Université de Montréal

Étude de contraintes spatiales bas niveau appliquées à la vision par ordinateur

Par
Pierre-Marc Jodoin

Département d'informatique et de recherche opérationnelle
Faculté des arts et des sciences

Thèse présentée à la Faculté des études supérieures
en vue de l'obtention du grade de Philosophiae Doctor
en informatique

Décembre, 2006

Université de Montréal
Faculté des études supérieures

Cette thèse intitulée:

Étude de contraintes spatiales bas niveau appliquées à la vision par ordinateur

présentée par:

Pierre-Marc Jodoin

a été évaluée par un jury composé des personnes suivantes:

Neil Stewart
président-rapporteur

Max Mignotte
directeur de recherche

Jean Meunier
membre du jury

Yvon Voisin
examinateur externe

Jean Meunier
représentant du doyen de la FES

Thèse acceptée le 20 mars 2007

RÉSUMÉ

Le domaine de l'imagerie numérique a connu une évolution constante au cours des trois dernières décennies. Une des raisons de ce dynamisme est la croissance du nombre d'applications requérant le traitement et l'analyse d'images pixélisées. Météorologie, océanographie, physique des particules, surveillance vidéo, imagerie médicale ne sont que quelques uns des domaines ayant recours de plus en plus à l'imagerie numérique. Alors que le traitement d'images porte sur des aspects bas niveau de l'image (débruitage, déconvolution, détection de contours, etc.) la vision par ordinateur porte sur des données à plus haut niveau (flux optique, stéréovision, reconnaissances de formes, etc.). Toutefois, malgré leurs différences, ces deux domaines ont un dénominateur commun : plusieurs problèmes leur étant dévolus trouvent leur solution au coeur de la théorie de l'apprentissage statistique. En effet, de nombreux problèmes fondamentaux en imagerie se réduisent à un problème d'apprentissage statistique. Autrement dit, plusieurs problèmes d'imagerie pouvent être formalisés par un processus d'optimisation cherchant à estimer une fonction «g»qui saurait le mieux satisfaire un critère de qualité «R». Le critère de qualité le plus fréquemment utilisé s'exprime sous la forme d'une *fonction de risque* définit comme l'espérance mathématique d'une *fonction de perte.* La minimisation d'une fonction de risque est un domaine en soi qui, comme le mentionne Vapnik [157], inclus trois problèmes fondamentaux, *i.e.*,

1. la reconnaissance de formes ;
2. la régression ;
3. et l'estimation de densité.

Malheureusement, minimiser une fonction de risque dans le contexte de l'imagerie numérique est souvent un problème difficile, parfois mal conditionné. De plus, de nombreux algorithmes en imagerie se fondent sur des hypothèses simplificatrices pour définir R. Or, ces hypothèses ont pour effet d'induire des imprécisions dans les résultats, imprécisions souvent difficiles à détecter.

L'objectif de cette thèse est de proposer une série de contraintes spatiales à bas niveau (*i.e.* au niveau pixel) afin de résoudre certains problèmes d'optimisation à haut niveau propres à la vision par ordinateur. En particulier, nous adressons les problèmes liés au flux optique, à la détection de mouvement, à la détection d'occlusions ainsi qu'à la segmentation de mouvement.

Le flux optique est un concept mathématique servant à décrire le mouvement apparent des objets dans une séquence vidéo. Parmi les méthodes proposées pour estimer le flux optique figure celle de Lucas et Kanade [22]. Leur méthode est fondée sur une régression par moindres carrés, est facile à implémenter et converge très rapidement. Toutefois, malgré les nombreux avantages qu'apporte cette méthode, elle n'en demeure pas moins sensible au manque de texture ainsi qu'aux valeurs extrêmes (les *outliers*). En réponse à ces problèmes, nous proposons au chapitre 1, une variante de l'algorithme de Lucas-Kanade utilisant deux contraintes à bas niveau. La première contrainte s'exprime sous la forme d'un filtrage de données de type *Best Linear Unbiased Estimate* (BLUE). Cette contrainte sert à régulariser les résultats dans les régions peu texturées et à minimiser l'effet du bruit. La seconde contrainte est fondée sur l'algorithme du mean-shift [37] et a pour but d'éliminer les valeurs extrêmes (les *outliers*) lors de l'estimation du mouvement. L'idée maîtresse de cette seconde contrainte est de légèrement déplacer le voisinage de calculs vers des régions dont le mouvement est susceptible d'être unimodal plutôt que multimodal.

Un deuxième problème que nous adressons est celui de la détection de mouvement. L'objectif de tout algorithme de détection de mouvement est de dissocier les régions mobiles des régions immobiles dans une séquence vidéo. La plupart des méthodes de détection de mouvement aujourd'hui utilisées se fondent sur une approche par soustraction de fond (*background substraction* en anglais). Pour ce faire, les méthodes les plus robustes modélisent chaque pixel du fond par une densité de probabilité apprise sur une série d'images absentes de tout mouvement. Malgré le fait que ces méthodes font preuve d'une grande fiabilité, apprendre des densités de probabilité sur plusieurs images peut être problématique. Cela est particulièrement

vrai lorsqu'aucune image absente de mouvement n'est disponible ou lorsque la mémoire disponible est insuffisante pour stocker les images nécessaires à l'apprentissage. Au chapitre 3, nous nous proposons d'étudier un nouveau paradigme de soustraction probabiliste. Ce paradigme permet d'entraîner des densités de probabilité sur une seule image au lieu d'une série d'images comme c'est généralement le cas. Nous démontrerons comment l'usage de contraintes spatiales locales peut permettre de réduire certaines limitations inhérentes aux méthodes statistiques classiques de détection de mouvement.

De nombreux problèmes en imagerie exigent l'estimation de ce qu'on appelle un *champ d'étiquettes*. Un champ d'étiquettes peut être vue comme une image dont chaque pixel contient une étiquette de classe. Parmi les algorithmes utilisés pour estimer un champ d'étiquettes, certains sont issus du domaine de l'apprentissage statistique. Les plus connus étant le *maximum de vraisemblance* ainsi que le *maximum a posteriori*. Malgré la puissance de ces approches, lorsque les données d'entrée sont bruitées ou encore imprécises, il leur arrive de générer des résultats imprécis. En réponse à ce problème, au chapitre 2, nous proposons une façon d'améliorer la précision d'un champ d'étiquettes. La méthode proposée fusionne des champs d'étiquettes au lieu de fusionner des données brutes comme le font traditionnellement les algorithmes d'apprentissage statistique. L'objectif de cette méthode est de contraindre les régions d'un champ d'étiquettes à épouser la forme des objets présents dans une scène. Pour ce faire, nous proposons deux fonctions d'énergie pouvant être optimisées à l'aide d'un algorithme ICM. La première fonction est une énergie de vraisemblance alors que la seconde est une énergie de vraisemblance combinée à une fonction *a priori*. Cette méthode de fusion s'est avérée être particulièrement efficace pour les applications de segmentation de mouvement ainsi que de détection d'occlusions.

Finalement, au chapitre 4, nous présentons une approche pour accélérer d'un facteur allant de 4 à 200, les calculs inhérents à certains algorithmes d'apprentissage numérique. Cette approche est fondée sur la propriété qu'ont certains algorithmes d'apprentissage à être implémentés sur une architecture parallèle. En

particulier, nous démontrerons comment ces algorithmes peuvent être implémentés sur une gamme de processeurs graphiques (les *GPU*) aujourd'hui disponibles sur la plupart des cartes graphiques vendues sur le marché. Nous nous concentrerons sur les méthodes de segmentation et d'apprentissage markoviens appliquées à la stéréovision, à l'estimation de mouvement, à la segmentation de mouvement ainsi qu'à la segmentation de couleur.

Mots clés: vision par ordinateur, contraintes spatiales, GPU, segmentation, fusion de données, flux optique, occlusion.

ABSTRACT

The field of imagery has been in constant evolution for the past three decades. One of the reasons for this evolution is the ever-growing number of applications requiring the analysis and processing of digital images and videos. Meteorology, oceanography, particle physics, video surveillance, medical imaging, are among the scientific areas taking advantage of digital imaging. Digital imaging is a research area with many branches among which are the fields of image processing and computer vision. Image processing is known to work with algorithms focused on the lower-level aspect of an image (denoising, deconvolution, edge detection, etc.). On the other hand, computer vision is concerned with higher-level image computation (optical flow, stereovision, shape recognition, etc.). Although these two research areas have their own specificities, they nonetheless share a common denominator : many solutions put forward to solve their problems come from Statistical Learning Theory. In other words, many image processing and computer vision problems can be formulated as an optimization problem in which we seek an optimal function "g" according to a quality criterion "R". The most frequently used quality criteria is the so-called *risk function* defined as the mathematical expectation of a *loss function*. The theory behind risk minimization is a research area on its own which encompasses three fundamental problems, *i.e.* [157],

1. pattern recognition;
2. regression estimation;
3. and density estimation.

Unfortunately, minimizing a risk function in the context of digital imaging is rarely an easy task and is sometimes ill-conditioned. Furthermore, many image processing and computer vision algorithms are based on simplistic hypotheses designed to make R easier to develop. Although these hypotheses make it simpler to solve the problem, they also add imprecision in the results, imprecision that is often hard to detect and correct.

The goal of this thesis is to study a series of low-level and spatial constraints to help better regularize higher-level optimization methods of computer vision. More specifically, we will address the problems of optical flow estimation, motion detection, occlusion detection and motion segmentation.

Optical flow is a mathematical concept used to describe the visual movement in a video sequence. Among the many methods used to estimate optical flow is the one proposed by Lucas and Kanade (LK) [22]. Their method is based on a sum of least-squares regression technique that is easy to implement and which quickly converges. Although their method has definite advantages, it is very sensitive to lack of texture and to the presence of outliers. As an answer to these two limitations, we propose in Chapter 1 a slightly modified version of the LK algorithm. Our modification includes two low-level constraints. The first constraint works as a *Best Linear Unbiased Estimate* (BLUE) low-pass filter that helps regularize the flow in noisy areas and in areas with little (or no) texture. The second constraint is based on the mean-shift algorithm [37]. The aim of this second constraint is to eliminate the presence of outliers during the optical flow estimation. The main idea of this constraint is to slightly move the local neighborhood toward regions where motion is more likely to be unimodal than multimodal.

A second problem that we address in this document is motion detection. The goal of motion detection is to segment a video sequence in *mobile* and *static* regions. Most motion detection methods implemented nowadays rely on the background subtraction principle. In this perspective, *robust* methods model the background pixels with a probability density function learned on a series of video images absent of motion. Although these robust methods work well in many real-life scenarios, they nonetheless have some basic limitations. Their most obvious limitations occur when no training frame absent of motion is available or when the available memory is too small to contain the images needed to learn the probability density function parameters to compensate for these limitations. In Chapter 3, we propose a new probabilistic background subtraction paradigm. This paradigm allows us to train the PDFs on only one background frame instead of many frames as is usually the

case. In this chapter, we show how the use of local spatial constraints can help reduce some basic limitations of most statistical background subtraction methods.

Many problems in digital imagery require the estimation of a so-called *label field*. A label field can be seen as an image whose pixels contain a class label. Some of the label-field estimation algorithms come from the Statistical Learning Theory. The most well known algorithms are those based on the *maximum likelihood* and the *maximum a posteriori* principles. Despite the many advantages of those approaches, they are sometimes sensitive to outliers and to noisy input data. In response to those weaknesses, we propose in Chapter 2 a way to enhance the precision of an estimated label field. In fact, the proposed method fuses label fields instead of features as is usually the case. The aim of this method is to force the label field regions to fit the shape of the most predominant object in the scene. To do so, we suggest two energy functions that we minimize with the deterministic Iterative Conditional Mode (ICM) algorithm. The first function is formed by a likelihood energy term whereas the second one is made up of a likelihood and an *a priori* energy term. We also show how this fusion procedure can be adapted to the context of motion segmentation and occlusion detection.

In Chapter 4, we present a way to drastically reduce the processing times related to many statistical learning algorithms applied to computer vision. In fact, we can reach acceleration factors from 4 to 200. Our approach is based on the "parallel" property some algorithms have. In fact, based on that property, we show how many Markovian density estimation and segmentation algorithms can be significantly accelerated when implemented on a Graphics Processor Unit (GPU). A GPU is a streaming processor (*i.e.* a processor with inherent parallel abilities) embedded in most graphics cards nowadays available on the market. In this chapter, we concentrate on Markovian algorithms applied to stereovision, motion estimation, motion segmentation and color segmentation.

Keywords computer vision, spatial contraints, GPU, segmentation, data fusion, optical flow, occlusion.

TABLE DES MATIÈRES

LISTE DES TABLEAUX

LISTE DES FIGURES

LISTE DES SIGLES

API	Application Programmer Interface
CDM	Change Detection Mask
DFD	Displace Frame Difference
DSI	Disparity Space Integration
EM	Expectation Maximization
GPU	Graphics Processor Unit
ICE	Iterative Conditional Estimation
IDD	Identiquement et Indépendamment Distribué
ICM	Iterated Conditional Modes
LK	Lucas Kanade
MAP	Maximum A Posteriori
MRF	Markov Random Field
MV	Maximum de Vraisemblance
PDF	Probability Density Function
SA	Simulated Annealing
WTA	Winner-Take-All

NOTATION

Introduction

B	Image de fond (*background*)
d^f	Champ de disparité *avant* exprimant la correspondance entre les images I^{Ref} et I^{Mat}.
d^b	Champ de disparité *arrière* exprimant la correspondance entre les images I^{Mat} et I^{Ref}.
$g(z)$	Fonction de discrimination
$H(z)$	Entropie
$\vec{I}_s, \vec{I}(s)$	Couleur du pixel s dans l'image I .
I_t	Image au temps t
I_x	Dérivé en x de l'image au temps t
I_y	Dérivé en y de l'image au temps t
I_T	Dérivé temporelle de l'image au temps t
I^{Ref}	Image de *référence*
I^{Mat}	Image de *matching*
O	Carte d'occlusions
$P(z)$	Densité de probabilité suivant laquelle se distribue z
$\hat{P}(z)$	Densité de probabilité estimée
$R(g(z))$	Fonction de risque
$R_{emp}(g(z))$	Fonction de risque empirique
S	Grille orthographique de taille $\mathcal{N} \times \mathcal{M}$ sur laquelle se définit les images, les champs vectoriels ainsi que les champs d'étiquettes.
s	Un site sur la grille S à la position (i, j) .
$\vec{v}_s$	Vecteur de déplacement 2D au site s
X	Champ d'étiquettes

z_i	Donnée complète
$\Phi(z, g(z))$	Fonction de perte
$\Psi(x)$	Fonction de coût
$\Gamma(x)$	Fonction de coût
λ	Constante
τ	Seuil
$\vec{\theta}$	Vecteur de paramètres

Chapitre 1

C_s	Matrice de variance-covariance.
$\mathcal{D}(s)$	Distance entre le site s et le contour le plus près
I_t	Image au temps t
I_x	Dérivé en x de l'image au temps t
I_y	Dérivé en y de l'image au temps t
I_T	Dérivé temporelle de l'image au temps t
K	Fonction *Kernel* utilisée par l'algorithme de mean-shift
S	Grille orthographique de taille $\mathcal{N} \times \mathcal{M}$ sur laquelle se définit les images, les champs vectoriels ainsi que les champs d'étiquettes.
s	Un site sur la grille S à la position (i, j).
$\vec{v}_s$	Vecteur estimé du déplacement 2D au site s ($\vec{v}_s = (u_s, v_s)$)
$\vec{\mathcal{V}}_s$	Vecteur de déplacement 2D réel au site s
W	Facteur de pondération
β_s	Voisinage de taille $\Lambda \times \Lambda$ autour du site s
γ	Site voisin de s
η_s	Voisinage de taille $N \times N$ autour du site s
λ	Constante

$\Psi(x)$ Fonction de coût

ψ_E Erreur angulaire moyenne

ρ Fonction de coût

ζ_s Voisinage de taille $\mathcal{X} \times \mathcal{X}$ autour du site s

$\vec{A}_{\zeta_i}$ Vecteur contenant les paramètres de mouvement de la classe ζ_i.

$B(g,x)$ Pourcentage de pixels mal classés entre l'image g et le champ d'étiquettes x.

E_s Fonction d'énergie associée au site s.

I Image d'entrée

r Carte de régions définit sur la grille orthographique S. r est une réalisation de la variable aléatoire R.

S Grille orthographique de taille $\mathcal{N} \times \mathcal{M}$ sur laquelle se définit les images, les champs vectoriels ainsi que les champs d'étiquettes.

s Un site sur la grille S à la position (i,j). $<s,t>$ est une clique binaire constituée des sites s et t.

$U(r,I)$ Fonction d'énergie globale. $U_1(r_s, I_s)$ est une fonction d'énergie locale d'attache aux données et $U_2(r_s)$ est une fonction d'énergie locale *a priori*.

$\vec{v}_s$ Vecteur de déplacement 2D au site s ($\vec{v}_s = (u_s, v_s)$)

x Champ d'étiquettes *d'application* (*application label field*). x est une réalisation de la variable aléatoire X et $x^{[0]}$ est un champ d'étiquettes initial et $\hat{x}$ est le résultat obtenu après fusion.

T Variable de température utilisée par l'algorithme recuit simulé.

α Constante

β Constante

Λ	Ensemble des étiquettes de région
Γ	Ensemble des étiquettes d'application
Ψ_s	Voisinage de taille $L \times L$ autour du site s
$\psi_E(V, V_g)$	Erreur angulaire moyenne entre les champs vectoriels V et V_g

Chapitre 3

B	Image du fond
D	Distance de Mahalanobis.
I_t	Image au temps t
L_t	Champ d'étiquettes au temps t
K_σ	Noyau gaussien d'écart-type σ
P^u	Densité de probabilité unimodale
P^m	Densité de probabilité multimodale

Chapitre 4

C	Fonction de coût.
x	Champ d'étiquettes
T	Température
w	Filtre passe-bas
U, V, W	Fonctions d'énergie
y	Image d'entrée

Jusqu'ici, tout va bien...

REMERCIEMENTS

Je tiens à remercier tout particulièrement mon directeur de recherche, le professeur Max Mignotte, sans qui cette thèse n'aurait jamais vu le jour. Je tiens à le remercier pour sa rigueur mathématique et pour les excellentes connaissances dans le domaine de l'apprentissage statistique qu'il a su me transmettre. Je tiens aussi à remercier le professeur Christophe Rosenberger de l'Université de Bourges pour son aide dans le projet sur la fusion de champs d'étiquettes. Je remercie aussi le professeur Janusz Konrad de l'Université de Boston pour sa précieuse aide dans le projet de détection de mouvement. Finalement, je remercie ma conjointe Sophie Charbonneau pour son support et Jean-Sébastien Bach pour avoir écrit la musique qui m'a accompagné tout au long de ces années.

INTRODUCTION

Les domaines du traitement d'images et de la vision par ordinateur ont fait preuve d'une vitalité remarquable au cours des trente dernières années. Ce dynamisme s'explique en bonne partie par l'émergence de nouvelles technologies et par le raffinement de technologies déjà existantes. En guise d'exemple, mentionnons l'arrivée récente sur le marché de nombreuses caméras numériques à bas prix, souvent équipées d'un processeur, d'une mémoire RAM et d'une carte de communication sans fil. Mentionnons aussi le développement des technologies liées à l'imagerie médicale ainsi que la commercialisation de processeurs graphiques (les *GPUs*) permettant des calculs parallèles. L'arrivée de ces technologies a entraîné l'explosion de nombreuses applications pouvant bénéficier de leur vertus. Météorologie, océanographie, physique des particules, surveillance vidéo, imagerie médicale ne sont que quelques uns des domaines bénéficiant du traitement et de l'analyse d'images numériques.

L'objectif du traitement d'images est, jusque dans une certaine limite, l'augmentation de la qualité visuelle d'une image dans un contexte d'application donné. Cette qualité d'image s'exprime souvent en fonction de la compréhension qu'aura un observateur de son contenu. C'est ainsi, par exemple, que des procédures de rehaussement, de déconvolution et de réduction de bruit sont utilisées pour améliorer la qualité visuelle d'images bruitées et floues. De même, les différentes procédures de segmentation spatiale mettant en relief les régions d'une image, sont utilisées pour mettre en valeur le contenu utile d'une image. Trouver une procédure automatique pour dissocier le bruit et le flou des informations utiles dans une image constitue souvent le *leitmotiv* du traitement d'images.

Pour ce qui est de la vision par ordinateur, ses objectifs se situent au-delà de ceux visés par le traitement d'images. Alors que ce dernier vise principalement la qualité de l'image, la vision par ordinateur s'intéresse davantage aux informations sémantiques de haut niveau contenues dans l'image. Les algorithmes de vision par ordinateur infèrent donc des informations abstraites en lien avec le contenu intelli-

gible d'une image. Citons en exemple les algorithmes de flux optique dont l'objectif est d'estimer le champ vectoriel optique dans une séquence animée ou les algorithmes de reconstruction 3D ayant pour but de localiser en 3D les objets contenus dans une scène stéréoscopique.

Bien que ces domaines soient distincts dans le domaine du traitement de signal, ils partagent tout de même un point en commun : leurs problèmes fondamentaux se réduisent souvent à un problème d'estimation de fonctions. En d'autres mots, étant donné un ensemble de fonctions admissibles à un problème, le but de nombreux algorithmes en imagerie numérique est de trouver *la* fonction qui saurait le mieux satisfaire un critère de qualité donné. De façon plus formelle, suivant un espace vectoriel Z, un ensemble de fonctions admissible $G = \{g(z), z \in Z\}$ et un critère de qualité C,

$$C = R(g), \tag{1}$$

le but de nombreux algorithmes d'imagerie est d'estimer une fonction $\hat{g}(z) \in G$ (qu'on appelle parfois *fonction de discrimination*) qui saura le mieux minimiser le critère $R(g)$. Lorsque les données sont indépendamment et identiquement distribuées (i.i.d.) selon une densité de probabilité $P(z)$, il est fréquent d'exprimer $R(g)$ sous la forme d'une espérance mathématique,

$$R(g) = \int \Phi(z, g(z))P(z)dz \tag{2}$$

où $\Phi(z, g(z))$ est parfois appelée fonction de perte (*loss function*). Ainsi défini, le critère de qualité $R(g)$ porte le nom de fonction de *risque*. En supposant connu un ensemble de données $\{z_1, z_2, ..., z_l\}$, minimiser la fonction (2) en vertu de celles-ci devient un problème de *minimisation du risque sur la base de données empiriques* [157]. Ainsi définis, les problèmes dévolus au traitement d'images et à la vision par ordinateur résident au coeur de la théorie de l'apprentissage statistique. Le domaine de la minimisation du risque est un domaine en soi qui, comme le mentionne Vapnik

[157], inclus trois problèmes fondamentaux, *i.e.*

1. la reconnaissance de formes ;
2. la régression ;
3. l'estimation de densité.

Ce qui distingue ces trois problèmes est le contenu des données z ainsi que la définition de la fonction de perte $\Phi(z, g(z))$. Dans les deux prochaines sections, nous démontrerons comment ces trois problèmes peuvent s'exprimer sous l'angle de la minimisation d'une fonction de risque. Nous démontrerons aussi comment de nombreux problèmes en vision par ordinateur se réduisent à l'un ou l'autre de ces trois problèmes. Nous porterons alors notre attention sur le flux optique, la détection de mouvement, la détection d'occlusions ainsi que la segmentation de mouvement.

0.1 La minimisation du risque

Comme nous l'avons mentionné, de nombreux problèmes en imagerie numérique se réduisent à des problèmes d'apprentissage statistique formulés sous la forme d'une fonction de risque à minimiser. Par conséquent, une question centrale en imagerie concerne la façon dont le risque peut être minimisé dans le contexte d'une application donnée. Malheureusement, pour de nombreuses applications, la distribution de probabilité $P(z)$ est inconnue et malgré la simplicité apparente de l'équation (2), il est impossible de minimiser $R(g)$ sur la base exclusive des données $\{z_1, z_2, ..., z_l\}$. La minimisation du risque est donc un problème difficile et souvent mal conditionné. Il existe toutefois trois solutions classiques à ce problème de minimisation.

La première solution au problème de minimisation du risque est d'approximer $P(z)$ par une fonction empirique $\hat{P}(z)$ apprise sur les données $\{z_1, z_2, ..., z_l\}$ *i.i.d.* de $\hat{P}(z)$. Une fois estimée, $P(z)$ est remplacée par $\hat{P}(z)$ dans (2) et cette nouvelle formulation (qu'on représente ici par $\hat{R}(g)$) peut en principe être minimisée. À la lumière de cette solution, le problème de minimisation du risque sur la base

de données se réduit à un problème d'estimation de densité. Toutefois, comme le mentionne Vapnik [157], le problème de l'estimation de densité ne peut être résolu qu'à condition d'avoir des connaissances *a priori* très fortes sur la nature de la distribution $P(z)$. Dans les faits, cela revient à dire que $P(z)$ ne peut être estimé que si la nature de sa distribution est connue. En somme, on peut résumer cette première méthode de minimisation du risque comme suit :

1. étant donné un ensemble de données $\{z_1, z_2, ..., z_l\}$, estimer la densité de probabilité $\hat{P}(z)$.
2. Redéfinir l'espérance mathématique

$$\hat{R}(g) = \int \Phi(z, g(z))\hat{P}(z)dz \tag{3}$$

   à l'aide de la densité de probabilité nouvellement estimée.
3. Trouver la fonction $\hat{g}(z)$ qui minimise le mieux la fonction (3), *i.e.*

$$\hat{g} = \arg\min_{g'} \hat{R}(g'). \tag{4}$$

Une deuxième façon de minimiser le risque est de remplacer la distribution $P(z)$ par une distribution uniforme [157],

$$R_{emp}(g) = \frac{1}{l}\sum_{i=1}^{l} \Phi(z_i, g(z_i)). \tag{5}$$

Cette approche porte le nom de *minimisation du risque empirique*. Alors que $R(g)$ exprime l'espérance mathématique de $\Phi(z, g(z))$ pour une fonction $g(z)$ donnée, $R_{emp}(g)$ exprime la moyenne empirique de $\Phi(z, g(z))$. Il est intéressant de savoir qu'en vertu de théorèmes classiques en statistique [157], la moyenne empirique converge vers l'espérance mathématique lorsque l devient grand. Conséquemment, minimiser le risque empirique est une option avantageuse lorsque le nombre d'échantillons l est élevé. Par contre, lorsque le nombre d'échantillons est petit, un risque empi-

rique faible ne garantie pas un risque réel faible, surtout lorsque la complexité du modèle estimé g_{emp} est élevée. Dans un tel cas de figure, il est préférable d'avoir recours à une approche par *minimisation du risque structurel* comme le mentionne Vapnik [157]. Cette approche d'estimer g sur la base d'un compromis entre la complexité de g et le nombre de données observées l. Toutefois, malgré l'engouement croissant pour cette approche, nous n'en ferons pas référence au cours de cette thèse. La raison étant que les méthodes proposées dans cette thèse disposent d'un nombre d'échantillons l suffisamment élevé pour justifier l'utilisation du risque empirique.

Dans les trois prochaines sous sections, nous verrons comment la reconnaissance de formes, la régression ainsi que l'estimation de densité peuvent s'exprimer sous la forme de l'équation (2). Nous soulignerons aussi les façons les plus usuelles de minimiser le risque pour chacun de ces problèmes.

0.1.1 La reconnaissance de formes

Le problème de la reconnaissance de formes peut se formuler ainsi. Soit un ensemble d'apprentissage $\{\vec{z}_1, \vec{z}_2, ..., \vec{z}_l\}$ où chaque donnée $\vec{z}_i = (\vec{y}_i, x_i)$ est constituée de deux éléments : une donnée à n dimensions $\vec{y}_i \in \mathbb{R}$ (distribuée suivant $P(\vec{y})$) et une étiquette $x_i \in \{0, ..., k-1\}$. $\vec{z}_i$ est donc un binôme constitué d'une valeur $\vec{y}_i$ appartenant à une classe x_i. Certains auteurs appellent $\vec{z}_i$ une *donnée complète*. On suppose aussi que la classification des données suit une distribution $P(x|\vec{y})$. Bien que $P(x|\vec{y})$ et $P(\vec{y})$ soient inconnues *a priori*, on suppose qu'elles existent, tout comme la distribution jointe $P(\vec{y}, x) = P(x|\vec{y})P(\vec{y})$. Le but de la reconnaissance de formes est d'estimer une fonction discriminante $\hat{g}(\vec{y})$ qui sache le mieux classifier les données $\vec{y}_i$.

En prenant la fonction $\Phi(x, g(\vec{y}))$ comme mesure d'erreur entre la valeur retournée par $g(\vec{y})$ et la vrai classe x,

$$\Phi(x, g(\vec{y})) = L(x - g(\vec{y})) \tag{6}$$

où $L(\cdot)$ est une fonction de coût. La fonction de risque liée à la reconnaissance de

formes peut désormais s'exprimer par

$$R(g) = \int L(x - g(\vec{y}))\hat{P}(\vec{y}, x)dxdy \tag{7}$$

ou, dans le cas discret,

$$R(g) = \sum_{i=1}^{l} L(y_i - g(\vec{y}_i))\hat{P}(\vec{y}_i, x_i). \tag{8}$$

Le problème de la reconnaissance de formes se réduit donc à un problème de minimisation de l'erreur de classification L, compte tenu d'un ensemble d'observations et d'une distribution de probabilité $P(x, \vec{y})$ initialement inconnue. En vertu du principe de minimisation du risque empirique, on peut inférer la fonction $\hat{g}$ en minimisant la fonction

$$R_{emp}(g) = \frac{1}{l}\sum_{i=1}^{l} L(x - g(\vec{y}_i)). \tag{9}$$

Il est intéressant de noter que cette formulation est celle retenue par le célèbre algorithme du Perceptron, algorithme proposé par Frank Rosenblatt en 1957. Pour le Perceptron, L est une fonction quadratique et $g = \text{sign}(\vec{y} \cdot \vec{w})$ où $\vec{w} = (w_1, ..., w_n)$ est le vecteur de paramètres du Perceptron.

Comme nous l'avons mentionné précédemment, une autre façon de minimiser l'équation (7) est d'estimer la distribution $\hat{P}(\vec{y}, x)$ et de la substituer à $P(\vec{y}, x)$. Ainsi,

$$\hat{R}(g) = \sum_{i=1}^{l} \Phi(x, g(\vec{y}_i))\hat{P}(\vec{y}_i, x_i). \tag{10}$$

Dans ce contexte, une fonction de perte souvent utilisée en est la fonction symétrique

(aussi appelée *zero-one function*) [130]

$$\Phi(x, g(\vec{y})) = \begin{cases} 0 & \text{si } y = g(\vec{y}) \\ 1 & \text{sinon.} \end{cases} \tag{11}$$

Suivant cette fonction de perte, on peut réécrire la fonction de risque (10) comme étant

$$\hat{R}(g) = \sum_{\substack{i=1 \\ g(\vec{y}_i) \neq x_i}}^{l} \hat{P}(\vec{y}_i, x_i)$$

et, suivant le théorème de Bayes,

$$\begin{aligned} \hat{R}(g) &= \sum_{\substack{i=1 \\ g(\vec{y}_i) \neq x_i}}^{l} \hat{P}(x_i|\vec{y}_i)\hat{P}(\vec{y}_i) \\ &= \sum_{\substack{i=1 \\ g(\vec{y}_i) \neq x_i}}^{l} \hat{P}(\vec{y}_i|x_i)\hat{P}(x_i). \end{aligned} \tag{12}$$

Suivant cette formulation, pour minimiser le risque on doit sélectionner, pour chaque donnée observée $\vec{y}_i$, la classe qui minimise le mieux le produit $\hat{P}(\vec{y}_i|x_i)\hat{P}(x_i)$. En d'autres mots, pour obtenir un taux d'erreurs minimum, il faut associer à $\vec{y}_i$ la classe x_j tel que

$$\hat{P}(\vec{y}_i|x_j)\hat{P}(x_j) > \hat{P}(\vec{y}_i|x_k)\hat{P}(x_k) \qquad \forall k \neq j. \tag{13}$$

C'est ce qu'on appelle la règle décision de Bayes, ou encore la règle du *Maximum a posteriori* [130]. Advenant le cas où $P(x)$ est une distribution constante, la règle de décision de Bayes se réduit à un *maximum de vraisemblance*, *i.e.*

$$\hat{P}(\vec{y}_i|x_j) > \hat{P}(\vec{y}_i|x_k) \qquad \forall k \neq j. \tag{14}$$

0.1.2 La régression

Soit un ensemble de données $(\vec{y}_1, x_1), ..., (\vec{y}_l, x_l)$ où $\vec{y}_i \in \mathbb{R}^n$ et $x_i \in \mathbb{R}$. On suppose que ces données sont la réalisation du couple de variables aléatoires $\vec{y}$ et x se distribuant suivant la distribution jointe $P(\vec{y}, x)$, distribution initialement inconnue. Le but de la régression est d'estimer une fonction g s'alignant le mieux sur les données observées. Formellement, le problème de la régression peut s'exprimer de la façon suivante,

$$x_i = g(\vec{y}_i) + e_i \qquad \forall i \in [1, l] \tag{15}$$

où $\vec{y}_i$ est une variable dite *explicative*, x_i est une variable dite *réponse* et e_i un terme d'erreur i.i.d. de $P(e)$ [112]. À noter que pour le modèle linéaire classique, la relation (15) devient

$$x_i = \theta_1 y_{i,1} + ... + \theta_n y_{i,n} + e_i \qquad \forall i \in [1, l]. \tag{16}$$

Le but de la régression est donc d'estimer les *meilleurs* paramètres de la fonction $g(\vec{y}_i)$ ($\{\theta_1, ..., \theta_n\}$ pour le modèle linéaire) tel que la valeur résiduelle r_i,

$$r_i = \Psi(x_i - \hat{x}_i) \tag{17}$$

soit minimale étant donné la fonction d'importance Ψ. En prenant la valeur r comme fonction de perte dans l'équation (2), on obtient que le problème de régression s'exprime sous la forme

$$R(g) = \int_{\vec{y},x} \Psi(x - g(\vec{y}))P(\vec{y}, x)dxdy$$
$$R(g) = \int_{\vec{y},x} \Psi(x - g(\vec{y}))P(x|\vec{y})P(\vec{x})dxdy$$

ou, dans le cas discret,

$$R(g) = \sum_{i=1}^{l} \Psi(x_i - g(\vec{y}_i)) P(x_i|\vec{y}_i) P(\vec{y}). \tag{18}$$

La façon la plus usuelle d'estimer $\hat{g}$ est de minimiser le risque empirique, c'est-à-dire de minimiser [157]

$$R_{emp}(g) = \frac{1}{l} \sum_{i=0}^{l} \Psi(x_i - g(\vec{y}_i)). \tag{19}$$

Lorsque Ψ prend la forme quadratique, on obtient la méthode bien connue de la minimisation par la *somme des moindres carrés* [112,119]. Fait intéressant, on peut facilement démontrer que si la distribution $P(x_i|\vec{y}_i)$ suit une loi gaussienne $\mathcal{N}(0, \sigma)$, minimiser (18) revient à une minimisation par la somme des moindres carrés [156]. Il est aussi intéressant de noter qu'en vertu du théorème de Gauss-Markov [112,156], l'estimateur des moindres carrés est le meilleur estimateur linéaire non biaisé (*Best Linear Unbiased Estimator* (BLUE)). Cet estimateur possède aussi l'avantage notable d'estimer le vecteur de paramètres $\vec{\theta}$ du modèle linéaire de l'équation (16) de façon explicite, à l'aide d'une simple inversion matricielle [112,119]. Nous reverrons à la section 0.2.1 que cette propriété peut être récupérée pour estimer le flux optique.

0.1.3 L'estimation de densité

Le problème de l'estimation de densité peut se comprendre par le biais de la théorie de l'information. Supposons qu'une source émette une série de n symboles $\{z_1, z_2, ..., z_n\}$ où chaque élément z_i est une réalisation d'une variable aléatoire z, i.i.d. de $P(z)$. Le but de l'estimation de densité est d'estimer $P(z)$ sur la base des données observées. Un concept fondamental en théorie de l'information est qu'un symbole z_i ayant une probabilité $P(z_i)$ contient,

$$S(z_i) = -\log P(z_i) \tag{20}$$

unités d'information [139] (c'est ce que certains auteurs appellent la *surprise* [130]). À noter que l'unité d'information résultante pour un logarithme en base 2 est le *bit*. Si la source émet au total «l»symboles, en vertu de la théorie des grands nombres, le symbole z_i devrait être émit en moyenne $l \times P(z_i)$ fois. Par conséquent, la surprise totale pour le symbole z_i dans une séquence contenant l éléments est

$$S_{\text{tot}}(z_i) = lP(z_i)S(z_i) = -lP(z_i)\log P(z_i) \tag{21}$$

et, pour l'ensemble des n symboles,

$$S_{\text{tot}}(z) = -l\sum_{i=1}^{n} P(z_i)\log P(z_i). \tag{22}$$

Il est clairement établi en théorie de l'information que l'information moyenne contenue dans une séquence donnée est la surprise moyenne, c'est-à-dire l'entropie [130]

$$H(z) = -\sum_{i=1}^{n} P(z_i)\log P(z_i) \tag{23}$$

ou, dans le cas continu

$$H(z) = -\int P(z)\log P(z)dz. \tag{24}$$

L'estimation de densité a pour objectif d'estimer une densité de probabilité $\hat{P}(z)$ qui minimise le mieux l'entropie. En d'autres mots, estimer une densité qui rende compte le mieux de l'ordre dans lequel les données sont distribuées. Il est intéressant de noter qu'en prenant $-\log P(z)$ comme fonction de perte, $H(z)$ peut s'exprimer sous la forme d'une fonction de risque,

$$R(P(z)) = \int Q(z, P(z))P(z)dz. \tag{25}$$

Comme nous l'avons mentionné pour la reconnaissance de formes et la régression,

$P(z)$ peut être estimée en minimisant le risque empirique, c'est-à-dire

$$R_{\text{emp}}(P(z)) = \sum_{i=1}^{l} Q(z_i, P(z_i)) = -\sum_{i=1}^{l} \log P(z_i). \tag{26}$$

Dans ce contexte, minimiser le risque empirique revient à un *maximum de vraisemblance* (*maximum likelihood* en anglais) [130].

0.2 La vision par ordinateur

Nombreux sont les problèmes en vision par ordinateur pouvant s'exprimer sous la forme d'un problème d'apprentissage statistique, c'est-à-dire comme un problème de minimisation d'une fonction de risque. En effet, de nombreux problèmes en vision ont pour objectif d'estimer une fonction $g(z)$ optimale au sens d'une fonction de risque. Malheureusement, estimer une telle fonction n'est pas toujours chose facile. Par exemple, il arrive que ce type d'optimisation se réduise à un problème dit *inverse mal posé*. En effet, comme le mentionne Vapnik [157], un problème est dit bien posé au sens de Hadamard, si et seulement si la solution au problème

1. existe ;
2. est unique ;
3. est stable.

Or, si l'une ou l'autre de ces conditions est violée, le problème est alors considéré mal posé. Pour rendre le problème bien posé, des termes de régularisation (sous forme de fonction *a priori*) sont souvent utilisées. Dans bien des cas cependant, ce terme de régularisation implique des masses de calculs, souvent prohibitifs pour les applications en temps réelle.

L'objectif de cette thèse est de proposer une série de contraintes spatiales à bas niveau pour résoudre des problèmes d'optimisation de fonctions de risque propres à la vision par ordinateur. En particulier, nous adressons les problèmes de flux optique, de détection de mouvement, de détection d'occlusion ainsi que de segmentation de mouvement. Dans les prochaines sous-sections, nous présentons en quoi

consiste ces quatre problèmes et comment ils peuvent s'exprimer sous la forme d'un problème d'apprentissage statistique.

0.2.1 Flux optique

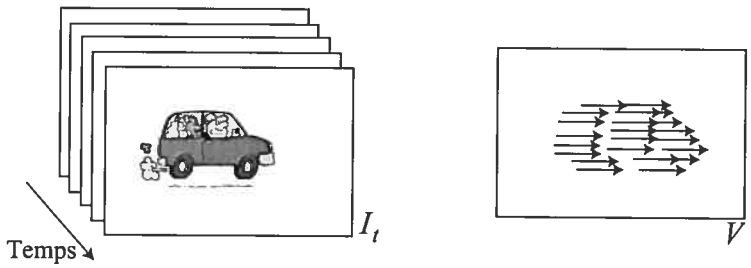

FIG. 1 – *Exemple schématique du flux optique : partant d'une séquence vidéo constituée d'une suite d'images I_t, l'objectif est d'estimer le mouvement apparent des objets à chaque temps t. Le mouvement estimé prend souvent la forme d'un champ vectoriel V.*

Soit une séquence vidéo I constituée d'une suite d'images I_t (aussi appelée *frames*) définies sur une grille orthographique $S = \{s = (i, j))|i \in [0, N[, j \in [0, M[\}$ et où t indique le temps. L'objectif de tout algorithme de flux optique est d'estimer le mouvement apparent des objets présents dans la séquence vidéo. Formellement, cela revient à estimer un champ vectoriel $V = \{\vec{v}_s|s \in S, \vec{v}_s \in \mathbb{R}^2\}$ où chaque vecteur 2D $\vec{v}_s$, exprime le déplacement du pixel s au temps t. Parfois dans la littérature, on appelle «donnée complète»le duo (I, V), où I est une donnée observée et V est une donnée à estimer. Comme l'illustre la figure 2, la plupart des méthodes d'estimation de mouvement émettent l'hypothèse qu'un point s observé au temps t possède la même intensité (ou couleur) que sa projection s' au temps $t + 1$ [13, 64, 70]. Cette hypothèse de constance de l'illumination (appelée *lightness constancy assumption* dans la littérature [100]) se représente mathématiquement comme suit :

$$I_t(s) = I_{t+1}(s') = I_{t+1}(s + \vec{v}_s). \tag{27}$$

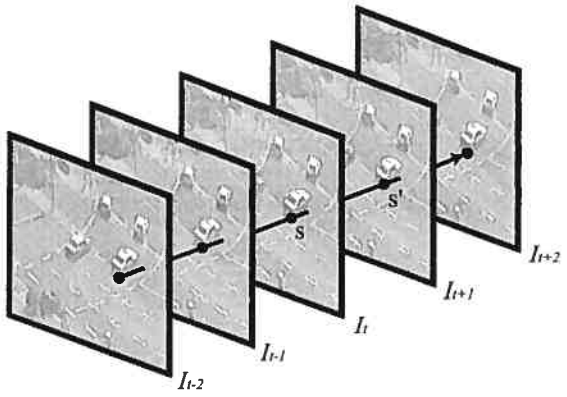

FIG. 2 – *Figure illustrant le concept de la constance de l'illumination.*

Sans perte de généralité, on peut dire qu'en vertu de l'hypothèse de la constance de l'illumination, l'erreur résiduelle

$$E(\vec{v}_s) = (I_t(s) - I_{t+1}(s + \vec{v}_s))^2 \tag{28}$$

est nulle lorsque le vecteur $\vec{v}_s$ décrit parfaitement le mouvement au site s. Il est fréquent de redéfinir cette équation à l'aide d'un développement en série de Taylor du premier ordre [51], c'est-à-dire

$$\begin{aligned} E(\vec{v}_s) &\approx (u_x I_x + u_y I_y + I_t)^2 \\ &\approx (\nabla I \vec{v}_s + I_t)^2 \end{aligned} \tag{29}$$

où $\vec{v}_s = (u_x, u_y)$ et I_x, I_y, I_t sont les dérivées partielles spatiales et temporelles de l'image au temps t. Une des façons d'estimer $\vec{v}_s$ est de trouver le minimum de cette fonction d'erreur, *i.e.* l'endroit ou la dérivée $dE(\vec{v}_s)/d\vec{v}_s$ s'annule,

$$\begin{aligned} \frac{dE(\vec{v}_s)}{d\vec{v}_s} = \nabla I(\nabla I \vec{v}_s + I_t) &= 0 \\ \nabla I \vec{v}_s + I_t &= 0 \end{aligned} \tag{30}$$

Cette dernière équation est fort célèbre et porte le nom «d'équation de contrainte du mouvement» (*motion constraint equation* en anglais). Malheureusement, trouver le *meilleur* vecteur $\vec{v}_s$ en vertu de l'équation du mouvement est un problème inverse mal posé. En effet, puisque la relation (30) contient une seule équation pour deux inconnus, il existe une infinité de solutions «optimales» à ce problème. La preuve étant que la solution algébrique à l'équation (30)

$$\vec{v}_s = -((\nabla I)^{\mathrm{T}} \nabla I)^{-1} (\nabla I)^{\mathrm{T}} I_t$$

ne peut être résolue, la matrice $(\nabla I)^{\mathrm{T}} \nabla I$ étant singulière. Pour rendre bien posé ce problème, on retrouve deux familles d'approches dans la littérature. La première famille est celle faisant appel à un *terme de régularisation* [6, 21, 43, 49, 62, 75, 80, 102, 150] de la forme

$$E(\vec{v}_s) = \Gamma(\nabla I \vec{v}_s + I_T) + \lambda \Psi(|\nabla u|^2 + |\nabla v|^2) \tag{31}$$

où λ est une constante et Γ et Ψ sont des fonctions d'importance (fonctions absolues, quadratiques, robustes, etc.). L'approche la plus répandue de cette famille est celle proposée par Horn et Schunck [21], approche fondée sur la fonction de régularisation de Tikhonov (aussi appelée le *weak membrane model* [102])

$$E(\vec{v}_s) = (\nabla I \vec{v}_s + I_t)^2 + \lambda(|\nabla u|^2 + |\nabla v|^2).$$

Malgré les avantages indéniables des méthodes basées sur l'équation (31), ces dernières sont contraintes à estimer V à l'aide d'optimiseurs itératifs, généralement très lents (Jacobi, Gauss-Seidel, Successive Over-Relaxation, etc.). Il existe toutefois une alternative à cette famille de fonctions. En effet, plusieurs auteurs ont proposé des méthodes dites *spatiales* qui, au lieu d'ajouter un terme de régularisation à l'équation (30), multiplient le nombre d'observations afin de rendre le problème bien posé [5, 22, 31, 85, 100]. Mathématiquement, cette famille d'approches cherche

à minimiser une fonction d'énergie de la forme,

$$E(\vec{v}_s) = \sum_{r \in \eta_s} \Psi(\nabla I_r \vec{v}_s + I_{T_r}) \tag{32}$$

où η_s est un voisinage (généralement carré) centré sur s. Il est intéressant de noter qu'en posant $g(\vec{v}_s) = -\nabla I_r \vec{v}_s$ et $y_i = I_{T_r}$, cette dernière formulation est équivalente à la régression de l'équation (19). En fait, l'équation (32) est un modèle de régression linéaire *multivariée* [112,119]. La solution spatiale la plus répandue est sans contredit celle de Lucas et Kanade [22] pour laquelle Ψ est une fonction quadratique,

$$E(\vec{v}_s) = \sum_{r \in \eta_s} (\nabla I_r \vec{v}_s + I_{T_r})^2. \tag{33}$$

En vertu de cette équation, le vecteur optimal $\vec{v}_s$ est celui pour lequel la dérivée $dE(\vec{v}_s)/d\vec{v}_s$ s'annule, c'est-à-dire

$$\begin{aligned}
\frac{dE(\vec{v}_s)}{d\vec{v}_s} &= 0 \\
\sum_{r \in \eta_s} (\nabla I_r \vec{v}_s + I_{T_r}) &= 0 \qquad (34) \\
\sum_{r \in \eta_s} ((\nabla I_r)^{\mathrm{T}} \nabla I_r \vec{v}_s + (\nabla I_r)^{\mathrm{T}} I_{T_r}) &= 0 \\
\left[\begin{pmatrix} \sum_r I_{x_r}^2 & \sum_r I_{x_r} I_{y_r} \\ \sum_r I_{x_r} I_{y_r} & \sum_r I_{y_r}^2 \end{pmatrix} \vec{v}_s + \begin{pmatrix} \sum_r I_{T_r} I_{x_r} \\ \sum_r I_{T_r} I_{y_r} \end{pmatrix} \right] &= 0 \\
M_s \vec{v}_s + b_s &= 0. \qquad (35)
\end{aligned}$$

Suivant cette formulation, le vecteur optimal de déplacement $\vec{v}_s$ peut donc être estimé à l'aide d'une simple inversion matricielle, *i.e.*

$$\vec{v}_s = -M_s^{-1} b_s. \tag{36}$$

Calculer ainsi $\vec{v}_s$ revient à résoudre un problème de régression *par la somme des moindres carrés*. L'avantage de cette solution réside autant dans sa simplicité

conceptuelle que dans la rapidité avec laquelle elle calcule $\vec{v}_s$. En effet, estimer $\vec{v}_s$ de façon explicite demande beaucoup moins de calculs qu'avec un optimiseur itératif. Autre avantage, si les valeurs I_T observées à l'intérieur du voisinage η_s sont des échantillons i.i.d. d'une densité gaussienne, alors $\vec{v}_s$ est le meilleur estimateur non biaisé [112, 157].

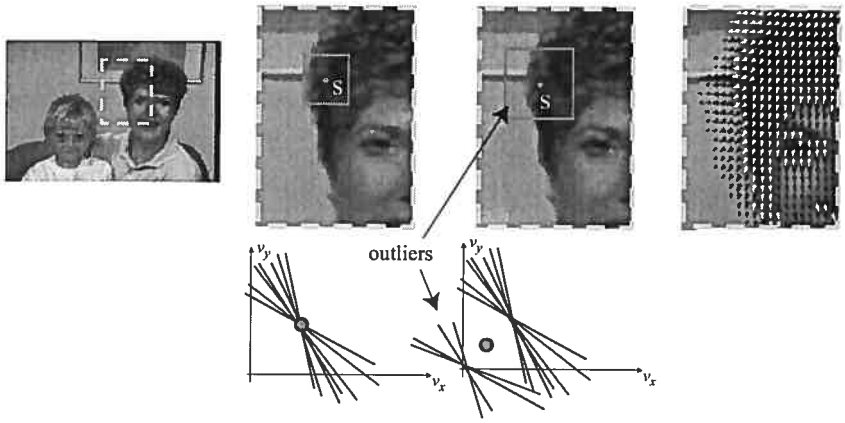

FIG. 3 – *(a) Résultat d'une régression par moindres carrés dans une région sans discontinuité de mouvement et (b) dans une région avec discontinuité de mouvement.*

Malheureusement, l'équation (36) souffre de deux limitations majeures. Tout d'abord, dans les régions peu texturées de l'image au temps t (les régions pour lesquelles $I_x \approx I_y \approx 0$) la matrice M_s devient singulière et ne peut être inversée. Par conséquent, l'approche de Lucas-Kanade peine à estimer le mouvement à l'intérieur de régions peu texturées. La façon classique de compenser pour cette limitation est d'utiliser un voisinage η_s plus grand afin d'inclure davantage de texture. Malheureusement, plus η_s est grand et plus grande est la possibilité de couvrir des zones de mouvement contenant une discontinuité (zones dites *bimodales*). En effet, tel qu'illustré à la figure 3, plus un voisinage est grand et plus il aura tendance à couvrir des zones dont le mouvement présente des discontinuités. Dans ce cas, $\vec{v}_s$ sera biaisé par des valeurs éloignées (des *outliers*). La méthode de Lucas-Kanade

est donc motivée par deux besoins contradictoires : (1) utiliser un grand voisinage η_s pour bien conditionner M_s et (2), utiliser un petit voisinage pour éviter de couvrir des zones présentant des discontinuités de mouvement. C'est ce qu'on appelle communément le *problème de l'ouverture*. Pour lutter contre l'effet des *outliers*, plusieurs solutions ont été proposées allant de la somme des moindres carrés pondérés [70], aux fonctions robustes [102], aux approches par *least median of squares* [50].

Dans le cadre de cette thèse, nous proposons l'utilisation de contraintes spatiales locales afin de mieux régulariser l'estimation de mouvement par la méthode de régression Lucas-Kanade. Notre méthode (présentée au chapitre 1) utilise deux contraintes. La première contrainte s'exprime sous la forme d'un filtrage de données de type *Best Linear Unbiased Estimate* (BLUE). Cette contrainte sert à régulariser les résultats dans les régions peu texturées et à minimiser l'effet du bruit. La seconde contrainte est fondée sur l'algorithme du mean-shift [37] et a pour but d'éliminer les valeurs extrêmes (les *outliers*) lors de l'estimation du mouvement. L'idée maîtresse de cette seconde contrainte est de légèrement déplacer le voisinage η_s vers des régions dont le mouvement a plus de chances d'être unimodal que bimodal.

0.2.2 Détection de mouvement

L'objectif de tout algorithme de détection de mouvement est de dissocier les zones mobiles des zones immobiles dans une séquence vidéo. Étant donné une séquence vidéo I telle que définit à la section précédente, l'objectif est d'estimer un champ d'étiquettes X contenant pour chaque pixel $s \in S$, une étiquette binaire de mouvement $x_s \in \{\text{immobile, mobile}\}$. Dans la littérature, X est parfois appelé *Displace Frame Difference* (DFD) où encore *Change Detection Mask* (CDM). La détection de mouvement peut être vue comme un problème de reconnaissance de formes. Autrement dit, comme un problème pour lequel il faut associer à chaque observation $\vec{I}_t(s)$ une étiquette de mouvement x_s. Avant d'expliciter la relation existant entre la détection de mouvement et l'apprentissage statistique, voyons d'abord comment fonctionnent les algorithmes de détection de mouvement classiques.

