THESE

présentée à

L'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

pour obtenir le titre de

DOCTEUR DE L'INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

(Arrêté ministériel du 30 Mars 1992)

spécialité :

INFORMATIQUE

par

Fabrice BELLET

Sujet de la thèse :

UNE APPROCHE INCREMENTALE A BASE DE PROCESSUS COOPERATIFS ET ADAPTATIFS POUR LA SEGMENTATION DES IMAGES EN NIVEAUX DE GRIS

Soutenue le 19 Juin 1998 devant le jury composé de :

MM.	Jean-Michel Jolion (INSA Lyon)	Rapporteur
	Dominique Barba (SEI EP Nantes)	Rapporteur
	Augustin Lux (INPG Grenoble)	Président du jury
	Catherine Garbay (TIMC-IMAG Grenoble)	Directrice de thèse
	Michel Riveill (Université de Savoie)	Examinateur
	Marc Salotti (VISIA Université de Corte)	Examinateur

Remerciements

Je tiens à remercier toutes les personnes qui ont contribué à la réalisation de cette thèse, et plus particulièrement mes chers parents pour leur soutien, ma directrice de thèse pour le temps hors norme qu'elle m'a consacré, mes rapporteurs pour leurs remarques et leur important travail d'évaluation, les membres du jury pour leur disponibilité, Marc Salotti pour avoir été l'initiateur de ce projet, mes collègues de l'équipe SIC et INFODIS pour leur endurance face à la machine à café, et enfin Guy pour ses compétences, et son talent à faire fonctionner un réseau dans des conditions parfois délicates. Qu'ils soient tous chaleureusement remerciés.

Table des matières

Introduction

Le travail effectué et présenté dans ce document se situe dans le domaine de la vision par ordinateur. Très tôt, des moyens techniques et informatiques ont permis l'acquisition et le stockage d'images sous forme numérique. Il ne s'agit ni plus, ni moins que de la manipulation d'une matrice, dont les valeurs traduisent des intensités lumineuses. Les possibilités de traitement automatique de ces images allaient pourtant s'avérer délicates, car des capacités aussi banales pour l'œil humain que la reconnaissance d'un objet ou d'un visage allaient poser de réelles difficultés pour l'outil informatique.

La recherche en vision par ordinateur a donc rapidement cherché à diminuer le volume de données à traiter, afin de concentrer le maximum d'information présente dans l'image dans un minimum de données. L'ambition était de pouvoir réaliser des traitements plus poussés sur des données restreintes. Le principe des détecteurs de contours et de régions sont apparus en réponse à ce besoin. Plutôt que de manipuler la matrice de l'image dans sa totalité, il suffit de conserver les points de l'image où il y a les plus forts changements de luminosité (les contours), et de regrouper sous une même étiquette les points qui ont une luminosité proche (les régions).

Cette volonté de simplification de la matière première de la vision par ordinateur s'est généralisée, et a abouti à une structuration du domaine. La vision de bas niveau met en œuvre des traitements de détection de contours et/ou de régions, afin de produire un résultat intermédiaire "simplifié" de la réalité. La vision de haut niveau procède à des tâches d'interprétation sur la base des contours et/ou des régions qui lui ont été fournis par le bas niveau.

D'importants problèmes apparaissent lorsque cette décomposition des tâches est utilisée :

- Il est toujours délicat de procéder à une simplification des données, car des éléments jugés peu pertinents par le bas niveau pourront manquer et s'avérer cruciaux pour le haut niveau. Le haut niveau est souvent guidé par un but, limité à un domaine précis d'utilisation, que le détecteur de contour de bas niveau ne connaît pas.
- La tâche réputée noble réside souvent dans la vision de haut niveau. Le bas niveau est alors relégué au rang d'étape préliminaire, sans grandes difficultés comparées à celles traitées par le haut niveau.
- Il existe généralement peu de communication entre le haut et le bas niveau. Les méthodes de haut niveau mettent ainsi en œuvre des méthodes lourdes et inadéquates pour pallier aux faiblesses des informations fournies par le bas niveau. Il serait plus logique de tenter de corriger les faiblesses du bas niveau, plutôt que de reporter ces erreurs sur le haut niveau.

Nous pensons que les premiers traitements en vision par ordinateur ont trop souvent été négligés. La détection de primitives dans des images naturelles est d'une difficulté particulièrement sous-estimée. Nombre de configurations locales liées à la scène, aux conditions d'éclairage, de masquage des objets les uns par les autres, génèrent autant de problèmes potentiels de segmentation, ombres, transitions floues, faibles, discontinues, textures, dégradés, jonctions, etc. Nous soutenons que la réponse à un problème difficile n'est pas forcément simple, et proposons l'idée d'appliquer des méthodes habituellement réservées au haut niveau, afin de tenter de résoudre des problèmes du bas niveau. De telles méthodes ont l'avantage de mettre opportunément l'accent sur la gestion des informations de l'image, et permettent d'exhiber un contrôle plus complet et plus fin que la plupart des approches de bas niveau classique. Ces travaux constituent une continuation des approches développées par Marc Salotti au cours de sa thèse en 1994 [Sal94], pour qui la clé se situe dans "l'accumulation d'information pertinente, au bon endroit et au bon moment [dans l'image]". Nous soutenons que cette accumulation peut être réalisée efficacement par des entités de segmentation spécialisées, générées et distribuées dynamiquement dans l'image, s'adaptant aux conditions locales, et coopérant entre elles lorsqu'elles en éprouvent le besoin. Notre ambition est donc de montrer comment des principes de gestion efficaces de l'information peuvent contribuer à s'acquiter d'une tâche de segmentation d'image en régions et en contours.

Ce mémoire s'articule autour de cinq parties relativement indépendantes les unes des autres, mettant chacune l'éclairage sur un aspect particulier du développement d'un système de vision de bas niveau, qui a été développé durant ces années de thèse. La première partie décrit les méthodes de segmentation utilisées, et justifie le choix qui a été fait de retenir des méthodes de construction incrémentales, tant pour l'élaboration des contours que des régions. Ces méthodes de segmentation sont tout d'abord présentées sous la forme d'un schéma très général, avant d'étudier quelles sont les possibilités de spécialisation offertes. La deuxième partie établit des liens entre les deux méthodes qui auront été étudiées précédemment, afin de pouvoir établir une coopération entre-elles, qui soit tout à la fois autonome, réciproque et distribuée dans l'image. La troisième partie sera un peu plus technique et décrira le mode d'implantation retenu, sous une forme très inspirée des systèmes d'exploitation d'ordinateur. Cette implantation doit permettre de manipuler les méthodes de segmentation, comme autant d'entités, localisées dans l'image, chargées d'une activité, et disposant de capacités de communication et de reproduction. La quatrième partie est une extension, qui introduit l'utilisateur, sous la forme d'une Interface Homme-Machine, comme un acteur à part entière dans le système à base de processus de vision de bas niveau précédemment décrit. L'utilisateur, avec une possibilité d'action opportuniste, ciblée, et adaptée à son expertise, peut orienter le fonctionnement global du système dans son intérêt. La cinquième et dernière partie aborde la question de la validation de la méthode. Un protocole d'évaluation est défini, justifié, puis appliqué. Divers aspects de la méthode sont étudiés qualitativement et quantitativement, tels la robustesse, la justesse, les capacités d'adaptation et de coopération.

Chapitre 1

La construction incrémentale

1.1 Introduction

Deux approches se partagent généralement le domaine de la vision de bas niveau pour la construction des primitives image. La première s'appuie sur des méthodes réalisant un nombre de passes fixe sur l'image, le plus souvent à base de filtrage de complexité variable sur les données. La seconde consiste à construire progressivement la primitive, par ajout d'éléments à une solution courante.

La gestion des informations dans les méthodes de bas niveau constitue le souci majeur qui motive notre point de vue, et la comparaison de ces deux types d'approche sous cet aspect bien précis permet de mettre en lumière des avantages et des faiblesses :

- Les approches de type filtrage sont intrinsèquement globales sur toute l'image, et des paramètres d'ajustement globaux ne permettent pas d'adapter localement le fonctionnement de ces algorithmes.
- Les approches de type filtrage sont des implantations rapides et efficaces, donc d'une utilisation simple pour l'utilisateur qui n'est pas expert du domaine. L'expert préférera une méthode qui lui offre davantage de contrôle.
- Les approches incrémentales mettent généralement en œuvre des mécanismes plus lourds, et plus lents, et répondent mal à des contraintes de temps réel.
- Cependant, elles proposent à tout instant une solution utilisable, même si elle n'a pas encore atteint un niveau de maturité suffisant. Elles tirent alors pleinement parti d'architectures de type pipe-line.
- Les approches incrémentales ont un cadre décisionnel beaucoup plus riche, car le choix d'ajouter un pixel plutôt qu'un autre se pose à tout instant de la construction de la solution.
- Les méthodes incrémentales sont des approches ouvertes, dans lesquelles les fonctions de décision sont accessibles et facilement modifiables, voire adaptables localement.

1.2 Les approches incrémentales dans la littérature

Notre approche de bas niveau s'intéresse conjointement aux primitives de type région et contour. Toutes deux se plient aisément à une construction incrémentale, même si cela est plus évident pour le contour. Les arguments du paragraphe précédent sont d'ailleurs valable pour les deux types d'objets.

Les approches incrémentales sont apparues très tôt en vision par ordinateur, dès les années soixante-dix, dans des systèmes de segmentation en régions et en contours. Cependant, elles ne sont jamais devenues aussi populaires que les méthodes de type filtrage, en particulier parce qu'elle sont plus lourdes algorithmiquement. Une des causes de la lourdeur de ces algorithmes réside dans le très grand nombre de choix possibles à une étape de construction de la primitive, ainsi que dans la préservation souhaitable mais coûteuse d'une primitive constamment utilisable. En contrepartie, et afin de conserver à ces algorithmes des temps d'exécution raisonnables, tous ces choix généralement reposaient sur des conditions très simplifiées, trop simplifiées. Ces limitations ne sont plus autant cruciales désormais, et des considérations plus fines, davantage calculatoires, peuvent être introduites dans ces algorithmes. Citons par exemple la possibilité d'étendre le "champ de vision" de l'algorithme, afin de se libérer de l'habituel masque 3x3 sur les pixels.

1.2.1 Construction de contours

La plupart des méthodes de construction de contours incrémentales consistent à rechercher un chemin optimal dans un arbre de recherche valué. Chaque nœud de l'arbre représente un pixel, un arc relie deux pixels voisins. Le contour recherché sera le chemin qui minimise une fonction d'énergie donnée, ajustée de manière heuristique. Martelli [Mar76] a développé une telle méthode fondée sur l'algorithme $A*$, Mérö en fit une extension incluant les "points de jonction". Ces techniques reposent sur l'évaluation que l'on va donner à un chemin dans l'arbre, correspondant à un contour dans l'image. Elles réalisent une minimisation globale d'une fonction d'énergie sur le chemin, et évitent ainsi les erreurs de décision locales. Mais peu de contrôle peut s'appliquer pour guider les décisions locales, autrement que par l'ajustement empirique de la mesure d'énergie d'un arc, qui mesure la qualité de transition d'un pixel à son voisin. Le contrôle complet de tels système réside entièrement dans la fonction de coût : mesure de gradient, évaluation de courbure locale (Martelli) ...

Liu [Liu77] ainsi que Chen et Siy [CS87] ont utilisé un point de vue beaucoup plus local sur la construction de contours. Les pixels contours sont étiquetés les uns après les autres, et une pile de pixels précédemment éligibles est conservée, ce qui permet d'implanter un mécanisme de "retour arrière" (backtracking) en cas d'échec local. Parmi plusieurs pixels voisins éligibles, celui ayant la plus forte norme du gradient est retenu (premier point de contrôle), l'algorithme décide de stopper la construction lorsque tous les pixels voisins ont un gradient inférieur à un seuil donné (second point de contrôle). Le "retour arrière" permet alors de remettre en cause le ou les derniers pixels agrégés, et de reprendre la croissance avec d'anciens pixels éligibles qui n'avaient pas été retenus. Liu a également généralisé cette méthode à la construction de surfaces 3D, ce qui s'avére être une anticipation ambitieuse pour l'époque. Ce principe est beaucoup plus flexible car il travaille directement au niveau du pixel (à la différence de Martelli, qui travaille au niveau du chemin dans l'arbre de décision) : il est ainsi très simple de rajouter ou bien de modifier des heuristiques, afin de contrôler un comportement local.

Bianchi [BMPR94] et Mraghni [Mra97] proposent un point de vue grammatical et lexicographique appliqué à la construction de contours, et plus généralement à l'élaboration d'une topologie de territoires dans l'image. La décision sur un pixel donné correspond à la mise en correspondance réussie avec un élément d'une grammaire, généralement un masque local de la situation du pixel. Cette décision repose ainsi sur l'historique des précédents points étiquetés, dont une partie est codée dans le masque local que l'on applique au pixel. Les points de contrôle résident complètement dans la définition de la grammaire, et présentent explicitement les possibilités de l'algorithme. L'introduction de nouvelles heuristiques par le biais d'une adaptation de la grammaire en est grandement facilitée. Mraghni peut ainsi "spécialiser" la détection de son algorithme en utilisant un jeu de règles de construction spécifiques à la forme devant être segmentée, forme circulaire, convexe, etc.

1.2.2 Construction de régions

Les méthodes de construction de régions peuvent s'appuyer sur des techniques de construction incrémentale similaires. En commençant par un petit nombre de pixels connexes, cette zone est itérativement fusionnée à d'autres petites régions avoisinantes. Horowitz et Pavlidis [HP76] ont été les précurseurs qui ont suggéré ce point de vue. L'aspect essentiel de cette approche est l'utilisation d'un arbre quaternaire de l'image ("quad tree") afin d'orienter les divisions et les fusions dans l'image. Le point de contrôle de cet algorithme consiste à définir le prédicat d'homogénéité, qui décide si une région doit être divisée ou pas, ainsi que le critère de similarité, qui choisit le meilleur couple de régions voisines devant être fusionné. L'aspect incrémental est assez évident lorsque l'algorithme fonctionne en mode "fusion de régions", à la différence que la primitive n'est pas construite par ajout de pixels, mais par ajout de paquets de pixels. Des problèmes combinatoires apparaissent avec cette méthode, en particulier la nécessité de maintenir à jour un graphe d'adjacence des régions, pour pouvoir sélectionner rapidement toutes les régions voisines d'une région courante. Il devient aussi indispensable d'ordonner les couples de régions fusionable en fonction de la valeur de leur prédicat d'homogénéité, dans le but de fusionner les régions les plus similaires en premier.

Brice et Fennema [BF70] ont étendu le contrôle de ces méthodes par l'introduction de deux heuristiques : l'heuristique *phagocyte* affirme que le couple de régions optimal est celui qui minimise la longueur de la frontière résultante, après fusion. Le prédicat d'homogénéité précédent est renommé *weakness heuristic*. Ce travail précurseur montre bien qu'il est nécessaire d'introduire une connaissance plus locale au sein de la segmentation que la simple comparaison de deux valeurs moyennes de niveau de gris, chacune se voulant représentation de plusieurs milliers de pixels. Et dans le cas présent, l'utilisation de mesures réalisées sur les pixels de la frontière des régions est un premier pas vers cette plus grande localité des traitements.

Gagalowicz et Monga [GM85] utilisent une cascade de fonctions heuristiques afin de guider un processus de fusion. Chaque critère est choisi pour être un peu moins restrictif que le précédent, et donc pour permettre de fusionner quelques régions supplémentaires par rapport au précédent critère. Le point de contrôle de situe dans la comparaison faite entre la région cible et ses régions avoisinantes, afin de choisir le meilleur couple pour cette fusion. On peut regretter le séquencement rigide de cette succession de critères, et envisager une solution plus consensuelle, qui consisterait à intégrer plusieurs de ces critères, de manière pondérée, au sein d'une même fonction de décision. Ceci permettrait de définir clairement l'importance que l'on apporte à chaque critère de fusion autrement que simplement par son ordre d'apparition dans le système. Car il apparaît que les premiers critères utilisés (respectivement la différence entre les niveaux de gris extrema puis la différence entre les niveaux de gris moyen) sont très globaux. Même si les seuils utilisés sont très stricts au départ, il n'en demeure pas moins vrai que les décisions sont prises sur la base de valeurs numériques qui sont censées englober les caractéristiques photométriques d'un très grand nombre de pixels, ce qui ne garantit donc pas que les décisions prises seront adaptées à une situation très locale. Des critères plus locaux tels que la prise en compte d'une information sur le gradient à la frontière des régions n'interviennent qu'ultérieurement, alors que les erreurs de segmentation ont pu être commises plus tôt, lors de l'utilisation des premiers critères globaux.

Chiarello et al. [CJC96] proposent une croissance de région fondée sur la gestion d'une pyramide stochastique. En partant de choix aléatoires pour le mécanisme de décimation et d'expansion, ils orientent la carte région résultat en ajustant les fonctions de décision – au départ purement aléatoires – selon les besoins de leur propre application (une application de génération de cartographie cohérente). Le choix des régions devant disparaître (décimation), ainsi que le choix des régions devant être étendues dans les zones décimées (expansion) sont les deux points de contrôle de cette approche. L'originalité de ce travail est de restreindre le nombre de régions candidates à la fusion aux seules zones issues d'une décimation, ce qui repose sur un critère de choix purement géographique.

Le travail de Pong et al. [PSWH84] est original car il propose un point de contrôle particulier consistant à scinder l'ensemble des régions courantes en deux, avec d'une part les régions qui vont contribuer au mécanisme de croissance de région, utilisant le modèle à facettes de Haralick, et d'autre part des régions qui vont demeurer non fusionables. Ces régions, nommées "edge regions", constituent ici une modélisation d'une sorte de contour épais, dans le sens ou elle sont définies comme étant des régions classiques, pour lesquelles la valeur moyenne du gradient (de leurs pixels intérieurs ?) est supérieure à un certain seuil. Cette approche a également l'effet de restreindre le nombre des objets candidats à être fusionnés à une région courante, non plus seulement selon un simple critère géographique comme précédemment, mais selon des considération photométriques.

Milgram [Mil79] ainsi que Hertz et Schafer [HS88] ne se placent pas dans une optique de fusion de régions comme les auteurs précédents puisque leurs approches ont pour but de faire coopérer une segmentation de contour, sous la forme d'une carte de contours robustes pré-calculée servant de référence, avec une méthodes de segmentation de région par seuillage successif de l'histogramme des niveaux de gris. Cette méthode peut être vue sous un aspect incrémental, car l'augmentation du seuil de l'histogramme de n à $n + k$ consiste simplement à ajouter à la région (dont les niveaux de gris par définition sont inférieurs à n) tous les pixels dont les niveaux de gris sont compris entre n et $n+k$. Les deux points de contrôle de cette approche sont d'une part le choix du seuil appliqué à l'histogramme, et la façon dont l'adéquation avec la carte contour de référence est calculée. Cette approche présente un intérêt incrémental dans la manière dont les ensemble de pixels candidats sont sélectionnés. Des critères photométriques sont utilisés comme dans le cas de Pong et al., mais à la différence des approches de type "Split and Merge" précédentes, les objets candidats à la fusion ne forment plus des ensembles disjoints de pixels, mais au contraire sont des groupes de pixels inclus les uns dans les autres. Il s'agit là d'une caractéristique qui peut s'avérer problématique, car des pixels indésirables pour la région peuvent être contenus dans toutes les régions candidates à la fusion (dans le cas où la région est seuillée au niveau de gris n et les pixels indésirables ont la valeur $n + 1$). La croissance doit alors se terminer si l'on ne désire pas que ces pixels soient inclus, alors d'autres pixels candidats, avec des niveaux de gris supérieurs à $n + 1$ seraient encore acceptables.

Un moyen brutal mais efficace pour pallier de tels inconvénients est de construire la région pixel par pixel, car ainsi les caractéristiques isolées d'un point n'avantagent pas ou ne pénalisent pas le groupe de points auquel il est artificiellement rattaché. Adams and Bishof [AB94] ont développé une approche qui repose sur ce principe de bon sens. Le démarrage se fait sur un ensemble de germes de régions, tous les pixels candidats situés dans le voisinage des régions courantes sont classés dans une liste commune. Le meilleur pixel est ajouté en premier à la région dont il est le voisin. Les pixels voisins de plusieurs régions apparaissent deux fois dans la liste, puisqu'ils sont évalués par rapport à chacune des régions qu'ils bornent. L'utilisation d'une liste unique de pixels candidats assure une croissance de région qui se fera en parallèle, dans laquelle les pixels sont fusionnés de façon concurrente aux régions pour lesquelles ils obtiennent la meilleure évaluation. Les régions homogènes seront donc naturellement privilégiées par ce mécanisme. Notre point de vue est très proche de cette méthode, à la différence qu'il nous semble plus intéressant de découpler l'aspect de parallélisme de l'aspect de segmentation afin de contrôler chacun séparément. Chaque région disposera de sa propre liste de pixels candidats, et sa croissance sera alors indépendante des autres régions. La croissance concurrente pourra être implantée par le biais d'une aide extérieure, par exemple un système d'exploitation multiprocessus.

1.3 Le schéma général

Un schéma général appliqué au cas de la vision de bas niveau a pour objectif de construire les premiers objets manipulables à partir des données extraites des images en niveaux de gris. Classiquement, les contours – définis comme des chaînes de pixels voisins les uns des autres – marquent les discontinuités visibles de l'image, les régions regroupent les pixels présentant certaines propriétés d'homogénéité. Les lieux de texture ou de dégradé apparaissent alors comme des régions de type particulier. Cette liste de primitives n'est pas exhaustive.

La construction commence à partir d'une petit ensemble de points, et la solution se développe par ajout consécutif de pixels, ou d'ensembles de pixels, à la solution courante. La boucle principale contient les étapes suivantes :

- La sélection de tous les pixels candidats, c'est à dire tous les points, dont l'emplacement par rapport à la primitive courante les rend potentiellement agrégeables, par rapport à des contraintes de connexité uniquement.
- L'évaluation des pixels. L'adéquation de chaque pixel candidat à la primitive doit être numériquement mesuré.
- Le tri des pixels candidats par rapport à la précédente évaluation. Le meilleur candidat sera retenu.
- L'intégration du meilleur pixel candidat. Les caractéristiques de la primitive peuvent alors être remises à jour.

Un critère d'arrêt supplémentaire permet de contrôler la terminaison de cette boucle. Ce schéma apparaît très général, car aucune hypothèse n'est faite sur la structure de la primitive, ni d'ailleurs sur les mécanismes de sélection et d'évaluation mis en jeu. La spécialisation de l'algorithme s'oriente alors vers :

- Le changement du type de primitive élaboré. Ce schéma s'applique à la fois à la construction incrémentale d'un contour, dans laquelle les pixels sont ajoutés à chaque extrémité, ou bien à une région, où les pixels sont sélectionnés dans le voisinage de la primitive.
- Le changement de la granularité de l'algorithme. Un contour est construit segments par segments, et une région se développe par ajouts successifs de blocs de pixels, ou d'ensembles plus exotiques de points. Pour faciliter la compréhension, nous continuerons à appeler *pixels* les objets élémentaires de la construction de la primitive.

L'autre point clé de ce modèle est la multitude de points de contrôle offerts à l'utilisateur, et dont voici quelques exemples :

- le contrôle de la sélection des pixels. La réduction du choix des pixels candidats oriente l'aspect de la primitive construite. Le choix des pixels candidats est en effet un point majeur, lorsque cela se traduit par l'exploration d'un arbre de recherche, comme dans les travaux de Martelli [Mar76] ou de Mérö [Mér81]. La limitation du nombre de pixels candidats réduit le degré de l'arbre de recherche, et par conséquent la complexité de l'algorithme, au risque néanmoins de réduire l'espace des solutions.
- les critères d'évaluation des pixels candidats.

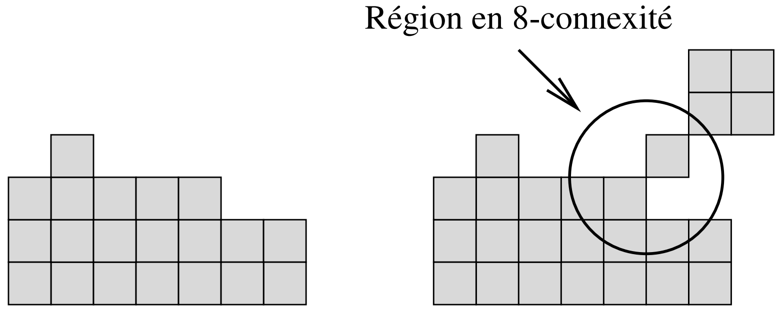


FIG. 1.1: A gauche, la région est construite en 4-connexité (pour chaque pixel, il existe un pixel de son 4-voisinage qui appartient à la même région). A gauche, la configuration encerclée place cette région en 8-connexité. Construire des régions en 4-connexité permet de réaliser des primitives plus compactes

- les heuristiques de tri des pixels candidats. Lorsque le nombre de pixels candidats est trop important pour pouvoir recalculer leur évaluation à chaque passage dans la boucle principale, il devient nécessaire de mettre en œuvre des heuristiques destinées à stocker cette évaluation lors de son premier calcul, et à savoir pendant combien d'itérations cette évaluation reste valide.
- le critère d'arrêt de la boucle d'agrégation.

1.4 La sélection des points

Le principe des méthodes incrémentales de croissance est de disposer à tout instant d'un ensemble de pixels potentiellement agrégeables à la primitive, pour ensuite pouvoir les évaluer et choisir le meilleur. La première étape de sélection est donc cruciale, car de l'ensemble des pixels sélectionnés dépend le potentiel de choix de l'algorithme, et plus généralement ce que l'on pourrait appeler le "champ de vision" de la méthode de construction.

1.4.1 Question de connexité

La primitive élaborée possède certaines contraintes de connexité qu'il convient de prendre en compte, et de refléter dans l'étape de sélection. Notre implantation construit des contours en huit-connexité.

La description des régions se fera au contraire en quatre-connexité, afin d'éviter des phénomènes de fuite de pixels, qui conduiraient trop facilement à des régions aux formes pathologiques, figure 1.1. Cette définition géométrique est d'ailleurs la seule envisageable selon le théorème de Jordan, puisqu'une région en huit-connexité permettrait à des contours en huit-connexité également de la traverser à certains endroits [1].

1.4.1.1 Cas d'un contour

La décision de construire un contour en huit-connexité impose des choix sur la manière de choisir les pixels suivants, en fonction de la disposition des deux derniers pixels agrégés à chaque extrémité. Nous proposons dans l'exemple qui suit de montrer l'influence que peut avoir ce choix des pixels suivants sur la forme globale du contour, selon que l'on teste trois ou cinq pixels à chaque extrémités dans certaines configurations.

[1]comme par exemple sur la zone encerclée de la figure 1.1

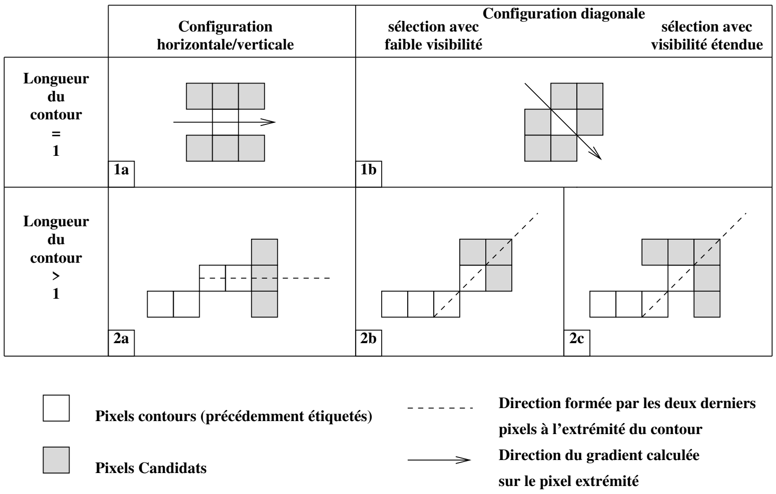


FIG. 1.2: Le mode de sélection des pixels candidats pour la croissance de contour. La mesure de la direction du gradient sert lorsque le contour ne contient qu'un seul pixel. Dans les autres cas, la configuration des pixels de l'extrémité et la nécessité de préserver la connexité permettent de retenir trois ou cinq pixels candidats

Si l'on appelle $P_1 = P(x_1, y_1)$, et $P_2 = P(x_2, y_2)$ respectivement le dernier et l'avant dernier pixel du contour, P_1 et P_2 sont des pixels voisins, ce qui se traduit par la relation $\max(|x_1 - x_2|, |y_1 - y_2|) = 1$, c'est-à-dire que la distance de P_1 à P_2 vaut 1 pour la norme ∞. Le déplacement (d_x, d_y) entre les pixels P_2 et P_1 se définit alors par $(d_x, d_y) = (x_2 - x_1, y_2 - y_1)$, avec comme contrainte $d_x, d_y \in \{-1, 0, 1\}$. On définit alors la direction de Freeman entre les pixels P_2 et P_1 par : $D = F(d_x, d_y)$, avec

F(-1,-1)=3	F(0,-1)=2	F(1,-1)=1
F(-1,0)=4	F(0,0) indéfini	F(1,0)=0
F(-1,1)=5	F(0,1)=6	F(1,1)=7

Dans le cas où la direction de Freeman F est paire (figure 1.2, cas 2a), la sélection des trois pixels grisés est effectuée, correspondant aux directions F, $(F+1)\text{mod } 8$ et $(F+7)\text{mod } 8$.

Dans le cas où la direction de Freeman est impaire (figure 1.2, cas 2b et 2c), il convient alors de sélectionner en plus les pixels correspondants aux directions $(F+2)\text{mod } 8$ et $(F+6)\text{mod } 8$. La limitation du nombre des pixels candidats en optant par exemple pour le mécanisme 2b au lieu de 2c présente des implications directes sur la construction du contour. Les deux types de sélection vont construire un contour en huit-connexité. Le cas 2b aura cependant des difficultés pour segmenter les coins aux angles aigus.

La solution la plus générique qui préserve la huit-connexité sera donc de tester trois pixels candidats lorsque l'extrémité est horizontale ou verticale, et cinq pixels lorsqu'elle est diagonale, comme l'exemple suivant va le montrer.

1.4.1.2 Exemple de l'influence de la sélection des pixels

L'exemple de l'influence de la sélection des pixels sur la forme du contour est présenté sur une image d'objets en bois, figure 1.3, dont on considère une forme triangulaire caractéristique. Cette forme n'est pas spécialement difficile à segmenter, car l'intérieur de cette forme est largement surexposé. Mais ce qui intéresse notre étude dans ce cas précis est davantage la justesse du contour obtenu. Les pixels retenus sont ceux qui réalisent la plus forte norme du gradient parmi les pixels candidats.

La figure 1.4 présente quelques étapes de la construction du contour. Son initialisation a été faite manuellement en plaçant le pixel initial sur la frontière de l'objet. Cette croissance

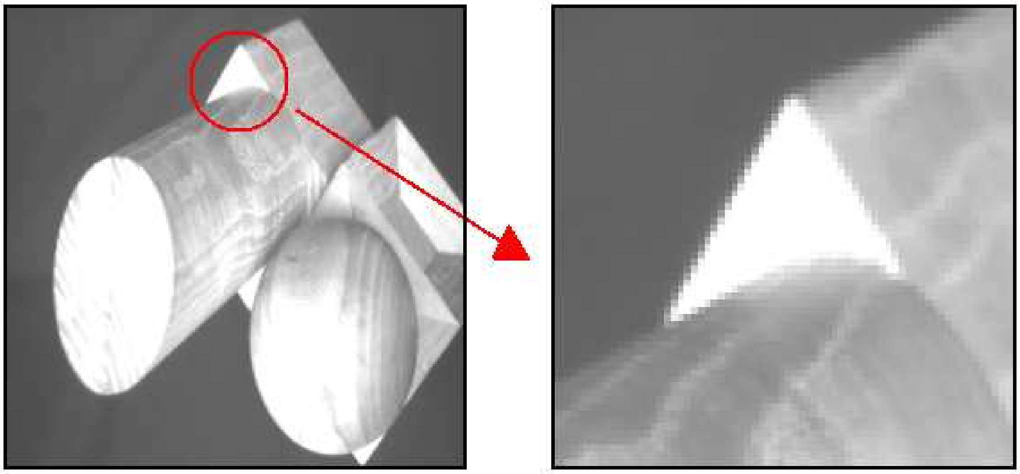


FIG. 1.3: Illustration de l'influence de l'algorithme de sélection des pixels candidats sur la segmentation d'une forme triangulaire extraite d'une image d'objets en bois

a utilisé le schéma de croissance 2b de la figure 1.2, et le contour obtenu présente justement quelques incohérences au voisinage de deux des angles qui constituent cette forme. La sélection 2b interdit au contour de former des angles à 90 degrés ce qui pénalise la détection correcte des angles (figure 1.4B et 1.4D. Le contour arrondit les angles. Plus grave, la construction se faisant de manière concurrente par les deux extrémité du contour, le coin de la forme situé en bas à droite de l'image empêche même le contour de poursuivre sa croissance, car l'angle est trop aigu.

La figure 1.5 utilise le schéma de croissance 2c, en commençant dans les mêmes conditions initiales que précédemment. Les trois angles encerclés sur l'image sont correctement segmentés, et tous correspondent à une configuration de segmentation du type 2c, comme le montrent les trois agrandissements des sous images B, C et D : le pixel retenu dans ces trois circonstances est effectivement celui qui est situé à 90 degrés par rapport à la direction formée par l'extrémité. De plus, le coin droit de la forme est correctement segmenté comme l'indique la sous-image C.

Ce petit exemple illustre plusieurs points. D'une part, il convient de ne pas négliger l'étape de sélection des pixels dans une méthode de construction incrémentale, car cela peut avoir des répercutions parfois subtiles sur la forme de la primitive obtenue. Il est souvent fait reproche aux détecteurs de contours classiques de ne pas correctement segmenter les coins, ou bien de trop arrondir ces discontinuités. Ces effets sont généralement liés à des étapes de pré-traitements consistant à lisser l'image afin d'en atténuer les bruits. L'effet de bord insidieux de tels pré-traitements est de perdre une précision de localisation lors de la détection. Une approche incrémentale, et donc par définition locale, permet tout au contraire une localisation très précise, et la segmentation correcte de formes angulaires en devient facilitée.

Une construction incrémentale de contour sera généralement robuste au bruit lorsque le pixel initial appartient effectivement à une transition, et pas à un artefact lié au bruit. En effet, le gradient d'une transition n'est pas du même ordre de grandeur qu'une perturbation des niveaux de gris liées au mécanisme d'acquisition [2]. Il n'est donc pas nécessaire de lisser uniformément l'image pour supprimer le bruit d'acquisition, il convient plutôt dans un premier temps de choisir des pixels de démarrage robustes, et dans un deuxième temps, de pouvoir reconnaître un faux contour qui serait construit en suivant des artefacts liés au bruit de l'image.

[2]Il est convenu qu'un bruit inhérent au matériel d'acquisition se limite à un écart-type σ autour de 2 ou 3.

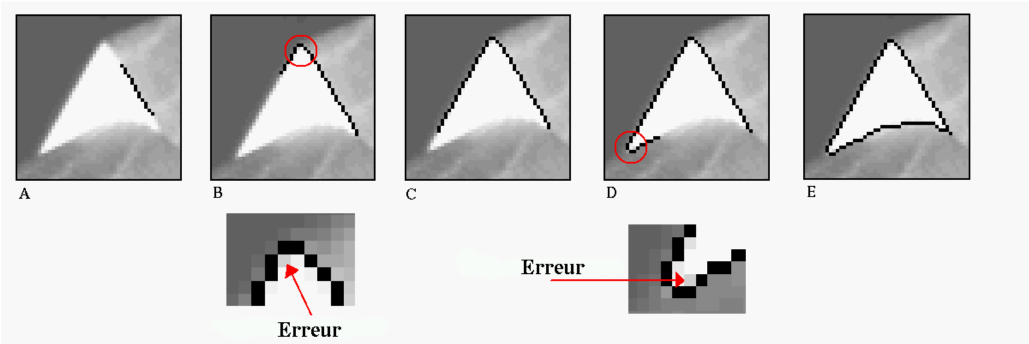


FIG. 1.4: La sélection des pixels candidats retient trois points dans les directions diagonales

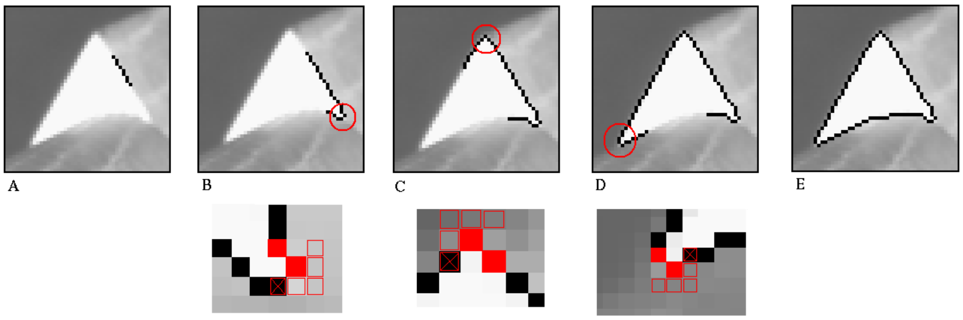


FIG. 1.5: La sélection des pixels candidats retient cinq points dans les directions diagonales

1.4.2 Le renouvellement des pixels sélectionnés

Le schéma général de notre approche incrémentale s'adapte à la fois à la construction des contours et des régions. Pourtant une différence majeure en terme de complexité va nous obliger à envisager quelques concessions à ce point de vue uniformisé.

Un contour en cours de construction dispose donc à tout instant d'un nombre N de pixels candidats à chacune de ses extrémités, N pouvant valoir 3 ou 5 en fonction de la configuration de l'extrémité [3]. Lorsque le meilleur pixel candidat a été choisi et ajouté à l'extrémité correspondante, tous les autres pixels qui étaient candidats pour cette même extrémité sont définitivement rejetés. En effet, si cela n'était pas le cas, cela impliquerait que trois pixels consécutifs d'un contour peuvent être mutuellement voisins, ce qui viole le principe de connexité qui guide la construction du contour. Cela présente plusieurs implications :

- Un pixel contour candidat n'a qu'une seule chance d'être intégré à la primitive. Les implications d'une erreur d'étiquetage sont donc grandes, car l'algorithme ne dispose pas de la possibilité de revenir en arrière. Il est donc nécessaire de motiver le plus possible le choix du pixel à fusionner au contour courant. Cela sera développé par la suite à travers des mécanismes qui évalueront ces pixels candidats.
- Entre deux étapes élémentaires de la croissance du contour, l'ensemble des pixels candidats sera en moyenne renouvelé à 50 %. Les pixels candidats conservés seront ceux correspondant à l'autre extrémité du contour, qui n'a pas évolué.

[3]La détection des intersections ne se pose pas encore à ce niveau. L'objectif de la construction incrémentale est de produire un ensemble de pixels chaînés sans ramifications. Une jonction sera ensuite une configuration où une extrémité du contour entre en contact avec un autre contour déjà construit. La construction ne peut plus progresser par cette extrémité en contact, car les pixels candidats soit appartiennent à l'autre contour, soit sont dans une configuration qui violerait la huit-connexité s'ils étaient intégrés au contour courant.

Ce dernier point mérite que l'on s'y attarde. Environ la moitié des pixels sélectionnés (puis évalués) l'avait déjà été à l'étape précédente [4]. Compte tenu du petit nombre de pixels que cela représente, ce surcoût en calcul peut être considéré comme négligeable. Tel n'est plus le cas dans la construction des régions. Le nombre de pixels candidats à chaque étape élémentaire de croissance est plus important. Tout pixel situé dans le quatre-voisinage de la région est un candidat potentiel à l'agrégation. L'ajout d'un pixel à la région ne remet plus en cause les autres pixels candidats, qui restent en lice pour l'étape suivante. Le renouvellement de la liste des pixels candidats est très faible. Il n'est donc plus envisageable d'appliquer le schéma générique consistant à sélectionner tous les pixels candidats à chaque étape élémentaire de croissance. Le schéma suivant peut s'adapter à la situation de façon plus efficace :

- *Nous considérons une liste de pixels candidats triée en fonction décroissante de leur fonction d'évaluation.*
- Le pixel avec la meilleure évaluation est extrait du sommet de la liste.
- Le pixel est ajouté à la primitive courante.
- Les nouveaux pixels candidats apparus sont sélectionnés, évalués et insérés dans la liste en fonction de leur évaluation.
- Des vérifications sont effectuées pour déterminer si les itérations peuvent se poursuivre. Ces tests incluent la vérification de la qualité de la primitive en construction, et la vérification que la liste des pixels candidats n'est pas vide.

1.4.3 Gestion de la complexité algorithmique

Le schéma décrit précédemment sera donc utilisé pour la segmentation incrémentale des régions, en raison du grand nombre de pixels potentiellement agrégeables à un instant donné. Cet algorithme manipule une liste de pixels candidats, stockant ses coordonnées ainsi qu'une évaluation numérique qui juge la qualité du pixel comparativement à la région. Cette évaluation n'est plus calculée qu'une seule fois, lorsque le pixel est inséré dans la liste. Comparativement au schéma générique dans lequel tous les pixels candidats sont réévalués, ce gain en performance pose un autre problème algorithmique de taille : est-ce que la valeur attribuée à ce pixel candidat est encore valable après plusieurs étapes d'agrégation ?

De manière évidente, cette approche est adaptée lorsque :

- le nombre de pixels à évaluer est important.
- l'évaluation attribuée à un pixel ne varie pas beaucoup lorsque la région se développe.
- la proportion de pixels dont l'évaluation change est négligeable.
- les pixels dont l'évaluation change sont facilement localisables, et peuvent donc être individuellement réévalués.

Considérons trois exemples afin d'illustrer ce point précis.

Premier exemple L'évaluation du pixel candidat est la mesure de l'intensité du niveau de gris du pixel. Les pixels candidats sont triés en fonction de cet ordre, et les premiers pixels ayant la plus forte évaluation – donc le plus haut niveau de gris – seront ajoutés en premier. L'évaluation de ces pixels est constante, et la liste n'a donc pas besoin d'être réévaluée. Un tel critère serait utilisé par exemple pour construire les régions les plus claires de l'image, dans un contexte non texturé.

[4] les pixels de l'extrémité qui n'a pas été retenue, puisque les deux extrémités peuvent croître simultanément.

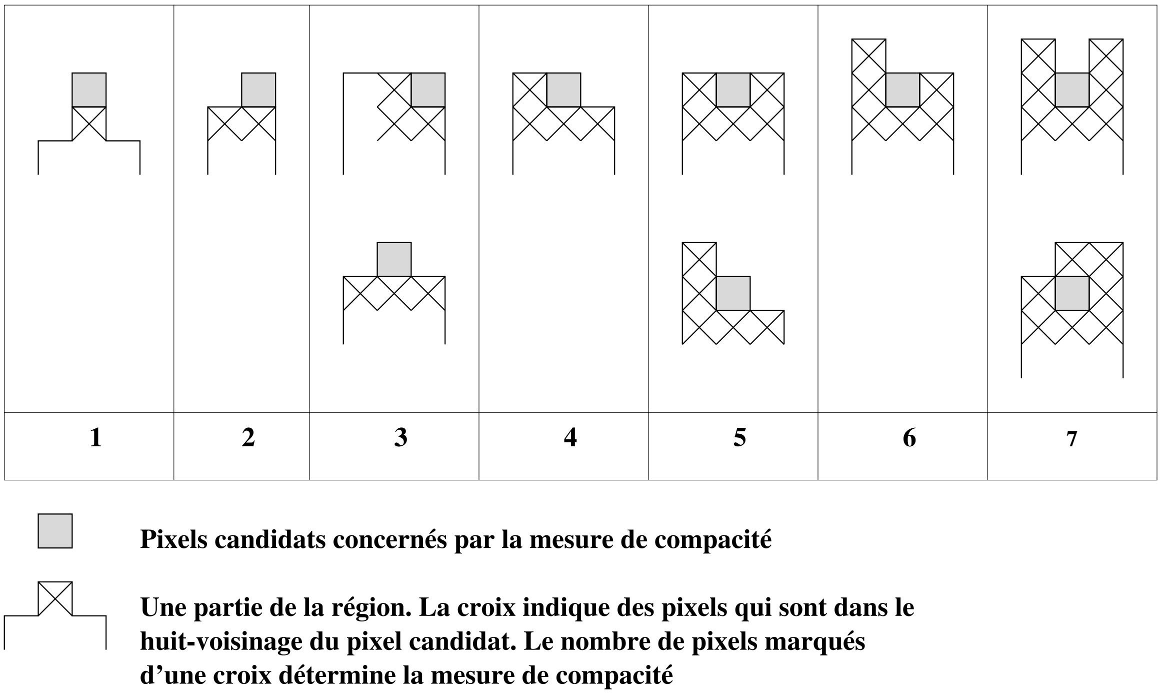


FIG. 1.6: Plusieurs exemples définissant la mesure de compacité d'un pixel candidat par rapport à sa région cible.

Deuxième exemple L'évaluation du pixel mesure cette fois la différence entre le niveau de gris du pixel et le niveau de gris moyen de la région courante. Cette évaluation associe les caractéristiques du point à celles de la région. A mesure que la région évolue, les évaluations de tous les pixels déjà en liste varient. Au bout d'un certain nombre d'itérations, l'ordre des pixels de la liste n'est plus significatif. Dans ce cas précis, on peut raisonnablement supposer que la moyenne des niveaux de gris de la région n'évolue plus sensiblement dès qu'elle a atteint un nombre suffisant de pixels. Une heuristique efficace serait alors de réévaluer fréquemment la liste complète des pixels candidats dans les premières étapes de la croissance, et de les espacer de plus en plus en fonction de la taille de la région.

Troisième exemple Dans ce dernier exemple, la fonction d'évaluation du pixel mesure un *degré de compacité* à la région courante. Pour un pixel candidat donné – en dehors de la région courante –, cette mesure compte le nombre de ses huit-voisins, qui sont déjà intégrés à la région, voir la figure 1.6. La principale différence entre cet exemple et le précédent réside dans le type de fluctuation de cette évaluation. Elle varie en effet très localement lorsqu'un pixel est fusionné à la région, et n'affecte que les pixels situés à son voisinage, voir la figure 1.7. La meilleure heuristique pour conserver une liste de candidats à jour serait d'utiliser ici une table de références croisées. Etant données des coordonnées, un pointeur correspondant dans la liste des pixels candidats permettrait de réévaluer et de réinsérer uniquement les pixels dont l'évaluation est assurée de changer (ceux indiqués précisément par des croix dans la figure 1.6).

Ces exemples montrent qu'il n'existe clairement aucune solution universelle qui puisse s'appliquer à tous les cas, et qui permette de conserver de bonnes performances, et une précision de calcul. Des heuristiques doivent être introduites, et plus précisément en accord avec la fonction ou les fonctions d'évaluation qui sont utilisées.

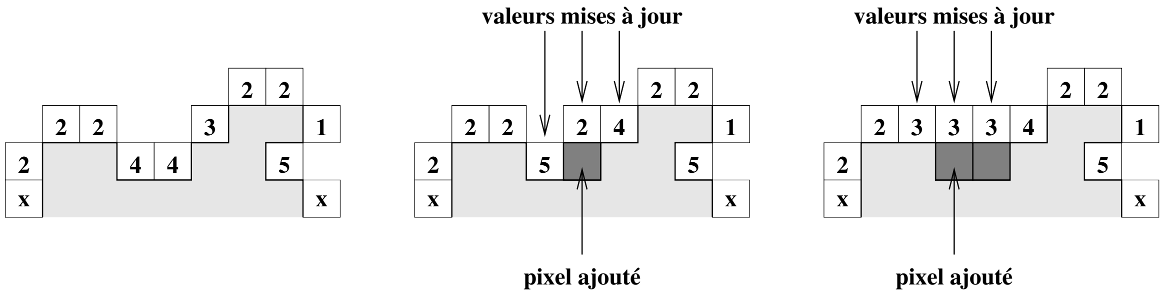


FIG. 1.7: Variation de la mesure de compacité en fonction de la croissance de la région.

1.4.4 Les choix algorithmiques de réduction de la complexité

Disposant d'une évaluation dans l'intervalle [0..1] des pixels, l'algorithme de segmentation utilise de plus un seuil permettant de séparer les pixels acceptables des autres. Ce seuil est pour l'instant fixe. Les pixels sont ainsi placés dans deux listes distinctes relativement à ce seuil. Seule la liste contenant les pixels acceptables est triée, l'autre contient des pixels en vrac. La réévaluation de tous les points consiste alors à recalculer les évaluation des pixels des deux listes, et à les reclasser en reconstruisant les deux listes par rapport à ce seuil.

