation d'une pénalisation non quadratique sur la régularisation, il y a la tendance à rendre les variations d'intensité sous forme de plateaux séparés par des différences importantes. C'est l'"effet cartoon" décrit par plusieurs auteurs, par exemple [Grasmair and Lenzen, 2010], et que nous illustrons sur la séquence Voilier en Figure 4.16.

Dans la version présentée à droite de la Figure 4.16, les chiffres inférieurs de l'immatriculation commencent à disparaître, car il sont lissés avec la zone homogène de la voile. Notons que les auteurs qui préfèrent ces approches à la SR quadratique dans le contexte de la SR [Mitzel et al., 2009, Unger et al., 2010], insistent souvent sur des zones très contrastées que cet effet cartoon améliore, typiquement les zones d'écriture comme des plaques d'immatriculation, et ne commentent pas toujours les pertes de nuances dans le rendu des zones présentant des variations faibles d'intensité. C'est le cas par exemple de la voiture de la Figure 7

Figure 4.16 – SR sur la séquence Voilier. De gauche à droite : SR quadratique itérative, SR itérative non quadratique avec une pénalisation robuste par fonction de Huber effet cartoon modéré, effet cartoon fort. Les paramètres sont en table B.3.

de l'article [Mitzel et al., 2009] : la plaque d'immatriculation est plus lisible en L1, mais les détails de l'intérieur visibles par le pare-brise sont perdus, alors qu'ils sont observables dans la version L2.

4.4.3 Gestion des données aberrantes

Les critères vus précédemment sont fondés sur une hypothèse de bruit gaussien homogène sur les données. La pondération $\mathbf{P}$ de la section 4.3.1 permet de tenir compte de la redondance des données BR colocalisées (au sens de la grille HR), mais pas de différences de qualité entre données BR. Dans la pratique, il y a des données aberrantes, c'est-à-dire des valeurs de l'intensité image dont la probabilité est très faible dans le cadre des modèles de formation d'image à bruit additif gaussien présentés en section 4.2.

Validation aller/retour. Une première observation consiste à dire que la plupart des données aberrantes proviennent d'exceptions locales à l'hypothèse de constance des niveaux de gris. Or cette hypothèse sous-tend aussi l'estimation du flot optique. Autrement dit, on peut espérer détecter un certain nombre de données aberrantes par un échec du recalage local par flot optique. Nous proposons de détecter ces échecs par la mesure de qualité du flot par aller/retour, mesure présentée dans la section 3.5.1. Cette mesure de qualité est ensuite utilisée pour pondérer les votes lors de l'étape de shift-and-mean qui débute le calcul SR lorsqu'il est fondé sur le modèle WarpBR. Avec des pondérations binaires, cela conduit à des trous supplémentaires dans les données, qui sont comblés soit par l'étape de bouchage dans l'algorithme rapide, soit au cours de l'optimisation par l'influence du terme de régularisation dans la version itérative.

Figure 4.17 – SR sur la séquence Aérien2. De haut en bas et de gauche à droite : Shift-and-Mean (SM) sans bouchage avec recalage par flot optique ; SM sans bouchage avec recalage par flot optique et validation aller/retour ; résultat SR sans validation aller/retour ; résultat SR avec validation aller/retour. Les paramètres sont en table B.3, le paramètre de sélection A/R est 0.2.

La Figure 4.17 présente un résultat sur la séquence Aérien2, séquence qui est affectée d'un très fort bruit de compression, ce qui gêne le recalage. La ligne du haut compare le résultat du Shift-and-Mean appliqué aux séquences recalées avec ou sans validation aller/retour du flot. Le remplissage sans validation est bien plus dense. Mais ces données sont souvent aberrantes, ce qui conduit à une image SR très dégradée (en bas à gauche) par rapport à celle utilisant moins de données sélectionnées par le critère de validation aller/retour.

Remarquons qu'on pourrait facilement étendre cette stratégie en utilisant plusieurs estimateurs de mouvements, avec la mesure de qualité aller/retour pour pondérer un ensemble d'hypothèses d'estimation de flot optique obtenus par différents algorithmes par exemple. Néanmoins pour des raisons de coût de calcul (qui augmente bien entendu avec le nombre d'hypothèses de mouvement à estimer) nous n'avons pas poursuivi dans cette voie, qui s'apparente aux méthodes de "probabilistic motion" de la référence [Protter et al., 2009]

Pénalisation robuste sur les données. L'approche précédente est utile pour améliorer le résultat dans les cas difficiles, cependant elle est tributaire de la performance de l'algorithme de flot optique. Nous avons fait le choix d'un algorithme rapide (FOLKI) mais qui peut localement conduire à des résultats biaisés, en particulier à cause de l'influence de sa fenêtre de corrélation qui tend à empâter les zones en mouvement. Si l'on veut pouvoir remettre en cause ces recalages et corriger d'autres aberrations il faut utiliser une pénalisation robuste sur le terme de données, comme cela est fait classiquement en traitement d'image. Cela conduit au critère :

$$\mathbf{E}_{\phi\phi}(\mathbf{I}_{\mathrm{HR}}) = \sum_{i=1}^{M} \phi^a \left(|\mathbf{I}^i_{\mathrm{BR}} - \mathbf{D}\mathbf{W}_i \, \mathbf{B}\mathbf{I}_{\mathrm{HR}}\|^2 \right) + \lambda \phi^r \left(\|\nabla \mathbf{I}_{\mathrm{HR}}\|^2 \right), \tag{4.17}$$

où ϕ^a désigne la pénalisation du terme d'adéquation et ϕ^r celle du terme de régularisation.

En adoptant une approche par moindres carrés repondérés, il est facile de voir que cette pénalisation ajoute simplement une pondération dans les opérations de vote qui conduisent à l'image SA, pondération variable au cours des itérations. La remise à jour s'écrit :

$$\mathbf{I}^{k+1}_{\mathrm{HR}} = \mathbf{I}^k_{\mathrm{HR}} - \eta \partial E_{\phi\phi} \tag{4.18}$$

avec le gradient

$$\begin{aligned} \partial E_{\phi\phi} &= -\mathbf{B}^t \sum_{i=1}^{N} w^a(||\mathbf{W}^t_i \mathbf{D}^t \mathbf{I}^i_{\mathrm{BR}} - \mathbf{I}_{\mathrm{HR}}||^2) \mathbf{W}_i \mathbf{D}^t \mathbf{I}^i_{\mathrm{BR}} && (4.19) \\ &+ \mathbf{B}^t (\sum_{i=1}^{M} \mathbf{W}_i \mathbf{D}^t w^a(||\mathbf{W}^t_i \mathbf{D}^t \mathbf{I}^i_{\mathrm{BR}} - \mathbf{I}_{\mathrm{HR}}||^2) \mathbf{D}\mathbf{W}_i) \mathbf{B}\mathbf{I}^k_{\mathrm{HR}} && (4.20) \\ &+ \lambda \nabla \left(w^r(||\nabla \mathbf{I}^k_{\mathrm{HR}}||) \otimes \nabla \mathbf{I}^k_{\mathrm{HR}} \right) && (4.21) \end{aligned}$$

Une conséquence de la repondération est qu'il est nécessaire de refaire les opérations de vote à chaque itération, alors qu'elles sont faites uniquement dans la phase initiale avec les autres algorithmes. Le surcoût sera évalué dans la section suivante. Notons qu'une fois encore le modèle direct WarpBR rend la mise en œuvre d'une méthode de rejet des données aberrantes beaucoup moins coûteuse que ce qu'on pourrait faire avec le modèle WarpHR.

Le résultat présenté en Figure 4.18 montre le potentiel de cette approche qui fournit une qualité nettement meilleure que les résultats précédents. D'autres exemples seront présentés dans la section résultats 4.5. Le prix à payer est le réglage de deux seuils (associés aux fonctions ϕ^a et ϕ^r) en plus du paramètre de régularisation λ.

Figure 4.18 – SR sur la séquence Voilier. SR itérative avec pénalisation de Huber sur la régularisation et sur les données. Les paramètres sont en table B.3. Contenu vidéo : séquence SR, à cause de la repondération des données, des divergences sont visibles sur le fond de mer qui ne respecte pas la contrainte de constance de l'intensité

4.4.4 Mise en oeuvre et durée d'exécution

La mise en œuvre des algorithmes itératifs a déjà été décrite dans les sections 4.4.1 et 4.4.2. La gestion des données aberrantes implique essentiellement le calcul de plusieurs champs de mouvement (flot aller/retour) et le calcul d'une pondération (opération pixel-à-pixel) avant les votes.

Comme pour la SR rapide, les algorithmes itératifs décrits ci-dessus ont été intégrés dans SRviewer, un logiciel de traitement de séquences BR avec une interface permettant la visualisation de la séquence SR résultat et du coût de calcul, le choix des variantes algorithmiques et le réglage des paramètres en ligne. L'effet des choix effectués par l'utilisateur est directement observable sur la séquence SR résultat, ce qui permet la comparaison rapide de solutions et de réglages. Cette interface est présentée dans l'annexe C.

En ce qui concerne le temps d'exécution, la table B.5 résume la situation pour une séquence d'images de taille 256×120. Les temps de calcul pour des tailles d'images croissantes sont repris plus systématiquement dans la section B.2. Dans ces tables seuls les paramètres ayant un impact sur le temps de calcul sont mentionnés. Toutes les méthodes étant basées sur les mêmes recalages, la comparaison porte uniquement sur la construction de l'image SR elle-même. Les principales conclusions sont les suivantes :

- le surcoût du modèle WarpHR est très important, d'un facteur 9 pour une SR $\times 3$ et 17 pour une SR $\times 6$: ce modèle ne permet pas d'approcher la cadence vidéo en SR ;
- la SR SMSK est *vraiment* rapide, avec un coût de quelques millisecondes négligeable devant le coût du recalage (10 fois supérieur) ;
- le coût des SR itératives est un ordre de grandeur supérieur à celui de SMSK, il est entre 1 à 2 fois le coût du recalage. Ces méthodes peuvent tourner pour cette taille d'image à 10Hz si on limite le nombre d'itérations à 20. En pratique, nous allons rarement au-delà, sauf pour des réglages très particuliers (régularisation faible ou très L1).
- le surcoût de la repondération des données (5ème colonne), méthode dont nous verrons qu'elle permet d'obtenir de très bons résultats dans certaines séquences difficiles, est entre 20% (cas à faible nombre d'images et grand facteur SR) et 70% (conditions inverses).

Pour entrer un peu plus dans les détails, nous présentons ensuite des chronogrammes obtenus avec le profileur CUDA. La figure 4.19 présente le chronogramme de la SR itérative quadratique sur modèle WarpHR, avec un zoom sur la partie reconstruction d'image en 4.20. La figure 4.21 présente le chronogramme de la SR itérative non quadratique sur modèle WarpBR, avec un zoom sur la partie reconstruction d'image en 4.22.

Ces représentations confirment que le modèle WarpHR conduit à un coût de calcul bien plus élevé que le modèle WarpBR (de l'ordre de 10 fois plus élevé). Ce surcoût est dû au

		SM+SK	SR-φ	SR-ϕ-ψ	WarpHR
GoP de 11 images (FO = 20.5ms) — (FO A/R = 42ms)					
sr x3	10	0.58	8.3	11	73
	20		16	22	147
	50		40	57	368
	100		80	113	734
sr x6	10	2.24	27	32	230
	20		54	65	464
	50		133	160	1152
	100		264	320	2310
GoP de 21 images (FO = 41ms) — (FO A/R = 83ms)					
sr x3	10	0.91	8.65	14	135
	20		16	28	271
	50		40	70	676
	100		81	140	1361
sr x6	10	2.63	28	36	424
	20		53	73	850
	50		133	183	2124
	100		265	366	4248

Table 4.1 – Tableaux des temps de calcul des algorithmes de super-résolution, pour une séquence de résolution 256×120 (les temps sont donnés en millisecondes).

plus grand nombre d'opérations en géométrie HR lors de l'évaluation du gradient du terme de données, cf. 4.20.

Cette comparaison s'appuie sur la version non quadratique du critère fondé sur le modèle WarpBR, mais le chronogramme 4.22 montre par ailleurs que le surcoût lié au critère non quadratique est faible, de moins de 10% de la partie reconstruction.

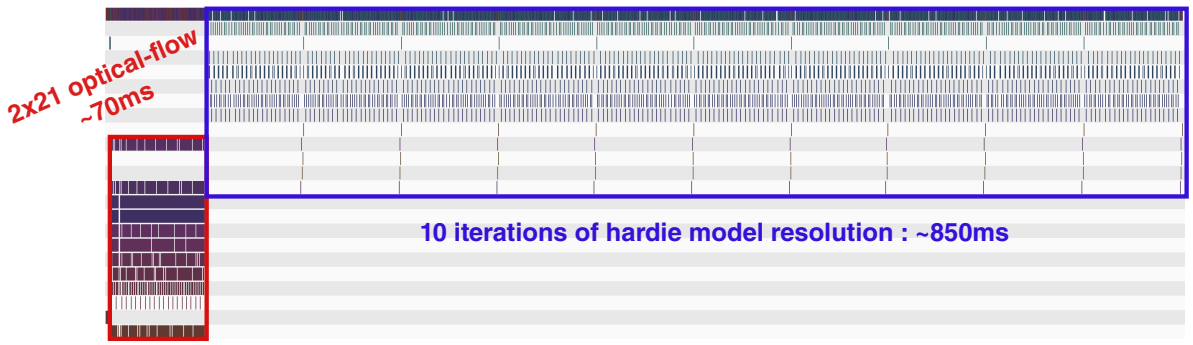


Figure 4.19 – Chronogramme de l'exécution de l'algorithme de descente de gradient à pas fixe sur le modèle WarpHR avec une régularisation quadratique.

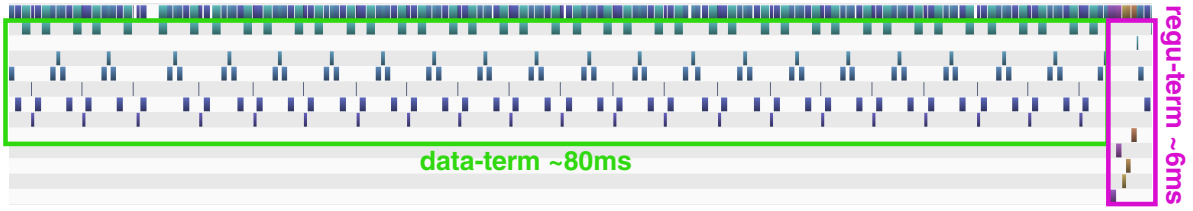


Figure 4.20 – Zoom sur une itération (soit environ 85ms) de la partie super-résolution du chronogramme de la figure 4.19. On voit que la partie fusion des données d'entrée (data-term) est prédominante par rapport au coût de la partie régularisation (regul-term).

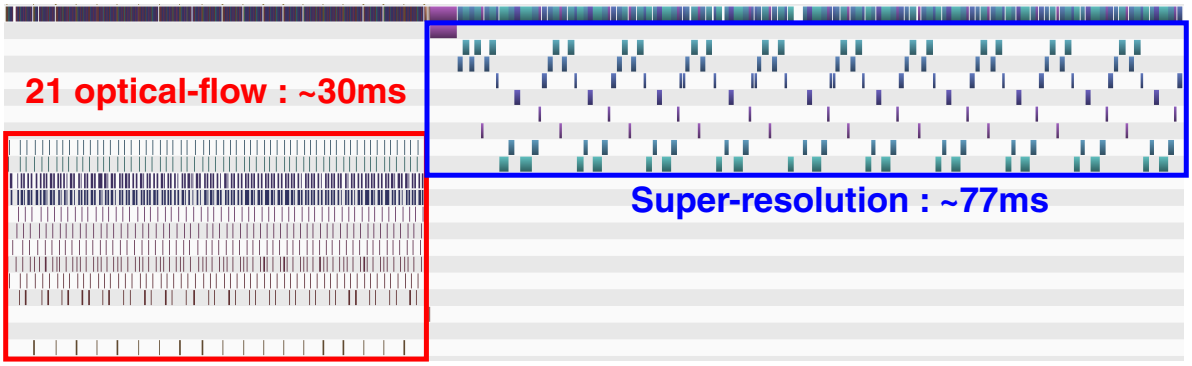


Figure 4.21 – Chronogramme de l'exécution de l'algorithme de descente de gradient à pas fixe sur le modèle WarpBR avec une régularisation non-quadratique.

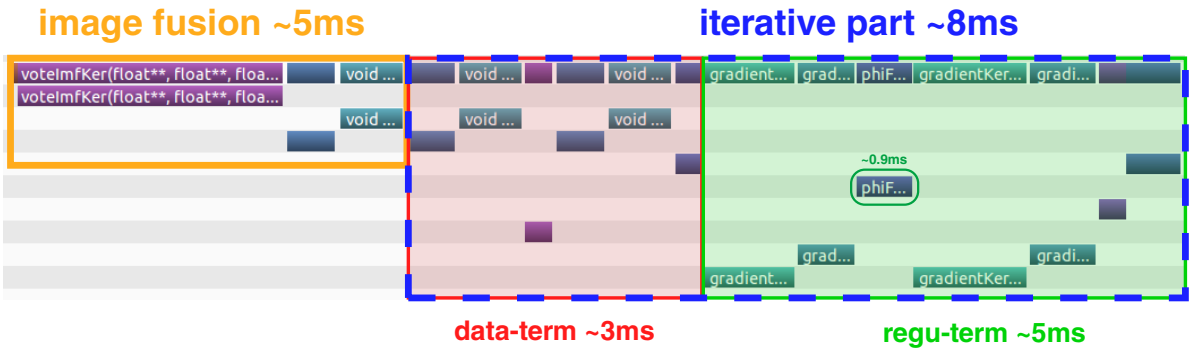


Figure 4.22 – Zoom sur la première itération (soit environ 13ms) de la partie super-résolution du chronogramme de la figure 4.21. La différence entre une version quadratique et non-quadratique réside dans le noyau entouré et est inférieur à 1ms par itération.

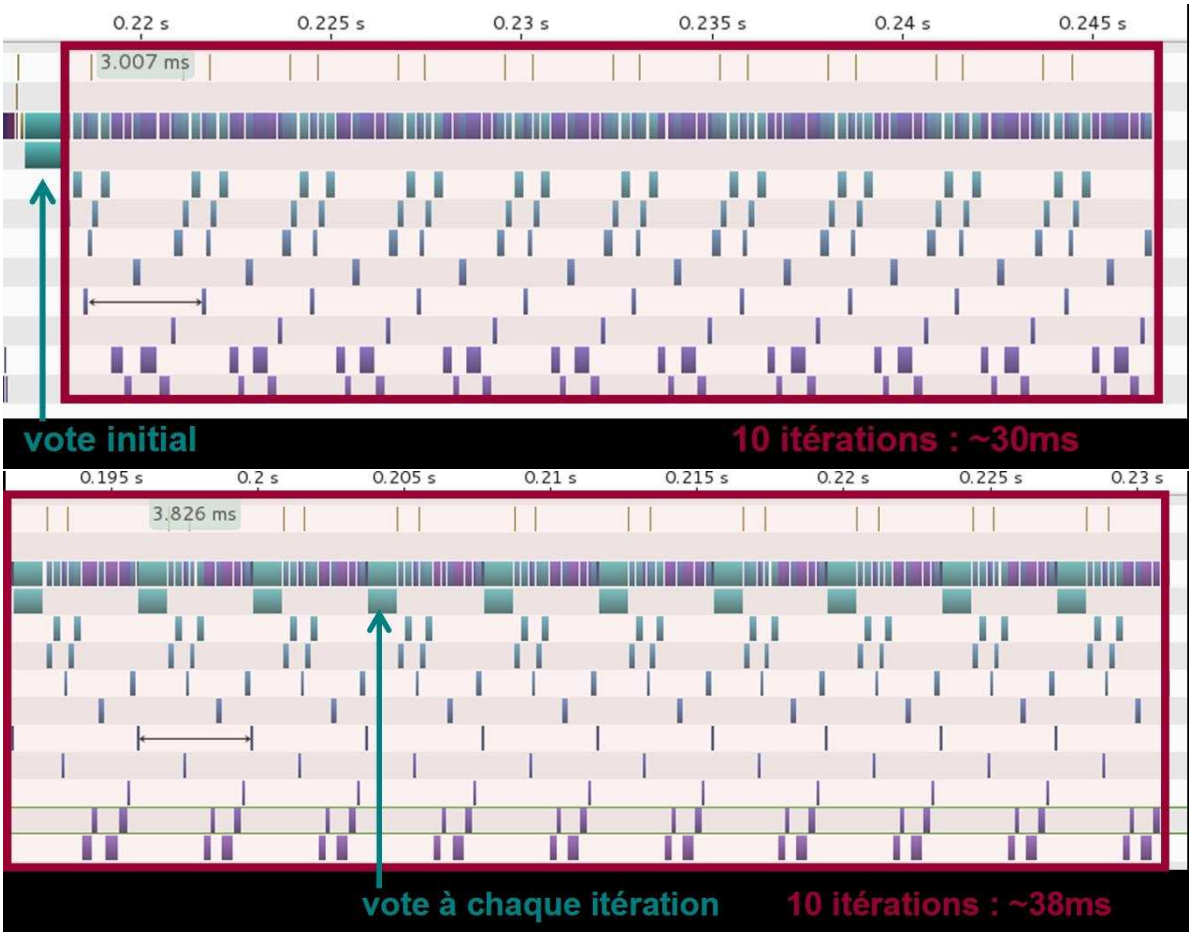


Figure 4.23 – Chronogrammes de la partie SR : en haut, algorithme SR itératif avec pénalisation non quadratique sur la régularisation, en bas avec pénalisation non quadratique sur terme de données et régularisation. La différence est la présence du noyau de vote à chaque itération de la seconde version. En termes de coût total, en tenant compte du vote initial obligatoire, cela ne fait que 7ms de différence sur 10 itérations.

4.5 Super-résolution sur données réelles : résultats et discussions

Après avoir proposé plusieurs algorithmes de SR rapide sur flot optique, nous présentons dans cette section une série de résultats sur données réelles, afin d'explorer le domaine d'emploi de ces techniques. Les sections suivantes présentent en particulier une série d'exemples sur des mires de résolution (section 4.5.1) ; des résultats sur des exemples très bruités ou dégradés par des artefacts de compression (section 4.5.2 et 4.5.3) ; enfin des résultats sur des champs de mouvement complexes, en fonction de l'estimateur de mouvement choisi (mouvement translationnel ou flot optique, flot optique) (section 4.5.4).

Il est important de souligner qu'en pratique, tous les traitements que nous effectuons sont appliqués sur un bloc glissant d'images extraites de séquences vidéo de sorte à fournir en sortie des séquences avec un échantillonnage temporel identique à celui de l'entrée. Ces résultats ne sont pas forcément fournis à la cadence d'origine, cela dépend de la performance du traitement choisi. Pour les besoins de ce manuscrit, ces résultats seront illustrés par la comparaison des images extraites des flux d'entrée (basse résolution) mises à l'échelle, et des images extraites des flux de sortie (SR) après application de différents traitements. Ce type de comparaison sur image fixe est sans doute plus objectif, car lors de la comparaison visuelle de séquences vidéo BR et SR, la vision humaine a tendance à faire sa propre super-résolution sur la séquence BR.

Le réglage des algorithmes est ad hoc, il a été fait grâce à SRviewer, cf. Annexe C. L'ensemble des paramètres utilisés pour générer les figures fait l'objet de la Table récapitulative B.3.

4.5.1 Mires de résolution en IR

Les résultats de cette section ont été obtenus sur une séquences IR d'évaluation réalisées par la DGA pour le groupe OTAN SET140, voir la référence [Weißet al., 2012]. Le dispositif d'acquisition comprend plusieurs caméra IR montées sur un support mobile animé d'un mouvement de précession : la figure 4.24 présente ce dispositif et les déplacements typiques effectués par le support.

La figure 4.25 présente une image de la séquence 'Camion', sur laquelle les zones de comparaison sont indiquées. Ces zones comprennent en particulier deux panneaux avec des mires, l'un avec des mires de résolution à bâtons classiques et l'autre avec des mires formées de triangles orientés (TOD pour *Triangle Orientation Discrimination*) proposées pour la caractérisation de performance de la SR par une équipe de TNO dans la référence [van Eekeren et al., 2007]. Les 8 mires à bâtons de la mire de droite, que nous dénommerons par leur numéro dans l'ordre

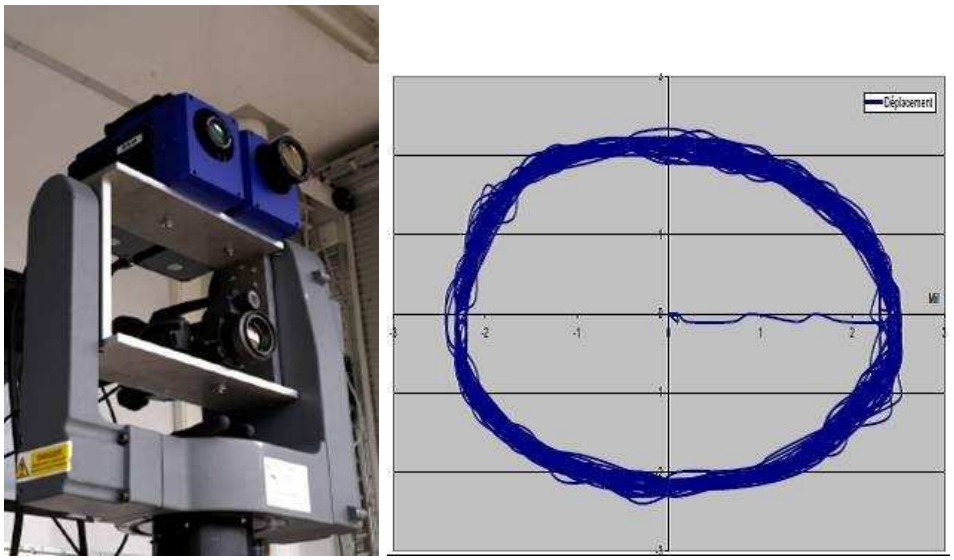


Figure 4.24 – Dispositif d'acquisition mis en place par la DGA pour l'évaluation d'algorithmes de SR sur images réelles Infra-rouges (images extraites de [Weißet al., 2012]). Trois caméras IR sont montées sur un support mobile : celle dont nous traitons les images dans cette section est une caméra bolométrique FLIR SC660 de résolution 640×480 pixels. Le support est animé de mouvements de précession tel que celui présenté à droite.

de leur taille présentent des tailles (donc des périodes spatiales) en progression quasi-linéaire. Elles permettent de quantifier grossièrement la limite de résolution d'une image sortie du capteur ou après traitement SR.

Paramétrage standard

La PSF de cette séquence peut être estimée par une technique de bord de plage (en anglais on parle de "*knife-edge method*", voir [Champagnat et al., 2009] et les références de cet article). Il s'agit d'identifier la réponse à un échelon en utilisant un bord droit séparant deux zones de niveaux différents, comme celui se trouvant sur la mire près de la roue arrière du camion en Figure 4.25 puis de fitter l'intégrale d'une PSF Gaussienne à cette réponse. En collectant plusieurs profils 1D perpendiculaires au contour qui doit être oblique par rapport aux lignes/colonnes de l'image, cette technique permet d'estimer la réponse du système non replié. La figure 4.26 illustre cette technique qui conduit ici à un écart-type estimé de 0.43 (en pixels BR).

Dans cette section, nous utilisons cette valeur de PSF pour utiliser des techniques de SR à facteur 5 en utilisant 21 images de la séquence.

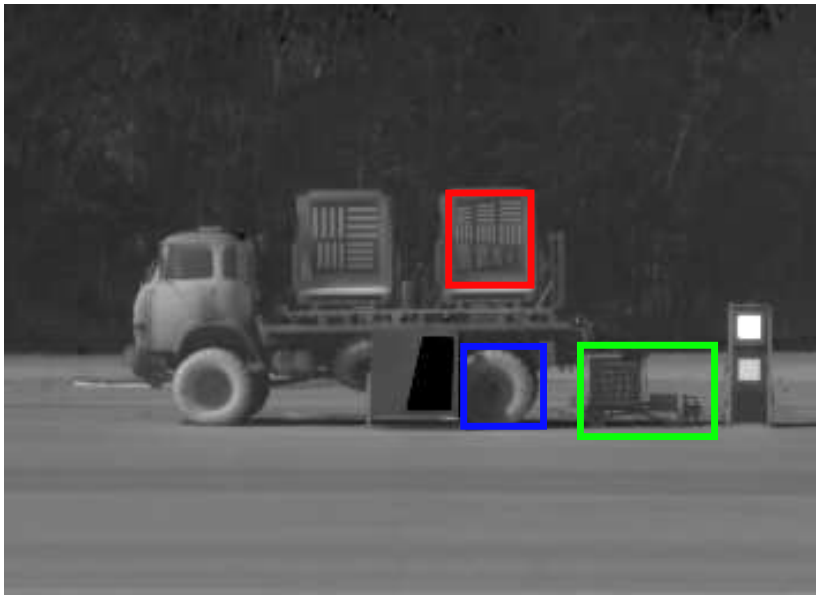


Figure 4.25 – Localisation des zones d'intérêt sur la séquence Camion. Dans la suite nous présentons des résultats dans les zones contenant des mires (zone rouge et verte).