FIG. 4 – *Illustration schématique du problème de la détection de mouvement. Partant d'une séquence vidéo I, l'objectif est d'estimer à chaque temps t, un champ d'étiquettes binaire X_t.*

Une configuration fréquente en détection de mouvement est constituée d'une caméra fixe filmant un fond (un *background*) immobile devant lequel s'animent des personnages et/ou des objets. C'est entre autres le cas pour les applications de surveillance vidéo, de contrôle de qualité le long d'une ligne de montage et de monitoring de trafic. Suivant cette configuration, l'image du fond est immobile et l'intensité de ses pixels est considérée stationnaire dans le temps. Une hypothèse très fréquemment émise par les algorithmes de détection de mouvement est qu'un objet mobile est fait de couleurs (ou de niveaux de gris) différentes de celles observées sur le fond. Par conséquent, une façon simple d'estimer le champ d'étiquettes X au temps t est d'appliquer un seuil sur la différence entre B, l'image du fond, et I_t, l'image au temps t,

$$X_t(s) = \begin{cases} 0 & \text{si } |\vec{B}(s) - \vec{I}_t(s)| < \tau \\ 1 & \text{sinon.} \end{cases} \tag{37}$$

où τ est un seuil donné par l'utilisateur [12]. C'est ce qu'on appelle communément une détection de mouvement par soustraction de fond (*background substraction*). Bien qu'une opération comme celle-ci soit simple et rapide, elle est toutefois très sensible aux variations d'intensité dans le fond et ce, même si τ peut être localement adapté pour ajouter de la robustesse [47, 149]. Une solution à ce problème consiste

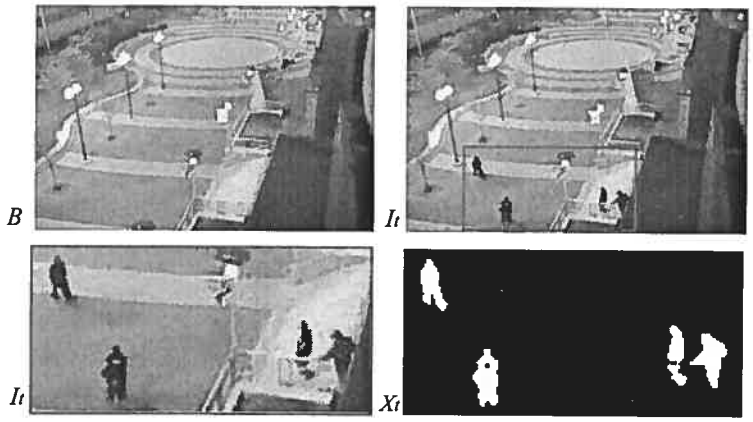

FIG. 5 – *Exemple d'une détection de mouvement. Le distribution temporelle de chaque pixel du fond est modélisée par une distribution gaussienne.*

à modéliser chaque pixel du fond par une distribution temporelle de niveaux de gris/couleurs. Dans ce cas, on peut modéliser chaque pixel du fond, non pas seulement par une seule valeur $\vec{B}(s)$, mais par une densité de probabilité $P(\vec{B}(s))$. Dans ce cas, le champs d'étiquettes X est estimé en appliquant un seuil sur P,

$$X_s = \begin{cases} 0 & \text{si } P(I_t(s)) > \tau' \\ 1 & \text{sinon.} \end{cases} \tag{38}$$

L'avantage de cette méthode est que $P(\vec{B}(s))$ peut modéliser les variations de couleur/d'intensité du background dans le temps. Par exemple, pour contrer l'effet du bruit numérique causé par une caméra bas de gamme, plusieurs auteurs modélisent la distribution temporelle des niveaux de gris/couleur de chaque pixel par une densité de probabilité unimodale (la plupart du temps gaussienne). Plusieurs articles ont d'ailleurs démontré l'efficacité de cette approche, en particulier pour les scènes d'intérieur [33, 143]. Par contre, pour les scènes dont le background contient des objets mobiles (des vagues sur l'eau ou un arbre secoué par le vent par exemple) ou encore des scènes dont la caméra est instable, la distribution temporelle des niveaux de gris/couleur ne peut être correctement modélisée par

une distribution unimodale. Dans ce cas, une distribution multimodale est à privilégier. Par exemple, un mélange de Gaussiennes [14, 29, 113] ou une distribution non-paramétrique (comme celle des fenêtres de Parzen par exemple [7,15]) peuvent être utilisées avec succès. Mentionnons aussi que des méthodes dites de prédiction basées sur les filtres de Kalman ont aussi été proposées pour détecter du mouvement dans des scènes dont le fond n'est pas stationnaire [84].

Ainsi définie, la détection de mouvement peut être vue comme une forme indirecte de reconnaissance de formes de type *maximum a posteriori* (équation (13)). En effet, soit la probabilité qu'un pixel s de couleur $\vec{I}_t(s)$ au temps t appartienne au fond : $P(\vec{I}_t(s)|0) = P(\vec{B}(s))$ Sans perte de généralité, on peut affirmer que la probabilité que ce pixel appartiennent à un objet en mouvement est : $P(\vec{I}_t(s)|1) = 1 - P(\vec{B}(s))$. Par conséquent, étant donnée l'équation (13), on peut établir que la détection de mouvement a pour objet d'assigner à chaque pixel s, l'étiquette ω_i maximisant le produit $P(\vec{I}_t(s)|\omega_j)P(\omega_j)$, *i.e.*

$$\begin{aligned}
P(\vec{I}_t(s)|\omega_j)P(\omega_j) &> P(\vec{I}_t(s)|\omega_i)P(\omega_i) \qquad \text{pour } i \neq j \\
P(\vec{I}_t(s)|\omega_j)P(\omega_j) &> (1 - P(\vec{I}_t(s)|\omega_j))P(\omega_i) \\
P(\vec{I}_t(s)|\omega_j)\frac{P(\omega_i)}{P(\omega_j)} &> 1 - P(\vec{I}_t(s)|\omega_j) \\
P(\vec{I}_t(s)|\omega_j)(1 + \frac{P(\omega_i)}{P(\omega_j)}) &> 1 \\
P(\vec{I}_t(s)|\omega_j) &> \frac{1}{1 + \frac{P(\omega_i)}{P(\omega_j)}}. \qquad (39)
\end{aligned}$$

Cette dernière relation correspond à l'équation (38) dans la mesure où $\tau' = \frac{1}{1+\frac{P(\omega_i)}{P(\omega_j)}}$. L'équation (37) peut aussi être vue comme un cas particulier de l'équation (39). Dans ce cas, $P(\vec{I}_t(s)|\omega_j) = \frac{1}{\lambda_s}\exp(-|\vec{B}(s) - \vec{I}_t(s)|)$ et $\tau = -\log(1 + \frac{P(\omega_i)}{P(\omega_j)})$.

La détection de mouvement peut donc être comprise comme un problème de reconnaissance de formes par *Maximum a posteriori*. Par contre, la détection de mouvement est aussi, pour de nombreuses méthodes, un problème d'estimation de densité. En fait, un problème pour lequel on doit estimer la distribution statistique

temporelle des niveaux de gris/couleurs de chaque pixel du fond $P(\vec{B}(s))$. Pour ce faire, la plupart des méthodes utilisant des modèles paramétriques de densité (gaussiennes, exponentielles, mélange de densité, etc.) ont recours au maximum de vraisemblance présenté à l'équation (26) pour estimer les paramètres. En d'autres mots, étant donné un ensemble d'entraînement de N images $\{I_1, I_2, ..., I_N\}$ absent d'objets en mouvement, on cherche pour chaque pixel le vecteur de paramètres $\vec{\theta}_s$ qui saura le mieux minimiser la fonction de risque empirique

$$R_{\text{emp}}(P(\vec{I}(s))|\vec{\theta}_s) = -\sum_{t=1}^{N} \log P(\vec{I}_t(s)|\vec{\theta}_s). \tag{40}$$

En résumé, la détection de mouvement peut être vue à la fois comme un problème de reconnaissance de formes (pour estimer X) et un problème d'estimation de densité (pour estimer $P(\vec{B}(s))$ à chaque pixel).

Malgré l'utilisation de plus en plus fréquente des approches probabilistes basées sur l'équation (38), ces dernières possèdent certains inconvénients souvent difficiles à contourner. Par exemple, apprendre des densités de probabilité à l'aide d'une série d'images $\{I_1, I_2, ..., I_N\}$ est un problème pour les applications ne disposant pas d'images absentes de mouvement. Au chapitre 3, nous démontrerons comment l'usage de contraintes spatiales locales peut permettre de réduire certaines limitations inhérentes aux solutions statistiques classiques de détection de mouvement. Nous montrerons comment ces contraintes peuvent servir à diminuer la quantité de mémoire utilisée, réduire les temps de calcul et simplifier les algorithmes d'apprentissage basés sur la fonction de risque empirique (40).

0.2.3 Détection d'occlusions

Le flux optique et la stéréovision sont deux applications en apparence différentes. Le flux optique a pour objectif d'estimer la vitesse de déplacement des pixels dans une séquence vidéo, alors que la stéréovision cherche à estimer la profondeur des objets d'une scène photographiée par deux caméras (parfois plus). Malgré leurs différences, ces deux applications cherchent à estimer une fonction dite de *corres-*

pondance. Tel qu'illustré à la figure 2.5 (b), cette fonction de correspondance a pour objet de relier ensemble les pixels des deux images. Pour le flux optique, la fonction de correspondance relie les pixels de l'image I_t à leur projection dans l'image I_{t+1}. Pour la stéréovision, la fonction de correspondance relie les pixels de l'image gauche à leur projection dans l'image droite. De façon générique, on appelle ces deux images : l'image de *référence* I^{Ref} et l'image de *matching* I^{Mat}. La fonction de correspondance reliant I^{Ref} à I^{Mat} porte souvent le nom de *carte de disparité d*. Une carte de disparité d est une fonction définie sur une grille rectangulaire S et dont chaque élément $\vec{d}(s)$ relie le pixel s de l'image I^{Ref} à sa correspondance s' dans l'image I^{Mat}. D'un point de vue général, comme en fait foi la figure 2.5 (b), il existe deux types de disparité : la disparité avant (*forward disparity*) et la disparité arrière (*backward disparity*). Ainsi, on appelle $\vec{d}^f(s)$ la disparité reliant $I^{\text{Ref}}(s)$ à $I^{\text{Mat}}(s')$ et $\vec{d}^b(s')$ la disparité reliant $I^{\text{Mat}}(s')$ à $I^{\text{Ref}}(s)$.

En général, une carte de disparité est estimée en minimisant une fonction d'énergie de la forme [45] :

$$d^f = \arg\min_d \sum_{s \in S} \left(\sum_{r \in \eta_s} \Gamma(I^{\text{Ref}}(r) - I^{\text{Mat}}(r + \vec{d}(s))) + \gamma \sum_{k \in \zeta_s} \Psi(|\vec{d}(s) - \vec{d}(k)|) \right) \quad (41)$$

où Γ et Ψ sont des fonctions de coût (souvent quadratiques) et γ est une constante. Malheureusement, estimer d^f (ou d^b) à l'aide de cette formule comporte deux limitations majeures. Tout d'abord, cette fonction est souvent mal-conditionnée dans les régions peu texturées. En d'autres mots, dans les régions peu texturées, plusieurs vecteurs $\vec{d}(s)$ peuvent avoir une énergie minimale alors qu'un seul n'est réellement valable. Dans ce cas, le minimum global n'est pas unique. Deuxièmement, estimer d^f à l'aide de l'équation (41) est problématique dans les zones dites *d'occlusion*. Ces zones sont des régions visibles dans une image, mais cachées dans la seconde. Les zones d'occlusion sont généralement engendrées par des objets en mouvement (dans le cas du flux optique) ou par l'effet de parallaxe (dans le cas de la stéréovision). N'étant visibles que d'une seule image, ces zones violent directement les conditions de l'équation (41).

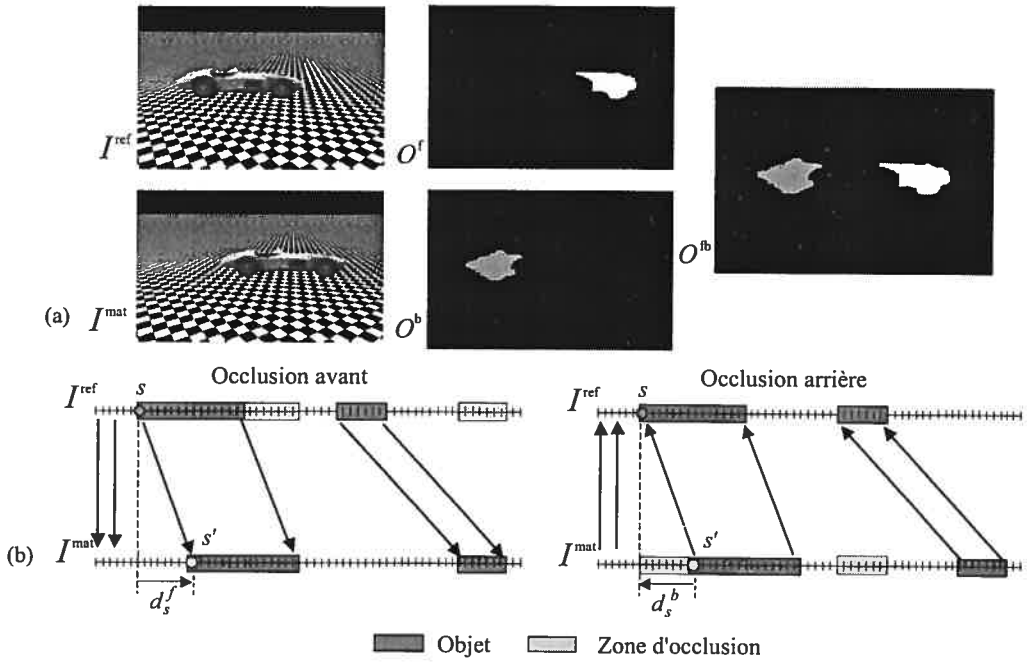

FIG. 6 – *(a) Exemple schématique de deux cartes d'occlusions : une carte avant O^f et une carte arrière O^b. (b) Exemple 1D illustrant le rôle des fonctions de disparité d^f et d^b.*

Comme son nom l'indique, une détection d'occlusions est une procédure ayant pour objectif de détecter les zones occluent, c'est-à-dire les zones vibles que d'une seule image. Pour y arriver, cette opération cherche à estimer une carte d'occlusions $O = \{O_s | s \in S\}$ par l'analyse des deux images I^{Ref} et I^{Mat}. Généralement, chaque élément O_s de la carte d'occlusions prend une valeur binaire $O_s \in \{\text{Non-Occlusion}, \text{Occlusion}\}$.

Comme le rapportent Egnal et Wildes [57], la détection d'occlusions est un problème généralement résolu à l'aide d'heuristiques simples. Citons en exemple les deux heuristiques les plus fréquemment utilisées dans le domaine : le test gauche-droite (*left-right check*) et le test de l'unicité (*uniqueness test*). Dans le premier cas, deux cartes de disparité sont estimées à l'aide de l'équation (41) : la carte avant d^f et la carte arrière d^b. En émettant l'hypothèse que d^f et d^b ne sont erronés que dans les zones d'occlusions, on peut estimer O à l'aide d'un simple seuil,

$$O_s = \begin{cases} 1 & \text{si } |s - d^b(s + d^f(s))| > \tau \\ 0 & \text{sinon.} \end{cases} \tag{42}$$

où τ est un seuil. En d'autre mots, pour le test gauche-droite, on émet l'hypothèse que la disparité $d^f(s)$ pointant au site s' (où $s' = s + d^f(s)$) doit être identique (à un signe près) à la disparité $d^b(s')$. En vertu de cet heuristique, seules les zones d'occlusions violent l'hypothèse gauche-droite. Pour ce qui est du test d'unicité, on suppose que chaque site s de l'image I^{Ref} doit être pointé par un et un seul vecteur $d^b(s')$. Si une région de I^{Ref} est pointée par un nombre anormalement restreint de vecteurs, c'est le signe que cette région est probablement occlue [142]. La détection d'occlusions par le test d'unicité se fait donc à l'aide d'un seuil appliqué sur la densité locale de vecteurs $\vec{d^b}(s)$. Il est à noter qu'à l'instar de la détection de mouvement, la détection d'occlusions peut être vue comme un problème de reconnaissance de formes.

Malheureusement, comme l'ont fait remarquer Egnal et Wildes [57], la plupart des méthodes d'estimation d'occlusions sont très sensibles au bruit ainsi qu'au

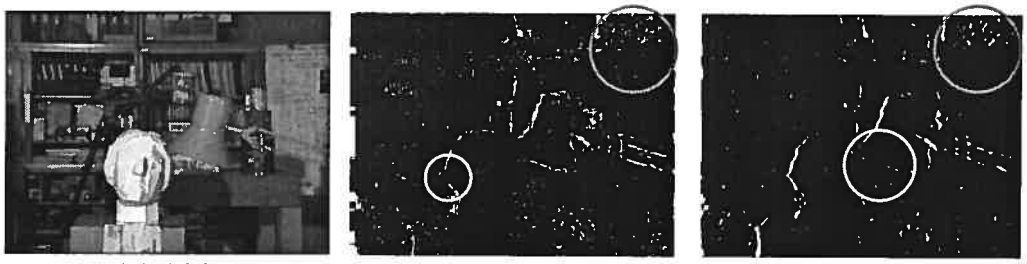

FIG. 7 – *De gauche à droite : une image de la séquence* TSUKUBA, *une carte d'occlusion obtenu par le test gauche-droite et une autre obtenue par le test d'unicité. Les zones vertes montrent des faux négatifs et les zones rouges des faux positifs.*

manque de texture (voir figure 7). En réponse à ces problèmes, au chapitre 2, nous proposons une procédure de fusion permettant de réduire considérablement l'effet du bruit et du manque de texture. Grâce à cette procédure de fusion, de nombreux faux positifs et faux négatifs sont éliminés. Cette procédure de fusion est conceptuellement simple, utilise au plus deux paramètres et peut être implémentée sur une architecture parallèle afin de permettre des traitements en temps-réel.

0.2.4 Segmentation de mouvement

L'objectif de la segmentation de mouvement est de regrouper ensemble les sites d'une scène ayant un déplacement uniforme. Tout algorithme de segmentation de mouvement prend donc en entrée une séquence vidéo I constituée d'une série d'images I_t définies sur une grille orthographique S. En sortie, l'algorithme retourne à chaque temps t, un champ d'étiquettes de déplacement x_t. Ce champ d'étiquettes est constitué de sites prenant une valeur parmi l'ensemble $\Gamma = \{\omega_1, \omega_2, ..., \omega_N\}$ où ω_i est une étiquette de déplacement. Ici, une étiquette de déplacement $x_t(s)$ indique à laquelle des N classes de mouvement appartient le site s. Mathématiquement, le champ d'étiquettes x_t est la réalisation de la variable aléatoire X_t distribuée suivant la densité de probabilité $P(X_t)$. Ainsi, tous les sites étiquetés ω_i ont un vecteur de

déplacement uniforme en vertu d'un modèle de déplacement prédéterminé A_{ω_i} :

$$\vec{v}(s) = \mathcal{X}_s \vec{A}_{\omega_i} \tag{43}$$

où $\mathcal{X}_s$ est une matrice liée à la position du site s. Par exemple, dans le cas d'un modèle de mouvement affine [5],

$$\begin{aligned} \vec{v}(s) &= \mathcal{X}_s \vec{A}_{\omega_i} \\ &= \begin{pmatrix} 1 & i & j & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & i & j \end{pmatrix} \begin{pmatrix} a_{i1} \\ a_{i2} \\ a_{i3} \\ a_{i4} \\ a_{i5} \\ a_{i6} \end{pmatrix} \end{aligned} \tag{44}$$

où (i, j) est la position euclidienne du site s sur la grille S, (a_{i1}, a_{i4}) sont les coefficients de translation et $(a_{i2}, a_{i3}, a_{i5}, a_{i6})$ sont les coefficients de rotation, changement d'échelle et cisaillement. La segmentation de mouvement est donc un processus ayant pour but d'estimer un champ d'étiquettes x_t ainsi que les paramètres de mouvement $\vec{A}_{\omega_i}$, tous deux initialement inconnus.

Il existe deux configurations classiques à la segmentation de mouvement. Le première est une configuration avec données *complètes* et la seconde avec données *incomplètes*. Pour ces deux configurations, des processus de reconnaissance de formes et de régression sont utilisées pour estimer conjointement x_t et $\vec{A}_{\omega_i}$.

Considérons tout d'abord la configuration avec données complètes. Pour cette configuration, l'application dispose en entrée d'une séquence vidéo I ainsi que d'un champ vectoriel V. Par conséquent, on cherche le *meilleur* champ d'étiquettes x_t ainsi que les meilleurs paramètres de mouvement $\vec{A}_{\omega_i}$ étant donnée I et V. On peut

formaliser ce critère à l'aide du *maximum a posteriori* tiré de l'équation (13),

$$\begin{aligned} \hat{x}_t &= \arg\max_{x_t} P(x_t|I,V) \\ &= \arg\max_{x_t} \frac{P(I,V|x_t)P(x_t)}{P(I,V)} \end{aligned} \tag{45}$$

ou, sans perte de généralité,

$$\hat{x}_t = \arg\max_{x_t} P(I,V|x_t)P(x_t)$$

car $P(I,V)$ ne dépend pas de x_t. Ici, $P(I,V|x_t)$ est la densité de probabilité de vraisemblance et $P(x_t)$ est la densité de probabilité *a priori*. En supposant que I, V et x_t sont des champs markoviens, suivant le théorème d'Hammersley-Clifford [71], $P(I,V|x_t)$ et $P(x_t)$ sont des densités de probabilité gibbsiennes de la forme : $\frac{1}{\lambda}\exp(-U)$, où U est un terme d'énergie et λ est un facteur de normalisation. En supposant aussi que I et V contiennent du bruit non corrélé, on peut affirmer que

$$\hat{x}_t = \arg\max_{x_t} \prod_{s\in S} P(I_t(s), \vec{v}(s)|x_t(s))P(x_t(s)) \tag{46}$$

$$= \arg\max_{x_t} \prod_{s\in S} \frac{1}{\lambda_s}\exp(-E(I_t(s), \vec{v}(s), x_t(s)) - \beta W(x_t(s))) \tag{47}$$

$$= \arg\min_{x_t} \sum_{s\in S} E(I_t(s), \vec{v}(s), x_t(s)) + \beta W(x_t(s)) \tag{48}$$

où $E(I_t(s), \vec{v}(s), x_t(s))$ l'énergie de vraisemblance, $W(x_t(s))$ est l'énergie *a priori* et β est une constante. Étant donnée que V et x_t sont liés suivant l'équation (44), on peut modéliser l'énergie de vraisemblance par

$$E(I_t(s), \vec{v}(s), x_t(s)) = \phi(||\vec{v}(s) - \mathcal{X}_s\vec{A}_{x_t(s)}||). \tag{49}$$

où ϕ est une fonction de coût, généralement quadratique [5, 31]. Comme cette dernière équation ne dépend pas de l'image I_t, on la redéfinie par $E(\vec{v}(s), x_t(s))$ pour alléger la notation. Pour ce qui est de l'énergie *a priori* $W(x_t(s))$, le modèle

de Potts est fréquemment retenu [5]. Ce modèle possède la forme,

$$U(x_t(s)) = \sum_{<r,s>} (1 - \delta(x_t(s), x_t(r))) \tag{50}$$

où δ est un delta de Kronecker et $< r, s >$ est l'ensemble des cliques binaires d'ordre deux. Comme en fait foi cette équation, le modèle de Potts est une fonction qui compte le nombre de site r dans le voisinage de s dont l'étiquette $x_t(r)$ est différente de $x_t(s)$. Ce modèle est donc identique au célèbre modèle d'Ising [141], à la différence que Potts accepte un nombre arbitraire d'étiquettes N. En combinant les équations (48), (49) et (50), on obtient que

$$\hat{x}_t = \arg\min_{x_t} \sum_{s \in S} \left(\underbrace{||\vec{v}(s) - \mathcal{X}_s \vec{A}_{x_t(s)}||^2}_{E(\vec{v}(s), x_t(s))} + \beta \underbrace{\sum_{<r,t>} (1 - \delta(x_t(s), x_t(r)))}_{W(x_t(s))} \right).$$

Évidemment, cette équation ne peut être résolue directement car $E(\vec{v}(s), x_t(s))$ dépend des vecteurs de paramètres $\vec{A}_{\omega_i}$, vecteurs initialement inconnus. Aussi, tel un cercle vicieux, l'estimation de $\vec{A}_{\omega_i}$ ne peut se faire que si x_t est donnée. Par conséquent, il faut utiliser un optimiseur qui sache estimer conjointement $\vec{A}_{\omega_i}$ et x_t. L'approche classique la plus fréquemment retenue consiste à estimer x_t et $\vec{A}_{\omega_i}$ en alternance [43, 110]. Pour ce faire, on estime tout d'abord les paramètres $\vec{A}_{\omega_i}$ pour chaque classe en fixant x_t. Par la suite, on estime x_t en fixant $\vec{A}_{\omega_i}$, et on répète ces deux étapes jusqu'à convergence. Cette optimisation en deux temps peut se faire à l'aide d'un estimateur déterministe [86, 110] ou stochastique [43] tel le recuit simulé ou encore l'algorithme Métropolis. Dans le cadre de cette thèse, nous avons implémenté au Chapitre 2, l'algorithme du recuit simulé que voici :

1. Initialiser x_t et $T = T_{\text{MAX}}$.
2. Mise à jour de $\vec{A}_{\omega_0}, \vec{A}_{\omega_1}, ..., \vec{A}_{\omega_{N-1}}$ suivant

$$\vec{A}_{\omega_i} = \arg\min_{\vec{A}} \sum_{\substack{s \in S \\ x_t(s)=\omega_i}} ||\vec{v}(s) - \mathcal{X}_s \vec{A}_{x_t(s)}||^2 \tag{51}$$

Cette équation peut être résolue à l'aide d'une minimisation par moindres carrés similaire à celle présentée à l'équation (19). Étant donné que $\vec{A}_{\omega_i}$ est estimé sur la base des pixels étiquetés $x_t(s) = \omega_i$, cette minimisation peut se faire rapidement à l'aide de l'inversion matricielle que voici :

$$\vec{A}_{\omega_i} = \left[\left(\sum_{\substack{s \in S \\ x_t(s)=\omega_i}} \mathcal{X}_s^{\mathrm{T}} \mathcal{X}_s \right)^{-1} \sum_{\substack{s \in S \\ x_t(s)=\omega_i}} \mathcal{X}_s^{\mathrm{T}} \vec{v}(s) \right]$$

3. Pour chaque site s, calculer la probabilité d'appartenance à chaque classe $\omega_i \in \Gamma$

$$\begin{aligned} U(x_t(s) = \omega_i | I) &= \sum_{s \in S} ||\vec{v}(s) - \mathcal{X}_s \vec{A}_{\omega_i}||^2 + \beta W(x_t(s)) \\ P(x_t(s) = \omega_i | I) &= \frac{1}{Z} \exp\left(-\frac{1}{T} U(\omega_i | I) \right) \end{aligned}$$

   et assigner une étiquette $\omega_i \in \Gamma$ à $x_t(s)$ suivant la probabilité $P(\omega_i|I)$.

4. $T = T \times$ Taux_De_Refroidissement

5. Reprendre les étapes 2, 3 et 4 jusqu'à ce que T atteigne la température T_{MIN}.

Effectuer une segmentation de mouvement à l'aide de cet optimiseur requiert donc l'emploi d'une régression par moindres carrés (étape 2) ainsi que d'une reconnaissance de formes de type maximum *a posteriori* (étape 3).

Considérons maintenant la configuration comportant des données incomplètes. Dans ce cas, l'algorithme ne dispose en entrée que d'une séquence vidéo, le champ vectoriel V étant inconnu au départ. Bien que l'objectif reste le même (estimer

conjointement x_t et $\vec{A}_{\omega_i}$ $\forall \omega_i \in \Gamma$) la fonction de coût à minimiser est différente. En raison de l'absence du champ vectoriel V, on ne peut utiliser l'équation (49) sans modification. En supposant que I respecte l'hypothèse de la constance de l'illumination (27), en combinant l'équation générale du mouvement (30) à l'équation (43), la fonction d'énergie de vraisemblance peut être redéfinie comme suit [5] :

$$E(I_t(s), x_t(s)) = \nabla I \left[\mathcal{X}_s \vec{A}_{x_t(s)}\right] + \partial_t I. \tag{52}$$

Suivant cette nouvelle formulation, l'estimation de $\vec{A}_{\omega_i}$ décrite à l'étape 2 du précédent algorithme s'exprime maintenant comme suit :

$$\vec{A}_{\omega_i} = \left[\left(\sum_{\substack{s \in S \\ x_t(s)=\omega_i}} (\nabla I \mathcal{X}_s)^{\mathrm{T}} (\nabla I \mathcal{X}_s) \right)^{-1} \sum_{\substack{s \in S \\ x_t(s)=\omega_i}} (\nabla I \mathcal{X}_s)^{\mathrm{T}} \partial_t I \right].$$

De même, l'équation de l'énergie globale $U(x_t(s) = \omega_i | I)$ utilisée à l'étape 3, doit être redéfinie comme suit,

$$U(x_t(s) = \omega_i | I) \;=\; \sum_{s \in S} \left(\nabla I \left[\mathcal{X}_s \vec{A}_{\omega_i}\right] + \partial_t I \right)^2 + \beta W(x_(s)).$$

Avec ces nouvelles fonctions d'énergie, le champ d'étiquettes x_t ainsi que les paramètres de mouvement $\vec{A}_{\omega_i}$ peuvent être estimés à l'aide des mêmes cinq étapes de l'algorithme de recuit simulé.

Malgré leurs différences, ces deux approches ont un point en commun : ce sont des approches dites *à base d'énergie*. Dans les deux cas, la probabilité de vraisemblance $P(I, V | x_t)$ est définie par l'exponentielle d'une fonction d'énergie *ad hoc*, tirée des équations (49) et (52). Il est cependant possible de segmenter une séquence vidéo à l'aide d'une approche probabiliste. Par cette approche, on modélise $P(I, V | x_t)$ avec une «vraie» densité de probabilité telle une gaussienne, une Student ou une distribution binomiale par exemple [23, 130]. Cette approche probabiliste est utilisée au chapitre 2 ainsi qu'au chapitre 4. Pour la segmentation probabiliste,

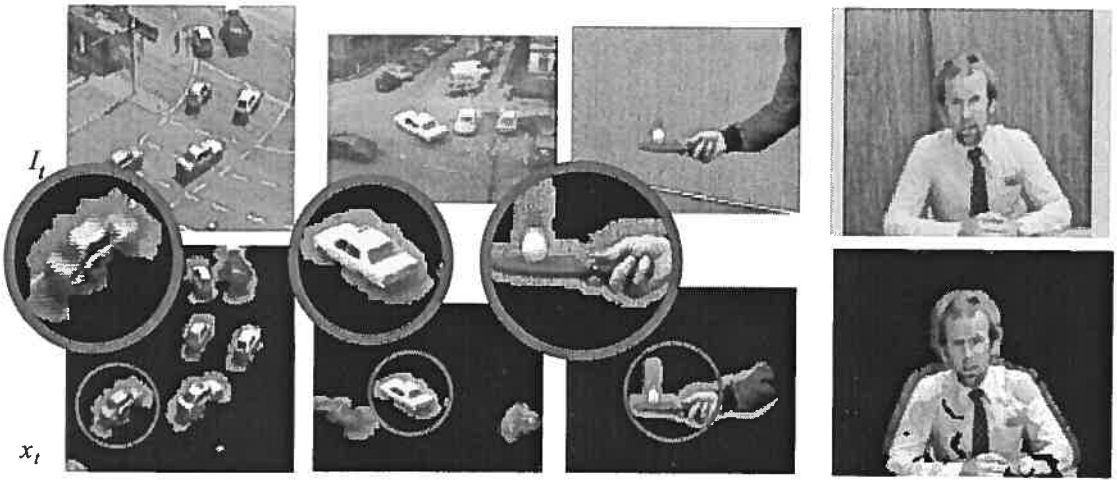

FIG. 8 – *Exemple des séquences vidéo* KARLSHRUE, TAXI, TENNIS *et* TREVOR WHITE *segmentées à l'aide d'une approche probabiliste.*

nous supposons une configuration avec données complètes. Ainsi, on modélise la distribution des vecteurs de chaque classe ω_i à l'aide d'une densité de probabilité $P(I, \vec{v}(s)|\omega_i)$. En l'absence de toute connaissance *a priori* quant à la nature de la distribution réelle des vecteurs de chaque classe, on modélise $P(I, \vec{v}(s)|\omega_i)$ par une loi gaussienne :

$$P(I, \vec{v}(s)|\omega_i) = \frac{1}{2\pi|\Sigma_s|^{1/2}} \exp\left(-\frac{1}{2}(\vec{v}(s) - \vec{\mu}_{\omega_i})^{\mathrm{T}}\Sigma_{\omega_i}^{-1}(\vec{v}(s) - \vec{\mu}_{\omega_i})\right) \qquad (53)$$

où $\vec{\mu}_{\omega_i}$ est le vecteur de déplacement moyen de la classe ω_i et Σ_{ω_i} est la matrice de covariance. À noter que l'équation (53) émet implicitement l'hypothèse que chaque classe suit un modèle de déplacement translationnel. Pour segmenter une séquence vidéo à l'aide d'une approche probabiliste, deux algorithmes doivent être implémentés : un algorithme d'estimation de paramètres et un algorithme de segmentation. Comme son nom l'indique, le premier algorithme sert à estimer les paramètres gaussiens $\vec{\mu}_{\omega_i}$ et Σ_{ω_i} de chaque classe. Cette étape peut se faire à l'aide d'un algorithme du type K-Moyennes, soft K-Moyennes, Estimation Maximization (EM), Stocahstic Estimation Maximization (SEM), Iterative Conditional

Estimation (ICE) ou tout autre algorithme équivalent. Aux chapitres 2 et 4, nous utilisons l'algorithme ICE [160] que nous initialisons à l'aide de l'algorithme des K-Moyennes [23]. À noter que l'ICE est une version stochastique et Markovienne du célèbre algorithme EM [130]. Une fois l'ensemble des paramètres $\vec{\mu}_{\omega_i}$ et Σ_{ω_i} estimé, le champ d'étiquettes x_t peut être inféré à l'aide du maximum a posteriori de l'équation (46). Cette deuxième étape peut se faire à l'aide d'un optimiseur stochastique tel le recuit simulé [141] ou encore un optimiseur déterministe tel l'ICM de Besag [71]. Ces deux algorithmes (ainsi que l'ICM et le K-Moyennes) seront décrits plus précisément au chapitre 4.

Malgré les différences entre l'approche probabiliste et celles à base d'énergie, les résultats obtenus sont visuellement très similaires. En fait, comme l'illustre la figure 8, le résultat d'une segmentation de mouvement comporte généralement des régions floues englobant plus ou moins bien les objets en mouvement. L'imprécision dans les résultats est généralement causée par un manque local de texture (par exemple, les taches noires sur la chemise de Trevor White à la figure 8) et par les gradients spatiaux (∇I) et temporels ($\partial_t I$). En fait, comme ces deux opérateurs gradient sont isotropiques, ils ont pour effet d'ajouter du flou dans la scène, un flou causant ces halos autour des objets dans x_t.

Au chapitre 2, nous proposons un algorithme de fusion ayant pour objectif de contraindre les régions du champ d'étiquettes x_t à épouser la forme des objets en mouvement. Cette fusion prend en entrée deux champs d'étiquettes : x_t et r une carte de régions contenant la forme des objets présents dans la scène. Ces champs sont combinés via une fonction de coût inspirée du modèle *a priori* de Potts. La minimisation se fait à l'aide de l'algorithme déterministe ICM.

0.2.5 Calculs temps réel

Les outils mathématiques issus de la théorie de l'apprentissage statistique ont maintes fois fait leur preuve au cours des dernières décennies. Toutefois, ces derniers ne sont pas toujours applicables, souvent en raison de la puissance de calcul qu'ils requièrent parfois. Par exemple, la segmentation de mouvement telle que définie

à la section 0.2.4, qu'elle soit probabiliste ou à base d'énergie, exige plusieurs secondes sinon plusieurs minutes pour inférer un seul champ d'étiquettes x_t. Par conséquent, en raison de ces temps de calcul prohibitifs, les ingénieurs rechignent à l'idée d'utiliser ces outils.

Afin de compenser pour ces limitations, au chapitre 4, nous démontrons comment des méthodes Markoviennes d'estimation de densité et de reconnaissance de formes peuvent fonctionner en temps réel lorsqu'implémentées sur une architecture parallèle. L'architecture retenue est celle disponible sur les cartes graphiques aujourd'hui disponibles sur le marché, à savoir les *Graphics Processor Units (GPU)*. Nous nous concentrerons sur les méthodes de segmentation et d'apprentissage statistiques appliquées à la stéréovision, à l'estimation de mouvement, à la segmentation de mouvement ainsi qu'à la segmentation de couleur. Tous les algorithmes présentés utilisent le processeur de *fragments* du GPU ainsi que la mémoire texture pour stocker les images à segmenter et les champs d'étiquettes. Les résultats obtenus démontrent qu'il est possible d'atteindre des taux d'accélérations allant de 4 à 200 et ce, sans aptitude particulière en programmation de bas-niveau.

CHAPITRE 1

ESTIMATION DU FLUX OPTIQUE BASÉE SUR UNE PROCÉDURE D'ÉVITEMENT DES CONTOURS

Article : Optical-Flow Based on an Edge-Avoidance Procedure

Cet article a été accepté avec modifications mineures au journal *Journal of Computer Vision and Image Understanding* comme l'indique la référence bibliographique.

P-M Jodoin M. Mignotte Optical-Flow Based on an Edge-Avoidance Procedure *Journal of Computer Vision and Image Understanding*, accepté, 2006.

Cet article est présenté ici dans sa version originale.

Résumé

Dans ce chapitre, nous proposons une solution à deux problèmes classiques en estimation de mouvement : (1) la régularisation du flux à l'intérieur de régions ayant un mouvement uniforme et (2) la préservation de contours bien définis autour des objets en mouvement. Pour ce faire, nous proposons une version modifiée du célèbre algorithme de Lucas et Kanade (LK). Comme nous l'avons souligné au chapitre précédent, cet algorithme fonctionne à l'aide d'une régression par la somme de moindres carrés. Par conséquent, il réagit difficilement dans les régions peu texturées ainsi que dans les régions dont le mouvement est discontinu. Plus spécifiquement, dans les régions de mouvement discontinu, l'algorithme de LK a tendance à brouiller les contours des objets en mouvement. La plupart des méthodes cherchant à préserver des contours nets autour des objets en mouvement utilisent différentes fonctions robustes afin de minimiser l'influence des valeurs extrêmes (les *outliers* en anglais). La méthode que nous proposons dans ce chapitre adopte une nouvelle façon de gérer les valeurs extrêmes. Tout comme pour la méthode de Lucas-Kanade, notre approche estime le flux optique à l'aide d'une somme par moindre

carrés. Toutefois, à l'aide d'une procédure non paramétrique nommée *mean-shift*, les voisinages locaux sont décalés en direction des régions dont le mouvement est plus vraisemblablement unimodal que multimodal. Cela a pour effet d'exclure les *outliers* des calculs. De plus, notre méthode régularise localement le champ vectoriel à l'aide d'un filtre covariant ayant pour effet de minimiser l'incertitude dans les régions peu texturées. Les résultats obtenus avec notre méthode se comparent avantageusement à ceux obtenus à l'aide d'autres méthodes fréquemment utilisées dans le domaine du flux optique.

Abstract This paper presents a differential optical flow method which accounts for two typical motion-estimation problems : (1) flow regularization within regions of uniform motion while (2) preserving sharp edges near motion discontinuities *i.e.*, where motion is multimodal by nature. The method proposed is a modified version of the well known Lucas Kanade (LK) algorithm. While many edge-preserving strategies seek to minimize the effect of *outliers* by using a line process or robust functions, our method takes a novel approach to solve the problem. Based on specified assumptions, our method computes motion with a classical least-square fit on a local neighborhood shifted away from where motion is likely to be multimodal. In this way, the inherent bias due to multiple motion around moving edges is avoided instead of being compensated. This edge-avoidance procedure is based on a non-parametric mean-shift algorithm that shifts the LK integration window away from local sharp edges. Our method also locally regularizes motion by performing a fusion of local motion estimates. The regularization is done with a covariance filter that minimizes the effect of uncertainties due in part to noise and/or lack of texture. Our method is compared with other edge-preserving methods on image sequences representing various challenges.

1.1 Introduction

In the past 30 years, a numerous solutions have been proposed to solve the optical flow problem [13, 64, 70]. As observed by Barron *et al.* [70], optical flow techniques can be divided into families among which are the phase-based methods [39], spectral-based methods [107], energy-based methods [17, 18, 115], Markovian methods [49, 54, 75]), and differential methods [21, 22, 62]. Choosing one approach *versus* another depends very much on the application and the nature of the flow to be estimated. For instance, a densely cluttered scene with different global movements might be amenable to a spectral-based method, whereas a differential method might be better suited for a scene exhibiting objects (such as cars for instance) moving in front of a fixed background. These techniques are various and adapted to all kinds of situations. However, most of them are expressed as global optimization problems involving a *data conservation constraint* likelihood term and a *spatial coherence constraint* prior term (also called *regularization* term) [102].

The data conservation constraint is generally built upon the *brightness constancy assumption* which stipulates that the brightness of a single point remains constant with time. This simple assumption is frequently used to develop simple optical flow methods that generate fairly good results. However, as one might expect, the brightness constancy assumption is only valid for ideal noise-free scenes and thus almost never holds exactly. Furthermore, due to lack of texture and occlusion, different motions within a local area can be indistinguishable even though only one is valid. In other words, the brightness constancy assumption only partially constrains the data leaving the problem ill-posed since this assumption allows several solutions as being "optimal", although only one is objectively correct. This problem is well documented and referred to as the *aperture problem* [13, 102].

To obtain accurate estimates, a *spatial coherence constraint* has to be added to the first constraint. While some authors see this additional constraint as an *integration window* [22, 51, 87], others see it as a *prior function* [21, 75] modeling the way motion vectors should be distributed in the final vector field. The choice of

spatial constraint is often the key element differentiating one approach from another. Typically, these spatial constraints assume that motion is locally uniform and changes smoothly across regions of the scene. Unfortunately, this isotropic assumption is violated when a region spans a motion discontinuity [102,152]. As explained by Black and Anandan [102], the spatial neighborhood must be large enough to sufficiently constrain the solution but also small enough to avoid spanning multiple motions.

Many popular motion estimation approaches were built upon these competing constraints [21,22,115], which often ignore the *multimodal* nature of motion near moving edges. As a result, the estimated motion is often imprecise and blurry in these areas. To gain more accuracy and better preserve motion discontinuities, several solutions have been proposed. As mentioned by Thompson [152], the optical flow literature proposes two broad families of solutions designed to preserve sharp motion edges : the *flow-based family* and the *image-based family.*

Flow-based methods

The flow-based methods typically allow for mixed motion distributions near boundaries. In fact, these methods assume that the imprecision around moving edges is due to the presence of "outliers" pooling from a spatial neighborhood and corrupting the final solution. Consequently, this family of solutions aim at minimizing the influence of these undesired values.

One popular way for dealing with outliers is to use a *robust* error function that give a relative influence to outliers, preventing them from corrupting the final solution. Black and Anandan [101,102] (followed soon after by Odobez and Bouthemy [85]) were the first to explicitly use robust functions to minimize the effect of those measures violating the data conservation or the spatial coherence constraint. They adapted their robust framework to two common motion estimation techniques : the recovery of multiple parametric motion models and the recovery of piecewise-smooth flow fields. More recently, Aubert *et al.* [56] presented a variational technique whose convergence was carefully demonstrated. Their method is

presented as a differential method similar to Horn and Schunck [21] but with an edge preserving regularization term and a half-quadratic optimizer. Weickert [80] also propose a variationnal technique implementing a robust isotropic regularization term. The use of such regularization term prevent the method from smoothing *accross* motion discontinuity. Shortly after, Weickert and Schnörr [82] proposed another variational method that could be seen as an extension of Weickert's technique [80]. In their paper, they propose an anisotropic flow-driven regularization technique which not only prevents from smoothing *across* flow discontinuity, but also encourages smoothing *along* flow edges. This method implements a diffusion tensor embedded in a diffusion-reaction system minimized with a downhill search technique. More recently, Brox *et al.* [150] proposed a variational technique minimizing an energy function composed of three terms : a brightness consistency term, a novel gradient constancy term and a robust spatio-temporal smoothness term.

Another class of flow-based optical flow methods are those implementing a *least-median-of-squares* technique. As opposed to the robust methods which minimize the influence of the outlying measures, these methods explicitely detect and reject outliers. The retained measures (called *inliers*) are then used to estimate the flow. Bab-Hadiashar and Suter [3] and Ong and Spann [50] proposed interesting work in that field.

Other flow-based methods consider motion vectors as "estimates" that are to be fused together. During the fusion process, each estimate is given an importance value which make it more or less influent locally. In this way, a region with a high confidence will propagate more information than a region with a low confidence. An early paper in that field has been published by Singh [17] in 1990. In this work, Singh uses a multiresolution-two-step sum of squared differences (SSD) procedure to help preserve flow discontinuity without any prior knowledge on the location of the boundary. In a first step, Singh's method estimates a variance-covariance matrix based on a SSD measure over a correlation window. This matrix is then used as a confidence measure to propagate the flow using neighborhood information. More recently, Comaniciu [36] presented a non-parametric fusion approach that appears

as an improvement of Singh's method [17]. Comaniciu's method pre-estimates the flow with a simple biased-least-squares method before locally fusing the motion vectors with a multiscale mean-shift procedure.

Let us also mention the work by Farneback [58] in which 3D orientation tensors are combined to affine parametric motion models for the estimation of the optical flow. Farneback shows that the fundamental relations between the 3D orientation tensors and optical motion makes his method a good solution for quickly estimating precise motion fields. The author argue that this approach can be combined with a region-growing segmentation algorithm to sharpen the results [58].

Image-Based Methods

The second class of edge-preserving optical flow methods are those which explicitly try to locate motion discontinuities [152]. These methods often assume that e flow boundary always corresponds to an intensity edge. Based on this assumption, most image-based methods make sure that measures gathered from opposite side of an intensity edge never influence each other. In that perspective, several approaches coping with motion discontinuities use the Markov Random Field (MRF) formulation inspired by Geman and Geman's work [141]. Most of these MRF methods rely on a *line process* to estimate border locations and keep points from opposite sides of the border from influencing each other [54, 75]. In particular, this is what Black [99] does when he jointly estimates a motion vector field together with a motion region map. To do so, this method minimizes an energy-based Markovian function made up of an intensity model, a border model and a motion model. Local constraints on motion and intensity allow to preserve sharp flow discontinuities.

Other image-based methods estimate motion based on a pre-estimated segmentation map. These methods make the underlying assumption that a motion discontinuity cannot span a region of uniform intensity. In that perspective, a color region map is pre-estimated and the optical flow is computed within the color regions. Among the first contribution in this area was the work by Fuh and Maragos [26] who proposed a region-based matching method combined with a post-

processing median filtering. A few years later, Meyer and Bouthemy [55], Dang *et al.* [153], and Black and Jepson [100] proposed methods to fit affine models within pre-estimated regions of uniform intensity. In the latter paper, the method has been made iterative to better refine the flow.

Many image-based variationnal techniques have also been proposed. These methods can be seen as extensions of the original Horn and Schnunk's method [21]. These variational methods use a regularization term to smooth out the flow based on the image content. For instance, in their 1999 paper, Alvarez *et al.* [92] proposed a variational method that reduces its regularization in the vicinity of every image edges. This is done with a non-linear regularizer inversely proportional to the image gradient. Such an approach is sometimes reffereed to as an *image-based isotropic* variational approach [82]. Another class of variational techniques comprises the *image-based anisotropic* methods [82]. These methods implement a regularization term that both prevents smoothing flow across image edges and encourage smoothing along image edges. Anisotropic image-based methods were proposed by Nagel-Enkelmann [65], Nagel [62], Schnörr [28], and more recently, Alvarez *et al.* [93].

Our Method

In this paper, a modification to the well known Lucas-Kanade (LK) algorithm [22] is proposed. The objective of our method is twofold : (1) minimize uncertainties (often caused by noise and lack of texture) by strongly constraining the flow within regions of uniform movement while (2) preserving flow discontinuities around moving objects. Since our method is based on a least-squares fit (and thus is sensitive to multimodal motion) the key idea is to avoid computing flow in areas where motion is likely to be multimodal. Following the same assumption made by most image-based methods, our approach assumes that every motion boundary correspond to an intensity edge. More specifically, in areas near a strong intensity gradient, our algorithm computes motion with a neighborhood window shifted away from the nearest intensity edge. In this way, the flow is computed

over sections of the scene where motion is likely to be unimodal. Our least-squares fitting algorithm can thus preserve sharp motion boundaries by *avoiding* having to deal with multiple motions. To our knowledge, such an avoidance procedure has never been investigated before. Also, to better constrain the solution, our method implements a fusion procedure similar to the flow-based method proposed by Singh [17]. This fusion procedure implements a covariance filter that locally averages motion. In the experimental section, our method is compared with other techniques whose purpose is also to preserve motion discontinuities. Results are obtained after processing synthetic, realistic, and real sequences.

The remainder of this paper is organized as follows. Because our method is a modified version of LK, an introduction of the LK method [22] is first presented in section 1.2. Section 1.3 then presents our modifications to LK, which includes a covariance filter and an edge-avoidance procedure based on the mean-shift [37] algorithm. Section 1.5 then presents the optical flow methods we compare our method to. This section also includes results obtained on various synthetic, realistic and real image sequences. Section 1.6 discusses our method and concludes.

1.2 Lucas-Kanade Motion Estimation

The Lucas-Kanade (LK) approach was first introduced as a least-squares fitting method applied to stereovision [22]. However, its extension to motion estimation is trivial and goes as follows. Let $S = \{s = (i,j) | i \in [0,\mathcal{N}[, j \in [0,\mathcal{M}[\}$ denote a 2D lattice of size $\mathcal{N} \times \mathcal{M}$ and $I(s,t)$ the intensity of the site s at time t. In our implementation, $I(s,t)$ takes a value between 0 and 255. Considering the brightness constancy assumption and assuming that the velocity is locally linear, LK looks for a vector field $V = \{\vec{v}_s = (u_s, v_s) | s \in S, \vec{v}_s \in \mathbb{R}^2\}$ that minimizes the residual quadratic error

$$E(\vec{v}_s) = \sum_{r \in \eta_s} [I(r,t) - I(r + \vec{v}_s, t+1)]^2, \quad \forall s \in S \tag{1.1}$$

where η_s is a neighborhood window of size $N \times N$ centered on site s. $E(\vec{v}_s)$ can be reformulated based on its Taylor expansion :

$$E(\vec{v}_s) \approx \sum_{r \in \eta_s} [u_s I_x + v_s I_y + I_T]^2 \tag{1.2}$$

$$= \sum_{r \in \eta_s} [\nabla I^{\mathrm{T}} \vec{v}_s + I_T]^2 \tag{1.3}$$

where I_x,I_y and I_T are respectively the spatial and temporal derivatives over site r at time t [51,52]. As mentioned by Lucas and Kanade, this quadratic error function can be minimized by setting its first derivative to zero : $\frac{\partial E(\vec{v})}{\partial \vec{v}} = 0$, which formally corresponds to

$$\sum_{r \in \eta_s} \nabla I (\nabla I^{\mathrm{T}} \vec{v}_s + I_T) = 0 \tag{1.4}$$

or, equivalently,

$$\left[\begin{pmatrix} \sum_r I_x^2 & \sum_r I_x I_y \\ \sum_r I_x I_y & \sum_r I_y^2 \end{pmatrix} \vec{v}_s + \begin{pmatrix} \sum_r I_T I_x \\ \sum_r I_T I_y \end{pmatrix} \right] = 0. \tag{1.5}$$

Although the assumption that velocity is locally linear is true in regions of constant flow, it can be a problem when N is large and/or when η_s spans motion discontinuities. Thus, to minimize the influence of outliers, many authors add a weighting term W_i which gives more influence to constraints that are close of the center of η_s than those at the periphery [70]. This added term is mathematically expressed as

$$\left[\begin{pmatrix} \sum_r W_r I_x^2 & \sum_r W_r I_x I_y \\ \sum_r W_r I_x I_y & \sum_r W_r I_y^2 \end{pmatrix} \vec{v}_s + \begin{pmatrix} \sum_r W_r I_T I_x \\ \sum_r W_r I_T I_y \end{pmatrix} \right] \tag{1.6}$$

where W typically contains Gaussian isotropic values. However, since the flow in our method is already constrained by a fusion procedure (see section 1.3), W is

assigned spatial gradient data ($W_r = ||\nabla I_r||^2$) in such a way that more influence is given to those sites located in textured areas.

To simplify the notation, it is common to rewrite equation (1.6) as

$$M_s \vec{v}_s + \vec{b}_s = 0 \tag{1.7}$$

where M_s is the 2×2 matrix in Eq.(1.6) and $\vec{b}_s$ is the $2D$ vector in the same equation. Following Eq.(1.7), the least-squares solution can be obtained after a simple matrix inversion

$$\vec{v}_s = -M_s^{-1} b_s. \tag{1.8}$$

Of course, LK provides only a solution to those sites $s \in S$ for which M_s is not singular. This singularity problem has first been addressed by Nagel [61] and then by Barron *et al.* [70]. In the latter paper, a simple solution is proposed : reject every unreliable estimate v_s for which the eigenvalues λ_1 and λ_2 of M_s, are below a given threshold τ. Although this approach is intuitively acceptable, it allows flow fields of density much lower that 100% (35% for the YOSEMITE sequence and 39% for the TRANSLATING TREE sequence [70]). Other authors [36,51] proposed adding a bias to M_s to ensure is invertibility. Mathematically, this can be formulated as

$$(M_s + \beta I_d)\vec{v}_s + \vec{b}_s = 0 \tag{1.9}$$

where I_d is the identity matrix and β may be a constant [36] or proportional to the covariance of the noise [51]. In this paper, we use a bias that is different from the one of Eq. (1.9). As shown in Algorithm 2, the bias we use is a small random white noise added to the input image sequence. This noise adds a bias to the spatial gradient (I_x, I_y) but doesn't affect the temporal gradient I_T. Thus, our bias makes M_s invertible everywhere without adding any significant error to the results, as can be seen in Fig.1.16 and 1.17.

Optical flow obtained with Eq. (1.8) (or Eq. (1.9)) is often considered as being

the standard LK solution. In practice, however, to gain more accuracy, it may be good idea to implement a Newton-Raphson-like iterative version of this scheme [87]. Furthermore, to better cope with large displacements, we implemented LK in a multiresolution framework.

Iterative LK

In this section, we present an iterative version of the traditional LK method, primarily inspired of the work by Bouguet [87] and Black and Anandan [102]. Let $\vec{v}_s^k$ be the motion vector on site s after k-1 iterations and $\Delta\vec{v}_s^k$ the incremental motion vector computed during the k^{th} iteration. Here, the goal is to estimate the $\Delta\vec{v}_s^k$ that will best minimize the residual error

$$E(\vec{v}_s^{k+1}) = \sum_{r\in\eta_s}[I(r+\vec{v}_s^k+\Delta\vec{v}_s^k,t) - I(r,t+1)]. \tag{1.10}$$

According to Eq. (1.8), the k^{th} motion increment can be computed byA

$$\Delta\vec{v}_s^k = -(M_s)^{-1}\vec{b}_s^k \tag{1.11}$$

where

$$\begin{aligned} \vec{b}_s^k &= \begin{pmatrix} \sum_r W_r I_T^k I_x \\ \sum_r W_r I_T^k I_y \end{pmatrix} \\ I_T^k &= I(r+\vec{v}_s^k,t) - I(r,t+1) \end{aligned}$$

where value $I(r+\vec{v}_s^k,t)$ is computed with a bilinear interpolation. After k iterations, the motion vector on site s is given by

$$\vec{v}_s^{k+1} = \vec{v}_s^k + \Delta\vec{v}_s^k \tag{1.12}$$

or equivalently

$$\vec{v}_s^{k+1} = \vec{v}_s^k - M_s^{-1}\vec{b}_s^k. \tag{1.13}$$

Multiresolution LK

As mentioned previously, a multiresolution framework is often a good solution to deal with large displacements. Its is also a good solution to help regularize the flow in noisy and/or textureless areas. When implementing a multiresolution framework, two pyramids based on $I(t)$ and $I(t+1)$ first need to be built. The process of building an image pyramid is often refereed to as "image decimation" [5, 106]. In most pyramid representation, the original image (of size $\mathcal{N} \times M$) appears at the bottom of the pyramid, *i.e.* at level 0. The original image is first convoluted by a low-pass filter and then decimated by a factor of two in each dimension. The resulting image (of size $\mathcal{N}/2 \times M/2$) is then placed at the level 1 of the pyramid. The same two operations are then applied to the image at level 1 to produce the next $\mathcal{N}/4 \times M/4$ pyramid level. This process is repeated up until when the desired number of pyramid level is reached. In our application, as shown in Figure 1.1 (a), a 3×3 low-pass Bartlett filter [106] is used. This filter is also called *pyramidal* or *weighted average* by some authors [139]. Since each level I^L has a size of $\mathcal{N}/2^L \times \mathcal{M}/2^L$, the pyramids can have up to $\min[\log_2(\mathcal{N}), \log_2(\mathcal{M})]$ levels. However, our tests revealed that more than four levels does not provide any major advantage, at least for those sequences we worked with for which motion never exceeds 5 pixels.