La procédure d'ajout d'un pixel à la région suit ainsi un schéma de ce type :

```
void AppendRegion (PtrPixel Pixel,PtrRegion Region, ... )
{
        PtrList NewPixelsList;

        /* Mise à jour des informations photométriques de la région */
        UpdateRegionPhotometric (Pixel,Region,...);

        /* Recherche des nouveaux pixels voisins */
        NewPixelsList = FindNewNeighbours (Pixel,Region,...);

        /* Mises à jour des listes de pixels voisins de la région */
        UpdateNeighbourList (Pixel,Region,NewPixelsList,...);
}

void UpdateNeighbourList (PtrPixel Pixel,
           PtrRegion Region,
           PtrList NewPixelsList, ...)
{
        /* Efface le pixel ajouté à la région : celui-ci est dans  */
        /* la liste des pixels acceptables                        */
        DeletePixel (PixelExtractList (Pixel,Region->NeighboursOK));

        /* Ajout des nouveaux pixels voisins */
        AddNeighbours (Region,NewPixelsList);
}

void AddNeighbours (PtrRegion Region,PtrList NewPixelsList)
{
        PtrPixel NewPixel;

        while (!EmptyList (NewPixelsList)) {
           NewPixel=ExtractTop (NewPixelsList);
           ThisEval=EvaluatePixel (NewPixel,Region);
           if (Value < Region->Threshold)

              /* Le pixel est placé en vrac en fin de liste des  */
```

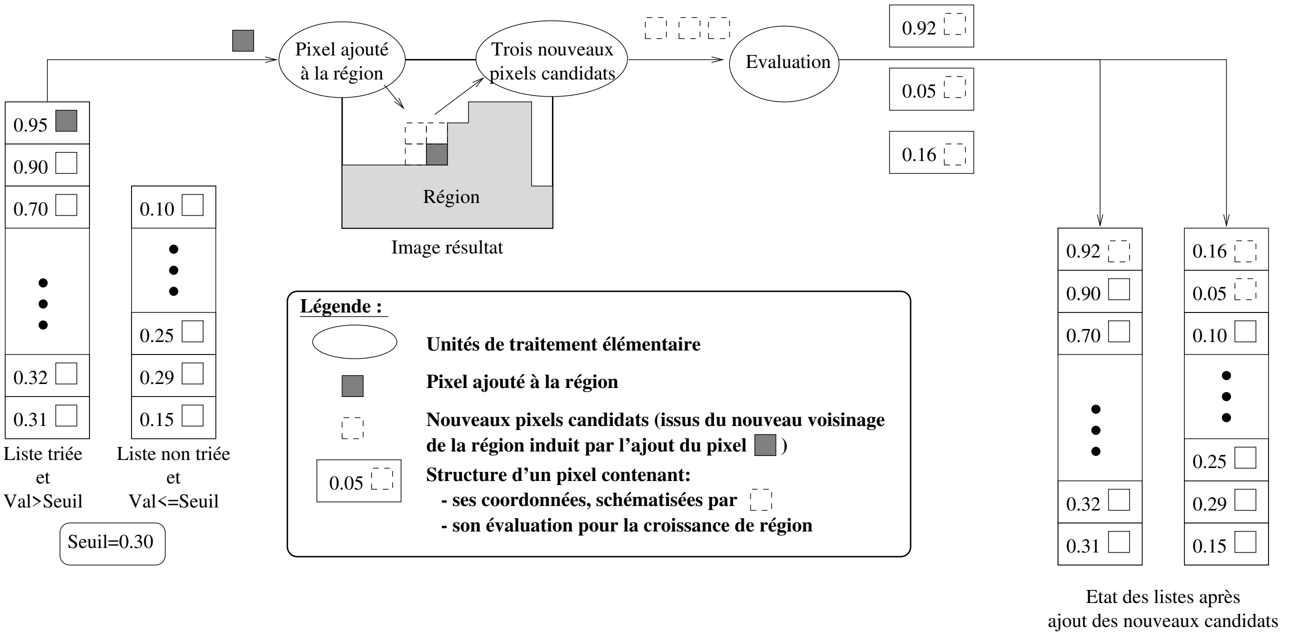


FIG. 1.8: Schéma fonctionnel de la mise à jour des listes de pixels candidats, et illustration de l'utilisation de deux listes de pixels distinctes.

```
               /* candidats non acceptables                          */
               AddBottom (Region->NeighboursKO, NewPixel);
            else

               /* Le pixel est inséré dans la liste triée des        */
               /* pixels avec une évaluation correcte                */
               AddSortedList (Region,NeighboursOK, NewPixel,DECREASING_LIST);
         }
}
```

Ce principe illustré par le schéma fonctionnel en figure 1.8, présente plusieurs intérêts :

- Un accès généralement au premier élément de la liste des candidats acceptables est suffisant pour trouver le pixel suivant à agréger.
- L'insertion d'un pixel dans la liste triée est plus rapide, car cette liste ne contient pas tous les pixels candidats.
- La liste des pixels non-acceptables peut toujours servir de *réservoir de points de secours*, lorsque la liste des pixels acceptable a été épuisée.

1.4.5 Exemple

Nous allons illustrer l'influence de la stratégie de réévaluation sur l'exemple concret suivant. L'image d'une souris en figure 1.9 montre la zone qui est recherchée. La croissance de la région commence dans tous les cas sur le pixel indiqué, situé aux coordonnées $(146, 168)$ de l'image.

Plusieurs captures d'écran ont été réalisées à des instants fixes, respectivement après que 250, 500, 1000 et 2000 pixels aient été agrégés. La région segmentée est superposée à l'image en niveaux de gris. Les pixels verts (gris clair sur l'image) correspondent aux pixels *intérieurs* à la région. Les pixels rouges (gris foncé sur l'image) sont aussi des points de la région, mais définissent sa frontière. La figure 1.10 indique que le *critère d'homogénéité* (décrit dans l'exemple numéro deux de la précédente section) utilisé pour trier les pixels candidats ne nécessite pas de réévaluation fréquente. La valeur calculée peut donc être considérée comme une information

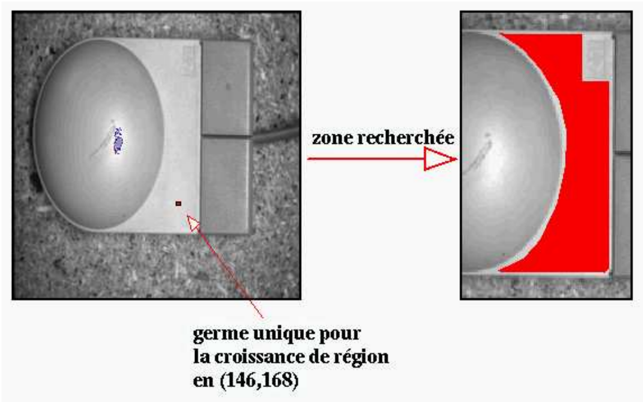
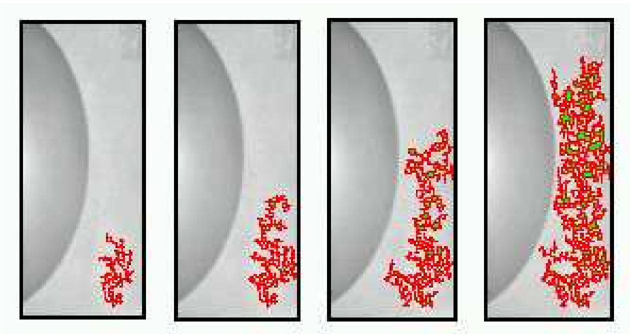
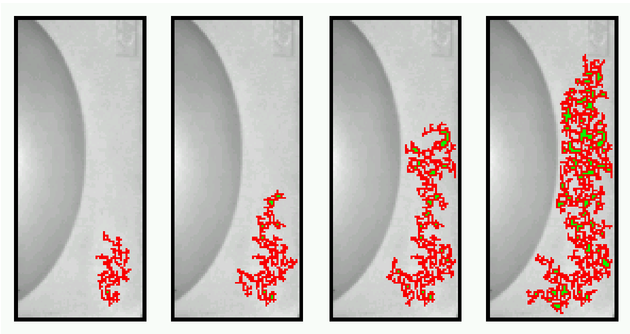


FIG. 1.9: Illustration de la méthode de croissance de région, commençant à partir d'un pixel germe aux coordonnées (146,168) de l'image de taille 256×256. La région recherchée est marquée sur l'image de droite.

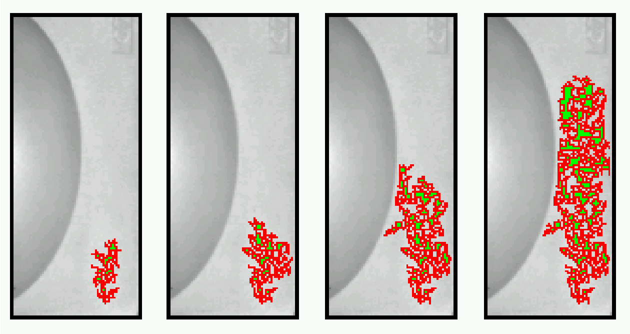


(a) La liste des candidats n'est jamais réévaluée.

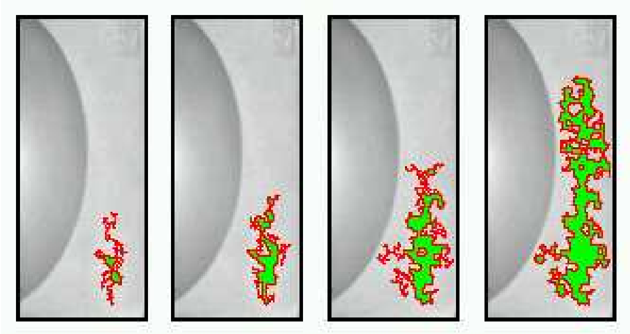


(b) La liste des candidats est réévaluée tous les 50 pixels agrégés.

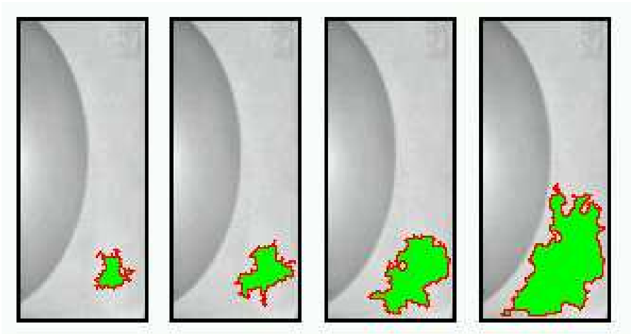
FIG. 1.10: Cet exemple présente une croissance de région utilisant un *critère d'homogénéité* décrit précédemment. Les pixels privilégiés sont ceux dont le niveau de gris est le plus proche du niveau de gris moyen de la région. Cet exemple montre que la réévaluation des pixels candidats a peu d'influence sur la manière dont la région se construit.



(a) La liste des candidats n'est jamais réévaluée.

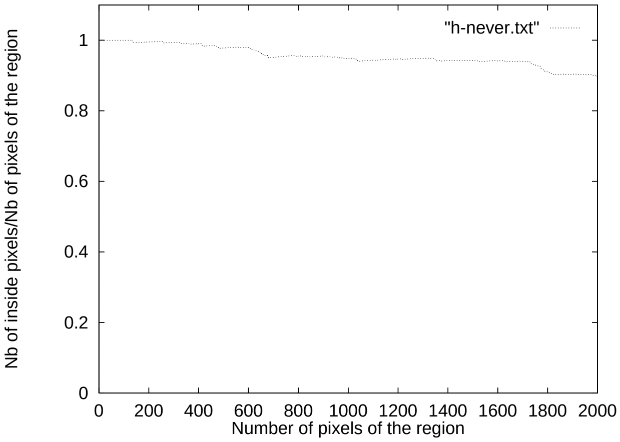


(b) La liste des candidats est réévaluée lorsqu'elle est vide.

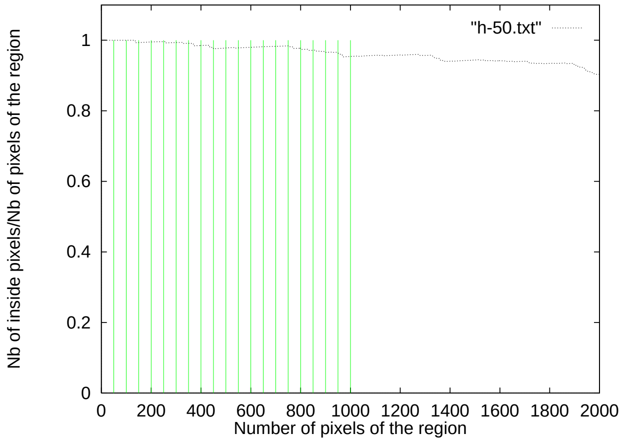


(c) La liste des candidats est réévaluée tous les 50 pixels agrégés.

FIG. 1.11: Cet exemple présente une croissance de région utilisant la *mesure de compacité* comme critère de tri des candidats. Contrairement à la mesure d'homogénéité précédente, cette évaluation est hautement sensible, et nécessite une réévaluation fréquente.

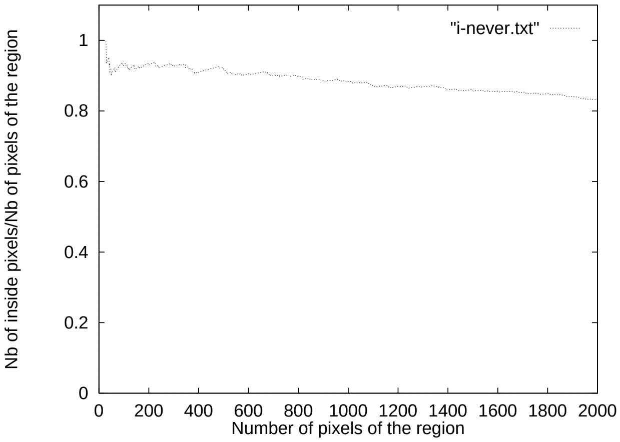


(a)

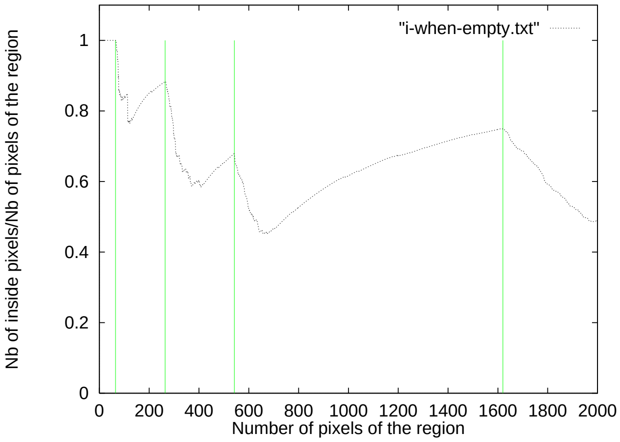


(b)

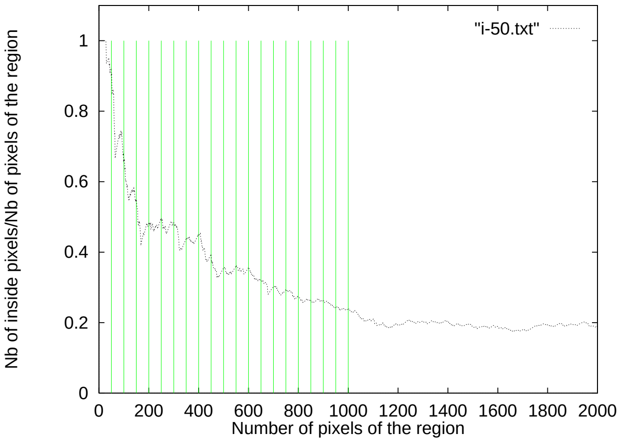
FIG. 1.12: Evolution du ratio *nombre de pixels frontière / nombre de pixels* de la région dans le cas de la croissance homogène, figure 1.10. Les graphiques h-never et h-50 correspondent respectivement aux figures 1.10a et 1.10b.



(a)



(b)



(c)

FIG. 1.13: Evolution du ratio *nombre de pixels frontière / nombre de pixels* de la région dans le cas de la croissance compacte, figure 1.11. Les graphiques i-never,i-when-empty, et i-50 correspondent respectivement aux figures 1.11a, 1.11b et 1.11c

stable. Au contraire, la figure 1.11 montre que le tri mesurant la *mesure de compacité* du pixel candidat (exemple numéro trois) est une information sensible. Les résultats sont obtenus dans les conditions expérimentales suivantes : la liste des pixels candidats n'est jamais réévaluée en (a), lorsqu'elle ne contient plus de pixels avec une évaluation suffisante en (b), et enfin systématiquement tous les 50 pixels ajoutés à la région en (c).

La figure 1.12 et 1.13 montrent une synthèse des trois précédents exemples sur le même graphe, et fournit l'évolution du ratio *nombre de pixels frontière / nombre de pixels* de la région, au cours de sa croissance. Ces figures montrent clairement que peu de différences se dégagent entre les courbes appelées `h-never`, `h-when-empty` et `h-50` sur la figure 1.12, montrant que la réévaluation n'est pas nécessaire dans ce cas pour fournir une croissance qui reflète les critères utilisés. Les courbes intitulées `i-never`, et `i-50` sur la figure 1.13 au contraire montrent comment la réévaluation peut améliorer l'effet de la mesure du *degré de compacité*, puisque la dernière courbe présente en effet le ratio le plus faible au cours de la croissance. Des droites verticales sont placées sur ces graphes lorsque la réévaluation se produit, et l'on notera que le ratio diminue immédiatement après cette réévaluation, jusqu'à atteindre un minimum. Ceci indique donc que la réévaluation produit l'effet escompté, puisque les pixels ajoutés juste après la réévaluation contribuent à faire diminuer la longueur de la frontière de la région. Leur nouvelle évaluation reflète donc mieux la réalité qu'antérieurement.

1.5 L'évaluation

1.5.1 Les conditions de l'évaluation

Le moment de l'évaluation numérique du pixel sélectionné est trivialement crucial pour le mécanisme global, aussi il convient de recueillir, à cet instant de l'algorithme, le maximum d'information disponible [SG94] [Sal94] afin d'obtenir une mesure robuste. Les éléments de l'environnement qui peuvent contribuer à cette mesure sont au nombre de trois : le pixel, la primitive en construction, et le reste de l'environnement (incluant d'autres primitives voisines par exemple).

Il nous semble donc très réducteur de se satisfaire de mesures qui occultent en grande partie l'environnement, comme cela est fait lorsque seul le niveau de gris du pixel est mesuré, ou encore simplement l'écart des niveaux de gris entre le pixel et la primitive. L'intérêt d'un système de vision, tel que nous le construisons est de regrouper ensemble des processus hétérogènes, où chacun peut profiter de la connaissance apportée par les autres.

Une évaluation, par exemple, sur le couple (pixel candidat, primitive courante) se veut par ailleurs complètement modulaire, et en particulier indépendante des autres points de contrôle du système. Il n'y a aucun lien entre la manière dont les points sont sélectionnés, et celle dont ils sont évalués. Chaque niveau inclut des étapes de décision distinctes. Cette partie va permettre de développer quelques idées qui ont été implantées avec succès dans le cadre du système de segmentation coopératif que nous avons développé [BSG94], [BSG95], [BG96], [GBB98].

1.5.2 Les types d'évaluation utilisés

L'évaluation a pour but de refléter l'adéquation du pixel à la primitive courante. D'un point de vue très générique, deux types de contributions se dégagent : des mesures sur la qualité intrinsèque du pixel, et des mesures sur le lien entre le pixel et la primitive. Cette distinction peut apparaître diffuse dans de nombreux travaux, ou bien insuffisamment mise en avant, mais elle devient d'un intérêt crucial lorsque les points sont évalués de manière concurrente par rapport à plusieurs primitives. Dans les travaux de Adams et Bishof [AB94], lorsque le pixel est candidat pour la fusion à plusieurs régions voisines, cette évaluation est calculée relativement à chaque région et la meilleure valeur est conservée. La région qui incorpore le pixel est celle dont l'évaluation relative était la plus élevée.

– Les caractéristiques propres au pixel peuvent être les suivantes : information photométrique, mesure du gradient en intensité et en direction, mesures locales d'écart type...
– Les caractéristiques liant le pixel à la primitive sont multiples, nous pouvons citer : la mesure de distance entre une des caractéristiques propres précédentes et la mesure correspondante pour la primitive, des mesures géométriques de distance, de compacité, ...

Une autre dichotomie potentielle des critères d'évaluation consiste à séparer les mesures photométriques des mesures géométriques :

– Les évaluations se fondant sur les valeurs de niveaux de gris ou sur des informations dérivées sont les principales mesures effectuées lors des segmentations en régions et contours. Le calcul du gradient, et la mesure de l'homogénéité d'une région sont des informations couramment appliquées pour grouper des primitives : ainsi Liu [Liu77] utilise une méthode de croissance incrémentale de contour, dont le critère consiste simplement à rechercher parmi les points candidats, celui qui dispose de la plus forte valeur du gradient. L'étape de chaînage par hystérésis implantée dans l'algorithme de Deriche [Der87], consiste à marquer les points dont la valeur de gradient est la plus forte dans la direction du gradient. Dans des algorithmes centrés sur les régions, Gagalowicz et Monga [GM85] utilisent successivement la conservation de la moyenne des niveaux de gris, puis la conservation de l'écart-type local des niveaux de gris, comme critère de croissance de région...
– Il est intéressant d'adjoindre à ces évaluations classiques des informations d'un autre type, par exemple géométriques, afin d'apporter une connaissance locale d'une autre nature sur le contexte : Brice et Fennema [BF70] ont par exemple utilisé les propriétés de la frontière de la région en construction, dans le cadre d'un algorithme de type *Split and Merge*. Une mesure heuristique calculait la longueur de la frontière de la région en construction (*phagocyte heuristic* dans le texte original), et un des critères qui orientait la fusion de deux régions était de réduire la taille de la frontière résultante. [5]

Le problème de définir quel type d'évaluation est nécessaire reste cependant intact, et l'autre point d'intérêt est d'établir comment des décisions locales (par le biais des évaluations au niveau des pixels) peuvent contribuer à finalement générer (au niveau global) une croissance de régions satisfaisante, en termes de forme de la primitive, de précision et de localisation de la frontière, d'utilisation de ressources. L'exemple qui va suivre va apporter un éclairage concret sur ces contraintes.

Considérons une méthode de croissance incrémentale de régions utilisant le schéma générique décrit au paragraphe 1.3, dont le but est de fournir des régions correspondant à des zones de luminosité homogène de l'image. L'algorithme d'évaluation des pixels devra donc logiquement privilégier les points candidats dont l'intensité des niveaux de gris est la plus proche du niveau de gris moyen de la région courante. Définissons ainsi $P(x, y)$ le pixel candidat, ayant une intensité $I(x, y)$. La région R courante contient n pixels référencés $P(x_0, y_0), P(x_1, y_1) \dots P(x_{n-1}, y_{n-1})$, et la moyenne des niveaux de gris de R est définie par :

[5] La construction incrémentale d'une région à partir de critères uniquement photométriques a la fâcheuse tendance de produire généralement des objets aux frontières très déchiquetées. Les méthodes de type contour contribuent à résoudre ces problèmes de localisation. Techniquement, ces frontières très déchiquetées posent également des problèmes de performance aux algorithmes, car la taille des listes de pixels voisins ainsi manipulées sont généralement disproportionnées par rapport au comportement espéré par le programmeur.

$$\overline{I_R} = \frac{1}{n}\sum_{i=0}^{n-1} I(x_i, y_i)$$

L'évaluation E_1 du pixel P par rapport à la région R se définit donc par :

$$\begin{aligned} E_1(x,y) &= 1 - min(1, \frac{\left|I(x,y) - \overline{I_R}\right|}{k}) \\ & \text{où } k \text{ est une constante} \end{aligned}$$

L'évaluation E_1 est maximum pour les points dont le niveau de gris correspond exactement à celui moyen de la région, et décroît lorsque cette différence devient plus importante. Le seuil k permet de normaliser l'expression dans l'intervalle [0..1].

L'application de ce critère unique fournira certainement un résultat adéquat, mais au prix d'une forte sensibilité au bruit et aux éléments texturés de l'image. Dans de telles conditions, l'algorithme de croissance génère une région à la frontière déchiquetée, et peu exploitable en l'état. En effet, le pixel recherché est le plus homogène, il peut donc être situé n'importe où à la frontière de la région. Aucune contrainte de forme ne vient forcer la région à conserver un aspect régulier.

D'autres considérations et contraintes d'implantation peuvent suggérer de construire des régions aux formes plus régulières, sans pour autant renoncer au critère d'homogénéité initial. L'idée est d'introduire une évaluation supplémentaire, notée E_2, dont le rôle est d'évaluer le *degré de compacité* du pixel $P(x,y)$, comparé à la région R : Définissons des relations de voisinage $N_8(P(x1,y1), P(x2,y2)) = 1$ si et seulement si $\max(|x2-x1|, |y2-y1|) = 1$, ensuite l'évaluation :

$$\begin{aligned} E_2(x,y) &= \frac{1}{8}\sum_{P(x_i,y_i)\in R} N_8(P(x_i,y_i), P(x,y)) \\ &= \frac{1}{8}\,(\text{Nombre de pixels du voisinage immédiat de P}) \end{aligned}$$

Cette seconde évaluation E_2 fournira de fortes valeurs pour les pixels les plus entourés par une région.

L'évaluation globale du couple (Pixel,Région) est déterminée par une combinaison linéaire des deux évaluation élémentaires. Les pondération sont fixées en fonction de l'importance que l'on attribue à chaque critère :

$$\begin{aligned} E &= \alpha E_1 + \beta E_2 \\ & \text{avec } \alpha + \beta = 1 \\ & \text{et } \alpha, \beta \in [0..1] \end{aligned}$$

Chaque évaluation considérée séparément correspond à un critère élémentaire. E_1 est le *critère d'homogénéité* tandis que E_2 reflète ce que l'on peut qualifier de *critère de compacité*, dont le comportement seul sera de combler les petits trous et aspérités au cours de la croissance de la région.

Ce principe permet donc d'ajuster finement et *numériquement* chacun des comportements élémentaires, pour obtenir un comportement hybride, qui tire avantage conjointement de contraintes d'homogénéité et de contraintes de compacité.

1.5.3 Evaluation des pixels contour candidats

Les deux paragraphes suivants décrivent quelques critères d'évaluation. Cette liste n'entend pas être exhaustive, mais se présente davantage comme une boite à outils, indiquant quelques moyens pour contrôler et diriger une construction incrémentale de primitives de type contour et région. Ces fonctions d'évaluation sont actuellement implantées au sein d'un système coopératif gérant l'activité parallèle de processus de segmentation région et contour, travaillant sur la même image [BSG94].

La description des fonctions d'évaluation est classée de la manière suivante : mesures de propriétés photométriques, considérations géométriques centrées sur les contours, considérations géométriques impliquant des relations entre des contours et des régions.

La principale évaluation relative à un pixel contour candidat sera la norme du gradient, les autres types d'évaluation ne serviront qu'à pondérer cette évaluation principale. Considérons le pixel candidat $P(x,y)$ pour le contour C dans l'image $\mathcal{P}$. Le gradient est calculé à partir des valeurs d'intensité dans le voisinage $\overrightarrow{G(x,y)}$, sous la forme de sa norme $G(x,y) = \left\|\overrightarrow{G(x,y)}\right\|$, et de sa direction G_{d8}, échantillonnée suivant les huit directions de Freeman $G_{d8} \in \{0\ldots7\}$.

- La première évaluation repose sur une mesure locale du gradient :

$$E_1 = \frac{G(x,y)}{max_{(x_i,y_i)\in\mathcal{P}} G(x_i,y_i)}$$

  Cette évaluation privilégie les points avec les plus fortes valeurs de la norme du gradient.

- $E_2 = 1 \Leftrightarrow$ la norme du gradient est localement maximum dans la direction du gradient. $E_2 = 0$ sinon.

$$E_2 = 1 \quad \Leftrightarrow \quad \left\{ \begin{array}{ll} G(x-1,y) < G(x,y) & \text{et } G(x+1,y) < G(x,y) \\ & \text{si } G_{d8} \in \{0,4\} \\ G(x,y-1) < G(x,y) & \text{et } G(x,y+1) < G(x,y) \\ & \text{si } G_{d8} \in \{2,6\} \\ G(x-1,y+1) < G(x,y) & \text{et } G(x+1,y-1) < G(x,y) \\ & \text{si } G_{d8} \in \{1,5\} \\ G(x+1,y+1) < G(x,y) & \text{et } G(x-1,y-1) < G(x,y) \\ & \text{si } G_{d8} \in \{3,7\} \end{array} \right.$$

  Cette évaluation peut évoluer suivant plusieurs possibilités, la première consistant à vérifier plus d'un pixel de chaque côté du point courant. Définissons d_X et d_Y les déplacements élémentaires horizontaux et verticaux dans chaque direction de Freeman. Posons :

$$\begin{array}{lll} d_X(x)\,, x \in \{0\ldots7\} & \text{avec} & \left\{ \begin{array}{ll} d_X(0) = d_X(1) = d_X(7) & = 1 \\ d_X(2) = d_X(6) & = 0 \\ d_X(3) = d_X(4) = d_X(5) & = -1 \end{array} \right. \\ d_Y(x)\,, x \in \{0\ldots7\} & \text{avec} & \forall x \in \{0\ldots7\}\, d_Y(x) = d_X((x+2) \bmod 8) \end{array}$$

  La précédente évaluation est étendue à l'examen de deux pixels de part et d'autre du point courant, et se réécrit :

$$E_2 = 1 \Leftrightarrow \left\{ \begin{array}{lll} \text{Posons } d = G_{d8} \text{ la direction du gradient du pixel } (x,y) & & \\ G(x - d_X(d), y - d_Y(d)) & < & G(x,y) \\ G(x + d_X(d), y + d_Y(d)) & < & G(x,y) \\ G(x - 2d_X(d), y - 2d_Y(d)) & < & G(x,y) \\ G(x + 2d_X(d), y + 2d_Y(d)) & < & G(x,y) \end{array} \right.$$

Cette évaluation E_2 peut enfin être directement déduite de la pente du gradient entre les pixels voisins, et permettra alors de quantifier numériquement le *degré de maximalité* du gradient. E_2 passe d'une évaluation binaire à une évaluation linéaire. Posons :

$$\begin{array}{rcl} d & = & G_{d8} \\ & & \text{la direction du gradient du pixel } (x,y) \\ d_1 & = & G(x,y) - G(x + d_X(d), y + d_Y(d)) \\ & & \text{la pente du gradient d'un côté du pixel} \\ d_2 & = & G(x,y) - G(x - d_X(d), y - d_Y(d)) \\ & & \text{la pente de l'autre côté} \\ E_2 & = & \left\{ \begin{array}{ll} 0 & \text{si } d_1 < 0 \\ 0 & \text{si } d_2 < 0 \\ \frac{\min(10, d1, d2)}{10} & \text{sinon} \end{array} \right. \end{array}$$

- La littérature regorge de classifications multiples des profils de contours. Les contours dits de *type marche* [6] sont les plus répandus, et correspondent aux transitions dans l'image entre deux zones d'homogénéité distincte. Des contours particuliers séparant deux régions avec les mêmes caractéristiques photométriques sont référencés comme des contours de *type ligne* ou de *type trait* [7]. Les contours de ce type n'ont pas reçu un intérêt aussi marqué, principalement, car ils peuvent être approximés par la plupart des détecteurs de type marche, comme deux contours de type marche parallèles proches l'un de l'autre (figure 1.14).

  Utiliser un opérateur de dérivation comme le gradient est un gaspillage de ressources, et induit de plus une perte de localisation pour identifier une discontinuité parfaitement identifiable [8] par un simple examen du profil des niveaux de gris. Il est plus logique de déduire du calcul de E_2, tel qu'il a été fait pour opérer sur les gradients, une fonction identique E_3 qui travaille sur les intensités en niveaux de gris, et qui permettra de déterminer les extrema des niveaux de gris, correspondant aux profils de *type toit*. Il convient de remarquer que le triplet (E_1, E_2, E_3) implique des fonctions d'évaluation qui sont incompatibles entre elles sur un pixel donné, dans le sens où elles ne peuvent pas toutes être maximisées simultanément : effectivement pour E_2 et E_3, le gradient vaut théoriquement zéro à l'emplacement d'un contour de *type toit*. Les deux évaluations E_2 et E_3 ne peuvent donc pas cohabiter au sein de la même évaluation globale.
- Des critères géométriques sont également envisageables. Un autre type d'évaluation privilégiera les pixels candidats situés dans la prolongation de l'extrémité du contour, afin de former globalement des contours à faible courbure, et d'éviter par exemple des phénomènes d'oscillation locaux de part et d'autre d'une crête de maximum de gradient, voir la figure 1.15.

[6] Cette appellation fait référence à la forme du profil des niveaux de gris orthogonalement à la direction du contour, présentant l'allure d'un échelon, dans la définition optimale que l'on donne usuellement.

[7] Toujours en référence à la forme du profil des niveaux de gris.

[8] Voire même plus facilement identifiable.

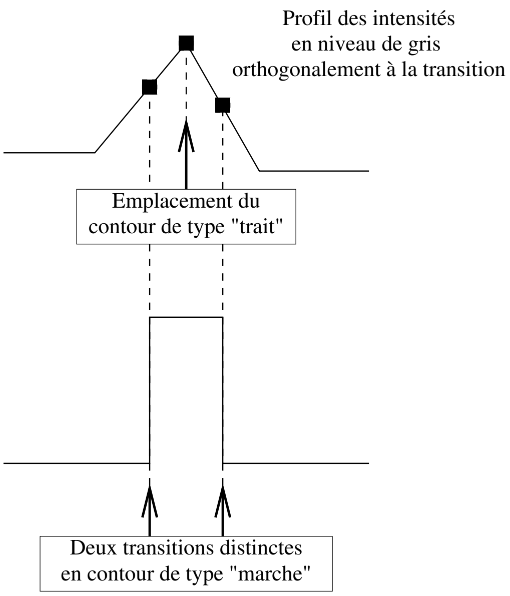


FIG. 1.14: Approximation d'un contour de type “trait” faite par un détecteur classique de type “marche”.

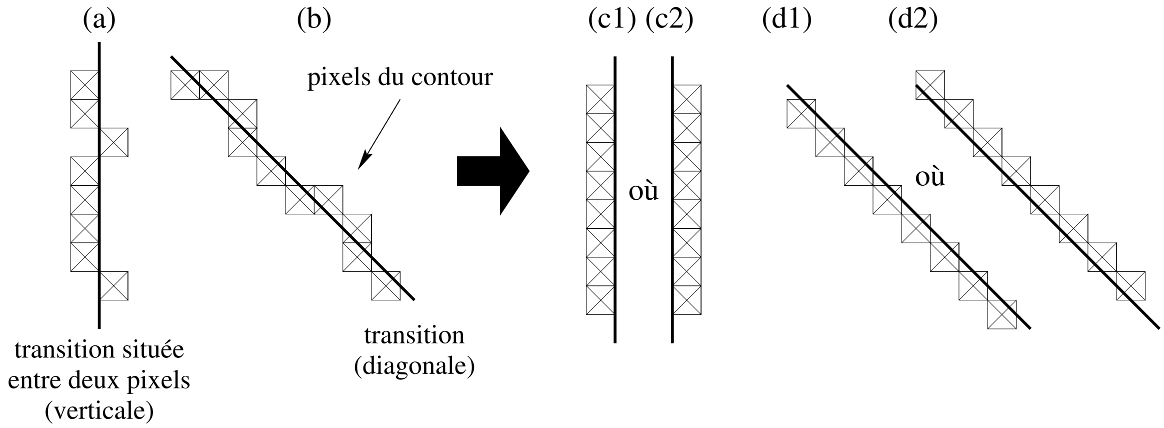


FIG. 1.15: Illustration d'un phénomène d'oscillation fréquent lorsque la transition est située entre deux pixels en (a) et (b). Une fonction d'évaluation qui privilégie les pixels situés dans le prolongement de l'extrémité permettra de résoudre ce problème en (c) et (d). Le problème de localisation n'est cependant pas résolu, car le contour obtenu peut être d'un coté ou de l'autre de la transition. Soulignons que l'objectif n'est pas ici de faire de la segmentation *sub-pixel*, mais de fournir des contours cohérents et manipulables pour les étapes de traitement ultérieures.

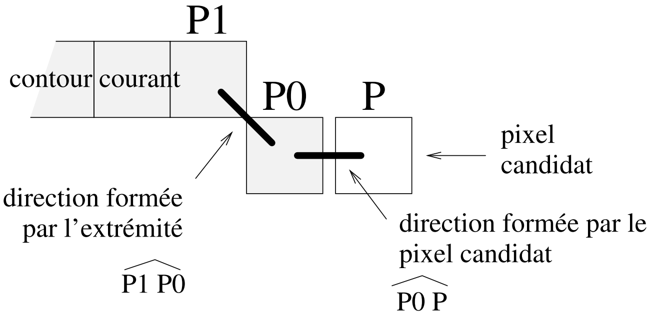


FIG. 1.16: Mesure de l'écart de direction entre l'extrémité du contour et le pixel candidat.

Considérons l'extrémité du contour, ainsi que les deux derniers pixels formant cette extrémité, P_1 et P_0, ce dernier étant extrémité libre du contour. Notons $d_{\text{ext}} = \widehat{P_1 P_0}$ la direction de Freeman formée par les pixels P_1 et P_0. Considérons le pixel candidat P, voisin du pixel P_0, sa direction de Freeman par rapport à P_0 définie par $d_{\text{candidate}} = \widehat{P_0 P}$. Calculons alors l'écart entre d_{ext} et $d_{\text{candidate}}$ (figure 1.16).

$$
\begin{aligned}
\sigma_1 &= \left| d_{\text{candidate}} - d_{\text{ext}} \right| \\
\sigma_2 &= \begin{cases} \sigma_1 & \text{si } \sigma_1 \leq \pi \\ 2\pi - \sigma_1 & \text{sinon} \end{cases} \\
\sigma &= \begin{cases} \sigma_2 & \text{si } \sigma_2 \leq \frac{\pi}{2} \\ \pi - \sigma_2 & \text{sinon} \end{cases} \\
E_4 &= 1 - \frac{2\sigma}{\pi}
\end{aligned}
$$

Ce critère, qui aura une forte valeur lorsque les valeurs des deux directions $d_{\text{candidate}}$ et d_{ext} sont proches, permet de privilégier la prolongation du contour dans la direction courante de son extrémité. Il est illustré sur l'exemple suivant, figure 1.17, montrant l'image d'un bateau de pêche, caractéristique pour les nombreux contours rectilignes de type toit [9] caractéristiques des divers cordages et mâts du navire. La sous-image C présente un résultat de détection de contour utilisant uniquement le critère d'intensité du gradient. La sous-image D présente le résultat obtenu dans les mêmes conditions initiales, en utilisant l'évaluation globale $E = \frac{E_1 + E_4}{2}$. L'image C montre clairement que le contour ne suit pas une ligne droite le long de l'objet, malgré l'aspect rectiligne de sa frontière. Ce comportement est en effet lié à la sensibilité du critère du gradient par rapport à de faibles fluctuations, car cet exemple dispose de deux lignes verticales de pixels voisins, avec des valeurs de gradients élevées et comparables. Le détecteur de contour saute donc naturellement de l'une à l'autre suivant celle qui réalise le plus fort gradient pour une ordonnée fixée (figure 1.18). L'image D fournit au contraire des contours rectilignes, qui respectent davantage les structures de l'image, et seront donc plus facilement exploitables aux plus hauts niveaux (par des détecteurs d'objets rectilignes par exemple).

– Les deux derniers points de cette description présentent des considérations géométriques mettant en relation les primitives de types contour et région. Jusqu'à présent, toutes les considérations faites ne reposaient que sur la primitive contour courante, le pixel candidat, les valeurs des niveaux de gris et les informations directement dérivées. Ce ne sont pas les seuls éléments de l'environnement de segmentation dont on dispose.

[9] que l'on segmentera ici cependant sous la forme de deux contours de type marche parallèles.

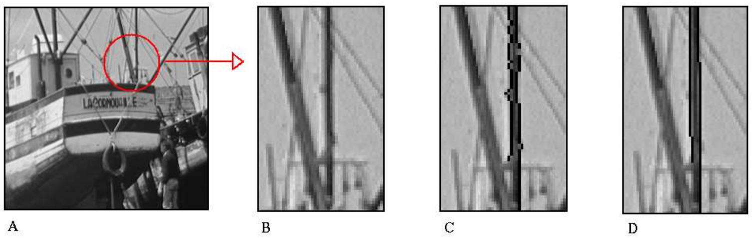


FIG. 1.17: La sous-image B indique la zone d'intérêt de l'image globale A qui a été utilisée. Dans l'image C, la croissance de contour suit classiquement les plus fortes valeurs de gradient avec le critère E_1, et présente de nombreuses irrégularités. Dans l'image D, au précédent critère d'évaluation s'ajoute le critère E_4 de préservation de direction, permettant aussi d'éviter le phénomène d'oscillation visible en C.

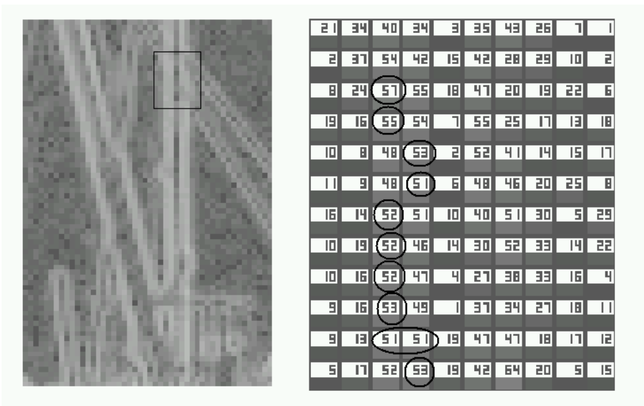
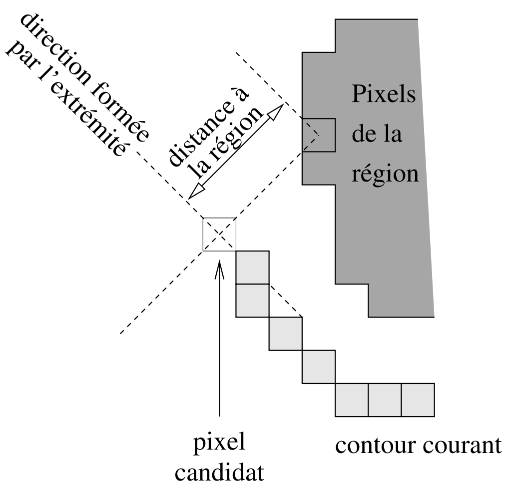
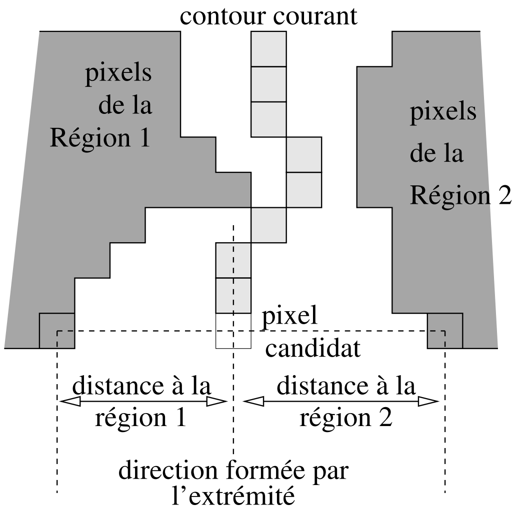


FIG. 1.18: Cette figure présente la carte du gradient correspondant à l'image 1.17B, ainsi qu'une zone sélectionnée, dans laquelle les valeurs numériques sont fournies. Les plus fortes valeurs numériques correspondent aux pixels étiquetés contour dans l'image 1.17C sont encerclées.



(a) Suivi de la frontière de la région



(b) passage au mieux entre les deux régions

FIG. 1.19: Deux exemples d'évaluations utilisant des critères géométriques impliquant des objets de type région. Les deux méthodes consistent à mesurer la distance qui sépare le pixel candidat et l'extrémité du contour à la frontière de la région la plus proche.

Elargissons le cadre de l'analyse à un système de segmentation, dans lequel cohabitent des processus de construction de contours et de régions au sein de la même image, chaque processus travaillant en parallèle avec les autres, en construisant un contour ou une région dans un endroit donné de l'image. L'objet de type région devient donc accessible comme une base d'information à part entière, sous la forme d'un ensemble de pixels étiquetés, et disposant de ses propres attributs et propriétés. Nous insistons bien sur le fait que l'image entière n'est pas segmentée complètement en régions, mais seulement certains groupes de pixels, ceux disposant par exemple de caractéristiques d'homogénéité robuste.

Le processus de construction de contour travaille donc dans un environnement entouré de structures de type région. Il est donc possible d'ajouter de nouveaux critères d'évaluation des pixels candidats, prenant en compte des interactions avec ces régions.

Nous présentons ici deux exemples : la première fonction d'évaluation E_5 a pour but de conserver une distance constante avec la frontière de la plus proche région (voir la figure 1.19a), et la seconde fonction d'évaluation E_6 permet de rester à équidistance de la frontière de deux régions environnantes (voir la figure 1.19b).

Ces critères sont typiquement une application au cas discret des techniques de navigation et d'évitement d'obstacles que pourraient utiliser des robots mobiles dans un environnement parsemé d'objets. L'implantation en est proche, puisqu'elle consiste à mesurer des distances sur des droites orthogonales à l'extrémité du contour, de la même manière qu'un robot mobile utiliserait des senseurs à ultrasons ou infrarouge pour évaluer les distances aux obstacles et corriger son cap.

Notons d_1 la distance du pixel candidat au plus proche pixel appartenant à une région, voir figure 1.20. La direction de recherche est choisie orthogonalement à l'orientation formée par le couple (pixel candidat, dernier pixel du contour) : $\widehat{P_0 P}$ sur la figure 1.16. La distance euclidienne est mesurée du centre du pixel candidat au centre du premier pixel étiqueté *région*. La même mesure d_2 est faite relativement au couple (dernier pixel du contour,

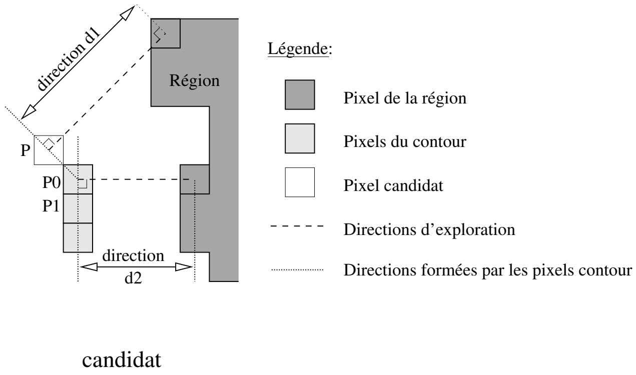


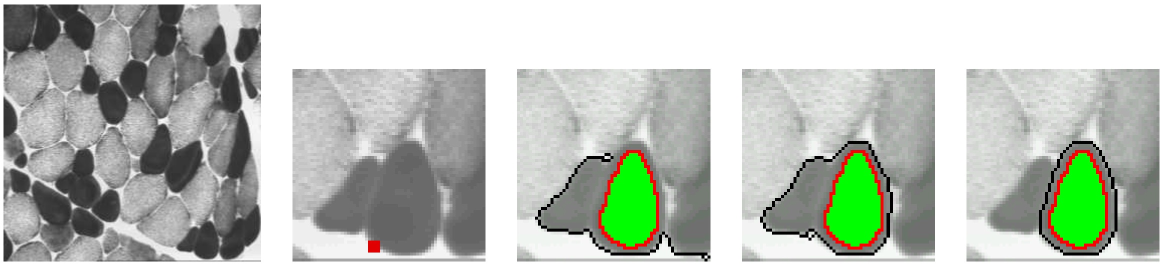
FIG. 1.20: La méthode de suivi de la plus proche région.

avant dernier pixel) : $\widehat{P_1P_0}$ sur la figure 1.16. $|d_1 - d_2|$ décrit la variation de distance à la région. La but de cette évaluation est de fournir une valeur élevée lorsque la distance est approximativement conservée, ainsi pour cet exemple : $E_5 = \max(0, 1 - \frac{|d_1-d_2|}{2})$.
L'exemple figure 1.21 illustre l'utilisation de ce critère. Cette image biomédicale montre une coupe de cellules musculaires avec des objets hautement texturés d'une part, et des objets noirs homogènes, l'ensemble étant sur un fond blanc, lui aussi homogène. Les objets de cette image présentent la caractéristique d'être très proches les uns des autres, provoquant des problèmes pour les isoler individuellement. Les régions sombres sont faciles à segmenter, et nous supposerons donc dans cet exemple qu'un processus de croissance de région a fourni séparément la segmentation de cet objet. Un processus de suivi de contour est lancé à la base des deux cellules sombres. Les pixels candidats sont évalués en utilisant une pondération de deux critères élémentaires :$E = \alpha E_1 + \beta E_5$ (plus fortes valeurs de gradient, et suivi de la frontière de la région).

- La figure 1.21c montre les résultats obtenus en utilisant $\alpha = 1$ et $\beta = 0$. Le lecteur notera que le contour ne peut pas faire le tour de la cellule sombre, car la proximité d'une autre cellule noire diminue l'intensité du gradient, comparée à la transition abrupte formée par le bas de la cellule et le fond clair de l'image. Il convient aussi de remarquer que le profil d'intensité a changé puisqu'il passe du *type marche* pour séparer la cellule du fond, à un *type toit* pour séparer deux cellules similaires.
- La contribution de l'évaluation E_5 ($\alpha = 0.6$ et $\beta = 0.4$) en figure 1.21d permet au contour de suivre avec succès la frontière de la région située à sa droite. Ceci n'est possible que grâce à la coopération implicite qui s'instaure avec la région pré-segmentée.
- Lorsque l'évaluation E_5 devient prédominante ($\alpha = 0.2$ et $\beta = 0.8$), la très faible frontière de la cellule sombre est trouvée, malgré un gradient peu élevé (entre 4 et 8), voir la figure 1.21e.

1.5.4 Evaluation des pixels région candidats

Cette partie décrit les types d'évaluation utilisés dans le cadre de la croissance de région incrémentale. A l'instar du paragraphe précédent, les critères sont classés, en commençant par les propriétés photométriques, puis en examinant les attributs géométriques de la région. Notons $P(x, y)$ le pixel candidat pour être ajouté à la région R sur l'image $\mathcal{P}$. $I(x, y)$ est l'intensité du niveau de gris du pixel $P(x, y)$. Cette mesure permet de calculer l'écart-type local $\sigma(x, y)$, obtenu sur le huit-voisinage de $P(x, y)$:



(a) image de cellules musculaires. (b) zone d'intérêt de taille 64x64. (c) (d) (e)

FIG. 1.21: Illustration de la combinaison de critères d'évaluation pour résoudre un problème local de segmentation. (a) est une image de coupe cellulaire. (b) est une sous image de taille 64×64. Une région a été pré-segmentée avant le lancement du suivi de contour dans les images (c)-(e), le point rouge (gris foncé) est le germe du suivi de contour. (c)-(e) fournit des résultats utilisant différentes pondérations des deux critères élémentaires.

$$\begin{aligned}
\overline{I(x,y)} &= \frac{1}{9}\sum_{i=-1}^{i=1}\sum_{j=-1}^{j=1} I(x+i,y+j) \\
&\quad \text{cela définit l'intensité moyenne en niveaux de gris.} \\
\sigma^2(x,y) &= \frac{1}{9}\sum_{i=-1}^{i=1}\sum_{j=-1}^{j=1} \left(I(x+i,y+j) - \overline{I(x,y)}\right)^2 \\
&\quad \text{cela définit une mesure de variance locale}
\end{aligned}$$

La structure de la région R_i est décrite sous la forme d'un ensemble de pixels connexes $R_i = \{P(x_{i,j}, y_{i,j}), j \in \{0 \dots N_i - 1\}\}$ pour lesquels, N_i étant le nombre de pixels de la région R_i :

$$\forall j, P(x_{i,j}, y_{i,j}) \in R_i \quad \exists k \text{ tel que} \quad \begin{aligned} &P(x_{i,k}, y_{i,k}) \in R_i \text{et} \\ &\|(x_{i,k}, y_{i,k}) - (x_{i,j}, y_{i,j})\|_\infty = 1 \end{aligned}$$

Définissons également les propriétés photométriques de la région R_i suivantes :

$$\begin{aligned}
\overline{I_{R_i}} &= \frac{1}{N_i}\sum_{j=0}^{j=N_i-1} I(x_{i,j}, y_{i,j}) \\
\sigma_{R_i} &= \frac{1}{N_i}\sum_{j=0}^{j=N_i-1} \sigma(x_{i,j}, y_{i,j})
\end{aligned}$$

$$E_1(x,y) = 1 - \frac{\min(10, |\sigma(x,y) - \sigma_{R_i}|)}{10}$$

Ce critère privilégie les pixels dont l'écart-type local est proche de l'écart-type moyen de la région [10].

