Architecture dédiée au traitement d'image base sur les équations aux dérivées partielles

Eva Dejnozkova

ECOLE DES MINES
DE PARIS
Collège doctoral

N° attribué par la bibliothèque

|__|__|__|__|__|__|__|__|__|__|

T H E S E

pour obtenir le grade de
Docteur de l'Ecole des Mines de Paris
Spécialité "Morphologie mathématique"

présentée et soutenue publiquement par
Eva DEJNOŽKOVÁ

le 9 mars 2004

ARCHITECTURE DEDIEE AU TRAITEMENT D'IMAGE BASE SUR LES EQUATIONS AUX DERIVEES PARTIELLES

Directeur de thèse : Jean-Claude KLEIN
Co-Directeur de thèse : Petr DOKLADAL

Jury

M. Michel BARLAUD	Président
M. Laurent COHEN	Rapporteur
M. Alain MERIGOT	Rapporteur
M. Christian GAMRAT	Examinateur

A mon père

Remerciements

Une thèse représente un travail mené pendant plusieurs années. Il serait très difficilement réalisable sans soutient et l'accompagnent de l'entourage. C'est pourquoi je tiens avant tout à remercier :

à Jean-Claude Klein, pour m'avoir permis de travailler sur le sujet aussi intéressant et pour sa confiance en me permettant de travailler dans une grande autonomie et liberté,

à Petr Dokladal, pour la rigueur avec laquelle il m'a aidé à mener ce travail à bien, et avec qui j'ai beaucoup aimé travailler,

aux membres de jury, M. Michel Barlaud, M. Alain Mérigot, M. Laurent Cohen et M. Christian Gamrat d'avoir participé au jury de soutenance et pour leurs remarques très pertinentes qui m'ont aidé à améliorer ce document,

à Fernand Meyer, Dominique Jeulin, Jean Serra et Serge Beucher pour m'avoir accueilli au Centre de Morphologie mathématique et qui m'ont permis d'acquérir et approfondir des connaissances en analyse d'image.

à Catherine Moysan, pour sa disponibilité et chez qui j'ai apprécié les qualité humaines,

à Marc Waroquier et Laura Andriamasinoro, pour leurs bonne humeur quotidienne.

à Michel Bilodeau, Béatriz Marcotegui et Etienne Decencière, pour leur gentillesse et chez qui j'ai apprécié leurs compétences.

à mes parents, pour leur infaillible confiance en moi et pour tous ci qu'ils m'ont transmis,

à Shahram Zahirazami qui m'a montré le chemin au Centre de morphologie mathématique,

et aux autres membres du Centre de morphologie mathématique que j'ai eu le plaisir de connaître, pour l'ambiance sympathique qu'ils ont su créer.

Résumé

Les méthodes de traitement d'images fondées sur les équations aux dérivées partielles (EDP) bénéficient d'une attention particulière de la part de la communauté scientifique. Le nombre d'applications a considérablement augmenté après la formulation du problème sous forme d'ensembles de niveaux. Les EDPs s'appliquent dans de nombreux domaines tels le filtrage des images (diffusion non-linéaire), les contours actifs utilisés pour la segmentation des images statiques (graphe de Voronoï, Ligne de Partage des Eaux, plus court chemin, détection d'objets), aussi bien que des séquences d'images (suivi d'objets) ou encore des méthodes plus récentes tel le *shape-from-shading*.

Les applications industrielles de ces méthodes sont néanmoins très limitées, d'une part par une complexité considérable de calculs (nombre d'itérations très élevé, par ex.), d'autre part par des difficultés rencontrées lors d'implantation embarquées (consommation d'énergie, exigences mémoire). Quelques expériences temps-réel ont été publiées avec des super-calculateurs ou des accélérateurs graphiques. Quant aux applications embarquées, elles sont à notre connaissance quasi-inexistantes.

Notre but est de proposer une architecture dédiée, facilitant tant l'implantation temps-réel qu'embarquée.

En vue de cet objectif nous proposons un nouvel algorithme de solution de l'équation Eikonale/calcul de fonction distance qui procède en parallèle, élimine l'usage des files d'attente hiérarchiques et permet d'obtenir la solution sur la totalité ou seulement sur une partie de l'image (le *narrow band*). La complexité de cet algorithme, nommé Massive Marching, est linéaire. Nous estimons que l'impact de Massive Marching est d'autant plus important pour la communauté de Morphologie Mathématique, qu'il s'agit du premier algorithme permettant d'obtenir *en parallèle* la ligne de partage des eaux *non-biaisée*.

Ensuite, nous proposons deux types d'architecture (i) SIMD et (ii) plusieurs cœurs de processeurs embarqués implantant Massive Marching en parallèle ou semi-parallèle. Ces mêmes types d'architecture peuvent être utilisés pour implanter un filtrage aussi bien que des méthodes à évolution d'interface. La même architecture peut donc être utilisée pour implanter une application complète, composée de différents types d'algorithmes comme par exemple filtrage suivi par segmentation.

Abstract

The image processing methods based on Partial Differetial Equations (PDEs) draw a groving attention of the scientific community. The number of applications has considerably increased with the introduction of the Level Set methods. The EDPs apply in many domains such the image improvement (non linear diffusion), the segmentation by active contours used both for static pictures (graphe of Voronoï, watershed, shortest paths, object detection) and image sequences (object tracking) or more recent methods such as *shape-from-shading*.

The industrial applications of these methods remain very limited due to a considerable computation complexity (high iteration number) on one hand, and the difficulties of embedded system implementation (energy consumption, memory requirements) on the other hand. To our knowledge, few real time experiments on supercomputers and graphics hardware were published, and embedded applications are almost inexistent.

Our goal is to propose a dedicated architecture facilitating the implementation of a real time embedded system.

Regarding this objective, we propose a new algorithm (called Massive Marching) solving the Eikonal equation for computation of the distance function. It proceeds in parallel and eliminates the usage of the ordered data structures. It allows to obtain the solution either on the entire image or only on its part : the *Narrow Band* around the propagation front. The complexity of Massive Marching is linear. We consider that its introduction is important also for the community of Mathematical Morphology, because Massive Marching represents the algorithm allowing to obtain the watershed in parallel.

Next, we propose two architecture types (i) SIMD and (ii) MIMD, based on several embedded processor cores, implementing Massive Marching in parallel or semi-parallèle. The same architecture types can be used to implement filtering algorithms as well as methods of interface evolution. The same architecture can therefore be used to implement all the steps of a complete application consisting of different types of algorithms, e.g. filtering followed by segmentation.

Table des matières

Table des Figures **5**

Liste des Tableaux **11**

Avant propos **13**
Motivation . . . 13
Equations aux dérivées partielles en traitement d'image . . . 15
Structure de thèse . . . 16

I Moyens mathématiques et algorithmiques **19**

1 Introduction **21**
1.1 Notions mathématiques . . . 21
1.1.1 Equations aux dérivées partielles . . . 21
1.1.2 Géométrie planaire différentielle . . . 24
1.1.2.1 Courbure euclidienne . . . 24
1.2 Evolution des courbes . . . 25
1.2.1 Théorie des courbes de niveaux . . . 27
1.2.2 Deux formulations des méthodes des ensembles de niveaux . . . 30
1.3 Analyse numérique . . . 32
1.3.1 Discrétisation spatiale . . . 32
1.3.2 Discrétisation temporelle . . . 32
1.3.2.1 Schéma explicite . . . 33
1.3.2.2 Schéma semi-implicite . . . 34
1.3.2.3 Remarque sur le traitement des bords . . . 35

2 Applications **37**
2.1 Amélioration des images . . . 38
2.1.1 Lissage . . . 38
2.1.1.1 Diffusion linéaire . . . 38

2.1.1.2 Diffusion non linéaire . . . 39
2.1.2 Lissage des contours . . . 41
2.2 Morphologie mathématique continue . . . 41
2.2.1 Opérateurs de base . . . 43
2.2.1.1 Les EDP d'érosion et de dilatation . . . 43
2.2.1.2 Erosions et dilatations "déformables" . . . 44
2.2.2 Reconstruction des images . . . 46
2.2.2.1 Nivellement continu . . . 46
2.2.2.2 Swamping . . . 48
2.2.3 Fonction distance . . . 48
2.2.4 Segmentation . . . 50
2.2.4.1 Ligne de partage des eaux . . . 51
2.2.4.2 Diagramme de Voronoï . . . 55
2.3 Contours actifs . . . 55
2.4 Conclusions . . . 59

3 Analyse des algorithmes 61
3.1 Type image entière . . . 62
3.1.1 Schéma explicite . . . 62
3.1.1.1 Diffusion linéaire . . . 63
3.1.1.2 Diffusion non linéaire . . . 63
3.1.1.3 Opérateurs morphologiques continus . . . 64
3.1.1.4 Nivellement continu . . . 64
3.1.2 Schéma semi-implicite . . . 64
3.1.3 Complexité de calcul et parallélisation . . . 66
3.2 Type Narrow Band . . . 68
3.2.1 Initialisation des contours . . . 68
3.2.1.1 Interpolation linéaire . . . 70
3.2.2 Contours actifs . . . 71
3.2.2.1 Schéma explicite . . . 72
3.2.2.2 Schéma semi implicite . . . 74
3.2.3 Formulation stationnaire . . . 74
3.2.3.1 Propagation équidistante . . . 77
3.2.3.2 Groupe Marching . . . 79
3.2.3.3 Balayages successifs . . . 80
3.2.3.4 Approximation . . . 82
3.3 Conclusion . . . 84

4 Massive Marching 89

4.1 Schéma numérique 90
4.2 Algorithme 90
4.2.1 Algorithme à deux étapes 93
4.2.2 Critère d'activation 95
4.2.3 Propagation des étiquettes 99
4.2.4 Erreur de calcul 102
4.2.5 Complexité de calcul et temps d'exécution 104
4.2.5.1 Complexité de calcul 104
4.2.5.2 Nombre d'opérations 104
4.3 Conclusions 106

II Architecture 109

5 Architectures parallèles 111

6 SIMD 119
6.1 Architecture globale 120
6.2 Contrôle global 122
6.2.1 Arrêt d'algorithme 124
6.3 Unité de traitement 125
6.3.1 Extraction du voisinage complet 127
6.3.2 Initialisation par interpolation 129
6.3.3 Approximation du schéma numérique 132
6.3.3.1 Linéarisation par morceaux 134
6.3.3.2 Look up Table 136
6.3.4 Ecriture des résultats 137
6.3.5 Activation 138
6.4 Résultats expérimentaux 140
6.5 Diffusion non linéaire 144
6.6 Conclusions 146

7 Architecture multiprocesseur 149
7.1 Vue générale des algorithmes 150
7.1.1 Analyse du flux de données 152
7.1.2 Analyse temporelle 154
7.1.2.1 Points de synchronisation 156
7.2 Architecture globale 156
7.2.1 Mémoire des points actifs 159
7.2.2 Sémaphores 161

7.2.3 Modèle de processeur 162
7.3 Distance topographique continue 164
7.3.1 Discussion des résultats 164
7.4 Conclusions 168
7.4.1 Perspectives 170

Conclusions **173**

Bibliographie **177**

Liste des autheurs **185**

Table des figures

1 Le progrès technologique respecte toujours la loi de Moore. (Les informations visualisées dans cette figure ont été obtenues à partir de la documentation technique disponible (Motorola, 2003), (Intel, 2003), (AMD, 2003)) 14

1.1 Courbure : cercle osculateur 25
1.2 Courbes et fonction de courbure 26
1.3 Image et courbes de niveaux. 28
1.4 Description implicite du contour 29
1.5 Direction de l'évolution en fonction de la définition de l'intérieur et de l'extérieur du contour. 29
1.6 Ensembles de niveau 30
1.7 Exemple : évolution contrôlée par la courbure du contour. En haut : l'extérieur et l'intérieur du contour pendant l'évolution, en bas : la description implicite du contour, seuls les contours dans l'intervalle $[-15, 15]$ sont visualisés. Le contour de l'objet correspond au niveau zéro. 31
1.8 Formulation stationnaire. Cet exemple montre le mouvement du contour gouverné par l'équation Eikonale. A chaque instant, la valeur du point est une fonction de la position et représente la distance parcourue depuis la source. . . 31

2.1 Diffusion linéaire. La première ligne montre des images résultantes, la deuxième ligne contient les images des courbes de niveaux (pour chaque 15 niveaux) pour n itérations. 39
2.2 Fonction $g()$, avec paramètre $k = 2$, $k = 5$ (-), $k = 10$ (:) 40
2.3 Diffusion non linéaire. La première ligne montre des images résultantes, la deuxième ligne contient les images des courbes de niveaux (pour chaque 15 niveaux) pour n itérations. 40
2.4 Lissage des contours par l'équation géométrique de la chaleur.(a) Objet binaire. (b) Objet binaire après le lissage des contours par l'équation géométrique de la chaleur. 41
2.5 Résultats de l'érosion et dilatation continue, $\tau = 0.2$ 45

2.6 Résultats de l'érosion et de la dilatation continue, $\tau = 0.2$ 45
2.7 Exemples des nivellements continus, $\tau = 0.2$ 47
2.8 Exemple de swamping pour une image des marqueurs choisis manuellement.(a) Image originale. (b) Minima régionaux de l'image originale. (c)Marqueurs choisis manuellement, en tant que minima régionaux à imposer. (d) Résultat de swamping continu. 48
2.9 Résultats du calcul de la fonction distance pondérée. (a) Distance pondérée calculée à partir d'une source unique et deux régions des vitesses différentes. (b) Courbes de niveaux de la fonction distance (en blanc) pondérée par niveau de gris de l'image originale. 49
2.10 Recherche du plus court chemin (en termes de variations minimales de la surface) entre deux points choisis arbitrairement dans l'image de la rétine. 50
2.11 Exemple de la segmentation morphologique continue. La LPE a été calculée à partir des marqueurs imposés manuellement. 54
2.12 Diagramme de Voronoï 55
2.13 Segmentation des vaisseaux rétiniens (Klein, Walter, Massin and Dejnožková, 2003). Les contours initiaux sont matérialisés par le squelette calculé sur l'image normalisée (Walter, 2003). 58
2.14 Exemple de suivie de visage du conducteur basé sur les contours actifs. Le modèle est décrit dans (Dokládal, Enficiaud and Dejnožková, 2004) 59

3.1 Classification des algorithmes basés sur des EDPs. 61
3.2 Schéma explicite 67
3.3 Schéma semi-implicite 69
3.4 Exemple de la fonction $\varphi(x, y)$. Le fond blanc représente l'extérieur de la courbe, le fond gris l'intérieur. Les points, dont la valeur est obtenue par l'interpolation, sont marqués en noir. 70
3.5 Exemple de détection de la courbe initiale par interpolation linéaire. (a) Fonction $\varphi(x, y)$, visualisée en tant que surface 3D. La ligne noire indique la courbe initiale détectée avec une précision sous-pixélique. (b) La courbe initiale, détectée par l'interpolation linéaire, superposée sur la fonction $\varphi(x, y)$. 70
3.6 Interpolation linéaire : diverses configurations des points voisins avec le signe différent du point au cours de traitement. 71
3.7 Contours actifs : schéma explicite 73
3.8 Contours actifs : schéma semi implicite 75
3.9 *Fast marching* : les ensembles $\mathcal{A}$, $\mathcal{Q}$, $\mathcal{T}$ 78
3.10 Fast marching 78
3.11 Sweeping 81

3.12 Exemples de configuration du voisinage : les points rouges q_i seront utilisés pour le calcul de la valeur du point p ; les points blancs n'ont pas encore obtenu leur valeur (supposons qu'ils aient été initialisés à ∞) et ne peuvent pas être pris en compte pour le calcul de $u(p)$. 82
3.13 Comparaisons du schéma de Godunov et de la méthode d'intersections 83

4.1 Premières itérations de l'algorithme Massive Marching. 92
4.2 Algorithme Massive Marching 93
4.3 Propagation de la fonction distance. (a) Image initiale et la fonction distance pondérée par des valeurs d'intensité de l'image. (b) Contours de la fonction distance. (c) Points actifs dans l'itération en cours. 94
4.4 Exemple des ondes de propagation et de la zone de re-activation (en gris). S_1 et S_2 sont des sources de propagation. Des lignes composées de tirets représentent la position des fronts de propagation après n_1 et n_2 itération. 95
4.5 Propagation simultanée de la fonction distance et des étiquettes 100
4.6 Comparaisons des différents algorithmes : diagramme de Voronoï. 101
4.7 Comparaison des différents algorithmes : distance pondérée. 101
4.8 Erreur de calcul sur le voisinage incomplet. La ligne indique la position du contour décrite par les valeurs de fonction distance signée des points. Les flèches montrent quels points sont considérés pour le calcul des nouvelles valeurs. . . . 103
4.9 Exemples avec différents nombres d'opérations. 105

5.1 Système enfoui. 112
5.2 Exemples des différents types d'architecture (Noguet, 2002) 113
5.3 Mémoire partagée et distribuée. Une unité de traitement est marquée comme PU ; une mémoire est marquée comme M. 114

6.1 Schéma bloc d'une architecture SIMD. 120
6.2 Réseau de communications 121
6.3 Partition de l'image dans les mémoires privées des unités de traitement. 121
6.4 Schéma bloc du module Contrôle global 122
6.5 Massive Marching en instructions de l'architecture proposée. 123
6.6 Le principe de détection de la fin de propagation au niveau global. 124
6.7 Principe de détection de la fin de propagation au niveau d'une unité de traitement. Les indices i et j représentent le numéro de l'unité de traitement et de la ligne. Remarque : La valeur de PUA_i est remise à zéro au début de l'image. . 125
6.8 Unité de traitement. 126
6.9 Directions des échanges des valeurs des points voisins. 127
6.10 Exemple du contenu des mémoires pour l'extraction de voisinage. 128
6.11 Bus bidirectionnels de communication 128

6.12 Diagramme de synchronisation de la lecture du voisinage d'un point. Les valeurs affichées sont des valeurs numériques lues de la mémoire des données. 129
6.13 Extraction de voisinage. (Bus bus_e et bus_w sont ici appelés rightside et leftside) 130
6.14 Détection de position du contour initial. 131
6.15 Interpolation. 131
6.16 Architecture interne du module d'approximation. 133
6.17 Recherche d'intervalle : schéma fonctionnel. 134
6.18 Recherche d'intervalle : schéma électronique. 135
6.19 Timing de calcul de la fonction distance. Nous avons séparé les résultats intermédiaires du multiplicateur (M) et de l'accumulateur (ACC) . L'indice j indique le numéro du point traité. 136
6.20 Timing de calcul de la fonction distance par approximation à LUT. L'indice j indique le numéro du point traité. 137
6.21 Mémoire double port 137
6.22 Propagation de l'adresse d'écriture. n est le nombre de cycles d'horloge utilisé pour le calcul. 138
6.23 Activation : schéma fonctionnel. 139
6.24 Demandes d'activation 139
6.25 Demandes d'activation 140
6.26 Activation : implantation matérielle. Remarque : signaux ARS, ARN, ARW, ARE sont ici appelés as, an, aw, ae. 140
6.27 Approximations de f_{diff} 141
6.28 Les courbes de niveaux de la fonction distance obtenues pour différentes approximations et précision de 8+8 bits. 141
6.29 Diagramme de Voronoï calculée sur la fonction distance obtenue pour différentes approximations et 8+8 bits de précision. 142
6.30 Erreur $\mathcal{L}_\infty$ mesurée en fonction de nombre de bits de précision pour différentes approximations et comparée à la solution exacte. 143
6.31 Estimation de surface occupée par bloc Approximation. (en nombre de portes NAND équivalentes) 143
6.32 Exemple de reconfiguration du bloc d'approximation pour diffusion non linéaire. 145
6.33 Diffusion non linéaire en instructions de l'architecture proposée. 146

7.1 Algorithme généralisé. 152
7.2 Flux de données. La largeur des branches correspond au volume des données qui sont transférées par ces branches. 153
7.3 Etude de synchronisation d'exécution des algorithmes EDPs. 154
7.4 Algorithme généralisé des EDPs. 156
7.5 Principe des points de synchronisation 157

7.6 Architecture globale. 158
7.7 Représentation d'une mémoire LIFO 159
7.8 Fonctionnement d'une mémoire LIFO 159
7.9 Gestion des LIFOs. Les points actuellement actifs sont marqués en noir. 160
7.10 Principe des sémaphores vue du côté de processeur. 161
7.11 Diagramme bloc du modèle de processeur. 162
7.12 La LPE continue par Massive Marching. 165
7.13 Rapport des temps d'exécution 166
7.14 Temps d'exécution en cycles d'horloge 167
7.15 Distribution de l'activité sur les différents processeurs. 167
7.16 Exemples des résultats obtenus. 169
7.17 Estimation du gain du temps d'exécution en nombre de cycles d'horloge. 170

Liste des tableaux

1 Tableau comparatif des processeurs. 14

3.1 Complexité des calculs 79

4.1 Le nombre de points recalculés (en %). La mesure a été effectuée pour une image 256×256 et la vitesse de propagation constante. 96

4.2 Comparaison des temps d'exécution pour une image 560×420 (Fig.) sur Pentium III sous Linux à 450MHz et des bandes passantes. 106

5.1 Différences de fonctionnement des systèmes SIMD et MIMD. 113

6.1 Format d'instruction. N est le nombre de bits choisi pour la représentation de la partie fractionnelle des nombres en virgule fixe. 122

6.2 OPCODE d'instructions 123

6.3 Exemple des configurations des signes des points de voisinage 132

6.4 Adresses des constantes d'approximation. 132

6.5 Suite des calculs pour la linéarisation par morceaux. (*) Si une des valeurs est infinie, l'autre est mise à sa place. La même approche est appliquée aux bords de l'image ou la valeur du point manquant est considérée comme infinie. 136

6.6 Suite des calculs pour la LUT. Si une des valeurs est infinie, le calcul se réduit à l'addition de 1. La même approche est appliquée aux bords de l'image. 136

6.7 L'erreur d'approximation du schéma de Godunov donné sur la Fig. 6.28 142

6.8 Les erreurs (approximation plus arrondi) ℓ_∞ des résultats de la Fig. 6.28(b, c, d et e) sont comparés avec le résultat exact de la Fig. 6.28(a). 143

6.9 Estimation de la surface occupée par l'implantation du bloc Approximation avec 8+8 bits de précision 144

6.10 Estimation de la surface occupée par l'implantation du bloc Approximation avec 8 + 16 bits de précision 144

6.11 Suite des calculs d'un terme $g(u)u$ par la linéarisation par morceaux. 145

6.12 Suite des calculs pour la LUT. 146

7.1 Statistique des points insérés dans la mémoire de points actifs à partir d'un point. Les résultats représentent les valeurs moyennes qui ont été obtenus pour le diagramme de Voronoï et la LPE par algorithme Massive Marching. 154
7.2 Bande passante des différents types d'architectures et temps de propagation par Massive Marching à partir d'une seule source placée dans un coin de l'image (de 256×256 points) 167

Avant-propos

L'augmentation incessante du nombre d'applications de la vision par ordinateur demande une utilisation de plus en plus large des systèmes embarqués (voitures, téléphones mobiles etc.). Pour répondre aux besoins de ces applications, l'industrie fait appels aux avancées technologiques et mathématiques. L'objectif de cette thèse est de rendre le monde de l'embarqué accessible aux algorithmes de traitement d'image les plus récents en proposant une architecture dédiée.

Motivation

Aujourd'hui, les systèmes embarqués sont utilisés pour accomplir des tâches de plus en plus complexes et souvent en temps réel. Cela impose des contraintes sur leur puissance de calcul de plus en plus élevée. La réponse à ces exigences est recherchée dans plusieurs axes :

- Mise en œuvre des composants électroniques à la fréquence plus élevée.
- Mise en œuvre des algorithmes parallèles.

En 1965, dans son article (Moore, 1965), G. Moore prédit l'évolution de l'électronique numérique. Il prévoit que l'utilisation des circuits intégrés va se répandre dans tous les domaines de la science ainsi que, par exemple, dans les ménages, l'équipement des voitures, les communications ou les ordinateurs personnels. Dans le même article, il donne une estimation du prochain progrès des technologies en termes de croissance des fréquences et de la densité en nombre de transistors.

Moore formule une hypothèse, connue comme la loi de Moore, selon laquelle *"le nombre de transistors intégrés sur une seule composante va doubler tous les 18 mois"*. La Fig. 1 montre que le développement des nouvelles technologies suit toujours cette loi. Personne ne sait combien de temps cette loi pourrait persister avant d'atteindre les limites physiques et technologiques de la fabrication des circuits intégrés.

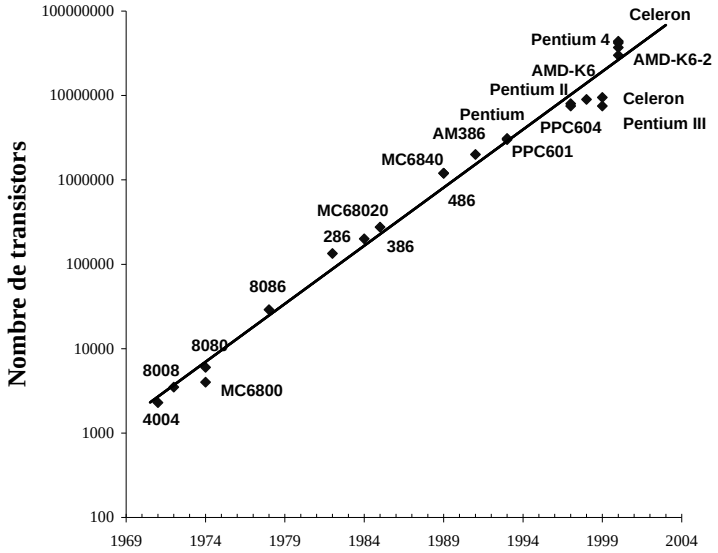


FIG. 1 – Le progrès technologique respecte toujours la loi de Moore. (Les informations visualisées dans cette figure ont été obtenues à partir de la documentation technique disponible (Motorola, 2003), (Intel, 2003), (AMD, 2003))

Cette augmentation de la densité des composants électroniques a pour conséquence une croissance de la vitesse de fonctionnement. La vitesse de fonctionnement est primordiale pour obtenir une meilleure puissance de calcul. En revanche, il est connu que l'augmentation des fréquences provoque l'augmentation de la consommation d'énergie (Gelsinger, 2001), un facteur pénalisant pour des systèmes embarqués.

	1997	1999	2001	2003	2006
Densité des trans. (M/mm^2)	3,7	6,2	10	18	39
Taille (mm^2)	300	340	385	430	520
Fréquence (GHz)	0,75	1,25	1,5	2,1	3,5
Consommation (W)	70	90	110	130	160
Technologie (μm)	0,20	0,14	0,12	0,10	70.10^{-3}

TAB. 1 – Tableau comparatif des processeurs.

La question de consommation de l'énergie devient prédominante. La technologie seule ne peut résoudre ce problème. Il est nécessaire de chercher des améliorations à tous les niveaux de conception des nouveaux systèmes : technologie, architecture (partage des tâches, représentation des données) et algorithmes (complexité).

Si l'on considère les processeurs dans des ordinateurs traditionnels, la surface active lors de l'exécution d'une instruction représente seulement quelques % de sa surface totale. Afin d'exploiter au maximum la puissance extraordinaire donnée par le nombre de transistors toujours croissant, il faut profiter des possibilités offertes par des algorithmes parallèles et paralléliser les calculs au maximum. En plus, cela permet de satisfaire des éventuelles contraintes

temporelles même avec une vitesse de traitement moins élevée et, par conséquent, avec une consommation d'énergie également moindre.

Aujourd'hui, le développement des architectures parallèles bénéficie du progrès récent dans le domaine des circuits programmables (Debes, 2003). Les nouvelles technologies permettent de concevoir des systèmes de plus en plus complexes à l'intérieur d'une seule composante.

Depuis un certain temps, les fabricants dotent les circuits programmables des blocs spéciaux et même des noyaux entiers de processeurs embarqués afin de pouvoir réaliser des applications toujours plus performantes. Avec la mise au point des techniques des architectures reconfigurables de plus en plus perfectionnées, les circuits programmables commencent même à jouer le rôle du concurrent des ASICs.

De plus, la durée de développement des nouveaux systèmes est de plus en plus courte ce qui représente un facteur très important pour l'industrie.

Equations aux dérivées partielles en traitement d'image

Les méthodes aux équations aux dérivées partielles (EDPs) bénéficient des progrès récents en analyse numérique et en géométrie différentielle. Grâce aux avancées informatiques, nous pouvons rencontrer maintenant les EDPs dans de nombreux domaines : physique, sciences des matériaux, médecine ou transport routier. Ces applications de pointe font appel à des outils de traitement d'images appartenant à des domaines très variés : morphologie mathématique continue, segmentation par contours actifs, filtrage, détection, reconnaissance, suivi des objets et autres.

Avantages des EDPs :

- Les EDPs permettent de modéliser les images dans le domaine continu.
- La formulation par EDPs est indépendante de la définition de la maille. Elle est isotrope.
- Les EDPs permettent de formuler de nouveaux filtres qui satisfont la causalité et la localité.
- Les algorithmes sont performants et allient précision et stabilité de solution.
- Les EDPs permettent d'obtenir des démonstrations de l'existence et l'unicité des solutions.

L'idée d'utiliser des EDPs dans le domaine du traitement d'images date des années 60 (Gabor, 1965). Un peu plus tard Jain (1977) a continué dans cet axe. Une contribution importante a été faite par Koenderink (1984) et Witkin (Witkin, 1983) qui ont introduit l'approche multi-échelle basée sur la convolution gaussienne. Hummel (1986) a démontré que toute équation qui satisfait le principe du maximum, définit un espace multi-échelle. Alvarez, Guichard, Lions and Morel (1993) ont établi des axiomes de base et les EDPs fondamentales.

L'introduction de la théorie des courbes de niveaux (Approche précédemment connue de la morphologie mathématique (Matheron, 1975)) dans le domaine des EDPs (Osher and Sethian, 1988) a joué un rôle important. La méthode des courbes de niveaux permet d'obtenir des algorithmes plus efficaces grâce à la description implicite des fronts qui se propagent. Elle permet également de gérer aisément des problèmes de changement de topologie lors de l'évolution.

A ce jour, les systèmes embarqués opérant selon des algorithmes EDPs sont quasi inexistants. Malgré leurs avantages théoriques les EDPs restent inaccessibles aux applications embarqués notamment à cause des calculs qui demandent le nombre élevé d'itérations. En revanche, elles possèdent des possibilités élevées de parallélisme des calculs dont une architecture dédiée pourrait bénéficier.

Structure du document

Introduction

Dans ce chapitre, nous présentons des principales définitions et outils mathématiques des équations aux dérivées partielles relatives aux traitements d'images. Après une introduction des notions du domaine de la géométrie planaire, nous nous concentrons sur les méthodes des ensembles de niveaux. Ensuite, dans la section de l'analyse numérique, nous présentons des méthodes de discrétisation.

Chapitre 2 : Applications

Le domaine du traitement d'image par les équations aux dérivées partielles est très varié. Nous essayons de montrer cette richesse par quelques exemples d'applications. Dans ce but, nous avons choisi des représentants typiques pour les opérations effectuées par traitement d'image : amélioration, filtrage et segmentation des images. Pour chaque opérateur mentionné, nous discutons l'équation d'évolution appropriée et le principe de son fonctionnement.

Chapitre 3 : Analyse des algorithmes

Nous distinguons deux groupes d'algorithmes : ceux, qui dans chaque itération parcourent l'image entière et ceux qui procèdent par propagation d'onde et ne touchent que des points proches de l'onde. En vue de trouver des caractéristiques communes à une implantation matérielle, nous étudions la discrétisation et des approches algorithmiques pour les opérateurs présentés dans le chapitre précédent. De cette analyse résulte que le goulet d'étranglement des calculs parallèles est l'inexistence d'un algorithme de calcul de la fonction distance euclidienne pondérée permettant de placer la courbe avec une précision sous-pixélique et efficace aussi bien sur l'image entière que sur une bande étroite autour de l'onde. Par conséquence, l'in-

existence d'un algorithme efficace parallèle de la fonction distance représente un vrai verrou technologique.

Chapitre 4 : Massive Marching

Pour lever ce verrou technologique, nous avons développé un nouvel algorithme entièrement parallèle de calcul de la fonction distance. Cet algorithme, appelé Massive Marching, matérialise la solution de l'équation Eikonale et permet d'être implanté de façon séquentielle, semi parallèle ou massivement parallèle. Ici, nous présentons la formulation de Massive Marching et la propagation simultanée des étiquettes des composantes connexes. Nous étudions également la complexité de calcul et l'erreur numérique de Massive Marching. En plus, nous démontrons que l'algorithme se termine en temps fini.

Chapitre 5 : Architectures parallèles

Ce chapitre est une courte introduction à l'état de l'art des architectures parallèles dédiées au traitement d'images. Nous y définissons nos objectifs et les contraintes posées par la conception d'une architecture dédiée aux équations aux dérivées partielles.

Chapitre 6 : SIMD

Le premier type d'architecture spécialisée proposée pour la mise en œuvre des EDPs est une architecture de type SIMD. Dans le passé, ce type d'architecture a été utilisé avec succès pour le filtrage par diffusion non linéaire des séquences vidéo en temps réel. En bénéficiant des avantages du Massive Marching, nous montrons la faisabilité d'implantation des algorithmes à évolution de courbe sur ce type d'architecture. Par l'implantation de Massive Marching, nous avons doté cette architecture de nouvelles fonctionnalités et par conséquent des possibilités de porter des opérateurs de segmentation. Une importante contribution est le calcul de la fonction distance avec le schéma de Godunov à l'aide des approximations : linéarisation par morceaux et LUT sans avoir à effectuer des calculs complexes comme les racines ou les puissances carrées. Dans la section des résultats expérimentaux, l'attention est portée sur l'étude de l'erreur numérique des approximations.

Chapitre 7 : Architecture multiprocesseur

L'architecture SIMD étant limitée en capacité de l'unité de calcul pour approximer des fonctions non linéaires par linéarisation par morceaux ou par LUT, nous proposons une architecture fondée sur plusieurs noyaux de processeurs. D'abord nous étudions des chemins de données et des contraintes temporelles et ensuite nous proposons une exécution asynchrone des algorithmes aux EDPs afin d'obtenir les performances maximales. Dans la réalisation finale, nous avons quatre processeurs qui exécutent l'algorithme de manière asynchrone sans aucune communication. Une section est consacrée à la gestion par sémaphores des accès simultanés à la mémoire.

L'architecture proposée a été testée sur la Ligne de partage des eaux continue. Nous avons effectué des mesures pour tester que l'activité des processeurs est équilibrée et pour estimer les performances.

Conclusions

Finalement, dans les conclusions, nous rappelons les principales contributions de cette thèse. En plus, nous développons les perspectives de notre travail et les futurs axes de recherche.

Première partie

Moyens mathématiques et algorithmiques

Chapitre 1

Introduction

Les méthodes de traitement d'images font appel à des branches très variées des mathématiques. Ce chapitre introduit des outils mathématiques nécessaires pour comprendre les principes de solution des EDPs. Ces outils représentent à la fois le guide de construction des algorithmes mais également des contraintes de réalisation.

1.1 Notions mathématiques

1.1.1 Equations aux dérivées partielles

Dans cette thèse nous nous focalisons sur des modèles basés sur des équations aux dérivées partielles (EDPs) linéaires et non linéaires. Les équations aux dérivées partielles peuvent être vues comme un ensemble de relations liant les dérivées partielles d'une fonction. Voici (à titre d'exemple) les trois types d'équations aux dérivées partielles que l'on peut rencontrer dans le traitement d'images :

- elliptique : équation de Poisson ou de Laplace
- parabolique : diffusion de la chaleur
- hyperbolique : équation de propagation d'onde

Il est connu que les modèles réalistes sont le plus souvent non linéaires. Nous pouvons également dire que les EDPs non linéaires permettent de décrire des situations concrètes à une certaine échelle, en étudiant le comportement moyen d'un grand nombre de "particules". Dans la recherche des solutions, l'analyse mathématique se révèle être un élément très important car elle permet de :

- valider des modèles
- trouver des discrétisations appropriées

- définir des algorithmes de solution

L'analyse mathématique des EDPs peut être très difficile non seulement à cause des non-linéarités des EDPs mêmes mais aussi à cause des instabilités ou des discontinuités des solutions. Elle s'appuie alors sur l'analyse des familles de solutions telles que solutions de viscosité ou solutions d'entropie pour construire des solutions, étudier la stabilité et effectuer l'analyse numérique. Une étude détaillée des axiomes fondamentaux de la théorie des EDPs peut être trouvée dans (Guichard and Morel, 1995) ou dans (Alvarez, Lions and Morel, 1992).

Solution faible et loi de conservation. Un front qui évolue sous le contrôle d'une EDP peut créer une forme singulière, par exemple un angle, même en temps fini. Dans ce cas, nous pouvons obtenir des solutions multiples. Ce problème peut être vu comme le problème d'ambiguïté du calcul du vecteur normal (à cause des discontinuités des dérivées) où il n'est pas clair, comment l'évolution du front devrait continuer. Il est nécessaire de trouver la *solution faible*, c'est à dire la solution qui satisfait la forme intégrale de l'équation, qui demande le degré de différenciabilité le moins élevé et fournit ainsi la solution la plus générale. Souvent la solution faible est obtenue par une étude des lois de conservation où la loi de conservation indique que le volume des changements dans le temps d'une substance à l'intérieur d'une région est égal au flux de cette substance passant par la frontière de la région.

Solution d'entropie et solution de viscosité. La condition d'entropie est équivalente à la loi hyperbolique de conservation pour les discontinuités qui se propagent. L'équation sous forme de loi hyperbolique de conservation peut développer des discontinuités même si les conditions initiales sont lisses. En ajoutant à cette loi le terme de diffusivité basé sur la dérivée seconde, ce terme agit comme un élément de lissage et empêche la création des discontinuités. Du point de vue physique, ce terme représente la viscosité des liquides. En plus, il a été prouvé que si ce terme est strictement positif, la solution est toujours lisse. Dans la propagation des fronts, la courbure prend souvent le rôle du terme de viscosité.

Du côté de l'analyse mathématique, l'élément inséparable de formulation complète de toute EDP est la définition des conditions initiales. Nous pouvons rencontrer plusieurs types de conditions initiales. Un type définit la valeur initiale à partir de laquelle un système évolue. Le deuxième type spécifie les conditions aux limites des régions.

Dans le domaine du traitement d'image, une image sert de support pour la définition des conditions initiales ainsi que des solutions. Mathématiquement, une image n-dimensionnelle est une fonction de n variables spatiales où les valeurs de cette fonction représentent par exemple la luminosité, la teinte ou un vecteur de couleurs. Dans le traitement numérique, les images sont digitalisées par le processus d'échantillonage lors de l'acquisition.

Définition 1 (Image discrète)
Une image I est définie comme un ensemble discret des nombres réels qui sont organisés dans un champ $n-$dimensionnel des points.

$$I : \mathbb{Z}^n \rightarrow \mathbb{R}^v$$

Ce champ de points peut avoir des formes différentes, les plus utilisées en deux dimensions sont :

- *rectangulaire avec des distances Δx et Δy entre les points dans la direction x et y*
- *hexagonale : consiste à regarder des lignes paires et impaires de champ rectangulaire décalées à droite*

En général, les valeurs de l'image appartenant à $\mathbb{R}^v$ peuvent être scalaires ($v = 1$) ou vectorielles ($v > 1$).

Dans le cas des valeurs scalaires qui varient entre le noir et le blanc nous appelons l'image "image à niveau de gris". Une image est une image binaire si elle ne contient que deux valeurs uniques, par exemple 0 *et* 1.

Pour la suite, s'il n'est pas spécifié autrement, sous le terme *image* nous sous-entendons une image rectangulaire unitaire et isotrope ($\Delta x = \Delta y$=1) aux valeurs à niveaux de gris.

Les relations entre les points d'image s'expriment en termes de voisinage ou de connexité. La connexité joue un rôle important pour la construction des algorithmes et l'analyse de leur propriété.

Définition 2 (4-voisinage d'un point)
Soit $I : \mathbb{Z}^2 \rightarrow \mathbb{R}$ et soit $p = [x_p, y_p]$ un point sur un support isotrope unitaire. Le 4-voisinage $V_4(p)$ du point p est défini :

$$V_4(p) = \{[x_p, y_p \pm 1], [x_p \pm 1, y_p]\} \tag{1.1}$$

Le point q est appelé 4-voisin de (ou 4-adjacent à) p si $q \in V_4(p)$

De la même façon, nous pouvons définir un voisinage plus large :

Définition 3 (8-voisinage d'un point)
Soit $I : \mathbb{Z}^2 \rightarrow \mathbb{R}$ et soit $p = [x_p, y_p]$ un point sur un support isotrope unitaire. Le 8-voisinage $V_8(p)$ du point p est défini :

$$V_8(p) = \{[x_p \pm 1, y_p \pm 1]\} \tag{1.2}$$

Le point q est appelé 8-voisin de (ou 8-adjacent à) p si $q \in V_8(p)$

1.1.2 Géométrie planaire différentielle

De nombreuses implantations des EDPs aboutissent à des réalisations des algorithmes basés sur l'évolution des fronts. En $2D$ ce front devient une courbe. L'étude des courbes permet de construire de nombreuses applications contrôlées par des caractéristiques de courbes allant de lissage de contours jusqu'à la segmentation par contours actifs.

Une courbe planaire peut être vue comme un contour d'un objet planaire.

Définition 4 (Courbe)
La courbe est un mapping d'une ligne réelle dans un plan réel (Sapiro, 2000) :

$$\mathcal{C}(p) : [a, b] \in \mathbb{R} \subset \mathbb{R}^2 \tag{1.3}$$

où p représente le paramètre de la courbe $\mathcal{C}$. Pour toute valeur du point $p_0 \in [a, b]$ nous obtenons un point dans le plan $\mathcal{C}(p_0) \in \mathbb{R}^2$.

*Si $\mathcal{C}(a) = \mathcal{C}(b)$ la courbe est **fermée**.*

*La courbe a une **intersection** s'il existe au moins une paire de paramètres p_0, $p_1 \in (a, b)$ et $p_0 \neq p_1$ tels que $\mathcal{C}(p_0) = \mathcal{C}(p_1)$. Sinon la courbe est appelée courbe simple.*

1.1.2.1 Courbure euclidienne

La plus importante caractéristique d'une courbe est la courbure κ. La Fig. 1.1 montre le principe de la définition de courbure. Considérons un arc infinitésimal ds de courbe $\mathcal{C}$ entre les points s_1 et s_2, nous pouvons remplacer ds par un arc de cercle osculateur avec rayon r. La courbure κ est l'inverse du radius r du cercle osculateur :

$$\kappa = \frac{1}{r} \tag{1.4}$$

Parmi d'autres expressions pour la courbure, nous pouvons mentionner une formulation dérivée de paramétrisation euclidienne de courbe. Si nous reprenons la définition (1.3) de courbe $\mathcal{C}(p)$, nous passons à une paramétrisation spéciale $\mathcal{C}(s)$ où la courbe est définie sur l'intervalle $s \in [0, L]$, L est la longueur de la courbe. La courbure est ensuite définie comme suit :

$$\kappa\vec{\mathcal{N}} = \frac{d^2C}{ds^2} \tag{1.5}$$

où $\vec{\mathcal{N}}$ est le vecteur normal et s est le paramètre d'une paramétrisation avec $s \in [0, L]$ (Sapiro, 2000).

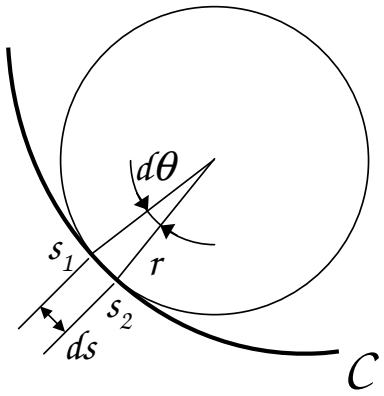


FIG. 1.1 – Courbure : cercle osculateur

$$\vec{\mathcal{N}} = \frac{\nabla u}{\|\nabla u\|} \tag{1.6}$$

$$\kappa = \nabla \frac{\nabla u}{\|\nabla u\|} = \frac{u_{xx}u_y^2 - 2u_x u_y u_{xy} + u_{yy}u_x^2}{\sqrt{\left(u_x^2 + u_y^2\right)^3}} \tag{1.7}$$

La longueur euclidienne. La longueur euclidienne d'une courbe est également l'un des concepts de base de la géométrie planaire. La longueur euclidienne L entre deux points de la courbe :

$$L(p_0, p_1) = \int_{p_0}^{p_1} \left[\left(\frac{dx}{dp}\right)^2 + \left(\frac{dy}{dp}\right)^2 \right]^{\frac{1}{2}} dp \tag{1.8}$$

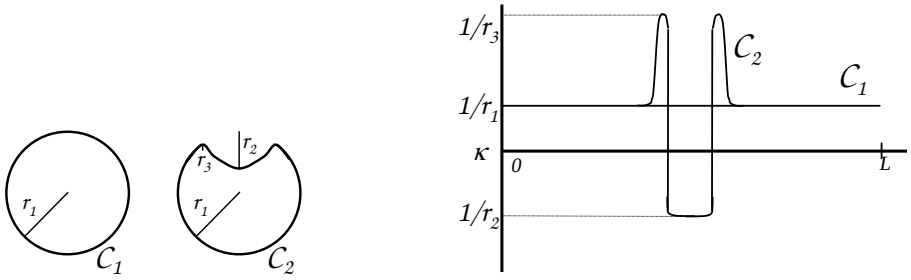
Quelques propriété relatives à la courbure :

- La courbure d'une droite est nulle car le radius r du cercle osculateur tend vers l'infini : $\frac{1}{\infty} = 0$.
- La courbure des cercles est constante, égale à l'inverse du radius du cercle.
- L'intégrale de la courbure d'une courbe fermée est le multiple de 2π (Sapiro, 2000).

Toute courbe $\mathcal{C}(p)$ peut être uniquement représentée par la fonction de courbure κ (Fig. 1.2). Cette notation a pour avantage d'être invariante sous rotation et translation (Sapiro, 2000).

1.2 Evolution des courbes

Le traitement d'images gouverné par les EDPs est modélisé par des courbes $\mathcal{C}(p,t) : \mathbb{R}^1 \times [0,T) \rightarrow \mathbb{R}^2$ qui se déforment dans le temps. L'EDP décrivant l'évolution des courbes est la



(a) Deux courbes différentes de la même longueur L.

(b) Exemple de fonction de courbure κ pour les deux courbes $\mathcal{C}_1$ et $\mathcal{C}_2$.

FIG. 1.2 – Courbes et fonction de courbure

suivante :

$$\begin{cases} \frac{\partial \mathcal{C}}{\partial t} = \mathcal{F}\vec{\mathcal{N}} \\ \mathcal{C}(t=0) = C_0 \end{cases} \tag{1.9}$$

où $\mathcal{F}$ est une fonction de vitesse d'évolution de la courbe et $\mathcal{N}$ est le vecteur normal de courbe. La fonction $\mathcal{F}$ peut être une fonction des arguments comme la courbure, le vecteur normal.

L'équation (1.9) est le résultat important des travaux de Epstein et Gage qui l'ont formulée sous forme de lemme (Epstein and Gage, 1987) disant que si la vitesse $\mathcal{F}$ ne dépend pas de la paramétrisation alors la vitesse dans la direction de tangente n'a pas d'influence sur la déformation géométrique de la courbe. Autrement dit, l'évolution de la courbe dans la direction tangentiale ne change pas la courbe mais c'est la paramétrisation qui changerait (Sapiro, 2000).

Le même raisonnement peut être appliqué aux surfaces $\mathcal{S} : \mathbb{R}^2 \times [0, T) \rightarrow \mathbb{R}^3$. L'évolution d'une surface est décrite par l'équation :

$$\begin{cases} \frac{\partial \mathcal{S}}{\partial t} = \mathcal{F}\vec{\mathcal{N}} \\ \mathcal{S}(t=0) = \mathcal{S}_0 \end{cases} \tag{1.10}$$

où $\vec{\mathcal{N}}$ est le vecteur normal en $3D$ de la surface.

La construction des algorithmes numériques de solution des EDPs (1.9) et (1.10) doit tenir compte de plusieurs problèmes :

- *Précision et stabilité.* Certains algorithmes exigent l'utilisation d'un pas d'intégration très petit afin de préserver leur stabilité ; également une discrétisation plus ou moins astucieuse influence la robustesse de la discrétisation.
- *Singularités.* Une courbe qui se déforme selon l'Eq. (1.9) peut développer des discontinuités même si la courbe initiale a été entièrement lisse. C'est la solution d'entropie qui permet de déterminer la suite d'évolution car elle permet de choisir la bonne solution

faible. Un exemple de solution d'entropie est représenté par le principe de Hüygens.

- *Changement de topologie.* Lors de l'évolution, la courbe peut changer la topologie : se diviser ou joindre une autre courbe.

De nombreux algorithmes ont été utilisés dans le passé pour calculer l'évolution des courbes. Parmi des exemples, nous pouvons citer l'approche par paramétrisation des courbes où la gestion des changements de topologie des courbes est très complexe, ou le contrôle de la position de la courbe par simulation du volume contenu par les points (Sethian, 1996).

1.2.1 Théorie des courbes de niveaux

Le passage de la description paramétrique vers une description implicite a signifié un pas très important permettant la construction des algorithmes performants. Cette approche a été introduite par (Osher and Sethian, 1988). Elle s'appuie sur le principe de base de la morphologie mathématique introduit par (Matheron, 1975) qui, dans l'espace des fonctions continues $\mathcal{I}$, interprète ces fonctions comme une famille imbriquée des ensembles de niveaux. L'équation 1.11 en fournit un exemple en 2D.

$$\mathcal{C}_i = \left\{ x \in \mathbb{R}^2 | I(x) = const_i \right\}, \quad I \in \mathcal{I} \tag{1.11}$$

La Fig. 1.3 montre le principe des courbes de niveau. Les valeurs représentent la hauteur des points qui créent la surface (Fig. 1.3(a)). En connectant les points à la même hauteur, nous obtenons des courbes de niveaux de l'image I (Fig. 1.3(b)). La projection dans le plan x–y se trouve sur (Fig. 1.3(c)). L'action des EDPs sur la surface de l'image peut s'interpréter comme une déformation spatiale à l'échelle du temps.

De ce raisonnement part l'idée de (Osher and Sethian, 1988) de représenter le front d'évolution en tant que l'ensemble de niveau zéro d'une fonction distance signée. Considérons une fonction continue u en $2D$:

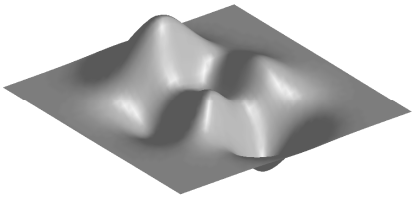
$$u(x) : \mathbb{R}^2 \rightarrow \mathbb{R}$$

La fonction $u(x)$ est définie de la manière suivante :

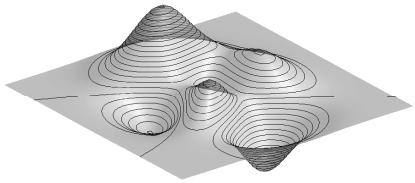
$$u(x) = \text{sign}\,(x)\,\text{dist}\,(x, \mathcal{C}_0) \;\; \text{où} \;\; \text{sign}(x) = \begin{cases} 1 \text{ si } x \text{ se trouve à l'extérieur de } \mathcal{C}_0 \\ -1 \text{ si } x \text{ se trouve à l'intérieur de } \mathcal{C}_0 \end{cases} \tag{1.12}$$

où dist est la distance du point x à l'ensemble de niveau zéro $\mathcal{C}_0 = \left\{ x \in \mathbb{R}^2 : I(x) = 0 \right\}$ de (1.11). Le signe de la fonction distance est donnée par la position du point x, si le point x se trouve à l'extérieur de l'ensemble zéro, on choisit le signe positif et vice-versa.

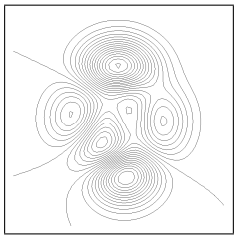
Remarque : la description implicite du contour (Fig. 1.4) est unique et équivalente à la



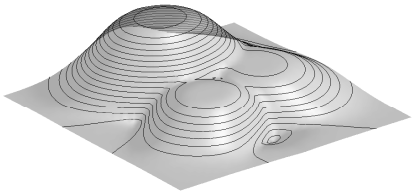
(a) Image visualisée comme une surface.



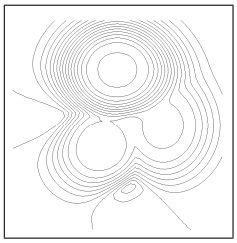
(b) Les courbes de niveaux superposées sur la surface de l'image (distance de 10).



(c) Les courbes de niveaux projetées dans le plan x–y.