Once the two input pyramids have been computed, the flow is estimated from the highest level of the pyramid down to level 0. At each level, vector field V^L is iteratively estimated with Eq. (1.13) after which it is projected down to level $L-1$. The downscaling operation is done with a bilinear interpolation as illustrated in Figure 1.1 (b). The iterative and multiresolution version of LK is presented in the Algorithm 1. Notice that at the highest level, the flow is initialized with zero values.

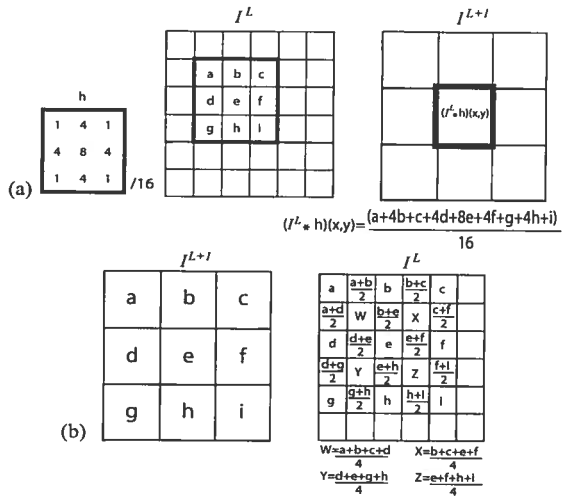

FIG. 1.1 – (a) The pyramid level I^{L+1} is obtained after convoluting I^L with h and decimating I^L by a factor 2 in each dimension. (b) When projecting a level $L+1$ down to level L, a simple bilinear interpolation method is used.

1.3 Our Method

The LK algorithm is simple and generates fairly good results over a wide range of real image sequences, hence its popularity. However, it is widely accepted that its implementation suffers from two fundamental limitations. First, since Eq. (1.13) (and (1.8)) do not model the inherent uncertainties caused by noise and low contrast regions [51], the output vector field V may be locally inaccurate, especially if the neighborhood η_s is small. Second, the LK solution hardly takes account of multiple motions and thus generates blurry edges around moving objects. This is especially true when LK is implemented with a multiresolution framework.

1.3.1 Dealing with Uncertainties

Several solutions are conceivable to alleviate the problem of uncertainties. While some have adapted Eq. (1.13) to a probabilistic framework [51] to account for noise, others have replaced V by a piecewise-smooth vector field made up of parametric models [31,73,78]. Also, a variety of filters has been proposed, from simple median

Algorithm 1 : Multiresolution and Iterative LK Algorithm

$I(s,t)$	Image at time t
V	Vector field
L	Pyramid-level index
N_i	Number of iterations

Build a n-level pyramid for $I(t)$ and $I(t+1)$.
$V \leftarrow 0$
For $L = Level_{Max}$ to 0 **do**
    compute I_x^L and I_y^L

    For $k = 0$ to N_i **do**
        For each site $s \in S^L$ **do**
            compute $(M_s^L)^{-1}$
            $I_T^{L^k} \leftarrow I^L(s + \vec{v}_s^k, t) - I^L(s, t+1)$
            Compute $\vec{b}_s^{L^k}$
            $\vec{v}_s^{k+1} \leftarrow \vec{v}_s^k - M_s^{L^{-1}} \vec{b}_s^k$

    If $L > 0$ **then**
        $V^{L-1} \leftarrow$ downscale V^L

filters [73] to more elaborate Kalman-like filters [18, 36].

For our method, every vector $\vec{v}_s$ is considered as an "estimate" that is to be fused locally with its neighbors to yield a better result [36]. Assuming that $\vec{v}_s$ has a 2×2 covariance matrix C_s proportional to the variance of the noise, the vectors surrounding site s can be fused with a linear combination [18, 36] of the form

$$\vec{v}_s = P_s \sum_{i \in \zeta_s} C_i^{-1} \vec{v}_i \qquad (1.14)$$
$$P_s = (\sum_{i \in \zeta_s} C_i^{-1})^{-1}$$

where ζ_s is a neighborhood window of size $\mathcal{X} \times \mathcal{X}$ around site s. Here, $C_i^{[-1]}$ can be seen as a "confidence measure" that give more or less influence to an estimate $\vec{v}_i$.

In this way, a vector $\vec{v}_i$ with a large confidence will influence more $\vec{v}_s$ than another one with a smaller confidence. Here, C_i is a 2×2 covariance matrix computed as follows :

$$C_s = \left(\begin{array}{cc} \sum_{t \in \beta_s} (u_t - u_s)^2 & \sum_{t \in \beta_s} (u_t - u_s)(v_t - v_s) \\ \sum_{t \in \beta_s} (u_t - u_s)(v_t - v_s) & \sum_{t \in \beta_s} (v_t - v_s)^2 \end{array} \right)_{/\Lambda * \Lambda} \tag{1.15}$$

where β_s is a neighborhood window of size $\Lambda \times \Lambda$ around s and $\vec{v}_t = (u_t, v_t)$. This matrix is similar to the one proposed by Singh [17]. Notice that such fusion procedure is sometimes refereed to as a *Best Linear Unbiased Estimate* (BLUE) by some authors [36,163]. Combining Eq. (1.13) and (1.14), the iterative LK procedure can be rewritten as

$$\vec{v}_s^{k+1} = P_s^k \sum_{i \in \zeta_s} C_i^{k^{-1}} (\vec{v}_i^k + M_i^{-1} \vec{b}_i^k). \tag{1.16}$$

This scheme makes sense intuitively since it encourages flow to propagate from high-confidence regions (regions with low covariance) to regions of low-confidence. To make sure C_i is invertible, its eigenvalues are forced to be larger or equal to 0.001 (here in pixel-distance units). This is done *via* a singular value decomposition.

Of course, for those applications for which processing time is a determining factor, the fusion procedure of Eq.(1.16) may be replaced by a simple, but faster, Gaussian filtering. However, it should be understood that such filtering may sometimes be prone to propagate erroneous flow.

1.3.2 Dealing with Multimodal Motion

As mentioned before, estimating motion with an isotropic approach such as LK often results in a flow with blurry edges, which explains why so many robust functions and anisotropic filters have been proposed so far. In this contribution, the way multiple motion is handled is based on the following four assumptions :

1. moving objects are textured enough to have their motion correctly estimated by a differential method ;

2. motion boundaries are close to sharp intensity edges;
3. in regions containing no sharp intensity edge, motion is locally invariant;
4. motion estimated away from flow discontinuities is reasonably accurate.

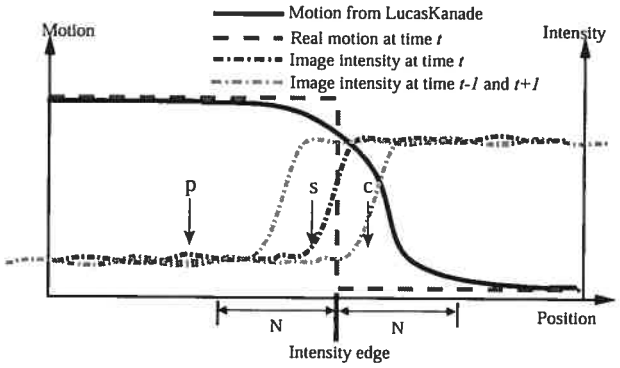

FIG. 1.2 – Synthetic motion returned by Eq. (1.13) *versus* the real motion boundary. The arrows point at three positions (s, p and c) for which $\vec{\mathcal{V}}_s \approx \vec{\mathcal{V}}_p$ and $\vec{\mathcal{V}}_s \neq \vec{\mathcal{V}}_c$.

From these four assumptions (that are generally accepted in the literature [102,152]) a fundamental observation can be made : two close sites, that are not separated by an intensity edge tend to have similar motion vectors. To illustrate this assertion, let us take the three sites s, p and c shown in Figure 1.2. Each of these sites has a *true* motion vector $\vec{\mathcal{V}}_s$, $\vec{\mathcal{V}}_p$ and $\vec{\mathcal{V}}_c$[1] that are to be estimated by an optical flow method (here LK). Since Eq. (1.13) includes no edge information, a vector $\vec{v}_s^k$ computed with the standard LK approach will be corrupted by the dual nature of the motion around s. In other words, because $\vec{v}_s$ (and $\vec{v}_c$) is estimated with a neighborhood window of size $N \times N$, the bimodal nature of motion around site s will locally bias the flow. Since s is a close neighbor of c, $\vec{v}_s \approx \vec{v}_c$ whereas $\mathcal{V}_s \neq \vec{\mathcal{V}}_c$, hence the blurry contour. As a consequence, *flow estimated with a least-squares-fit method close to a sharp intensity edge is likely to be corrupted by multiple motions.*

Also, with the above four assumptions, it may be assumed (without prior information on the true nature of the flow) that $\vec{\mathcal{V}}_s \approx \vec{\mathcal{V}}_p$ and $\vec{\mathcal{V}}_s \neq \vec{\mathcal{V}}_c$. In fact, because

[1] $\mathcal{V}$ is the *true* vector field whereas v is the one estimated by LK.

no intensity edge separates s and p, motion has to change smoothly between these neighbors and thus $\vec{\mathcal{V}}_s \approx \vec{\mathcal{V}}_p$. Similarly, since s and c are separated by a sharp intensity edge, it cannot be assumed that $\vec{\mathcal{V}}_s = \vec{\mathcal{V}}_c$, hence why $\vec{\mathcal{V}}_s \neq \vec{\mathcal{V}}_c$. Of course, there are many cases for which two close sites separated by an intensity edge have the same motion (a background made of stationary objects would be a good example). Thus, the conclusion that can be derived from the previous four assumptions is that site s and c are *likely* (not necessarily) to have different motion.

From these observations, it may be inferred that :

1. when two neighbors s and p are not separated by an intensity edge and s is closer than p to a motion boundary, then $||\vec{v}_s - \vec{\mathcal{V}}_s|| \geq ||\vec{v}_p - \vec{\mathcal{V}}_s||$. In other words, a motion vector estimated away from a strong intensity edge is more likely to be accurate;
2. when the distance between a site p and the closest motion boundary is larger than N (N being the size of η_s), then $\vec{v}_p \approx \vec{\mathcal{V}}_p$ (a similar conclusion was already stated by Thompson in [152]).

Consequently, to fight against the influence of multiple motions and thus keep sharp edges, Eq. (1.16) will be rewritten as :

$$\vec{v}_s^{k+1} = P_\delta^k \sum_{i \in \zeta_\delta} C_i^{k^{-1}} (\vec{v}_i^k + M_i^{-1} \vec{b}_i^k) \tag{1.17}$$

where

$$\delta = \begin{cases} s & \text{if } \mathcal{D}(s) \geq N \\ p & \text{otherwise} \end{cases} \tag{1.18}$$

where N denotes the LK window size (see Eq.(1.1)). Here, $\mathcal{D}(\cdot)$ denotes the distance to the nearest intensity edge and p is a neighbor of site s located further away from that same edge (see Fig. 1.3). The latter equations may be understood as follows : when s is close to an intensity edge, $\vec{v}_s$ estimated with the standard LK approach is likely to be corrupted by multiple motions. It is thus preferable to compute $\vec{v}_s$

with a neighborhood window η_p, shifted away from the nearest intensity edges. In this way, $\vec{v}_s$ is computed with a neighborhood that is more likely to contain an unimodal motion.

As mentioned previously, in real-life scenarios, many strong edges may not correspond to a motion discontinuity. In these cases, estimating the flow with Eq.(1.17) could seem, at first glance, to be a source of error. Fortunately though, when an intensity edge do not correspond to a motion discontinuity, this means that s and p are located in an area of uniform motion. Thus, since s and p are neighbors, motion is likely to be nearly constant between them in such a way that $\vec{\mathcal{V}}_s \approx \vec{\mathcal{V}}_p$. Also, the fusion procedure of Eq.(1.17) helps significantly constrain the flow and thus reduce, if not prevent, error propagation.

1.3.3 Mean-Shift

As shown in Eq. (1.18), when $\mathcal{D}(s) < N$, $\vec{v}_s^*$ is computed over a neighboring site p, located further away from the nearest intensity edge. In this way, $\vec{v}_s$ will be less likely to be corrupted by multiple motion. From section 1.3.2's four assumptions, a good site p must respect the following two criteria

1. p must be a neighbor of site s with $\mathcal{D}(p) \geq \mathcal{D}(s)$ (assumptions 2 and 4).
2. $||\nabla I(p,t)|| \approx 0$ (assumptions 3).

From these criteria, we found that the *mean-shift* procedure [37] offers an appropriate strategy to determine p given s and $I(t)$. Mean-shift is a simple iterative nonparametric estimator of density gradient that was first introduced by Fukunaga and Hostetler [88] and adapted to imagery by Comaniciu and Meer [37]. Mean-shift is based on the multivariate kernel density estimate

$$\hat{f}(x) = \frac{1}{nh^d} \sum_{j=1}^{n} K(\frac{x - x_j}{h}) \tag{1.19}$$

where $K(\cdot)$ is a kernel of radius h and $\{x_j\}_{j=1...n}$ is a set of n points of dimension d. With this density estimate, the density gradient can be expressed as

$$\nabla \hat{f}(x) = \frac{1}{nh^d} \sum_{j=1}^{n} \nabla K(\frac{x - x_j}{h}) \tag{1.20}$$

where, as suggested by Comaniciu and Meer [37], $K(\cdot)$ can be replaced by the Epanechnikov kernel. With such a kernel, the last equation can be redefined as

$$\nabla \hat{f}(x) = \frac{n_x}{n(h^d c_d)} \frac{d+2}{h^2} M_h(x) \tag{1.21}$$

where $M_h(x)$ is called the *sample mean-shift*, *i.e.*,

$$M_h(x) = \frac{1}{n_x} \sum_{x_j \in S_h(x)} (x_j - x) \tag{1.22}$$

where $S_h(x)$ is a hypersphere of radius h centered on x and containing n_x points. Mean-shift is thus a fairly simple iterative algorithm performing the following operations.

For a given point x_j in a d-dimensional space :

1. compute the sample mean-shift $M_h(x_j)$;
2. translate the hypersphere $S_h(x_j)$ by $M_h(x_j)$;
3. repeat steps 1 and 2 until convergence.

When using mean-shift to filter an image, the iterative procedure is applied on data x_j located in a so-called *spatial-range* domain. The *spatial* domain refers to the 2D space of lattice S while the *range* domain refers to the pixel color/intensity level. In this context, each site $s \in S$ is associated with a point x_s in a d-dimensional spatial-range domain. Here, the first two dimensions of x_s correspond to the $I \times J$ spatial Euclidian coordinates while the other dimensions correspond to the intensity (or color) observed over site s. Thus, d is always set to 5 $(i, j, Red_s, Green_s, Blue_s)$ for color images and to 3 $(i, j, Intensity_s)$ for grayscale images. After successive

mean-shift iterations, the hypervolume S_h is shifted from its initial location x_s to a final position x_p where the local gradient is null. We define the *mean-shift vector* as vector $\vec{P}_s$ linking site s to site $p : p = s + \vec{P}_s$. This last relation is the one we have chosen to use in Eq. (1.18). By the very nature of mean-shift, p is always located further away from the nearest intensity edge than s and $||\nabla f(x_p)|| \approx 0$. Also, in general, the stronger the intensity gradient is around site s, the larger $\vec{P}_s$ will be. These are the reasons why we consider that mean-shift meets the two criteria presented at the beginning of this section. To make sure p is a neighbor of s, the length of $\vec{P}_s$ is clamped to a maximum value : $||\vec{P}_s|| = \min(||P_s||, N)$. The reason why the mean-shift vector length is limited to N can be understand as follows. As illustrated in Figure 2 and mentioned in the second observation of section 3.2, a pixel located at a distance $\geq N$ of the nearest edge is less likely to be corrupted by multiple motion. On the other hand, a motion vector $\vec{v}_p$ estimated with a window located too far away from its original pixel s will violate the first observation of section 3.2. Therefore, by making sure the mean-shift displacement is limited to N, both observations are being respected.

For more details on mean-shift, please refer to [37].

Fig. 1.3 shows a mean-shift vector field P with vectors linking site s to site p. As can be seen, the closer to the edge a site s is, the larger the mean-shift displacement is.

1.4 Other methods implemented

In section 1.5, the proposed method is compared with eight other optical flow methods whose objective is to preserve sharp motion discontinuities (except for Horn and Schunck). In this section, we review in detail these methods and underline how they were implemented numerically.

Horn and Schunck We have compared our method with differential optical flow methods among which some seek to preserve sharp discontinuities. All differential

Algorithm 2 : Our Method Processing one Pyramid Level

$I(t)$	Image at time t
P_s	Mean-shift vector
V, V^{prev}	Vector fields
C_s	Variance-covariance matrix of site s.
N_l	Noise Level

$P \leftarrow$ Apply mean-shift on $I(t)$

for each site $s \in S$ **do**
  rnd $\leftarrow$ random value between 0 and N_l
  $I(s,t), I(s,t+1) \mathrel{+}= \text{rnd}$

$\nabla I \leftarrow$ Spatial gradient of $I(t)$
$V, V^{prev} \leftarrow 0$

do
  $I_t \leftarrow I(s + V^{prev}, t) - I(s, t+1)$
  /* **Motion Estimation** */
  for each site $s \in S$ **do**
    $\delta \leftarrow s + P_s$
    Compute M_δ^{-1} and $\vec{b}_\delta^k$
    $\vec{v}_s \leftarrow \vec{v}_s^{prev} + M_\delta^{-1} \vec{b}_\delta^k$
  $V^{prev} \leftarrow V$

  /* **Covariance filtering** */
  Compute $C_s, \forall s \in S$
  for each site $s \in S$ **do**
    $\delta \leftarrow s + P_s$
    $\vec{v}_s \leftarrow P_\delta \sum_{i \in \zeta_\delta} C_i^{-1} \vec{v}_i^{prev}$

while $||V - V^{prev}|| >$ accuracy threshold

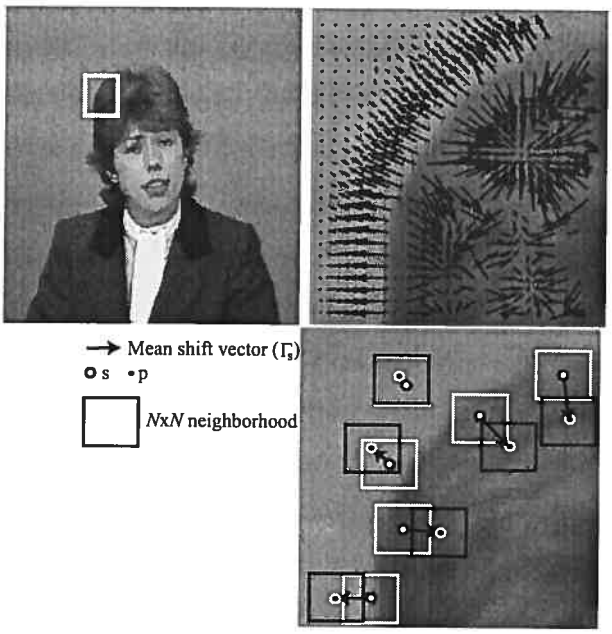

FIG. 1.3 – Zoom on a frame of CLAIRE sequence. Every vector shows the estimated mean-shift displacement between a site s and a site p.

methods we have implemented are based on the data conservation constraint :

$$I_x u + I_y v + I_T = 0 \tag{1.23}$$

where I_x and I_y are the spatial derivatives, I_T the temporal derivative and (u, v) the horizontal and vertical image velocity at a site $s \in S$. Because this formulation involves one equation and two unknowns, it admits an infinite number of solutions. As mentioned previously, the standard solution for handling this ill-posed problem is to add a spatial coherence term E_{sp} whose essential role is to constrain the solution. One common formulation of E_{sp} is the *membrane model* proposed by Horn and Schunck [21]

$$E_{sp}(\vec{v}) = ||\nabla u||^2 + ||\nabla v||^2 \tag{1.24}$$

or, when considering the discrete version of ∇,

$$E_{sp}(\vec{v}) = \frac{1}{4}\sum_{r\in\mathcal{G}_s}[(u_r - u_s)^2 + (v_r - v_s)^2] \tag{1.25}$$

where $\mathcal{G}_s$ contains the four neighbors of size s. In this context, the energy function $E(\vec{v})$, to be minimized at every site $s \in S$, is represented by

$$E(\vec{v}) \;=\; (I_x u + I_y v + I_T)^2 + \frac{\alpha}{4}\sum_{r\in\mathcal{G}_s}[(u_r - u_s)^2 + (v_r - v_s)^2] \tag{1.26}$$

where α is a constant giving more or less importance to the spatial constraint. With the Euler-Lagrange equations, the underlying computation in Eq. (1.26) reduces to a Jacobi optimization, namely

$$u^{[k+1]} = \bar{u}^{[k]} - I_x\frac{I_x u^{[k]} + I_y v^{[k]} + I_T}{\alpha^2 + I_x^2 + I_y^2} \tag{1.27}$$

$$v^{[k+1]} = \bar{v}^{[k]} - I_y\frac{I_x u^{[k]} + I_y v^{[k]} + I_T}{\alpha^2 + I_x^2 + I_y^2} \tag{1.28}$$

where $(\bar{u}, \bar{v})$ is the local mean of the four nearest neighbors s. In our implementation, a total of 1000 iterations is used and α is set between 5 and 30 depending on the sequence.

Black and Anandan It is well known that the quadratic functions of Eq. (1.26) give an overwhelming importance to outliers which is a major cause of blurry edges. The first to propose a *robust* formulation of Eq. (1.26) was the pioneer work by Black and Anandan [102]. The robust gradient-based formulation they proposed has the following form :

$$E(\vec{v}) \;=\; \rho(I_x u + I_y v + I_T, \sigma_D) + \alpha\sum_{r\in\mathcal{G}_s}[\rho(u_r - u_s, \sigma_S) + \rho(v_r - v_s, \sigma_S)]$$

where, in our implementation, the importance function is ρ is the Lorentzian function [102]

$$\rho(x, \sigma) = \log(1 + \frac{1}{2}(\frac{x}{\sigma})^2). \tag{1.29}$$

As suggested by Black and Anandan, $E(\vec{v})$ can be minimized with a multiresolution Successive Over-Relaxation method with a fixed number of iterations (twenty in our case) on each level of the pyramid. Following Black and Anandan recommendations, the values of σ_D and σ_S are also lowered according to an annealing schedule [102]. This schedule comprises six stages for which σ_D and σ_S vary respectively from $18/\sqrt{2}$ to $5/\sqrt{2}$ and $3/\sqrt{2}$ to $0.03/\sqrt{2}$. The variable α was set to 0.2.

Image-based Isotropic Regularization Although Horn and Schunck's method enjoys a great deal of popularity and is still frequently used, as mentioned previously its "blind" quadratic regularizer E_{sp} is well known to blur-out motion discontinuities. Consequently, variational methods with a different regularization term have been proposed to better preserve flow in these areas. A survey of some of these methods has been published by Weickert and Schnörr [82].

One such method we have implemented uses an image-based robust isotropic regularization factor similar to the one proposed by Alvarez *et al.* [92] :

$$E_{sp} = \Psi(||\nabla I||^2)(||\nabla u||^2 + ||\nabla v||^2). \tag{1.30}$$

The goal of this regularization function is to strongly regularize flow in textureless areas and reduce smoothing at image boundaries. Since $\Psi(\cdot)$ is a decreasing and strictly positive function, the Charbonnier function has been implemented : $\Psi(s^2) = 1/\sqrt{1 + s^2/\lambda^2}$. In this way, as opposed to Horn and Schunck's method which uniformly regularizes the flow, the Ψ function prevent Alvarez' method from smoothing accross an image edge. In other words, in the vicinity of an image edge (*i.e.* where $||\nabla I||$ is large) the influence of the prior function E_{sp} is significantly reduced. This makes the method less prone to smooth out the flow where a motion

discontinuity is likely to be.

As mentioned by Alvarez *et al.* [92], following the Euler-Lagrange equations, the to-be-optimized optical flow (u, v) must satisfy the following diffusion-reaction system at its steady state (*i.e.*, $k \rightarrow \infty$)

$$\frac{\partial u}{\partial k} = \alpha \ \mathrm{div}(\Psi(||\nabla I||^2)\nabla u) - I_x(I_x u + I_y v + I_T) \tag{1.31}$$

$$\frac{\partial v}{\partial k} = \alpha \ \mathrm{div}(\Psi(||\nabla I||^2)\nabla v) - I_y(I_x u + I_y v + I_T). \tag{1.32}$$

where k denotes an artificial evolution parameter that should not be confused with time parameter t of the image sequence. This system can be solved with a modified-explicit scheme whose numerical structure is given by

$$\frac{u^{[k+1]} - u^{[k]}}{\tau} = I_x(I_x u^{[k+1]} + I_y v^{[k]} + I_T) - \alpha A(u^{[k]}, v^{[k]}) \tag{1.33}$$

where $A(u^k)$ is the numerical approximation of $\mathrm{div}(\alpha\Psi(||\nabla I||^2)\nabla u)$, k is the iteration index and τ is the optimization time step (not to confuse with the time step T) that we set to 0.25. This optimization scheme was taken in Weickert and Schnörr's 2001 paper [82] and will also be used for the implementation of the *Flow-based Isotropic method* and the *Flow-based Anisotropic method.* Here, a total of 1000 iterations is used together with α set between 100 and 400 and λ between 0.5 and 10. Variables α and λ are manually adjusted to each video sequence. Notice that v is optimized in similar way.

Image-based Anisotropic Regularization The anisotropic image-based variational method we have implemented is the one proposed by Nagel and Enkelmann [62,65] which uses second-order spatial derivatives to constrain the flow. The key idea of their method is the use of an "oriented smoothness" constraint that suppresses smoothing across image edges while encouraging smoothing along image

edges. The energy function their method minimizes has the following shape [62,70] :

$$E(\vec{v}) = (I_x u + I_y v + I_T)^2 + \alpha^2 \mathrm{tr}((\nabla \vec{v})^{\mathrm{T}} W (\nabla \vec{v})) \tag{1.34}$$

with

$$W = \frac{1}{I_x^2 + I_y^2 + 2\gamma} \begin{pmatrix} I_y^2 + \gamma & -I_x I_y \\ -I_x I_y & I_x^2 + \gamma \end{pmatrix} \tag{1.35}$$

where γ and α are two constants set between 2 and 5 depending on the nature of the flow to be estimated and "tr" is the trace operator. This functional may be minimized by a Gauss-Seidel optimizer as explained in [62, 70]. All details of our implementation have been taken from Appendix A of Barron *et al.*'s technical report [69]. However, in contrast to what Barron *et al.* [70] suggested, a fixed number of 500 iterations has been used.

Flow-based Isotropic Regularization Another kind of modification to Horn and Schunck's variational model leads to the so-called "flow-based" methods [56,80, 82,83,129]. These isotropic flow-based methods are implemented on top of a robust regularizer that reduces the influence of outliers. The method we have implemented is similar to the one proposed by Weickert [80] and to the *2-D version* of Weickert and Schnörr [83] spatio-temporal method. For this method, the energy functional to be minimized is expressed as :

$$E(u, v) = (I_x u + I_y v + I_T)^2 + \alpha \Psi(||\nabla u||^2 + ||\nabla v||^2) \tag{1.36}$$

where $\Psi(\cdot)$ is a robust function. In this way, this flow-based isotropic method strongly regularizes the flow in areas where $||\nabla u||^2$ and $||\nabla v||^2$ are small and reduce regularization where $||\nabla u||^2$ and/or $||\nabla v||^2$ is strong, *i.e.* in the vicinity of a motion discontinuity.

The gradient descent of this functional leads to the reaction-diffusion system :

$$\frac{\partial u}{\partial k} = \nabla(\Psi'(||\nabla u||^2 + ||\nabla v||^2)\nabla u) - \frac{I_x}{\alpha}(I_x u + I_y v + I_T) \tag{1.37}$$

$$\frac{\partial v}{\partial k} = \nabla(\Psi'(||\nabla u||^2 + ||\nabla v||^2)\nabla v) - \frac{I_y}{\alpha}(I_x u + I_y v + I_T) \tag{1.38}$$

where $\Psi'(s^2)$ is the derivative of $\Psi(s^2)$ (with respect to s^2) that we set equal to $\frac{1}{\sqrt{(1+s^2/\lambda^2)}}$ as suggested by Weickert [80]. As is the case for the image-based isotropic method, we resort to a modified-explicit scheme [83] to perform the gradient descent. A total number of 1000 iterations is used, with λ set between 0.01 and 0.1 and $\alpha = 300$.

Flow-based Anisotropic Regularization Flow-based anisotropic regularization motion estimaton methods are similar to the anisotropic image-based methods previously introduced. The flow-based anisotropic methods encourage regularization *along* flow discontinuities while discouraging regularization *across* flow discontinuities [81, 82, 134]. This family of methods is thus clearly different from the Flow-based isotropic methods which only prevent smoothing across motion discontinuity. The anisotropic method we have implemented was taken from Weickert and Schnörr [82] which minimizes the following functional :

$$E(u, v) = (I_x u + I_y v + I_T)^2 + \alpha \text{ tr } \Psi(\nabla u \nabla u^{\mathrm{T}} + \nabla v \nabla v^{\mathrm{T}}) \tag{1.39}$$

where $\Psi(\cdot)$ is a robust function. Using the Euler-Lagrange theorem, the gradient descent of this functional leads to the reaction-diffusion system [82]

$$\frac{\partial u}{\partial k} = \frac{I_x}{\alpha}(I_x u + I_y v + I_T) + \text{div}(\Psi'(J)\nabla u) \tag{1.40}$$

$$\frac{\partial v}{\partial k} = \frac{I_y}{\alpha}(I_x u + I_y v + I_T) + \text{div}(\Psi'(J)\nabla v) \tag{1.41}$$

where

$$J = \nabla u \nabla u^{\mathrm{T}} + \nabla v \nabla v^{\mathrm{T}} \tag{1.42}$$

and

$$\Psi'(J) = \Psi'(\sigma_1)\vec{\mu}_1\vec{\mu}_1^{\mathrm{T}} + \Psi'(\sigma_2)\vec{\mu}_2\vec{\mu}_2^{\mathrm{T}} \tag{1.43}$$

where σ_1, σ_2 and $\vec{\mu}_1, \vec{\mu}_2$ are the eigenvalues and the eigenvectors of J, and $\Psi'(s^2)$ is fixed to $\frac{1}{\sqrt{(1+s^2/\lambda^2)}}$. The optimization is performed through a modified-explicit scheme [83] together with a total of 1000 iterations. The variable λ is set between 0.01 and 0.1 and α between to 100 and 500. As is the case for the other methods, these values are adjusted according to the nature of the flow to be estimated.

Combined Local-Global Method The traditional Lucas-Kanade method which estimates flow with a simple matrix inversion (see Eq.(1.8)) is clearly a local approach since it provides no means to propagate flow. However, a basic iterative scheme such as the one of Algorithm 1, or a more elaborate one such as the one we propose in Algorithm 2, can allow a LK-based method to propagate flow. Thus, we believe that our method should be compared with a recently-published method which explicitly combines the Lucas-Kanade and Horn and Schunck method into a synthetic "local-global" method [6]. Using the notations

$$\begin{aligned} \vec{w}_s &= (u_s, v_s, 1)^{\mathrm{T}} \\ ||\vec{w}_s||^2 &= ||\nabla u_s||^2 + ||\nabla v_s||^2 \\ \nabla_3 I &= (I_x, I_y, I_T)^{\mathrm{T}} \\ J_\rho(\nabla_3 I) &= W_\rho * (\nabla_3 I \nabla_3 I^{\mathrm{T}}) \end{aligned} \tag{1.44}$$

where W_ρ is a Gaussian kernel with standard deviation ρ and $J_\rho(\nabla_3 I)$ is a *stucture tensor*. The authors argue that the Lucas-Kanade and Horn and Schunck energy

functionals can be respectively expressed as

$$E_{\text{LK}}(\vec{w}) = \vec{w}^{\text{T}} J_\rho(\nabla_3 I)\vec{w} \tag{1.45}$$

$$E_{\text{HS}}(\vec{w}) = \int \vec{w}^{\text{T}} J_0(\nabla_3 I)\vec{w} + \alpha||\nabla w||^2 dxdy. \tag{1.46}$$

where $J_0(\nabla_3 I)$ is a structure tensor with a zero standard deviation. A local-global energy function can thus be obtained by simply replacing $J_0(\nabla_3 I)$ by $J_\rho(\nabla_3 I)$ with $\rho > 0$:

$$E_{\text{HS\_LK}}(\vec{w}) = \int \vec{w}^{\text{T}} J_\rho(\nabla_3 I)\vec{w} + \alpha||\nabla w||^2 dxdy. \tag{1.47}$$

As can be seen, $E_{\text{HS\_LK}}$ implements two spatial coherence constraints : an integration window (first term) and a prior function (second term). These two constants implicitly compensate for their mutual limitations and thus generate a better constrained flow. The authors argue that more accurate results can be obtained with a non-quadratic variation of $E_{\text{LK\_HS}}$ whose formulation is as follows :

$$E_{\text{HS\_LK}}(\vec{w}) = \int \Psi_1(\vec{w}^{\text{T}} J_\rho(\nabla_3 I)\vec{w}) + \alpha\Psi_2(||\nabla w||^2) dxdy \tag{1.48}$$

where $\Psi_1(s^2)$ and $\Psi_2(s^2)$ are non-quadratic robust functions. All results reported in this section with the subscript "HS_LK" have been obtained after minimizing Eq.(1.48). As mentioned by the authors, the Euler-Lagrange equations of $E_{\text{HS\_LK}}$ are given by

$$\begin{aligned} 0 = {} & \text{div}(\Psi_2'(||\nabla_3 w||^2)\nabla u) \\ & -\frac{1}{\alpha}\Psi_1'(\vec{w}^{\text{T}} J_\rho(\nabla_3 I)\vec{w})(W_{11}u + W_{12}v + W_{13}) \\ 0 = {} & \text{div}(\Psi_2'(||\nabla_3 w||^2)\nabla v) \\ & -\frac{1}{\alpha}\Psi_1'(\vec{w}^{\text{T}} J_\rho(\nabla_3 I)\vec{w})(W_{21}u + W_{22}v + W_{23}) \end{aligned}$$

where W_{ij} is a component of the structure tensor $J_\rho(\nabla_3 I)$. In our implementation, $\Psi_i(s^2) = \frac{1}{\sqrt{1+s^2/\lambda_i^2}}$ and the vector field v is computed with a Successive Over-Relaxation optimization scheme [6]. The number of iterations is set to 1000 and variables λ_1 and λ_2 are assigned values ranging between 0.05 and 0.005, depending on the nature of the scene.

Bruhn *et al.* [6] stipulate that the method can be made "3D" through the use of spatio-temporal filters. Although this modification may help better estimate motion, to make the comparison fair for the other methods implemented in a "2-D" style, we have implemented what Bruhn *et al.* call the *non-quadratic 2D* energy function (here Eq. (1.48)).

1.5 Results

In this section, an extensive set of results obtained with our method and the ones introduced in the previous section are presented. These results have been obtained on synthetic, realistic, and real sequences. For those sequences with a groundtruth vector field, two angular error metrics taken from Barron *et al.* [70] are used to provide a quantitative measure of quality. For the ones with unknown groundtruth, vector fields snapshots are provided.

Setup and Implementation Details

In order to correctly gauge performance, every optic flow method were implemented in a similar matter. They share the same spatial and temporal gradient function, the same multiresolution framework and they process video sequences that were pre-filtered by the same low-pass spatio-temporal filter. In fact, all image sequences presented in this paper have been pre-filtered by a spatio-temporal Gaussian filter. For each sequence, the spatial standard deviation of the filter is set to 1.5 whereas, the temporal standard deviation is set to 1.5 for sequences YOSEMITE, TRANSLATION TREE, DIVERGENCE TREE, CARS OVER PARK, FLOWER, and MOM AND DAUGHTER and to 0.5 for sequences ROTATING BONSAI, PARTHENON, TRANSLATING SHAPES, CLAIRE,TAXI, and KARLSRUHE. The spatial

derivatives in x and y used by every method is approximated with a central difference. As for the temporal derivative, a simple two-frame difference is used through every sequence.

Since our method is built upon a multiresolution framework, to make the comparison fair for every approach, each method has also been implemented in a multiresolution fashion. Here, the multiresolution framework is the same for every method : the coarse-scale solution obtained at level L serves as initial data for estimating the flow at the level $L-1$. The reader should be aware that the number of pyramid levels is a fundamental issue when estimating motion. For instance, we observed that for scenes with *global* motion such as *Yosemite* or the *Translating tree* sequence, a high pyramid with many levels improves estimation of the flow. On the other hand, scenes with *local* motion, *i.e.*, motion with small-scale details such as the *Taxi* or the *Bonsai* sequences for instance, a pyramid implementation with fewer levels is preferable. For these reasons, the number of pyramid levels used by each method has been manually adjusted to each sequence. Typically, the number of levels is set between 1 and 3 for each method.

Since a uniform implementation framework is used for every method, some results reported in this paper differ slightly from those previously reported in other papers. For instance, our implementation of Horn and Schunck's method produces, for the *Yosemite* sequence, an average angular error of 7.35 as opposed to 11.26 in Barron *et al.* [70]. This is mainly due to the fact that our implementation of HS has been made multiresolution.

Also, when we implemented these methods, we first used the parameter sets provided by the authors. However, we came to realize that these parameters are not well suited for every sequences. For instance, the α regularization term of the Horn and Schunck method needs to be large when estimating a "global" optical flow (*i.e.* a vector field with little or no motion discontinuities) such as the TRANSLATING TREE,the DIVERGING TREE and the YOSEMITE SEQUENCES. Indeed, when α is large, the flow diffusion is maximal and regions with little or no texture can be compensated by a strong regularization. On the other hand, using a large α value

on "local" sequences such as KARLSRUHE ,TAXI and the MOVING SHAPES, has the effect of over-regularizing the flow and thus generate "super-blurry" motion fields. Thus, for those local sequences, a smaller α value is better suited. In other words, a good set of parameters for a global sequence is not necessarely appropriate for a local sequence. Thus, to fairly compare the methods together, we beleive that the parameters of every method need to be adjusted to the content of the sequence. Using only one parameter set per method would be harmful for most methods. Unfortunately, since most original papers do not provide a parameter set for local and global sequences, we had to heuristically adjust it. Consequently, the parameters were optimized to obtain the best results based on the content of the scene.

Let us also mention that since processing speed is not the core of this paper, large number of iterations (sometimes much greated than necessary) has been given to every method. This is to make sure that every method converge towards a stable solution. In most cases, the specific number of iterations has been taken from the original papers and slightly increased in some cases.

For our method, $\mathcal{X} = \Lambda = N = 9$ or 11[2] (for Algorithm 1, $N \in \{7, 9, 11\}$) and the noise level bias (variable N_l in Algorithm 2) between 2.0 and 5.0 for every sequence.

Notice that the noise level has also been used in the implementation of Algorithm 1 to make sure LK always produces vector fields of 100% density. In fact, the Algorithm 1 is identical to our method exept for the edge-avoidance procedure and the fusion procedure (Eq. (1.16)). In this way, the results clearly illustrate the difference between our method and the traditionnal iterative LK algorithm. Also, as shown in Algorithm 2, our method requires that the covariance matrix C_s be computed for each $s \in S$ at each optimization iteration. However, we observed empirically that computing C_s only once at the first iteration and reusing it afterward does not significantly reduce the quality of the results. In this way, C_s is estimated

[2] $\mathcal{X}$ is the fusion window half-size in Eq.(1.14) and Λ is the window half-size used to compute the covariance matrix C_s in Eq.(1.15).

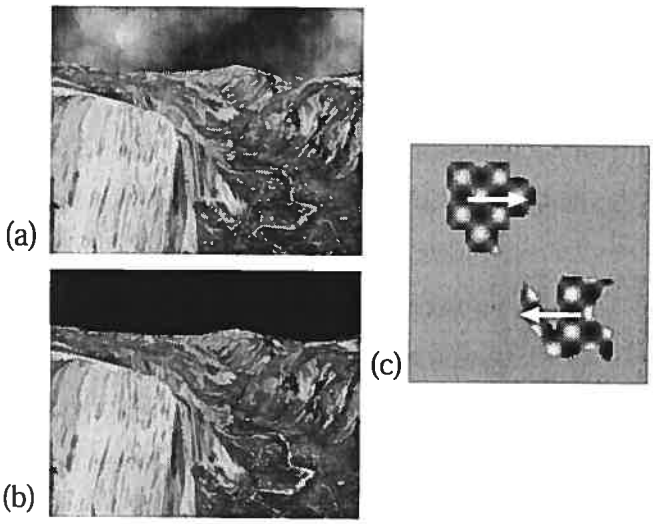

FIG. 1.4 – Synthetic sequences.(a) YOSEMITE WITH SKY (b) YOSEMITE WITHOUT SKY and (c) the SHAPES sequence.

once at every scale of the pyramid. We thus adopted that strategy as a means to save processing time.

The number of iterations for Algorithm 1 and 2 is set to three whereas the number of mean shift iterations is set to four. For each example, the radius of the mean-shift hypervolume $S_h(x)$ is set to N in the spatial domain and to 10 in the range domain. Notice that all flow fields presented in this section have a density of 100%.

1.5.1 Metrics

As mentioned before, three kinds of sequences are used to compare the methods : synthetic, realistic and real sequences. While synthetic sequences are composed of pure computer-generated images, realistic sequences are made up of real-world images with simulated motion. Both synthetic and realistic sequences come with a groundtruth vector field which makes it possible to quantitatively compare the methods. Following Barron *et al.* [70], we implemented the average angular error

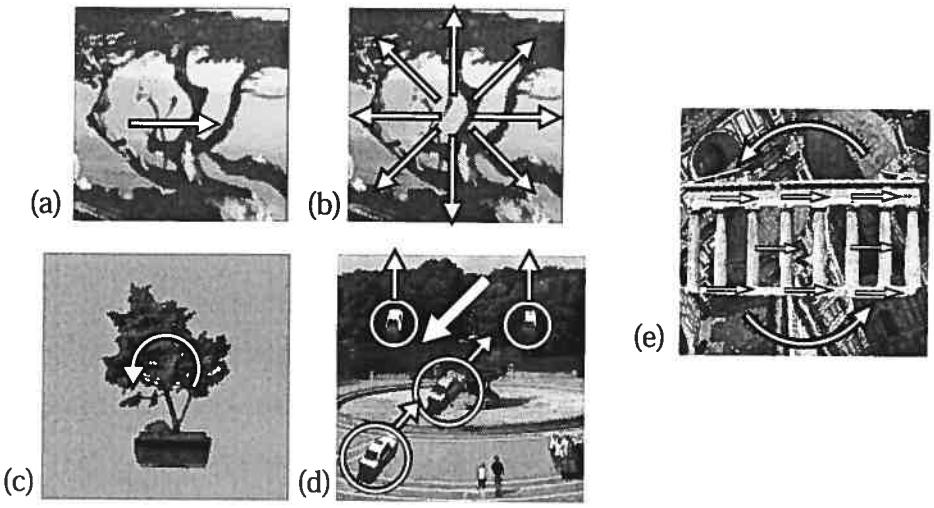

FIG. 1.5 – Realistic sequences. (a) TRANSLATING TREE (b) DIVERGING TREE (c) ROTATING BONSAI (d) CARS OVER PARK and (e) PARTHENON.

metric to evaluate the distance between the ground truth vector field $\mathcal{V}$ and the estimated vector field $\hat{v}$, namely

$$\bar{\psi}_E = \frac{1}{\mathcal{N} \times M} \sum_{s \in S} \arccos(\vec{v}_s . \vec{\mathcal{V}}_s) \tag{1.49}$$

where $\vec{v}_s$ and $\vec{\mathcal{V}}_s$ are normalized 3D vectors : $\vec{v}_s = \frac{(u,v,1)}{\sqrt{u^2+v^2+1}}$. Because our method is meant to preserve sharp discontinuities, the metric was also implemented on vectors located at a distance lower or equal than 10 pixels from a motion edge, *i.e.*,

$$\bar{\psi}^e_E = \frac{1}{\mathcal{N} \times M} \sum_{\substack{s \in S \\ \mathcal{D}_s \leq 10}} \arccos(\vec{v}_s . \vec{\mathcal{V}}_s). \tag{1.50}$$

where $\mathcal{D}_s$ is the distance in pixels between site s and the nearest motion edge. This metrics is used to evaluate how accurate the optical flow algorithms are near flow

discontinuities. The results are also presented in terms of the standard deviation

$$\sigma_{\psi_E} = \sqrt{\frac{1}{\mathcal{N} \times M} \sum_{s \in S} (\arccos(\vec{v}_s.\vec{\mathcal{V}}_s) - \bar{\psi}_E)^2} \tag{1.51}$$

and

$$\sigma_{\psi_E^e} = \sqrt{\frac{1}{\mathcal{N} \times M} \sum_{\substack{s \in S \\ \mathcal{D}_s \leq 10}} (\arccos(\vec{v}_s.\vec{\mathcal{V}}_s) - \bar{\psi}_E^e)^2}. \tag{1.52}$$

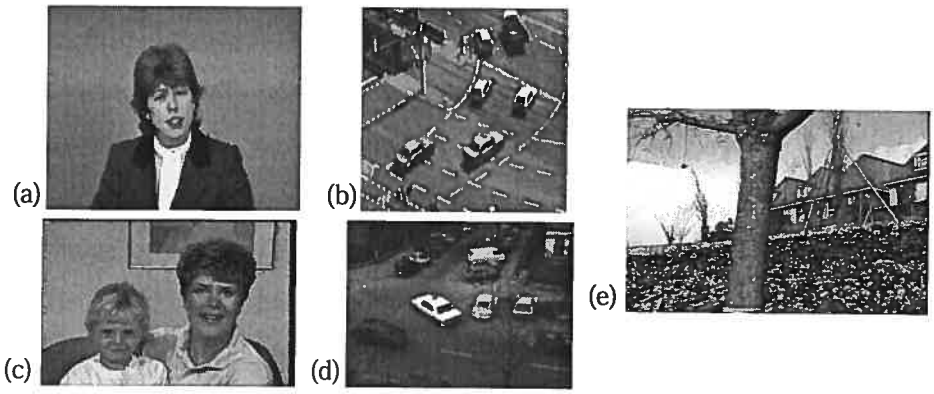

FIG. 1.6 – Real sequences. (a) CLAIRE (b) KARLSRUHE (c) MOM AND DAUGHTER (d) TAXI and (e) FLOWER.

1.5.2 Synthetic Sequences

We have compared the 9 methods over three synthetic sequences, namely YOSEMITE WITH SKY, YOSEMITE WITHOUT SKY, and TRANSLATING SHAPES. Notice that in the case of the YOSEMITE WITH SKY sequence, although the first version of the ground truth file had a sky with a translational velocity of 1.0, we used the more recent one with a velocity of 2.0. The two YOSEMITE sequences were taken from Barron *et al.* [70] and Micheal Black's web site (http ://www.cs.brown.edu/people/black/) while the TRANSLATING SHAPES sequence was computer generated. The latter sequence exhibits two arbitrary shapes running horizontally in front of a uniform

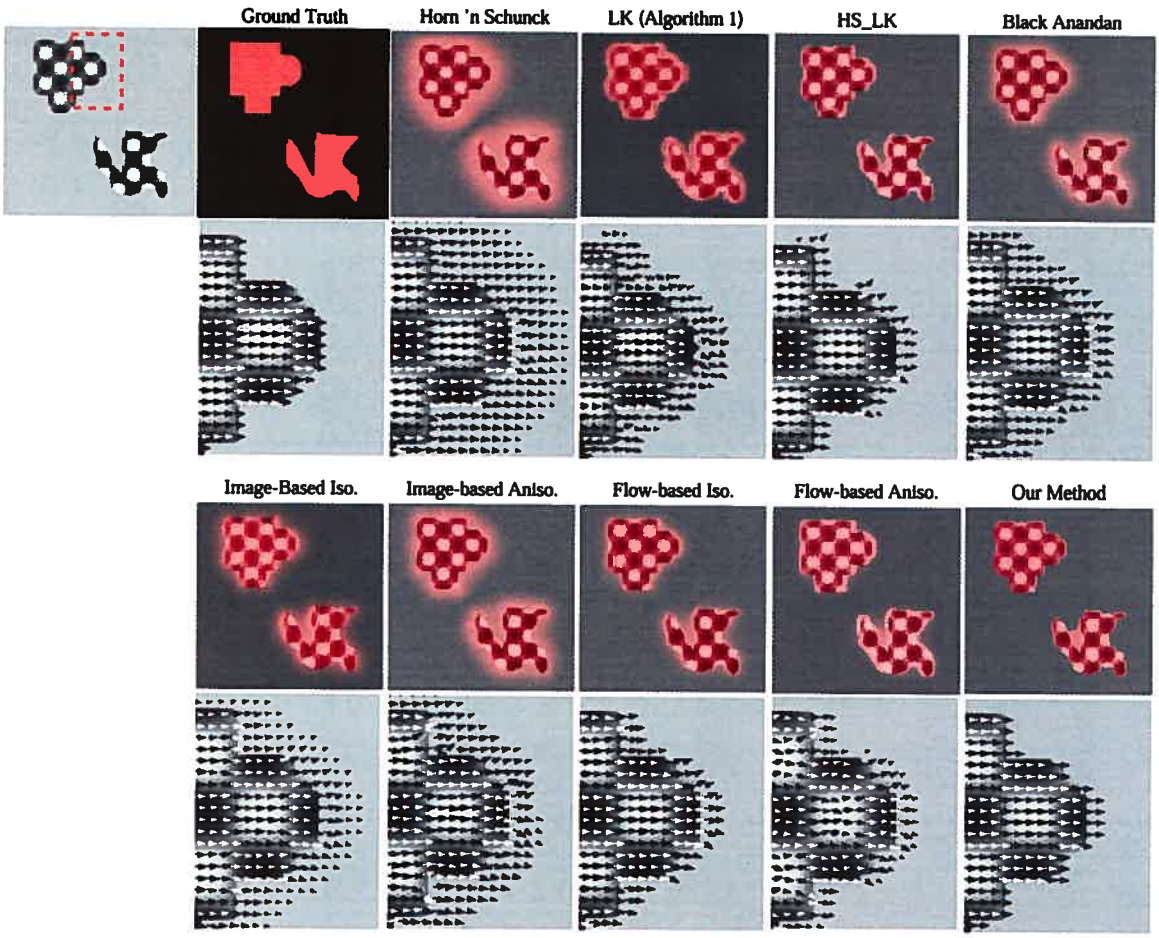

FIG. 1.7 – Results for the TRANSLATING SHAPES sequence.

background. The three sequences are shown in Fig. 1.4 and the quantitative results are presented in Tables 1, 2 and 3. The results for the TRANSLATING SHAPES sequence are also plotted in Fig. 1.7. In this figure, the first row presents the magnitude of the estimated vector field (in red) overlapped with a time frame. These images (as well as the zoom on the vector field) illustrate visually how precise each method is on that sequence.

Notice that for these three synthetic sequences, our method generates results that are at least, if not more precise than the ones produced by the other methods. This is especially true near the motion edges of the TRANSLATING SHAPES

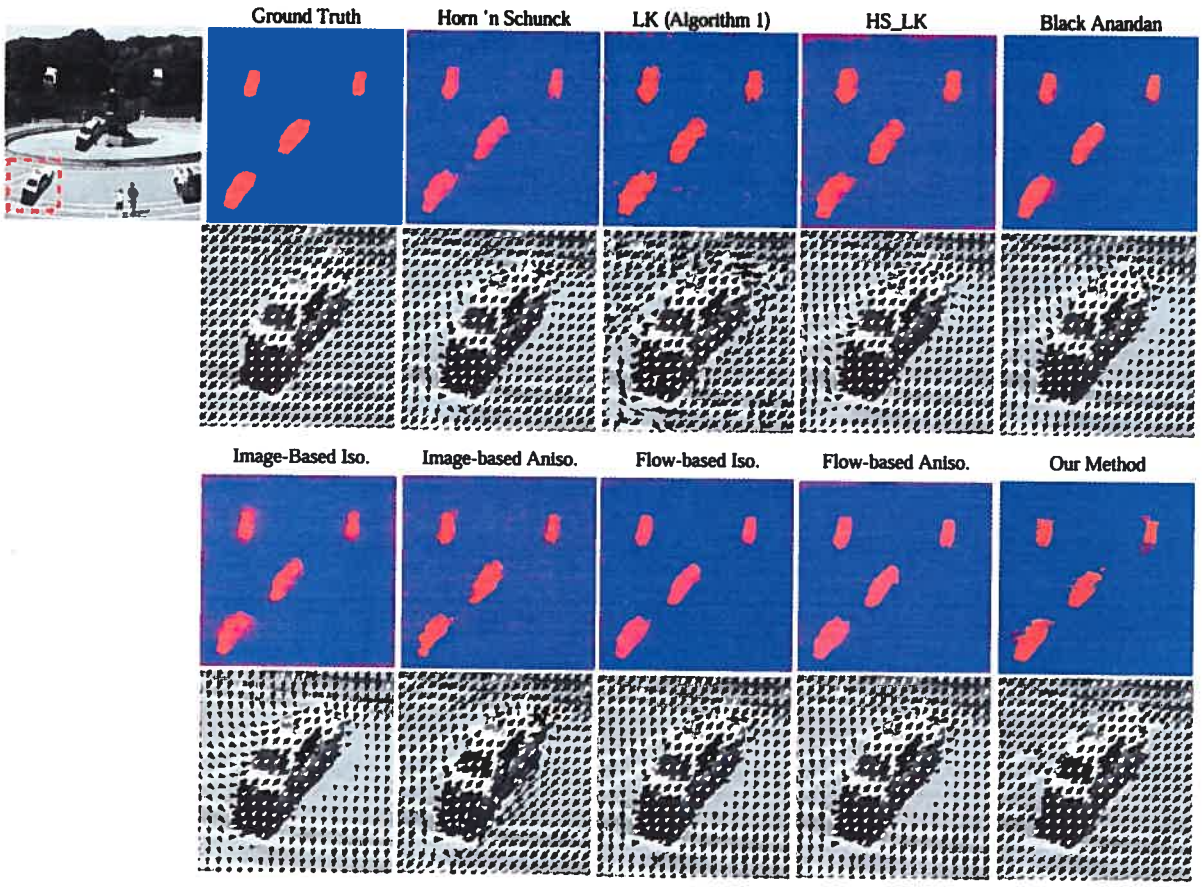

FIG. 1.8 – Results for the CARS OVER PARK sequence. The red channel contains the magnitude of the vectors pointing upward and the blue channel contains the magnitude of the vectors pointing downward.