- Cet autre critère caractérise les propriétés géométriques de la région, et mesure la position relative du pixel candidat par rapport au centre de gravité de la région. Notons $G_i(x_g, y_g)$ le centre de gravité de la région R_i et $P_i^m(x_m, y_m)$,$P_i^M(x_M, y_M)$ respectivement le pixel *voisin de la région* qui est le plus proche et le plus éloignée du centre de gravité G_i. Nous devons tout d'abord définir l'ensemble des pixels *voisins de la région* R_i :

$$
\begin{aligned}
X_i &= \{\ P(x_{i,j}, y_{i,j}), j \in \{0..M_i - 1\} \\
&\quad \text{avec} \\
&\quad P(x_{i,j}, y_{i,j}) \notin R_i \text{ et} \\
&\quad \exists P(x,y) \in R_i \text{ où } \|P(x,y) - P(x_{i,j}, y_{i,j})\|_i nfty = 1\ \} \\
x_g &= \frac{1}{N_i} \sum_{j=0}^{j=N_i-1} x_{i,j} \\
y_g &= \frac{1}{N_i} \sum_{j=0}^{j=N_i-1} y_{i,j} \\
P_i^m(x_m, y_m) \quad &\text{vérifie} \quad \begin{cases} P_i^m(x_m, y_m) \in X_i \\ \forall P(x,y) \in X_i\ ,\ \|\overrightarrow{G_iP}\| \geq \|\overrightarrow{G_iP_i^m}\| \end{cases} \\
P_i^M(x_M, y_M) \quad &\text{vérifie} \quad \begin{cases} P_i^M(x_M, y_M) \in X_i \\ \forall P(x,y) \in X_i\ ,\ \|\overrightarrow{G_iP}\| \leq \|\overrightarrow{G_iP_i^M}\| \end{cases}
\end{aligned}
$$

Notons $P(x, y)$ le pixel candidat courant, et calculons E_2 de la manière suivante. Posons tout d'abord :

$$
\begin{aligned}
d_i^M &= \|\overrightarrow{G_iP_i^M}\| \\
&\quad \text{définit la plus grande distance du centre} \\
&\quad \text{de gravité à un pixel voisin de la région.} \\
d_i^m &= \|\overrightarrow{G_iP_i^m}\| \\
&\quad \text{définit la plus petite distance du centre} \\
&\quad \text{de gravité à un pixel voisin de la région.} \\
d &= \|\overrightarrow{G_iP}\| \\
&\quad \text{définit la distance du centre de gravité au} \\
&\quad \text{pixel voisin candidat.} \\
E_2 &= 1 - \frac{|d - d_i^m|}{|d_i^M - d_i^m|}
\end{aligned}
$$

- Le dernier critère est l'évaluation du *degré de compacité* déjà abordé dans la partie 1.5.2 :

$$
\begin{aligned}
N_8(P(x1, y1), P(x2, y2)) = 1 \quad &\Leftrightarrow \quad \max(|x2 - x1|, |y2 - y1|) \leq 1 \\
E_3(x, y) \quad &= \quad \frac{1}{8} \sum_{P(x_i, y_i) \in R_i} N_8(P(x_i, y_i), P(x, y))
\end{aligned}
$$

[10] L'écart-type global de la région est défini dans notre contexte comme la moyenne des écart-types locaux des pixels de la région. L'intérêt de cette moyenne des écarts-type locaux est de pouvoir éventuellement adapter la taille du voisinage autour duquel l'écart-type local de chaque pixel est calculé.

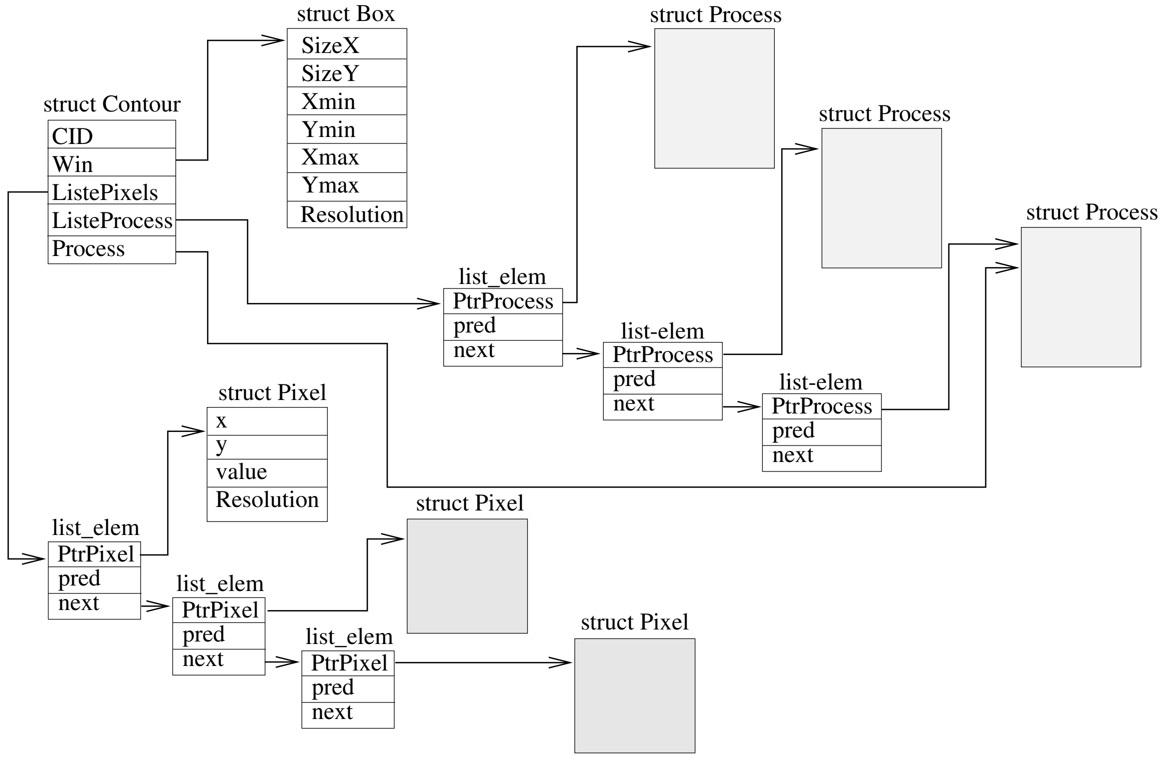


FIG. 1.22: Structure simplifiée de type contour.

1.6 L'agrégation

Cette partie va donner davantage de détails sur l'implantation du mécanisme de construction incrémental, à travers un examen plus en profondeur des structures de données, et des contrôles mis en œuvre.

1.6.1 Données stockées relatives à un contour

La structure de contour (voir figure 1.22) est définie de la manière suivante :

```
typedef struct contour
{
    int CID;
    byte GradMini;
    byte GradMaxi;
    float GradMoyen;

    PtrBox Win;
    PtrList ListePixels;
    PtrList ListeProcess;
    PtrProcess Process;
} TypeContour;
```

- L'entier défini par `CID` permet d'identifier de manière unique un contour dans le système, chaque nouveau processus de segmentation qui débute sur un nouveau pixel germe se voit attribuer une nouvelle valeur de `CID`.
- `GradMini`, `GradMaxi`, `GradMoyen`, sont les informations photométriques tenues à jour à mesure que le contour se construit. Ce sont ces mesures qui serviront à valider la primitive et à déterminer si sa construction peut continuer.

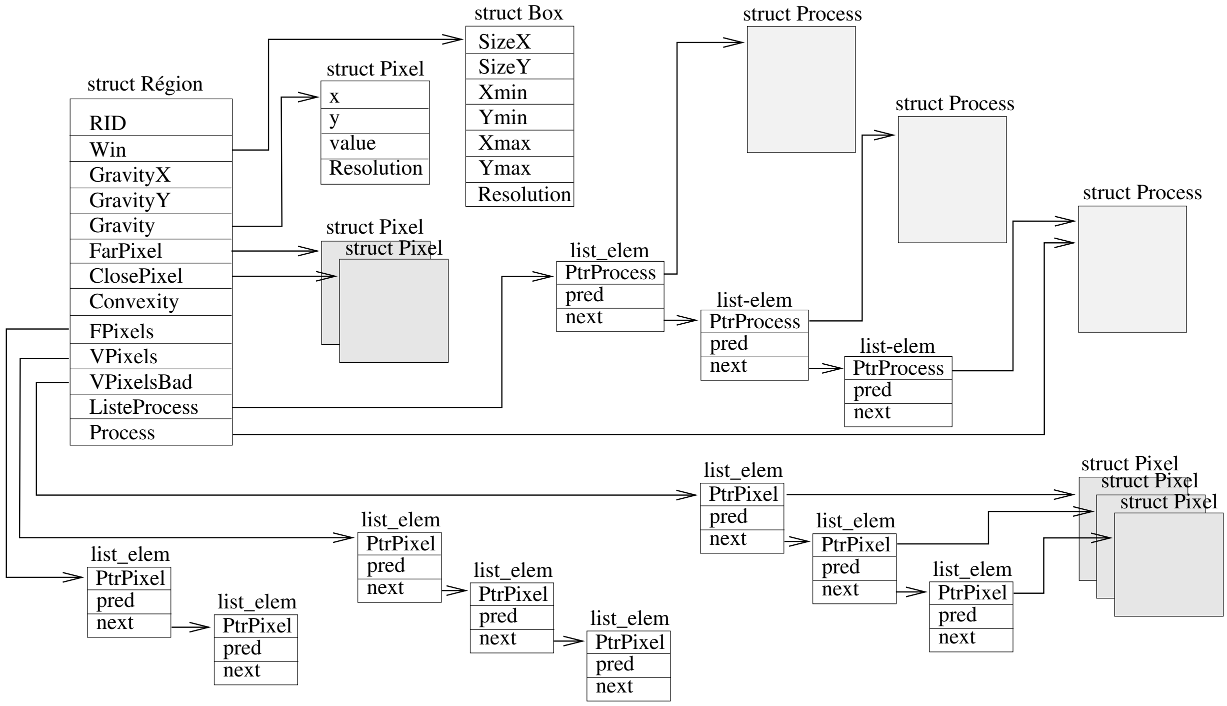


FIG. 1.23: Structure simplifiée de type région.

- `Win` définit la boite englobante du contour, utilisée à des fins d'optimisation des accès aux données.
- `ListePixels` contient la liste ordonnée (par la relation de voisinage) des pixels qui constituent ce contour. Chaque ajout d'un nouveau pixel aux extrémités se traduit par un ajout en tête ou en queue de liste.
- Chaque processus de segmentation connaît logiquement la structure de données de la primitive sur laquelle il travaille. Au cours de la croissance, il est amené à identifier d'autres primitives voisines (au moins celles qu'il a lui-même créées). Il conserve donc des pointeurs sur ces objets. `ListeProcess` est la liste symétrique, qui permet à une primitive contour ou région de référencer tous les processus qui "la connaissent". Ceci permet d'implanter une gestion propre de la destruction des primitives : la libération effective des ressources se fait lorsque aucun processus ne pointe plus sur la structure, c'est-à-dire, lorsque la liste `ListeProcess` est vide.
- `Process` est le pointeur qui référence la structure de processus qui segmente actuellement le contour.

1.6.2 Données relatives à une région

La structure de région (voir figure 1.23) est définie de la manière suivante :

```
PtrPixel     typedef struct region
{
        int             RID;
        float           EcartMini;
        float           EcartMaxi;
        byte            GrisMini;
        byte            GrisMaxi;
        float           EcartMoyen;
        float           GrisMoyen;
```

```
        float           GradMoyen;
        PtrBox          Win;
        float           GravityX;
        float           GravityY;
        PtrPixel        Gravity;
        PtrPixel        FarPixel;
        PtrPixel        ClosePixel;
        float           Convexity;
        int             *DirHisto;
        int             NbPixels;

        PtrList         FPixels;
        PtrList         VPixels;
        PtrList         VPixelsBad;

        PtrList         ListeProcess;
        PtrProcess      Process;
} TypeRegion;
```

- `RID` permet d'identifier un objet de type région de façon unique pendant l'exécution du système.
- `EcartMini`, `EcartMaxi`, `EcartMoyen`, `GrisMini`, `GrisMaxi`, `GrisMoyen` ainsi que `GradMoyen` sont des informations photométriques concernant la région, remises à jour à chaque ajout d'un nouveau pixel. Par exemple, une mesure d'écart-type local est calculée sur chacun des pixels ajoutés à la région (calculée sur un voisinage 3×3 centré sur le pixel). Les valeurs minimales, maximales et moyennes sont donc obtenues à partir de l'ensemble des points agrégés.
- `Win` définit une boite englobante de la région de manière analogue aux contours.
- `GravityX`, `GravityY` et `Gravity` tiennent à jour le *centre de gravité* de la région. Cette dernière variable est une structure de type pixel, définie par deux entiers (x, y), qui approximent les valeurs flottantes `GravityX` et `GravityY`.
- `FarPixel` et `ClosePixel` sont les pixels de la frontière respectivement le plus éloigné et le plus proche du centre de gravité. Cette proximité au centre de gravité est conservée dans la structure de région car elle constitue un critère d'agrégation à part entière. Ces pixels sont particulièrement délicats à tenir à jour à chaque ajout de nouveau point, car le centre de gravité est lui-même susceptible de se déplacer. (Voir le paragraphe 1.6.3.2).
- `Convexity` mesure un rapport de surface, qui permet de mesurer le degré de convexité de la région. L'algorithme qui mène à ce calcul est consommateur de temps CPU, et donc ne peux pas servir raisonnablement de critère d'agrégation, car cela nécessiterait son recalcul pour chaque nouveau pixel. Cette mesure est calculée à des fins statistiques. Voir le paragraphe 1.6.3.3.
- `DirHisto` est un tableau de 8 entiers, qui définit l'histogramme des directions de Freeman du gradient, calculé sur l'ensemble des pixels de la région. Des concentrations importantes de pixels sur certaines directions peuvent permettre d'identifier le caractère dégradé d'une région, et donc orienter le choix des pixels candidats pour poursuivre la construction du dégradé.
- Il n'était pas réaliste de conserver une liste de tous les pixels intérieurs d'une région, pour des raisons d'occupation mémoire, et d'accès ralenti aux données. L'essentiel est, si besoin est, de pouvoir parcourir rapidement les pixels à partir d'une autre information moins encombrante. Une matrice de pointeurs de la taille de l'image est utilisée. Chaque élément (x, y) de la matrice renvoie un pointeur sur une structure de région, si le pixel

(x, y) appartient à cette région. Une utilisation de cette matrice, en conjonction avec `Win`, permet de passer en revue rapidement tous les pixels intérieurs d'une région. La variable `NbPixels` indique le nombre de pixels de la région.

- `FPixels` définit la liste des pixels qui constituent la frontière de la région.
- Deux listes de pixels extérieurs à la région sont tenues à jour, `VPixels` et `VPixelsBad`, et contiennent les pixels candidats pour être agrégés, ou encore appelés pixels voisins. La première liste regroupe les points avec une évaluation *correcte*, et qui les rendent potentiellement agrégeables. Cette liste est triée suivant une fonction d'évaluation détaillée dans le paragraphe 1.5.4. Le seconde rassemble en vrac tous les autres pixels candidats, avec une *mauvaise* évaluation.
- De manière analogue au cas des contours, des références croisées sont maintenues, afin de savoir quels sont les processus qui ont connaissance de cette région dans leurs structures de données. Ces processus sont référencés dans la liste `ListeProcess`. `Process` est l'unique processus en charge de segmenter ladite région.

1.6.3 Remise à jour des données sensibles

Nous allons donner dans cette partie quelques points algorithmiques, sur la remise à jour de données sensibles pour les primitives de type région et contour. Ces remises à jour sont considérées comme "sensibles", car elles sont effectuées à chaque ajout de pixel, et doivent donc être optimisées afin de ne pas vampiriser les performances du système.

1.6.3.1 Le centre de gravité

Les coordonnées du centre de gravité sont réactualisées de manière incrémentale à chaque nouveau point ajouté. Pour ne pas perdre en précision, on sauvegarde la somme des abscisses et des ordonnées, sous une forme entière, plutôt que la valeur décimale rapportée au nombre de pixels de la région.

```
Region->SumX+=GetPixelX (NouveauPixel);
Region->SumY+=GetPixelY (NouveauPixel);
Region->NbPixels++;
Region->GravityX=(float)Region->SumX/(float)Region->NbPixels;
Region->GravityY=(float)Region->SumY/(float)Region->NbPixels;
DeletePixel (Region->Gravity);
Region->Gravity=CreateSimplePixel (Region->GravityX,Region->GravityY);
```

La dernière instruction permet de convertir les coordonnées du centre de gravité en valeurs entières. Nous allons voir dans l'étape suivante comment utiliser ces valeurs.

1.6.3.2 Les pixels frontières extrema

La difficulté pour remettre à jour les deux pixels de la frontière, respectivement le plus proche, et le plus éloigné du centre de gravité de la région, provient du fait que le centre de gravité se déplace avec la croissance de la région. La remise à jour optimale consistant à parcourir systématiquement tous les pixels frontières est exclue pour des raisons de complexité algorithmique. Il est cependant possible d'obtenir un compromis dans les traitements en séparant le cas où le pixel qui représente le centre de gravité s'est déplacé ou pas. Cela consiste à verifier si l'arrondi entier des coordonnées flottantes a varié.

Cas 1. Les coordonnées entières du centre de gravité n'ont pas changé par rapport à l'ajout du pixel précédent :

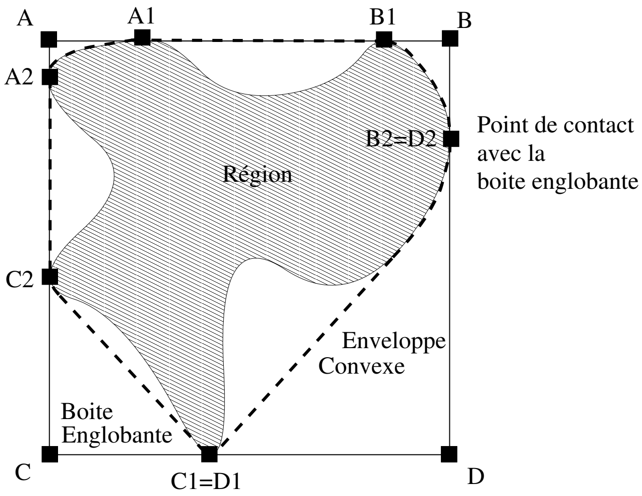


FIG. 1.24: Définition de l'enveloppe convexe d'une région, en pointillé. La boite englobante (A,B,C,D), ainsi que les points de contacts entre la région et ladite boîte servent à la construction de cette enveloppe.

– Le pixel le plus proche du centre de gravité ne change pas, tant que celui-ci demeure un pixel voisin. S'il n'est plus un pixel du voisinage, on parcourt la liste des points voisins à la recherche du nouveau plus proche pixel.

– Le pixel voisin le plus éloigné du centre de gravité peut éventuellement être remplacé par les nouveaux pixels voisins issus du nouveau point ajouté (une comparaison des distances suffit alors).

Cas 2. Le pixel centre de gravité s'est déplacé. Il faut parcourir la liste des pixels frontière à la recherche des deux nouveaux pixels extrema.

On notera que le centre de gravité devient de plus en plus stable à mesure que la région grossit. Les traitements faits dans le cas 1 deviennent donc prépondérants par rapport au cas 2.

1.6.3.3 La mesure de convexité

L'algorithme proposé pour mesurer la convexité calcule le rapport de surface entre la région courante, et l'enveloppe convexe de la région comme cela est illustré dans la figure 1.24. Le calcul de cette mesure répond à un besoin concret. La prise en compte de l'évolution de cette valeur au cours de la croissance de la région, et plus précisément sa maximisation permettrait de générer une croissance plus compacte, et donc économiserait les ressources mises en œuvre pendant sa construction.

La notion de *région convexe englobante* d'une région a été introduite dans le cas discret par Kim et Rosenfeld [KR80], [Kim82], et dans le cas analogique pour une forme quelconque fermée par Freeman et Shapira [FS75]. L'ouvrage de Chassery et Montanvert [CM91] présente un état de l'art précis sur ce domaine. Nous utiliserons une variante de l'algorithme de Freeman pour calculer cette enveloppe convexe, qui s'appuie pareillement sur les points de contact entre la région et sa boîte englobante, voir la figure 1.24, et 1.25.

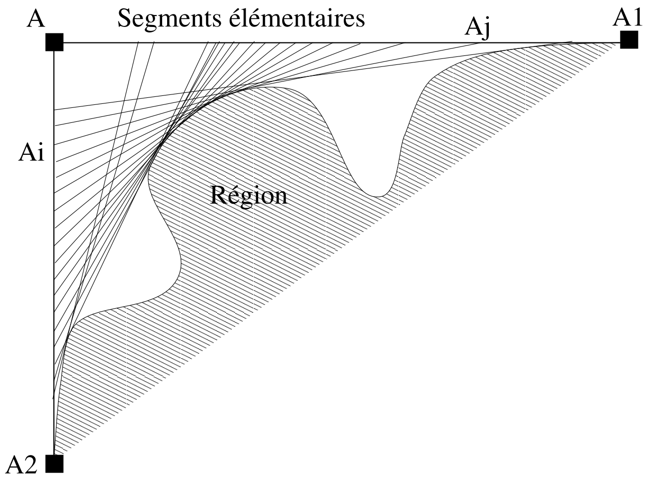


FIG. 1.25: La construction des segments élémentaires A_iA_j, avec A_i parcourant le segment AA_2 et A_j parcourant le segment AA_1.

1.7 La validation de la primitive

Les précédents paragraphes ont montré comment les pixels candidats sont évalués, et comment le meilleur pixel est ajouté à la primitive courante. Ceci définit la boucle élémentaire de la construction de la primitive. Une condition d'arrêt doit permettre de sortir de cette boucle, et le cas échéant de valider ce qui vient d'être construit. Ce traitement constitue l'étape de *validation de la primitive*, et est effectuée à chaque nouveau pixel ajouté. Plusieurs mesures sont effectuées sur la primitive et sur le dernier pixel ajouté afin de s'assurer de leur qualité. Ces traitements sont réalisés en dehors du mécanisme d'évaluation de la primitive pour plusieurs raisons :

- Ces mesures sont assimilables à des interruptions, signifiant que la construction de la primitive ne doit pas se poursuivre, et qu'un problème vient d'être détecté. Une baisse de la valeur du gradient sur les N derniers pixels contours ajoutés, un gradient dont la norme est trop irrégulière pour les pixels d'un contour venant d'être construit : toutes ces mesures sortent du cadre de la simple comparaison entre le pixel et la primitive, comparaison qui guide notre démarche d'évaluation dans les précédents paragraphes. Il est donc logique d'isoler ces mesures de contrôle.
- Les mesures effectuées dans le cadre des fonctions d'évaluation précédentes apportent une indication *relative* entre le pixel et la primitive. Les mesures de validation que nous suggérons sont des mesures *absolues.*

1.7.1 Validation de la primitive contour

Chacun de ces tests est effectué séparément, sans préjuger du résultat des autres tests. Les résultats sont placés dans une variable de type "champ de bits". Ceci permet à terme de définir des conditions complexes portant sur une combinaison de plusieurs variables.

1.7.1.1 Fluctuation de la direction du gradient

Le premier test mesure la plus ou moins grande variation de la direction du gradient le long des pixels du contour. Ce test a pour but de discriminer les contours bien formés des faux contours trouvés à tort dans les textures. Dans les deux cas, la norme du gradient est

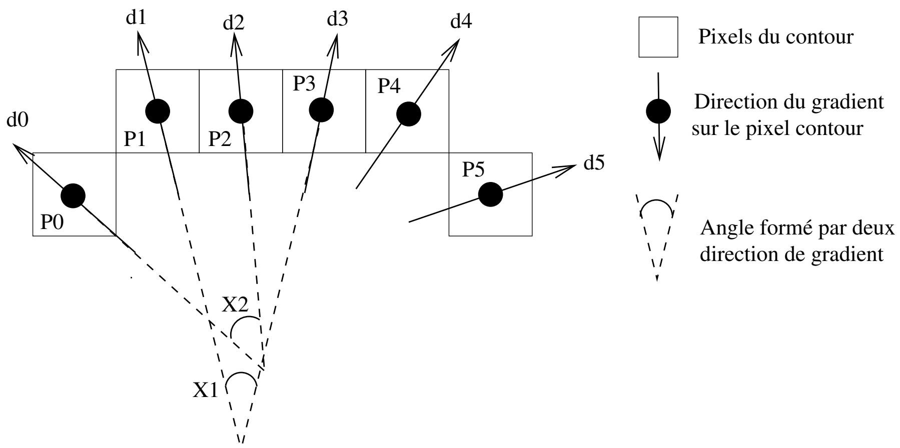


FIG. 1.26: Mesure de l'écart entre les directions du gradient. Cette mesure d'écart, afin d'être moins sensible au bruit, n'est pas réalisée sur des pixels consécutifs.

significative. Un contour bien formé doit avoir un gradient qui ne varie pas trop en direction, alors qu'un contour trouvé dans une zone texturée aura une direction de gradient très fluctuante.

La figure 1.26 illustre ce calcul. Les direction du gradient d_i sont obtenues pour les N derniers pixels du contour. $N = 20$ dans nos expérimentations. Les écarts de direction $X_1, X_2, \ldots$ sont calculés entre les pixels P_i et $P_i + 2$, pour i variant entre 0 et N. La valeur moyenne $\overline{X}$ des X_i est calculée. Un drapeau `C_DIRECTION` est activé si $\overline{X} \geq \frac{\pi}{4}$. Ceci correspond à une variation d'angle de $\frac{\pi}{8}$ entre deux pixels consécutifs, ce qui correspondrait, dans une image de synthèse non bruitée , à un cercle discrétisé de 16 pixels, soit de rayon $2,5$.

1.7.1.2 Fluctuation de la norme du gradient

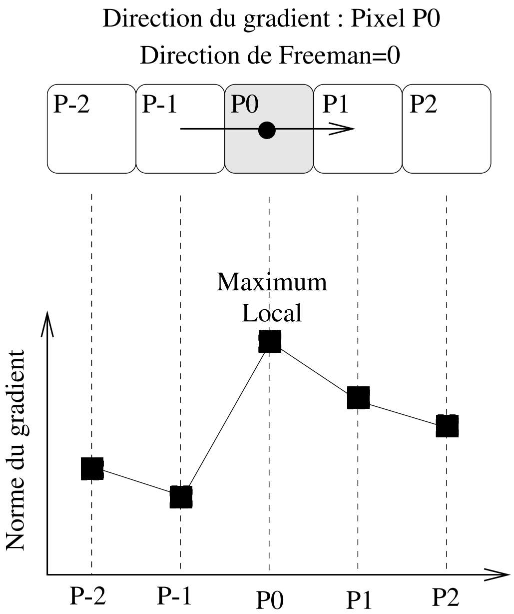
Le deuxième test valide de manière analogue la fluctuation de la norme du gradient le long du contour. Un contour doit présenter une norme régulière du gradient pour chaque point qui le compose. Ceci est particulièrement important pour les contour dont le gradient est faible. La moyenne de la norme du gradient est calculée sur les N derniers pixels du contour à une extrémité donnée, et est notée $\overline{G}$. La variation de norme du gradient entre deux pixels consécutifs est calculée, et notée $\sigma_i = |G_i - G_{i+1}|$. La moyenne de ces variations $\overline{\sigma}$ est comparée à un seuil, dépendant linéairement du gradient moyen. Un drapeau `C_IRREGULAR_GRD` est activé si $\overline{G} < 20$ et $\overline{\sigma} > \alpha.\overline{G}$. Nous utilisons expérimentalement la valeur $\alpha = \frac{1}{5}$.

1.7.1.3 Baisse de la norme du gradient

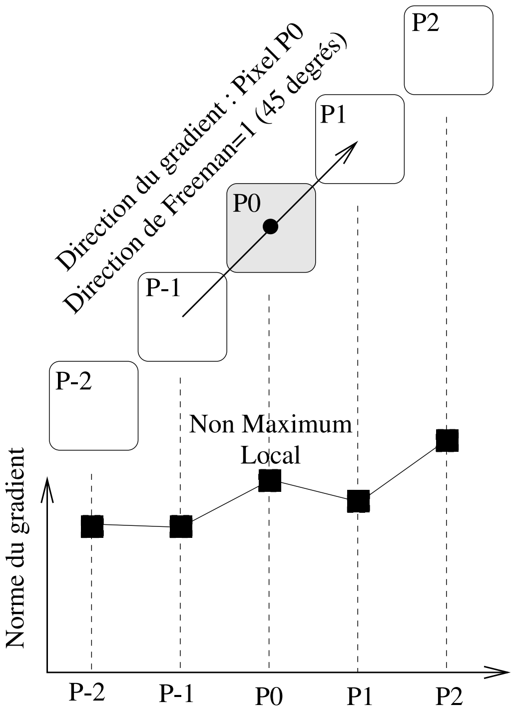
Le troisième test vérifie si le gradient du dernier pixel est significativement faible, comparé aux valeurs sur les précédents points agrégés. Le gradient du pixel extrémité est défini par G_e, et le gradient moyen des N derniers pixels ajoutés à une même extrémité (nous choisissons $N = 6$) est noté $\overline{G}$, extrémité non incluse. Un drapeau `C_LOW_GRAD` est activé si $G_e < 20$ et $G_e < \beta.\overline{G}$. Nous utilisons expérimentalement la valeur $\beta = 0,6$.

1.7.1.4 Proportion de maxima locaux du gradient

Le quatrième test vérifie si le contour présente suffisamment de pixels ayant un gradient qui réalise le maximum local dans la direction du gradient. La figure 1.27 indique comment est vérifiée la notion de "gradient maximum local dans la direction du gradient". La direction du gradient est échantillonnée suivant les huit directions de Freeman, et deux pixels de part et



(a) Le pixel $P-0$ réalise le maximum local du gradient.



(b) Le pixel P_0 ne réalise pas le maximum local du gradient.

FIG. 1.27: Vérification de la condition de maximalité de la norme du gradient dans la direction du gradient sur le pixel P_0.

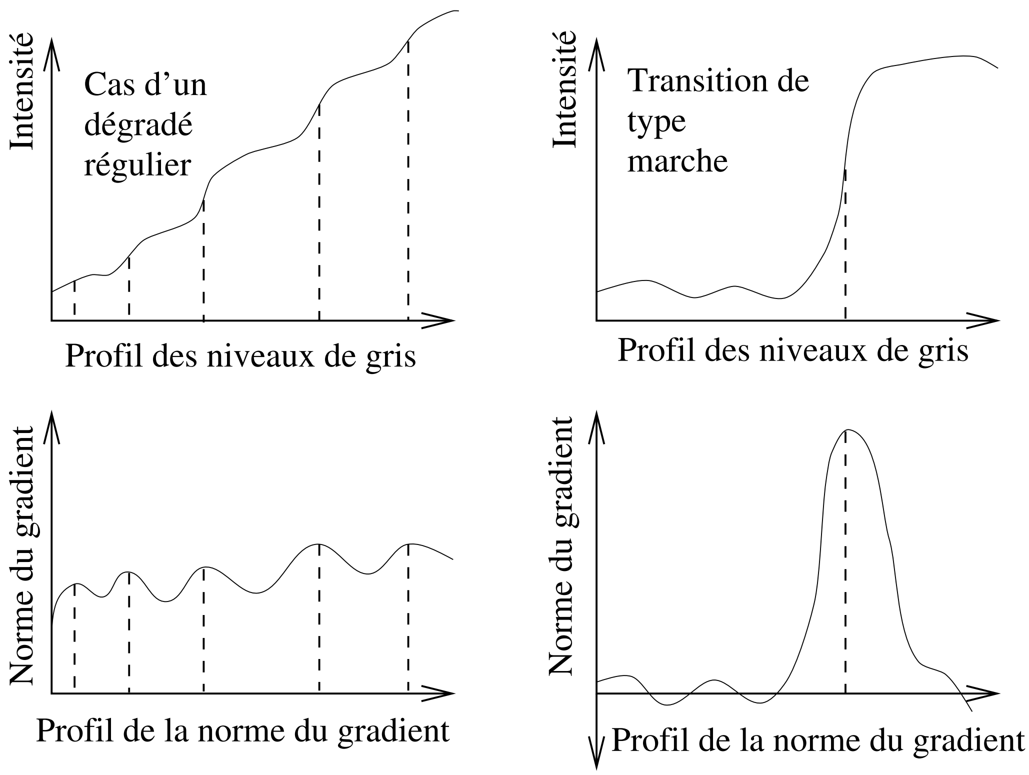


FIG. 1.28: Exemple de profils de gradient en zone dégradée et sur une transition de type marche.

d'autre de l'extrémité sont testés. La figure 1.27a donne l'exemple d'une direction de gradient horizontale pour laquelle la norme du gradient sur les pixels P_{-2}, P_{-1}, P_1 et P_2 est inférieure à la norme du gradient sur le pixel P_0. La figure 1.27b donne l'exemple d'une direction de gradient diagonale, pour laquelle le pixel P_0 ne réalise pas le maximum local du gradient.

Ce test permet de discriminer les contours bien formés avec une forte proportion de maxima locaux, des autres contours pouvant par exemple être trouvés dans un dégradé, présentant un gradient significatif, mais ne réalisant pas le maximum local. La figure 1.28 montre un profil typique d'une zone dégradée et celui d'une transition de type marche. Le dégradé génère un gradient de norme significative, pratiquement constant, mais dont les légères fluctuations (exagérées sur cet exemple) peuvent tromper le détecteur de contour qui suit les pixels ayant “la plus forte norme du gradient”. L'espacement très faible entre ces faux maxima locaux (en pointillés sur l'exemple), ne permettent pas de les valider comme “maxima locaux du gradient” : on aboutit à la configuration de l'exemple 1.27b, dans lequel le pixel P_2 invalide la maximalité.

La statistique est effectuée sur les N derniers pixels agrégés à une même extrémité, avec $N = 20$ dans notre système. Notons $\overline{G}$ la moyenne de la norme du gradient sur les N pixels étudiés, et N_{max} le nombre de pixels réalisant le maximum local du gradient. Nous définissons deux seuils pour tester cette proportion. Un drapeau `C_FEW_MAX_LOC` est activé si $\overline{G} < 10$ et $N_{max} < \gamma_1.N$ ou bien lorsque $10 \leq \overline{G} < 20$ et $N_{max} < \gamma_2.N$. Nous utilisons expérimentalement les valeurs $\gamma_1 = 0,3$ et $\gamma_2 = 0,5$.

1.7.1.5 Visibilité

Ce test reprend les travaux de thèse de Marc Salotti [Sal94] sur la définition d'un critère de visibilité d'un contour, en fonction de sa longueur et de son gradient moyen. Le problème de visibilité est considéré pour les petits contours (moins de 8 pixels). Salotti a montré que plus un contour était petit, plus son gradient moyen devait être important afin de rester visible à l'œil humain. Notons L la longueur du contour, et $\overline{G}$ son gradient moyen [11]. Un drapeau `C_VISIBILITY` est activé lorsque $L \leq 3$ (un contour de moins de trois pixels n'est pas jugé significatif), ou lorsque $3 < L \leq 6$ et $\overline{G} < 20$, ou lorsque $6 < L \leq 8$ et $\overline{G} < 10$.

1.7.1.6 Faible fonction d'évaluation

Ce dernier test relatif à une primitive contour teste la valeur de la fonction d'évaluation du dernier pixel agrégé à la primitive par rapport à un seuil. Ce seuil a été défini pour ce processus de segmentation au moment de sa création, et peut fluctuer en fonction de l'environnement dans lequel évolue ce processus. Cette adaptation est détaillée dans le paragraphe 2.4.3.2. Un drapeau `C_LOW_EVAL` est activé si l'évaluation du pixel agrégé est inférieure au seuil σ du processus.

1.7.1.7 Divers

Un drapeau `C_SHORT_CUT` est activé lorsque le contour boucle sur lui-même, et n'a donc plus de possibilité pour se développer.

1.7.2 Validation de la primitive région

Chacun de ces tests est effectué séparément, sans préjuger du résultat des autres tests. Les résultats sont placés dans une variable de type "champ de bits".

1.7.2.1 Homogénéité de la région

Ce test permet de s'assurer que la région conserve une certaine homogénéité. Il consiste à mesurer la différence entre le pixel ayant le plus fort et le plus faible écart-type local des intensités des niveaux de gris. L'écart-type local sur un pixel $P(x, y)$ est mesuré sur un petit voisinage de taille $n \times n$, dans notre cas, $n = 5$, centré sur le pixel $P(x, y)$. L'intensité moyenne est calculée sur ce petit voisinage, $\overline{I} = \frac{1}{25} \cdot \sum_{i=-2}^{i=2} \sum_{j=-2}^{j=2} I(x+i, y+j)$. Puis la variance est calculée de façon similaire, $\sigma^2 = \frac{1}{25} \cdot \sum_{i=-2}^{i=2} \sum_{j=-2}^{j=2} (I(x+i, y+j) - \overline{I})^2$.

La consistance d'une région ne se limite pas seulement à regrouper les pixels avec une faible valeur de σ, auquel cas on éliminerait d'office les zones texturées ou bruitées, quand bien même cette texture, ou plus généralement ce désordre des intensités présenterait une certaine régularité, qui en ferait une région à part entière. Afin d'englober à la fois les zones homogènes et les zones texturées dans notre critère de validation, nous proposons plutôt que de seuiller σ, de valider l'amplitude de variation de σ sur tous les pixels composant la région.

Notons σ_{min} et σ_{max} les valeurs extrémales de σ pour tous les pixels de la région. Un drapeau `R_NO_HOMOGEN` est activé lorsque $\sigma_{max} - \sigma_{min} > \zeta$. Nous utilisons expérimentalement la valeur $\zeta = 10$.

1.7.2.2 Faible fonction d'évaluation

Ce dernier test présente les mêmes caractéristiques que le test relatif aux contours. Un drapeau `R_LOW_EVAL` est activé si l'évaluation du pixel agrégé est inférieure au seuil η du processus.

[11] la moyenne de la norme du gradient des pixels qui le constitue.

1.8 Conclusion

Ce chapitre a introduit l'approche incrémentale qui sert de fondement à notre système de segmentation, ainsi que les points de contrôle qu'il nous parait important de mettre en exergue, afin de pouvoir développer des méthodes de segmentation qui répondent le plus fidèlement possible aux besoins de l'utilisateur.

Ces nombreux points de contrôle ne vont pas manquer de poser des problèmes d'ajustement et de paramétrisation. Le chapitre 4 apportera une contribution sur les moyens qui permettront à un utilisateur non expérimenté de les gérer sans être noyé sous un flot de valeurs à configurer [12]. La multiplicité des points de contrôle posera également des problèmes lors de l'étape de validation, en particulier pour la comparaison avec d'autres méthodes disposant de peu de paramètres à ajuster et pour lesquelles il est possible de faire une recherche exhaustive du jeu de "paramètres optimaux".

Le chapitre suivant s'efforcera enfin de détailler la manière de faire fonctionner ces méthodes incrémentales, de façon simultanée, et au sein d'un même environnement de travail, en utilisant des principes de séquencement proches de ceux des systèmes d'exploitation.

[12]Des valeurs par défaut fourniront un comportement satisfaisant dans la plupart des cas. Ces valeurs par défaut sont obtenues empiriquement, par des méthodes de type essai-erreur. Des images atypiques, ne satisfaisant pas aux paramètres usuels, pourront cependant être manipulées sur intervention explicite de l'utilisateur.

Chapitre 2

Un système coopératif

2.1 Introduction

Le chapitre précédent s'est attaché à décrire des mécanismes incrémentaux permettant la construction de primitives image de type contour et région. Cette construction incrémentale permet de prendre en compte les configurations locales de l'environnement, tant en termes photométriques qu'en termes de primitives préexistantes dans le système. L'incrémentalité dans un système de vision apporte donc une richesse de fonctionnement à travers une évolution temporelle, des possibilités de prises de décision dynamiques, et plus globalement une adaptabilité des méthodes.

L'étape suivante qui vient naturellement à l'esprit est de disposer non plus d'une unique méthode de construction incrémentale, mais de tout un ensemble, éparpillés dans diverses zones de l'image. Un tel choix repose sur la constatation simple qu'une image est constituée de plusieurs primitives, tant de type région que contour, et que chacune de ces primitives peut prétendre être segmentée par une *instanciation* [1] d'une méthode incrémentale *générique*. Une coopération efficace peut alors résulter d'un tel choix. La coopération, selon l'approche proposée, consiste à ne pas donner un rôle prépondérant à une méthode par rapport aux autres, mais au contraire, à utiliser de manière opportuniste [2] leurs compétences. Une telle coopération symétrique entre des méthodes présente des intérêts importants :

- une meilleure adaptation individuelle de chaque méthode incrémentale (en fonction de ses conditions d'initialisation, et de ses interactions avec d'autres méthodes) peut offrir de meilleures performances locales.
- une méthode individuelle devient située dans le temps et l'espace par rapport aux autres méthodes. Des dépendances temporelles entre les méthodes peuvent s'instaurer.
- le lien qui unit la primitive à segmenter de l'*instance* qui a la charge de sa segmentation n'est pas unique dans le temps. Une même primitive image pourra tenter d'être segmentée par une autre méthode ultérieurement [3], si la méthode courante n'y parvient pas.
- une interaction opportuniste de méthodes permet un enrichissement de l'environnement de termes de primitives segmentées [4].

[1] au sens des langages orientés objets

[2] dans le contexte, cela signifie qu'une méthode fera appel à une autre méthode dans certaines circonstances clairement établies, pour répondre à un besoin, et pas de manière désintéressée. Le besoin sera l'accumulation d'information locale pertinente pour une prise de décision plus robuste.

[3] avec d'autres paramétrages.

[4] Cet effet de bord demeure le but ultime de notre système, car de cette coopération de méthodes de segmentation, l'utilisateur obtient au terme de l'exécution un ensemble de primitives régions ou contours. Ces primitives

A l'aspect méthode de segmentation décrit dans le chapitre précédent, vient donc s'ajouter un niveau supérieur, où chaque méthode de segmentation est individualisée (et répondra par la suite au terme de *processus* de segmentation), et où cet ensemble de méthodes constitue le corps actif du système qui est présenté.

Le présent chapitre s'articule autour des thèmes suivants. Après une brève présentation des méthodes coopératives existantes dans le domaine de la vision, la notion de système d'exploitation sera introduite comme une possibilité pour l'implantation des méthodes de segmentation incrémentales décrites au chapitre précédent. Les points suivants expliciteront davantage l'ancrage des méthodes de segmentation dans un système multi-processus : les moyens pour initialiser les processus de segmentation, l'utilisation des autres méthodes du point de vue du processus de type contour, puis de type région. La négociation de territoires constituera la dernière partie de ce chapitre.

2.2 Coopération de méthodes en vision

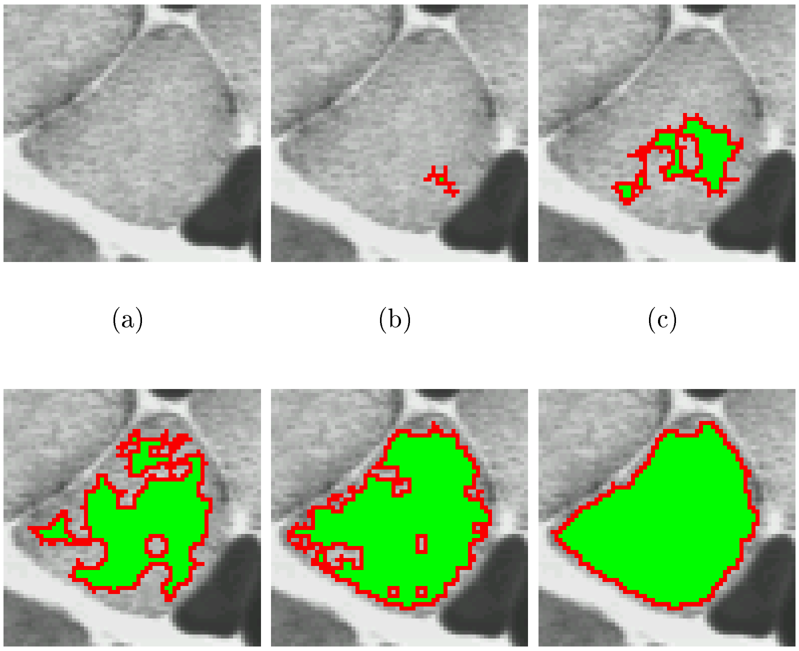
Les approches coopératives en vision par ordinateur peuvent se scinder en deux catégories : les approches *orientées modèle*, et les approches *orientées contrôle*. Les premières s'appuient sur une conceptualisation théorique des éléments recherchés dans l'image. La définition d'un modèle théorique du contour, d'une région, d'un dégradé permettent d'en dégager les caractéristiques *a priori*, les relations, les limites, le domaine où le modèle sera performant, la complexité algorithmique, la mesure de qualité et de performance *a posteriori*. Le deuxième type d'approches repose davantage sur la manière d'intégrer deux méthodes, en étudiant leurs dépendances, quelles données échanger, à quel moment, ou comment intégrer les résultats d'une méthode dans l'algorithme de calcul de l'autre méthode.

Parmi les méthodes à base de modèle, citons la méthode proposée par Haddon et Boyce [HB90], qui transposent le problème de la segmentation en région et en contour en un problème de partitionnement de la matrice de cooccurence, les pics situés sur la diagonale correspondant aux régions, et les autres pics de l'image se rapportant aux frontières. Une étape postérieure permet un raffinement du résultat en adaptant la cohérence locale dans une approche utilisant la relaxation. Cette méthode effectue une classification des pixels à la différence de la nôtre, ce qui signifie qu'elle n'autorise pas de laisser des pixels non étiquetés à l'issue de la première étape, quand bien même les pixels intégrés à une région sont sujets à caution. Nous suggérons au contraire de n'étiqueter que des pixels robustes, et sûrs, et de laisser le cas échéant des pixels non étiquetés. De plus, leur première étape de classification est suivie d'un traitement de raffinement visant à restituer la consistance locale du voisinage des pixels, ce qui laisse à penser que cette première étape est trop globale, puisqu'il convient de rectifier certaines de ses décisions qui ont été prises hâtivement.

Les méthodes coopératives fondées sur le contrôle constituent la majorité des techniques proposées. L'idée de base qui sous-tend ces coopérations de méthodes est que :

- Chaque méthode seule présente des défauts intrinsèques. Les détecteurs de contours travaillent trop localement, et ne permettent pas de construire des objets très structurés entre eux. Les détecteurs de contours sont sensibles au bruit de l'image. Les méthodes de croissance de région utilisent des critères souvent trop globaux, et les frontières délimitant ces objets ne sont généralement pas significatives. La figure 2.1 présente un petit exemple de construction de région incrémentale par ajout de pixels guidé par un unique critère

sont les seuls éléments qui subsistent à la fin de l'exécution, et sur lesquels peuvent porter les traitements ultérieurs. La coopération de méthodes n'est donc qu'un moyen pour arriver à ce but.



(a) (b) (c)

(d) (e) (f)

FIG. 2.1: Exemple de croissance de région présentant une frontière déchiquetée difficilement exploitable au cours de sa construction. Lorsque la construction arrive à son terme, la frontière de la région épouse plus régulièrement la frontière de la zone à segmenter.

d'homogénéité. Il est facile de remarquer que la frontière de cette région [5] au cours de sa construction est particulièrement déchiquetée, même si elle tend à s'améliorer lorsque la région épouse avec plus de précision les limites réelles de l'objet à segmenter dans la dernière image. L'utilisation de cette frontière est donc rendue délicate par son manque de localisation. La construction incrémentale d'une région est de plus un processus lent comparé à la construction de contours.

– Les atouts de chaque méthode peuvent donc permettre de corriger leur défauts.
– Plus précisément, la détection des contours peut aider à contrôler l'expansion ou la fusion des région, tandis que la détection des région contribue à la localisation et à la fermeture des contours ([ABM87], [BPG91], [BB92], [Bon91], [CA93], [FBC94], [MC92], [NL84], [PL90], [WM87], [XYJH+92]).

Cependant peu de travaux exploitent totalement la dualité entre ces deux types de primitives. Ainsi une première sous-classe de méthodes va consister à assister une technique de construction de régions par des informations de type contour. La région est créée par une méthode de type *Split And Merge*, ou bien de type *incrémental* pixel par pixel. Par exemple, Wrobel et Monga [WM87] proposent d'appliquer d'abord un détecteur de contours, puis de contraindre une méthode de fusion des régions en prenant en compte la présence des contours à leurs frontières. Gagalowicz et Monga [GM85] utilisent une approche de type *Split and Merge* pour la construction des régions. Plusieurs critères sont utilisés pour décider de la fusion de couples de régions adjacentes. Parmi ceux-ci, le quatrième critère mesure le gradient moyen à

[5] La frontière de la région fait ici référence aux pixels de la primitive présentant au moins au pixel voisin en dehors de la primitive. Elle ne doit donc pas être confondue, ni assimilée à la frontière qui sépare visuellement deux objets dans l'image des niveaux de gris, cette dernière étant une caractéristique statique de l'image.

la frontière entre les deux régions considérées. La fusion est effectuée si ce critère est inférieur à un seuil donné. Benois et Barba [BB92] utilisent une carte précalculée des contours robustes de l'image pour contraindre une méthode de segmentation en régions de type *Split and Merge*. L'algorithme s'appuie sur une structuration hiérarchique pyramidale de l'image de type *quad-tree* [6]. Le choix de l'ordre de fusion des régions est résolu à l'aide de la carte des contours, car la fusion privilégie les zones du *quad-tree* [7] les plus éloignées de tout pixel contour. Ces zones sont supposées par conséquent les plus homogènes. Kara Falah et *al.* [FBC94] proposent une coopération région-région, qui permet dans une première étape de générer des "noyaux" de régions robustes par superposition de plusieurs cartes de régions différentes. Ces noyaux sont supposés robustes puisqu'ils correspondent à des région obtenues par plusieurs détecteurs, dans différentes conditions expérimentales. L'expansion de ces noyaux de régions se fait ensuite sous le contrôle de l'information fournie par le gradient, selon une coopération région-contour. Dans le même ordre d'idées, Xiaohan et *al.* [XYJH$^+$92] intègrent directement l'information sur la maximalité du gradient comme un critère de sélection des pixels dans une croissance de région classique.

Une deuxième sous-classe de méthodes repose sur la dualité entre les informations de type région et de type contour. Pavlidis [PL90] propose d'appliquer d'abord une méthode de croissance de région, permettant d'extraire des contours fermés à partir des régions formées. La position de ces contours est ensuite affinée par des techniques de déformation de contours (*snakes*). Cette approche résout ainsi une des difficultés importante de la détection consistant à obtenir des objets fermés. Chu et Aggarwal [CA93] intègrent des informations de type région et contour dans une structure résultat. Les cartes de régions sont converties en cartes de contours par simple extraction des frontières, et l'image résultante provient d'un judicieux compromis entre différentes images sources, toutes contenant des contours.