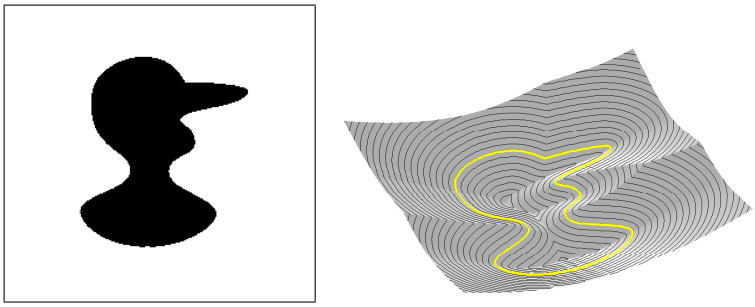


(d) Image dilatée avec les courbes de niveaux superposées sur la surface.



(e) Les courbes de niveaux de l'image dilatée, projetées dans le plan $x - y$.

FIG. 1.3 – Image et courbes de niveaux.



(a) $I(x)$: image d'entrée avec un objet dont l'intérieur est marqué en noir et l'extérieur en blanc.

(b) $u(x)$: fonction distance en tant que description implicite du contour de l'objet $\mathcal{C}_0$ en blanc.

FIG. 1.4 – Description implicite du contour

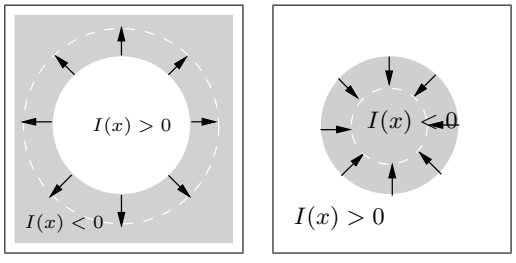
description paramétrique si la fonction distance satisfait (Gomez and Faugeras, 1992) :

$$\|\nabla u(x)\| = 1 \tag{1.13}$$

Le contour de (1.12) évolue dans la direction normale $\vec{\mathcal{N}}$ où $\vec{\mathcal{N}} = \frac{\nabla u}{\|\nabla u\|}$ avec une vitesse de déformation $\mathcal{F}$ donnée par la EDP qui gouverne l'évolution : $\mathcal{N} = \mathcal{F}$. Après l'ajout du terme de l'évolution temporelle, l'équation fondamentale des méthodes des ensembles de niveau basées sur EDPs s'écrit :

$$\begin{cases} \frac{\partial u}{\partial t} + \mathcal{F}\|\nabla u\| = 0 \\ u(x, t = 0) = u_0 \end{cases} \tag{1.14}$$

A chaque instant t, la position du front dans l'équation (1.14) est donnée par la position de l'ensemble de niveau zéro. Le choix du signe de l'intérieur et de l'extérieur du contour détermine la direction du mouvement du contour (1.5).



(a) (b)

FIG. 1.5 – Direction de l'évolution en fonction de la définition de l'intérieur et de l'extérieur du contour.

1.2.2 Deux formulations des méthodes des ensembles de niveaux

Comme il résulte de l'équation (1.14), la vitesse de déplacement du contour est donnée par la fonction $\mathcal{F}$. En général, $\mathcal{F} = \mathcal{F}(G, l, g)$ est une fonction des propriétés globales (G) et locales (l) de l'image d'entrée et en même temps des caractéristiques géométriques (g) du contour lui-même. En théorie, les propriétés de la fonction $\mathcal{F}$ sont données par le type de problème à résoudre par rapport à l'équation (1.14). En théorie, nous pouvons rencontrer deux types de problèmes.

1) Problème général

Ce type de problème reprend l'équation fondamentale. Théoriquement, la fonction $\mathcal{F}$ peut changer de signe au cours de l'évolution et le long du contour. En revanche, il est nécessaire de choisir des schémas numériques robustes (en fonction des valeurs de $\mathcal{F}$) afin de ne pas perturber la stabilité de la solution. Le principe de l'évolution contrôlée par l'équation (1.15) est visualisé sur la Fig. 1.6. La Fig. 1.7 montre l'exemple de l'évolution contrôlée par la courbure.

$$\begin{cases} \frac{\partial u}{\partial t} + \mathcal{F}\|\nabla u\| = 0 \\ u(x, t = 0) = \text{sign}\,(x)\,\text{dist}\,(x, \mathcal{C}_0) \\ \mathcal{C}_t = \{x | u(x, t) = 0\} \\ \mathcal{F} \text{ arbitraire} \end{cases} \tag{1.15}$$

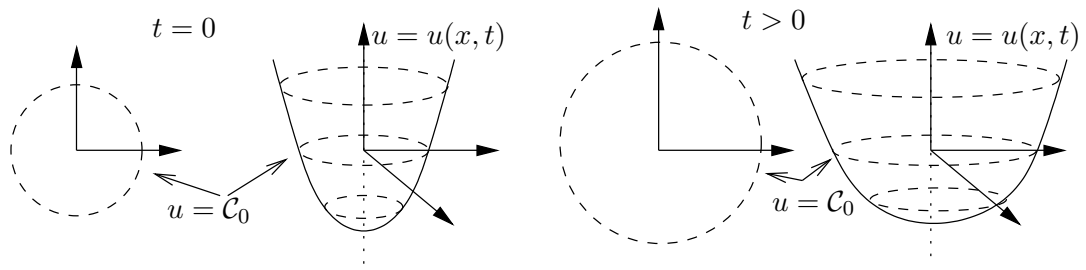


FIG. 1.6 – Ensembles de niveau

2) Problème stationnaire

Dans des cas particuliers, l'équation fondamentale (1.14) se transforme en problème indépendant du temps. La vitesse de propagation $\mathcal{F}$ doit impérativement être ou positive ou négative le long de la propagation. Si $\mathcal{F}$ ne dépend que de la position, nous obtenons l'équation Eikonale (1.16).

$$\begin{cases} \mathcal{F}\|\nabla u\| = 1 \\ \mathcal{C}_t = \{x | u(x, t) = t\} \\ \mathcal{F} > 0 \end{cases} \tag{1.16}$$

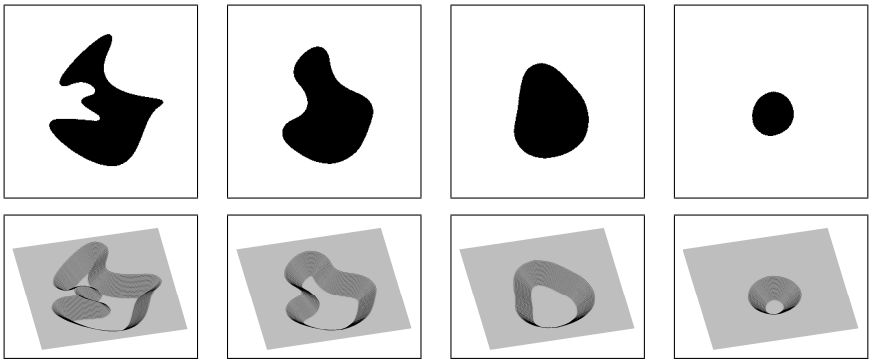


FIG. 1.7 – Exemple : évolution contrôlée par la courbure du contour. En haut : l'extérieur et l'intérieur du contour pendant l'évolution, en bas : la description implicite du contour, seuls les contours dans l'intervalle $[-15, 15]$ sont visualisés. Le contour de l'objet correspond au niveau zéro.

Le problème stationnaire est souvent interprété comme une propagation d'ondes à partir des sources données. Ainsi, les résultats représentent le temps de passage de l'onde par les points particuliers.

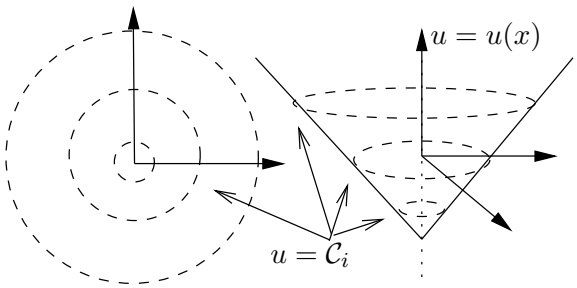


FIG. 1.8 – Formulation stationnaire. Cet exemple montre le mouvement du contour gouverné par l'équation Eikonale. A chaque instant, la valeur du point est une fonction de la position et représente la distance parcourue depuis la source.

Les deux problèmes utilisent la description implicite des courbes de niveau (ou des contours en général) ce qui permet de mettre en œuvre des algorithmes efficaces dont les points forts :

- Gestion simple des changements de topologie.
- Mesure facile des propriétés géométriques à partir de la fonction des courbes de niveaux $u(x)$ (Eq. (1.6)).
- Solution numérique par les différences finies.
- Le contour peut former des angles.
- Extension directe en $3D$.

1.3 Analyse numérique

Dans la plupart de cas problématiques des EDPs, la solution analytique est difficile ou impossible à déterminer. Par conséquent, nous sommes contraints de chercher la solution à l'aide des algorithmes d'approximation numérique. En plus, les images représentant les conditions initiales sont, elles aussi, définies sur un support discret en tant que résultat d'échantillonage lors du processus d'acquisition.

La définition des meilleures approches de discrétisation représente, en soi, un grand axe de recherche à part entière. Dans cette section, nous essayons d'abord d'établir des formules de base pour ensuite les utiliser dans le reste du document. En même temps nous visons par quelques exemples à dévoiler les points importants de l'analyse numérique liée à notre domaine d'intérêt.

1.3.1 Discrétisation spatiale

Comme nous avons déjà annoncé, l'utilisation des différences finies fait partie des points forts de l'approche par courbes de niveaux. Ainsi, les différences finies spatiales servent à estimer le gradient, vecteur normal ou courbure le long du front et sont nécessaires pour calculer l'évolution des équations. L'ordre des dérivées le plus utilisé est un et deux.

Considérons $I = I(i, j)$ une image isotrope, les dérivées spatiales au point p_{ij} sont obtenues selon les expressions suivantes :

$$\begin{aligned}
I_x^+(p_{i,j}) &= \frac{I(p_{i+1,j}) - I(p_{i,j})}{\Delta x} \\
I_x^-(p_{i,j}) &= \frac{I(p_{i,j}) - I(p_{i-1,j})}{\Delta x} \\
I_x(p_{i,j}) &= \frac{I(p_{i+1,j}) - I(p_{i-1,j})}{2\Delta x} \\
I_{xx}(p_{i,j}) &= \frac{I(p_{i+1,j}) - 2I(p_{i,j}) + I(p_{i-1,j})}{\Delta x} \\
I_{xy}(p_{i,j}) &= \frac{I(p_{i+1,j+1}) - I(p_{i-1,j+1}) - I(p_{i+1,j-1}) + I(p_{i-1,j-1})}{4\Delta x}
\end{aligned} \tag{1.17}$$

1.3.2 Discrétisation temporelle

La solution numérique appropriée de l'équation $u_t + \mathcal{F}\|\nabla u\| = 0$ doit satisfaire la condition d'entropie. Cela permet de choisir la *solution faible* correcte parmi les alternatives. Afin de choisir la solution faible la plus appropriée aux applications où le front sépare deux régions, Sethian (Sethian, 1996) a formulé la *condition d'entropie*. Sous cette condition, le front est vu comme un feu qui se propage dans un champ : une fois une particule brûlée, elle le reste. Cela

signifie que la solution n'est pas réversible. Ainsi, nous perdons un certain volume d'information ce qui a pour conséquence que des données initiales ne peuvent pas être retrouvées à partir de la solution. La condition d'entropie définit ainsi l'ordre dans lequel les points obtiennent leurs valeurs. Les valeurs elles-mêmes sont obtenues selon les schémas numériques ci-dessous.

1.3.2.1 Schéma explicite

L'équation des méthodes des ensembles de niveaux peut être réécrite sous forme :

$$u_t + H(u_x, u_y) = 0 \tag{1.18}$$

où $H(u_x, u_y)$ est connu comme l'Hamiltonien. Dans les schémas numériques pour la discrétisation de l'Hamiltonien, on utilise les opérateurs des dérivées finies.

Les schémas numériques les plus utilisés sont divisés en deux classes, selon le type de l'Hamiltonien. On distingue deux cas : convexe et non-convexe. L'Hamiltonien est convexe si H est lisse dans N dimensions et si $\frac{\partial^2 H(u)}{\partial p_i p_j} \geq 0$ où $P = (p_1, \ldots, p_N)$ (Sethian, 1996).

En fonction de la précision numérique exigée, on a le choix entre les schémas numériques d'ordre plus ou moins élevé. Il est clair que les schémas d'ordre plus élevé sont plus complexes et peuvent faire appel au calcul des différences finies d'ordre plus élevé ou utiliser un voisinage plus large pour leurs calculs. A titre d'exemple, nous donnons ici le schéma du premier ordre, qui est le plus utilisé.

Hamiltonien convexe

$$u_{ij}^{n+1} = u_{ij}^n - \tau \left[\max\{\mathcal{F}_{ij}, 0\}\nabla^+ + \min\{\mathcal{F}_{ij}, 0\}\nabla^- \right] \tag{1.19}$$

où

$$\begin{aligned} \nabla^+ &= [\max\{I_x^-, 0\}^2 + \max\{I_y^-, 0\}^2 \\ &\quad + \min\{I_x^+, 0\}^2 + \min\{I_y^+, 0\}^2]^{\frac{1}{2}} \\ \nabla^- &= [\min\{I_x^-, 0\}^2 + \min\{I_y^-, 0\}^2 \\ &\quad + \max\{I_x^+, 0\}^2 + \max\{I_y^+, 0\}^2]^{\frac{1}{2}} \end{aligned} \tag{1.20}$$

Hamiltonien non-convexe

$$u_{ij}^{n+1} = u_{ij}^n - \tau \left[H\left(\frac{I_x^- + I_x^+}{2}, \frac{I_y^- + I_y^+}{2} \right) - \frac{1}{2}\alpha_u \left(I_x^+ + I_x^- \right) - \frac{1}{2}\alpha_v \left(I_y^+ + I_y^- \right) \right] \tag{1.21}$$

où α_u, α_v sont les bornes des dérivées partielles de H relatives au premier ou deuxième argument et $H = \mathcal{F}\|\nabla u\|$.

1.3.2.2 Schéma semi-implicite

La facilité d'implantation des schémas explicites est en général payée par l'obligation d'utiliser le pas d'intégration τ très petit pour ne pas perturber la stabilité de la solution numérique (Sethian, 1996). Ceci introduit un grand nombre d'itérations nécessaires afin d'obtenir la convergence. Ainsi les applications concernées sont pénalisées par un temps de traitement élevé.

(Weickert, 1998) a introduit un schéma semi-implicite d'intégration nommé *Additive Operator Splitting (AOS)* avec une stabilité de solution numérique absolue et indépendant de τ. Afin de pouvoir utiliser ce schéma semi-implicite, il est obligatoire que l'équation différentielle. partielle soit formulée sous une forme particulière :

$$\begin{cases} \frac{\partial u}{\partial t} = \operatorname{div}\left(g\left(\|\nabla u\|\right)\nabla u\right) \\ u(t=0) = u_0 \end{cases} \tag{1.22}$$

L'Eq. (1.22) est le modèle du filtrage par diffusion non linéaire introduit par (Perona and Malik, 1990). Elle peut être écrite de la façon suivante :

$$\operatorname{div}\left(g\left(\|\nabla u\|\right)\nabla u\right) = \sum_{\ell=1}^{m} \frac{\partial}{\partial x_\ell}\left(g\left(\|\nabla u\|\right)\frac{\partial u}{\partial x_\ell}\right) \tag{1.23}$$

où ℓ est l'indice indiquant la dimension jusqu'à la dimension maximale m. Pour une image $2D$: $m = 2$ et $x_1 = x,\ x_2 = y$. Le schéma semi-implicite s'écrit :

$$u^{k+1} = \frac{1}{m}\sum_{\ell=1}^{m}\left[\mathbf{I} - m\tau\mathbf{A}_\ell\left(u^k\right)\right]^{-1} u^k \tag{1.24}$$

avec $\mathbf{A}_\ell\left(u^k\right) = \frac{\partial}{\partial x_\ell}\left(g\left(\|\nabla u^k\|\right)\frac{\partial}{\partial x_\ell}\right)$, k le numéro de l'itération en cours et $u_0 = u^0$.

Pour obtenir la solution de l'Eq. (1.24) il faut résoudre un système d'équations linéaires par l'algorithme de Thomas (Weickert, 1998).

Dans (Goldenberg, Kimmel, Rivlin and Rudzsky, 2001), les auteurs montrent comment modifier ce modèle pour obtenir une segmentation par contours actifs. Néanmoins, l'application de ces modèles se limite aux problèmes qui peuvent être décrits par l'Eq. (1.22).

1.3.2.3 Remarque sur le traitement des bords

Le fait que les images représentent un support de calcul fini nous oblige à choisir une stratégie de traitement des effets de bord, surtout en ce qui concerne le calcul des différences finies. Evidemment, cette stratégie doit correspondre à l'application donnée et elle ne doit pas influencer l'évolution imposée par la EDP en question.

En général, on remplace les différences centrales par les différences gauches ou droites. Dans le cas d'une évolution plus complexe du front, on recourt aux conditions limites périodiques, c'est à dire que l'on recopie des valeurs de u le long des bords actuels (Sethian, 1996).

Chapitre 2

Applications

Les EDPs s'utilisent pour des images scalaires mais également pour des images vectorielles. Nous présentons ici le classement général des applications des EDP pour des images statiques et des séquences (Suri, Singh and Reden, 2001), et nous montrons quelques exemples des résultats ensuite.

1. Amélioration des images
   (a) Lissage basé sur la diffusion
   (b) Lissage basé sur la diffusion géométrique
   (c) Augmentation de contraste
   (d) Filtres morphologiques
2. Segmentation
   (a) Segmentation par modèles déformables
      i. Avec régulariseurs : la force de propagation est dérivée des données statistiques des régions.
      ii. Sans régulariseurs : l'information des régions n'est pas utilisée.
      iii. Combinée avec un autre modèle.
   (b) Morphologie mathématique : Ligne de Partage des Eaux continue
3. Flux optique
4. Shape from Shading

Le domaine EDP est extrêmement riche en applications. Nous avons effectué une étude des opérateurs-types qui représentent à chaque fois une classe de calculs. Dans les sections suivantes, nous présentons les types correspondant aux deux premiers groupes d'applications mentionnées ci-dessus.

2.1 Amélioration des images

Dans cette section, nous présentons quelques contributions principales d'utilisation des EDPs pour améliorer des images, c'est à dire réduire le bruit et renforcer le contraste (Guichard and Morel, 2001).

2.1.1 Lissage

Lissage étant un outil important pour la vision par ordinateur basée sur l'analyse multi-échelle, il existe de très nombreuses méthodes d'extractions des caractéristiques telles que les crêtes, les vallées ou les jonctions. Le lissage par les méthodes des courbes de niveaux ((Kimia and Siddiqi, 1996), (Lindeberg, 1994)) est considéré par beaucoup d'auteurs comme un choix de l'espace multi-échelle approprié à l'analyse donnée des informations continues dans l'image.

2.1.1.1 Diffusion linéaire

La diffusion linéaire joue un rôle important dans certains détecteurs de contours. Elle sert souvent de modèle dans l'analyse multi-échelle linéaire ((Guichard and Morel, 2001) ou (Lindeberg, 1994)).

L'équation de la diffusion linéaire s'écrit :

$$\begin{cases} \frac{\partial u}{\partial t} = \Delta u \\ u(x, t = 0) = u_0(x) \end{cases} \tag{2.1}$$

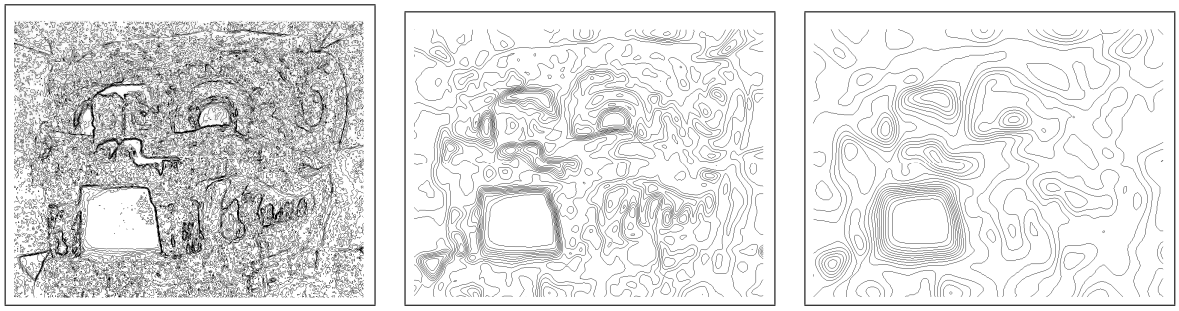
Cette équation respecte le principe de maximum-minimum sur $\mathbb{R}^2 \times [0, \infty)$. L'équivalence avec le lissage par une Gaussienne a été prouvée (Hellwig, 1977). Elle possède la solution suivante :

$$\begin{cases} u(x, t) = u_0(x) \text{ pour } t = 0 \\ u(x, t) = (\mathcal{G}_\sigma * u_0)(x) \text{ pour } t > 0 \text{ avec } \sigma = \sqrt{2t} \end{cases} \tag{2.2}$$

où $\mathcal{G}_\sigma = \frac{1}{\sqrt{2\pi\sigma}} \exp\left(\frac{x^2}{2\sigma^2}\right)$ est une gaussienne d'écart-type σ et $u_0(x)$ est le niveau de gris dans le point x. L'unicité de solution est prouvée si la fonction u est bornée $u(x, t) \leq M \exp(ax^2)$, avec $M > 0$ et $a > 0$.

Le lissage par diffusion linéaire régularise l'image et élimine le bruit. En même temps il introduit également du flou car aucune contrainte sur la préservation des contours n'est imposée. De même, ce type de lissage cause un décalage des contours Fig. 2.1. La préservation et le non-déplacement des contours représentent l'intérêt des modèles non linéaires.

(a) Original (b) $n = 20$ (c) $n = 100$



(d) $n = 0$ (e) n=20 (f) $n = 100$

FIG. 2.1 – Diffusion linéaire. La première ligne montre des images résultantes, la deuxième ligne contient les images des courbes de niveaux (pour chaque 15 niveaux) pour n itérations.

2.1.1.2 Diffusion non linéaire

La diffusion non linéaire a été introduite par Perona and Malik (Perona and Malik, 1988) et ils ont également tout de suite proposé son modèle par l'EDP. En même temps ils ont établi de manière explicite le principe de maximum ayant la même importance que le principe de comparaison en morphologie mathématique en tant que propriété des érosions, dilatations, ouvertures et fermetures (Alvarez et al., 1992), (Serra, 1982). Malik et Perona proposent l'équation suivante :

$$\begin{cases} \frac{\partial u}{\partial t} = \operatorname{div}\left(g\left(\|\nabla u\|\right)\nabla u\right) \\ u(t=0) = u_0 \end{cases} \tag{2.3}$$

ou $g()$ est le terme de diffusion. $g()$ est une fonction régulière, positive et non croissante, avec $g(0) = 1$ et $\lim_{v\to\infty} g(v) = 0$. Cette fonction représente un teste d'atténuation de la diffusivité du filtre dans chaque point. Il est défini comme suit :

$$g(\nabla u) = \exp\left(-\left(\frac{\|\nabla u\|}{k}\right)^2\right) \tag{2.4}$$

k est une constante à déterminer par l'utilisateur. Ce paramètre correspond à la hauteur des contours à préserver.

Le principe est de contrôler la diffusion en fonction des informations locales. Il s'agit d'un filtre qui diminue la diffusion dans les régions à fort gradient (pour préserver des contours) et maintient la diffusion dans les zones à faible gradient. Ce modèle permet ainsi de créer des zones plates dans l'image même dans le cas de faible contraste. Ainsi il diminue la sur-segmentation résultante de l'application des détection des contours.

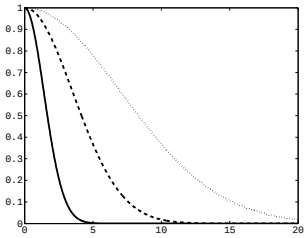
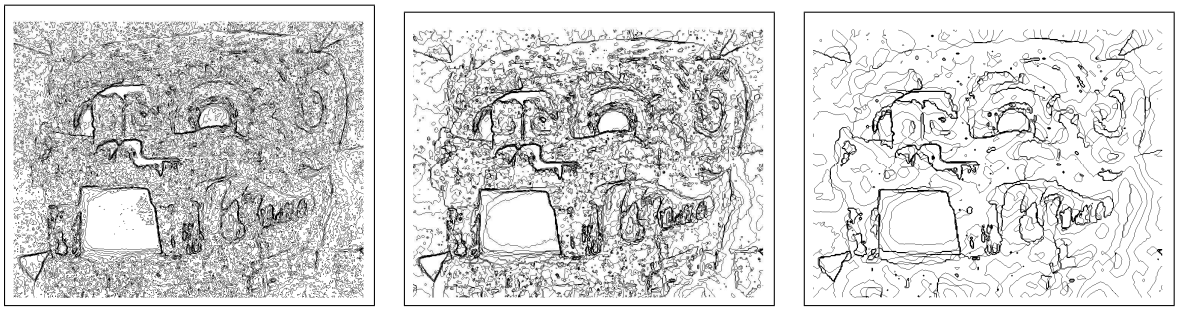


FIG. 2.2 – Fonction $g()$, avec paramètre $k = 2$, $k = 5$ (-), $k = 10$ (:)

(a) Original (b) $n = 100$ et $k = 2$ (c) $n = 100$ et $k = 5$



(d) $n = 0$ (e) $n = 100$ et $k = 2$ (f) $n = 100$ et $k = 5$

FIG. 2.3 – Diffusion non linéaire. La première ligne montre des images résultantes, la deuxième ligne contient les images des courbes de niveaux (pour chaque 15 niveaux) pour n itérations.

Beaucoup de modèles ont été développés, parmi lesquels, nous pouvons citer : (Alvarez et al., 1992), (Osher and Rudin, 1988), (Guichard and Morel, 1995) ou (Weickert, ter Haar Romeny and Viergver, 1998). Nous ne les présentons pas, car ils sont principalement liés à la diffusion non linéaire expliquée ci-dessus. Aussi du point de vue des calculs et de l'implantation hardware elles appartiennent au même type.

2.1.2 Lissage des contours

Le lissage des contours sert à éliminer des effets de discrétisation ou des artefacts d'une segmentation préalable. Ces artefacts sont représentés localement et peuvent fausser une estimation des caractéristique globales. Dans ce but, (Kimmel, Shaked, Kiryati and Bruckstein, 1995) a lissé les contours des objets afin de calculer le squelette à partir des contours lisses. Beaucoup de méthodes sont basées sur un lissage par Gaussienne (Lindeberg, 1994) qui est présentée dans la section 2.1.1.1. Kimia and Siddiqi (1996) ont introduit un opérateur de lissage gouverné par l'équation géométrique de chaleur :

$$\frac{\partial u}{\partial t} = \kappa \|\nabla u\| \tag{2.5}$$

où κ est la courbure de la fonction à lisser.

L'avantage d'utilisation de cette équation par rapport au lissage par une Gaussienne est qu'un lissage plus fort s'applique là où la courbure locale est plus importante. En plus, il a été prouvé que l'évolution des ensembles des courbes de niveaux contrôlées par la courbure converge vers les formes circulaires sans développer des intersections avec soi-même (Kimia and Siddiqi, 1996).

(a) (b)

FIG. 2.4 – Lissage des contours par l'équation géométrique de la chaleur.(a) Objet binaire. (b) Objet binaire après le lissage des contours par l'équation géométrique de la chaleur.

2.2 Morphologie mathématique continue

Typiquement, la morphologie mathématique considère des images binaires et des images en niveau de gris comme des fonctions. Elle travaille avec ces fonctions à l'aide des transformations morphologiques par l'intermédiaire des éléments structurant (Matheron, 1975), (Serra, 1982), (Tek and Kimia, 1998). Comme le prouvent de nombreux exemples dans la littérature, la morphologie mathématique peut être appliquée aux espaces plus abstraits comme des treillis (Serra, 2001), des graphes (Vincent, 1990) ou des espaces topologiques (Roerdink, 1994).

A partir de la classification des méthodes des EDPs, où les transformations morphologiques interviennent dans tous les domaines, nous pouvons voir l'intérêt croissant pour l'interprétation

géométrique des transformations morphologiques, où les opérateurs basiques sont définis sous forme d'équations aux dérivées partielles.

1. *Meilleure modélisation mathématique.* Elle permet de bénéficier des définitions de base de la morphologie mathématique en $\mathbb{R}^n$ ainsi que de leurs propriétés importantes. Ces propriétés sont perturbées dans les cas discrets, par exemple la définition des cercles digitaux avec tous les radius possibles.
2. *Précision sous-pixélique.* Nous obtenons la possibilité d'utiliser les transformations de distance les plus précises. La représentation des contours des objets et des frontières des régions est accessible.
3. *Indépendance de la définition de la maille de discrétisation.* Elle est en partie liée à la modélisation mathématique. En plus, nous pouvons utiliser des éléments structurants dont la forme ne dépend plus de la maille. Nous ne sommes plus obligés de faire un choix entre la maille carrée et la maille hexagonale. Dès maintenant, nous ne faisons que le choix de la précision du schéma numérique.
4. *Extension directe en* $3D$. L'équation d'évolution reste la même, il nous suffit d'ajouter la troisième coordonnée afin d'obtenir la formulation en dimensions plus élevées.
5. *Possible introduction d'une information sur la vitesse de propagation.* Il est également plus facile d'agir sur la vitesse d'évolution. Nous pouvons ajouter des critères basés sur les informations locales et ainsi créer des opérateurs morphologiques déformables.

Le principe de la morphologie continue part du raisonnement où l'on suppose que la frontière (ou contour) de la forme transformée est le résultat d'un processus de déformation d'une courbe représentant la forme originale. Or, cette déformation (transformation) d'une courbe à une autre est le résultat d'une séquence des déformations infinitésimales qui ne dépend que localement de la forme originale (Tek and Kimia, 1998), à condition d'utiliser un éléments structurant convexe. Ainsi, toute évolution de courbe est formulée à l'aide de l'équation suivante :

$$\begin{cases} \frac{\partial \mathcal{C}}{\partial t} = \mathcal{F}\vec{\mathcal{N}} \\ \mathcal{C}(t=0) = \mathcal{C}_0 \end{cases} \tag{2.6}$$

où $\mathcal{C}$ est une courbe qui se déforme. On considère une évolution de courbe avec la vélocité $\mathcal{F}\vec{\mathcal{N}}$. La vélocité $\mathcal{F}\vec{\mathcal{N}}$ dépend de l'élément structurant (Sapiro, 2000). Imaginons un objet binaire avec les contours représentés par une courbe $\mathcal{C}_0$. Si on dilate cet objet avec un disque de rayon δt, le résultat sera une famille de courbes matérialisant l'évolution selon Eq. (2.6) avec $\mathcal{F} = 1$.

Plusieurs modèles basés sur des méthodes des courbes de niveaux des transformations morphologiques ont été proposés afin de surmonter les difficultés du calcul selon l'Eq. (2.6). Osher and Sethian (1988) proposent de décrire les contours des formes par fonction distance et

de faire évoluer ses courbes de niveau (voir section 1.2.1). Pour le suivant, nous allons utiliser des EDPs qui ont été développées par Brockett and Maragos (1992).

Il a été montré que des courbes de niveaux évoluant sous l'équation Eq. (2.6) sont équivalentes aux courbes de niveau de toute image dont l'évolution est contrôlée par l'équation suivante :

$$\begin{cases} \frac{\partial u}{\partial t} = \pm s(I)\|\nabla u\| \\ u(t=0) = u_0 \end{cases} \tag{2.7}$$

avec la fonction s en tant que fonction structurante qui peut être une fonction de l'image d'entrée I.

2.2.1 Opérateurs de base

Toute opération morphologique est générée par dilatations δ et érosions ε. Les équations suivantes donnent les formules de ces opérateurs basiques sur une fonction u, $s()$ est une fonction d'élément structurant (Serra, 2001), .

$$\delta(x,y) = (u \oplus s)(x,y) = \sup_{a,b} \{u(x-a, y-b) + s(a,b)\} \tag{2.8}$$

$$\varepsilon(x,y) = (u \ominus s)(x,y) = \inf_{a,b} \{u(x-a, y-b) - s(a,b)\} \tag{2.9}$$

Pour l'analyse multi-échelle, ces équations sont réécrites sous forme (Maragos and Meyer, 1999) :

$$\delta(x,y,t) = \left(u \oplus s^{(t)}\right)(x,y) = \sup_{a,b} \left\{ u(x-a, y-b) + ts\left(\frac{a}{t}, \frac{b}{t}\right) \right\},\ t > 0 \tag{2.10}$$

$$\varepsilon(x,y,t) = \left(u \ominus s^{(t)}\right)(x,y) = \inf_{a,b} \left\{ u(x-a, y-b) - ts\left(\frac{a}{t}, \frac{b}{t}\right) \right\},\ t > 0 \tag{2.11}$$

où $\delta(x,y,0) = u(x,y)$. Ces opérateurs sont obtenus en remplaçant l'élément structurant $s()$ à noyau unitaire dans les formules classiques (2.9,2.8) par l'élément structurant multi-échelle $s^{(t)}()$. Si la fonction structurante $s()$ est une fonction continue sur $\mathbb{R}^n$ l'image dans l'espace multi-échelle est continue (van den Boomgaard and Smeulders, 1994), (Jackway and Deriche, 1996), (Brockett and Maragos, 1992).

2.2.1.1 Les EDP d'érosion et de dilatation

Nous montrons quelque exemples des équations aux dérivées partielles qui créent les dilatations et les érosions multi-échelle continues pour différentes formes des éléments structurants. Il faut noter qu'il s'agit des EDPs simples mais non linéaires. Cela signifie que même si la

fonction initiale u est lisse, des discontinuités des dérivées peuvent apparaître lors de l'évolution. Par conséquent, les érosions et dilatations multi-échelle sont des solutions faibles des équations relatives (Schüpp, 2000).

- Disque

$$Dilatation : \frac{\partial u}{\partial t} = \quad \|\nabla u\| \tag{2.12}$$

$$Erosion : \frac{\partial u}{\partial t} = -\|\nabla u\| \tag{2.13}$$

- Carré

$$Dilatation : \frac{\partial u}{\partial t} = \quad \left(\left|\frac{\partial u}{\partial x}\right| + \left|\frac{\partial u}{\partial y}\right|\right) \tag{2.14}$$

$$Erosion : \frac{\partial u}{\partial t} = -\left(\left|\frac{\partial u}{\partial x}\right| + \left|\frac{\partial u}{\partial y}\right|\right) \tag{2.15}$$

- Losange

$$Dilatation : \frac{\partial u}{\partial t} = \quad \max\left\{\left|\frac{\partial u}{\partial x}\right|, \left|\frac{\partial u}{\partial y}\right|\right\} \tag{2.16}$$

$$Erosion : \frac{\partial u}{\partial t} = -\max\left\{\left|\frac{\partial u}{\partial x}\right|, \left|\frac{\partial u}{\partial y}\right|\right\} \tag{2.17}$$

- Elément linéaire

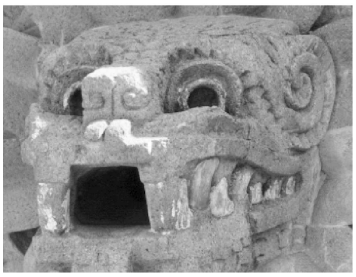
$$Dilatation : \frac{\partial u}{\partial t} = \quad \|\nabla u\||\cos\alpha|, \alpha = \arctan\left(\frac{\frac{\partial u}{\partial x}}{\frac{\partial u}{\partial y}}\right) - \theta \tag{2.18}$$

$$Erosion : \frac{\partial u}{\partial t} = -\|\nabla u\||\cos\alpha|, \alpha = \arctan\left(\frac{\frac{\partial u}{\partial x}}{\frac{\partial u}{\partial y}}\right) - \theta \tag{2.19}$$

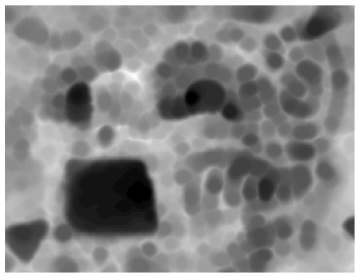
Etant donné que les érosions et dilatations par un élément structurant plat commutent avec l'opération de seuillage, les EDPs mentionnées ci-dessus s'appliquent aussi directement aux images binaires. Quand une image à niveau de gris est dilatée, chaque image binaire issue du seuillage est simultanément dilatée par le même élément structurant plat à la même échelle. En revanche, ceci n'est plus vrai dans le cas des fonctions structurantes à niveau de gris (Maragos and Meyer, 1999).

2.2.1.2 Erosions et dilatations "déformables"

L'un des avantages intéressants de la formulation continue est de pouvoir y intégrer des contraintes sur la vitesse d'évolution des courbes de niveaux. Nous obtenons ainsi une sorte d'élément structurant déformable en fonction de la définition de $s(I)$.

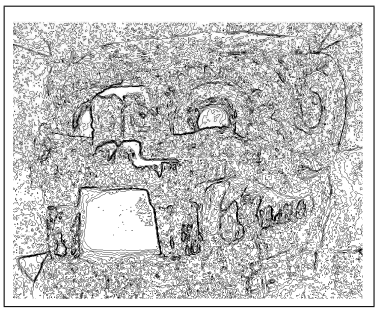


(a) Image originale

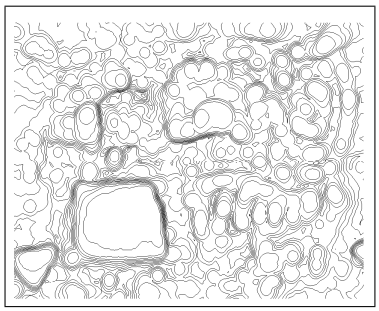


(b) Erosion continue par un disque $n = 200$

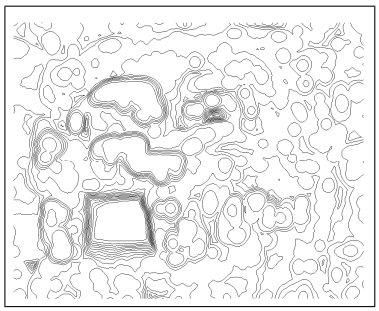
(c) Dilatation continue par un disque $n = 200$



(d) Courbes de niveaux de l'image originale

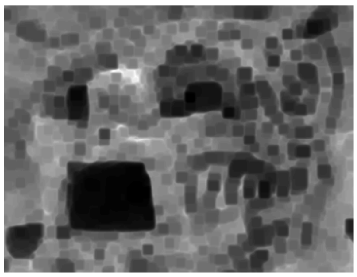


(e) Courbes de niveaux de l'érosion continue par un disque



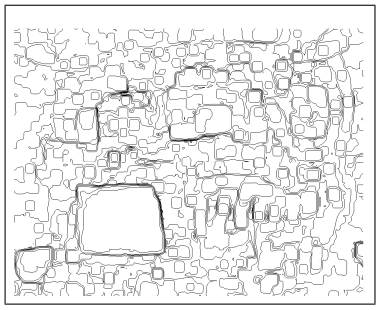
(f) Courbes de niveaux de la dilatation continue par un disque

FIG. 2.5 – Résultats de l'érosion et dilatation continue, $\tau = 0.2$

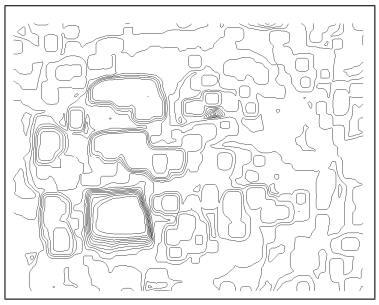


(a) Erosion

(b) Dilation



(c) Courbes de niveaux de l'érosion par un carré



(d) Courbes de niveaux de la dilatation par un carré

FIG. 2.6 – Résultats de l'érosion et de la dilatation continue, $\tau = 0.2$

On peut citer des exemples de (Schüpp, 2000) :

- $s(I) = \frac{1}{\|\nabla I\|}$: l'évolution est freinée si le gradient est plus important
- $s(I) = \frac{I-\hat{I}}{\max I}$: $\hat{I}$ représente diverses informations de l'image I, on peut ralentir l'évolution dans les zones sombres ou brillantes.

2.2.2 Reconstruction des images

La reconstruction est un opérateur encore plus performant que les ouvertures ou fermetures (suites des érosions et dilatations). La reconstruction commence avec une image de référence I et une image de marqueur $I_{\mathcal{M}}$. La reconstruction permet de reconstruire l'image en préservant les frontières des objets. Elle élargit les zones plates dans l'image et la simplifie. A titre d'exemple, nous présentons les opérateurs de nivellement continu et de swamping. D'autres exemples de la reconstruction continue peuvent être trouvés dans (Schüpp, 2000).

2.2.2.1 Nivellement continu

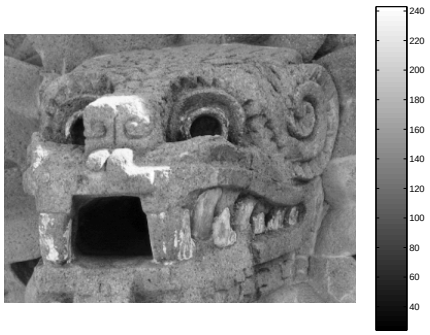
L'un des filtres morphologiques fort qui peut également représenter l'espace multi-échelle est le nivellement. Parmi ses propriétés importantes, on trouve (Meyer and Maragos, 1999a) :

- L'invariance spatiale (invariance par translation)
- L'isotropie (invariance par rotation)
- L'invariance par changement d'illumination

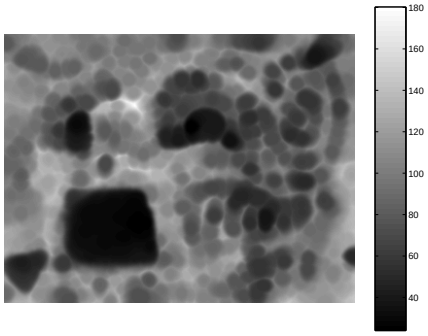
La formulation continue de nivellement et des opérateurs de reconstruction ajoutent, en plus de tous les avantages des EDP, l'avantage de pouvoir arrêter l'évolution avant d'atteindre la convergence tout en préservant l'isotropie de l'évolution. Cette possibilité peut être valorisante pour certaines applications (Maragos and Meyer, 1999) car la modification du critère d'arrêt s'effectue par un simple changement du générateur de l'EDP en question, sans compliquer l'algorithme. Nous présentons L'EDP qui permet de générer l'opérateur de nivellement classique.

$$\begin{cases} \frac{\partial u}{\partial t} = \text{sign}\,(I-u)\,\|\nabla u\| \\ u(t=0) = I_{\mathcal{M}} \end{cases} \quad \text{avec sign}(I-u) = \begin{cases} -1 \text{ si } (I-u) < 0 \\ 0 \text{ si } (I-u) = 0 \\ 1 \text{ si } (I-u) > 0 \end{cases} \tag{2.20}$$

Cette équation procède par la déformation du marqueur $I_{\mathcal{M}}$ qui est contrôlée par la référence I. La différence entre l'évolution est la référence qui détermine le type de cette déformation : expansion ou contraction. Ces déformations sont réalisées par les érosions et dilatations multi-échelle.

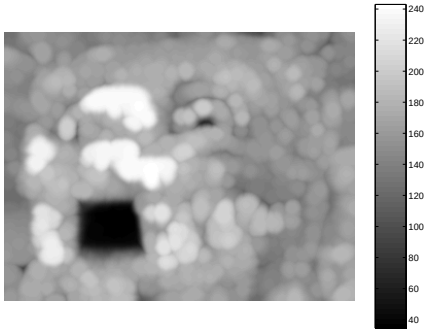


(a) Image originale



(b) Marqueur obtenu par l'érosion continue de l'image 2.7(a)

(c) Résultat du nivellement continu obtenu avec le marqueur 2.7(b)



(d) Marqueur obtenu par la dilatation continue de l'image 2.7(a)

(e) Résultat du nivellement continu obtenu avec le marqueur 2.7(d)

FIG. 2.7 – Exemples des nivellements continus, $\tau = 0.2$

Remarque : zones quasi-plates. Il est également possible de créer des zones plates avec une pente constante linéaire ou avec une surface parabolique. Ceci est fait par remplacement de l'élément structurant plat par un générateur parabolique des érosions, dilatations multi-échelle (Maragos and Meyer, 1999) :

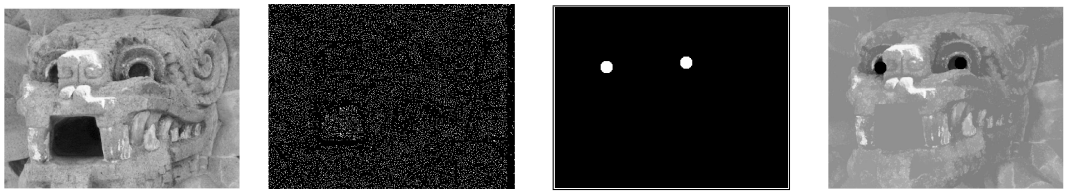
$$\begin{cases} \frac{\partial u}{\partial t} = \frac{1}{2}\text{sign}\,(I-u)\,\|\nabla u\|^2 \\ u(x,y,t=0) = u^0 = I_{\mathcal{M}} \end{cases} \tag{2.21}$$

2.2.2.2 Swamping

Le swamping est un autre exemple de reconstruction d'image. La surface topographique est modifiée de façon que les marqueurs deviennent les seuls minima régionaux de l'image. Cette opération s'appelle *swamping*. Les marqueurs sont des images binaire, avec valeur minimale à l'intérieur des marqueurs et maximale à l'extérieur. La surface topographique est modifiée de façon que la valeur minimale est attribuée à tous les minima régionaux. Après, on procède à la fermeture par reconstruction de l'image à partir de la fonction marqueur $I_{\mathcal{M}}$ (Serra, 2001) :

$$\begin{cases} u^{n+1} = \varepsilon(u^n) \vee I \\ u^0 = I_{\mathcal{M}} \vee I \end{cases} \tag{2.22}$$

La fonction u^∞ est similaire à l'image de départ à l'exception que ses minima sont des marqueurs.



(a) (b) (c) (d)

FIG. 2.8 – Exemple de swamping pour une image des marqueurs choisis manuellement.(a) Image originale. (b) Minima régionaux de l'image originale. (c)Marqueurs choisis manuellement, en tant que minima régionaux à imposer. (d) Résultat de swamping continu.

2.2.3 Fonction distance

La distance entre deux points p_1 et p_2 d'un ensemble $X \in \mathbb{Z}^2$ est définie comme la longueur minimale du chemin entre les deux points appartenant à X (Serra, 2001).

$$\text{dist}(p_1,p_2) = \min\left\{l\left(C_{p_1,p_2}\right)\right\} \tag{2.23}$$

avec C_{p_1,p_2} représentant le chemin entre p_1 et p_2 et l la longueur du chemin C.

Si l'image est considérée comme un signal continu, l'expression de la distance prend la forme suivante :

$$\text{dist}(p_1, p_2) = \inf \int_{p_1}^{p_2} C(s)ds \tag{2.24}$$

où s est l'abscisse curviligne.

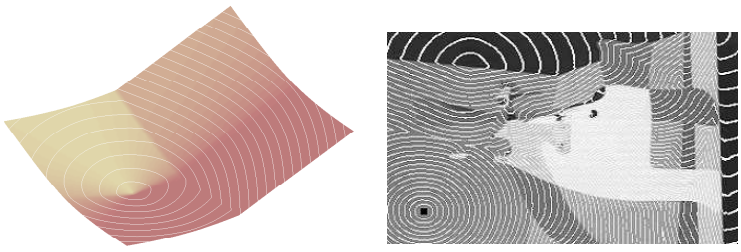
Cette expression peut être généralisée pour le calcul d'une fonction distance pondérée. Considérons une fonction de coût (ou de poids) $\mathcal{P} : \mathbb{Z}^2 \rightarrow \mathbb{R}^+$, la fonction distance pondérée est définie comme la longueur du chemin entre deux points p_1, p_2 qui minimise le coût :

$$\text{dist}_{\mathcal{P}}(p_1, p_2) = \min \int_{p_1}^{p_2} \mathcal{P}(C(s))ds \tag{2.25}$$

Dans (Schüpp, 2000) il a été montré que le résultat satisfait la formulation stationnaire des méthodes des courbes de niveaux, l'équation Eikonale :

$$\begin{cases} \|\nabla u\| = \mathcal{P} \\ u_0 = \mathcal{C}_0 \end{cases} \tag{2.26}$$

Les conditions initiales limites u^0 sont des contours $\mathcal{C}_0$ des sources de la fonction distance. En même temps, les contours des sources de la fonction distance sont des courbes de niveaux zéro. Comme il a été mentionné auparavant, les valeurs ainsi obtenues peuvent être interprétées comme les temps de passage des ondes propagées à partir des sources.



(a) (b)

FIG. 2.9 – Résultats du calcul de la fonction distance pondérée. (a) Distance pondérée calculée à partir d'une source unique et deux régions des vitesses différentes. (b) Courbes de niveaux de la fonction distance (en blanc) pondérée par niveau de gris de l'image originale.

Recherche du plus court chemin

Une des applications directes de la fonction distance est le calcul du plus court chemin sur une surface. Le principe est le suivant : après avoir obtenu la fonction distance, on effectue une descente sur le gradient du point p_2 jusqu'à atteindre le point p_1 (il est possible de tracer

plusieurs chemins en même temps). Dans le cadre des méthodes basées sur les EDPs, il a été proposé par (Cohen and Kimmel, 1997) et repris dans (Kimmel, 1995) et (Sethian, 1996) où les auteurs montrent comment utiliser ces outils dans la navigation des robots. Le principe est d'attribuer un poids très élevé aux obstacles (murs ou objets à éviter) et de calculer le chemin le plus court (optimal en termes de variation de la surface).

Dans le domaine de la morphologie mathématique classique, ce principe a été proposé dans (Jeulin, Osmont and Larquetoux, 1987) où la distance minimale est cherchée sur les graphes. Plus tard, (Vincent and Jeulin, 1989) et (Jeulin, Vincent and Serpe, 1990) utilisent le calcul de la distance minimale entre des points donnés pour la détection des fissures des matériaux.

La recherche du plus court chemin peut également être employée pour l'extraction d'objets comme il a été montré dans (Schüpp, 2000). Une autre application intéressante est présentée dans (Deschamps and Cohen, 2000) où la recherche des plus courts chemins est employée pour l'endoscopie virtuelle.

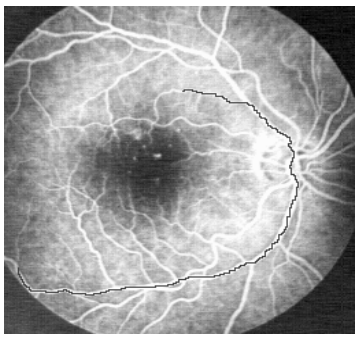


FIG. 2.10 – Recherche du plus court chemin (en termes de variations minimales de la surface) entre deux points choisis arbitrairement dans l'image de la rétine.

2.2.4 Segmentation

En théorie, la segmentation des images consiste en un isolement des objets de l'arrière plan. En pratique on crée une partition de l'image en régions disjointes tout en respectant certaines propriétés de ces régions comme la texture, le niveau de gris, etc...

Nous avons précédemment discuté le calcul de la fonction distance. Elle représente un outil efficace pour la construction des algorithmes de segmentation car la procédure de segmentation peut être regardée comme la détection des lignes de partage des eaux de la fonction distance.

2.2.4.1 Ligne de partage des eaux

La ligne de partage des eaux (LPE) est une approche de segmentation basée sur des régions, initialement elle a été proposée par (Digabel and Lantuéjoul, 1977) et ensuite améliorée par (Beucher and Lantuéjoul, 1979). Deux interprétations intuitives de la LPE sont :

- La surface topographique (vue comme un paysage) est inondée par la pluie et la surface est divisée en régions d'attraction de l'eau.
- La surface est inondée à partir des minima locaux considérés comme les sources (puits). Là où les eaux des différents bassins versants se rencontrent, les barrages sont construits.

Le processus de l'algorithme LPE résulte en :

- Des zones d'influence associées à chaque minimum de l'image.
- Un ensemble des lignes de partage des eaux qui séparent les zones d'influence.

La grande variété des définitions des chemins par lesquels les gouttes d'eau tombent dans des bassins versants a pour conséquence de nombreuses définitions formelles de l'algorithme de la LPE. A titre d'exemple et pour comparer avec la formulation continue, nous présentons dans la suite quelques définitions dites classiques de la LPE. La discussion plus détaillée peut être trouvée dans (Najman and Couprie, 2003) ou (Roerdink and Meijster, 2000).

D'abord, nous présentons la définition des zones d'influence géodésique. Les zones d'influence géodésique sont basées sur la notion de la distance géodésique.