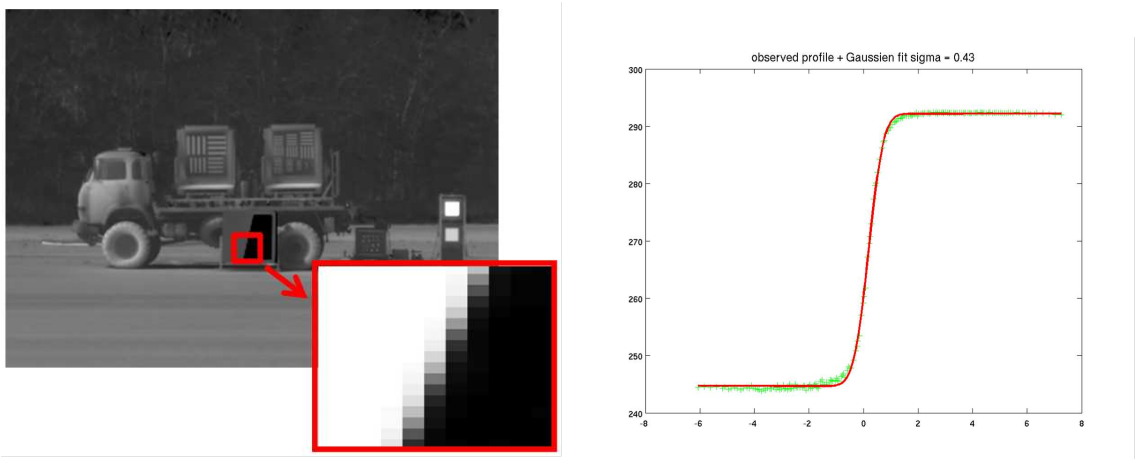


Figure 4.26 – Estimation de la PSF du capteur IR sur la séquence Camion. A gauche, contour extrait de l'image ; à droite, adéquation des profils avec l'intégrale d'une gaussienne d'écart-type 0.43, voir texte.

La Figure 4.27 montre d'abord l'apport brut de la SR rapide par SMSK en comparaison d'un zoom spatial (bilinéaire) x5 sur une image de la séquence BR d'origine. On constate que sur l'image BR les mires sont résolues jusqu'à la mire 3, alors que l'image BR permet de décompter les barres jusqu'à la mire 5. La mire 6 est en limite de résolution, nous verrons qu'elle peut être résolue avec un réglage adapté des paramètres. Connaissant la progression des fréquences spatiales des mires on peut en déduire que le gain effectif en résolution est entre $1,5$ (mire 5 résolue facilement) et $2,0$ (mire 6 à la limite de résolution), plus proche de 2. Notons que le facteur de SR utilisé dans les reconstructions présentées est beaucoup plus important ($f_{\mathrm{SR}} = 5$). Comme la référence [Champagnat et al., 2009] nous avons nous aussi constaté qu'il est utile d'avoir un facteur de SR supérieur au gain de résolution effectif pour avoir les meilleurs résultats. De plus, comme nous utilisons des déplacements tronqués en pas entier de la grille HR, ces facteurs élevés permettent aussi de limiter l'erreur de troncature.

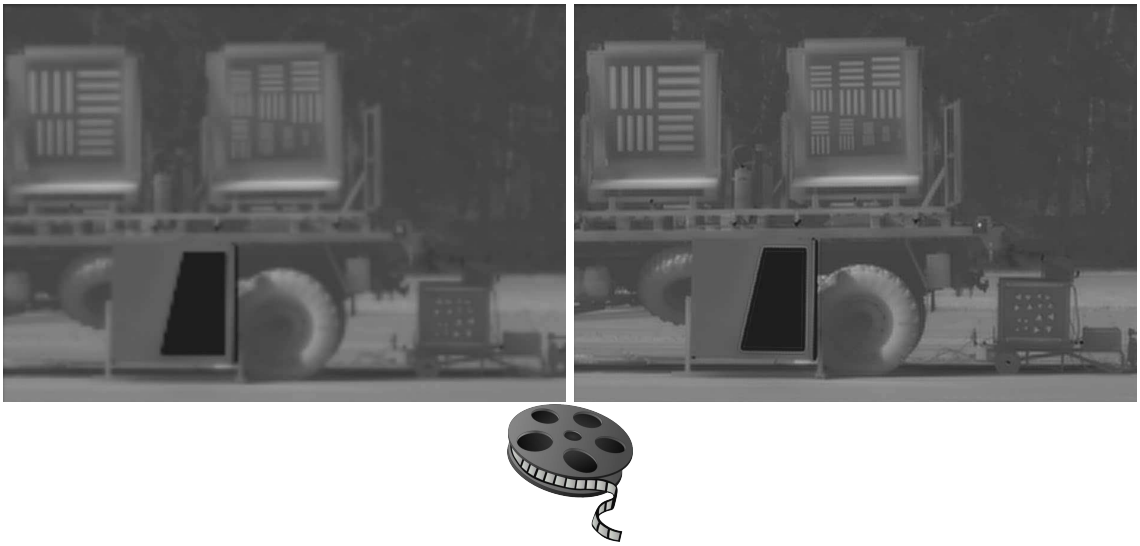


Figure 4.27 – Apport de la SR sur la séquence Camion. A gauche : image BR, à droite : SR rapide par SMSK. Le gain en fréquence spatiale (mires "dépliées") est net.

La Figure 4.28 se focalise sur les résultats SR sur la mire à bâtons en comparant les différentes techniques proposées, de haut en bas et de gauche à droite : SR rapide SMSK, SR itérative quadratique, idem sous-régularisée, SR non quadratique avec pénalisation de Huber. On observe d'abord peu de différences dans cet exemple entre la SR rapide SMSK et la SR quadratique (il y a uniquement un peu plus d'oscillations autour de certains bâtons) ce qui s'explique par le bon remplissage de cette séquence à mouvement contrôlé. Le résultat quadratique sous-régularisé montre que la mire 6 verticale peut être résolue — au prix d'un bruit de reconstruction important — en revanche la mire 6 horizontale n'est pas résolue. Nous

reviendrons sur la résolution limite dans cet exemple un peu plus loin. Enfin la pénalisation de Huber permet de lisser de manière différente les zones homogènes et les zones de gradient, ce qui conduit à un résultat plus favorable sur ces images IR de mire. Mais il n'y a pas de gain en résolution, voire même une perte, surtout lorsque l'on pousse trop loin le paramètre de seuil (effet cartoon), cf. Figure 4.29.

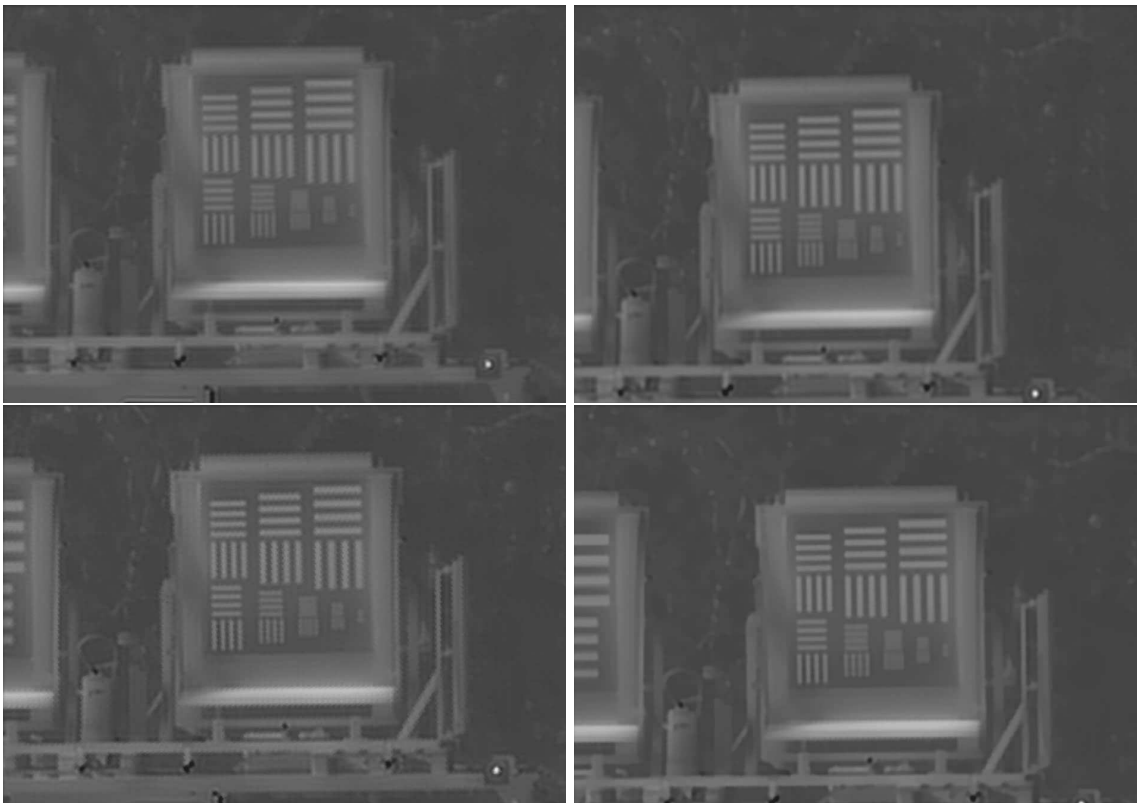


Figure 4.28 – Comparaison de techniques de SR sur la séquence Camion. HB-GD : SR rapide SMSK, SR itérative quadratique, idem sous-régularisée, SR non quadratique avec pénalisation de Huber. Les paramètres de traitement sont donnés en Table B.3.

En ce qui concerne la mire TOD, on présente deux résultats en Figure 4.30, l'un par SMSK et l'autre par pénalisation de Huber avec un réglage très non linéaire. Le résultat Huber peut sembler meilleur car les triangles sont quasiment segmentés. En regardant de plus près, on voit que la régularisation Huber isotrope a tendance à raboter les triangles et que leur orientation est sans doute plus difficile à estimer que sur la SR quadratique.

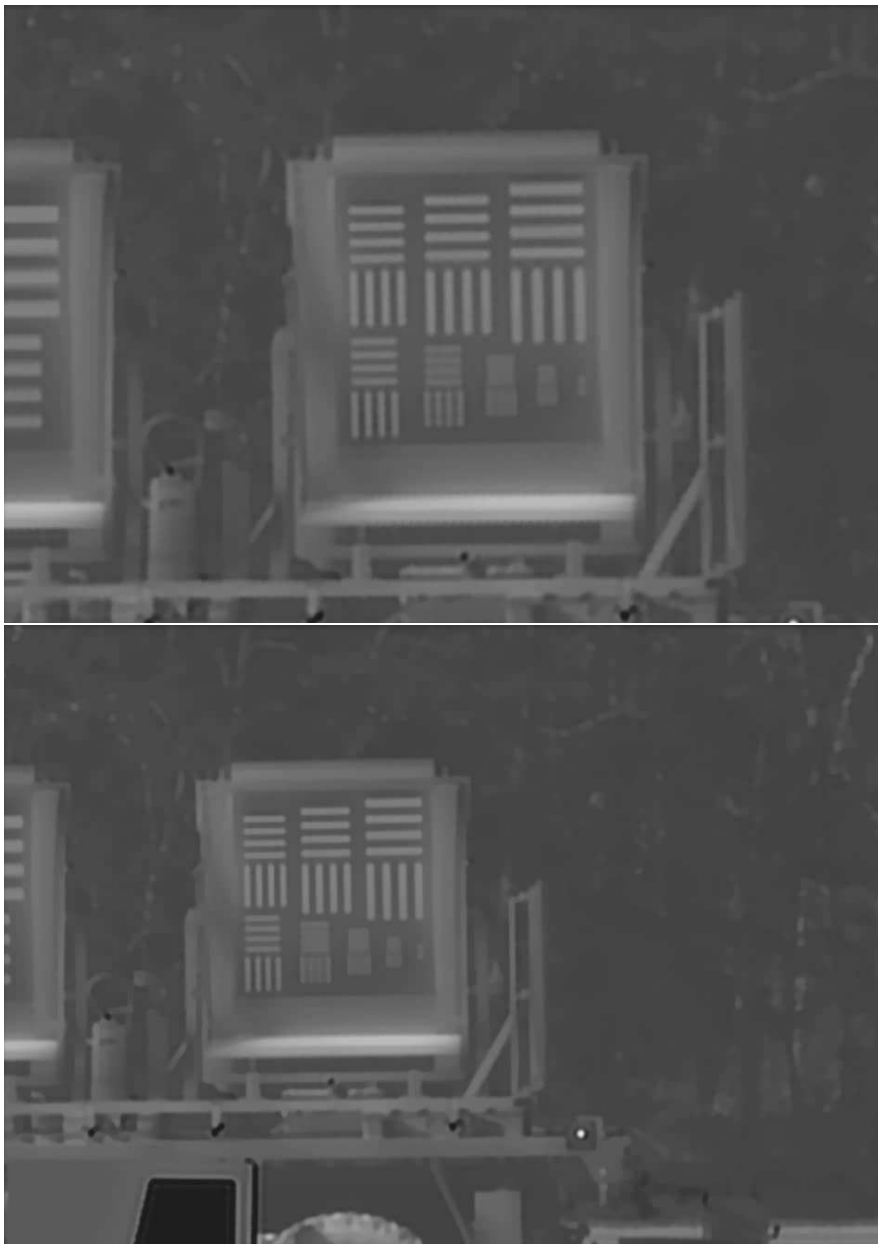


Figure 4.29 – Effet cartoon de la régularisation non quadratique sur la mire à bâton de la séquence Camion : en bas réglage de seuil amplifiant cet effet, la mire numéro 6 commence à devenir non résolue. Les paramètres de traitement sont donnés en Table B.3.

Figure 4.30 – Effet cartoon de la régularisation non quadratique sur la mire TOD de la séquence Camion : en haut, SMSK, en bas SR itérative et pénalisation Huber avec un réglage très non linéaire. Les paramètres de traitement sont donnés en Table B.3.

Etude de paramètres limites

Dans cette section nous cherchons à pousser les algorithmes de SR à leurs limites en jouant sur le facteur SR et sur la largeur de la PSF. En effet, nous nous sommes rendus compte que l'utilisation d'un facteur de SR élevé (typiquement 7 et plus) et d'une PSF plus étroite que la PSF estimée permet de restaurer plus de hautes fréquences, au prix d'artefacts importants. Un exemple est présenté en figure 4.31 : on a utilisé une SR itérative quadratique avec une PSF gaussienne d'écart-type $\sigma = 0.33$ (au lieu de la valeur estimée de 0.43) et un facteur de SR $f_{\mathrm{SR}} = 15$. Lorsque l'on affiche tout le champ, le résultat semble excellent puisque la mire 6 est résolue (la verticale nettement, l'horizontale moins). Le zoom présenté dessous confirme ce gain en résolution effectif de 2, mais montre que l'image est dégradée par un effet de damier.

Cette dégradation résulte du procédé de Shift-and-Mean et de la troncature des déplacements. La figure 4.32 montre un détail extrait d'une reconstruction SR utilisant 21 images et, en regard, l'image "Shift-and-Mean" (SM) avant bouchage de trous. On peut observer que la trame qui apparaît sur la reconstruction correspond aux trous de l'image SM. Autrement dit, le procédé SM qui est à la base de toutes les méthodes de SR présentées dans ce travail, qu'il s'agisse de la SR rapide ou des techniques itératives, est à l'origine d'un biais sur les reconstructions d'autant plus important que l'on régularise peu, qu'on utilise un facteur de SR élevé et qu'on sous-estime la taille réelle de la PSF.

A ce stade, on peut penser que le modèle WarpHR, que nous avons laissé de côté dans les sections précédentes, ne conduit peut-être pas aux mêmes artefacts. C'est vrai, mais cela ne suffit pas à lui assurer de meilleurs résultats. La figure 4.33 compare les résultats de SR quadratique avec un modèle WarpHR (à gauche) et WarpBR (à droite) pour deux valeurs du paramètre de régularisation, dans un cas sous-régularisé (en haut) ou légèrement sur-régularisé (en bas).

On constate que, dans le second cas (bien régularisé), les deux méthodes donnent des résultats très proches. Dans le premier, c'est-à-dire pour une régularisation faible permettant de faire apparaître les détails haute fréquence, l'image SR utilisant le modèle WarpHR ne présente pas le motif du SM, mais elle est aussi agitée par des oscillations liée aux limitations de la régularisation quadratique. Surtout, cette image ne permet pas de résoudre la mire 6, alors qu'elle est résolue dans l'image utilisant le WarpBR. Cet exemple est typique des résultats que nous avons obtenus sur un grand nombre d'essais avec des paramétrages variés. Sur cette séquence au moins, le WarpHR ne permet pas d'obtenir un meilleur compromis que WarpBR — tout en étant beaucoup plus coûteux d'emploi.

Revenons à la recherche d'une image aux limites de la résolution maximale possible en utilisant le modèle WarpBR. La Figure 4.34 montre que le gain en résolution (et l'importance

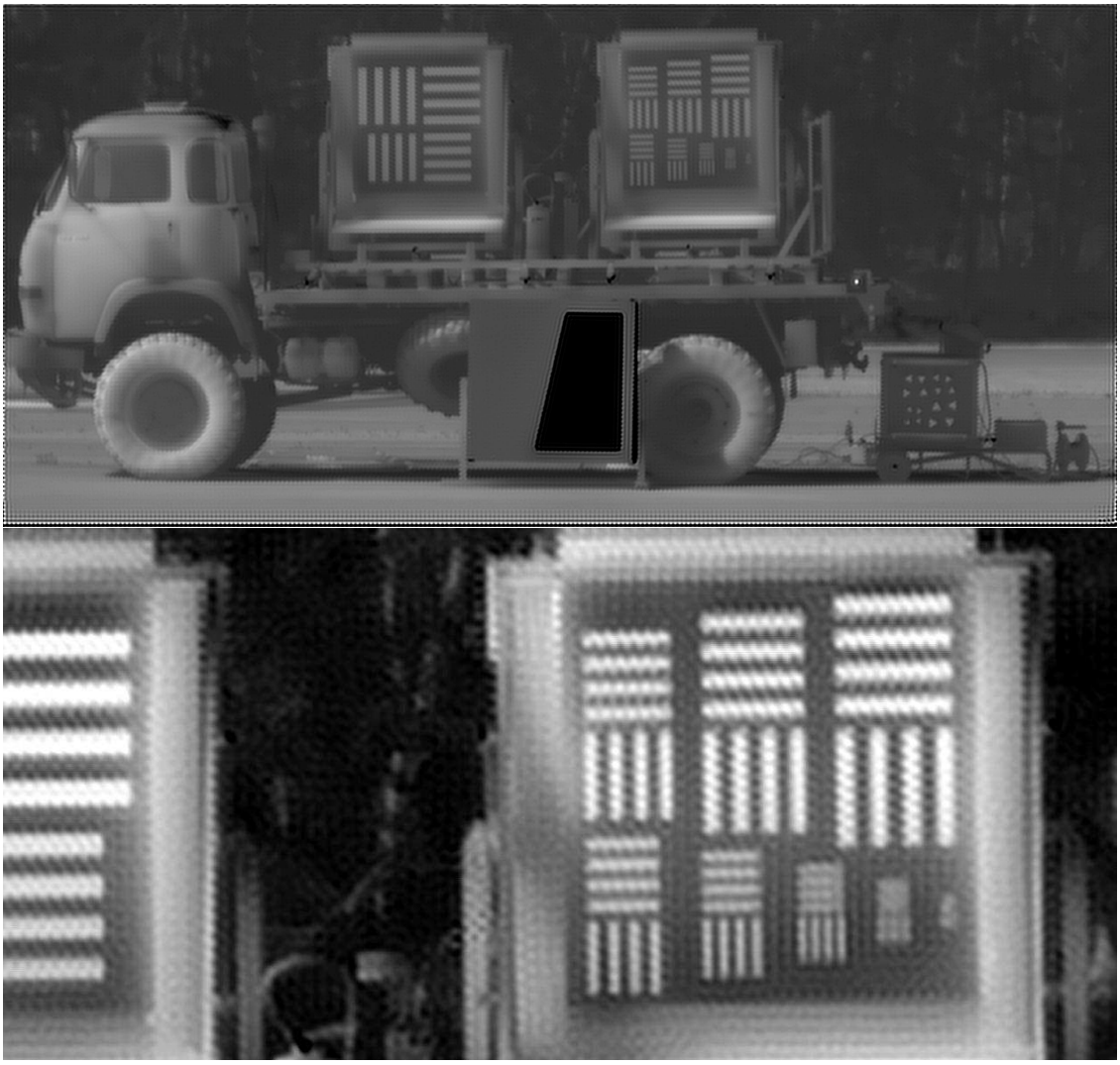


Figure 4.31 – SR quadratique pour une PSF plus étroite que l'estimée ($\sigma = 0.330$) et un facteur de SR très élevé (x15). L'image de dessous est un extrait de celle de dessus, pour voir le phénomène de damier. Les paramètres de traitement sont donnés en Table B.3.

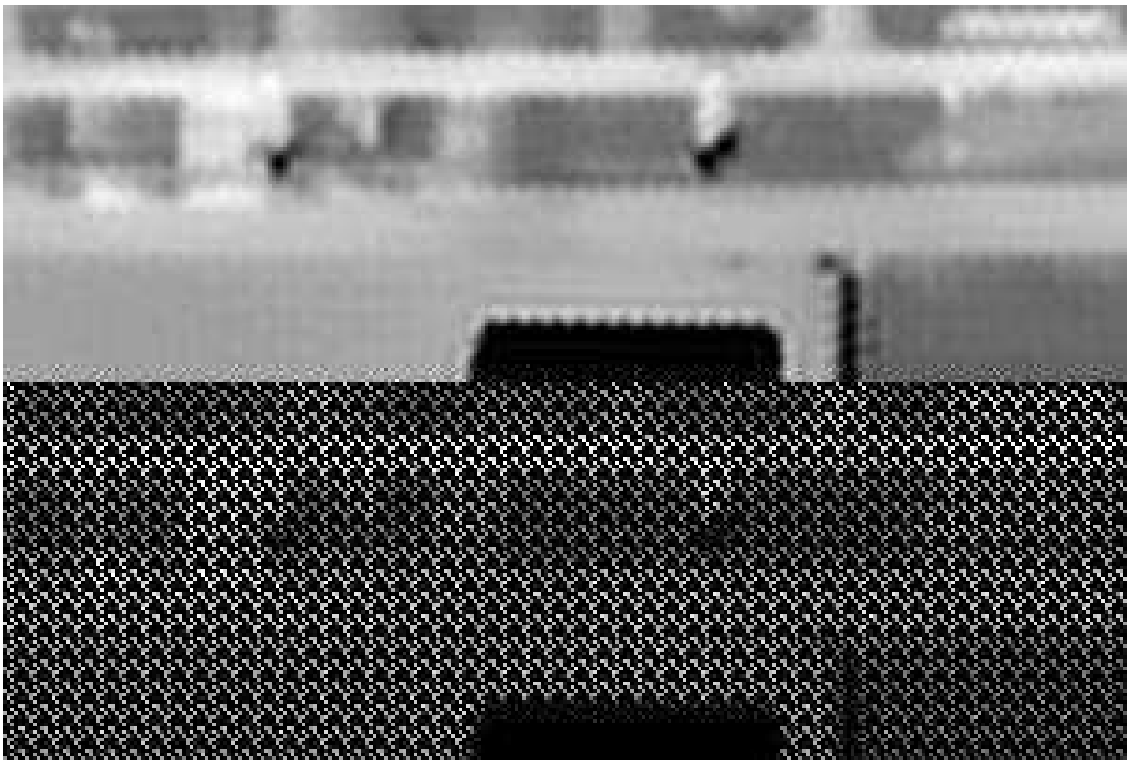


Figure 4.32 – Artefact en "trame" sur un détail d'une image SR et image SM avant bouchage de trous sur la même zone.

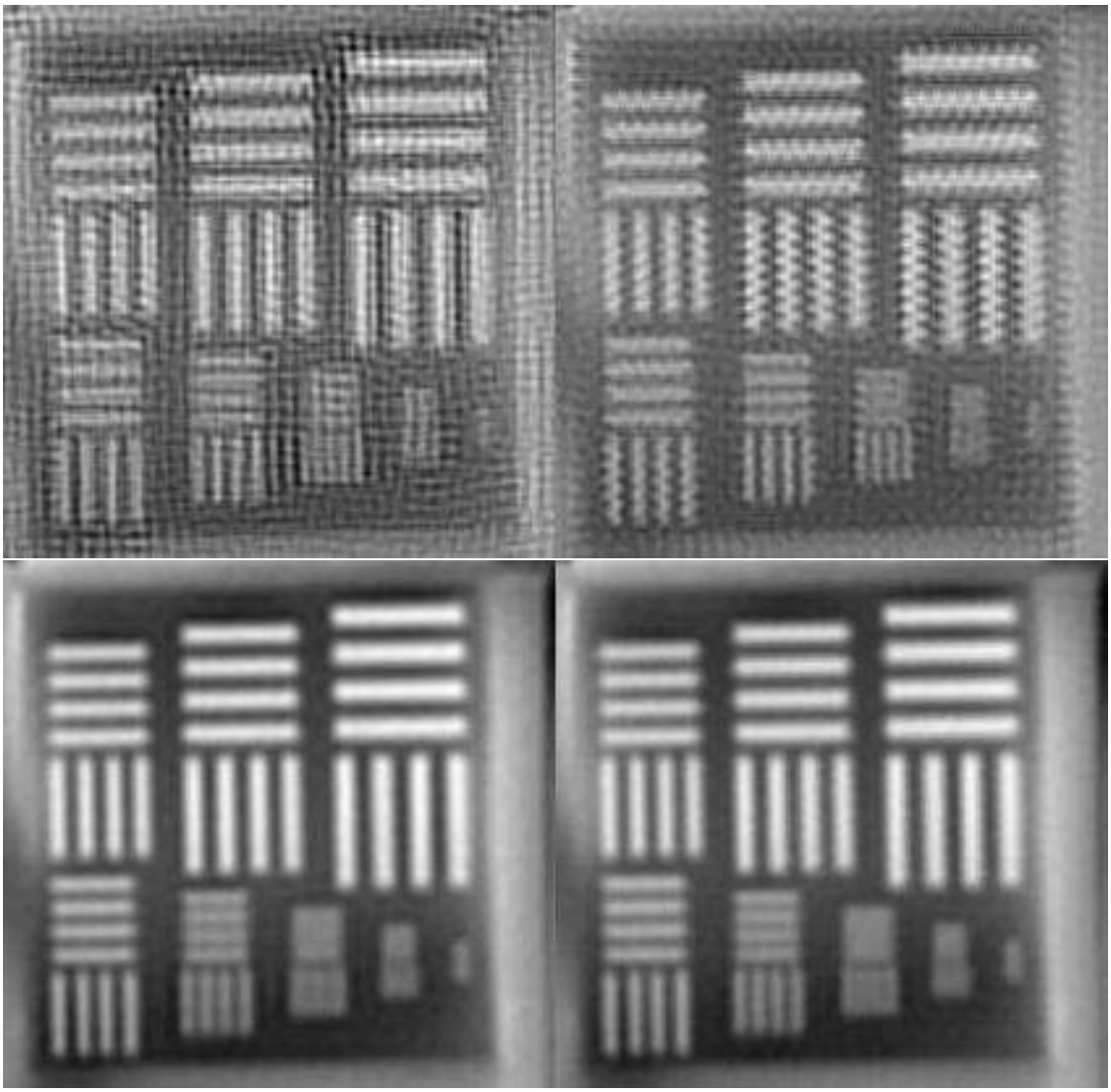


Figure 4.33 – Comparaison des modèles WarpHR (à gauche) et WarpBR (à droite) pour deux niveaux de régularisation.

des artefacts) croît avec le facteur de SR, les autres paramètres étant fixés. Sans aller jusqu'à un facteur de 15, on voit que la reconstruction à facteur 5 est légèrement meilleure que celle à facteur 5 et nettement meilleure que celle à facteur 3.