La dernière sous-classe réunit plus dynamiquement les deux approches. Anderson, Bajcsy et Mintz [ABM87] intègrent effectivement une croissance de régions et une détection de contours dans un même processus itératif de raffinement, dont l'objectif est le meilleur compromis entre les deux segmentations, par rapport à des contraintes définies par l'utilisateur, comme par exemple le nombre de régions à détecter. Ce mécanisme itératif se traduit dans l'implémentation par un mécanisme d'ajustement mutuel des paramètres des deux méthodes. Nazif et Levine [NL84] proposent enfin une coopération d'opérateurs de segmentation de bas niveau à travers un véritable système expert, dont les règles opèrent de façon originale sur des connaissances de bas niveau, hors de toute sémantique liée à l'image, et portant sur des configurations locales de contours et de régions.

La plupart de ces approches hormis [ABM87] et [NL84] présupposent en fait l'existence d'informations de bas niveau, par exemple sous la forme d'une carte des contours, et recherchent la meilleure façon de combiner les résultats issus de chaque méthode. La coopération devient une sorte de règle de raffinement final du résultat, au lieu d'être partie intégrante de l'analyse des primitives à détecter. En conséquence cette coopération est fondée sur des informations déjà calculées, qui peuvent être entachées d'erreur, car calculées séparément, et indépendamment.

Nous pensons au contraire que la coopération doit être considérée comme un moyen pour créer de l'information supplémentaire au moment d'une prise de décision, de manière opportuniste, et réactive au contexte. L'incrémentalité de la construction présentée dans le chapitre précédent nous semble à ce titre indispensable. Elle permet en effet une évolution couplée des primitives, à la différence de nombreuses autres approches qui apposent l'étiquette coopérative

[6] chaque zone carrée de dimension N de l'image, à commencer par l'image dans sa totalité peut être récursivement découpée en sous-carrés de coté $N/2$. Chaque région construite dans l'image et s'appuyant exclusivement sur des carrés de ce type est alors représentable par une structure arborescente de degré 4.

[7] qui correspondent aux régions élémentaires

à des systèmes dont la caractéristique est d'utiliser une méthode pour servir de référence à une autre. L'incrémentalité dans cette optique permet ainsi d'exploiter et d'explorer des phénomènes d'influence mutuelle.

La coopération passe tout d'abord par la délégation des activités. Les processus de bas niveau doivent pouvoir travailler localement dans l'image. Cela répond à un besoin précis. Les difficultés de segmentation portent généralement sur des endroits très locaux, et un processus qui travaille localement sera naturellement plus efficace pour les déceler, et éventuellement ensuite pour les résoudre. La coopération passe ensuite par l'incrémentalité, qui permet d'analyser progressivement la situation, et de faire face aux difficultés de façon séquentielle, à mesure qu'elle sont identifiées. Les moyens pour résoudre ces difficultés de segmentation passent par l'émergence d'information locale supplémentaire (en faisant appel à d'autres processus spécialisés). Il semble opportun de requérir cette aide extérieure lorsque le processus de segmentation se trouve en situation d'échec, c'est-à-dire lorsque ses propres compétences ne lui permettent plus de poursuivre de manière robuste [8]. Cette coopération avec d'autres méthodes a donc pour fondement la résolution de problèmes et comme soucis une efficacité dans l'intervention. Cette accumulation d'information dans l'environnement de travail permet de prendre des décisions plus adaptées, qui ne se fondent plus seulement sur les compétences propres de la méthode, mais sur la synergie acquise grâce aux autres méthodes sollicitées.

2.3 Le schéma général

L'implantation sous la forme d'un système d'exploitation permet de manipuler un ensemble de processus. Ces processus sont des méthodes incrémentales disposant d'une certaine durée de fonctionnement, et effectuant des actions sur leurs structures de données internes et sur leur environnement au cours de cette période de fonctionnement. Le fonctionnement de deux processus quelconques s'effectue généralement de manière asynchrone dans le système [9], dans le sens où chacun fonctionne de façon autonome, sans besoin particulier de synchronisation par rapport à une condition externe. Un processus a une période de vie dans le système : il est créé à un certain moment [10], il a une période d'exécution, puis une terminaison. A l'issue de son exécution, le système ne conserve de ce processus que la primitive qu'il a segmentée.

Une population de processus constitue le système à tout instant. Cette population évolue, se reproduit, communique. Cette reproduction entre les processus induit la notion de filiation entre les processus, ainsi que la création d'un arbre des processus courants dans le système. Chaque processus possède un *père* unique, et peut lui-même avoir plusieurs *fils*. La notion de flux d'informations et plus généralement de communication entre les processus est un enjeu crucial pour la coopération. Elle est présente sous deux formes complémentaires :

- Une communication de type hiérarchique entre un processus *père* et les processus *fils* qu'il a créé. De nouveaux processus sont créés afin de faire émerger localement de l'information nouvelle, qui aidera la segmentation des processus existants. Cette aide se traduit par la connaissance de nouvelles primitives, qui par leur existence, peuvent modifier le comportement du processus qui avait demandé leur construction. Un premier flot de communication consiste donc à faire "remonter" l'information accumulée par un processus vers son père, aussi rapidement que possible.

[8]Une politique de construction non coopérative consisterait dans ce cas à poursuivre la construction de la primitive, bien que des signes laissent penser que l'objet construit risque d'être entaché d'erreur.

[9]pour deux processus quelconques, ne présentant pas de lien de filiation particuliers, et n'étant pas en cours de négociation pour un terriroire de l'image.

[10]la création d'un processus de segmentation ne coïncide pas nécessairement avec l'initialisation du système.

- Une communication de type topologique entre deux processus physiquement proches dans l'image. Deux processus de segmentation travaillant dans des zones proches peuvent éprouver le besoin d'échanger des informations sur leur primitive respective, afin d'envisager des fusions ou des annexions de territoire.

Les méthodes de segmentation se trouvent désormais encapsulées dans une structure de processus. Elles sont définies à ce titre par des caractéristiques sur lesquelles elles ont plus ou moins d'influence. Les paramètres sur lesquels un processus de segmentation n'a pas contrôle sont les suivants :

- Son lieu d'initialisation. La méthode ne choisit pas le pixel germe à partir duquel elle va débuter la construction de sa primitive. Cet enplacement, qui sera appelé *lieu de focalisation* par la suite, est déterminé par son processus père.
- Sa filiation. Le processus est rattaché à son processus père. Celui-ci sera le destinataire de la primitive segmentée lorsque le processus se terminera.
- L'environnement de l'image déjà segmenté. Le processus s'interdit d'entrer en conflit pour l'étiquetage de pixels avec d'autres processus. La seule possibilité pour incorporer des pixels déjà segmentés consiste donc à entrer dans une phase de négociation avec les processus en charge de ces autres pixels.

Les paramètres sur lesquels un processus peut au contraire exercer son contrôle sont les suivants :

- Les paramétrages de la méthode de segmentation utilisée.
- La politique de mise en relation avec d'autres méthodes. Le processus choisit le lieu, le type, les caractéristiques des processus fils qu'il désire créer localement.
- Les protocoles de coopération visant à permettre des négociations de territoire.

Un point clé apparaît déjà de cette classification sommaire. Si la possibilité de déléguer des tâches à d'autres processus de segmentation s'avére possible dans le système, le processus qui sollicite cette délégation n'a que très peu de contrôle sur l'entité sur laquelle il délègue une activité, une fois cette dernière initialisée. Ceci relève d'une volonté nette de décentraliser les tâches, et d'offrir une autonomie à chacun des processus du système. Par ailleurs, sauf dans des circonstances spécifiques telle la négociation de territoire, aucun processus n'a de rôle prépondérant par rapport à un autre. Cette gestion sous forme d'entités autonomes s'apparente clairement aux systèmes multi-agents de type réactif [11].

2.4 Une coopération Région/Contour

La coopération entre les segmenteurs se réalise en utilisant un principe d'invocation mutuel des différentes méthodes localement. Des processus de segmentation vont en créer d'autres suivant certains critères qui vont être étudiés dans les paragraphes qui suivent. Les deux méthodes de segmentation utilisées sont les algorithmes incrémentaux de détection de contours et de régions présentés au chapitre précédent. Dans un premier temps, une étude sur les moyens de la coopération sera effectuée du point du vue du détecteur de contours. Dans un deuxième temps, la même analyse sera faite pour les régions. Mais avant tout, les mécanismes permettant l'initialisation de ces processus de segmentation sont décrits.

[11] Les processus sont ici réactifs à leur environnement, et réactifs à leurs congénères.

2.4.1 La focalisation

Dans l'optique de pouvoir faire émerger localement de l'information, il est souhaitable qu'un processus de segmentation puisse être initialisé à un endroit précis de l'image. La construction incrémentale d'une primitive implique donc d'initialiser le mécanisme de construction, en fournissant au processus le premier pixel devant appartenir à la primitive. Ce premier point sera appelé un *germe* par la suite [12].

Il n'est donc pas interdit de choisir correctement ce premier pixel, car de son emplacement dépendra le succès ou l'échec probable de la construction. Trivialement, le type du processus va guider le choix de l'emplacement de ce germe. Un germe de contour et un germe de région ne seront pas placés au même endroit.

2.4.2 Les fenêtres de focalisation

Dans l'optique où un processus de segmentation va initialiser un autre processus de segmentation quelque-part dans l'image, il devient rapidement nécessaire de définir ce que l'on entend par *quelque-part dans l'image*, ainsi que les rôles respectifs des protagonistes de cette création.

Le lieu de l'initialisation du nouveau processus est clairement connu par le processus qui demande sa création. Il reste ensuite à déterminer quelle est la part de contrôle qui revient au processus créateur, par rapport aux deux autres entités de ce système, qui peuvent légitimement prétendre pouvoir aussi partager ce contrôle :

– Le système. Cet objet a pour rôle de manipuler les processus de segmentation comme autant de tâches anonymes. Il supervise leur création, leur exécution et leur terminaison. A ce titre, une extension logique de son contrôle serait de gérer leur initialisation.
– Le nouveau processus créé. Il peut prétendre également pouvoir influer sur son pixel de démarrage, si l'on suppose qu'il dispose d'une meilleure adaptation locale que le processus qui l'a créé pour définir si son pixel d'initialisation est "prometteur" ou pas. Mais ce contrôle peut rester centré sur le processus créateur, en arguant du fait qu'un processus mal situé peut se terminer en mode d'échec, en laissant à la charge de son créateur le soin de trouver un pixel germe plus adéquat.

Nous avons opté pour une approche partageant le contrôle entre le processus créateur et le système. L'introduction d'un mécanisme de niveau système, c'est-à-dire externe au processus de segmentation, permet de plus une certaine uniformisation dans les méthodes de création de processus. Le processus définit une fenêtre dans l'image, avec un certain nombre de processus désirés, leurs types respectifs, ainsi que la méthode de sélection de leurs germes. Le contrôle du niveau système consiste alors à exécuter un algorithme de recherche de pixel germe dans cette fenêtre selon le type de sélection souhaité.

Le contrôle est effectivement partagé entre le processus et le système, et est de plus largement modulable, puisque le processus peut choisir arbitrairement la taille de la fenêtre de focalisation. Le cas d'une fenêtre de dimension 1×1 pixel laisse alors peu de choix pour placer le germe au niveau système. Tout le contrôle réside alors dans le processus de segmentation. Hors de cet extrême, la taille de fenêtre 5×5 actuellement implémentée permet un partage réel du contrôle.

La suite de ce paragraphe décrit les algorithmes de choix mis en œuvre en fonction du type de processus souhaité. L'étape de localisation des germes consiste à fournir une fenêtre dans l'image, la méthode de sélection des germes, et le nombre de germes souhaités dans cette fenêtre. Trois méthodes de sélection sont proposées [13], mais cette liste n'est pas limitative :

[12] Un seul pixel est nécessaire pour initialiser les détecteurs, puisqu'ils fonctionnent de façon incrémentale, en construisant la solution point par point.

[13] Elles seront référencées par la suite sélection de type *photométrique*, de type *grille* ou de type *aléatoire*.

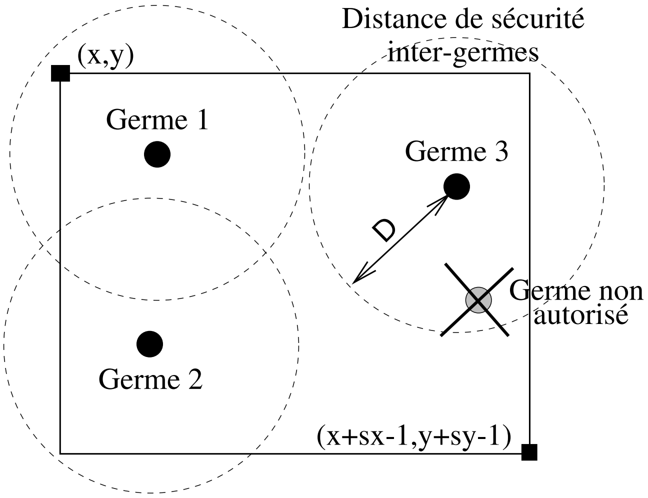


FIG. 2.2: Espacement entre les pixels germes dans la fenêtre de focalisation.

1. Choisir le pixel sur des critères photométriques, dont la nature dépend du type du processus région ou contour.

1.1. Choisir le pixel réalisant la plus forte valeur de la norme du gradient dans la fenêtre de focalisation. Cette méthode sera bien adaptée pour positionner des germes de contour.

1.2. Choisir le pixel réalisant la plus faible valeur de l'écart-type local, caractérisant pour un pixel donné l'homogénéité des niveaux de gris dans son voisinage 5x5. Cette méthode permettra de placer des germes de région.

2. Choisir le pixel de manière indépendante des données, par exemple, centré dans la fenêtre de focalisation

3. Choisir le pixel de manière indépendante des données, par exemple, sélectionné au hasard dans la fenêtre de focalisation.

La transition entre la sélection d'un germe unique et la sélection d'un ensemble de germes dans une même fenêtre de focalisation nécessite quelques aménagements. Empiriquement, il est fréquent que les pixels réalisant le maximum d'une mesure photométrique, par exemple le gradient, soient groupés dans l'image. Ces pixels se retrouvent par conséquent aussi groupés dans la fenêtre de focalisation. Le choix de plusieurs pixels germes sur un même critère de plus forte valeur de gradient implique donc des germes groupés, qui empiriquement ont de fortes probabilités d'être rattachés à la même primitive, ce qui n'est pas satisfaisant [14].

Il nous a donc semblé intéressant d'ajouter une contrainte d'espacement à la sélection des germes, permettant de les répartir au mieux dans la fenêtre, tout en continuant de respecter le critère de sélection sous-jacent. Ce critère d'espacement ne garantit pas l'émergence de primitives distinctes, mais choisir des germes éloignés les uns des autres tend à augmenter cette potentialité.

Cette distance minimum entre deux germes est définie par la valeur D, voir la figure 2.2, et dépend de la taille de la fenêtre ainsi que du nombre de germes recherchés. Nous définissons la fenêtre de focalisation $\mathcal{F}$, de taille $sx \times sy$ localisée dans l'image par son coin supérieur gauche (x, y). Posons N, le nombre de germes à initialiser dans la fenêtre. La distance minimum entre deux pixels germes est définie par $D = \alpha\sqrt{\frac{sx.sy}{N}}$, avec $\alpha = 0,8$ dans notre application. La valeur

[14]Si l'on souhaite créer plusieurs processus dans une même fenêtre de focalisation, c'est dans le but de segmenter autant de primitives distinctes les unes des autres.

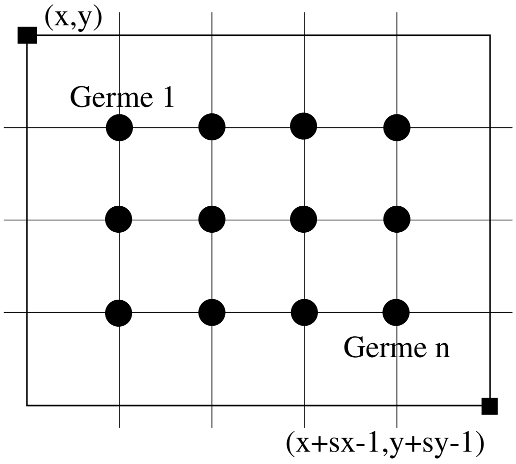


FIG. 2.3: Méthode de sélection des pixels germes par placement sur une grille régulière.

de cette constante doit être choisie inférieure à 1, qui correspondrait au cas optimal, selon lequel chaque germe serait exactement à une distance D de ses voisins.

L'algorithme de sélection est ensuite très simple, puisqu'il se limite à trier tous les pixels de la fenêtre en fonction de la valeur de la fonction mesurée (gradient ou écart-type), on supposera que la liste est triée par ordre décroissant de la valeur de la fonction :

- Le pixel en tête de liste est choisi comme germe, et est extrait de la liste.
- Tant que le nombre de germes à créer n'est pas atteint, et tant qu'il reste des pixels en liste, faire :
  - Si le pixel en tête de liste est à une distance supérieure à D de tous les autres germes déjà choisis, alors ce pixel est extrait et ajouté à la liste des germes.
  - Sinon, il est rejeté et est extrait de la liste.

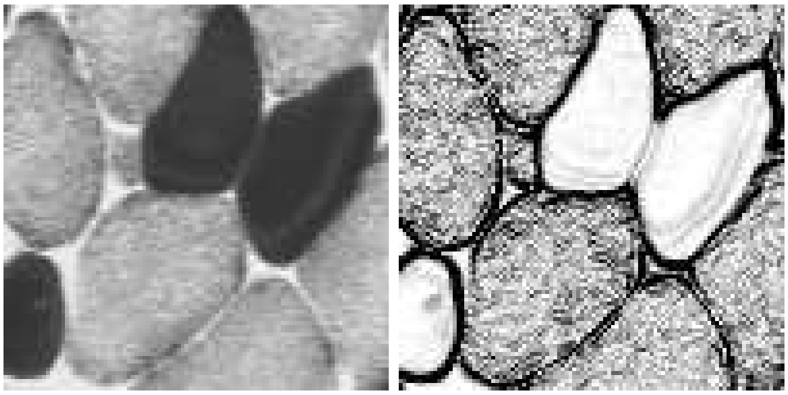
Ce principe s'applique donc sans difficultés aux méthodes de sélection 1.1 et 1.2 précédemment décrites. L'extension de la méthode de sélection 2. à plusieurs germes se fait pareillement sans douleur en plaçant les germes sur une grille fictive à l'intérieur de la fenêtre, et dont la taille dépend elle-aussi de N, sx et sy. En se plaçant sous la condition $sx \geq sy$, le nombre de germes par lignes est défini par $N_x = \lfloor\sqrt{N.\frac{sx}{sy}}\rfloor$, de même le nombre de germes par colonne $N_y = \lfloor N/N_x \rfloor$, aux approximations entières près. Voir l'exemple en figure 2.3.

La méthode de sélection 3. se fait par un calcul de coordonnées pseudo-aléatoires, limitées à la fenêtre de focalisation, pour autant de germes que le processus en a demandé, sans contraintes d'espacement.

2.4.3 Les contours utilisent les régions

2.4.3.1 La motivation

Les primitives de type région et contour sont clairement duales, et difficilement dissociables l'une de l'autre au cours de leur construction. Salotti dans ses travaux de thèse [Sal94] indique qu'il convient de ne pas appliquer le même seuil pour la détection des contours, selon que les régions situées de chaque côté sont homogènes ou pas. Une discontinuité avec un gradient extrêmement faible (de norme inférieure à dix par exemple) sera parfaitement visible si elle sépare deux régions homogènes, alors qu'un gradient comparable en zone texturée n'est absolument plus significatif. La figure 2.4 montre à ce titre un petit exemple illustrant cette



(a) Les niveaux de gris (b) La norme du gradient

FIG. 2.4: Cet exemple montre la nécessité d'adapter le seuillage appliqué sur la norme du gradient selon les caractéristiques des régions dans le voisinage de la transition. La figure (a) présente une partie 100×100 d'une image cellulaire, et la figure (b) est une représentation de la norme du gradient des niveaux de gris. Les parties sombres correspondent aux fortes valeurs du gradient. Cette image présente à la fois une très faible transition entre deux cellules noires de niveau de gris homogène, et une transition délicate à mettre en évidence entre deux cellules texturées. Il apparaît clairement qu'un seuillage permettant de délimiter les deux cellules noires détectera une multitude de faux pixels contours au sein des textures, où le gradient moyen est largement supérieur à celui de la transition entre les cellules noires.

situation. Une transition à faible gradient sépare les deux cellules noires, alors qu'un gradient comparable est présent sur une grande quantité des pixels des zones texturées et ne préjugent en rien de l'existence d'une transition à ces endroits là.

Il nous apparaît donc indispensable qu'un processus de détection de contour puisse adapter son comportement en fonction de l'environnement. Plus précisément, en repérant l'existence des primitives de type région dans son environnement de travail, et en consultant leurs paramètres, il pourra ainsi ajuster ses seuils de segmentation. Le processus se sert ainsi de régions pré-existantes.

Mais il se peut que ces informations ne soient pas disponible. Plusieurs cas sont alors envisageables :

- La fonction d'évaluation des pixels candidats indique des points très robustes. Le processus poursuit la construction de manière autonome, tant que des pixels candidats valides sont disponibles. Il n'est pas utile de solliciter des interventions extérieures intempestives aussi longtemps que le processus peut travailler de manière autonome, avec confiance.
- Le processus de segmentation ne dispose plus de pixels candidats valides. Plutôt que de poursuivre malgré tout, au risque de commettre des erreurs d'étiquetage [15], le processus de type contour va susciter la création d'une région dans sa périphérie. En faisant émerger l'information qui lui manque pour continuer à progresser de manière robuste, le processus de segmentation pourra ainsi renforcer ou remettre en cause la suite de la construction du contour.

[15] Ces erreurs ne seraient pas corrigées par la suite, car notre approche ne remet plus en cause des pixels déjà étiquetés. Ceci garantit entre autres que notre algorithme se termine, car chaque processus région ou contour ne peut qu'étiqueter de nouveaux pixels dans un univers borné.

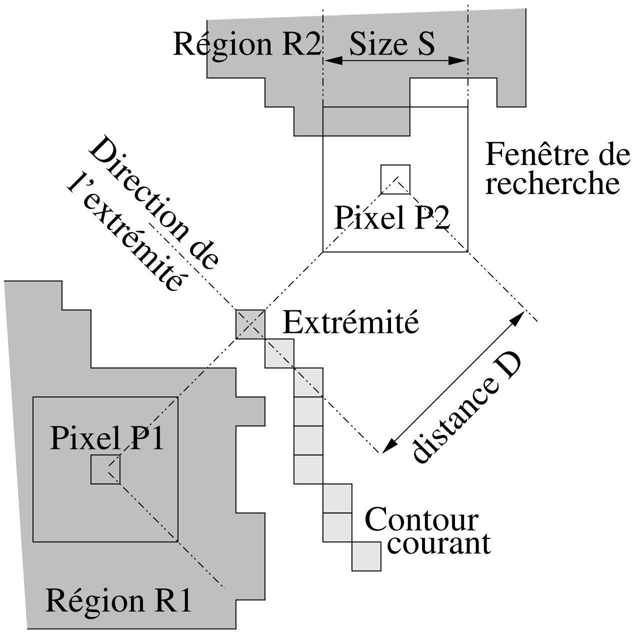


FIG. 2.5: Recherche des régions adjacentes existantes

2.4.3.2 Utilisation de régions existantes

L'objectif est de permettre au processus de type contour de modifier son comportement par la connaissance sur des régions de son voisinage de construction. Ce paragraphe adresse le cas où ces régions existent déjà.

La mesure de l'homogénéité des régions adjacentes au contour permet d'ajuster le seuil à appliquer sur le gradient. Deux zones de recherche rectangulaires sont initialisées de part et d'autre de l'extrémité du contour. Une région est recherchée dans ces deux zones, voir la figure 2.5. Pour notre application, nous avons choisi de fixer les paramètres suivants : $D = 5$, et $S = 5$.

- Si une région adjacente R_i est trouvée dans la fenêtre de recherche, son écart-type est mémorisé σ_i [16].
- Si aucune région n'est trouvée dans la fenêtre de recherche, l'écart-type local du pixel au centre de la fenêtre P_i (voir la figure 2.5) remplacera l'information région manquante, σ_i [17].

Ce principe de recherche d'information dans l'environnement montre clairement qu'aucun processus de segmentation ne détient une connaissance centralisée de la segmentation. Chaque processus doit rechercher l'information dont il a besoin dans l'environnement. Il s'agit typiquement d'une illustration de la communication inter-processus utilisant l'environnement comme médium de communication.

Un seuil sur le gradient est calculé à partir de la table 2.4.3.2, en fonction de σ_1 et σ_2, mesure d'homogénéité de part et d'autre de l'extrémité. Cette table n'introduit pas de dissymétrie entre σ_1 et σ_2, et définit un seuil croissant lorsque σ_1 et σ_2 augmentent. L'idée intuitive étant à l'origine

[16] L'écart-type de la région est la moyenne des écart-types locaux des pixels qui la constituent. L'écart-type local d'un pixel est précalculé à partir de l'image des niveaux de gris, sur une petite fenêtre 5×5 centrée sur le point.

[17] Il est bien sûr préférable d'utiliser les caractéristiques d'une région lorsqu'elle est accessible, car elle concerne davantage de pixels, et permet donc une "vue" plus globale de la situation.

	$\sigma_1 < 2$	$\sigma_1 < 5$	$\sigma_1 < 8$	$\sigma_1 < 10$	$\sigma_1 \geq 10$
$\sigma_2 < 2$	4	7	9	11	15
$\sigma_2 < 5$	.	10	15	19	23
$\sigma_2 < 8$	.	.	18	23	27
$\sigma_2 < 10$	.	.	.	29	35
$\sigma_2 \geq 10$	.	.	.	.	40

TAB. 2.1: Table des seuils de gradient adaptatifs.

de ces valeurs consiste à remarquer qu'une transition, pour être visible par l'oeil humain, devra être d'autant plus marquée que les régions avoisinantes sont texturées. Le choix de ces valeurs est empirique, et est largement fondé sur les critères de visibilité définis par Salotti dans ses travaux de thèse [Sal94].

2.4.3.3 Création de nouvelles régions

L'objectif reste encore pour le contour d'utiliser les caractéristiques des régions avoisinantes. La différence de traitement réside dans le fait que les régions n'existent pas à l'avance, et que le processus de segmentation de type contour ne dispose plus de pixels candidats présentant une évaluation satisfaisante pour poursuivre. Il devient donc nécessaire de créer de nouveaux processus, permettant de faire émerger localement de nouvelles primitives régions, qui guideront le processus de type contour pour la suite de sa progression.

Le processus de segmentation ne peut plus poursuivre de manière certaine, car tous les pixels candidats examinés présentent une évaluation qui est inférieure au seuil utilisé par ce processus. Il suspend ses activités et transmet une requête de *création de processus fils* en direction du séquenceur de tâches. A chaque extrémité, deux processus de type région, de chaque coté de l'extrémité du contour, et un autre processus de type contour, dans le prolongement de l'extrémité, sont initialisés.

- Une fenêtre de focalisation, dont le processus choisit la taille et l'emplacement (le principe de la focalisation est détaillé en paragraphe 2.4.1), est placée dans l'image. Le processus indique le nombre, le type de processus fils à créer, ainsi que la méthode à utiliser (forts gradient, faible écart-type local, ...)
- Les informations photométriques extraites de cette fenêtre de focalisation servent à initialiser les seuils du processus fils associé. Il s'agit donc ici de l'instanciation d'un processus générique de segmentation, avec des seuils qui en font une entité localement adaptée à l'image sur laquelle il travaille.
- Les deux processus de type région ont un rôle *global* et *local*. Globalement, ils contribuent à enrichir la segmentation de l'image. Il est intéressant à ce titre que le système automatise la détection de nouveaux objets sans solliciter l'intervention de l'utilisateur. Ce dernier peut ainsi espérer pouvoir obtenir une segmentation complète de l'image avec un petit nombre de processus initiaux. Localement, un enrichissement de l'environnement permettra au processus de type contour de sécuriser ses futures décisions en ayant une connaissance des régions environnantes.
- Le processus de type contour initialisé dans le prolongement présente les mêmes objectifs que les régions, à la différence près qu'il s'agit dans ce cas d'une délégation de la tâche de segmentation qui est donnée par le père à son fils. Le processus père, à la terminaison de ce fils, pourra fusionner sa primitive avec celle du contour fils nouvellement segmenté, si les conditions photométriques le permettent (voir figure 2.8). Chaque processus fait remonter

l'information segmentée à son père. Un processus contour père peut donc disposer de plusieurs choix pour tenter une fusion, comme cela est le cas sur cette même figure.

La figure 2.6 explicite graphiquement le mécanisme de focalisation. Le schéma 2.6(a) montre le placement des fenêtres de focalisation, et le schéma 2.6(b) présente le résultat théorique à l'issue de la terminaison des trois processus fils. On notera plusieurs points. La construction du contour se fait indifféremment par ses deux extrémités. A ce titre, la focalisation est appliquée simultanément aux deux extrémités. Le processus contour choisit l'emplacement des fenêtres de focalisation sur des critères géométriques uniquement, liés à l'orientation de son extrémité. Le mécanisme de focalisation va ensuite opérer une recherche du pixel germe le plus prometteur dans cet espace. Cependant, rien ne garantit le succès des processus fils générés. Obtenir un contour en prolongement, et deux régions de part et d'autre de l'extrémité est donc le cas le plus favorable que le processus peut rencontrer à l'issue de la focalisation.

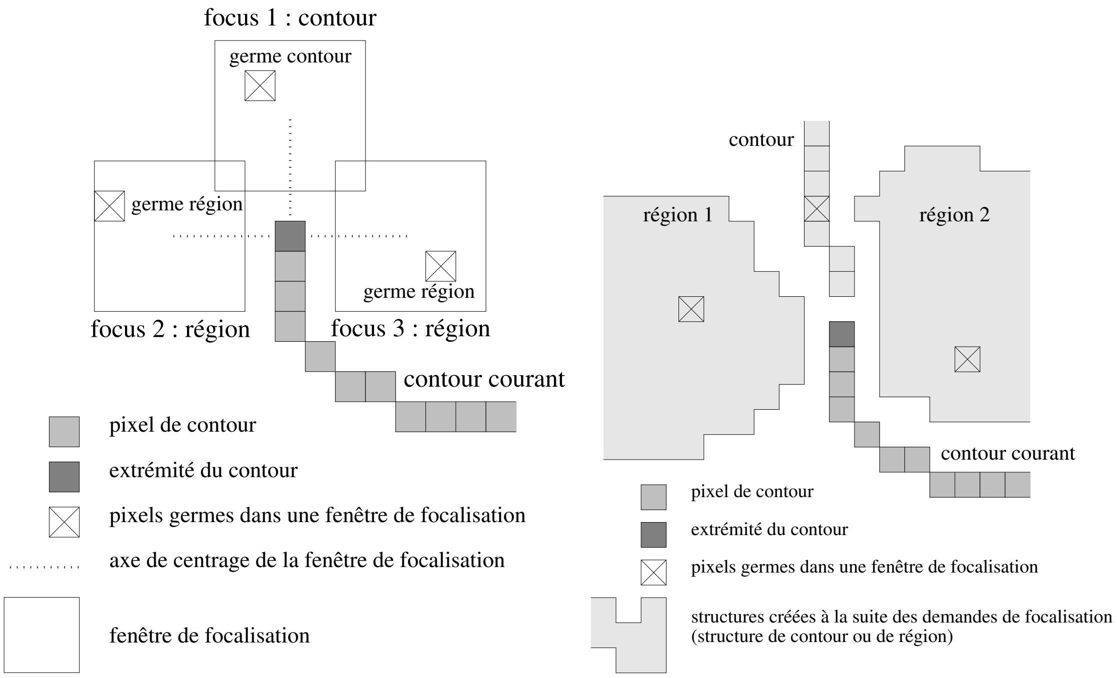
La figure 2.7 présente une approche plus dynamique de la focalisation avec un chronogramme. Le processus contour père est figuré sur la première ligne, les trois processus fils créés apparaissent sur les lignes inférieures. Ce chronogramme est une simulation d'exécution en environnement multi-tâches, puisqu'un seul processus est actif à un instant donné. En particulier les trois processus ne peuvent pas s'exécuter en même temps. Cet émiettement des tâches sera justifié dans le chapitre 3, relatif à l'implantation sous la forme d'un système d'exploitation simplifié. Les flèches descendantes correspondent à l'initialisation de l'environnement des fils réalisée par le processus père. Les flèches ascendantes figurent la remontée d'information lorsque les processus se termine. La primitive segmentée est transmise au processus père.

2.4.3.4 L'instanciation des processus fils

L'instanciation du processus contour consiste à initialiser sa méthode de travail (une détection de contour ou de région), et à déterminer le seuil à appliquer sur la fonction d'évaluation [18]. Ce seuil est calculé à partir du gradient moyen sur la fenêtre de focalisation.

$$
\begin{aligned}
F_{[x_0,y_0,x_1,y_1]} &= \text{La fenêtre de focalisation} \\
\overline{G_F} &= \frac{\sum_{i=x_0}^{i\leq x_1}\sum_{j=y_0}^{j\leq y_1}\|G(\vec{i,j})\|}{(x_1+1-x_0)(y_1+1-y_0)} \\
E &= \alpha_1E_1+\alpha_2E_2+\ldots+\alpha_iE_i+\ldots+\alpha_{n-1}E_{n-1}+\alpha_nE_n \\
E_i &= \frac{G(x,y)}{\max_{(x_i,y_i)\in\mathcal{P}}G(x_i,y_i)} \\
& \quad \text{l'évaluation de la norme du gradient} \\
S &= \alpha_1S_1+\alpha_2S_2+\ldots+\alpha_iS_i+\ldots+\alpha_{n-1}S_{n-1}+\alpha_nS_n \\
S_i &= \beta\frac{\overline{G_F}}{\max_{(x_i,y_i)\in\mathcal{P}}G(x_i,y_i)} \\
& \quad \text{avec } \beta = \text{constante} = 2.25 \\
S_j &= \frac{1}{2},\ \forall j\neq i \\
\sum_k \alpha_k &= 1
\end{aligned}
$$

[18] La description des méthodes incrémentales de segmentation en régions et en contours est faite dans le chapitre 1. Rappelons succinctement que chaque étape de la construction consiste à choisir le meilleur des pixels candidats pour être ajouté à la primitive courante. Chaque pixel candidat est donc numériquement évalué. Un seuil sur cette fonction d'évaluation permet de définir une limite sur la qualité des pixels candidats à l'incorporation.



(a) Avant l'exécution des processus fils (b) Après l'exécution des processus fils

FIG. 2.6: La figure (a) montre un processus de détection de contours qui génère trois requêtes de création de processus fils, à l'aide de trois *fenêtres de focalisation* positionnées dans les directions formées par l'extrémité du contour. Deux processus de type région sont initialisés de part et d'autre de l'extrémité, un processus de type contour est créé dans le prolongement de l'extrémité. La figure (b) indique les résultats que le processus père peut espérer obtenir après la terminaison de ses trois processus fils.

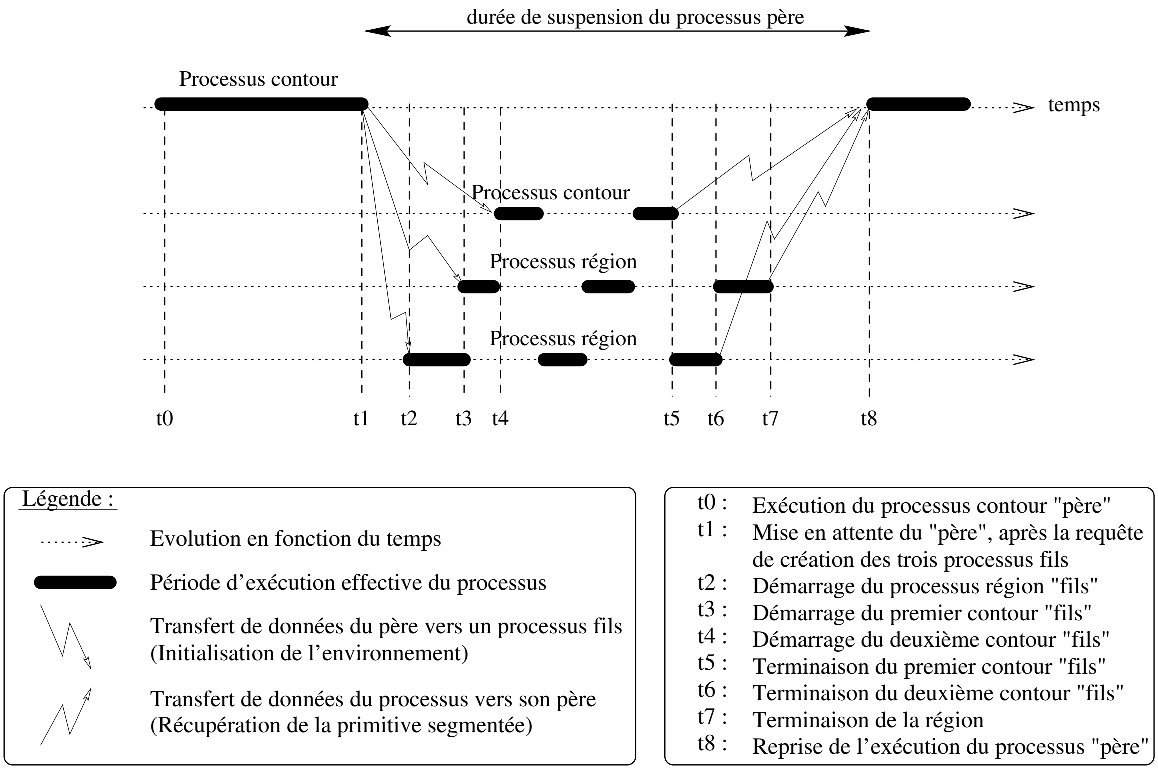


FIG. 2.7: Chronogramme de l'exécution du système lors des requêtes de focalisation du processus de type contour.

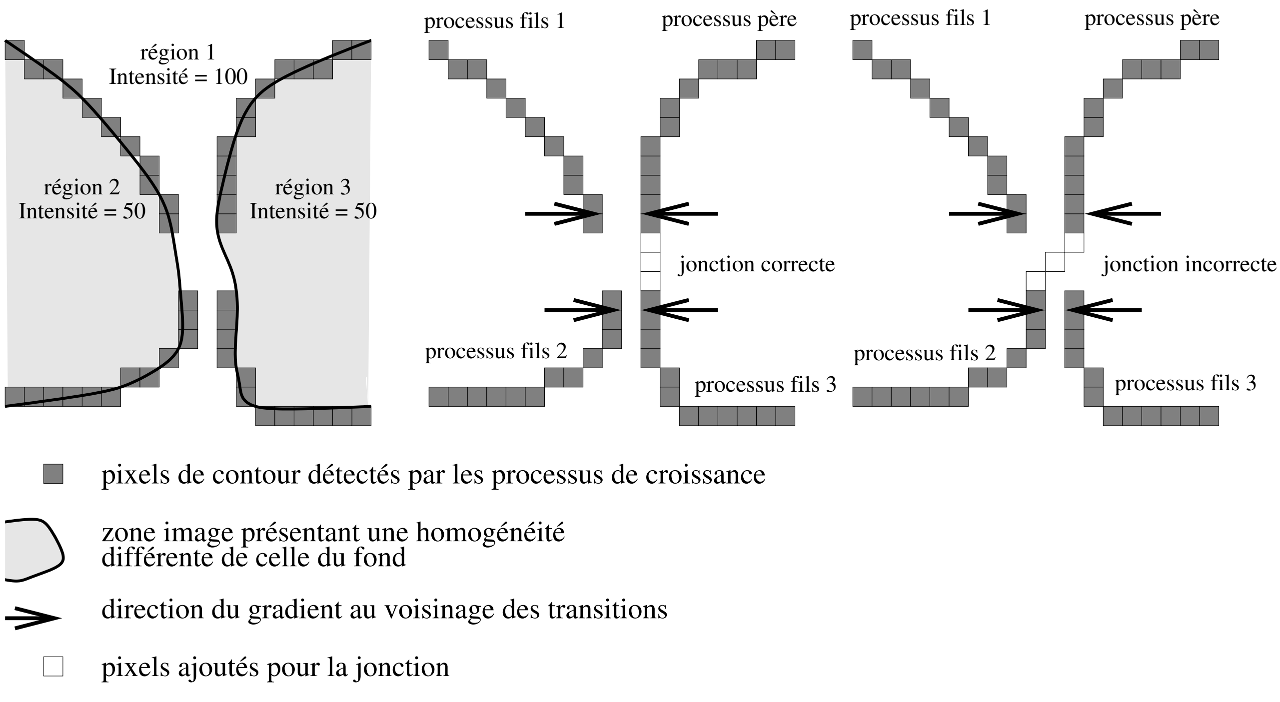


FIG. 2.8: Cohérence de la fusion entre des contours proches.

L'évaluation E_i de la détection de contour est celle mesurant la norme du gradient. L'évaluation E est la pondération linéaire des évaluations élémentaires. Le seuil S utilisé par le processus contour sera donc également une pondération des seuils élémentaires s_j, parmi lesquels s_i est obtenu par la mesure du gradient moyen sur la fenêtre de focalisation F.

Il apparaît de ces définitions que tous les seuils élémentaires sont obtenus en prenant la valeur moyenne de leur intervalle de variation, soit $\frac{1}{2}$, à l'exception de la mesure du gradient, pour laquelle le seuil appliqué est une pondération du gradient moyen sur la fenêtre de focalisation. Le seuil global du processus sera obtenu à partir des seuils élémentaires par la même combinaison linéaire qui permet d'obtenir l'évaluation globale en fonction des évaluations élémentaires [19].

2.4.3.5 La récupération des informations segmentées

Chaque processus de segmentation a été créé par un autre processus de segmentation selon le schéma décrit précédemment [20] [21]. L'information sur la primitive segmentée doit donc logiquement revenir au processus père après la terminaison d'un processus. Cela permet d'enrichir la connaissance du processus père sur l'environnement.

Cette approche constitue une alternative à la communication par l'environnement décrite dans le paragraphe précédent. La communication est ici de type "point à point", avec une contrainte forte entre les deux entités communicantes : le lien de filiation.

Chaque processus doit alors distinguer deux types de primitives :

- La primitive sur laquelle il travaille. Elle est unique, non interchangeable, et typée région ou contour.
- Un ensemble d'autres primitives régions ou contours qui auront été acquises au gré des focalisations successives. Elles correspondent, en quelque sorte, à la connaissance acquise sur l'environnement. Cet ensemble est vide au démarrage du processus.

Chaque processus lorsqu'il se termine transmet donc la totalité de ces données à son processus père. Il y a plusieurs avantages à ce principe :

- Peu à peu, l'ensemble des primitives segmentées "remonte" vers le processus racine du système, ou encore appelé "processus initial". Ce processus initial joue ainsi un rôle centralisateur, puisqu'il regroupe à la fin du système l'ensemble des objets segmentés.
- Cette remontée d'information permet de ne pas perdre de primitives en cours d'exécution. Une perte de primitive se caractérise alors par le fait qu'aucun processus ne dispose de pointeur sur une primitive. Cette perte de primitive pose alors un problème pour le processus initial, dont le rôle naturel est de *centraliser* toutes les primitives segmentées à l'issue de l'exécution des autres processus. Cette perte de primitive se traduira pour le processus initial par des objets régions ou contours manquant, bien qu'ayant été segmenté à un moment donné dans l'image, et bien que disposant encore de pixels étiqueté à cette primitive dans l'image.

Les inconvénients qui peuvent apparaître :

[19] Les pondérations utilisées sont constantes pour chaque processus durant toute l'exécution du système. La levée de cette contrainte permettrait de spécialiser les segmenteurs. Une initiative de ce type est proposée dans le cadre de l'Interface Homme Machine décrite au chapitre 4. Elle reste cependant complètement pilotée par l'utilisateur.

[20] A l'exception du processus initial, où processus *racine*, qui correspondra par exemple à l'initialisation d'un germe initial par l'utilisateur.

[21] En supposant qu'un processus de type région peut créer à son tour d'autres processus de type contour, voir à ce titre le paragraphe 2.4.4.

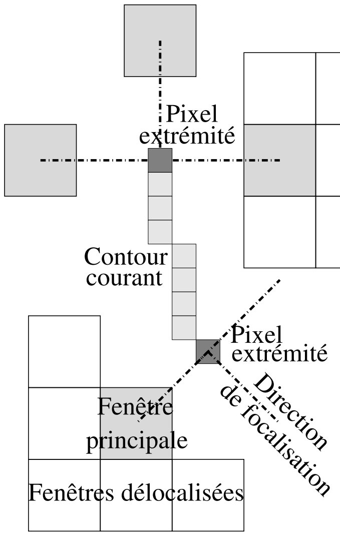


Fig. 2.9: La délocalisation des fenêtres de focalisation.

– Les processus hiérarchiquement proches du processus initial seront ceux qui concentreront le plus d'information. Ils risquent de perdre leur caractéristique de localité. Dans les étapes de recherche parmi leurs primitives connues, ils perdront en particulier beaucoup de temps à passer en revue des primitives qui sont éloignées de leur zone d'intérêt.
– Cette approche enfin n'est plus en accord avec les principes de connaissances distribuées propres aux systèmes multi-agents, puisqu'un seul processus (même s'il a un statut très particulier) regroupe toute la connaissance du système. Cependant, le processus initial n'a pas d'activité propre hormis ce rôle de concentrateur de la connaissance du système, ce qui réduit l'impact de l'entorse faite au principe de distribution des connaissances.

2.4.3.6 Les fenêtres de focalisation alternatives

L'emplacement des fenêtres de focalisation est directement lié au territoire qui est exploré par le système. Placer une fenêtre dans un endroit de l'image offre une "chance" pour la primitive située à cet endroit d'être découverte. L'objectif étant d'obtenir la meilleure couverture de l'image en terme de primitives à l'issue de l'exécution, une possibilité pour favoriser cette couverture est de limiter globalement au maximum le recouvrement des fenêtres de focalisation. De plus, les processus initialisés dans de telles fenêtres ont davantage de chance de démarrer dans une zone inexplorée, et ainsi de travailler utilement [22].

Le principe consiste pour un processus de segmentation à proposer, non plus une unique fenêtre, mais une ensemble de fenêtres disjointes, formant un pavage local de l'espace selon la schéma décrit en figure 2.9. Cette figure ne montre pas toutes les fenêtres délocalisées pour des raisons de clarté. Les directions de recherche différentes aux deux extrémité génèrent éventuellement des zones de recouvrement entre les pavages correspondant à deux fenêtres principales distinctes. Le terme de *fenêtre principale* référence l'unique fenêtre de base utilisée précédemment

Le système conserve globalement une carte mesurant la densité de focalisation. Cette mesure indique pour chaque pixel le nombre de fenêtres de focalisation auxquelles il a appartenu. La densité de focalisation étendue à une fenêtre de pixels sera la plus forte densité parmi tous les

[22]Un processus dont le germe initial est placé à l'endroit d'une primitive existante n'exécute aucune action, et retourne simplement un pointeur informatif à son processus père.

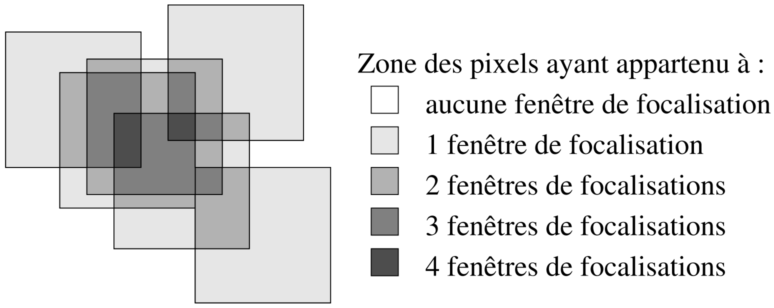


FIG. 2.10: La densité des focalisations.

pixels de la fenêtre. La figure 2.10 présente un exemple de mesure de la densité de focalisation, après que plusieurs requêtes aient eu lieu dans des zones proches. Les pixels ayant les valeurs les plus foncées correspondent aux pixels qui ont eu le plus d'opportunités de devenir des pixels germes d'un nouveau processus de segmentation.

Le système retient, parmi toutes les fenêtres proposées par le processus, celles réalisant le minimum de la mesure de densité de focalisation. Cette mesure de densité apparaît comme un enrichissement de l'environnement, élaborée dynamiquement par l'ensemble des processus de segmentation, et constitue à ce titre un nouvel exemple de communication inter-processus utilisant l'environnement image comme médium de communication.

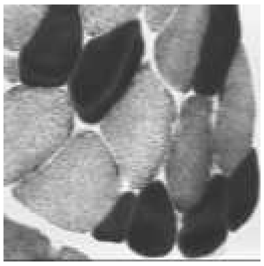
L'exemple suivant présente concrètement l'intérêt de ces fenêtres de focalisation supplémentaires sur la qualité finale de la segmentation. Une partie de l'image de coupe de cellules musculaires de taille 128×128 est utilisée à cet effet. Les figures 2.11 et 2.12 présentent respectivement le cas où une unique fenêtre de focalisation est utilisée pour créer un nouveau processus (il n'y a pas de délocalisation), et le cas où un pavage de fenêtres de focalisations alternatives est utilisé (la fenêtre choisie sera donc délocalisée). Les conditions d'initialisation sont les mêmes dans les deux expériences. Cinq germes de type région et contour sont sélectionnés automatiquement dans l'image selon un critère photométrique. Le système commence donc avec dix processus initiaux. Les images 2.11(b), 2.11(e), 2.12(a) et 2.12(d) montrent l'emplacement des fenêtres de focalisation sous une forme cumulée.

La figure 2.11 présente un résultat final dans lequel deux régions importantes en bas à gauche n'ont pas été segmentées. L'image 2.11(f) met en évidence une zone vierge de toute focalisation, qui explique pourquoi ces deux régions n'ont pas été segmentées : peu ou pas de processus ont été initialisés à ces endroits-là.

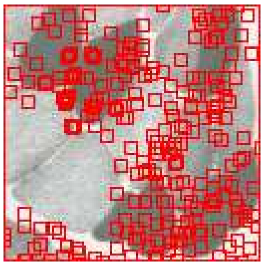
La figure 2.12 présente un résultat final plus complet. Les images prises en cours d'exécution du système indiquent que l'objectif est atteint, puisque les fenêtres de focalisation se répartissent plus harmonieusement dans toute l'image. Les deux régions qui étaient restées non-segmentées dans l'exemple précédent contiennent à présent chacune une dizaine de fenêtres de focalisation, ce qui constitue autant d'opportunités pour un processus de segmenter ces régions.