Soit $A \subseteq \mathcal{E}$ avec $\mathcal{E} = \mathbb{R}^d$ ou $\mathcal{E} = \mathbb{Z}^d$ et soient a, b deux points appartenant à A. La distance géodésique $d_A(a, b)$ entre a et b est le chemin minimal de tous les chemins de a à b appartenant à A. Considérons B comme un sous-ensemble de A, nous pouvons définir ensuite : $d_A(a, B) = \min_{b \in B}(d_A(a, b))$. Soit $B \subseteq A$ qui est partitionné en k composantes connexes $B_i,\ i = 1, \ldots, k$. La zone d'influence géodésiques de l'ensemble B_i dans A est définie :

$$iz_A(B_i) = \{p \in A | \forall j \in [1 \ldots k] \backslash \{i\} : d_A(p, B_i) < d_A(p, B_j)\} \tag{2.27}$$

L'union des zones d'influence des composantes connexes de B :

$$IZ_A(B) = \bigcup_{i=1}^{k} iz_A(B_i) \tag{2.28}$$

et le squelette par zones d'influence :

$$SKIZ_A(B) = A \backslash IZ_A(B) \tag{2.29}$$

Le $SKIZ$ est l'ensemble des points équidistants à (au moins deux) composantes les plus proches. Pour $E = \mathbb{Z}^d$, cet ensemble peut être vide.

Définition de LPE discrète I. : algorithmique Cette définition algorithmique fréquemment utilisée a été introduite par (Vincent and Soille, 1991). Considérons une image à niveau de gris $I : (D \subseteq \mathbb{Z}^2) \rightarrow \mathbb{N}$ avec des valeurs minimales et maximales h_{min}, h_{max}. Nous définissons l'opération de seuillage sur I à niveau h comme suit :

$$T_h = \{p \in D | I(p) \leq h\} \tag{2.30}$$

X_h est l'union de l'ensemble des bassins versants calculés sur le niveau h. Une composante connexe du seuil T_{h+1} au niveau $h+1$ est soit un nouveau minimum soit une extension du bassin X_h. Dans ce dernier cas, on calcul la zone d'influence géodésique de X_h sur T_{h+1} qui résulte en une mise à jour de X_{h+1}.

$$\begin{cases} X_{h_{min}} = \{p \in D | I(p) = h_{min}\} = T_{h_{min}} \\ X_{h+1} = \text{MIN}_{h+1} \cup \text{IZ}_{T_{h+1}}(X_h), \quad h \in [h_{min}, h_{max}] \end{cases} \tag{2.31}$$

IZ est la zone d'influence calculée pour le niveau correspondant. MIN_{h+1} est l'union de tous les minima avec l'altitude $h+1$.

La ligne de partage des eaux $LPE(I)$ est le complément de $X_{h_{max}}$ sur D :

$$LPE(I) = D \backslash X_{h_{max}} \tag{2.32}$$

Définition de la LPE discrète II, Meyer (1991) : part d'une image à niveau de gris I et d'un ensemble de marqueurs M qui correspondent au minima de l'image I. Par la suite, on procède par l'expansion des marqueurs en préservant le nombre des composantes connexes de M.

1. Insérer chaque voisin x de chaque région marquée dans une file d'attente hiérarchique où la priorité est inversement proportionnelle à la valeur $I(x)$. Aucun point ne peut être inséré deux fois.
2. Extraire le point avec la priorité maximale (la plus petite valeur) de la queue hiérarchique. Si tous les voisins du point ont la même étiquette, marquer ce point par cette étiquette. S'il existe des voisins non marqués, les insérer dans la file. Répéter jusqu'à ce que la file d'attente soit vide.

Les lignes de partage des eaux sont ensuite obtenues comme le complément des points marqués.

Définition de la LPE discrète III. : distance topographique Pour cette définition, nous partons des travaux publiés dans (Meyer, 1994). Considérons une image à niveau de gris I et supposons que tous les points qui ne sont pas des minima locaux ont un voisin

d'une valeur inférieure. La distance topographique entre les points p_0 et p_ℓ le long du chemin $C = (p_0, \ldots, p_\ell)$:

$$\mathrm{Tdist}_I(p_0, p_\ell) = \min_{C \in \mathcal{C}(p_0, p_\ell)} \sum_{i=0}^{\ell-1} \mathcal{P}(p_i, p_{i+1}) \tag{2.33}$$

où $\mathcal{C}(p_0, p_\ell)$ désigne l'ensemble de tous les chemins de p_0 à p_ℓ. $\mathcal{P}$ représente une fonction du coût (du poids) pour aller d'un point vers les voisins. Ce coût peut être interprété comme la valeur absolue de la différence des altitudes de deux points (Roerdink and Meijster, 2000). La distance topographique entre un point p et un ensemble A est définie comme : $T\mathrm{dist}_I(p, A) = \min_{a \in A} T\mathrm{dist}_I(p, a)$.

Pour ce type d'algorithme de LPE, il faut traiter correctement les plateaux dans l'image. La solution est de calculer la distance géodésique à la frontière inférieure du plateau (Roerdink and Meijster, 2000). Appelons une image ainsi créée I^*.

Rappelons que D est un support sur lequel cette image est définie, et que D est muni d'une connéxité. L'ensemble $M = \{m_k\}_{k \in K}$ contient les minima de l'image I. Le bassin versant $CB(m_k)$ correspondant au minimum m_k est ensuite obtenu d'après l'expression :

$$CB(m_k) = \{p \in D | \forall j \in I \backslash \{k\} : I^*(m_k) < I^*(m_j) + T\mathrm{dist}_{I^*}(p, m_j)\} \tag{2.34}$$

La LPE est l'ensemble des points qui n'appartient à aucun bassin versant :

$$LPE(I) = D \cap \left(\bigcup_{k \in K} CB(m_k) \right)^c \tag{2.35}$$

Définition de la LPE continue La définition de la LPE continue est basée sur le calcul de la fonction distance. Même dans le cas continu, la définition de la LPE dépend de la définition de la fonction distance. Dans ce document, nous nous basons sur des publications (Meyer, 1994) et (Najman and Schmitt, 1994).

Supposons que l'image I soit une fonction continue réelle deux fois différentiable sur le domaine de l'image. La distance topographique entre deux points p_0 et p_ℓ est définie :

$$T\mathrm{dist}_I(p_0, p_\ell) = \inf_{C \in \mathcal{C}(p_0, p_\ell)} \int_C \|\nabla I(\wp(s))\| ds \tag{2.36}$$

où $\mathcal{C}$ est l'ensemble des chemins C de p_0 à p_ℓ.

Le bassin versant $CB(m_k)$ du minimum m_k est défini comme l'ensemble des points qui sont topographiquement plus proches de m_k que d'un autre minimum m_j :

$$CB(m_k) = \{x \in D | \forall j \in K \backslash \{k\} : I(m_k) + T\mathrm{dist}_I(x, m_k) < I(m_j) + T\mathrm{dist}_I(x, m_j)\} \tag{2.37}$$

LPE :

$$LPE(I) = D \cap \left(\bigcup_{k \in K} CB(m_k) \right)^c \tag{2.38}$$

Pour une étiquette $W \notin K$. Le transformé par la LPE est un mapping : $\lambda : D \to K \cup \{W\}$ tel que $\lambda(p) = i$ si $p \in CB(m_k)$ et $\lambda(p) = W$ si $p \in LPE(I)$.

Dans le cas où tous les minima ne sont pas les sources de la segmentation, il existe deux solutions : soit on considère les marqueurs comme des sources, par conséquent les bassins versants sont inondés à partir des bassin versant voisins déjà inondés (Beucher and Meyer, 1993) ; soit on modifie la surface topographique par swamping (voir la section 2.2.2.2).

En formulation continue, la distance topographique le long d'un chemin devient l'integrale curviligne de $\|\nabla I\|$ le long de ce chemin (Meyer and Maragos, 1999b). Ce raisonnement mène à l'équation aux dérivées partielles dont le résultat est la fonction distance pondérée :

$$\|\nabla u(x, y)\| = \mathcal{P}(x, y) \tag{2.39}$$

Nous retrouvons l'équation Eikonale, le cas stationnaire de la formulation générale des ensembles de niveaux (voir section 1.1).

Si $\mathcal{P}(I) = \frac{1}{\|\nabla I\|}$ et si les courbes initiales coïncident avec les contours des minima régionaux de $\|\nabla I\|$, Eq. (2.39) devient l'équivalente de la LPE classique définie comme un squelette de zones d'influence de tous les minima régionaux relativement à une distance pondérée dépendante de $\|\nabla I\|$ (Najman and Schmitt, 1994). Si les sources diffèrent des minima régionaux, il faut utiliser l'approche de swamping afin de les imposer (voir section 2.2.2.2).

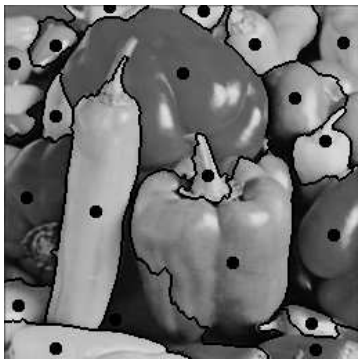


FIG. 2.11 – Exemple de la segmentation morphologique continue. La LPE a été calculée à partir des marqueurs imposés manuellement.

L'avantage de la LPE continue est une inondation plus isotrope par rapport aux méthodes classiques (Beucher, 2002). En revanche le défi est de disposer des algorithmes efficaces de calcul de la fonction distance pondérée avec une précision sous-pixélique. Dans la section 3

nous présentons les algorithmes le plus souvent utilisés et ensuite nous allons proposer un nouvel algorithme parallèle.

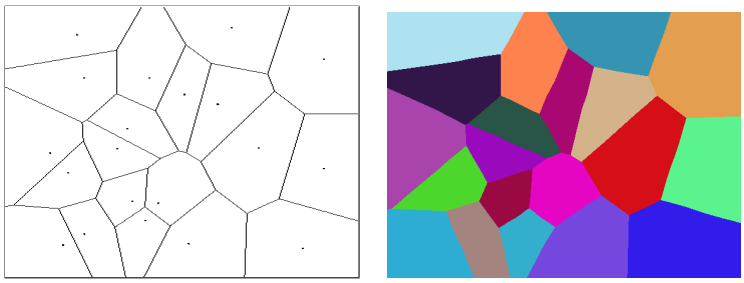
2.2.4.2 Diagramme de Voronoï

Un autre représentant des opérateurs de la partition de l'espace par le calcul des zones d'influence est le diagramme de Voronoï. Le diagramme de Voronoï réalise une partition du plan basée sur la distance euclidienne entre les points $p_i \in \mathbb{R}^2$ de la façon suivante :

$$\text{Diagramme}_V(p_i) = \left\{ p \in \mathbb{R}^2 | \, \text{dist}(p, p_i) < \text{dist}(p, p_j), \forall j \neq i \right\} \tag{2.40}$$

Cette partition peut être construite à partir de l'équation Eikonale (Montanvert and Chassery, 1991). Cet exemple des zones d'influence du diagramme de Voronoï obtenues par le calcul de l'équation Eikonale est montré sur Fig. (2.12).

$$\begin{cases} \|\nabla u\| = 1 \\ u(t=0) = u_0 \end{cases} \tag{2.41}$$



(a) Source et le résultat exact

(b) Résultat par la fonction distance

FIG. 2.12 – Diagramme de Voronoï

2.3 Contours actifs

Cette méthode est utilisée notamment pour la segmentation des images et la recherche des contours des objets. La segmentation commence par la définition du contour initial qui est ensuite déformé afin qu'il corresponde au mieux à la frontière de l'objet recherché.

L'équation de propagation est la suivante :

$$\begin{cases} \frac{\partial \mathcal{C}}{\partial t} = \mathcal{F}\vec{\mathcal{N}} \\ \mathcal{C}(p, t=0) = \mathcal{C}_0 \end{cases} \tag{2.42}$$

La courbe $\mathcal{C}(p,t)$ représentant le contour déformé est vue comme le front d'onde qui se propage dans sa direction normale $\vec{\mathcal{N}}$ avec une vitesse $\mathcal{F}$. La vitesse peut être une fonction de la courbure, de l'intensité, de la texture pour les méthodes contrôlées par les données ou par un modèle de l'objet. L'art consiste à savoir traduire le but de la segmentation en équation qui contrôle la déformation du contour initial

La déformation du contour est une réponse à l'action des forces diverses. Les forces agissent selon un critère d'optimisation :

- Minimisation d'énergie : le contour est déformé afin que le contour résultant corresponde au minimum de l'énergie du contour. (présenté dans (Kass, Witkin and Terzopoulos, 1987) et connu également sous le terme "snakes")
- Point d'équilibre : le contour évolue afin de trouver son équilibre ce qui est une autre façon de chercher un minimum local.

Le modèle classique des contours actifs présente quelques inconvénients telles qu'une grande sensibilité au conditions initiales, une dépendance de la paramétrisation et une extension difficile en $3D$. Cohen (1991) a introduit une force de gonflage qui permet au modèle de pallier le premier problème.

Les algorithmes des contours actifs classiques rencontrent des problèmes liés aux difficultés de la stabilité numérique et à la gestion des changements de topologie des contours.

Ces derniers problèmes sont palliés par l'utilisation de la description implicite des contours où le contour est construit en tant que l'ensemble de niveau zéro de la fonction distance (section 1.2.1).

$$\frac{\partial u}{\partial t} = -\mathcal{F}\|\nabla u\| \tag{2.43}$$

avec

$$u(x, t=0) = \begin{cases} \text{dist}(x, \mathcal{C}_0) & \text{si } x \in \text{ à l'intérieur } (\mathcal{C}_0) \\ 0 & \text{si } x \in \mathcal{C}_0 \\ -\text{dist}(x, \mathcal{C}_0) & \text{si } x \in \text{ à l'extérieur } (\mathcal{C}_0) \end{cases} \tag{2.44}$$

Les contributions principales sont dues à Casselles, Catte, Coll and Dibos (1993) et Malladi and Sethian (1996). Un grand nombre de méthodes de calcul de $\mathcal{F}$ a été développé afin d'améliorer la qualité de la segmentation et sa robustesse ainsi qu'un large choix des schémas numériques pour accélérer la convergence.

Comme un exemple des contours actifs par méthodes des courbes de niveaux, nous pouvons citer **le modèle géométrique** introduit par Casselles et al. (1993). L'équation d'évolution

est la suivante :

$$\begin{cases} \frac{\partial u}{\partial t} = g(I)\left[v + \operatorname{div}\left(\frac{\nabla u}{\|\nabla u\|}\right)\right]\|\nabla u\| \\ u(t=0) = u_0 \end{cases} \tag{2.45}$$

Le terme v agit comme une force externe qui fait évoluer la courbe vers l'intérieur ou l'extérieur. Il permet aux courbes convexes de devenir non-convexes. $\operatorname{div}\frac{\nabla u}{\|\nabla u\|} = \kappa$ est la courbure de la courbe et préserve la courbe lisse le long de l'évolution (terme de viscosité). Fonction $g(I)$ sert à arrêter l'évolution, sa forme la plus utilisée est : $g(I) = \frac{1}{1+\|\nabla I\|^2}$.

Une autre approche a été proposée par (Kichissammany, Kumar, Olver, Tannenbaum and Yezzi, 1995), elle est basée sur la minimisation de la longueur du contour. Ce modèle est appelé **modèle géométrique géodésique**. L'équation du modèle formulé par les méthodes des courbes de niveaux :

$$\begin{cases} \frac{\partial u}{\partial t} = g(I)\,(v + \kappa)\,\|\nabla u\| + \nabla g \nabla u \\ u(t=0) = u_0 \end{cases} \tag{2.46}$$

Le dernier terme de cette équation permet d'attirer ou de repousser le contour vers les objets à détecter. Le choix du critère de convergence est déterminant pour arrêter l'évolution. Le choix de $g(I)$ présenté ci-dessus est suffisant pour des objets contrastés ; cependant il peut être insuffisant dans le cas des caractéristiques internes de l'objet comme par exemple la texture.

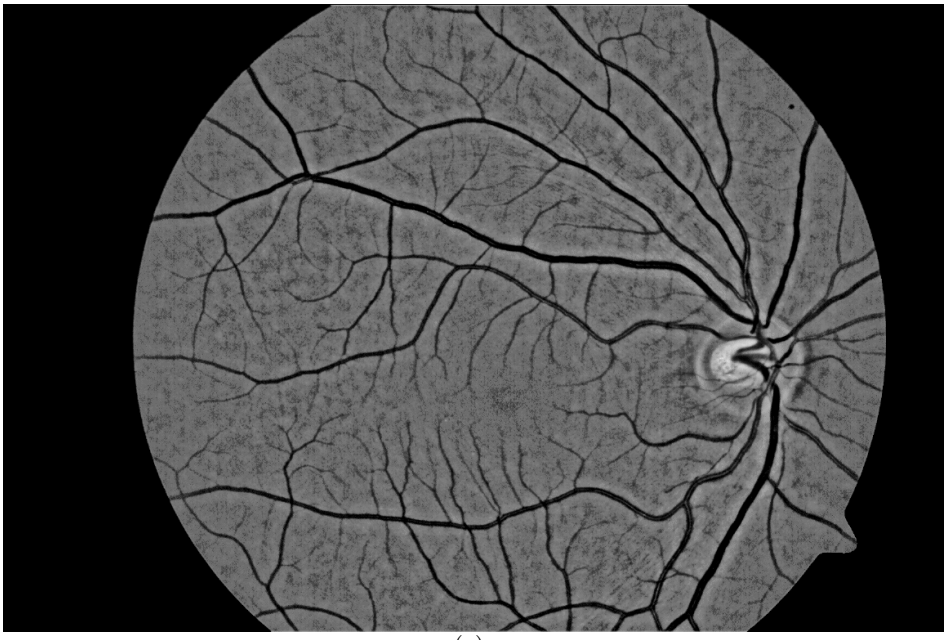
Les deux modèles précédents peuvent être unis par une seule équation (Weickert and Kühne, 2003) :

$$\begin{cases} \frac{\partial u}{\partial t} = a(x)\|\nabla u\| \operatorname{div}\left(\frac{b(x)}{\|\nabla u\|}\nabla u\right) + \|\nabla u\| k g(x) \\ u(t=0) = u_0 \end{cases} \tag{2.47}$$

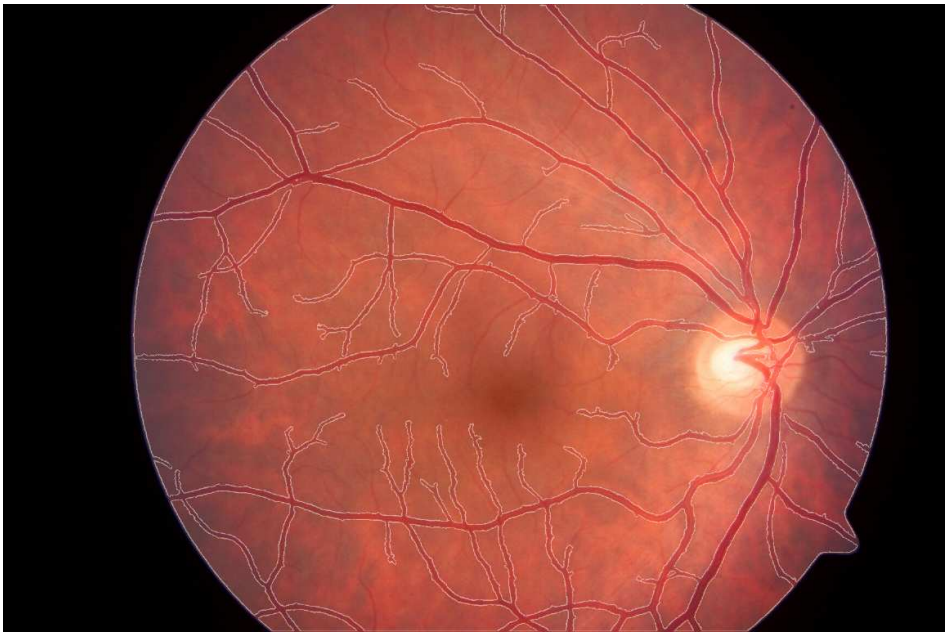
pour $a = g$ et $b = 1$ nous obtenons le modèle géométrique alors que pour $a = 1$ et $b = g$ nous aboutissons au modèle géodésique. $a = b = 1$ et $k = 0$ donnent l'évolution contrôlée par la courbure moyenne.

Les modèles présentés ci-dessus visent à minimiser un critère donné sur les frontières des régions séparées par les contours. Les forces de déformation des contours sont locales ce qui peut engendrer des problèmes d'attraction des contours vers les minima locaux. (Ronfard, 1994), (Cohen, 1991) et (Zhu and Youille, 1996) et d'autres introduisent le type des contours actifs basé sur des régions. Ici, chaque région est caractérisée par des descripteurs (moyenne de l'intensité, variance, histogramme ou données statistiques).

Le principal problème de cette approche est la sensibilité à l'initialisation du contour. Ce problème peut être pallié par l'ajout des informations à priori sur l'objet comme par exemple sa forme. Des méthodes destinées à la segmentation des séquences vidéo des images couleur sont introduites dans (Gastaud, Barlaud and Aubert, 2004) et (Jehan, Barlaud and Aubert, 2003).

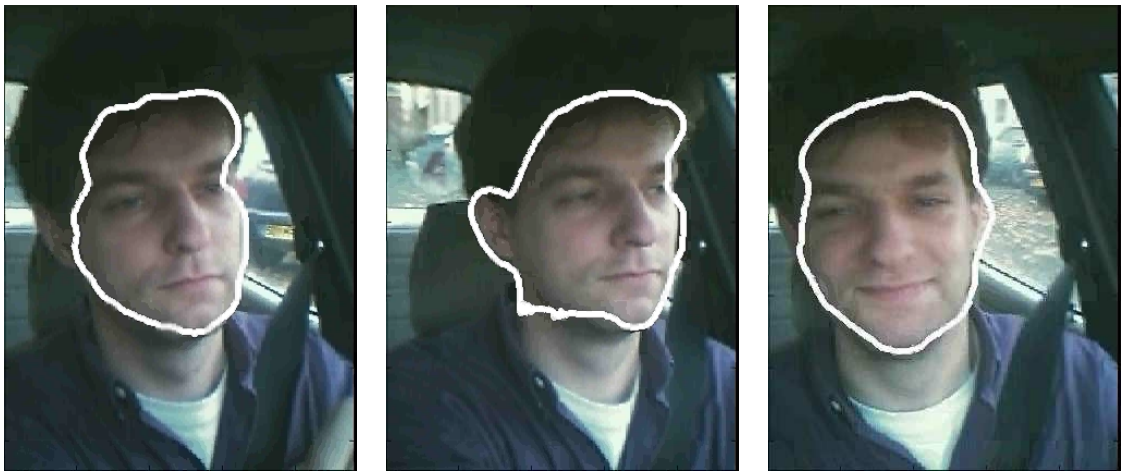


(a)



(b)

FIG. 2.13 – Segmentation des vaisseaux rétiniens (Klein et al., 2003). Les contours initiaux sont matérialisés par le squelette calculé sur l'image normalisée (Walter, 2003).



(a) (b) (c)

FIG. 2.14 – Exemple de suivie de visage du conducteur basé sur les contours actifs. Le modèle est décrit dans (Dokládal et al., 2004)

Les auteurs procèdent à la minimisation de la distance entre le contour de référence et le contour actif ou entre l'histogramme de référence et l'histogramme de la région concernée.

Rappelons que les contours actifs basés sur les méthodes des courbes de niveaux permettent de gérer facilement les changements de topologie, bénéficient d'une grande stabilité numérique et sont indépendants de la paramétrisation des contours. En revanche, leur inconvénient majeur est la complexité de calcul très élevée et une convergence souvent très lente. De nombreuses solutions ont été proposées afin de pallier ce problème et d'accélérer l'évolution. En principe, les solutions proposées concernent la stabilité de la discrétisation temporelle et la vitesse d'intégration liée. Nous pouvons citer l'approche par l'évolution simultanée de plusieurs contours (Paragios, 2000), (Suri, Wu, Reden, Gao, Singh and Laxminarayan, 2001) ou l'approche par le schéma d'intégration semi-implicite (Weickert and Kühne, 2003).

2.4 Conclusions

Le domaine du traitement d'images par des méthodes aux équations aux dérivées partielles contient une large palette d'opérateurs. Le chapitre qui vient d'être présenté vise à montrer les richesses du domaine. En même temps elle justifie le choix du type de méthodes à étudier c'est à dire des méthodes d'ensembles de niveau.

Deux grands groupes d'opérateurs sont présentés : amélioration et segmentation des images. Pour chaque groupe, nous avons choisi quelques représentants typiques. Chaque application est accompagnée de l'équation de l'évolution et d'une discussion des principes d'évolution.

En étudiant ces équations, nous pouvons observer l'aspect générique des méthodes présentées. Ceci pourrait être un point intéressant du point de vue d'une implantation matérielle car l'approche générique permettrait qu'une architecture porte plusieurs groupes de traitement.

Cette observation sert de motivation pour le chapitre suivant où nous nous intéressons à rechercher des points communs pour la conception d'une architecture dédiée.

Finalement, la partie concernant les problèmes de segmentation par EDPs a dévoilé l'importance de l'équation Eikonale qui sert pour le calcul de la fonction distance avec la précision sous-pixélique. Nous avons vu qu'elle est utilisée dans l'algorithme de la LPE continue ainsi que dans les contours actifs pour décrire la position du contour. Pour cette raison nous allons porter une attention particulière au calcul efficace de l'équation Eikonale.

Chapitre 3

Analyse des algorithmes

Cette section se focalise sur l'analyse d'implantation numérique des algorithmes présentés préalablement. Or, l'objectif est de concevoir une architecture générale et efficace pour le plus large nombre d'applications. La première étape consiste à trouver les points communs des algorithmes afin de pouvoir définir la stratégie de parallélisation.

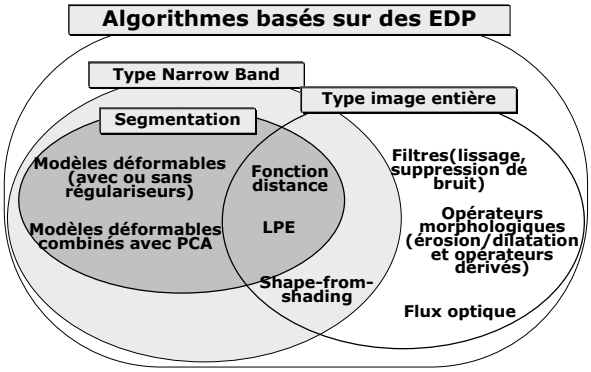


FIG. 3.1 – Classification des algorithmes basés sur des EDPs.

Selon la classification établie (Fig. 3.1), nous distinguons deux types d'algorithmes. Nous appelons le premier **type image entière** car des algorithmes de ce type procèdent de manière itérative au traitement de tous les points dans l'image, jusqu'à la convergence. Le second groupe est de **type Narrow Band**. Ce type d'algorithmes n'effectue le calcul que sur une bande étroite (*Narrow Band*) autour du contour en évolution. Par conséquent, tous les points ne sont pas nécessairement traités. Cette technique a été introduite par (Osher and Sethian, 1988). A l'intersection du type Narrow Band et du type image entière, se trouvent des algorithmes qui effectuent des calculs sur une bande étroite autour du front. En revanche, la solution est cherchée pour tous les points dans l'image.

Dans la Fig. 3.1, nous avons marqué des algorithmes de segmentation. Bien évidemment,

certains algorithmes de type image entière peuvent également fournir une partition du plan par création des zones plates. Cependant, les algorithmes appartenant au groupe segmentation procèdent par extraction des objets, par déformation des contours, ou par détermination des zones d'influence.

3.1 Type image entière

Tous les algorithmes présentés dans la section 2.1 appartiennent aux algorithmes de type image entière. Il s'agit de l'amélioration des images à base de filtres à diffusion linéaire ou non linéaire, les opérateurs morphologiques utilisant l'érosion et la dilatation.

L'équation fondamentale est :

$$\begin{cases} \frac{\partial u}{\partial t} = \mathcal{F}(\nabla u) \\ u(t=0) = u_0 \end{cases} \tag{3.1}$$

Les conditions initiales u_0 représentent directement l'image originale I à filtrer ou à lisser. La fonction $\mathcal{F}$ représente la vitesse de déformation de la surface de l'image. La plupart des algorithmes ne se différencient que par la façon de calculer $\mathcal{F}$. Dans les sections suivantes, nous allons présenter la discrétisation de l'équation (3.1) avec des conséquences sur la complexité de calcul, le nombre d'itérations et la vitesse de convergence.

3.1.1 Schéma explicite

Le plus souvent, les algorithmes adoptent le schéma d'intégration explicite d'Euler.

$$\begin{cases} u^{n+1} = u^n + \tau \mathcal{F}(u^n) \\ u(t=0) = u^0 \end{cases} \tag{3.2}$$

où n est le numéro d'itération et τ est le pas d'intégration. En principe, la condition initiale est équivalente à l'image d'entrée $u^0 = I$.

Le schéma d'Euler est très facile à mettre en œuvre. La mise à jour d'un point est un calcul local et indépendant de la mise à jour de son voisinage. Ceci permettrait d'obtenir un premier niveau de parallélisation de granularité moyenne. En revanche, l'augmentation de τ peut provoquer l'instabilité de la solution numérique ou ralentir la convergence. La condition de stabilité CFL (selon ses auteurs : Couran, Friedrichs et Lewy) limite le rapport du pas d'intégration et de la discrétisation spatiale $\frac{\tau}{\Delta x}$ où $\tau \to 0$ et $\Delta x \to 0$. La condition CFL impose le choix τ de telle façon qu'en aucune itération, on ne dépasse pas le taux du flux

maximale possible d'information (Kimmel, 1995). Pour l'équation du type $u_t + (H(u))_x = 0$, la condition CFL s'exprime sous la forme suivante :

$$1 \geq \frac{\tau}{\Delta x}|H'| \tag{3.3}$$

3.1.1.1 Diffusion linéaire

La diffusion linéaire opère localement sur le voisinage du points à traiter. Il ne s'agit pas d'un filtre directionnel, par conséquent, la discrétisation est simple. Le schéma numérique sert à calculer l'équation de la chaleur (chapitre 2) :

$$\mathcal{F} = \Delta u = \frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} \tag{3.4}$$

où Δ est l'opérateur de Laplace. Pour calculer les dérivées secondes, nous appliquons directement les dérivées secondes calculées par des différences finies telles qu'elles ont été introduites par les équations (1.17). Ainsi l'équation de la diffusion linéaire devient : $\mathcal{F} = u_{xx} + u_{yy}$.

3.1.1.2 Diffusion non linéaire

La mise en œuvre de la diffusion non linéaire représente, par rapport à la diffusion linéaire, un filtre directionnel. Par conséquent, il faut utiliser des différences dans les directions particulières. Rappelons la fonction de la vitesse d'évolution :

$$\mathcal{F} = \operatorname{div}\left(g(\|\nabla u\|)\nabla u\right) \quad \text{avec} \quad g(\|\nabla u\|) = \exp\left(-\left(\frac{\|\nabla u\|}{k}\right)^2\right) \tag{3.5}$$

En fonction des différences, nous estimons la valeur de la fonction $g()$ et effectuons l'intégration et la mise à jour du point traité. En fonction du modèle utilisé, la fonction $g()$ peut avoir la forme suivante :

$$g(\|\nabla u\|) = \frac{1}{1 + \left(\frac{\|\nabla u\|}{k}\right)^2} \tag{3.6}$$

Nous pouvons trouver de nombreuses façons d'obtenir $g()$, des exemples peuvent être trouvés dans (Schüpp, 2000).

Le terme de vitesse d'évolution prend la forme de l'équation suivante :

$$\mathcal{F} = g(u_x^-)u_x^- + g(u_x^+)u_x^+ + g(u_y^-)u_y^- + g(u_y^+)u_y^+ \tag{3.7}$$

3.1.1.3 Opérateurs morphologiques continus

Dans le cas des opérateurs morphologiques, l'application directe des différences finies à la discrétisation de l'équation continue (3.8) n'est pas possible.

$$\mathcal{F} = \pm g(\|\nabla u\|) \tag{3.8}$$

le signe $\pm$ est choisi suivant s'il s'agit de l'érosion ou de la dilatation. La discrétisation doit préserver la condition d'entropie afin d'obtenir une solution stable. Il faut employer une discrétisation spatiale plus complexe

$$\|\nabla u\| = \left[\max(g,0)\nabla^+ + \min(g,0)\nabla^-\right] \tag{3.9}$$

$$\nabla^+ = \left[\max(u_x^-,0)^2 + \min(u_x^+,0)^2 + \max(u_y^-,0)^2 + \min(u_y^+,0)^2\right]^{\frac{1}{2}} \tag{3.10}$$

$$\nabla^- = \left[\max(u_x^+,0)^2 + \min(u_x^-,0)^2 + \max(u_y^+,0)^2 + \min(u_y^-,0)^2\right]^{\frac{1}{2}} \tag{3.11}$$

Ce schéma a été proposé par Brockett and Maragos (1992). Il existe d'autres types de discrétisation spatiale de l'Hamiltonien (section 1.3.2). L'effort principal est consacré à éliminer l'effet des marches d'escalier sur l'élément structurant qui est dû au pas de discrétisation et à la diffusion provoquée par les schémas numériques. Parmi d'autre modèles proposés nous pouvons citer le modèle des facettes par (van den Boomgaard, 1998), le schéma de Rouy et Tourine (Rouy and Tourin, 1992) ou l'approche de (Tek and Kimia, 1998).

3.1.1.4 Nivellement continu

Etant donné que la reconstruction d'image par nivellement consiste en application alternée des érosions et des dilatations, la discrétisation du générateur du nivellement reprend la discrétisation des opérateurs basiques et n'ajoute que le terme qui permet de déterminer le signe de l'opération.

$$\mathcal{F} = \left[\max\{S,0\}\nabla^- + \min\{S,0\}\nabla^+\right] \tag{3.12}$$

avec $S = \text{sign}(I_{ij} - u_{ij})$. Pour la stabilité, il faut satisfaire $\tau/\Delta x + \tau/\Delta y \leq 0.5$ (Maragos and Meyer, 1999).

3.1.2 Schéma semi-implicite

Dans (Weickert et al., 1998), l'auteur propose de mettre en œuvre un schéma d'intégration semi-implicite, nommé *Additive Operator Spliting (AOS)* (chapitre 2). Ce schéma est insensible

à la grandeur du pas d'intégration, il bénéficie d'une stabilité absolue. En augmentant le pas d'intégration, nous pouvons considérablement diminuer le nombre d'itérations.

En revanche, les schémas *AOS* nécessitent de résoudre un système d'équations linéaires de la même taille que celle de l'image. Ce schéma ne peut s'appliquer à tous les types de problèmes. Il est, en effet, indispensable que l'EDP en question soit d'un type spécifique. Il est nécessaire que la force d'évolution soit une fonction de $\operatorname{div}\big(g(\|\nabla u\|)\nabla u\big)$ ce qui représente la base de la diffusion non linéaire.

$$\begin{cases} \frac{\partial u}{\partial t} = \mathcal{F}\Big(\operatorname{div}\big(g(\|\nabla u\|)\nabla u\big), \nabla u, u\Big) \\ u(t=0) = u_0 \end{cases} \tag{3.13}$$

Fonction $g()$ est le terme de diffusivité dans le cas du filtrage (Weickert et al., 1998).

Pour ce type des EDPs, nous pouvons utiliser le schéma semi-implicite pour l'intégration numérique. Plus de détails sur la dérivation de ce schéma sont présentés dans (Weickert and Kühne, 2003). L'équation suivante présente l'expression résultante, prête à être utilisée.

$$\begin{cases} u^{n+1} = \frac{1}{2}\sum_{\ell\in\{x,y\}} \left(E - 2\tau A_\ell(u^n)\right)^{-1} \\ u(t=0) = u^0 \end{cases} \tag{3.14}$$

où E est une matrice unité et A_ℓ est une matrice de taille de l'image traitée.

$$A_\ell(u) = \{a_{ij\ell}(u)\}; \quad a_{ij\ell} = \begin{cases} s(g(i), g(j)) \text{ si } j \in V_4(i) \\ -\sum_{n\in V_4(i)} s(g(i), g(n)) \text{ si } j = i \\ 0 \text{ sinon} \end{cases} \tag{3.15}$$

L'équation (3.14) définit un système tridiagonal des équations linéaires qui est résolu par l'algorithme de Thomas. Le problème général est formulé : $Bu = d$. La solution est cherchée par l'élimination Gaussienne en sachant que la matrice B est de taille $N \times N$.

Algorithme de Thomas : commence par la décomposition LR de la matrice B.

$$B = \begin{bmatrix} \alpha_1 & \beta_1 & & & \\ \gamma_1 & \alpha_2 & \beta_2 & & \\ & \ddots & \ddots & \ddots & \\ & & & \gamma_{N-1} & \alpha_N \end{bmatrix} = \underbrace{\begin{bmatrix} 1 & & & \\ l_1 & 1 & & \\ & \ddots & \ddots & \\ & & l_{N-1} & 1 \end{bmatrix}}_{L} \underbrace{\begin{bmatrix} m_1 & r_1 & & \\ & \ddots & \ddots & \\ & & m_{N-1} & r_{N-1} \\ & & & m_N \end{bmatrix}}_{R} \tag{3.16}$$

Pour tout i, les coefficients $r_i = \beta_i$. m_i avec l_i sont obtenus de la manière suivante :

$$\begin{array}{l} m_1 = \alpha_1 \\ \text{for} \quad i = 1, 2, \ldots, N-1 \\ \qquad l_i = \gamma_i / m_i \\ \qquad m_{i+1} = \alpha_{i+1} - l_i \beta_i \end{array}$$

Ensuite, la solution de $LRu = d$ est effectuée en deux étapes. D'abord, le systèmes des équations $Ly = d$ est résolu. Les résultats sont utilisés dans $Ru = y$.

$$\begin{array}{l} y_1 = d_1 \\ \text{for} \quad i = 2, \ldots, N \\ \qquad y_i = d_i - l_{i-1} y_{i-1} \end{array} \qquad \begin{array}{l} u_N = y_N / m_N \\ \text{for} \quad i = N-1, \ldots, 1 \\ \qquad u_i = (y_i - \beta_i u_{i+1}) / m_i \end{array}$$

L'algorithme de Thomas est numériquement stable pour toute matrice diagonale, strictement dominante.

3.1.3 Complexité de calcul et parallélisation

Schéma explicite

Le schéma d'Euler est très facile à mettre en œuvre. La mise à jour d'un point est un calcul local et indépendant de la mise à jour du voisinage. Ceci permettrait d'obtenir un premier niveau de parallélisation de granularité moyenne. En revanche, l'augmentation de τ peut provoquer l'instabilité de la solution numérique ou ralentir la convergence. De l'analyse de l'équation (3.2) résulte la complexité de calcul linéaire de N (le nombre de points à traiter). Nous avons N multiplications et également N additions.

Nous pouvons réécrire les algorithmes présentés sur la Fig. 3.1 sous la forme sur la Fig. 3.2. L'objectif est de mettre en évidence le potentiel de parallélisation des algorithmes image entière à schéma explicite. L'expression "Pour tous p ... faire **en parallèle**" signifie que chaque opération du code est effectuée en parallèle sur tous les points p mais le code est exécuté de manière séquentielle.

Comme il a été déjà évoqué, l'inconvénient est que pour préserver la stabilité de la solution, il est nécessaire d'intégrer avec un τ très petit ce qui provoque un grand nombre d'itération à effectuer. Par conséquent, le temps de traitement peut être long.

Les divers exemples dans les sections 3.1.1.1-3.1.1.4 nous montrent que le calcul n'est basé que sur le voisinage local même si la discrétisation spatiale est plus ou moins complexe en fonction de la préservation de la condition d'entropie par le schéma numérique. En plus, la mise à jour d'un point est individuelle, l'ordre de traitement des points est indifférent. En profitant de toutes les possibilités de parallélisation, nous pouvons raccourcir considérablement

```
// Initialisation :
Initialiser τ ;   //pas d'intégration
Initialiser u^0 ;   //image originale u^0 = u_0
// Evolution :
Jusqu'à la convergence :
// terme "convergence" représente un critère d'arrêt :
// 1. abs(u^n − u^{n+1}) < const
// 2. Le nombre d'itérations effectuées
{
    Pour tous p appartenant à l'image faire en parallèle
        {
            Extraire voisinage u^n(V_4(p)) et u^n(p) ;
            Calculer F(p) ;
            Intégrer u^{n+1}(p) = u^n(p) + τF(p)
            Mettre à jour u^{n+1}(p) ;
        }
}
```

FIG. 3.2 – Schéma explicite

le temps d'exécution. Grâce aux calculs locaux et indépendants nous avons le choix entre le parallélisme massif ou l'approche semi-parallèle.

Pour conclure, regroupons des possibilités de parallélisation :

- Mise à jour indépendante de tous les points
- Calcul de $\mathcal{F}$
- Opérations arithmétiques sur les données

Schéma semi-implicite

Nous avons présenté en détails l'algorithme de Thomas qui se compose de plusieurs étapes de calcul. D'abord, on effectue la décomposition LR et après on applique l'élimination de Gauss dans deux directions : causal et anticausal (Weickert et al., 1998). L'implantation de cet algorithme demande au total :

$$2(N-1) + (N-1) + 1 + 2(N-1) = 5N - 4$$

multiplications et divisions. Et

$$(N-1) + (N-1) + (N-1) = 3N - 3$$

additions et soustractions. Ainsi, la complexité de calcul est linéaire en N. En revanche, il

ne faut pas oublier que l'algorithme de Thomas doit être appliqué deux fois, pour chaque direction de coordonnées séparément. Le temps d'exécution d'une itération est quelque fois plus long que le temps d'exécution du schéma explicite (Weickert et al., 1998). De l'autre côté, nous pouvons utiliser le pas d'intégration τ plus grand car le schéma semi-implicite reste stable.

L'algorithme de Thomas offre la possibilité en deux niveaux de parallélisation :
- Les évolutions $1D$ dans les directions des coordonnées sont indépendantes
- Les opérations arithmétiques parallèles sur des données

En revanche, il faut respecter l'ordre des points dans l'élimination de Gauss ainsi que la suite des étapes de l'algorithme de Thomas, où la suivante ne peut commencer avant que la précédente ne soit terminée.

Fig. 3.3 montre comment cet algorithme peut s'écrire à l'aide des pseudo-instructions.

3.2 Type Narrow Band

L'algorithme contient les phases suivantes :

- Initialisation : détection du contour initial avec la précision souhaitée et construction de la représentation des contours.
- Propagation : liée à la formulation du problème et au type du schéma numérique choisi.
  - Formulation générale : le contour initial est déformé, son évolution est réalisée par le déplacement de l'ensemble de niveau zéro de la fonction distance.
  - Formulation stationnaire : le contour initial représente la source de propagation d'onde.

3.2.1 Initialisation des contours

Les méthodes basées sur l'évolution de courbe (surface) contrôlée par EDP ont besoin d'une fonction initiale $\varphi = \varphi(x, y)$. Il est imposé que l'ensemble de niveau zéro de la fonction φ corresponde à la position de la courbe initiale (Sethian, 1996). En pratique, φ est défini sur la maille discrète de l'image de telle façon qu'elle ait la valeur négative à l'intérieur et la valeur positive à l'extérieur (ou l'inverse, selon la direction de l'évolution souhaitée). L'ensemble de niveau zéro de φ est souvent placé entre les points de la maille de l'image. Pour le détecter, il faut calculer par l'interpolation la distance exacte signée à la courbe initiale pour tous les points p qui vérifient la condition suivante :

$$p : \exists q, q \in V(p), sign\left(\varphi\left(p\right)\right) \neq sign\left(\varphi\left(q\right)\right) \tag{3.17}$$

Les interpolations les plus utilisées sont l'interpolation bilinéaire et l'interpolation linéaire.

```
// Initialisation :
Initialiser τ ;   //pas d'intégration
Initialiser u^0 ;   //image originale u^0 = u_0
// Evolution :
Jusqu'à la convergence
// terme "convergence" représente un critère d'arrêt :
// 1. abs(u^n − u^{n+1}) < const
// 2. Le nombre d'itérations effectuées
{
//Initialisation des matrices g() et A pour différentes directions ℓ
    Pour tous p appartenant à l'image faire en parallèle
    {
        Calculer g(p) ;
        Calculer A_ℓ ; // ℓ = {x, y}
    }
    Pour ∀ℓ faire en parallèle
    {
        Décomposition L_ℓ R_ℓ ;
        Calculer L_ℓ y_ℓ = d_ℓ ;
        Calculer R_ℓ u_ℓ = y_ℓ ;
    }
    Pour tous p appartenant à l'image faire en parallèle
    }
        Calculer 1/2 Σ_{ℓ∈{x,y}} u_ℓ(p)
        Mettre à jour u^{n+1}(p) ;
    }
}
```

FIG. 3.3 – Schéma semi-implicite

Ces types d'interpolation ne donnent pas les meilleurs résultats du point de vue du respect de la condition sur la qualité du gradient : $\|\nabla u\| = 1$. Ils ne sont pas non plus capables de détecter la présence de plusieurs courbes entre deux points. En revanche, ils sont très faciles à implanter et dans beaucoup d'applications les erreurs sont acceptables.

Il existe un nombre important de méthodes d'interpolation d'ordres plus élevés qui permettent d'augmenter la précision (par exemple (Nessyahu and Tadmor, 1990) ou (Osher and Shu, 1991)). Cependant, cette précision est payée par des calculs plus compliqués et souvent, on est obligé de considérer un voisinage plus large.

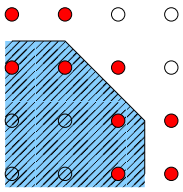
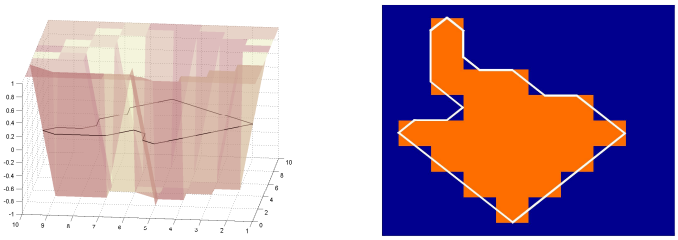


FIG. 3.4 – Exemple de la fonction $\varphi(x, y)$. Le fond blanc représente l'extérieur de la courbe, le fond gris l'intérieur. Les points, dont la valeur est obtenue par l'interpolation, sont marqués en noir.



(a) (b)

FIG. 3.5 – Exemple de détection de la courbe initiale par interpolation linéaire. (a) Fonction $\varphi(x, y)$, visualisée en tant que surface 3D. La ligne noire indique la courbe initiale détectée avec une précision sous-pixélique. (b) La courbe initiale, détectée par l'interpolation linéaire, superposée sur la fonction $\varphi(x, y)$.

3.2.1.1 Interpolation linéaire

Pour calculer l'interpolation linéaire, on détermine les distances d_x et d_y du point à l'ensemble de niveau zéro dans les directions x et y. L'équation de l'interpolation linéaire pour un point $p = [i, j]$ est la suivante :

$$d_x(p_{i,j}) = \frac{\varphi(p_{i,j})}{\varphi_x^{\pm}(p_{i,j})} \tag{3.18}$$

$$d_y(p_{i,j}) = \frac{\varphi(p_{i,j})}{\varphi_y^{\pm}(p_{i,j})} \tag{3.19}$$

$\varphi_x^{\pm}$ désigne la différence finie droite/gauche selon x, la direction droite/gauche est choisie en fonction de l'emplacement du voisin de signe opposé.

Si la fonction φ est constante à l'intérieur et à l'extérieur de la courbe : $|\varphi| = const$, nous ne pouvons avoir que quatre configurations possibles des voisins (en ce qui concerne les positions des voisins avec le signe opposé) sur le 4-voisinage (Fig. 3.6).

De plus, l'hypothèse que la fonction φ soit constante à l'intérieur et à l'extérieur de la courbe signifie que les distances d_x et d_y auront toujours la même valeur.

$$d_x = d_y = const \tag{3.20}$$

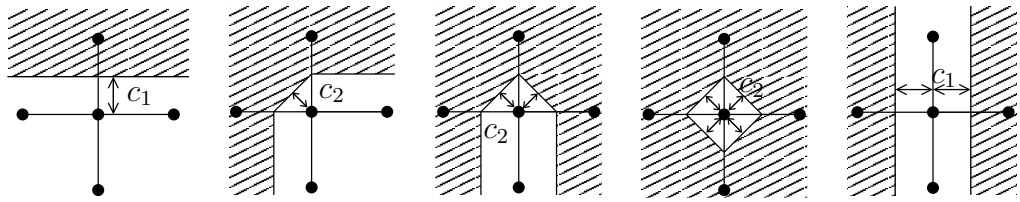


FIG. 3.6 – Interpolation linéaire : diverses configurations des points voisins avec le signe différent du point au cours de traitement.

C'est à dire le résultat numérique de l'interpolation ne dépend que des deux faits suivants :

1. Les points voisins avec le signe opposé se trouvent dans une seule direction (x **ou** y).
2. Les points voisins avec le signe opposé se trouvent dans les deux directions (x **et** y).

Pour tous les points concernés par le cas 1 le résultat de l'interpolation est le même. Ce résultat est égal à la valeur de $d_x = d_y$ (dépendant dans quelle direction est le voisin avec le signe opposé). Dans le cas 2, le résultat est la fonction de d_x et d_y :

$$u = \frac{d_x \, d_y}{\sqrt{d_x^2 + d_y^2}} \tag{3.21}$$

3.2.2 Contours actifs

Les applications concernent avant tout la segmentation des images, l'extraction ou le suivi des objets. L'évolution est contrôlée par la formulation générale des méthodes des courbes de niveau. Rappelons ici l'équation d'évolution fondamentale :

$$\begin{cases} \frac{\partial u}{\partial t} = -\mathcal{F}\|\nabla u\| \\ u(t=0) = u_0 \end{cases} \tag{3.22}$$

Les méthodes de type Narrow Band ont besoin d'une fonction initiale u_0 dont l'ensemble de niveau zéro de la fonction φ correspond à la position de la courbe initiale. Cette fonction initiale se construit par la solution de l'équation Eikonale. L'algorithme de solution de l'équation Eikonale sera présenté plus tard, dans la section 3.2.3.

La discrétisation de l'équation 3.22 est similaire aux exemples des schémas numériques présentés dans la section 1.3.2. La différence de l'implantation consiste dans le traitement des seuls points qui se trouvent dans une distance définie au contour initial. Cette distance est appelée demi-largeur de Narrow Band. La technique de Narrow Band a été introduite par Osher and Sethian (1988). Elle part du principe même de la description implicite du contour.

La position exacte du contour est définie par la position de l'ensemble de niveau zéro, c'est à dire par la distance des points les plus proches du contour. Par conséquent, il suffit de connaître et de ne faire évoluer que les valeurs de ces points. Néanmoins, la largeur de Narrow Band est toujours choisie plus grande pour deux raisons :

- Eliminer les effets de bord et les erreurs éventuelles dues à l'absence des valeurs hors Narrow Band.
- A chaque fois, quand le contour traverse un point, il faut ajouter de nouveaux points au Narrow Band et reconstruire la description par la fonction distance. Il est donc plus avantageux d'effectuer cette reconstruction moins souvent.

Or, au cours de l'évolution, la fonction distance décrivant la position du contour se déforme parce que chaque point de Narrow Band peut évoluer avec une vitesse différente. Par conséquent, au bout d'un certain temps (exprimé en nombre d'itérations ou en vitesse de déplacement de contour) la réinitialisation de Narrow Band doit avoir lieu. Cela est un point de faiblesse des contours actifs implicites.

Plusieurs méthodes ont été proposées pour pallier ce problème. Gomez and Faugeras (1992) effectuent une descente sur gradient pour chaque point jusqu'à l'ensemble de niveau zéro où ils vont chercher la vitesse de propagation. Malgré la suppression de la réinitialisation, cette technique est coûteuse en temps de traitement même si la complexité reste inchangée. Une autre approche est mentionnée dans (Suri, Wu, Reden, Gao, Singh and Laxminarayan, 2001). Elle est basée sur une évolution simultanée de plusieurs contours et minimise ainsi la distance de déplacement des contours et ainsi leurs déformations.

3.2.2.1 Schéma explicite

L'algorithme procède de la même façon que dans le cas du type image entière, à l'exception que le traitement ne s'effectue plus sur l'image entière mais seulement sur les points appartenant au Narrow Band. En ce qui concerne la discrétisation du calcul de la fonction $\mathcal{F}$ et ∇u, selon le modèle appliqué, nous appliquons directement les différences finies ou les schémas numériques respectant la condition d'entropie (section 1.1).

A partir des exemples de modèles géométriques nous voyons que le calcul de $\mathcal{F}$ reste toujours un problème local, n'utilisant que des voisins les plus proches. Nous retrouvons les mêmes possibilités de parallélisation grâce à l'indépendance de la mise à jour des points appartenant au Narrow Band. L'ordre des points reste toujours indifférent.