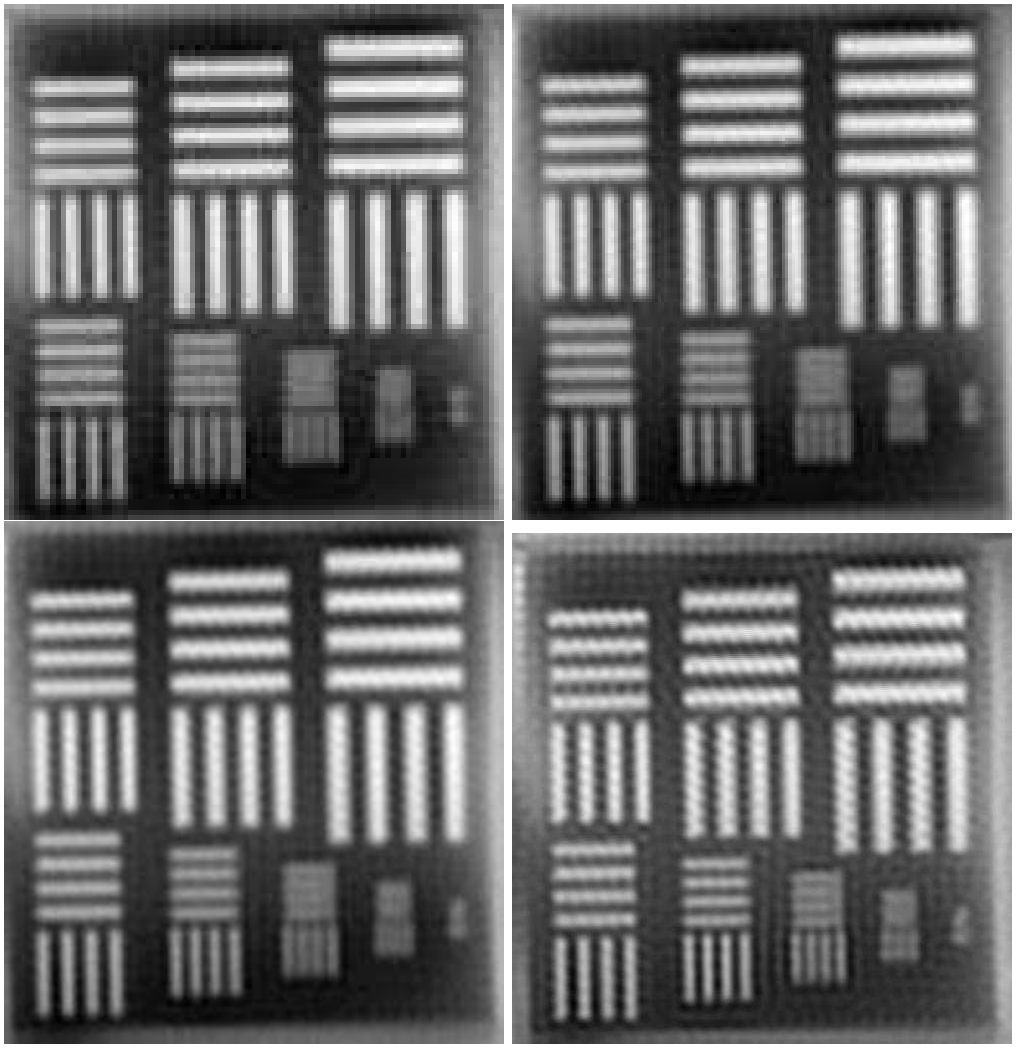


Figure 4.34 – SR quadratique pour une PSF plus étroite que l'estimée et un facteur de SR croissant : x3, x5, x7 et x15. Les paramètres de traitement sont donnés en Table B.3.

Les phénomènes de damier peuvent être en partie corrigés par l'utilisation de pénalisations non quadratiques, mais c'est au détriment de la résolution, comme on peut le voir en Figure 4.35 qui présente des images SR à facteur 10. Ces images sont obtenues avec une pénalisation de Huber à la fois sur les données et sur la régularisation : on joue ensuite sur le paramètre de régularisation. La reconstruction la plus régularisée (à droite) permet une excellente segmentation des triangles de la mire TOD, et de la mire à bâton 6 verticale, mais la mire horizontale est lissée,

alors que sa période est visible sur la reconstruction moins régularisée à gauche.

L'image du champ complet de la caméra pour la régularisation la plus élevée est présentée en Figure 4.36. Cette reconstruction exceptionnelle en termes de restitution des hautes fréquences de la scène nécessite un réglage délicat des nombreux paramètres de la méthode, ce qui serait très lourd en pratique.

4.5.2 Bruit élevé

La séquence "Tank" fait partie des séquences de tests réalisées par la DGA pour le groupe OTAN SET140, voir la référence [Weißet al., 2012]. Le dispositif d'acquisition est placé en haut d'une tour, visant les pistes en contrebas sur lesquelles se trouve positionnées plusieurs cibles, dont un tank au premier plan. Les images utilisées ici sont obtenues avec un temps d'intégration faible, d'où un niveau de bruit très élevé, avec en particulier un bruit spatial fixe ("effet pyjama") important, voir la Figure 4.37. Les gains sont corrigés avant recalage en normalisant chaque colonne indépendamment afin d'obtenir une séquence égalisée dont une image est présentée à droite de la même figure. C'est sur cette séquence que la SR est appliquée.

Sur ces images très bruitées, les différences entre méthodes sont faibles comme le montre la Figure 4.38 qui présente un zoom sur la zone du tank pour l'image BR remise à l'échelle, et les images SR par SMSK, SR quadratique et SR non quadratique (pénalisation Huber).

Dans ce cadre, qui revient essentiellement à du zoom avec débruitage, le principal facteur de la qualité image est le nombre d'images BR qui sont utilisées. La figure 4.39 présente des résultats grand champ sur la même séquence, obtenus par traitement de 11 (milieu) et 51 images BR (en bas, pénalisation Huber) comparés à l'image BR d'origine (en haut). Le gain associé au traitement d'un grand nombre d'images est très important.

4.5.3 Artefacts de compression

En pratique, il est courant d'avoir à traiter des séquences affectées d'artefacts de compression importants. C'est le cas par exemple lorsque l'on exploite a posteriori des vidéos transmises et stockées : les limites de la bande passante du système de transmission ou de stockage ont pu conduire à utiliser des facteurs de compression élevés. La recherche d'informations précises sur ces vidéos pose alors un problème de qualité image. Nous montrons dans cette section que la SR peut permettre de réduire très significativement ces artefacts, même s'ils n'ont pas été explicitement pris en compte dans le modèle de formation d'image de la section 4.2.

Un premier exemple est présenté en Figure 4.40 sur la séquence Aerien2 (cf. section B.2) : le zoom sur une image BR de la séquence montre les artefacts de compression. L'image SR

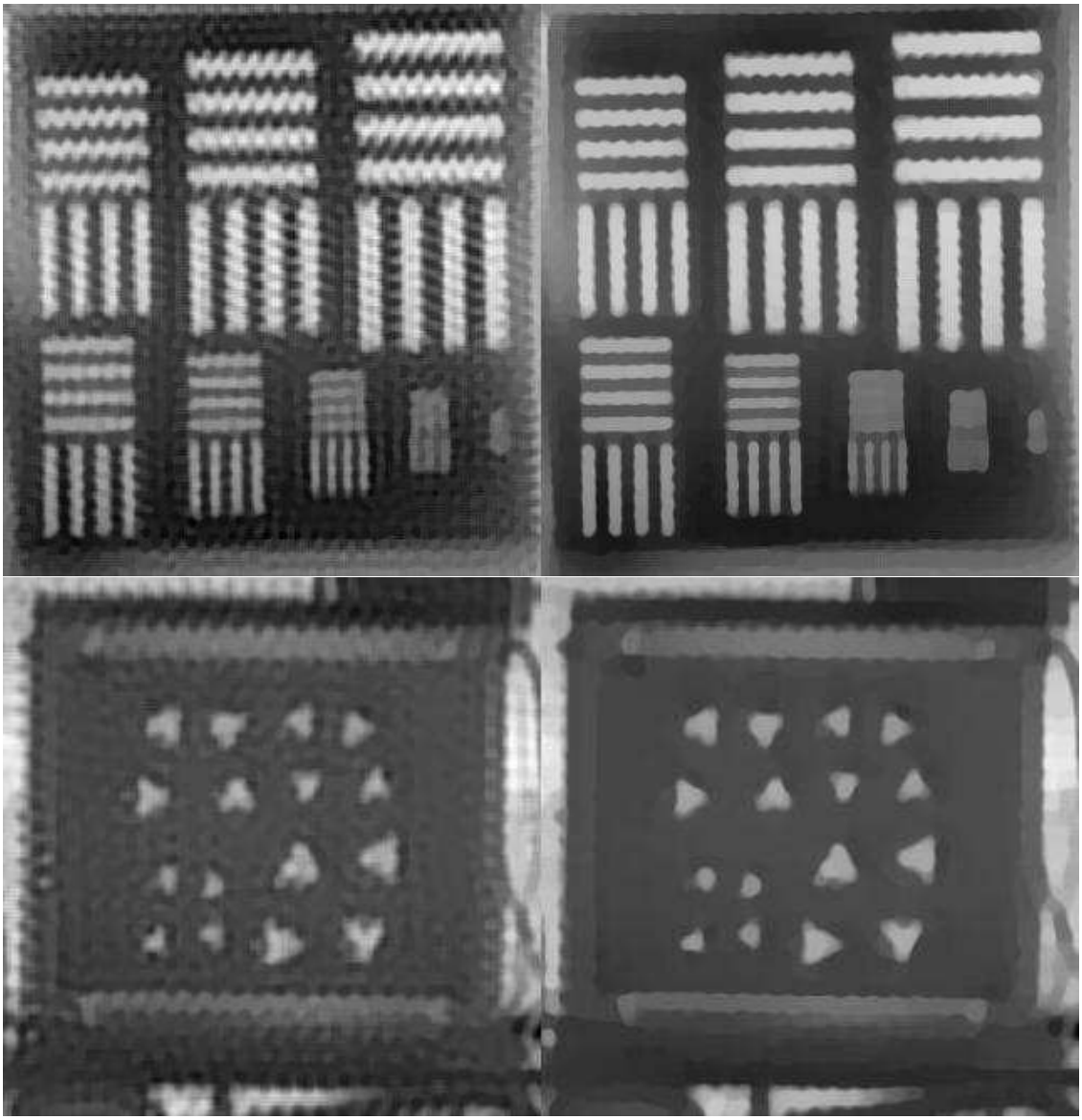


Figure 4.35 – SR non-quadratique avec pénalisation de Huber pour une PSF plus étroite que l'estimée ($\sigma = 0.330$) et deux valeurs du paramètre de régularisation. Les paramètres de traitement sont donnés en Table B.3.

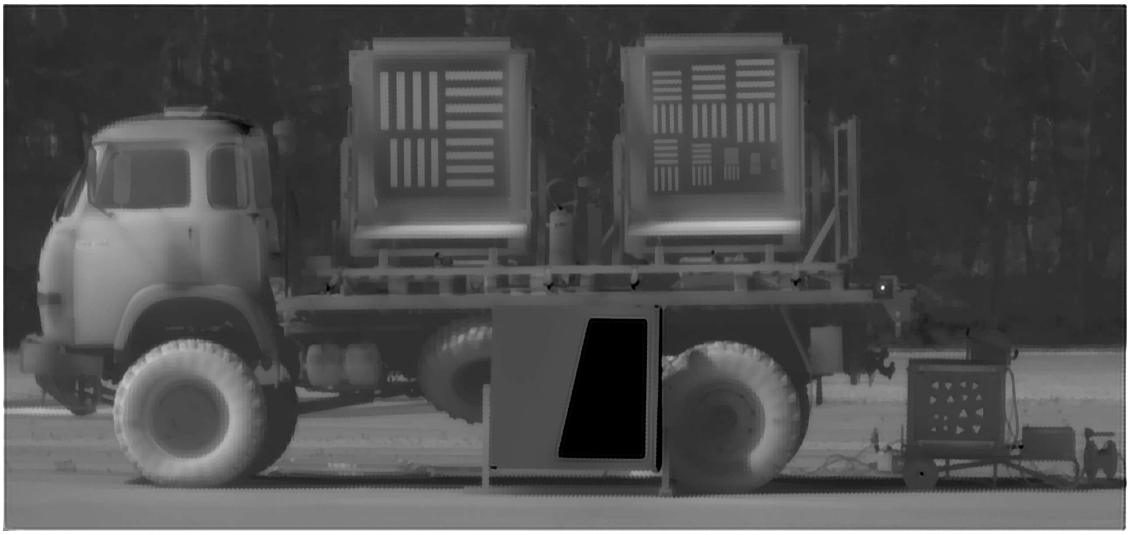


Figure 4.36 – SR non-quadratique avec pénalisation de Huber pour une PSF plus étroite que l'estimée ($\sigma = 0.330$) et un paramètre de régularisation fort. Les paramètres de traitement sont donnés en Table B.3.

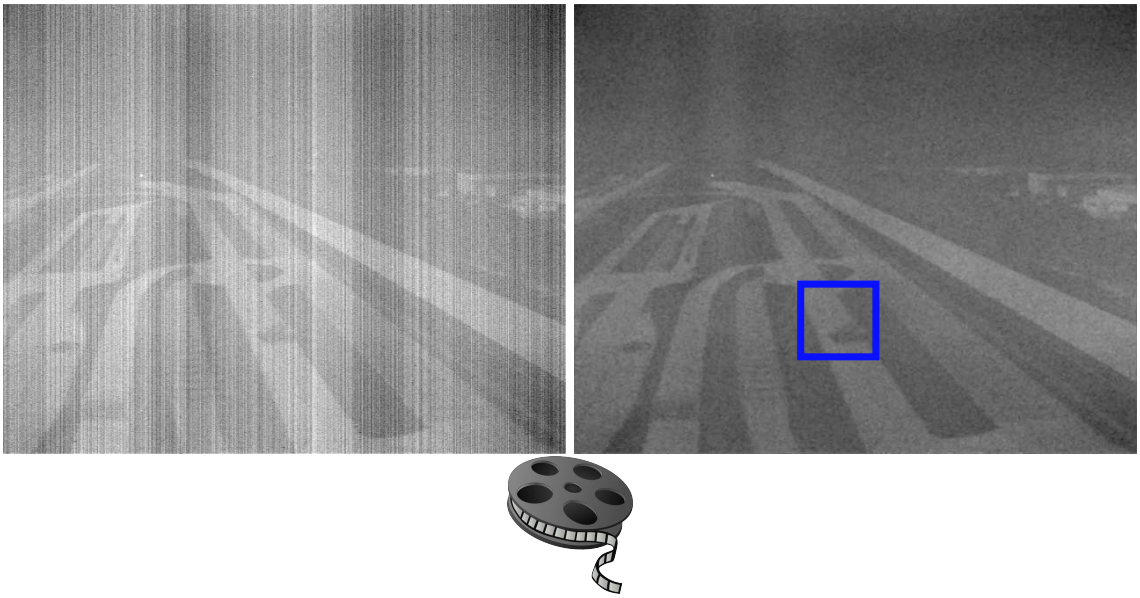


Figure 4.37 – Une image de la séquence Tank avant correction des gains (à gauche) puis après correction à droite. Sur cette deuxième image est indiquée la localisation de la zone observée sur les comparaisons de résultats SR ci-dessous.

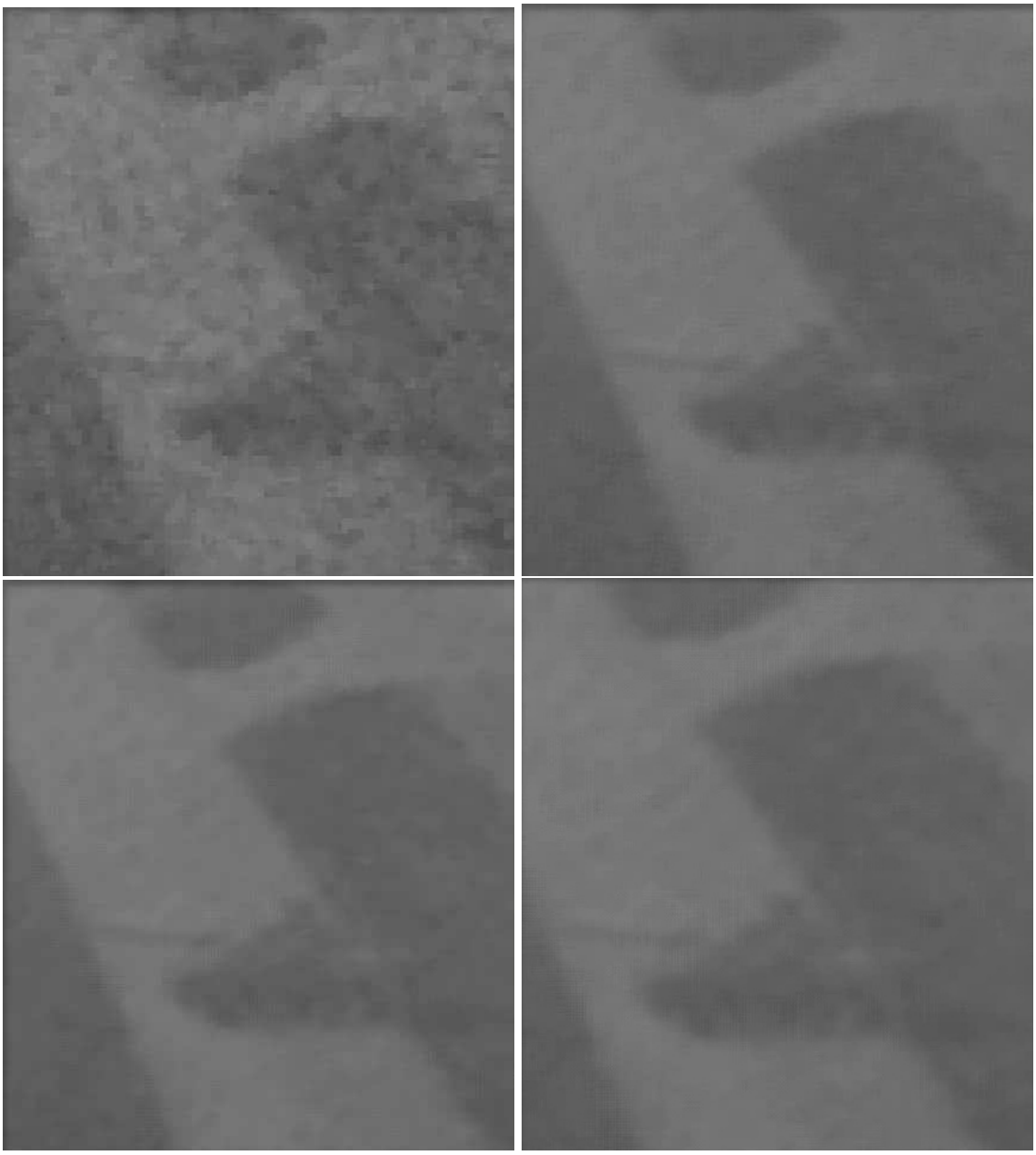


Figure 4.38 – de gauche à droite et haut en bas : image BR mise à l'échelle, image SR par SMSK, par méthode itérative avec une régularisation quadratique et Huber. Les paramètres de traitement sont donnés en Table B.3.

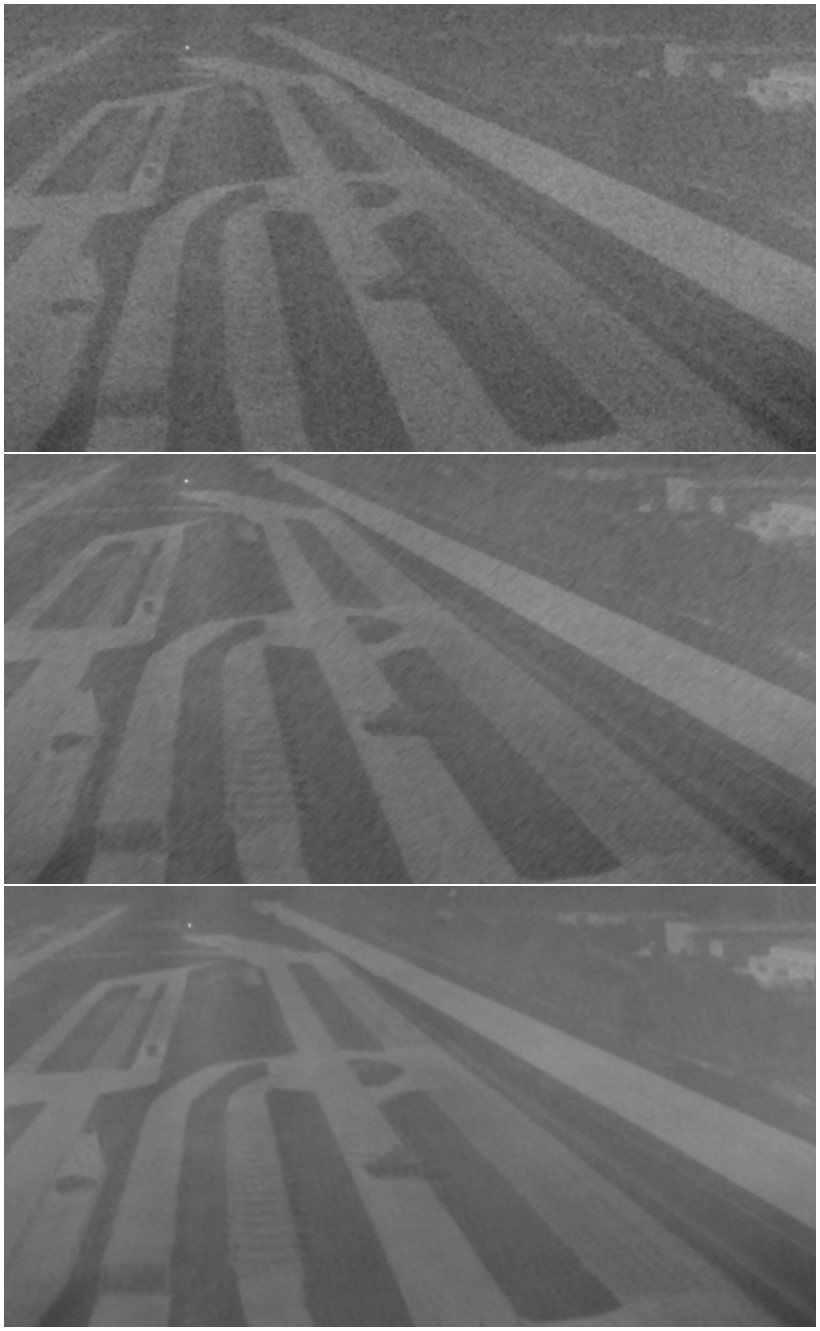


Figure 4.39 – En haut, image BR mise à l'échelle, résultat de SR non quadratique (pénalisation Huber) par traitement de 11 images BR (milieu) et 51 images BR (bas). Les paramètres de traitement sont donnés en Table B.3.

utilisant 17 images de la séquence d'origine est de bien meilleure qualité visuelle.

Figure 4.40 – SR en facteur 6 avec 17 images de la séquence Aerien2 affectée d'un fort bruit de compression. A gauche, image BR d'origine zoomée ; à droite, résultat SR. Les paramètres de traitement sont donnés en Table B.3.

On peut voir une zone zoomée en Figure 4.41 et comparer l'effet des différents traitements proposés. Le Shift-and-Mean améliore nettement l'image, mais reste un peu flou, le SM-SK est moins flou, mais présente des instabilités locales. La régularisation non quadratique permet d'obtenir un compromis entre le lissage des artefacts de compression et la déconvolution plus intéressant que les méthodes linéaires.

D'une manière générale, dans ces exemples très dégradés, la pénalisation non quadratique permet d'améliorer significativement la qualité des images. Si elle ne permet pas réellement de faire apparaître de nouveaux détails haute fréquence, elle peut faciliter par exemple l'identification de véhicules, comme le démontre la comparaison d'un arrêt sur image zoomé et d'une reconstruction SR en figure 4.42.

Cependant, comme nous l'avons déjà noté, l'emploi des régularisations non quadratiques avec de mauvais réglages peut donner des résultats dégradés par un fort effet cartoon, comme cela est montré dans la vignette du bas de la même figure 4.42. De bons compromis peuvent être trouvés, au prix d'un réglage plus complexe, en utilisant une pénalisation non quadratique à la fois sur le terme de régularisation et sur le terme de données, comme le montre la figure 4.43. Ces méthodes offrent de plus des perspectives très intéressantes pour la gestion des champs de mouvements complexes, comme nous le verrons en section suivante.

Figure 4.41 – SR en facteur 6 avec 17 images de la séquence Aerien2 affectée d'un fort bruit de compression. HBGD : image BR d'origine zoomée ; en bas, résultat SR. Les paramètres de traitement sont donnés en Table B.3.

Figure 4.42 – Detail d'images SR sur la séquence Aerien1 : en haut, image BR mise à l'echelle, au milieu pénalisation non-quadratique (Huber), en bas pénalisation non quadratique avec fort effet cartoon (Geman). Les paramètres de traitement sont donnés en Table B.3.

Figure 4.43 – Comparaison image BR et image SR avec régularisation non quadratique sur la régularisation et les données pour la séquence Aerien1. Les paramètres de traitement sont donnés en Table B.3.

4.5.4 Mouvements complexes et influence de l'estimateur de mouvement

Comparaison des modèles de recalage dans un cas simple

La précision de l'estimation du mouvement influence bien entendu la qualité de la reconstruction SR. La Figure 4.44 compare, sur la séquence Camion, les images obtenues par SR rapide SMSK pour les mêmes réglages en utilisant différents modèles de recalage. On utilise soit un flot optique avec une fenêtre de taille 17×17, ou 33×33, soit un recalage sous hypothèse de translation globale.

Comme on pouvait s'y attendre, sur cette séquence qui est effectivement animée d'un mouvement global, il est préférable d'estimer le mouvement en intégrant spatialement le plus possible. On peut cependant noter que la différence de qualité est limitée et, en fait, de l'ordre de ce qu'on avait observé comme différences entre la SR rapide SMSK et la SR quadratique itérative en Figure 4.28. On peut donc se dire que ces différences pourraient être gommées par un réglage approprié de la régularisation, au prix d'une perte de résolution — d'ailleurs, pour approcher les limites de résolution de la SR en Section 4.5.1, nous avons utilisé un modèle de recalage paramétrique. Par ailleurs, lorsque le champ de mouvement n'est pas connu ou bien n'est pas paramétrique, il sera indispensable d'utiliser un flot optique, comme dans les exemples que nous verrons dans la suite de cette section. Enfin, à qualité équivalente, le recalage par flot

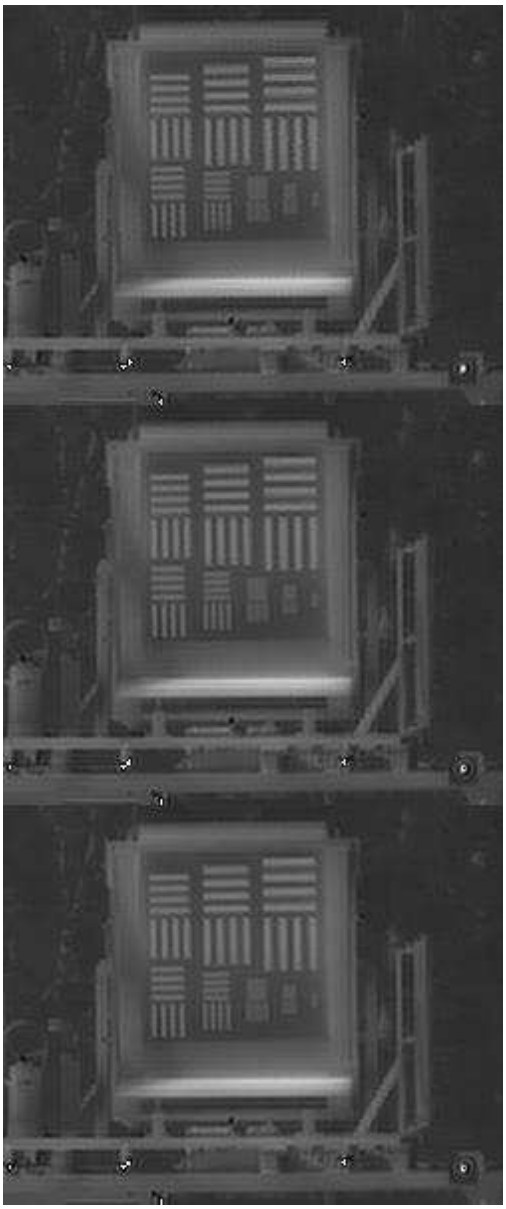


Figure 4.44 – SR et modèle de recalage. De haut en bas : recalage par flot optique FOLKI, fenêtre 17×17 et 33×33, recalage paramétrique sous hypothèse de translation globale. Les paramètres de traitement sont donnés en Table B.3.

optique est moins coûteux que le recalage paramétrique dense, comme montré au chapitre 3.