Cet exemple illustre ici la nécessité de faire un compromis sur la localisation des processus. L'information nécessaire à un processus de segmentation se situe [23] dans son proche voisinage. Cependant, le besoin de traiter l'ensemble de l'image autorise une certaine délocalisation de la recherche, afin de forcer l'exploration de zones un peu plus éloignées des processus en cours d'exécution.

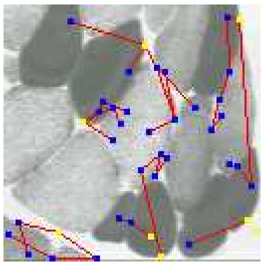
[23] Cette recherche d'information justifie la création de nouveaux processus.



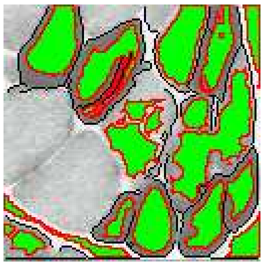
(a) Niveaux de gris.



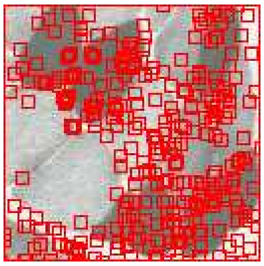
(b) Les fenêtres de focalisation.



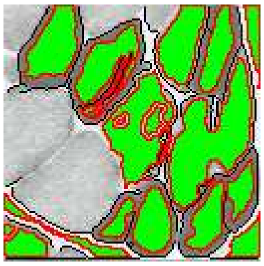
(c) Les liens entre les processus.



(d) Le résultat partiel.

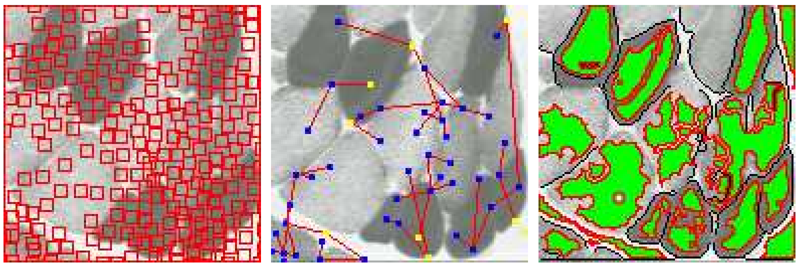


(e) Les fenêtres de focalisations.

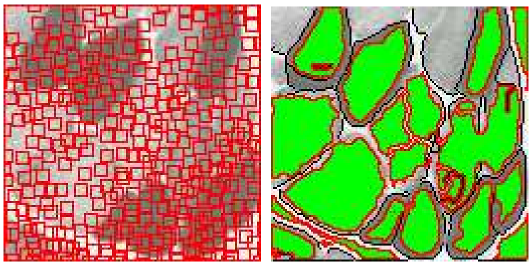


(f) Le résultat partiel.

FIG. 2.11: La création de processus fils n'utilise qu'une unique fenêtre de focalisation. Les images (b), (c) et (d) montre l'état d'avancement du système après 30 secondes d'exécution. Les images (e) et (f) montrent l'état final de la segmentation. Il apparaît qu'une zone importante de l'image reste non-segmentée.



(a) Les fenêtres de focalisations. (b) Les liens entre les processus. (c) Le résultat partiel.



(d) Les fenêtres de focalisations. (e) Le résultat partiel.

Fig. 2.12: La création de processus fils délocalise les fenêtres de focalisation. Les images (a), (b) et (c) montre l'état d'avancement du système après 30 secondes d'exécution. Les images (d) et (e) montrent l'état final de la segmentation. La zone non-segmentée dans la précédente image a, cette fois, été trouvée.

2.4.4 Les région utilisent les contours

2.4.4.1 La motivation

Le mécanisme décrit au paragraphe 2.4.3 permet aux processus de type contour de solliciter la création de processus de type région afin de faire émerger de l'information dans le voisinage du contour. Le système, dans son état actuel présente un biais puisque les processus de type région sont exploités dans une relation de type maître-esclave par les processus de segmentation de type contour. Dans un soucis de rééquilibrage des influences, un processus de type région doit pouvoir à son tour déléguer des tâches à des processus de type contour. Le système ne présente plus ainsi de lien de dépendance entre des méthodes qui soit statiquement codé. Chaque processus dépend d'un processus père qui l'a créé, et dispose lui-même de la possibilité de créer d'autres processus fils.

Cette création de processus contours doit répondre à un besoin précis. La littérature a insisté sur le fait que les segmentations de type région présentaient des défauts de localisation : la construction d'une région se fait sur les critères photométriques [24], et toute notion spatiale est retirée des critères de décision. En conséquence, les frontières [25] de ces régions sont déchiquetées et irrégulières.

Des détections de contours opportunes sont adaptées pour répondre à ce problème de localisation de la frontière de la région. Ces contours vont mettre en évidence des éléments constituant le contour réel [26] de la primitive sur laquelle travaille le détecteur de région. La simple existence de ces contours agit comme autant d'obstacles qui ont pour effet de bloquer la progression de la région au-delà de ces contours.

L'objectif est atteint si la construction de ces contours est suffisamment anticipée pour permettre de mettre en évidence la limite de la primitive à segmenter avant que le processus de construction de région ne risque de déborder vers un autre objet voisin de l'image aux caractéristiques photométriques proches. Il en résulte une forte dépendance entre le processus région et ses processus contours :

- Une dépendance temporelle d'abord, puisque le processus région doit initialiser ces contours suffisamment tôt par rapport à l'avancement de sa propre région, pour que les contours puissent travailler avant qu'un éventuel débordement ne se produise. Si un débordement a lieu, la primitive contour ne pourra plus être obtenue, puisque les pixels situés sur la transition auront été étiquetés à la région auparavant.
- Une dépendance topologique ensuite, puisque le processus région est le seul à pouvoir les placer judicieusement. Leur lieu d'initialisation devant se situer quelques pixels au delà de la frontière courante de la région. Empiriquement, cette initialisation repose sur l'hypothèse qu'il ne manque plus à la région courante qu'une couronne de quelques pixels d'épaisseur afin d'épouser la limite réelle de la primitive à segmenter.

La figure 2.13 illustre le but recherché. Le contour tracé en noir, créé par la région antérieurement, localise précisément l'emplacement de la discontinuité, et forme une limite empêchant la région de déborder au cours de son expansion. La figure 2.14 fournit une illustration de cet effet. L'exemple choisi est une partie de l'image des cellules musculaires. Cette partie contient

[24] On compare le niveau de gris des pixels du voisinage au niveau de gris moyen de la région, et on choisit le pixel avec la plus proche valeur. Une alternative consiste à travailler sur les mesures de l'écart-type local.

[25] au sens de *boundary* en anglais, le contour fermé obtenu en chaînant les pixels périphériques de la région.

[26] Le terme de *contour réel* est ici à opposer au terme de *frontière de la région.* en construction. Le premier est une caractéristique photométrique statique de l'image. Le second est un paramètre dynamique lié à la construction de la primitive région.

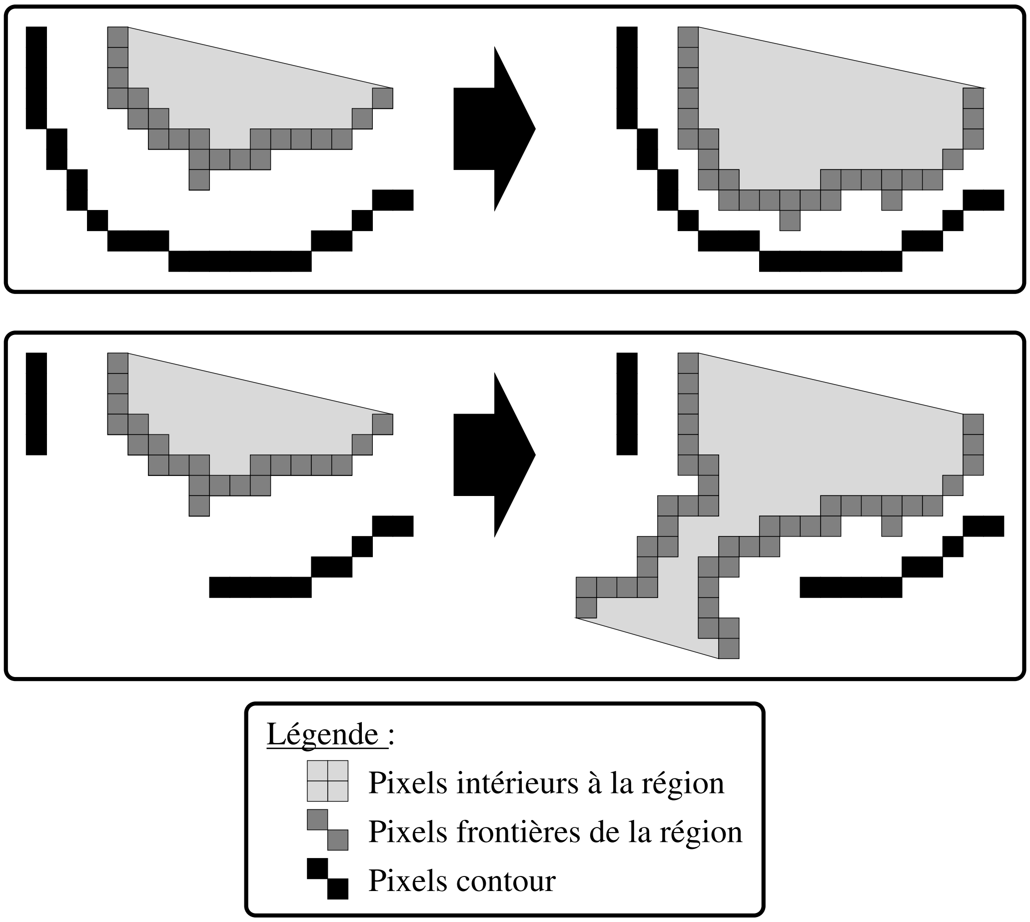


FIG. 2.13: Rôle des contours pour cerner l'expansion d'une région.

deux cellules noires séparées par un contour très faiblement marqué [27]. La première séquence d'images de (b) à (e) montre comment, à un moment donné de la croissance, la primitive déborde sur la cellule voisine. L'endroit où se produit le débordement n'a rien d'aléatoire, car la "fuite" de la région se réalise toujours à l'endroit de la frontière où la transition est la plus faiblement marquée. La seconde séquence d'images de (f) à (i) montre le même processus de type région, qui cette fois initialise des processus de type contour pendant sa construction. Le contour séparant les deux cellules noires forme alors une barrière infranchissable pour la région initiale, qui reste par conséquent limitée dans sa progression. Accessoirement, les germes contour ainsi initialisés ont segmenté davantage que la simple frontière de la région courante, puisqu'une bonne partie de la frontière de l'autre cellule noire a également été trouvée.

2.4.4.2 Création de nouveaux contours

Afin de prendre en compte les échecs potentiels de certains processus contours créés par la région, et afin de couvrir au mieux le voisinage proche de la région, plusieurs détections de contours sont initialisées à différentes étapes de la construction de la région. La construction de la région se fait par un cycle contenant un petit nombre d'itérations (5 dans notre implémentation), effectuant les actions suivantes :

1. Croissance de la région tant que des pixels candidats ont une *bonne* évaluation.

[27]Les deux cellules de cette partie bien spécifique de l'image forment un regroupement généralement appelé *la cellule papillon*.

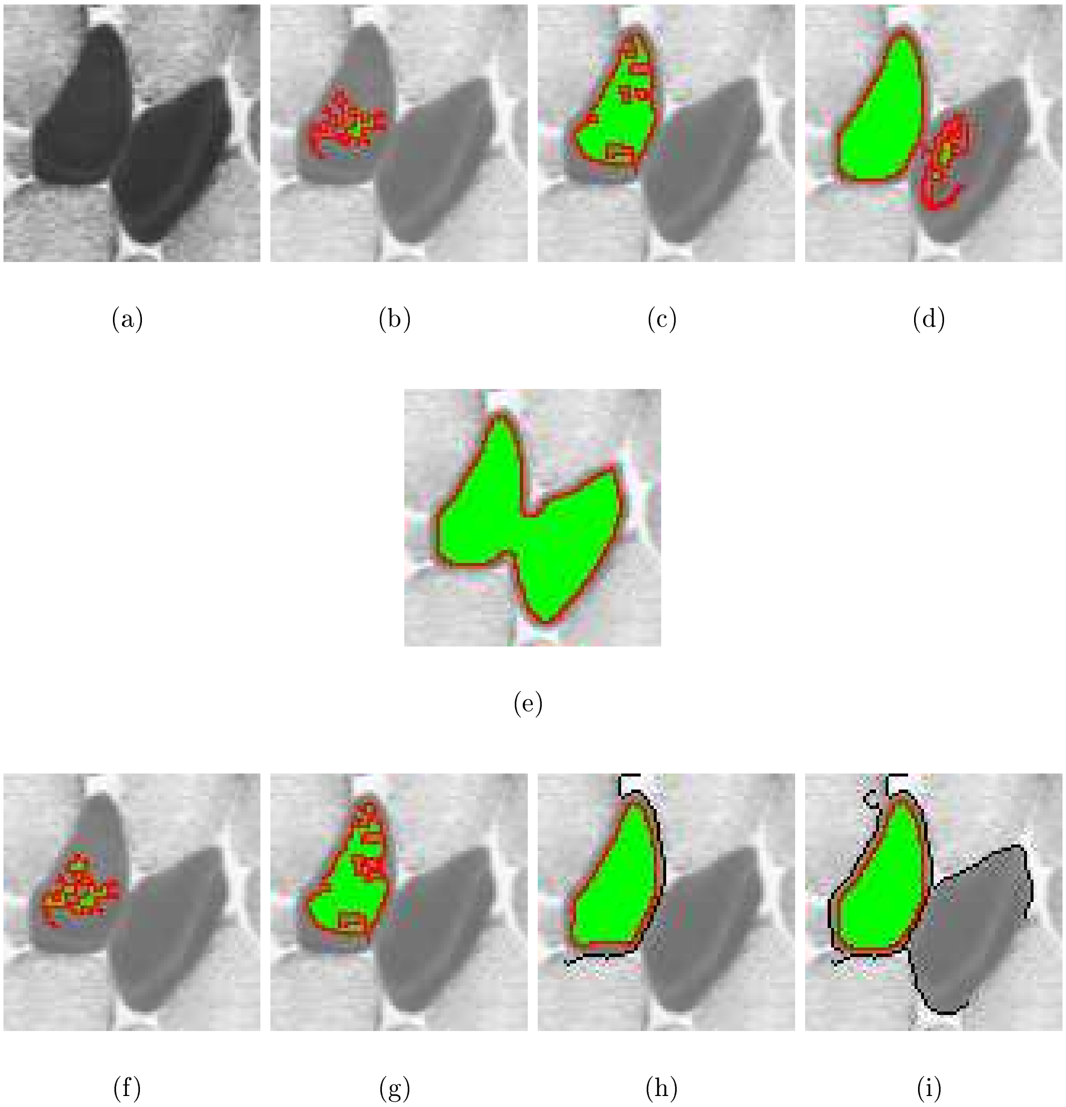


FIG. 2.14: Illustration de l'expansion d'une région contrôlée par la présence de contours.

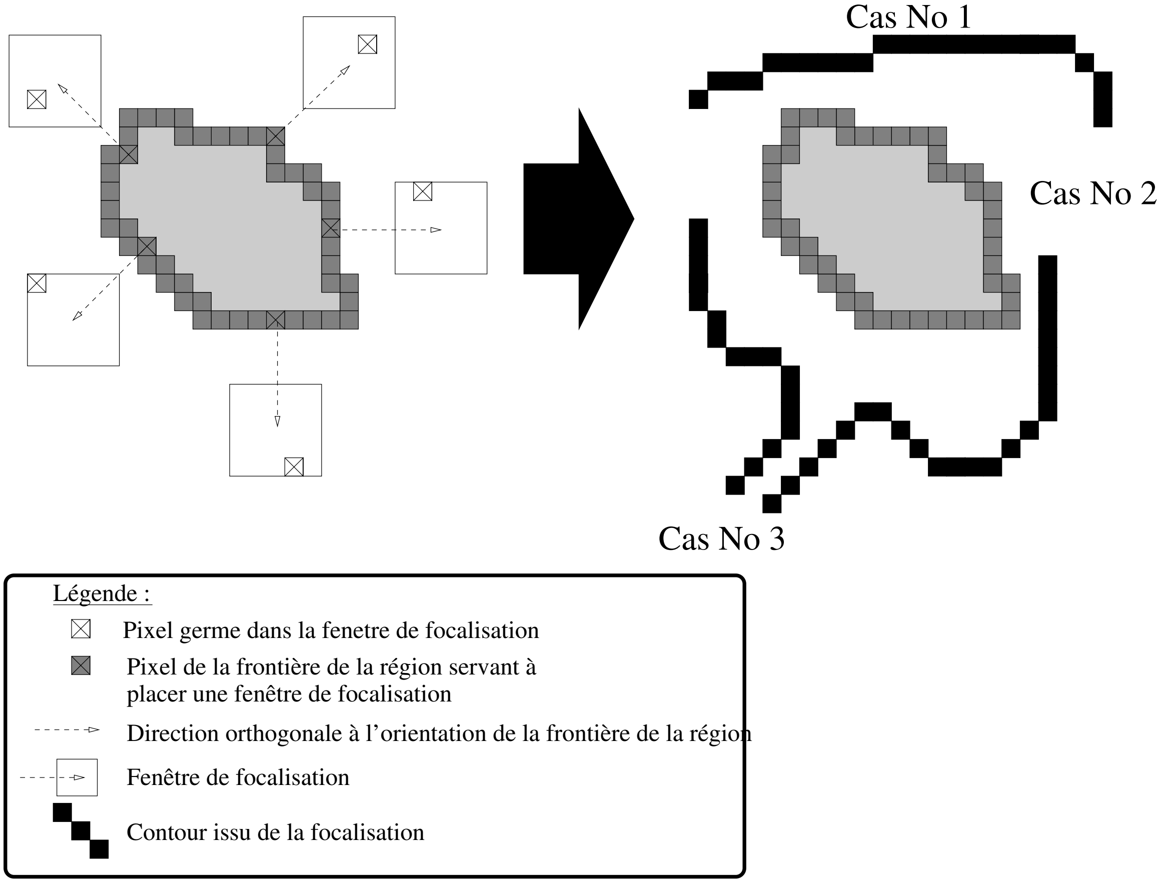


FIG. 2.15: Principe de la focalisation pour la segmentation d'une région.

2. Interruption de la croissance, lancement de processus de type contour dans le voisinage de la région. Mise en attente jusqu'à la terminaison de ces processus.

3. Ajustement des seuils utilisés par le processus de type région, permettant de relaxer le mécanisme de croissance, et d'incorporer éventuellement de nouveaux pixels.

La construction de la région se fait par vagues successives d'ajout de pixels. Entre chacune de ces vagues intervient une étape dite *de focalisation*, au cours de laquelle le processus de type région fait une pause dans sa construction, et génère un certain nombre de processus de type contour dans son voisinage proche. Ce voisinage évolue entre deux vagues successives d'ajout de pixels à la région. L'étape de focalisation suivante ne placera donc pas les germes des processus contours au mêmes endroits dans l'image qu'à l'étape de focalisation précédente. Cela garantit ainsi une progression constante dans le territoire ainsi exploré par les seuls processus contours issus de cette région.

La figure 2.15 présente un exemple de coopération entre le processus de type région et un certain nombre de processus contour fils. La frontière de la région a un double rôle pour le placement des fenêtres de focalisation.

- Un échantillonnage plus ou moins grossier des pixels de cette frontière permet de sélectionner un certain nombre de points, dont chacun va servir à déterminer l'emplacement d'une fenêtre de focalisation.
- La courbure discrète de la frontière à cet endroit permet de définir une direction pour l'emplacement de la fenêtre (flèches en pointillés sur la figure).

Le choix de ce pixel, et le calcul de cette direction ne garantissent pas que la fenêtre ainsi définie sera située en dehors de la région. On peut par exemple imaginer une région à la frontière très déchiquetée, contenant deux morceaux de frontière en vis-à-vis, ou bien encore une région présentant des trous. Les fenêtres de focalisation risquent alors d'être situées totalement ou partiellement à l'intérieur de la région. Ces cas particuliers apparaissent marginaux. Ainsi, sur le nombre de processus contour initialisés, l'échec d'un certain nombre d'entre eux n'est pas paralysant pour la viabilité de l'ensemble du système.

La région suspend son activité pendant l'exécution des processus contours fils. Lorsqu'elle redevient active, une configuration telle que celle présentée dans la partie droite de la figure 2.15 est espérée. Plusieurs cas peuvent être mis en exergue (voir la figure 2.15) :

Cas 1. La fusion de deux contours *frères*.

Cas 2. L'échec d'un des processus contour. La fenêtre de focalisation peut avoir été placée dans un endroit hors de toute zone contrastée de l'image.

Cas 3. Les contours obtenus marquent une transition avec une autre région que celle qui les a créés (changement de concavité du contour).

Cette énumération de cas montre que chaque processus de segmentation dispose d'une importante autonomie, et que peu de contrôle s'opère de la part du processus qui les a créés, mis à part leur lieu d'initialisation, et leurs seuils initiaux. Chaque processus apparaît davantage guidé par le besoin d'enrichir un environnement global, que par une dépendance rigide à un processus initiateur auquel il doit rendre des comptes. L'aide qu'un processus pourra tirer des autres méthodes qu'il va initialiser apparaît d'une certaine manière plus comme un “heureux concours de circonstances” que comme un élément indéfectible du système.

2.4.4.3 Les aspects unifiés de la méthode

Plusieurs points sont semblables à ceux déjà décrits dans le paragraphe 2.4.3. L'instanciation des processus fils, la récupération des informations segmentées par le processus père, ainsi que la gestion des fenêtres de focalisation alternatives sont des concepts pareillement applicables. Nous n'insisterons pas sur ces points.

2.4.5 La coopération et la localité

La coopération entre un détecteur de régions et un détecteur de contours repose sur l'idée simple que l'on connaît le lieu où l'on s'attend à trouver un certain type de primitive, en fonction de celles qui ont déjà été trouvées. Il s'agit donc ici de définir une contrainte *topologique* sur l'organisation des primitives qui constituent le système.

Les hypothèses réalisées dans le système sont :

- L'existence probable de deux primitives de type région de part et d'autre d'un contour.
- L'existence probable d'un réseau de contours entourant une primitive région.

Il est important pour les performances du système, et pour l'intérêt de la coopération, de pouvoir faire une hypothèse sur le lieu d'existence d'une primitive, et ceci avant même le début de sa construction. Le processus qui est créé s'avére “utile”, dans le sens où il construit effectivement une primitive à l'endroit où il a été créé. Un échec lié à un mauvais placement du germe fait perdre du temps au niveau du système en terme d'allocation de ressources, sans apporter d'enrichissement de la segmentation en contrepartie.

Cette hypothèse sur le lieu d'existence est facile à faire dans le cas des contours et des régions, car elle s'appuie sur la dualité entre ces deux types d'objets. Le cas général est plus délicat.

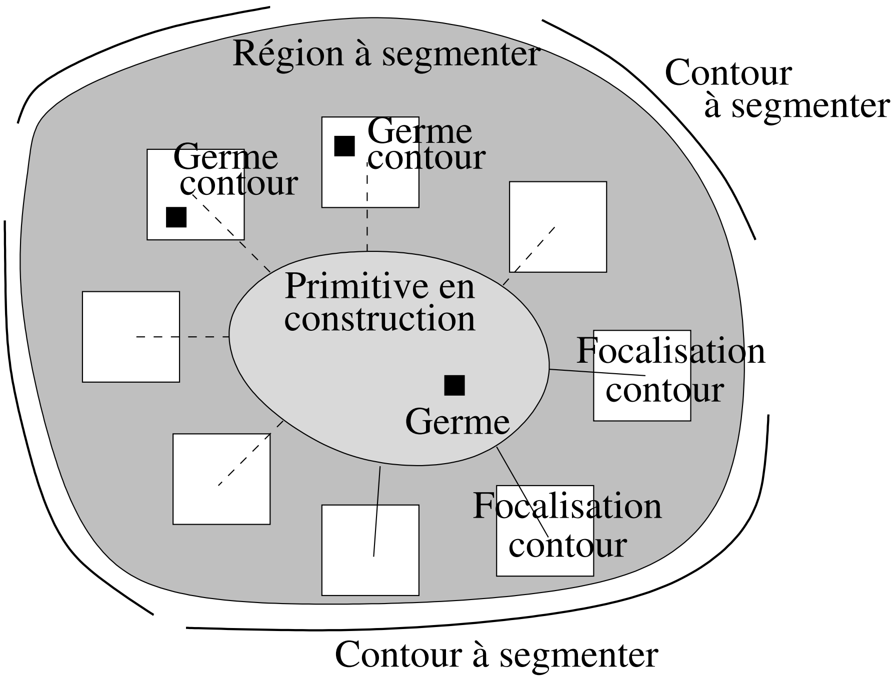


FIG. 2.16: Un exemple des limites de la focalisation.

Comment faire une hypothèse sur le lieu d'existence d'une primitive, sans lancer le détecteur de cette primitive ? Une possibilité est dans ce cas d'utiliser un algorithme plus simple, dont le coût d'application localement est plus intéressant que le détecteur lui-même. Citons plusieurs possibilités :

- Effectuer des mesures, et déterminer des critères simples sur l'environnement.
- Réaliser une version simplifiée du détecteur lui-même.

2.4.5.1 Les limitations

La limitation de la coopération apparaît lorsqu'il n'a pas été possible de faire cette hypothèse sur le lieu d'existence, ou bien lorsqu'elle a été faite de manière insuffisamment précise. Le processus de détection est alors créé, et les chances d'un résultat positif sont hypothéquées. La manière dont les processus de type région génèrent des contours en est une illustration.

L'exemple proposé par la figure 2.16 présente une région en cours de construction et dont les dimensions sont restreintes par rapport à la taille de l'objet à segmenter. Lorsque la focalisation intervient de manière trop anticipée, la plupart des fenêtres risquent d'être situées à l'intérieur de l'objet, là où il n'y a aucun contour à segmenter.

Notre hypothèse, que les contours cernent la région en cours de construction s'avére donc en ce sens trop générique, puisqu'elle ne tient pas compte du fait que la frontière de la région en construction peut être très éloignée de la frontière de la zone à segmenter. Cependant, sa facilité d'implantation, et la relative difficulté pour ajouter des critères de décision plus précis sans alourdir démesurément les temps de calcul nous ont poussés à conserver ce principe, malgré cette faiblesse.

2.4.5.2 Les extensions

Des hypothèses plus élaborées peuvent être introduites, permettant de cibler davantage la topologie de l'environnement que l'on segmente. Nous en donnons ici quelques exemples, qui

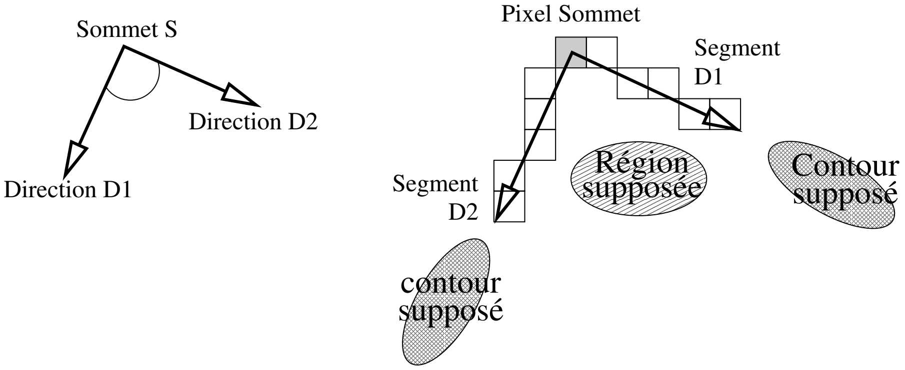


FIG. 2.17: Extension à d'autres types de processus. Exemple d'un détecteur de coins.

n'ont pas été implantés actuellement.

En se plaçant dans le cas du processus de type région qui génère des contours à sa périphérie, une évaluation qualitative de la photométrie de la région [28] peut aider à cibler le type de contour que l'on s'attend à trouver à la périphérie de la région. Plus la région est sombre, plus il est probable que le contour marquant la séparation de la région avec le reste de l'environnement sera de type *marche* avec une transition de type *sombre* vers *clair*, ou bien de type ligne avec une transition de type *sombre* vers *clair* vers *sombre*. Une telle banalité permet pourtant de typer efficacement l'orientation du gradient sur le contour recherché, et ainsi d'éliminer d'éventuels faux contours n'ayant pas ces caractéristiques.

Dans le même ordre d'idée reposant sur l'affinage des hypothèses par des indices photométriques, il est raisonnable de supposer qu'un contour de type *ligne* [29] sépare deux régions ayant une intensité des niveaux de gris proche. Ainsi la segmentation d'une des deux régions bordant ce contour devient un indicateur sur le type de région à chercher de l'autre coté du contour.

D'un tout autre genre, une extension possible de la coopération consisterait à spécialiser d'autres types de processus. Un détecteur de contours devient alors spécifique à certaines formes, par exemple un détecteur de *coins*. Définissons le coin comme un pixel sommet S, et deux segments de quelques pixels D_1 et D_2 partant de ce sommet, voir la figure 2.17.

En supposant la réalisation d'un détecteur de telles primitives, consistant juste à placer correctement le pixel définissant le sommet, et quelques pixels de part et d'autre du sommet indiquant les prémisses de ses deux segments, il est clair qu'une coopération efficace avec les détecteurs précédemment décrits passera par la recherche de deux contours *classiques* situés dans le prolongement de l'extrémité des deux segments D_1 et D_2, ainsi que d'une région située entre ces deux segments.

Inversement, si l'on désire conserver l'idée d'un système où chaque méthode peut indifféremment utiliser d'autres méthodes, et être utilisée par d'autres méthodes, il faut alors se poser la question de la manière dont les processus régions et contours pourraient utiliser un tel détecteur de coins. Ce second aspect de la coopération n'est pas trivial. Les détecteurs classiques régions et contours doivent pouvoir déterminer avec une forte probabilité l'endroit où initialiser de tels détecteurs de coins. Le but est d'éviter de lancer *à l'aveugle* des détecteurs de coins dans des zones où il n'y a rien à détecter. A la différence des contours et des régions, qui peuvent s'appuyer sur leur complémentarité, rien ne permet de supposer l'existence d'une configuration de type coin quelque-part dans l'image avant d'avoir lancé le détecteur proprement dit. Une solution

[28] C'est-à-dire pouvoir évaluer s'il s'agit globalement d'une région claire ou d'une région sombre.

[29] le profil des niveaux de gris à l'aspect d'un V, à l'endroit ou à l'envers.

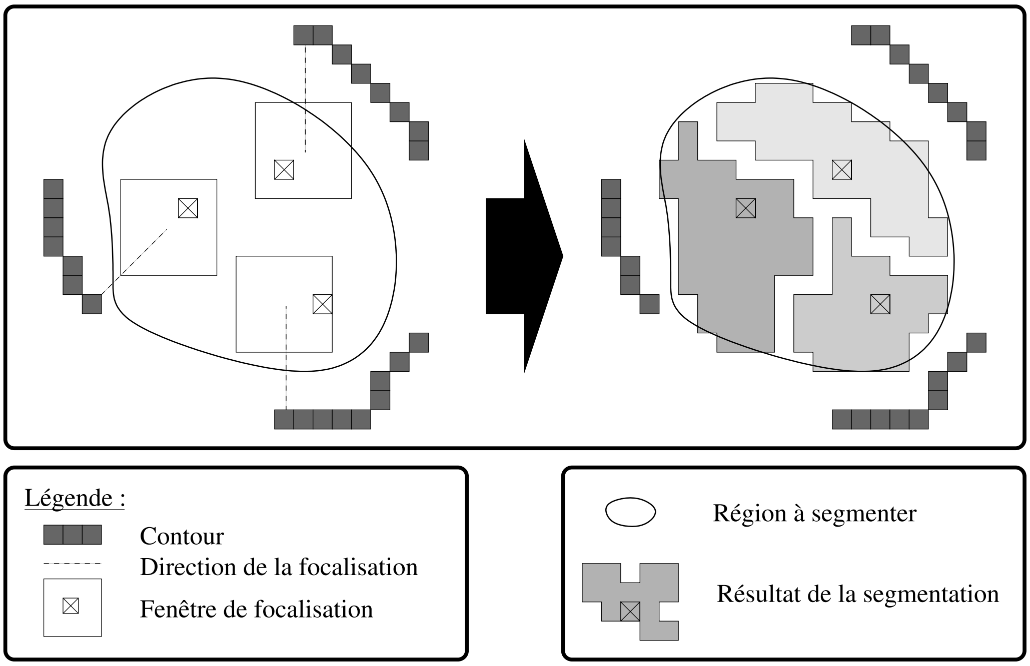


FIG. 2.18: Plusieurs processus de segmentation de type région lancés dans une zone correspondant à un même objet.

serait alors d'invoquer la détection des coins *par défaut* de la méthode générique. Le détecteur spécialisé est activé lorsque le détecteur générique a échoué dans un endroit donné. Cette approche ne permet cependant pas de décider, parmi plusieurs types de détecteurs spécifiques, lequel activer lorsque la méthode par défaut échoue.

2.4.6 La coopération par la fusion de données

2.4.6.1 Motivation

La fusion de primitives est un moyen de régler les conflits entre plusieurs processus de segmentation, segmentant conjointement la même primitive. Sans faire intervenir les notions d'accès concurrents aux données [30], il est probable que plusieurs segmentations de type région soient initialisées à différents endroits d'un même objet de l'image.

La figure 2.18 illustre ce cas. Des détecteurs de contours créent conjointement des germes de détection de région au cœur d'un objet de l'image, qui s'avére constituer une même entité à segmenter. Chaque processus région dispose d'une vision limitée de son environnement, et n'a donc pas la possibilité de remarquer la présence des autres segmenteurs autrement que par un effet de bord : une région déjà constituée empêchera la progression d'une autre région sur ses pixels déjà étiquetés. Les conflits d'appartenance de pixels ne sont en effet pas gérés [31]. Un

[30]Cela signifie qu'un processus dans notre implémentation distingue *immédiatement* qu'un pixel est étiqueté par un autre processus. Il ne peut donc pas se produire d'état instable, dans lequel un pixel serait étiqueté simultanément par plusieurs processus, si l'information de mise à jour ne circulait pas assez vite entre les processus, ou bien si l'étape d'étiquetage n'intervenait pas en condition d'exclusion mutuelle.

[31]Une approche visant à retarder la prise de décision pourrait substituer au principe de partitionnement de l'image celui de recouvrement, autorisant ainsi un pixel à appartenir à plusieurs classes. L'intérêt est de ne pas engager trop le processus de segmentation dans un mécanisme de décision, en utilisant des notions proches de celles de la logique floue, afin de faire remonter au niveaux supérieurs les moyens de prendre cette décision selon le contexte. L'inconvénient majeur qui apparaît est la mise sous condition des traitements ultérieurs, ainsi que

pixel est rattaché à la première primitive qui l'étiquète. Sur l'exemple de la figure 2.18, chaque segmenteur de région progresse donc indépendamment dans l'image. Chacun va "conquérir" une partie du territoire à partir du pixel d'où il a été initialisé. La différence par rapport à la présence d'un seul germe sera que chaque région sera gênée dans sa progression par les frontières fictives imposées par les deux autres régions [32].

La segmentation finale présente donc à l'utilisateur un objet constitué de plusieurs régions élémentaires, ce qui s'apparente à un cas de *sur-segmentation* de l'image. Il ne s'agit pas d'une erreur de segmentation à proprement parler, puisque d'une part aucune des primitives régions ne couvre deux objets à la fois, et d'autre part l'objet à segmenter a été identifié dans son intégralité. Cette sur-segmentation constitue davantage un handicap pour les actions de reconnaissance ultérieures, pour lesquelles il est plus séduisant de s'appuyer sur l'hypothèse que la segmentation de bas niveau fournit une seule primitive par objet.

Le système que nous proposons doit donc s'efforcer de minimiser cette sur-segmentation, effet de bord non désiré de l'implémentation, en fusionnant les primitives se rattachant manifestement à un même objet. Une approche globale consisterait à effectuer un traitement postérieur à l'exécution du système, visant à fusionner les primitives *topologiquement* proches, et présentant des similarités, tant pour les régions que les contours. Cette approche présente l'avantage de la modularité des traitements, puisque l'on sépare les étapes de détection et de fusion, et a d'ailleurs séduit la plupart des chercheurs dans le cadre de la segmentation en contours. Une phase de chaînage, c'est-à-dire de fusion, suit généralement la phase de détection.

La fusion des primitives constitue une simplification du système de segmentation en terme de ressources utilisées, puisque l'on réduit le nombre des processus qui travaillent sur l'image, ainsi que le nombre des primitives [33]. Ainsi, pourquoi faudrait-il attendre la terminaison du système pour bénéficier de la simplification induite par les fusions pouvant avoir lieu entre des primitives ? Il semble plus intéressant de gérer les fusions entre primitives – et donc les fusions entre les processus associés – dynamiquement, c'est-à-dire dès que cela s'avére possible pendant l'exécution du système. Une telle fusion dynamique respecte ainsi le principe de fonctionnement opportuniste des processus, ainsi que la délégation des tâches, qui fait de chaque processus un centre de décision autonome, y compris dans le cadre de la fusion évoqué ici.

Les paragraphes qui suivent vont s'attacher à décrire les protocoles de fusion proposés dans le cas des primitives régions et contours, leurs particularités, leurs ressemblances et leurs différences.

---

la complexité qui en découle lorsque les décisions à prendre dépendent des décisions prises antérieurement. Le chaînage des pixels contours en est un exemple. Le choix de retenir un pixel plutôt qu'un autre à une intersection conditionne laquelle des deux jonctions va être suivie dans la suite de la construction.

[32]Un tel partage homogène de l'objet à segmenter réussit grâce à un séquencement efficace des processus régions, dans le cadre d'une exécution en pseudo multi-tâches du système. Aucun ne doit avoir une durée d'exécution prépondérante afin que chacun puisse segmenter une zone de taille comparable. Si deux processus de même nature ont des durées d'exécution comparables, il n'en est pas de même entre les processus de type contour et région (la construction des régions est beaucoup plus lourde en nombre de pixels à étudier). Cela n'est pas pénalisant dans une implémentation sur une architecture mono-processeur, où un unique processeur effectue l'ensemble des traitements, quelles que soient les conditions de synchronisation entre ces traitements. La prise en compte de la vitesse relative entre des tâches filles prend tout son intérêt sur une architecture parallèle, où un processus qui attend la terminaison de ses fils se traduit par un processeur qui ne fait plus rien, si la répartition des tâches sur les processeurs est trop statique.

[33]Rappelons que chaque processus est associé à une primitive dès sa création, et que la primitive *survit* – en termes de ressources utilisées – après la terminaison du processus.

2.4.6.2 La fusion des contours

Le terme générique de *fusion* appliqué au cas des contours se rapporte ici au chaînage, intervenant lorsque deux extrémités, aux propriétés similaires peuvent soutenir l'hypothèse empirique suivante : ces deux extrémités sont suffisamment proches et suffisamment similaires, pour que l'on admette qu'elles marquent une même transition dans l'image. Cette transition unique a subi localement une perte de qualité qui n'a permis à aucun des deux détecteurs, ayant travaillé de chaque coté de cette perte de qualité, de poursuivre.

La fusion peut donc permettre ici de résoudre un problème de localité. Un détecteur de contours seul n'a généralement pas un "champ de vision" suffisant pour pouvoir détecter ce genre de configurations, qui demandent d'explorer davantage qu'un simple masque de pixels 3×3 ou 5×5. Plus précisément, son champ de vision limité est suffisant dans la plupart des cas : transition évidente, fort gradient, pas de jonctions, peu de bruit. Mais il s'avére limité dès que des configurations plus complexes apparaissent. Ces configurations sont bien souvent très marginales dans l'image, mais ces dernières doivent cependant recevoir un intérêt tout particulier, car c'est de leur traitement que dépendra la qualité d'un détecteur de contours. *Un détecteur de contours trouve toujours les contours évidents.*

La focalisation faisant émerger les primitives du voisinage offre un cadre idéal pour envisager leur fusion. Le contour a suspendu sa construction, et a créé des processus de type région sur les cotés de son extrémité, ainsi qu'un processus de type contour dans le prolongement de son extrémité. Le contour fils obtenu peut éventuellement être fusionné au contour qui avait suspendu sa construction. L'objectif est alors atteint : un détecteur de contour ne peut plus progresser car la transition n'est plus clairement marquée. Le processus de type contour ne cherche pas à étendre son propre champ de vision, il délègue au contraire cette tâche. Un nouveau contour est initialisé au delà de l'extrémité, là où la transition est à nouveau évidente. Le nouveau contour est construit. Il peut être fusionné au contour initial qui l'avait créé : quelques pixels sont éventuellement ajoutés pour combler la distance existant entre les deux extrémités. Le chaînage devient ainsi récursif : un contour se suspend, crée un autre contour, qui se suspend un peu plus loin, qui crée un autre contour, etc.

Ce principe permet donc :

- un apport de connaissance globale – existence ou pas d'un contour dans le prolongement des extrémités – permettant des décisions locales motivées d'un processus de type contour, sans avoir besoin d'augmenter le champ de vision de ce dernier, ce qui serait d'une part très pénalisant en termes de performances, et d'autre part inutile dans la grande majorité des cas, lorsque la transition est clairement marquée.
- un chaînage dynamique des contours, intervenant au sein même du mécanisme de construction.
- un allégement des ressources mises en œuvre dans le système à chaque fusion effectuée.

2.4.6.3 La fusion des régions

La fusion fait ici intervenir deux primitives de type région. L'objectif d'une fusion entre deux régions est de regrouper dans une même entité deux ensembles de pixels, inclus auparavant dans deux primitives distinctes, lorsque ces régions possèdent des propriétés photométriques communes, laissant à penser qu'elles correspondent au même objet.

La fusion entre deux régions est abordée différemment de la fusion entre deux contours. La fusion de contours s'effectue entre la primitive d'un processus en cours de fonctionnement, et une autre primitive contour issue de la création antérieure d'un processus fils. Le mécanisme de coopération consiste pour le processus père à attendre la terminaison de ses fils pour redevenir actif. Cette autre primitive contour n'est donc plus rattachée à un processus en cours de fonctionnement au moment où le processus père tente la fusion. Il peut donc fusionner cette primitive à la sienne sans spolier un autre processus.

Cette contrainte de filiation n'existe plus dans le cas général de la fusion, et en particulier dans la fusion de deux primitives de type région. Un processus de type région souhaitant fusionner sa primitive avec une autre doit donc envisager le cas où la primitive cible est rattachée à un autre processus de type région actif. Cela signifie par ailleurs que la région cible est encore en construction, dans un état non figé. Il convient alors de distinguer deux cas, suivant que cette région cible est en cours de développement ou pas.

Le mécanisme de fusion doit pouvoir s'intégrer à moindre coût dans l'algorithme de construction incrémental actuel. De plus, il est nécessaire de définir les contraintes à satisfaire pour les régions potentielles.

- La construction incrémentale de la région se fait par une boucle contenant :
  1. Une étape de construction élémentaire, effectuée jusqu'à épuisement de la liste des pixels candidats.
  2. Une étape dite de *focalisation*, au cours de laquelle des processus contours fils sont initialisés.
  3. Une étape où les seuils de la fonction d'évaluation utilisée dans l'étape **1.** sont recalculés de manière à pouvoir ajouter de nouveaux pixels à la région.

  La capacité de fusion, apparaissant comme une perception supplémentaire de l'environnement [34], il est nécessaire de définir le moment, au cours du fonctionnement de la méthode, où cet acte de perception va avoir lieu. L'endroit où cette capacité de perception est insérée dans la méthode influe sur sa fréquence d'activation. Cet acte de perception sera introduite, sous la forme d'une étape supplémentaire – la quatrième –, afin de ne tenter des fusions qu'après l'achèvement de l'étape de construction élémentaire, et d'éventuelles focalisations de type contour. Les tentatives de fusions se feront ainsi dans un environnement image enrichi.
- Le critère de choix de la région cible, avec laquelle une fusion va être tentée, porte sur la longueur de la frontière commune entre la région initiatrice et la région cible. La région cible retenue est celle présentant la plus grande fontière commune. La fusion s'effectue alors si des critères auxiliaires relativement stricts sont satisfaits (norme du gradient faible sur cette frontière commune, faible différence entre les niveaux de gris moyen des deux régions en présence, ...). Le succès de cette fusion permet éventuellement d'itérer à nouveau ce même mécanisme, tant qu'il existe une région à la plus longue frontière commune satisfaisant les critères auxiliaires, qui seront donnés par la suite.

La région cible n'est plus en construction : elle n'est plus rattachée à un processus. L'étape de fusion pour le processus initiateur se réduit simplement à l'annexion d'un ensemble de pixels. La figure 2.19 montre un exemple d'un tel comportement. Supposons que la région 1a est en cours de croissance, et que les deux régions voisines 2a et 3a sont terminées. Ces dernières seront appelées *régions mortes* par la suite. Après chaque étape de croissance élémentaire, la région 1a tente de fusionner avec les régions mortes de son voisinage, en autant d'itérations que nécessaire, c'est-à-dire tant que des régions satisfaisantes sont trouvées dans le voisinage. Les trois schémas de la figure 2.19 représentent ainsi deux fusions consécutives ayant eu lieu entre deux étapes de croissance élémentaire. Lors de la

[34] une perception de l'environnement, non plus au niveau du pixel, mais au niveau de la primitive.

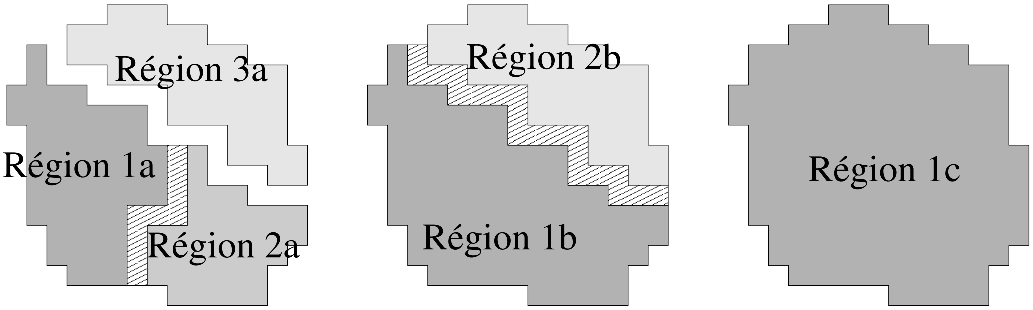


FIG. 2.19: Fusion itérative classique.

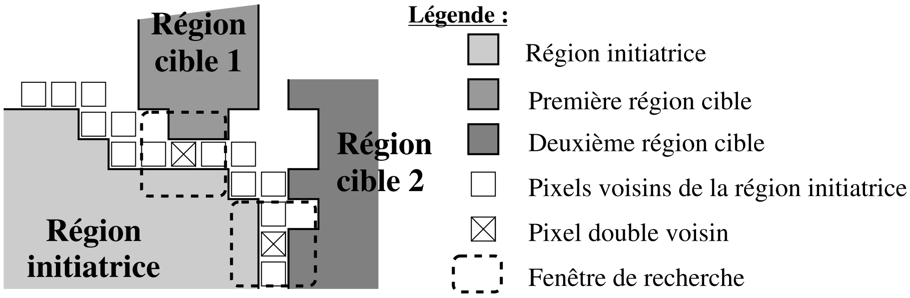


FIG. 2.20: Fusion itérative classique.

première tentative de fusion, la région 1a dispose de deux régions avec lesquelles elle peut potentiellement fusionner. La région 2a est choisie. La fusion consiste ici à :

- Ajouter les pixels de la région 2a à ceux de la région 1a.
- Ajouter les pixels du voisinage situés entre la région 1a et 2a. Ces pixels sont voisins des deux régions, ils doivent donc être intégrés à la région résultante. L'existence de ces pixels résulte d'un choix d'implantation, dans lequel un pixel ne peut pas avoir le double statut de pixel étiqueté *région* et de pixel étiqueté *voisin* pour une autre région.
- Mettre à jour les informations photométriques de la région 1a.
- Changer les références faites à la région 2a dans le système, pour les transformer en références à la région 1a.

La région 1a, devenue la région 1b sur la figure 2.19, peut ensuite tenter une nouvelle fusion avec les autres régions restantes de son voisinage. Dans cet exemple, il ne reste plus que la région 2b.

La phase de recherche de région cible se fait en deux temps. Le premier temps consiste à répertorier les régions potentielles du voisinage. Ceci pourrait être effectué par un graphe d'adjacence, mais cette structure centralisée ne serait pas compatible avec notre approche décentralisée. Chaque région parcourt donc sa liste de pixels voisins, et repère ceux qui ont le statut de *double voisins*, c'est-à-dire ceux qui sont voisins de plus d'une seule région. La figure 2.20 présente un tel exemple. Les carrés constituent les pixels voisins de la région *initiatrice* : la région qui cherche à fusionner. Les pixels marqués d'une croix sont des exemples de pixels faisant partie du voisinage de deux régions distinctes. Le parcours de tous les voisins, et la récupération de cette information sur le double voisinage permettent de construire une liste des régions voisines de la région initiatrice.