L'algorithme général des contours actifs se résume en pseudo-instructions fournies par la Fig. 3.2. La réinitialisation du Narrow Band par le calcul de la fonction distance avec la précision sous-pixélique représente le goulet d'étranglement pour une parallélisation efficace. Ce problème est discuté dans la section décrivant l'implantation de l'équation Eikonale.

```
// Initialisation :
Initialiser τ ;   //pas d'intégration
Initialiser u^0 ;   //image originale
// Initialiser le Narrow Band (NB) avec la largeur w :
Pour tous p appartenant à l'image faire en parallèle
    {
        Interpoler ; // détection des contours initiaux
        Calculer distance u < NB_w ;
    }
// Evolution :
Jusqu'à la convergence
// terme "convergence" représente un critère d'arrêt :
// 1. abs(u^n − u^{n+1}) < const
// 2. Le nombre d'itérations effectuées
{ // Déformation de contour :
    Jusqu'au besoin de réinitialiser
    // Par exemple le nombre d'itérations effectuées
    // ou l'approchement des bords de Narrow Band
    {
        Pour tous p appartenant au Narrow Band faire en parallèle
        {
            Extraire voisinage u^n(V_4(p)) et u^n(p) ;
            Calculer F(p) ;
            Intégrer u^{n+1}(p) = u^n(p) + τF(p)
            Mettre à jour u^{n+1}(p) ;
        }
    }
    Pour tous p appartenant au Narrow Band faire en parallèle
    {
        Interpoler ; // détection des contours initiaux
        Calculer distance u < NB_w ;
    }
}
```

FIG. 3.7 – Contours actifs : schéma explicite

3.2.2.2 Schéma semi implicite

La discussion du schéma semi implicite pour le type image entière est valide également pour les contours actifs. La contrainte concernant la forme de l'équation persiste. En revanche, même en utilisant le schéma semi implicite, nous pouvons appliquer la technique de Narrow Band (Weickert and Kühne, 2003) et réduire le nombre de points traités. L'algorithme peut être réécrit comme le montre la Fig. 3.8.

Néanmoins, nous sommes toujours obligés de reconstruire le Narrow Band et nous sommes à nouveau confrontés au problème de l'absence d'un calcul efficace (c'est à dire parallèle) de la fonction distance avec précision sous-pixélique.

3.2.3 Formulation stationnaire

Le chapitre 2 montre quelques exemples des applications pratiques comme la LPE continue ou le calcul du squelette. Ces applications sont contrôlées par l'équation Eikonale :

$$\begin{cases} \|\nabla u\| = \mathcal{P} \\ u(t=0) = u_0 \end{cases} \tag{3.23}$$

La fonction u_0 est une condition initiale qui contient la courbe placée par une méthode d'interpolation (section 3.2.1).

Pour résoudre l'équation (3.23) nous devons résoudre une équation quadratique et considérer seulement sa solution maximale. Afin de pouvoir choisir l'algorithme de calcul de la fonction distance qui serait efficace non seulement du point de vue de la rapidité et de la précision mais aussi du point de vue de sa réalisation sur une architecture dédiée, nous consacrons cette partie à l'analyse des méthodes existantes et les plus utilisées dans le contexte de l'implantation des méthodes d'évolution de courbe (surface) contrôlée par EDP.

Notre étude a pour but de trouver les moyens de simplification et de parallélisation possibles des calculs. Le procédé de calcul de la fonction distance étant composé de trois étapes spécifiques, nous allons présenter chacune de ces étapes dans l'ordre suivant :

Initialisation : on utilise l'interpolation pour détecter la position de la courbe initiale avec la précision sous-pixélique à partir d'une fonction $\varphi \in \mathbb{R}^2$. (Fig. 3.4).

Propagation : la définition de l'ordre de traitement des points, c'est à dire l'ordre dans lequel on parcourt les points de l'image afin de leur donner une valeur de distance en fonction des résultats de l'étape précédente.

Approximation : le calcul de la valeur de fonction distance en fonction des valeurs des voisins du point en cours de traitement.

```
// Initialisation :
Initialiser τ ;   //pas d'intégration
Initialiser u^0 ;   //image originale
// Initialiser le Narrow band :
Pour tous p appartenant à l'image faire en parallèle
{
    Interpoler ; // détection des contours initiaux
    Calculer distance u < NB_w ;
}
// Evolution :
Jusqu'à la convergence
// terme "convergence" représente un critère d'arrêt :
// 1. abs(u^n − u^{n+1}) < const
// 2. Le nombre d'itérations effectuées
{   Jusqu'au besoin de réinitialiser
    {
        //Initialisation des matrices g() et A pour différentes directions ℓ
        Pour tous p appartenant au Narrow Band faire en parallèle
        {       Calculer g() ;
                Calculer A_ℓ ; // ℓ = {x, y}
        }
        Pour tous A_ℓ faire en parallèle
        {
                Décomposition L_ℓ R_ℓ ;
                Calculer L_ℓ y_ℓ = d_ℓ ;
                Calculer R_ℓ u_ℓ = y_ℓ ;
        }
        Pour tous p appartenant au Narrow Band faire en parallèle
        }
                Calculer 1/2 Σ_{ℓ∈{x,y}} u_ℓ(p)
                Mettre à jour u^{n+1}(p) ;
        }
    }
    Pour tous p appartenant au Narrow Band faire en parallèle
    {
        Interpoler ; // détection des contours initiaux
        Calculer distance u < NB_w ;
    }
}
```

FIG. 3.8 – Contours actifs : schéma semi implicite

Rappelons que nous utilisons le 4-voisinage de point pour les parties suivantes de ce chapitre.

Cette partie est consacrée à l'étude des techniques de propagation. Une fois l'interpolation effectuée, l'étape suivante est de propager la fonction distance vers le voisinage des points qui ont déjà été traités (ils ont une valeur finie de la fonction distance). Cela veut dire que l'on définit un ordre dans lequel les points reçoivent la valeur de la fonction distance ou dans lequel cette valeur est mise à jour.

Il existe un grand nombre d'algorithmes de calcul de la fonction distance. Ces algorithmes calculent une fonction distance plus ou moins précise avec un nombre d'opérations plus ou moins élevé ((Verbeek and Verwer, 1990), (Tsai, 2000), (Eggers, 1997) ou (Menhert and Jackway, 1999)).

L'algorithme le plus souvent utilisé avec les méthodes basées sur les équations aux dérivées partielles est l'algorithme à progression rapide (*Fast Marching*). Cet algorithme fournit des résultats assez précis dans la zone d'intérêt et ne nécessite pas d'effectuer des parcours de l'image entière. En revanche, il est nécessaire de gérer des structures en forme de tas ordonnés. L'inconvénient est que la gestion du tas ordonné rend cet algorithme séquentiel parce que l'on ne peut traiter qu'un seul point à la fois (celui avec la priorité maximale).

Il existe des algorithmes qui n'utilisent pas de structures ordonnées et fournissent une estimation précise de la fonction distance. On peut citer l'approche de Danielsson (Danielsson, 1980), qui calcule la fonction distance seulement en deux passes de l'image. L'inconvénient est le parcours obligatoire de l'image entière deux fois et le deuxième parcours ne peut s'effectuer avant que le premier ne soit terminé. L'idée principale est que l'on travaille avec des coordonnées des points dont on obtient le carré de la fonction distance, ce qui représente un deuxième inconvénient.

La simplicité et la transparence de cet algorithme se compliquent en ajoutant l'inévitable initialisation par interpolation. Le fait de travailler avec la précision sous-pixélique a pour conséquence l'augmentation de la taille de la mémoire (pour stocker les coordonnées en nombres réels) et également l'augmentation des temps de calcul parce que toute opération arithmétique s'effectue avec des nombres réels.

Une amélioration est l'algorithme de *sweeping* introduit dans (Boué and Dupuis, 1999) (et repris dans (Tsai, 2000)). Cet algorithme, très simple à implanter, est de même type que celui de Danielsson. Il parcourt l'image en quelques passes et propage la fonction distance de manière directionnelle. En revanche, il travaille carrément avec des valeurs de la fonction distance, et non avec les coordonnées.

Afin de comparer ces deux types d'algorithmes *Fast Marching* et *sweeping*, les sections suivantes fournissent une description plus détaillée de chacun d'eux.

3.2.3.1 Propagation équidistante

L'algorithme *Fast Marching* (Sethian, 1996) a été mis au point en partant du principe de l'algorithme de Dijkstra (Sethian, 1996). Cet algorithme a été étendu et est actuellement appliqué à un grand nombre de problèmes généraux.

Dans l'algorithme suivant, nous utilisons les notations suivantes.

- L'ensemble des points $\mathcal{Q}$ est défini comme un ensemble de tous les points de l'image qui ont obtenu leur valeur finale de fonction distance et qui ne seront plus traités au cours de la propagation
- L'ensemble $\mathcal{A}$ contient tous les points P qui sont en attente d'être traités. Ils sont placés dans un tas ordonné avec une priorité associée.
- L'ensemble $\mathcal{T}$ contient tous les points qui n'appartiennent ni à l'ensemble $\mathcal{A}$ ni à l'ensemble $\mathcal{Q}$.

La priorité des points appartenant à $\mathcal{A}$ est donnée par la valeur de la fonction distance. Le point avec la valeur minimale a la priorité maximale. S'il existe plusieurs points avec la même priorité, ils sont rangés dans l'ordre d'arrivée.

Algorithme 1 (Fast Marching)

1. *Initialisation*
   (a) *Les points de $\mathcal{Q}$ sont initialisés aux valeurs de fonction distance exacte résultantes de l'interpolation (ou à une valeur constante fixée)*
   (b) *L'ensemble $\mathcal{A}$ est initialisé avec des valeurs $u(\mathcal{Q})$ calculées soit d'après le schéma de Godunov soit d'après la méthode d'intersections. Les points appartenant à l'ensemble $\mathcal{A}$ sont mis dans une structure ordonnée, d'après les valeurs de distance croissantes.*
   (c) *L'ensemble $\mathcal{T}$ est initialisé à l'infini $u(\mathcal{T}) = \infty$*
2. *Propagation*
   (a) *Retirer le point p_{min} avec la distance minimale de $\mathcal{A}$*
   (b) *p_{min} passe à l'ensemble $\mathcal{Q}$*
   (c) *Examiner tout voisin q de p_{min}*
      i. *Si q appartient à $\mathcal{T}$, on calcule sa distance $u(q)$ et on range q dans la structure ordonnée. q passe à l'ensemble $\mathcal{A}$*
      ii. *Si q appartient à $\mathcal{A}$, on recalcule sa valeur et si cette valeur est plus petite que la précédente, on met à jour sa valeur $u(q)$*

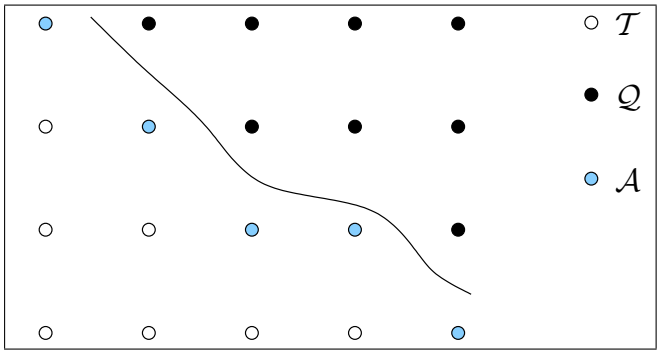


FIG. 3.9 – *Fast marching* : les ensembles $\mathcal{A}$, $\mathcal{Q}$, $\mathcal{T}$

A chaque instant lors de la propagation, la structure ordonnée permet de déterminer le point dont le temps de passage par zéro est minimal. Cette structure est réalisée sous une forme de tas ordonné avec une clé prioritaire (la distance minimale a la priorité maximale). Dans la majorité des implantations, il s'agit d'un arbre binaire dont les valeurs en chaque nœud sont inférieures aux valeurs des nœuds fils. Nous avons implanté le tas ordonné en utilisant le tas de Fibonnacci et nous pouvons comparer les complexités des calculs de deux différentes implantations :

// Initialisation :
Pour tous p appartenant à l'image faire **en parallèle**
{
  Interpoler ; *// détection des sources*
  Initialiser $\mathcal{A}$; *// points ayant obtenus une valeur* $< \infty$
}
// Propagation :
Jusqu'à $\mathcal{A} \neq \{\emptyset\}$
{ *// Pour le point avec la priorité maximale*
  Extraire $p_i = \min u(\mathcal{A})$;
  Passer p_i dans $\mathcal{Q}$;
  Extraire $u(V_4(q_i))$ et $u(q_i)$
  Calculer $u(q_i)$; *//* $q_i \in V_4(p)$ *sauf* $q_i \in \mathcal{Q}$
  Ranger q_i dans $\mathcal{A}$ si la priorité $u(q_i)$ a diminué ;
}

FIG. 3.10 – Fast marching

Complexité de calculs :

Pour un nombre n_p de nœuds d'un arbre équilibré, le coût des opérations liées à l'insertion et à la reorganisation du tas est estimé à $\mathcal{O}(n_p \log n_p)$ Si la structure ordonnée est matérialisée à l'aide de tas de Fibonnacci, la complexité du calcul est la suivante :

Opération	Arbre binaire	Tas de Fibonacci
Insérer()	$\mathcal{O}(\log n_p)$	$\mathcal{O}(\log n_p)$
Minimum()	$\mathcal{O}(1)$	$\mathcal{O}(1)$
EffacerMin()	$\mathcal{O}(\log n_p)$	$\mathcal{O}(\log n_p)$
Union()	$\mathcal{O}(n_p)$	$\mathcal{O}(1)$
BaisserPriorité()	$\mathcal{O}(\log n_p)$	$\mathcal{O}(1)$
EffacerElement()	$\mathcal{O}(\log n_p)$	$\mathcal{O}(\log n_p)$

TAB. 3.1 – Complexité des calculs

Du tableau TAB. 3.1 il est évident que le tas de Fibonacci est plus efficace en opérations de changement de priorité et d'insertion de nouvelles valeurs. En revanche l'arbre binaire est moins exigeant sur le plan de l'utilisation de la mémoire.

3.2.3.2 Groupe Marching

Afin de réduire la complexité de calcul, (Kim, 2001) propose un algorithme qui n'utilise pas de structure ordonnée pour la propagation. Comme l'indique le nom, il est possible de traiter plusieurs points en même temps. L'auteur définit un critère qui permet de sélectionner un groupe de points appartenant au front ou à son voisinage, qui seront traités dans une itération. Le critère d'activation des points est mis à jour après chaque itération. Il est définit comme suit :

$$TM = TM + \delta_t \tag{3.24}$$

$$\delta_t = \frac{1}{\sqrt{3}}\, h\, s_{\min} \tag{3.25}$$

où TM est le minimum de la distance des points appartenant au front de propagation, $h = \Delta x = \Delta y$ et $s_{\min}$ est la vitesse minimale de propagation des points appartenant au front.

Dans chaque itérations, l'algorithme *Groupe Marching* calcule ensuite les voisins des points appartenant au front de propagation dont les valeurs de la distance sont inférieures ou égales à TM. Les points calculés sont parcourus deux fois dans les sens opposés.

La complexité de calcul est linéaire $\mathcal{O}(N)$. Toutefois, le besoin de remettre à jour le critère d'activation à chaque itération représente toujours l'approche globale qui simule la propagation équidistante. Cette approche globale qui nécessite de maintenir une variable globale avec, en plus, un ordre de traitement défini représente un obstacle pour la parallélisation efficace.

3.2.3.3 Balayages successifs

La méthode de *sweeping* a été introduite dans (Boué and Dupuis, 1999). Plus tard dans (Tsai, 2000) ont repris cette méthode et ils ont présenté un algorithme qui n'utilise plus de tas ordonnés et qui devient purement algébrique.

L'idée principale vient du domaine des chaînes de Markov (Boué and Dupuis, 1999). Elle consiste à considérer le problème stationnaire comme une limite des problèmes à temps fini. Cela est connu comme **l'itération de Jacobi**. En fait, l'itération de Jacobi signifie que pour le calcul d'une valeur d'une fonction $F(x,t)$ à l'instant t_{n+1}, on n'utilise que des valeurs de $F(x,t_n)$. Boué et Dupuis proposent d'utiliser **l'itération de Gauss-Seidel** où chaque nouvelle valeur de $F(x,t_{n+1})$ remplace successivement $F(x,t_n)$ pour les prochains calculs. La convergence est prouvée et les propriétés discutées dans le même article. L'inconvénient est que l'on doit parcourir et traiter tous les points de l'image dans toutes les directions de *sweeping* et, pour chaque parcours nous avons besoin de connaître le résultat du parcours précédent.

Nous présentons l'algorithme tel qu'il a été introduit dans (Tsai, 2000), c'est à dire en 3D. Les notations utilisées dans l'algorithme sont les suivantes :

- L'ensemble $\mathcal{A}$ contient des points qui font le voisinage de la courbe (surface) initiale.[1]
- L'ensemble $\mathcal{T}$ contient tous les points qui n'appartiennent pas à l'ensemble $\mathcal{A}$.

Algorithme 2 (Sweeping)

1. *Initialisation*
   (a) *Ensemble $\mathcal{A}$ est initialisé à la distance exacte (l'interpolation) à la surface initiale.*
   (b) *Les points dans $\mathcal{T}$ sont initialisés à l'infini : $u(\mathcal{T}) = \infty$.*
2. *Propagation*
   (a) *Pour tout point $P \in \mathcal{T}$ itérer dans les directions suivantes*
      i. $(x+, y+, z+)$

      for $k = 1 : N_z - 1$
      for $j = 1 : N_y - 1$
      for $i = 1 : N_x - 1$

      A. *Trouver des voisins V_x et V_y tels que : $u(V_x) = \min(u(x_{i+1}, y_j), u(x_{i-1}, y_j))$ et $u(V_y) = \min(u(x_i, y_{j+1}), u(x_i, y_{j-1}))$*
      B. *Rejeter V_x ou V_y si : $V_x = \infty$ ou $V_y = \infty$*
      C. *Rejeter V_l, $l = x, y$ si : $u(V_l) - u(\min(V_x, V_y)) > |V_l - \min(V_x, V_y)|$*

[1] Il est nécessaire de noter que cet algorithme ne touche plus les points appartenant à $\mathcal{A}$ (le voisinage de la courbe (surface) initiale).

D. Calculer $u(P)$

ii. $(z-, y-, x-)$

for $i = N_x - 1 : 1$
for $j = N_y - 1 : 1$
for $k = N_z - 1 : 1$

répéter A - D

iii. $(y+, x+, z-)$ *: répéter A - D*

iv. $(y-, x-, z+)$ *: répéter A - D*

v. $(x-, y+, z+)$ *: répéter A - D*

vi. $(z-, y-, x+)$ *: répéter A - D*

vii. $(x+, y-, z+)$ *: répéter A - D*

viii. $(z-, y+, x-)$ *: répéter A - D*

Pour comparer des possibilités de parallélisation, nous réécrivons l'algorithme de *sweeping* sous forme sur la Fig. 3.11. On n'utilise plus des tas ordonnés. Mais cet avantage est payé par plusieurs balayages de l'image entière dans les directions définies. En plus, un balayage ne peut pas commencer avant que le précédent ne soit pas terminé.

```
// Initialisation :
Pour toutes les directions définies
{
        Parcourir tous les points p dans l'image :
        {
            Extraire u(V4(p)) ;
            Calculer u(p) ;
            Mettre à jour u(p) ;
        }
}
```

FIG. 3.11 – Sweeping

Complexité de calculs :

Cet algorithme est très facile à implanter car il évite l'utilisation des tas ordonnés. Les auteurs estiment que la complexité de calcul est de $\mathcal{O}(MN)$ (Tsai, 2000). M est le nombre des données et N est le nombre des points de la maille.

3.2.3.4 Approximation

Le choix parmi les approximations numériques de la fonction distance est assez large. Les méthodes de calcul peuvent être divisées en deux catégories :

1. Le résultat est une fonction des coordonnées relatives par rapport à la courbe (surface) et il a souvent la forme de puissance de deux de la fonction distance (Danielsson, 1980). Ainsi, on peut travailler en nombres entiers.
2. Le résultat est la valeur de la fonction distance (Sethian, 1996) et on doit utiliser des nombres réels

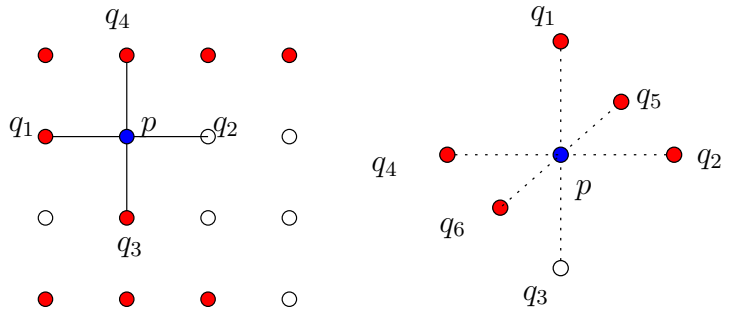
Schéma de Godunov

Le schéma de Godunov est une solution numérique de l'équation Eikonale (2.39) avec $F = 1$. Le schéma s'écrit de la façon suivante :

$$H_G(u_{x_-}, u_{x_+}, u_{y_-}, u_{y_+}) = \sqrt{\max(u_{x_-}, -u_{x_+}, 0)^2 + \max(u_{y_-}, -u_{y_+}, 0)^2} = 1 \qquad (3.26)$$

avec les dérivées partielles : $u_{x_\pm} = D_{x_\pm} u_{i,j}, D_{x_\pm} u_{i,j} = \pm(u_{i_\pm,j} - u_{i,j})/h$ et de la même façon pour la direction y. h est la distance de deux points sur la grille.

L'équation (3.26) nous mène à résoudre une équation quadratique afin d'obtenir la distance du point actuel. L'idée principale est de résoudre cette équation quadratique pour toute paire (ou triplet en 3D) des voisins du point au cours du traitement. On ne considère que la solution qui est plus grande que la valeur maximale des voisins. Le résultat est le minimum des solutions obtenues pour toute combinaison des voisins dans toutes les directions.



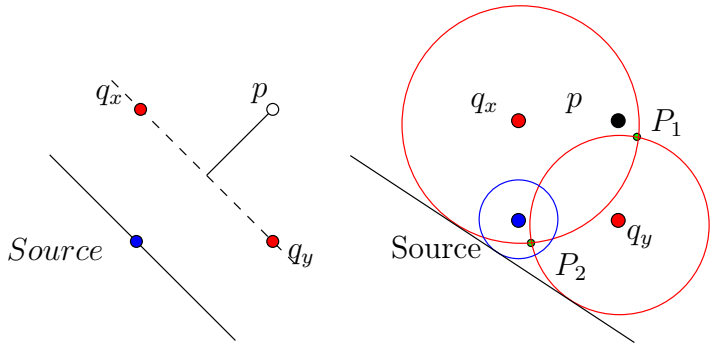
(a) Cas 2D (b) Cas 3D

FIG. 3.12 – Exemples de configuration du voisinage : les points rouges q_i seront utilisés pour le calcul de la valeur du point p ; les points blancs n'ont pas encore obtenu leur valeur (supposons qu'ils aient été initialisés à ∞) et ne peuvent pas être pris en compte pour le calcul de $u(p)$.

La méthode de Godunov n'obtient pas de résultats numériques précis pour le calcul de la distance des points isolés. Par exemple, considérons un point de source $\mathbf{s}$ avec $u(\mathbf{s}) = 0$, voir

la Fig. 3.13. La distance des points est égale à h (h est la distance entre les points, maille isotrope). Nous cherchons la valeur de la fonction distance dans le point p. Cette valeur est obtenue à partir des valeurs des points q_x et q_y. Par la résolution de l'équation Eikonale on obtient $u(p) = (1 + 1/\sqrt{2})h$.

En réalité, la méthode de Godunov se comporte comme si une onde planaire se propageait à partir du point de source et le résultat obtenu est le temps de passage de cette onde par le point traité.



(a) Propagation d'ondes planaires (schéma de Godunov)

(b) Propagation d'ondes circulaires (méthode d'intersections)

FIG. 3.13 – Comparaisons du schéma de Godunov et de la méthode d'intersections

Ainsi on n'obtient pas de formes parfaitement rondes mais une forme de polyèdre (losange).

Méthode d'intersections

Le fait que la méthode de Godunov ne donne pas de résultats précis pour le calcul de la fonction distance des points isolés est provoqué par la propagation d'ondes linéaires (Fig. 3.13(a)). Dans (Tsai, 2000), l'auteur propose un algorithme de propagation des ondes circulaires afin d'obtenir la distance euclidienne précise.

La méthode consiste à calculer les points d'intersection des cercles en 2D (des sphères en 3D) à partir de toute combinaison des paires (des triplets) des points voisins et parmi les résultats obtenus choisir la solution minimale qui respecte la monotonicité de la fonction distance.

Pour tout point p au cours de traitement le procédé du calcul est le suivant : les coordonnées des points sont les coordonnées des centres des cercles et les valeurs de la fonction distance u représentent leurs rayons.

- Pour toute combinaison des voisins, calculer les points d'intersection P_1, P_2 (en 2D). La valeur de distance est $u(p) = \max\{|p - P_1|, |p - P_2|\}$
- Si l'intersection n'existe pas : $u(p) = \min\{\mathrm{V}\} + h$, où h est l'espace de la maille.

Si l'on travaille avec le quatre-voisinage et la maille isotope, avec h étant l'espacement de la maille, la condition de l'existence de l'intersection :

$$r_{q_1} + r_{q_2} \geq \sqrt{h} \tag{3.27}$$

$$|r_{q_1} - r_{q_2}| \leq \sqrt{h} \tag{3.28}$$

où r_{q_1} et r_{q_2} sont les rayons des cercles relatifs aux points q_1, q_2.

Dans le cas où l'intersection n'existerait pas, les auteurs proposent, dans un premier temps, d'additionner la valeur h au voisin minimal. Cela peut résulter en valeurs contradictoires de la fonction distance, c'est-à-dire que la fonction distance ainsi obtenue ne dispose pas de la qualité du module de gradient exigé. C'est pourquoi les auteurs proposent ensuite de combiner les deux méthodes : le schéma de Godunov et les intersections. Il s'agit d'utiliser le schéma de Godunov dans les cas où les intersections n'existent pas et dans les autres cas, de calculer la fonction distance à l'aide des intersections. A force de combiner les deux méthodes, on obtient une meilleure estimation de la fonction distance. Toutefois, combiner les deux méthodes mène à utiliser deux types de calculs très différents. Cela provoquerait les complications de l'architecture éventuelle.

3.3 Conclusion

L'objectif du chapitre actuel est de considérer algorithmes aux EDPs du point de vue du concepteur d'architecture. D'abord, nous avons classé les algorithmes selon la façon dont ils parcourent les images. Ensuite, nous discutons des questions algorithmiques de type : linéarité des fonctions de propagation, complexité des calculs et réutilisation des valeurs. Tous ces points influent sur la conception en imposant des contraintes sur la complexité des unités de traitement et la communication entre elles. Cette étude a également un rôle important pour la recherche sur la parallélisation des calculs.

Ainsi nous avons défini deux groupes d'algorithmes : type à image entière et type Narrow Band. Dans le cas du type image entière, tous les points sont traités en un certain nombre d'itérations. La complexité de calcul est donnée par deux aspects : i) la fonction non linéaire d'évolution et ii) la méthode d'intégration choisie. La première joue un rôle dans la complexité d'une unité arithmétique alors que la deuxième peut influencer la parallélisation des calculs. Nous avons vu que le schéma semi implicite réutilise des valeurs des points traité précédemment alors que le schéma explicite est indépendant pour tous les points et donc entièrement parallèle.

Les deux aspects sont également présents dans le cas du type Narrow Band. Ici, on ajoute en plus le problème de la description du contour ou de l'onde de propagation par la solution de l'équation Eikonale. Du point de vue de la réalisation d'une architecture, les algorithmes de calcul de l'équation Eikonale posent les problèmes suivants :

1. Dans le cas des tas ordonnés (*Fast Marching*) :
   (a) Le maintien obligatoire de variables globales (lecture et modification). A chaque instant on ne peut traiter que le point avec la priorité maximale. Cela exclut la possibilité de parallélisation complète de cette approche.
   (b) La gestion de tas ordonnés où la priorité est en nombres réels. Lors de l'évolution de l'algorithme, nous avons besoin de mettre à jour la priorité des points qui se trouvent déjà dans le tas. Cela pose d'énormes problèmes car il faut aller chercher le point concerné, recalculer sa fonction distance et mettre le point dans le tas à la place correspondant à la nouvelle priorité. Dans le cas où la nouvelle priorité est plus basse que la précédente, on laisse le point à sa place.
2. La méthode de *sweeping* :
   (a) Elle est extrêmement simple à implanter, il n'y a plus de tas ordonnés mais elle a l'inconvénient de devoir parcourir tous les points de l'image dans toutes les directions définies. Ceci n'est pas très économique en nombre d'itérations. Cela devient encore plus désavantageux dans les applications où le calcul de la fonction distance est effectué très souvent.

On voit quels sont les problèmes à résoudre par une réalisation de l'architecture de l'algorithme de *Fast Marching*. D'abord, il faudrait matérialiser un des types de tas ordonné (par exemple arbre binaire ou tas de Fibonacci). Le tas ordonné aurait à respecter la priorité exprimée en nombres réels des éléments stockés. On ne peut pas connaître la taille du tas à priori. En revanche, on peut noter que la taille est bornée pour une taille d'image bornée. Le nombre total des points dans le tas n'est pas constant mais change dynamiquement à chaque itération, au fur et à mesure de l'évolution de la courbe.

Pendant l'exécution de l'algorithme de *Fast marching*, il est nécessaire non seulement de pouvoir ajouter au tas de nouveaux points mais aussi de pouvoir y accéder plus tard afin de recalculer éventuellement la priorité et le cas échéant, de pouvoir la changer. Etant donné que l'algorithme est défini de telle façon que la priorité ne peut qu'augmenter, nous pouvons avoir des cas où la priorité ne va pas changer. Ainsi, un grand nombre d'opérations est inutile.

En ce qui concerne la parallélisation, le plus grand inconvénient est représenté par le traitement de la variable globale. Cette variable globale signifie qu'à chaque instant, on ne travaille qu'avec le point qui a la priorité maximale (la valeur de la fonction distance minimale) et qui est le premier dans le tas ordonné (le sommet de l'arbre binaire ou du tas de Fibonacci). La

gestion du tas ordonné est un processus séquentiel. La lecture et la mise à jour d'autres points du tas ordonné ne peuvent commencer qu'une fois le traitement du point en cours terminé. En effet, le résultat de ce traitement influence l'ordre des priorités des autres points actifs en attente dans le tas. Par conséquent, l'implantation de ce type d'algorithme sur une machine multiprocesseur serait peu efficace. Les seules opérations qu'il est possible de paralléliser sont seulement les accès au voisinage.

On peut imaginer le cas d'utilisation d'une architecture basée sur la parallélisation massive ou d'une architecture *Divide-and-Conquer*. Un seul processeur serait actif et le reste des processeurs inactifs, ce qui n'apporterait pas un gain important en temps de traitement.

Le *Sweeping* apporte une simplification des calculs importante. On n'utilise plus de tas ordonnés. Mais cet avantage est payé par plusieurs balayages de l'image entière dans les directions définies.

Dans la plupart des applications basées sur l'évolution de courbes, nous n'avons besoin de calculer la fonction distance que sur une bande étroite autour de la courbe initiale. Dans ce cas, calculer la fonction de distance pour tous les points de l'image devient peu efficace. Surtout si l'on a besoin de recalculer la fonction distance chaque fois après un certain nombre d'itérations.

L'utilisation de *sweeping* serait encore moins efficace avec les méthodes de *Level Set* sans réinitialisation (Gomez and Faugeras, 1992). Etant donné que ces méthodes ne provoquent pas la déformation de la carte distance, nous n'avons besoin d'appeler la fonction distance que dans le cas où la courbe atteint les bords de la bande étroite. L'algorithme de *sweeping* ne peut que traiter tous les points au lieu d'en ajouter quelques-uns à la bande étroite de la courbe.

En revanche, un incroyable potentiel de parallélisation existe dans l'exécution des boucles "pour tous $p \ldots$ faire en parallèle {}", utilisées fréquemment dans tous les algorithmes analysés dans ce chapitre. Le fait que la mise à jour de tous les points p soit purement locale signifie que l'ordre de traitement de ces points est indifférent et qu'il peut même être exécuté sur tous les points en parallèle.

Le seul verrou technologique de cette implantation est l'inexistence d'un algorithme efficace pour calculer en parallèle la fonction distance. Dans le chapitre suivant, nous proposons un nouvel algorithme *Massive Marching* de calcul de la fonction distance euclidienne exacte qui est complètement parallèle et qui n'utilise pas non plus de tas ordonnés. Pour obtenir ces propriétés, nous remplaçons la variable globale par une constante globale qui nous permet d'assurer les qualités demandées de la fonction distance.

Notons que l'introduction d'un algorithme efficace de calcul de la fonction distance a une grande l'importance car elle pourrait également augmenter l'efficacité de toutes les méthodes

qui emploient la distance pour d'autres objectifs comme par exemple des contours actifs basés sur la minimisation de la distance entre un modèle de référence (histogramme, forme de l'objet) et le contour actif ((Jehan et al., 2003), (Aubert, Barlaud, Faugeras and Jehan, 2003)).

Chapitre 4

Massive Marching

L'obtention de la fonction distance avec une précision sous-pixélique représente un vrai verrou technologique. Le calcul de la distance est coûteux, or souvent répété. Comme nous avons montré dans l'analyse des algorithmes, le calcul de la fonction distance représente le goulet d'étranglement pour une implantation parallèle efficace et souvent il ne sert que de support pour d'autres calculs.

Pour l'optimiser il est nécessaire de trouver un algorithme simple et efficace qui combine les avantages de ces deux approches :

- placer le front de propagation avec précision sous-pixélique et
- calculer sur
  - l'image entière et
  - la bande étroite autour du front.

En partant de l'analyse des algorithmes dans le chapitre précédent, il est clair que l'élimination des structures ordonnées permettrait de concevoir un algorithme parallèle, tout en respectant la propagation de la solution depuis les sources sur le Narrow Band ou sur l'image entière.

En sachant que les tas ordonnés servent à la mise en œuvre de la propagation équidistante, il fallait trouver une façon de propager la solution où l'ordre de traitement des points serait indifférent. C'est à dire trouver un mécanisme pour une propagation non équidistante qui préserverait les qualités de la propagation équidistante. Ainsi nous avons défini l'algorithme présenté dans les sections suivantes et nous l'avons appelé Massive Marching puisqu'il parcourt l'image par des marches (itérations) qui sont exécutées en parallèle sur tous les points à traiter.

La description de Massive Marching suit la logique de son développement. Elle commence par une discussion de la discrétisation de l'équation Eikonale ce qui permet ensuite de définir le mécanisme de la propagation non équidistante appelée critère d'activation. L'analyse de la complexité de calcul et de l'erreur est également présentée.

4.1 Schéma numérique

Dans le chapitre précédent, nous avons discuté des méthodes de discrétisation de l'équation Eikonale :

$$\|\nabla u\| = \mathcal{F} \tag{4.1}$$

où $\mathcal{F}$ est la fonction de coût de la distance pondérée.

Pour la définition de l'algorithme Massive Marching, nous supposons que la valeur $u(p)$ du point p peut s'exprimer en fonction de son voisinage de la manière suivante :

$$u(p) = u_{min}(p) + f_{diff}(V_4(p), \mathcal{F}(p)) \tag{4.2}$$

$u_{min}(p)$ est la valeur minimale des points voisins de p :

$$u_{min}(p) = \min_{q_i \in V_4(p)} \{u(q_i)\}$$

Nous faisons l'hypothèse que f_{diff} est une fonction strictement positive et croissante. L'allure exacte de la fonction f_{diff} dépend du choix du schéma numérique. Nous pouvons montrer en exemple le schéma de Godunov, très largement utilisé dans ce domaine (Sethian, 1996).

$$\left[\max\left\{u(p) - u\left([x_p \pm 1, y_p]\right), 0\right\}^2 - \max\left\{u(p) - u\left([x_p, y_p \pm 1]\right), 0\right\}^2\right]^{\frac{1}{2}} = \mathcal{F}(p) \tag{4.3}$$

Pour obtenir f_{diff}, typiquement, il faut résoudre une équation quadratique où on ne considère que la solution minimale positive sur le voisinage du point traité. Selon notre proposition, le schéma de Godunov peut être réecrit en respectant l'Eq. (4.2) où :

$$f_{diff} = \frac{|u_x(p) - u_y(p)|}{2} + \sqrt{\frac{\mathcal{F}(p)^2}{2} - \left(\frac{u_x(p) - u_y(p)}{2}\right)^2} \tag{4.4}$$

où $u_x(p) = \min\{u([x_p \pm 1, y_p])\}$ et $u_y(p) = \min\{u([x_p, y_p \pm 1])\}$

Egalement dans l'Eq. (4.4) seule la solution minimale est considérée. Si l'Eq. (4.4) n'a pas de solution réelle la distance est obtenue à partir de $f_{diff} = \mathcal{F}(p)$.

4.2 Algorithme

L'algorithme n'utilise aucun ordre de traitement des points, ainsi la propagation n'est pas équidistante à la courbe initiale. La propagation non équidistante engendre deux problèmes que nous devons gérer pendant les calculs :t

1. La valeur de chaque point est obtenue à partir des valeurs des voisins dépendant l'un de l'autre. Cela impose un algorithme à deux estimations de u.
2. Les points ne sont pas forcément envahis par l'onde provenant depuis la source la plus proche. Si c'est le cas, la valeur u du point en question n'est pas correcte. Elle sera rectifiée lors de l'une des prochaines itérations. La détection de ces points est l'un des rôles du critère d'activation.

Pour définir l'algorithme, nous utilisons les ensembles suivants : $\mathcal{Q}$ est l'ensemble de points initialisés par l'interpolation, $\mathcal{A}$ est l'ensemble de points marqués comme actifs obtenus par $\mathcal{A} = \{q_i \mid q_i \notin \mathcal{Q} \text{ et } V_4(q_i) \cap \mathcal{Q} \neq \emptyset\}$.

Algorithme 3 (Massive Marching)
Initialisation

- *Détecter la courbe initiale par interpolation. Les points interpolés appartiennent à l'ensemble $\mathcal{Q}$ avec les voisins de la courbe.*
- *Initialiser* $u(p) = \infty,\ \forall p_i \notin Q$.
- *Marquer comme actifs les points adjacents de l'ensemble $\mathcal{Q}$, ils constituent $\mathcal{A}$.*

Propagation

Tant que $\mathcal{A} \neq \{\}$, *faire en parallèle pour tous les points actifs* ($p \in \mathcal{A}$) :

{

⋆ *Calculer la nouvelle valeur de distance :*

- *Etape de Jacobi :*

$$u^{n+1}(p) = u^{n}_{min}(p) + \min\{f_{diff}(V_4(p), \mathcal{F}(p)),\ \mathcal{F}(p)\} \tag{4.5}$$

- *Etape de Gauss-Seidel :*

$$u^{n+1}(p) = u^{n+1}_{min}(p) + \min\{f_{diff}(V_4(p), \mathcal{F}(p)),\ \mathcal{F}(p)\} \tag{4.6}$$

⋆ *Activer des nouveaux points pour l'itération suivante :*

- *Effacer p de $\mathcal{A}$ et insérer p à $\mathcal{Q}$.*
- *Si* $u(p) < NB_{width}$ *alors* $\forall\ q_i,\ q_i \in V_4(p)$ *tel que :*

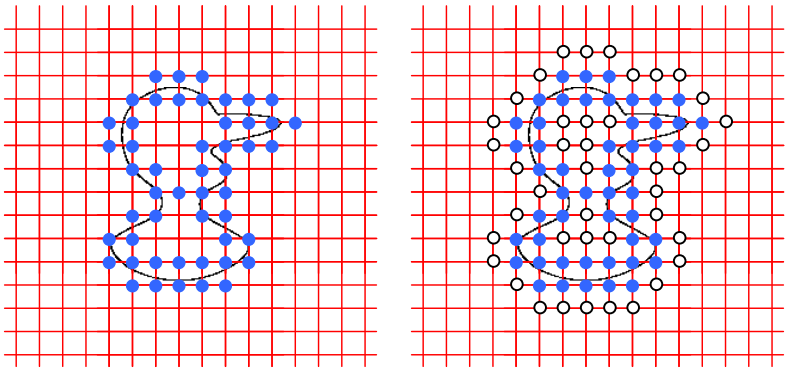
$$u^{n+1}(q_i) - u^{n+1}(p) > \varepsilon(q_i) \tag{4.7}$$

insérer $q_i \rightarrow \mathcal{A}$

}

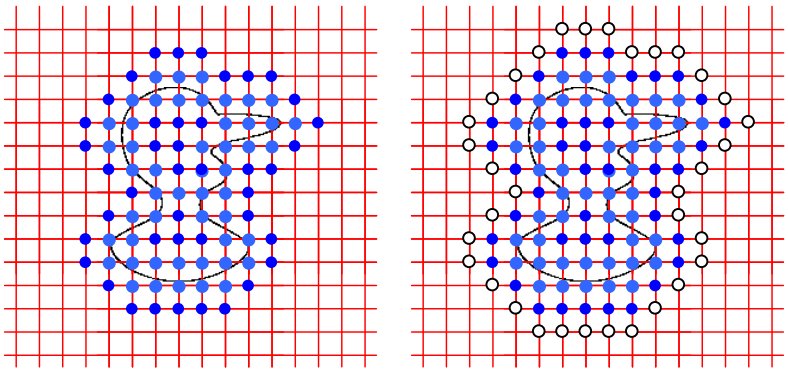
où $\mathrm{NB}_{\mathrm{width}}$ est la demi largeur choisie de Narrow Band. Si nous souhaitons obtenir la solution sur l'image entière, il suffit de mettre $\mathrm{NB}_{\mathrm{width}} = \infty$. Des valeurs des points non traités en

t_n sont automatiquement gardées pour l'itération suivante en t_{n+1}. $\varepsilon(q_i)$ est une constante positive qui sert de critère d'activation. L'obtention de $\varepsilon(q_i)$ est traitée dans la section 4.2.2, page 95.



(a) Détection du contour : les voisins les plus proches du contour sont initialisés à la distance au contour par l'interpolation (points bleus).

(b) Les voisins des points initialisés sont marqués comme *actifs* (points blancs).



(c) Après les deux étapes de calcul Jacobi et Gauss-Seidel, tous les points actifs sont désactivés.

(d) Les voisins des points, qui viennent d'être traités, sont marqués comme *actifs* s'ils satisfont le critère d'activation.

FIG. 4.1 – Premières itérations de l'algorithme Massive Marching.

Dans le contexte de l'analyse de la granularité de parallélisation (chapitre 3), Massive Marching peut être réécrit de la façon indiquée sur la Fig. 4.2. L'algorithme Massive Marching n'utilise aucune structure ordonnée grâce à la propagation non équidistante. Par conséquent, la réalisation hardware des EDPs devient possible et nous obtenons un large éventail des implantations possibles, allant du parallélisme massif à une implantation avec l'approche divide-and-conquer. La parallélisation peut se faire sur plusieurs niveaux :

- Traitement des points entièrement parallèle.
- Opérations arithmétiques et comparaisons sur le voisinage.

Remarque : Il faut assurer que l'étape Gauss-Seidel ne commence pas avant que celle de Jacobi ne soit terminée, à cause de la dépendance réciproque des valeurs des points voisins.

```
// Initialisation :
Pour ∀p ∈ I faire en parallèle
{
      Interpoler ; //initialiser Q
      Initialiser A ;
}
// Evolution :
Tant que u() ≤ NB_w
{
      Pour tous p ∈ A faire en parallèle
            {
                  Extraire voisinage u(V_4(p)) et u(p) ;
                  Calculer la nouvelle valeur u(p) ; // étape Jacobi
                  Mettre à jour u(p) ;
                  Extraire voisinage u(V_4(p)) et u(p) ;
                  Calculer la nouvelle valeur u(p) ; // étape Gauss-Seidel
                  Mettre à jour u(p) ;
                  Activer nouveaux points ;
            }
}
```

FIG. 4.2 – Algorithme Massive Marching

4.2.1 Algorithme à deux étapes

Nous disons que la valeur d'un point est calculée sur un voisinage incomplet puisque les points adjacents (réciproquement dépendants) peuvent être traités simultanément. Le calcul est exécuté par conséquent en deux étapes. Les étapes sont nommées d'après leur similarité avec l'algorithme d'approximation de chaîne de Markov par EDPs comme introduit (Boué and Dupuis, 1999). La première, *l'étape de Jacobi*, calcule la valeur de la fonction de distance à t_{n+1} en fonction des valeurs obtenues à t_n. La seconde, *l'étape de Gauss-Seidel*, recalcule la valeur de distance à t_{n+1} en utilisant aussi les valeurs obtenues à t_{n+1}.

L'algorithme ne calcule la valeur $u(p)$ qu'à partir du voisin minimal. Ainsi les points initialisés à l'infini ne sont pas considérées pour le calcul. Comme mentionné ci-dessus, la répétition du calcul par l'étape de Gauss-Seidel a pour but de préciser la première estimation de $u^{n+1}(p)$ en utilisant les valeurs des voisins.

La Fig. 4.3 montre comment Massive Marching procède pour calculer la fonction distance pondérée à partir des sources données. Nous pouvons y observer comment la valeur de la fonction distance est propagée sur l'image entière (Fig. 4.3(a)) et comment les contours (et des valeurs) déjà calculés sont recalculés grâce au dépassement des différentes ondes (Fig. 4.3(b)).

(a) (b) (c)

FIG. 4.3 – Propagation de la fonction distance. (a) Image initiale et la fonction distance pondérée par des valeurs d'intensité de l'image. (b) Contours de la fonction distance. (c) Points actifs dans l'itération en cours.

La partie Fig. 4.3(c) visualise en blanc les points qui sont marqués comme actifs lors de l'itération en cours. Les points initialisés par l'interpolation autour des sources sont également marqués en blanc.

4.2.2 Critère d'activation

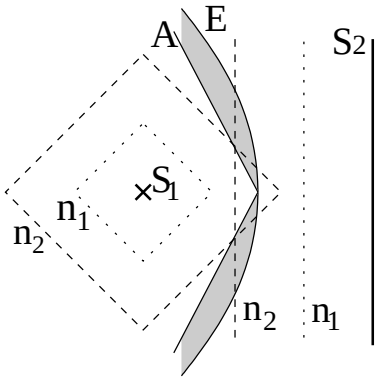


FIG. 4.4 – Exemple des ondes de propagation et de la zone de re-activation (en gris). S_1 et S_2 sont des sources de propagation. Des lignes composées de tirets représentent la position des fronts de propagation après n_1 et n_2 itération.

Le critère d'activation a deux rôles importants : contrôler la fin de la propagation et permettre aux ondes de propagation de se chevaucher. Considérons l'exemple fourni par la Fig. 4.4.

Nous calculons la fonction distance $u(x) = \min[\text{dist}(x, S_1), \text{dist}(x, S_2)]$, S_1 étant un point, S_2 une droite. L'ensemble E contient les points aussi éloignés de S_1 que de S_2, c'est à dire $E = \{x | \text{dist}(x, S_1) = \text{dist}(x, S_1)\}$. Les points à gauche (resp. à droite) de E sont plus près de S_1 (resp. de S_2). Notons que dans cet exemple E est une parabole. A indique l'ensemble des points activés simultanément par les deux fronts venant de deux sources. Puisque le front de propagation de Massive Marching n'est pas équidistant aux sources de propagation, les ensembles A et E ne coïncident pas. La zone délimitée par A et E contient des points qui ont été activés par S_2 alors qu'ils sont plus près de S_1. Ces points seront réactivés par le front venant de S_1 afin de diminuer leur valeur de $\text{dist}(x, S_2)$ à $\text{dist}(x, S_1)$. L'ensemble E représente le SKIZ par u.

Supposons que $u^n(p)$ vient d'être calculé et qu'ensuite p a été désactivé. Nous cherchons un estimateur de $u^{n+1}(q_i)$ pour savoir si le voisin q_i de p devrait être activé afin de calculer ou de recalculer sa valeur. Puisque l'objectif est d'obtenir la solution minimale, l'idée principale est de tester si la valeur $u(q_i)$ pourrait être baissée en considérant p comme le voisin minimal de q_i. Supposons que p est le voisin minimal de q_i, auquel cas, dans l'itération prochaine, q_i recevra sa valeur à partir de p. De l'Eq. (4.2), $f_{diff}(p)$ est la différence entre la valeur de

distance $u(p)$ d'un point donné p et $u_{min}(p)$ le moindre des voisins. La nouvelle valeur $u(q_i)$ satisferait $u(q_i) \geq u(p) + \inf f_{diff}$.

Soit K_{min}la borne inférieure de f_{diff} :

$$K_{\min}(p) = \inf f_{diff}(V_4(p), \mathcal{F}(p)) \tag{4.8}$$

$\mathcal{F}(p)$ est une fonction arbitraire mais invariante dans le temps. $K_{\min}$ est le prédicteur du plus petit incrément de u dans une itération. Tous les voisins q_i de p tels que $u(q_i) - u(p) > K_{\min}(p)$ devraient donc être activés à nouveau et recalculés car la nouvelle valeur $u(p)$ pourrait affecter $u(q_i)$ dans l'itération suivante. ε dans l'Eq. (4.7) doit satisafire :

$$\varepsilon(p) \geq K_{\min}(p) > 0 \tag{4.9}$$

Remarque : La borne inférieure de f_{diff} du schéma Godunov est

$$K_{\ \min}(p) = \mathcal{F}(p)\frac{1}{\sqrt{2}} \tag{4.10}$$

K_{min}est constant si $\mathcal{F}$ est constant et devient une fonction de $\mathcal{F}$ si $\mathcal{F}$ varie sur l'image.

Le réglage $\varepsilon < K_{\min}$ serait inutile parce qu'il autoriserait l'activation des points dont la valeur ne pourrait être diminuée et la propagation pourrait marcher en arrière. Par le choix de $\varepsilon > K_{\min}$ nous pouvons autoriser moins de réactivations (raccourcir le temps d'exécution). Le raccourcissement est en revanche payé par une certaine erreur (proportionnelle à $\varepsilon - K_{\min}$) (Dejnožková, 2002).

TAB. 4.1 – Le nombre de points recalculés (en %). La mesure a été effectuée pour une image 256×256 et la vitesse de propagation constante.

Nombre d'activations	1	2	3	4
$\varepsilon = 1.0$	97.37 %	0.45 %	0.0%	0.0%
$\varepsilon = \sqrt{0.5}$	97.09 %	0.65 %	0.06 %	0.02 %

Le Tab. 4.1 montre le nombre de réactivations des points en fonction du critère d'activation ε. Les mesures ont été effectués pour $\mathcal{F} = 1$ et le schéma de Godunov : $K_{min} = \frac{1}{\sqrt{2}}$. Les valeurs obtenues expriment en pourcentage le nombre de points calculés une ou plusieurs fois. Les points initialisés par l'interpolation et les sources ne sont pas inclus (c'est pourquoi la somme des pourcentages est inférieure à 100%).

Par la suite nous allons montrer que le nombre de réactivation est limité et que, par conséquent, Massive Marching, se termine en un temps d'exécution inférieur à l'infini.

Proposition 4.2.1 *Sur un support dénombrable, l'algorithme Massive Marching s'effectue en un nombre d'itérations fini.*

Démonstration 4.2.1

1. ***Critère d'activation :*** *Nous montrerons qu'un point ne peut réactiver son voisin à partir duquel il a reçu sa valeur. Soit $\mathcal{F}$ la fonction de poids satisfaisant $\mathcal{F} > 0$. Q^n désigne l'ensemble des points actifs lors de n-itième itération, Q_p l'ensemble des points activés par le point p.*

   Considérons un seul point actif dans l'itération n, i.e. $Q^n = \{p\}$. De l'hypothèse sur le schéma numérique (4.2)-(4.4), page 90, p obtient sa valeur u en fonction de son voisinage $V(p)$ déjà, au moins partiellement, connu.

   La valeur $u(p)$ satisfaira : :

$$u(p) \geq \min_{\forall a_i \in V(p)}(u(a_i)) + K_{min}\mathcal{F}\big(\arg\min_{\forall a_i \in V(p)}(u(a_i))\big)$$

   où $K_{min} > 0$. Soit $a = \arg\min_{\forall a_i \in V(p)}(u(a_i))$ L'ensemble des points activés par p pour l'itération suivante :

$$Q_p^{n+1} = \{q_i | q_i \in V(p) \text{ et } u(q_i) > u(p) + K_{min}\mathcal{F}(u(q_i))\}$$

   et l'ensemble de tous les points à traiter :

$$Q^{n+1} = (Q^n \setminus \{p\}) \cup Q_p^{n+1}$$

   par conséquent, $a \notin Q_p^{n+1}$ et par transition $a \notin Q_p^m$, $\forall m > n$.

   Ceci est également valide pour le cas où Q^n contient plus de points actifs. Q^{n+1} ne va contenir que les points dont la valeur est supérieure au plus petit incrément de la fonction distance dans une itération. Ainsi, on ne peut pas ré-activer les points qui ont contribué au calcul ou ceux qui se trouvent au même niveau (avec une tolérance de $K_{min}\mathcal{F}$) que le point en cours de traitement.