Mappemonde

La séquence Mappemonde est un cas d'école pour la SR sur flot optique, puisque le mouvement apparent sur la sphère en rotation ne rentre pas dans les modèles paramétriques classiques (affine, homographique).

La figure 4.45 présente un résultat : le mouvement complexe de la sphère en rotation est bien estimé par FOLKI et le résultat SR (obtenu ici avec un critère non quadratique sur la régularisation et les données) est très bon. Le texte sur la mappemonde devient lisible, y compris sous incidence rasante comme montré dans les vignettes du bas.

Oscillations locales

Cette section présente deux contextes de mouvements locaux indépendant de la volonté de l'utilisateur qui peuvent, dans certaines conditions, être exploités pour faire de la SR si on utilise un recalage par flot optique.

Dans la séquence Aérien1, l'origine des mouvements image est le déplacement de la ligne de visée par l'opérateur. Quand on regarde la vidéo, on se rend compte que ce déplacement génère en fait un mouvement apparent de fluctuation des lignes de l'image qui est dû au mode d'acquisition progressive ("rolling shutter") du capteur. Avec cette technologie, les lignes ne sont pas toutes acquises au même instant, ce décalage temporel induit un champ de mouvement non translationnel dans l'image même dans le cas de mouvements simples de la ligne de visée. La figure 4.46 illustre l'échec des stratégies de recalage global dans ce cas.

L'ensemble des résultats présentés sur cette séquence (Figures4.41 et 4.43) a donc utilisé un recalage local. Il s'agit d'une situation assez courante (les capteurs "rolling shutter" sont très répandus) et où la SR avec recalage par une technique de flot optique comme FOLKI trouve tout son intérêt. En effet le champ de mouvement étant régulier, on peut travailler avec une fenêtre de corrélation d'assez grande taille et obtenir ainsi des résultats précis.

Un autre effet qui dégrade la qualité des vidéos est la turbulence atmosphérique. Dans le contexte de la super-résolution, cette turbulence entraîne des mouvements images qui peuvent permettre de super-résoudre la vidéo, et ce même lorsque capteur est fixe (cette approche a été proposée dans [Yaroslavsky et al., 2007]). Pour expérimenter et valider cette situation, nous avons réalisé une expérimentation simple en laboratoire grâce à la turbulence produite par la chaleur d'un grille-pain, cf. Figure 4.47. Nous avons ainsi pu obtenir un facteur de super-résolution suffisant permettant la lecture d'un texte situé derrière la zone de turbulence, celui-ci

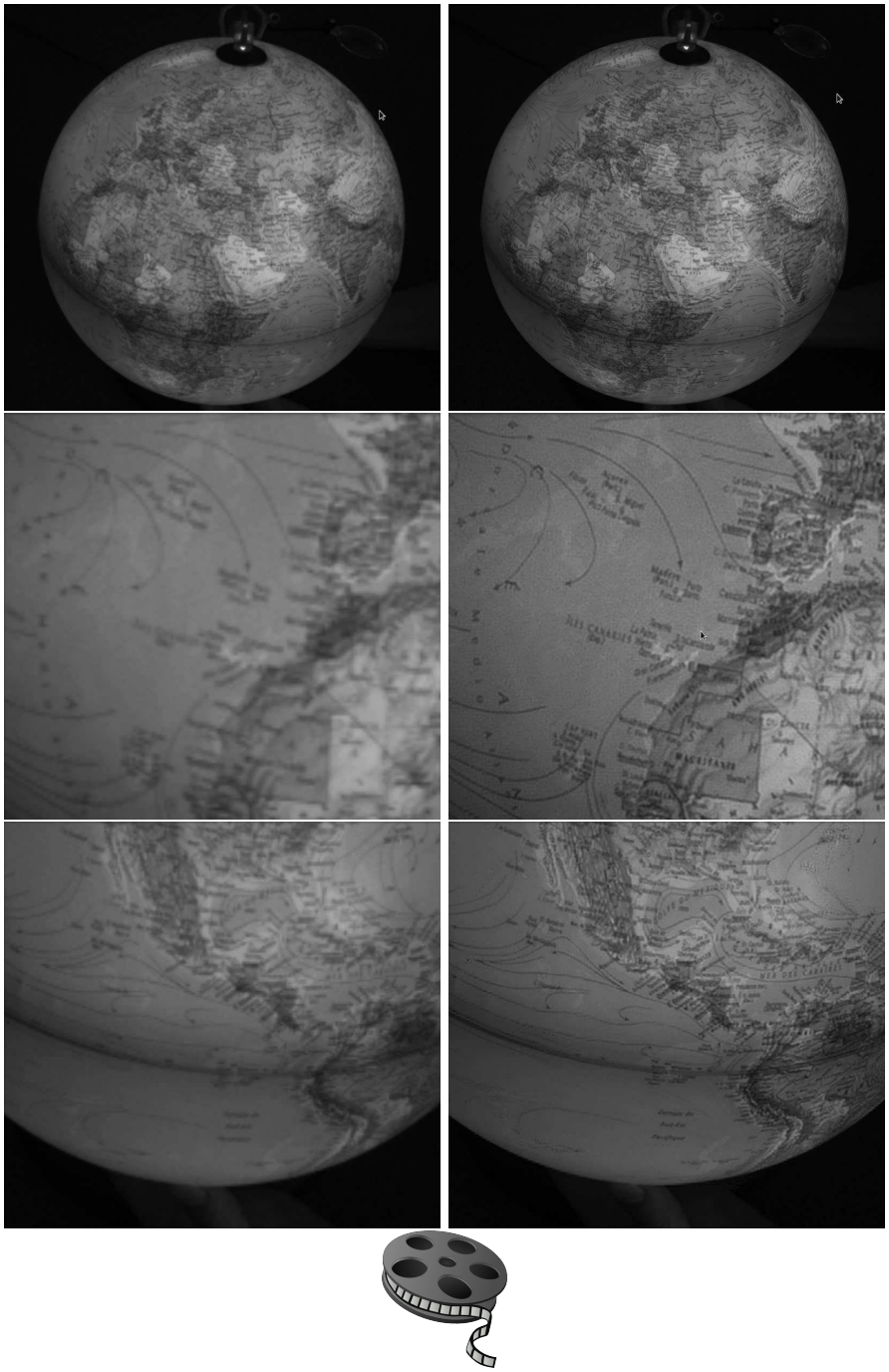


Figure 4.45 – SR sur la séquence Mappemonde. Les paramètres de traitement sont donnés en Table B.3.

Figure 4.46 – Comparaison de techniques de recalage pour la SR sur la séquence Aerien1. A gauche, recalage sous hypothèse de mouvement global de translation 2D ; à droite, recalage par flot optique FOLKI. Les paramètres de traitement sont donnés en Table B.3.

étant illisible avant le processus de super-résolution, voir Figure 4.47.

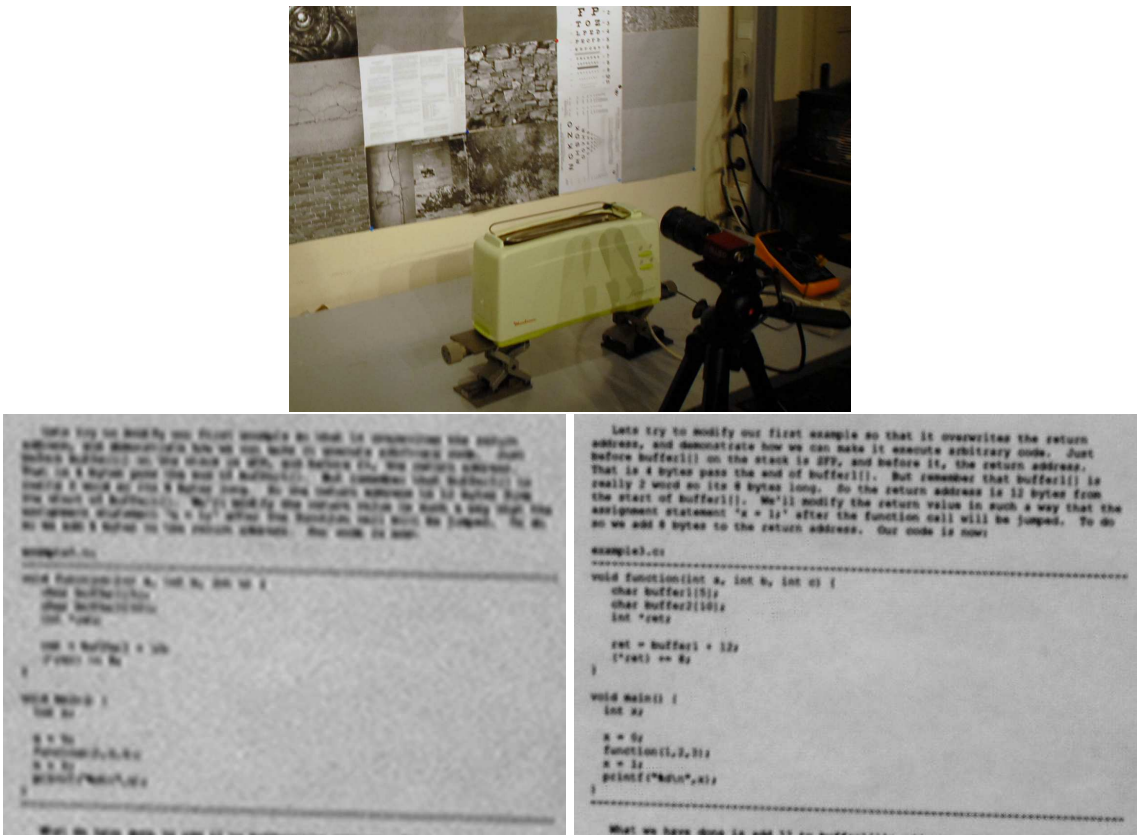


Figure 4.47 – Super-résolution exploitant le mouvement induit par la turbulence afin d'améliorer la lecture d'un texte accroché. En haut, le dispositif expérimental (la turbulence est produite par un grille-pain), en bas à gauche, une image basse résolution extraite de la séquence, à droite une image super-résolue.

Petits objets en mouvement

La Figure 4.48 présente un cas intéressant de SR sur une vidéo aérienne avec des véhicules de petite taille en mouvement. Ce contexte difficile a été étudié par exemple dans la référence [van Eekeren et al., 2008], mais avec des techniques dédiées (segmentation objet/fond, modification de la technique de SR pour l'objet lui-même) assez coûteuses : nous montrons ici que l'emploi de l'algorithme de SR avec repondération des données permet de restaurer correctement les objets en mouvement en effectuant toujours la SR du fond.

Sur la ligne du bas de la Figure 4.48, l'image de gauche présente le résultat sans pénalisation non quadratique sur le terme de données. Le mouvement du véhicule qui est de taille inférieure

à la fenêtre de FOLKI est mal estimé, ce qui conduit à des artefacts classiques (l'objet semble "partir en poudre"). A droite, l'emploi de la pénalisation non quadratique permet d'éliminer les données faussement associées à cause des mouvements contradictoires du fond et du véhicule et de récupérer une image correcte. Même si l'image obtenue de l'objet mobile ne profite pas complètement de l'effet SR parce qu'elle ne collecte pas autant de données BR, elle n'est en tous cas pas dégradée par rapport à l'image BR (visible au dessus).

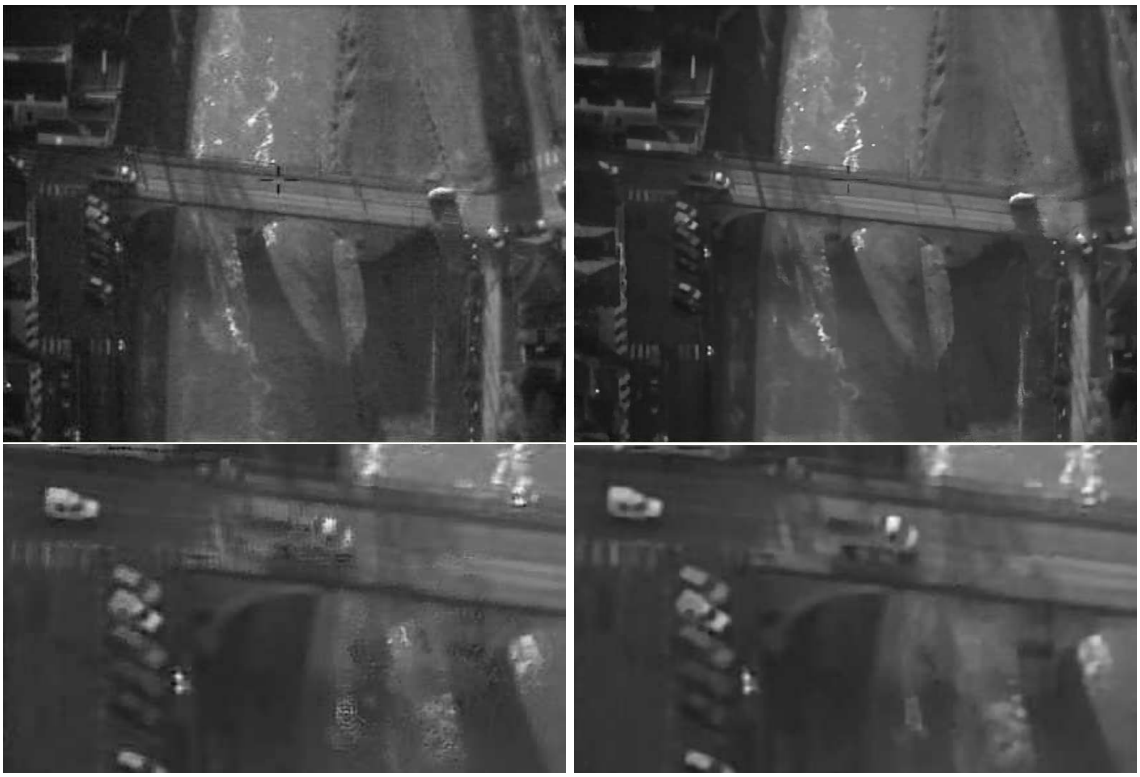


Figure 4.48 – SR et petits objets mobiles. En haut à gauche, une image BR de la séquence d'origine, à droite, résultat SR. En bas à gauche, SR sans pénalisation non quadratique des données, à droite avec. Les paramètres de traitement sont donnés en Table B.3.

Piétons

Dans cette section, nous abordons réellement les limites de la SR, en essayant de l'utiliser sur une vidéo IR de piétons marchant et se croisant dans la rue — il s'agit d'une des vidéos du jeu de données URBAN fait par la DGA pour le SET 140 OTAN, cf. [Weißet al., 2012]. Le champ de mouvement est ici très complexe : il y a des rotations des personnages, des déformations, des occultations lors des croisements. Les essais que nous avons menés montrent cependant

qu'il existe un cadre d'application pour les algorithmes présentés dans ce travail, en réduisant le nombre d'images dans le GoP et en exploitant un terme de pénalisation non quadratique sur les données. De plus, nous avons testé pour l'occasion la SR non seulement sur FOLKI mais aussi sur un flot optique régularisé, obtenu par un algorithme de type MCLG (cf. section 3.5.1) avec une régularisation robuste par *variable-splitting*, et un groupe de cinq images pour la super-résolution . Les figures 4.49 et 4.50 présentent deux snapshots de la vidéo BR d'origine et des videos SR obtenues avec ces deux types de flot.

La figure 4.49 montre d'une part que les traitements de SR apportent effectivement un gain en qualité d'image par rapport à la séquence d'origine. Le gain est encore plus net avec l'utilisation d'un flot optique régularisé (en bas). La figure 4.50 considère un cas difficile de croisement des personnages (au milieu et à droite). Des artefacts apparaissent dans les deux résultats SR. Le résultat utilisant un flot régularisé est un peu meilleur, car les effets de "poudre" sont bien localisés aux zones d'occultation et plus faciles à détecter. Dans le cas de FOLKI, on peut avoir des détails qui peuvent sembler réels, mais qui sont des artefacts (zone blanche à droite de la tête du personnage du milieu par exemple).

4.5.5 Conclusions et perspectives

Nous avons présenté dans ce chapitre nos travaux et résultats en super-résolution. Leur principale originalité est d'utiliser des estimateurs de mouvements non paramétrique (flot optique) permettant ainsi de pouvoir traiter des vidéos avec des mouvements complexes. Par rapport aux travaux existants, nous avons privilégié l'obtention d'algorithmes réellement rapides en visant des cadences comprises entre quelques Hz et la cadence vidéo (25 Hz).

Nous avons envisagé différentes solutions algorithmiques, selon un arbre présenté en Figure 4.51. Le premier choix important concerne le modèle de formation d'image adopté. Nous avons comparé deux modèles :

- le modèle WarpHR, modèle de référence largement utilisé dans la littérature [Mitzel et al., 2009]
- le modèle WarpBR, simplificateur, qui a fait l'objet travaux précédents dans l'équipe [Rochefort, 2005, Létienne, 2010]

Nous avons montré que le modèle WarpBR réduit considérablement la quantité de calcul nécessaire, et ce sans perte trop importante sur la qualité du résultat.

A partir de ce choix du modèle WarpBR, nous avons proposé deux stratégies de résolution pour le problème de la super-résolution. Premièrement, nous avons développé et validé une chaîne de traitement très rapide SM+SK, fondée sur la fusion des images BR par Shift and Mean puis la déconvolution de l'image obtenue par une implémentation rapide et approximative

Figure 4.49 – Séquence Piétons : en haut image BR zoomée, ; au milieu, image SR sur flot optique FOLKI (avec repondération des données) ; en bas, image SR sur flot régularisé (avec repondération). Les paramètres de traitement sont donnés en Table B.3.

Figure 4.50 – Séquence Piétons : en haut image BR zoomée, ; au milieu, image SR sur flot optique FOLKI (avec repondération des données) ; en bas, image SR sur flot régularisé (avec repondération). Les paramètres de traitement sont donnés en Table B.3.

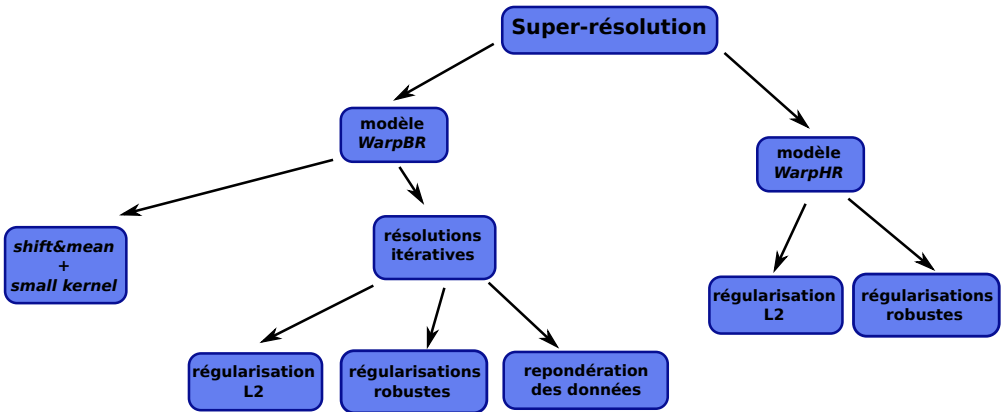


Figure 4.51 – Synthèse des algorithmes de super-résolution proposés dans ce chapitre.

du filtrage de Wienner : les filtres Small Kernel. Cette première solution profite pleinement d'une structure de calcul homogène, ce qui permet d'atteindre de très grandes vitesses de traitement sur GPU. On atteint par exemple la cadence vidéo pour une SR $\times 6$ sur un GoP de 21 images 256×120.

En parallèle, nous avons étudié des stratégies de résolutions itératives, permettant des modifications du schéma de résolution en utilisant des fonctions de repondérations robustes, tout d'abord sur le terme de régularisation (SR-ϕ), puis sur le terme d'attache aux données (SR-$\phi\psi$). Cette seconde stratégie, bien que plus coûteuse que les méthodes SM+SK, nous permet d'atteindre une grande qualité dans la reconstruction SR. La méthode SR-$\phi\psi$ permet de plus de corriger certaines erreurs locales de l'estimation du mouvement. Cela se révèle très utile en pratique sur les séquences réelles qui comprennent souvent des mouvements complexes : véhicules en mouvement ou fumées sur les vidéo aériennes ; piétons dans un cadre plus proche de la vidéo-surveillance, etc. Nous obtenons ainsi une méthode plus chère, mais restant compatible avec le traitement en ligne (typiquement 10Hz sur l'exemple précédent), avec un domaine d'emploi très large : ce dernier point constitue, à notre avis, la principale innovation de notre travail.

Enfin, pour démontrer les résultats de la SR sur ces nombreuses séquences réelles, nous avons développé SRviewer une interface de traitement et de paramétrage en ligne, fondée sur un noyau implémenté sur GPU, cf. Annexe 3.5.1.

Perspectives : Une extension à court terme est le traitement de vidéo couleur. Dans un premier temps, on peut voir cela comme un problème de mise en œuvre, car les algorithmes sont essentiellement inchangés. Pour affiner, il faut modifier la régularisation pour tenir compte de corrélations entre les canaux de couleur.

La principale limitation de nos travaux concerne le choix des paramètres de traitement. Nous avons privilégié dans notre interface la possibilité de paramétrer complètement les méthodes proposées, en affichant en temps réel les effets sur la séquence SR pour aider l'utilisateur à choisir les paramètres. Même s'il nous semble qu'une part du réglage devra toujours être effectuée par l'opérateur, il est clair qu'un outil réaliste et opérationnel devra intégrer des choix automatiques ou des "factorisations" de certains paramètres pour éviter que l'usage ne soit trop complexe. C'est donc une voie de travail importante pour compléter nos travaux.

Même si nous avons étudié des cas dégénéré, comme des séquences très bruitées ou affectées d'une forte compression, en ce qui concerne l'effet de la turbulence, nous avons considéré un cas de dégradation légère. On pourrait se poser le problème dans le cas de forts effets de turbulence. Les premiers essais que nous avons mené indiquent que dans de telles situations l'hypothèse de constance de l'intensité devient invalide dans beaucoup de zones de l'image. On observe ainsi des phénomènes de dédoublement de structures (mirages), ainsi que d'importantes variations locales (en spatial et en temporel) de la PSF — ces variations apparaissent aussi dans les vidéos affectées d'un flou de bougé. Tous ces facteurs rendent la SR (mais aussi et d'abord d'estimation de mouvement) difficilement applicable. Pour traiter ces cas, il nous semble qu'il faudrait combiner nos approches de SR avec des méthodes dites de *Lucky-imaging* [Fried, 1978]. Ces méthodes sélectionnent les images BR "exploitables" selon un critère de qualité image à définir, mais qui pourrait s'inspirer des techniques d'identification locales du flou développées dans l'équipe par Pauline Trouvé [Trouvé et al., 2011]. Dans leur principe, ces méthodes pourraient être rapprochées de la repondération des données effectuée dans l'algorithme SR-$\phi\psi$.

Pour finir, et pour introduire le sujet du chapitre suivant, on peut penser qu'utiliser la super-résolution uniquement dans la géométrie image est une limitation, principalement dans le cas de scènes dynamiques complexes. Une extension intéressante est donc d'intégrer la super-résolution dans un processus de reconstruction 3D et de rendu de texture (en utilisant un banc stéréo ou un autre capteur 3D). Il existe des travaux sur ce thème [Graber et al., 2011], mais il pourrait être intéressant de développer des techniques "rapides" en s'appuyant sur des choix d'algorithmie et d'architecture pertinents, comme nous l'avons fait ici pour la SR 2D.

Bibliographie

[Buades et al., 2005] Buades, A., Coll, B., and Morel, J.-M. (2005). A non-local algorithm for image denoising. *Computer Vision and Pattern Recognition, IEEE Computer Society Conference on*, 2 :60–65.

[Champagnat, 2011] Champagnat, F. (2011). Small kernel deconvolution using 10 lines of (matlab) code (internal report onera).

[Champagnat et al., 2009] Champagnat, F., Le Besnerais, G., and Kulcsár, C. (2009). Statistical performance modeling for superresolution : a discrete data-continuous reconstruction framework. 26(7) :1730–1746.

[Defretin, 2011] Defretin, J. (2011). *Stratégies de Vision Active pour la Reconnaissance d'Objets et d'Événements sur des Séquences Vidéo Aériennes*. PhD thesis, Ecole Normale Supérieur de Cachan, Onera, France.

[Elad and Hel-Or, 2001] Elad, M. and Hel-Or, Y. (2001). A fast super-resolution reconstruction algorithm for pure translational motion and common space-invariant blur. *IEEE Transactions on Image Processing*, 10(8) :1187–1193.

[Fried, 1978] Fried, D. L. (1978). Probability of getting a lucky short-exposure image through turbulence. *J. Opt. Soc. Am.*, 68(12) :1651–1657.

[Glasner et al., 2009] Glasner, D., Bagon, S., and Irani, M. (2009). Super-resolution from a single image. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 349–356. Ieee.

[Graber et al., 2011] Graber, G., Pock, T., and Bischof, H. (2011). Online 3d reconstruction using convex optimization. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 708–711. IEEE.

[Grasmair and Lenzen, 2010] Grasmair, M. and Lenzen, F. (2010). Anisotropic total variation filtering. *Applied Mathematics and Optimization*, 62 :323–339.

[Idier, 2001] Idier, J. (2001). *Approche bayésienne pour les problèmes inverses*. Hermès.

[Létienne, 2010] Létienne, A. (2010). *Super-Résolution : Développement d'algorithmes rapides et évaluation de performance*. PhD thesis, Université Paris 13, Villetaneuse, France.

[Mitzel et al., 2009] Mitzel, D., Pock, T., Schoenemann, T., and Cremers, D. (2009). Video super resolution using duality based tv-l1 optical flow. In *Pattern Recognition (Proc. DAGM)*, Jena, Germany.

[Pereira and Goussard, 1997] Pereira, S. and Goussard, Y. (1997). Unsupervised 3d restoraiton of tomographic images by constrained wiener filtering. In *Proceedings of 19th Intl Conf. EMBS*, Chicago, Il. IEEE.

[Protter et al., 2009] Protter, M., Elad, M., Takeda, H., and Milanfar, P. (2009). Generalizing the nonlocal-means to super-resolution reconstruction. *Image Processing, IEEE Transactions on*, 18(1) :36–51.

[Reichenbach and Park, 1991] Reichenbach, S. E. and Park, S. K. (1991). Small convolution kernels for high-fidelity image restoration. *IEEE Transactions on Signal Processing*, 39(10) :2263–2274.

[Rochefort, 2005] Rochefort, G. (2005). *Amélioration de la résolution de séquences d'images. Application aux capteurs aéroportés.* PhD thesis, Université Paris-Sud, Orsay, France.

[Rochefort et al., 2006] Rochefort, G., Champagnat, F., Le Besnerais, G., and Giovannelli, J.-F. (2006). An improved observation model for super-resolution under affine motion. 15(11) :3325–3337.

[Shechtman et al., 2005] Shechtman, E., Caspi, Y., and Irani, M. (2005). Space-time super-resolution. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27 :531–545.

[Takeda et al., 2009] Takeda, H., Milanfar, P., Protter, M., and Elad, M. (2009). Super-resolution without explicit subpixel motion estimation. *Image Processing, IEEE Transactions on*, 18(9) :1958–1975.

[Trouvé et al., 2011] Trouvé, P., Champagnat, F., Le Besnerais, G., and Idier, J. (2011). Single image local blur identification. In *Proceedings of IEEE Intl. Conf. Image Processing (ICIP'11)*, pages 613–616, Bruxelles (Belgique). IEEE.