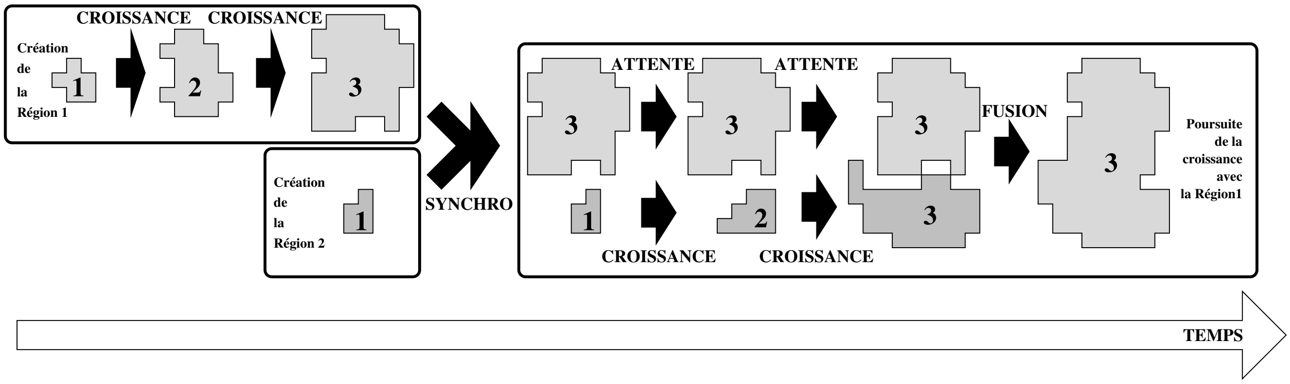


FIG. 2.21: Fusion entre deux régions simultanément en construction

Le deuxième temps consiste à choisir la meilleure région pour la fusion. La région privilégiée sera celle qui a la plus de pixels voisins en commun avec la région initiatrice [35]. Le gradient moyen est calculé sur ces pixels voisins communs. La fusion sera effectuée si les critères suivants sont satisfaits :

- La frontière commune est suffisamment grande. Empiriquement, cela consiste à vérifier que le nombre de pixels voisins en commun est supérieur à un seuil L, fixé dans l'application à $L = 10$.
- Le gradient moyen sur cette frontière commune n'est pas significatif. Une première comparaison par rapport à un seuil absolu $S_{abs} = 2$ est faite, s'appuyant sur les critères de visibilité des contours de Salotti [Sal94]. Dans le cas où le gradient est supérieur à ce seuil absolu de visibilité, un autre seuil, relatif celui-ci, est appliqué. $S_{rel} = \alpha\frac{1}{2}(\overline{G_1}+\overline{G_2})$, déduit du gradient moyen des deux régions (moyenne du gradient des deux régions). Empiriquement $\alpha = 2$ est utilisé dans l'application. La contrainte obtenue peut sembler faible. Dans la pratique cependant, le gradient moyen sur toute une région depasse rarement la norme de 10 ou 15, ce qui conduit à une contrainte tout à fait raisonnable de 20 ou 30 pour le gradient des pixels frontières. Ce coefficient α traduit donc approximativement la différence d'ordre de grandeur supposée entre le gradient à l'intérieur d'une région et le gradient sur sa frontière.
- Un autre critère de fusion porte, non plus sur la qualité de la transition, mais sur la comparaison entre les deux régions à fusionner, pour lesquelles la moyenne des niveaux de gris doit être relativement proche. Cela permet entre autres de ne pas fusionner un fond homogène avec un dégradé, comme dans l'exemple de l'image de synthèse de la figure 5.17 du chapitre 5. Empiriquement, la moyenne des niveaux de gris des deux régions candidates à la fusion ne devront pas différer de plus de 15.

Ces critères sont choisis de façon empirique, leur but étant de rester relativement restrictifs pour ne pas fusionner abusivement. Si ces trois critères sont satisfaits, la fusion des deux régions s'effectue.

La région cible est toujours en construction : cela signifie qu'un processus de segmentation est attaché à la région cible, et que celle-ci poursuit sa croissance. Du point de vue de la région initiatrice, il est préférable de fusionner ayant une région avec un niveau de con-

[35]Cela signifie visuellement que la région retenue est celle qui a la plus grande frontière commune avec la région initiatrice.

struction avancé [36], ainsi la zone annexée est plus importante, les valeurs photométriques de la région cible sont plus fiables, et enfin, la tâche de segmentation restant à effectuer postérieurement à la fusion sera moindre.

Notons N, le nombre de phases de croissance déjà effectuées par la région initiatrice. La fusion avec une région cible sera envisagée lorsque la région cible aura effectué le même nombre de phases de croissance N, ou bien un nombre plus faible $N_2 < N$, lui laissant alors la possibilité de se *mettre à niveau par rapport à la région initiatrice*. Cette mise à niveau consiste à envisager la fusion entre deux régions qui auront chacune relâché leurs seuils un même nombre de fois, l'hypothèse sous-jacente étant que la fusion aura ainsi lieu entre deux régions de taille comparable d'une part et significative d'autre part. Le cas $N_2 \geq N$ n'a pas besoin d'être traité, puisqu'il pourra être pris en charge ultérieurement, si la région initiatrice et la région cible permutent leur rôle.

Ce protocole de négociation de la fusion introduit une nouvelle contrainte dans le système. Un processus doit effectuer un acte de communication [37] avec un autre processus, et l'objet de cet acte de communication ne peut pas être obtenu de manière indirecte, en utilisant par exemple l'environnement comme médium. Il faut donc envisager un moyen de communication direct entre les processus. La remontée d'information des processus fils vers leur père constitue une communication hiérarchique, permettant un flux unidirectionnel d'information entre des processus liés par la filiation. Une communication point-à-point directe entre les processus repose quant à elle sur des liens topologiques entre les processus, et à ce titre s'avére donc complémentaire du moyen de communication précédent.

Un protocole simple d'échange de messages en mode point-à-point est utilisé, et sera décrit dans le paragraphe 3.5.1. Chaque processus dispose d'une boîte à messages, et de la possibilité d'envoyer des messages à un (point-à-point) ou plusieurs processus (*broadcast*). Un processus destinataire d'un message pouvant être en attente de terminaison de ses fils, il faut donc gérer un mécanisme de signaux permettant de réveiller un processus en attente, afin qu'il puisse répondre à ces messages, avant de se remettre en attente immédiatement après. L'absence d'un tel mécanisme de réveil conduirait immanquablement le système dans un état de "dead-lock" [38] entre les processus. Il suffit pour cela qu'un processus actif – donc une feuille de l'arbre des processus – envoie un message à un de ses *aïeux*. Le processus feuille se met en attente de réponse, et plus aucun processus ne redevient actif sur cette branche de l'arbre des processus.

La sémantique des messages est fortement typée et dépend du contexte de l'application : des messages de synchronisation. L'objectif de cet échange de message est de permettre à la région fille de poursuivre sa construction, afin de se mettre au niveau de la région initiatrice en terme de nombre de phases de croissance réalisées. Par abus de langage, nous utiliserons le terme *la région* à la place du terme *le processus de type région* :

- La région initiatrice envoie à la région cible une requête de synchronisation, sous la forme d'un message, contenant le nombre de phases de croissance souhaité N.
- La région cible accepte ou pas la requête (elle peut elle-même déjà être en cours de synchronisation avec une troisième région, auquel cas, elle décline l'offre).

[36] Nous rappelons à cette occasion que la construction d'une région se fait par étapes de croissance successive, chacune d'elles étant séparée par une série de focalisations à la recherche de contours, puis par une phase au cours de laquelle les seuils appliqués à la fonction d'évaluation sont relâchés, de manière à pouvoir incorporer d'autres pixels. L'implémentation actuelle du système propose cinq étapes de croissance successives.

[37] Cet acte de communication avec un autre processus afin de proposer une synchronisation en vue d'une fusion de sa primitive sera appelé par la suite une *requête de synchronisation*.

[38] Etreinte fatale.

- Lorsqu'une réponse positive est enregistrée, la région initiatrice se met en attente, le temps que la région cible atteigne le nombre de phases de croissance souhaité N.
- Entre-temps, la région cible continue sa croissance. La différence entre l'exécution classique du processus région réside dans le fait que la région cible renonce à faire des focalisations de type contour lorsqu'elle est en cours de synchronisation, afin de terminer au plus vite sa croissance. En effet, la région initiatrice attend ses résultats, et même si elle n'est pas active dans le système, ses ressources existent et ne sont pas utilisées. Il convient donc de minimiser dans la mesure du possible le temps pendant lequel la région initiatrice est en attente de synchronisation.
- La région cible a terminé sa mise à niveau, envoie un message à la région initiatrice, et se termine. La région initiatrice peut tenter de fusionner avec la région cible.

La figure 2.22 représente la synchronisation entre deux processus de type région telle qu'elle vient d'être décrite. Les deux automates d'états finis correspondant à la région initiatrice (maître) et à la région en synchronisation (esclave) sont mis en vis-à-vis, de manière à identifier quels envois de messages correspondent à quelles réceptions pour l'autre processus. Il faut cependant garder à l'esprit que cette représentation est une version simplifiée de la réalité, dans le sens où un processus doit pouvoir indifféremment jouer le rôle du maître ou de l'esclave. Il doit donc posséder tous les états correspondant à ces deux statuts dans le code de son automate. La boîte en pointillé au cœur des deux automates de la figure 2.22 représente les états de base qui ne mettent pas en jeu de synchronisation ni de fusion. Le protocole permettant cette synchronisation est donc relativement indépendant du fonctionnement générique implémentant la croissance de région, et à ce titre aisément modifiable.

2.4.6.4 Les points délicats de la fusion

La synchronisation, puis la fusion de primitives posent des difficultés dans la gestion des ressources. Le plus gros problème à régler consiste à gérer le fait qu'un processus de segmentation n'est plus attaché de manière unique à une primitive image, qu'il a la charge de segmenter. L'allocation et la libération des nombreuses ressources mémoires demeurent donc une opération des plus délicates dans ce contexte, car une libération de mémoire ne peut intervenir de manière certaine que lorsqu'il n'existe plus aucune référence vers l'objet à supprimer. Cette nécessité s'avére difficile à gérer dans un environnement pseudo distribué comme celui qui est proposé.

D'autre part, les mises en attentes des processus sont toujours délicates à implanter, car à toute mise en attente doit correspondre un réveil effectué postérieurement par un autre processus, sous peine de voir se multiplier des processus de segmentation indéfiniment bloqués, à mesure que le système évolue. Cette contrainte peut être facilement traitée lorsque la seule cause de mise en attente dans le système est liée à la terminaison des processus fils. Dès lors qu'une cause supplémentaire, telle l'attente de synchronisation par exemple, vient s'ajouter, la gestion en devient plus délicate. Des solutions génériques seront proposées dans le chapitre suivant pour résoudre ces difficultés.

2.4.6.5 Conclusion

La fusion de primitives peut apparaître comme un raffinement non indispensable dans le cadre d'un système de segmentation de bas niveau, pouvant être laissé à la charge d'un algorithme indépendant, effectuant ce post-traitement ultérieurement. Pourtant, il nous a semblé intéressant d'intégrer cette fonctionnalité au cœur même du système, afin de mettre davantage

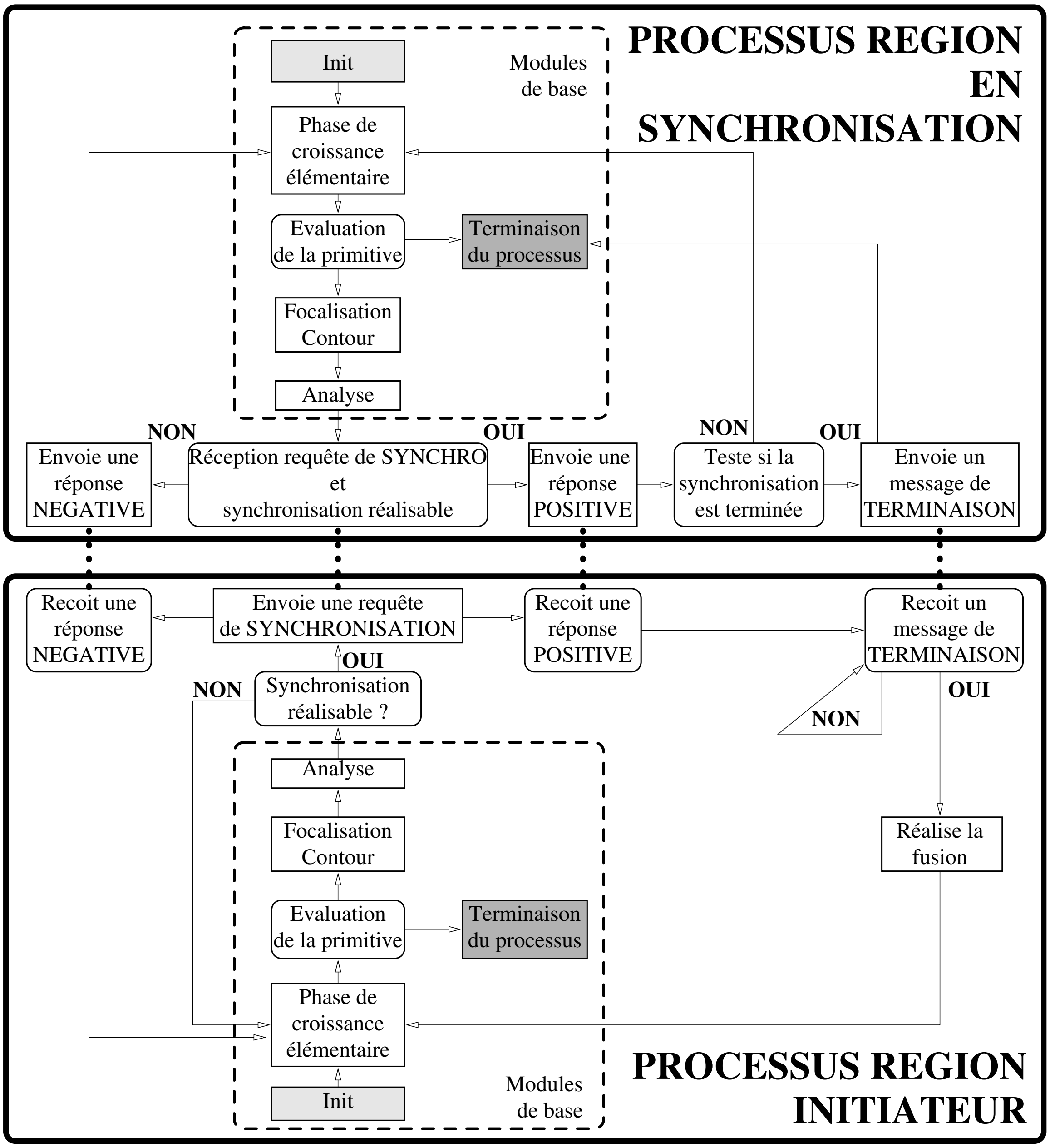


FIG. 2.22: La synchronisation entre deux processus de type région

l'accent sur l'aspect de communication locale entre les processus. Le système proposé ne propose pas seulement un ensemble de segmenteurs travaillant conjointement dans l'image, mais des entités pouvant appréhender leur environnement, et engager des actes de communication de manière autonome entre elles.

2.4.7 Conclusion

Cette partie s'est attachée à décrire l'implémentation d'une coopération efficace entre des processus hétérogènes de segmentation. Les points clés qui nous semblent généralisables à d'autres type d'approches sont :

- La gestion d'une coopération distribuée, dans laquelle aucun des acteurs ne dispose d'une vue et d'une connaissance globale sur le système, et n'a donc la possibilité de piloter le fonctionnement global de l'ensemble de tous les processus du système.
- La gestion d'une coopération locale, qui, à partir de la définition de contraintes très locales, mettant en jeu un nombre réduit de processus, permet par propagation de dégager une dynamique de groupe d'un point de vue global. La dynamique de groupe observée ici est la couverture progressive de l'image, jusqu'à son traitement complet, en ayant simplement défini une politique de délégation de tâches entre des processus de segmentation hétérogènes.
- La réalisation d'une coopération mutuelle, dans laquelle une méthode n'a pas d'ascendant *a priori* sur une autre. L'autonomie des agents de segmentation est un point fort, selon lequel l'enrichissement global de l'environnement, sous la forme de primitives image venant peu à peu s'ajouter à l'espace de travail des processus de segmentation, est prioritaire sur la satisfaction de l'objectif particulier d'un processus. La coopération apparaît alors davantage comme la gestion d'effets de bords judicieusement initiés.
- L'intégration de méthodes. Des mécanismes lourds et onéreux, tels la fusion ou le chaînage de primitives, sont généralement effectués sous la forme de *post-traitements* après la fin de l'algorithme de segmentation, et gagneraient pourtant à être intégrés au sein même du système de segmentation, pour peu que ce dernier offre suffisamment de points d'entrée dans ses mécanismes internes.
- L'utilisation de méthodes incrémentales est à nos yeux le point crucial qui soutend les arguments précédemment cités. L'incrémentalité est une des rares possibilités permettant d'exhiber un réel contrôle sur le fonctionnement des méthodes implémentées, et donc d'en exploiter les finesses.

Chapitre 3

Un système d'exploitation pour la vision

3.1 Introduction

Il peut sembler incongru de parler de système d'exploitation dans le cadre d'un système de vision. En effet ces deux domaines d'activités semblent radicalement éloignés l'un de l'autre. Le système d'exploitation est l'interface indispensable entre l'utilisateur (souvent exigeant) et le matériel informatique qu'il utilise (souvent intensivement), qui permet par une couche logicielle de bas niveau de lui présenter une représentation conceptuelle unifiée, quel que soit le type de matériel utilisé [Bou83], [Kra87].

Parmi les représentations conceptuelles permises par les systèmes d'exploitation, on peut par exemple citer celle des "ressources infinies" [Rus97]. La notion de tâche élémentaire (*job* ou encore *task*) permet de développer le concept de "nombre de processeurs infini". L'utilisateur peut lancer plusieurs tâches simultanément, sans s'occuper du nombre de processeurs présents sur sa machine. Charge est laissée au système d'exploitation de les séquencer (*scheduling*) en fonction des ressources en processeurs disponibles à un instant donné. Si la machine dispose de deux CPU ou plus, chaque tâche peut s'exécuter sur un processeur distinct, et dans le cas contraire, le système d'exploitation doit donc allouer une période de temps élémentaire à chaque processus pour que les deux tâches puissent d'exécuter sur le même processeur de manière entrelacée.

La mémoire supporte pareillement cette conceptualisation. Chaque tâche travaille dans son propre espace de mémoire linéaire, virtuellement infini, et (sauf exception) distinct de celui des autres processus. Le système d'exploitation doit effectuer la translation entre les espaces d'adresse virtuels des tâches et l'espace d'adressage réel de la machine.

Les ressources physiques étant rarement infinies dans le domaine de l'informatique, le rôle du système d'exploitation permet donc, dans un environnement multi-tâches et/ou multi-utilisateurs, de gérer les accès à des ressources limitées. Dans cette optique, les similarités commencent à apparaître avec un système de vision, dès lors qu'il est conçu selon des principes de coopération entre des méthodes hétérogènes fonctionnant de façon concurrente et distribuée dans la même image, selon la description qui en a été faite au chapitre 2 :

– La répartition des tâches sur un processeur se pose dans le cadre d'un système de vision, qui manipule plusieurs *processus* de segmentation, où chacun est chargé de la segmentation d'un objet de type région ou contour, où chacun travaille indépendemment et parallèlement aux autres [1], et où chacun fonctionne dans un environnement unique et limité

[1] Le chapitre 2 consacré à la coopération de méthodes montre que cela n'est pas toujours le cas

en termes de ressources (les données image). Les tâches d'un système d'exploitation vont donc correspondre aux processus de segmentation de notre système de vision.

- La gestion de la ressource mémoire pour les systèmes d'exploitation possède son pendant dans notre système de vision. Tout comme les tâches d'un système d'exploitation ont une représentation virtuelle de la mémoire qui leur est allouée [2], les processus de segmentation ont une représentation virtuelle de leur environnement de travail : l'image. Aucun n'a une vue complète de l'image, et tous travaillent localement, et de façon *relative* à l'emplacement de leur primitive en construction (choix des nouveaux points à explorer par exemple).
- Le système d'exploitation dispose de mécanismes permettant aux tâches de prendre conscience de leur coexistence. Les signaux et les mécanismes tels que les *System V IPC Mechanisms*, apparus dans la version de Unix System V en 1983, et contenant les sémaphores, les queues de messages, et la mémoire partagée, permettent d'implémenter des communications entre des processus. Un système de vision fondé sur des méthodes coopératives doit également permettre de telles communications, pour pouvoir réaliser la fusion de primitives, ou bien encore réaliser des synchronisations.

Nous nous proposons par ailleurs de spécialiser le concept de système d'exploitation :

- La vision par ordinateur permet de développer le concept d'agent de segmentation *situé*, dans le sens où un processus est topologiquement localisé par la primitive sur laquelle il travaille, et entre en contact avec d'autres processus sur des considérations spatiales : deux processus segmentant la même région fusionneront leurs résultats car ils sont voisins. Il est intéressant d'intégrer au niveau du système cette dimension spatiale : citons par exemple la notion de densité de processus de segmentation dans une zone de l'image, ou encore l'idée de focaliser l'exploration de l'image sur un endroit précis.
- Un processus de segmentation est assimilé à une tâche au sens du système d'exploitation. Tous les processus ont pour but commun de construire une primitive, avec ses attributs associés, afin d'enrichir l'image de connaissances de plus haut niveau. Il semble logique d'introduire un mécanisme de retour d'information au processus qui a créé un fils, l'objet de cette information étant précisément la primitive segmentée, dès lors que le fils s'est correctement terminé. La généricité des tâches dans un système d'exploitation ne justifie pas un tel lien entre le processus fils et le processus père : un simple entier de statut est renvoyé au processus père, ainsi qu'un signal d'indication de terminaison.

3.2 La notion de processus de segmentation

La question de l'implantation de notre système de vision s'est posée très tôt. Il se compose d'un ensemble de processus de segmentation, qui doivent fonctionner simultanément dans l'image, qui disposent d'une durée de vie limitée dans le temps, et qui doivent réaliser des actes de communication élémentaires. Ces trois principes énoncés établissent un lien immédiat avec des tâches au sens des systèmes d'exploitation. Il est donc logique de tenter d'implémenter ces processus de segmentation sous la forme de processus dans un système d'exploitation de type multi-tâches.

[2] Le système de mémoire virtuelle consiste à effectuer une conversion d'adresses entre la mémoire virtuelle, telle que la “perçoit” un processus, et la mémoire physique, telle que l'adresse le processeur. Ceci permet en particulier à tous les processus d'avoir leur espace de mémoire qui débute à la même adresse virtuelle, et qui apparaît constitué d'un seul bloc, alors qu'en mémoire réelle, l'espace de mémoire d'un processus peut être dispersé, dans des blocs non contigüs.

La principale question concerne la manière de faire exécuter en pseudo parallèle plusieurs processus de segmentation, avec les contraintes et caractéristiques suivantes :

1. Tous les processus de segmentation doivent accéder aux données fixes de l'image. De plus, ils effectuent régulièrement des mises à jour dans l'image résultat, par l'ajout de nouveaux pixels à leur primitive respective.

2. Un processus de segmentation est très consommateur de temps CPU (gestion de listes, tri de pixels, évaluation) lorsqu'il fonctionne de manière autonome.

3. Le nombre de processus de segmentation sur une image n'est pas limité arbitrairement, et dépend fortement des données. L'expérience montre qu'un cinquantaine de processus de segmentation simultanés en activité est fréquent pour une image 256×256. Ce nombre sera plus important sur une image plus grande si aucune contrainte de limitation n'est introduite.

4. La finesse de la granularité du séquencement n'est pas primordiale. Elle l'est pour un système d'exploitation, lorsque deux tâches de deux utilisateurs différents sont en compétition pour l'accès au processeur. Il n'est dans ce cas pas envisageable de bloquer un utilisateur totalement pendant plusieurs secondes au profit d'un autre. Les deux tâches doivent donner l'impression de s'exécuter continuement.

Les points 2. et 3. ne rendent pas l'utilisation de processus Unix très réaliste pour l'implantation de ces processus de segmentation. Aussi l'approche choisie est de simuler l'activité de plusieurs processus de segmentation au sein d'un unique processus Unix. Le point 1. est trivialement résolu, puisque un et un seul processus de segmentation s'exécutera à un instant donné dans notre système. Le processus Unix contiendra toutes les données de type image, et toutes les données de segmentation, ce qui résoudra donc la gestion des accès concurrents. Les points 2. et 3. ne posent plus de problèmes, puisque la machine qui supporte le système de segmentation n'aura qu'un seul processus Unix à gérer, quelle que soit l'activité interne de ce dernier.

La contrepartie à cette apparente facilité va résider dans la nécessité de programmer un séquenceur de tâches anonymes dédié à notre système de vision d'une part, et d'écrire des processus de segmentation supportant de s'interrompre régulièrement pour que d'autres puissent s'exécuter. Le point 4. dépendra donc de cette implémentation.

La programmation d'un séquenceur de processus au bas niveau permettra également de contrôler finement la manière dont les processus vont s'exécuter. En particulier, les endroits où les processus se suspendront seront clairement définis en tant que tels, et de ce fait les primitives sur lesquelles ils interrompront leur travail seront dans un état "stable" et cohérent [3].

3.2.1 Un automate d'états finis

Le choix a donc été fait de réaliser un système multi-tâches coopératif. C'est une différence majeure avec les systèmes d'exploitation évolués, qui pour la plupart utilisent un multi-tâches préemptif. La *coopération* signifie que le système d'exploitation attend que la tâche qui s'exécute sur le processeur se suspende *volontairement* (éventuellement par une mise en attente sur une entrée/sortie) pour faire exécuter une autre tâche. La *préemption* signifie que le système d'ex-

[3]Une interruption d'un processus de segmentation au moment où celui-ci remet à jour les listes de pixels voisins de sa région laisserait par exemple apparaître des incohérences d'étiquetage dans le voisinage pendant la période où le processus est interrompu. Des parties importantes du système nécessiteraient d'être exécutées en exclusion mutuelle, c'est-à-dire de manière non interruptible.

ploitation interrompt une tâche de manière autoritaire au bout d'un certain temps d'exécution (*time slice*) [4].

Du point de vue de la tâche utilisateur, la différence est importante. La préemption peut intervenir à n'importe quel moment, et est réalisée de façon transparente : le contexte de la tâche est sauvegardé. Dans le séquencement coopératif, le processus choisit à quel moment il va se suspendre, cela lui permet donc de "stabiliser" son contexte de travail avant de rendre la main au séquenceur [5].

Pour répondre à cet impératif, le code des processus de segmentation est séparé en plusieurs modules. Une variable d'état interne au processus permet de diriger l'ordre d'exécution de chaque module. Cette variable est modifiée à la fin de l'exécution d'un module, et définit ainsi le module suivant à exécuter pour ce processus. De façon pratique, le code d'un processus de segmentation est encapsulé dans une structure de contrôle de type `switch`, `case X:`, `case Y:` :

```
void ProcessContour (PtrData Data, ...)
{
   switch (Data->Stage) {
      case STAGE1 :
         /*
          * Initialisation du processus
          */
         ...
         Data->NextStage=STAGE2;
         Data->ExitCode =EXIT_OK;
         break;
      case STAGE2 :
         /*
          * Construction de la primitive
          */
         ...
         if (...) {
            /*
             * Terminaison du processus
             */
            Data->NextStage=NO_STAGE;
            Data->ExitCode=EXIT_OK;
         }
         else {
            Data->NextStage=STAGE3;
            Data->ExitCode=EXIT_OK;
         }
         break;
      case STAGE3 :
         /*
          * Focalisation
          * Création de processus fils
          */
         ...
         Data->NextStage=STAGE4;
         break;
      case STAGE4 :
         /*
```

[4]La manière la plus simple étant d'activer une interruption sur le timer (horloge interne).

[5]Les deux cas présentent à l'utilisateur une impression d'exécution parallèle des tâches, la seule différence réside dans le fait que le séquencement préemptif ne fait pas confiance aux tâches pour n'utiliser le processeur que pendant une courte durée. En effet, le séquencement coopératif n'assure pas une utilisation équitable du processeur, puisqu'il ne contrôle pas la durée pendant laquelle une tâche va l'utiliser.

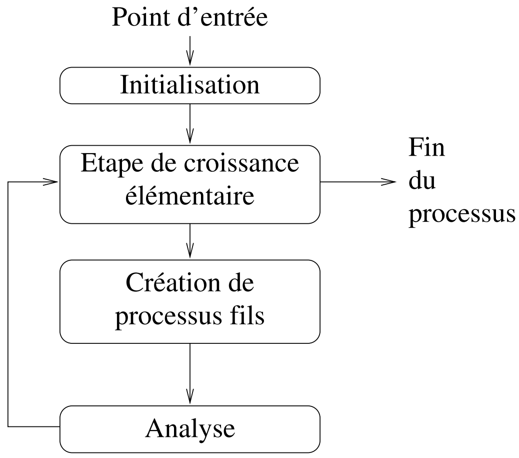


FIG. 3.1: Le schéma de contrôle élémentaire du processus Contour.

```
          * Analyse
          */
         ...
         Data->NextStage=STAGE2;
         break;
      default :
   }
}
```

Ce principe permet de disposer de plusieurs processus, ayant chacun une structure de données `Data` propre, et se situant à des niveaux d'avancement différents [6].

L'extrait de programme C précédent permet donc de diviser le code du processus de suivi de contour en un certain nombre de modules élémentaires, suivant le schéma de contrôle en figure 3.1.

Le code des processus peut également être vu sous la forme d'un automate d'états finis classique. Un automate de Moore, par exemple, change d'état en fonction de la valeur de certaines fonctions de transition, et émet des commandes propres à chaque état. Il dispose d'un état d'entrée, et la somme des fonctions de transition sur chaque état est égale à un. Les affectations `Data->NextStage=STAGE4;` correspondent à ces fonctions de transition, et les commandes émises dans chaque état correspondent aux actions effectuées dans chaque module du processus. La seule différence réside dans le fait qu'il n'est pas nécessaire d'*émietter* les tâches, de manière à ce que chaque état de l'automate n'ait que des commandes élémentaires à exécuter.

3.2.2 Le séquenceur

Le rôle du séquenceur de tâches consiste donc à implémenter la stratégie suivant laquelle chaque tâche se verra autorisée à utiliser le processeur à tour de rôle. Le séquenceur manipule simplement une liste de processus dits *actifs* [7].

```
while (!ActiveList->Empty) {
   RunningProc=ActiveList->Top;
   Data=RunningProc->Data;
   switch (Data->Type) {
```

[6]Cette variable *NextStage* est comparable au rôle du registre “pointeur programme” (*Program Counter*), utilisé dans la micro-programmation des processeurs afin de localiser l'instruction machine suivante à décoder puis à interpréter.

[7]que nous opposerons par la suite aux processus *bloqués* ou encore appelées *en attente*.

```
      ...
      case P_REGION :
         ProcessRegion (Data); break;
      case P_CONTOUR :
         ProcessContour (Data); break;
   }
   ...
   /*
    * Round-robin
    */
   RunningProc=ExtractTop (ActiveList);
   AddBottom (ActiveList,RunningProc);
}
```

L'exemple montre l'activité du séquenceur, qui consiste à utiliser le processus en tête de liste des processus actifs, sa structure de données `Data` contenant tout son environnement de travail. En fonction du type de ce processus, le code adéquat est invoqué. La fin du séquencement consiste à placer le processus qui vient de s'exécuter en queue de liste. Le séquenceur est également en charge de traitements supplémentaires, il doit par exemple veiller à la remise à jour de certaines structures des processus, et aux acheminements de leurs diverses requêtes (création de nouveaux processus [8], mises en attente, terminaison, envoi de signaux, etc).

Notion de priorité

Dans le cadre d'un système de vision, tout comme celui d'un système d'exploitation d'ordinateur, il peut être intéressant de définir des tâches plus prioritaires que d'autres, c'est à dire des tâches qui obtiennent plus souvent l'accès au processeur que d'autres.

L'implémentation actuelle ne permet pas de changer la priorité d'un processus au niveau du séquenceur. Ceci pourrait être réalisé de deux manières différentes :

- En changeant la durée élémentaire pendant laquelle un processus est autorisé à s'exécuter dans le système. Ce temps élémentaire ou *time slice* se traduit dans notre système de vision par un nombre maximum de pixels qu'un processus peut agréger avant de se suspendre. Il est donc possible de rendre un type de processus plus prioritaire qu'un autre en augmentant ce *quantum de pixels élémentaire*. Dans notre implémentation actuelle, ce quantum est fixé initialement pour les processus de type contour, et région, et n'a pas la possibilité d'évoluer par la suite. Il est nécessaire de prévoir deux valeurs différentes pour les contours et les régions, et le temps d'agrégation moyen d'un pixel est différent pour ces deux primitives, essentiellement car un nombre de pixels voisins très différent est mis en jeu. Voir le chapitre 1 pour davantage de précisions.

- En changeant la règle d'attribution du processeur au niveau du séquenceur. Au lieu de choisir systématiquement le processus en tête de liste, on peut supposer un compteur de préférence dans chaque processus, dont la valeur initiale dépend de la priorité, qui est augmentée chaque fois que le processus dans la liste n'est pas choisi, et qui est diminué lorsqu'il devient le processus en cours d'exécution. Le séquenceur choisit à chaque itération le processus de la liste ayant la plus forte valeur de ce compteur de préférence. Cette possibilité n'a pas été exploitée.

[8] La possibilité de créer de nouveaux processus se fait de manière analogue à l'appel système *fork()*, suivi de *execl()* sous Unix. Elle répond à des besoins apparus lors la description de la coopération de méthodes décrite dans les paragraphes 2.4.3 et 2.4.4.

3.3 Les données de travail d'un processus

Nous séparons dans cette partie les données relatives au séquencement des données relatives à la segmentation.

3.3.1 Les données de séquencement

La structure d'un processus contient des données nécessaires au séquenceur. La figure 3.2 montre une partie de la structure d'un processus, ainsi que diverses interdépendances pouvant exister entre ces structures. Les case grisées correspondent soit à des simplifications pour la clarté de la figure, soit à des structures étudiées dans une autre partie de ce document.

```
typedef struct process
{
   /********* INFOS SPECIFIQUES AU SYSTEME ******************************/
   int            PID;        /* Numéro d'identification              */
   int            Depth;      /* Profondeur dans l'arbre des processus */
   PtrProcess     Father;     /* Numéro du père                        */
   PtrList        ChildList;  /* Liste des processus fils              */
   byte           State;      /* Etat de vie actuel                    */
   short          NbWaits;    /* Nombre de réveils nécessaires         */
   /********* INFOS SUR LE TEMPS D'EXECUTION ****************************/
   int            LifeTime;   /* Durée de vie                          */
   int            WaitTime;   /* Nombre de cycles d'attente            */
   int            ActiveTime; /* Nombre de cycles d'action             */
   /********* POINTEUR SUR L'ENVIRONNEMENT DU PROCESSUS *****************/
   PtrData     Data;          /* Les données connues du processus      */
   PtrMailBox  MailBox;       /* La boite à lettre du processus        */
   PtrSignal   SignalManager; /* Le gestionnaire de signaux du process */
} TypeProcess;
```

- `PID` permet d'identifier un objet de type processus de manière unique pendant l'exécution du système.
- `Depth` indique la profondeur de ce processus dans l'arbre des processus, initialisé par le processus *racine* initial, de profondeur 1.
- `Father` permet de remonter dans l'arbre vers le processus père.
- `ChildList` décrit une liste de pointeurs vers les processus fils.
- `State` indique l'état courant du processus, pouvant prendre la valeur :
  - `RUNNING` si le processus est celui qui s'exécute sur la machine.
  - `SLEEPING` si le processus est dans la liste d'attente pour être exécuté.
  - `WAITING` si la processus est bloqué (en attente de la terminaison de ses fils).
  - `TERMINATED` si le processus vient de se terminer et attend que ses structures internes soient détruites.
- `NbWaits` est un compteur gérant la mise en attente du processus.
- Les valeurs `LifeTime`, `WaitTime` et `ActiveTime` permettent d'établir des statistiques sur l'activité du processus.
- `Data` pointe sur les données du processus relatives à la segmentation, et décrites dans le paragraphe 3.3.2. La primitive en cours de segmentation, les autres primitives connues du processus, le pixel germe à la base de la segmentation, les différents seuils nécessaires à ce processus sont le genre d'information stockée dans cette structure.

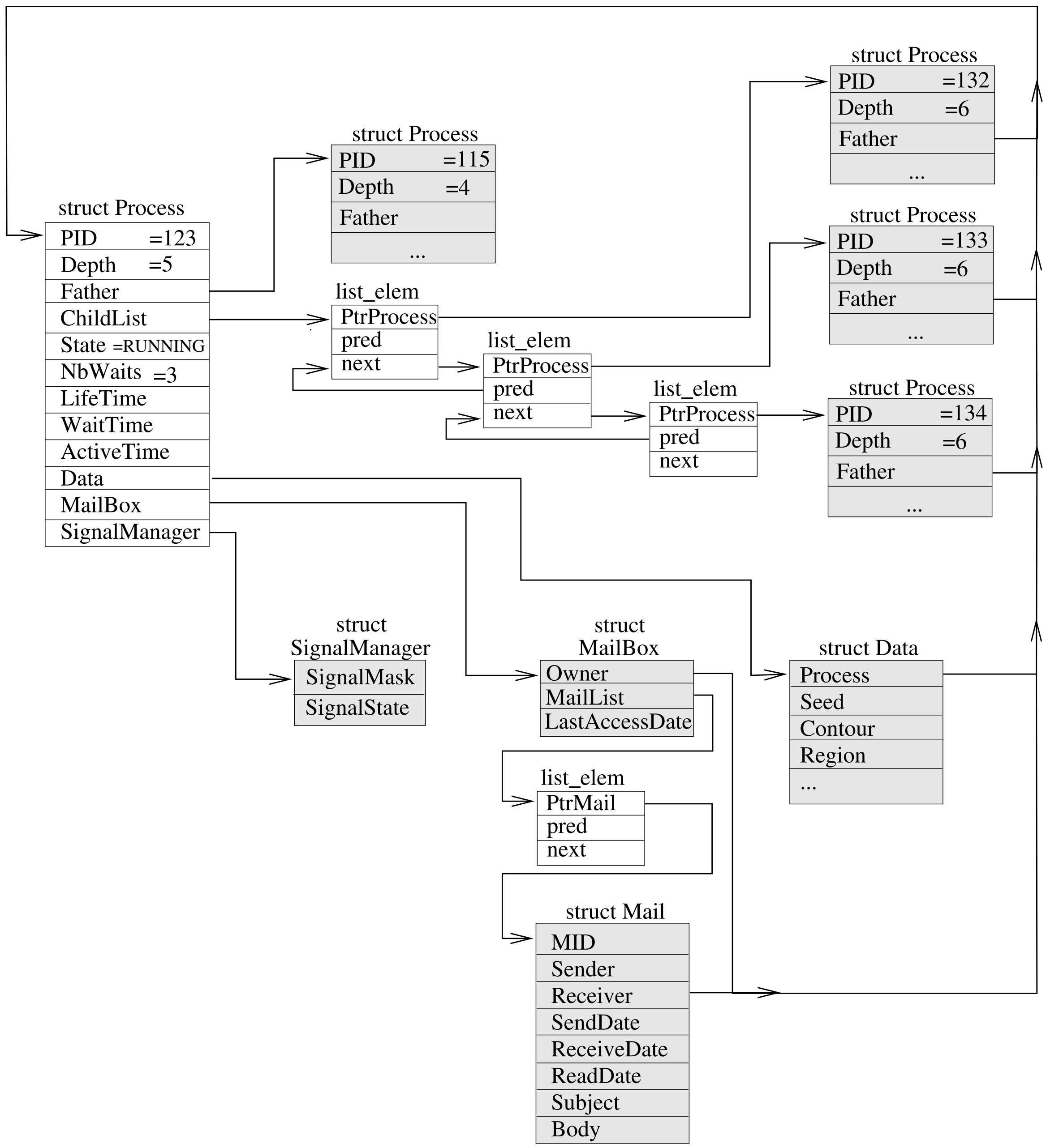


Fig. 3.2: Les données de séquencement d'un processus.

- `MailBox` définit une structure permettant d'émettre et de recevoir des messages avec les autres processus. Cette structure sera décrite plus précisément dans le paragraphe 3.5.1. La sémantique de ces messages doit répondre à un protocole partagé au sein de chaque processus. Cet outil permet à titre d'exemple dans notre système de segmentation d'instaurer une négociation entre deux processus de type région, dans le but de fusionner les deux primitives. Le paragraphe 2.4.6 explique l'intérêt de la fusion dans le cadre de la coopération entre les détecteurs.
- `SignalManager` définit une structure pour la réception de signaux, informations élémentaires, venant s'accumuler dans le processus au sein d'une structure de champ de bits. Le processus a la possibilité de masquer la réception de certains signaux. Cette structure sera décrite plus précisément dans le paragraphe 3.4.3.3, et 3.5.2. Cette structure permet de réveiller un processus sur certaines conditions, comme par exemple la réception d'un message dans la structure `MailBox`.

3.3.2 Les données de segmentation

Notre implémentation s'est efforcée le plus possible de cloisonner les structures, et ainsi de regrouper toutes les données spécifiques à l'activité des processus dans une structure commune. Le reste du système apparaît alors comme un moteur relativement générique permettant de gérer l'activité parallèle de tâches anonymes. Le paragraphe 3.3.1 a présenté ces données génériques. Les informations utilisées par les processus [9] sont regroupées dans la structure `Data`, voir la figure 3.2. Chaque processus dispose d'une structure unique pour les données de segmentation.

```
typedef struct data
{
   PtrProcess      Process;        /* Le processus concerné         */
   PtrPixel        Seed;           /* Le germe du processus         */
   PtrContour      Contour;        /* Le contour courant            */
   PtrRegion       Region;         /* La région courante            */
   PtrList         OtherContours;  /* Les autres contours           */
   PtrList         OtherRegions;   /* Les autres régions            */
   struct TypeData *Variables;     /* Les variables                 */
} TypeData;

typedef struct variables
{
                                   /*** VARIABLES DE TACHE **********/
        byte       Type;           /* Le type du processus          */

                                   /*** VARIABLES DE SEQUENCEMENT ***/
        byte       ExitCode;       /* Code de sortie (OK/ERREUR)    */
        byte       LastStage;      /* Etat précédent                */
        byte       Stage;          /* Etat courant                  */
        byte       NextStage;      /* Etat suivant                  */

        boolean    TimerLocked;    /* switch d'activation du timer  */
        short      Timer;          /* Compte à rebours (timer)      */
        short      ResetValue;     /* La valeur par défaut au reset */

        short      NbCycles;       /* Nombre de cycle d'agrégation  */
        short      NbAgregStages;  /* Nombre de phases d'agrégation */
```

[9]Ces informations peuvent être assimilé à l'espace de travail *utilisateur* dans la terminologie des systèmes d'exploitation, par opposition à l'espace *noyau*.

```
                               /*** VARIABLES MIXTES ************/
        boolean    Redraw;         /* nécessité d'un refresh grafix */
        int        LastNbPoints;   /* Nombre de points agrégés      */

                               /*** VARIABLES CONTOUR ***********/
        PtrPixel   LastPixel;      /* Dernier pixel de contour      */
        byte       LastExtrem;     /* Dernière extrémité agrégée    */
        short      LastValidHead;  /* Dernier code validation       */
        short      LastValidQueue; /*                               */
        PtrRegion  Adjacent1;      /* Région adjacente d'un côté    */
        PtrRegion  Adjacent2;      /* Région adjacente de l'autre   */

                                       /*** SEUILS ADAPTATIFS ***/
        float      EdgeThreshold;          /* seuil contour         */
        float      EdgeThreshold2;         /* seuil contour 2       */
        float      RegionThreshold;        /* seuil région          */

                                       /*** PONDERATIONS DES ****/
                                       /*** PROCESSUS ***********/
        float      EdgeWeights   [NB_EDGE_EVAL_TYPES];
        float      RegionWeights [NB_REGION_EVAL_TYPES];
} TypeData;
```

La description de ces variables sera succincte, car elles n'apportent pas d'éclairage particulier sur le gestion du multi-tâches, qui demeure l'objet de ce chapitre.

- `Type` est un drapeau spécialisant l'activité contour ou région du processus. Chaque méthode dispose donc d'une structure de données de segmentation unifiée, même si elle n'en utilise pas tous les champs, selon qu'elle segmente un contour ou une région.
- `LastStage`, `Stage`, `NextStage`, reflètent l'état de séquencement dans lequel se situe le processus, au sens de l'automate de Moore décrit précédemment.
- Des variables permettent de gérer un *timer* [10] sur le nombre de pixels agrégés, afin de déterminer à quel moment le processus doit se suspendre, pour rendre la main au séquenceur lorsque son *quantum de pixels élémentaire* est épuisé.
- Une variable stocke le seuil à appliquer sur la fonction d'évaluation. D'autres variables conservent les pondérations élémentaires permettant de calculer l'évaluation globale. Le paragraphe 1.5 du chapitre 1 décrit à ce propos les évaluations élémentaires, et les moyens de les combiner pour en déduire l'évaluation "pondérée" globale d'un pixel candidat.
- D'autres variables consistent simplement à conserver un pointeur sur des accointances, et d'autres informations de proximité : le pixel initial du processus (`Seed`), la région ou le contour en cours de construction (`Contour`, `Region`), les autres primitives de l'image dont le processus a eu connaissance par l'exécution de ses processus fils (`OtherContours`, `OtherRegions`), la structure de processus associée à la primitive (`Process`), etc.

3.4 Le multi-tâches coopératif

Cette partie va s'attacher à décrire quelques aspects du multi-tâches proposé dans ce système de vision, ainsi que les moyens permettant leur mise en œuvre. La résolution des changements de contexte, l'étude du quantum de temps élémentaire, la mise en attente et le réveil des processus, et enfin leur création et destruction seront les différents aspects décrits dans cette partie.

[10] Un compte à rebours.

3.4.1 Les changements de contexte

3.4.1.1 Le principe

Le contexte d'un processus définit son environnement de travail. Il s'agit d'un ensemble de données qui lui sont propres, et qu'il est le seul à pouvoir manipuler, à l'exclusion de tout autre processus. Dans le cadre du système de processus de segmentation proposé, ce contexte contient en pratique les structures décrites aux paragraphes 3.3.1 et 3.3.2. A ce titre, entre le moment où un processus de segmentation se suspend pour rendre la main au séquenceur, et le moment où le séquenceur lui redonne la main à nouveau, il est légitime que le processus retrouve son contexte dans le même état que celui dans lequel il l'avait laissé avant de s'interrompre. Cette préservation des contextes s'avére indispensable pour prétendre gérer une population de processus s'exécutant de façon concurrente.

La remarque précédente permet déjà de dégager la nécessité de définir un contexte par processus, en ajoutant une contrainte forte interdisant à chaque processus d'accéder à un contexte qui ne lui appartient pas. Une programmation à base d'objet permet de gérer efficacement cette contrainte. Un langage de plus bas niveau, tel que le C, n'interdit cependant pas une programmation reposant sur les mêmes principes d'encapsulation des données.

Parmi l'ensemble des processus du système à un instant donné, un au moins est en cours d'exécution [11]. A ce processus correspond donc la notion de *contexte actif*. Lorsque le processus Unix qui implémente notre système de segmentation fonctionne, le code qui s'exécute sur le processeur peut correspondre :

- soit à une partie de code se rattachant à la gestion de la partie *système d'exploitation*, par exemple la création, la destruction des processus, l'expédition des messages entre les processus, etc. Le système d'exploitation dispose du privilège de modifier le contexte des processus, afin de pouvoir effectuer les actions énumérées précédemment.
- soit à une partie de code se rattachant au fonctionnement d'un des *processus de segmentation* [12] : le processus de segmentation [13] en cours d'exécution. Dans ce cas, toutes les manipulations faites par nos algorithmes doivent porter sur des variables appartement **exclusivement** :
  - soit au *contexte actif*, c'est-à-dire le contexte du processus en cours d'exécution.
  - soit à l'environnement commun à tous les processus, par exemple l'image résultat dans laquelle les pixels sont étiquetés à mesure qu'ils sont incorporés à une primitive région ou contour.

  Si une manipulation est effectuée par un processus de segmentation en dehors de ce cadre, l'intégrité du système risque de ne plus être assurée. Le contexte d'un processus, autre que celui actuellement actif, aura en effet été modifié hors de son contrôle.

L'énumération précédente montre que, dans le cadre d'un processus de segmentation, il convient d'isoler attentivement les actions qui peuvent être faites, soit à l'environnement, soit au contexte courant. Dans la pratique, un simple pointeur `RunningProcess` permet de référencer le processus qui a été choisi au niveau du séquenceur pour s'exécuter, et le code des méthodes de

[11] Par souci de simplification, et car il s'agit du cadre de travail effectivement implémenté, nous limiterons l'explication au cas *monoprocesseur* se traduisant par l'activité d'un seul processus de segmentation simultanément.

[12] Un système multi-tâches sera d'autant plus efficace que le temps passé dans la partie *système d'exploitation* sera minimisée. Intuitivement, cela consiste à rendre l'activité du système d'exploitation la plus efficace possible, afin d'offrir aux tâches de l'utilisateur le plus de temps CPU possible.

[13] Le terme processus de segmentation n'est pas à confondre avec celui de processus Unix. Le système, sous la forme d'un unique processus Unix simule un ensemble de processus de segmentation.

segmentation accède simplement les données du processus en cours d'exécution en déréférençant ce pointeur : `RunningProcess->Data->....`

Le changement de contexte à effectuer au niveau du séquenceur en devient trivial, puisqu'il consiste à choisir un nouveau processus à exécuter parmi la liste des processus actifs, afin de l'affecter à `RunningProc`. Le code d'une même procédure utilisera alors les données du nouveau processus actif au lieu de l'ancien.

3.4.1.2 Le choix du processus à exécuter

Le rôle du séquenceur consiste à choisir un processus, parmi tous ceux qui sont actifs dans le système, et qui peuvent donc prétendre accéder au processeur. Le séquenceur lance ou poursuit l'exécution du processus choisi. De manière tout à fait semblable à un système d'exploitation classique, une liste de processus actifs est maintenue à jour. Cette liste est gérée en FIFO (*First In First Out*). Le séquenceur boucle, tant que des processus existent dans cette liste :

- En fonction du statut du processus qui vient de suspendre son exécution, le séquenceur le replace dans la liste des processus *actifs* en queue de liste, ou bien le bascule dans la liste des processus *bloqués* [14].
- Le processus en tête de liste des actifs est extrait, il devient le processus en cours d'exécution.

Le séquencement ne gère pas de priorité à ce niveau pour l'accès au processeur. La gestion de la liste en FIFO garantit un accès équitable pour tous les processus de la liste des actifs.

3.4.2 Le quantum de temps élémentaire

3.4.2.1 Motivation

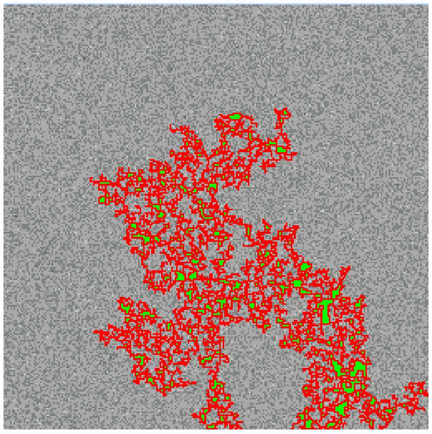
Le paragraphe 3.4.1.2 décrit une politique équitable pour décider de l'accès au processeur. Il ne dispose cependant pas de tous les moyens utiles pour s'assurer qu'en pratique chaque processus de la liste des actifs disposera d'autant de temps CPU que les autres. Le multi-tâches coopératif que nous avons retenu ne permet pas de retirer l'utilisation du processeur à un processus lorsque la période de temps qui lui avait été allouée est dépassée. Lorsqu'un processus devient actif sur le processeur, il est le seul à pouvoir décider de se suspendre.