2. ***Propagation d'onde :*** *Ensuite, nous montrons l'arrêt de l'algorithme à la rencontre de deux ondes provenant de sources différentes. Nous considérons deux points isolés a, b tels que $a \notin V(b)$ et tels qu'il existe un point c tel que $c \in V(a)$ et $c \in V(b)$. Supposons que les points a et b aient été envahis par deux ondes des différentes sources qui se rencontreront dans c. Le point c est actif. La valeur $u(c)$ satisfaira :*

$$u(c) \geq \min_{\forall k_i \in V(p)}(u(k_i)) + K_{min}\mathcal{F}(\arg\min_{\forall k_i \in V(p)}(u(k_i)))$$

   On fait l'hypothèse que $u(a) < u(b)$ (sans atteinte à la généralité, le cas contraire n'est

pas considéré car symétrique), alors

$$a = \arg \min_{\forall k_i \in V(c)} (u(k_i)) \tag{4.11}$$

et le point c obtient sa valeur de a. La nouvelle valeur $u(c)$ vérifiera l'inégalité

$$u(c) \geq u(a) + K_{min}\mathcal{F}(c)$$

Le point c active pour l'itération $n+1$ ses voisins qui satisfont :

$$Q_c^{n+1} = \{q_i \mid q_i \in V(c) \text{ et } u(q_i) > u(c) + K_{min}\mathcal{F}(q_i)\}$$

Deux cas surgissent :

(a) le point b n'est pas réactivé, c'est à dire

$$b \notin Q_c^{n+1}$$

car $u^n(b) < u^n(c) + K_{min}\mathcal{F}(b)$ et la propagation s'arrête, ou

(b) le point b est réactivé.

Lors de l'itération suivante, le point b obtient sa nouvelle valeur

$$u^{n+1}(b) \geq u^n(c) + K_{min}\mathcal{F}(b) \geq u^{n-1}(a) + K_{min}(\mathcal{F}(c) + \mathcal{F}(b))$$

Soit k_1 un tel point que $k_1 \in V(b)$ et $k_1 \notin V(c)$.
La propagation continue par k_1 si k_1 peut être activé par b, c'est à dire si

$$u^{n+1}(k_1) > u^{n+1}(b) + K_{min}\mathcal{F}(k_1)$$

où $u(k_1)$ est la valeur actuelle de k_1. S'il s'agit de la première activation de k_1 alors $u^{n+1}(k_1) = \infty$, sinon $u^{n+1}(k_1) < \infty$.
Si le point b avait été activé par k_1 alors $u^{n+1}(k_1) < u^{n+1}(b) - K_{min}\mathcal{F}(k_1)$, et k_1 ne pourra pas être réactivé par b, voir la partie 1 de la démonstration. (Le cas où $u^{n+1}(k_1) = \infty$ ne fait pas partie de la démonstration.)
La nouvelle valeur du point k_1 sera :

$$u^{n+2}(k_1) > u^{n+1}(b) + K_{min}\mathcal{F}(k_1) > u^{n-1}(a) + K_{min}(\mathcal{F}(k_1) + \mathcal{F}(b) + \mathcal{F}(c))$$

Ensuite, la propagation de l'onde continue jusqu'au point $u(k_i)$ qui obtiendrait une nouvelle valeur

$$u^{n+i+1}(k_i) > u^{n-1}(a) + K_{min}(\mathcal{F}(k_i) + \cdots + \mathcal{F}(k_1) + \mathcal{F}(b) + \mathcal{F}(c))$$

si sa valeur actuelle est

$$u^{n+i}(k_i) < u^{n-1}(b) - K_{min}(\mathcal{F}(k_i) + \cdots + \mathcal{F}(k_1))$$

par comparaison de $u^{n+i+1}(k_i)$ *et* $u^{n+1}(k_i)$ *on obtient*

$$u^{n-1}(b) - K_{min}(\mathcal{F}(k_i) + \cdots + \mathcal{F}(k_1)) > u^{n-1}(a) + K_{min}\big(\mathcal{F}(k_i) + \cdots + \mathcal{F}(k_1) + \mathcal{F}(b) + \mathcal{F}(c)\big)$$

d'où

$$\frac{u^{n-1}(b) - u^{n-1}(a)}{2K_{min}} > \sum_{i=1}^{N} \mathcal{F}(k_i) + \frac{\mathcal{F}(b) + \mathcal{F}(c)}{2} \tag{4.12}$$

Lors d'une rencontre de deux ondes dans le point c en itération n, arrivant par les points a et b, l'onde plus lente va traverser l'onde plus rapide ; sa propagation continuera et elle s'arrêtera au point k_N *en N itérations au pire.*

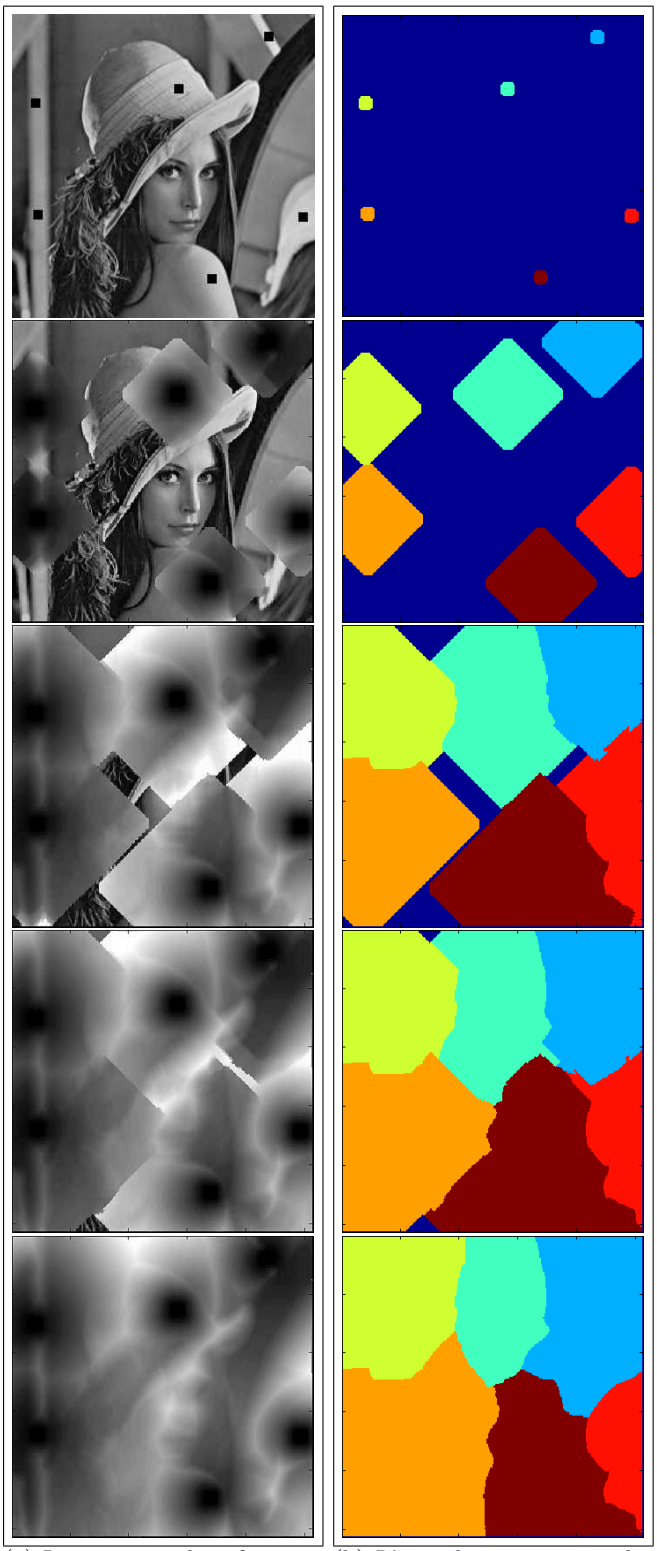
L'équation (4.12) peut être interprétée comme suit. Plus la différence des vitesses de deux ondes est grande $|u^{n-1}(b) - u^{n-1}(a)|$, plus loin ira le dépassement. Si les deux ondes arrivent avec la même vitesse ($|u^{n-1}(b) - u^{n-1}(a)| < K_{min}(\mathcal{F}(a) + \mathcal{F}(b))$), la propagation s'arrête.

Egalement, plus le poid $\mathcal{F}$ est localement important, moindre sera le dépassement des ondes.

4.2.3 Propagation des étiquettes

La propagation des étiquettes des régions peut être exécutée simultanément avec le calcul de la fonction de distance pour obtenir les zones d'influence du diagramme de Voronoï ou de la LPE continue. Les étiquettes des régions sont initialisées pendant l'étape d'initialisation de Massive Marching. Il suffit d'une légère modification de l'algorithme Massive Marching. Désormais, la fonction distance doit être calculée à partir des voisins ayant la même étiquette.

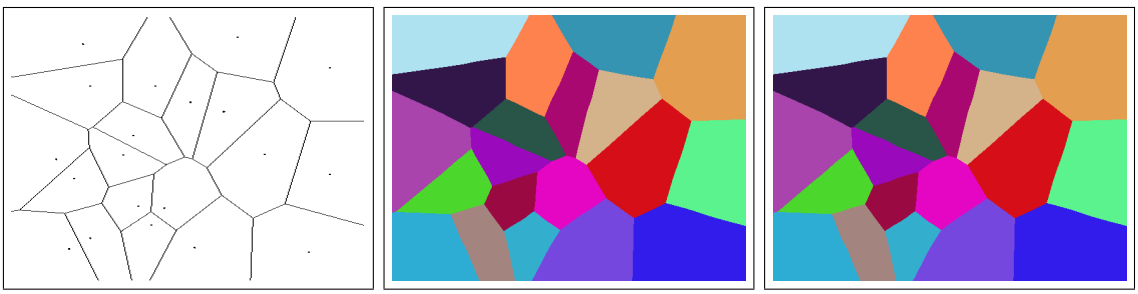
- *Jacobi step*
  Si $u_x(p)$ et $u_y(p)$ ont les mêmes étiquettes, calculer l'Eq. (4.5)
  sinon calculer : $u^{n+1}(p) = u^n_{min}(p) + F(p)$
- *Gauss-Seidel step*
  1. Si $u_x(p)$ et u_y ont les mêmes étiquettes, calculer l'Eq. (4.5)
     sinon calculer $u^{n+1}(p) = u^{n+1}_{min}(p) + F(p)$
  2. p reçoit l'étiquette du voisin minimal dont la valeur est $u_{min}(p)$



(a) Image originale et fonction distance pondérée superposée. (b) L'état de propagation des étiquettes en itération donnée.

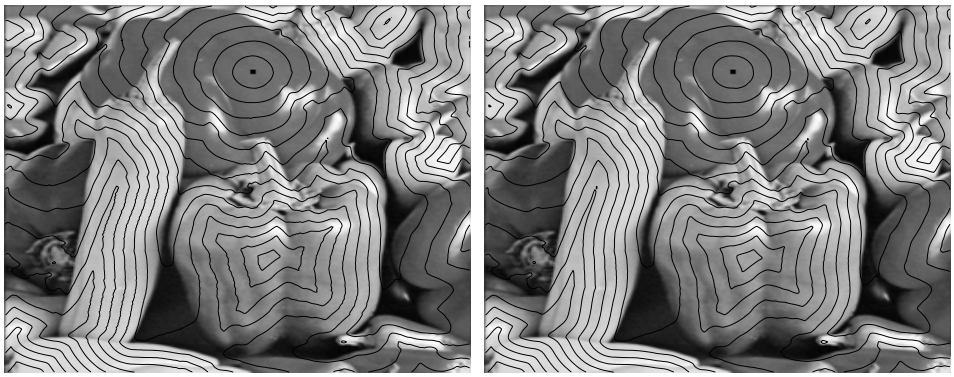
FIG. 4.5 – Propagation simultanée de la fonction distance et des étiquettes

La propagation simultanée des valeurs de u et des étiquettes des régions est présentée sur la Fig. 4.5. La deuxième colonne (Fig. 4.5(b)) visualise l'état de la propagation des étiquettes par rapport au calcul de la fonction distance pondérée (Fig. 4.5(a)) dans différentes itérations. Nous pouvons observer le principe du dépassement des ondes avec des vitesses différentes. D'abord, les ondes envahissent l'espace de l'image en fonction de la définition du voisinage utilisé pour le schéma numérique. A la rencontre de deux ou plusieurs ondes, les plus "rapides" s'arrêtent et la plus "lente" continue à propager son étiquette.



(a) Triangulation de Delaunay. (b) Zones d'influence par Massive Marching. (c) Zones d'influence par Fast Marching.

FIG. 4.6 – Comparaisons des différents algorithmes : diagramme de Voronoï.



(a) Massive Marching (b) Propagation équidistante

FIG. 4.7 – Comparaison des différents algorithmes : distance pondérée.

4.2.4 Erreur de calcul

Nous avons deux types d'erreur : erreur du schéma numérique et erreur dûe au calcul sur un voisinage incomplet.

Erreur du schéma numérique

Comme la majorité des schémas du premier ordre, le schéma de Godunov souffre aussi d'une certaine "myopie". Le schéma de Godunov utilise seulement le 4-voisinage le plus proche. Ainsi nous avons besoin de deux itérations pour estimer la distance dans la direction diagonale par rapport à une seule dans les directions des axes de coordonnées. Si deux ondes avec des vitesses comparables se rencontrent, l'une arrivant horizontalement et l'autre dans une direction diagonale, l'onde diagonale est plus lente que l'onde horizontale. L'onde horizontale pourrait envahir certains points alors que ces points appartiennent à l'onde diagonale (Danielsson, 1980).

Tsai (2000) montre que quand la propagation commence d'un point isolé, ce schéma crée des formes de losange au lieu de cercles. (Sethian, 1999) propose un mécanisme de commutation entre le premier et le deuxième ordre afin d'améliorer la précision. Néanmoins, Godunov converge vers la solution exacte à ∞.

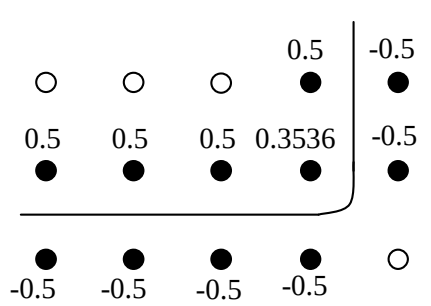
Erreur de voisinage incomplet

Tous les algorithmes de la fonction distance référencées dans l'introduction permettent de recalculer les points afin d'obtenir la solution plus précise de l'Eq.(4.1). Les méthodes des balayages successifs recalculent pendant chaque balayage tous les points de l'image. La succession des balayages doit être répétée jusqu'à la convergence.

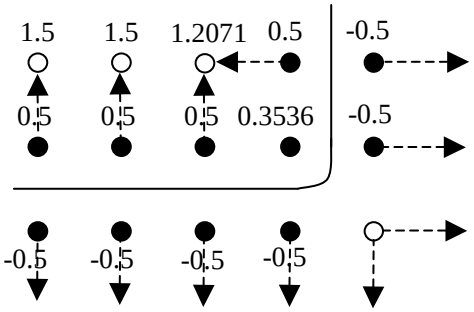
Les méthodes à propagation équidistante, exécutées en utilisant un tas ordonné, recalculent certains points plusieurs fois. Cela dépend localement du voisinage de chaque point particulier. A chaque nouveau calcul, le point reçoit une nouvelle valeur de la distance selon les nouvelles valeurs des voisins.

L'algorithme Massive Marching calcule simultanément les valeurs de points adjacents. Les valeurs des points adjacents peuvent s'influencer mutuellement. Ainsi, le calcul est exécuté en deux étapes : Jacobi et Gauss-Seidel. Une erreur additive (typiquement à la quatrième place décimale) apparaît dans des cas spéciaux (Fig. 4.8). Les expériences ont montré que le calcul de deux étapes donne une précision suffisante pour la plupart des applications pratiques (section 2). Si des résultats plus précis sont exigés, l'étape de Gauss-Seidel peut être répétée.

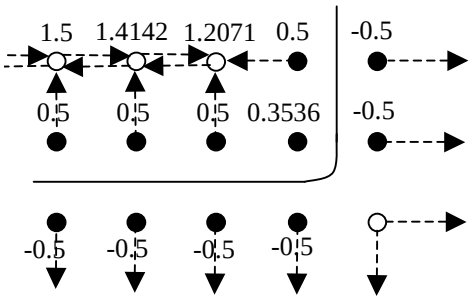
En plus, Massive Marching également autorise de réactiver le voisin q_i si le point p reçoit une nouvelle valeur $u(p)$ inférieure à $u(q_i) - \varepsilon$ (condition de l'Eq. (4.7)).



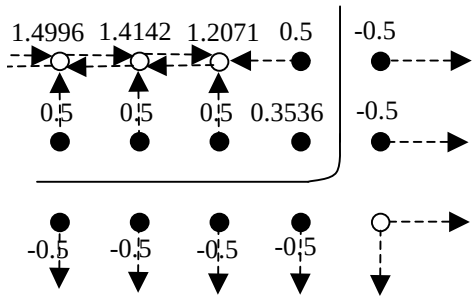
(a) Les points ayant obtenu leurs valeurs par l'interpolation sont marqués en noir. Les points restant sont initialisés à l'infini.



(b) Itération de Jacobi : dans le premier calcul, seules les valeurs résultant de l'interpolation interviennent.



(c) Itération de Gauss-Seidel : les valeurs obtenues dans l'itération de Jacobi sont maintenant utilisées pour la deuxième estimation de u.



(d) Il est possible que des valeurs de certains points soient diminuées par une itération supplémentaire. Massive Marching n'exécute pas de troisième itération. Par conséquent, une erreur sur la quatrième place derrière la virgule peut subsister.

FIG. 4.8 – Erreur de calcul sur le voisinage incomplet. La ligne indique la position du contour décrite par les valeurs de fonction distance signée des points. Les flèches montrent quels points sont considérés pour le calcul des nouvelles valeurs.

4.2.5 Complexité de calcul et temps d'exécution

Pour cette analyse nous séparons la notion de complexité de calcul, qui est liée aux accès aux données et à la difficulté de l'implantation de l'algorithme, et la notion du nombre d'opérations, qui aide à estimer le temps d'exécution en fonction du nombre de points à traiter.

4.2.5.1 Complexité de calcul

Quelle que soit la vitesse de propagation $\mathcal{F}$, la valeur d'un point actif p est obtenue dans un temps constant $\mathcal{O}(1)$ après quoi le point lui-même devient inactif. Le point active tous ses voisins qui vérifient la condition (4.7). La fonction (4.2) est strictement positive, aucun point ne peut réactiver le voisin duquel il a reçu l'activation.

Par conséquent, la complexité de calcul est linéaire $\mathcal{O}(N)$ où N est le nombre de points à traiter.

4.2.5.2 Nombre d'opérations

Le dépassement des ondes autorisé par Massive Marching peut impliquer que le nombre de points à traiter dépasse le nombre de points dans l'image N. Le nombre de points à traiter varie en fonction du nombre des sources dans l'image et en fonction des variations de la vitesse de propagation $\mathcal{F}$.

Afin d'estimer le nombre d'opérations nous considérons d'abord $\mathcal{F} = const.$ sur l'image entière.

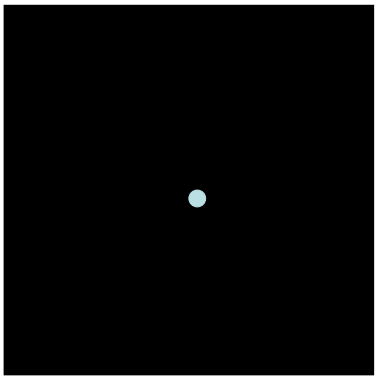
Si la propagation est faite à partir d'une seule source (Fig. 4.9(a)), le nombre d'opérations atteint la limite inférieure $O(N)$ donnée par la complexité de calcul de l'algorithme.

Pour la propagation à partir de plusieurs sources (Fig. 4.9(b)) ou de sources ayant des formes géométriques plus compliquées, la complexité peut dépasser $O(N)$ puisque quelques points peuvent être activés plus d'une fois à cause des dépassements. L'onde plus lente continuera sa propagation jusqu'au squelette de la distance où elle s'arrête. Nous avons montré que le nombre de réactivations est limité. Pour les images bornées le terme $|u(b) - u(a)|$ représente une limite supérieure (Eq. 4.13).

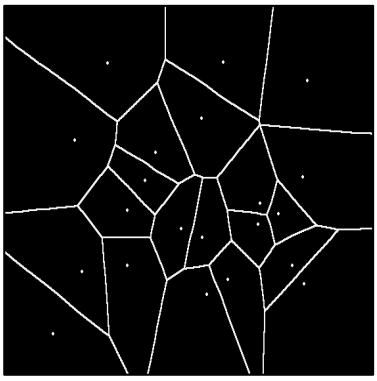
$$\frac{|u(b) - u(a)|}{2K_{min}} - 1 > N_{\text{itérations}} \tag{4.13}$$

dans les images avec $\mathcal{F} = 1$.

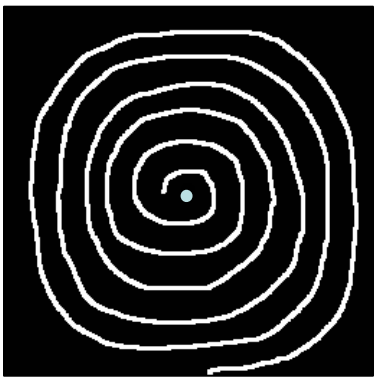
Par conséquent le nombre de réactivations est borné. Autrement dit, la propagation s'arrêtera après $N_{\text{itérations}}$ itérations. Voir l'illustration sur la Fig. 4.4.



(a) Une source dans l'image plate $\mathcal{F} = const.$



(b) Plusieurs sources dans l'image plate $\mathcal{F} = const.$



(c) Une (ou plusieurs) source(s) dans l'image avec $\mathcal{F} \neq const.$

FIG. 4.9 – Exemples avec différents nombres d'opérations.

Dans l'exemple pratique du diagramme de Voronoï (Fig. 4.9(b)), le dépassement des ondes est provoqué par le fait que les ondes se propagent avec la graduation d'unité de u dans une répétition dans la direction verticale et horizontale tandis que dans les directions diagonales la graduation est obtenue après deux itérations seulement.

Le nombre de points réactivés est minimal, le nombre d'opérations est estimé à $O(N+k_1)$, k_1 étant une constante indépendante de N.

Algorithme	Temps d'exécution (s)	Bande passante (points/s)
Propagation équidistante	4.3	0.55×10^5
Massive Marching	2.03	1.16×10^5

TAB. 4.2 – Comparaison des temps d'exécution pour une image 560×420 (Fig.) sur Pentium III sous Linux à 450MHz et des bandes passantes.

Pour les cas où $\mathcal{F} \neq const.$, l'estimation du nombre d'opérations est très complexe car dépendant de $\mathcal{F}$ (voir démonstration 4.2.1), du nombre de sources et de leur forme en même temps. Néanmoins, le nombre d'opérations reste toujours borné comme le montre l'Eq. 4.12.

Dans le cas extrême, présenté sur la Fig. 4.9(c), le nombre de points recalculés est élevé parce que la réactivation des points peut engendrer la propagation des ondes sur une grande partie de l'image. Par conséquent, le nombre d'opérations peut être estimé à $O(k_2N)$ où k_2 est une constante indépendante de N. Les expériences effectuées ont toutefois montré que le temps de traitement est toujours de même ordre de grandeur par rapport à d'autres algorithmes.

4.3 Conclusions

Dans ce chapitre, nous avons présenté un nouvel algorithme parallèle, appelé Massive Marching, pour calculer la fonction distance pondérée. Massive Marching propage la solution de l'équation Eikonale à partir d'un contour initial qui peut être placé avec la précision sous-pixélique. Grâce au mécanisme de propagation non équidistante, Massive Marching ne nécessite d'utiliser aucune structure des tas ordonnés. La complexité de calcul est proche de $\mathcal{O}(N)$. Or, Massive Marching permet de propager des étiquettes des régions de manière simultanée aux calculs de la fonction distance.

Nous avons également présenté l'étude de l'erreur de Massive Marching et nous avons démontré que Massive Marching se termine dans le temps fini. De plus, nous avons montré des résultats obtenus et nous avons comparé la bande passante de Massive Marching et de l'algorithme à propagation équidistante.

L'algorithme Massive Marching permet une implantation de façon séquentielle, semi parallèle ou massivement parallèle. Dans la partie suivante de ce document, nous allons étudier

la faisabilité et la performance de l'implantation matérielle du Massive Marching.

Une importante contribution est la réécriture du schéma numérique de façon quelle permet d'employer plus tard, dans l'implantation matérielle, des diverses approximations de type linéarisation par morceaux ou LUT.

Deuxième partie

Architecture

Chapitre 5

Architectures parallèles

L'analyse préalable des algorithmes nous a permis d'identifier les contraintes suivantes pour la conception d'une architecture dédiée. Elles sont imposées par la nature des méthodes aux EDPs.

- Calculs en nombres réels.
- Calculs de fonctions non linéaires.
- Réalisation d'algorithmes aux accès aléatoires à la mémoire.
- Réalisation d'algorithmes à balayage de l'image entière.

Les calculs en nombres réels sont nécessaires afin de pouvoir préserver l'évolution continue avec une précision sous-pixélique. Les deux possibilités, représentation en virgule flottante et fixe, sont envisageables. Néanmoins, il est à déterminer quelle représentation d'un nombre réel peut être utilisée sans atteinte à la précision acceptable des résultats. En effet, dans la plupart de cas, nous sommes amenés à calculer des fonctions non linéaires pour connaître la vitesse de propagation.

A ces contraintes nous ajoutons des objectifs technologiques que nous visons :

- Système enfoui.
- Traitement temps réel.
- Programmabilité.

Selon la définition introduite par (Wong, Vassiliadis and Cotofana, 2002), un système enfoui est un élément d'un système plus large et il fournit à ce système des services dédiés sous des contraintes temporelles. Le système complet visé est montré sur la Fig. 5.1.

Il s'agit d'un système autonome, qui peut faire partie d'un système plus large, pour les applications liées à l'analyse d'image comme : contrôle de qualité de fabrication, vidéo surveillance ou assistance à la conduite des automobiles. Le système comporte un capteur d'images

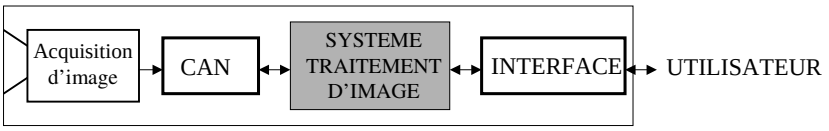


FIG. 5.1 – Système enfoui.

converties en images numériques à l'aide d'un convertisseur analogique numérique CAN (le convertisseur est souvent intégré dans le capteur). Le système de traitement d'image extrait des informations pertinentes pour l'utilisateur. Les informations peuvent être communiquées via une interface sous différentes formes, par exemple image, texte ou son.

Par définition, un système enfoui doit fournir des services dans des délais déterminés. Les délais se traduisent en contraintes temporelles que nous avons appelées traitement en temps réel. Dans le milieu des multimédia et des applications industrielles les contraintes de temps réel sont de l'ordre de quelques millisecondes et se situent généralement entre 10 et 25 images traitées par seconde.

Parmi les objectifs visés, nous avons ajouté la programmabilité de l'architecture. Cette mesure devrait assurer l'obtention d'une palette plus large d'opérateurs et faciliter la réutilisation de cette architecture.

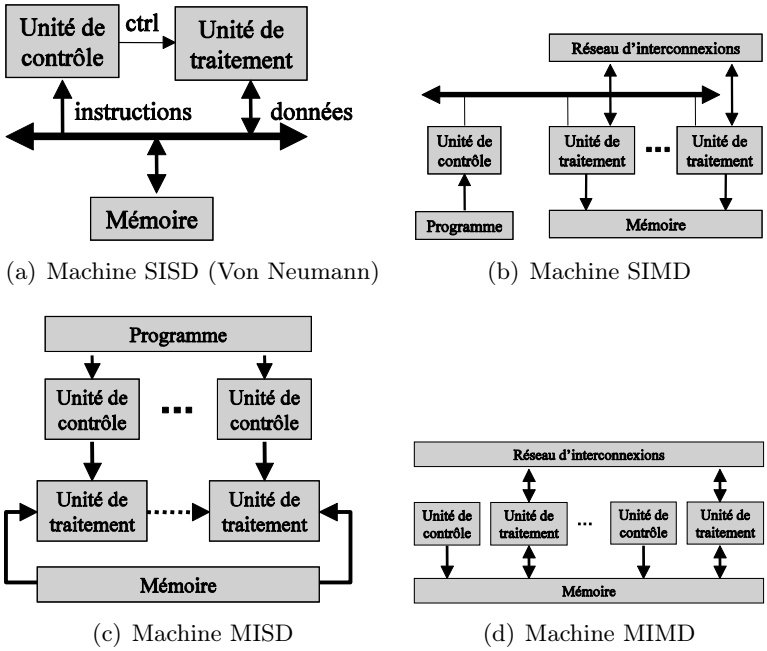
Pour les contraintes fixées, nous disposons d'un choix parmi de nombreux modèles d'architectures parallèles. Flynn (1966) a établi une classification d'une grande notoriété :

SISD : **Single Instruction Single Data**. Cette architecture est la plus largement utilisée. Une opération est effectuée sur un ensemble de données à chaque instant. L'architecture conventionnelle de Von Neumann et Harvard sont deux exemples classiques.

SIMD : **Single Instruction Multiple Data**. Les unités de traitement sont répliquées et gérées par une seule unité de contrôle. La même opération est effectuée en même temps sur différents ensembles de données. En tant qu'exemple, nous pouvons citer les réalisations des calculateurs *Connection Machine* ou *MasPar MP-1 systems* basés sur un parallélisme massif.

MISD : **Multiple Instructions Single Data**. Ce type d'architecture est caractérisé par un nombre donné d'unités de traitement que exécutent différentes opérations sur le même ensemble de données. Souvent, ce type de traitement est appelé pipe-line.

MIMD : **Multiple Instructions Multiple Data**. Des opérations différentes sont effectuées sur un flux de données multiples, par exemple *CM-5* ou *Intel Paragon*. Chaque unité de traitement peut travailler indépendamment des autres. Ce type d'architecture est caractérisé par un niveau d'asynchronisme d'exécution entre les unités de traitement.



(a) Machine SISD (Von Neumann) (b) Machine SIMD

(c) Machine MISD (d) Machine MIMD

FIG. 5.2 – Exemples des différents types d'architecture (Noguet, 2002)

Les catégories les plus efficaces utilisées sont SIMD et MIMD. Les systèmes MIMD sont plus puissants que SIMD mais plus coûteux et complexes à mettre en œuvre. La Tab. 5.1 indique trois points majeurs faisant la différence entre SIMD et MIMD.

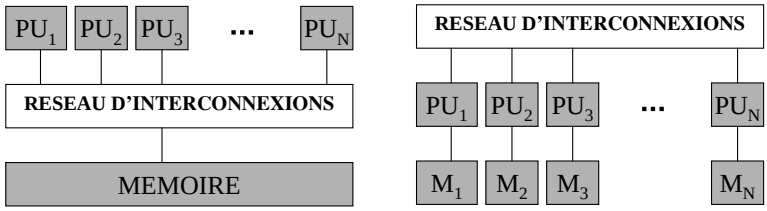
TAB. 5.1 – Différences de fonctionnement des systèmes SIMD et MIMD.

	SIMD	MIMD
Opérations à un instant donné	*Identiques*	*Différentes*
Adresses de mémoire	*Identiques*	*Différentes*
Réception des valeurs des voisins	*Dans le même instant*	*Indépendant*

La discussion des différentes configurations des processeurs (mailles, arbres, pyramides etc.) peut être trouvée dans (Cypher and Sanz, 1989) ou (Bieniek, 2000). Des détails directement liés à notre recherche seront présentés dans les sections correspondantes au cas par cas.

- Mémoire partagée : tous les processeurs accèdent à la même mémoire.
- Mémoire distribuée : chaque unité de traitement dispose d'un bloc privé de mémoire.

En général, les systèmes avec la mémoire partagée sont plus faciles à programmer car l'utilisateur ne doit pas gérer les opérations de communication. En revanche, les systèmes avec mémoire distribuée sont plus efficaces parce qu'il n'y a que des accès locaux à la mémoire. Pour cette raison, une grande partie des architectures dédiées au traitement des images emploie la catégorie des systèmes à mémoire distribuée.



(a) Mémoire partagée (b) Mémoire distribuée

FIG. 5.3 – Mémoire partagée et distribuée. Une unité de traitement est marquée comme PU ; une mémoire est marquée comme M.

Les problèmes engendrés par ces choix sont le *mapping* d'une application sur l'architecture en question et la communication efficace à l'intérieur du système. Le choix de l'algorithme et du type d'architecture vont toujours ensemble.

Pour un *mapping* efficace, les créateurs des systèmes font souvent appel à la technique de *Divide-and-Conquer*. Il s'agit d'une division d'un problème en sous problèmes de nature similaire (Gibbons and Rytter, 1988). En générale cette division algorithmique correspond à la distribution physique des tâches entre des unités de traitement ou entre des modules d'un processeur. Une approche classique de type *Divide-and-Conquer* est de diviser une image en sous images, chacune traitée par un processeur.

Des exemples des architectures existantes vont du parallélisme massif avec une unité de traitement par point ou semi parallèle avec une unité de traitement par bloc d'image. Manzanera (2002) applique le parallélisme massif à la rétine artificielle. L'architecture proposée est dotée d'un niveau d'asynchronisme. A la fréquence 10 MHz, elle permet d'obtenir la LPE en moins de 1 ms (sur image 200×200). Un autre exemple de parallélisme massif est présenté dans (Broggi, Conte, Gregoretti, Sansoè and Reyneri, 1994). Une approche SIMD semi parallèle est introduite dans (Kyo, Koga and Okazaki, 2001). Des processeurs sont réalisés avec la technologie VLIW (Very Large Instruction Word). Ils disposent des instructions spécialisées à la propagation des étiquettes des régions. Les performances indiquées par les auteurs sont des unités de millisecondes pour une fréquence de 100 MHz. Le problème majeur des architectures VLIW est que la portabilité des programmes n'est pas possible en cas de changement du nombre de processeurs. L'approche semi parallèle est également employée par (Peyrard and Klein, 1991) pour la conception d'un ASIC de morphologie mathématique. Le système comprend douze unités de traitement et permet de traiter cinq images par seconde. (Lemonnier, 1996) continue dans cette axe en intégrant des nouvelles approches algorithmiques pour des files d'attente.

Parmi des exemples d'architectures MIMD, nous pouvons citer (Moga, Viero, Dobrin and Gabbouj, 1994) qui décrit comment rechercher des minima régionaux et effectuer l'étiquetage des régions sur ce type d'architecture. Les auteurs utilisent le mode SPMD (Single Program Multiple Data) où chaque processeur exécute le même programme de manière asynchrone.

(Noguet, 2002) introduit une architecture MIMD pour le calcul de la LPE. L'effort principal est porté sur la minimisation des besoins de synchronisation lors des calculs afin d'obtenir des performances maximales. Dans les deux cas, les temps de traitement sont de l'ordre de dizaines ou centaines de millisecondes en fonction de la taille de l'image (jusqu'à 512×512).

Discussion des architectures

Le développement croissant des multimédia et des communications influence de plus en plus la conception des processeurs. Par conséquence, les processeurs intègrent de plus en plus de fonctionnalités du domaine DSP et des multimédia. Pour exprimer la complexité des processeurs, le terme *granularité* est utilisé dans la littérature (Miller and Stout, 1999) :

- Fine : la machine est composée d'un grand nombre de processeurs simples.
- Grosse : la machine comporte moins de processeurs mais ils sont plus complexes.

La fine granularité est typique des machines SIMD qui opèrent de manière synchrone sur la mémoire locale privée. En revanche l'architecture MIMD représente les processeurs à grosse granularité qui travaillent avec la mémoire partagée. Miller and Stout (1999) parle également d'une granularité moyenne utilisant des machines situées entre les deux modèles ci-dessus. Le plus souvent elles sont construites comme des machines MIMD fondées sur des processeurs à usage général soit avec la mémoire partagée soit avec la mémoire distribuée. Les machines à granularité moyennes sont les plus présentes sur le marché actuel grâce à un bon rapport coût/performance. De manière similaire, la granularité peut décrire également la complexité des données ou des algorithmes.

Le traitement d'image est caractéristique d'un parallélisme avec différents niveaux de granularité (Debes, 2003) :

- Niveau de données
- Niveau de processus
- Niveau d'instruction

En revanche, la réutilisation des données au cours d'une même itération est relativement peu fréquente par rapport aux autres applications. Les applications multimédia ont en plus des contraintes temporelles. Debes (2003) conclut que des applications multimédia peuvent être essentiellement implantées sur deux classes d'architectures basées sur des processeurs à usage général :

- Des systèmes du type des puissants ordinateurs de bureau , limités par une consommation d'énergie croissante.
- Des systèmes alimentés par piles où la performance est limitée par l'énergie disponible.

Dans les deux cas, l'énergie joue un rôle important et nous ramène à sur la nécessité de paralléliser les calculs afin de pouvoir baisser les fréquences d'exécution.

Dans le passé, une autre caractéristique des processeurs embarqués était une structure statique, c'est à dire que le processeur optimisé n'était pas à usage général et que l'utilisateur avait des possibilités limitées de programmation. Aujourd'hui, le temps de développement doit être de plus en plus court et le coût de plus en plus bas. L'architecture doit donc être flexible et réutilisable. Dans cet objectif, les fonctions des processeurs sont séparées en deux groupes : critique et non critique dans le temps. Les fonctions non critiques peuvent être exécutées sur des circuits moins rapides, en revanche, les fonctions critiques doivent être implantées avec la performance maximale (Wong et al., 2002). Deux approches principales existent pour la conception des nouvelles architectures :

- Adapter une architecture à usage générale. Des modifications de telles architectures sont nécessaires car souvent elles ne sont pas destinées aux systèmes enfouis.
- Concevoir un nouveau processeur enfoui à partir d'un modèle. La conception est plus longue mais l'architecture résultante est même adaptée à l'application en question.

Au départ, le domaine des processeurs embarqués était gouverné par les composants ASIC (Application Spécifique Integrated Circuits). L'effort pour réduire la longue durée de développement a mené à l'utilisation des techniques de reconfiguration à l'aide des FPGAs. Depuis peu, depuis récemment, les performances des ASICs et des FPGAs se rapprochent rapidement. En plus, les FPGAs reconfigurables ont pour avantage une création rapide du prototype, sa correction et sa mise à jour sont faciles. L'un des avantages important est que la programmabilité permet l'utilisation des langages de programmation comme *C*. Une présentation des langages actuels spécialisés en programmation parallèle peut être trouvée dans (Michel, Vito and Sansonnet, 1996).

Actuellement, les processeurs embarqués vont des micro-contrôleurs à 8 bits aux processeurs RISC 64 bits en passant par les DSPs 32 bits. Souvent, il s'agit de processeurs spécifiques à l'application. L'application, étant connue à priori, permet d'optimiser les processeurs et facilite la conception des logiciels en même temps que du matériel. Parmi des exemples de noyaux adaptables selon la première approche, nous pouvons citer les processeurs ARM, PowerPC, ou Motorola Coldfire. La deuxième approche est représentée par processeur le Trimedia VLIW.

Les progrès de fabrication permettent d'intégrer plusieurs processeurs sur un seul composant, le nombre courant de processeurs étant quatre. L'apparition sur le marché de nombreux composants multiprocesseur montre qu'ils vont jouer un rôle important dans le futur des applications des multimédia et des communications. L'avantage des composants multiprocesseur est de pouvoir distribuer des tâches de différente granularité, en plus, les processeurs peuvent partager des blocs des mémoires présentes sur le composant. La communication est plus rapide si les processeurs sont situés sur un même composant.

EDPs et des architectures dédiées

Nous avons montré que dans le domaine des EDPs, le prétraitement ainsi que les algorithmes de segmentation sont contrôlés par des propriétés géométriques des ensembles de niveaux u tel que le gradient ou la courbure. Nous savons que ces propriétés géométriques sont directement obtenues à partir des ensembles de niveaux u ou de leurs dérivées (Sethian, 1996), (Sapiro, 2000). Les dérivées pour chaque point sont indépendantes les unes des autres. La séparation et la parallélisation des dérivées dans des directions différentes représentent le niveau le plus bas de parallélisation exécutée sur le voisinage le plus proche. Le calcul des dérivées est une opération élémentaire et représente ainsi le niveau de granularité fine. D'autres niveaux de parallélisme (granularité d'algorithme) doivent être identifiés au cas par cas pour des différents algorithmes.

A notre connaissance, il existe très peu de réalisations d'architectures dédiées directement au traitement d'image par EDPs. Le problème des méthodes aux EDPs est l'impossibilité actuelle de transformer les calculs des fonctions non linéaires en règles simples comme suite de comparaisons de nombres entiers en préservant l'avantage d'une évolution continue.

Des travaux publiés récemment nous permettent de constater l'attention croissante qui est portée à l'accélération des calculs. Certains auteurs proposent de nouvelles méthodes numériques (Weickert et al., 1998), essentiellement moins sensibles au pas d'intégration, qui permettent de réduire le nombre d'itérations à effectuer. D'autres propositions visent de telles techniques de calculs qui réduisent le nombre de points à traiter (méthode de Narrow Band (Sethian, 1996)) ou qui simplifient la complexité des structures de données (Tsai, 2000).

Beaucoup d'auteurs ont orienté leur recherche à limiter le nombre d'itérations par accélération de convergence en reformulant l'algorithme d'une manière parallèle ou en éliminant le besoin de structures de données ordonnées (Tsai, 2000).

Nous savons maintenant qu'en général, l'application exige de résoudre une EDP non linéaire. La solution numérique est obtenue par les algorithmes récursifs caractérisés par un grand nombre d'itérations. De plus, certains algorithmes font appel à l'usage des structures de données hiérarchiques.

Nous avons vu pourquoi il est préférable d'utiliser des algorithmes de type Narrow Band pour l'obtention de la fonction distance. Dans le chapitre 3 (page 61), nous avons démontré que l'approche classique par des files d'attente hiérarchiques pénaliserait l'efficacité du calcul puisqu'elle impose des calculs séquentiels.

Pour ces raisons, nous allons étudier l'implantation des méthodes aux EDPs à l'aide de l'algorithme Massive Marching (chapitre 4, à partir de la page 89). L'algorithme Massive Marching a été conçu comme entièrement parallèle et permet d'utiliser des architectures de

différents types : massivement parallèle, semi parallèle ou même séquentiel. Nous avons choisi d'étudier l'implantation des algorithmes EDP sur deux types d'architecture :

- Architecture SIMD. Le type SIMD convient au type image entière et nous voulons étudier l'efficacité de l'utilisation de Massive Marching pour la segmentation sur ce type d'architecture.
- Architecture MIMD. Ce type d'architecture est mieux adapté aux algorithmes avec une granularité de parallélisation moins fine ce qui est le cas de la LPE.

Remarque : Afin de pouvoir évaluer les performances des architectures proposées, elles ont été programmées, testées et simulées à l'aide des langages de programmation de matériel : l'architecture SIMD a été réalisée en VHDL et l'architecture MIMD en HandelC.

Chapitre 6

SIMD

En ce qui concerne la résolution des EDPs générales, pas forcément liées au traitement d'image, nous pouvons citer l'architecture orthogonale introduite par Hwang, Tseng and Kim (1989). Selon les auteurs, cette architecture est avantageuse pour les calculs matriciels. Elle est construite avec n unité de traitement et n^2 blocs de mémoires. Chaque processeur est connecté aux mémoires par des bus dédiés à un seul processeur et tous les processeurs sont équipés d'un contrôleur d'accès à la mémoire. L'inconvénient de cette conception est que le nombre d'interconnexions et des bus augmentent considérablement avec le nombre de processeurs. Ceci pourrait se montre comme la limite de l'utilisation pour le traitement d'image où la tendance est de partager le traitement entre un maximum de processeurs.

L'architecture la plus proche des applications de traitement d'images a été présentée par (Gijbels, Six, Gool, Catthoor, Man and Oosterlinck, 1994). Il s'agit d'une architecture conçue pour effectuer une sorte de pré traitement en vue d'amélioration des séquences d'images. Les auteurs utilisent une architecture SIMD avec mémoire distribuée pour la diffusion non linéaire parallèle, c'est à dire pour le type d'algorithme image entière dans une application de vision. Les performances estimées sont de 100 itérations effectuées sur une image 256×256 toutes les 0.25 s alors que les unités de traitement elles-mêmes sont cadencés à 20 MHz.

Dans ce chapitre, nous présentons implantation de l'algorithme Massive Marching sur une architecture du type SIMD. Cette étude a pour but de montrer que, grâce à l'algorithme Massive Marching, l'architecture SIMD est utilisable également pour exécuter les opérateurs de segmentation, c'est à dire le type Narrow Band. En raison de plus est que le schéma de Godunov peut être approximé par la linéarisation par morceaux selon l'équation (4.2), page 90, qui permet d'utiliser la même approche pour les calculs arithmétiques que celle employée par l'architecture de Gijbels.

D'abord, nous décrivons l'architecture globale où nous présentons ses différentes parties et la façon dont elles communiquent entre elles. Ensuite, nous nous consacrons en particulier à

la description de la réalisation de l'unité de traitement et du calcul de Massive Marching.

6.1 Architecture globale

Le schéma bloc de l'architecture proposée est montré sur la Fig. 6.1. Elle consiste en un ensemble de processeurs (PU_i). Le réseau de communication représente toutes les interconnexions des différents modules. Un contrôleur (Contrôle global) diffuse des instructions à tous les processeurs qui les exécutent simultanément à un moment donné.

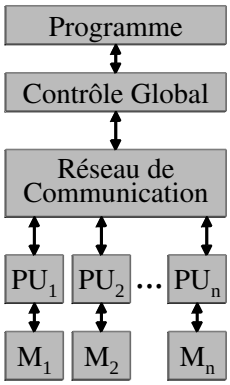


FIG. 6.1 – Schéma bloc d'une architecture SIMD.

Toutes les informations sont transmises par le réseau de communication (Fig. 6.2). Les instructions sont envoyées par le contrôle global vers les unités de traitement (les flèches noires Fig. 6.2). Des informations sont également échangées entre les processeurs voisins (les flèches noires composées de tirets Fig. 6.2). Rappelons que Massive Marching utilise le 4-voisinage. Afin de réduire le nombre d'interconnexions, nous utilisons des bus bidirectionnels pour la communication des PUs adjacents. Pour les algorithmes de type Narrow Band, le réseau de communication est équipé avec le module Arrêt d'algorithme qui collecte des informations sur présence des points actifs dans les parties d'image attribuées aux processeurs (les flèches grises Fig. 6.2).

Chaque processeur possède une mémoire privée non partagée. Nous avons choisi de la diviser en colonnes. Cela signifie qu'une unité de traitement opère sur une colonne de l'image. A un moment donné, les processeurs PU_n traitent en parallèle tous les points sur une ligne de l'image (Fig. 6.3).

Nous allons montrer la gestion de l'accès aux voisins dans la section 6.3.1 ci-après.

Puisque les données sont contenues dans les mémoires non partagées, toutes les unités de traitement peuvent accéder simultanément aux données. Il s'agit de mémoires à double porte. Ainsi, les données sont transmises en utilisant l'un des ports pour l'accès global (accès

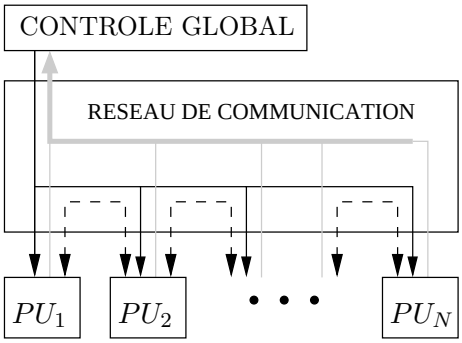


FIG. 6.2 – Réseau de communications

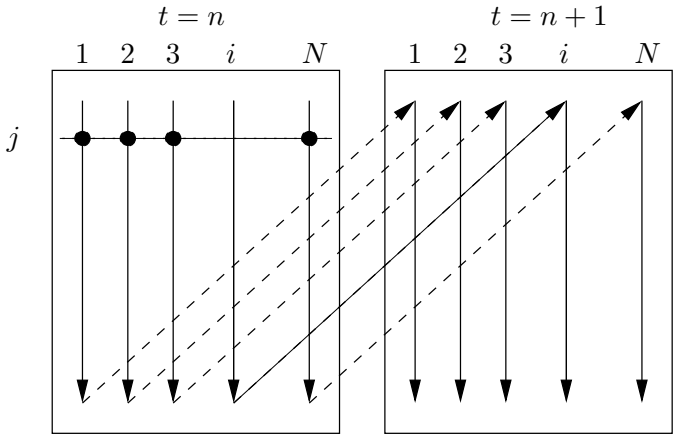


FIG. 6.3 – Partition de l'image dans les mémoires privées des unités de traitement.

à l'ensemble des mémoires) avant et après l'exécution de l'algorithme.

6.2 Contrôle global

Nous avons déjà dit que le Contrôle global est utilisé pour la lecture des instructions de la mémoire de Programme et leur transmission vers les unités de traitement.

Le bloc Contrôle Global gère non seulement l'exécution de l'algorithme mais permet aussi de changer les valeurs des registres internes utilisés pour approximation dans chaque unité de traitement. Ainsi nous pouvons réaliser un large nombre de fonctions non-linéaires.

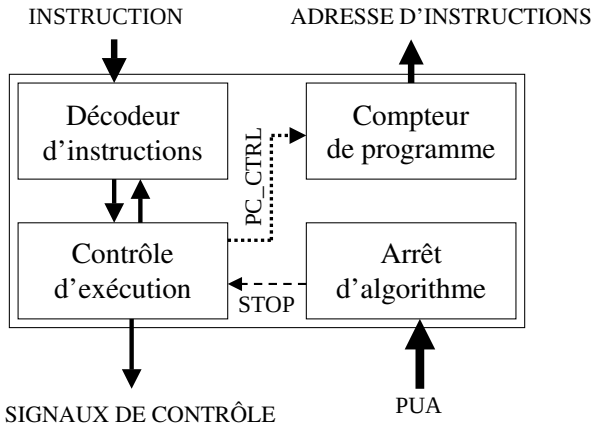


FIG. 6.4 – Schéma bloc du module Contrôle global

Le programme est composé des instructions qui représentent différents micro-codes. Un micro-code consiste une séquence d'opérations exécutées dans certain ordre. Cela signifie qu'une instruction indique directement l'action qui sera exécutée par tous les PUs sur leur partie de la mémoire.

TAB. 6.1 – Format d'instruction. N est le nombre de bits choisi pour la représentation de la partie fractionnelle des nombres en virgule fixe.

N+6	N+5 N	N-1 . . . 0
VAL=1	NOREG	VALUE

N+6	N+5	N+4	N+3	N+2 . . . 0
VAL=0	ACT	INTERP	DIFF	0

Le format d'instruction est présenté sur la Tab. 6.1. Nous distinguons deux formats différant par la valeur du bit VAL. Dans le premier, VAL=1, nous changeons les valeurs des registres utilisés pour l'approximation ou des registres contenant le nombre d'itérations à exécuter. La partie NOREG contient le numéro du registre dont la valeur va être modifiée.

NOREG est long de 6 bits pour adresser jusqu'à 63 registres (30 pour l'approximation et le reste pour des registres de contrôle d'exécution)). La partie restante, VALUE, d'instruction contient la nouvelle valeur. VALUE est long de N-1 bits où N est la précision choisie pour la partie fractionnelle des nombres en virgule fixe. Nous ne travaillons qu'avec la partie fractionnelle car les fonctions non-linéaires à approximer appartiennent dans notre cas à l'intervalle $[0, 1]$.

Le deuxième format, VAL=0, est destiné aux opérations où l'unité de traitement parcourt sa partie de mémoire. Le bit ACT indique si, pendant le parcours des points actifs, l'activation des nouveaux points est exécutée (Tab. 6.2). C'est le cas pendant l'interpolation qui initialise les premiers points actifs et pendant l'étape Gauss-Seidel où nous activons les points pour l'itération suivante. L'interpolation est lancée seulement si le bit INTERP est égal à 1. Dans le cas inverse, fonction distance est calculée mais seulement si la valeur de DIFF est zéro. Or DIFF signal que la diffusion non linéaire va être exécutée. Le cas échéant, le bit INTERP n'a aucune influence et tous les points sont considérés comme actifs.

TAB. 6.2 – OPCODE d'instructions

Instruction	VAL	NOREG ACT INTERP DIFF	VALUE
NonLinearDiffusion	0	0 0 1	X
Interpolation	0	1 1 0	X
JacobiStep	0	0 0 0	X
GaussSeidelStep	0	1 0 0	X
LoadRegister	1	ADRESSE	VALEUR

Une instruction est appliquée à tous les points dans la mémoire appropriée. Avec la fin de parcours de la mémoire, une nouvelle instruction est chargée. L'algorithme pour calculer la fonction distance par Massive Marching sur l'image entière s'écrit avec les instructions sur la Fig. 6.5

```
Interpolation
Tant que STOP = 1
{
      JacobiStep
      GaussSeidelStep
}
```

FIG. 6.5 – Massive Marching en instructions de l'architecture proposée.

La condition tant que $\overline{\text{STOP}} = 1$ est équivalente à la formulation : *au moins un point est actif* et représente ici la détection de la fin de propagation.