[Unger et al., 2010] Unger, M., Pock, T., Werlberger, M., and Bischof, H. (2010). A convex approach for variational super-resolution. In Goesele, M., Roth, S., Kuijper, A., Schiele, B., and Schindler, K., editors, *Pattern Recognition*, volume 6376 of *Lecture Notes in Computer Science*, pages 313–322. Springer Berlin / Heidelberg.

[van Eekeren et al., 2007] van Eekeren, A. W. M., Schutte, K., Oudegeest, O. R., and van Vliet, L. J. (2007). Performance evaluation of super-resolution reconstruction methods on real-world data. *EURASIP Journal on advances in signal processing*.

[van Eekeren et al., 2008] van Eekeren, A. W. M., Schutte, K., and van Vliet, L. J. (2008). Super-resolution on small moving objects. In *Proc. IEEE ICIP (Image Processing)*.

[Weißet al., 2012] Weiß, A., R., Adomeit, U., Chevalier, P., Landeau, S., Bijl, P., Champagnat, F., Dijk, J., Göhler, B., Landini, S., Reynolds, J. P., and Smith, L. N. (2012). A standard data set for performance analysis of advanced ir image processing techniques. volume 8355, page 835512. SPIE.

[Yaroslavsky et al., 2007] Yaroslavsky, L., Fishbain, B., Shabat, G., and Ideses, I. (2007). Super-resolution in turbulent videos : making profit from damage. *Optics letters*, 32(20) :3038–3040.

[Zhang, 1995] Zhang, Z. (1995). Parameter estimation techniques : A tutorial with application to conic fitting. Technical report, INRIA Research Report No.2676.

[Zhao and Sawhney, 2002] Zhao, W. and Sawhney, H. (2002). Is super-resolution with optical flow feasible ? In Heyden, A., Sparr, G., Nielsen, M., and Johansen, P., editors, *Computer Vision — ECCV 2002*, volume 2350 of *Lecture Notes in Computer Science*, pages 599–613. Springer Berlin / Heidelberg.

5 Conclusion et perspectives

Dans ce manuscrit nous avons décrit l'application de principes de programmation massivement parallèle à deux contextes principaux, l'estimation du flot optique et la super-résolution. Les perspectives associées à ces thématiques ont été présentées dans les sections 3.7 et 4.5.5. Nous allons discuter ici d'autres problématiques, sur lesquelles nous avons débuté des travaux, qui ne nous paraissent pas encore assez mûres pour constituer des chapitres à part entière mais qui constituent des directions de travail futures. Ces problématiques sont les suivantes

- la numérisation d'un environnement 3D dynamique ;
- la perception pour la navigation autonome des micro-drones ;
- la métrologie volumique.

Nous présentons donc ici l'avancement de ces travaux, puis nous finirons par une perspective d'ordre plus générale.

5.1 Numérisation d'un environnement 3D dynamique

Dans les travaux que nous avons présenté, nous n'avons traité de l'information vidéo que dans une représentation 2D, image courante pour le flot optique, ou image de référence pour la SR. Une extension naturelle de ces travaux est de fournir le même type d'information dans une représentation 3D.

Pour pouvoir passer du repère 2D au 3D nous avons besoin d'une estimation de profondeur. Il existe de très nombreux capteurs ou systèmes de capteurs, passifs ou actifs, qui fournissent une carte de profondeur en plus d'une image d'intensité. Pauline Trouvé propose une taxonomie de ces capteurs dans sa thèse [Trouvé, 2012], où elle présente le développement d'une nouvelle solution fondée sur le flou de focalisation. Comme on l'a vu précédemment l'utilisation de FOLKI sur les deux images d'un système d'acquisition stéréoscopique nous fournit une carte de profondeur instantanée. Cette solution est dépendante de la quantité de texture dans l'environnement, ce qui pose des problèmes en particulier en intérieur (murs homogènes).

Dans un contexte intérieur, la Kinect [1] permet aujourd'hui de donner des cartes de profondeur fiables avec une portée limitée à quelques mètres. Rappelons que ce capteur utilise un principe stéréoscopique, mais remplace une des caméras par un projecteur de texture infra-rouge. Il est donc fortement perturbé par le rayonnement solaire, ce qui le rend essentiellement utilisable en intérieur.

L'étape suivante est de combiner cette information de profondeur avec une association temporelle, que nous obtenons ici par FOLKI. L'information géométrique que l'on en déduit est appelée flot de scène "scene flow". Il s'agit d'estimer le déplacement 3D des éléments de la scène observable par la caméra, et de les représenter dans un repère 3D lié à la caméra. En 2009, nous avons proposé une solution associant Kinect et FOLKI, dont un résultat est présenté en figure 5.1. L'utilisation de la Kinect fournit une carte de profondeur Z(t) superposable à l'image I(t) à l'instant t. Chaque pixel $\mathbf{x}$ est donc associé à l'instant t à un point 3D de coordonnées $\mathbf{X}(xv,t)$. Le flot optique fournit une association entre deux positions images temporellement successives : $\mathbf{x} \Leftrightarrow \mathbf{x}+\mathbf{u}(\mathbf{x},t)$. Cette association est directement transposable aux représentations 3D, ce qui fournit un déplacement 3D $\mathbf{U}(\mathbf{x}) = \mathbf{X}(\mathbf{x}+\mathbf{u}(\mathbf{x},t),t+1)-\mathbf{X}(\mathbf{x},t)$ indicé par la position du pixel. En répétant cette opération sur un ensemble d'instans successifs, les flots optiques étant calculés par rapport à une image de référence commune $I(t_0)$, on peut tracer des morceaux de trajectoires dans le repère caméra à l'instant t0. Le traitement temporel consiste simplement à décaler le groupe d'images. Les "filaments" représentés sur la figure 5.1 sont formés avec trois instant consécutifs. Les vitesses des différentes parties de la scène sont observables. On note aussi en empâtement du flot de scène lié à l'utilisation d'une fenêtre de recalage dans FOLKI.

Plutôt que de reprojeter une information géométrique en 3D, on peut chercher à reproduire une information radiométrique 3D. En effectuant une association entre un point 3D et une succession de positions images, comme dans l'application précédente, on peut faire voter les radiométries de chaque image dans l'espace 3D, ce qui permet d'envisager un processus de SR "Shift-and-Add" 3D. Ce processus pourrait ensuite être raffiné par la prise en compte des angles d'incidence des rayon sur la surface observée afin de permettre la correction des effets spéculaires (rejet des intensités aberrantes) ou même l'estimation des propriétés de spécularité de la surface observée. Ceci pourrait permettre d'obtenir des systèmes de numérisation à coût réduit.

1. Pour plus de détail sur son fonctionnement, vous pouvez consulter `www.aurelien.plyer.fr/the-kinect/how-the-kinect-work/`

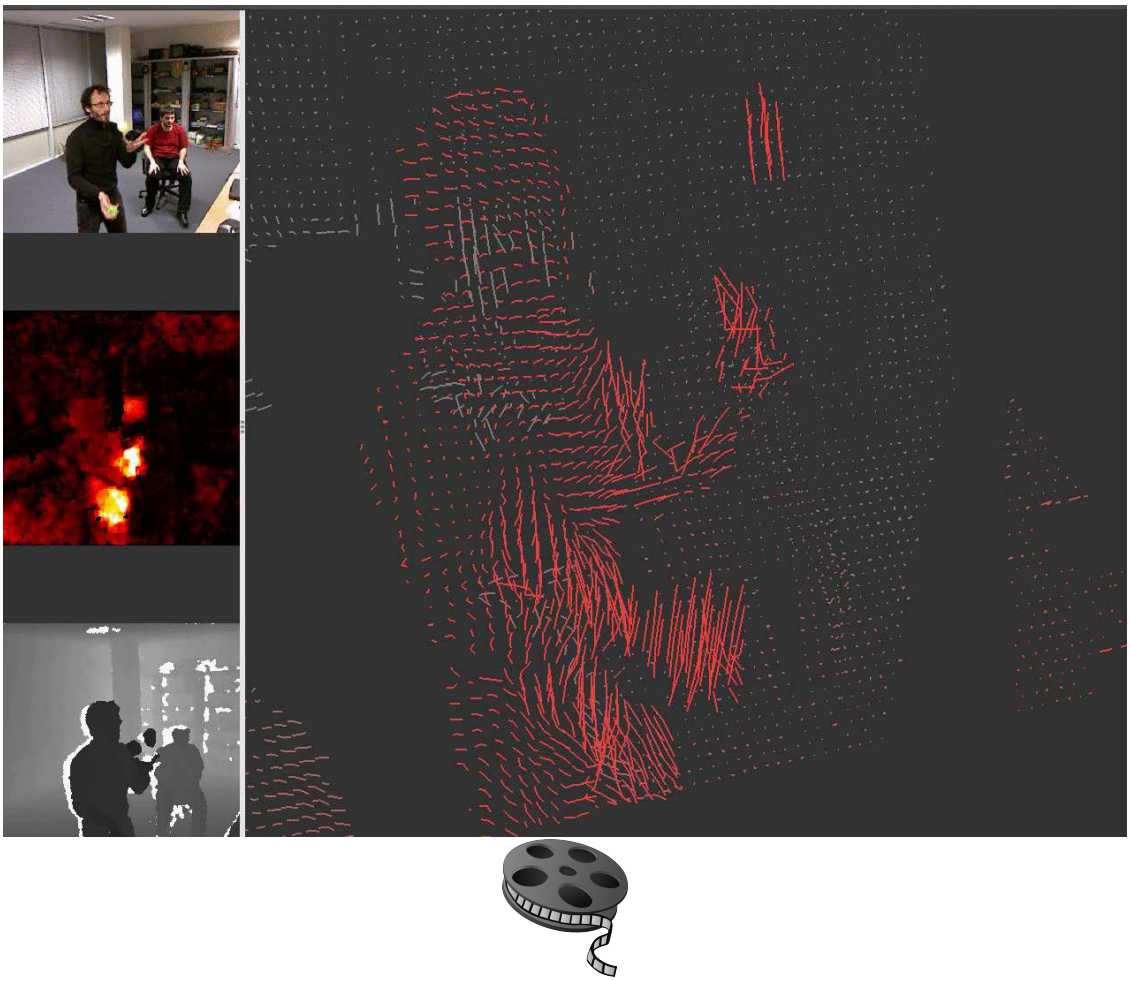


Figure 5.1 – FOLKInect : Flot de scène (à droite) par utilisation d'un Kinect (carte de profondeur présentée dans la vignette à gauche en bas) et du flot optique (vignette à gauche au mileu) calculé à cadence vidéo par FOLKI sur l'image issue du Kinect (vignette en haut à gauche).

5.2 Perception embarquée pour les micro-drones

Dans les applications précédentes, on reste dans un repère lié à une caméra à un instant donné, en utilisant les images temporellement proches pour obtenir une reconstruction 3D. Cependant on n'estime pas les changements de repères successifs de la caméra, c'est-à-dire l'odométrie. Au contraire pour la navigation et l'exploration autonome d'une environnement, c'est une information essentielle.

Les solutions classiques, regroupées sous le terme générique de SLAM (Simultaneous Localization and Mapping), utilisent des associations de primitives dans les images et des structures de filtrage pour estimer les poses successives de la caméra dans un repère fixe, souvent lié à la position initiale. Nous avons durant nos travaux proposé une méthode d'estimation de vraisemblance de la pose d'un capteur de profondeur [Israël and Plyer, 2013]. En utilisant des éléments caractéristiques extraits de cartes de profondeurs (courbure, gradient) nous avons proposé un algorithme massivement parallèle permettant d'évaluer un grand nombre d'hypothèses rapidement en profitant au maximum de la puissance d'un GPU. Une illustration de ce travail est présentée en Figure 5.2.

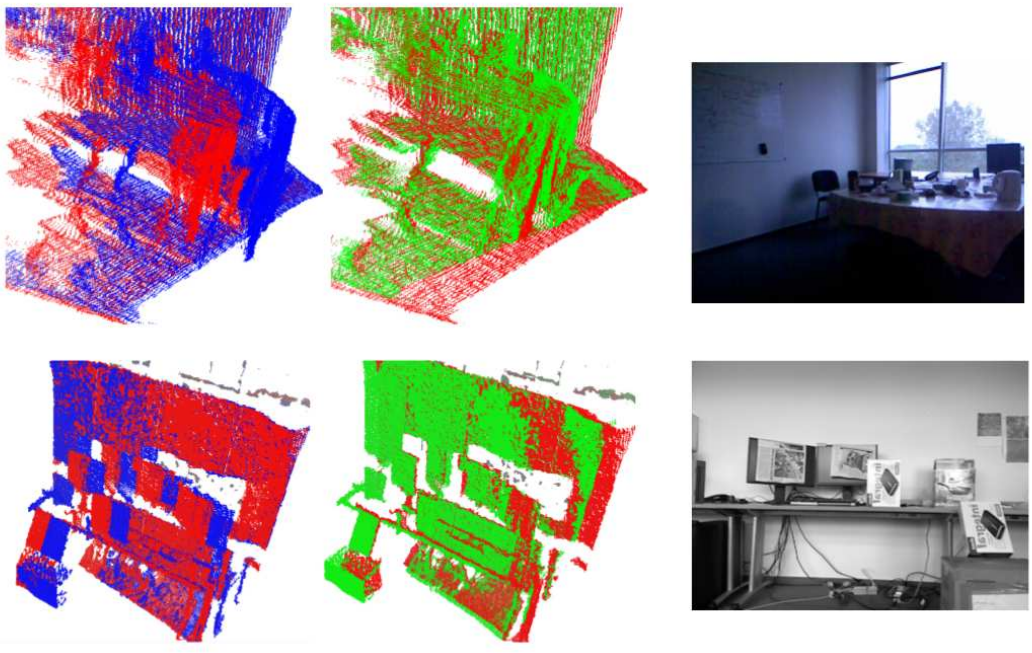


Figure 5.2 – Résultat d'un calcul de pose par un algorithme massivement parallèle utilisant la correspondance de segments extraits des images de profondeur fournies par un Kinect (extrait de [Israël and Plyer, 2013])

Ce type d'approche est utile dans les situations où il n'y a pas ou très peu d'éléments caractéristiques image (milieux très peu texturés en intérieur), pour servir de brique de base à un SLAM visuel ou un RGBD SLAM. Très récemment, des travaux dérivés de [Molyneaux, 2012] utilisant une reconstruction volumétrique de la scène sous une forme de level-set 3D ont conduit à une solution offrant de très bonnes performances en robustesse et précision dans ce contexte.

Une perspective qui nous intéresse plus, étant donné l'orientation de nos travaux, est de travailler directement sur le flot de scène, introduit précédemment, pour y identifier le mouvement propre de la caméra et les objets mobiles par ailleurs. On pourrait utiliser la technique présentée dans [Umeyama, 1991] pour effectuer l'odométrie par la régression du mouvement rigide associé au déplacement capteur. De nouveau, cette technique pourrait servir de base à un filtrage temporel fournissant la trajectoire.

5.3 Métrologie volumique

Au cours de la dernière année de la thèse, nous avons contribué à l'extension des méthodes de métrologie en soufflerie, jusqu'alors essentiellement planes, à des méthodes de mesure volumique : PIV3D et BOS3D. Ces travaux s'inscrivent dans des projets menés en collaboration entre le département DTIM et les département DAFE et DMAE de la Branche Mécanique des Fluides de l'ONERA.

Ces deux méthodes consistent à imager un volume simultanément avec plusieurs caméras et se réfèrent à un modèle de tomographie pour la reconstruction du volume.

La PIV3D utilise en général 4 caméras pour imager une expérimentation PIV où le laser éclaire non plus une nappe plane, mais une nappe d'une certaine épaisseur. Le nuage de particules est alors reconstruit par un procédé de tomographie, puis les volumes à deux instants sont corrélés pour obtenir un champ de déplacement 3D. Nous avons développé une première version de ce pipeline, dont le résultat, un champ de mouvement 3D est présenté en figure 5.3.

En BOS3D, on utilise un grand nombre de caméras pour imager les déviations des rayons lumineux provoquées par les gradients d'indice de l'air dans un écoulement. De nouveau, à partir de ces images de déviation un processus qui s'apparente à une tomographie vectorielle permet de reconstruire le volume d'indice et donc, information importante, la masse volumique du fluide. Nous avons contribué à une première version de ce processus de reconstruction, qui a été développé par Violaine Todoroff et qui fait l'objet des publications [Todoroff et al., 2012b, Todoroff et al., 2012a].

En passant au 3D, on fait bien entendu augmenter la taille des problèmes d'un ordre de grandeur, ce qui nécessite des architectures plus puissantes (multi-GPU, clusters). En revanche,

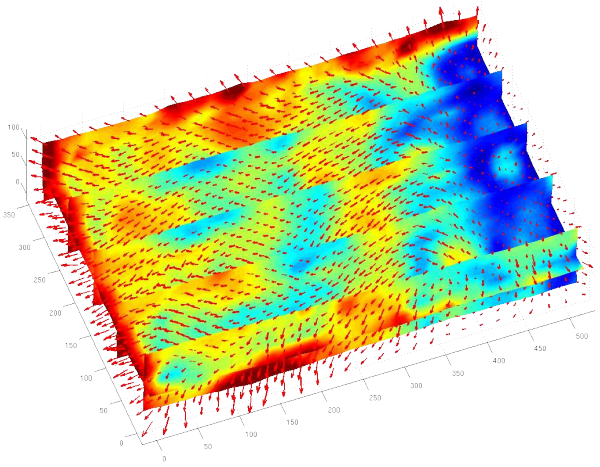


Figure 5.3 – Résultat de calcul de vitesse 3D par une version préliminaire de FOLKI3D (en Matlab) sur des données de PIV3D mises en ligne par B. Wienecke sur le site http ://fluid.irisa.fr/

on accède ainsi à la vraie nature des phénomènes physique au sens où dans la plupart des cas d'intérêt les modèles physiques sont 3D.

5.3.1 Conclusion générale

L'orientation de nos travaux a été d'aborder de manière pragmatique et empirique des problématiques classiques de la vision bas-niveau en nous concentrant sur la performance de calcul et la qualité du résultat sur données réelles. Avec cette approche, nous avons montré que la prise en compte des nouvelles architectures de calcul massivement parallèles conduit à des choix algorithmiques différents de ceux effectués sans considération de l'efficacité du système final, en termes de compromis qualité/performance.

Que ce soit pour l'estimation du flot optique ou l'amélioration de vidéo par super-résolution, nous avons ainsi mis en avant des structures de calcul considérées comme sous-optimales (ou même dépassées, dans le cas de Lucas-Kanade en flot optique) et montré qu'elles permettent de répondre à la problématique. Ces structures, bien adaptées aux architectures massivement parallèles, offrent des performances calculatoires sans équivalent, conduisant à la possibilité de les utiliser dans des produits opérationnels, pour lesquels le délai de réponse est essentiel : robotique, traitement interactif, traitement de données massives.

Cette approche pragmatique nous conduit aussi à éviter les problèmes de sur-apprentissage ou de sur-spécification qui sont encouragés par la sélectivité des processus de publications dans certaines communauté et l'importance donnée, dans ces processus, au ranking sur les benchmarks. On peut très bien obtenir un bon résultat dans une fonction intégrée comme

la super-résolution avec une brique considérée comme peu performante au regard de ces comparatifs, comme FOLKI. De ce point de vue, le site Kitti, quoique proposant des images réelles représentatives d'un domaine concret (la navigation urbaine) ne va pas au bout de cette démarche, car il ne propose pas de critère d'évaluation de l'estimation du flot optique qui soit associé à cette problématique (performance de détection d'obstacles par exemple).

La possibilité de mettre en œuvre, sur données réelles, des systèmes de plus en plus complexes avec des moyens informatiques accessibles à tous doit selon nous, conduire à l'avenir à des évaluations de performances plus globales et non plus focalisées sur une sous-fonction. SRviewer est un exemple limité de ce type plate-forme d'évaluation pour la SR. Nous chercherons à étendre cette démarche aux systèmes plus complexes que nous avons abordés en section précédente.

Bibliographie

[Israël and Plyer, 2013] Israël, J. and Plyer, A. (2013). A brute force approach to depth camera odometry. In Fossati, A., Gall, J., Grabner, H., Ren, X., and Konolige, K., editors, *Consumer Depth Cameras for Computer Vision*, Advances in Computer Vision and Pattern Recognition, pages 49–60. Springer London.

[Molyneaux, 2012] Molyneaux, D. (2012). Kinectfusion rapid 3d reconstruction and interaction with microsoft kinect. In *Proceedings of the International Conference on the Foundations of Digital Games*, pages 3–3. ACM.

[Todoroff et al., 2012a] Todoroff, V., Plyer, A., Le Besnerais, G., Champagnat, F., Donjat, D., Micheli, F., and Millan, P. (2012a). 3d reconstruction of the density field of a jet using synthetic bos images. In *15th International Symposium on Flow Visualization*, Minsk, Belarus.

[Todoroff et al., 2012b] Todoroff, V., Plyer, A., Le Besnerais, G., Champagnat, F., Donjat, D., Micheli, F., and Millan, P. (2012b). Reconstruction tridimensionnelle du champ de masse volumique d'un jet à partir d'images bos simulées. In *13ième Congrès Francophone de Techniques Laser, CFTL 2012*, Rouen, France.

[Trouvé, 2012] Trouvé, P. (2012). *Conception conjointe optique/traitement d'un imageur compact à capacité 3D*. PhD thesis, Onera.

[Umeyama, 1991] Umeyama, S. (1991). Least-squares estimation of transformation parameters between two point patterns. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 13(4) :376–380.

A Publications

Nous listons ici les différentes publication et communications que nous avons effectué durant cette thèse.

A.1 Flot optique

Mes travaux que le **flot optique rapide** sur GPU ont donné lieu à l'ouverture d'un site web ou une version d'évaluation de Folki est disponible au téléchargement et à un congrès sans actes [Plyer et al., 2009].

Flot optique rapide pour la **PIV** : un article de conférences avec actes et comité de lecture [Champagnat et al., 2009], deux articles de revue à comité de lecture [Le Besnerais et al., 2009], et [Champagnat et al., 2011], ainsi que plusieurs présentations dans des séminaires [Le Sant et al., 2010].

Exp Fluids (2011) 50:1169–1182
DOI 10.1007/s00348-011-1054-x

RESEARCH ARTICLE

Fast and accurate PIV computation using highly parallel iterative correlation maximization

F. Champagnat · A. Plyer · G. Le Besnerais · B. Leclaire · S. Davoust · Y. Le Sant

Received: 31 December 2009 / Revised: 26 January 2011 / Accepted: 1 February 2011 / Published online: 17 March 2011

Abstract Our contribution deals with fast computation of dense two-component (2C) PIV vector fields using Graphics Processing Units (GPUs). We show that iterative gradient-based cross-correlation optimization is an accurate and efficient alternative to multi-pass processing with FFT-based cross-correlation. Density is meant here from the sampling point of view (we obtain one vector per pixel), since the presented algorithm, FOLKI, naturally performs fast correlation optimization over interrogation windows with maximal overlap. The processing of 5 image pairs (1,376 × 1,040 each) is achieved in less than a second on a NVIDIA Tesla C1060 GPU. Various tests on synthetic and experimental images, including a dataset of the 2nd PIV challenge, show that the accuracy of FOLKI is found comparable to that of state-of-the-art FFT-based commercial softwares, while being 50 times faster.

1 Introduction

Particle Image Velocimetry (PIV) has become an essential tool for flow diagnosis and is therefore widely used in industrial as well as academic situations. Its current limitation is however, the time necessary to compute the vector fields from the images, which often imposes specific constraints in the schedule of test campaigns. In that respect, the important development of high-speed PIV systems over the last decade appears even more challenging. We propose a solution to shorten dramatically this processing time, based on an algorithm that computes dense 2C vector fields using Graphics Processing Units (GPUs).

GPU has already been compared to other architectures for PIV processing in previous works (Schiwietz and Westermann 2004; Venugopal et al. 2009). These studies concentrated on cross-correlation using FFT, but the speed-up factor for FFT using GPU versus CPU architecture does not exceed three. In this context, real-time computation therefore requires large PC clusters with a GPU at each node (Venugopal et al. 2009). Former real-time realizations also involve parallelization on Field-Programmable Gate Arrays (FPGA) (Iriarte Munoz et al. 2009; Yu et al. 2006). Although efficient and convenient for embedded systems, this solution is far more expensive than GPU to implement, both in terms of hardware cost and software development effort. Interestingly, these architectures get rid of FFT in favor of direct correlation, which is better suited to FPGA architectures. In contrast to these works, the approach proposed hereafter relies on a technique for cross-correlation maximization that departs from the classical FFT method or from direct correlation. Its structure is ideally matched to massively parallel architectures and therefore allows a 50 times speed-up using a single GPU.

The algorithm FOLKI (French acronym for Iterative Lucas–Kanade Optical Flow, Le Besnerais and Champagnat 2005) was originally designed in the context of computer vision for motion estimation in video sequences. But FOLKI proved also very robust and adaptive to many other kinds of images such as those obtained in photomechanics and PIV. It is based on the classical interrogation window

F. Champagnat (✉) · A. Plyer · G. Le Besnerais
Modeling and Information Processing Department,
French Aerospace Lab (ONERA), Chemin de la Hunière,
91761 Palaiseau Cedex, France
e-mail: fchamp@onera.fr

B. Leclaire · S. Davoust · Y. Le Sant
Fundamental and Experimental Aerodynamics Department,
French Aerospace Lab (ONERA), 8 rue des Vertugadins,
92190 Meudon, France

paradigm, but belongs to the family of Lucas–Kanade (LK) algorithms (see Baker and Matthews 2004, for a review). The basic LK method is already known in the PIV community but it is most often associated with Particle Tracking Velocimetry (Miozzi 2004; Stanislas et al. 2008), i.e., low-seeding densities and sparse estimation of displacements. In contrast, the improvement from the basic LK approach implemented in FOLKI naturally relies on the computation of dense fields, i.e., a displacement vector for each image pixel. This leads to a highly regular and parallel algorithm which is much more efficient than previous sparse LK techniques and furthermore specially suited to GPU architectures. Of course, the fact that one vector per pixel be obtained should not be confused with the spatial resolution of the method, which is tightly linked with the window size, as for any other window-based PIV technique.

The principle of FOLKI is the following: around each pixel, a fixed size interrogation window (IW) is defined, and a cross-correlation measure is defined as a Sum of Squared Differences (SSD) between the IW and a displaced window in the consecutive image. In contrast to mainstream PIV algorithms that perform extensive search over discrete pixel grid with a FFT correlation, this SSD is minimized using an iterative Gauss–Newton (GN) descent. On a general point of view, when initialized not too far (say 3 pixels) from the true displacement, it is known that the convergence of GN is fast, reaching a precision of the order of a tenth of a pixel in typically less than 5 iterations. As PIV images may often be characterized by larger displacements, a multiresolution scheme is used to avoid local minima. An image pyramid is built, starting from the acquired images, which correspond to the ground level. This is done by successively performing low-pass filtering and decimation, leading to successively smaller images. As each step also divides the displacements by two, this has to be done until the top-level images have displacements compatible with GN iterations initialized with a displacement equal to zero. This leads to first rough estimates, whose values are successively refined by descending the pyramid levels (whereby the spatial resolution is also refined). Such a coarse-to-fine multiresolution scheme has proven very efficient in optical flow methods in computer vision (Bergen et al. 1992) and is also used in PIV (see for instance Ruhnau et al. 2005). As will be shown in Sect. 2, an iterative image deformation technique (Lecordier and Trinite 2003; Stanislas et al. 2008) is implicitly embedded in the descent iteration.

Multiresolution, gradient descent and a dense velocity output are more often encountered in so-called "optical flow" methods (Corpetti et al. 2006; Ruhnau et al. 2005). However, FOLKI is a window-based method, with no spatial regularization such as a Horn & Schunk-like term (Corpetti et al. 2006). It should be compared to classical FFT correlators, which we will do in the assessment part of this paper.