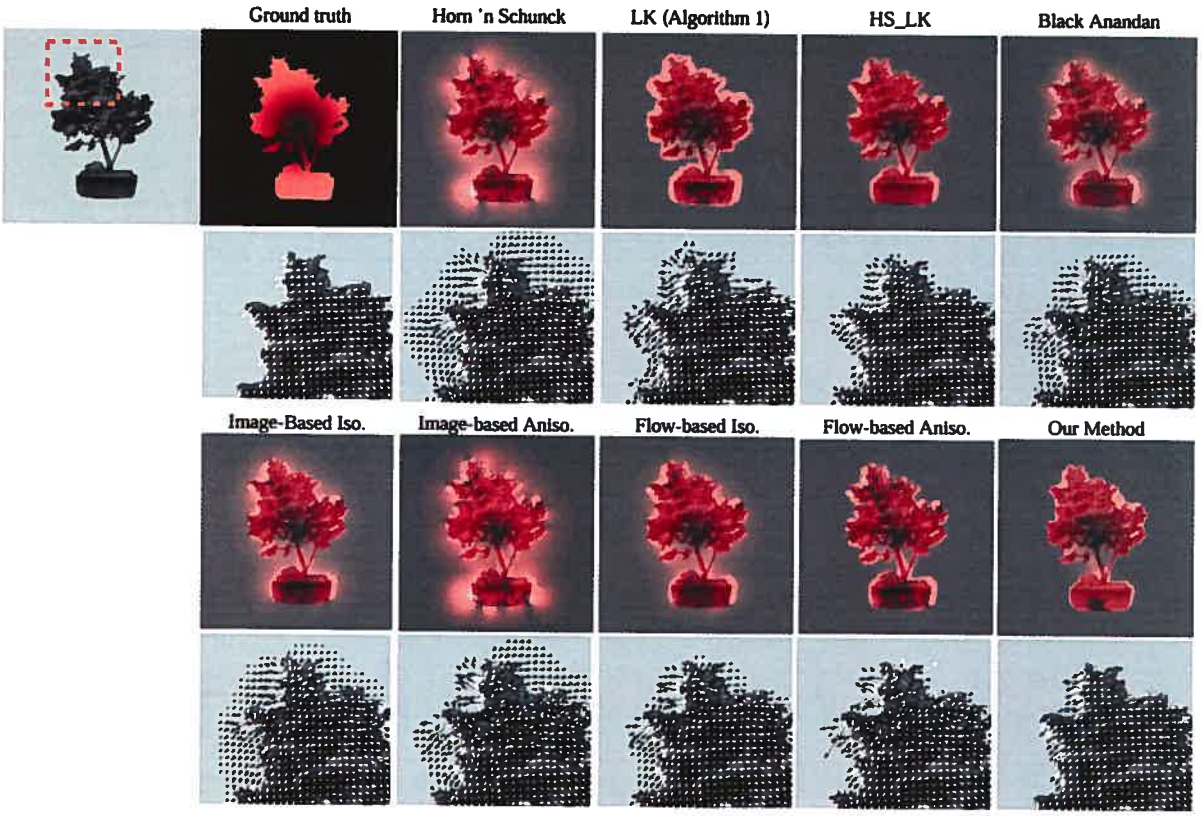

FIG. 1.9 – Results for the ROTATING BONSAI sequence.

sequence.

1.5.3 Realistic Sequences

The methods were also tested over the five realistic sequences presented in Fig. 1.5. The first two are the famous TRANSLATING TREE and DIVERGING TREE sequences [70] for which motion is global with no discontinuities. The other three sequences were computer generated with real images. For the CARS OVER PARK sequence, four cars (cropped from the KARLSRUHE sequence) are pasted on top of a picture of Central Park. The four cars move upward while the background is diagonally shifted toward the lower left corner. For the ROTATING BONSAI sequence, an image of a bonsai (a small tree in a pot) rotates in front of a flat motionless background. The last realistic sequence is the PARTHENON sequence for which an

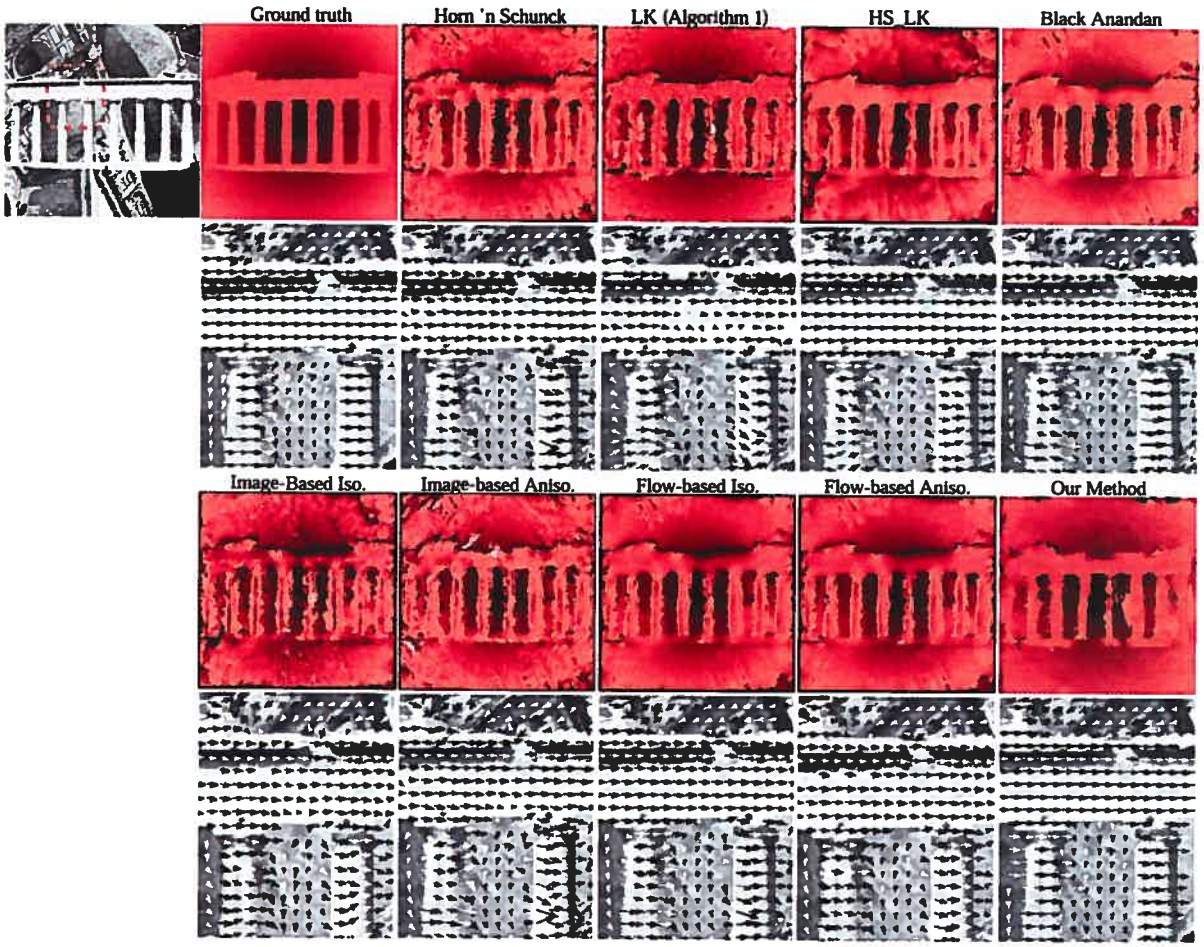

FIG. 1.10 – Results for the PARTHENON sequence.

image of the Parthenon is plotted in front of an image of Florence's Duomo. In this sequence, the Parthenon moves horizontally toward the right whereas the background has a counterclockwise rotation. The reason why we picked these sequences was to illustrate how good our method is with sequences having different amounts of texture.

Quantitative results for the realistic sequences are presented in Tables 4 to 8. As we did for the synthetic TRANSLATING SHAPES sequence, the magnitude of the estimated vector fields have been overlapped with a time frame to illustrate how precise the methods are near motion boundaries. This is shown in Fig. 1.8, 1.9,

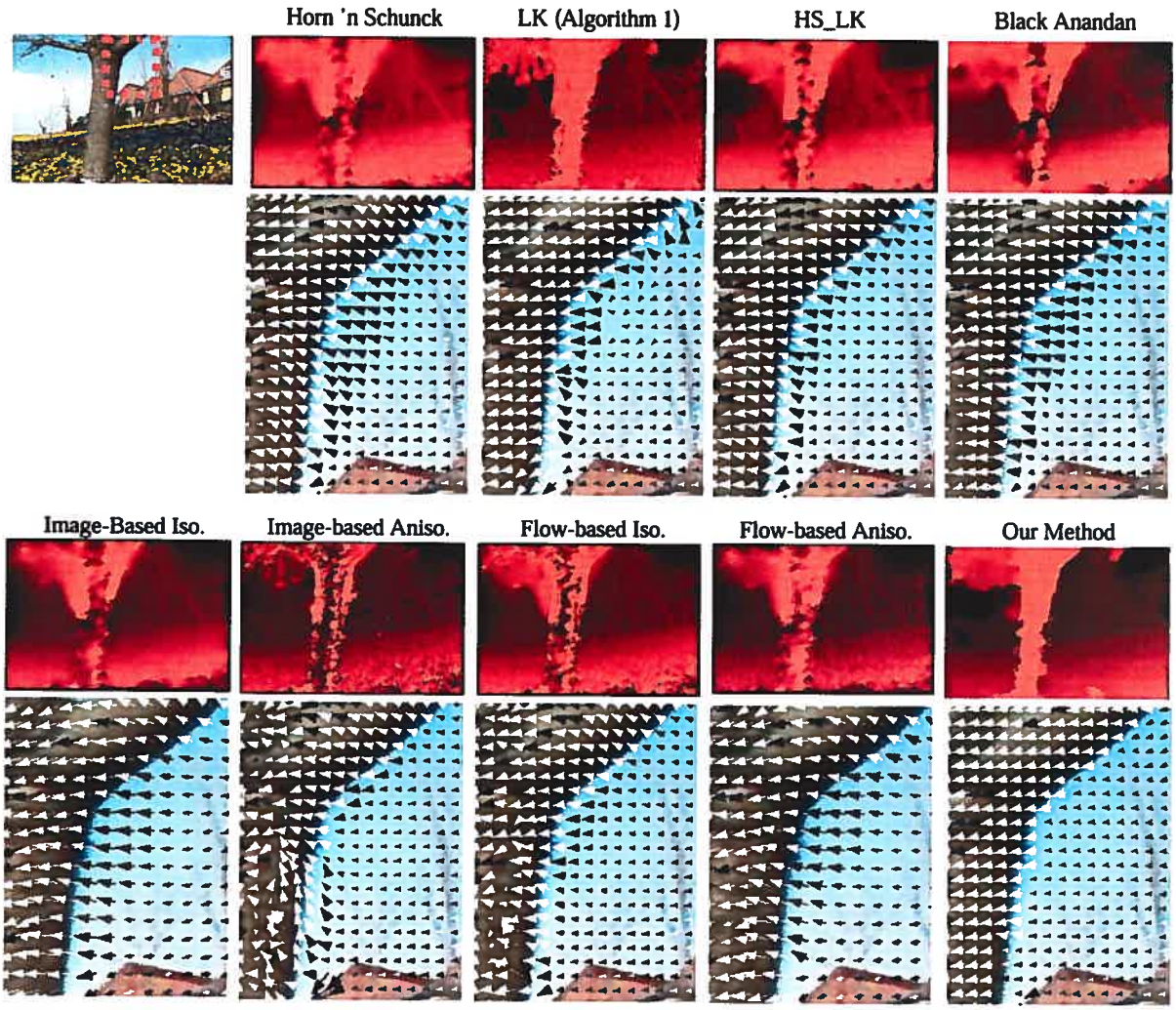

FIG. 1.11 – Results for the FLOWER sequence.

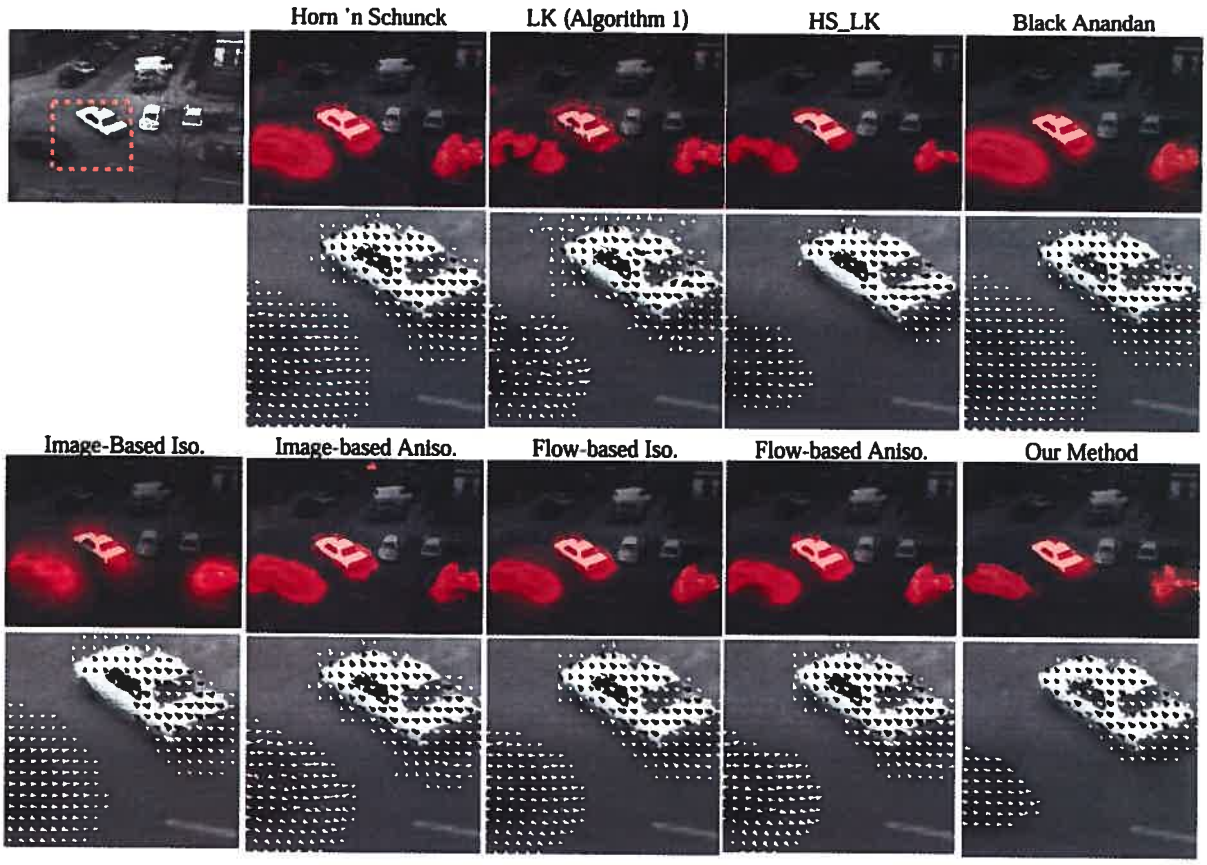

FIG. 1.12 – Results for the TAXI sequence.

and 2.10. Exept for the DIVERGING TREE sequence, our approach appears to be either the best or the second best method according to both quantitative measures. This is true whether the moving objects have a translational (CARS OVER PARK) or a rotational motion (ROTATING BONSAI). Even on highly textured scenes, *i.e.* scenes for which most strong color/intensity edges do not correspond to a motion discontinuity, the results suggest that our method is as good, if not better, than the other ones.

1.5.4 Real Sequences

As shown in Fig. 1.6, five well known image sequences have been used to compare the methods. These sequences are CLAIRE, KARLSRUHE, MOM AND DAUGH-

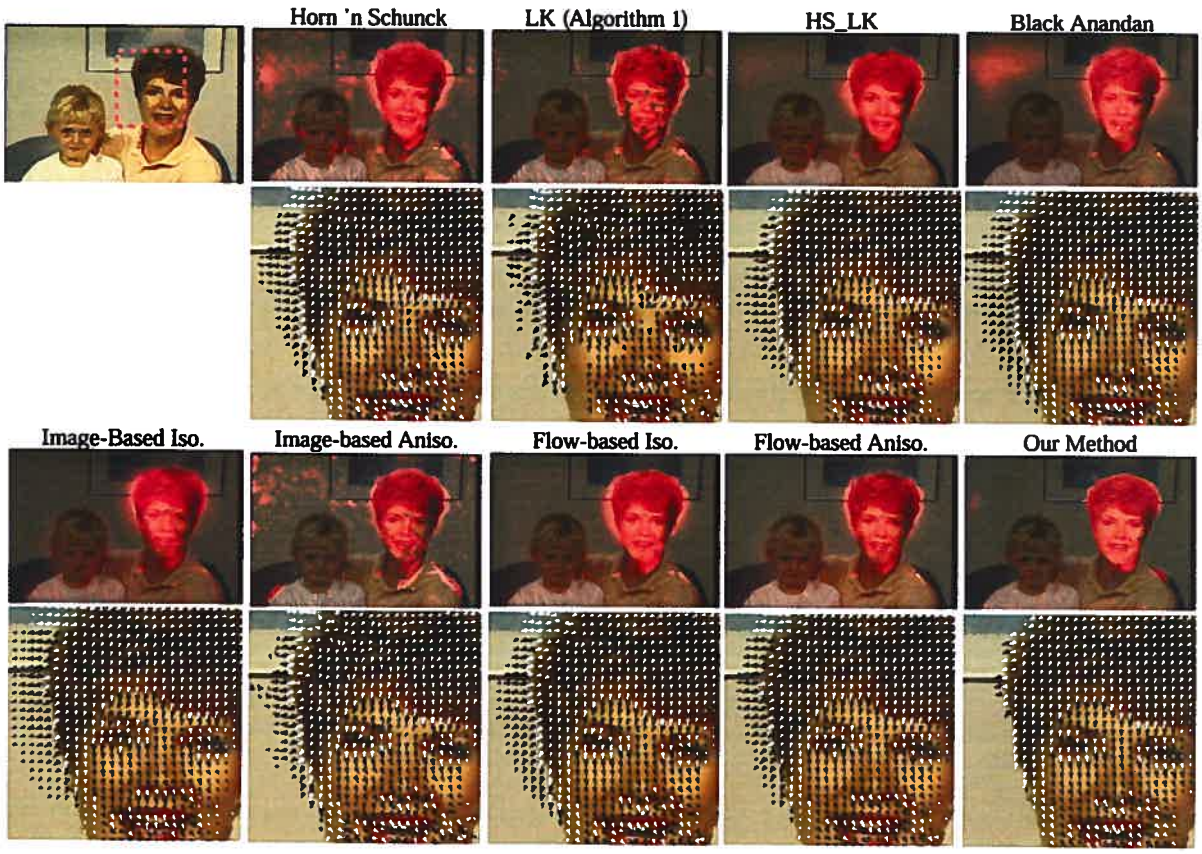

FIG. 1.13 – Results for the MOM AND DAUGHTER sequence.

TER, TAXI, and FLOWER. Qualitative results are presented in Fig. 1.11 to 1.15.

1.6 Discussion

The results presented in the previous section demonstrate that our method is competitive on all kinds on sequences. This includes sequences with global motion (YOSEMITE with and without sky, TRANSLATING and DIVERGING tree) and local motion (KARSHRUE, TAXI, TRANSLATING SHAPES and BONSAI). Our method also shows good performance in estimating translating motion, rotating motion and diverging motion. Our method works well over sequences exhibiting large textureless backgrounds (BONSAI, TRANSLATING SHAPES, and CLAIRE) and on more highly textured sequences (PARTHENON, YOSEMITE, and FLOWER). These results

FIG. 1.14 – Results for the CLAIRE sequence.

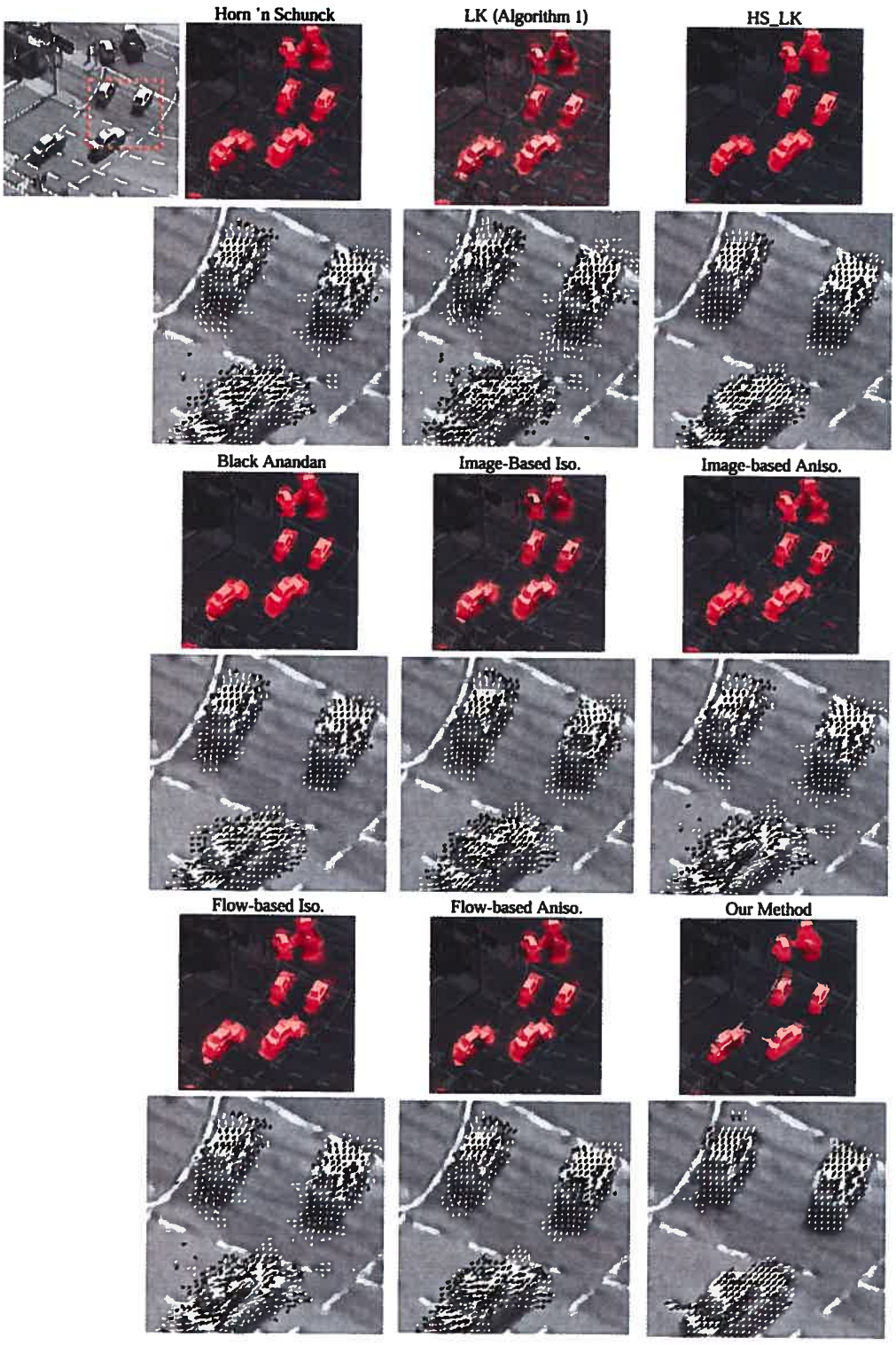

FIG. 1.15 – Results for the KARLSRUHE sequence.

TAB. 1.1 – *Results :* YOSEMITE *sequence with sky.*

Technique	ψ_E	σ_{ψ_E}	ψ_E^e	$\sigma_{\psi_E^e}$
Horn and Schunck	7.35	9.77	−−	−−
LK (Algorithm 1)	8.54	14.94	−−	−−
Black	5.94	9.31	−−	−−
IB-Iso	8.25	9.16	−−	−−
IB-Aniso	8.28	11.55	−−	−−
FB-Iso	7.16	12.80	−−	−−
FB-Aniso	8.65	10.31	−−	−−
HS_LK	7.9	11.20	−−	−−
Our method	6.20	13.7	−−	−−

TAB. 1.2 – *Results :* YOSEMITE *sequence without sky.*

Technique	ψ_E	σ_{ψ_E}	ψ_E^e	$\sigma_{\psi_E^e}$
Horn and Schunck	3.50	4.44	−−	−−
LK (Algorithm 1)	3.42	3.65	−−	−−
Black	2.30	1.66	−−	−−
IB-Iso	5.87	6.31	−−	−−
IB-Aniso	4.35	5.51	−−	−−
FB-Iso	2.92	3.20	−−	−−
FB-Aniso	4.96	6.45	−−	−−
HS_LK	3.66	3.21	−−	−−
Our method	1.81	1.78	−−	−−

illustrates the fact that the combination of a fusion procedure (eq. (1.14)) and an edge avoidance procedure makes the method efficient on various sequences representing different challenges. While the mean-shift-based avoidance procedure preserves sharp motion boundaries, the covariance filter smooths out the vector field and thus minimizes errors due to lack of texture, occlusion and noise. More specifically, the fact that our method relies on the assumption that image and motion boundaries coincide might suggest that our method is error-prone in regions where the number of motion boundaries is larger than the number of image edges (such as in YOSEMITE, TRANSLATING and DIVERGING tree sequences). This would

TAB. 1.3 – *Results :* TRANSLATING SHAPES *sequence.*

Technique	ψ_E	σ_{ψ_E}	ψ_E^e	$\sigma_{\psi_E^e}$
Horn and Schunck	18.7	22.1	31.1	27.6
LK (Algorithm 1)	8.9	19.0	25.8	27.3
Black	10.7	16.8	24.4	22.5
IB-Iso	9.5	15.7	23.7	19.6
IB-Aniso	14.0	19.1	29.2	23.8
FB-Iso	8.0	14.4	19.7	19.8
FB-Aniso	7.2	13.2	18.6	18.2
HS_LK	5.0	12.2	13.0	18.7
Our method	2.3	10.0	6.9	17.3

TAB. 1.4 – *Results :* TRANSLATING TREE *sequence.*

Technique	ψ_E	σ_{ψ_E}	ψ_E^e	$\sigma_{\psi_E^e}$
Horn and Schunck	1.83	2.15	—	—
LK (Algorithm 1)	1.65	4.20	—	—
Black	1.14	1.03	—	—
IB-Iso	4.21	3.84	—	—
IB-Aniso	1.89	1.41	—	—
FB-Iso	1.30	1.64	—	—
FB-Aniso	2.00	2.10	—	—
HS_LK	3.71	3.80	—	—
Our method	0.80	1.68	—	—

certainly be true if only the edge-avoidance procedure (first part of algorithm 2) was used to estimate the flow. However, since the covariance filter is used at each iteration, errors that could eventually be induced by the mean-shift-based motion estimation are significantly reduced. Also, since the magnitude of the mean-shift vector P_s (see section 1.3.3) is limited to N, the avoidance procedure cannot induce a large error. Our method is thus efficient for estimating motion over highly textured scenes.

While testing our approach, we observed two limitations with our method. The first one concerns the processing time. Even by calculating C_s only once during the

TAB. 1.5 – *Results :* DIVERGING TREE *sequence.*

Technique	ψ_E	σ_{ψ_E}	ψ_E^e	$\sigma_{\psi_E^e}$
Horn and Schunck	2.30	2.06	—	—
LK (Algorithm 1)	4.60	3.39	—	—
Black	2.25	1.30	—	—
IB-Iso	6.40	5.01	—	—
IB-Aniso	2.50	3.85	—	—
FB-Iso	2.16	1.79	—	—
FB-Aniso	2.69	2.47	—	—
HS_LK	5.00	3.39	—	—
Our method	2.48	1.71	—	—

TAB. 1.6 – *Results : realistic* CARS OVER PARK *sequence.*

Technique	ψ_E	σ_{ψ_E}	ψ_E^e	$\sigma_{\psi_E^e}$
Horn and Schunck	12.0	18.7	33.0	31.2
LK (Algorithm 1)	9.5	20.6	25.9	38.4
Black	8.0	5.7	25.4	28.4
IB-Iso	13.9	18.4	32.8	29.0
IB-Aniso	10.1	19.2	29.0	35.7
FB-Iso	7.8	15.2	20.0	29.2
FB-Aniso	7.2	16.6	22.1	32.3
HS_LK	9.2	18.3	27.2	34.4
Our method	4.0	16.9	16.2	34.1

first iteration, our method is approximately 5 times slower than LK. Although this might appear to be a major limitation for some applications, there is a technical solution to that problem. In fact, our method can be implemented on a parallel architecture (such as a programmable graphics card for example) andmade to work in interactive time. Such implementation is possible because calculation over each site $s \in S$ (at each stage of the algorithm) is independent of the processing of its neighbors. The parallel implementation would thus be effective for the mean shift calculation, motion estimation, and covariance filtering. Also, as we mentioned previously, another way to accelerate our method is to replace the covariance filter

TAB. 1.7 – *Results : realistic* PARTHENON *sequence.*

Technique	ψ_E	σ_{ψ_E}	ψ_E^e	$\sigma_{\psi_E^e}$
Horn and Schunck	12.2	18.5	18.2	21.7
LK (Algorithm 1)	14.7	24.1	22.5	28.3
Black	10.4	18.1	15.5	21.4
IB-Iso	16.0	17.6	21.3	19.0
IB-Aniso	14.1	21.2	18.7	23.8
FB-Iso	10.7	18.0	16.1	21.0
FB-Aniso	11.0	16.9	17.0	19.5
HS_LK	13.1	16.7	17.3	19.4
Our method	9.80	20.7	15.2	25.5

TAB. 1.8 – *Results : realistic* ROTATING BONSAI *sequence.*

Technique	ψ_E	σ_{ψ_E}	ψ_E^e	$\sigma_{\psi_E^e}$
Horn and Schunck	22.1	22.3	36.1	25.2
LK (Algorithm 1)	10.4	21.5	25.4	28.1
Black	14.1	19.4	30.1	23.0
IB-Iso	15.6	18.5	28.9	21.1
IB-Aniso	22.8	21.9	37.0	25.1
FB-Iso	13.7	19.2	29.1	22.3
FB-Aniso	11.8	20.9	27.2	25.3
HS_LK	8.8	15.8	20.3	20.9
Our method	5.3	15.0	13.2	20.1

by a simple Gaussian low-pass filter. This would greatly improve the processing times, although at the expense of precision.

The second limitation of our method comes from the fact that, in some specific cases, despite the covariance filter, some mean-shift vectors P_s induce errors that could not be compensated for. This is typically true when the intensity edges of two objects crosses. Fig. 1.18 shows two examples for which a background edge is taken as a part of the moving object. As mention previously though, since the magnitude of the mean-shift vectors is limited to N these errors are very local. However, one way to minimize this problem would be to include a motion segmentation step to

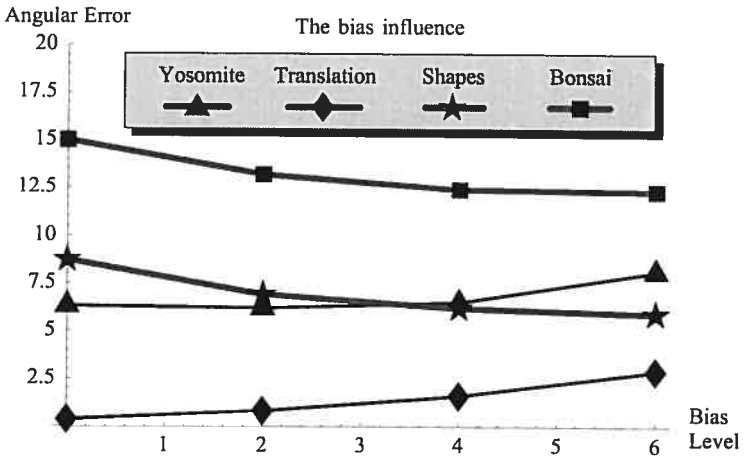

FIG. 1.16 – The effect of the noise level variable on the angular error of four sequences.

the motion estimation process. In this way, only the mean-shift vectors P_s located in the vicinity of a moving edge would be retained. With such modification, our approach would then become a motion segmentation/estimation method.

Finally, we have tested the influence of the most important variables of our method. These variables are the neighborhood window size N and the noise level N_l. As mentioned before, for every example presented in this paper, N was set to 9 or 11 and the noise level between 2 and 5. We observed that the resulting vector fields react smoothly to a change of these variables and that other values could have been used. To illustrate this assertion, Fig. 1.17 presents different vector fields obtained on the CLAIRE and the TAXI sequences with different values for N and for the noise level variable. As shown, the resulting vector fields react smoothly to a change of these two variables. Also, in Fig 1.17, we plotted the angular error (ψ_E) of four sequences processed with different noise levels. For those highly textured sequences (YOSEMITE and TRANSLATION TREE) a larger noise level tends to raise slightly the angular error whereas for less textured sequences, large noise level clearly reduces the angular error (ψ_E). In other words, the use of such a bias makes our method estimate vector field with 100% density without inducing any significant error, even

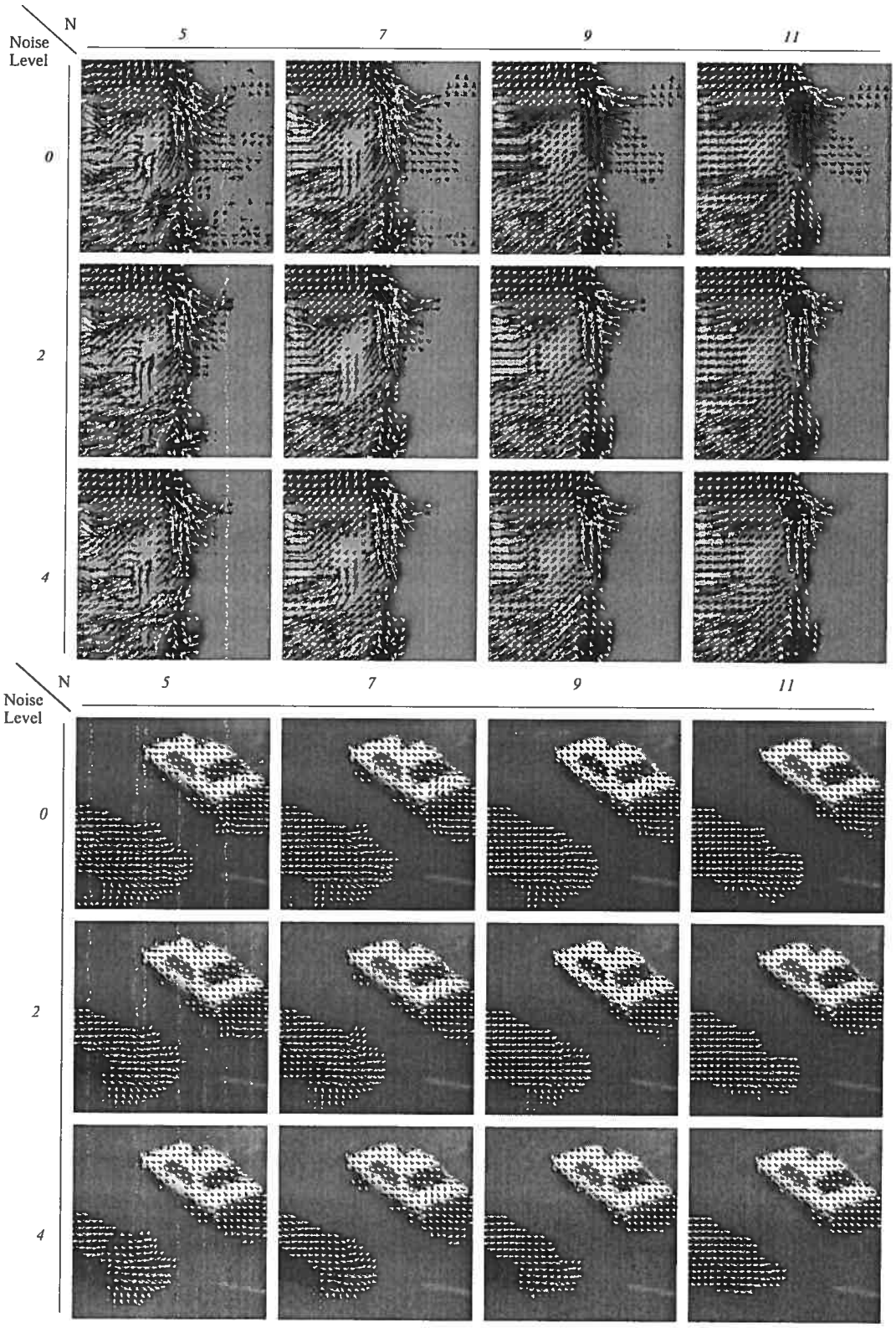

FIG. 1.17 – Vector fields obtained with our method with different neighborhood window size N and different noise levels.

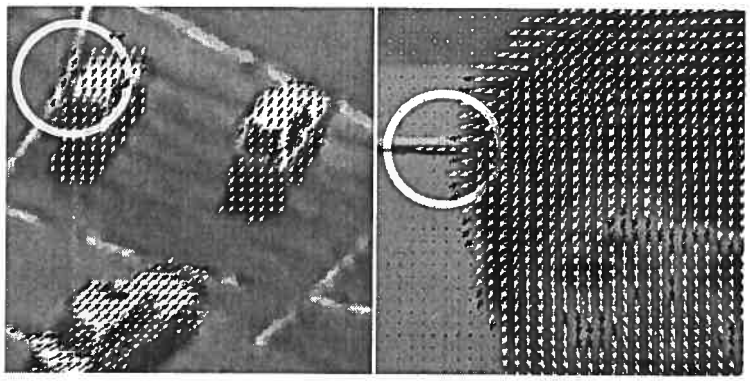

FIG. 1.18 – Two examples for which the Mean-shift-based avoidance procedure has locally induced an error.

on highly textured sequences that might require little or no bias.

Acknowledgments

The authors are very grateful to Amar Mitiche for his constructive comments and to John Barron for his precious help.

CHAPITRE 2

FUSION DE CHAMPS D'ÉTIQUETTES COMPLÉ-MENTAIRES

Article : Label field fusion in computer vision

Cet article a été accepté avec modifications mineures au journal *IEEE Transactions on Image Processing* comme l'indique la référence bibliographique

P-M Jodoin, M. Mignotte, C. Rosenberger[1] Label field fusion in computer vision *IEEE Transactions on Image Processing*, accepté, 2007.

Cet article est présenté ici dans sa version originale.

Résumé

Dans ce chapitre, nous proposons une nouvelle approche de fusion de données appliquée à l'imagerie. La méthode ici présentée fusionne des champs d'étiquettes au lieu de fusionner des données brutes comme le font traditionnellement les algorithmes d'apprentissage statistique. L'objectif de cette méthode est de contraindre les régions d'un champ d'étiquettes à épouser la forme d'objets présents dans une scène. Nous démontrons que cette approche aide sensiblement à régulariser les résultats obtenus en segmentation de mouvement, en flux optique et en détection d'occlusions. Notre méthode prend donc en entrée deux champs d'étiquettes : une carte de régions et un champ d'étiquettes d'application. Ces deux champs sont fusionnés ensemble afin de contraindre les régions du champ d'application à épouser la forme des objets décrits dans la carte de régions. Cette procédure de fusion s'exprime sous la forme d'une fonction d'énergie globale minimisée à l'aide de l'optimiseur déterministe Iterative Conditional Mode (ICM). La fonction d'énergie peut être une fonction de fusion dite *pure*, ou encore une fonction de type *fusion-réaction*. Pour cette dernière, un terme d'attache aux données est ajouté afin de rendre le problème bien posé. Nous croyons que la simplicité conceptuelle de notre méthode,

[1]Christophe Rosenberger est professeur à l'École Nationale Supérieure d'Ingénieurs de Bourges

le faible nombre de paramètres et l'utilisation d'un optimiseur déterministe rapide pouvant être implémenté sur une architecture parallèle constituent les principaux avantages de notre méthode.

Abstract In this paper, we propose a novel fusion framework that combines *label fields,* instead of observation data, as is usually the case. Our framework takes as input two label fields : a quickly-estimated and to-be-refined segmentation map and a spatial region map that exhibits the shape of the main objects of the scene. These two label fields are combined with a global energy function that is minimized with a deterministic ICM algorithm. As explained in the paper, the energy function may implement a pure fusion strategy or a fusion-reaction function. In the latter case, a data-related term is used to make the optimization problem well-posed. We believe that the conceptual simplicity, the small number of parameters, and the use of a simple and fast deterministic optimizer that admits a natural implementation on a parallel architecture are among the main advantages of our approach. Our fusion framework is adapted to various computer vision applications among which are motion segmentation, motion estimation and occlusion detection.

2.1 Introduction

In the field of imagery, computer vision is frequently considered as a research area in which applications aim at estimating high-level models learned from input images. This is the case for applications such as stereovision [63], motion estimation [100] and motion detection whose goal is to estimate depth labels, optical flow vectors and the presence (or absence) of motion in a video sequence. Most methods used to solve these kinds of imagery problems are built upon a to-be-optimized energy function made up of low-level image features such as color, spatial gradient or texture features. Years of research have demonstrated that significant improvements may be achieved by using more complex features (*e.g.*, wavelets coefficients instead of Fourier coefficients), better designed energy models (*e.g.*, robust instead of quadratic or L1-norm energy functions) and better optimizers (stochastic instead of deterministic optimizers and/or a multiresolution instead of monoresolution optimization schemes).

With the ever growing computational power of modern computers, researchers tend to use an increased number of features to enforce the result accuracy. The main advantage of using many features resides in the fact that many features blended together often mutually compensate for their respective limitations. For example, when segmenting cluttered color images, the use of color *and* texture features has shown great improvement as compared to color-only or texture-only segmentation approaches [167].

But using numerous features raises the question of how these metrics can be fused together. Probably the most intuitive and simple solution is to fuse those features inside one single N-dimensional vector. In this perspective, each input pixel may be associated not only with 3D RGB color values, but also with texture values, gradient values, edge data or any other input features. Among the applications that benefit from this approach of "all features in one vector" are motion segmentation [144], texture segmentation [19,35,72,114], image retrieval [133,159] and face recognition [27,66,111] to name a few.

However, despite the obvious advantages of using high dimensional data vectors, an increased number of features raises new challenges. One such challenge is the well known problem of the "curse of dimensionality" [23, 131] related to the rapid increase of extra dimensions. David Donoho [38] points out that if we consider a unit dimension divided in bins of size 1/10, a minimum of 10 points is needed to fill each bin with at least 1 point. However, for a 20-dimension unit hypercube, no less than 10^{20} points are needed to fill the bins. This means that an increase in dimension often means a need for more input data which, for some applications, is not realistic.

The typical solution used to avoid the curse of dimensionality is to reduce the number of dimensions. To do so, one may try to identify the "right" features in the stream of input data and minimize the dimensionality by getting rid of the "less useful" features [9, 19, 23, 67]. One simple but efficient way for selecting features is to retain a subset of X features that best help the algorithm produce precise results [23]. Although this approach is viable in many applications, it has the disadvantage of requiring a training data set and thus being deficient for unsupervised applications. Other approaches for reducing dimensionality focus more on the "right" data space dimension than on the "right" feature space. This is the case for methods such as principal component analysis (PCA) [19, 23, 35, 72, 111, 131], singular value decomposition (SVD) [133] and the Fisher linear discriminant (FLD) [67, 131] which projects linearly the input data onto a lower dimensional subspace. In cases where features have complicated interactions, a nonlinear component analysis may also be considered [131]. Another way to deal with high-dimensional data is to give more *weight* to some features and less to others. This strategy has been used by Pichler *et al.* [35, 114] with their Feature Contrast method and indirectly by many authors who implement fuzzy logic-based fusion procedures [8, 53, 60, 158]. In a similar perspective, some researchers have underlined the fact that data gathered form many sensors can be fused together with the help of Dempster-Shafer theory of evidence [4, 161]. Also, feature weighting is what back-propagation training algorithms (neural networks) are meant to do [23, 131]. However, since a training

data set is needed to weigh the input features, this approach is not suited to all applications.

Another problem that often occurs when fusing different features concerns the need for *axis rescaling*. This situation occurs when features with different units are blended together, making the N-dimensional space anisotropic. This typically happens when blending, say, RGB values ranging between $(0, 0, 0)$ and $(256, 256, 256)$ with motion vectors ranging between $(-5.0, -5.0)$ and $(5.0, 5.0)$. In this case, the usual similarity measures (used to evaluate the distance between data points) will give an overwhelming importance to features having a larger unit range such as the RGB values. Although data normalization [35, 131] may be used to rescale the axes, for some applications rescaling may have the effect of reducing the class separability [131].

To alleviate dimensionality problems, other methods use one large energy function [54,77,99,144,148,150] made of a series of smaller energy functions built around one (or sometimes two) specific features. A typical example is the method used by Heitz and Bouthemy [54] in which a large Markovian energy function composed of five gradient-based, edge-based and motion-based energy functions is minimized. This kind of approach has the advantage of using multiple features without having to deal explicitly with the inherent problems related to multiple dimensions. It also allows intuitive and yet *ad hoc* energy function formulation. However, despite the undeniable advantages of using large energy functions, these applications often contain many parameters to tweak. Furthermore, large and complex energy functions are more likely to have an erratic profile with several local *minima* that are not trivial to minimize, especially with a deterministic optimizer.

Another family of fusion approaches used in imagery are the so-called *decision fusion* approaches [10,74,143]. With these methods, a series of energy functions are first minimized before their outputs (their decisions) are merged. In this case, the energy functions are defined on different basic features and/or different cost functions. With this perspective, Reed *et al.* [147] successfully implemented a similar fusion method to segment sonar images. Their method fuses segmentation maps

involving identical classes (here, segmentation maps of the seafloor in sonar imagery) with a voting scheme followed by a Markov Random Field (MRF) in-painting procedure.

The last data fusion trend that we mention involves the so-called region-based approaches that are typically used in stereovision [63,77,96,103,166], motion segmentation [120,162,164], motion estimation [26,100] and image deconvolution [109]. Region-based methods generally use a region map R initially obtained after segmenting an input image into regions of uniform color. Under the assumption that these regions contain precise information on the main objects of the scene, the regions are used to help regularize the optimization process. Although some of these methods implement a *soft* region-based constraint [77], an imprecise region map R often generates errors that are difficult if not impossible to compensate for.

In this paper, we propose a new fusion framework that mixes together label fields instead of features; label fields containing different and yet complementary information. More specifically, our framework takes as input two label fields : a region map (called r) obtained after segmenting one (or two) input images into regions of uniform color, and a rough estimate (called $x^{[0]}$) of the application label field. Note that $x^{[0]}$ is application specific and may contain occlusion labels, motion labels, or any other high-level information. Once r and $x^{[0]}$ have been estimated, they are merged together with a fusion procedure expressed as an energy function minimization. Here, the optimization process searches for a new label field $\hat{x}_{opt}$ whose content is close to that of $x^{[0]}$ but adapted to fit the regions of r. As is the case for most conventional region-based methods, r is assumed to contain precise information on the overall shape of the scene. However, by the very nature of our fusion procedure, our framework is tolerant to imprecisions in r and reacts smoothly to any modification of its parameters. Let us mention that the to-be-minimized energy function U may be a pure *fusion* function or a *a fusion-reaction* function including a data-related term. In both cases, our framework can be implemented on a parallel architecture such as a graphics processor unit (GPU).

To our knowledge, fusion of label fields, involving labels of different natures

(i.e., classes estimated with different image features), has never been proposed in the literature before, and/or developed (up to now) into a coherent theoretical framework.

In this work, we will show that a MRF framework can be efficiently used to fuse, in a versatile way, the knowledge of these two preliminary label maps through a data-related term and a regularizing prior. The latter term measuring the joint spatial homogeneity and interactions of this couple of label fields and allowing to infer, in the Maximum *a posteriori* (MAP) sense, the final segmentation map to be estimated.

Besides, since we use two label fields instead of observation data as is usually the case, our a priori knowledge about these interactions can be efficiently expressed at a higher level of abstraction into the MRF model (or within the prior Gibbs distribution). In our fusion MRF model, it allows to originally integrate a priori information at different levels of representation (pixel and regions), thus efficiently modeling the spatial coherence of the different regions preliminary segmented with different image features. We also believe that the conceptual simplicity, the small number of parameters, the absence of dimensionality problems, and the scalability of our approach are appealing advantages of our framework.

The rest of the paper is organized as follows. In Section 3.2, our framework is first introduced and summarized with an algorithm. Then, Section 2.3 illustrates three applications (namely, motion segmentation, motion estimation, and occlusion detection) that can benefit from our framework. Finally, some experimental results are presented in Section 4.8. Section 3.6 draws conclusions.

2.2 Label Field Fusion

2.2.1 Framework

As mentioned previously, our method fuses together two label fields, both estimated with different low-level image features [121, 122]. The reason for blending label fields is to alleviate dimensionality problems and, more specifically, the re-

scaling problem that arises when blending features with different units such as color and motion vectors for example. Furthermore, since the two label fields are estimated separately, they allow the minimization of simple energy functions (*i.e.*, functions with few local *minima*) defined on few features.

The first label field considered is a region map $r = \{r_s | s \in S\}$ defined on a rectangular lattice S made up of $\mathcal{N} \times \mathcal{M}$ sites. The map r is obtained by segmenting one (or two) input images into uniform regions. Here, "uniformity" may be defined in the sense of color, texture or any other image features that best suit the image content. For the purpose of this paper, r is estimated based on the color feature. Every element r_s in r takes a value in $\Lambda = \{\omega_0, \omega_1, ..., \omega_{c-1}\}$ where "c" is the number of color classes in which the input image is segmented. In this way, every input pixel associated with a given class $r_s = \omega_i$ has a color with distribution $P(_{\text{color}}|\omega_i)$.

The second input label field for our framework is the so-called *application label field i.e.*, a field made of application-specific labels. For instance, this label field (which we call $x^{[0]}$) may contain motion labels, optical flow labels, occlusion labels or any label specific to the current application. As is the case for r, $x^{[0]}$ is defined on a rectangular lattice of size $\mathcal{N} \times \mathcal{M}$ whose sites take a value in $\Gamma = \{\zeta_0, \zeta_1, ..., \zeta_{d-1}\}$ where d is the number of categories. For instance, for a typical occlusion detection application, $d = 2$ and $\zeta_0 = \{\text{No\_Occlusion}\}$ and $\zeta_1 = \{\text{Occlusion}\}$. In this case, occlusion is related to stereovision or video applications and refers to pixels that are visible in one image but occluded in a second image (c.f. Fig. 2.5).

In our framework, $x^{[0]}$ is a rough estimate of the *true* label field that we wish to estimate. Thus, $x^{[0]}$ is typically obtained with a simple method and may be imprecise near edges, and contain false positives and false negatives due to the presence of noise or the lack of texture. In fact, $x^{[0]}$ serves as an initialization for the ensuing iterative fusion process (hence the "[0]" exponent).

Once r and $x^{[0]}$ have been estimated, they are blended together with a fusion procedure (see Fig. 2.1 for a schematic view). The goal of this fusion procedure is to modify the content of $x^{[0]}$ based on the regions of r which describe the overall shape of the predominant objects of the scene. Here, by the very nature of r and

$x^{[0]}$, it is reasonable to assume that the regions of the to-be-estimated label field $\hat{x}$ follow the ones in r. In other words, we assume that no transition in $\hat{x}$ occurs inside a uniform region of r and thus that the edges in $\hat{x}$ correspond to edges in r.

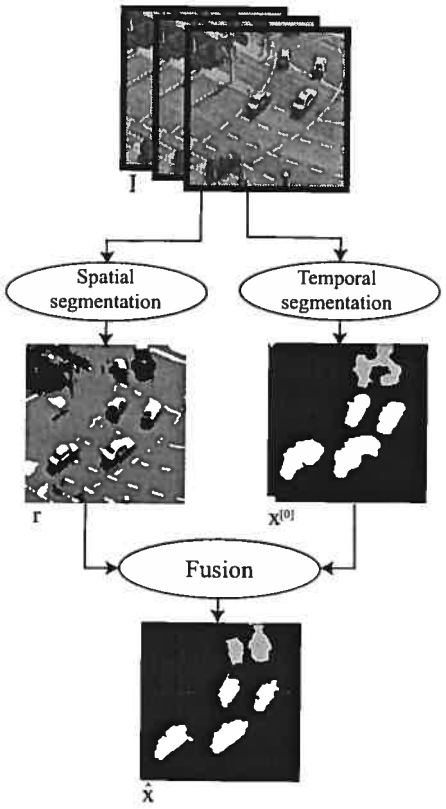

FIG. 2.1 – Schematic view of our fusion framework adapted to motion segmentation. In this example, the fusion of the region map r and the motion segmentation field $x^{[0]}$ makes the resulting label field $\hat{x}$ more precise and less blobby.

2.2.2 Fusion Procedure

In the light of the assumption described at the end of the previous section, the validity of a solution $\hat{x}$ may be evaluated by measuring how well the regions in $\hat{x}$ follow the ones in r. Locally, this means that a uniform section in r will also be uniform in $\hat{x}$ and that an edge in $\hat{x}$ should also correspond to an edge in r. In other

words, no transition in $\hat{x}$ is expected to occur inside a uniform region of r.

In our model, we assume that r and $\hat{x}$ are realizations of a pair of joint Markov Random Fields (MRF) and that, by the properties of the Hammersley-Clifford theorem [71], the joint probability $P(r_s, \hat{x}_s|\Psi_s)$ is a Gibbs distribution such as

$$P(r_s, \hat{x}_s|\Psi_s) = \frac{1}{Z}\exp(-E_s(r, \hat{x}|\Psi_s)) \tag{2.1}$$

where Z is a normalization factor and $E_s(r, \hat{x}|\Psi_s)$ is a local energy function measuring how well $\hat{x}$ and r fit together around site s. Also, Ψ_s is the $L \times L$ local joint neighborhood surrounding site s in both label fields $\hat{x}$ and r. In the absence of any prior geometric knowledge on $\hat{x}$ and r, we define the potential function E_s as :

$$E_s(r, \hat{x}|\Psi_s) = -\sum_{t\in\Psi_s} \delta(r_t, r_s)\delta(\hat{x}_t, \hat{x}_s) \tag{2.2}$$

where $\delta(a, b)$ is the Kronecker delta function ($\delta(a, b)$=1 if $a = b$ and 0 otherwise). Note that this potential function to some extent resemble the Potts model, *i.e.*, an N-class generalization of the well-known Ising model [141]. Equation 2.2 is what we call the *fusion model* that, by its very nature, measures the spatial homogeneity of the joint couple of Markovian random fields x and r. More specifically, E_s is small when the regions in $\hat{x}$ fit locally the regions in r, *i.e.* when no edges in $\hat{x}$ cross a uniform region in r. Also, E_s can be seen as a function that counts the number of neighbors "t" around site "s" whose label "r_t" and "x_t" is the same as "r_s" and "x_s". In this way, the best label field $\hat{x}$ may be expressed as

$$\hat{x} = \arg\min_{x} \sum_{s\in S} E_s(r, x|\Psi_s). \tag{2.3}$$

Although this formulation can produce very decent results ($\hat{x}$ in Fig. 2.1 has been computed with equation (2.3)) it nonetheless contains a weakness. In fact, E_s allows more than one global *minimum* that corresponds to trivial and uninteresting

solutions such as the constant label field $\hat{x}_s = \zeta_0$, $\forall s \in S$. Of course, most of the time when Eq.(2.3) is minimized with a deterministic downhill search algorithm, the estimated solution $\hat{x}$ lies in a *minima* close to the initial estimate $x^{[0]}$. However, when minimizing Eq.(2.3) with a stochastic optimizer such as simulated annealing, the resulting solution $\hat{x}$ may be quite far from $x^{[0]}$ and thus not useful in practice. To overcome this problem, a *reaction* term is added to E_s to make sure the problem is well-posed and that there exists only one global solution. Mathematically, this is formulated as

$$E'_s(r, \hat{x}, x^{[0]}|\Psi_s) = -\alpha\delta(x_s^{[0]}, \hat{x}_s) - \sum_{t\in\Psi_s} \delta(r_t, r_s)\delta(\hat{x}_t, \hat{x}_s) \tag{2.4}$$

where $\delta(x_s^{[0]}, \hat{x}_s)$ can be viewed as a *reaction* term and α is a constant. In this case, the best label field $\hat{x}$ is given by

$$\hat{x} = \arg\min_x \sum_{s\in S} E'_s(r, x, x^{[0]}|\Psi_s). \tag{2.5}$$

Since neither Eq.(2.5) nor Eq.(2.3) have an analytical solution, we use Besag's Iterative Conditional Mode (ICM) optimization algorithm [71] to estimate $\hat{x}$ as shown in Algorithm 1. ICM is a deterministic update optimization algorithm introduced by Besag [71] to optimize the energy function of a Gibbs distribution. More precisely it consists in finding the conditional modes, i.e., for each site, the value that maximizes the local conditional probability density function (PDF). This deterministic algorithm is not guaranteed to find the global *minima*; nevertheless, it drastically reduces computational time as compared to stochastic relaxation techniques such as simulated annealing.