6.2.1 Arrêt d'algorithme

Au niveau de l'image entière, l'arrêt d'exécution est contrôlé par le signal STOP (Fig. 6.6). La valeur du signal STOP résulte d'une opération OR effectuée sur l'ensemble des signaux PUA_i (PROCESSING UNIT ACTIVITY) indiquant l'état d'activité de chacun des blocs de mémoire. Les signaux PUA_i sont générés par chaque unité de traitement, le signal PUA avec

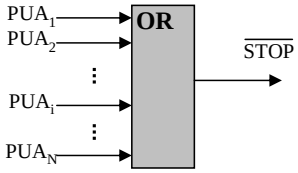


FIG. 6.6 – Le principe de détection de la fin de propagation au niveau global.

l'indice i correspondant à l'unité de traitement PU_i. Pour expliquer comment la valeur de PUA est obtenue, supposons que l'image traitée ait a la taille N×M où N est le nombre de colonnes (et d'unités de traitement) et M est le nombre de lignes (la longueur de chaque bloc de mémoire). Le calcul de la valeur PUA_i est fondé sur la connaissance de l'état d'activité de tous les points dans le bloc de mémoire appropriée à une unité de traitement PU_i. La valeur PUA est obtenue selon l'expression suivante :

$$PUA_i = PUA_{i1} \cup PUA_{i2} \cup PUA_{i3} \cup \cdots \cup PUA_{iM} \tag{6.1}$$

où le deuxième indice indique le numéro de ligne de mémoire ce qui est équivalent au numéro de point dans la mémoire. Puisque les unités de traitement parcourent l'image ligne par ligne, nous profitons de l'associativité de l'opération OR logique et nous calculons la valeur PUA_i au fur et à mesure, lors de la propagation, sur les drapeaux d'activité des points dans son bloc de mémoire :

$$PUA_i = (((PUA_{i1} \cup PUA_{i2}) \cup PUA_{i3}) \cup \cdots \cup PUA_{iM}) \tag{6.2}$$

Le principe du calcul est montré sur la Fig. 6.7. La question qu'il faut poser est combien de points vont être actifs dans l'itération suivante ? D'où nous voyons qu'il faut examiner les demandes d'activation qui sont générées lors de l'itération actuelle. La propagation du résultat intermédiaire (selon l'équation (6.2)) est réalisée à l'aide du retour de la valeur de PUA_i.

Les signaux matérialisant les demandes d'activation sont appelés : ARN (ACTIVATION REQUEST NORTH), ARS (ACTIVATION REQUEST SOUTH), ARW (ACTIVATION REQUEST WEST) et ARE (ACTIVATION REQUEST EST). Nous allons expliquer plus tard comment ces signaux sont générés (section 6.3.5, page 138). Nous avons deux cas :

- Sens horizontal : les demandes d'activation proviennent des voisins à l'est et à l'ouest. Les signaux sont communiqués par des unités de traitement adjacentes $i + 1$ et $i - 1$,

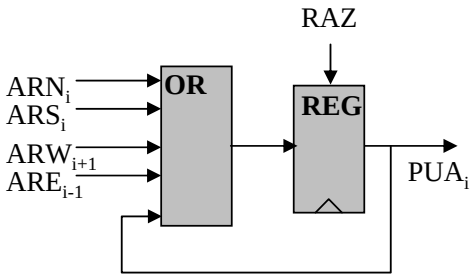


FIG. 6.7 – Principe de détection de la fin de propagation au niveau d'une unité de traitement. Les indices i et j représentent le numéro de l'unité de traitement et de la ligne. Remarque : La valeur de PUA$_i$ est remise à zéro au début de l'image.

nous les appelons : ARW$_{i+1}$ et ARE$_{i-1}$.

- Sens vertical : les demandes d'activation proviennent de l'unité de traitement i et elles sont envoyées vers les voisins nord et sud. Elles sont appelées ARN$_i$ et ARS$_i$

Théoriquement, il faut examiner des demandes d'activation arrivant des voisins car après avoir été traité, un point se désactive et ne peut pas s'activer lui-même. Notons, qu'un point peut être activé ou réactivé seulement à partir des voisins. Pour le test de fin de propagation, il suffit de connaître que nous avons demandé d'activer au moins un point pour le bloc de mémoire appropriée.

Dans le sens vertical, nous examinons des demandes d'activation envoyées (ARN$_i$ et ARS$_i$) vers les voisins par le point en cours de traitement. Cette approche permet de simplifier la matérialisation sans devoir revenir sur l'état d'activation des points déjà traités (des voisins au nord).

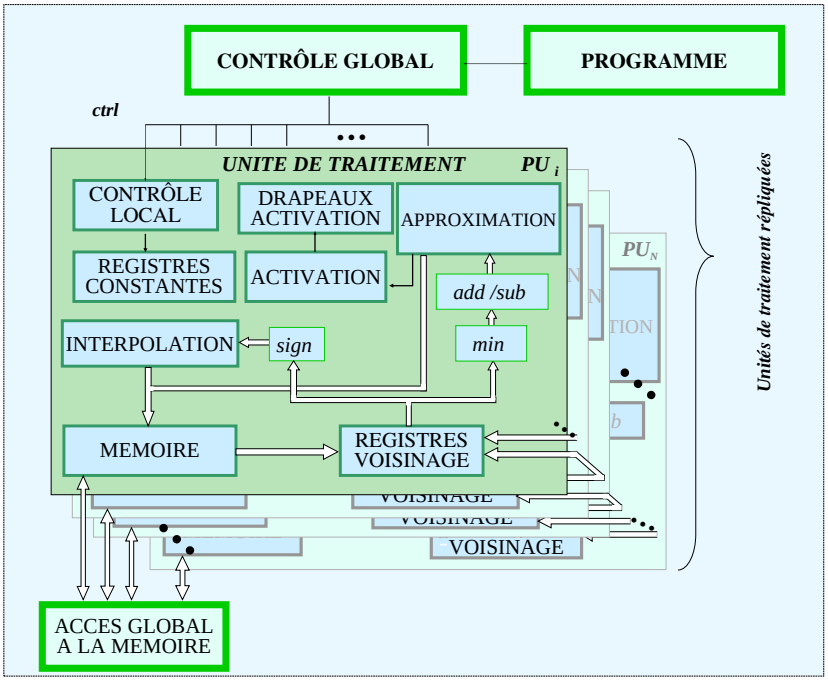
La valeur finale PUA$_i$ est valide après le parcours du bloc entier de mémoire.

6.3 Unité de traitement

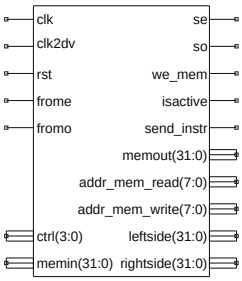
Dans cette section, nous présentons une description des blocs spécifiques exécutant les étapes différentes parties de l'algorithme (Fig. 6.8).

L'action de tous les blocs est gérée par le **CONTROLE LOCAL**. Ce bloc a plusieurs rôles :

- Décoder les instructions.
- Détecter la fin de la propagation.
- Gérer la communication avec les voisins.
- Changer les valeurs de constantes dans **REGISTRES CONSTANTES**.
- Contrôler accès à la mémoire.



(a) Blocs fonctionnels d'une unité de traitement.



(b) Composante matérialisée à l'aide d'un outils de synthèse

FIG. 6.8 – Unité de traitement.

L'exécution du module **INTERPOLATION** est démarrée par l'instruction *Interpolation*. Son rôle est la détection du contour initial avec la précision sous-pixélique par interpolation linéaire. L'**APPROXIMATION** effectue des opérations arithmétiques pour calculer les valeurs des points voisins qui sont contenus dans **REGISTRES VOISINAGE**. Les constantes utilisées pour la linéarisation par morceaux ou LUT sont contenues dans les **REGISTRES CONSTANTES**. Chaque PU possède un registre contenant les drapeaux d'activité des points **DRAPEAUX ACTIVATION**, pour sa partie de l'image. Le drapeau d'activité est utilisé comme un masque pour contrôler l'activité de la PU. Si le point est "actif", la PU effectue des calculs pour ce point. Autrement PU n'assure que la lecture de mémoire et envoie la valeur aux voisins. L'activation des points est assurée par le bloc **ACTIVATION**. Le module **ACCES GLOBAL A LA MEMOIRE** sur la Fig. 6.8 est chargé d'initialiser les mémoires privées et il permet également de lire les résultats du traitement.

L'unité de traitement parcourt une colonne de l'image, le générateur d'adresses est donc réalisé comme un compteur de lignes.

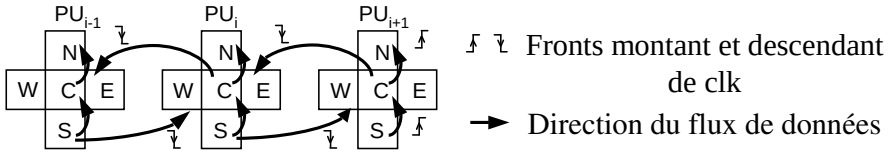


FIG. 6.9 – Directions des échanges des valeurs des points voisins.

6.3.1 Extraction du voisinage complet

Chaque unité de traitement opère sur une colonne de l'image (de 1 à N). Puisque toutes les unités de traitement sont synchronisées, une ligne j des points est traitée à un instant donné. Quand la dernière ligne de l'image est atteinte, les unités de traitement commencent une nouvelle itération $t = n + 1$ sur la première ligne de l'image. Cette division permet de réduire le nombre d'accès à la mémoire. Lors d'un parcours de la colonne, une valeur lue est utilisée trois fois. Les valeurs précédentes du voisin SOUTH et du point à la position CENTER sont décalées vers le nord lors de la lecture et deviennent ainsi CENTER et NORTH pour le point suivant (Fig. 6.9).

Afin de limiter le nombre d'interconnexions entre les unités de traitement, nous avons décidé d'utiliser des bus bidirectionnels pour la communication. Grâce à ceci, nous pouvons économiser jusqu'à 50% des interconnexions ce qui peut sembler moins important au niveau d'une unité de traitement mais, sachant que la tendance est d'avoir le plus d'unités de traitement possible pour réduire le temps de traitement, au niveau de l'ensemble des unités de traitement le nombre d'interconnexions pourrait devenir considérable. Supposons que nous utilisons n_{bits} bits pour coder les valeurs des points, dans le cas des bus bidirectionnels, nous avons $n_{bits}(N-1)$ interconnexions alors que dans le cas d'un bus dans chaque direction, nous aurions $2n_{bits}(N-1)$ interconnexions.

L'extraction du voisinage complet est réalisée à l'aide de deux signaux d'horloge clk et clk/2 avec la période $T_{clk/2} = \frac{T_{clk}}{2}$ (Fig. 6.12). Pour la gestion des bus bidirectionnels de communication, nous utilisons les deux fronts, montant et descendant, de clk ce qui nous permet d'obtenir le voisinage complet d'un point en deux cycles d'horloge. Notons que les calculs arithmétiques sont complètement pipelinés et le résultat de la fonction distance est obtenu également en deux cycles d'horloge (section 6.3.3, page 132).

Pour une meilleure explication du mécanisme d'extraction de voisinage, supposons que les mémoires de l'unité PU_i et des unités voisines PU_{i-1} et PU_{i+1} contiennent les valeurs spécifiées sur la Fig. 6.10.

Nous profitons du fait que l'image est découpée en colonnes pour réduire le nombre d'accès en lecture à la mémoire. Tous les deux cycles de clk, une nouvelle valeur est lue dans la mémoire. Cette valeur est la valeur du point SOUTH. A chaque lecture mémoire, les valeurs présentes

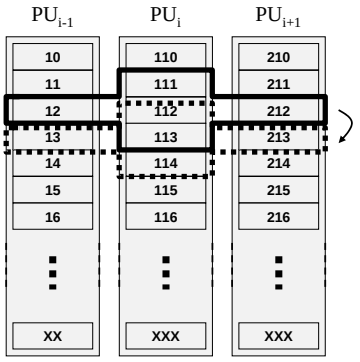


FIG. 6.10 – Exemple du contenu des mémoires pour l'extraction de voisinage.

dans SOUTH et CENTER sont décalées vers le nord et deviennent respectivement CENTER et NORTH (Fig. 6.10).

Les valeurs des voisins EAST et WEST sont échangées via les bus bidirectionnels bus_e et bus_w. La direction des bus ainsi que leur origine et destination sont contrôlées par des signaux t_east et t_west qui fonctionnent comme des commutateurs en fonction des valeurs de clk/2.

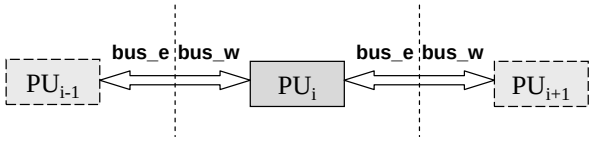


FIG. 6.11 – Bus bidirectionnels de communication

La valeur de l'ouest WEST (W) est lue par le PU adjacent à l'ouest alors que la valeur de point SOUTH (S) est mise à la disposition au PU à l'est. Sur le prochain front descendant le point EAST (E) est lu de l'est. En même temps le point CENTER (C) est envoyé au PU adjacent à l'ouest. Le voisinage complet est disponible immédiatement après la lecture du point EAST (indiqué par les flèches sur la Fig. 6.12).

L'implantation matérielle est montrée sur la Fig. 6.13. La gestion de la communication des unités de traitement fait partie du module CONTROLE LOCAL.

Il est important de noter que le choix retenu pour la division d'image affecte seulement la procédure de communication entre les unités de traitement adjacentes et n'a pas d'influence sur leur architecture interne. Ce protocole de communication s'appliquerait également aux applications de filtrage. Dans le cas d'un partage différent de l'image, il suffirait de modifier ce bloc sans changer la structure d'autres modules fonctionnels. La seule contrainte est d'assurer que toutes les valeurs sont disponibles tous les deux cycles d'horloge.

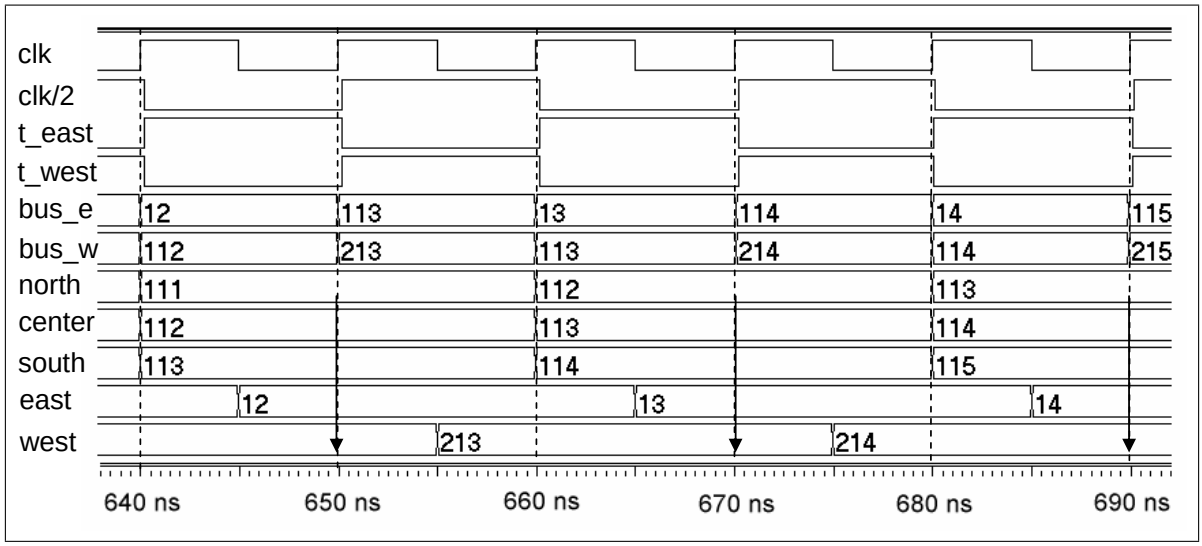


FIG. 6.12 – Diagramme de synchronisation de la lecture du voisinage d'un point. Les valeurs affichées sont des valeurs numériques lues de la mémoire des données.

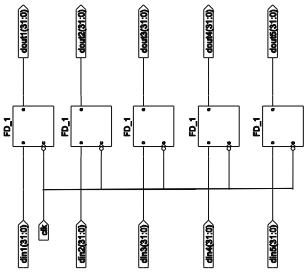
6.3.2 Initialisation par interpolation

L'interpolation est la première étape des algorithmes de type Narrow Band. Son rôle est de détecter la position initiale des contours et d'initialiser en même temps les premiers points actifs. Dans le chapitre 3, section 3.2.1 (page 68), nous avons indiqué comment est représentée la position du contour initial dans l'image d'entrée. La position du contours est matérialisée par l'ensemble de niveau zéro d'une fonction $\varphi = \varphi(x, y)$. φ a une valeur négative à l'intérieur et une valeur positive à l'extérieur (ou l'inverse, selon la direction de l'évolution souhaitée). La position initiale du contour est définie par l'utilisateur.

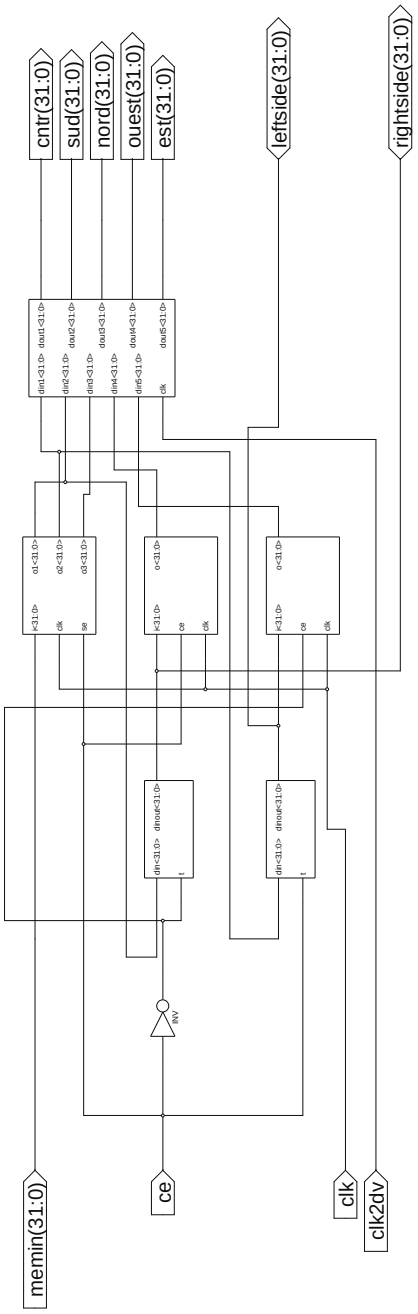
Nous avons montré que le nombre de cas possibles de configuration des voisins avec des signes différents est dénombrable. Sous l'hypothèse que $\text{abs}(\varphi) = const$, le nombre exact de configurations possibles est 5 (Fig. 3.6, page 71) pour le 4-voisinage. Seuls les voisins qui ont des signes différents par rapport au point central sont considérés pour le calcul. Les termes d_x et d_y sont obtenue par l'interpolation linéaire (section 3.2.1.1, page 70). Ils expriment la distance d'un point à l'ensemble de niveau zéro (Fig. 6.14). Si les voisins ayant des signes différents de φ se trouvent simultanément dans les directions x et y, la distance à l'ensemble de niveau zéro est évaluée selon l'expression suivante :

$$u = \frac{d_x d_y}{\sqrt{d_x^2 + d_y^2}} \tag{6.3}$$

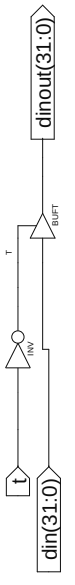
L'hypothèse $\text{abs}(\varphi) = const$ nous permet d'implanter ces calculs de manière efficace car les différences pour l'axe x et y sont numériquement équivalentes : $d_x = d_y = const$ (dans le



(a) Registres internes contenant les cinq valeurs du voisinage complet.



(b) Module d'extraction de voisinage. Les blocs intérieurs : Fig. 6.13(a) et Fig. 6.13(c)



(c) Ecriture sur le bus.

FIG. 6.13 – Extraction de voisinage. (Bus bus_e et bus_w sont ici appelés rightside et leftside)

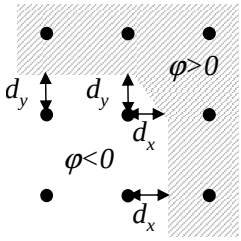


FIG. 6.14 – Détection de position du contour initial.

cas où les signes des points sont différents. Autrement les différences d_x et d_y sont nulles).

Les points ne peuvent obtenir qu'une des trois valeurs suivantes en fonction de la configuration du voisinage :

1. $u_0 = c_0$
   c_0 est une valeur choisie par défaut. Elle est attribuée dans le cas où il n'y a pas de signe différents sur le voisinage : $d_x = d_y = 0$.
2. $u_0 = c_1$
   c_1 est une constante, elle est la solution numérique de l'interpolation dans le cas où nous avons des signes différents dans une seule direction x ou y.
3. $u_0 = c_2$
   c_2 résulte de l'interpolation si les signes sont différents dans les deux directions.

La constante c_0 initialise des points à l'infini, elle est affectée à des points qui ne sont pas interpolés. Les constantes c_1 et c_2 sont connues en avance.

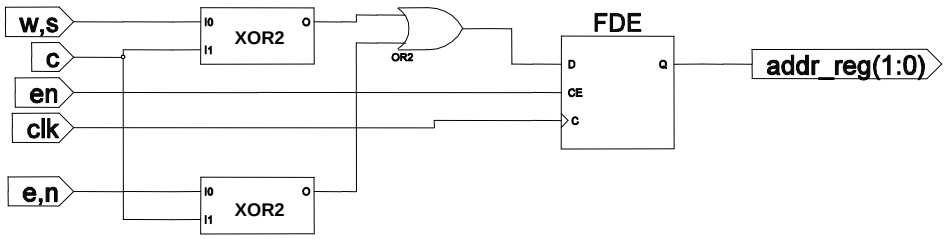


FIG. 6.15 – Interpolation.

De la discussion précédente il résulte qu'il n'est pas nécessaire d'effectuer de calculs numériques. Il suffit d'examiner les signes des valeurs des voisins et de les comparer avec le signe du point en cours de traitement. Appelons les signaux d'entrée représentant les signes des points c (CENTER), s (SOUTH), n (NORTH), e (EAST) et w (WEST). Dans l'implantation électronique (Fig. 6.15), nous avons rangé les signes dans un vecteur nommé signs = {c,n,s,e,w}. La Tab. 6.3 montre des configurations des signes c,s,n,e,w et le résultat correspondant de l'interpolation. La même table d'attribution des constantes d'interpolation est appliquée pour des valeurs opposées du vecteur sign.

TAB. 6.3 – Exemple des configurations des signes des points de voisinage

c	n	s	e	w	Résultat d'interpolation
0	0	0	0	0	c_0
0	0	0	0	1	c_1
0	0	0	1	0	c_1
0	0	0	1	1	c_1
0	0	1	0	0	c_1
0	1	0	0	0	c_1
0	1	1	0	0	c_1
0	1	0	1	0	c_2
0	1	0	0	1	c_2
0	1	0	1	1	c_2
0	1	1	1	0	c_2
0	1	1	0	1	c_2
0	1	1	1	1	c_2

Nous pouvons ainsi décrire le comportement du module d'interpolation par les conditions indiquées dans la Tab. 6.4. Les relations d'ordre s'appliquent à la représentation numérique du vecteur binaire.

TAB. 6.4 – Adresses des constantes d'approximation.

Registre	Valeur numérique	Adresse
c_0	∞	addr_reg = 0b00
c_1	0.5	addr_reg = 0b10 ou addr_reg = 0b01
c_2	0.35	addr_reg = 0b11

La valeur du vecteur signs est comparée aux différentes combinaisons des configurations du voisinage. A la sortie, nous obtenons l'adresse d'une des constantes d'interpolation.

6.3.3 Approximation du schéma numérique

Afin d'éviter les calculs complexes sur les racines et les puissances carrées propres aux schémas numériques classiques, nous proposons d'utiliser une approximation au lieu de calculer la valeur exacte de l'Eq. (4.4). En se basant sur l'Eq. (4.2), l'approximation permet de conserver la précision sous-pixélique nécessaire en réduisant le coût d'implantation.

Deux types d'approximation ont été testés :

- Linéarisation par morceaux
- Look up table (LUT)

Le raisonnement est le suivant. L'Eq. (4.4) peut s'écrire sous forme :

$$u(p) = u_{\min}(p) + f_{diff}(|u_x(p) - u_y(p)|) \tag{6.4}$$

où $u_x(p) = \min\{u([x_p \pm 1, y_p])\}$ et $u_y(p) = \min\{u([x_p, y_p \pm 1])\}$. Cette équation peut être réécrite comme suit :

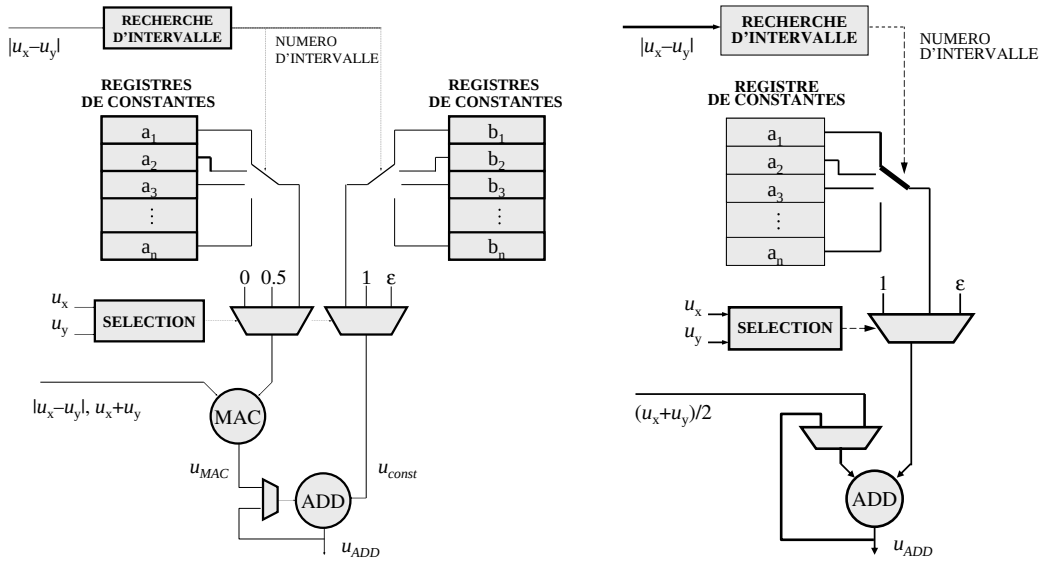
$$u(p) = \frac{u_x(p) + u_y(p)}{2} + g_{diff}(|u_x(p) - u_y(p)|) \tag{6.5}$$

En approximant l'Eq. (6.5) par des segments linéaires ou par une LUT, on obtient :

$$\hat{u}_{Lin.\ Approx}(p) = \frac{u_x(p) + u_y(p)}{2} + a_i|u_x - u_y| + b_i \tag{6.6}$$

$$\hat{u}_{LUT}(p) = \frac{u_x(p) + u_y(p)}{2} + a_i \tag{6.7}$$

Le nombre d'opérations nécessaire pour obtenir $\hat{u}_{Lin.\ Approx}(p)$ se réduit à trois additions, une soustraction et deux multiplications. Dans le cas de la LUT, $\hat{u}_{LUT}(p)$ demande seulement deux additions et une multiplication. En revanche, cette réduction des opérations pour la LUT est payée par le besoin d'utiliser plus d'intervalles afin d'atteindre une précision comparable à celle de la linéarisation par morceaux.



(a) Linéarisation par morceaux (b) Look up table

FIG. 6.16 – Architecture interne du module d'approximation.

Les calculs arithmétiques sont complètement pipelinés et cadencés à la fréquence clk. Rappelons, que nous avons besoins de deux cycles de clk pour extraire tous les points de voisinage. Par conséquence, deux opérations arithmétiques sont effectuées lors de la lecture d'un voisinage ce qui permet de réduire la longueur des pipelines où les données sont en attente. Après une latence initiale de 10 périodes d'horloge, le résultat pour chaque point est obtenu dans deux périodes d'horloge (un cycle d'extraction du voisinage). Les calculs sont réalisés en préci-

sion virgule fixe. Une étude comparative du coût d'implantation est présentée dans la section des résultats expérimentaux, cf. page 140.

La description fonctionnelle d'implantation de l'approximation est donnée par Fig. 6.16. Les signaux d'entrée sont $|u_x - u_y|$ et $u_x + u_y$. Les termes u_x et u_y sont obtenus par deux comparateurs.

Le calcul de nouvelle valeur commence par la recherche des constantes d'approximation. La valeur $|u_x - u_y|$ entre dans le bloc RECHERCHE D'INTERVALLE. Ensuite, elle est comparée en parallèle aux limites des intervalles et les résultats sont à la sortie convertis (Fig. 6.17) en adresse (le numéro d'intervalle) du registre contenant la constante d'approximation a_i et b_i. La Fig. 6.18 montre le schéma de son implantation matérielle.

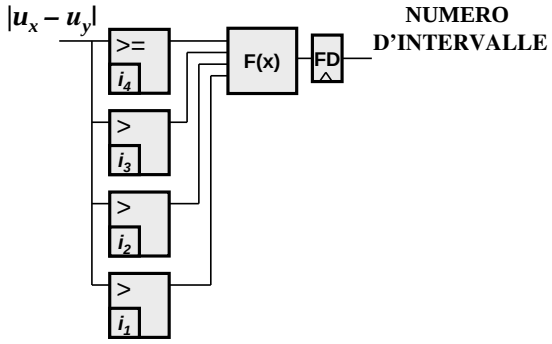


FIG. 6.17 – Recherche d'intervalle : schéma fonctionnel.

L'unité SELECTION est chargée de contrôler les multiplexeurs qui sont utilisés pour produire au fur et à mesure les différents termes de calcul de distance selon les équations (6.6) et (6.7). L'unité SELECTION teste des valeurs u_x, u_y :

1. $u_x < \infty$ et $u_y < \infty \rightarrow$ l'approximation est calculée selon l'Eq. 6.6.
2. $u_x = \infty$ ou $u_y = \infty \rightarrow$ on additionne 1 à la valeur finie car la propagation ne va que dans une seule direction.

le cas où $u_x = \infty$ et $u_y = \infty$ ne peut arriver par définition de l'algorithme Massive Marching.

6.3.3.1 Linéarisation par morceaux

Considérons d'abord la linéarisation par morceaux. Le multiplicateur accumulateur (MAC) calcule deux termes : u_{M1} et u_{M2} dont l'addition fournit à la sortie du MAC. La Tab. 6.5 montre la forme des termes calculés en fonction du test des valeurs u_x et u_y. A la fin, l'addition du résultat u_{MAC} et de la constante b_i est effectuée.

Sur la Fig. 6.19 se trouve la synchronisation des calculs avec l'horloge clk. Le résultat pour point est disponible en quatre cycles d'horloge. Le multiplicateur accumulateur est occupé

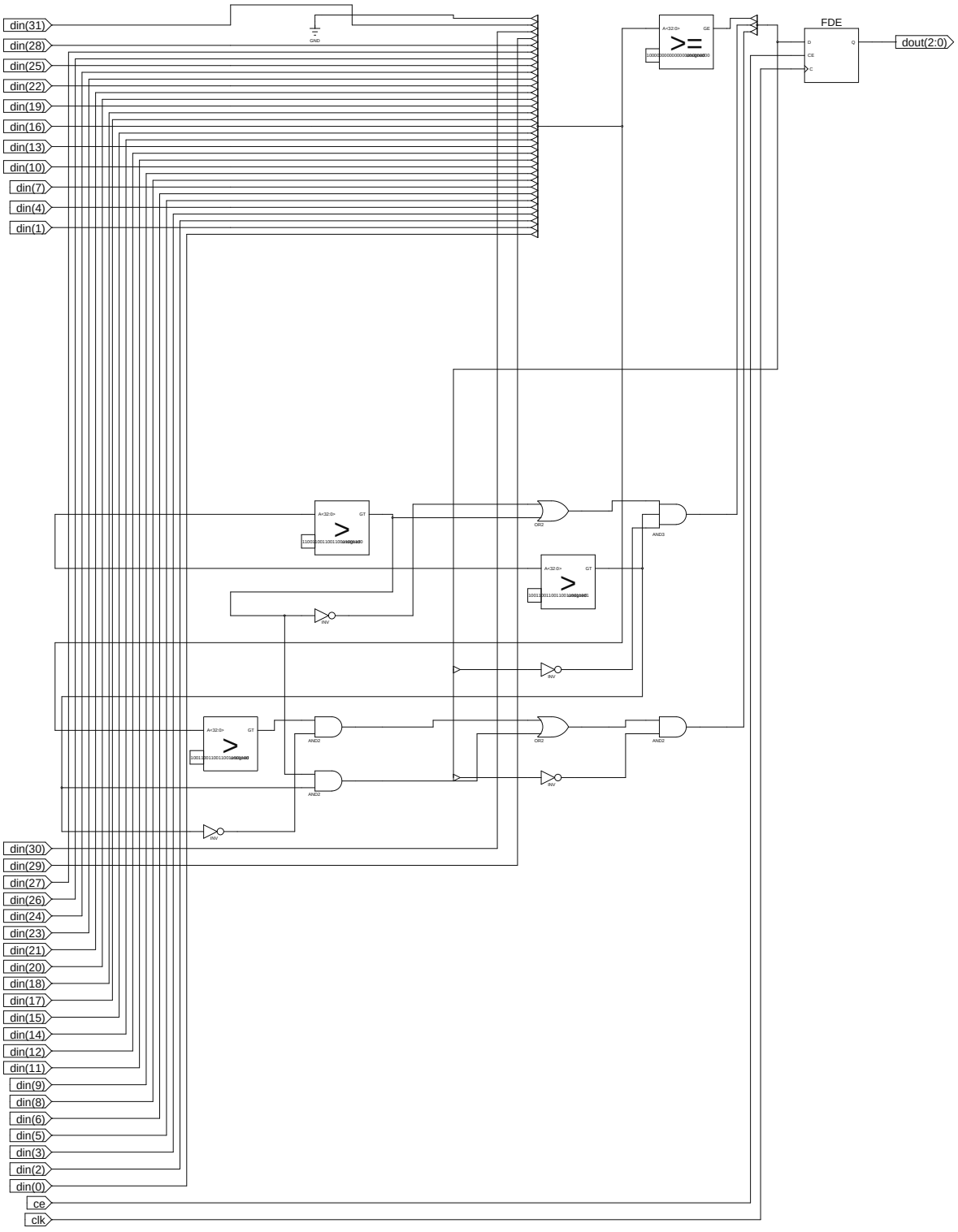


FIG. 6.18 – Recherche d'intervalle : schéma électronique.

TAB. 6.5 – Suite des calculs pour la linéarisation par morceaux. (*) Si une des valeurs est infinie, l'autre est mise à sa place. La même approche est appliquée aux bords de l'image ou la valeur du point manquant est considérée comme infinie.

Sortie	$u_x < \infty$ et $u_y < \infty$	$u_x = \infty$ ou $u_y = \infty$
	$u_{M1} = 0.5(u_x + u_y)$	$u_{M1} = 0.5(u_x + u_x)$ (*)
	$u_{M2} = a_i \lvert u_x - u_y \rvert$	$u_{M2} = 0 \lvert u_x - u_y \rvert$
u_{MAC}	$u_{MAC} = u_{M1} + u_{M2}$	$u_{MAC} = u_{M1} + u_{M2}$
u_{ADD}	$u_{ADD} = u_{MAC} + b_i$	$u_{ADD} = u_{MAC} + 1$

toutes les périodes de clk alors que l'addition ne s'exécute que toutes les trois périodes. Nous bénéficions des périodes libres de l'additionneur que nous utilisons pour obtenir la valeur du critère d'activation qui s'écrit : $u(p_i) + K_{min}$ (l'Eq. 4.7, page 91) où $u(p_i) = u_{ADD}(p_i)$. Le module Activation est détaillé dans la section suivante.

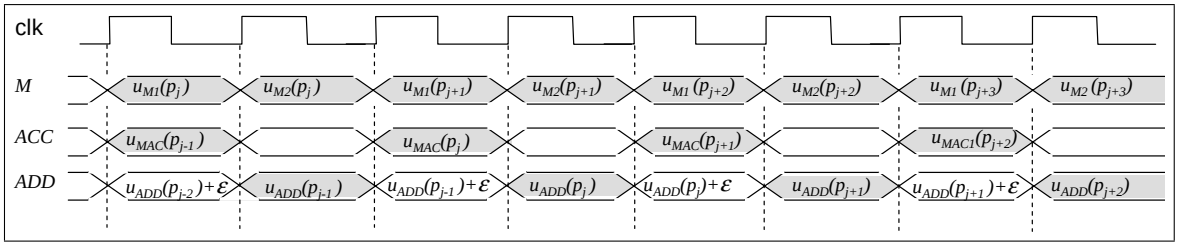


FIG. 6.19 – Timing de calcul de la fonction distance. Nous avons séparé les résultats intermédiaires du multiplicateur (M) et de l'accumulateur (ACC) . L'indice j indique le numéro du point traité.

6.3.3.2 Look up Table

Dans le cas de la LUT, le calcul se réduit à une seule addition (Tab. 6.6). Le contrôle du multiplexeur (Fig. 6.16) est simplifié car si seulement une des valeurs u_x, u_y est finie alors le calcul de distance est réduit à la simple addition d'une constante ($const = 1$) afin d'ajouter la vitesse de propagation dans une direction. Rappelons que les deux valeurs ne peuvent pas être infinies en même temps puisqu'un tel point ne serait pas activé.

TAB. 6.6 – Suite des calculs pour la LUT. Si une des valeurs est infinie, le calcul se réduit à l'addition de 1. La même approche est appliquée aux bords de l'image.

Sortie	$u_x < \infty$ et $u_y < \infty$	$u_x = \infty$ ou $u_y = \infty$
u_{ADD}	$u_{ADD} = \frac{\lvert u_x + u_y \rvert}{2} + a_i$	$u_{ADD} = \frac{\lvert u_x + u_y \rvert}{2} + 1$

L'obtention du résultat prend une période de clk alors que pour l'extraction des cinq point de voisinage, il nous faut deux périodes. Cela signifie que nous disposons d'une période libre d'additionneur qui est utilisée pour le calcul du critère d'activation $u_{ADD} + K_{min}$, comme le

montre la Fig. 6.20.

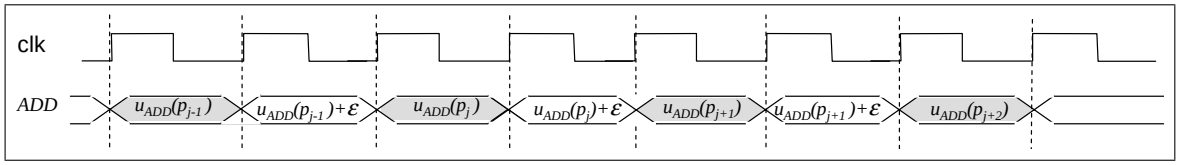


FIG. 6.20 – Timing de calcul de la fonction distance par approximation à LUT. L'indice j indique le numéro du point traité.

Remarque :

Le bloc d'approximation comme il est ici présenté, peut calculer la fonction distance euclidienne, c'est à dire pour $\mathcal{F} = 1$. Si $\mathcal{F} \neq 1$, des multiplications supplémentaires seraient nécessaires. Sans regarder l'approximation utilisée, il faut multiplier le critère d'activation, $K_{min}\mathcal{F}$, et les constantes d'approximation a_i et éventuellement b_i.

Dans le cas de l'approximation par linéarisation, les multiplications peuvent être réalisées à l'aide du multiplicateur accumulateur déjà existant. Cela aurait pour conséquence de rallonger le temps de calcul et la bande passante générale de l'architecture serait plus basse à moins que deux multiplicateurs supplémentaires ou approximer les multiplications soit utilisés. Le bloc d'approximation par la LUT serait obligatoirement équipé d'au moins d'un multiplicateur.

6.3.4 Ecriture des résultats

La mémoire des données est réalisée comme une mémoire double port. C'est à dire qu'elle a deux bus d'adresses et deux bus de données (Fig. 6.21). En théorie, les deux ports peuvent être utilisés pour la lecture et l'écriture sous condition que les accès simultanés à la même adresse soient évités.

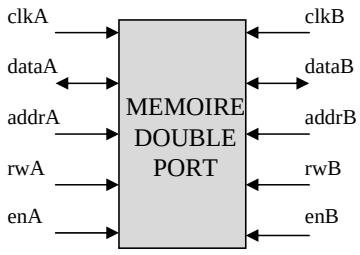


FIG. 6.21 – Mémoire double port

Nous utilisons un port (port A) uniquement pour lecture et le deuxième (porte B) uniquement pour l'écriture des résultats. Les résultats sont écrits dès qu'ils sont disponibles à la sortie du module d'approximation.

La question qui se pose est comment obtenir l'adresse de mémoire pour l'écriture ? Les calculs étant entièrement pipelinés, l'unité de traitement lit des points de la mémoire un par un alors que les résultats pour chaque point sont disponibles avec certain retard. Il existe trois façons de générer l'adresse d'écriture : soit nous pouvons ajouter un deuxième générateur d'adresses, soit l'adresse d'écriture est calculée à partir de l'adresse de lecture ou l'adresse de lecture est propagée le long des calculs afin d'être utilisée pour l'écriture.

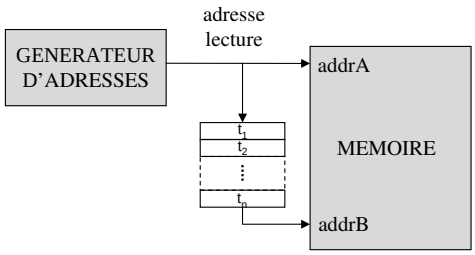


FIG. 6.22 – Propagation de l'adresse d'écriture. n est le nombre de cycles d'horloge utilisé pour le calcul.

Nous avons choisi la troisième façon, c'est à dire la propagation de l'adresse (Fig. 6.22). Le principe de propagation peut être vu comme un registre à décalage de longueur n où n est le nombre de cycles d'horloge écoulés depuis la lecture d'un point jusqu'à l'obtention du résultat pour ce point. La propagation est utilisable dans le cas de l'approximation par linéarisation ou par la LUT et ne demande que la modification de la durée de propagation. En ce qui concerne deux autres possibilités, le calcul de l'adresse signifierait de réaliser des soustractions de constantes ou d'ajouter un deuxième compteur d'adresse qui démarrerait avec retard après le générateur d'adresses de lecture.

6.3.5 Activation

Les unités de traitement procèdent au traitement ligne par ligne de manière synchrone. Si un point n'est pas actif, la nouvelle valeur n'est pas calculée et l'unité de traitement ne se charge que de la communication avec des autres unités et éventuellement de la réception des demandes d'activation.

L'unité de traitement reconnaît l'activité du point en cours de traitement selon l'état du drapeau approprié (indiquant s'il est actif ou non). Cette information peut être codée en un seul bit pour chaque point. Dans notre réalisation, la valeur "1" indique l'état "actif", la valeur "0" indique l'état "non actif".

Selon la définition du Massive Marching, le drapeau d'activité du point x reçoit la valeur "actif" quand la condition de l'Eq. (4.7), page 91, est vérifiée. La condition est vérifiée par le bloc Activation (Fig. 6.23).

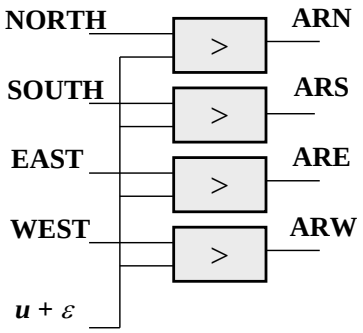


FIG. 6.23 – Activation : schéma fonctionnel.

L'activation est réalisée par quatre comparateurs qui, en fonction des résultats, procède à l'activation par envoi des demandes d'activation aux voisins (matérialisées par des signaux ARN (Activation Request North), ARS (Activation Request South), ARW (Activation Request West et ARE (Activation Request East)). Les signaux ARW et ARE sont envoyés vers les unités de traitement voisines.

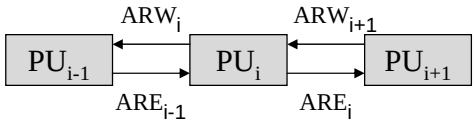


FIG. 6.24 – Demandes d'activation

Si un point a été déjà activé, il le reste jusqu'à il soit traité et ensuite désactivé. Le problème de l'implantation matérielle de l'activation est que nous pouvons rencontrer des cas où il faut activer en même temps les points centre, sud et nord. Les mises à jour des trois drapeaux ne peuvent pas être faites en même temps. Nous avons divisé l'activation de voisinage en deux étapes. D'abord, le point nord est activé par l'écriture de la mise à "1" de la valeur de son drapeaux si ARN = 1. Pour l'activation du point central, les signaux ARW et ARE en provenance des unités de traitement voisines sont décisifs. Nous les appelons ARW_{i+1} et ARE_{i-1} (Fig. 6.24). Nous considérons en plus, la valeur de la demande ARS qui a été générée lors du traitement du point précédent (p_{j-1}) car le point en cours de traitement (p_j) est son voisin sud (Fig. 6.25).

Le point centre est activé en cours de traitement actuel si l'expression $\mathrm{ARS}_{j-1} \cup \mathrm{ARE}_{i-1} \cup \mathrm{ARW}_{j+1} = 1$. La valeur ARS_{j-1} est mémorisée et introduite dans la procédure d'activation du point p_j.

Rappelons que l'activation est effectuée uniquement lors de l'étape Gauss-Seidel du Massive Marching. Aussitôt que tous les drapeaux sont inactifs, l'algorithme se termine.

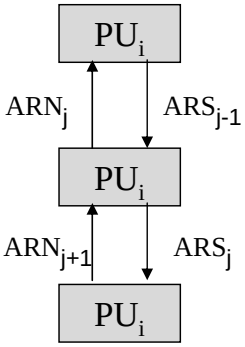


FIG. 6.25 – Demandes d'activation

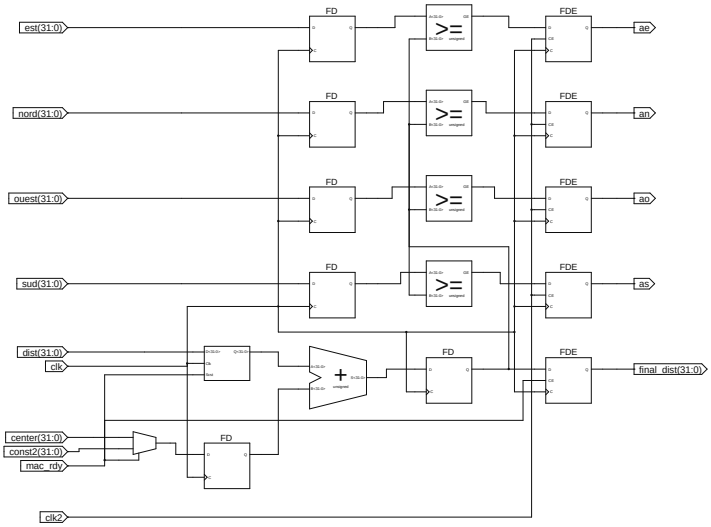


FIG. 6.26 – Activation : implantation matérielle. Remarque : signaux ARS, ARN, ARW, ARE sont ici appelés as, an, aw, ae.

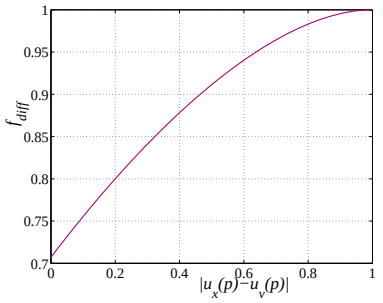
6.4 Résultats expérimentaux

Nous avons testé la précision des résultats obtenus par rapport à deux facteurs :

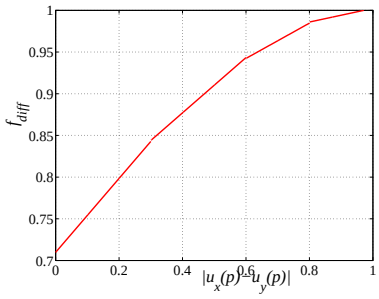
- Type d'approximation du schéma numérique de Godunov.
- Nombre de bits utilisés pour effectuer les calculs en virgule fixe.

Nous avons mesuré l'erreur $\mathcal{L}_\infty$ des résultats par rapport à la solution exacte en précision double. Rappelons que la norme $\mathcal{L}_\infty$ correspond au maximum des éléments d'un vecteur. Ici, il représente la limite supérieure de l'erreur.

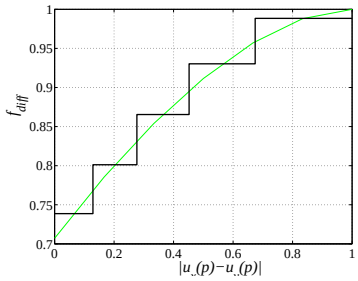
Les résultats peuvent être évalués visuellement dans la Figure 6.28. L'image de test contient trois sources de formes différentes : une lettre M, une ligne droite et un point. Nous montrons des courbes de niveaux de la distance obtenue pour une précision de 8 bits après la virgule. L'erreur d'approximation (dans ℓ_∞) par rapport au résultat exact est donnée dans la Table 6.7.



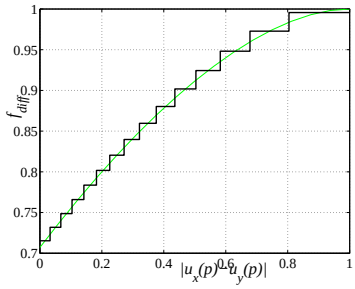
(a) Schéma de Godunov exact



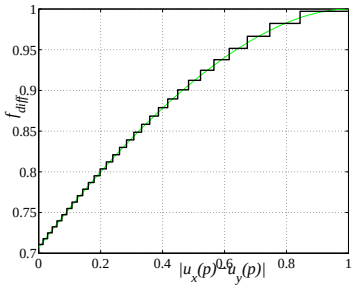
(b) Linéarisation par morceaux avec 4 intervalles



(c) LUT avec 5 intervalles non équidistants

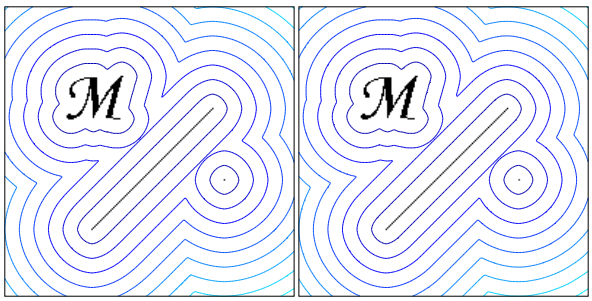


(d) LUT avec 15 intervalles non équidistants



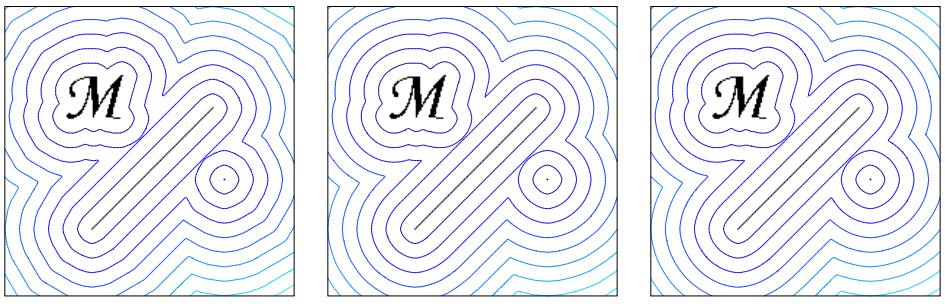
(e) LUT avec 30 intervalles non équidistants

FIG. 6.27 – Approximations de f_{diff}



(a) Schéma de Godunov exacte

(b) Linéarisation par morceaux avec 4 intervalles

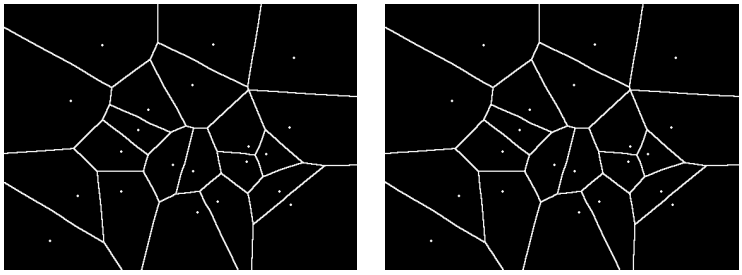


(c) LUT avec 5 intervalles

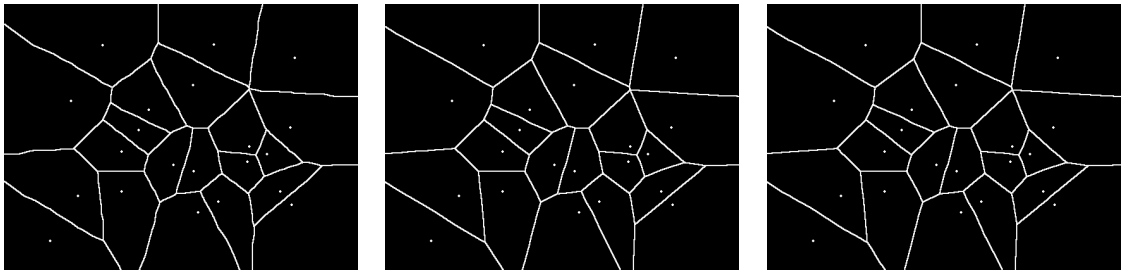
(d) LUT avec 15 intervalles

(e) LUT avec 30 intervalles

FIG. 6.28 – Les courbes de niveaux de la fonction distance obtenues pour différentes approximations et précision de 8+8 bits.