The paper is organized as follows: Sects. 2 and 3 are devoted to a description of the basic FOLKI algorithm and to the principle of its GPU implementation. They consist in a more detailed version of the material presented in PIV'09 (Champagnat et al. 2009). Section 4 then describes specific improvements which were added to address situations typically encountered in PIV, the corresponding GPU implementation is then referred to as FOLKI-PIV. A detailed performance assessment follows, where FOLKI-PIV is characterized and benchmarked against a state-of-the-art commercial PIV software using FFT-based cross-correlation. First, synthetic images are specifically generated in order to determine its spatial resolution and its sensitivity to low-seeding densities and to noise. This is done in Sect. 5. Then, the comparative assessment is extended to experimental images, in Sect. 6. The level of peak-locking bias and sensitivity to actual measurement noise are explored by considering case A of the second PIV challenge (Stanislas et al. 2005), and results from a test campaign recorded at ONERA are introduced to show the advantages of dense sampling and illustrate how FOLKI-PIV deals with solid walls thanks to the use of masks. Finally, conclusive remarks and perspectives on future work are gathered in Sect. 7.

2 Basic FOLKI algorithm

2.1 Multiresolution setting and notations

The notations for the following derivations are illustrated in Fig. 1: observed image intensity at discrete positions $\boldsymbol{k} = [k, l]^t \in \mathcal{G} = \{0, \ldots, K-1\} \times \{0, \ldots, L-1\}$ and time indexes $t \in \{0, dt\}$ is denoted $I(\boldsymbol{k}, t)$. In the sequel, all summations $\sum_{\boldsymbol{k}}$ refer to summation on $\mathcal{G}$. We will sometimes use the notation $I(\cdot, t)$ for the function $\boldsymbol{k} \mapsto I(\boldsymbol{k}, t)$.

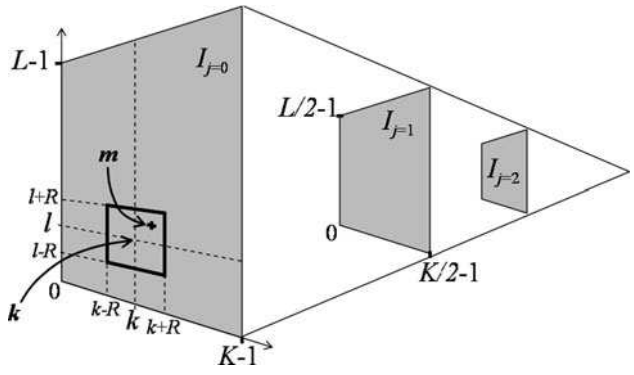


Fig. 1 Notations for image intensities within image pyramid and interrogation window

We use a multiresolution framework (Bergen et al. 1992): image intensity $I_j(\boldsymbol{x},t)$ at any real position $\boldsymbol{x}=[x,y]^t$, and any resolution level $j>0$ is computed by means of a Gaussian pyramid (Burt and Adelson 1983): starting from a level j, the image of level $j+1$ is obtained by applying a low-pass filter on the intensity $I_j(.,t)$ and then retaining one pixel out of a square of 2×2 pixels. Thus, image $j+1$ is four times smaller than image j, while displacements are divided by two. For each level, the spatial image gradient $\nabla I_j(\boldsymbol{x},t)$ is computed by a first-order centered difference scheme.

In this framework, displacements of the initially recorded image (level $j=0$) are thus divided by 2^j at level j. This allows to settle the question of the first estimate for the Gauss–Newton iterations: indeed, initialization at the highest level $J-1$ (where J is the total number of levels) can be done with zero displacement, as long as J is chosen so that displacements to find at level $J-1$ are sufficiently small in the whole image. In practice, for standard PIV images with an 8 pixel dynamic range, $J=3$ is enough for this process to work successfully, without being trapped in local minima.

2.2 A Lucas–Kanade algorithmic core

FOLKI relies on a Lucas–Kanade paradigm (Baker and Matthews 2004), which has been extended in Le Besnerais and Champagnat (2005) so as to provide a convergent iterative estimation of the dense displacement field $\boldsymbol{u}$.

In a majority of current PIV algorithms, the displacement of a given IW is found by first calculating the cross-correlation score of all possible displacements, then finding the maximum correlation peak, and finally refining its position by sub-pixel fit or interpolation. Usually, this process is repeated iteratively with decreasing window sizes, and at each step, the IWs are shifted using the previous estimation of displacement. The LK algorithm is also a window matching technique, but differs on both the objective criterion and on the way to obtain sub-pixel displacements. Cross-correlation maximization is in fact achieved by minimizing a Sum of Squared Differences (SSD), in which the displacement to be found appears directly as a real-valued (and not integer-valued) quantity. This is achieved thanks to a Gauss–Newton iterative descent. The SSD criterion around pixel $\boldsymbol{k}$ at level j writes

$$\sum_{\boldsymbol{m}} w(\boldsymbol{m}-\boldsymbol{k})\big(I_j(\boldsymbol{m},0)-I_j(\boldsymbol{m}-\boldsymbol{u}(\boldsymbol{k}),dt)\big)^2 \tag{1}$$

where w is a weight function whose support defines the interrogation window $\mathcal{W}(\boldsymbol{k})$:

$$\mathcal{W}(\boldsymbol{k})=\{\boldsymbol{m}\in\mathcal{G}\mid w(\boldsymbol{m}-\boldsymbol{k})>0\}. \tag{2}$$

The following derivations are valid for any kind of weight function. Popular choices are rectangular and Gaussian weights. All the experimental results presented in this paper use a standard rectangular IW ($w_{\mathrm{I}}(\boldsymbol{m})=1/(2R+1)^2$ for $\boldsymbol{m}\in\{-R,\ldots,R\}\times\{-R,\ldots,R\}$). For convenience of coding, we use only odd IW dimensions.

Now addressing the minimization process, let us assume that an initial guess $\boldsymbol{u}_0(\boldsymbol{k})$ of the displacement is available and is a good approximation of the sought displacement $\boldsymbol{u}(\boldsymbol{k})$, i.e., $\boldsymbol{u}(\boldsymbol{k})-\boldsymbol{u}_0(\boldsymbol{k})\approx 0$. The Gauss–Newton iteration derives from the following first-order expansion of Eq. 1 around $\boldsymbol{u}_0(\boldsymbol{k})$, with $\boldsymbol{u}(\boldsymbol{k})-\boldsymbol{u}_0(\boldsymbol{k})$ as a small parameter:

$$\sum_{\boldsymbol{m}} w(\boldsymbol{m}-\boldsymbol{k})\big(I_j(\boldsymbol{m},0)-I_j(\boldsymbol{m}-\boldsymbol{u}_0(\boldsymbol{k}),dt) + \nabla I_j(\boldsymbol{m}-\boldsymbol{u}_0(\boldsymbol{k}),dt)^t(\boldsymbol{u}(\boldsymbol{k})-\boldsymbol{u}_0(\boldsymbol{k}))\big)^2. \tag{3}$$

Equation 3 is a linear least-squares criterion, which can already be optimized to yield $\boldsymbol{u}(\boldsymbol{k})$ by solving a 2×2 linear system.

In Bouguet (2000), a faster scheme was proposed. It relies on a slightly different form of Eq. 1:

$$\sum_{\boldsymbol{n}} w(\boldsymbol{m}-\boldsymbol{k})\big(I_j(\boldsymbol{m}+\boldsymbol{u}(\boldsymbol{k})-\boldsymbol{u}_0(\boldsymbol{k}),0) -I_j(\boldsymbol{m}-\boldsymbol{u}_0(\boldsymbol{k}),dt)\big)^2. \tag{4}$$

In this criterion, instead of searching $\boldsymbol{u}(\boldsymbol{k})$ in the image at time dt as in Eq. 1, image at time dt is shifted by the estimate $\boldsymbol{u}_0(\boldsymbol{k})$ and the increment $\boldsymbol{u}(\boldsymbol{k})-\boldsymbol{u}_0(\boldsymbol{k})$ is applied to the image at time 0. A Taylor expansion of $I_j(\cdot,0)$ around $\boldsymbol{m}$ then yields

$$\sum_{\boldsymbol{m}} w(\boldsymbol{m}-\boldsymbol{k})\big(I_j(\boldsymbol{m},0)-I_j(\boldsymbol{m}-\boldsymbol{u}_0(\boldsymbol{k}),dt) + \nabla I_j(\boldsymbol{m},0)^t(\boldsymbol{u}(\boldsymbol{k})-\boldsymbol{u}_0(\boldsymbol{k}))\big)^2. \tag{5}$$

The advantage of this "inverse additive" approach (Baker and Matthews 2004) is that the spatial intensity gradient $\nabla I_j(\boldsymbol{m},0)$ is computed only once for each resolution level j, while in Eq. 3 the spatial gradient, $\nabla I_j(\boldsymbol{m}-\boldsymbol{u}_0(\boldsymbol{k}),dt)$ has to be computed at each iteration.

2.3 Dense LK algorithm

As shown in Le Besnerais and Champagnat (2005), if one wants to apply iterative techniques based on expansions (3) or (5) *at each pixel*—which is the usual goal in computer vision—the overall cost is prohibitive, because, for each iteration, it requires $(2R+1)^2$ interpolations per pixel due to the $I_j(\boldsymbol{m}-\boldsymbol{u}_0(\boldsymbol{k}),dt)$ term (and also because of the gradient $\nabla I_j(\boldsymbol{m}-\boldsymbol{u}_0(\boldsymbol{k}),dt)$ which appears in Eq. 3). Faster schemes can in fact be obtained by using only one interpolated image per iteration. To do so, we introduce the following notation:

$$I_j^{\boldsymbol{u}_0}(\boldsymbol{m},dt) \triangleq I_j(\boldsymbol{m}-\boldsymbol{u}_0(\boldsymbol{m}),dt). \quad (6)$$

As this expression shows, image $I^{\boldsymbol{u}_0}$ is "warped" according to the current displacement field estimate $\boldsymbol{u}_0$ *evaluated at each pixel* $\boldsymbol{m}$, as opposed to a warping with one value of $\boldsymbol{u}_0$ per IW. In order to obtain a convergent scheme based on the unique warped image (6), FOLKI thus uses the approximation proposed in Le Besnerais and Champagnat (2005):

$$\boldsymbol{u}(\boldsymbol{k}) - \boldsymbol{u}_0(\boldsymbol{m}) \approx 0, \quad \forall \boldsymbol{m} \in \mathcal{W}(\boldsymbol{k}) \quad (7)$$

Using Eq. 7, one then derives a first-order expansion of Eq. 1:

$$\sum_{\boldsymbol{m}} w(\boldsymbol{m}-\boldsymbol{k})\Big(I_j(\boldsymbol{m},0) - I_j^{\boldsymbol{u}_0}(\boldsymbol{m},dt) + \nabla I_j(\boldsymbol{m},0)^t(\boldsymbol{u}(\boldsymbol{k}) - \boldsymbol{u}_0(\boldsymbol{m}))\Big)^2. \quad (8)$$

Minimization of Eq. 8 finally amounts to solving a 2×2 local system

$$\boldsymbol{H}(\boldsymbol{k})\boldsymbol{u}(\boldsymbol{k}) = \boldsymbol{c}(\boldsymbol{k}). \quad (9)$$

Let us detail the computation of the matrices $\boldsymbol{H}(\boldsymbol{k})$ for all pixel index $\boldsymbol{k}$. While searching for the stationary point which minimizes Eq. 8, one obtains:

$$\boldsymbol{H}(\boldsymbol{k}) = \sum_{\boldsymbol{m}} w(\boldsymbol{m}-\boldsymbol{k})\big(\nabla I_j(\boldsymbol{m},0)\nabla I_j(\boldsymbol{m},0)^t\big). \quad (10)$$

If $\boldsymbol{H}$ denotes the matrix valued function $\boldsymbol{k} \mapsto \boldsymbol{H}(\boldsymbol{k})$ of the pixel index, Eq. 10 for all pixels $\boldsymbol{k}$ can be globally written as a convolution:

$$\boldsymbol{H} = w * \big(\nabla I_j(\cdot,0)\nabla I_j(\cdot,0)^t\big), \quad (11)$$

where $*$ stands for the convolution of the scalar weight function w with each component of the matrix valued function which is inside the parenthesis. In the same way, the right-hand side vectors $\boldsymbol{c}(\boldsymbol{k})$ of Eq. 9 can be all computed by convolutions as follows

$$\boldsymbol{c} = w * (\epsilon \nabla I_j(\cdot,0)) \quad (12)$$

$$\epsilon = I_j(\cdot,0) - I_j^{\boldsymbol{u}_0}(\cdot,dt) - \nabla I_j(\cdot,0)^t \boldsymbol{u}_0 \quad (13)$$

As a result, at each iteration, local systems (9) for all pixels can be constructed *simultaneously* by Eqs. (6–11–13)—note however, that Eq. 11 can be computed once for all iterations, as already mentioned.

2.4 Overall algorithm and general comments

The global structure of the algorithm, summarized in Table 1, is a coarse-to-fine multiresolution scheme over J levels, with a fixed number N of Gauss–Newton iterations per level. As mentioned above, J should be chosen depending on the expected displacements in the image, and N may depend on the quality of the images and on the radius R of the IWs. More details on the way to choose these parameters will be given in Sects. 5 and 6. Also, note that the current version of FOLKI at use in ONERA gathers additional features specially adapted to PIV, which will be described in Sect. 4. Here, we simply comment on some specificities of the algorithm which are already contained in the above derivation.

A first remark is that, as shown in Table 1, each iteration begins with an image warp (6). Hence, FOLKI can be related to *image deformation techniques* (Lecordier and Trinite 2003; Stanislas et al. 2008). But, as FOLKI computes a dense vector field, the deformation is available at each pixel without velocity interpolation.

The dense character of the vector field also deserves further comment. First, it should be mentioned that it is an unavoidable building block of the algorithm: solving Eq. 9 for a restricted ensemble of spatial locations cannot be envisaged here, since computations (12, 13) require the availability of velocities at a much larger number of locations. Viewed in the PIV context, this by-product of the computer vision origin of FOLKI may however, appear useless, or even detrimental in terms of computational time. Indeed, as will be shown in Sect. 5, similarly to other PIV approaches based on window matching, FOLKI's spatial resolution remains related to the window size. A first important remark which justifies our choice is that this density is not an overload to the computational time: tests performed on a CPU implementation showed that the

Table 1 Pseudo GPU code of FOLKI

input: $I(\cdot,0)$ and $I(\cdot,dt)$
output: $\boldsymbol{u} = (u,v)$
begin
send $I(\cdot,0)$, $I(\cdot,dt)$ from CPU memory to GPU global memory
GPU: compute Burt pyramids *(SC)*
for $j = J-1:-1:0$
  GPU: compute $\nabla I_j(\cdot,0)$ *(SC)*
  GPU: compute $\boldsymbol{H}$ *(PW)*
  for $n = 1:N$, **iterate:**
    GPU: compute $I_j^{\boldsymbol{u}_0}(\cdot,dt)$ *(II)*
    GPU: compute ϵ *(PW)*
    GPU: compute $\boldsymbol{c}$ *(PW+SC)*
    GPU: solve local systems (9) *(PW)*
  GPU: upsample $\boldsymbol{u}$ vector fields *(SC)*
(option 1) GPU: compute output image result and transfer it into GPU visualization memory
(option 2) send $\boldsymbol{u} = (u,v)$ result from GPU to CPU
end

GPU: GPU functions, *II*: image interpolation, *SC*: separable convolution, *PW*: pixelwise operation

computational time of a dense vector field with FOLKI was comparable to that of a classical sparse computation with a commercial PIV software. This is due to the high degree of optimization of FOLKI. Besides, and paradoxically, it is in fact this dense character that leads to a highly regular and parallel algorithm which precisely allows the considerable speed-up provided by the GPU. In addition, density provides an appreciable degree of freedom of result sampling, e.g., to finely evidence vortex cores or investigate flows close to walls, see Sect. 6.2 for an example.

3 Implementation on a GPU

An implementation has been developed in C++ and CUDA language for NVIDIA GPU and tested on different hardwares (generic graphic unit of a laptop, and a dedicated GPU on a PC workstation) with Linux and Windows OS.

Different packages of FOLKI are freely available on the ONERA website, at the address: http://www.onera.fr/dtim-en/gpu-for-image/index.php. Note that the open source Linux package strictly corresponds to the algorithm described in Sect. 2, whereas the Windows packages include the additional features described in Sect. 4.

The efficiency of the GPU implementation stems from the fact that FOLKI relies mainly on three types of computations, image interpolation (II), pixelwise operations (PW) and separable convolution (SC), see Table 1. These computations are performed very efficiently on a GPU, see Champagnat et al. (2009) for a more detailed account on GPU architecture and how to make profit of it. Two main features can be highlighted:

1. Image bilinear interpolation is hardwired on a GPU, it is thus performed at a cost which is negligible compared to a CPU.
   Higher order interpolation can also be performed very efficiently thanks to the algorithm of Ruijters et al. (2008) that combines multiple bilinear interpolations to perform one bicubic B-spline interpolation.
2. It is fundamental to limit the number of CPU-GPU transfer which are particularly time-consuming. The GPU pseudo-code presented in Table 1 is designed to minimize the number of CPU-GPU image transfers. Note the optional steps at the end of Table 1: if the code is used only to visualize an output image which depends on the computed velocity field (for instance an image of the vorticity field), it is much faster to compute this image with the GPU and then to transfer it directly into the visualization memory of the GPU. This mode can be very useful for fast parameter tuning of an experiment.

4 Adapting FOLKI to PIV context

We now discuss some extensions of FOLKI, directly dictated by the typical constraints of PIV experiments. These developments principally aim at increasing the accuracy, properly handling boundaries and giving the user a quality criterion on the obtained vector fields. Results using this improved version, which we call FOLKI-PIV, are presented in Sects. 5 and 6.

4.1 Third-order B-spline interpolator

As mentioned in Sect. 4, the user may choose whether the image interpolation is performed via simple bilinear interpolation, or using third-order B-splines. Having such a choice may prove relevant in order to adapt to the image characteristics, as will be shown for instance in Sect. 6.1.

4.2 Symmetric matching cost

Following symmetric SSD criteria like

$$\sum_{\boldsymbol{m}} w(\boldsymbol{m}-\boldsymbol{k})\left(I_j\left(\boldsymbol{m}+\frac{\boldsymbol{u}(\boldsymbol{k})}{2},0\right)-I_j\left(\boldsymbol{m}-\frac{\boldsymbol{u}(\boldsymbol{k})}{2},dt\right)\right)^2, \tag{14}$$

have been proposed by many authors (Keller and Averbuch 2004; Zhao and Sawhney 2002), in order to suppress the dissymmetry of classical SSD costs, increase precision and robustness against occlusions. When upgrading toward FOLKI-PIV, we chose to implement this approach rather than the simple original SSD criterion (1).

In practice, Eq. 14 can be handled in a very similar manner as Eq. 1. Replace $u(\boldsymbol{k})$ by $u_0(\boldsymbol{m})+(u(\boldsymbol{k})-u_0(\boldsymbol{m}))$ in Eq. 14, then take first-order approximation for both images based on Eq. 7. Finally, one gets modified expressions for Eqs. 11, 12 and 13. For instance, the 2×2 matrix $\boldsymbol{H}(\boldsymbol{k})$ associated to pixel $\boldsymbol{k}$ now writes:

$$\begin{aligned}\sum_{\boldsymbol{m}} w(\boldsymbol{m}-\boldsymbol{k})\Bigg(&\nabla I_j\left(\boldsymbol{m}-\frac{\boldsymbol{u}_0(\boldsymbol{m})}{2}\right)\nabla I_j\left(\boldsymbol{m}-\frac{\boldsymbol{u}_0(\boldsymbol{m})}{2}\right)^t\\&+\nabla I_j\left(\boldsymbol{m}+\frac{\boldsymbol{u}_0(\boldsymbol{m})}{2},dt\right)\nabla I_j\left(\boldsymbol{m}+\frac{\boldsymbol{u}_0(\boldsymbol{m})}{2},dt\right)^t\Bigg),\end{aligned} \tag{15}$$

(the '0' in $\nabla I_j(.,0)$ has been omitted for concision). The overall structure of the symmetric algorithm remains similar to the one in Table 1, except that expression (15) has to be recomputed at each iteration using spatial gradients and interpolations of both images. In this process, the computational time is multiplied by 2 compared to the basic algorithm of Sect. 2.

4.3 Robustness to varying illumination

In contrast to a zero-normalized cross-correlation (ZNCC) maximization objective, which is classically used in PIV, objectives defined by SSD minimization such as Eq. 14 are less robust to varying illumination conditions. Of course, global illumination changes can easily be handled by equalization using image gain and offset adjustment before feeding the algorithm with the corrected image pairs. But this approach will not work when, for instance, the lighting difference varies across the field of view, a situation which is encountered in PIV, see for instance both examples of Sect. 6. In this case, some kind of local equalization is required.

We follow hereafter the logic of mean and standard deviation normalization; note that the min–max normalization of Westerweel (1993) could also be implemented cheaply on a GPU. The principle of such a local equalization is to replace the image intensity values $I(\cdot,t)$ ($t = \{0, dt\}$) by normalized intensities $\widetilde{I}(\cdot,t)$. For an IW centered on a pixel $\boldsymbol{k}$, the vector of normalized intensities of pixels $\boldsymbol{m}$ inside the IW, $\{\widetilde{I}(\boldsymbol{m},t)\}_{\boldsymbol{m}\in\mathcal{W}(\boldsymbol{k})}$, should have approximately zero mean and unit standard deviation. If the displacement field is zero, such a normalization simply writes

$$\widetilde{I}(\boldsymbol{m},t) = (I(\boldsymbol{m},t) - M(\boldsymbol{k},t))/\sigma(\boldsymbol{k},t), \quad t = \{0, dt\},$$

where the local empirical mean $M(\cdot,t)$ and standard deviation $\sigma(\cdot,t)$ are computed simultaneously for all pixels by pixelwise operations and separable convolution.

For each iteration of FOLKI-PIV, the current displacement field is not zero anymore and local means and standard deviation should be computed on the warped images $I^{-\boldsymbol{u}_0/2}(\cdot,0)$ and $I^{\boldsymbol{u}_0/2}(\cdot,dt)$ (using the notation from Eq. 6), in order to write a normalized symmetric criterion from Eq. 14. A fast approximation is to perform the normalization only once, at the beginning of each level, and then performing the GN iterations on these images. With such a strategy, there is nearly no extra cost and empirical comparisons of both schemes—exact or approximate—show their equal effectiveness.

4.4 Boundary handling

This problem should be adequately addressed not only to process pixels located near the boundary of the field of view but also to take masks into account. A mask is a binary image aimed at excluding some pixels from the estimation process, because they belong to some rigid object (wing, measurement device, etc.) present in the field of view; see for instance the real dataset of Sect. 6.2. In the sequel, the image support refers to the set of non-masked pixels.

It is quite delicate to find an optimal way to handle boundaries in window-based displacement estimation. Indeed the *support of the estimation*, i.e., the pixels for which the system (9) can be constructed and inverted, varies with the estimate: if the displacement field locally tends to escape from the image, the support of the estimation consequently "moves back" away from the boundary. Such effect is even more pronounced in FOLKI, because of the dense estimation and also of the multiresolution process.

The proposed solution retained in FOLKI-PIV relies on dynamic masks which are updated at each iteration. Excluded pixels are (*i*) those whose current displacement vector falls outside of the image support (in the symmetric case, the displacement fields which are used are either $\boldsymbol{u}_0/2$ or $-\boldsymbol{u}_0/2$); (*ii*) those whose IW contains more than 80% of already excluded pixels. Displacement vectors are computed for the remaining pixels, then the holes are filled with nearest valid vectors before the next iteration.

4.5 Correlation quality

When analyzing PIV images, imperfect lighting, particle loss, and CCD noise will impact the quality of the correlation and thus increase the uncertainty of a computed vector. To yield a quality criterion to the user, we included in FOLKI-PIV a computation of the correlation peak height, as is traditionally done in PIV. This is done by first warping the images by the final displacement, so as to get images $I^{-\boldsymbol{u}/2}(\cdot,0)$ and $I^{\boldsymbol{u}/2}(\cdot,dt)$. Then, mean and standard deviation normalization is applied on these images, yielding $\widetilde{I}^{-\boldsymbol{u}/2}(\cdot,0)$ and $\widetilde{I}^{\boldsymbol{u}/2}(\cdot,dt)$. As a preliminary step, the ZNSSD score S_{ZNSSD} is then computed. This quantity is simply the residual of the symmetric SSD criterion (14) applied to these zero- normalized images:

$$S_{\text{ZNSSD}} = \sum_{\boldsymbol{m}} w(\boldsymbol{m}-\boldsymbol{k})\left(\widetilde{I}_0^{-\boldsymbol{u}/2}(\boldsymbol{m},0) - \widetilde{I}_0^{\boldsymbol{u}/2}(\boldsymbol{m},dt)\right)^2. \tag{16}$$

It finally turns out that S_{ZNSSD} is directly related to the classical correlation score S_{ZNCC}, or height of the correlation peak, which is defined as

$$S_{\text{ZNCC}} = \sum_{\boldsymbol{m}} w(\boldsymbol{m}-\boldsymbol{k})\widetilde{I}_0^{-\boldsymbol{u}/2}(\boldsymbol{m},0)\widetilde{I}_0^{\boldsymbol{u}/2}(\boldsymbol{m},dt). \tag{17}$$

Simple algebra, see for instance Pan et al. (2007, Appendix A), indeed shows that one has

$$S_{\text{ZNCC}} = 1 - \frac{S_{\text{ZNSSD}}}{2}. \tag{18}$$

Consequently, even though FOLKI-PIV's objective is formulated differently as in classical PIV, the quality of its results

may be assessed—and vectors validated or not—in the same way, using S_{ZNCC}. $S_{ZNCC} = 1$ will indicate a perfect matching, while $S_{ZNCC} = 0$ is the theoretical limit of no correlation. Further elements on the way we use this score in practice will be given in Sects. 5 and 6.

5 Performance assessment: synthetic data

In the following, we use synthetic PIV images to study FOLKI-PIV's spatial frequency response, along with the effects of low seeding and noise on the reliability of the results. In order to determine whether the computation choices underlying FOLKI-PIV result in a different behavior as traditional PIV algorithms, we first present simple test cases for which the behavior of these algorithms is already documented. Then, we provide comparison of FOLKI-PIV's result to that of a state-of-the-art commercial software, hereafter denoted CPIV.

In the following, CPIV's estimation of an IW's displacement relies on the FFT-based computation of the cross-correlation map, followed by a Gaussian sub-pixel interpolation of the correlation peak. This process is embedded in an adaptive multi-pass scheme, in which the displacements are progressively refined from their previous estimates; this can be done with IWs of gradually decreasing size. Between each pass, rejection of spurious vectors is performed using a median filter, outliers being replaced either by other correlation maxima or by local interpolation, and the vector field is filtered with a 3×3 pixels Gaussian.

5.1 Parameters for the tests

The synthetic images used in the present work were generated with the EUROPIV Synthetic Image Generator (S.I.G.) which is described in Lecordier and Westerweel (2003). Keeping physical units in pixels, we use $1{,}025 \times 1{,}025$ images with a fully covered 8 bit range. Particles show as 2 pixels diameter Gaussian intensity distribution. The intensity level of a given particle depends on its out-of-plane position with respect to the $l_w = 2$ pixels width Gaussian-shaped light sheet that illuminates the scene. If N_p is the total number of particles in a volume V illuminated by the laser sheet, then particle density seen in the image is $N_d = \frac{N_p l_w}{V}$. Unless specified otherwise, particle density is set to $N_d = 0.02$ and no CCD noise is added, yielding sample images such as shown in Fig. 2. Displacement fields are applied symmetrically forward and backward to an initial cloud of randomly located particles. For each test case, identical displacement patterns are repeated in different zones of a given image pair, so that 25 image pairs are sufficient to achieve statistical convergence of the results.

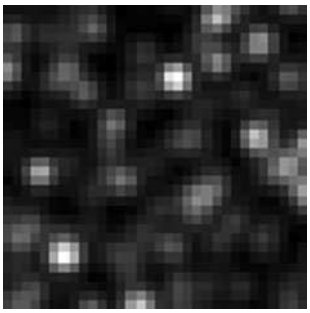


Fig. 2 32×32 pixel close-up at a S.I.G. image with $N_d = 0.02$

FOLKI-PIV's window radius R is varied from 5 to 31 pixels, and the interrogation window has a standard rectangular weight. We use $J = 1$ level because the pixel displacements in these tests are less than 2 pixels. Convergence is reached for $N = 3$ iterations. The interpolation is performed with a third-order B-spline.

CPIV is also used with various IW sizes (8×8, 16×16, 32×32 and 64×64 pixels) with rectangular weight. For a given result, the calculation is done thanks to a multi-pass scheme composed of 4 passes with the same IW size. We use a Whittaker pixel interpolation method and overlap is set to 75%.