Cette politique présente des avantages décrits dans le paragraphe 3.2. En revanche, il est nécessaire d'ajuster, au niveau de chaque processus, une durée au terme de laquelle il suspendra son activité.

3.4.2.2 Les alternatives

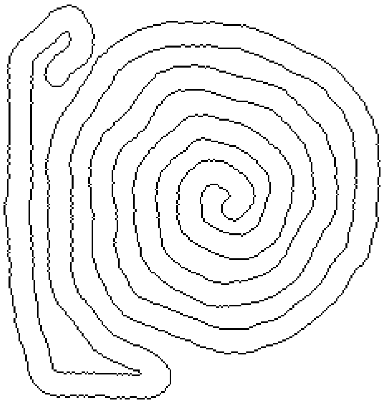
La première possibilité consiste à déclencher un *timer* au moment où le séquenceur désigne le processus à exécuter. Le processus de segmentation construit sa primitive pixel par pixel, et se suspend lorsque le *timer* expire. Cette méthode assure un temps d'exécution pratiquement constant pour chaque processus entrant dans une étape de construction de sa primitive. L'inconvénient majeur de ce *timer* est que son fonctionnement dépend d'une donnée indépendante du programme : le temps. Selon la charge de la machine sur laquelle fonctionne le système, le quantum de temps "réel" utilisé par le processus de segmentation correspondra une durée "effective" variable, se traduisant par un nombre de pixels agrégés variable entre deux exécutions du système, malgré des conditions initiales identiques. Cette non reproductibilité d'une exécution

[14] Ce cas correspond à une mise en attente liée au lancement d'un processus fils par exemple. Le processus sera réveillé par la terminaison du processus fils.



(a) région en construction sur un fond homogène.

(b) image de synthèse en niveaux de gris.



(c) image du contour obtenu.

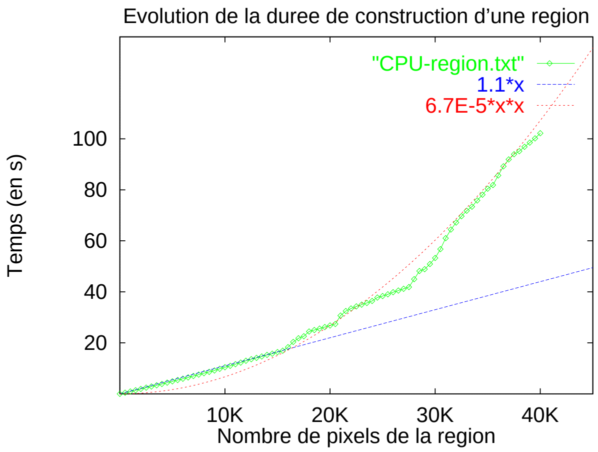
FIG. 3.3: Images test utilisées pour mesurer le temps de construction des primitives de type contour et région sur une grande quantité de pixels.

est inacceptable pour des raisons de debuggage en particulier. Ce principe n'a donc pas été retenu.

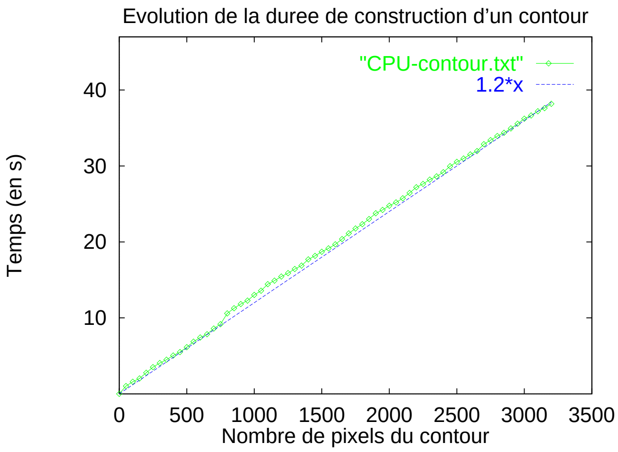
La deuxième possibilité qui a été appliquée consiste à utiliser une unité de mesure dont le système a le contrôle. Le nombre de pixels agrégés par un processus jouera le rôle du *quantum de temps élémentaire*. Un processus se suspendra lorsqu'il aura incorporé un nombre fixé de pixels à sa primitive. Ce nombre est différent selon le type du processus, puisqu'un processus de segmentation de type contour et de type région ne travaillent pas à la même cadence.

3.4.2.3 Application

La mesure du temps de construction d'une primitive région puis contour s'est faite sur des images de synthèse, données en figure 3.3. Un unique processus de construction de région est initialisé sur une image homogène avec une moyenne des niveaux de gris de 128, présentant un bruit gaussien d'écart-type $\sigma = 8$. Le processus contour est initialisé sur un motif de synthèse en forme de spirale, légèrement bruité. L'objectif de ces deux images est qu'un unique segmenteur



(a) région.



(b) contour.

FIG. 3.4: La relation entre le temps de construction d'une primitive de type région ou contour et le nombre de pixels incorporés.

Nombre de pixels de la région	1000	2000	3000	4000
Durée de construction (en s)	0.91	1.93	2.89	3.87
Nombre de pixels du contour	200	600	1000	2000
Durée de construction (en s)	0.28	0.74	1.30	2.48

TAB. 3.1: La correspondance entre le nombre de pixels d'une primitive, et le temps effectif mesuré pour leur construction.

	Nombre de pixels	Pourcentage de l'image
Pixels contour	1059	7 %
Pixels région	10416	63 %
Non étiqueté	4909	30 %
Total	16384 (128x128)	100 %

TAB. 3.2: Statistique sur l'étiquetage des pixels dans un exemple concret, en utilisant un quantum de 600 pixels pour les contours, et 2000 pixels pour les régions.

	Nombre de pixels	Pourcentage de l'image
Pixels contour	1115	7 %
Pixels région	10209	62 %
Non étiqueté	5060	31 %
Total	16384 (128x128)	100 %

TAB. 3.3: Statistique sur l'étiquetage des pixels en utilisant un quantum de pixels faible de 20 pixels à la fois pour les contours et pour les régions.

puisse construire la primitive, contour ou région, dans sa totalité, sans aide extérieure, afin de pouvoir mesurer le temps écoulé depuis le début de la construction, en fonction du nombre de pixels incorporés. Ces images ne présentent donc pas de difficultés de segmentation dans ce but.

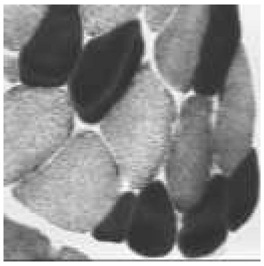
Les deux graphiques de la figure 3.4 montrent l'évolution de la durée de construction en fonction du nombre de pixels incorporés à la primitive, dans le cas du processus de type contour et de type région. On notera le progression linéaire de la construction du contour en $\mathcal{O}(N)$, et la progression en $\mathcal{O}(N^2)$ pour la région, où N est le nombre de pixels incorporés à la primitive. Cette complexité en $\mathcal{O}(N^2)$ s'explique par le fait qu'il soit nécessaire d'insérer des nouveaux éléments régulièrement dans une liste triée de pixels candidats, dont la taille augmente à mesure que la région se développe. Toutefois pour les 5000 premiers pixels d'une région, une approximation linéaire peut être réalisée. Il est donc assez légitime d'utiliser le nombre de pixels agrégés à une primitive comme valeur de *quantum de temps élémentaire* pour le séquencement des tâches, puisqu'il existe une correspondance linéaire entre ces deux quantités dans le cas du contour, et linéaire mais limitée aux premiers pixels dans le cas de la région. A titre d'exemple, la table 3.1 donne quelques valeurs de quantum de pixels, et leur équivalence en terme de temps CPU utilisé par le système [15]. Notre application utilise actuellement les valeurs de quantum de 2000 pixels pour les processus de type région, et de 600 pour les processus de type contour.

3.4.2.4 Discussion

Le choix de ce quantum de temps élémentaire s'avére souvent délicat, car il définit la granularité des processus pour le séquenceur de tâches. Un quantum de temps faible laisse peu de temps d'exécution à chaque processus. Cela permet de boucler rapidement sur toute la liste des processus actifs, et donne une impression de parallélisme efficace. En contrepartie, le système d'exploitation est davantage mis à contribution, et il utilise autant de temps CPU qui ne sera plus disponible pour les processus de segmentation eux-mêmes.

A titre d'exemple, nous proposons l'exécution du système avec d'une part le quantum classique de 600 pixels pour les contours, et de 2000 pixels pour les régions, et d'autre part avec

[15] Les temps mesurés sont obtenus sur un PC équipé d'un processeur Intel MMX à 291MHz, fonctionnant sous Linux, version 2.0.33, avec 96 Mo de RAM, en utilisant le compilateur gcc version 2.7.2.3.



(a) niveaux de gris.

(b) quantum de 600/2000 pixels.

(c) quantum de 20/20 pixels.

FIG. 3.5: Exemple de segmentation complète montrant la prédominance des pixels étiquetés *région* – ici en gris clair et gris foncé – par rapport aux pixels étiquetés *contour* – en noir dans l'image – au terme de la segmentation.

```
-----------------------------------------------
                0.01   23.46       1/1          Cooperation [4]
[5]     99.5    0.01   23.46       1        Scheduler [5]
                0.01   17.99    1571/1571       ProcessRegion [6]
                0.04    3.62     393/393        ProcessContour [12]
                0.00    0.40       8/8          ProcessRoot [86]
                0.00    0.33    2147/2147       CreateRequest [95]
                0.00    0.32    2147/2147       RequestProcessing [98]
                0.05    0.22    2147/2147       UpdateProcesses [110]
                0.02    0.20    2147/2491       SortData [117]
                0.00    0.19    2147/2147       DeleteRequest [143]
                0.00    0.02    2147/2147       RoundRobin [287]
                0.00    0.02    2147/2147       TestEndProcess [303]
                0.00    0.01    2147/2147       ChooseRunningProcess [326]
                0.01    0.00    2148/2148       EmptyActiveList [338]
                0.01    0.00   12882/12882      GetKernelStat [354]
                0.00    0.00     175/175        DaemonSleep [456]
                0.00    0.00       1/1          LoadInit [514]
                0.00    0.00       1/1          LoadClose [525]
                0.00    0.00    2147/31549      GetProcessData [560]
                0.00    0.00    2147/14691      GetDataType [565]
                0.00    0.00    2147/2147       ShiftDataStage [616]
                0.00    0.00    2147/2147       LoadUpdate [615]
                0.00    0.00    2147/3711       GetKernelCurrentTime [591]
-----------------------------------------------
```

FIG. 3.6: Information de *profiling* obtenue à l'issue d'une segmentation type grâce à la commande Unix *gprof* appliquée au lancement du système de segmentation. Le quantum utilisé est de 600 pixels pour les contours, et de 2000 pixels pour les régions.

```
-----------------------------------------------
                0.00   27.17       1/1          Cooperation [4]
[5]     99.6    0.00   27.17       1        Scheduler [5]
                0.03   20.86    3135/3135       ProcessRegion [6]
                0.00    3.80     455/455        ProcessContour [13]
                0.01    0.71    4200/4200       CreateRequest [69]
                0.00    0.44      12/12         ProcessRoot [86]
                0.00    0.37    4200/4200       DeleteRequest [103]
                0.06    0.24    4200/4200       UpdateProcesses [118]
                0.00    0.29    4200/4200       RequestProcessing [121]
                0.00    0.25    4200/4551       SortData [125]
                0.01    0.04    4200/4200       RoundRobin [222]
                0.02    0.01    4200/4200       TestEndProcess [277]
                0.01    0.02   25200/25200      GetKernelStat [281]
                0.00    0.01    4200/4200       ChooseRunningProcess [381]
                0.00    0.00     598/598        DaemonSleep [422]
                0.00    0.00    4201/4201       EmptyActiveList [450]
                0.00    0.00    4200/57893      GetProcessData [306]
                0.00    0.00    4200/29737      GetDataType [344]
                0.00    0.00       1/1          LoadInit [525]
                0.00    0.00       1/1          LoadClose [534]
                0.00    0.00    4200/4200       ShiftDataStage [602]
                0.00    0.00    4200/4200       LoadUpdate [599]
                0.00    0.00    4200/7265       GetKernelCurrentTime [590]
-----------------------------------------------
```

FIG. 3.7: Information de *profiling* obtenue avec un quantum de pixels très faible. Le quantum utilisé est de 20 pixels pour les contours et pour les régions.

un quantum de pixels très faible de 20 pixels à la fois pour les contours et les régions. Toutes les autres conditions initiales sont identiques par ailleurs.

Les deux résultats obtenus sont présentés sur la figure 3.5(b) et 3.5(c), et ils permettent déjà de souligner le peu de différences qui existent entre eux. Tous deux présentent environ la même cartographie de contours. Le résultat 3.5(b) présente un peu de fragmentation sur les régions situées à droite dans l'image. Notre souci dans cet exemple n'est pas de mesurer la qualité intrinsèque du résultat par rapport à une quelconque mesure de référence, mais d'étudier la manière dont cohabitent les deux types de processus de segmentation dans le système. A ce titre, les tables 3.2 et 3.3 montrent la répartition des pixels segmentés de l'image. Ces deux tables montrent :

- que très peu de différences existent entre les deux résultats 3.5(b) et 3.5(c), ce qui s'avére être une confirmation au vu des images elles-mêmes.
- que les pixels étiquetés *région* sont largement prépondérants par rapport aux pixels étiquetés *contour*, dans un rapport de un à neuf dans l'exemple.

Cette dernière remarque, ainsi que les valeurs de temps CPU fournies dans la figure 3.4 nous permettent de déduire que l'ensemble des processus de type région utiliseront globalement largement plus de temps CPU que les processus de type contour, ce qui n'est pas surprenant puisqu'un contour est une structure mono-dimensionnelle alors qu'une région est bi-dimensionnelle. Le temps de construction *par pixel* n'est pas fondamentalement différent entre les régions et les contours, surtout lors de l'étiquetage des premiers pixels, malgré la complexité des algorithmes, en $\mathcal{O}(N)$ pour les contours et en $\mathcal{O}(N^2)$ pour les régions, où N est le nombre de pixels.

Les tables 3.6 et 3.7 en apportent la confirmation. Elle montrent le temps passé dans chacune des procédures à partir de la procédure `Scheduler()`, qui effectue le séquencement des tâches dans le système. Le chiffre seul dans la colonne de gauche montre le pourcentage de temps CPU passé dans cette procédure `Scheduler()` par rapport au temps d'exécution total du système, 99.5 et 99.6 %, ce qui indique qu'il n'est pas nécessaire de remonter davantage dans la hiérarchie des procédures, car elle seule et celles qu'elle appelle cumulent la quasi totalité du temps passé dans le système. Les chiffres de la troisième colonne classent par ordre décroissant le temps CPU passé dans chacune des procédures appelées par `Scheduler()`. La quatrième colonne comptabilise le nombre d'appels fait à une procédure, par rapport au nombre total d'appel qui lui sont faits dans tout le programme [16].

L'analyse de ces deux traces montre que la disproportion est confirmée entre le temps utilisé par les contours et par les régions, dans un rapport de un à six dans l'exemple. L'autre remarque entre la table 3.6 et 3.7 indique que le système de segmentation, pour un résultat comparable, est globalement plus lent (27 secondes contre 24) lorsque la granularité du séquencement est plus fine, comme cela avait été souligné au début de ce paragraphe.

A titre d'exemple, la procédure `ProcessRegion()`, qui contient l'ensemble du code utilisé par les processus de type région, est appelée deux fois plus dans la table 3.7, ce qui constitue l'effet direct de la diminution du *quantum de pixels*. Cette augmentation du nombre d'appel (2 fois) n'est cependant pas en accord avec la diminution du quantum de pixels, passé de 2000 à 20. On pourrait s'attendre à 1000 fois plus d'appels à la procédure `ProcessRegion()`. Cela n'est pas le cas, pour une raison très simple. Cette procédure n'est pas invoquée uniquement pour étiqueter des pixels. Elle est appelée pour traiter chaque état de l'automate du processus de type région (initialisation, focalisation, communication, etc). L'étape au cours de laquelle le processus de type région étiquette effectivement des pixels ne constitue qu'un état parmi d'autres. Par ailleurs, une part importante de ces appels à `ProcessRegion()` constituent la première étape du processus, au cours de laquelle il initialise ses structures. Le processus ne poursuivra pas au delà, si son pixel germe associé se situe déjà sur une région existante. Il retournera un pointeur sur la région correspondante, et se terminera ainsi. Il ne contribuera donc pas à l'étiquetage de pixels, mais son initialisation sera cependant comptabilisée par le *profiling*.

3.4.3 La mise en attente

Ce paragraphe décrit deux méthodes de mise en attente des processus dans le cadre du système de vision proposé. La première consiste à bloquer le processus jusqu'à la réalisation d'une condition dans le système. La seconde opère une mise en sommeil pour une durée arbitraire du processus, sans condition spécifique de réactivation.

3.4.3.1 Mise en état de blocage

A l'instar d'un système d'exploitation réel, un processus n'a pas toujours d'opération à effectuer dans le système. Il peut être en attente d'une opération sur un périphérique lent, auquel cas il n'est pas judicieux qu'il continue à utiliser du temps CPU, en bouclant dans une attente active. La mise en attente d'un processus consiste donc à l'extraire *temporairement* de la

[16] Cette trace est un outil standard de développement, nommé *profiling*. Il permet entre autres de repérer les goulots d'étranglement d'un programme, et les procédures anormalement consommatrices de temps CPU. C'est un outil d'optimisation intéressant, puisqu'il permet de désigner les procédures prédominantes d'un algorithme, sur lesquelles devront porter en majorité les efforts d'optimisation. Cette trace s'obtient avec le compilateur **gcc** en rajoutant l'option **-pg** à la fois à la compilation et à l'édition de liens. L'exécution du programme génère un fichier **gmon.out**, sur lequel un filtre est appliqué postérieurement **gprof exécutable gmon.out**.

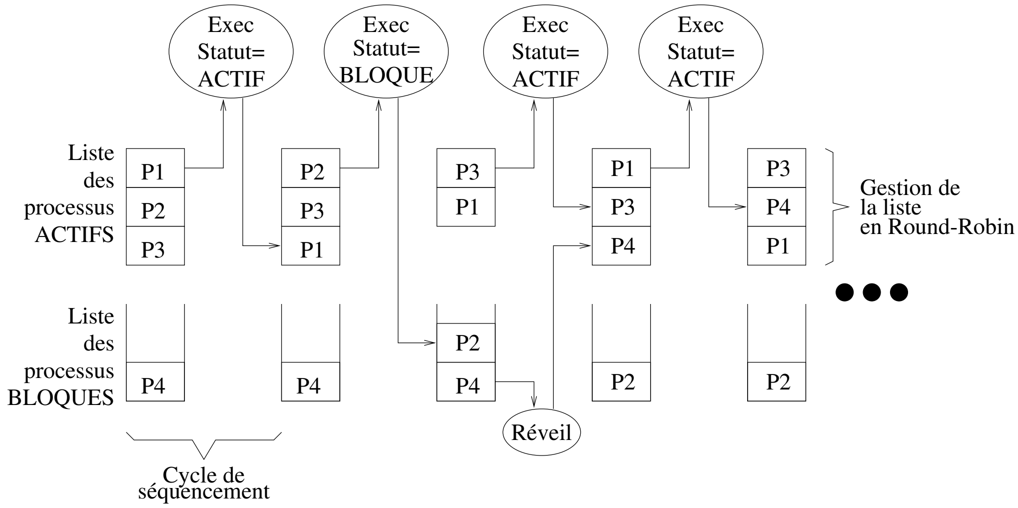


FIG. 3.8: La gestion des mises en attente au niveau du séquenceur.

liste des processus actifs, afin qu'il ne soit plus sélectionné par le séquenceur pendant la période au cours de laquelle il n'a plus d'action à effectuer sur le processeur. Celui ci est donc utilisé plus efficacement, puisque tout processus de la liste des actifs en a une utilisation effective.

Le système d'exploitation doit, pour sa part, gérer le moment où le processus bloqué a besoin de redevenir actif puisque, par définition, le processus lui-même n'est plus en état de le faire – il est bloqué –. Les systèmes d'exploitation gèrent généralement ces blocages et déblocages de processus sous la forme de sémaphores. Un sémaphore est attaché à une ressource physique, l'entrée du processus dans ce sémaphore provoque son blocage tant que la ressource n'est pas disponible, et le processus est débloqué par le système d'exploitation lorsque la ressource se libère. En ce sens, la seule connaissance du sémaphore sur lequel un processus est bloqué permet d'identifier l'objet du blocage du processus, et le moment où il devra être réveillé. Le réveil consiste alors à basculer le processus à nouveau dans la liste des processus actifs.

Le système de vision proposé a des contraintes de séquencement analogues. Deux conditions distinctes de mise en attente existent dans l'implémentation actuelle :

– Le processus de segmentation initialise des processus fils, et suspend son activité en attendant leur terminaison. La condition de réveil porte sur la fin du dernier processus fils.

– Le processus effectue un acte de communication à destination d'un tiers. Il se met en attente le temps que le processus tiers traite ce message, et lui fasse parvenir sa réponse. La condition de réveil porte sur la réception d'une réponse. Par extension, toute réception de message doit provoquer le réveil du processus destinataire, afin d'accélérer autant que possible le délai de traitement.

  Ce point implique des contraintes strictes de communication, qui doivent être gérées par chaque processus. Ainsi une certaine discipline individuelle garantit à ce niveau la cohérence au niveau du groupe :

  – Toute réception de message doit entraîner une réponse, car le réveil du processus expéditeur dépend de la réception de cette réponse.

  – Plus spécifiquement, cette réponse doit intervenir même dans les cas extrêmes. Par exemple, tout processus sur le point de se terminer doit *impérativement* vider sa boîte à message, en répondant à tous les messages en attente, avant de libérer ses structures.

La figure 3.8 présente le séquencement, à travers une succession de plusieurs cycles élémentaires. Le code de retour du processus qui vient de s'exécuter indique au séquenceur s'il doit placer ce dernier dans la liste des processus *bloqués*, où s'il peut demeurer dans la liste des processus *actifs*. Dans ce dernier cas, le processus est replacé en queue de liste. A l'issue d'un cycle de séquencement, un processus qui était bloqué peut revenir à l'état actif, sur la condition de réveil. Par exemple, la terminaison du dernier fils envoie un signal de type *fin de processus fils* à destination de son père, et réveille ce dernier, qui se voit alors réinséré dans la liste des *actifs*.

Il est important de jumeler ces deux actions (réveil et envoi de signal), afin qu'un processus qui redevient actif puisse connaître la cause de cette réactivation – fin des processus fils, ou réception de message – en consultant les signaux associés à son réveil.

3.4.3.2 Mise en sommeil

Le mécanisme de mise en attente décrit au paragraphe 3.4.3.1 fonctionne sur des condition de blocage, telles que l'attente de la terminaison de processus fils, et/ou la réception de messages. Il peut s'avérer utile pour un processus d'entrer dans un état d'attente temporaire, et de rester ainsi pour une durée déterminée, choisie par le processus lui-même. Cela permet par exemple de disposer d'un processus qui scrute périodiquement une ressource du système, et que l'on ne souhaite pas voir constamment actif, car cette ressource système évolue lentement. Ce mécanisme est comparable à la fonction `sleep()` de la librairie C.

La condition de réveil est alors l'expiration d'un *timer* armé par le processus au moment de sa mise en sommeil. Pour gérer cet ensemble de processus en sommeil, avec leur *timer* associé, plusieurs possibilités ont été envisagées. La plus cohérente par rapport à l'architecture du système a été retenue. Elle consiste à disposer d'un processus spécifique dans le système, obéissant aux mêmes règles de séquencement que les processus de segmentation, dont le rôle est de gérer les autres processus une fois qu'ils ont effectué une demande de mise en sommeil. Le *démon* [17] utilise deux étapes élémentaires pour son séquencement :

Etape d'initialisation .
: Le démon crée une liste pour stocker les processus en sommeil, et une liste contenant leur date de réveil.

Etape de traitement .

- Le démon insert en liste les processus dont il a reçu un message, avec leur date de réveil associée.
- Il passe en revue tous les processus de la liste, et procède au réveil de ceux dont la date de réveil est dépassée.

3.4.3.3 Gestion de signaux

Le paragraphe 3.4.3.1 a mis en lumière la nécessité de définir une structure supplémentaire propre à chaque processus, lui permettant – entre autres utilités – de connaître la cause de son réveil. Un protocole générique de stockage et de transmission de signaux inter-processus a été implémentée. La motivation reste toujours de développer des outils génériques, dont la fonctionalité puisse facilement être étendue à d'autres tâches que celles auxquelles ils sont cantonnés dans ce système de vision. Le gestionnaire de signaux en est une parfaite illustration.

[17] Un processus ayant ainsi un rôle intermitant, offrant un service sur une requête d'autres processus du système est couramment appelé *daemon* ou démon dans les systèmes Unix.

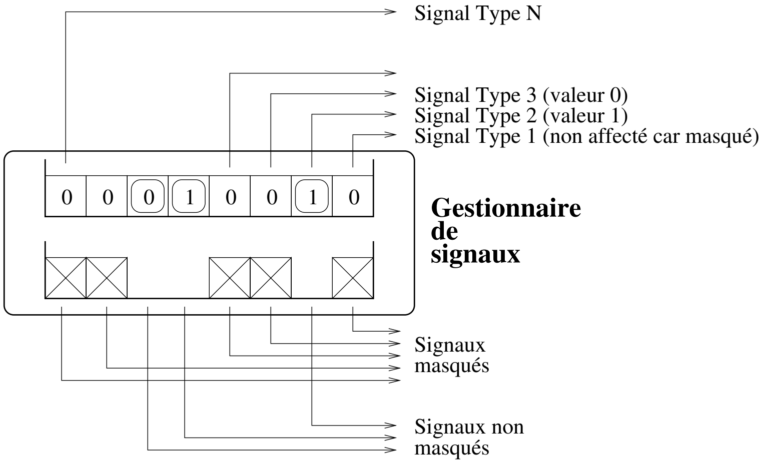


FIG. 3.9: Le gestionnaire de signaux.

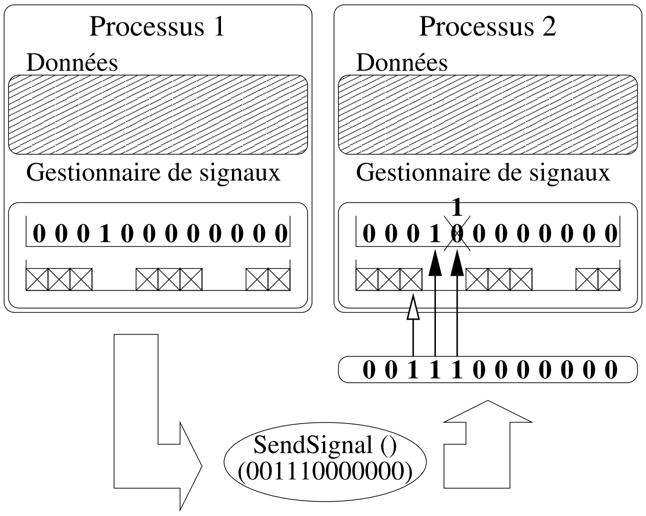


FIG. 3.10: L'envoi de signaux entre les processus.

```
#define ALL_SIGNALS     0x0003
#define NO_SIGNAL       0x0000

#define SIGNAL_MAIL              0
#define SIGNAL_CHILDREN_END      1

#define SIGNAL_MAIL_MASK         1L<<0
#define SIGNAL_CHILDREN_END_MASK 1L<<1

typedef struct signal
{
   unsigned short   SignalMask;    /* Le masque des signaux admissibles */
   unsigned short   SignalState;   /* Les signaux reçus                 */
}
```

Le masque permet à un processus d'ignorer certains signaux. La structure est celle d'un champ de bits. Chaque bit est porteur d'une information sémantique d'un type particulier. Il n'y a donc pas la possibilité de conserver plus d'un seul signal par type, voir la figure 3.9.

L'envoi d'un signal à destination d'un autre processus se fait par un appel à une procédure `SendSignal()`, dont le rôle est d'effectuer une opération binaire entre le champ de bits `val` fourni en paramètre, et le gestionnaire de signaux du processus cible :

`val AND SignalMask OR SignalState.`
`SignalMask` et `SignalState` sont les deux champs de bits constituant le gestionnaire de signaux du processus cible, voir la figure 3.10.

Chaque processus dispose de fonctions permettant de consulter, et de réinitialiser son propre gestionnaire de signaux, tant au niveau de `SignalMask` que de `SignalState`.

L'implémentation actuelle n'utilise que deux champs, comme l'indique l'extrait de code précédent. La sémantique de ces deux champs est fixée par le concepteur. `SIGNAL_MAIL` indique l'arrivée d'un message, et `SIGNAL_CHILDREN_END` indique la terminaison des processus fils.

L'émission d'un signal est accompagnée du réveil du processus destinataire, afin qu'il réagisse rapidement à l'événement qui lui est signalé par le message [18].

3.5 La communication

Une particularité de notre système de segmentation est d'utiliser l'environnement de travail comme médium de communication entre les processus. Cette approche se replace au cœur de la philosophie des systèmes multi-agents, dans lesquels chaque entité ne possède pas une vision complète du "monde" dans lequel elle agit. Elle en est donc réduite à exploiter une vision très partielle, souvent faite d'interaction avec l'environnement, et avec d'autres entités.

La communication par l'environnement consiste, à la manière des travaux de Luc Steels [Ste90], à permettre à chaque agent de laisser des traces de son passage dans l'environnement à destination des autres agents. Notre système utilise déjà pleinement cette caractéristique, puisque chaque processus de segmentation enrichit l'environnement de primitives régions ou contours, dont l'existence et les caractéristiques sont intégrées dans les mécanismes de décision des processus eux-mêmes.

Ce principe favorise paradoxalement un comportement hautement indésirable dans l'informatique, et précisément dans l'algorithmique "académique" : les effets de bords. L'utilisation d'une variable globale au cœur d'une procédure locale constitue en soi un effet de bord, puisqu'une modification extérieure de cette variable modifiera le comportement de la procédure locale. Cela rend un algorithme difficilement "démontrable", "évaluable" en terme de complexité algorithmique. Et la mise au point en est hasardeuse. Cependant les effets de bords permettent selon nous de générer une dynamique de groupe intéressante, qui serait difficilement simulable par d'autres biais. L'intérêt des systèmes multi-agents, principalement les systèmes réactifs, est d'utiliser justement au maximum les effets de bords à leur profit, par le biais des diverses interactions "opportunistes" entre les agents, afin de faire émerger d'un algorithme un comportement de groupe qui n'a pas été écrit explicitement.

L'architecture de notre système aspire à être très générale, et donc ne pouvait pas se limiter à une communication implicite entre les processus. Très vite, des nécessités de communications individuelles, en point-à-point, sont apparues. Deux processus devaient pouvoir établir un canal de communication propre afin de pouvoir par exemple négocier des territoires, comme dans le cas de la fusion par exemple. Le paradigme du système d'exploitation nous a donc invité à reproduire le schéma de communication implanté dans la version d'Unix écrite initialement à Berkeley, BSD Unix, et utilisant le mécanisme des échanges de signaux.

Parallèlement, un protocole d'échange de messages a aussi été réalisé. Inspiré du démon *sendmail* [19], ce deuxième moyen de communication permet de continuer à respecter le principe de

[18] Un processus est en attente de terminaison de ses processus fils. Il est réactivé par l'arrivée d'un message, avec le signal adéquat. Il répond négativement à ce message, et doit alors se remettre en attente de terminaison de ses fils, puisque cette condition n'est encore pas réalisée. La situation inverse est possible. Plus généralement, un réveil non souhaité, sera suivi par une remise en attente sur la même condition initiale.

[19] *sendmail* est un démon sous Unix permettant le routage et la distribution des *mails* des utilisateurs. Ce

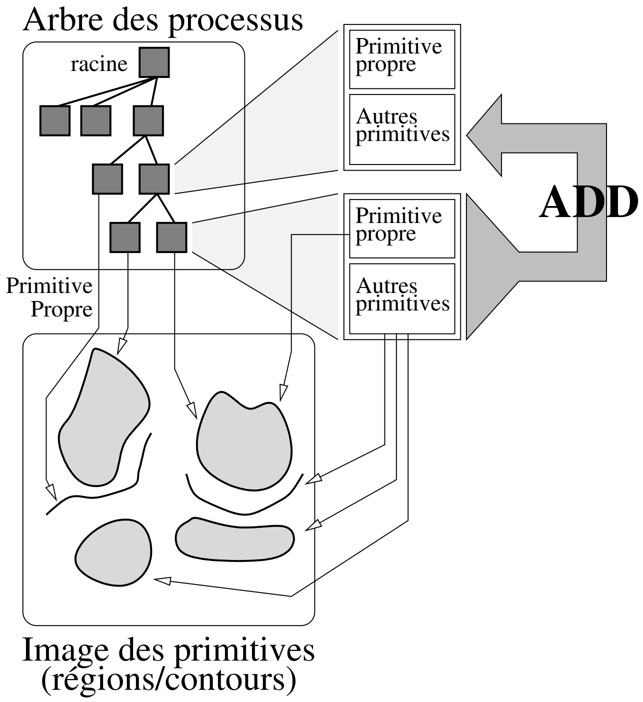


FIG. 3.11: La communication par remontée des primitives du processus fils vers son père lors de sa terminaison.

séparation des rôles entre le système d'exploitation et les processus. Ainsi les signaux véhiculent des informations de niveau "système d'exploitation" (terminaison d'un processus fils, arrivée d'un message, etc), tandis que les messages, eux, peuvent contenir une information dédiée à la tâche des processus (demande de fusion de région, fusion acceptée, fusion en cours, etc).

3.5.1 La communication par échange de messages

3.5.1.1 Motivation

La communication par échange de messages trouve son intérêt dans la nécessité de faire parvenir de l'information à un processus. Une circulation mono-directionnelle de données existe déjà entre deux processus, mais elle s'est vite avérée insuffisante pour les besoins de négociation de territoire auxquels elle était destinée.

La figure 3.11 rappelle les conditions strictes sous lesquelles sont placées ce transfert d'information monodirectionnel entre deux processus.

- Les deux processus doivent être liés par une relation de type *père-fils*. Un processus fils a segmenté un objet, qu'il fait parvenir à son père, pour répondre à la demande d'information qui avait motivé sa création.
- Chaque processus dispose d'un pointeur vers sa propre primitive, ainsi qu'une liste d'autres primitives, acquises au cours des diverses étapes de focalisation. Ces deux informations sont transmises au processus père, avant que le processus fils ne soit détruit. Elles vont venir se rajouter à la liste *Autres primitives* du processus père, voir la figure 3.11.

démon travaille au niveau utilisateur et pas au niveau noyau dans Unix. Il se différencie en ce sens de notre approche, ou le système de distribution de message –ici centralisé– peut aisément être réalisé "dans" le système d'exploitation.

Ce type de communication est donc mono-directionnel, hiérarchiquement ascensionnel, et ponctuel [20]. Toutes ces contraintes ne le rendent pas suffisamment généraliste.

L'autre type de communication passe par le médium image. Les processus, par les marques qu'ils déposent dans l'environnement image, sous la forme de pixels étiqueté à une région ou à un contour, effectuent indirectement un acte de communication. Le destinataire de cet acte n'est pas connu *a priori*, mais se révélera lorsqu'un processus quelconque accédera à ces pixels, et devra modifier son comportement du fait de leur étiquetage antérieur. L'utilisation de ce type de communication, implicite et complètement dépendant de la sémantique du domaine, ne satisfait pas non plus les contraintes de généralité fixées.

3.5.1.2 La structure de gestion de message

Pour ces raisons, un outil permettant une gestion explicite d'échange de messages entre des processus a été implémentée. Cet outil présente les caractéristiques suivantes :

- Une structure de base de message, avec des champs identifiant le processus expéditeur, le ou les processus destinataires, le sujet du message, et le corps proprement dit du message.
- Chaque processus possède une structure permettant la réception, le stockage, et le traitement de ces messages.

```
typedef struct mail
{
   int          MID;          /* Numéro d'identification       */
   PtrProcess   Sender;       /* L'expéditeur du message       */
   PtrListDesc  Receiver;     /* Les destinataires du message  */
   int          SendDate;     /* La date d'expédition          */
   int          ReceiveDate;  /* La date d'arrivée             */
   int          ReadDate;     /* La date de lecture            */
   char         *Subject;     /* Le sujet du mail              */
   char         *Body;        /* Le contenu du message         */

} TypeMail;

typedef struct mailbox
{
   PtrProcess   Owner;          /* Le processus propriétaire     */
   PtrListDesc  MailList;       /* La liste des mails non lus    */
   int          LastAccessDate; /* La date du dernier accès      */
} TypeMailBox;
```

La procédure gérant l'acheminement des messages dans les boîtes à lettres de chaque destinataire ne pose pas de problèmes de routage, puisque le système est centralisé dans sa configuration actuelle. Il suffit alors de dupliquer le message et de rajouter cette copie dans la liste `MailList` du destinataire. Pour des contraintes de séquencement décrites précédemment, un signal est parallèlement envoyé au processus destinataire, et ce dernier est réveillé si besoin est.

3.5.1.3 La sémantique des messages

Après une description générique de l'outil, ce paragraphe décrit l'utilisation qui en est faite, dans le cadre du système de segmentation d'images présenté. Cet outil est appliqué dans l'étape de synchronisation qui précède l'étape de fusion de régions décrite au paragraphe 2.4.6.3. La figure 2.22 présente les étapes successives qui mènent à la synchronisation, permettant à un processus région de fusionner avec une autre région.

[20] Il est ponctuel, dans le sens où il ne se produit qu'une seule fois dans la vie du processus.

Code	Signification du msg	Paramètres
SYNC_REQUEST	Requête de synchro	Numéro d'étape pour la mise à niveau
SYNC_ACCEPT	Acceptation de la synchro	Numéro d'étape pour la mise à niveau
SYNC_REJECT	Rejet de la synchro	
SYNC_TERM	Terminaison de la synchro	

TAB. 3.4: La sémantique des messages échangés au cours de la synchronisation entre deux régions.

Dans cette optique, les messages échangés prennent une signification. Les contraintes de synchronisation ont permis d'en identifier plusieurs, référencés en table 3.4. Un processus région sollicite une région voisine pour réaliser une fusion. Chaque région est à un état de construction variable. Le but de la synchronisation précédent la fusion, est de mettre à niveau la région cible, afin que celle-ci se retrouve au même état d'avancement à l'instant de sa fusion. Le paramètre passé dans le message `SYNC_REQUEST`, permet au processus cible de comparer l'état d'avancement du processus qui sollicite la fusion avec son propre état, et ainsi de répondre positivement par `SYNC_ACCEPT`, si la mise à niveau est possible. La fin de la mise à niveau par le processus cible se traduira par l'expédition d'un message `SYNC_TERM` au processus initiateur.

La sémantique des messages échangés à cette occasion constitue un protocole très simple de négociation. Cette sémantique est complètement indépendante du gestionnaire de message, et peut donc aisément être étendue, modifiée, et affinée. Le fonctionnement du protocole passe par le respect de chacune de ces étapes au sein de chaque processus qui y prend part.

3.5.2 La communication par échanges de signaux

Le gestionnaire de signaux a été présenté dans le paragraphe 3.4.3.3. Les différences majeures entre les échanges de message et les signaux montrent la complémentarité de ces deux outils :

- Les messages peuvent contenir des données importantes et complexes, tandis que les signaux sont uniquement porteurs d'une information binaire.
- Le processus peut stocker plusieurs messages, alors qu'un seul signal par type peut être conservé.
- Le message est porteur d'une information plus riche en sémantique que ne l'est le signal.
- Le message permet d'identifier son expéditeur.
- Les messages mettent en œuvre une structure lourde, tandis que la gestion des signaux repose sur des objets simples, deux champs de bits uniquement.
- L'expédition de messages peut se faire en *multicast*, c'est-à-dire à plusieurs destinataires à la fois, qui recevront chacun une copie du message original [21]. Le signal a un unique destinataire.
- La sémantique du message est contenue dans le message lui-même. La sémantique du signal est contenue dans la structure de réception du processus destinataire.

[21] Cette fonctionnalité n'est pas actuellement exploitée dans le système.

3.6 Discussion et perspectives

3.6.1 Conclusions

L'objectif de ce chapitre a été de montrer dans quelle mesure une philosophie d'implémentation de type système d'exploitation pouvait s'avérer utile dans la gestion d'un ensemble de processus de segmentation coopérants.

Il a semblé particulièrement intéressant de maintenir aussi longtemps que possible une séparation entre d'une part des processus, dont le rôle est d'effectuer une tâche de traitement d'image, et d'autre part un système devant assurer le fonctionnement concurrent de ces processus, comme autant de tâches anonymes de son propre point de vue. Les processus mettent en œuvre des algorithmes de segmentation. Ils disposent d'une connaissance sur l'évaluation des pixels et des primitives, sur la localisation des éléments d'intérêt dans l'image, sur les mécanismes à mettre jeu entre eux, pour coopérer, pour communiquer, pour négocier, pour s'exploiter mutuellement, etc. Le séquenceur, quant à lui, assure la cohabitation et le fonctionnement d'une famille de processus, sans disposer de connaissances particulières sur la sémantique de leurs actions. Il gère le parallélisme des traitements, la transmission des données, la communication inter-processus. Il limite les ambitions individuelles pour la cohérence de l'ensemble. *Le séquenceur joue un rôle d'arbitre, sans vraiment savoir de quel jeu il s'agit.*

A ce titre, nous pensons que le système de gestion de processus présenté est largement adaptable :

- Tout d'abord vers une plus grande diversité de types de processus de segmentation. La plus rude tâche consiste à définir les moyens de faire coopérer ces nouvelles entités avec les méthodes de segmentation existantes. Ces considérations relèvent du chapitre précédent, et des extensions ont déjà été suggérées. La seule contrainte envisageable serait de devoir incorporer en conséquence de nouveaux outils de niveau système rendus nécessaires par ces nouveaux processus. Citons par exemple des outils permettant d'utiliser des données partagées par certains processus, autrement qu'en utilisant le médium de communication que représente l'image.
- Ensuite vers d'autres types d'applications, pas nécessairement dans le domaine de la vision, manipulant une population réactive d'individus, avec des possibilités de reproduction, et disposant d'un pouvoir cognitif important. Les simulations démographiques pourraient, à titre d'exemple, constituer une généralisation intéressante de ce système.

3.6.2 Autres approches

L'utilisation du paradigme du système d'exploitation ne constitue pas la solution unique au problème de la mise de commun d'entités hétéroclites censées coopérer entre elles. La recherche dans le domaine des systèmes multi-agents [22] peut également offrir des voies de réflexions vers d'autres styles d'implantation.

Les SMA constituent le domaine de l'Intelligence Artificielle Distribuée. L'IAD est une extension logique de l'Intelligence Artificielle, dont l'appellation laisse à penser (souvent faussement) que l'objectif est de construire des systèmes présentant extérieurement toutes les caractéristiques de l'intelligence humaine [Fer95]. Cette définition, restreinte au cas d'une unique entité se généralise aisément à la mise en commun de plusieurs agents, et constitue les bases des SMA. Parallèlement aux concepts internes régissant les agents se posent de nouveaux problèmes en termes de partage d'informations, d'interaction et de buts [FG88]

[22] appelé SMA par la suite.

La gestion d'une société d'agents a conduit à deux types d'approches différentes, *cognitive* ou *réactive*. Dans un premier cas, les agents présentent un aspect cognitif très développé, et la dynamique de résolution du problème repose essentiellement sur la compétence propre de chaque entité du système [BG93], réagissant comme autant de mini-systèmes experts. Dans un deuxième cas, les concepteurs de systèmes misent davantage sur la réactivité du système, en adoptant le présupposé qu'il n'est pas indispensable que les agents soient individuellement intelligents pour qu'un comportement de groupe cohérent puisse cependant émerger. Nous pouvons citer les travaux de Luc Steels [Ste90], qui illustre ces principes dans le domaine de la robotique mobile. Brooks [Bro90], [Bro91] utilise les mêmes principes appliqués à la robotique. Il argumente en particulier qu'il n'est pas nécessaire de disposer de mécanismes complexes pour interpréter les informations fournies par les capteurs de ses robots afin de donner extérieurement un aspect cohérent à leur fonctionnement. Des mécanismes simples et réactifs aux informations provenant des capteurs et se combinant entre eux sont souvent davantage efficaces.

La communication entre les agents utilise deux approches, souvent induites par des nécessités techniques, le *tableau noir* ou les *envois de messages*. Les approches par tableau noir permettent un partage des informations, chaque agent pouvant accéder à l'information apportée par un autre agent. Ce mode de communication est par exemple utilisé dans BB-1 [HR85]. Les approches par envoi de messages sont utilisées par exemple dans les langages à acteurs [Hew77]. Elle permettent une communication directe entre deux agents, sans passer par une unité de stockage intermédiaire.

La mise en œuvre de la coopération entre agents peut s'envisager sous plusieurs aspects, par exemple la *passation d'appels d'offre*, ou bien encore *l'échange de solutions partielles*. Dans le premier cas, un agent va sous-traiter la résolution de certains problèmes, issus de la décomposition de sa tâche principale en sous-tâches élémentaires. Il va donc demander à l'agent le plus compétent d'effectuer la sous-tâche pour laquelle il sera jugé le plus adapté [RD83]. Dans le deuxième cas, des solutions partielles sont échangées entre les agents, jusqu'à ce qu'une solution globale satisfaisante se dégage [CL83].

Chapitre 4

Un système ouvert

4.1 Introduction

Les précédents chapitres ont présenté un système de segmentation utilisant un grand nombre de processus fonctionnant conjointement sur l'image. La place et le rôle de l'utilisateur n'ont, jusqu'à présent, pas été clairement définis dans le cadre de ce système. Tout semble avoir été prévu pour que la population de processus évolue en dehors de tout contrôle humain, pour peu que celui-ci ait procédé à une initialisation correcte. Le système de vision est décrit comme une entité autonome et déterministe.

Cependant, il est légitimement souhaitable que l'utilisateur prenne une place plus importante dans le fonctionnement même du système de vision. L'architecture utilisée se prête particulièrement bien à une telle extension, puisqu'il est très simple de considérer l'utilisateur, comme un processus supplémentaire, pouvant agir lui aussi, à son niveau, sur le système sans perturber le fonctionnement des autres processus de segmentation.

Cette discipline, visant à étudier les moyens proposés à l'utilisateur par un programme pour contrôler son fonctionnement [1], porte le nom générique d'interaction homme-machine. Elle adresse donc l'étude des points de contrôle offerts à l'utilisateur pour une application donnée afin de lui permettre de la manipuler avec le plus d'aisance et d'efficacité possible. Nous allons étudier la manière dont certaines de ces notions peuvent s'intégrer dans le domaine de la vision par ordinateur et plus particulièrement à un système de vision de bas niveau.

La conception d'interfaces entre l'utilisateur et une application informatique pose des problèmes récurrents, souvent indépendants de l'application, et qui sont posés par l'élément le plus versatile et le moins fiable de la chaîne des traitements, son élément terminal : l'homme. L'utilisateur présente des besoins et des caractéristiques souvent très variables par rapport à une application, qui par définition a fixé un certain nombre de fonctionnalités au sein d'un programme exécutable. Il n'est donc pas évident que les fonctionnalités prévues par le programmeur, même si elles sont nombreuses, puissent convenir au besoin d'un utilisateur particulier. Une autre particularité d'une interface sera de tenter de sécuriser les opérations effectuées sur le système, ainsi que de permettre d'améliorer son efficacité. Une approche classique consiste par exemple à introduire plusieurs niveaux de contrôle, présentés à l'utilisateur en fonction de son niveau d'expertise dans le domaine de l'application. L'idée sous-jacente étant de fournir à l'utilisateur "lambda" le minimum de points de contrôle, dont il puisse facilement comprendre la signification, sans le noyer sous des détails plus obscurs, relevant plutôt des mécanismes internes de l'application, et qui concernent davantage à ce titre l'utilisateur expert.

Le besoin de convivialité a poussé les développeurs d'IHM (interface homme-machine) vers

[1] On parle de l'ergonomie d'un programme pour l'utilisateur, c'est-à-dire de son confort d'utilisation.

l'idée que personne mieux que l'utilisateur ne pouvait définir l'environnement de travail qu'il souhaitait utiliser. Ainsi, des systèmes ont permis à l'utilisateur de créer, puis de disposer de son propre environnement de travail avant même de pouvoir accéder à l'application. L'équipe italienne de Piero Mussio a tenté de réaliser un tel système dans le domaine de la vision par ordinateur. L'objectif de leurs travaux est de permettre l'analyse d'image d'électro-cardiogramme à partir d'outils de traitement d'image [MFGC92],[BBMP93]. L'utilisateur commence avec un espace de travail vierge et peut créer et disposer à sa guise les objets avec lesquels il va travailler. Il sélectionne les différents types de sortie pour chaque objet. Une communication s'opère alors entre les objets sélectionnés.

Cette approche séduisante présente conceptuellement un défaut rebutant pour le néophyte, à cause de l'important travail préliminaire qui lui est demandé avant de pouvoir accéder à l'application. La construction de l'environnement de travail initial nécessite de prendre connaissance de nombreux détails sur le fonctionnement de l'application, sur les notions d'entrée/sortie, avant de pouvoir définir le premier objet dans l'environnement de travail. L'interface que nous proposons présente les caractéristiques inverses, dans le sens où l'utilisateur est tout de suite capable de manipuler le système, ceci sans avoir besoin de connaissances particulières sur les aspects internes du fonctionnement du système.

Nous pensons que les interfaces homme-machine doivent évoluer vers des architectures davantage tournées vers des approches d'apprentissage, afin de pouvoir dynamiquement modifier leur interface graphique ainsi que les points de contrôle qu'elles présentent, afin de se rapprocher au plus près d'un profil d'utilisateur tel qu'elles "le perçoivent". La complexité de ce genre d'approches les rend assez rares dans la littérature.

4.1.1 Les IHM dans le domaine de la vision

L'état de l'art est faible à la frontière diffuse entre les IHM et la vision par ordinateur, malgré l'activité de chacun de ces domaines séparément. On s'accorde pour hiérarchiser les approches de la vision par ordinateur en deux niveaux, le premier étant réservé aux traitements de base effectués sur l'image, afin d'en extraire les primitives significatives. Ces méthodes travaillent donc fréquemment sur les informations photométriques de l'image en appliquant des techniques à base de filtrage [Der87],[Can86]. Le second niveau a davantage pour but des interprétations structurées de la scène, et utilise à cet effet les informations collectées au bas niveau. Ces systèmes utilisent une représentation complexe des données, et les manipulent par des méthodes à base de règles logiques [RM89], des systèmes experts, ou encore des systèmes à base de connaissances [DCB+89]. On dénombre assez peu de travaux qui ne se situent pas dans cette stratification du domaine, en introduisant par exemple des connaissances de haut niveau pour résoudre des problèmes de segmentation de bas niveau [NL84], [Mat89].