The way our fusion procedure works is illustrated in Fig.2.2 (in this case, $\alpha = 0$ in this example). In image r, site γ is part of the black class (which is a section of the moving vehicle) but has the *static* label in $x^{[0]}$. When considering the sites that are *part of the black section of the vehicle in* r inside Ψ_γ, we see that a majority of

Fusion Procedure

I	Input image
r	Region map
$x^{[k]}$	Label field after the k^{th} iteration
Ψ_s	$L \times L$ neighborhood centered on site s
$\delta(a,b)$	Kronecker delta

1. Initialization
Estimate $x^{[0]}$ based on features picked in I
$r \leftarrow$spatial segmentation of I into "c" classes
$k \leftarrow 0$

2. ICM Optimization (Fusion)
do
  $k \leftarrow k+1$
  for *each site* $s \in S$ **do**
    for *each class* $\zeta_i \in \Gamma$ **do**
      $tab[\zeta_i] \leftarrow -\alpha\delta(\zeta_i, x^{[0]})$
        $-\sum_{t\in\Psi_s}\delta(\zeta_i, x_t^{[k]})\delta(r_s, r_t)$
    $x_s^{[k]} \leftarrow \arg\min_{\zeta_i\in\Gamma} tab[\zeta_i]$
while $x^{[k-1]} \neq x^{[k]}$

Algorithm 1: Our fusion-reaction algorithm.

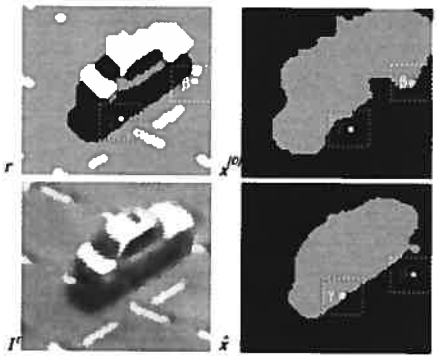

FIG. 2.2 – Zoom on KARLSRUHE sequence. Top left is the label field r and top right is motion label field $x^{[0]}$. In this example, the motion label field $x^{[0]}$ contains two classes which can be understood as the "static" and the "moving upward" classes. Bottom left is the image frame at time t while bottom right shows the motion label field at convergence (here $\alpha = 0$). Note how the region in $\hat{x}$ is well localized as compared to the one in $x^{[0]}$.

those sites have a *mobile* label in $x^{[0]}$. In other words, within the nearest neighbors around site γ with a black label in r, there is a majority of *mobile* sites. For this reason, after minimizing the energy function E', the site γ is assigned a *mobile* label in $\hat{x}$. Note that since $\alpha = 0$ in this example, each ICM iteration of our fusion method works in a similar way to the well-known K nearest neighbor algorithm does.

2.2.3 Markovian Segmentation

As mentioned previously, r and $x^{[0]}$ are label fields estimated with different image features. Since $x^{[0]}$ contains application-specific labels, we will see in Section 2.3 how it can be estimated in the context of three specific applications. As for r, although any valid segmentation algorithm may be used to estimate it, we use a Markovian approach that we shall describe in the following paragraphs. The reason for this choice is twofold. First, the segmentation method we have implemented is *unsupervised* and thus requires no human intervention at runtime. Second, this segmentation method can be parallelized and implemented on a parallel architecture

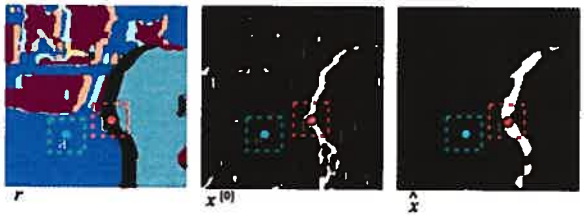

FIG. 2.3 – Zoom on the TSUKUBA scene. After fusing r and $x^{[0]}$, the number of isolated false positives and false negatives in $x^{[0]}$ is significantly reduced because the region map r is locally homogeneous.

such as a graphics processor unit (GPU) [123]. With such an implementation, r can be computed in interactive time.

Let us consider $Z = \{R, I\}$, a pair of random fields where $R = \{R_s, s \in S\}$ and $I = \{\vec{I}_s, s \in S\}$ represent respectively the color label field and the observation field, both defined on a $\mathcal{N} \times \mathcal{M}$ lattice $S = \{s = (i,j) | i \in [0, \mathcal{N}[, j \in [0, \mathcal{M}[\}$. Here, I is a known input image and r (a realization of R) is to be estimated. As mentioned previously, r_s takes a value in $\Lambda = \{\omega_0, \ldots, \omega_{c-1}\}$, where c is the number of color classes. Note that $\vec{I}_s$ is a $3D$ vector for color images and a scalar for grayscale images.

Segmentation can be viewed as a statistical labeling problem, *i.e.*, a problem for which each observation vector $\vec{I}_s$ most be associated with the *best* color class $r_s = \omega_i \in \Lambda$. Thus, inferring a label field can be seen as an optimization problem that searches for *the best* r in the sense of a given statistical criterion. Among the available statistical criteria, the Maximum *a posteriori* criterion states that a label field r is optimal according to I when it maximizes the *a posteriori* PDF : $P(r|I)$. In this way, r is optimal whenever $r = \arg\max_{r'} P(r'|I)$ [71].

Because $P(r|I)$ is often complex or undefined, it is common to assume that r and I are realizations of MRFs and that, by the Hammersley-Clifford theorem [71], the posterior distribution is defined by a Gibbs distribution of the form $P(r|I) \propto \exp -U(r, I)$ where $U(r, I)$ is an *energy* function [71]. By the properties of the Bayes

theorem [23], the *a posteriori* distribution can be represented as

$$P(r|I) \propto \exp\{-(U_1(r,I) + U_2(r))\} \tag{2.6}$$

where U_1 and U_2 are the likelihood and prior energy functions. By assuming independence between the random variables $\vec{I}_s$ (*i.e.*, $P(I|r) = \prod_{s\in S} P(\vec{I}_s|r_s)$), the corresponding posterior energy to be minimized is

$$U(r,I) = \sum_{s\in S}\Big(\underbrace{L_s(r_s,\vec{I}_s)}_{U_1(r_s,I_s)} + \beta \underbrace{\sum_{<s,t>}[1-\delta(r_s,r_t)]}_{U_2(r_s)} \Big),$$

where U_2 corresponds to the isotropic Potts model. Here, $\delta(a,b)$ is the Kronecker function, β is a constant, $<s,t>$ is a set of binary *cliques*, and $L_s(r_s,\vec{I}_s) = -\ln P(\vec{I}_s|r_s)$. Note that the cliques defined here are on a second-order neighborhood.

The conditional distribution $P(\vec{I}_s|r_s)$ models the distribution of the observed data $\vec{I}_s$ given a class $r_s \in \Lambda$. In this paper, this distribution is modeled with a Normal law that depends on the two parameters $(\vec{\mu}_{r_s}, \Sigma_{r_s})$. Since there are c different classes, there are c different Normal laws and a total of $2c$ Gaussian parameters $\Phi = [(\vec{\mu}_0,\Sigma_0)\ ,\ldots,(\vec{\mu}_{c-1},\Sigma_{c-1})]$. Because these parameters are initially unknown, they need to be estimated. To this end, we use a Markovian and stochastic method called Iterated Conditional Estimation (ICE) [160]. Note that the K-means or the EM algorithms could have also been used.

Once Φ has been estimated with ICE, r can be obtained by minimizing the global energy function U :

$$r = \arg\min_{r'} U(r',I). \tag{2.7}$$

To do so, we again use Besag's ICM algorithm [71]. For more details on our implementation of ICM and ICE, please refer to [123].

2.3 Computer Vision Applications

In this Section, three computer vision applications are described and adapted to our fusion framework. These applications are motion segmentation, motion estimation / segmentation, and occlusion detection. As the name suggests, motion segmentation is a procedure that groups together pixels having a uniform displacement in a video sequence. As for motion estimation, it refers to the task of estimating the optical flow visible in a video sequence. Since our motion estimation procedure estimates a parametric flow together with a motion label field, we call this operation *motion estimation/segmentation*. Finally, occlusion detection is a procedure that takes as input two images and that locates the pixels that are visible in one image but occluded in the second image. Consequently, occlusion detection is closely related to optical flow and stereovision.

2.3.1 Motion Segmentation

2.3.1.1 Introduction

Motion segmentation refers to the general task of labeling pixels with uniform displacement [48, 135]. Consequently, motion segmentation has often been linked to motion estimation. Actually, a common way to segment an image sequence is to estimate an optical flow field and then segment it into a set of regions with uniform motion vectors. Such an approach is sometimes called *motion-based* [48] since segmentation is performed on the basis of displacement vectors only. This kind of segmentation is rather easy to implement and generates more accurate results than, say, an 8×8 block segmentation procedure.

To enforce precision, some authors propose segmentation models based on additional features, such as brightness and edges. These models are sometimes referred to as *spatio-temporal* segmentation techniques. In this context, Black [99] presented an MRF approach that minimizes a three-term energy function using a stochastic relaxation technique. In Black's work, the motion label field is estimated on the basis of motion *and* intensity. In [164], Altunbasak *et al.* proposes a region-based

motion segmentation approach. Assuming that color regions are more accurate than the motion regions, a region-based motion segmentation is performed, whereby all sites contained in a color region are assigned the same motion label. In a similar vein, Bergen and Meyer [95] show that an image segmentation may be used to eliminate error due to occlusion in an animated scene. For completeness, let us also mention the work by Khan and Shah [144] in which a MAP framework is proposed to softly blend color, position and motion cues to extract motion layers. In this contribution, each cue has its own PDF. These PDFs are combined together with feature weights that give more or less importance to a cue depending on certain specified observations.

2.3.1.2 Motion Segmentation and Our Framework

As mentioned in Section 3.2, our framework is supplied with two label fields : r, a region map and $x^{[0]}$ a motion map. Although $x^{[0]}$ could be obtained with any valid motion segmentation approach, we decided to use the same unsupervised statistical Markovian procedure that we use to compute r. In this way, $x^{[0]}$ is obtained by segmenting V, a vector field computed with an iterative and multiresolution version [87] of the well known Lukas-Kanade algorithm [22, 70]. Note that for this segmentation, the vector field V stands for the observation field that we called I in Section 2.2.3 and that every element $\vec{V}_s$ is a two-dimensional real vector. For every sequence we have tested, V was computed with a two-level pyramid and an integration window of size 7×7 pixels [87].

2.3.2 Motion Estimation

2.3.2.1 Introduction

Motion estimation is one of the most studied area in computer vision. Among the solutions proposed for this problem, let us mention variational methods [21, 56, 62, 80, 83, 93, 150], local methods [6, 21, 51, 79], frequency-based methods [107], correlation-based methods [115], phase-based methods [39] and Markovian methods

[49, 75].

Another class of motion estimation algorithms included those assuming that the overall motion in a video sequence is piecewise parametric [31,48], *i.e.* that the motion field may be divided into regions whose motion can be expressed with a parametric motion model. Thus, the goal of these approaches is to jointly estimate the motion regions together with their associate parametric motion model. To this end, the motion regions and the motion model parameters are generally estimated with a two-step procedure [30,104,117] that iterates until convergence. The first step consists in estimating the motion model parameters according to the current motion label field [31,86]. In contrast, the second step estimates new motion regions while the motion models are kept unchanged. Following the work of Murray and Buxton [43], Odobez and Bouthemy [86] and Stiller [31], Tekalp [5,110] summarizes these two steps with a *Maximum Likelihood* (ML) and *Maximum a Posteriori* (MAP) procedure. The difference between the former and the latter is the use of an *a priori* energy function that helps smooth the resulting motion label field.

2.3.2.2 Motion Estimation and Our Framework

As usual, the goal is to estimate a label field $x^{[0]}$ whose pixels are associated with labels ranging between ζ_0 and ζ_{d-1}. Since the optical flow field is assumed to be piecewise parametric, each class ζ_i is assigned a parameter vector $\vec{A}_{\zeta_l}$. A commonly used parametric model is the six-parameter *affine* model $\vec{A}_{\zeta_l} = (a_{l1}, a_{l2}, a_{l3},$

a_{l4}, a_{l5}, a_{l6}) defined as [5] :

$$\vec{v}_s = \begin{pmatrix} v_x \\ v_y \end{pmatrix} \tag{2.8}$$

$$= \begin{pmatrix} 1 & i & j & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & i & j \end{pmatrix} \begin{pmatrix} a_{l1} \\ a_{l2} \\ a_{l3} \\ a_{l4} \\ a_{l5} \\ a_{l6} \end{pmatrix}$$

$$= \mathcal{X}_s \vec{A}_{\zeta_l} \tag{2.9}$$

where (i, j) are the Euclidean coordinates of site s and (v_x, v_y) are the horizontal and vertical components of motion vector $\vec{v}_s$.

In this paper, the motion label field $x^{[0]}$ (as well as the parameters $\vec{A}_{\zeta_l}$, $l \in [0, d[$) are estimated with a MAP procedure [5, 43] whose objective is to maximize the *a posteriori* PDF $P(x^{[0]}|I)$ where I is the input image sequence (note that for the rest of this subsection, $x^{[0]}$ will be replaced by x to simplify the notation). By the well known Bayes theorem, $P(x|I)$ may be rewritten as

$$P(x|I) = \frac{P(I|x)P(x)}{P(I)} \tag{2.10}$$

where $P(I|x)$ is the likelihood PDF that measures how well the motion label field x (together with its d motion models) fits the input observation I and $P(x)$ is the prior PDF. Observe that since $P(I)$ is constant with respect to x, it will be ignored during the optimization process.

If we assume that the *true* vector field is perturbed with a zero mean white noise with standard deviation σ and that the mismatch between I and x is modeled with

the well-known motion constraint equation

$$E(x_s) = (\partial_x I v_x + \partial_y I v_y + \partial_t I)^2 \tag{2.11}$$
$$= (\nabla I \vec{v} + \partial_t I)^2 \tag{2.12}$$

where $\partial_x I, \partial_y I, \partial_t I$ denote the spatial and temporal partial derivatives at site s and time t, the likelihood PDF $P(I, x)$can be expressed at each pixel as

$$P(I_s|x_s) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp\{-E(x_s)/2\sigma^2\}. \tag{2.13}$$

As for the prior PDF $P(x_s)$, the Potts model may be used to incorporate local constraints on the segmentation. This is given by

$$P(x_s) = \frac{1}{Z} \exp\{-\beta \underbrace{\sum_{<t,s>} (1 - \delta(x_s, x_t))}_{U(x_s)}\} \tag{2.14}$$

where Z is a normalizing constant, $\delta(x_s, x_t)$ is the Kronecker delta, and β is a constant. If we assume that each random variable I_s is independent, the *optimal* solution can be formulated as

$$\hat{x} = \arg\max_x \prod_{s \in S} P(I_s|x_s) P(x_s) \tag{2.15}$$

or, if we assume that the noise level σ is the same for each class :

$$\hat{x} = \arg\min_x \sum_{s \in S} (\nabla I \vec{v}_s + \partial_t I)^2 + \beta U(x_s). \tag{2.16}$$

Combining Eq.(2.16) and Eq.(2.9) leads to

$$\hat{x} = \arg\min_x \sum_{s \in S} \left(\nabla I \left[\mathcal{X}_s \vec{A}_{x_s} \right] + \partial_t I \right)^2 + \beta U(x_s)$$

which emphasizes the need for a joint estimation of $\vec{A}_i$ and x. As proposed by

Murray and Buxton [43] and Tekalp [5], x and $\vec{A}_i$ may be estimated with stochastic optimizers such as simulated annealing or the Metropolis algorithm. In this paper, we use the simulated annealing approach presented in Algorithm 2.

Once the motion label field x has been estimated, it can be fused with r using our fusion procedure (Algorithm 1).

2.3.3 Occlusion Detection

2.3.3.1 Introduction

The goal of most optical flow and stereovision algorithms is to estimate a matching function (be it a disparity map [46] or an optical flow field [70]) between the pixels of two (or more) input images. Due to motion or to a parallax effect between a *left* and a *right* image, most scenes contain areas that are visible in only one frame. Generally speaking, these half-occluded areas are either *newly exposed* or *newly occluded* [128, 142]. Since these areas have no direct correspondence in the second image, they are a classical source of error for most motion or depth estimation algorithms.

While many authors have considered occlusion as a source of noise that is to be fought with spatial smoothing [46], others have explicitly included an occlusion criterion in their energy function [11, 32, 40, 77, 90, 118]. During the past few years, a variety of occlusion metrics have been proposed among which the one Egnal and Wildes [57] call the *left-right-check* (LRC) has drawn a lot of attention. This approach stipulates that the matching function between the left and the right image should differ only by sign with the right-left matching function. In this context, every pixel for which the difference between the left-right match and the right-left match is above a given threshold is considered as being occluded. Although the LRC can be useful within a global energy function [24, 90, 94], many researchers have noted that the LRC is error-prone in noisy areas [142] and in areas having little or no texture [57, 118]. Others have also argued that estimating the forward *and* the backward matching functions can be prohibitive time wise.

Simulated Annealing

1. Initialize x and $T = T_0$ the initial temperature
2. Update $\vec{A}_0, \vec{A}_1, ..., \vec{A}_{d-1}$ as

$$\vec{A}_l = \arg\min_{\vec{A}} \sum_{\substack{s \in S \\ x_s = \zeta_l}} (\nabla I[\mathcal{X}_s \vec{A}] + \partial_t I)^2 \quad (2.17)$$

This minimization can be done with a least-square estimation of $\vec{A}_l$ on every pixel whose motion label is $x_s = \zeta_l$. This is expressed as

$$\vec{A}_l = \left[\left(\sum_{\substack{s \in S \\ x_s = \zeta_l}} (\nabla I \mathcal{X}_s)^{\mathrm{T}} (\nabla I \mathcal{X}_s) \right)^{-1} \sum_{\substack{s \in S \\ x_s = \zeta_l}} (\nabla I \mathcal{X}_s)^{\mathrm{T}} \partial_t I \right]$$

3. For each site, compute the probability related to each label $\zeta_i \in \Gamma$ as

$$E_s(\zeta_l | I) = \sum_{s \in S} \left(\nabla I \left[\mathcal{X}_s \vec{A}_{\zeta_l} \right] + \partial_t I \right)^2 + \beta U(x_s)$$
$$P(x_s = \zeta_l | I) = \frac{1}{Z} \exp -E_s(\zeta_l | I)/T$$

and randomly assign a label $\zeta_j \in \Gamma$ to x_s according to its probability $P(\zeta_j | I)$.

4. $T = T \times \text{coolingRate}$
5. Repeat Step 2, 3 and 4 until T reaches a minimum temperature.

Algorithm 2: Algorithm used for the motion estimation/segmentation procedure. This algorithm returns a motion label field x as well as the affine motion model $\vec{A}_l$ for each motion class.

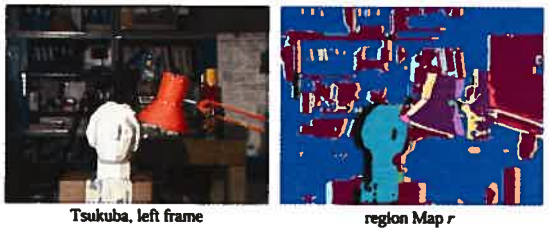

FIG. 2.4 – TSUKUBA left image with the region map r obtained after segmenting the two input images $I^{[ref]}$ and $I^{[mat]}$

Another idea that enjoys a great deal of popularity is Marr-Poggio's [41] uniqueness assumption. This assumption stipulates that there is always a one-to-one correspondence between the pixels of the two frames. Kolmogorov and Zabih [154] incorporated that assumption into their graph-cut algorithm and stipulated that each pixel in one image should correspond to *at most* one pixel in the other image. A pixel with no match would then be considered as being occluded. A variation of this approach has been proposed by Sun *et al.* [77] for which a non-occluded pixel must have *at least* one match. Although the difference between the two approaches is conceptually slight, Sun *et al.* [77] demonstrate that their method is superior in scenes containing slanted surfaces.

Let us also mention that some authors use the so-called *Ordering constraint* [40, 57, 77] which stipulates that a point P laying to the right of a point Q in one image should also lie to the right of Q in the other image. Although this assumption is often true, it can easily be violated by narrow front-ground objects (what Sun *et al.* [77] call the "double nail illusion").

2.3.3.2 Occlusion Detection and Our Framework

After thorough evaluations of many occlusion detection criteria, we came to realize that the ones based on Marr-Poggio's uniqueness assumption are the most accurate, at least in the context of our framework (in their review paper, Egnal and Wildes [57] came to a similar conclusion). More specifically, the Ince-Konrad [142]

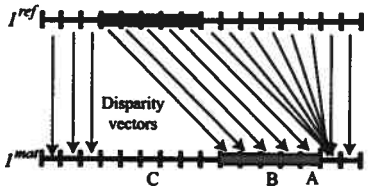

FIG. 2.5 – Synthetic example of a disparity map between two images $I^{[ref]}$ and $I^{[mat]}$. In this example, "C" is an occluded pixel since it is visible in $I^{[mat]}$ but occluded by the blue object in $I^{[ref]}$.

metric was retained to compute $x^{[0]}$, a "rough" occlusion map estimate. The Ince-Konrad [142] metric can be seen as a generalization of the uniqueness constraint : instead of counting the number of matches for each pixel independently, they count the number of matches within a given local neighborhood. Consider $\Lambda = \{s | s \in S\}$ the set of pixels in the reference image I^{ref} and $\Delta = \{u | u = s + d_s, s \in S\}$, the set of *matching* pixels in I^{mat}[2]. Based on Δ, an accumulation function M is computed

$$M_s = \sum_{i \in \Delta} \zeta_{i,s} \tag{2.18}$$

where $\zeta_{i,s} = 1$ if the Euclidean distance between pixel $s \in S$ and $i \in \Delta$ is lower than or equal to D, and zero otherwise. The occlusion map $x^{[0]}$ is obtained by thresholding M_s :

$$x_s^{[0]} = \begin{cases} 1 & \text{if } M_s < \tau \\ 0 & \text{otherwise.} \end{cases} \tag{2.19}$$

As suggested by the authors [142], we set D to 2. The Ince-Konrad method can be intuitively understood following the synthetic example of Figure 2.5. In this example, with $D = 2$, the accumulation function M_s equals 11 for pixel A, 5 for

[2] I^{ref} and I^{mat} stands for the "left" and "right" image in stereovision and for the images at time t and $t+1$ in optical flow. As for d_s, it represents the disparity value linking pixel I_s^{ref} to its projection in I_s^{mat} as shown in Figure 2.5.

pixel B and zero for pixel C. With a threshold τ of 3 for example, pixel C would be considered as being occluded.

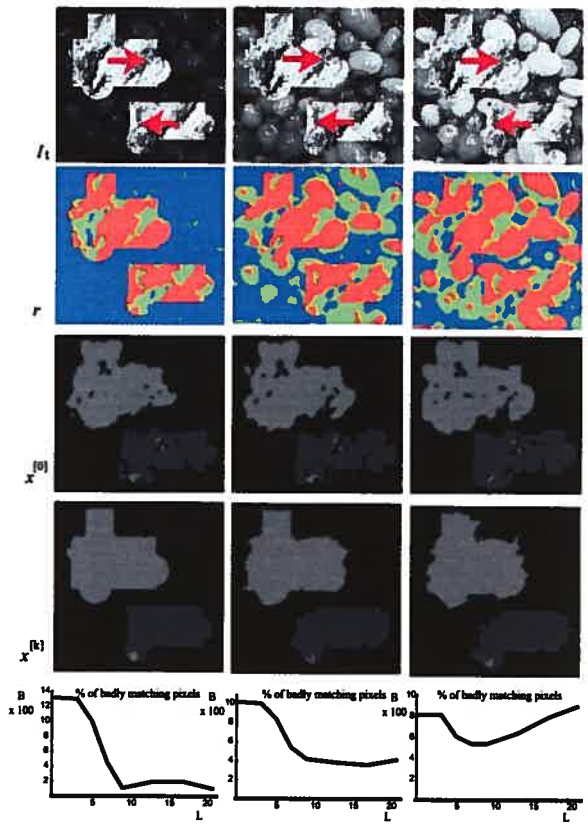

FIG. 2.6 – Three different versions of SEQUENCE A. From left to right, the sequence exhibits a precise, a medium and an imprecise region map r. In every case, the resulting label field $x^{[k]}$ is more precise than the initial one $x^{[0]}$. The last row contains graphics of the percentage of badly matching pixels versus the window size $\Psi = L \times L$. These curves are discussed in Section 2.4.4.

Because occlusion is a mismatch between *two* images, the way the region map r is computed is slightly different than for the other applications. In fact, the input frames I^{ref} and I^{mat} are respectively segmented into two label fields, namely r^{ref} and r^{mat}. These two region maps are then linearly combined together : $r = r^{\text{ref}} + c \times r^{\text{mat}}$ where c is the number of classes in which I^{ref} and I^{mat} have been segmented. This

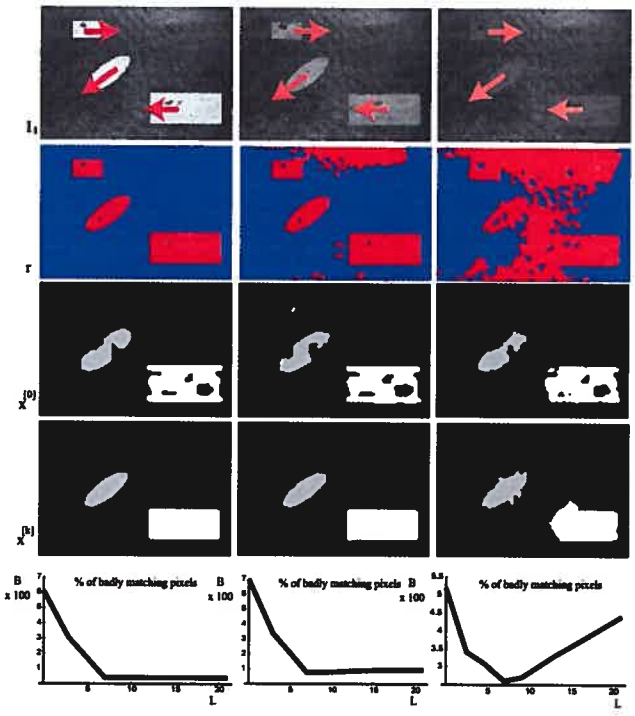

FIG. 2.7 – Three different versions of SEQUENCE B. From left to right, the sequence exhibits a precise, a medium and an imprecise region map r. In every case, the resulting label field $x^{[k]}$ is more precise than the initial one $x^{[0]}$. The last row contains graphics of the percentage of badly matching pixels versus the window size $\Psi = L \times L$. These curves are discussed in Section 2.4.4.

last operation results in a label field r whose regions are uniform in the sense of both input images. An example of such a region map is presented in Fig 2.4 (b). Once r and $x^{[0]}$ have been computed, they can be fused with Algorithm 1.

2.4 Experimental Results

In order to gauge performance of our algorithm, we segmented sequences representing different challenges. Some sequences are real while others have been computer generated and come with a perfect ground-truth label field g. The results presented in this Section illustrate how stable and robust our algorithm is

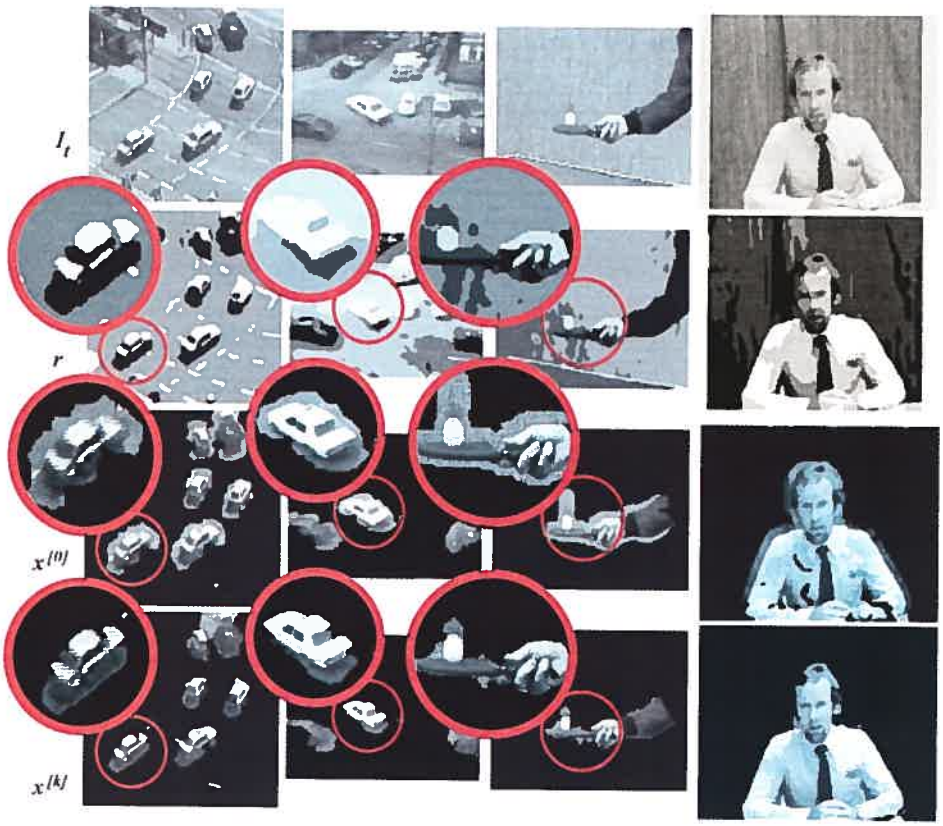

FIG. 2.8 – Sequences KARLSRUHE, TAXI, TENNIS, TREVOR WHITE, SEQUENCE A, and SEQUENCE B. The first row presents frames at time t, the second row spatial partitions r and the last two rows the motion label fields $x^{[0]}$ and $x^{[k]}$ superposed to I^t. As can be seen, the moving objects are more precisely located after the fusion process ($x^{[k]}$) than before ($x^{[0]}$).

Sequence A

Partition r	Alt.	$x^{[0]}$	3×3	7×7	11×11	21×21	31×31
precise	0.8	13.2	13.1	5.0	1.9	1.0	0.9
medium	12.5	10.8	10.7	5.4	4.0	4.2	5.3
imprecise	25.5	8.1	8.1	5.4	5.3	8.3	9.3

Sequence B

Partition r	Alt.	$x^{[0]}$	3×3	7×7	11×11	21×21	31×31
precise	0.4	6.2	2.9	0.4	0.4	0.4	0.5
medium	8.9	6.7	3.3	0.7	0.8	0.9	1.3
imprecise	42.6	5.2	3.3	2.0	2.6	2.7	5.4

TAB. 2.1 – Percentage of badly matching pixels computed with three different versions of two synthetic image sequences. From left to right : results obtained with Altunbasak *et al.* [164], our unsupervised statistical Markovian segmentation algorithm and results obtained with our fusion algorithm. The five rightmost columns measure the effect of the window size ($L \times L$). The quality of the spatial partition r is ranked from *precise* to *imprecise* depending on how well objects have been segmented (see Fig. 2.6 and 2.7). Note that our fusion method can reduce the percentage of badly matching pixels by a factor between 35% to 94% depending on the precision of r.

with respect to the window size $\Psi = L \times L$, the α coefficient (Eq. 2.4) and the precision of the region map r. Note that for each sequence presented here, the region map r has been computed with a number of classes ranging between 4 and 7 and that the window size Ψ ranges between 5×5 and 11×11. Also, the number of iterations $[k]$ needed by the fusion procedure to converge (algorithm 1) depends on the nature of the scene and the application. For the motion segmentation and the motion estimation applications, an average of 30 iterations is needed whereas an average of 5 iterations is needed for the occlusion detection.

2.4.1 Motion Segmentation

To test the robustness of our motion segmentation framework, different real and synthetic sequences have been segmented. At first, we segmented two synthetic sequences (called sequence "A" and sequence "B") both having a ground truth label field g (see Fig 2.6 and 2.7). Note that both synthetic sequences are made of real images pasted on computer generated shapes. To measure how precise the label fields $x^{[k]}$ returned by our algorithm are, we use the percentage of badly matching

pixels between $x^{[k]}$ and the ground-truth label field g, *i.e.*,

$$B(g, x^{[k]}) = \frac{100}{N_S} \sum_{s \in S} (1 - \delta(x_s^{[k]}, g_s)) \tag{2.20}$$

where N_S is the number of sites in S and $\delta(x_s^{[k]}, g_s)$ is the Kronecker delta function.

We computed the label field $x^{[k]}$ for both sequences with a different region map r exhibiting precise, medium and imprecise regions. These region maps are used to illustrate how robust our method is with respect to r. Here, the percentage of badly matching pixels is presented in Table 2.1 (and on the last row of Fig. 2.6 and 2.7). In Table 2.1, our fusion procedure is compared with Altunbasak *et al.*'s method [164] which also relies on a pre-estimated region map r. Since our fusion method does not assign one motion class to every pixel of a region (as is the case for Altunbasak *et al.*'s), our approach is less sensitive to imprecisions in r. This observation illustrates the fact that our algorithm reacts smoothly to a change in its parameters L and r.

Also, as shown in Fig. 4.3.3, four real video sequences have been segmented. To illustrates the precision of the resulting label fields, $x^{[0]}$ and $x^{[k]}$ have been superimposed to the image I^t. From left to right, the sequences were segmented with respectively three, three, four, and six motion classes. As can be seen in most cases, the label field $x^{[k]}$ returned by our fusion framework is more accurate than the ones with no fusion procedure ($x^{[0]}$).

2.4.2 Motion Estimation

For this application, we segmented four synthetic sequences having a ground-truth label field g and ground-truth vector field V_g. These sequences are presented in Fig.2.9, 2.10, and 2.11. In Fig. 2.9 (a), the sequence (which we call sequence "C") exhibit two shapes moving in different directions on a flat grayscale background. In this case, since the region map r contains highly precise edges, the results $x^{[k]}$ also exhibits precise regions. This is shown by the percentage of badly matching pixels (in Table 2.2) which is much larger for $x^{[0]}$ than for $x^{[k]}$. As for the sequence

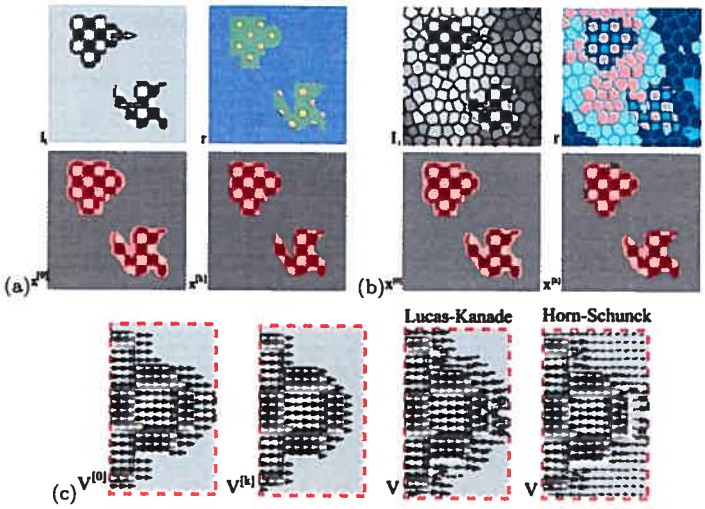

FIG. 2.9 – Sequence "C" (a) with flat background and (b) with textured background. Note that the textured background has been removed from $x^{[0]}$ and $x^{[k]}$ to help visualize the results. In (c), vector fields obtained with Algorithm 2 ($V^{[0]}$), our fusion method ($V^{[k]}$), Lucas-Kanade [22] and Horn-Schunck [21] optical flow methods.

of Fig.2.9 (b), it exhibits the same moving shapes as in Fig.2.9 but in front of a textured background. With this textured background, the region map r contains regions less precise than those of Fig.2.9. This is because the background is composed of shades similar to the ones on the moving shapes. But nevertheless, as shown in Table 2.2, even with an imprecise region map r, $x^{[k]}$ is still significantly more precise than $x^{[0]}$.

The third synthetic sequence we segmented is the one shown in Fig. 2.10. It contains an image of the Parthenon moving to the right, on top of an image performing a counterclockwise rotation. From left to right are the ground-truth, the scene estimated with the MAP procedure presented in Section 2.3.2, and the result of our fusion procedure. Because of the highly textured background, the region map r contains local imprecisions, especially around the first, the second, the third and the fifth column. As can be seen in image $x^{[k]}$ and $||V^{[k]}||$, the local imprecisions in r have induced local errors in the resulting fields, especially around the first and

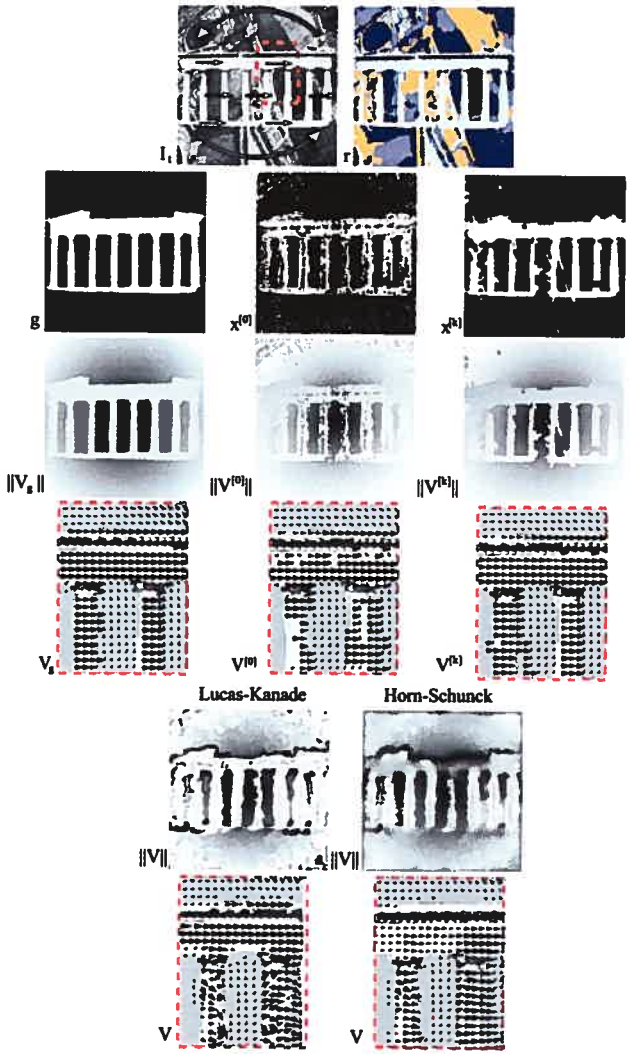

FIG. 2.10 – The PARTHENON sequence with the region map r, ground-truth images, initial estimates and the result of our fusion procedure. Note that the third row exhibits the vector fields' magnitude.

Sequence	$x^{[0]}$	$x^{[k]}$
Seq.C, flat back.	4.1%	0.006%
Seq.C, textured back.	4.4%	2.4%
Parthenon	16.4%	5.5%

TAB. 2.2 – Percentage of badly matching pixels computed for three synthetic sequences.

the fifth column. But nevertheless, the results returned by our fusion procedure are less noisy and globally exhibit more precise edges than the ones returned by the MAP procedure. This observation is underlined by the percentage of badly matching pixels presented in Table 2.2 which is almost three times smaller for $x^{[k]}$ than for $x^{[0]}$.

The fourth synthetic sequence is the famous YOSEMITE [70] sequence presented in Fig. 2.11. As can be seen, even if r has local imprecisions (especially in the top left section of the scene) the resulting label field $x^{[k]}$ (as well as $V^{[k]}$) is much smoother than $x^{[0]}$.

We also implemented a metric to measure how good the vector fields $V^{[k]}$ returned by our framework are. Following Barron *et al.* [70], we used to the average angular error metric to evaluate the distance between the ground truth vector field V_g and the estimated vector field (be it $V^{[0]}$ or $V^{[k]}$), namely

$$\bar{\Psi}_E(V, V_g) = \frac{1}{\mathcal{N} \times \mathcal{M}} \sum_{s \in S} \arccos(\vec{v}_s . \vec{v}_{g_s}) \tag{2.21}$$

where $\vec{v}_s$ and $\vec{v}_{g_s}$ are normalized 3D vectors : $\vec{v}_s = \frac{(u,v,1)}{\sqrt{u^2+v^2+1}}$. Using this metric, the vector field $V^{[k]}$ of the four synthetic sequences are compared to the ones returned by the MAP procedure. The angular errors are presented in Table 2.3 and again, the results clearly favors our method.

We also segmented a real image sequence, namely the RHEINHAFEN sequence presented in Fig. 2.12. Again, our results shows sharper edges and less isolated false positives and false negatives.

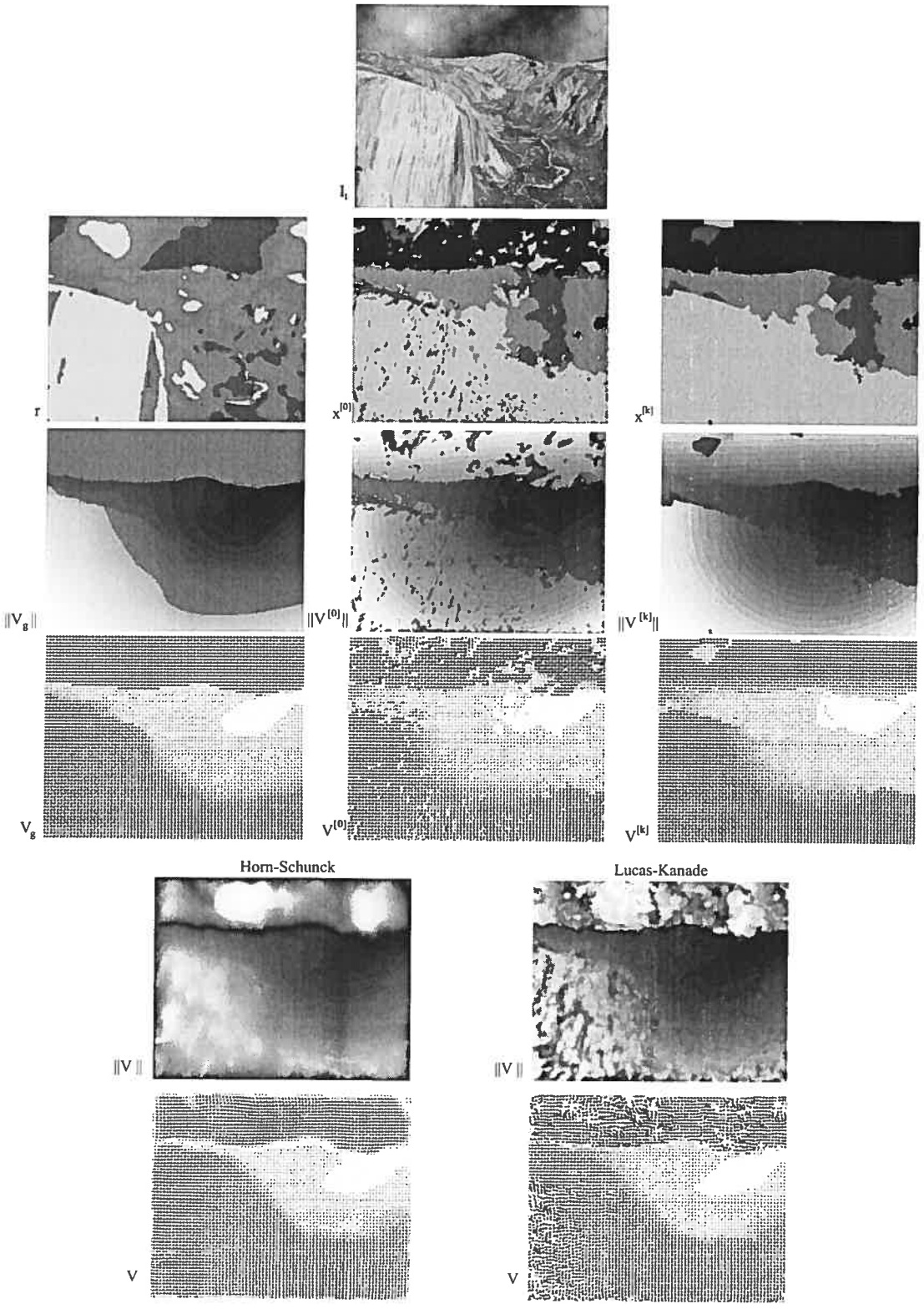

FIG. 2.11 – Yosemite Sequence with ground-truth, results from the MAP estimation / segmentation procedure and results from our fusion framework. Although r contains local imprecisions, the fusion procedure significantly reduces the number of false positives and false negatives in x.

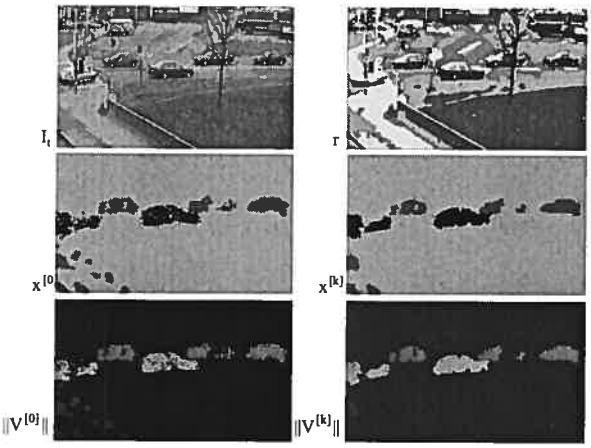

FIG. 2.12 – Zoom on the RHEINHAFEN sequence. In this case, the fusion process has significantly reduced the number of false positives and false negatives. As a result, the moving areas are more homogeneous.

2.4.3 Occlusion Detection

To validate our method, we detected occlusion on various data sets commonly used for such an application. Here, our framework is compared with other frequently-used approaches, namely the *left-right check* (LRC) [57], the *ordering constraint* (ORD) [57] and Ince-Konrad's [142] uniqueness-based approach.

Four sequences with ground truth taken from Middlebury web page [1] have been used to test the methods. As mentioned in Section 2.3.3, occlusion detection is closely related to applications involving a disparity map (such as optical flow and stereovision). A disparity map is a function that links the pixels of one image (I^{ref}) to their projection in a second image (I^{mat}). In this context, an occluded area is a section of the scene that is visible in one image but hidden in the second image. Although a disparity map can be estimated in a variety of ways as mentioned in most optical flow and stereovision review papers [46, 70], we estimate it with a simple pixel-based window matching strategy taken from the work of Scharstein and Szeliski [46]. This method, called *winner-take-all*, is a greedy algorithm looking at each pixel for the disparity value that best minimizes a matching cost. Here, the matching cost is a simple squared difference of intensity values. The resulting

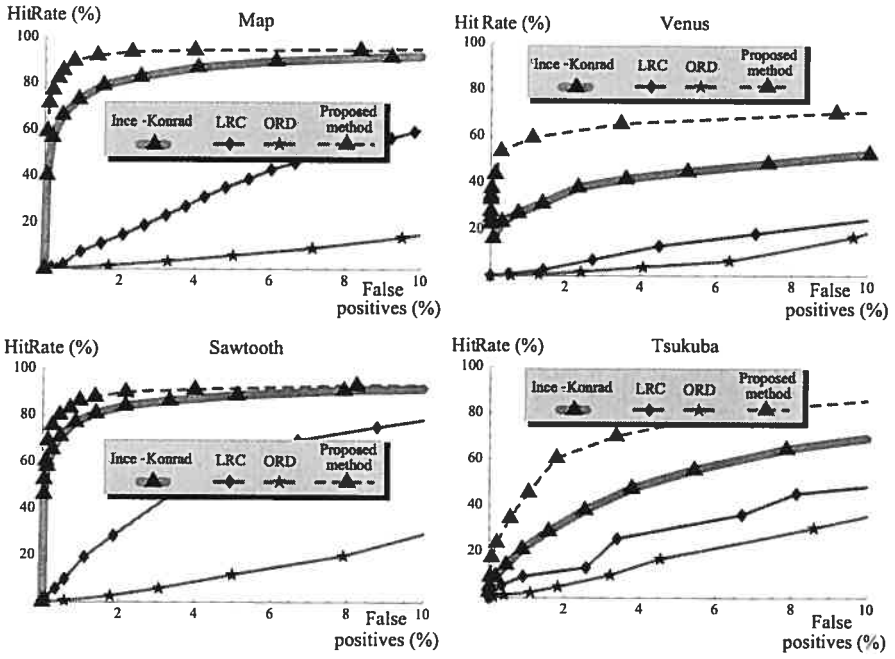

FIG. 2.13 – Hit rate versus false positive rates obtained with four different data sets. The proposed method significantly reduces the number of false positives and false negatives.

disparity map is filtered out with a 3×3 *shiftable* aggregation filter, *i.e.* a box filter that locally adapts to the scene. For more details on this algorithm, please refer to the work of Scharstein and Szeliski [46]

Following Egnal and Wildes' methodology [57], we have plotted the hit rate / false positive rate curve of every method by varying their threshold (see Fig. 2.13). According to these graphs, an optimal method is one that maximizes the surface below its curve. Consequently, as can be seen on every graphic of Fig. 2.13, our method appears to be more precise than the others we have implemented. This is especially true for those sequences containing large textureless areas such as VENUS and TSUKUBA. This can be explained by the fact that, as mentioned by Egnal and Wildes [57], most common occlusion detection methods are error-prone in textureless areas. In this context, using a region-based approach to eliminate

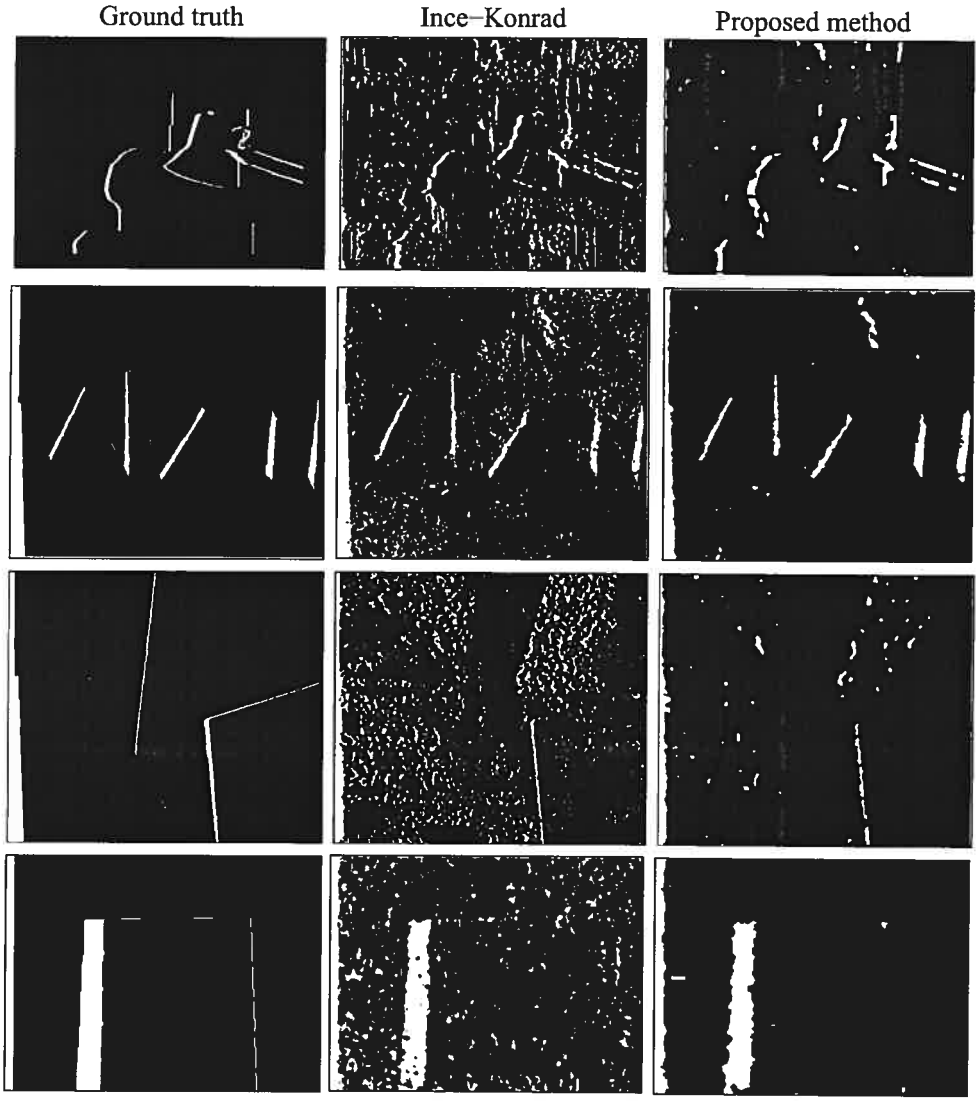

FIG. 2.14 – From top to bottom, ground truth and results obtained for TSUKUBA, SAWTOOTH, VENUS, and MAP data set. Hit rate for every result is respectively 60%, 90%, 45%, and 90%.

Sequence	$V^{[0]}$	$V^{[k]}$
Seq.C, flat back.	(4.1, 15.4)	(0.3, 0.5)
Seq.C, textured back.	(4.0, 15.3)	(3.2, 13.8)
Parthenon	(13.7, 24.0)	(8.1, 15.2)
Yosemite	(16.3, 18.5)	(9.5, 10.9)
	HS	LK
Seq.C, flat back.	(8.0,18.1)	(16.5,20.1)
Seq.C, textured back.	(8.0,18.1)	(16.5,20.1)
Parthenon	(13.3, 21.6)	(13.6, 17.8)
Yosemite	(11.0, 16.6)	(10.6, 11.3)

TAB. 2.3 – Average angular error with standard deviation computed for four different sequences. These results show the angular error with ($V^{[k]}$) and without ($V^{[0]}$) fusion as well as for the Lucas-Kanade (LK) and the Horn-Schunck (HS) methods.