5.2 Spatial frequency response

Following Scarano and Riethmuller (2000), the frequency response of a PIV algorithm can be evaluated using a sinusoidal shear displacement test:

$$(U, V) = \left(A \sin\left(2\pi \frac{Y}{\lambda} \right), 0 \right) \tag{19}$$

where X and Y are the horizontal and vertical coordinates and U and V the associated displacement components.

We here reproduce this displacement field with amplitude A set to 2 pixel, and the wavelength λ varied from 20 to 400 pixels. Figure 3 compares the ground truth and the average value found by FOLKI-PIV for the U component, for case $\lambda = 200$ processed with $R = 10$ IWs. For each image corresponding to a given (λ, R) couple, we computed the ratio between the estimated amplitude of the sinusoid $A_{\text{FOLKI}-\text{PIV}}$ and the ground truth value $A = 2$ pixel. The evolution of this ratio as a function of the normalized IW size $2R/\lambda$ is plotted in Fig. 4. Although we used IW radii ranging from $R = 5$ to $R = 31$, all values of $A_{\text{FOLKI}-\text{PIV}}/A$ nearly collapse on a cardinal sine curve, which is the frequency response of a $[-R, R]$ sliding average. In comparison, as shown by Scarano and Riethmuller (2000), iterative multigrid methods with isotropically weighted

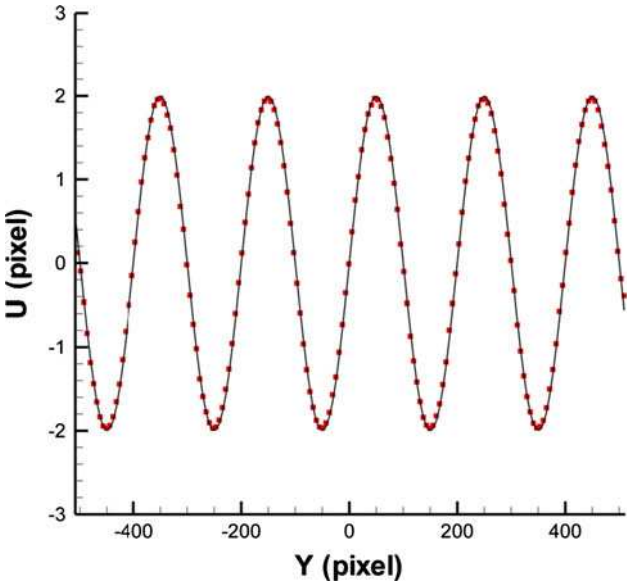


Fig. 3 Sinusoidal shear displacement with $A = 2$ pixels and $\lambda = 200$ pixels. Ground truth (*black curve*) and average displacement found by FOLKI-PIV with $R = 10$ IWs, downsampled every 6 pixels

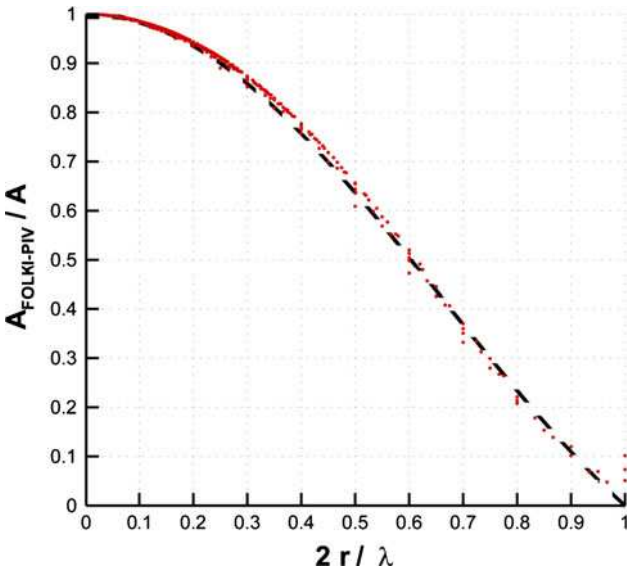


Fig. 4 Sinusoidal shear displacement with $A = 2$ pixels: amplitude ratio $A_{\text{FOLKI-PIV}}/A$ as a function of the normalized window size $2R/\lambda$ (*red symbols*), for IW radii R ranging from 5 to 31. The *dashed line* is the response of a $[-R, R]$ sliding average (cardinal sine function)

IWs follow a similar trend, but with an amplitude damping compared to the ideal sliding average, which we do not observe with FOLKI-PIV.

5.3 Resolution versus noise

One other way to evaluate the effective spatial resolution is to recover the response of the algorithm to a sharp spatial

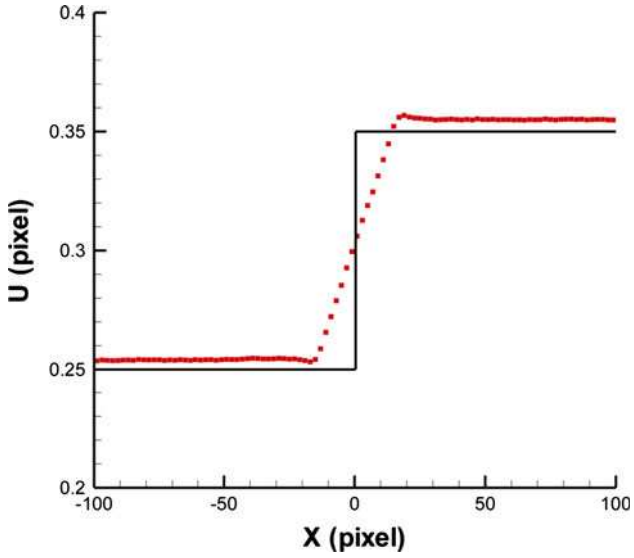


Fig. 5 Sharp horizontal step: displacement estimated by FOLKI-PIV with $R = 15$ (*red dots*, downsampling every 2 pixels) and ground truth (*black curve*). Note that the displacement error for FOLKI-PIV which appears from this figure (roughly 0.005 pixel) remains well below the usual PIV uncertainty

step in the displacement field, similar to the high velocity gradient that can be found across a shock wave. The following test was suggested to us by B. Wieneke (private communication): the displacement field is a sudden step from a $U = 0.25$ to a $U = 0.35$ pixel horizontal translation. Figure 5 compares the average result found for U by FOLKI-PIV with $R = 15$ to the ground truth. For a given window radius R, we determine the effective spatial resolution by measuring the width over which the PIV algorithm integrates the sharp edge. In practice, this width can be recovered as the inverse of the slope of the estimated $U(X)$ at the center of the step. Figure 6, in which this quantity is plotted against R, shows that FOLKI-PIV integrates the step exactly like the $[-R, R]$ moving average, which has a $2R$ effective spatial resolution.

An experimentalist seeking a better spatial resolution will be tempted to use smaller window sizes, but the drawback is an increased measurement uncertainty. Previous studies have shown that this resolution versus uncertainty trade-off also depends on the quality of the images (Raffel et al. 2007, pp. 174–176). To show how FOLKI-PIV behaves in that respect, we have considered three test cases keeping the same sudden step displacement, each of them characterized with a different type of image degradation or added noise compared to the above ideal situation. For each of them, we compute the rms displacement error U_{RMS} of both FOLKI-PIV and CPIV, for various IW sizes.

First, we have added an out-of-plane displacement $W = 0.5$ pixels so that some particles are lost (case I). W is to be compared to the Gaussian light sheet width $l_w = 2$

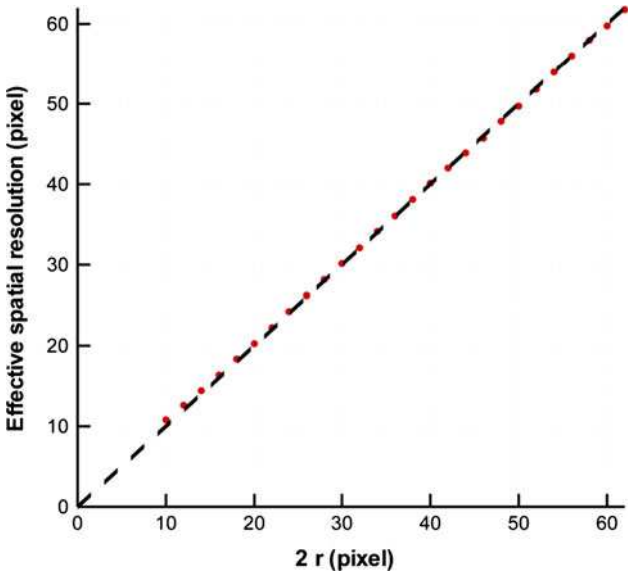


Fig. 6 Effective spatial resolution of FOLKI-PIV as a function of the total IW width $2r$ (*red symbols*) computed from the sharp horizontal step test case, compared with the effective spatial resolution of the $[-R, R]$ sliding average (*dashed black curve*)

pixels. Second, we have added to the images a Gaussian CCD noise with 20 gray level standard deviation, while reducing the 8 bit dynamic range down to 7 bit, thus reducing the signal to noise ratio (case II). Finally, using the same CCD noise as in case II, we have reduced the seeding density from $N_d = 0.02$ down to $N_d = 0.005$ (case III). In Fig. 7, FOLKI-PIV's rms displacement error U_{RMS} is compared to CPIV's for different IW sizes. For all these

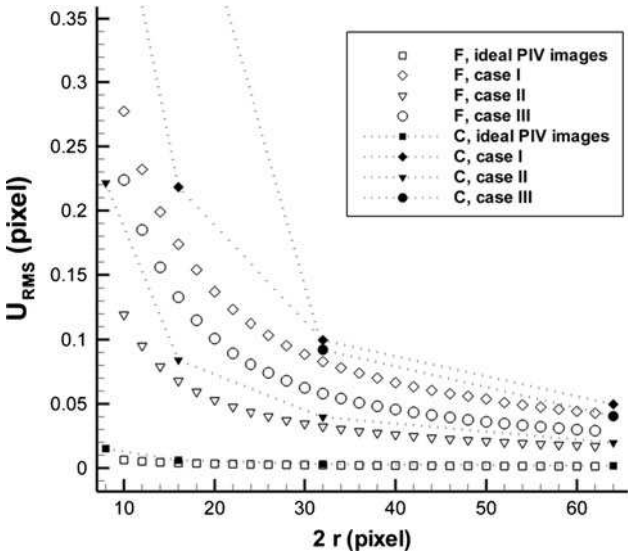


Fig. 7 Rms displacement error U_{RMS} against IW size $2R$ for three test cases of sharp horizontal step displacement (see the text for their exact characteristics). FOLKI-PIV (F) and CPIV (C) results in open and filled symbols, respectively. For clarity, CPIV results are linked by *dotted lines*

cases, FOLKI-PIV and CPIV have a similar behavior, the rms error rising as the IW size decreases. But remarkably, it turns out that FOLKI-PIV performs equally or better than CPIV depending on the cases, whereas it does not involve any data post-processing between two successive iterations, as CPIV does. This is especially true for small windows sizes.

The results presented in this section show a reassuring behavior of FOLKI-PIV with respect to noise and resolution, in other terms FOLKI-PIV does not sacrifice measurement accuracy for speed.

6 Performance assessment: real data

In the following, we provide results on two experimental PIV datasets using FOLKI-PIV with the improvements described in previous sections. The main purposes of these tests are to refine the assessment of FOLKI-PIV's rms displacement error for difficult experimental conditions, as well as its peak-locking bias error, to show how FOLKI-PIV deals with solid boundaries and illustrates the advantages of density for result sampling.

6.1 Case A of the second PIV challenge

Our first real dataset is case A of the second PIV challenge (Stanislas et al. 2005). In this experiment, a round turbulent jet is imaged at a distance from its exhaust large enough for the flow to be self-similar there. This way, a quantitative performance assessment is made possible, by comparison with canonical self-similar turbulent jets, even though the data are extracted from a real experiment with typically encountered difficulties. The sample retained in the 2003 PIV challenge consists of 100 images of resolution 992 × 1,004 pixels, in which the particles images have a diameter estimated to 1 pixel. As described by Stanislas et al. (2005), the main goals of proposing this test case were to assess the ability of PIV algorithms to deal with strongly turbulent flows, as well as to evaluate their rms displacement error and their peak-locking error.

6.1.1 Choice of the vector computation parameters

In the following, as done in the previous paragraph, we compare the results obtained by both FOLKI-PIV and CPIV on these data. For the former, we use $J = 3$ levels, $N = 7$ iterations and $R = 15$ IWs, and for the latter, a multi-pass iterative scheme composed of 2 passes with 64 × 64 IWs followed by 4 passes with 32 × 32 IWs (50% overlap at each pass), with the same intermediate spurious vector rejection and data processing between passes as described in Sect. 5. After the last pass of CPIV, this post-processing is applied once again to yield the final vector fields. For both

algorithms, local zero normalization of the image intensities is applied, and a bilinear interpolator is used, instead of the more accurate interpolators used in the previous paragraph. It is indeed known that for data having a low signal to noise ratio, bilinear interpolation yields optimal performance, especially regarding peak-locking (Lecordier and Trinité 2006; Yamamoto and Uemura 2009). Our tests (not shown here for conciseness) confirmed this conclusion for the present dataset.

Concerning FOLKI-PIV's settings, the choice $N = 7$ stems from a convergence study performed on two image pairs, labeled 1 and 5 in the dataset. We processed these images with N increasing from 1 to 20, keeping all other parameters constant. For each value of $N \geq 2$, we here compute at each image point (X, Y) the norm $\Delta(X, Y)$ of the difference between the displacement vectors found at iterations N and $N - 1$. The optimal choice for the iteration number then results from the fact that for $N \geq 7$, Δ is observed to be inferior to 0.1 pixel in the whole field, this limit being a traditional estimate of PIV accuracy on experimental images. Figure 8 below yields an illustration of this convergence, by showing the evolution with N of Δ, together with the correlation score S_{ZNCC} at $(X, Y) = (497, 502)$. As can be observed, at this location, an accurate result is reached for N well below 7. More importantly, the correlation score convergence is seen to be similar to that of Δ. This shows that the user may tune the parameters of FOLKI-PIV by a visual diagnosis of either the velocity fields or the correlation score S_{ZNCC}, by monitoring when the fields cease to change as N is increased. In practice, this quick preliminary convergence study should be led bearing the following typical values in mind: good quality images usually yield converged results for N as low as 3 (as was the case for the synthetic images of Sect. 5), whereas experimental images of poor quality may require values of N reaching up to 10.

6.1.2 Comparative results

To compare FOLKI-PIV's and CPIV's results, we downsample the dense vector fields of FOLKI-PIV every 15 × 15 pixels. Upon applying for both datasets the same a posteriori detection of outliers as in Stanislas et al. (2005), which amounts to keeping vectors lying in the inside of a given ellipse in a (U, V) scatter plot, we found 5 outliers with FOLKI-PIV, over a total of 442,200 vectors. In comparison, 87 vectors were found outside of the ellipse with CPIV, over a total of 390,600. It is worth noticing that FOLKI-PIV yields a remarkably small number of outliers despite the fact that it does not include any spurious vector rejection step between iterations—whereas CPIV does.

Results for the peak-locking bias are shown in Figs. 9 and 10, where the global histogram for U and the histogram for the fractional part of U are plotted, respectively. These curves show that for images with particles of small size, FOLKI-PIV displays a level of peak-locking bias comparable with that of CPIV, with a slightly higher level for even values of the displacement. To date, we have no explanation for this difference, as both algorithms use a symmetrical window shift before each iteration or pass. Overall, both figures thus show that, even though it does not involve any interpolation of the correlation peak, FOLKI-PIV performs comparably to algorithms based on multi-pass schemes involving a three-point Gaussian

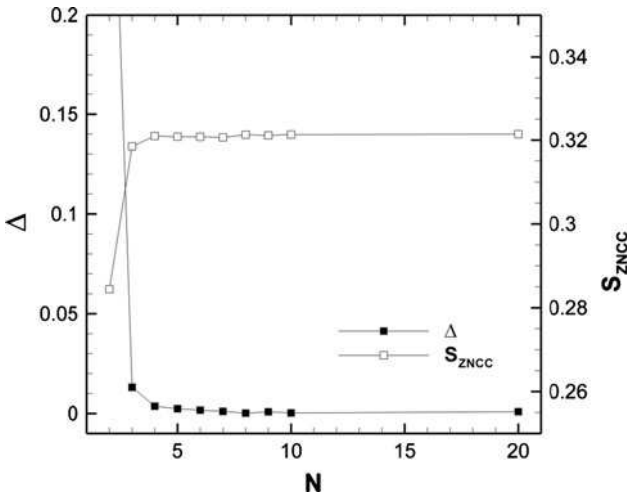


Fig. 8 Convergence with N of the displacement found by FOLKI-PIV at $X = 497$, $Y = 502$, image 1 of PIV challenge 2003 A's dataset. Plotted quantities are the correlation score S_{ZNCC} and the norm $\Delta(X, Y)$ of the difference between the displacement vector at iterations N and $N - 1$

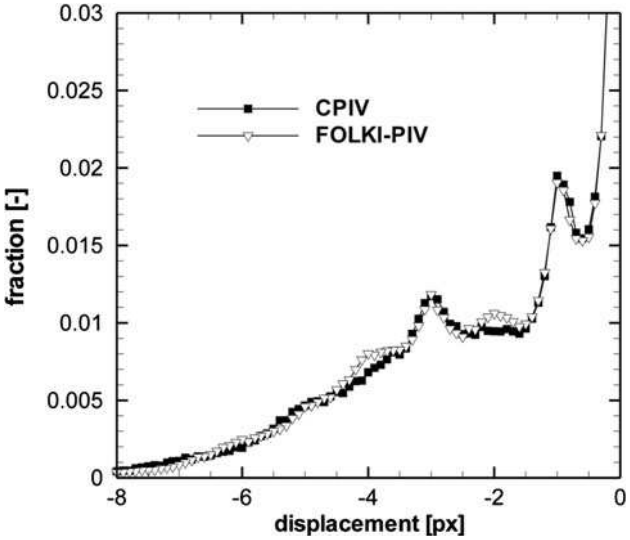


Fig. 9 Histograms of the U displacement

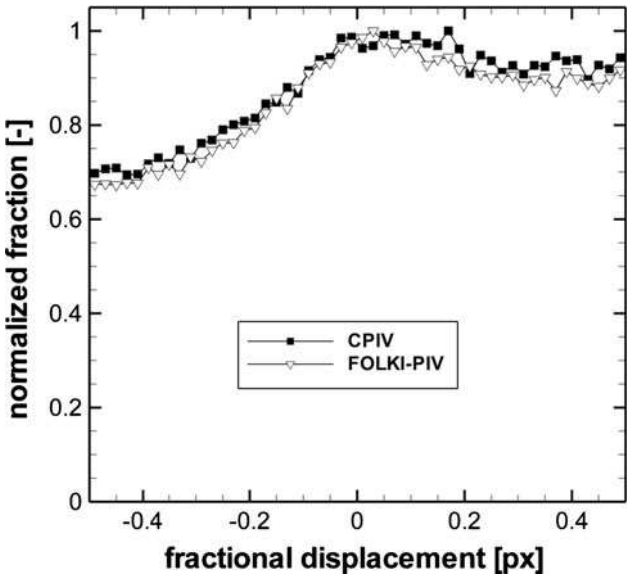


Fig. 10 Fractional part of the U displacement

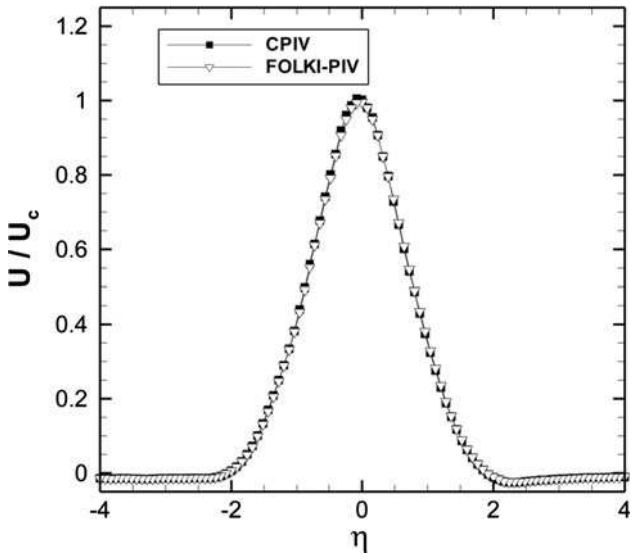


Fig. 11 Normalized mean horizontal displacement U/U_c as a function of the self-similar variable η

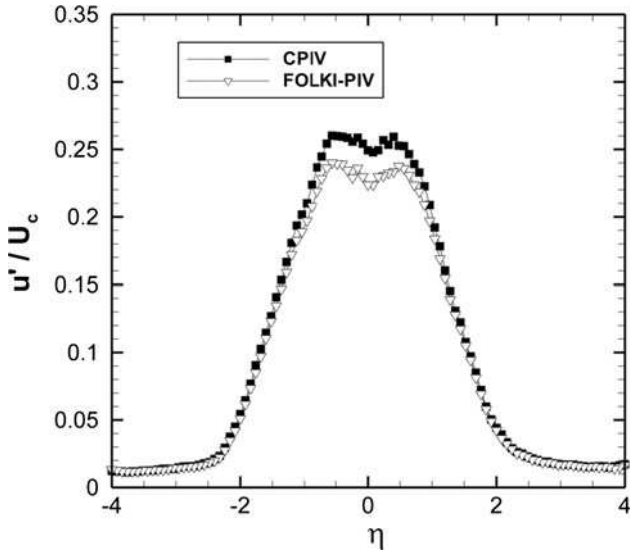


Fig. 12 Normalized rms horizontal displacement u'/U_c as a function of the self-similar variable η

sub-pixel interpolation (Westerweel 2000a, b; Westerweel et al. 1997).

Still following Stanislas et al. (2005), we now represent the mean and rms axial velocities U and u' as single one-dimensional profiles. This can be achieved by taking advantage of the jet self-similarity: first, the ensemble averaged axial velocity U must be fitted to

$$U(x-x_0, y-y_0) = U_c(x)\exp\left(-\eta^2\right) = U_c(x)\exp\left(-\frac{y^2}{l(x)^2}\right) \quad (20)$$

with

$$U_c(x) = \frac{A}{x-x_0} \quad \text{and} \quad l(x) = B(x-x_0), \quad (21)$$

using parameters B (jet spreading rate), x_0 and y_0 (jet virtual origin), and A. Then, values of U/U_c and u'/U_c from each profile at a given x may be gathered so as to represent one single profile depending on η. This is done in Figs. 11 and 12, respectively.

Concerning the mean velocity, an excellent agreement is found between CPIV and FOLKI-PIV. Both curves collapse almost perfectly, both in the jet itself and in the outer flow. The same remark also holds for the rms u' in the outer flow and in the jet shear layers (see Fig. 12). In the middle of the jet, for $-1.5 \le \eta \le 1.5$, discrepancies between both softwares are however observed, which are maximal in the plateau region, for $-0.6 \le \eta \le 0.6$. The average plateau value found by CPIV is of roughly 0.255, while that found by FOLKI-PIV is of roughly 0.235. Note that discrepancies of the same order of magnitude were found between algorithms tested during the second PIV challenge. To us, this is not surprising, since the present test case corresponds to difficult experimental conditions (seeding inhomogeneities in the individual images, differences in light intensity between two images in a pair, small particles). In practice, we find quite low values of the correlation score S_{ZNCC} close to the jet centerline, especially near the right edge of the images (typically, between 0.1 and 0.3). This may explain why algorithms may perform differently there.

As a last quality assessment from this test case, we consider the level of u'/U_c obtained in the outer flow region, where turbulence should progressively return to zero when $|\eta|$ increases. Similarly to the analysis performed

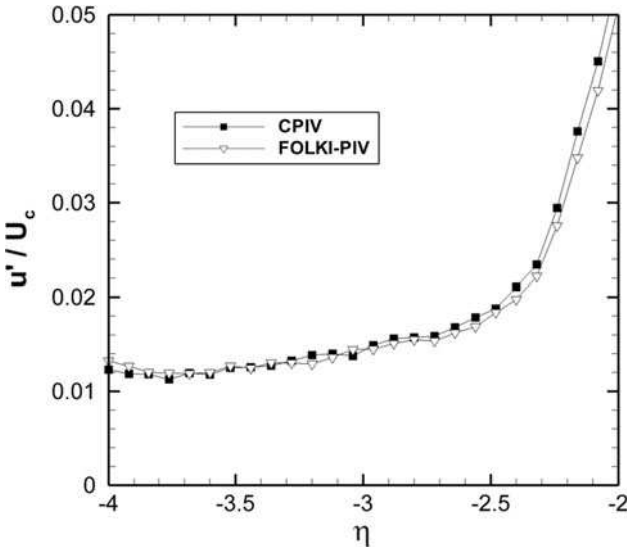


Fig. 13 Normalized rms horizontal displacement u'/U_c as a function of η, close-up

Table 2 Per image average computational time (in s), PIV challenge 2003 A dataset

FOLKI-PIV	CPIV standard	CPIV-high accuracy
0.2	9.4	19.4

All computations were performed on a PC with an Intel Core2 CPU at 3.16 *GHz*, 3 *Go* RAM and a NVIDIA Tesla C1060 GPU. See the text for more details on the settings used for each algorithm

by Stanislas et al. (2005), we plot in Fig. 13 a close-up of u'/U_c in the region $-4 \leq \eta \leq -2$. For $-4 \leq \eta \leq -3.5$, both codes reach approximately the same asymptotic limit of $u'/U_c \approx 0.012-0.013$. There as well, it appears that FOLKI-PIV performs nearly identically to a state-of-the-art PIV software using a FFT-based cross-correlation and a three-point Gaussian sub-pixel interpolation (Westerweel 2000a, b; Westerweel et al. 1997).

Finally, we compare in Table 2 the average time of computation of one vector field of this dataset for each algorithm. For CPIV, we also performed a simpler computation, composed of 1 pass with 64 × 64 IWs followed by 2 passes with 32 × 32 IWs, still with 50% overlap. Such a setting is usually a standard for a majority of PIV images, leading to a high enough accuracy. In Table 2, it is denoted by "CPIV-standard", as opposed to the setting used for the above comparisons, which is denoted by "CPIV-high accuracy". All these computations were performed on a machine equipped with an Intel Core2 CPU at 3.16 *GHz*, 3 *Go* RAM and a NVIDIA Tesla C1060 GPU.

As shown by Table 2, the high level of optimization of FOLKI together with the GPU implementation leads to computational gains ranging from 50 to 100 compared to a state-of-the-art traditional PIV approach, with a similar accuracy of the results.

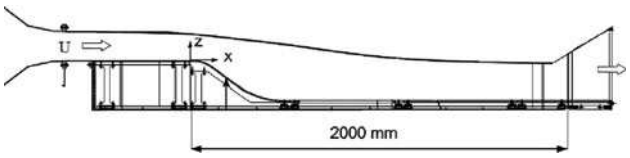


Fig. 14 Principle sketch of the ONERA S19Ch wind-tunnel

6.2 Massive flow separation over a rounded ramp (ONERA data, Gardarin et al. 2008)

The data considered in this paragraph are extracted a from High-Speed PIV (HS-PIV) test campaign on massive boundary layer separation and its control Gardarin et al. (2008), performed in the S19Ch wind-tunnel of ONERA Meudon center (see Fig. 14). In this closed-loop wind-tunnel, the test section is 300 mm wide and 2 m long and allows convenient optical access thanks to transparent lateral walls. As seen in Fig. 14, the flow enters the test section with a horizontal velocity $U = 30$ m s^{-1}. Separation then occurs due to the strong adverse pressure gradient created by the rounded ramp. While the purpose of the global test campaign has been to evaluate the efficiency of several control devices [in particular mechanical and fluidic vortex generators, Gardarin et al. (2008)], we here present the reference case, where the unforced separation occurs.