(a) Schéma de Godunov exact (b) Linéarisation par morceaux 4 intervalles



(c) LUT avec 5 intervalles (d) LUT avec 15 intervalles (e) LUT avec 30 intervalles

FIG. 6.29 – Diagramme de Voronoï calculée sur la fonction distance obtenue pour différentes approximations et 8+8 bits de précision.

TAB. 6.7 – L'erreur d'approximation du schéma de Godunov donné sur la Fig. 6.28

Type d'approximation (nombre d'intervalles)	Linear. 4	LUT 5	LUT 15	LUT 30
Erreur $\mathcal{L}_\infty$ (%)	0.3	3.9	1.2	0.9

Deux types d'approximation de la fonction g_{diff} ont été utilisés : *linéarisation par morceaux* avec quatre intervalles et LUT avec cinq, quinze et trente intervalles (Fig. 6.27). Le premier et le dernier élément dans le LUT sont exactes (égale à $1/\sqrt{2}$ et 1) afin de minimiser l'erreur dans les directions de gauche et de droite, du haut et du bas ainsi que les directions diagonales. Les autres valeurs sont obtenues par distribution de l'erreur également de manière égale sur l'intervalle]0, 1[.

L'implantation du bloc d'approximation (cf. Fig. 6.16) a été testé dans la précision de virgule fixe avec à 8 bits pour la partie de nombre entier et 4, 6, 8, 10, 12, 16, 20 et à 24 bits pour la partie fractionnelle. La partie entière pourrait être augmentée en cas de nécessité. L'erreur totale (l'approximation plus l'arrondi) donnée par la Table 6.8.

En plus de l'erreur introduite par l'approximation et l'erreur d'arrondi (Fig. 6.30), nous avons observé le coût d'implantation en termes d'occupation de surface estimée en nombre de portes NAND équivalentes dans la netlist. Ce nombre est indiqué par le compilateur. Il s'agit de l'estimation après l'optimisation et le routage pour la précision 8+8 et 8+16 (bits pour la

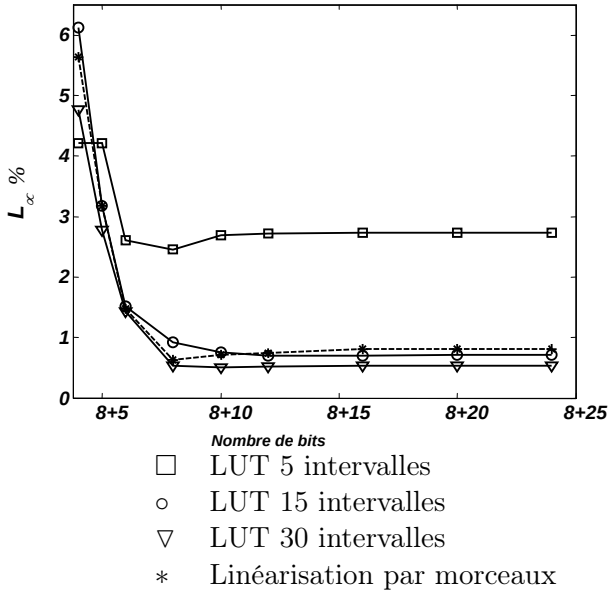


FIG. 6.30 – Erreur $\mathcal{L}_\infty$ mesurée en fonction de nombre de bits de précision pour différentes approximations et comparée à la solution exacte.

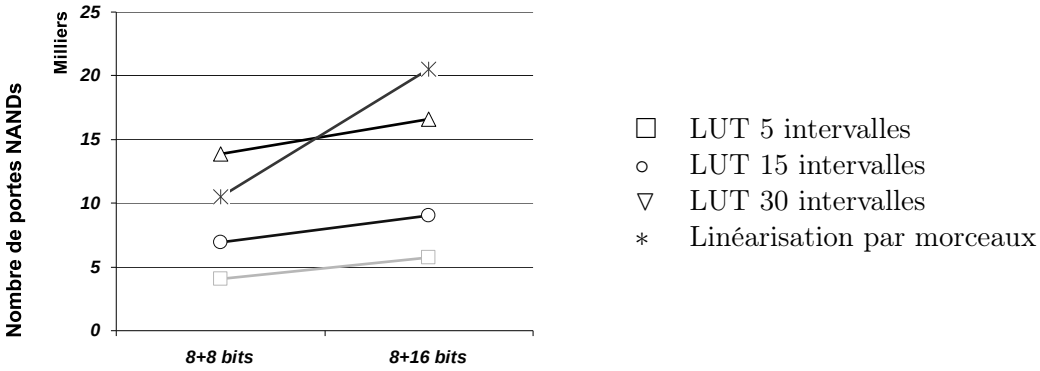


FIG. 6.31 – Estimation de surface occupée par bloc Approximation. (en nombre de portes NAND équivalentes)

TAB. 6.8 – Les erreurs (approximation plus arrondi) ℓ_∞ des résultats de la Fig. 6.28(b, c, d et e) sont comparés avec le résultat exact de la Fig. 6.28(a).

Type d'approximation (nombre d'intervalles)	Linéaire. 4	LUT 5	LUT 15	LUT 30
Erreur $\mathcal{L}_\infty$ 8 bits (%)	0.64	2.45	0.92	0.54
Erreur $\mathcal{L}_\infty$ 16 bits (%)	0.81	2.74	0.70	0.53

partie entière plus bits pour la partie fractionnelle.

TAB. 6.9 – Estimation de la surface occupée par l'implantation du bloc Approximation avec 8+8 bits de précision

Type d'approximation (nombre d'intervalles)	Linéarisation 4	LUT 5	LUT 15	LUT 30
Surface après optimis. (NANDs)	10132	4031	6920	13856
Bits de mémoire	128	80	240	480

TAB. 6.10 – Estimation de la surface occupée par l'implantation du bloc Approximation avec 8 + 16 bits de précision

Type d'approximation (nombre d'intervalles)	Linéarisation 4	LUT 5	LUT 15	LUT 30
Surface après optimis. (NANDs)	20044	5743	9056	16592
Bits de mémoire	192	120	360	720

Notons qu'en utilisant la linéarisation par morceaux l'occupation de surface augmente de deux fois en NANDs équivalents dans le cas où la précision augmente de 8+8 à 8+16 bits (le nombre entier plus la partie fractionnelle). La taille plus élevée de la surface occupée par le module d'approximation est due à l'usage d'un multiplicateur accumulateur optimisé pour la vitesse. D'autre part l'occupation de surface croit de manière linéaire dans le cas de l'utilisation d'une LUT : l'augmentation est de 20 % à 40 % de NANDs équivalents et par d'un tiers de bits de mémoire.

6.5 Diffusion non linéaire

Nous allons montrer comment réaliser la diffusion non linéaire sur l'architecture présentée. Rappelons que l'équation discrétisée d'évolution s'écrit :

$$u^{n+1} = u^n + \tau\left(g(u_x^-)u_x^- + g(u_x^+)u_x^+ + g(u_y^-)u_y^- + g(u_y^+)u_y^+\right) \tag{6.8}$$

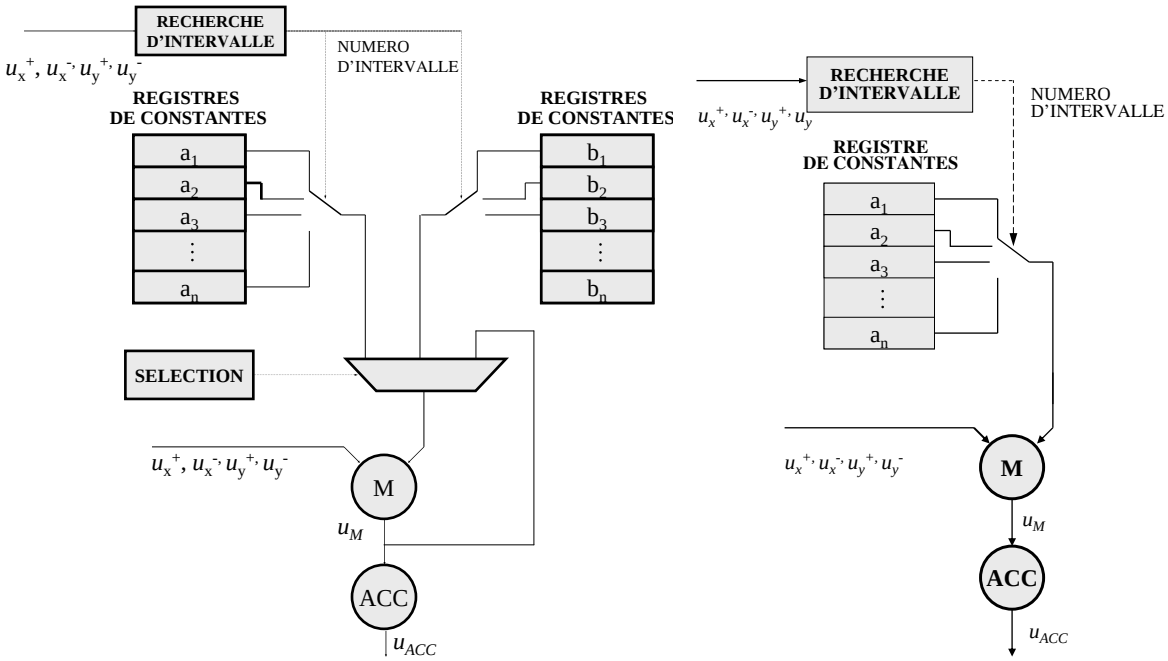
la fonction $g()$ est non linéaire (chapitre 3, page 61), elle peut être approximée par linéarisation par morceaux ou par LUT :

$$\hat{u}_{Lin.\ Approx}(p) = a_i u + b_i \tag{6.9}$$

$$\hat{u}_{LUT}(p) = a_i \tag{6.10}$$

Dans ces cas, le bloc d'approximation serait modifié de la façon montrée sur la Fig. 6.32.

Pour les deux types d'approximation, les calculs commencent par détecter l'intervalle d'approximation approprié dans le bloc Recherche d'intervalle. Ensuite, les valeurs des constantes d'approximation sont récupérées dans les registres de constantes concernés.



(a) Linéarisation par morceaux (b) LUT

FIG. 6.32 – Exemple de reconfiguration du bloc d'approximation pour diffusion non linéaire.

Dans le cas de la linéarisation par morceaux, un multiplexeur gère la suite des calculs. D'abord, l'accumulateur multiplicateur charge valeur b_i à la quelle, ensuite, il additionne le résultat de $a_i u$. Le tout est multiplié par u. La suite des opérations élémentaires est montrée sur la Tab. 6.11. Le multiplicateur fournit les résultats des multiplications et l'additionneur a cette fois le rôle d'accumulateur. Au fur et à mesure, on calcule la somme des quatre termes $g(u_x^{\pm})u_x^{\pm},\ g(u_y^{\pm})u_y^{\pm}$. L'addition de u^n (Eq. 6.8) se fait par l'initialisation de l'accumulateur avec cette valeur. La multiplication par le pas d'intégration est évitée par l'utilisation de la distributivité de multiplication et τ est intégré dans les valeurs des constantes d'approximation.

TAB. 6.11 – Suite des calculs d'un terme $g(u)u$ par la linéarisation par morceaux.

Sortie	Terme
u_M	$u_{M1} = a_i u$
u_M	$u_{M2} = u_{M1} u$
u_M	$u_{M3} = b_i u$
u_{ACC}	$u_{ACC1} = u_{ACC} + u_{M1}$
u_{ACC}	$u_{ACC2} = u_{ACC1} + u_{M2}$
u_{ACC}	$u_{ACC3} = u_{ACC2} + u_{M3}$

TAB. 6.12 – Suite des calculs pour la LUT.

Sortie	Terme
u_M	$u_M = a_i u$
u_{ACC}	$u_{ACC1} = u_{ACC} + u_M$

Pour l'approximation à l'aide de la LUT, nous somme obligés d'ajouter un multiplicateur afin de pouvoir calculer $g(u)u$. Le résultat est la somme des termes de l'équation (6.8) qui est produite par l'accumulateur. La suite des opération est présentée par la Tab. 6.12.

La diffusion non linéaire appartient au type d'algorithme à image entière. Cela signifie que tous les points sont considérés comme actifs et le bloc d'Activation n'est pas utilisé. En conséquence, les calculs sont effectués pour tous les points en fonction du nombre d'itérations. Le programme de la diffusion non linéaire est montré sur la Fig. 6.33. La valeur $N_{\text{itérations}}$ représente le nombre d'itérations à effectuer, il est choisi par l'utilisateur.

```
k = 0
Tant que k < N_itérations
{
      NonlinearDiffusion
      k = k+1
}
```

FIG. 6.33 – Diffusion non linéaire en instructions de l'architecture proposée.

Les autres blocs de l'unité de traitement fonctionnent de la même façon quelle est décrite pour le calcul de la fonction distance.

6.6 Conclusions

Le résultat principal de cette partie est l'implantation hardware de l'algorithme Massive Marching sur une architecture SIMD :

- Nous avons montré qu'il est possible d'utiliser l'architecture SIMD, qui avait déjà été utilisée pour le filtrage, également pour les applications de segmentation, c'est à dire les algorithmes à évolution de courbe (type NB).
- Nous avons montré la faisabilité pour le calcul de la fonction distance avec une précision sous-pixélique avec une approximation par linéarisation par morceaux ou par LUT.

L'architecture proposée est composée d'un vecteur linéaire des unités de traitement. Chaque unité de traitement dispose d'un bloc de mémoire privée. Dans notre cas, un bloc de mémoire est une colonne de l'image et nous avons besoin d'une unité de traitement par colonne. Cependant, l'unité de traitement a été conçue avec une attention portée sur la scalabilité de

l'architecture. L'unité de traitement a été proposée comme la plus simple possible afin de permettre de la dupliquer au maximum sur une seule composante. La seule condition de son fonctionnement est d'avoir toutes les valeurs à l'entrée de l'unité de traitement à un instant donné. Si l'image est découpée différemment, les seuls blocs fonctionnels affectés par le changement seraient l'extraction du voisinage et la gestion d'activation des points.

Nous avons étudié la surface occupée par l'unité de traitement en fonction de la précision de calcul et du choix d'approximation. La linéarisation par morceaux demande une unité de calcul plus complexe pour une meilleure précision. En revanche la LUT est beaucoup plus simple et occupe moins de surface mais à cause des discontinuités dans l'approximation, les résultats sont moins précis. Nous avons présenté et visualisé des résultats expérimentaux et nous avons comparé l'influence de la représentation des nombres en virgule fixe sur les résultats obtenus.

Nous avons également mesuré les performances de l'architecture présentée avec des contraintes physiques de réalisation sur un FPGA. Pour une image 256×256, la fréquence maximale estimée par les outils de synthèse a été de 150 MHz pour les calculs arithmétiques et 75 MHz pour l'accès à la mémoire. Avec cette vitesse de traitement, le temps d'exécution estimé pour calculer la fonction distance sur l'image entière est de 4 ms dans le pire des cas. Il s'agit de propagation en diagonale à partir d'une seule source placée dans un coin de l'image.

Nous tenons à préciser, que le but de ce chapitre était de démontrer la **faisabilité** de l'utilisation d'une architecture SIMD, c'est à dire d'une architecture à granularité fine, pour des algorithmes à propagation de front. L'utilisation de ce type d'architecture est loin d'être optimale. L'activité des unités de traitement n'est pas équilibrée et la bande passante de 16,3 Mpix/s (256×256 pixs/4 ms) pour une segmentation est obtenue au moyen d'un coût en surface très élevé.

Rappelons que l'utilisation de l'architecture SIMD pour les algorithmes à évolution continue de front est possible grâce à l'introduction de Massive Marching.

Dans la partie suivante de ce document est étudiée une autre architecture, qui est mieux adaptée à ce type d'applications et qui permettra d'obtenir le rapport bande passante/coût plus favorable.

Chapitre 7

Architecture multiprocesseur

Dans le chapitre précédent, nous avons montré comment adapter une architecture SIMD, conçue pour le prétraitement, à la segmentation. Bien que l'approche SIMD permette d'avoir une meilleure exploitation en parallèle des données, sa capacité reste cependant limitée en ce qui concerne la programmabilité de l'unité de traitement. Il est évident que l'architecture proposée SIMD peut être mal adaptée pour les applications avec accès aléatoires à la mémoire.

Quelques expériences temps-réel ont été publiées avec des super-calculateurs (architecture MIMD). Quant aux applications embarquées, elles sont à notre connaissance quasi-inexistantes.

Parmi d'autres réalisations des EDPs sur une architecture MIMD, nous pouvons mentionner l'implantation des EDPs sur une architecture NUMA (Holmgren and Wallin, 2001). La description de NUMA est présentée dans (Noordergraaf and van der Pas, 1999). En bref, il s'agit d'une machine avec plusieurs processeurs (jusqu'à 112 processeurs de famille UltraSparc, organisés en groupes, chacun connecté à un nœud) et mémoire partagée. Ainsi, il est possible d'avoir en même temps des processus parallèles partagés ou individuels. Bien que cette architecture ne soit pas destinée aux systèmes enfouis, type de parallélisme utilisé par Holmgren and Wallin (2001) est intéressant car il propose de convertir le calcul des EDP par FFT multidimensionnelle à l'aide de l'approche fréquentielle (Loan, 1992). Les filtres à diffusion peuvent ainsi être reformulés à l'aide de convolution directionnelle et par conséquent réalisés par FFT multidimensionnelle. Un autre exemple est mentionné dans (Sethian, 1996).

Un des avantages principaux des méthodes aux EDPs est que pour changer l'application il suffit de ne changer que le générateur de la vitesse de propagation. L'objectif de ce chapitre est de proposer une architecture multiprocesseur pour des systèmes enfouis qui corresponde aux besoins des applications des EDPs tout en profitant des derniers progrès technologiques. Notre approche veut offrir une architecture suffisamment générale et facilement programmable avec les outils de développement classiques.

D'abord, nous reprenons la discussion des algorithmes et nous avançons encore plus dans leur généralisation afin d'effectuer une analyse du flux de données en vue de trouver sa meilleure optimisation. Lors de la conception de l'architecture proposée, nous avons consacré un soin particulier à l'activité équilibrée de toutes les unités de traitement de l'architecture.

Finalement nous proposons une architecture multiprocesseur construite autour de plusieurs noyaux de processeurs opérant de manière asynchrone. Nous présentons également une estimation des performances de l'architecture proposée.

7.1 Vue générale des algorithmes

D'abord, reprenons la classification des algorithmes des courbes de niveaux. Afin d'analyser les données, nous présentons ici leur vue générale :

1. **Evolution de surface**
   Elle contient les filtres à diffusion, le lissage géométrique, la suppression de bruit, les opérateurs morphologiques érosion/dilatation et les opérateurs composés. L'évolution temporelle est basée sur le voisinage local et engendre l'évolution spatiale des courbes de niveaux (Sapiro, 2000). Elle s'arrête aussitôt que la convergence ou un nombre donné d'itérations est atteint. L'équation d'évolution s'écrit :

   $$\frac{\partial u}{\partial t} = \mathcal{F}(u)|\nabla u| \tag{7.1}$$

   u est l'image et représente une fonction qui évolue dans une dimension plus élevée, $\mathcal{F}$ la vitesse d'évolution. Il s'agit du type d'algorithme image entière : dans chaque itération, tous les points de l'image sont traités.

2. **Propagation d'onde.**
   Elle contient les algorithmes représentés par la fonction distance euclidienne ou pondérée, la LPE continue, le diagramme de Voronoï ou "Shape-from-Shading. La solution est propagée à partir des sources données, sur l'image entière selon la vitesse $\mathcal{F}$ définie ou seulement sur un voisinage choisi des sources de propagation. L'évolution est contrôlée par l'équation Eikonale :

   $$|\nabla u| = \mathcal{F} \tag{7.2}$$

   La condition initiale est représentée par les sources de propagation. L'algorithme opère localement, seulement sur une bande étroite autour du front d'onde (type Narrow Band). Typiquement, le front est propagé de manière équidistante aux sources en utilisant la structure de données ordonnées (Kimmel, 1995).

3. **Modèles déformables.**
   Nous distinguons les types suivant : avec régulariseurs (contrôlés par l'information statistique des régions) (Paragios, 2000), sans régulariseurs (l'information des régions n'est pas utilisée) (Sapiro, 2000) ou combiné avec un modèle (Bresson, Vandergheynst and Thiran, 2003). Les algorithmes procèdent par déformation d'un contour initial en fonction des propriétés géométriques du contour (la courbure, la pente) ou des caractéristiques statistiques de l'image (par exemple la moyenne estimée de l'intensité de la région ou l'information sur la texture).

   L'équation d'évolution généralisée s'écrit sous forme :

$$\frac{\partial u}{\partial t} = \mathcal{F}_{courbure}(u) + \mathcal{F}_{grad}(u) + \mathcal{F}_{région}(u) \tag{7.3}$$

   u est une surface définie dans un espace de dimension $N+1$ et sa projection dans l'espace de dimension N représente la courbe. Les modèles déformables appartiennent au type Narrow Band car la déformation est gérée par une force calculée pour chaque point à chaque itération seulement sur un voisinage défini du contour.

Dans les applications EDPs, l'évolution de l'image peut être considérée comme l'évolution d'une surface en continu. Cette évolution se traduit en itérations d'un filtre local (Sapiro, 2000). Cette notion de localité des opérations est très importante du point de vue de l'implantation, car cela signifie que nous n'avons besoin que de l'information sur le voisinage local comme l'exprime l'équation suivante :

$$u^{n+1}(p) = f(u^n(q_i), u^n p) \text{ où } q_i \in V_4(p) \tag{7.4}$$

Par conséquent, le potentiel de parallélisme des calculs est important. Sous l'hypothèse de disposer des moyens appropriés, tous les algorithmes à schéma explicite,(Fig. 3.2, Fig. 3.7, page 67) ainsi que l'algorithme Massive Marching (Fig. 4.2, page 93), peuvent être réécrits sous l'unique forme présentée sur la Fig. 7.1. Chaque commande sur une ligne de l'algorithme est exécutée **en parallèle pour tous** $p \in \mathcal{A}$. Notons que l'exécution de la suite des lignes est séquentielle.

L'algorithme généralisé sur la Fig. 7.1 représente la conclusion principale sur l'analyse des algorithmes du point de vue de l'implantation matérielle. Il met en évidence le potentiel de parallélisme des algorithmes présentés jusqu'à maintenant. Désormais, l'information sur le type d'algorithme, Narrow Band ou image entière, n'intervient que dans la définition de l'ensemble $\mathcal{A}$. L'ensemble $\mathcal{A}$ contient les points à traiter dans une itération. Rappelons la définition de l'ensemble $\mathcal{A}$ qui change en fonction du type d'algorithme comme suit :

```
// Initialisation :
Pour tous p ∈ I faire en parallèle :
{
  Initialiser constantes ;   //par exemple τ
  Initialiser A ;   //Initialiser points actifs
}
// Evolution :
Pour tous p ∈ A faire en parallèle :
{
  Extraire voisinage u^n(V_4(p)) ;
  Calculer u^{n+1}(p) ;
  Mettre à jour u(p) ;
  Activer nouveaux points ; // (insertion dans A)
}
```

FIG. 7.1 – Algorithme généralisé.

1. **Image entière** : les algorithmes de type image entière agissent dans chaque itération sur tous les points. Par conséquent, chaque point dans l'image est considéré comme actif.

$$\boxed{\mathcal{A} = \text{supp}(I), I = image.} \tag{7.5}$$

2. **Narrow Band** : les algorithmes de type Narrow Band agissent seulement sur les points actifs, c'est à dire sur les points proches de la position actuelle du front qui se déplace.

$$\boxed{\mathcal{A} = \{p \,||\, \text{dist}(p)| < NB_{\text{width}}/2\}} \tag{7.6}$$

où NB_{width} est la largeur du Narrow Band.

Un autre aspect à considérer pour l'analyse du flux de données est le type de voisinage utilisé. La définition du voisinage dépend du schéma numérique pour la discrétisation (voir la section 1.1 et le chapitre et 3). Nous considérons le 4- voisinage et le schéma explicite d'Euler pour l'intégration non seulement parce qu'il est simple à implanter mais également parce qu'il a un potentiel de parallélisme comme cela a été expliqué dans le chapitre 3 (à partir de la page 61).

7.1.1 Analyse du flux de données

Les blocs fonctionnels, entre lesquels les données transitent, sont les suivants :

1. **Unités de traitement ($\mathbf{PU}_i$) :** Elles calculent des valeurs des points selon le schéma numérique choisi avec la précision exigée. Nous supposons que plusieurs PUs travaillent en parallèle.

2. **Mémoire de données :** Elle contient des images et d'autre type de données comme les étiquettes des régions ou les drapeaux.
3. **Mémoire de points actifs :** Cette mémoire contient des informations sur les points actifs représentant l'ensemble $\mathcal{A}$ pour les algorithmes de type Narrow Band.

De manière générale, la mémoire de points actifs peut contenir différents types d'information comme les valeurs des points, leurs coordonnées ou des drapeaux. Le type d'information utilisé dépend de l'application et sera spécifié plus tard. Pour le type image entière la mémoire de points actifs n'est pas indispensable car tous les points sont considérés comme actifs.

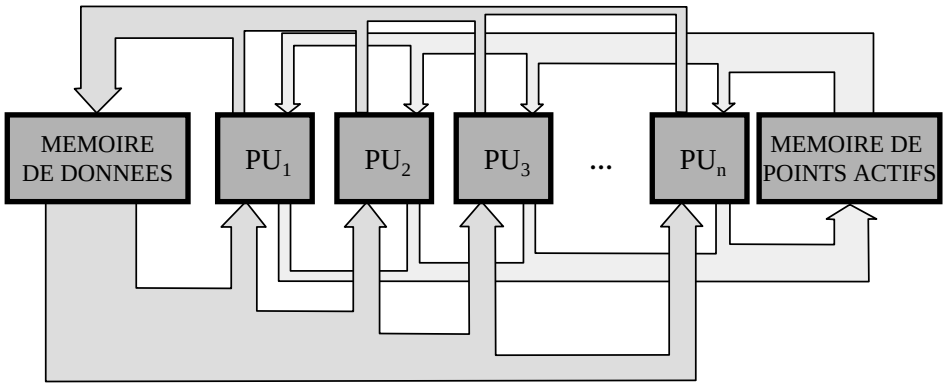


FIG. 7.2 – Flux de données. La largeur des branches correspond au volume des données qui sont transférées par ces branches.

Le flux de données correspondant à l'algorithme généralisé de la Fig. 7.1 est montré sur la Fig. 7.2. Pour établir une stratégie d'optimisation, regardons le flux de données du point de vue des volumes qui entrent et sortent des blocs fonctionnels.

Le volume des données sortant du bloc de **MEMOIRE DE DONNEES** est plusieurs fois plus élevé que le volume entrant de données. Cela est causé par le fait que pour traiter un point p, une PU donnée doit extraire son voisinage complet (le point en cours de traitement compris). En revanche une seule valeur de la mémoire est mise à jour. Ainsi, pour le 4-voisinage, le volume sortant est cinq fois plus élevé (un point p et ses quatre voisins) que le volume entrant (une mise à jour du point p).

De même, chaque **PU** lit la valeur du point dans la mémoire de données ainsi que les valeurs des voisins. En revanche, elle écrit une seule valeur dans la mémoire de données ou de une à quatre valeurs dans la mémoire des points actifs.

Le nombre des accès simultanés attendus à la mémoire est prédominant pour la complexité de l'architecture (Noguet, 2002). Il s'agit surtout des accès aléatoires aux mémoires dans le cas des algorithmes du type Narrow Band.

Sans perdre en généralité, nous supposons qu'à un moment donné, une PU ne peut lire qu'un point de la **MEMOIRE DE POINTS ACTIFS** (elle peut traiter un seul point) mais

elle peut engendrer l'activation d'aucun à quatre des voisins. Par conséquent, le volume moyen entrant de données dans la mémoire de points actifs est légèrement plus élevé que le volume sortant.

	Nombre de points testés	0 p	1 p	2 p	3 p	4 p
Points	1256650	96393	1070842	83809	5606	0
%	100	7.67	85.21	6.67	0.45	0.00

TAB. 7.1 – Statistique des points insérés dans la mémoire de points actifs à partir d'un point. Les résultats représentent les valeurs moyennes qui ont été obtenus pour le diagramme de Voronoï et la LPE par algorithme Massive Marching.

Nous avons effectué des mesures statistiques pour estimer combien de voisins sont activés par un point en cours de traitement pour Massive Marching. Selon les résultats obtenus, un point traité insère dans la mémoire le plus souvent un nouveau point actif (85% des cas) (Tab. 7.1).

7.1.2 Analyse temporelle

Pour concevoir une gestion optimale des accès à la mémoire, il faut tenir compte également des aspects de synchronisation d'exécution des algorithmes.

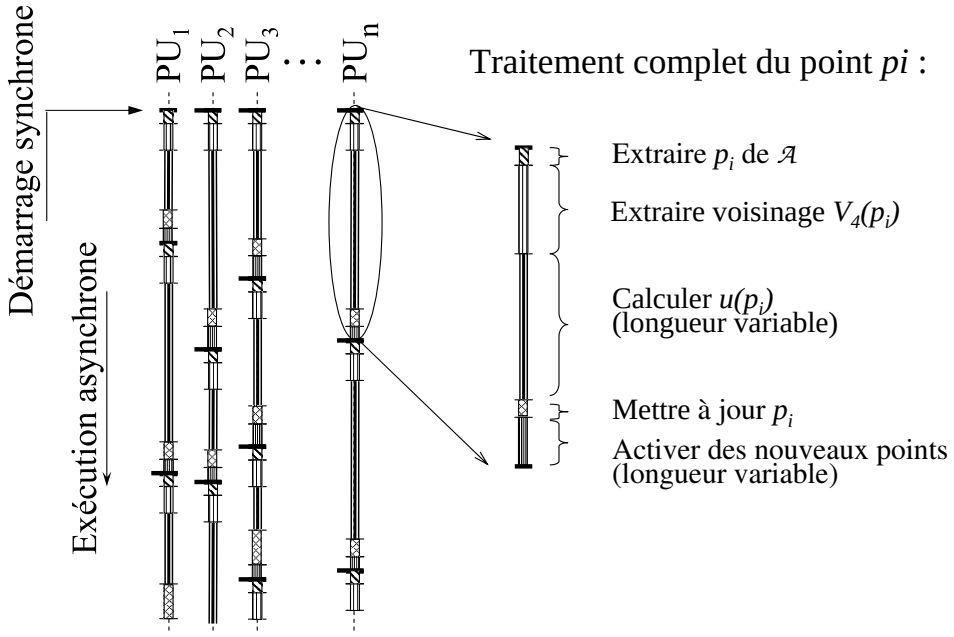


FIG. 7.3 – Etude de synchronisation d'exécution des algorithmes EDPs.

La Fig. 7.3 montre une étude temporelle de l'exécution des algorithmes. Nous supposons que N points sont traités en parallèle par N processeurs. Tous les processeurs démarrent en

même temps et exécutent le même algorithme. Pour l'instant, nous négligeons le problème des éventuels accès simultanés à la mémoire. Notons que pour les algorithme à type Narrow Band, nous avons séparé l'extraction de l'ensemble $\mathcal{A}$ du point à traiter et l'extraction du voisinage.

Nous avons pu constater que la structure de l'exécution de code a deux caractéristiques majeures :

- L'extraction du point et des valeurs des voisins est significativement plus rapide que les calculs. Son temps d'exécution est constant.
- Le calcul des valeurs u des différents points peut avoir des longueurs différentes à cause de l'exécution conditionnelle de certaines parties du code.

Par conséquent, après quelque temps, l'exécution de l'algorithme par les processeurs n'est plus synchrone. Cela signifie que l'exécution efficace sur plusieurs PUs devrait être asynchrone afin d'éviter la perte de temps par les pauses des PUs plus rapides.

L'exécution asynchrone pourrait être avantageuse parce qu'elle favorise naturellement des accès à la mémoire de manière aléatoire dans le temps tout en réduisant le nombre de manipulations simultanées de mémoire.

Nous savons que pour les algorithmes de type image entière la relation entre les valeurs de deux itérations successives est :

$$u^{n+1} = f(u^n) \tag{7.7}$$

Les points de synchronisation sont la fin des itérations car nous avons besoin de connaître des valeurs u^n avant de commencer l'itération $n + 1$.

Dans le cas de Massive Marching, l'algorithme procède en deux étapes :

$$u^{n+1}_{\text{Jacobi}} = f(u^n) \tag{7.8}$$

$$u^{n+1} = f(u^n, u^{n+1}_{\text{Jacobi}}) \tag{7.9}$$

d'où l'on voit que les points de synchronisation se situent à la fin de chacune de deux étapes (étape de Jacobi et étape de Gauss-Seidel).

L'exécution asynchrone est possible parce que le traitement des points est indépendant. En revanche, elle doit respecter les points de synchronisation de chaque algorithme. Nous appelons les points de synchronisation des instants où tous points à traiter doivent être mis à jour avant de commencer le traitement suivant.

7.1.2.1 Points de synchronisation

Une première synchronisation est effectuée au début d'exécution. Ensuite, nous avons deux points majeurs de synchronisation : en début d'une itération et avant l'activation des nouveaux points pour l'itération suivante. La stratégie générale de synchronisation est visualisée sur la Fig. 7.4.

```
   // Initialisation :
   Pour tous p ∈ I faire en parallèle :
   {
⇒ Synchronisation
      Initialiser constantes ;   //par exemple τ
      Initialiser A ;   //Initialiser points actifs
   }
   // Evolution :
   Pour tous p ∈ A faire en parallèle :
   {
⇒ Synchronisation
      Extraire voisinage u^n(V_4(p)) ;
      Calculer u^{n+1}(p) ;
      Mettre à jour u(p) ;
⇒ Synchronisation
      Activer nouveaux points ; // (insertion dans A)
   }
```

FIG. 7.4 – Algorithme généralisé des EDPs.

De l'Eq. (7.7)-(7.9) il résulte, que l'itération $n+1$ ne peut commencer qu'après la mise à jour de tous les points traités dans l'itération n. Si certains processeurs terminent plus tôt que les autres elles arrêtent leur exécution et attendent jusqu'à la mise à jour de tous les points actuellement actifs, par exemple les unités de traitement PU_1, PU_2 et PU_3, dans l'itération n sur la Fig. 7.4. En début de chaque itération, les unités de traitement commencent à exécuter l'algorithme en même temps (Fig. 7.5). L'exécution se poursuit tant que tous les points actifs dans cette itération ne sont pas mis à jour.

Tous les points de l'image étant considérés comme actifs pour le type d'algorithme à image entière, la synchronisation ne s'effectue qu'au début de chaque itération.

7.2 Architecture globale

Nous avons adopté une approche semi parallèle, où la mémoire des données et la mémoire des points actifs sont partagées, afin d'obtenir une activité équilibrée de tous les processeurs.

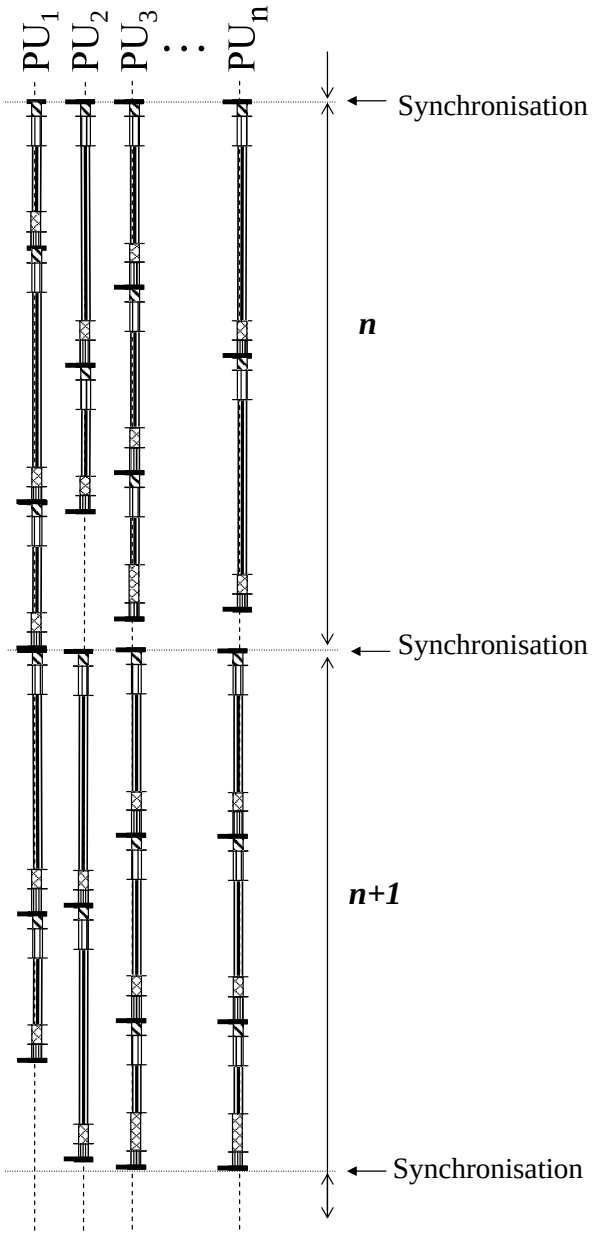


FIG. 7.5 – Principe des points de synchronisation

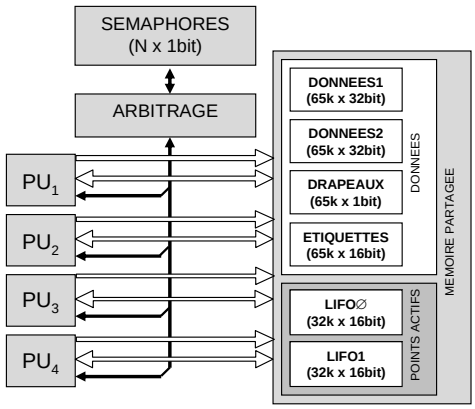


FIG. 7.6 – Architecture globale.

Autrement dit, qu'il n'y ait pas de processeur inactif que l'accès à la mémoire soit aléatoire (type Narrow Band) ou séquentiel (type image entière).

L'architecture proposée est de type MIMD et elle comporte quatre processeurs qui opèrent en parallèle (Fig. 7.6). Les processeurs travaillent en mode SPMD (*Single Program Multiple Data*) où ils exécutent le même programme de manière asynchrone. De manière générale, la partie essentielle du processeur est une unité arithmétique (ALU) performante permettant d'implanter les calculs de $\mathcal{F}$, qui est généralement une fonction non linéaire. En plus, le processeur contient des modules pour la lecture et le décodage des instructions et pour la gestion des accès à la mémoire.

Les interconnexions avec la mémoire partagée sont organisées de façon que tous les processeurs puissent accéder à toutes les mémoires. La mémoire des données contient les images d'entrée et de sortie. En fonction de l'application, elle peut également contenir d'autres informations. Dans le cas de la LPE, nous ajoutons les images des étiquettes et des drapeaux. La mémoire des points actifs est divisée en deux parties (Fig. 7.6). Les points en cours de traitement se trouvent dans une partie et les points activés pour l'itération suivante dans l'autre.

Les accès simultanés à la mémoire sont contrôlés à l'aide des sémaphores. Les sémaphores permettent de verrouiller et rendre disponible un bloc mémoire seulement pour un processeur et le bloquer pour les autres. La mémoire n'est déverrouillée pour les autres processeurs que lorsque le processeur qui y accède termine ses opérations. La gestion de la priorité des processeurs dans l'attribution de la mémoire est nécessaire un mécanisme d'arbitrage. Le fonctionnement des sémaphores sera expliqué dans une des sections suivantes.

7.2.1 Mémoire des points actifs

Nous avons réalisé la mémoire des points actifs à l'aide de deux piles LIFO (*Last-In-First-Out*) (cf. Fig. 7.7) qui se justifient par le fait que l'ordre de traitement des points est indifférent. L'avantage des mémoires LIFO est l'absence du délai de transport contrairement aux files FIFO (*First-In-First-Out*). Pour les opérations d'insertion et d'extraction voir Figs. 7.8(a) et 7.8(b).

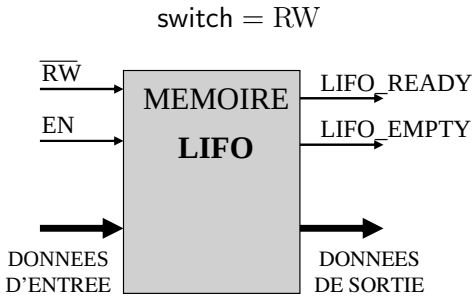
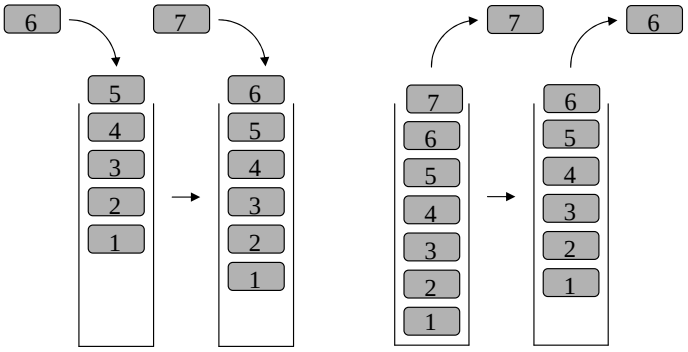


FIG. 7.7 – Représentation d'une mémoire LIFO



(a) Insertion (b) Extraction

FIG. 7.8 – Fonctionnement d'une mémoire LIFO

Nous avons dit qu'une mémoire LIFO est lue, la deuxième est remplie. Le contrôle de la direction de lecture/écriture est effectué par un signal `switch` (Fig. 7.7) qui commute à la fin de chaque itération (Fig. 7.12). Nous pouvons déterminer quelle LIFO est utilisée pour la lecture et quelle pour l'écriture en fonction du signal switch par :

- Extraction : LIFO(switch)
- Insertion : LIFO($\overline{\text{switch}}$)

Dans notre implantation, les LIFOs contiennent les coordonnées des points actifs (Fig. 7.9) :

$$\boxed{i - j}$$

La présence d'un point dans une des LIFOs est indiqué par la valeur de drapeaux approprié.

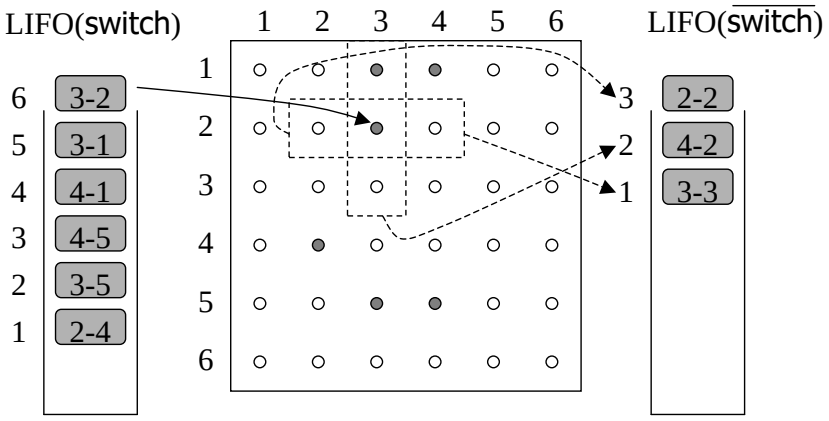


FIG. 7.9 – Gestion des LIFOs. Les points actuellement actifs sont marqués en noir.

Quand un point est extrait d'une LIFO (par exemple LIFO(switch)), on récupère ses coordonnées et on procède au traitement en fonction de son voisinage. Après avoir calculé et mis à jour sa nouvelle valeur, on active les voisins selon un critère d'activation propre à l'algorithme exécuté. Les coordonnées des nouveaux points actifs sont insérées dans la deuxième LIFO (LIFO($\overline{\text{switch}}$) dans notre exemple sur la Fig. 7.9).

Nous savons que chaque PU exécute le même programme de façon asynchrone vis à vis des autres PUs. En dépit de l'exécution asynchrone, les algorithmes EDPs ont un ou plusieurs points de synchronisation. Les point de synchronisation sont reconnus par :

- La LIFO en mode de lecture est vide (Algorithmes de type Narrow Band)
- La fin du balayage de l'image (Algorithmes de type Image entière)

Le test vérifiant si la LIFO lue est vide est valide également pour la synchronisation avant l'activation des nouveaux points.

La fin de l'algorithme pour les deux types d'algorithmes est indiquée par :

- Nombre d'itérations atteint :
  - Le nombre d'itérations fixé par l'utilisateur
  - Le nombre d'itérations nécessaire pour propager la solution sur l'image entière ou Narrow Band
- Convergence (I)
  Donné par le critère choisi. Le plus souvent, il s'agit d'une limite de différence entre deux itérations consécutives.
- Convergence (II)
  Les deux LIFOs sont vides. Le critère d'activation n'a été satisfait pour aucun point et l'algorithme s'arrête.

Les LIFOs sont également partagées par les processeurs. Nous appliquons le même principe

de partage à l'aide des sémaphores, comme pour les autres blocs de mémoire.

7.2.2 Sémaphores

Les sémaphores matérialisent le moyen de verrouillage des blocs de mémoire partagée. Le principe des sémaphores commence à être largement utilisé dans des systèmes multiprocesseur (Raineault, 2001). Même des noyaux de processeurs DSP modernes commencent à être équipés par des instructions dédiées à la synchronisation à l'aide des sémaphores (Xilinx, 2002).

Le sémaphore doit assurer les fonctions suivantes :

- Verrouiller le bloc de mémoire pour le processeur qui demande à y accéder, le garder verrouillé jusqu'à la fin des opérations du processeur et le déverrouiller ensuite.
- Effectuer l'arbitrage dans le cas où le même bloc de mémoire est sollicité par plusieurs processeurs en même temps.

Dans notre architecture, nous avons utilisé des sémaphores tels qu'ils sont réalisés dans la bibliothèque des fonctions du langage HandelC et dont le principe est le suivant. Quand un processeur, par exemple PU_2, veut accéder à un bloc de mémoire, il procède d'abord à un test de l'état du sémaphore attribué à ce bloc. Si le sémaphore est libre, la mémoire est immédiatement verrouillée pour le processeur PU_2. Si la mémoire a été verrouillée par un autre processeur, par exemple PU_1, le processeur PU_2 continue à tester le sémaphore en boucle tant qu'il ne peut pas y accéder à son tour (Fig. 7.10).

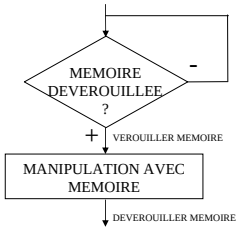


FIG. 7.10 – Principe des sémaphores vue du côté de processeur.

Si plusieurs processeurs demandent d'accéder au même bloc de mémoire, un arbitrage est effectué. Dans notre cas, l'arbitrage est basé sur la priorité. La priorité correspond à l'inverse de l'ordre des processeurs $PU_1 \succ PU_2 \succ PU_3 \succ PU_4$, c'est à dire que PU_1 emporte sur PU_2 et ainsi de suite, cf. (Celoxica, 2002). Les processeurs ayant demandé d'accéder à la mémoire et n'étant pas servis, doivent attendre jusqu'à pouvoir y accéder. Quand le processeur PU_1 déverrouille la mémoire, il ne peut la redemander qu'au prochain cycle d'horloge, ainsi un autre processeur, par exemple PU_2 peut être servi. Si PU_1 redemande d'accéder à la mémoire, il regagnera la mémoire immédiatement après la fin des opérations de PU_2. Si PU_3 et PU_4 sont également en attente, ils devront attendre. Cela pourrait provoquer un déséquilibre dans les délais d'attente des processeurs. Ce problème pourrait être résolu en utilisant un

autre mécanisme d'arbitrage basé sur des files d'attente, capables de contenir l'information sur l'ordre d'arrivée des processeurs. Une discussion des algorithmes d'arbitrage peut être trouvée dans (Shreedhar and Varghese, 1995) ou (Gjessing and Maus, 2002) et un exemple d'un algorithme de ce type présenté dans (Trynosky, 2003).

Nous avons parlé de l'exécution asynchrone qui permet aux processeurs d'exécuter, à un moment donné, différentes lignes du même code. L'exécution asynchrone réduisant ainsi le nombre d'accès simultanés aux blocs partagés, un arbitrage simple pourrait être suffisant. Il s'agit d'une mesure simple avec une implantation facile sans devoir réaliser des structures complexes. La surface estimée occupée par un sémaphore est de 300 portes NANDs. Notons que nous avons un sémaphore par bloc de mémoire, les deux LIFOs y comprises.

Avec cette façon d'utiliser les sémaphores, deux problèmes peuvent surgir :

- Charge des processeurs non équilibrée.
- Temps de traitement plus long par rapport au temps de traitement idéal pour le nombre donné de processeurs.

Nous avons étudié ces problèmes sur l'implantation de la LPE continue sur l'architecture proposée. La discussion est présentée dans la section suivante.

7.2.3 Modèle de processeur

Sous le mot processeur, nous entendons surtout une unité logique arithmétique (ALU) performante avec une précision nécessaire pour l'application. L'ALU est entourée des modules assurant la lecture des instructions, les manipulations avec la mémoire et la communication avec les sémaphores. Actuellement, un tel processeur peut être matérialisé soit avec le noyau d'un processeur existant embarqué sur le composant comme le PowerPC (Xilinx, 2003) ou le processeur ARM (ARM, 2000), soit avec un processeur partiellement modifiable (Altera, 2003) ou entièrement programmé (Gumm, 1995). L'avantage d'un noyau embarqué est qu'il est réalisé avec des circuits optimisés. Il est donc plus performant qu'un noyau programmé. En revanche, un noyau programmé peut être plus facilement adapté aux besoins de l'application.

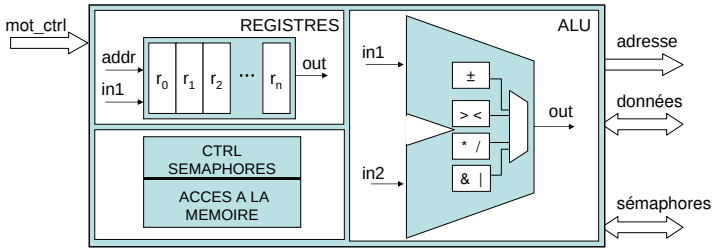


FIG. 7.11 – Diagramme bloc du modèle de processeur.

En vue de la vérification du fonctionnement des points de synchronisation et des sémaphores, nous avons adapté une approche générique et nous avons programmé un processeur qui nous sert de modèle de processeur à usage général. Dans notre modèle (Fig. 7.11), les calculs arithmétiques sont implantés en virgule flottante. Le modèle dispose de 64 registres à usage général.

Remarque : Notons que notre modèle utilise un mot de 32 bits pour représenter les nombres réels en virgule fixe (24 bits pour la partie décimale et 8 bits pour la partie fractionnelle). Pour l'adressage des blocs des mémoires, nous avons besoin d'au moins 19 bits, 16 pour l'adresse d'un point (pour une image de 256×256), plus 3 bits pour les sémaphores.

Toutes les opérations demandant d'accéder à la mémoire partagée font appel au bloc contrôle des sémaphores (CTRL SEMAPHORES sur la Fig. 7.11). Le code qui réalise la synchronisation par sémaphores est le suivant :

```
loop : demander_sémaphore_x     // Et verrouiller s'il est libre
       si x not libre jump loop // Si échec de verrouillage, réessayer
       lecture/écriture         // Manipulation avec mémoire
       libérer_sémaphore        // Déverrouiller
```

Lorsqu'un sémaphore est testé, il est immédiatement (par la même instruction) verrouillé, s'il est libre. S'il a été verrouillé par un autre processeur, le test est répété tant que le sémaphore ne peut être attribué au processeur - demandeur d'accès. Ensuite, la lecture ou l'écriture est exécutée et le sémaphore est libéré. Nous permettons que plusieurs lectures ou écritures soient exécutées par le même processeur une fois la mémoire verrouillée. La gestion des sémaphores devrait être invisible à l'utilisateur. Lors d'un accès à un bloc partagé, le compilateur doit assurer la génération du code correspondant.

L'avantage de cette approche est que le modèle fonctionnel peut être remplacé par un autre modèle de processeur ou par un noyau embarqué. Le choix de la famille du processeur va influencer le jeu d'instructions. En fonction du processeur utilisé, le code ci-dessus pourrait avoir de légères modifications. Le noyau de PowerPC, que nous avons mentionné (Xilinx, 2002), dispose des instructions de sémaphores spécialisées et le code de synchronisation prendrait la forme suivante :

```
loop : lwarx  adresse_x  registre_y  // Lire données et verrouiller si sémaphore est libre
       bne    loop                  // Si échec de verrouillage, réessayer
       stwcx  adresse_x  registre_y  // Déverrouiller
```

Nous voyons que le code est réduit car il permet de lire immédiatement la valeur. Si la lecture a été effectuée sans que le sémaphore du processeur ait été attribué au processeur en question, la valeur est considérée comme invalide et la lecture est répétée. Le sémaphore est ensuite libéré par une écriture.