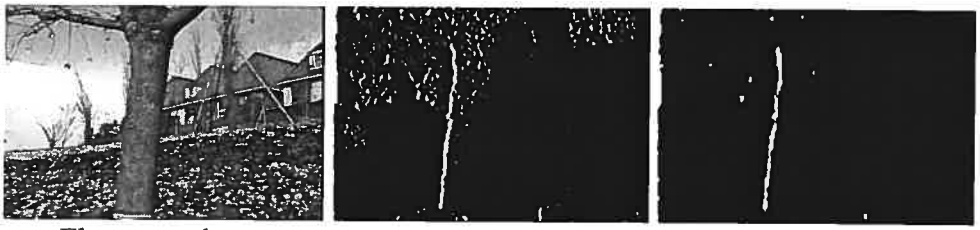

FIG. 2.15 – Comparison of the proposed method with the Ince Konrad's method on the FLOWERGARDEN sequence.

isolated false positives provides a clear advantage.

A qualitative comparison has also been made in Fig. 2.14. To make the results objectively comparable, each method has been tuned to return an occlusion map with a specific hit rate. In this way, the results in the second and third column of Fig. 2.14 have respectively hit rates of 60%, 90%, 45%, and 90%. Although the hit rate is the same for both approaches, the false positive rate is clearly better for our method.

As for the FLOWERGARDEN sequence of Fig. 2.15, our method produce again a significantly lower number of false positives. Note that for this sequence, the matching function was computed with a pixel-based window-matching strategy [70].

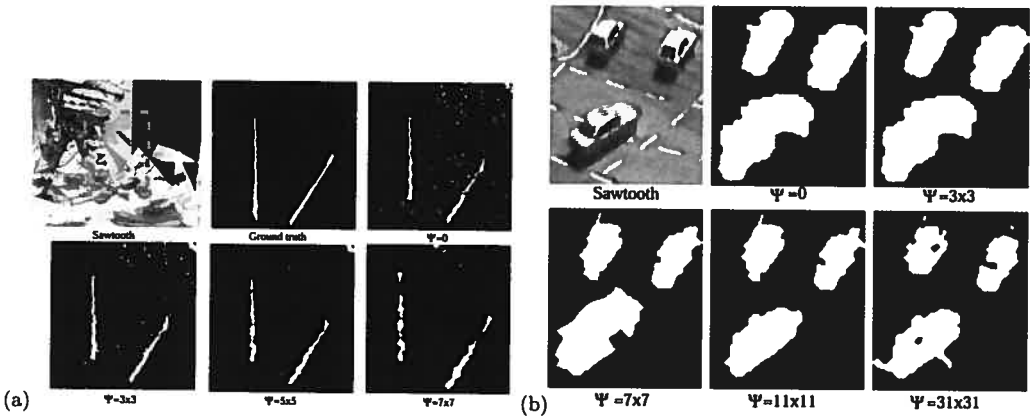

FIG. 2.16 – This figure illustrates the influence of the neighborhood size $\Psi = L \times L$. In (a), occlusion maps of the SAWTOOTH sequence and in (b), motion maps of the KARLSRUHE sequence.

2.4.4 Testing the influence of Ψ and α

Since our fusion framework depends very much on the size of the 2D neighborhood Ψ, we illustrate its influence with various qualitative and quantitative results. In Fig. 2.16, the effect of varying the window size is shown on a synthetic and a real sequence. As can be seen, a modification of this variable brings a smooth and predictive variation in the resulting image. However, it should be noted that the use of a too large window size can cause unexpected local errors as shown in the lower right image of both thumbnails of Fig. 2.16.

We also evaluated quantitatively the influence of Ψ on the percentage of badly matching pixels for two synthetic sequences. This is presented in the last row of Fig. 2.6 and 2.7 (and in Table 2.1). These examples show that the use of our fusion procedure does indeed help enhance the quality of the results even for sequences having an imprecise region map r. However, using a too large window Ψ for sequences having an imprecise region map can induce local errors and thus raise the percentage of badly matching pixels. That being said, we observed that for every sequence segmented in this paper, a window size ranging between 5×5 and 11×11 produces the most satisfying results.

As for the coefficient α of Eq. (2.4), we tested it on three sequences shown in

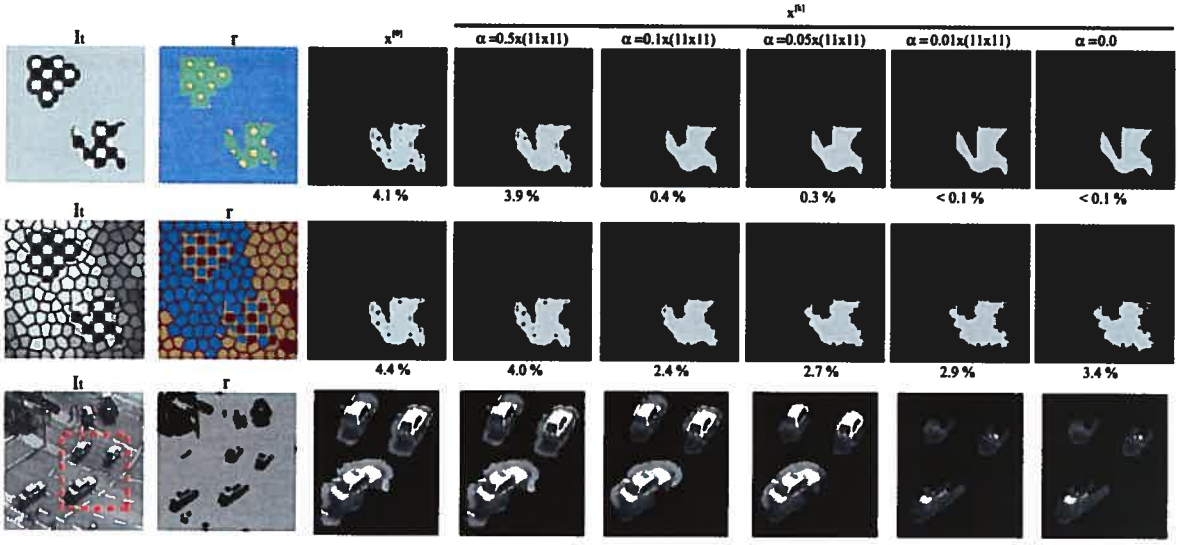

FIG. 2.17 – Three sequences segmented with different α values. The fusion procedure has a window size $\Psi = 11 \times 11$ and the percentage of badly matching pixels is shown below the synthetic label Fields. Among the three region maps, the first one exhibits precise regions, the second one less precise regions and the last one very imprecise two-class regions.

Fig. 2.17. As we mentioned previously, the reason for this coefficient is to give a relative influence to $x^{[0]}$ and to r during the fusion procedure (and thus make the optimization process well-posed). In fact, when r exhibits sharp regions (as is the case for the first sequence on top to Fig. 2.17) a small value might be assigned to α (*i.e.* a value that gives more weight to r than to $x^{[0]}$). However, when the region map r contains imprecise regions as is the case for the two other examples in Fig. 2.17, a non-zero value for α (here between $0.1 * (11 \times 11)$ and $0.05 * (11 \times 11)$) is preferable. In this way, the to-be-minimized energy function has a non-zero reaction term that prevents the resulting vector field $x^{[k]}$ from being too different from the initial guess $x^{[0]}$. As mentioned previously, a non-zero α value allows stochastic optimizers to converge towards a meaningful solution $x^{[k]}$. To illustrate that assertion, we minimized Eq. (2.4) with the deterministic optimizer ICM and with the stochastic simulated annealing optimizer. The results are illustrated in Figure 2.18. As can be seen, although the global energy of the ICM label field is slightly higher, the results are very much similar. This illustrates the fact Eq. (2.4) is nearly convex *i.e.* not

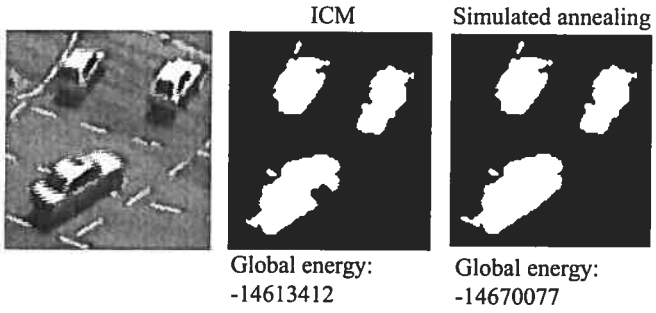

FIG. 2.18 – Results obtained after minimizing Eq. (2.4) with ICM and Simulated annealing. In this example, $\Psi = 9 \times 9$ and $\alpha = 0.01 \times (9 \times 9)$. The ICM label field has been obtained after 24 iterations whereas 250 iterations have been needed for Simulated annealing.

too erratic and can be efficiently minimized with a simple (and sub-optimal) ICM minimization procedure.

2.4.5 Real-time processing

Since our fusion framework process every pixel independently, it can be implemented on a parallel architecture. To this end, we implemented our method on a Graphics Processor Unit (GPU) [2]. A GPU is a processor embedded on most graphics card nowadays available on the market which, among other things, can load, compile and execute programs implemented with a C-like language. The key feature of the GPUs is their fundamental ability to process *in parallel* each pixel of the scene, making all kinds of applications much more efficient than when implemented on traditional sequential CPUs. For example, the fusion procedure (with $\Psi_s = 5 \times 5$) can process at a rate of 25 fps a scene of size 384×288 such as TSUKUBA[3]. Also, the region map r of this scene can be computed in 1 second or, if the Gaussian parameters are re-used from a previous calculation, in 0.05 second. These processing rates outperformed by a factor of almost 100 what we obtained with a traditional CPU implementation. For an application such as motion estima-

[3]Since there are no efficient ways to access the framebuffer content to verify if the ICM algorithm has converged (see part 2 of Algorithm 1), a predefined number of 10 ICM iterations has been used.

tion/segmentation (Algorithm 2), computing $x^{[0]}$ requires approximately 50 seconds whereas estimating $x^{[n]}$ ($x^{[0]} + r +$ fusion) requires no more than 52 seconds for a 256×256 scene. For more details on how r can be computed with a GPU, please refer to [123].

2.5 Conclusion

In this paper, we considered the issue of fusing label fields instead of features as is usually the case. The core of our method is a fusion framework that blends together a region map r and an application label field $x^{[0]}$. With the assumption that the color regions in r are more detailed than the regions in $x^{[0]}$, the goal of the fusion procedure is to iteratively modify the application field x to make its regions fit the color regions. In this way, false positives and false negatives are filtered out and blobby shapes are sharpened resulting in a more precise label field $\hat{x}$. The fusion is performed by an ICM optimization procedure that minimizes an energy function that may be a pure fusion function (and thus works at each ICM iteration in a similar way to the well-known K-nearest neighbor algorithm) or a fusion-reaction energy function involving a data-related term.

Although such a segmentation framework based on pre-estimated label fields might appear as a step backward when compared to methods that minimize one large multidimensional energy function [99,144] or to methods using dimensionality reduction algorithms [23, 131], our method has decided advantages. To start off with, traditional multidimensional methods often rely on weighting factors that give more or less influence to some features. Since a bad choice of these weighting parameters can lead to a bad segmentation, they must be carefully adjusted at runtime which, of course, can be cumbersome for most unsupervised applications. Also, because these parameters generally depend on the sequence content, they need to be re-estimated when a new sequence is processed. Furthermore, tweaking these weighting factors is not a small task, especially when their number is large. For instance, the method proposed by Black requires that a number of 8 weighting factors need to be tweaked [99]. Moreover, large energy functions are generally less

stable than smaller ones and thus often need to be implemented with a stochastic (and slow) optimizer such as simulated annealing.

The point of our method is to alleviate these problems by individually minimizing two energy functions in order to blend their label fields. Thus, our method does not rely on weighting factors, scaling functions or projection functions often used when blending features. Our method uses simple energy functions that can be minimized with a deterministic optimizer (such as ICM) which is much faster than say, simulated annealing. Finally, we believe our fusion method is simple to implement and can be easily transposed to a parallel architecture such as a GPU.

Results obtained on real and synthetic image sequences show that our algorithm is stable and precise. It reacts well to changes of its parameters Ψ and α and performs well even when supplied with poorly estimated region maps r.

As mentioned previously, our method mostly depends on two parameters, namely the window size Ψ and the coefficient α. In our implementation, these two parameters are specified by the user in a supervised way. However, if an application had access to a corpus of images with ground-truth data, it would be possible to automatically estimate an "optimal" value for Ψ and α. In this context, the search for the "best" value for Ψ and α could be done by successively segmenting every image of the corpus and comparing the resulting segmentation maps with the ground-truth maps. This search could be done in a brute force way by testing a large number of values for Ψ and α and keeping the ones associated to the best results. On the other hand, a simplex optimizer could also be implemented to reduce the computational cost.

2.6 Acknowledgments

The authors would like to thank the Middlebury stereovision research team (http ://cat.middlebury.edu/stereo/) for providing the stereo-pair images, John Barron for his precious help with the Yosemite sequence, and the group of Prof. Dr. H.-H. Nagel for providing most video sequences (http ://i21www.ira.uka.de/image_sequences/).

CHAPITRE 3

DÉTECTION DE MOUVEMENT À L'AIDE D'INFORMATION SPATIALE

Article : Statistical Background Subtraction Using Spatial Cues.

Cet article a été accepté sans condition au journal *IEEE Transactions on Circuits and Systems for Video Technology* comme l'indique la référence bibliographique

P-M Jodoin M. Mignotte, J.Konrad[1]. Statistical Background Subtraction Using Spatial Cues . *IEEE Transactions on Circuits and Systems for Video Technology* , accepté 2007.

Cet article est présenté ici dans sa version originale.

Résumé

Comme nous l'avons souligné dans l'introduction, la plupart des techniques de soustraction de fond modélisent chaque pixel du fond à l'aide d'une densité de probabilité apprise sur une série d'images temporelles. Toutefois, comme le mentionnent plusieurs auteurs, apprendre des densités de probabilité sur plusieurs images peut être problématique, surtout lorsqu'aucune image absente de mouvement n'est disponible ou lorsque la mémoire disponible est insuffisante pour stocker les images nécessaires à l'apprentissage. En réponse à ces problèmes, nous proposons d'explorer deux alternatives *spatiales* aux traditionnels approches *temporelles*. En effet, notre proposition est d'étudier un nouveau cadre de travail permettant la soustraction de fond à l'aide de distributions statistiques apprises sur une seule image. Pour ce faire, nous proposons deux approches complémentaires. La première, que nous appelons «robuste», gère des scènes dont le fond est instable en raison d'une caméra qui vibre ou d'objets du fond en mouvement. Pour cette approche,

[1]Janusz Konrad est professeur à l'Université de Boston.

chaque pixel du fond est modélisé par deux distributions statistiques : une distribution unimodale et une autre multimodale, toutes deux entraînées sur une seule image du fond. La deuxième approche, que nous appelons «légère et rapide», a pour but d'utiliser le moins de puissance de calculs et d'espace mémoire possible. En se basant sur l'hypothèse que des pixels voisins partagent souvent une distribution temporelle similaire, cette deuxième approche modélise le fond à l'aide d'un mélange global de $\mathcal{M}$ gaussiennes.

Abstract Most statistical background subtraction techniques are based on the analysis of temporal color/intensity distributions. However, learning statistics on a series of time frames can be problematic, especially when no frame without moving objects is available or when the available memory isn't sufficient to store the series of frames needed for learning. In this paper, we propose a *spatial* variation to the traditional *temporal* framework. The proposed framework allows statistical motion detection with methods trained on one single frame instead of a series of frames as is usually the case. Our framework includes two *spatial* background subtraction approaches suitable for different applications. The first approach is meant for scenes having a non-static background due to noise, camera jitter or motion in the scene (e.g.,waving trees, fluttering leaves). This approach models each pixel with two PDFs : one unimodal PDF and one multimodal PDF, both trained on one background frame. In this way, the method can handle backgrounds with static and non-static areas. The second *spatial* approach is designed to use as little processing time and memory as possible. Based on the assumption that neighboring pixels often share similar temporal distributions, this second approach models the background with one global mixture of $\mathcal{M}$ Gaussians.

3.1 Introduction

Motion detection methods are used to locate the presence (or absence) of motion in a given animated scene. A specific class of motion detection methods is the one meant for video surveillance [7], traffic monitoring [113] and various commercial applications [163] using a static camera. Because the background is constant in time, a class of solutions that enjoys tremendous popularity is the family of background subtraction methods [12]. These methods are based on the assumption that the animated objects to be segmented have different visual characteristics than the static background. As the name suggests, the most intuitive background subtraction method involves one background image and an animated sequence containing moving objects [12]. In this case, motion is detected by simply thresholding the intensity/color difference between a frame at time t and the background image [25,108]. Although the threshold can be estimated on the fly [126, 127] and locally adapted [16], these methods are sensitive to phenomena that violate the basic assumptions of background subtraction. For instance, noise induced by a low-quality camera or jitter caused by an unstable camera are typical situations that can't be handled properly by simplistic background subtraction methods. There are also numerous situations in which some background objects aren't perfectly static and induce local false positives. This is the case, for instance, when a tree is shaken by the wind or when the background includes animated texture such as wavy water. Another common source of background variations is when the global illumination isn't constant in time and alters the appearance of the background. Such variation can be gradual, for example, when a cloud occludes the sun, or sudden, when a light is turned on or off.

For all these scenarios, a more elaborate background subtraction strategy is required. In this perspective, many authors propose modeling each background pixel with a probability density function (PDF) learned over a series of training frames. In this case, the motion detection problem often becomes a PDF-thresholding problem. For instance, to account for noise, some authors [33, 143] use a Gaussian

distribution for each pixel. Also, to account for unstable backgrounds, multimodal PDF models have been proposed such as a mixture of Gaussians (MoG) [14,29,113], kernel-based methods [7,15], and predictive methods [84,105]. Let us also mention that several block-based methods [44], codebook methods [89] and statistical Markovian methods [165], to name a few, have been proposed.

The main limitation of most traditional statistical solutions is their need for a series of training frames absent of moving objects. Without these training frames, a non-trivial outlier detection method needs to be implemented [29]. Another limitation of these methods is the amount of memory some require. For example, in [113], every training frame needs to be stored in memory to estimate the MoG parameters and, for some non-parametric methods [7,15], many frames need to be kept in memory during runtime which may be costly memory-wise.

In this paper, we investigate a framework that uses one background frame to train statistical models (be they parametric or not). Our aim for such a framework is fourfold :

1. reduce the number of frames (thus the amount of memory) needed during the training period;
2. reduce the amount of memory required during the detection phase;
3. better deal with sequences having no training frames absent of moving objects;
4. investigate how speed and memory can be improved in the context of statistical background subtraction.

3.2 Motivation

Two kinds of background variation may be identified. The first one is variations due to noise typically induced by a low-quality camera or by an environmental phenomenon, such as rain (see Fig.3.1 (c)). In this case, the background B is considered static and the intensity/color of a pixel at site s and discrete time t is defined as $\vec{B}_s^t = \vec{B}_s + n$ where n is an uncorrelated noise variable and $\vec{B}_s$ is the

ideal noise-free background color (or intensity) at site s. In this case, the temporal distribution $P(\vec{B}_s^t)$ is considered unimodal with expectation $\vec{B}_s$. The second kind of variations is due to background movements caused, for example, by an animated texture or by camera jitter. In this case, the temporal distribution $P(\vec{B}_s^t)$ is often multimodal. But whether the temporal distribution is unimodal or multimodal, the goal is to estimate at every time instant t a *label field* L^t (sometimes called a *motion mask*) containing the motion status of each site s (typically, $L_s^t = 0$ when s is motionless and $L_s^t = 1$ otherwise). Since each background pixel is modeled by PDF $P(\vec{B}_s^t)$, the detection criterion can be formulated as :

$$L_s^t = \begin{cases} 0 & \text{if } P(\vec{I}_s^t) > \tau \\ 1 & \text{otherwise.} \end{cases} \tag{3.1}$$

where I^t the intensity of examined frame at time t and τ is a predetermined threshold.

In this paper, the goal is to use the same thresholding procedure and the same distribution functions proposed in the literature, but with different training data gathered inside a single frame instead of a series of frames as is usually the case. The use of spatial data is motivated by the fact that the color distribution of a given pixel observed over a certain period of time is also frequently observed spatially around that same pixel. For example, in the case of background movements, considering that variations are due to local movements, it can be assumed that the distribution of $\vec{B}_s^t$ in time is similar to a spatial distribution around s, *i.e.*, $\{\vec{B}_r, \forall r \in \eta_s\}$ where η_s is a $M \times M$ neighborhood centered on s. Fig.3.1 (a)-(b) shows that when a site s is locally animated, the color/intensity distribution observed over a certain period of time often corresponds to the distribution observed locally around s. As shown in Fig.3.1 (c), the same observation may also hold for unimodal distribution. In the next sections, two *spatial* approaches will be presented. The first one is designed to be robust to background movements while the second one minimizes processing complexity and memory usage during runtime.

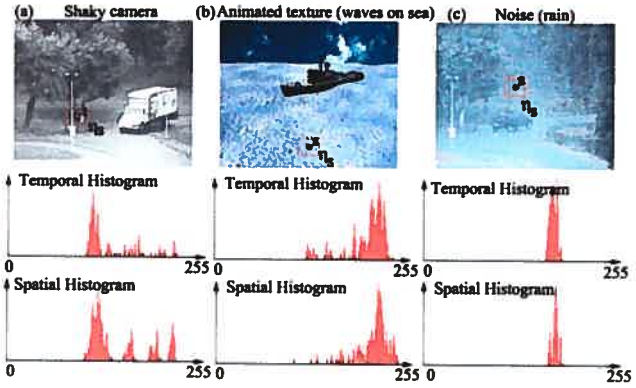

FIG. 3.1 – Sample frame (top row), temporal histogram (middle row) and spatial histogram (bottom row) for three test sequences : "Shaky camera" captured with unstable camera (left column), "Boat" that includes animated texture modeling water (middle column) and "Rain" that is an example of severe noise (right column). The spatial histograms were calculated from 11×11 neighborhoods while the temporal histograms were calculated from 90, 200, and 100 frames, respectively. Note a good correspondence between spatial and temporal histograms.

3.3 Robust Method

As we mentioned previously, background variation may be unimodal or multimodal. Unfortunately, with only one training frame, determining whether the distribution of a site s in time is unimodal or multimodal is an ill-posed problem. In order to overcome this, we use a *decision fusion* strategy. To this end, each pixel is modeled by two PDFs : one unimodal PDF (that we call P^u) and one multimodal PDF (called P^m), both being trained on one background frame B (see Section 3.3.1 for more details on training). Having each pixel modeled by two PDFs, the detection criterion may be formulated as follows :

$$L_s^t = \begin{cases} 0 & \text{if } P^u(\vec{I}_s^t) > \tau \text{ OR } P^m(\vec{I}_s^t) > \tau \\ 1 & \text{otherwise.} \end{cases} \tag{3.2}$$

Estimating L^t with this equation turns out to be the same as blending two label fields resulting from thresholding $P^u(\vec{I}_s^t)$ and $P^m(\vec{I}_s^t)$ separately. The reader

should be aware that this fusion strategy is meant for sequences whose content is unknown *a priori*. However, if the scene is known to have either a perfectly static background or camera jitter, only P^u or P^m may be thresholded.

3.3.1 Spatial Training

In this section, we present how P^u and P^m can be trained on data gathered inside a single background training frame B.

3.3.1.1 Single Gaussian

As mentioned in Section 3.1, $P^u(\vec{B}_s^t)$ can be modeled by a single Gaussian distribution,

$$\frac{1}{(2\pi)^{d/2}|\Sigma_s|^{1/2}} \exp\left(-\frac{1}{2}(\vec{B}_s^t - \vec{\mu}_s)^{\mathrm{T}}\Sigma_s^{-1}(\vec{B}_s^t - \vec{\mu}_s)\right) \tag{3.3}$$

where $d = 1$ for grayscale sequences and $d = 3$ for color sequences. Note that for color sequences, Σ_s is a 3×3 variance-covariance matrix that, as suggested by Stauffer and Grimson [29], is assumed to be diagonal for efficiency. Since only one training frame is available in this framework, $\vec{\mu}_s$ and Σ_s are estimated with data gathered inside a spatial neighborhood η_s centered on site s. Of course, the spatial data should be drawn from a unimodal distribution that resembles the one observed temporally. Although some spatial neighborhoods of a scene have that kind of unimodal distribution (neighborhood A in Fig. 3.2), others are obviously multimodal (neighborhood C in Fig. 3.2) and cannot be used for training as is. Thus, to prevent $\vec{\mu}_s$ and Σ_s from being corrupted by outliers (such as gray pixels of the pavement near C in Fig. 3.2), a robust function ρ is used to weigh the influence of each sample $\vec{B}_r$ [125]. More specifically, the parameter estimation can

be expressed as :

$$\vec{\mu}_s(j) = \frac{1}{\sum_{r\in\eta_s}\rho_{s,r}}\sum_{r\in\eta_s}\rho_{s,r}\vec{B}_r(j) \tag{3.4}$$

$$\Sigma_s(j) = \frac{1}{\sum_{r\in\eta_s}\rho_{s,r}}\sum_{r\in\eta_s}\rho_{s,r}(\vec{B}_r(j)-\vec{\mu}_s(j))^2 \tag{3.5}$$

where $j \in \{1,...,d\}$, $\Sigma_s(j)$ is the j^{th} variance. As suggested by Huber [125], we use,

$$\rho(s,r) = \begin{cases} 1 & \text{if } ||\vec{B}_s - \vec{B}_r||_2 \leq c \\ \frac{c}{||\vec{B}_s-\vec{B}_r||_2} & \text{otherwise} \end{cases} \tag{3.6}$$

where c is a constant and $||.||_2$ denotes the Euclidean norm. Clearly, with Eq.(3.6), as intensity/color of pixels in η_s deviate from those of the central pixel at s, their contribution to the mean and variance estimates diminishes. Note that global illumination variations can be compensated by updating $\vec{\mu}_s$ and Σ_s at every time t [33,143] as follows :

$$\vec{\mu}_s(j) \leftarrow (1-\alpha)\vec{\mu}_s(j) + \alpha\vec{I}_s^t(j) \tag{3.7}$$

$$\Sigma_s(j) \leftarrow (1-\alpha)\Sigma_s(j) + \alpha(\vec{I}_s^t(j)-\vec{\mu}_s(j))^2 \tag{3.8}$$

for all s such that $L_s^t = 0$, $j \in \{1,...,d\}$, and $\alpha \in [0,1[$ is the so-called *learning rate* [113].

3.3.1.2 Mixture of Gaussians

a mixture of K Gaussians is used to model multimodal histograms such as those in Fig. 3.1 (a)-(b) ,

$$P^m(\vec{I}_s^t) = \frac{1}{\sum_i^K w_{s,i}}\sum_{i=1}^{K} w_{s,i}\,\mathcal{N}(\vec{I}_s^t, \vec{\mu}_{s,i}, \Sigma_{s,i}) \tag{3.9}$$

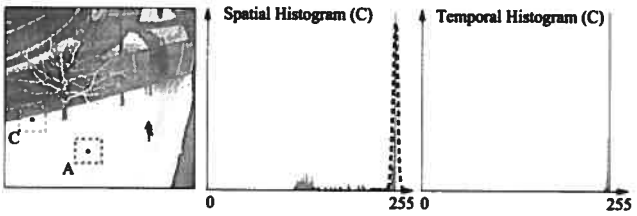

FIG. 3.2 – Sample frame from a sequence captured by a perfectly static camera. While the temporal intensity distribution of pixel C is unimodal and centered at intensity 254, the spatial intensity distribution around C is bimodal. Estimating the Gaussian parameters with Eq. (3.4) and (3.5) leads to a distribution (in black) centered on the main mode, uncorrupted by the gray levels of the street.

where $\mathcal{N}(.)$ is a Gaussian similar to the one of Eq.(3.3), $w_{s,i}$ is the weight of the i^{th} Gaussian and K is the total number of Gaussians (between 2 and 5 typically). Thus, $P^m(\vec{I}_s^t)$ has $3 \times K$ parameters to be estimated : $w_{s,i}, \vec{\mu}_{s,i}$, and $\Sigma_{s,i}$. To do so, we use the well-known K-means algorithm. In our framework, K-means takes as input for each pixel s, the $M \times M$ background pixels $\{\vec{B}_r | r \in \eta_s\}$ contained within the square neighborhood η_s. When the algorithm converges, the mean $\vec{\mu}_{s,i}$ of every cluster is known and every pixel $r \in \eta_s$ has been assigned a class label $i \in [0, K[$. The covariance matrix $\Sigma_{s,i}$ and the weight $w_{s,i}$ of each class i can then be estimated from the pixels $r \in \eta_s$ having label i. Note that the EM [130] algorithm could eventually be used to refine these estimates.

As we mentioned for P^u, the MoG parameters can be updated at every frame to account for illumination variations. As suggested by Stauffer and Grimson [29], at each time frame I^t, parameters of the Gaussian that matches the observation $\vec{I}_s^t$ can be updated as follows :

$$\vec{\mu}_{s,i}(j) \leftarrow (1-\alpha)\vec{\mu}_{s,i}(j) + \alpha \vec{I}_s^t(j) \tag{3.10}$$

$$\Sigma_{s,i}(j) \leftarrow (1-\alpha)\Sigma_{s,i}(j) + \alpha(\vec{I}_s^t(j) - \vec{\mu}_{s,i})^2 \tag{3.11}$$

$$w_{s,i} \leftarrow (1-\alpha)w_{s,i} + \alpha\xi_{s,i} \tag{3.12}$$

for all s such that $L_s^t = 0$, $j \in \{1, ..., d\}$, where $\xi_{s,i}$ is 1 for the Gaussian that matches and 0 for the other models.

3.3.1.3 Non-parametric Density Estimation

An unstructured approach can also be used to model multimodal distributions. We propose a kernel-based density estimation inspired by the work of Elgammal *et al.* [7]. The density at s is estimated using,

$$P^m(\vec{I}_s^t) = \frac{1}{M \times M} \sum_{r \in \eta_s} \prod_{j=1}^{d} K_{\sigma_j}(\vec{I}_s^t(j) - \vec{B}_r(j)). \tag{3.13}$$

where K_{σ_j} is a kernel function, j is the color-space index and B is a background image. As suggested by Elgammal *et al.* [7], we implemented K_σ with a zero-mean, σ^2-variance Gaussian. In this way, a single global 1D lookup table with 256 entries can be pre-calculated to allow speedup during runtime. The table values are then accessed with the intensity difference $\vec{I}_s^t(j) - \vec{B}_r(j)$ as index. Let us also mention that the background frame B used in P^m can be updated at every time t to account for illumination variations as follows : $\vec{B}_s \leftarrow (1-\alpha)\vec{B}_s + \alpha\vec{I}_s^t$.

3.4 Light and Fast Method

Our *light and fast* method is based on two assumptions. The first one stipulates that the background distribution is temporally stationary, *i.e.* $P(\vec{B}_s^t)$ is independent of t in such a way that $P(\vec{B}_s^0) \approx P(\vec{B}_s^1) \approx P(\vec{B}_s^2) \approx \ldots$ (illumination variation will be addressed later). In other words, the background is assumed to be motionless. The second assumption stipulates that the background is piecewise temporal ergodic in time *i.e.*, composed of piecewise constant regions for which spatial average color equals its temporal and also its ensemble average [155]. This means that we assume that the average temporal color/intensity of a pixel s is the same as the average color of a uniform region containing s.

Here, we represent the background as a whole with a Gaussian mixture of the

form :

$$P(\vec{B}_s) = \sum_{c=0}^{\mathcal{M}-1} P(\vec{B}_s|\omega_c, \vec{\theta}_c)P(\omega_c) \tag{3.14}$$

where ω_c is a class label, $\theta = \{\vec{\theta}_c = (\vec{\mu}_c, \Sigma_c)|c \in [0, \mathcal{M}[\}$ is the set of Gaussian parameters to be estimated, $P(\omega_i)$ is the prior probability of class ω_i, and B is the background image from which the mixture is estimated [130]. This equation stipulates that each pixel s is a member of a class ω_c with proportion $P(\omega_c)$ and likelihood $P(\vec{B}_s|\omega_c, \vec{\theta}_c)$.

This being said, B can be segmented into regions of uniform color (or intensity) by estimating a label field x for which x_s takes a value in $\{\omega_0, ..., \omega_{\mathcal{M}-1}\}$. In other words, it is possible to assign one class ω_i to each pixel s in such a way that every pixel having label ω_i has a color that follows the Gaussian distribution $\mathcal{N}(\vec{\mu}_i, \Sigma_i)$. Since the color of a background pixel s is assumed to be constant in time (more or less a noise factor), its class label x_s is also assumed to be constant in time. Therefore, the probability of observing color $\vec{I}_s^t$ at time t is well approximated by $P(\vec{B}_s|x_s, \vec{\theta}_{x_s})$ where $x_s \in \{\omega_0, \omega_1, ..., \omega_{\mathcal{M}-1}\}$. Thus, a consequence of our piecewise-ergodic assumption is that the spatial noise within a class is assumed to be the same as the temporal background variation (noise) of the pixels belonging to that class, *i.e.* $P(\vec{I}_s^t) \approx P(\vec{B}_s|x_s, \vec{\theta}_{x_s})$. In this way, our method can model the background with a global mixture of $\mathcal{M}$ Gaussians as opposed to $\mathcal{A} \times \mathcal{B}$ Gaussians [33] or $\mathcal{A} \times \mathcal{B} \times \mathcal{M}$ Gaussians [29] where $\mathcal{A} \times \mathcal{B}$ is the total number of pixels.

3.4.1 Gaussian Parameter Estimation

In order to obtain a reliable estimate for θ (Gaussian mixture parameters) and x (label field) we use Pieczynski's Iterative Conditional Estimation (ICE) algorithm [160] since it allows simultaneous estimation of θ and x. The ICE algorithm can be outlined as follows :

1. [Initialization Step] The parameters $\hat{\theta}^{[0]}$ and the label field $\hat{x}^{[0]}$ are initialized

with the K-Means [138] algorithm. $p = 1$.

2. [Stochastic Step] With a Gibbs sampler, a label field $\hat{x}^{[p]}$ is generated according to the posterior distribution $P(\hat{x}^{[p-1]}|B, \hat{\theta}^{[p-1]})$.
3. [Estimation Step] With a maximum likelihood estimator, $\hat{\theta}^{[p]}$ is recomputed based on $\hat{x}$ and B. In our implementation, $\vec{\mu}_c^{[p]}$ and $\Sigma_c^{[p]}$ are computed with a classical empirical mean and variance-covariance estimator for each class.
4. If $||\vec{\mu}^{[p]} - \vec{\mu}^{[p-1]}|| < T$, where T is a fixed threshold, then Stop. Else, $p = p + 1$ and go back to the Stochastic Step.

Here, the posterior distribution is modeled after Bayes' theorem,

$$P(\hat{x}^{[p]}|B, \hat{\theta}^{[p]}) \propto P(B|\hat{x}^{[p]}, \hat{\theta}^{[p]})P(\hat{x}^{[p]}). \tag{3.15}$$

Assuming independence of $\vec{B}_s$ given $\hat{x}_s^{[p]}$ and $\hat{\theta}^{[p]}$,

$$P(\hat{x}^{[p]}|B, \hat{\theta}^{[p]}) \propto \prod_s P(\vec{B}_s|\hat{x}_s^{[p]}, \hat{\theta}^{[p]})P(\hat{x}_s^{[p]}) \tag{3.16}$$

where $P(\vec{B}_s|\hat{x}_s^{[p]}, \hat{\theta}^{[p]})$ is a Gaussian distribution $\mathcal{N}(\vec{B}_s; \vec{\mu}_{\hat{x}_s}^{[p]}, \Sigma_{\hat{x}_s}^{[p]})$. As for $P(\hat{x}_s^{[p]})$, we use the simple Potts model based on a Gibbs distribution of the form $P(\hat{x}_s^{[p]}) \propto \exp[-\gamma U_{\eta_s}(\hat{x}_s^{[p]})]$ where γ is a constant and $U_{\eta_s}(\hat{x}_s^{[p]})$ is an energy function that counts the number of sites in the neighborhood η_s with a label different than $\hat{x}_s^{[p]}$. In our implementation, we use binary cliques linking site s to its eight spatial neighbors (second-order neighborhood).

3.4.2 Detecting Motion

Once the ICE algorithm has converged, we have in hand both the $\mathcal{M}$-class Gaussian mixture parameters modeling the background and the label field x indicating to which class each pixel has been assigned. Like most already-published motion detection methods, we assume that the color distribution of the moving objects is different from the background. In this way, since each pixel s is modeled

by a global Gaussian distribution with parameters $\vec{\theta}_{x_s} = (\vec{\mu}_{x_s}^{[p]}, \Sigma_{x_s}^{[p]})$, we can expect that $P(\vec{I}_s^t) \approx P(\vec{B}_s|x_s, \vec{\theta}_{x_s})$ when pixel s in image I^t is part of the background and $P(\vec{I}_s^t) \neq P(\vec{B}_s|x_s, \vec{\theta}_{x_s})$ when pixel s is covered by a moving object. In this way, the detection criterion may be formulated as :

$$L_s^t = \begin{cases} 1 & \text{if } P(\vec{I}_s^t|x_s, \vec{\theta}_{x_s})/P(\vec{B}_s|x_s, \vec{\theta}_{x_s}) < \tau \\ 0 & \text{otherwise.} \end{cases} \tag{3.17}$$

Alternatively, for computational reasons, the Mahalanobis distance [33] can also be used as follows :

$$L_s^t = \begin{cases} 1 & \text{if } |D(\vec{I}_s^t, \vec{\theta}_{x_s}) - D(\vec{B}_s, \vec{\theta}_{x_s})| > \tau' \\ 0 & \text{otherwise.} \end{cases} \tag{3.18}$$

where $D(\vec{a}_s, \vec{\theta}_{x_s}) = (\vec{a}_s - \vec{\mu}_{x_s})^T \Sigma_{x_s}^{-1} (\vec{a}_s - \vec{\mu}_{x_s})$. To further reduce processing times, for grayscale sequences, $D(\vec{I}_s^t, \vec{\theta}_{x_s})$ is pre-calculated and kept in memory in a global look-up table. Also, because empirically $D(\vec{B}_s, \vec{\theta}_{x_s})$ rarely goes above 32, we pre-calculate $D(\vec{B}_s, \vec{\theta}_{x_s})$ and store it in a 2D array allowing five bits per pixel. To account for illumination variation, the background pixels may have their statistics updated at each time t [124].

3.5 Experimental Results

The tests aim to evaluate the performance of our two spatial methods compared to temporally-trained methods. Three temporal methods have been implemented which, respectively, model every pixel with one Gaussian, a mixture of 3 Gaussians and a non-parametric kernel summation. Each of these temporal methods is trained on 15 to 80 frames depending on the sequence. A neighborhood η_s of size between 11×11 and 15×15 has been used for our *robust* method and a number of classes $\mathcal{M}$ between 7 and 10 was used for the *light and fast* method. Also, the threshold τ for every method has been varied to estimate the ROC curves of Fig. 3.4 and tuned

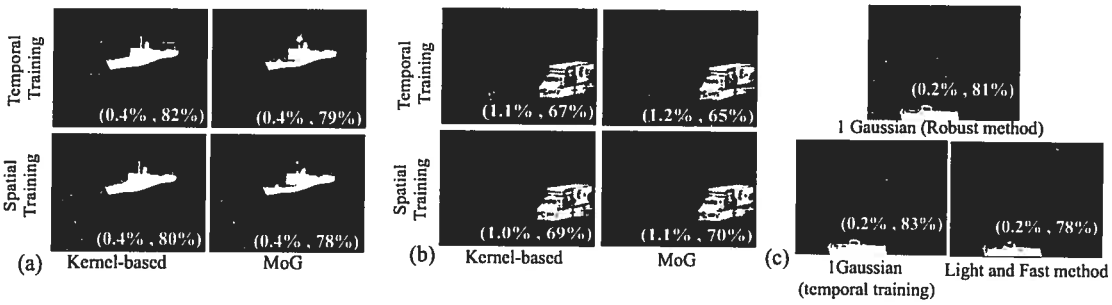

FIG. 3.3 – (a) A 200-frame synthetic sequence of a boat sailing on a wavy sea and (b) 90-frame sequence shot with an unstable camera. In each case, the MoG and the kernel-based method have been used. Both were trained either temporally or spatially. (c) Results for a 100-frame sequence corrupted by rain segmented with one Gaussian per pixel. The numbers in the lower right corner indicate the average percentage of false positives and true positives. These sequences are shown in Figure 3.1.

to produce the best possible results for the sequences presented in Fig.3.6 and 3.5. Also, $c = 5$, $\sigma = 2$, $\gamma = 2$, the learning rate α was set to 0 for every sequence (thus exploiting statistics of a single frame), and a simple 3×3 median filter was used to smooth out every motion mask L_t.

In order to gauge performance of our approach, sequences with known ground truth have been used (see Fig. 3.3). The first sequence is computer generated and exhibits a boat sailing on a wavy sea. Since the pixels representing water are multimodal, we used the kernel-based and MoG approaches to segment the sequence. The second sequence shows a moving truck filmed with an unstable camera making the background highly multimodal. The MoG and the Kernel-based approaches have been used again. The third sequence (Fig. 3.3 (c)) with known groundtruth, contains fixed background, strongly corrupted by rain. For this sequence, three methods have been used to detect motion : our *light and fast* method, our *robust* method with one Gaussian, and a single-Gaussian temporal method similar to Wren *et al.*'s [33]. Note that for this sequence, only the unimodal PDF P^u has been used for the *robust* methods. For these three sequences, ROC curves [130] have been generated and, as it can be seen, the difference between the spatially-trained methods and the temporally-trained methods is small. Note that for the Shaky

camera and Rain sequences, the percentages of false positives and true positives have been obtained based on hand-segmented ground truth images.

We also segmented a sequence having no training frame without moving objects (see Fig. 3.5). The background frame B used to learn the Gaussian parameters was computed with a basic five-frame median filter : $\vec{B}(s) = \text{Med}[I_s^0, I_s^{40}, I_s^{80}, I_s^{120}, I_s^{160}]$. The reader should note that, aside from the median filter, no outlier-detection method has been used.

To evaluate the computational complexity, we ran the methods on color and grayscale sequences of different sizes. Here, the MoG mixes two Gaussians, the temporal kernel-based method uses 30 training frames, and our *light and fast* method uses 8 classes. As it can be seen in Table 1, our *light and fast* method is significantly faster, especially for grayscale sequences for which look-up tables have been used to store $D(\vec{I}_t(s), \vec{\theta}_{x_s})$ in memory (these processing rates include the 3×3 median filtering). Also, the minimum amount of memory required to model the background is significantly smaller for our *light and fast* method than for the other ones. Every program has been executed on a 2.2 GHz AMD Athlon processor.

3.6 Conclusion

In this paper, a novel spatial framework for the background subtraction problem has been presented. Our framework is based on the idea that, for many applications, the temporal intensity/color distribution observed over a pixel corresponds to the statistical distribution observed spatially around that same pixel.

Our framework offers three main advantages. First, the statistical parameters can be learned with one background frame instead of a series of frames as is usually the case for single-Gaussian and the MoG model. This has the advantage of requiring less memory and being more flexible in the presence of sequences having no training frames without moving objects. Second, as opposed to the kernel-based method, only one frame (instead of N) is kept in memory at runtime. This again is a major advantage memory-wise. Last, but not least, our framework maintains

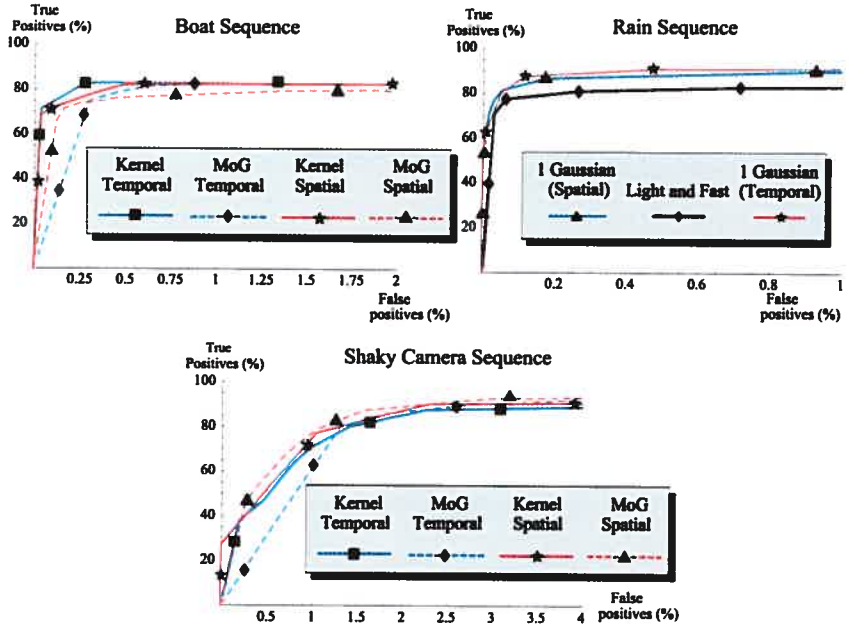

FIG. 3.4 – ROC curves obtained for three sequences shown in Figure 3.1.

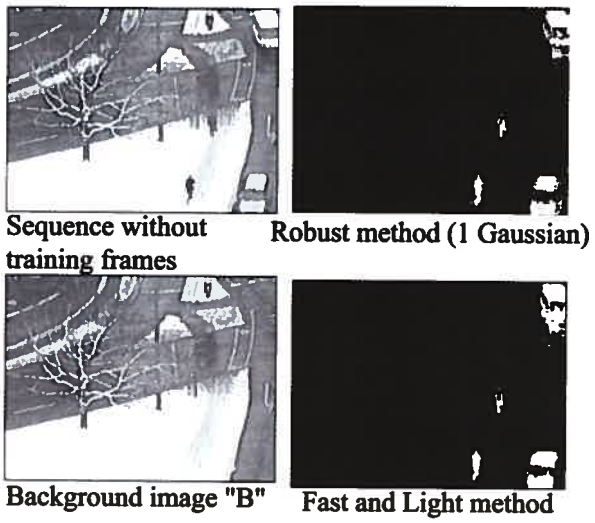

FIG. 3.5 – Results for a 200-frame sequence with no frame absent of moving objects. A median filter was used to produce the background frame B.

TAB. 3.1 – Tables showing respectively the frame rate and the minimum amount of memory needed to model the background.

Frame rate (fps)				
	Grayscale sequence		Color sequence	
Methods	256 x 256	640 x 480	256 x 256	640 x 480
Light and Fast	240	44	136	26
One Gaussian	128	25	108	20
Mixture	23	5	12	3
Kernel	10	4	9	2
Minimum memory to model the background (KB)				
	Grayscale sequence		Color sequence	
Methods	256 x 256	640 x 480	256 x 256	640 x 480
Light and Fast	72	308	64	300
One Gaussian	512	2400	1536	7200
Mixture	1536	7200	3584	16800
Kernel	1921	9001	5761	27001

the conceptual simplicity and strength of the background subtraction methods it is based on. The detection function, the adaptation to global illumination variations and the statistical learning phase are implemented in a way that is very similar to the one originally proposed in the temporal framework. In spite of these advantages, our framework does have one limitation. As we mentioned previously, our approach is based on the assumption that the temporal distribution of a pixel s can be approximated by a spatial distribution. Although this assumption is true for many real-life scenarios, it nonetheless fails to recognize "temporal textures" arising from a temporally-periodic event such as a blinking light. To compensate for this limitation, a specific updating scheme such as the one from Eq.(3.10) - (3.12), would need to be implemented.

In summary, we have shown that results obtained with our framework, on sequences with noise, camera jitter and animated background are, for all practical purposes, similar to the ones obtained with temporal methods.

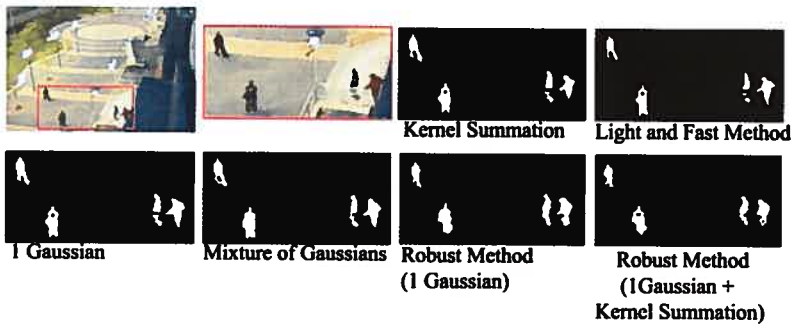

FIG. 3.6 – Results for a 100-frame noisy sequence segmented with one Gaussian per pixel whose parameters have been spatially and temporally trained.

Acknowledgements

The authors are grateful to the group of Prof. H.-H. Nagel for the winter sequence of Fig. 3.5 and to Dr. Ahmed Elgammal for the sequences of Fig. 3.1 (a) and (c).

CHAPITRE 4

SEGMENTATION MARKOVIENNE ET ESTIMATION DE PARAMETRES À L'AIDE D'UN PROCESSEUR GRAPHIQUE

Cet article a été publié dans le journal *Journal of electrical imaging* comme l'indique la référence bibliographique

P-M. Jodoin, M. Mignotte. Markovian Segmentation and Parameter Estimation on Graphics Hardware

Journal of Electrical Imaging, septembre 2006, Volume 15(3),033005.

Cet article est présenté ici dans sa version originale.

Résumé

Dans ce chapitre, nous démontrons comment des algorithmes de segmentation et d'apprentissage markoviens peuvent être accélérés de façon significative lorsqu'implémentés sur une architecture parallèle programmable. Parmi les applications étudiées figurent l'estimation de mouvement, la détection de mouvement, la segmentation de mouvement, la stéréovision et la segmentation de couleur. Nous démontrons comment les processeurs GPU (les *Graphics Processing Units*) disponibles sur la plupart des cartes graphiques en vente aujourd'hui, peuvent être utilisés pour inférer des champs d'étiquettes. Les applications étudiées sont ici exprimées sous la forme de problèmes à résoudre à l'aide d'un maximum *a posteriori* (MAP). Dans chaque cas, le champ d'étiquettes est inféré avec un algorithme déterministe tel l'ICM de Besag ou encore un algorithme stochastique tel le recuit simulé. Pour les segmentations de type probabiliste, les paramètres statistiques sont estimés à l'aide des algorithmes K-Moyennes et ICE. Tous les algorithmes de segmentation et d'estimation de paramètres utilisent le processeur de *fragments* du GPU ainsi que la mémoire texture pour stocker les images à segmenter ainsi que les champs d'étiquettes. Les résultats obtenus démontrent que les applications Markoviennes peuvent être accélérées d'un facteur allant de 4 à 200 et ce, sans

exiger de connaissances particulières en programmation bas-niveau.

Abstract This paper shows how Markovian strategies used to solve well known segmentation problems such as *motion estimation*, *motion detection*, *motion segmentation*, *stereovision* and *color segmentation* can be significantly accelerated when implemented on programmable graphics hardware. More precisely, it exposes how the parallel abilities of a standard Graphics Processing Unit usually devoted to image synthesis can be used to infer the labels of a segmentation map. The problems addressed in this paper are expressed a the maximum a posteriori with an energy-based or probabilistic formulation, depending on the application. In each case, the label field is inferred with an optimization algorithm such as ICM or simulated annealing. In the case of probabilistic segmentation, parameters are estimated with the K-means and ICE procedure. For both the optimization and the parameter estimation algorithms, the GPU's *fragment processor* is used to update in parallel each label of the segmentation map while rendering passes and graphics textures are used to simulate optimization iterations. The hardware results, obtained with a mid-end graphics card, show that these Markovian applications can be accelerated by a factor of 4 to 200 without requiring any advanced skills in hardware programming.

4.1 Introduction

Image segmentation is generally understood as a means of dividing an image into a set of uniform regions. Here, the concept of *uniformity* refers to image features such as color, depth or motion. Among the existing segmentation methods proposed in the literature, segmentation models can roughly be divided between *feature-space based* and *image-space based* families [97]. As the name suggests, feature-space algorithms segment images on the basis of a *feature* criterion whereas space-based algorithms segment images on the basis of a *feature-space* criterion. Because image-space based techniques incorporate information from the image to be segmented and the segmentation map (typically *likelihood* and *prior* information), the results they produce are often more precise, although at the cost of a heavier computational load.

Among the *image-space based* techniques are the Markovian algorithms [71,141] which incorporate spatial characteristics by using Markov Random Fields (MRF) as *a priori* models. The first contribution in that field was made by Geman et al. [141] who proposed the concept of *Maximum a Posteriori* (MAP) as an *image-space* probabilistic criterion. Shortly afterward, many Markovian models were adapted to solve all kinds of segmentation problems ranging from motion estimation to stereovision [146]. While some authors proposed *ad-hoc* energy-based solutions, others used probabilistic functions to model the way the desired (hidden) label field is distributed. The shape of these probabilistic functions depends on parameters that are either supposed to be known (or manually adjusted) or estimated in a first step of processing. In the latter case, estimation algorithms such K-means or the Iterative Conditional Estimation (ICE) [23,160] have demonstrated their efficiency.

Although Markovian approaches have several appealing advantages, they are relatively slow algorithms and thus inappropriate in applications for which time is an important factor. This paper explains how Markovian segmentation algorithms used to solve segmentation problems such as *motion detection* [116], *motion estimation* [75], *stereovision* [46], *color segmentation* [97], and *motion segmentation* can be

significantly accelerated when implemented on programmable graphics hardware. This is made possible by the use of the so-called *Graphics Processor Unit* (GPU) embedded on most graphics hardware nowadays available. This unit can load and execute *in parallel* general purpose programs independently of the CPU and the central memory. As the name suggests, the GPU architecture was optimized to solve typical graphics problems with the goal of rendering complex scenes in real-time. Because of the very nature of conventional graphics scenes, graphics hardware has been designed to efficiently manipulate texture, vertices and pixels. What makes these graphics processors so efficient is their fundamental ability to process vertices and fragments (read pixels) in parallel, resulting in attractive acceleration factors.

In spite of appearances, it is possible to take advantage of the parallel abilities of programmable graphics hardware to solve problems that go beyond graphics. This is what people call *general-purpose computation on GPUs* (GPGPU) [2]. Various authors have shown that applications such as fast Fourier transforms [91], linear algebra [76], motion estimation and spatial segmentation with level sets could run on graphics hardware [2,136]. Even if these applications have little in common with traditional computer graphics, they all share a common characteristic : they are problems that can be solved by parallel algorithms.

Parallel implementations of Markovian algorithms applied to motion detection [25] and picture restoration [42] have been proposed in the past. Unfortunately, these methods were build upon dedicated, expensive and sometimes obsolete architectures. This paper shows how cheap and widely distributed graphics hardware can be used to significantly accelerate Markovian segmentation. Even though GPUs are cutting edge technologies made for graphics rendering, implementing a MAP segmentation algorithm on a fragment processor isn't much more difficult than writing it in a C-like procedural language.