Figure 15 shows a sample image obtained with a HS-PIV system composed of a Litron LDY 303HE25 Nd:YLF laser and a Phantom V12.1 camera (1,280 × 800 pixels CCD). The generated laser sheet is 1 mm thick, and the flow is seeded using DiEthylHexyl Sebacate (DEHS) droplets. The repetition rate of the laser is set to 3 kHz, and the time interval between two images in a pair is 60 μs. The zone of interest is a rectangular field located near the

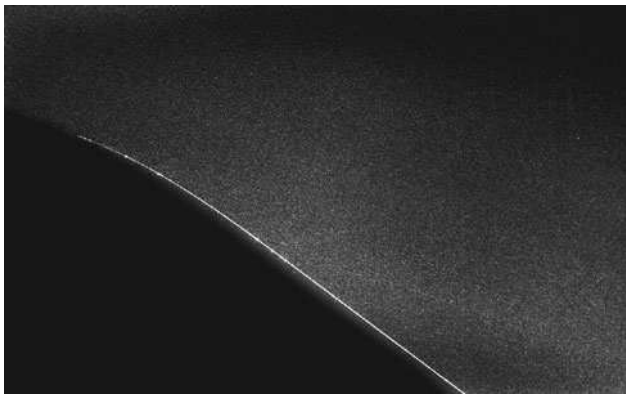


Fig. 15 PIV image sample of ONERA S19Ch HS-PIV experimental dataset

beginning of the rounded ramp, where boundary layer separation occurs. As mentioned above, this test case is specially interesting in that it combines several common experimental difficulties: a solid wall (the rounded ramp) is included inside the field of view so that a light reflection occurs on it, and the light intensity is not homogeneous in the bulk of the image, with in particular irregularities in the light sheet.

We have processed this PIV image pair with FOLKI-PIV using mean and standard deviation local normalization, and a mask following the contour of the rounded ramp. We used the following parameters: $R = 16$ IWs, $N = 6$ iterations, $J = 3$ pyramid levels. The value of N was chosen after a qualitative convergence study in the same way as described in Sect. 6.1.1. Results are shown in Figs. 16 and 17, which represent the obtained velocity field, together with vorticity iso-contours post-processed from this velocity field. To compute the velocity gradients involved in the vorticity, we used a 5×5 Gaussian filter before applying a traditional second-order finite difference scheme. Note that in both figures, the velocity field has been downsampled, respectively, to every 32×32 and 6×6 pixels.

Figure 16 shows that despite the difficult configuration, the flow physics is fully retrieved, since the boundary layer separation is precisely captured (here it is visible at around $x = -0.055$), as well as the various vortical structures which develop on the downstream mixing layer. Figure 17 confirms that FOLKI-PIV and its improvements described in Sect. 4 yield physically sound results even close to the wall, as seen from the various recirculation vortices which are captured until the mask edge. In this respect, it is important to emphasize that the dense result provided by the improved FOLKI-PIV is a real advantage, since it allows to sample the vector field even very close to obstacles.

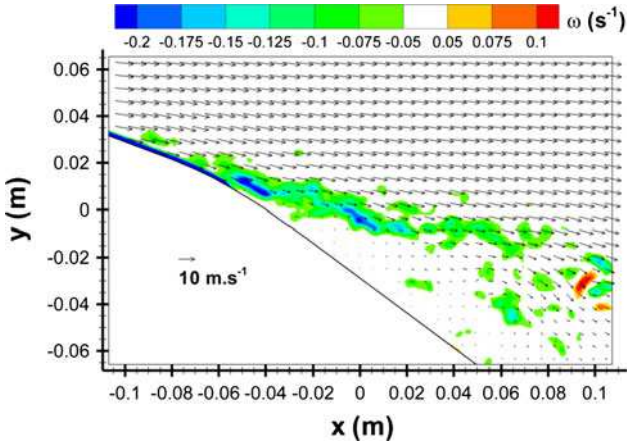


Fig. 16 Instantaneous velocity field obtained with FOLKI-PIV, together with iso-contours of vorticity post-processed from the velocity field by using a 5×5 pixels Gaussian filter. Vectors are displayed every 32×32 pixels

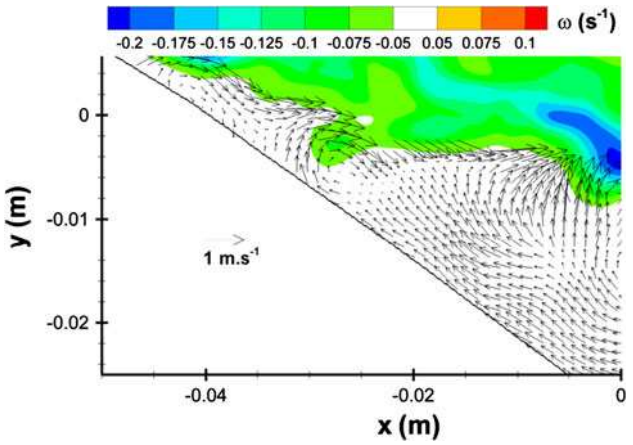


Fig. 17 Instantaneous velocity and vorticity field, close-up on the recirculation region. For clarity, we only plot velocity vectors which have a modulus inferior to 1 m s^{-1}. These vectors are displayed every 6×6 pixels

7 Conclusions

We have presented FOLKI-PIV an algorithm that performs fast computation of dense two-component (2C) PIV vector fields using Graphics Processing Units. Tested on both synthetic and real images, FOLKI-PIV proved as accurate as a state-of-the-art standard PIV software, while being 50 times faster and allowing a convenient degree of freedom for result sampling due to its dense character.

Our simulation study confirms that the behavior of FOLKI-PIV is that of a moving average filter—with $1/(2R)$ bandwidth, R being the window's radius. Robustness to noise is equal or better than state-of-the-art PIV software, without the need for any spurious vector rejection process between iterations. As a result, FOLKI-PIV appears very simple to tune for the practitioner.

To date, these features of FOLKI-PIV already offer to the experimentalists at ONERA a precious help in the way to conduct their PIV experiments, by allowing faster diagnosis on their experimental settings. Indeed, the accuracy of optical parameter tunings may now in particular be assessed also by investigating the mean and rms of the flows. This is especially precious when dealing with turbulent flows, which require a large number of image pairs for the turbulent statistics to converge. More generally, we believe that such a fast PIV code opens the way to significant changes in the way to perform experiments in fluids with PIV, allowing a much faster flow diagnosis and extending considerably possibilities of trial-and-error in the optimization phasis of experimental campaigns.

The basic rationale of the algorithm is furthermore generic enough to enable various extensions. On-going work considers Stereo PIV [see Leclaire et al. (2010)] and 3D velocity estimation for tomographic PIV, both settings which can be cast into the FOLKI-PIV scheme, so as to derive fast GPU implementation in the near future.

Acknowledgments Benoît Gardarin, Laurent Jacquin and Gilles Losfeld are gratefully acknowledged for providing their experimental TR-PIV dataset. Moreover the authors salute the helpful comments of the referees.

References

Baker S, Matthews I (2004) Lucas-Kanade 20 years on: a unifying framework. Int J Comput Vis 56(3):221–255

Bergen JR, Anandan P, Hanna KJ, Hingorani R (1992) Hierarchical model-based motion estimation. In: ECCV 92, Proceedings of the second European conference on computer vision, pp 237–252

Bouguet JY (2000) Pyramidal implementation of the Lucas-Kanade feature tracker: description of the algorithm. Tech. rep., Open CV Documentation

Burt P, Adelson E (1983) The Laplacian image pyramid as a compact image code. IEEE Trans Commun 31:532–540

Champagnat F, Plyer A, Le Besnerais G, Leclaire B, Le Sant Y (2009) How to calculate dense PIV vector fields at video rates. In: Proceedings of 8th international symposium on particle image velocimetry—PIV09, Melbourne

Corpetti T, Heitz D, Arroyo G, Mémin E, Santa-Cruz A (2006) Fluid experimental flow estimation based on an optical-flow scheme. Exp Fluids 40(1):80–97

Gardarin B, Jacquin L, Geffroy P (2008) Flow separation control with vortex generators. In: AIAA 4th flow control conference, Seattle, WA, 23–26 June

Iriarte Munoz J, Dellavale D, Sonnaillon M, Bonetto F (2009) Real-time particle image velocimetry based on FPGA technology. In: Programmable logic, 2009. SPL. 5th Southern conference on, pp 147–152

Keller Y, Averbuch A (2004) Fast motion estimation using bi-directional gradient methods. IEEE Trans Image Process 13(8):1042–1054

Le Besnerais G, Champagnat F (2005) Dense optical flow by iterative local window registration. In: ICIP'05, Proceedings of the IEEE international conference on image processing, vol I, pp 137–140

Leclaire B, Le Sant Y, Davoust S, Le Besnerais G, Champagnat F (2010) FOLKI-SPIV: a new, ultra-fast approach for stereo PIV. submitted to Experiments in Fluids

Lecordier B, Trinite M (2003) Advanced PIV algorithms with image distortion validation and comparison using synthetic images of turbulent flow. In: Stanislas M, Westerweel J, Kompenhans J (eds) Particle image velocimetry: recent improvements. Proceedings of the EUROPIV 2 workshop, Springer, pp 115–132

Lecordier B, Trinité M (2006) Accuracy assessment of image interpolation schemes for PIV from real images of particle. In: Proc. 13th int. symp. on appl. of laser techn. to fluid mechanics, Lisbon, Portugal

Lecordier B, Westerweel J (2003) The EUROPIV Synthetic Image Generator (SIG). In: Stanislas M, Westerweel J, Kompenhans J (eds) Particle image velocimetry: recent improvements. EUROPIV 2 workshop, Springer

Miozzi M (2004) Particle image velocimetry using feature tracking and Delaunay Tessellation. In: Proceedings of the XII international symposium on application of laser technique to fluid mechanics, Lisbon

Pan B, Xie H, Guo Z, Hua T (2007) Full-field strain measurement using a two-dimensional Savitzky-Golay digital differentiator in digital image correlation. Opt Eng 46(3):1–10

Raffel M, Willert C, Wereley C, Kompenhans J (2007) Particle image velocimetry. A practical guide, 2nd edn. Springer, Heidelberg

Ruhnau P, Kohlberger T, Schnörr C, Nobach H (2005) Variational optical flow estimation for particle image velocimetry. Exp Fluids 38:21–32

Ruijters D, ter Haar Romeny BM, Suetens P (2008) Efficient GPU-based texture interpolation using uniform B-splines. J Graphics GPU Game Tools 13(4):61–69

Scarano F, Riethmuller M (2000) Advances in iterative multigrid PIV image processing. Exp Fluids 29:51–60

Schiwietz T, Westermann R (2004) GPU-PIV. In: Girod B, Magnor MA, Seidel HP (eds) Proceedings of the vision, modeling, and visualization conference, Aka GmbH, Stanford, CA, pp 151–158

Stanislas M, Okamoto K, Kähler CJ, Westerweel J (2005) Main results of the second international PIV challenge. Exp Fluids 39:170–191

Stanislas M, Okamoto K, Käler CJ, Westerweel J, Scarano F (2008) Main results of the third international PIV challenge. Exp Fluids 45:27–71

Venugopal V, Patterson C, Shinpaugh K (2009) Accelerating particle image velocimetry using hybrid architectures. In: Proceedings of symposium on application accelerators in high performance computing (SAAHPC'09), Urbana, Illinois

Westerweel J (1993) Digital particle image velocimetry—theory and application. PhD thesis, University Press, Delft

Westerweel J (2000a) Effect of sensor geometry on the performance of PIV interrogation. Laser techniques applied to fluid mechanics. Springer, Berlin, pp 37–55

Westerweel J (2000b) Theoretical analysis of the measurement precision in particle image velocimetry. Exp Fluids 29:3–12

Westerweel J, Dabiri D, Gharib M (1997) The effect of a discrete window offset on the accuracy of cross-correlation analysis of digital PIV recordings. Exp Fluids 23(1):20–28

Yamamoto Y, Uemura T (2009) Robust particle image velocimetry using gradient method with upstream difference and downstream difference. Exp Fluids 46(4):659–670

Yu H, Leeser M, Tadmor G, Siegel S (2006) Real-time particle image velocimetry for feedback loops using FPGA implementation. J Aerosp Comput Inf Commun 3(2):52–62

Zhao W, Sawhney H (2002) Is super-resolution with optical flow possible? In: ECCV02, Proceedings of the European conference on computer vision, pp 153–162

A.2 Reconstruction Tomographique

Travaux sur la **tomographie** : Les développements en reconstruction de type BoS ont fait l'objet d'un article de conférence nationale [Todoroff et al., 2012b] et un article de conférence à comité de lecture [Todoroff et al., 2012a].

A.3 Detection

Notre activité de détection sur vidéo aérienne basée sur une décomposition du champ de mouvement à été présenté dans un article de conférence avec actes [Pollard et al., 2009].

A.4 Perception et Robotique

Nous avons initié dans nos travaux un ensemble d'étude et développement autour de la problématique de la perception embarquée pour les petits drones. Dans ce contexte, nous avons publié dans un journal à comité de lecture un tour d'horizon sur la complexité de ces systèmes [Sanfourche et al., 2012].

Nous avons de plus étudié une méthode d'odométrie basée features pour un capteur de profondeur possédant peu de textures, utilisant une stratégie massivement parallèle pour évaluer un grand nombre d'hypothèses [Israel and Plyer, 2011, Israël and Plyer, 2013].

Bibliographie

[Champagnat et al., 2011] Champagnat, F., Plyer, A., Le Besnerais, G., Leclaire, B., Davoust, S., and Le Sant, Y. (2011). Fast and accurate piv computation using highly parallel iterative correlation maximization. *Experiments in Fluids*, 50 :1169–1182. 10.1007/s00348-011-1054-x.

[Champagnat et al., 2009] Champagnat, F., Plyer, A., Le Besnerais, G., Leclaire, B., and Le Sant, Y. (2009). How to calculate dense piv vector fields at video rates. In *Proceedings of 8th International Symposium on Particle Image Velocimetry-PIV09*.

[Israel and Plyer, 2011] Israel, J. and Plyer, A. (2011). A brute force approach to depth camera odometry. In *Computer Vision Workshops (ICCV Workshops), 2011 IEEE International Conference on*, pages 1141–1146. IEEE.

[Israël and Plyer, 2013] Israël, J. and Plyer, A. (2013). A brute force approach to depth camera odometry. In Fossati, A., Gall, J., Grabner, H., Ren, X., and Konolige, K., editors, *Consumer Depth Cameras for Computer Vision*, Advances in Computer Vision and Pattern Recognition, pages 49–60. Springer London.

[Le Besnerais et al., 2009] Le Besnerais, G., Champagnat, F., Plyer, A., Fezzani, R., Leclaire, B., and Le Sant, Y. (2009). Advanced processing methods for image-based displacement field measurement. *Aerospace Lab Journal*, 1.

[Le Sant et al., 2010] Le Sant, Y., Leclaire, B., Davoust, S., Champagnat, F., Plyer, A., and Le Besnerais, G. (2010). FOLKI-PIV : une nouvelle approche ultra-rapide pour le calcul de champs PIV Stéréo. In *CFTL, Congrès Francophone de Techniques Laser*.

[Leclaire et al., 2010] Leclaire, B., Le Sant, Y., Davoust, S., Le Besnerais, G., and Champagnat, F. (2010). Folki-spiv : a new, ultrafast approach for stereo piv. *submitted to Experiments in Fluids*.

[Plyer et al., 2009] Plyer, A., Le Besnerais, G., and Champagnat, F. (2009). Folki-gpu : A powerful and versatile cuda code for real-time optical flow computation. In *GPU Technology Conference*.

[Pollard et al., 2009] Pollard, E., Plyer, A., Pannetier, B., Champagnat, F., and Le Besnerais, G. (2009). Gm-phd filters for multi-object tracking in uncalibrated aerial videos. In *Information Fusion, 2009. FUSION'09. 12th International Conference on*, pages 1171–1178. IEEE.

[Sanfourche et al., 2012] Sanfourche, M., Delaune, J., Le Besnerais, G., de Plinval, H., Israel, J., Cornic, P., Treil, A., Watanabe, Y., and Plyer, A. (2012). Perception for uav : Vision-based navigation and environment modeling. *Aerospace Lab Journal*, 4.

[Todoroff et al., 2012a] Todoroff, V., Plyer, A., Le Besnerais, G., Champagnat, F., Donjat, D., Micheli, F., and Millan, P. (2012a). 3d reconstruction of the density field of a jet using synthetic bos images. In *15th International Symposium on Flow Visualization*, Minsk, Belarus.

[Todoroff et al., 2012b] Todoroff, V., Plyer, A., Le Besnerais, G., Champagnat, F., Donjat, D., Micheli, F., and Millan, P. (2012b). Reconstruction tridimensionnelle du champ de masse volumique d'un jet à partir d'images bos simulées. In *13ième Congrès Francophone de Techniques Laser, CFTL 2012*, Rouen, France.

B Tables

B.1 Chapitre Estimation de Mouvements

Table B.1 – Table Kitti au 24/11/2012

Rank	Method	Setting	Out-Noc	Out-All	Avg-Noc	Avg-All	Density	Runtime
1	PCBP-Flow	ms	5.45 %	10.15 %	1.4 px	2.8 px	100.00 %	3 min
2	fSGM		11.03 %	22.90 %	3.2 px	12.2 px	100.00 %	60 s
3	TGV2CENSUS		11.14 %	18.42 %	2.9 px	6.6 px	100.00 %	4 s
4	GC-BM-Bino	ms	18.93 %	29.37 %	5.0 px	12.0 px	83.73 %	1.3 s
5	GC-BM-Mono	ms	19.49 %	29.88 %	5.0 px	12.1 px	84.33 %	1.3 s
6	HS		19.92 %	28.86 %	5.8 px	11.7 px	100.00 %	3 min
7	RSRS-Flow		20.74 %	29.68 %	6.2 px	12.1 px	100.00 %	4 min
8	ALD		21.35 %	30.65 %	10.9 px	16.0 px	100.00 %	110 s
9	LDOF		21.86 %	31.31 %	5.5 px	12.4 px	100.00 %	1 min
10	C+NL		24.64 %	33.35 %	9.0 px	16.4 px	100.00 %	3 min
11	HMM		24.76 %	34.16 %	7.2 px	15.0 px	100.00 %	10 min
12	DB-TV-L1		30.75 %	39.13 %	7.8 px	14.6 px	100.00 %	16 s
13	GCSF		33.23 %	41.74 %	7.0 px	15.3 px	48.27 %	2.4 s
14	HAOF		35.76 %	43.36 %	11.1 px	18.2 px	100.00 %	16.2 s
15	PolyExpand		47.54 %	53.95 %	17.2 px	25.2 px	100.00 %	1 s
16	OCV-BM		63.46 %	68.16 %	24.4 px	33.3 px	100.00 %	1.5 min
17	Pyramid-LK		65.74 %	70.09 %	21.7 px	33.1 px	99.90 %	1.5 min

B.2 Chapitre Super-résolution

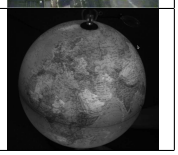
Nom	Origine	Type	Résolution	Exemple
Voilier	OTAN (SET-132)[GÖHLER,]	Infra-rouge	384×288	
Camion-Mires	OTAN (SET140)	Bolométrique	256×120	
Tank	OTAN (SET140)	IR MW	640×512	
Boys-Band	OTAN (SET140)	Bolométrique	261×205	
Aérien	SIDM	RGB	352×288	
Aérien2	SIDM	RGB	352×288	
MapMonde	Onera	niveau de gris	384×384	

Table B.2 – Origine et type des séquences utilisées pour les illustrations de ce chapitre.

Figure	Nom	algo	nb BR	fSR	recalage	sigBR	rég	type	seuil
4.10	Voilier	SMSK	21	5.5	FOLKI	0.33	0.21	cov lin	-
4.14(a),4.15(a)	Voilier	SMSK	21	7	FOLKI	0.4	0.2	cov lin	-
4.14(b)	Voilier	SM+deconvL2	21	7	FOLKI	0.4	0.	gradient	-
4.14(c),4.15(a)	Voilier	SR L2	21	7	FOLKI	0.4	0.01	gradient	-
4.15(c)	Voilier	SR L1-L2	21	7	FOLKI	0.4	0.5	Huber/grad	2.0
4.16(a)	Voilier	SR L2	21	7	FOLKI	0.33	0.5	Gradient	-
4.16(b)	Voilier	SR NQ	21	7	FOLKI	0.33	0.5	Huber/grad	0.42
4.16(c)	Voilier	SR NQ	21	7	FOLKI	0.33	0.5	Huber/grad	2.0
4.17(d)(e)	Aerien2	SMSK	17	5	FOLKI	0.4	0.29	cov lin	-
4.18(a)(b)	Voilier	SR NQ Rob	21	7	FOLKI	0.4	0.5	Gradient	-

Table B.3 – Paramètres utilisés dans les figures.

	iter.	SM+SK	SR-φ	SR-ϕ-ψ	WarpHR
GoP de 11 images (OF = 17.2ms)					(OFar = 34.5ms)
sr x3	10	0.24	3.5	4.2	36
	20		6.8	8.3	72
	50		19	21	179
	100		33	42	362
sr x6	10	0.65	9.4	9.7	77
	20		16	19	154
	50		41	48	403
	100		83	96	774
GoP de 21 images (OF = 34.1ms)					(OFar = 68ms)
sr x3	10	0.36	3.5	4.8	67
	20		6.8	9.6	133
	50		17	24	335
	100		34	48	671
sr x6	10	0.78	8.35	11	142
	20		17	21	284
	50		41	53	712
	100		82	107	1424

Table B.4 – Tableaux des temps de calculs des algorithmes de super-résolution calcul du Flot Optique (FO) et des différents algorithmes de super-résolutions présenté dans le chapitre, pour une séquence de résolution $128x60$ (les temps sont donné en mili-seconde).

	iter.	SM+SK	SR-φ	SR-ϕ-ψ	WarpHR
GoP de 11 images (OF = 20.5ms)					(OFar = 42ms)
sr x3	10	0.58	8.3	11	73
	20		16	22	147
	50		40	57	368
	100		80	113	734
sr x6	10	2.24	27	32	230
	20		54	65	464
	50		133	160	1152
	100		264	320	2310
GoP de 21 images (OF = 41ms)					(OFar = 83ms)
sr x3	10	0.91	8.65	14	135
	20		16	28	271
	50		40	70	676
	100		81	140	1361
sr x6	10	2.63	28	36	424
	20		53	73	850
	50		133	183	2124
	100		265	366	4248

Table B.5 – Tableaux des temps de calculs des algorithmes de super-résolution calcul du Flot Optique (FO) et des différents algorithmes de super-résolutions présenté dans le chapitre, pour une séquence de résolution $256x120$ (les temps sont donné en mili-seconde).

	iter.	SM+SK	SR-φ	SR-ϕ-ψ	WarpHR
GoP de 11 images (OF = 26ms)					(OFar = 52ms)
sr x3	10	1.9	26	38	212
	20		51	75	424
	50		124	188	1058
	100		247	375	2130
sr x6	10	8.8	97	118	833
	20		191	253	1675
	50		499	606	4215
	100		946	1172	8452
GoP de 21 images (OF = 52ms)					(OFar = 101ms)
sr x3	10	3.0	29	50	398
	20		52	98	798
	50		139	246	2077
	100		248	488	3995
sr x6	10	10	98	138	1567
	20		192	279	3119
	50		493	702	7919
	100		968	1378	15656

Table B.6 – Tableaux des temps de calculs des algorithmes de super-résolution calcul du Flot Optique (FO) et des différents algorithmes de super-résolutions présenté dans le chapitre, pour une séquence de résolution $512x240$ (les temps sont donné en mili-seconde).

	iter.	SM+SK	SR-φ	SR-ϕ-ψ	WarpHR
GoP de 11 images (OF = 36ms)					(OFar = 71ms)
sr x3	10	4.5	55	84	444
	20		106	168	887
	50		261	419	2218
	100		521	839	4439
sr x6	10	20	210	263	1850
	20		414	523	3657
	50		1028	1301	9150
	100		2060	2596	18322
GoP de 21 images (OF = 72ms)					(OFar = 141ms)
sr x3	10	6.6	59	113	814
	20		109	226	1650
	50		265	566	4068
	100		527	1132	8143
sr x6	10	23	214	307	3438
	20		419	614	6739
	50		1039	1531	16856
	100		2077	3058	33710

Table B.7 – Tableaux des temps de calculs des algorithmes de super-résolution calcul du Flot Optique (FO) et des différents algorithmes de super-résolutions présenté dans le chapitre, pour une séquence de résolution $768x360$ (les temps sont donné en mili-seconde).

Bibliographie

[GÖHLER,] GÖHLER, B. Set-132 "imaging ladar technology & algorithms for tactical applications". goehler@fom.fgan.de.

C Produits logiciels

C.1 FolkiGPU et libFolkigpu

FolkiGPU désigne une interface Qt de démonstration d'estimation de mouvements. Une version basique est disponible au téléchargement sur le site Onera (`www.onera.fr/dtim-eng/gpu-for-image`).

Cette interface permet d'évaluer rapidement la qualité de l'estimation de mouvement. Un panneau de controle permet de plus un paramétrage aisé en ligne.


FolkiGpu
video Control
folki
visuParams
stats
pistes
type Pyramide
Haar
type fenetre
Flat
nombre de niveaux
3
nombre d'iterations
2
unRolling Convolution
census
2
rayon de la fenetre
5
max
8
pas
2
min
1
bords
10
talon : 10^(-x)
lminSeuil
6.0
utilisation de strategies mutliGrids
strategie multiGrid
V cycle
flush flow
FolkiGpu
video Control
folki
visuParams
stats
pistes
temps de calcul du flot (sur GPU)
5
ms
temps totale (boucle de l'application)
34
ms
temps de transfert (1 image)
0
ms
norme max du flot
0
pxls
FolkiGpu
video Control
folki
visuParams
stats
pistes
load mask
Load
cam
play
PIV mod
active m
backwar
pause
fli
normalize
stop
quit
save vector
repeat
ONERA
THE FRENCH AEROSPACE LAB

C.2 SRViewer

SRViewer est une interface multi-usage que nous avons développé à l'origine pour évaluer des algorithmes de super-résolution. Celui-ci ayant beaucoup évolué, il nous a aussi permis d'expérimenter un grand nombre de variantes d'estimation de mouvements, par exemple les méthodes régularisées par "variable splitting". Nous avons de plus commencé à y développer les algorithmes de calcul de flot de scène tel que Folkinect, cf. les perspectives dans le chapitre de conclusion.

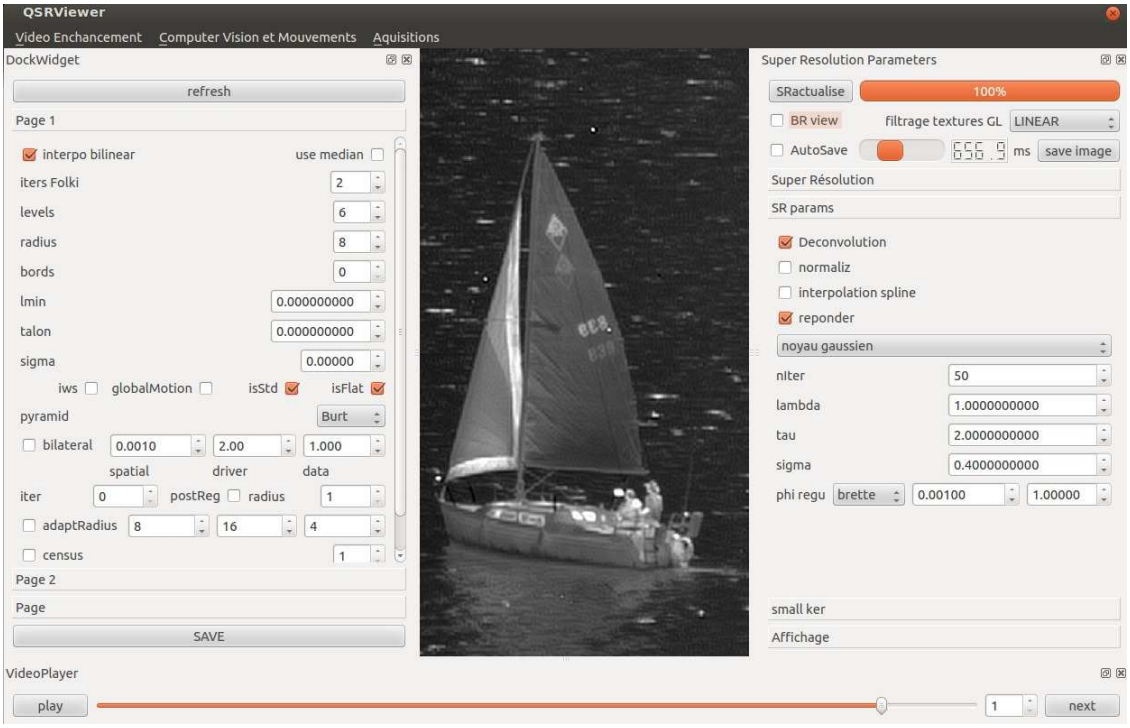


Figure C.2 – Interface SRViewer