Rappelons que dans notre design nous utilisons un modèle de processeur. La surface occupée par notre modèle est de $250k$ de portes NANDs équivalentes. Il s'agit d'une estimation fournie par le compilateur, sans optimisation.

7.3 Distance topographique continue

Rappelons que sur le plan de EDPs, la transformation de la LPE peut être obtenue en calculant la fonction de distance topographique à un ensemble de sources qui doit être identique à la série de minima locaux dans l'image. Si ce n'est pas le cas, il faut imposer les minima à l'aide de swamping (Najman and Schmitt, 1994), (Meyer and Maragos, 1999b).

Dans notre implantation nous avons utilisé l'algorithme parallèle Massive Marching présenté dans le chapitre Massive Marching (chapitre 4, page 89). Nous avons montré que Massive Marching peut être utilisé pour la bande étroite ou sur l'image entière sans perte d'efficacité. En même temps, il permet une parallélisation massive, semi-parallèle ou séquentielle. Rappelons que la complexité de calcul dépasse légèrement $\mathcal{O}(N)$ parce que quelques points peuvent être calculés à plusieurs reprises (Dejnožková and Dokládal, 2003).

L'implantation du Massive Marching nécessite deux images d'entrée, une image $I_{\mathcal{F}}$ contenant la vitesse de propagation $\mathcal{F}$ pour chaque point et l'autre $I_{\mathcal{E}}$ contenant des minima étiquetés. Des algorithmes à étiquetage des régions peuvent être trouvés dans (Lemonnier, 1996) ou (Bieniek, 2000).

L'étape d'initialisation détecte les contours des minima par l'interpolation linéaire. Les résultats de l'interpolation sont écrits dans l'image de sortie $I_{\mathcal{S}}$. La LIFO$\emptyset$ est initialisée avec les points interpolés. Simultanément, les drapeaux indiquant la présence des points dans la LIFO sont mis à jour dans image $I_{\mathcal{D}}$. Une fois les points actifs mis dans la LIFO$\emptyset$, nous pouvons commencer à propager la solution de la LPE sur l'image. L'algorithme est présenté sur la Fig. 7.12.

Remarque : Les points de synchronisation coïncident avec les opérations `switch` = $\overline{\texttt{switch}}$, comme indiqué sur la Fig. 7.4 et la Fig. 7.12.

7.3.1 Discussion des résultats

Nous nous intéressons particulièrement à mesurer les performance de l'implantation du Massive Marching sur l'architecture MIMD présentée précédemment. Pour chaque étape du Massive Marching, nous avons calculé le rapport relatif des temps d'exécution en fonction du

// INITIALISATION
`switch`$=\emptyset$;
Parcourir l'image
{
Extraire voisinage $V_4(p_i)$ et p_i de $I_{\mathcal{F}}$;
Calculer interpolation $u_{intp}(p_i)$;
Si $u_{intp}(p_i) < \infty$ et p_i est à l'extérieur des minima
{
Insérer coordonnées de p_i dans LIFO(`switch`) ;
Mettre à jour $I_{\mathcal{D}}(p_i)$;
Extraire voisinage $V_4(p_i)$ et p_i de $I_{\mathcal{E}}$;
Calculer étiquette de p_i ;
Mettre à jour $I_{\mathcal{E}}(p_i)$; } }
// PROPAGATION
Tant que (LIFO(switch) et LIFO($\overline{\texttt{switch}}$) non vides)
{
Tant que LIFO(`switch`) non vide **faire** *// Etape Jacobi*
{
Extraire p_i de LIFO(`switch`) ;
Extraire voisinage $V_4(p_i)$ et p_i de $I_{\mathcal{D}}$;
Calculer $u^{n+1}(p_i)$;
Mettre à jour $I_{\mathcal{D}}(p_i)$;
Insérer coordonnées de p_i dans LIFO($\overline{\texttt{switch}}$) ; }
`switch` = $\overline{\texttt{switch}}$;
Tant que LIFO(`switch`) non vide **faire** *// Etape Gauss-Seidel*
{
Extraire p_i de LIFO(`switch`) ;
Extraire voisinage $V_4(p_i)$ et p_i de $I_{\mathcal{S}}$;
Extraire étiquettes $V_4(p_i)$ et p_i de $I_{\mathcal{E}}$;
Calculer $u^{n+1}(p_i)$;
Calculer étiquette de p_i ;
Mettre à jour $I_{\mathcal{D}}(p_i)$;
Mettre à jour $I_{\mathcal{E}}(p_i)$;
Insérer coordonnées de p_i dans LIFO($\overline{\texttt{switch}}$) ; }
Remettre des drapeaux $I_{\mathcal{D}}$ à zéro ; } *// Désactivation des points*
`switch` = $\overline{\texttt{switch}}$;
Tant que LIFO(`switch`) non vide **faire** *// (Re)Activation*
{
Extraire p_i de LIFO(`switch`) ;
Extraire voisinage $V_4(p_i)$ et p_i de $I_{\mathcal{S}}$;
Calculer le critère d'activation pour $q_i \in V_4(p_i)$;
Pour tout point q_i satisfaisant le critère d'activation faire
{
Insérer coordonnées de q_i dans LIFO($\overline{\texttt{switch}}$) ;
Mettre à jour drapeaux $I_{\mathcal{D}}(q_i)$; } }
`switch` = $\overline{\texttt{switch}}$; } }

FIG. 7.12 – La LPE continue par Massive Marching.

nombre de processeurs (N) utilisés, comme indique l'expression suivante (Bieniek, 2000) :

$$\text{Rapport des temps d'exécution}(N) = \frac{t_{exec}(1)}{t_{exec}(N)} \tag{7.10}$$

Nous avons effectué des mesure séparément pour les étapes de l'interpolation et de la propagation. L'objectif était de comparer les rapports des temps d'exécution pour les différentes parties du Massive Marching. Ces deux parties ont une longueur du code différente et elles peuvent ainsi présenter un nombre différent d'accès simultanés à la mémoire. Les rapports des temps d'exécution obtenus sont présentés sur la Fig. 7.13 et sont comparés au cas idéal où le rapport des temps d'exécution est linéaire.

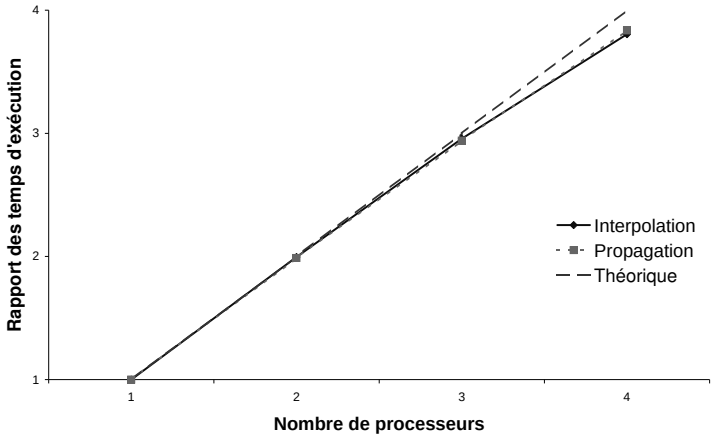
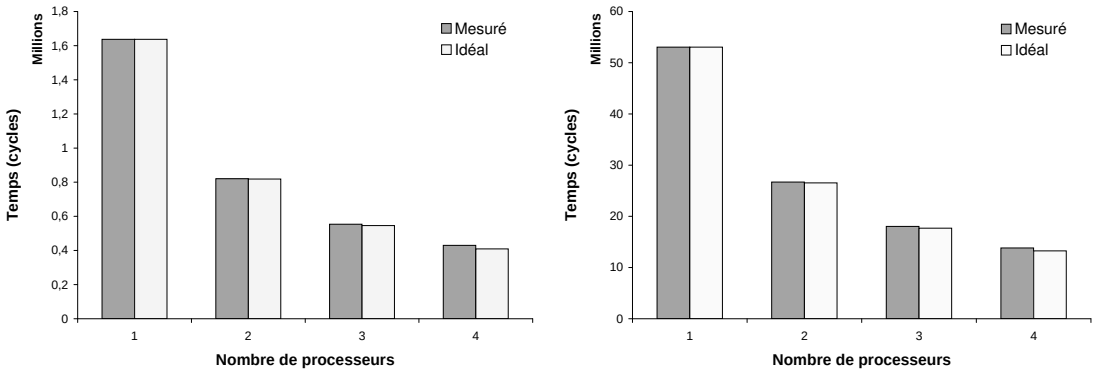


FIG. 7.13 – Rapport des temps d'exécution

Les temps d'exécution en termes de nombre de cycles d'horloge en fonction du nombre des processeurs sont montrés sur la Fig. 7.14. Ce graphique nous permet de faire une conclusion importante : bien que le contrôle simple d'accès à la mémoire utilisant des sémaphores introduit un certain délai, le nombre mesuré de cycles d'horloge est proche du temps d'exécution théorique. Le nombre mesuré de cycles d'horloge (y compris la propagation simultanée des étiquettes des régions et la phase d'initialisation) est comparable au temps d'exécution théorique.

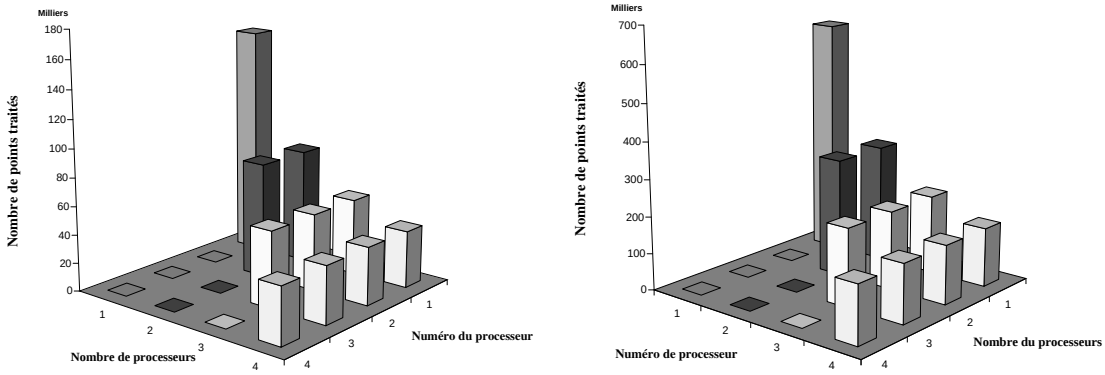
Afin de vérifier si la gestion simple des sémaphores n'a pas pour conséquence une activité déséquilibrée des processeurs, nous avons mesuré l'activité des processeurs. La Fig. 7.15 montre la distribution de l'activité des unités de traitement. L'activité est exprimée en termes de nombre de points traités par chaque PU. Nous pouvons observer que l'activité est uniformément distribuée sur tous les PUs. Cela confirme encore une fois, que le choix d'une gestion simple des accès simultanés à la mémoire est suffisant (au moins jusqu'à quatre processeurs).

La Tab. 7.2 compare la bande passante du calcul et les temps d'exécution obtenus sur différentes plate-formes hardware, pour calculer la LPE continue avec une propagation simul-



(a) Interpolation (b) Propagation

FIG. 7.14 – Temps d'exécution en cycles d'horloge



(a) Nombre de points réactivés peu élevé. (b) Nombre de points réactivés très élevé.

FIG. 7.15 – Distribution de l'activité sur les différents processeurs.

TAB. 7.2 – Bande passante des différents types d'architectures et temps de propagation par Massive Marching à partir d'une seule source placée dans un coin de l'image (de 256×256 points)

Type d'architecture	Fréquence	Bande passante (pix/s)	Temps d'exéc. (s)
4 processeurs	120MHz	26.1×10^5	0.025
PentiumIV(PC)	1.6GHz	8.27×10^5	0.078
PentiumIV(PC)	800MHz	4.96×10^5	0.130
PentiumIII(Linux)	450MHz	2.08×10^5	0.310
XScale(iPAQ)	400MHz	0.12×10^5	5.228
StrongARM(iPAQ)	206MHz	0.05×10^5	12.800

tanée des étiquettes des sources. La bande passante a été mesurée pour Massive Marching exécuté sur diverses plate-formes ; elle exprime le nombre de points traités par seconde. Pour l'architecture proposée, le programme a été écrit en assembleur. Les autres plateformes ont exécuté le programme écrit en langage **C**. Il s'agit des ordinateurs de bureau (PC) comportant des processeurs Pentium cadencés à différentes fréquences, et équipés de différents systèmes d'exploitation : Windows et Linux.

Les résultats des calculs sont montrés sur la Fig. 7.16. Le premier exemple, une image synthétique (Fig. 7.16(a)) contient les valeurs de coût et les marqueurs (carrés noirs). Les contours de la fonction distance et les zones d'influence sont visualisés sur les Figures. 7.16(b) et 7.16(c).

Les Figures 7.16(e) - 7.16(g) permettent de comparer les résultats obtenus sur une image réelle. Les zones d'influence ont été calculées d'abord par le Massive Marching sur l'architecture proposée avec 4 processeurs (Fig. 7.16(e)), ensuite par Massive Marching sur une machine séquentielle (Fig. 7.16(f)) et finalement par la propagation équidistante (*Fast Marching*) (Fig. 7.16(g)). Nous pouvons observer des légères différences de l'ordre de quelques points. Les différences sont dues, d'une part, à l'erreur liée à la propagation non équidistante et les calculs sur le voisinage incomplet. D'autre part, la représentation des nombres réels en virgule fixe peut influencer la précision numérique.

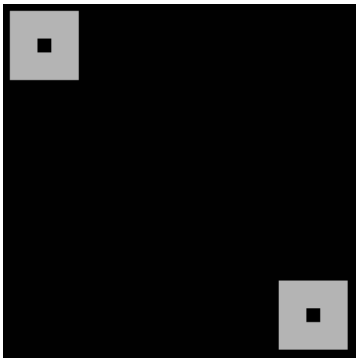
7.4 Conclusions

Dans ce chapitre, nous présentons une architecture multiprocesseur dédiée aux méthodes des ensembles de niveaux. A l'aide de l'analyse des algorithmes, nous avons montré les contraintes à respecter lors de la conception de cette architecture, notamment l'asynchronisme de l'exécution des algorithmes.

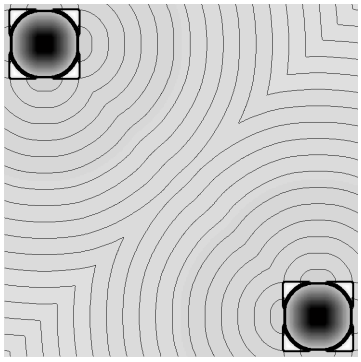
L'architecture proposée est construite autour de quatre noyaux de processeurs à usage général. Nous avons également démontré qu'un moyen simple et efficace peut être utilisé pour la gestion des accès simultanés à la mémoire partagée. Ce moyen est réalisé par des sémaphores, le contrôle est invisible à l'utilisateur et, par conséquent, facilement programmable et réutilisable.

Les mesures et les tests de performance de l'architecture proposée ont été obtenus sur l'implantation de la LPE continue. Il a été montré que le contrôle d'accès à la mémoire par sémaphores est suffisant au moins jusqu'à quatre PUs. La comparaison avec le temps d'exécution théorique et la charge équilibrée des unités de traitement montrent qu'une architecture plus compliquée serait inutile.

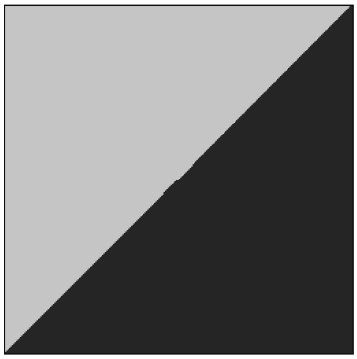
Bien que la fréquence ait été réduite pour des problèmes de consommation, les résultats



(a) Image de test avec deux marqueurs.



(b) Contours de la foncion distance pondérée.



(c) Zones d'influence.

(d) Marqueurs superposés sur l'image originale.

(e) Zones d'influence obtenues sur l'architecture proposée (4 processeurs).

(f) Zones d'influence obtenues par Massive Marching implanté de manière séquentielle.

(g) Zones d'influence obtenues par *Fast Marching*.

FIG. 7.16 – Exemples des résultats obtenus.

obtenus montrent que l'architecture proposée est plus performante que d'autres systèmes portables et qu'elle est comparable à Pentium IV.

Dans ce chapitre, nous avons étudié une architecture multiprocesseur pour un nombre de processeurs limité, allant jusqu'à quatre. Bien que le nombre de ces processeurs ne soit pas limité en lui-même, augmenter leur nombre aurait pour conséquence la saturation de la bande passante des autres éléments présents dans la chaîne de traitement, telle la mémoire des données.

7.4.1 Perspectives

Jusqu'à maintenant, l'attention était essentiellement portée à la gestion optimale des accès simultanés à la mémoire. Cette voie d'optimisation a un effet sur la minimisation du rapport des temps d'exécution en fonction du nombre de processeurs utilisés.

Un autre moyen de rendre l'architecture plus performante est d'ajouter des instructions adaptées aux applications. Etant donné que les méthodes aux EDPs demandent beaucoup d'accès à la mémoire, il est naturel de se concentrer sur l'optimisation des opérations avec la mémoire.

Rappelons que pour mettre à jour un point, on doit accéder à son voisinage. La mémoire est donc très sollicitée. L'utilisation d'une mémoire à accès parallèle au voisinage est envisageable et présente une piste à exploiter, notamment pour des images de grande taille.

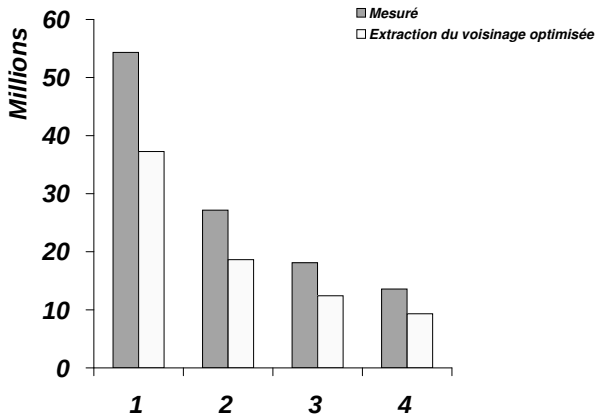


FIG. 7.17 – Estimation du gain du temps d'exécution en nombre de cycles d'horloge.

Plusieurs approches de l'extraction du voisinage en parallèle peuvent être trouvées dans la littérature. Nous pouvons citer les exemples de (Lemonnier, 1996) et de (Noguet, 2002) qui proposent différentes organisations de mémoire permettant l'extraction des voisins en un seul cycle d'horloge. Le gain théorique de l'utilisation d'une telle instruction dédiée peut être

exprimé par :

$$gain = \alpha \frac{4}{5\ N} \tag{7.11}$$

où α est le nombre total de lectures de voisinage. Le raisonnement est tel que nous gagnons 4 cycles d'horloge sur 5 pour l'extraction d'un 4-voisinage avec le point central. Cela est multiplié par le nombre α. Si l'algorithme est exécuté par plusieurs processeurs, le gain est distribué entre les processeurs (divisé par N).

Le temps de traitement estimé après cette optimisation est montré sur la Fig. 7.17. La différence entre le temps d'exécution mesuré (en gris) et le temps d'exécution estimé après l'optimisation (en blanc) représente le gain théorique. Toutes les étapes de calcul de la LPE continue, y compris l'initialisation et l'activation, sont incluses dans cette estimation.

Conclusions

Grâce aux progrès technologiques, la densité des composants électroniques ne cesse d'augmenter. Aujourd'hui nous disposons de suffisamment de ressources pour réaliser des systèmes complexes sans que la surface de composant ne représente de limite. Les vraies limitations reposent sur la consommation de l'énergie qui augmente avec la fréquence des systèmes.

La puissance et la variété des méthodes de traitement d'image fondées sur les équations aux dérivées partielles a pour conséquence que le nombre d'applications les utilisant ne cesse d'augmenter. Malgré le succès de l'application des méthodes aux EDPs à de nombreux problèmes pratiques, leurs applications industrielles restent quasi inexistantes. Une des raisons est sans doute la complexité de calcul qui exige l'utilisation des processeurs cadencés à des fréquences très élevées afin d'obtenir des temps de traitement raisonnables.

Dans ce document, nous présentons les résultats de notre recherche sur la conception d'une architecture dédiée aux méthodes basées sur les EDPs afin de les rendre accessibles à l'industrie.

Depuis plusieurs années beaucoup d'auteurs travaillaient sur l'accélération et le parallélisme algorithmique. Nous avons réalisé une étude des algorithmes aux EDPs qui couvrent les problèmes majeurs du traitement d'image : l'amélioration et la segmentation des images. Cette étude nous a servi pour mettre en commun leurs caractéristiques algorithmiques qui ont été utilisées pour la définition d'une stratégie de parallélisme.

En premier lieu, nous avons classé des algorithmes en deux groupes : type "image entière" et type "Narrow Band". Ce classement vise une généricité du point de vue de concepteur d'architecture aussi bien que du point de vue des algorithmes. La généricité des algorithmes est reflétée par l'implantation matérielle et donne accès à un nombre plus large de traitements.

Lors de l'étude des algorithmes, nous avons identifié un verrou technologique empêchant une implantation efficace principalement des opérateurs de segmentation des images. Ce verrou technologique est le calcul de la fonction distance qui pourrait fournir des résultats avec une précision sous-pixélique et serait plus efficace aussi bien sur l'image entière que sur le Narrow Band d'une courbe.

Nous avons levé ce verrou technologique par la définition d'un algorithme parallèle pour la fonction distance (classique ou pondérée). L'algorithme a été nommé Massive Marching d'une part pour refléter sa façon de propager la solution et d'autre part, parce qu'il procède de manière entièrement parallèle. Il est fondé sur une propagation non équidistante qui ne nécessite aucune structure ordonnée. Bien qu'il soit conçu comme entièrement parallèle, Massive Marching peut être implanté de façon semi parallèle par approche *Divide and Conquer* ou de façon séquentielle. Les expériences que nous avons menées ont montré que même implanté séquentiellement, Massive Marching reste compétitif avec d'autres algorithmes en termes de complexité de calcul et de temps d'exécution.

Le choix du type d'architecture pour l'implantation finale a été donné par l'analyse de l'état de l'art du domaine. Nous proposons deux types d'architecture. Parmi les rares implantations des méthodes basées sur les EDPs, une architecture SIMD est proposée pour le filtrage par diffusion non linéaire. Nous avons décidé d'implanter l'algorithme Massive Marching sur ce type d'architecture afin de pouvoir effectuer le filtrage et la segmentation sur la même architecture. Les performances estimées de l'architecture présentée dans cette thèse reflètent les contraintes physiques de réalisation sur un FPGA. Pour une image 256×256, la fréquence maximale estimée par les outils de synthèse a été de 150 MHz pour les calculs arithmétiques et 75 MHz pour l'accès à la mémoire. Avec cette vitesse de traitement, le temps d'exécution estimé du Massive Marching est de 4ms dans le pire des cas.

Afin de profiter au maximum de la généricité des algorithmes aux EPDs, nous proposons une deuxième architecture de type multiprocesseur. Elle est de type MIMD et opère en mode SPMD. Les accès à la mémoire partagée sont gérés par le principe des sémaphores dont l'efficacité est démontrée par l'implantation de la LPE continue sur cette architecture. Les mesures effectuées permettent de comparer les bandes passantes et les temps d'exécution des différentes plateformes allant d'un ordinateur de bureau jusqu'aux outils portables de type PDA (Personnal Digital Assistant). De cette comparaison il résulte que notre architecture a, à 120 MHz, une performance comparable à un Pentium à 1.6 GHz et qu'elle devance largement des processeurs de type ARM à l'intérieur des PDAs.

Perspectives

Dans l'avenir, le travail présenté dans ce document pourrait avoir une suite aussi bien sur le plan théorique que sur le plan de la conception des architectures dédiées.

Sur le plan théorique : En 3D, le front de propagation peut contenir un nombre considérable de points. L'extension en 3D de l'algorithme de calcul de la fonction distance Massive Marching aurait pour avantage d'éliminer la gestion des structures ordonnées de grande taille.

Rappelons que la gestion d'une telle structure est caractérisée, dans le meilleur des cas, par une complexité (en moyenne) logarithmique, alors que le Massive Marching n'utilise qu'une simple pile à complexité constante.

Sur le plan de l'implantation matérielle : Actuellement, le nombre habituel de processeurs embarqués sur les composants disponibles est de quatre. Avec le progrès technologique, ce nombre va, dans les mois à venir, sans doute augmenter. Il sera donc nécessaire d'effectuer une étude de la scalabilité de l'architecture multiprocesseur proposée dans ce document et, éventuellement de mettre en œuvre un partage des périphériques mieux équilibré, à l'aide des sémaphores dotés de la capacité de respecter l'ordre d'arrivée des demandes.

A plus long terme, les avantages principaux des architectures proposées dans ce document, l'importante bande passante de l'architecture SIMD (convenant aux filtres) d'un côté, et la programmabilité logicielle de l'architecture multiprocesseur de l'autre côté (convenant plus aux méthodes à évolution de courbes), suscite l'envie de combiner les deux avantages.

La formulation des méthodes à évolution de courbes sous forme vectorielle nous permettra d'effectuer des calculs vectoriels sur la totalité du Narrow Band.

Travailler avec des fonctions non linéaires imposera sans doute l'utilisation d'une ALU complète, à mot très large, nécessitant une surface considérable. L'implantation d'une telle ALU se heurte aux limitations physiques des chips actuels, ne permettant de traiter le Narrow Band que par parties. Or, la surface des composants étant en constante augmentation, le traitement pourrait être, dans le futur, étendu sur la totalité du Narrow Band.

Bibliographie

Altera (2003). Nios embedded processor, *Technical Report 3.1*, Altera.

Alvarez, L., Guichard, F., Lions, P. and Morel, J. (1993). Axioms and fundamental equations af image processing, *Arch. Rational Mech. Anal.* **123** : 199–257. Springer-Verlag.

Alvarez, L., Lions, P. and Morel, J. (1992). Image selective and edge detection by nonlinear diffusion, *SIAM J. Numer. Analysis*, Vol. 43, pp. 845–866.

AMD (2003). *Documentation techniques AMD*, AMD, http ://www.amd.com/.

ARM (2000). ARM architecture, *Technical report*, ARM Limited.

Aubert, G., Barlaud, M., Faugeras, O. and Jehan, S. (2003). Image segmentation using active contours : calculus of variations or shape gradients ?, *SIAM* **63**(6) : 2128–2154.

Beucher, S. (2002). Algorithmes sans biais de ligne de partage des eaux, *Technical report*, Centre de Morphologie Mathématique / ENSMP.

Beucher, S. and Lantuéjoul, C. (1979). Use of watersheds in contour detection, *International Workshop on Image Processing : Real-time edge and motion/detection/estimation. - CCETT - INSA - IRISA,*, Rennes, pp. 2.1–2.12. 609.

Beucher, S. and Meyer, F. (1993). *The morphological approach to segmentation : the watershed transformation*, Marcel Dekker, chapter 12, pp. 433–481.

Bieniek, A. (2000). *Divide-and-conquer parallelisation methods for digital image processing algorithms*, Vol. 10 of *VDI Fortschritt-Berichte*, VDI Verlag, Düsseldorf.

Boué, M. and Dupuis, P. (1999). Markov chain approximations for deterministic control problems with affine dynamics and quadratic cost in the control, *SIAM Journal on Numerical Analysis* **36**(3) : 667–695.

Bresson, X., Vandergheynst, P. and Thiran, J. (2003). A priori information in image segmentation : Energy functional based on shape statistical model and image information, *IEEE ICIP03, Spain.*

Brockett, R. and Maragos, P. (1992). Evolution equations for continuous–scale morphology, *ICASSP*, Vol. 3, pp. 125–128.

Broggi, A., Conte, G., Gregoretti, F., Sansoè, C. and Reyneri, L. M. (1994). The PAPRICA massively parallel processor, *IEEE Intl. Conf. on Massively Parallel Computing Systems*, pp. 16–30.

Casselles, V., Catte, F., Coll, T. and Dibos, F. (1993). A geometric model of active contours, *Numerishe mathematick* **66** : 1–31.

Celoxica (2002). Technical note cel-000049, *Technical Report November*, Celoxica Inc.

Cohen, L. (1991). On active contour models and balloons, *Computer Vision, Graphics, and Image Processing. Image Understanding* **53**(2) : 211–218.

Cohen, L. and Kimmel, R. (1997). Global minimum for active contour models : A minimal path approach.

Cypher, R. and Sanz, J. L. C. (1989). Simd architecture and algorithms for image processing and computer vision, *IEEE Transaction on Acoustics, Speech, and Signal Processing* **37**(12) : 2158–2174.

Danielsson, P. E. (1980). Euclidean distance mapping, *Computer Graphics and Image Processing* **14**(3) : 227–248.

Debes, E. (2003). Recent changes and future trends in general purpose processor architectures to support image and video applications, *IEEE International Conference on Image Processing*.

Dejnožková, E. (2002). Massive marching : A parallel computation of distance function for PDE-based applications, *Technical Report N-17/02/MM*, ENSMP, Center of Mathematical Morphology.

Dejnožková, E. and Dokládal, P. (2003). A parallel architecture for curve-evolution PDEs, *Image Analysis and Stereology* **22**. June.

Deschamps, T. and Cohen, L. (2000). Fast extraction of minimal paths in 3D images and applications to virtual endoscopy, *Technical Report 0026*, Les cahiers du Cérémade.

Digabel, H. and Lantuéjoul, C. (1977). Iterative algorithms, *Proceedings, 2nd European Symposium on quantitative analysis of microstructures in material science, biology and medicine*, Caen, Octobre 1977, pp. 85–89.

Dokládal, P., Enficiaud, R. and Dejnožková, E. (2004). Contour-based object tracking with gradient-based contour attraction field, *International Conference on Acoustics, Speech and Signal Processing, ICASSP*, IEEE. Proceedings.

Eggers, H. (1997). Fast parallel euclidian distance transformation in $\mathbf{Z^n}$, *SPIE Proceedings* **3168** : 33–40.

Epstein, C. and Gage, M. (1987). *Wave motion : theory, modelling and computation*, Vol. 7 of *Math. Sci. Res. Int. Publ.*, Springer, chapter The curve shortening flow, pp. 15–59.

Flynn, M. J. (1966). Very high-speed computing systems, *Proc. of the IEEE*, number 54, pp. 1901–1909.

Gabor, D. (1965). Information theory in electron microscopy, *Lab. Invest.* **14** : 801–807.

Gastaud, M., Barlaud, M. and Aubert, G. (2004). Combining shape prior and statistical features for active contour segmentation, *IEEE trans on CSVT* .

Gelsinger, P. (2001). Présentation à l'ISSCC, Intel Corporation.

Gibbons, A. and Rytter, W. (1988). *Efficient parallel algortihms*, Cambridge university press, Cambridge.

Gijbels, T., Six, P., Gool, L. V., Catthoor, F., Man, H. D. and Oosterlinck, A. (1994). A VLSI architecture for parallel non-linear diffusion with applications in vision, *IEEE, Workshop on VLSI Sig. Proc.* **7** : 398–407.

Gjessing, S. and Maus, A. (2002). A fairness algorithm for high-speed networks based on a resilient packet ring architecture, *Proceedings of 2002 IEEE International Conference on Systems, Man and Cybernetics.*

Goldenberg, R., Kimmel, R., Rivlin, E. and Rudzsky, M. (2001). Fast geodesic active contours, *IEEE Trans. on Image Processing* **10(10)** : 1467–75.

Gomez, J. and Faugeras, O. (1992). Reconciling distance functions and level sets, Technical report No : 3666, INRIA.

Guichard, F. and Morel, J. (1995). Introduction to partial differential equations in image processing, *Tutorial Notes, IEEE Int. Conf. Image Proc., Washington.*

Guichard, F. and Morel, J. (2001). Image analysis and P.D.E.'s, Cours DEA, ENS Cachan.

Gumm, M. (1995). *VLSI Design Course : VHDL-Modelling and synthesis of the DLXS RISC Processor*, University of Stuttgart.

Hellwig, G. (1977). *Partial Differential Equations*, Teubner, Stuttgart.

Holmgren, S. and Wallin, D. (2001). *Performance of High-Accuracy PDE Solvers on a Self-Optimizing NUMA Architecture*, number LNCS 2150, Springer-Verlag, pp. 602–610.

Hummel, R. A. (1986). Representations based on zero-crossings in scale-space, *Computer Vision and Pattern recognition*, IEEE, New york, pp. 204–209.

Hwang, K., Tseng, P. S. and Kim, D. (1989). An orthogonal multiprocessor for parallel scientific computations, *IEEE Transaction on computers*, Vol. 38, pp. 47–61.

Intel (2003). *Intel consumer desktop PC microprocessor history timeline*, Intel, http ://www.intel.com/.

Jackway, P. and Deriche, M. (1996). Scale-space properties of the multiscale morphological dilation-erosion, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **18**(1) : 38–51.

Jain, A. K. (1977). Partial differential equations and finite-difference methods in image processing, part 1 : image representation, *J. Optimiz. Theory Application* **23** : 65–91.

Jehan, S., Barlaud, M. and Aubert, G. (2003). DREAM^{2}S : Deformable Regions driven by an Eulerian Accurate Minimization Method for image and video segmentation, *International Journal of Computer Vision* **53** : 40–70.

Jeulin, D., Osmont, P. and Larquetoux, G. (1987). Simulation of crack propagation in a a porous material by image analysis, *Acta stereologica* **6**(3) : 381–386. Proceeding, I.C.S. 7, Caen, 2-9 Septembre.

Jeulin, D., Vincent, L. and Serpe, G. (1990). Propagation algorithms on graphs for physical applications, *To be publ. in the journal of visual communication and image representation*, CG/CMM Fontainebleau, ENSMP. 1969.

Kass, M., Witkin, A. P. and Terzopoulos, D. (1987). Snakes : active contour models, *International Journal of Computer Vision* **1** : 321–331.

Kichissammany, S., Kumar, A., Olver, P. J., Tannenbaum, A. and Yezzi, A. (1995). Gradient flows and geometric active contours, *Fifth International Conference on Computer Vision*, Cambridge, pp. 810–815.

Kim, S. (2001). An o(n) level set method for eikonal equations, *SIAM J. Sci Comput.* **22** : 2178–2193.

Kimia, B. and Siddiqi, K. (1996). Geometric heat equation and nonlinear diffusion of shapes and images, *Computer Vision and Image Understanding : CVIU* **64**(3) : 305–322.

Kimmel, R. (1995). *Curve Evolution on Surfaces*, PhD thesis, Technion Israel Institute of Technology.

Kimmel, R., Shaked, D., Kiryati, N. and Bruckstein, A. M. (1995). Skeletonization via distance maps and level sets, *Computer Vision and Image Understanding : CVIU* **62**(3) : 382–391.

Klein, J., Walter, T., Massin, P. and Dejnožková, E. (2003). Computer aided diagnosis of diabetic retinopathy : A survey, *Computer Assisted Fundus Image Analysis*, CAFIA, Torino, Italy.

Koenderink, J. J. (1984). The structure of images, *Biol. Cyber.* **50** : 363–370.

Kyo, S., Koga, T. and Okazaki, S. (2001). Imap-ce : A 51.2 gops video rate image processor with 128 vliw processing elements, *ICIP01* (III) : 294–297.

Lemonnier, F. (1996). *Architecture électronique dédiée aux algorithmes rapides de segmetation basés sur la morphologie mathématique*, PhD thesis, Ecole nationale supérieure des Mines de Paris.

Lindeberg, T. (1994). *Scale-Space Theory In Computer Vision*, Kluwer Academic Publishers, Monograph 1994.

Loan, C. V. (1992). Computational frameworks for the fast fourier tranform, *Soc. for Industr. and Appl. Math.* .

Malladi, R. and Sethian, J. A. (1996). Image processing : Flows under min/max curvature and mean curvature, *Graphical models and image processing* **58**(2) : 127–141.

Manzanera, A. (2002). Morphological segmentation on the programmable retina : towards mixed synchronous/asynchronous algorithms, *in* H. Talbot and R. Beare (eds), *ISMM02*, Sydney.

Maragos, P. and Meyer, F. (1999). Nonlinear PDEs and numerical algorithms for modeling levelings and reconstruction filters, *in* M. Nielsen, P. Johansen, O. Olsen and J. Weickert (eds), *Scale-Space Theories in Computer Vision*, number 1682 in *Lecture Notes in Computer Science*, Springer-Verlag, pp. 363–374.

Matheron, G. (1975). *Random sets and integral geometry*, Wiley, New York.

Menhert, A. J. and Jackway, P. T. (1999). On computing exact euclidian distance transform on rectangular and hexagonal grids, *Journal of Mathematical Imaging and Vision* **11** : 223–230.

Meyer, F. (1991). Un algorithme optimal de ligne de partage des eaux, *Actes 8ème Congrès AFCET Reconnaissance des Formes et Intelligence Artificielle*, Lyon-Villeurbanne, 25-29 Novembre 1991, pp. 847–857. 2422 EX N-10/91/M.

Meyer, F. (1994). Topographic distance and watershed lines, *Signal Processing* **38**(1) : 113–125.

Meyer, F. and Maragos, P. (1999a). Morphological scale-space representation with levelings, *in* Mads (ed.), *Scale-space Theories in Computer Vision*, Vol. 1682, Springer, Berlin, pp. 187–198.

Meyer, F. and Maragos, P. (1999b). Multiscale morphological segmentations based on watershed, flooding, and Eikonal PDE, *in* M. Nielsen, P. Johansen, O. Olsen and J. Weickert (eds), *Scale-Space Theories in Computer Vision*, number 1682 in *Lecture Notes in Computer Science*, Springer-Verlag, pp. 351–362.

Michel, O., Vito, D. D. and Sansonnet, J. (1996). $8\frac{1}{2}$: Data-parallelism and data-flow, *Intensional Programming II : 9th Int. Symp. on Lucid and Intensional Programming* .

Miller and Stout (1999). Algorithmic techniques for networks of processors, *Algorithms and Theory of Computation Handbook*, CRC Press.

Moga, A., Viero, T., Dobrin, B. and Gabbouj, M. (1994). *Mathematical Morphology and its applications to Image and Signal Processing*, Kluwer Academic Publishers, chapter Implementation of a distributed watershed algorithm, pp. 281–288.

Montanvert, A. and Chassery, J. M. (1991). *Géométrie discrète en analyse d'images*, Hermès Paris.

Moore, G. E. (1965). Cramming more components onto integrated circuits, *Electronics* **38**(8) : 114–117.

Motorola (2003). *PowerPC Technical documentation*, Motorola, http ://e-www.motorola.com/.

Najman, L. and Couprie, M. (2003). *Watershed algorithms and contrast preservation*, Vol. 2886 of *DGCI'03*, Springer Verlag, chapter Lecture note in Computer Sciences.

Najman, L. and Schmitt, M. (1994). Watershed of a continuous function, *Signal Processing* **38** : 99–112.

Nessyahu, H. and Tadmor, E. (1990). Non-oscillatory central differencing for hyperbolic conservation laws.

Noguet, D. (2002). *Architectures parallèles pour la morphologie mathématique géodésique*, PhD thesis, Institut National Polytechnique De Grenoble, Techniques de l'Informatique et de la Microélectronique pour l'Architecture des ordinateurs.

Noordergraaf, L. and van der Pas, R. (1999). Performance experiences on sun's wildfire prototype, *Proceedings of the 1999 ACM/IEEE conference on Supercomputing (CDROM)*, ACM Press, p. 38.

Osher, S. and Rudin, L. (1988). Featured-oriented image enhancement using shock filters, *SIAM J. on Numerical Analysis* (27) : 919–940.

Osher, S. and Sethian, J. (1988). Fronts propagating with curvature-dependent speed : Algorithms based on Hamilton-Jacobi formulations, *Journal of Computational Physics* **79** : 12–49.

Osher, S. and Shu, C.-W. (1991). High-order Essentially Non-oscillatory schemes for Hamilton-Jacobi equations, *SIAM Journal of Numerical Analysis* **28**(4) : 907–922.

Paragios, N. (2000). *Geodesic Active Regions and Level Set Methods : Contributions and Applications in Artificial Vision*, PhD thesis, Université de Sophia Antipolis.

Perona, P. and Malik, J. (1988). A network for multiscale segmentation, *Proceedings IEEE, International Symposium Circuits and Systems CISCAC'88* pp. 2565–2568.

Perona, P. and Malik, J. (1990). Scale-space and edge dectecting using anisotropic diffusion, *PAMI* (12) : 629–639.

Peyrard, R. and Klein, J. (1991). Speeding up mathematical morphology processes by using a new asic : Pimm1, *Proc. Computer Architecture for Machine Perception - CAMP 91, ETCA/CNRS, IEEE/AFCET, Zavidovique B., Wendel P-L. (Eds)*, pp. 439–452. 2411.

Raineault, T. (2001). Semaphore aid multiprocessor design, *Embedded Edge, Texas Instruments* .

Roerdink, J. B. T. M. (1994). Manifold shape : from differential geometry to mathematical morphology, *in* Y.-L. O, A. Toet, D. Foster, H. J. A. M. Heijmans and P. Meer (eds), *Proceedings of the Workshop "Shape in Picture", 7–11 September 1992, Driebergen, The Netherlands*, Springer, Berlin, pp. 209–223.

Roerdink, J. B. T. M. and Meijster, A. (2000). The watershed transform : Definitions, algorithms and parallelization strategies, *Fundamenta Informaticae* **41**(1-2) : 187–228.

Ronfard, R. (1994). Region-based strategies for active contour models, *IJCV* **13**(2) : 229–251.

Rouy, E. and Tourin, A. (1992). A viscosity solution approach to shape from shading, *SIAM J. Num. Analysis* **29**(3) : 867–884.

Sapiro, G. (2000). *Geometric Partial Differential Equations and Image Analysis*, Cambridge University Press.

Schüpp, S. (2000). *Pretraitement et segmentation d'images pas mise en œuvre de techniques aux dérivées partielles : application en imagerie microscopique biomedicale*, PhD thesis, Université de Caen/Basse Normandie.

Serra, J. (1982). *Image analysis and mathematical morphology*, Academic Press, London. ENSMP - CMM Fontainebleau.

Serra, J. (2001). *Lecture Notes on Morphological Operators*, number 1 in *ISSN 1103-467X*, Institut Mittag-Leffler, The Royal Swedish Academy of Sciences.

Sethian, J. (1996). *Level Set Methods*, Cambridge University Press.

Sethian, J. A. (1999). Fast marching methods, *SIAM Review* **41**(2) : 199–235.

Shreedhar, M. and Varghese, G. (1995). Efficient fair queueing using deficit round robin, *SIGCOMM*, pp. 231–242.

Suri, J., Singh, S. and Reden, L. (2001). Computer vision and pattern recognition techniques for 2-D and 3-D MR cerebral cortical segmentation : A state-of-the-art review, Accepted for publication in International Journal of Pattern Analysis and Applications, 2001.

Suri, J., Wu, D., Reden, L., Gao, J., Singh, S. and Laxminarayan, S. (2001). Modeling segmentation via geometric deformable regularizers, pde and level sets in still and motion imagery : a revisit, *International Journal of Image and Graphics* **1**(4) : 681–734.

Tek, H. and Kimia, B. (1998). Curve evolution, wave propagation, and mathematical morphology, *Fourth International Symposium on Mathematical Morphology* p. 115 :126.

Trynosky, S. (2003). Serial backplane interface to a shared memory, *Technical Report XAPP648*, Xilinx.

Tsai, Y. (2000). Rapid and accurate computation of the distance function using grids, *Technical Report 17*, University of California, USA.

van den Boomgaard, R. (1998). *Numerical solutions of morphological partial differential equations based on the moprhological facet model*, chapter Mathematical Morphology and its Applications to Image and Signal Processing.

van den Boomgaard, R. and Smeulders, A. (1994). The differential equations of morphological scale-space, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **16**(11) : 1101–1113.

Verbeek, P. and Verwer, B. (1990). Shading from shape, the eikonal equation solved by grayweighted distance transform, *Pattern Recognition Letters* **11**(10) : 681–690.

Vincent, L. (1990). *Algorithmes morphologique a base de files d'attente et de lacets, Extension aux graphes*, PhD thesis, Ecole nationale supérieure des Mines de Paris.

Vincent, L. and Jeulin, D. (1989). Minimal paths and crack propagation simulations, *5 ECS*, Vol. 8 of *Acta Stereologica*, Freiburg, Sept. 89, pp. 487–494. 1708.

Vincent, L. and Soille, P. (1991). Watersheds in digital spaces : An efficient algorithm based on immersion simulations, *IEEE Transactions on Pattern Analysis and Machine Intelligence* **13**(6) : 583–598.

Walter, T. (2003). *Application de la Morphologie Mathématique au diagnostic de la Rétinopathie Diabétique à partir d'images couleur*, PhD thesis, Centre of Mathematical Morphology, Paris School of Mines.

Weickert, J. (1998). Efficient image segmentation using partial differential equations and morphology, *Technical report*, Department of Computer Science, University of Copenhagen.

Weickert, J. and Kühne, G. (2003). *Geometric level set methods in imaging, vision and graphics*, Springer Verlag, chapter 2, pp. 43–58.

Weickert, J., ter Haar Romeny, B. M. and Viergver, M. A. (1998). Efficient and reliable schemes for nonlinear diffusion filtering, *Transactions on Image Processing*, Vol. 7 :3, IEEE, pp. 398–410.

Witkin, A. P. (1983). Scale-space filtering, *in* A. Inc. (ed.), *International Joint Conference on Artificial Intelligence*, New York, pp. 1019–1021.

Wong, S., Vassiliadis, S. and Cotofana, S. (2002). Future directions of (programmable and reconfigurable) embedded processors, *2nd Workshop on System Architecture Modeling and Simulation (SAMOS2002)*.

Xilinx (2002). *Virtex-II Pro Platform, FPGA Documentation : PowerPC 405 Processor*, Xilinx. ML300 Release.

Xilinx (2003). *User's guide Virtex II*, Xilinx, http ://www.xilinx.com/.

Zhu, S. and Youille, A. (1996). Region competition : unifying snakes, region growing, and bayes/mdl for lultiband image segmentation, *PAMI* **18**(9) : 884–900.

Liste des auteurs

Altera, 162, 177
Alvarez, L., 15, 22, 39, 40, 177
AMD, 5, 14, 177
ARM, 162, 177
Aubert, G., 57, 87, 177, 179, 180

Barlaud, M., 57, 87, 177, 179, 180
Beucher, S., 51, 54, 177
Bieniek, A., 113, 164, 166, 177
Boué, M., 76, 80, 93, 177
Bresson, X., 151, 177
Brockett, R., 43, 64, 177
Broggi, A., 114, 178
Bruckstein, A. M., 41, 180

Casselles, V., 56, 178
Catte, F., 56, 178
Catthoor, F., 119, 179
Celoxica, 161, 178
Chassery, J. M., 55, 181
Cohen, L., 50, 56, 57, 178
Coll, T., 56, 178
Conte, G., 114, 178
Cotofana, S., 111, 116, 184
Couprie, M., 51, 182
Cypher, R., 113, 178

Danielsson, P. E., 76, 82, 102, 178
Debes, E., 15, 115, 178
Dejnožková, E., 6, 58, 59, 96, 164, 178, 180
Deriche, M., 43, 179
Deschamps, T., 50, 178
Dibos, F., 56, 178
Digabel, H., 51, 178
Dobrin, B., 114, 181
Dokládal, P., 6, 59, 178
Dokládal, P., 164, 178
Dupuis, P., 76, 80, 93, 177

Eggers, H., 76, 178
Enficiaud, R., 6, 59, 178
Epstein, C., 26, 178

Faugeras, O., 29, 72, 86, 87, 177, 179
Flynn, M. J., 112, 178

Gabbouj, M., 114, 181
Gabor, D., 15, 179
Gage, M., 26, 178
Gao, J., 59, 72, 183
Gastaud, M., 57, 179
Gelsinger, P., 14, 179
Gibbons, A., 114, 179
Gijbels, T., 119, 179
Gjessing, S., 162, 179
Goldenberg, R., 34, 179
Gomez, J., 29, 72, 86, 179
Gool, L. V., 119, 179
Gregoretti, F., 114, 178
Guichard, F., 15, 22, 38, 40, 177, 179
Gumm, M., 162, 179

Hellwig, G., 38, 179
Holmgren, S., 149, 179
Hummel, R. A., 15, 179
Hwang, K., 119, 179

Intel, 5, 14, 179

Jackway, P., 43, 179
Jackway, P. T., 76, 181
Jain, A. K., 15, 179
Jehan, S., 57, 87, 177, 180
Jeulin, D., 50, 180, 184

Kass, M., 56, 180
Kichissammany, S., 57, 180
Kim, D., 119, 179
Kim, S., 79, 180
Kimia, B., 38, 41, 42, 64, 180, 183
Kimmel, R., 34, 41, 50, 63, 150, 178–180
Kiryati, N., 41, 180
Klein, J., 6, 58, 114, 180, 182
Koenderink, J. J., 15, 180
Koga, T., 114, 180
Kumar, A., 57, 180
Kyo, S., 114, 180
Kühne, G., 57, 59, 65, 74, 184

Lantuéjoul, C., 51, 177, 178
Larquetoux, G., 50, 180
Laxminarayan, S., 59, 72, 183
Lemonnier, F., 114, 164, 170, 180
Lindeberg, T., 38, 41, 180
Lions, P., 15, 22, 39, 40, 177
Loan, C. V., 149, 180

Malik, J., 34, 39, 182
Malladi, R., 56, 180
Man, H. D., 119, 179
Manzanera, A., 114, 181
Maragos, P., 43, 44, 46, 48, 54, 64, 164, 177, 181
Massin, P., 6, 58, 180
Matheron, G., 16, 27, 41, 181
Maus, A., 162, 179
Meijster, A., 51, 53, 182
Menhert, A. J., 76, 181
Meyer, F., 43, 44, 46, 48, 52–54, 64, 164, 177, 181
Michel, O., 116, 181
Miller, 115, 181
Moga, A., 114, 181
Montanvert, A., 55, 181
Moore, G. E., 13, 181
Morel, J., 15, 22, 38–40, 177, 179
Motorola, 5, 14, 181

Najman, L., 51, 53, 54, 164, 182
Nessyahu, H., 69, 182
Noguet, D., 7, 113, 115, 153, 170, 182
Noordergraaf, L., 149, 182

Okazaki, S., 114, 180
Olver, P. J., 57, 180
Oosterlinck, A., 119, 179
Osher, S., 16, 27, 40, 42, 61, 69, 71, 182
Osmont, P., 50, 180

Paragios, N., 59, 151, 182
Perona, P., 34, 39, 182
Peyrard, R., 114, 182

Raineault, T., 161, 182
Reden, L., 37, 59, 72, 183
Reyneri, L. M., 114, 178
Rivlin, E., 34, 179
Roerdink, J. B. T. M., 41, 51, 53, 182
Ronfard, R., 57, 183
Rouy, E., 64, 183
Rudin, L., 40, 182
Rudzsky, M., 34, 179
Rytter, W., 114, 179

Sansonnet, J., 116, 181
Sansoè, C., 114, 178
Sanz, J. L. C., 113, 178
Sapiro, G., 24–26, 42, 117, 150, 151, 183
Schüpp, S., 44, 46, 49, 50, 63, 183

Schmitt, M., 53, 54, 164, 182
Serpe, G., 50, 180
Serra, J., 39, 41, 43, 48, 183
Sethian, J., 16, 27, 32–35, 42, 50, 61, 68, 71, 77, 82, 117, 149, 182, 183
Sethian, J. A., 56, 102, 180, 183
Shaked, D., 41, 180
Shreedhar, M., 162, 183
Shu, C.-W., 69, 182
Siddiqi, K., 38, 41, 180
Singh, S., 37, 59, 72, 183
Six, P., 119, 179
Smeulders, A., 43, 183
Soille, P., 52, 184
Stout, 115, 181
Suri, J., 37, 59, 72, 183

Tadmor, E., 69, 182
Tannenbaum, A., 57, 180
Tek, H., 41, 42, 64, 183
ter Haar Romeny, B. M., 40, 64, 65, 67, 68, 117, 184
Terzopoulos, D., 56, 180
Thiran, J., 151, 177
Tourin, A., 64, 183
Trynosky, S., 162, 183
Tsai, Y., 76, 80, 81, 83, 102, 117, 183
Tseng, P. S., 119, 179

Vandergheynst, P., 151, 177
van den Boomgaard, R., 43, 64, 183
van der Pas, R., 149, 182
Varghese, G., 162, 183
Vassiliadis, S., 111, 116, 184
Verbeek, P., 76, 183
Verwer, B., 76, 183
Viergver, M. A., 40, 64, 65, 67, 68, 117, 184
Viero, T., 114, 181
Vincent, L., 41, 50, 52, 180, 184
Vito, D. D., 116, 181

Wallin, D., 149, 179
Walter, T., 6, 58, 180, 184
Weickert, J., 34, 40, 57, 59, 64, 65, 67, 68, 74, 117, 184
Witkin, A. P., 15, 56, 180, 184
Wong, S., 111, 116, 184
Wu, D., 59, 72, 183

Xilinx, 161–163, 184

Yezzi, A., 57, 180
Youille, A., 57, 184

Zhu, S., 57, 184