The remainder of the paper is organized as follows. In Section 4.2, a review of the Markovian segmentation theory is proposed after which Section 4.3 presents three energy-based segmentation problems. Section 4.4 follows with the probabilistic (and unsupervised) problem of motion and color segmentation which is followed

by Section 4.5 and 4.6 present the optimization algorithms ICM and SA and the parameter estimation algorithms K-means and ICE. Section 4.7 gives an in-depth look at graphics hardware architecture and explains how the algorithms previously presented can be implemented on a graphics hardware. Finally, Section 4.8 gives experimental results. Section 4.9 is the conclusion.

4.2 Markovian Segmentation

The applications this contribution tackles aim at subdividing observed input images into uniform regions by grouping pixels having features in common such as color, motion or depth. Starting with some observed data Y (typically one or more input images), the goal of a segmentation program is to infer a label field X containing class labels (i.e. labels indicating whether or not a pixel belongs to a moving area or a certain depth range for instance). The variables X and Y are generally defined over a rectangular lattice of size $\mathcal{N} \times \mathcal{M}$ represented by $S = \{s|0 \leq s < \mathcal{N} \times \mathcal{M}\}$ where s is a site located at the Cartesian position (i, j) (for simplicity, s is sometimes defined as a pixel). It is common to represent by a lower-case variable such as x or y, a realization of the label field or the observation field. For each site $s \in S$, its corresponding element x_s in the label field takes a value in $\Gamma = \{e_1, e_2, ..., e_N\}$ where N is the total number of classes. In the case of motion detection for example, N can be set to 2 and $\Gamma = \{\text{StaticClass,MobileClass}\}$. Similarly, the observed value y_s takes a value in $\Lambda = \{\epsilon_1, \epsilon_2, ..., \epsilon_\zeta\}$ where ζ can be set, for instance, to 2^8 for gray-scale images and 2^{24} for color images.

In short, a typical segmentation model is made up of two fields x and y, i.e. an observation field (y) that is to be decomposed into N classes by inferring a label field (x).

In the context of this paper, the goal is to find an *optimal* labeling $\hat{x}$ that maximizes the *a posteriori* probability $P(X = x|Y = y)$ (represented by $P(x|y)$ for simplicity), also called the *maximum a posteriori* (MAP) estimate [141] : $\hat{x}_{\text{MAP}} =$

$\arg\max_x P(x|y)$. With Bayes theorem, this equation can be rewritten as

$$\hat{x}_{\text{MAP}} = \arg\max_x \frac{P(y|x)P(x)}{P(y)} \tag{4.1}$$

or equivalently $\hat{x}_{\text{MAP}} = \arg\max_x P(y|x)P(x)$ since $P(y)$ isn't related to x. Assuming that X and Y are Markov Random Fields (MRF) and according to the Hammersley-Clifford theorem [141], the *a posteriori* probability $P(x|y)$ –as well as the likelihood $P(y|x)$ and the prior $P(x)$– follows a Gibbs distribution, namely

$$P(x|y) = \frac{1}{\lambda_{x|y}} \exp(-U(x,y)) \tag{4.2}$$

where $\lambda_{x|y}$ is a normalizing constant and $U(x,y)$ is an *energy function*. Combining Eq. (4.1) and (4.2), the optimization problem at hand can be formulated as an *energy minimization problem* i.e.,

$$\hat{x}_{\text{MAP}} = \arg\min_x \big(W(x,y) + V(x)\big) \tag{4.3}$$

where $W(x,y)$ and $V(x)$ are respectively the likelihood and prior energy functions. If we assume that the noise in y isn't correlated, the global energy function $U(x,y)$ can be represented by a sum of local energy functions

$$\hat{x}_{\text{MAP}} = \arg\min_x \sum_{s \in S} \left(W_s(x_s, y_s) + V_{\eta_s}(x_s)\right). \tag{4.4}$$

Here, η_s is the neighborhood around site s and $V_{\eta_s}(x_s) = \sum_{c \in \eta_s} V_c(x_s)$ is a sum of potential functions defined on so-called cliques c. Notice that function $V_c(x_s)$ defines the relationship between two neighbors in c, a binary clique linking a site s to a neighbor r.

An ad hoc definition of $W_s(x_s, y_s)$ and $V_{\eta_s}(x_s)$ lead to a so-called *energy-based segmentation*. By opposition, when $W_s(x_s, y_s)$ and $V_{\eta_s}(x_s)$ are defined by a probabilistic law linking x_s to y_s, the segmentation is called *probabilistic*. In both cases though, $\hat{x}_{\text{MAP}}$ is estimated with an optimization procedure such as SA or ICM which

are typically slow algorithms. Details of these algorithms are discussed in Section 4.5.

4.3 Energy-based segmentation

This Section shows how typical computer vision problems can be expressed as the minimum of a global energy function made of a likelihood and a prior term.

4.3.1 Motion Detection

The goal of motion detection is to segment an animated image sequence into *mobile* and *static* regions. For this kind of application, moving pixels are typically the ones with a non-zero displacement, no matter what direction or speed they might have. Motion detection is thus a spatial case of motion segmentation. The solution presented in this Section was inspired by the work of Bouthemy and Lalande [116] which proposed one of the first energy-based Markovian solution to that problem.

Let us mention that their paper influenced many subsequent contributions including the one by Dumontier et al. [25] who proposed a parallel hardware architecture to detect motion in real time. Unfortunately, the hardware they used was specifically designed and is not, to our knowledge, available on the market. In addition, the design of their hardware architecture is different from standard graphics hardware.

The solution here proposed is based on the concept of temporal gradient and doesn't require the estimation of an optical flow. From two successive frames $f(t)$ and $f(t+1)$, the observation field y is defined as the temporal derivative of the intensity function df/dt namely $y = ||f(t+1) - f(t)||$. Assuming that scene illumination variation is small, the likelihood energy function linking the observation field to the label field is defined by the following equation

$$W(x_s, y_s) = \frac{1}{\sigma^2}(y_s - m_p x_s)^2 \tag{4.5}$$

where m_p is a constant and σ is the variance of the Gaussian noise. Because of the very nature of the problem, $N = 2$ and $x_s \in \{0, 1\}$ where 0 and 1 correspond to static and moving labels. As for the prior energy term, as in [116] and [25], the following Potts model was implemented

$$V_c(x_s) = \begin{cases} 1 & \text{if } x_s \neq x_r \\ 0 & \text{otherwise.} \end{cases} \tag{4.6}$$

The overall energy function to be minimized is thus defined by

$$U(x, y) = \sum_{s \in S} (\underbrace{\frac{1}{\sigma^2}(y_s - m_p x_s)^2}_{W(x_s, y_s)} + \beta_{\text{MD}} \underbrace{\sum_{c \in \eta_s} V_c(x_s)}_{V_{\eta_s}(x_s)}) \tag{4.7}$$

where η_s is a second order neighborhood (eight neighbors) and β_{MD} is a constant. Please remark that this solution makes the implicit assumption that the camera is still and that moving objects were shot in front of a static background. To help smooth out interframe changes, one can add a temporal prior energy term $V_\tau(x_s)$ linking label x_s estimated at time t and the one estimated at time $t - 1$.

4.3.2 Motion Estimation

The goal of motion estimation is to estimate the direction and magnitude of optical motion over each site $s \in S$ of an animated sequence [13, 64, 70]. Among the solutions proposed in the literature, many are based on an hypothesis called *lightness consistency*. This hypothesis stipulates that a site $s \in S$ at time t keeps its intensity after it moved to site $s + \vec{v}_s$ at time $t + 1$. Although this hypothesis excludes noise, scene illumination variation, and occlusions (and thus is an extreme simplification of the true physical nature of the scene) it allows simple energy functions to generate fairly accurate results. Under the terms of this hypothesis, the goal of motion estimation is to find, for each site $s \in S$, an optical displacement vector $\vec{v}_s$ for which $f_s(t) \approx f_{s+\vec{v}_s}(t + 1)$. In other words, the goal is to find a vector

field $\hat{v}$ for which

$$\hat{\vec{v}}_s = \arg\min_{\vec{v}_s} ||f_s(t) - f_{s+\vec{v}_s}(t+1)||, \quad \forall s \in S. \tag{4.8}$$

Notice that the absolute difference could be replaced by a cross-correlation distance for more robustness. Such strategy is sometimes called in the literature *region-based matching* [70]. But anyway, when estimating motion, the observation field y is the input image sequence f and $y(t)$ is a frame at time t. Furthermore, the label field x is a vector field made of 2D vectors defined as $x_s = \vec{v}_s = (\zeta_i, \zeta_j)$ where ζ_i, ζ_j are integers taken between $-d_{max}$ and d_{max} as shown in Fig. 4.3.2.

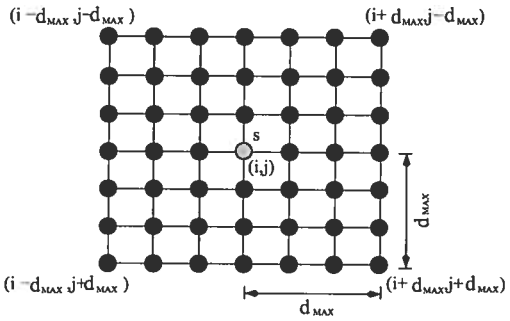

FIG. 4.1 – *The total number of possible displacement vector for a site* $s \in S$ *is* $(2d_{max} + 1)^2$.

Eq. (4.8) has one major limitation though which comes from the fact that real-world sequences contain textureless areas and/or areas with occlusions. Typically, over these areas several vectors x_s can have a minimum energy, although only one is valid. This is the well known *aperture problem* [102]. In order to guaranty the uniqueness of a consistent solution, some approaches have been proposed [64]. Among these approaches, many opt for a regularization term (or *smoothness constraints*) whose essential role is to rightly constrain the ill-posed nature of this inverse problem. These constraints typically encourage neighboring vectors to point in the same direction with the same magnitude. In the context of the MAP, these constraints can be expressed as a prior energy function such as the

Potts model of Eq. (4.6). However, since the number of labels can be large (here $(2d_{max}+1)^2$), we empirically observed that a smoother function is better suited. In fact, the following linear function was implemented

$$V_{\eta_s}(x_s) = \sum_{\substack{c\in\eta_s \\ s,r\in c}} (|x_s[0]-x_r[0]| + |x_s[1]-x_r[1]|) \tag{4.9}$$

where c is a binary clique linking site s to site r. Notice that other smoothing functions are available [102]. The global energy function $U(x,y)$ to be minimized is obtained by combining Eq. (4.8) and (4.9) as follows

$$U(x,y) = \sum_{s\in S} \underbrace{(||y_s(t) - y_{s+x_s}(t+1)||}_{W_s(x_s,y_s)} + \beta_{\text{ME}} \underbrace{\sum_{c\in\eta_s} (|x_s[0]-x_r[0]| + |x_s[1]-x_r[1]|))}_{V_{\eta_s}(x_s)} \tag{4.10}$$

where β_{ME} is a constant. Let us mentioned that Konrad and Dubois [75] proposed a similar solution involving a *line process* to help preserve edges.

4.3.3 Stereovision

The goal of stereovision is to estimate the relative depth of 3D objects from two (or more) images of a scene. For simplicity purposes, many stereovision methods use two images taken by cameras aligned on a linear path with parallel optical axis (this setup is explained in detail in Scharstein and Szelisky's review paper [46]). Stereovision algorithms often make some assumptions on the true nature of the scene. One common assumption (which is similar to motion estimation's lightness consistency assumption) states that every point visible in one image is also visible (with the same color) in the second image. Based on that assumption, the goal of a stereovision algorithm is to estimate the distance between each site s –with coordinate (i,j)– in one image to its corresponding site t –with coordinate $(i+d_s,j)$– in the second image. Such distance is called *disparity* and is proportional to the inverse depth of the object projected on site s. In this contribution, $d_s \in [0, D_{\text{MAX}}]$.

This gives rise to a *matching cost* function that measures how good a disparity d_s is for a given site s in a reference image y_{ref}. This is expressed mathematically by

$$C(s,d,y) = ||y_{\text{ref}}(i,j) - y_{\text{mat}}(i+d_s,j)|| \tag{4.11}$$

where $y = \{y_{mat}, y_{ref}\}$ and y_{mat} is the second image familiarly called the *matching image*. Notice that the function $||.||$ can be replaced by a robust function [46] if needed. In the context of the MAP, $C(.)$ is the likelihood energy function and the disparity map d is the label field to be estimated. Thus, to ensure uniformity with Section 4.2's notation, the cost function of Eq. (4.11) will be defined as $C(s,x,y)$.

To ensure spatial smoothness, two strategies have been traditionally proposed. The first one is to convolute $C(s,x,y)$ with a low-pass filter or a so-called *aggregation filter* w (see [46] for details on aggregation). Although a pre-filtering step slows down the segmentation process, it can significantly reduce the effect of noise and thus enhance result quality. The second strategy to ensure spatial smoothness is to take advantage of a prior energy term $V_{\eta_s}(x)$ of the form

$$V_{\eta_s}(x) = \sum_{\substack{c \in \eta_s \\ s,r \in c}} |x_s - x_r| \tag{4.12}$$

where the absolute value could be replaced by another cost function if needed. The global energy function $U(x,y)$ can thus be written as

$$U(x,y) = \sum_{s \in S} (\underbrace{(w * C)(s,x,y)}_{W_s(x_s,y_s)} + \beta_{\text{S}} \underbrace{\sum_{\substack{c \in \eta_s \\ s,r \in c}} |x_s - x_r|}_{V_{\eta_s}(x_s)}) \tag{4.13}$$

where β_{S} is a constant.

Minimizing the energy function of Eq. (4.13) can be time consuming, especially when the number of disparities D_{MAX} is large. To save on processing time, two simple strategies are conceivable. The first one is to minimize the likelihood function only and ignore the prior term : $\hat{x}_s = \arg\min_{x_s}(w * C)(s,x,y))$. In this way, the

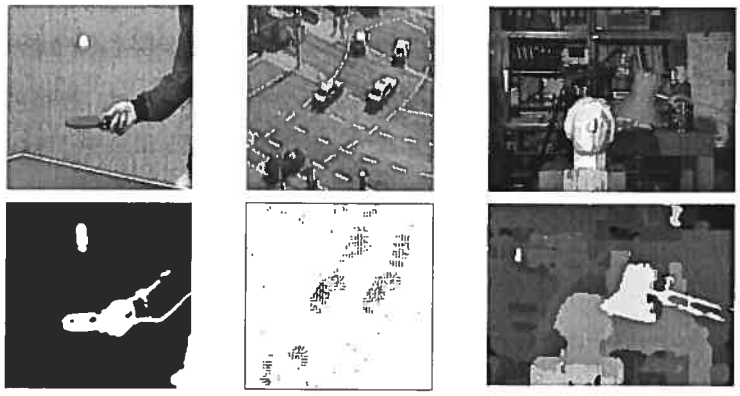

FIG. 4.2 – *Motion detection, motion estimation, and stereovision label fields obtained after a Markovian optimization.*

filter w is assumed to be good enough to ensure spatial smoothness. This simple greedy strategy is called *Winner-Take-All* (WTA) and converges after only one iteration. Another way to reduce processing time is to pre-compute $C(s,x,y)$ in a 3D table. Such table –also called the *Disparity Space Integration* (DSI) table– contains $\mathcal{N} \times \mathcal{M} \times D_{\text{MAX}}$ cost values. Once the DSI table has been computed, is can be filtered by w after which the optimization process can be launched.

4.4 Probabilistic segmentation

In this paper, both the color and the motion segmentation are based on a *probabilistic* criterion which relates the observed data y_s to its label x_s with a distribution $P(y_s|x_s)$. For the color segmentation, y_s takes a value between 0 and 255 for gray-scale images and between $(0,0,0)$ and $(255,255,255)$ for color images. As for motion segmentation, y_s is a two-dimensional vector represented by $y_s = \vec{v}_s = (u_s, v_s)$ where u_s and v_s are real values. Notice that since y_s is an observation, the motion vector $\vec{v}_s$ $\forall s \in S$ is assumed to have been priorly estimated.

Because y_s is related to x_s by a probability distribution, the energy function $W_s(x_s, y_s)$ is designed according to that distribution, namely $W_s(x_s, y_s) \propto -\ln P(y_s|x_s)$. A very popular function used to model $P(y_s|x_s)$ is the multidimen-

sional Gaussian distribution

$$P(y_s|x_s) = \frac{1}{\sqrt{(2\pi)^d|\Sigma_{x_s}|}} \exp\{-\frac{1}{2}(y_s-\mu_{x_s})\Sigma_{x_s}^{-1}(y_s-\mu_{x_s})^{\mathrm{T}}\} \tag{4.14}$$

where d is the dimensionality of y_s ($d = 1$ or 3 for color segmentation and $d = 2$ for motion segmentation) and $(\mu_{x_s}, \Sigma_{x_s})$ are the mean and variance-covariance matrix of class x_s. Using a classical Potts function as prior model, the global energy function of Eq. (4.4) can be written as

$$\sum_{s\in S}\{\frac{1}{2}(\ln|\Sigma_{x_s}| + (y_s - \mu_{x_s})\Sigma_{x_s}^{-1}(y_s - \mu_{x_s})^{\mathrm{T}}) + V_{\eta_s}(x_s)\}. \tag{4.15}$$

In the case of *unsupervised* segmentation, the Gaussian parameters $\Phi = \{(\mu_k, \sigma_k) \mid 1 \leq k < N\}$ has to be estimated conjointly with x or preliminary to the segmentation step. To do so, many parameter estimation algorithms are available among which K-means [23] and ICE [160] are commonly used. For further details on probabilistic Markovian segmentation, the reader is invited to consider the following books [130, 146].

4.5 Optimization Procedures

Since Eq. (4.4) has no analytical solution, $\hat{x}$ has to be estimated with an *optimization* procedure. The first optimization procedure we have implemented is the simulated annealing (SA) which is a stochastic relaxation algorithm based on a Gibbs sampler. The concept of SA is based on the manner in which some material recrystallize when slowly cooled down after being heated at a high temperature. The final state (called the *frozen* ground state) is reached when temperature gets down to zero. Similarly, SA searches for the global minima by cooling down a temperature factor T [145] from an initial temperature T_{MAX} down to zero.

The major advantage with SA is its ability to always reach the global minima with the appropriate decreasing cooling temperature schedule. This is made possible because SA authorizes energy increases to escape form local minima. To do

so, SA stochastically samples the system probability distribution and randomly generates new configurations. In this paper, the system probability is made of the global energy function (here $U(x, y)$) and a temperature factor T. This probability function is similar to Boltzmann's probability function [145] which can be written as

$$P(x, y) = \frac{1}{\lambda} \exp\{-\frac{U(x, y)}{T}\}. \tag{4.16}$$

where λ is a normalization factor. The simulated annealing algorithm is presented in the upper section of Table 4.1. $!h$

The main limitation with SA is the number of iterations it requires to reach the frozen ground state. This makes SA unsuitable to many applications for which time is a decisive factor. This explains the effort of certain researchers to find faster optimization procedures. One such optimization procedure is Besag's ICM algorithm [71]. Starting with an initial configuration $x^{[0]}$, ICM iteratively minimizes $U(x, y)$ in a deterministic manner by selecting, for every site $s \in S$, the label $e_k \in \Gamma$ that minimizes the local energy function at that point. Since ICM isn't stochastic, it cannot exit from local minima and thus, requires x to be initialized near the global minima. In practice however, this limitation is rarely a problem since ICM generates fairly good results, always at a fraction of the time needed by SA (Fig. 4.3 illustrate the difference between ICM and SA). The ICM algorithm is presented in the lower section of Table 4.1.

FIG. 4.3 – *Difference between a label field x estimated with ICM (7 iterations) and SA (247 iterations).*

4.6 Parameter Estimation

The two parameter estimation algorithms we have implemented for this paper are K-means [23] and ICE [160]. K-means is an iterative clustering method that assumes input data $\{y_s\}$ is distributed within K spherical clusters of equal volume. At each iteration, every site s is first assigned to the nearest cluster before a second step re-estimates the center of mass of every cluster. The resulting K-means clusters minimize the sum-of-square error function : $\sum_{k=1}^{N} \sum_{x_s=e_k} ||y_s - \mu_k||^2$ [23]. The variance-covariance of each cluster is estimated once the algorithm has converged. $!h$ $!h$

K-means has two well known limitations. First, its assumption that all clusters are spherical with equal volume is simplistic an often unsuited to some observation fields. Second, because K-means is a deterministic algorithm, it is sensitive to noise and is likely to converge toward local minima. Consequently, some authors suggest to refine Φ with a more realistic model, less sensitive to noise and local minima such as the stochastic (and Markovian) ICE estimation algorithm. Further details on this algorithm are presented in [160] while Table 4.3 presents a version of ICE adapted to this paper.

4.7 Graphics Hardware Architecture

As mentioned previously, *graphics hardware* is highly optimized to solve traditional computer graphics problems. Nowadays, graphics hardware is generally embedded on a graphics card which can receive/send data from/to the CPU and the main memory via the system bus, be it PCI, AGP or PCIe (see Fig. 4.4). To our knowledge, most graphics hardware are designed to fit the graphics processing pipeline shown in Fig. 4.5 [20, 140]. This pipeline is made of various stages which sequentially transforms images and geometric input data into an output image stored in a section of graphics memory called the *framebuffer*. Part of the framebuffer (the front buffer) is meant to be visible on the display device.

Until recently, graphics pipelines have presented a flexible but static interface

1	$T \leftarrow T_{\text{MAX}}$
2	For each site $s \in S$
2a	$P(x_s = e_k \mid y_s) = \frac{1}{\lambda} \exp\left\{\frac{-1}{T} U(e_k, y_s)\right\}, \quad \forall e_k \in \Gamma$
2b	$x_s \leftarrow$ according to $P(x_s \mid y_s)$, randomly select $e_k \in \Gamma$
3	$T \leftarrow T *$cooling Rate
5	Repeat lines 2-3 until $T \leq T_{\text{MIN}}$

1	Initialize x
2	For each site $s \in S$
3	$x_s = \arg\min_{e_k \in \Gamma} U(e_k, y_s)$
4	Repeat lines 2-3 until x stabilizes

TAB. 4.1 – *Simulated annealing and ICM algorithms.*

1	$\mu_k \leftarrow$ random initialization $\qquad \forall \mu_k \in \Phi_\mu$
2	For each site $s \in S$
2a	$x_s \leftarrow \arg\min_{e_k \in \Gamma} \|\|y_s - \mu_{e_k}\|\|^2$
3	$\mu_k \leftarrow \frac{1}{N_k} \sum_{x_s = e_k} y_s \qquad \forall \mu_k \in \Phi_\mu$
4	Repeat lines 2-3 until each mean μ_k no longer moves
5	$\Sigma_k^{nm} \leftarrow \frac{1}{N_k} \sum_{x_s = e_k} (y_s^n - \mu_k^n)(y_s^m - \mu_k^m), \forall \Sigma_k \in \Phi_\Sigma$

TAB. 4.2 – *K-Means algorithm.*

1	$\Phi \leftarrow K$-means
2	For each site $s \in S$
2a	$P(e_k \mid y_s) = \frac{1}{Z_s} \exp\{(W(e_k, y_s) + V_{\eta_s})\}, \forall e_k \in \Gamma$
2b	$x_s \leftarrow$ according to $P(x_s \mid y_s)$, randomly select $e_k \in \Gamma$
3a	$\mu_k \leftarrow \frac{1}{N_k} \sum_{x_s = e_k} y_s \quad \forall \mu_k \in \Phi_\mu$
3b	$\Sigma_k^{nm} \leftarrow \frac{1}{N_k} \sum_{x_s = e_k} (y_s^n - \mu_k^n)(y_s^m - \mu_k^m), \quad \forall \Sigma_k \in \Phi_\Sigma$
5	Repeat lines 2-3 until Φ stabilizes

TAB. 4.3 – *ICE algorithm.*

to application programmers. Although many parameters on each processing stage could be tweaked to adjust the processing, the fundamental graphics operations couldn't be changed. To answer this limitation, hardware manufacturers have made *programmable* the vertex and fragment processing stages. These two stages can now be programmed using C-like languages to process vertex and fragments with user-defined operations. Let us mention that a fragment is a per-pixel data structure created at the rasterization stage. A fragment contains data such as color, texture coordinates, depth, and so on. Each fragment is used to update a unique location in the framebuffer. For example, a scene made of a five-pixel-long horizontal line will generate five fragments whereas a 100×100 plan perpendicular to an orthographic camera will generate 10000 fragments.

Because of the very nature of graphics applications, the *graphics processing unit* (the GPU) have been designed as a streaming processor, that is a processor with inherent parallel processing abilities. With such processor, the vertices and the fragments are processed in parallel, thus providing all graphics applications a significant acceleration factor.

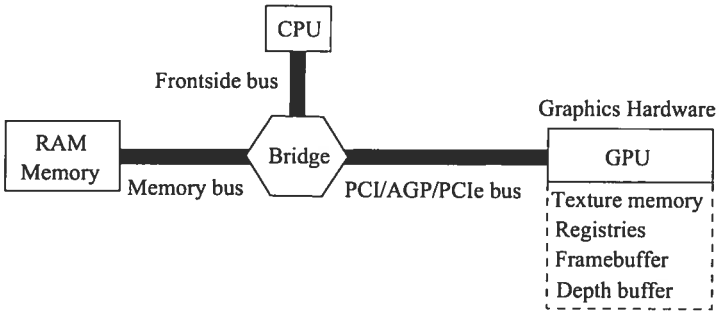

FIG. 4.4 – *Diagram showing physical relation between the CPU, the main memory and the graphics hardware.*

4.7.1 Fragment Programs

With a GPU, general-purpose applications going beyond traditional computer graphics can now be implement on graphics hardware [2] to take advantage of its pa-

rallel abilities. It is especially true for various image processing applications. These applications are generally executed over the fragment processor, mostly because it is the only part of the graphics pipeline that has access to both input memory (texture memory) and output memory (the framebuffer). It has also traditionally been the most powerful stage of the GPU.

A fragment processor is designed to load and execute in parallel a program (also called *shader*) on each fragment generated during the rasterization stage. As shown in Fig. 4.5(b), such program has typically access to three kinds of input values [132, 140]. First, a fragment shader has a read-only access to the texture memory. In this contribution, the texture memory contains the observation field y and the label field $x^{[t-1]}$ estimated during the previous optimization iteration. Secondly, fragment shaders have access to build-in variables containing general graphics informations such as the modelview matrix, the light sources, the fog, and the material to name a few. The only build-in variable used by our shaders is the one containing the fragment coordinates (i, j). Thirdly, fragment shaders have also access to user-defined variables containing all kinds of application-specific data such as weighting factors, class parameters, window size, and so on. This data is typically stored in arrays, vectors, integer or floating point variables.

A fragment shader can return *color*, *alpha* and *depth* values. The first two values are stored in the framebuffer where as the last one is copied in the depth buffer. In this contribution, only the values copied in the framebuffer are taken into account.

In short, a fragment shader is a program executed on the fragment processor that processes *in parallel* every fragment (see pixel) returned by the rasterization stage. This shader has access to user-defined parameters, has a read-only access to texture, a write-only access to the framebuffer and cannot exchange information with the neighboring fragments. It should be noted that while the GPU can be a very powerful processing tool, sending and receiving data from/to the CPU across the system bus introduces significant latency. As such, data traffic between the CPU application and the GPU should be kept at a strict minimum.

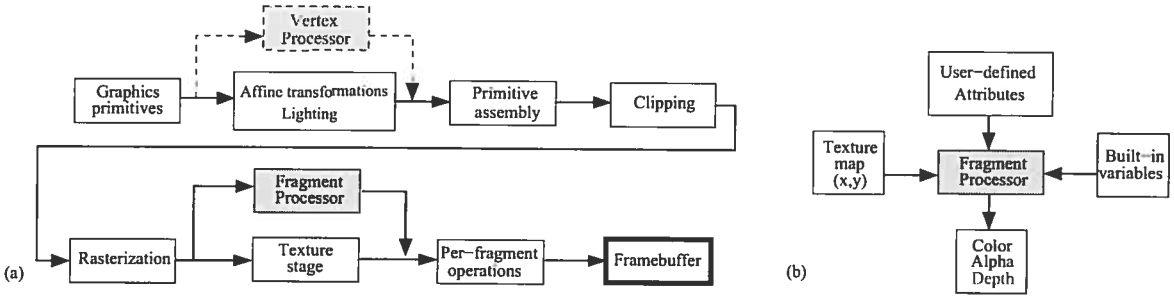

FIG. 4.5 – *(a) Logical diagram of a typical processing pipeline with programmable vertex and fragment processor. (b) Fragment processor inputs and outputs.*

4.7.2 Loading and Executing a Fragment Shader

As mentioned before, a fragment processor is made to load and execute in parallel a program called the *fragment shader.* !*h* As shown in Fig. 4.6, the shader source code is first located in a C or C++ application running on the CPU. Typically, this source code is listed in a $1D$ "unsigned char" array. While the C/C++ application is running, the shader source code is compiled and linked by the Application Program Interface (API)'s driver. Once the linking has finished, the linker returns a program object that is loaded on the graphics hardware for execution. The shader is executed when one or more graphics primitives are rendered. This procedure is illustrated in Table 4.4. Notice that most common APIs such as OpenGL and DirectX provide easy to use high-level driver functions [132, 140].

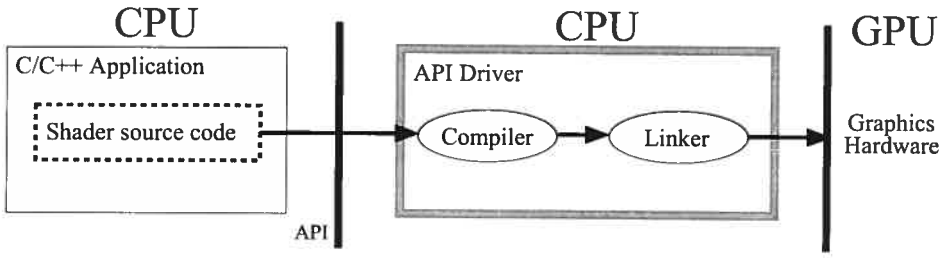

FIG. 4.6 – *Execution model of a typical shader.*

1	Buf ← Copy the shader source code in a 1D buffer
2	Give Buf to the Driver.
3	Compile the shader code
4	Link the compiled shader code
5	Send the linked code to the graphics hardware

TAB. 4.4 – *Algorithm of a C/C++ application compiling, linking and loading a shader. Notice that lines 2 to 5 are done by functions provided by the API driver.*

4.7.3 Markovian Segmentation on GPU

Although fragment shaders (as well as vertex shaders) can be written in a C-like procedural languages [132, 140], they have some specificities as compared to ordinary C/C++ programs. The most important ones are the following :

1. a fragment shader is made to process every fragment in parallel ;
2. the only memory in which a fragment shader can write into is the *framebuffer* and the *depth buffer* ;
3. the only data a fragment shader can read is contained in the texture memory, in built-in variables and in user-defined variables. As such, it cannot read the content of the framebuffer of the depth buffer ;
4. since fragments are processed in parallel, fragment shaders cannot exchange information. GPUs do not provide its shaders with access to general-purpose memory.

With such specificities, minimizing a global Markovian energy function such as Eq. (4.4) can be tricky. In fact, three main problems have to be overcome. Firstly, when performing a Markovian segmentation, the fragment operations should obviously be performed on every pixel of the input scene. As such, a perfect $1:1$ mapping from the input pixels to the output buffer pixels is necessary. This is achieved by rendering a screen-aligned rectangle covering a *window* with exactly the same size than the input image. To alleviate all distortion due to perspective,

we also used an orthographic camera. In this way, the rasterization stage generates $\mathcal{N} \times \mathcal{M}$ fragments, one for every pixel of the input images. This is illustrated by the ICM algorithm presented in Table 4.5. In this example, the fragment shader is executed when the rectangle primitive is rendered (line 4). Here, the fragment shader minimizes in parallel the energy function $U(x_s, y_s)$ on each site $s \in S$.

```
1  Copy the input images y into texture memory.
2  Compile, link and load the ICM shader on the GPU.
3  Specify shader parameters (β_MD, β_ME or β_S for example).
4    Render a rectangle covering a window of size N × M
5    Copy the framebuffer into texture memory
6  Repeat line 4 and 5 until convergence
7  Copy the framebuffer into a C/C++ array if needed
```

```
1  x̂_s ← arg min_{x_s} U(x_s, y_s)
2  framebuffer_s ← x̂_s
```

TAB. 4.5 – *High level representation of an ICM hardware segmentation program. The upper section (line 1 to 7) is a C/C++ CPU program used to load the shader, render the scene and manage textures. The second program (line 1-2) is the ICM fragment shader launched on every fragment (pixel) when the scene is rendered (line 4). Notice that images x and y are contained into texture memory.*

The second problem comes from the fourth limitation. Since GPUs provide no general-purpose memory, one might wonder how the prior energy function V_{η_s} can be implemented on a fragment program since it depends on the neighboring labels x_t contained in the (write-only) framebuffer. As shown in Table 4.5, after rendering the scene, the CPU program copies the framebuffer into texture memory (line 5). In this way, the texture memory (which can be read by fragment shaders) contains not only the input images, but also the label field $x^{[t-1]}$ computed during the previous iteration. Thus, V_{η_s} is computed with labels iteratively updated and not sequentially updated as it is generally the case. Such strategy was already proposed by Besag

[71] and successfully tested by other authors [25]. This iterative updating scheme corresponds to the Jacobi-type version of the initial Gauss-Seidel ICM procedure. However, as mentioned by Besag [71], such parallel updating scheme can induce small oscillations. In fact, it turns out that indeed, on several examples, energy oscillations can occur after 10 or 20 iterations (see Fig. 4.8 (a)). These oscillations are caused by some labels (rarely more than few dozen, sparsely distributed in the label field) that endlessly switched between two classes. Although the Gauss-Seidel and the Jacobi versions of ICM don't strictly converge toward the same minima, the results they return are similar both visually and energy-wise (see Fig.4.7 and 4.8). Notice that the iterative nature of ICM is reproduced with multiple renderings of the rectangle (lines 4-5-6 of Table 4.5), the fragment shader containing no iteration loop.

The last problem with shaders comes with their inability to generate random numbers such as needed by the stochastic algorithms SA and ICE. We thus had to implement a workaround that goes as follows. First, in the C/C++ application, an image with random values is created. This random image is then copied in texture memory where the shader can access it. In this way, a stochastic shader processing a site (i, j) has access to an array of random numbers. This procedure is illustrated by the algorithms of Table 4.6 and 4.9. Please remark that the shaders are fed with a vector $\delta = (\delta_I, \delta_J)$ whose value is in-between $-\Delta$ and Δ. This is to make sure the same random value isn't reused at each iteration.

Although this workaround isn't as efficient as a good random number generator, the results obtained with our hardware programs are very close to the ones obtain with the software programs. To illustrate the effectiveness of our workaround, we have segmented a gray scale and a color image with a good software random number generator and with our hardware method. As can be seen in Fig. 4.7 and Fig.4.8 (b), our hardware method generates results very close to the ones obtained with the traditional software method, both visually and energy wise. Let us mention that this random value problem will surely be solved when hardware graphics companies will include a pseudo-random number generator to their shading language library.

But in the mean time, our workaround can be used will little lost of precision.

```
1   Rnd← Create an N × M image with random values.
2   Copy the images y and Rnd into texture memory.
3   Compile, link and load the SA shader on the GPU.
4   T ← T_MAX
5     δ_I, δ_J ← random integers between -5 and 5
6     Specify shader parameters (δ_I, δ_J, T and β_S for example)
7     Render a rectangle covering a window of size N × M
8     Copy the framebuffer into texture memory
9     T ← T*coolingRate
10  Repeat lines 5 to 9 until T < T_MIN
11  Copy the framebuffer into a C/C++ array if needed
```

```
1   P(x_s = e_k|y_s) ← (1/λ) exp{(-1/T) U(x_s = e_k, y_s)},  ∀e_k ∈ Γ
2   r ← Rnd_{s+δ}
3   According to r and P(x_s|y_s), randomly select e_k ∈ Γ
4   Framebuffer_s ← x̂_s
```

TAB. 4.6 – *High level representation of a SA hardware segmentation program. The upper section (line 1 to 11) is a C/C++ CPU program used to load the shader, render the scene and manage textures. The second program (line 1 to 4) is the fragment shader launched on every fragment (pixel) when the scene is rendered (line 7). Since shaders aren't equipped with a random number generator, random values are stored in the texture memory (Rnd).*

4.7.4 Energy-Based Segmentation on GPU

With the techniques described in the previous Section, performing motion detection, motion estimation and stereovision on a GPU is fairly straightforward. Since the shading languages (NVIDIA's *Cg* language in our case) have a syntax similar to C, the software programs could be almost directly reused in the shader. The implementation of the three fragment programs is conceptually very similar

since they all minimize an energy function made of a likelihood term and a prior term.

There is one exception however that occurs when a pre-filtering step is required by the stereovision application. To handle this situation, the cost function $C(s,x,y)$ is first precomputed and stored in a 3D DSI table located in texture memory. Then, the DSI table is filtered by "w" after which the optimization procedure (be it ICM, SA or WTA) is launched. In the context of a typical GPU, these operations can be done with one fragment shader made of three functions : one to compute the matching cost $C(s,x,y)$, one to filter $C(s,x,y)$ with w, and one to minimize the global energy function $U(x,y)$. More specifically, the first shader function has to compute, for each site $s \in S$, the cost value $C(s,x,y)$ associated to each disparity $d_s \in [0, D_{\text{MAX}}]$). The cost values are then copied in texture memory which stands as a DSI table. Immediately after, the second shader function filters the DSI table ($(w * C)(s,x,y)$) and copies the result in texture memory. At this point, the likelihood values $W_s(x_s,y_s), \forall s \in S$, $\forall x_s \in \Gamma$ are stored in texture memory. The third shader function is finally used to minimize the energy function $U(x_s,y_s)$ with ICM, SA or WTA. The overall stereovision *hardware* algorithm is presented in Table 4.7.

4.7.5 Probabilistic Segmentation on GPU

Optimizing Eq. (4.15) with SA or ICM is done the same way as for energy-based applications. Hardware algorithms of Table 4.5 and 4.6 can be directly reused, with the difference that shader parameters include the Gaussian mixture parameters Φ. The delicate aspect of probabilistic segmentation concerns more the parameter estimation procedures K-means and ICE which aren't perfectly suited to a mapping to the GPU. While the first step of these algorithms (assigning *the best* cluster for each image pixel for each site $s \in S$, line 2 of Table 4.2 and 4.3) is perfectly implementable in parallel, the second step (Gaussian parameters computation, line 3 of Table 4.2 and 4.3) is not. As such, we have to take an hybrid approach : execute line 2 on the GPU (parallel processing) and line 3 on the CPU (sequential

```
1    Copy input images y_MAT, y_REF in texture memory
2    Compile, Link and Load the stereovision shader
       on the GPU

     //**** Compute the DSI table ****//
3    Tell the shader to use the DSI function.
4    For d = 0 to D_MAX
5      Specify shader parameters (d)
6      Render a rectangle covering a window of size N x M.
7      Copy the framebuffer into texture memory
8      d <- d + 4

     //**** Apply filter w ****//
9    Tell the shader to use the filter function.
10   For d = 0 to D_MAX
11     Specify shader parameters (d and filter parameters)
12     Render a rectangle covering a window of size N x M.
13     Copy the framebuffer into texture memory
14     d <- d + 4

     //**** Launch ICM to minimize U(x,y) ****//
15   Tell the shader to use the ICM function.
16   Specify shader parameters (D_MAX,beta_S)
17     Render a rectangle covering a window of size N x M.
18     Copy the framebuffer into texture memory
19   Repeat line 17 and 18 until convergence
20   Copy the framebuffer into a C/C++ array if needed
```

TAB. 4.7 – *Stereovision program using three shader functions to compute and filter the cost function and segment the scene. From line* 3 *to* 8*, matching cost function* $C(s,x,y)$ *is computed and stored in texture memory which stands as a DSI table. Then, from line* 9 *to* 14*, the cost function is filtered* $(w * C)$ *before the label field is estimated with ICM.*

1	Copy the input images y into texture memory.
2	Compile, link and load the K-**means** shader on the GPU.
3	$\Phi \leftarrow$ Init Gaussian parameters.
4	Specify shader parameters (N and Φ_μ).
5	Render a rectangle covering a window of size $\mathcal{N} \times \mathcal{M}$
6	$E \leftarrow$ Copy the framebuffer into a C/C++ array.
7	$\mu_k \leftarrow \frac{1}{N_k} \sum_{E_s=e_k} y_s, \quad \forall \mu_k \in \Phi_\mu$
8	Repeat lines 4 to 7 until convergence.
9	$\Sigma_k^{nm} \leftarrow \sum_{E_s=e_k} (y_s^n - \mu_k^n)(y_s^m - \mu_k^m), \quad \forall \Sigma_k \in \Phi_\Sigma$
10	Copy the framebuffer into a C/C++ array if needed

1	$x_s \leftarrow \arg\min_{e_k} \|\|y_s - \mu_{e_k}\|\|^2$
2	$\text{framebuffer}_s \leftarrow x_s$

TAB. 4.8 – *High level representation of a K-means program. The upper section (line 1 to 10) is the C/C++ CPU program used to load the shader, render the scene and compute the Gaussian parameters Φ. The second program (line 1-2) is the fragment shader launched on every fragment (pixel) when the scene is rendered (line 5).*

processing).

At first, the input images y are put in texture memory so it is accessible by the fragment shaders. Each site $s \in S$ are then assigned *the best* class e_k by the fragment shader, before the Gaussian parameters of every class are re-estimated. Because this last operation is global and thus can't be parallelized, the framebuffer containing the class of each pixel is read back to the CPU memory, where the computation takes place. Once the parameters are re-estimated, they are passed back to the GPU after which a new iteration can begin. The implementation of K-means and ICE is illustrated in Table 4.8 and 4.9.

1	Rnd$\leftarrow$ Create an $\mathcal{N} \times \mathcal{M}$ image with random values.
2	Copy the input images y and Rnd into texture memory.
3	Compile, link and load the **ICE** shader on the GPU.
4	$\Phi \leftarrow$ Init Gaussian parameters.
5	$\delta_I, \delta_J \leftarrow$ random integers between -5 and 5.
6	Specify shader parameters (δ_I, δ_J, N and Φ).
7	Render a rectangle covering a window of size $\mathcal{N} \times \mathcal{M}$
8	$E \leftarrow$ Copy the framebuffer into a C/C++ array.
9	Copy the framebuffer into texture memory
10	$\mu_k \leftarrow \frac{1}{N_k} \sum_{E_s=e_k} y_s, \quad \forall \mu_k \in \Phi_\mu$
11	$\Sigma_k^{nm} \leftarrow \sum_{E_s=e_k} (y_s^n - \mu_k^n)(y_s^m - \mu_k^m), \quad \forall \Sigma_k \in \Phi_\Sigma$
12	Repeat lines 5 to 11 until convergence.
13	Copy the framebuffer into a C/C++ array if needed

1	$P(e_k \mid y_s) = \frac{1}{Z_s} \exp\{(W(e_k, y_s) + V_{\eta_s})\}, \forall e_k \in \Gamma$
2	$r \leftarrow \text{Rnd}_{s+\delta}$
3	$x_s \leftarrow$ according to r and $P(x_s \mid y_s)$, randomly select $e_k \in \Gamma$

TAB. 4.9 – *High level representation of an ICE program. The first section (line* 1 *to* 13*) is the C/C++ CPU program used to load the shader, render the scene and compute the Gaussian parameters* Φ*. The second program (line* 1 *to* 3*) is the fragment shader launched on every fragment (pixel) when the scene is rendered (line* 7*).*

4.8 Experimental Results

Results compare software and hardware implementations of the energy-based and probabilistic applications we have discussed so far. The goal being to show how fast a segmentation program implemented on a GPU is compared to its implementation on a CPU. The software programs were implemented in C++[1] and the NVIDIA Cg language was used to implement the fragment programs. Because

[1]C and C++ are by far the most utilized languages for hardware programming. Empirical tests have shown that the two languages provides similar processing times.

Cg's syntax is very much similar to C and C++, the C++ code of our software applications were partly reused to implement the fragment shaders. As such, the differences between the software and hardware implementations were kept at a strict minimum and thus, could be fairly compared.

Every results were made after varying some variables. In Figs. 4.9 to 4.13, the lattice size vary between 64×64 and 1024×1024 and the number of classes (or disparities) between 4 and 32, depending on the application. The number of iterations was set to 10 for ICM, K-Means and ICE, and to 500 for SA. Thus, because the number of iterations is fixed for each algorithm, the processing times are independent of the image content.

Notice that for ICE and SA, Δ was set to 5. The reason for this choice is mostly technical. As shown in Fig. 4.8 (c), using a value larger than 5 doesn't minimize much more the global energy. Also, because a value larger than 5 adds a latency to the algorithm, we conclude that 5 is a good compromise between speed and accuracy.

Every results are expressed as an acceleration factor between the software and hardware programs. However, the results do not include the time needed to load, compile and link the shaders which can vary between 0.05 second and 5 seconds. Although this might seem prohibitive, this initialization step is done only once at the beginning of the application. In this way, when segmenting more than one scene (or segmenting a scene with a lattice size larger than 128×128), this initialization time soon gets negligible as compared to the acceleration factor. This is especially true when simulated annealing is used as optimization procedure.

All programs were executed on a conventional home PC equipped with a AMD Athlon 2.5 GHz, 2.0 Gig of RAM and a NVIDIA 6800 GT graphics card. NVIDIA fp40 Cg profile was used in addition to the cgc compiler version 1.3.

4.8.1 Energy-based segmentation

Figs. 4.9, 4.10 and 4.11 contain the acceleration factor for the three energy-based applications. In Figs. 4.9 and 4.10 are the results for SA and ICM over grayscale

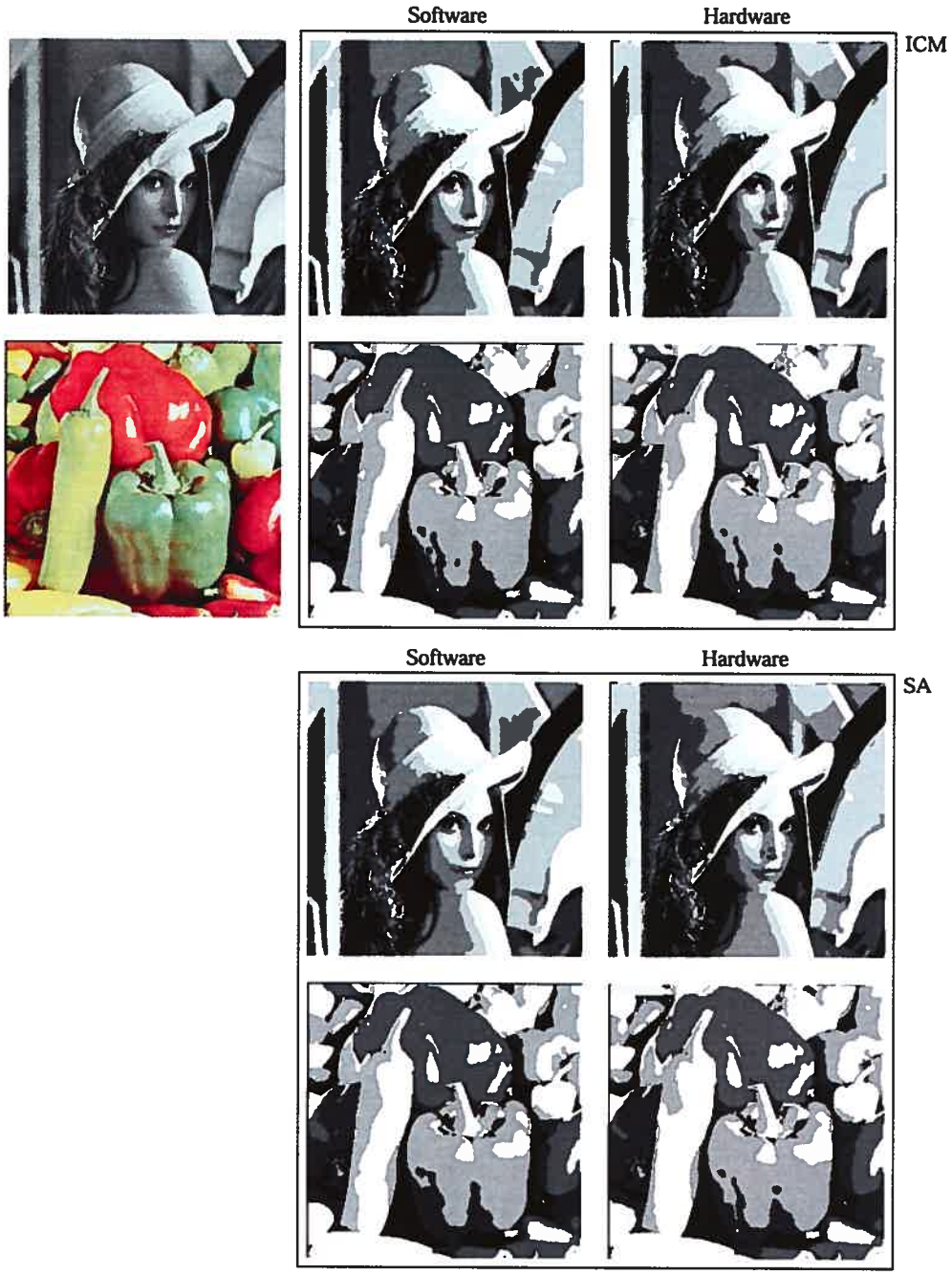

FIG. 4.7 – *Gray scale and color image segmented with the software (left column) and hardware (right column) version of ICM (top four images) and SA (four lowest images). In every cases, the Gaussian parameters have been estimated with K-means.*

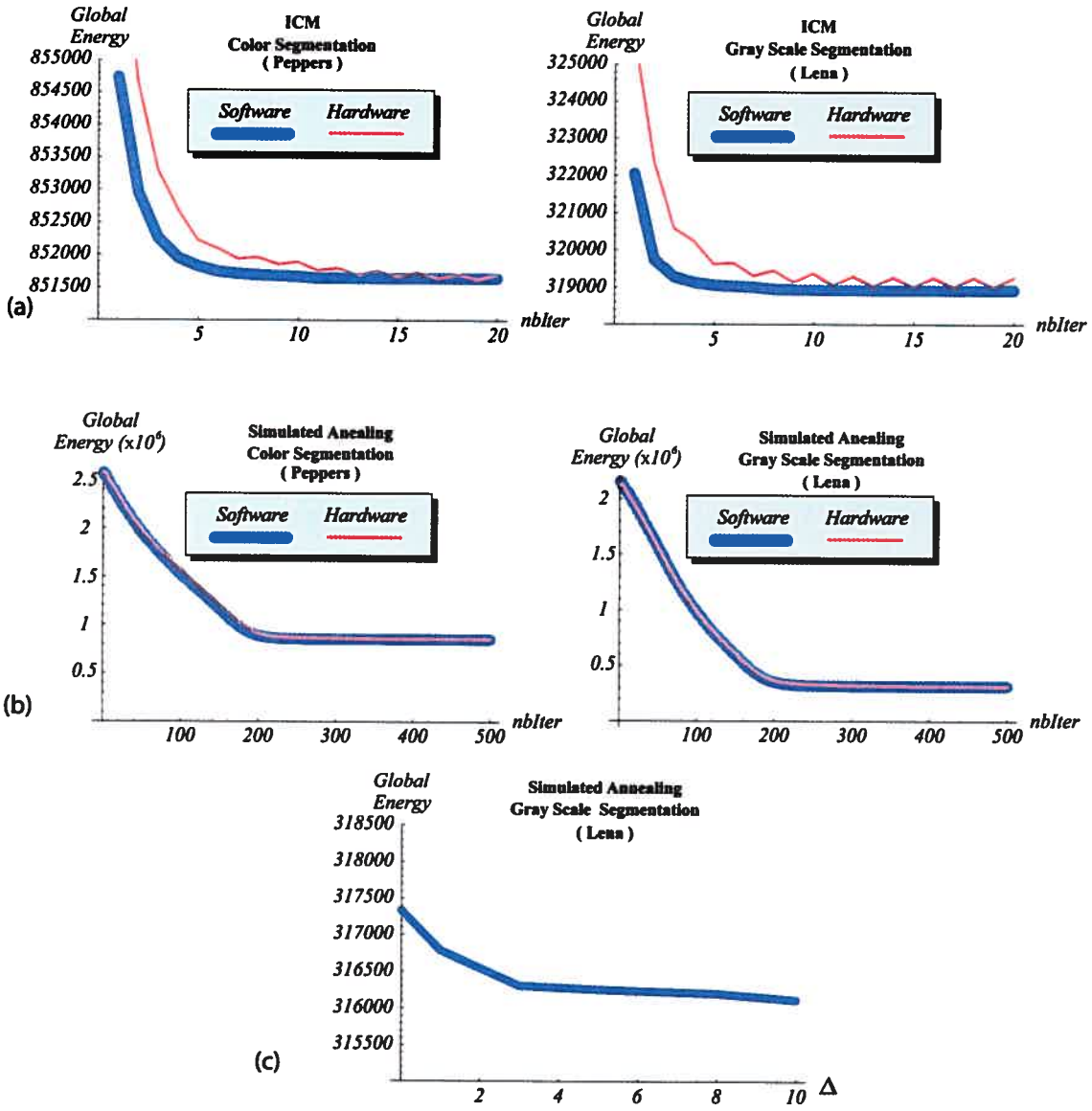

FIG. 4.8 – *Plot of Eq.(4.15) when segmenting a color or a gray scale image (see Fig.4.7) with ICM (first two graphs) or with SA (two middle graphs). The fifth graph shows the influence of variable* Δ *on the global energy after* 500 *SA iterations.*

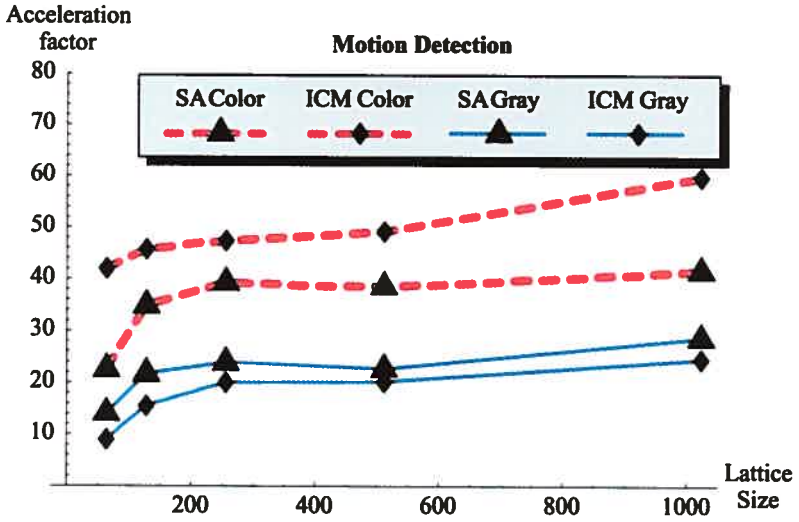

FIG. 4.9 – *Acceleration factor for motion detection programs over square image sequences.*

and color sequences. The way SA and ICM are initialized vary from one application to another. For motion detection, a label field obtained after a trivial thresholding operation is used to initialize the two optimizers. As for motion estimation, the label field is simply initialize to zero whereas, for stereovision, ICM and SA are fed with the disparity map generated by a simple WTA.

In every case, the hardware implementation is faster than its software counterpart by a factor between 10 and 60. Notice that the acceleration factor is more important for color sequences than for grayscale sequences. This is explained by the fact that the likelihood energy function W of the motion estimation and motion detection programs is more expensive to compute with RGB values than with grayscale values. Thus, distributing this extra load on a fragment processor results in a more appreciable acceleration factor. In Fig. 4.10, d_{MAX} was set to 4 in the first graphic and the lattice size was set to 256×256 in the second graphic.

For stereovision (Fig. 4.11), we have tested our programs for the three tasks presented in Tab. 4.7 namely the computation of the DSI table, the aggregation filtering, and the optimization procedure (SA, ICM and Winner-Take-All (WTA)).

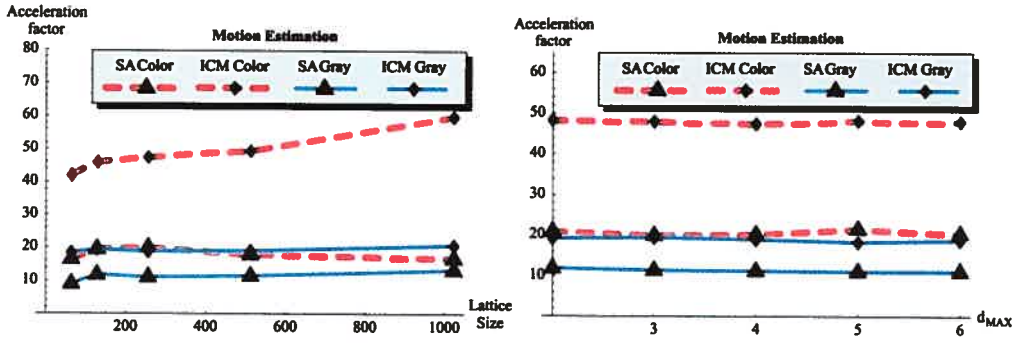

FIG. 4.10 – *Acceleration factor for the motion estimation programs.*

The three optimization procedures have been tested on scenes of various size and with different number of disparities. In the leftmost graphics, D_{MAX} was set to 16 and the lattice size was set to 256×256 in the other graphics. As can be seen, the acceleration factor for ICM and SA is more important than the one for WTA. This can be explained by the fact that WTA is a trivial and efficient algorithm (it converges in only one iteration) with a less impressive amount of computation to distribute on the GPU than for ICM and SA.

As for the task of computing the DSI table, we have compared our hardware and software implementations over color and grayscale input images. Again, since the likelihood cost function $C(s, x, y)$ is more expensive to compute with RGB values than with grayscale values, the acceleration factor for the color DSI is more important than the one for the gray scale DSI.

4.8.2 Statistical segmentation

The statistical applications presented in Section 4.5 and 4.6 have been also implemented in C++ and in Cg. The performances of each implementation was evaluated by varying the number of segmentation classes and the size of the images to be segmented. The acceleration factor between the software and hardware version of the programs is presented in Fig. 4.12. In both cases, ICM and SA are initialized by the label field returned by ICE.

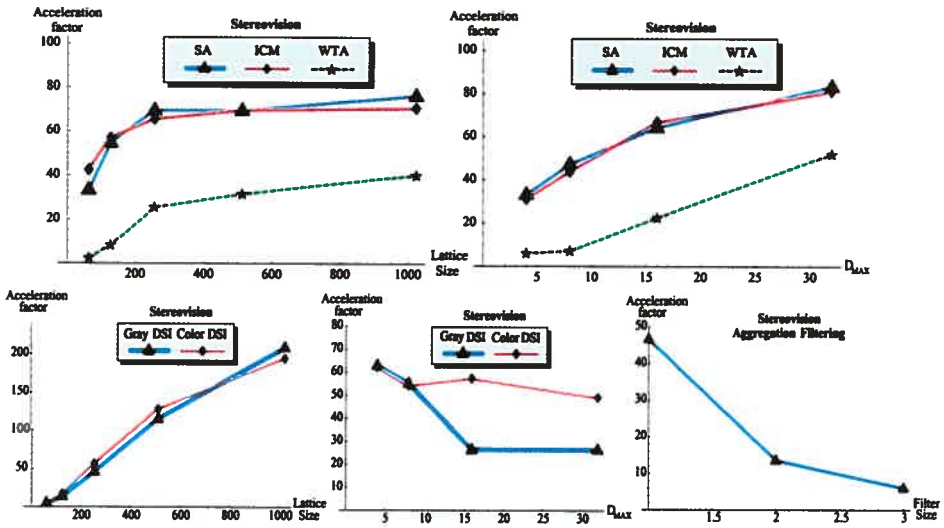

FIG. 4.11 – *Acceleration factor for the stereovision programs.*

Notice that the speedup factor between hardware and software version of ICM and SA (between 20 and 120) is more important than the one for K-means and ICE (between 2 and 8). The reason for this is that K-means and ICE have to exchange information (for the Gaussian parameter estimation) with the CPU which is a major bottleneck for such hardware programs. Hence why the parameter estimation programs seem less efficient than ICM and SA.

Also, as can be seen, the speedup factor for K-means is larger than for ICE. This is explained by the fact that ICE has to estimate (and invert) the variance-covariance matrix at each iteration which isn't required for K-means. This extra load on the CPU makes ICE less efficient than K-means.

As is the case for most energy-based applications, the speedup factor for SA and ICM is more important on color images than on grayscale images. Again, this is explained by the fact that the energy function of Eq. (4.15) is more expensive to compute for color images than for grayscale images. Thus, parallelizing this costly CPU operation leads to a more important acceleration factor. Notice that the acceleration factor is larger when segmenting large images and/or segmenting

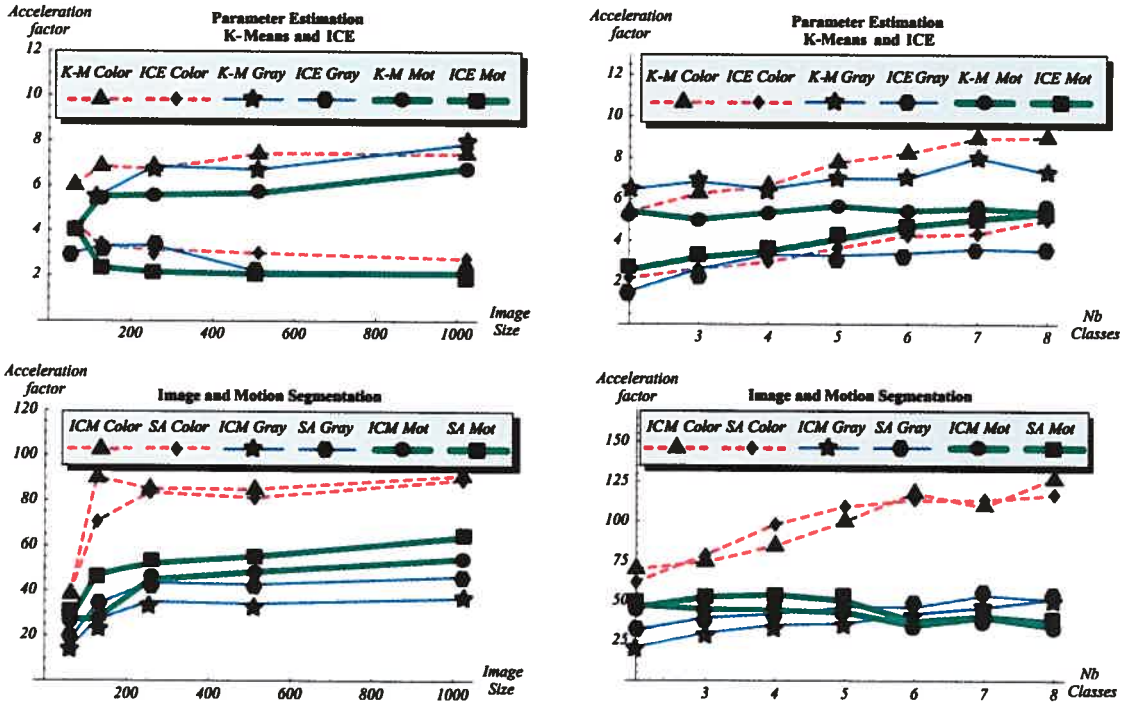

FIG. 4.12 – *Acceleration factor for K-means, ICE, SA and ICM obtained on grayscale/color images and on vector fields (motion segmentation).*

images with many classes.

With our actual hardware implementation, a color image of size 128 × 128 is segmented in 4 classes at a rate of 76 fps with ICM, 1.4 fps with SA, 2.5 fps with ICE and 14 fps with K-means. Although K-means and ICE estimate parameters at an interactive rate, they can be seen as slow procedures, at least compared to ICM. Thus, to save on processing time, when segmenting an image sequence with frames having mostly the same color distribution, the Gaussian parameters estimated on the first frame can be reused for the rest of the sequence. As an example, Fig. 4.13 shows an image sequence of size 352 × 240 segmented in 6 classes. At first, the fragment shader is loaded, compiled and linked (approximately 1.5 second). The Gaussian parameters are then estimated on the first frame with K-means and ICE (approximately 2 seconds). Assuming the color distribution of

every frame is similar, the Gaussian parameters are reused to segment the rest of the sequence. Segmenting the 30 frames with our hardware implementation of ICM took approximately 1.5 second for the entire sequence, i.e. an average of 0.05 second per frame. This represents a segmentation rate of 20 frames per second. Notice how little the difference is between the segmentation map of the last frame (Fig. 4.13 (e)) inferred with the Gaussian parameters initially computed and the one obtained with the Gaussian parameters estimated on the last frame (Fig. 4.13 (f)).

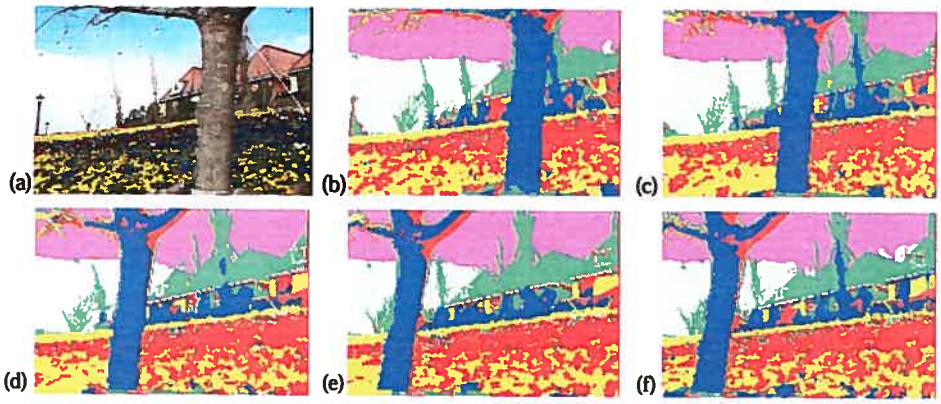

FIG. 4.13 – *Image sequence of size* 352×240 *segmented in 6 classes. The Gaussian parameters estimated on the first frame (a) are used to segment the entire sequence (b)-(e). (f) Last frame segmented with newly estimated Gaussian parameters.*

4.9 Conclusion

This paper exposes how programmable graphics hardware can be used to performed typical Markovian segmentation applied to energy-based and statistical problems. Results show that the parallel abilities of GPUs significantly accelerate these applications (by a factor of 4 to 200) without requiring any advanced skills in hardware programming. Such hardware implementation is useful especially when the image size is large, when the number of labels is large or when the observation field y is made of color images. Notice that a multiresolution version of every pro-

gram could be implemented on GPU, at the expense though of a more elaborated setup.

Acknowledgments

The authors would like to thank Jean-François St-Amour for his precious help with hardware programming.

CONCLUSION GÉNÉRALE ET PERSPECTIVES D'AVENIR

Au cours de cette thèse, nous avons exploré plusieurs contraintes spatiales bas niveau que nous avons utilisé pour résoudre des problèmes classiques de vision par ordinateur. Comme nous l'avons mentionné dans l'introduction, ces problèmes d'imagerie trouvent leurs solutions au coeur de la théorie de l'apprentissage statistique. En fait, ces problèmes peuvent se formaliser via la minimisation d'une fonction de risque ou comme l'espérance mathématique d'une fonction de perte. La minimisation d'une fonction de risque rentre dans un cadre très générale qui inclut plusieurs problèmes fondamentaux en apprentissage statistique. Ainsi, dépendant de la nature des données ainsi que de la fonction de perte retenue, les problèmes de vision que nous avons étudié se réduisent à l'un ou l'autre des problèmes suivants :

1. la reconnaissance de formes ;
2. la régression ;
3. l'estimation de densité.

Parmi les applications auxquelles nous nous sommes attardées figurent le flux optique, la détection de mouvement, la segmentation de mouvement ainsi que la détection d'occlusions.

Le flux optique est un concept mathématique utilisé pour décrire le mouvement apparent des objets présents dans une séquence vidéo. La plupart des méthodes d'estimation du flux optique se fondent sur une hypothèse simplificatrice qui stipule qu'un pixel au temps t possède la même couleur que sa projection au temps $t + 1$ (hypothèse de la constance de l'illumination). Malgré le fait que cette hypothèse soit souvent fausse (surtout pour les séquences vidéo bruitées et dans les régions occluent) elle permet d'établir une équation fort utile, l'équation générale de mouvement (voir équation (30)). Malheureusement, trouver le *meilleur* vecteur $\vec{v}_s$ pour chaque pixel en vertu de l'équation du mouvement est un problème inverse mal posé. Parmi les solutions pour rendre cette équation bien posée figure celle proposée par Lucas et Kanade. Mathématiquement, leur approche cherche

à minimiser une fonction d'énergie quadratique à l'aide d'une régression linéaire multivariée. La solution à leur méthode peut donc s'obtenir à l'aide d'une simple inversion de matrice, $\vec{v}_s = -M_s^{-1} b_s$. Malgré ses avantages, cette équation souffre de deux limitations majeures. Premièrement, la matrice M_s peut devenir singulière dans les régions peu texturées. Deuxièmement, puisque Lucas-Kanade implémente une régression par la somme des moindres carrés, le mouvement à l'intérieur de η_s ne doit contenir aucune discontinuité. Ainsi, lorsque η_s couvre une discontinuité de mouvement (proche du contour d'un objet en mouvement par exemple) le champ vectoriel résultant est flou.

En réponse à ces deux problèmes, nous avons proposé au chapitre 1 deux contraintes à bas niveau. La première est un filtre passe bas de type *Best Linear Unbiased Estimate* ayant pour objet de régulariser le champ vectoriel dans les régions peu texturées. La seconde contrainte est une procédure de déplacement local du voisinage de calculs η_s. À l'aide de l'algorithme mean-shift, les voisinages η_s sont localement déplacés vers des régions dont le mouvement est plus susceptible d'être unimodal que multimodal. Les résultats obtenus démontrent que notre méthode fonctionne aussi bien sur des séquences locales que globales. Pour les séquences locales, grâce au mean-shift, le champ vectoriel estimé à proximité du contour d'objets en mouvement est mieux conservé qu'avec les autres méthodes implémentées. Nous avons d'ailleurs souligné cette propriété en mesurant l'erreur angulaire moyenne près du contours d'objets en mouvement dans des scènes synthétiques avec vérité-terrain. Pour ce qui est des séquences globales, grâce à la procédure de filtrage, les résultats sont plus réguliers que ceux obtenus à l'aide d'autres méthodes implémentées. À nouveau, cette propriété a été vérifiée à l'aide d'une mesure d'erreur angulaire moyenne.

Une autre application sur laquelle nous nous sommes attardée est la détection de mouvement. L'objectif de cette application est d'estimer un champ d'étiquettes X dont chaque élément indique si un site est *mobile* ou *immobile*. Une des configurations les plus couramment utilisées pour estimer X est basée sur la soustraction de fond. Pour cette configuration, la caméra est fixe et l'image du fond est station-

naire. Suivant cette configuration, X peut être estimé en appliquant un simple seuil sur la différence entre l'image au temps t, I_t et l'image du fond, B. Pour rendre la détection plus robuste, plusieurs auteurs modélisent chaque pixel à l'aide d'une densité de probabilité (unimodale ou multimodale dépendant de l'application). La densité de probabilité de chaque pixel est apprise sur la base d'une séquence vidéo comprenant N images absentes de tout mouvement. La détection de mouvement probabiliste est donc, pour l'essentiel, un problème d'estimation de densité. Bien que cette façon d'estimer les densités soit largement répandue aujourd'hui, elle possède toutefois des limites. Ces limites se manifestent lorsqu'aucune image absente de mouvement n'est disponible ou lorsque l'espace mémoire disponible est insuffisant pour stocker les images nécessaires à l'apprentissage. Au chapitre 3, nous avons proposé deux nouvelles méthodes pour effectuer une soustraction de fond probabiliste. L'aspect novateur de ces deux méthodes réside dans le fait qu'elles permettent d'entraîner des modèles statistiques sur une seule image, au lieu d'une série d'images comme c'est le cas habituellement. Notre première méthode (que nous appelons *robuste*) est faite pour gérer des séquences dont le fond n'est pas stationnaire. Pour y arriver, cette méthode modélise chaque pixel à l'aide de deux densités de probabilité : une densité unimodale et une densité multimodale. Pour ce qui est de notre seconde méthode (appelée *légère et rapide*) elle a pour objet de minimiser les temps de calcul et l'espace mémoire requis pour effectuer la détection. Pour ce faire, l'image du fond est modélisée par un mélange global de $\mathcal{M}$ gaussiennes. Les résultats obtenus démontrent que nos deux méthodes *spatiales* génèrent des résultats aussi précis que les méthodes *temporelles* classiques. Nous avons démontré à l'aide de courbes R.O.C., que notre méthode robuste arrive à bien compenser pour différentes instabilités pouvant être observées dans le fond. Ces instabilités incluent celles causées par une caméra qui vibre, un fort niveau de bruit ou une texture animée. Les résultats démontrent aussi que notre méthode légère et rapide (*light and fast*) requière sensiblement moins de calculs et d'espace mémoire que les autres méthodes que nous avons implémenté. Nous avons aussi démontré qu'avec nos deux méthodes, il est facile d'effectuer une soustraction de

fond probabiliste lorsqu'aucune image absente de mouvement est disponible pour l'entraînement.

Au chapitre 2, nous avons proposé une nouvelle approche de fusion de données appliquée à l'imagerie. Cette méthode fusionne des champs d'étiquettes au lieu de fusionner des données brutes comme le font traditionnellement les algorithmes de segmentation. Tel que mentionné au chapitre 2, la fusion de champs d'étiquettes possède plusieurs avantages. Tout d'abord, puisqu'elle fusionne des champs d'étiquettes et non des données brutes (des *features* en anglais) on réduit considérablement les problèmes liés à la dimensionnalité du problème. Aussi, notre approche est conceptuellement très simple. Elle peut être implémentée en moins de 30 lignes de code C/C++, ne dépend que d'un seul paramètre et peut facilement être implémentée sur une architecture parallèle. De plus, notre approche est plus robuste aux imprécisions que d'autres algorithmes comparables de segmentation à base de régions. Pour fonctionner, notre méthode prend en entrée deux champs d'étiquettes : un champ d'application $x^{[0]}$ et une carte de régions r. Ces champs d'étiquettes sont inférés sur la base de différentes données (*features*) tirées d'images d'entrée. De cette façon, leur contenu est complémentaire. Ainsi, le contenu du champ d'application $x^{[0]}$ contient une version grossière du champs d'étiquettes qu'on cherche à estimer. Quant à la carte de régions r, elle contient la forme des principaux objets présents dans la scène. Ainsi, la fusion de ces deux champs d'étiquettes permet d'inférer un nouveau champ d'étiquettes $x^{[k]}$ proche de $x^{[0]}$, mais dont la forme des objets épouse la géométrie des régions présentes dans r. La fusion se fait en minimisant une fonction d'énergie à l'aide de l'algorithme d'optimisation déterministe ICM. Pour ce faire, deux fonctions d'énergie ont été proposées. La première est une fonction de fusion *pure* alors que la seconde est une fonction de fusion de type *fusion-réaction*. Pour cette dernière, un terme de réaction a été ajouté afin que $x^{[k]}$, le champ d'application résultant, ne diffère pas trop de $x^{[0]}$, le champ d'application en entrée. Nous avons montré que les applications de détection d'occlusions et de segmentation de mouvement pouvaient clairement bénéficier de notre méthode. Les résultats démontrent que notre méthode

de fusion est robuste et réagit graduellement aux variations de ses paramètres. De plus, contrairement à certaines approches concurrentes, notre méthode réagit bien aux imprécisions présentes dans la carte de régions r et peut fonctionner en temps réel lorsqu'implémentée sur une architecture parallèle comme celle des GPUs.

Plusieurs algorithmes d'apprentissage statistique appliqués à l'imagerie sont reconnus pour exiger des masses de calculs souvent prohibitifs pour des applications temps-réel. C'est entre autre le cas des algorithmes markoviens de segmentation et d'estimation de paramètres. En réponse à ce problème, nous avons exploré au chapitre 4, la possibilité d'implémenter certains de ces algorithmes sur une architecture parallèle, à savoir les GPU (Graphics Processor Unit). Les GPU sont des processeurs graphiques disponibles sur la plupart des carte graphiques aujourd'hui en vente sur le marché. Bien que les GPUs aient été développés pour effectuer des tâches liées au rendu d'images, ils possèdent deux caractéristiques fondamentales les rendant intéressants pour nous. Tout d'abord, les GPU possèdent un processeur *fragment* ayant la capacité de traiter en parallèle tous les pixels d'une scène. C'est entre autre cette capacité qui permet aux GPUs de réduire considérablement les temps de calculs. Deuxièmement, les GPUs permettent de charger, compiler et exécuter du code produit par un programmeur. De cette façon, les GPUs permettent d'effectuer des opérations allant bien au-delà des calculs de synthèse d'images. Nous avons montré que les GPUs peuvent être utilisés pour résoudre différents problèmes de segmentation d'images, d'estimation de paramètres, d'estimation de mouvement, de segmentation de mouvement ainsi que de stéréovision.

Les résultats obtenus démontrent qu'une implémentation sur GPU permet d'atteindre des taux d'accélération allant de 4 à 200. Plus les images d'entrée sont grosses, plus le nombre de données à traiter par pixel est élevé et plus le nombre de classes (ou de disparités) est élevé, plus les taux d'accélération seront élevés. Nous avons aussi démontré que la précision des résultats obtenus sur GPU sont à toutes fins pratiques identiques à ceux obtenus sur CPU. Cela est vrai tant qualitativement que quantitativement au niveau de la mesure d'énergie globale. Nous avons aussi souligné que les algorithmes d'estimation de paramètres K-Moyennes

et ICE présentent des taux d'accélération inférieurs car ce sont les seuls à exiger des calculs simultanés sur CPU et sur GPU.

Perspectives d'avenir

Nous avons exploré au cours de cette thèse des contraintes à bas niveau appliquées à certains problèmes en vision par ordinateur. Au cours des prochains mois, nous prévoyons appliquer ce même type de contraintes à d'autres applications en imagerie numérique. Parmi ces applications, les plus prometteuses sont, selon nous, l'estimation du flux optique 3D, la segmentation de profondeur et la fusion de données tirées d'un réseau de caméras.

Flux optique 3D

Les méthodes d'estimation de flux optique présentées au chapitre 1 fonctionnent dans la perspective d'estimer un champ vectoriel 2D. Bien que ce paradigme bidimensionnel soit satisfaisant pour une majorité d'applications, il n'en demeure pas moins que certaines applications exigent un autre type de flux. Par exemple, certaines applications exigent l'estimation d'un flux optique 3D et non 2D. Pour ces applications, au lieu d'avoir en entrée une séquence vidéo faite d'images bidimensionnelles se succédant dans le temps, on dispose d'une séquence de volumes tridimensionnels se succédant dans le temps. Ainsi l'élément de base n'est plus le pixel 2D de coordonnée (i, j), mais le voxel 3D de coordonnée (i, j, k). Le domaine ayant recours le plus souvent au flux optique 3D, est sans nul doute l'imagerie médicale [59,68,98,151]. Estimer le flux optique dans un contexte d'imagerie médicale présuppose toutefois certaines contraintes. Tout d'abord, bien que les données médicales peuvent être obtenues à l'aide de différentes technologies ayant leur caractéristiques propres, il n'en demeure pas moins que ces données sont souvent bruitées et difficiles à utiliser sans post- traitement. Aussi, dans certains cas (comme l'estimation du champ de déplacement d'un flux sanguin par exemple) le mouvement est fortement localisé à l'intérieur d'un organe. Dans ce cas, le champ vectoriel estimé doit présenter de fortes discontinuités à certains endroits, chose

difficilement réalisable avec les algorithmes de base. Par conséquent, nous croyons que les deux contraintes présentées au Chapitre 1 peuvent s'appliquer au contexte du flux optique 3D en imagerie médicale. Par exemple, la contrainte liée au filtrage passe bas *BLUE* pourrait certainement aider à contrer les effets du bruit. Nous croyons aussi que la contrainte d'évitement de contours basée sur l'algorithme mean-shift pourrait contribuer à préserver les discontinuités de mouvement. L'extension 3D de notre méthode pourrait donc générer des résultats intéressants sur des images tirées de PET-scans (*Positron Emission Tomography*) de CT-scans (*Computed axial Tomography*) et de MRI-scans (*Magnetic Resonance Imaging*). Nous croyons aussi que notre méthode pourrait facilement s'adapter aux images dites *in-vivo* retournées par certains scanners MRI. Ces images contiennent des informations hautement bruitées de magnitude et de phase relative au mouvement à l'intérieur d'un organe.

Segmentation de profondeur

La stéréovision a pour but d'estimer la profondeur des objets situés dans une scène photographiée par deux caméras (ou plus). Comme nous l'avons mentionné au chapitre 2, le résultat retourné par un algorithme de stéréovision s'exprime souvent sous la forme d'une carte de disparités d. Une fois estimée, il est possible de segmenter cette carte afin d'en extraire des régions de disparité uniforme. C'est ce qu'on appelle une opération de segmentation de profondeur [34,137]. Pour ce faire, les différents algorithmes de segmentation de mouvement présentés aux chapitres 2 et 4 pourraient facilement être adaptés au contexte de la segmentation de profondeur. Nous croyons toutefois que l'approche la plus prometteuse est de réutiliser l'équation (48) avec, comme fonction d'énergie de vraisemblance, $E(d(s), x(s))\phi(||d(s) - \mathcal{X}_s \vec{A}_{x(s)}||)$. Il est à noter que Wren et Ivanov [34] ont proposé une approche similaire quoique fort simplifiée. Une fois le champ d'étiquettes x obtenu à l'aide d'un algorithme d'optimisation comme le recuit simulé, on pourrait fusionner x avec une carte de régions r afin d'éliminer les valeurs isolées et forcer les formes présentes dans x à épouser les régions de r. Il est fort à parier que la

précision des résultats s'en verrait grandement améliorée, tout comme ce fut le cas pour les différentes applications présentées au chapitre 2.

Fusion de multiple caméras

Parmi les projets de recherche sur lesquels nous prévoyons travailler au cours des prochains mois, il en est un portant sur le traitement d'images tirées de multiples caméras. Alors que la plupart des méthodes en imagerie traitent d'images (ou de séquences vidéo) tirées d'une seule caméra, l'idée ici est de traiter plusieurs images tirées d'autant de caméras. Ainsi, nous voulons voir comment peut s'améliorer la précision de certains résultats lorsque des données issues de plusieurs caméras localisées à différentes positions sont fusionnées ensemble. Nous pensons que notre algorithme de fusion de données présenté au chapitre 2 pourrait être facilement adapté à ce contexte. Dans ce cas, il nous faudrait ajuster notre approche pour fusionner un nombre arbitrairement grand de champs d'étiquettes. Nous croyons que la fusion de plusieurs champs d'étiquettes obtenus en segmentant différentes images sur la base de différents *features* pourrait permettre de lever certaines ambiguïtés et ainsi augmenter la précision des résultats. Ainsi, cette approche pourrait donner des résultats intéressants pour des applications en détection de mouvement, en estimation de mouvement ainsi qu'en segmentation de mouvement.

BIBLIOGRAPHIE

[1] www.middlebury.edu/stereo.

[2] http ://www.gpgpu.org/, 2006.

[3] Bab-Hadiashar A. and Suter D. Robust optic flow computation. *Int. J. Comput. Vision*, 29(1) :59–77, 1998.

[4] Bendjebbour A., Delignon Y., Fouque L., Samson V., and Pieczynski W. Multisensor image segmentation using dempster-shafer fusion in markov fields context. *IEEE Trans. on Geo. and Rem. Sens.*, 39(8) :1789–1798, 2001.

[5] Bovik A., editor. *Handbook of Image and Video Processing, 2nd Edition.* Academic Press, 2005.

[6] Bruhn A., Weickert J., and Schnörr. Lucas/kanade meets horn/schunck : combining local and global optic flow methods. *Int. J. Comput. Vision*, 61(3) :211–231, 2005.

[7] Elgammal A., Duraiswami R., Harwood D., and Davis L.S. Background and foreground modeling using nonparametric kernel density for visual surveillance. In *Proc of the IEEE*, volume 90, pages 1151–1163, 2002.

[8] Filippidis A., Jain L.C., and Martin N. Using genetic algorithms and neural networks for surface land mine detection. *IEEE Trans. on Sig. Process.*, 47(1) :176–186, 1999.

[9] Jain A. and Farrokhnia F. Unsupervised texture segmentation using gabor filters. *Pattern Recogn.*, 24(12) :1167–1186, 1991.

[10] Kushki A., Androutsos P., Plataniotis K.N., and Venetsanopoulos A.N. Retrieval of images from artistic repositories using a decision fusion framework. *IEEE Trans. on Image Processing*, 13(3) :277–292, 2004.

[11] Luo A. and Burkhardt H. An intensity-based cooperative bidirectional stereo matching with simultaneous detection of discontinuities and occlusions. *Int. J. Comput. Vision*, 15(3) :171–188, 1995.

[12] McIvor A. Background subtraction techniques. In *Proc. of Image and Video Computing*, 2000.

[13] Mitiche A. and Bouthemy P. Computation and analysis of image motion : a synopsis of current problems and methods. *Int. J. Comput. Vision*, 19(1) :29–55, 1996.

[14] Mittal A. and Huttenlocher D. Scene modeling for wide area surveillance and image synthesis. In *Proc. of CVPR*, pages 160–167, 2000.

[15] Mittal A. and Paragios N. Motion-based background subtraction using adaptive kernel density estimation. In *Proc. of CVPR*, pages 302–309, 2004.

[16] Neri A., Colonnese S., Russo G., and Talone P. Automatic moving object and background separation. *Signal Processing*, 66(2) :219–232, April 1998.

[17] Singh A. An estimation-theoretic framework for image-flow computation. In *Proc. of ICCV*, pages 168–177, 1990.

[18] Singh A. Incremental estimation of image flow using a kalman filter. *J. Vis. Commun. Image Represent.*, 3 :39–57, 1992.

[19] Solberg A. and Jain A. Texture fusion and feature selection applied to SAR imagery. *IEEE Trans. Geosci. Remote Sens.*, 35(2) :475–479, 1997.

[20] Tomas Akenine-Möller and Eric Haines. *Real-time Rendering.* AK Peters, 2e edition, 2002.

[21] Horn B. and Schunck B. Determining optical flow. *Artificial Intelligence*, 17 :185–203, 1981.

[22] Lucas B. and Kanade T. An iterative image registration technique with an application to stereo vision (darpa). In *Proc. of DARPA Image Unders. Workshop*, pages 121–130, 1981.

[23] Bishop C. *Neural Networks for Pattern Recognition.* Oxford University Press, 1996.

[24] Chang C., Chatterjee S., and Kube P.R. On an analysis of static occlusion in stereo vision. In *Proc. of CVPR*, pages 722–723, 1991.

[25] Dumontier C., Luthon F., and Charras J-P. Real-time dsp implementation for mfr-based video motion detection. *IEEE Transaction on Image Processing*, 8(10) :1341–1347, 1999.

[26] Fuh C. and Maragos P. Region-based optical flow estimation. In *Proc. of CVPR*, pages 130–135, 1989.

[27] Liu C. Capitalize on dimensionality increasing techniques for improving face recognition grand challenge performance. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(5) :725–737, 2006.

[28] Schnörr C. On functionals with greyvalue-controlled smoothness terms fordetermining optical flow. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(10) :1074–1079, 1993.

[29] Stauffer C. and Grimson E.L. Learning patterns of activity using real-time tracking. *IEEE Trans. Pattern Anal. Mach. Intell.*, 22(8) :747–757, 2000.

[30] Stiller C. Object-oriented video coding employing dense motion fields. In *Proc. of IEEE ICASSP*, pages 273–276, 1994.

[31] Stiller C. and Konrad J. Estimating motion in image sequences : A tutorial on modeling and computation of 2d motion. *IEEE Signal Process. Mag.*, 16 :70–91, 1999.

[32] Strecha C., Fransens R., and Van Gool L. A probabilistic approach to large displacement optical flow and occlusion detection. In *proc of ECCV Workshop SMVP*, pages 71–82, 2004.

[33] Wren C.R., Azarbayejani A., Darrell T., and Pentland A.P. Pfinder : Real-time tracking of the human body. *IEEE Trans. Pattern Anal. Mach. Intell.*, 19(7) :780–785, 1997.

[34] Wren C.R and Ivanov Y.A. A fast algorithm for depth segmentation. In *proc of Conference on Computer Vision and Pattern Recognition*, 2001.

[35] Clausi D. and Deng H. Design-based texture feature fusion using gabor filters and co-occurrence probabilities. *IEEE Trans on Image Processing*, 14(7) :925–936, 2005.

[36] Comaniciu D. Nonparametric information fusion for motion estimation. In *Proc. of CVPR*, pages 59–68, 2003.

[37] Comaniciu D. and Meer P. Mean shift analysis and applications. In *Proc. of ICCV*, pages 1197–1203, 1999.

[38] Donoho D. High-dimensional data analysis : The curses and blessings of dimensionality. Lecture for American Math. Society "Math Challenges of the 21st Century", 2000.

[39] Fleet D. and Jepson A. Computation of component image velocity from local phase information. *Int. J. Comput. Vision*, 5(1) :77–104, 1990.

[40] Geiger D., Ladendorf B., and Yuille A. Occlusions and binocular stereo. *Int. J. Comput. Vision*, 14(3) :211–226, 1995.

[41] Marr D. and Poggio T.A. Cooperative computation of stereo disparity. 194(4262) :283–287, 1976.

[42] Murray D., Kashko A., and Buxton H. A parallel approach to the picture restoration algorithm of Geman and Geman on a simd machine. *Image and Vision Computing*, 4 :141–152, 1986.

[43] Murray D. and Buxton B. Scene segmentation from visual motion using global optimization. *IEEE Trans. Pattern Anal. Machine Intell.*, 9(2) :220–228, 1987.

[44] Russel D. and Gong S. A highly efficient block-based dynamic background model. In *Proc. of IEEE conf on Adv. Video and Sig. Based Surv.*, pages 417 – 422, 2005.

[45] Scharstein D. and Szeliski R. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *Int. J. Comput. Vision*, 47(1-3) :7–42, 2002.

[46] Scharstein D., Szeliski R., and Zabih R. A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. In *Proc. of the IEEE Workshop on Stereo and Multi-Baseline Vision*, 2001.

[47] Toth D., Aach T., and Metzler V. Bayesian spatio-temporal motion detection under varying illumination. In *proc. of EUSIPCO*, pages 2081–2084, 2000.

[48] Zhang D. and Lu G. Segmentation of moving objects in image sequence : A review. *Circuits, Systems and Signal Process.*, 20(2) :143–183, 2001.

[49] Memin E. and Perez P. Hierarchical estimation and segmentation of dense motion fields. *Int. Journal of Computer Vision*, 46(2) :129–155, 2002.

[50] Ong E. and Spann M. Robust optical flow computation based on least-median-of-squares regression. *Int. J. Comput. Vision*, 30(1) :51–82, 1999.

[51] Simoncelli E., Adelson E., and Heeger D. Probability distributions of optical flow. In *Proc. of CVPR.*, pages 310–315, 1991.

[52] Trucco E. and Verri A. *Introductory Techniques for 3-D Computer Vision.* Prentice Hall, 1998.

[53] Bujor F., Trouve E., Valet L., Nicolas J-M., and Rudant J-P. Application of log-cumulants to the detection of spatiotemporal discontinuities in multitemporal SAR images. *IEEE trans. on Geo and Rem. Sens.*, 42(10) :2073–2084, 2004.

[54] Heitz F. and Bouthemy P. Multimodal estimation of discontinuous optical flow using markov random fields. *IEEE Trans. Pattern Anal. Mach. Intell.*, 15(12) :1217–1232, 1993.

[55] Meyer F. and Bouthemy P. Region-based tracking using affine motion models in long image sequences. *CVGIP : Image Understanding*, 60(2) :1994, 119–140.

[56] Aubert G., Deriche R., and Kornprobst P. Computing optical flow via variational techniques. *SIAM Journal on Applied Mathematics*, 60(1), 1999.

[57] Egnal G and Wildes R.P. Detecting binocular half-occlusions : Empirical comparisons of five approaches. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(8) :1127–1133, 2002.

[58] Farneback G. Very high accuracy velocity estimation using orientation tensors, parametric motion and simultaneous segmentation of motion field. In *Proc of ICCV*, pages 171–177, 2001.

[59] Klein G.J. and Huesman R.H. A 3d optical flow approach to addition of deformable pet volumes. In *Procc of the IEEE Workshop on Motion of Non-Rigid and Articulated Objects*, pages 136–145, 1997.

[60] Li H., Deklerck R., De Cuyper B., Hermanus A., Nyssen E., and Cornelis J. Object recognition in brain ct-scans : Knowledge-based fusion of data from multiple feature extractors. *IEEE Trans. on Med. Imaging*, 14(2) :212–229, 1995.

[61] Nagel H. Recent advances in image sequence analysis. In *proc. of Prem. Coll. Image : Trait., Synt., Tech. et App.*, pages 545–558, 1984.

[62] Nagel H. On the estimation of optical flow : relations between different approaches and some new results. *Artificial. Intelligence.*, 33(3) :298–324, 1987.

[63] Tao H., Sawhney H.S., and Kumar R. A global matching framework for stereo computation. In *proc. of ICCV*, pages 532–539, 2001.

[64] Nagel H-H. Image sequence evaluation : 30 years and still going strong. In *proc. of ICPR*, pages 1149–1158, 2000.

[65] Nagel H-H and Enkelmann W. Investigation of second order gray value variations to estimate corner point displacements. In *proc. of ICPR*, pages 768–773, 1982.

[66] Dagher I. and Nachar R. Face recognition using IPCA-ICA algorithm. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(6) :996–1000, 2006.

[67] Guyon I. and Elisseeff A. An introduction to variable and feature selection. *J. Mach. Learn. Res.*, 3 :1157–1182, 2003.

[68] Barron J. Experience with 3d optical flow on gated mri cardiac datasets. In *Proceedings of the 1st Canadian Conference on Computer and Robot Vision*, volume 0, pages 370–377, 2004.

[69] Barron J., Fleet D., and Beauchemin S. Performance of optical flow techniques. Technical Report 299, Univ. of Western Ontario, Dept. of CS, 1993.

[70] Barron J., Fleet D., and Beauchemin S. Performance of optical flow techniques. *Int. J. Comput. Vision*, 12(1) :43–77, 1994.

[71] Besag J. On the statistical analysis of dirty pictures. *J. Roy. Stat. Soc.*, 48(3) :259–302, 1986.

[72] Bigun J. Unsupervised feature reduction in image segmentation by local transforms. *Pattern Recognition Letters*, 14 :573–583, 1993.

[73] Choi J., Lee S., and Kim S. Segmentation and motion estimation of moving objects for object-oriented analysis-synthesis coding. In *Proc. of ICASSP*, pages 2431–2434, 1995.

[74] Kittler J., Hatef M., Duin R., and Matas J. On combining classifiers. *IEEE Trans. Pattern Anal. Mach. Intell.*, 20(3) :226–239, 1998.

[75] Konrad J. and Dubois E. Bayesian estimation of motion vector fields. *IEEE Trans. Pattern Anal. Mach. Intell.*, 14(9) :910–927, 1992.

[76] Kruger J. and Westermann R. Linear algebra operators for gpu implementation of numerical algorithms. *ACM Trans. Graph.*, 22(3) :908–916, 2003.

[77] Sun J., Li Y., Kang S.B., and Shum H-Y. Symmetric stereo matching for occlusion handling. In *proc of CVPR*, pages 399–406, 2005.

[78] Wang J. and Adelson E. Representing moving images with layers. *The IEEE Trans. on Image Proces. Sp. Issue : Image Seq. Comp.*, 3(5) :625–638, September 1994.

[79] Weber J. and Malik J. Robust computation of optical flow in a multi-scale differential framework. *Int. J. Comput. Vision*, 14(1) :67–81, 1995.

[80] Weickert J. On discontinuity-preserving optic flow. In *Proc. of CVMR*, pages 115–122, 1998.

[81] Weickert J. Coherence-enhancing diffusion of colour images. *Image Vision Comput.*, 17(3-4) :201–212, 1999.

[82] Weickert J. and Schnörr C. A theoretical framework for convex regularizers in pde-based computation of image motion. *International Journal of Computer Vision*, 45(3) :245–264, 2001.

[83] Weickert J. and Schnörr C. Variational optic flow computation with a spatio-temporal smoothness constraint. *International Journal of Computer Vision*, 14(3) :245–255, 2001.

[84] Zhong J. and Sclaroff S. Segmenting foreground objects from a dynamic textured background via a robust kalman filter. In *Proc of ICCV*, pages 44–50, 2003.

[85] Odobez J.-M. and Bouthemy P. Mrf-based motion segmentation exploiting a 2d motion model robust estimation. In *proc. of ICIP*, pages 628–631, 1995.

[86] Odobez J.-M. and Bouthemy P. Robust multiresolution estimation of parametric motion models. *Journal of Visual Communication and Image Representation*, 6(4) :348–365, 1995.

[87] Bouguet J.-Y. Pyramidal implementation of the lucas kanade feature tracker : Description of the algorithm. Technical report, Intel Corporation, 1999.

[88] Fukunaga K. and Hostetler L. The estimation of the gradient of a density function. *IEEE Trans. on Info. Theory*, 21 :32–40, 1975.

[89] Kim K., Chalidabhongse T., Harwood D., and Davis L. Real-time foreground-background segmentation using codebook model. *Real-Time Imaging*, 11(3) :172–185, 2005.

[90] Lim K., Das A., and Chong M. Estimation of occlusion and dense motion fields in a bidirectional bayesian framework. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(5) :712–718, 2002.

[91] Moreland K. and Angel E. The fft on a gpu. In *in proc. of Workshop on Graphics Hardware*, pages 112–119, 2003.

[92] Alvarez L., Esclarin J., Lefebure M., and Sanchez J. A pde for computing the optical flow. In *Proc. of XVI Congreso de Ecuaciones Diferenciales y Aplicaciones*, pages 1349–1356, 1999.

[93] Alvarez L., Weickert J., and Sánchez J. Reliable estimation of dense optical flow fields with large displacements. *Int. J. Comput. Vision*, 39(1) :41–56, 2000.

[94] Alvarez L., Deriche R., Papadopoulo R., and Sanchez J. Symmetrical dense optical flow estimation with occlusions detection. In *proc of ECCV*, pages 721–735, 2002.

[95] Bergen L. and Meyer F. A novel approach to depth ordering in monocular image sequences. In *Proc. of CVPR*, pages 536–541, 2000.

[96] Hong L. and Chen G. Segment-based stereo matching using graph cuts. In *proc. of CVPR*, pages 74–81, 2004.

[97] Lucchese L. and Mitra S. Color image segmentation : A state-of-the-art survey. In *Proc. of INSA-A*, pages 207–221, 2001.

[98] Abramoff M. and Viergever M.A. Computation and visualization of three-dimensional motion in the orbit. *IEEE Transactions on Medical Imaging*, 21(4) :296–304, 2002.

[99] Black M. Combining intensity and motion for incremental segmentation and tracking over long image sequences. In *Proc. of the Sec. European Conf. on Compt. Vis.*, pages 485–493, 1992.

[100] Black M. and Jepson A. Estimating optical flow in segmented images using variable-order parametric models with local deformations. *IEEE Trans. Pattern Anal. Mach. Intell.*, 18(10) :972–986, 1996.

[101] Black M. and Anandan P. A model for the detection of motion over time. In *Proc. of ICCV*, pages 33–37, 1990.

[102] Black M. and Anandan P. The robust estimation of multiple motions : parametric and piecewise-smooth flow fields. *Comput. Vis. Image Underst.*, 63(1) :75–104, 1996.

[103] Bleyer M. and Gelautz M. A layered stereo matching algorithm using image segmentation and global visibility constraints. *ISPRS Journal of Photogrammetry and Remote Sensing*, 59(3) :128–150, 2005.

[104] Chang M., Tekalp M., and Sezan I. Simultaneous motion estimation and segmentation. *IEEE Trans. on Image Processing*, 6(9) :1326–1333, 1997.

[105] Heikkila M. and Pietikainen M. A texture-based method for modeling the background and detecting moving objects. *IEEE Trans. Pattern Anal. Mach. Intell.*, 28(4) :657–662, 2006.

[106] Kraus M. and Strengert M. Pyramid filters based on bilinear interpolation. In *Accepted for publication in 2nd Int. Conf. on Comp. Graph. Theo. and App.*, 2007.

[107] Langer M. and Mann R. Optical snow. *Int. J. Comput. Vision*, 55(1) :55–71, 2003.

[108] Mech R.and Wollborn M. A noise robust method for 2d shape estimation of moving objects in video sequences considering a moving camera. *Signal Processing*, 66(2) :203–217, 1998.

[109] Mignotte M. A segmentation-based regularization term for image deconvolution. *IEEE Trans on Image Processing*, 15(7) :1973–1984, 2006.

[110] Tekalp M. *Digital video processing.* Prentice-Hall, Inc., 1995.

[111] Turk M. and Pentland A. Face recognition using eigenfaces. In *CVPR*, pages 586–591, 1991.

[112] Golberg M.A. and Cho H.A. *Introduction to Regression Analysis.* WIT Press, 2004.

[113] Friedman N. and Russell S.J. Image segmentation in video sequences : A probabilistic approach. In *UAI*, pages 175–181, 1997.

[114] Pichler O., Teuner A., and Hosticka B. An unsupervised texture segmentation algorithm with feature space reduction and knowledge feedback. *IEEE Trans. on Image Processing*, 7(1) :53–61, 1998.

[115] Anandan P. A computation framework and an algorithm for the measurement of visual motion. *Int. J. Comp. Vis.*, 2 :219–232, 1989.

[116] Bouthemy P. and Lalande P. Motion detection in an image sequence using gibbs distributions. In *Proc. of ICASSP*, pages 1651–1654, 1989.

[117] Bouthemy P. and Lalande P. Recovery of moving object masks in an image sequence using local spatiotemporal contextual information. *Optical Engineering*, 32(6) :1205–1212, 1993.

[118] Fua P. A parallel stereo algorithm that produces dense depth maps and preserves image features. *Machine Vision and Applications*, 6 :35–49, 1993.

[119] Rousseeuw P. and Leroy A. *Robust Regression and Outlier Detection.* Probability and Mathematical Statistics. John Willey & Son, 1987.

[120] Smith P., Drummond T., and Cipolla R. Layered motion segmentation and depth ordering by tracking edges. *IEEE Trans. Pattern Anal. Mach. Intell.*, 26(4) :479–494, 2004.

[121] Jodoin P-M., Rosenberger C., and Mignotte M. Detecting half-occlusion with a fast region-based fusion procedure. In *Proc. of BMVC*, pages 417–426, 2006.

[122] Jodoin P-M. and Mignotte M. Motion segmentation using a k-nearest-neighbor-based fusion procedure of spatial and temporal label cues. In *Proc. of ICIAR*, pages 778–785, Toronto, 2005. Springer.

[123] Jodoin P-M. and Mignotte M. Markovian segmentation and parameter estimation on graphics hardware. *Journal of Electrical Imaging*, 15(3) :033005, 2006.

[124] Jodoin P-M., Mignotte M., and Konrad J. Light and fast statistical motion detection method based on ergodic model. In *Proc. of IEEE ICIP*, pages 1053–1056, Atlanta, 2006.

[125] Huber P.J. *Robust Statistical Procedures, 2nd Ed.* SIAM, Philadelphia, 1996.

[126] Rosin P.L. Unimodal thresholding. *Pattern Recognition*, 34(11) :2083–2096, 2001.

[127] Rosin P.L. and Herva J. Remote sensing image thresholding methods for determining landslide activity. *Int. J. Remote Sensing*, 26(6) :1075–1092, 2005.

[128] Depommier R. and Dubois E. Motion estimation with detection of occlusion areas. In *Proc. of ICASSP*, pages 269–272, 1992.

[129] Deriche R., Kornprobst P., and Aubert G. Optical-flow estimation while preserving its discontinuities : A variational approach. In *proc of ACCV '95 : Invited Session Papers*, pages 71–80, 1996.

[130] Duda R., Hart P., and Stork D. *Pattern Classification (2nd Edition).* Wiley-Interscience, 2000.

[131] Duda R., Hart P., and Stork D. *Pattern Classification (2nd Edition).* Wiley-Interscience, 2000.

[132] Fernando R. and Kilgard M. *The Cg Tutorial : The Definitive Guide to Programmable Real-Time Graphics.* Addison-Wesley, 2003.

[133] Kanth R., Agrawal D., and Singh A. Dimensionality reduction for similarity searching in dynamic databases. In *Proc. ACM SIGMOD inter. conf. on Manag. of data*, pages 166–176. ACM Press, 1998.

[134] Kimmel R., Malladi R., and Sochen N. Images as embedded maps and minimal surfaces : Movies, color, texture, and volumetric medical images. *Int. J. Comput. Vision*, 39(2) :111–129, 2000.

[135] Megret R. and DeMenthon D. A survey of spatio-temporal grouping techniques. Technical report, University of Maryland, College Park, 2002.

[136] Strzodka R. and Rumpf M. Level set segmentation in graphics hardware. In *Proc. of ICIP*, 3, pages 1103–1106, 2001.

[137] Whitaker R. A level-set approach to 3d reconstruction from range data. *Int. J. Comput. Vision*, 29(3) :203–231, 1998.

[138] Dubes R.C. *Cluster Analysis and Related Issues.* Handbook of Pattern Recognition and Computer Vision, C.H. Chen, L.F. Pau, and P.S.P. Wang (editors), 1992.

[139] Gonzalez R.C. and Woods R.E. *Digital Image Processing.* Prince Hall, 2002.

[140] Randi J. Rost. *OpenGL Shading Language.* Addison-Wesley, 1st edition, 2004.

[141] Geman S. and Geman D. Stochastic relaxation, gibbs distributions, and the bayesian restoration of images. *IEEE Trans. Pattern Anal. Machine Intell.*, 6(6) :721–741, 1984.

[142] Ince S. and Konrad J. Geometry-based estimation of occlusions from video frame pairs. In *Proc. of ICASSP*, volume 2, pages 933–936, 2005.

[143] Jabri S., Duric Z., Wechsler H., and Rosenfeld A. Detection and location of people in video images using adaptive fusion of color and edge information. In *ICPR*, pages 4627–4631, 2000.

[144] Khan S. and Shah M. Object based segmentation of video color, motion and spatial information. In *Proc. of CVPR*, pages 746–751, 2001.

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

[146] Li S. *Markov random field modeling in computer vision.* Springer-Verlag, 1995.

[147] Reed S., Ruiz I. R., Capus C., and Petillot Y. The fusion of large scale classified side-scan sonar image mosaics. *IEEE Trans. on Image Processing*, 15(7) :2049–2060, 2006.

[148] Lai S-H. and Vemuri B. Reliable and efficient computation of optical flow. *Int. J. Comput. Vision*, 29(2) :87–105, 1998.

[149] Aach T., Dumbgen L., and Mester R.and Toth D. Bayesian illumination-invariant motion detection. *Signal Processing*, 31(2) :165–180, 1993.

[150] Brox T., Bruhn A., Papenberg N., and Weickert J. High accuracy optical flow estimation based on a theory for warping. In *Proc of ECCV*, pages 25–36, 2004.

[151] Guerror T., Zhang G., Huang T-C., and Lin K-P. Intrathoracic tumour motion estimation from ct imaging using the 3d optical flow method. *Physics in Medicine and Biology*, 49 :4147–4161, 2004.

[152] William B. Thompson. Exploiting discontinuities in optical flow. *Int. J. Comput. Vision*, 30(3) :163–173, 1998.

[153] Dang V., Mansouri A.-R., and Konrad J. Motion estimation for region-based video coding. In *Proc of ICIP*, pages 2189–2192, 1995.

[154] Kolmogorov V. and Zabih R. Computing visual correspondence with occlusions via graph cuts. In *Proc. of ICCV*, pages 508–515, 1999.

[155] Oppenheim V. and Schafer R. *Digital Signal Processing.* Prentice-Hall, Englewood Cliffs, NJ, 1975.

[156] Vapnik V. *Estimation of Dependences Based on Empirical Data.* Springer-Verlag, New-York Heidelberg Berlin, 1982.

[157] Vapnik V. *Statistical Learning Theory.* Wiley-Interscience, New-York, 1998.

[158] Onana V-P., Trouve E., Mauris G., Rudant J-P, and Tonye E. Linear features extraction in rain forest context from interferometric SAR images by fusion of coherence and amplitude information. *IEEE Trans. on Geo. and Rem. Sens.*, 41(11) :2540–2556, 1999.

[159] Jiang W., Er G., Dai Q., and Gu J. Similarity-based online feature selection in content-based image retrieval. *IEEE Trans. on Image Process.*, 15(3) :702–712, 2006.

[160] Pieczynski W. Statistical image segmentation. *Machine Graphics and Vision*, 1(1) :261–268, 1992.

[161] Pieczynski W. and Benboudjema D. Multisensor triplet markov fields and theory of evidence. *Image Vis. Comput.*, 24(1) :61–69, 2006.

[162] Thompson W.B. Combining motion and contrast for segmentation. *IEEE Trans. Pattern Anal. Mach. Intell.*, 2(6) :543–549, 1980.

[163] Zhou X-S., Comaniciu D., and Gupta A. An information fusion framework for robust shape tracking. *IEEE Trans. Pattern Anal. Mach. Intell.*, 27(1) :115–129, 2005.

[164] Altunbasak Y., Eren P., and Tekalp M. Region-based parametric motion segmentation using color information. *Graph. Models Image Process.*, 60(1) :13–23, 1998.

[165] Sheikh Y. and Shah M. Bayesian object detection in dynamic scenes. In *Proc of CVPR*, pages 74–79, 2005.

[166] Wei Y. and Quan L. Region-based progressive stereo matching. In *proc. of CVPR*, pages 106–113, 2004.

[167] Kato Z., Pong T., and Qiang S. Multicue mrf image segmentation : combining texture and color features. In *Proc. of ICPR02*, volume 1, pages 660– 663, 2002.