Les IHM sont nécessaires aux applications de haut niveau, dans lesquelles l'utilisateur dispose d'un rôle de coordinateur, pour sélectionner les buts, orienter l'utilisation des connaissances, construire des plans d'action, ou encore résoudre les problèmes de conflits. De tels systèmes de haut niveau ont naturellement nécessité une IHM, car les mécanismes mis en œuvre ne sont pas autonomes. Nous proposons au contraire d'introduire une IHM dans une situation où elle n'est pas indispensable paradoxalement. Elle serait davantage un moyen de contrôle en temps-réel d'un système autonome par ailleurs.

De fait, la vision par ordinateur de bas niveau s'intéresse à l'extraction de primitives de base de la structure de l'image, souvent sans utiliser de connaissances sur le domaine d'application. Elle apparaît alors comme le développement de stratégies multiples, souvent empiriques, afin de déterminer l'ordre idéal dans lequel il convient d'appliquer une série de filtrages sur l'image, chacun d'eux disposant d'un jeu de paramètres ajustable, étroitement liés les uns aux autres par ailleurs. L'utilisateur dispose donc des images brutes d'une part, et d'une imposante librairie d'opérateurs de traitement d'image d'autre part. Il lui revient la tache délicate consistant à décider quels opérateurs appliquer, dans quel ordre, et avec quel jeu de paramètres. Certains systèmes de vision ont pour but d'assister l'utilisateur dans cette étape. Tamura [TSY+89] a développé un environnement nommé *View Station*, dont l'objectif est de faciliter l'écriture de programmes de traitement d'image. Matsuyama [Mat89] a développé *LLVE*, un système expert contenant de la connaissance sur un ensemble d'opérateurs pouvant être appliqués sur l'image. Le rôle de ces IHM est alors d'assister le travail de prospective de l'utilisateur dans le but de développer la meilleure séquence algorithmique pour résoudre un problème de vision donné. Le système de vision ainsi généré pourra fonctionner ensuite de manière autonome, et l'IHM n'est utilisé ici que dans une étape de développement. L'étape d'exploitation fonctionne de manière indépendante. Il nous semble intéressant au contraire d'orienter une IHM sur l'exécution même de l'algorithme plutôt que sur sa phase d'ajustement.

Pour d'autres, la vision par ordinateur peut enfin apparaître comme un problème décisionnel purement algorithmique [Mar82], [SG92], dans lequel les décisions prises par l'algorithme à un instant donné ont une importance cruciale sur les traitements qui suivent. Salotti a proposé des principes allant dans ce sens pour la vision de bas niveau : accumuler de l'information locale afin de prendre des décisions plus robustes, reporter les décisions délicates, faire des choix adaptatifs, considérer l'information locale et contextuelle de l'image. Avec un même souci sur les choix algorithmiques, Capdevielle [Cap95] réalise une étude sur la manière de formuler des buts en segmentation d'image. Il introduit notamment une typologie précise des modes et des niveaux d'interaction : description des connaissances sur l'image, des buts, des connaissances sur les opérateurs, etc. A la différence des algorithmes de type "boite noire", où par définition l'utilisateur dispose de peu de points de contrôle, une telle approche de la vision de bas niveau, où les prises de décision constituent des points de contrôle visibles de l'algorithme, offre un important champ d'actions pour l'élaboration d'une IHM, pour laquelle l'utilisateur pourra agir finement sur un algorithme ouvert.

La réalisation d'une interface entre l'homme et la machine est selon nous une excellente opportunité pour introduire une connaissance de haut niveau apportée par l'utilisateur au sein d'un système de bas niveau. Ce niveau de coopération doit être finement dosé, en laissant le système fonctionner la plupart du temps de manière autonome. L'utilisateur a ainsi toute latitude pour se concentrer sur certaines actions spécifiques, qui peuvent de façon opportuniste influencer le comportement du système. La difficulté d'une telle interface est de fournir à l'utilisateur un ensemble de points de contrôles suffisamment utiles, intuitifs et variés. L'utilisateur peut alors orienter davantage sa réflexion sur *le type* de tâche à réaliser, plutôt que sur *la façon* de réaliser une tâche.

4.2 L'Interface Homme-Machine

4.2.1 Une interface à plusieurs niveaux

Nous proposons une interface à trois niveaux, qui permet de s'adapter à la multiplicité des utilisateurs, et à leur degré d'expertise.

Le niveau de base laisse fonctionner le système de segmentation de manière autonome. Les processus se développent en fonction des données de l'image, et utilisent les seuils et paramètres standards définis dans le système. Ces paramètres offrent des résultats satisfaisants dans un grand nombre de cas classiques. Le résultat final est obtenu sans nécessiter l'intervention de l'utilisateur. Ce niveau laisse fonctionner le système comme si aucune IHM n'avait été implémentée, et ne nécessite donc aucune connaissance particulière de la part de l'utilisateur sur les mécanismes internes du système.

Le niveau intermédiaire propose une interaction simplement intuitive avec l'utilisateur, par le biais d'actions simples à la souris (citons par l'exemple la suppression d'une simple primitive région ou contour à la souris, alors même que le système poursuit sa segmentation).

Le niveau d'interaction le plus élevé permet de contrôler plus finement l'exécution du système, en influant sur des paramètres plus internes. En contrepartie, ce genre d'interaction requiert une plus grande expertise de la part de l'utilisateur dans le domaine de la vision et sur le comportement du système qu'il manipule. Il a la possibilité d'ajuster certains paramètres, tels les pondérations qui définissent les fonctions d'évaluation, et certains seuils de décision. Pour ce faire, il doit avoir une représentation de l'effet probable de ses actions, afin de pouvoir diriger le système pour qu'il réponde selon ses désirs.

4.2.2 Une interface homme-machine en temps réel

L'utilisateur doit pouvoir agir sur le système en temps-réel. L'interface doit rendre possible cette action, avec le minimum de délai dans le meilleur des cas. Ce court délai est indispensable par exemple pour des actions de correction d'erreur, où il est indispensable que l'utilisateur puisse réagir dès qu'il décèle une anomalie, sans devoir attendre la fin de l'exécution. Cette réactivité est indissociable d'une certaine inertie du système de segmentation, dans le sens où un tel contrôle devient inutile lorsque le système a une durée d'exécution trop courte par rapport au temps de réaction de l'utilisateur. Le cas ne se pose pas dans notre système.

Le rôle de l'utilisateur ne doit donc pas se limiter au simple ajustement de quelques paramètres initiaux. L'interface doit lui permettre d'agir alors même que le système est en cours de fonctionnement. Les corrections d'erreur sont alors possibles, juste après que ces dernières aient été commises, et pas seulement en fin d'exécution. L'utilisateur n'est donc plus un spectateur passif devant le travail de son système de segmentation. Il devient un acteur dont les connaissances de haut niveau sont transmises sous formes d'actions vers le système de segmentation.

Cette interaction en temps réel doit s'insérer de façon transparente au sein du système de segmentation. Plusieurs possibilités sont envisageables à ce niveau :

- Le système de segmentation et l'interface graphique fonctionnent de façon séparée et parallèle sous la forme de deux processus distincts. Il convient alors de suspendre le système de segmentation temporairement, afin de procéder aux actions de l'utilisateur. La cohérence des structures de données nécessite que le système ne fonctionne pas durant la courte section critique où l'interface utilisateur va modifier les données du système. Une telle implémentation est concevable en définissant l'IHM et le système de segmentation comme deux processus affiliés, et utilisant une mémoire partagée.
- L'interface est intégrée au système de segmentation, et est activée par le biais d'événements graphiques. Les deux entités sont rassemblées au sein d'un même processus. Les problèmes de suspension de tâches ne se posent donc plus, puisque le système ne s'exécute plus au moment où sont invoquées les procédures de gestion des événements de l'interface graphique. Cette deuxième solution a été retenue, d'abord pour sa relative facilité d'implémentation par rapport à la proposition précédente, et ensuite pour des raisons de cohérence : les processus de segmentation du système lui-même ont été intégrés selon ce principe. Cela permet à l'utilisateur de devenir un acteur à part entière dans le système de segmentation, au même titre que n'importe quel autre processus.

Une interaction en temps-réel sur le système est indissociable de la possibilité de revenir en arrière, afin par exemple d'annuler les effets d'une action introduite précédemment par l'utilisateur. Il est particulièrement important que l'utilisateur se sente en sécurité lorsqu'il manipule l'interface, et qu'il réalise qu'aucune de ses actions ne peut causer une erreur irrécupérable dans le système, et que toute manipulation faite peut en cas de doute être défaite.

4.2.3 Les caractéristiques d'une interface temps-réel en vision par ordinateur

Les caractéristiques d'une interface homme-machine sont décrites sur l'exemple particulier du système de vision étudié dans les précédents chapitres. Cette interface devrait permettre, du point de vue de la segmentation, de fournir un résultat plus complet, de meilleure qualité, de s'adapter, et à terme d'apprendre à reproduire le comportement de l'utilisateur. Nous proposons que l'interface permette à l'utilisateur de compléter, de corriger, de modifier, d'ajuster les éléments de ce système de vision, et à terme de "montrer" de nouvelles connaissances :

- L'interface doit permettre de réaliser une segmentation de l'image plus complète. L'utilisateur peut souhaiter poursuivre la segmentation après que le système ait terminé son exécution dans le but de compléter le résultat déjà obtenu. L'objectif est de forcer le système à retravailler sur une carte de régions et de contours issue d'une précédente segmentation. Cela peut être le cas par exemple car l'utilisateur pense que ce résultat ne contient pas suffisamment de primitives contours et régions détectées, ou bien car d'importantes primitives (importantes du point de vue d'un utilisateur particulier – deux utilisateurs n'ont pas forcément les mêmes nécessités de segmentation) n'ont pas pu être segmentées par le système.
- L'interface doit permettre de réaliser une segmentation plus correcte de l'image. Les erreurs qui ont pu être commises par le système dans le mode d'exécution autonome doivent pouvoir être corrigées. L'utilisateur peut ainsi détruire une grande région qui recouvre plusieurs objets distincts de l'image, il place manuellement des germes de processus de type région à l'intérieur de chaque objet devant être segmenté individuellement. Il tire ainsi profit du pseudo parallélisme qui guide la segmentation : les deux processus régions, sous réserve que les germes soient correctement placés, travaillent simultanément. Deux objets de type région seront donc trouvés.
- Le comportement des processus de segmentation peut être modifiable en temps et en espace, par l'ajustement manuel des seuils des processus en cours d'exécution, ou des processus de même type qui travaillent à des endroits différents de l'image. Le comportement est modifié dans le temps, si tous les nouveaux processus créés postérieurement à cette modification sont affublés des nouveaux attributs en vigueur. Il est modifié dans l'espace si ces modifications s'appliquent à des processus de même type, situés à différents endroits de l'image.
- L'utilisateur peut appliquer localement des stratégies d'essai-erreur. Il ajuste certains paramètres, lance des processus dans l'image, stoppe le système, efface les primitives obtenues, et ajuste encore les paramètres jusqu'à ce qu'il juge suffisante la qualité des résultats obtenus. L'interface autorise d'appliquer cette stratégie très localement, sur des zones très réduites de l'image, par exemple pour un seul contour, ou une seule région que le système ne parvient pas à segmenter correctement avec les paramètres par défaut au cours d'un fonctionnement autonome. La localité permet d'une part un gain appréciable en temps de calcul, puisque la stratégie d'essai-erreur ne s'applique que sur l'élément déterminant, et pas sur l'ensemble de l'exécution du système. La localité permet d'autre part de ne pas introduire d'erreurs supplémentaires à d'autres endroits de l'image durant

l'ajustement de l'essai-erreur. Tout le reste des primitives segmentées reste ainsi figé durant cette stratégie d'ajustement.

– L'objectif de l'utilisateur est d'introduire une expertise supplémentaire au sein du système de segmentation construit antérieurement. Le système devrait à terme pouvoir imiter le comportement de l'utilisateur pour l'appliquer par extension à d'autres configurations similaires au problème que l'utilisateur tente de résoudre par son interaction. Un système à base d'apprentissage pourrait être adapté pour affronter ce problème. L'utilisateur devient un “tuteur” pour l'interface, dont le savoir-faire permet alors d'enrichir une bibliothèque d'actions. Au rôle classique de l'interface s'ajoute celui d'incorporer à bon escient la connaissance de l'utilisateur ainsi stockée, dans le but d'anticiper les actions prévisibles que ce dernier pourrait avoir sur le système. Ce point apparaît extrêmement puissant, mais également d'une grande difficulté à mettre en œuvre.

4.3 Implantation

L'interface homme-machine présente la particularité d'avoir été développée longtemps après l'écriture de la première version du système de segmentation. Les travaux d'implantation ont été en partie réalisés par Fabien Loubiat [Lou95] au cours d'un stage de DEA réalisé au sein du laboratoire TIMC-IMAG. Ce système fonctionnait ainsi auparavant à l'aide de fichiers de démarrage, qui contenaient tous les paramètres du système, et qui étaient chargés à l'initialisation. Le système de segmentation était donc une entité autonome, qui fonctionnait sans intervention humaine. L'interface apparaît donc ici comme un processus supplémentaire, avec un caractère proche de celui d'un parasite, ou d'un virus (se développer au détriment de son hôte, prendre le contrôle sur lui, de manière transparente, à son insu). Sylvain Giroux dans ses travaux de thèse [Gir93], puis plus tard dans des travaux connexes [GSL94], introduit la notion d'agent *épiphyte*, dans le cadre d'un processus de méta-niveau chargé de surveiller un système de base. Des agents épiphytes se caractérisent par des aspects parasitaires semblables à ceux de notre interface graphique.

L'interface homme-machine travaille en parallèle avec le système d'exploitation. Elle peut modifier le comportement de ce système au moment où l'utilisateur le décide, mais ne prendra pas d'initiative autonome par ailleurs. Le système de segmentation est donc en conséquence autonome si l'utilisateur n'agit pas sur l'interface. Dans la pratique, le niveau de base de cette IHM est donc réalisé sans qu'il ne soit nécessaire de rajouter une ligne de code, figure 4.1.

L'IHM utilise les fenêtres de visualisation des informations du système comme des fenêtres de dialogues afin de collecter les actions de l'utilisateur. La première fenêtre est l'*image résultat* (result window), qui présente l'état d'avancement de la segmentation, montrant à la fois les contours et les régions dans une même image. L'utilisateur peut observer les primitives qui se construisent. La seconde fenêtre de visualisation est l'*image de l'arbre des processus* (link window) qui présente les liens de filiation associant les processus dans l'image. Chaque processus est représenté par l'emplacement de son pixel germe, la forme de cet arbre épouse naturellement les structures de l'image, et est mise en contexte par rapport à l'environnement de la segmentation, par superposition à l'image des niveaux de gris. L'*image résultat* offre des contrôles sur la croissance des primitives régions et contours. L'*image de l'arbre des processus* offre des contrôles relatifs aux processus eux-mêmes, en donnant accès à leurs structures internes par des panneaux de configuration.

Du point de vue architectural, la nature du système de vision envisagé, fonctionnant sur le modèle d'un système d'exploitation, nécessitait de respecter ce fonctionnement lors de la création de l'interface. Ainsi, les fonctionnalités existantes et le caractère autonome du système

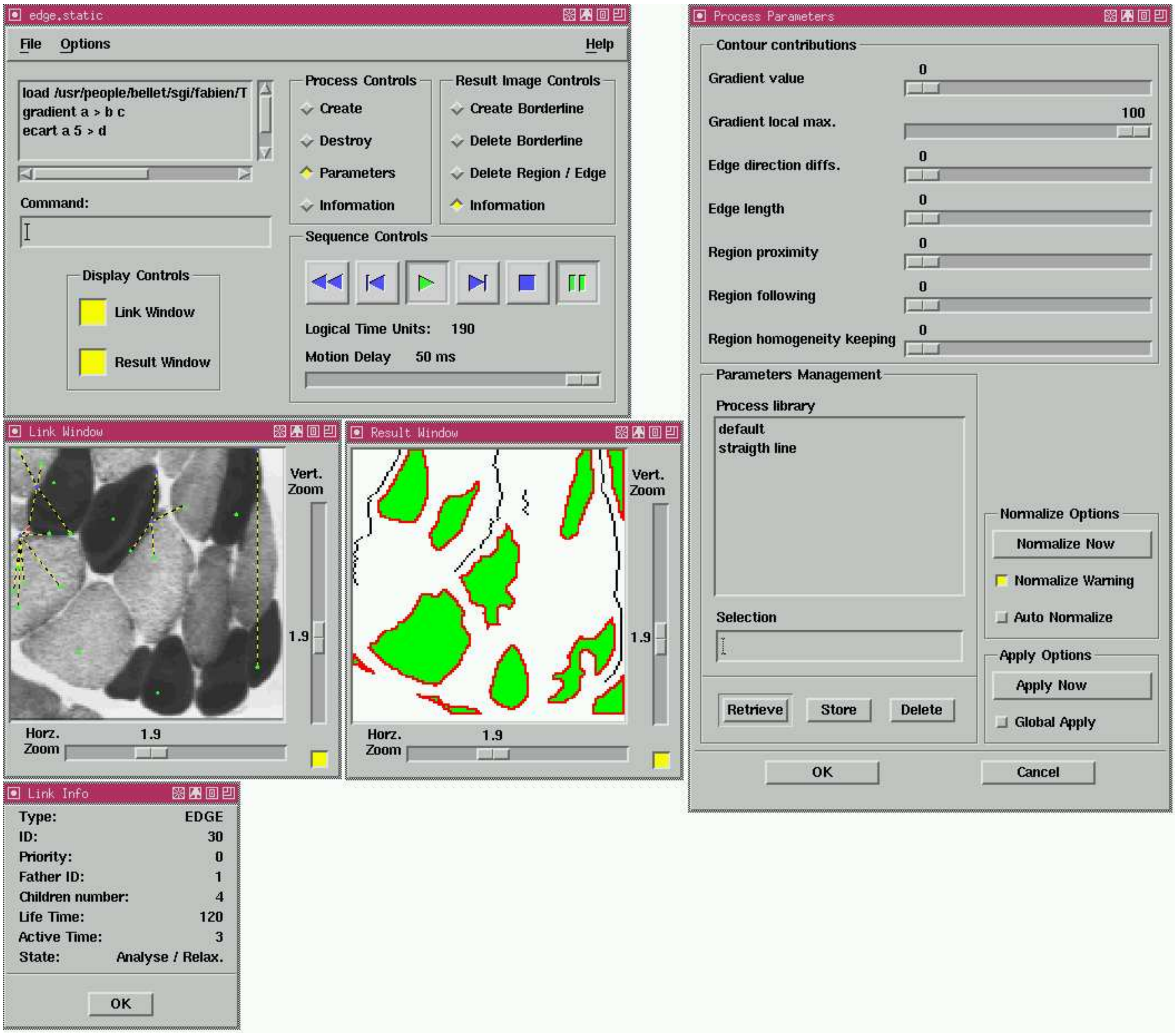


FIG. 4.1: Vue globale du système de segmentation : la fenêtre principale est *edge.static*, elle contient les contrôles de séquencement ; *Link Window* présente les relations de parenté entre les processus ; *Result Window* montre les résultats courants de la segmentation ; *Link Info* indique les caractéristiques d'un certain processus qui a été sélectionné par l'utilisateur et *Process Parameters* offre un accès à un panneau permettant de contrôler les comportements d'un processus de segmentation.

ont été préservés, rendant plus difficile une implantation classique fondée sur le modèle événementiel d'une interface. Il a donc fallu greffer à l'application existante, séquentielle de nature, une architecture permettant de traiter les messages en provenance de l'environnement graphique de manière asynchrone. Les contraintes précédentes ont fixé quelque peu le cadre de développement de l'application, le fonctionnement de celle-ci est basé sur le modèle présenté en figure 4.2.

4.3.1 Les contraintes du modèle événementiel

L'interface a été réalisé sous système XWindow, avec la librairie d'objets d'interface standard OSF [2]/ Motif. Ce choix d'environnement concernant l'environnement était naturel, puisque le système initial avait été développé sous ce même environnement, et que la librairie OSF/Motif s'impose de fait comme le standard dans l'implémentation des objets d'interface, puisqu'elle est maintenue de manière permanente, et qu'elle est très complète en matière de contrôles de haut niveau mis à disposition du programmeur.

4.3.1.1 Les contraintes imposées par le système XWindow

L'architecture du système XWindow est basée sur un modèle client-serveur, ce qui peut provoquer des problèmes, lors de l'implémentation de traitements en temps-réel, comme dans le cas qui nous concerne. En effet, lorsque, par exemple, l'utilisateur déplace le pointeur de souris à l'intérieur d'une fenêtre qui appartient à l'application, le serveur XWindow va générer un message adéquat (ici MouseMove) à destination de la fenêtre concernée, et l'application a ensuite la charge de réagir en fonction du message reçu. Mais le protocole X ne prévoit pas de synchronisation entre le moment où l'utilisateur déplace effectivement la souris, et le moment où sera traité le message correspondant dans l'application, qui est l'instant où l'utilisateur pourra véritablement constater le résultat de son action. Le serveur traite les événements dans leur ordre d'arrivée, et il peut s'écouler un certain temps entre le moment où l'utilisateur doit provoquer la génération du message en déplaçant la souris, et le moment où l'application reçoit effectivement le message. Même si on peut forcer la priorité de certains messages générés, il n'est pas possible d'assurer une parfaite synchronisation avec ce type d'architecture. Ce problème a d'ailleurs amené certains chercheurs spécialisés dans les systèmes d'interaction à proposer un autre type d'architecture permettant de forcer le traitement en temps réel, et ainsi de passer outre ces difficultés [BCW93].

4.3.1.2 Les XtIntrinsics

Le système XWindow comprend une interface de programmation de "bas niveau" : la "XLib", permettant d'accéder à toutes les fonctions fondamentales de manipulation des messages et de gestion des fenêtres. Cependant, on peut considérer qu'elle est lourde à utiliser. Une couche supplémentaire (les "XtIntrinsics") au dessus de la XLib a donc été conçue, notamment pour permettre la création aisée d'objets graphiques (les *Widgets*), et pour faciliter le traitement des messages, puisque ceux-ci sont automatiquement acheminés vers des routines de traitement appropriées pour chaque objet graphique : les *Callbacks*. Il s'agit ici d'une approche orientée objet, mais dont l'inconvénient majeur est qu'elle est centralisée : tous les messages à traiter par l'application passent par une sorte d'aiguilleur qui se charge de les répartir sur les callbacks adéquats.

---

[2] Open Software Foundation

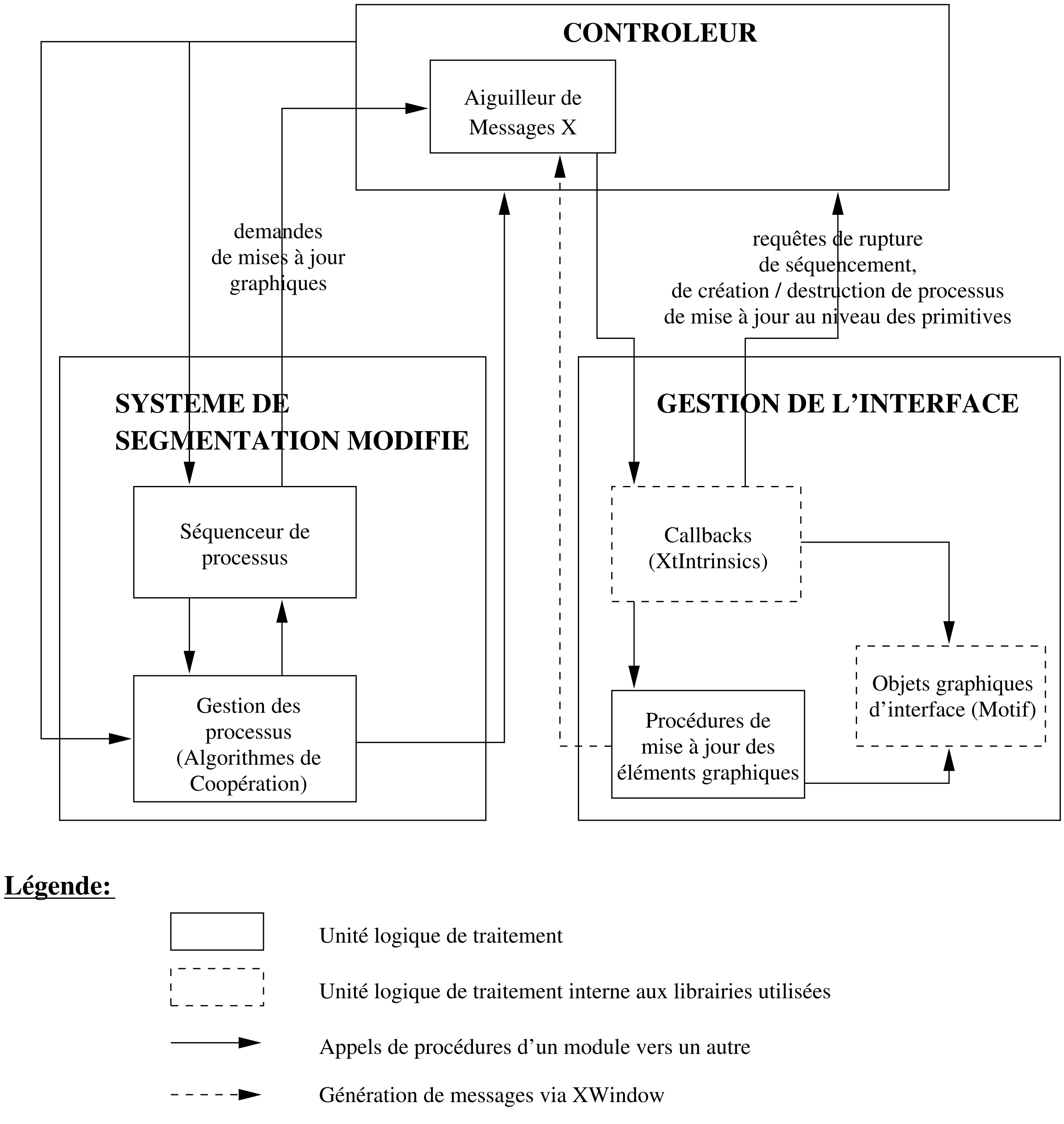


Fig. 4.2: Architecture générale de l'application. L'interface a été écrite en utilisant la bibliothèque d'objets graphiques Motif. Le schéma fonctionnel montre la modularisation effectuée, ainsi que les principaux liens de contrôles.

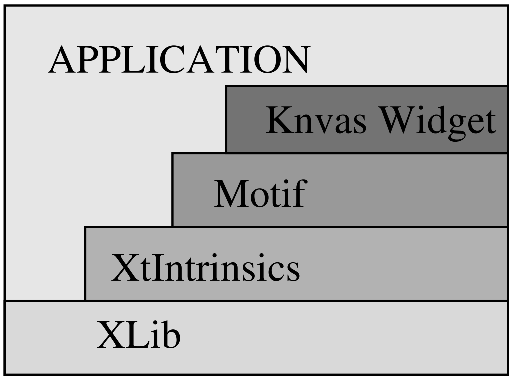


FIG. 4.3: Les niveaux d'abstraction de l'application.

4.3.1.3 La librairie Motif

La librairie Motif est construite elle-même sur la base des widgets de XtIntrinsics, et permet une apparence normalisée pour les objets graphiques, et offre de nombreux contrôles de gestion s'y rapportant.

Il est cependant dommage qu'aucun objet de gestion d'affichage de primitives rudimentaires, telles des points ou des lignes n'ait été prévue dans la librairie, ce qui oblige à utiliser pour cela la XLib... ou une librairie destinée à cet effet, comme *KnvasWidget*, mises au point par des membres du groupe Bull. Cette librairie permet entre autre de partager un même objet graphique (image, droite, cercle, ...) entre plusieurs fenêtres et facilite grandement la gestion de l'affichage des primitives graphiques. Schématiquement, on peut résumer les barrières d'abstraction présentes dans l'application du fait de l'utilisation des différentes librairies dans le diagramme en figure 4.3.

4.3.2 Le séquencement

La granularité de système de vision est définie par la boucle de séquencement des processus, voir le paragraphe 3.2.2 à ce sujet. Au sein de cette boucle, un processus est choisi, il s'exécute pendant une période de temps élémentaire [3] (en fait l'unité de temps se traduit par un nombre de pixels pouvant être agrégés à la primitive courante), puis il est replacé dans la liste des processus *actifs*.

Le choix a été fait de fournir à l'utilisateur un contrôle total au niveau de cette boucle de séquencement. Il dispose de fonctions permettant une pause, de continuer, d'exécuter le système en mode *pas à pas* [4], ou en mode temporisé (une boucle de ralentissement variable vient s'introduire au cours du séquencement). Le contrôle le plus fin est obtenu en mode pas à pas, puisque l'utilisateur peut suivre l'exécution élémentaire de chaque processus.

Une alternative aurait consisté à placer le contrôle au niveau de l'ajout du pixel, plutôt qu'au niveau du changement de processus. Ces deux contrôles ne sont d'ailleurs pas antagonistes. Un contrôle au niveau du pixel permet de travailler plus finement au sein même d'un processus. Il serait possible par exemple de déterminer la raison pour laquelle un processus de type contour bifurque à un mauvais endroit, ou pourquoi une région déborde, et de savoir exactement sur quel pixel porte la mauvaise décision. En contrepartie, ce genre de contrôle s'adresse à des utilisateurs ayant le plus de connaissances sur les mécanismes internes du système, et constitue

[3] L'appélation conventionnelle dans les systèmes d'exploitation est le *time slice*.

[4] ce qui se traduit ici par un contrôle de l'exécution processus par processus, ou encore tranche de temps élémentaire par tranche de temps élémentaire.

davantage un outils de *debuggage* qu'une aide à la segmentation.

4.3.3 Les contrôles sur la fenêtre des résultats

La fenêtre des résultats permet d'effacer des primitives déjà construites par le système, quel que soit leur état d'avancement [5]. Dès que l'utilisateur remarque que la région se développe dans une mauvaise direction (par exemple lorsqu'une région traverse une transition des niveaux de gris jugée importante par l'utilisateur), ou bien lorsqu'un processus contour suit à tort un chemin guidé par des valeurs de gradient trompeuses à l'intérieur d'une texture, il peut détruire la primitive elle-même, ainsi que le processus qui lui est associé. Toute l'arborescence des processus à partir de ce dernier sera par effet de bord également supprimée [6]. Ces processus fils correspondent en effet à des requêtes d'information formulées par le processus que l'utilisateur est en train de détruire. Ce mode de contrôle est très proche du mode d'exécution du système, et fournit ainsi un bon moyen de corriger des erreurs locales de segmentation.

Le second contrôle implémenté dans cette fenêtre de visualisation évite qu'une même primitive erronée ne soit reconstruite exactement au même endroit que la précédente. La notion de frontières fictives est introduite et permet de borner l'expansion de primitives spécifiques. Une frontière est tracée sur l'image par l'utilisateur, puis est rattachée à une primitive en construction, de telle sorte que l'ajout de pixels en dehors de cette limite est rejeté par le mécanisme de croissance. Ces barrières virtuelles (car elles n'apparaissent pas dans la segmentation finale) orientent donc la croissance des primitives dans certaines directions.

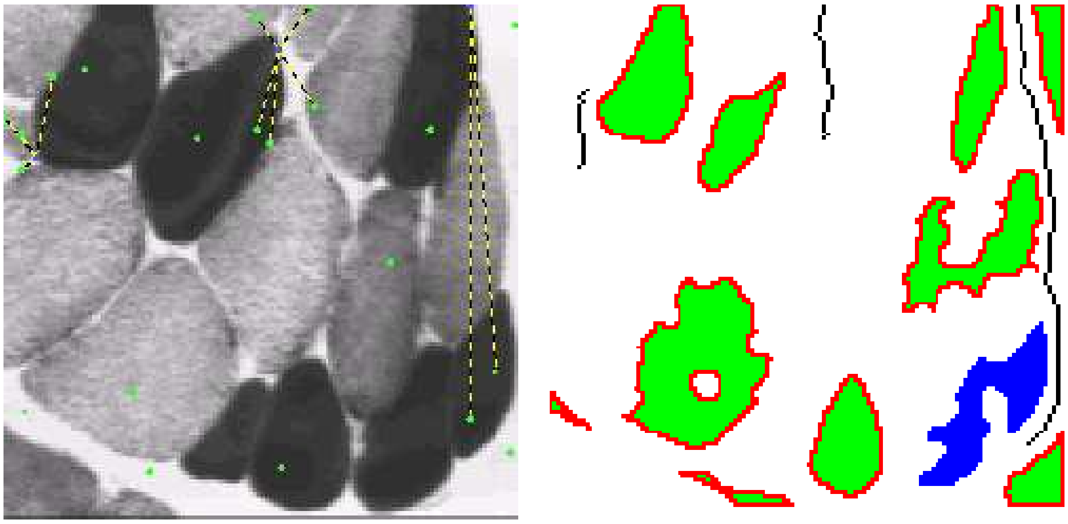
4.3.4 Les contrôles sur l'image des processus

L'utilisateur peut forcer le système à explorer certaines zones de l'image, qui ont été "oubliées" par le système, en ajoutant à ces endroits de nouveaux processus spécifiques. Il ajuste interactivement chaque paramètre de ces nouveaux processus, générés artificiellement après la terminaison normale du système. Cette création est assimilable à l'instanciation d'une classe générique dans les langages orientés objets. Un processus générique est adapté pour segmenter un objet présentant des caractéristiques classiques (par exemple pour les contours, la fonction d'évaluation de base est classiquement orientée afin de suivre les pixels ayant les plus fortes valeurs de la norme du gradient, et le seuillage effectué sur cette fonction d'évaluation est suffisamment élevé pour n'accepter que les pixels correspondant à des transitions fortes). Le création de ces nouveaux processus manuellement permet à l'utilisateur de choisir des paramétrages inhabituels, mais qui pourront être adéquats dans un cas spécifique, très localement dans l'image (certains paramétrages peuvent conduire le processus contour à suivre les valeurs plus faibles du gradient afin de mettre en évidence de faibles transitions, ou bien à suivre la frontière d'une région pré-existante).

La création de nouveaux processus peut intervenir avant la terminaison du système, et auquel cas il peut être intéressant d'attacher le nouveau processus à un père déjà existant, si besoin est. Une stratégie consiste, par exemple, à forcer un processus de type région à découvrir le maximum de primitives de type contour qui sont situées dans son voisinage. L'utilisateur crée un certain nombre de nouveaux processus contour. Il les rattache ensuite tous à la région courante. La croissance de région, si le processus était dans cette phase de son activité, s'interrompt jusqu'à ce que les processus contour aient terminé leur exécution, et que de nouveaux

[5] La granularité de base de l'interface est la boucle de séquencement. La construction d'une région complète nécessite plusieurs passages dans la boucle de séquencement.

[6] Une alternative à cette destruction en chaîne serait de rattacher toute la filiation du processus supprimé à son père.



(a) Processus (b) Résultats

FIG. 4.4: première étape : l'erreur est détectée

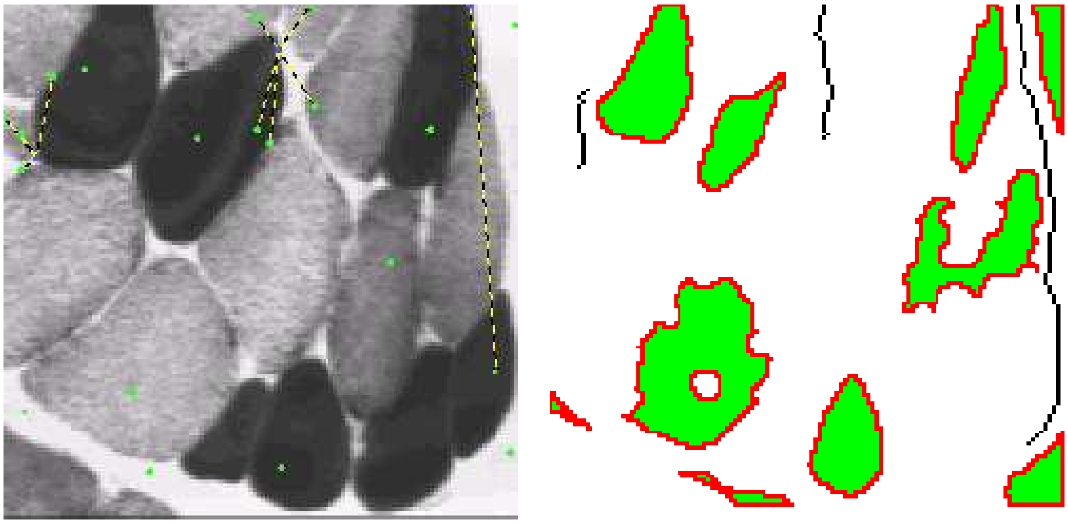
morceaux de contour aient contribué à mieux délimiter la croissance de la région.

4.4 Expérimentations de l'interface

Nous proposons dans ce paragraphe d'illustrer les principes précédents sur un exemple concret, démontrant la manière dont l'utilisateur peut interagir avec le système, en effectuant un nombre limité d'actions sur l'interface. L'image utilisée dans cet exemple est une coupe de cellules musculaires présentant des cellules noires très homogènes, et des cellules grises hautement texturées sur un fond blanc également homogène.

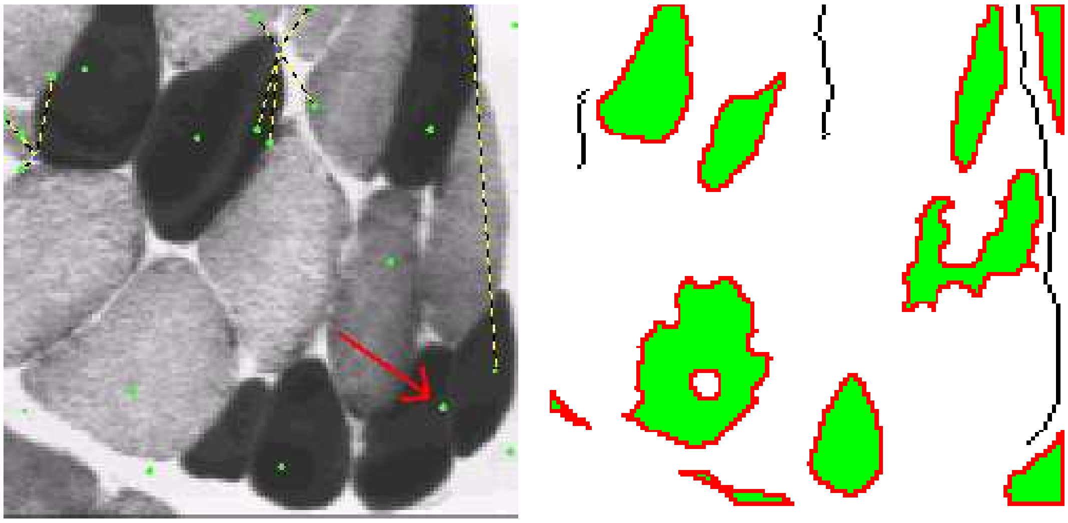
- Le système fonctionne de façon autonome jusqu'à atteindre l'état présenté sur la figure 4.4. L'utilisateur remarque une erreur commise en bas à gauche sur cette *image résultat* car deux cellules ont été regroupées en une seule région.
- L'utilisateur décide d'effacer cette primitive sur la figure 4.5. Il interrompt l'exécution du système, et ensuite efface la primitive erronée en cliquant sur cette région. Le processus associé ainsi que ses éventuels fils sont détruits.
- L'utilisateur positionne manuellement un pixel germe pour un nouveau processus de type contour, en indiquant l'endroit approximatif souhaité. Il trace un petit rectangle à cheval sur la frontière estimée des deux régions. Le meilleur pixel germe de ce petit rectangle est calculé [7] et un nouveau processus est créé par le système à cet endroit en figure 4.6. En fonction du niveau d'expertise de l'utilisateur, il est possible alors d'accéder au panneau de configuration du processus afin d'ajuster ses paramètres. Par défaut, des paramètres standards seront utilisés (dans ce cas, la maximalité du gradient est le critère prépondérant de la fonction d'évaluation, voir le paragraphe 1.5.3).
- Le processus produit un contour correct, qui permet de mettre en évidence cette faible frontière entre les deux cellules homogènes très proches l'une de l'autre, figure 4.7.

[7] utilisant la méthode de focalisation décrite au paragraphe 2.4.1, utilisée par ailleurs dans le fonctionnement normal du système.



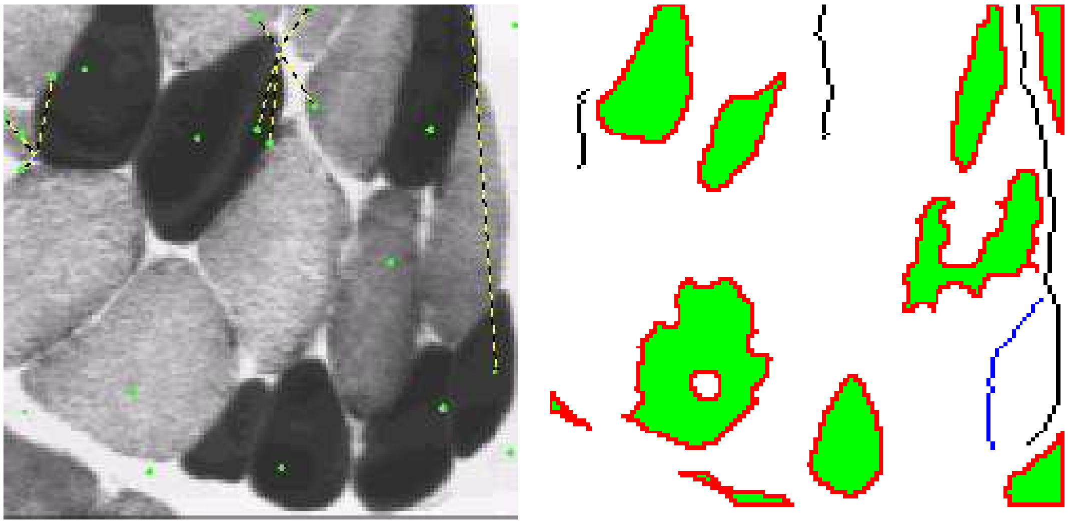
(a) Processus (b) Résultats

FIG. 4.5: deuxième étape : la région erronée est manuellement effacée



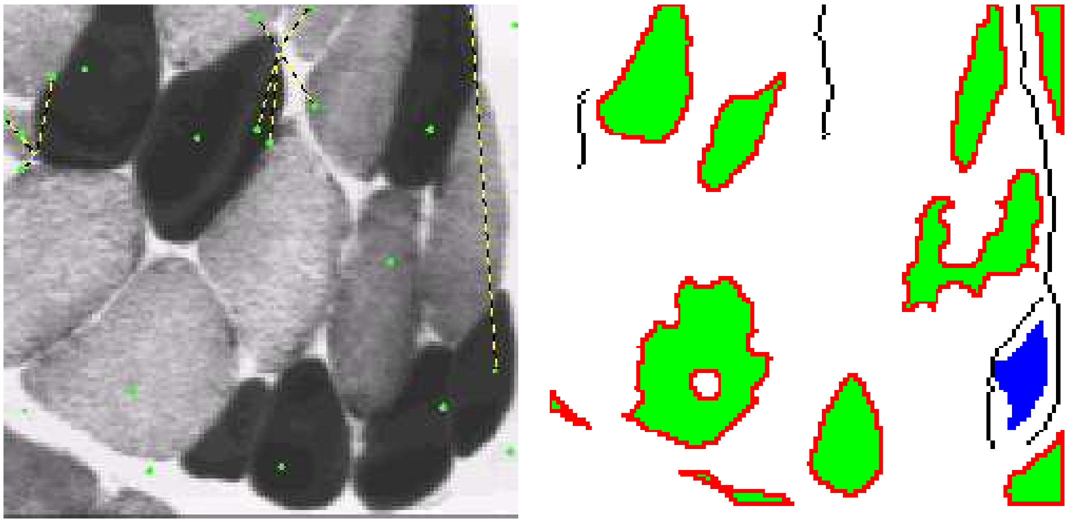
(a) Processus (b) Résultats

FIG. 4.6: troisième étape : un processus contour est ajouté manuellement



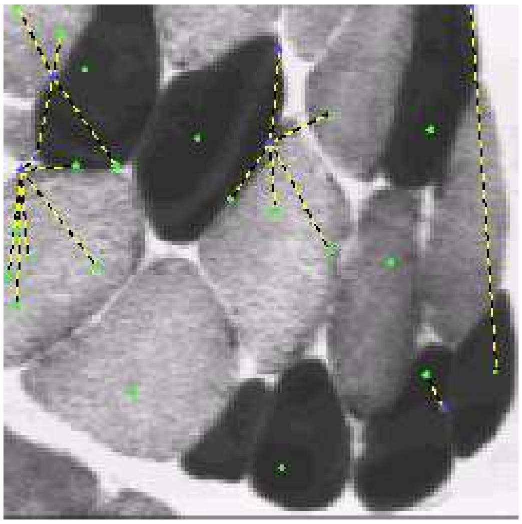
(a) Processus (b) Résultats

FIG. 4.7: quatrième étape : le contour est correctement segmenté

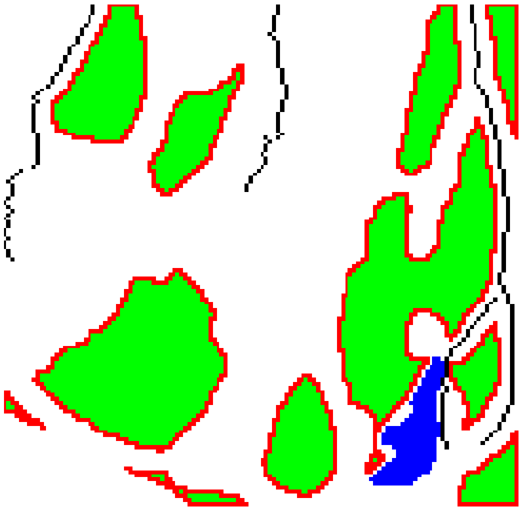


(a) Processus (b) Résultats

FIG. 4.8: cinquième étape : la région de droite est correctement segmentée



(a) Processus



(b) Résultats

FIG. 4.9: sixième étape : la région de gauche est correctement segmentée

- L'utilisateur repasse du mode d'exécution en pas à pas, au mode d'exécution normal, afin que le système reprenne son activité en figure 4.8. L'ancien processus région (situé à droite du contour), qui avait été segmenté avant que l'utilisateur n'interrompe le système, réalise cette fois une segmentation correcte qui ne s'étend pas sur la cellule voisine.
- La région située à gauche est finalement elle-aussi segmentée en figure 4.9, mais en utilisant une autre voie d'accès. La fenêtre montrant l'*arbre des processus* montre que cette région est un processus fils issu directement du processus contour créé par l'utilisateur à la troisième étape. Un comportement de ce type montre comment un nombre minimum d'actions de la part de l'utilisateur peut suffire pour corriger un problème dans une situation complexe. Cela souligne l'importante autonomie ainsi que la robustesse du système, dont le fonctionnement n'a pas été géné par la légère perturbation introduite par l'utilisateur dans l'environnement de travail (la suppression d'une région). La robustesse est liée en partie à la gestion redondante de l'information. Les processus créent plusieurs germes de processus fils lorsqu'une seule information est recherchée. Ainsi l'échec d'un des fils ne déstabilise pas le système, car des possibilités de récupération, et une prise en compte de l'incertitude ont été intégrés. Un processus père n'est pas assuré que ses fils vont réussir à obtenir l'information souhaitée, même dans le cadre du fonctionnement autonome du système.

Cette trace d'exécution montre simplement une manière parmi d'autres afin de corriger une erreur de segmentation. L'utilisateur, suivant son degré d'expertise du domaine, pourra produire un tout autre plan d'action dans le but d'arriver à un résultat similaire :

- L'utilisateur *expert du domaine* efface la région litigieuse. Il sait que les deux cellules correspondent à des régions homogènes, et que les valeurs de gradient entre ces deux objets sont particulièrement faibles, bien que la transition soit parfaitement visible, ce qui explique pourquoi la région a traversé cette faible frontière. Il crée deux nouveaux processus de type région, situés dans chacune des deux cellules recherchées, et ajuste la valeur de leurs paramètres de segmentation afin de disposer d'un critère d'homogénéité plus restrictif au sein de la fonction d'évaluation globale de ce processus région.

– L'utilisateur *intuitif* efface la région erronée, afin de reprendre le même processus, sans changer ses paramètres. Ceci est possible en utilisant le mode "retour arrière" en pas à pas. Il trace juste une frontière virtuelle relative à cette région, qui évite ainsi que la région ne se propage au-delà de cette limite, et ne déborde sur la cellule voisine.

4.5 Conclusion

Les interfaces homme-machine ont souvent été réservées aux problèmes de haut niveau en vision par ordinateur, interprétation de scènes, reconnaissance de formes, suivi d'objets en trois dimensions, etc. Ces activités nécessitent une intervention humaine dans le but de guider l'algorithme, ou d'encadrer son comportement. Dans cette optique, les étapes de bas niveau de la vision par ordinateur sont souvent admises comme des tâches élémentaires autonomes, et qui ne requièrent pas d'intervention humaine à ce titre. L'étiquette de "boite noire" est trop souvent apposées aux méthodes de bas niveau, et constitue une vision beaucoup trop réductrice des problèmes de segmentation qui sont sous-jacents. Une interaction humaine peut profitablement apporter une connaissance de haut niveau à des systèmes de vision de bas niveau. Par voie de conséquence, une segmentation de bas niveau sous contrôle permettrait de simplifier de manière substantielle les traitements de plus haut niveau qui sont souvent obligés de prendre en compte la qualité approximative de leurs données d'entrée. L'interface proposée se greffe de manière transparente sur un système de bas niveau coopératif. Cette interface est certes très spécifique au système piloté, mais elle ne met pas moins en lumière certains principes génériques, tels l'interaction en temps-réel, la notion d'historique des actions, les principes d'adaptation comme instanciation d'une méthode générique, les possibilités de correction d'erreurs à la volée...

Chapitre 5

Intérêt et limites de l'approche

5.1 Introduction

L'évaluation des méthodes de segmentation est un sujet délicat, aux limites floues, aux enjeux multiples, aux protocoles expérimentaux non consensuels. Autant de méthodes de segmentation disposent d'autant de méthodes d'évaluation *ad-hoc*, développées chacune afin de mettre en valeur les points forts des méthodes proposées, et de masquer autant que possible les points faibles, qui ne manquent pas d'exister.

Ce chapitre n'échappera pas à la règle. Mais avant d'ébaucher une analyse numérique, il est indispensable d'apporter un éclairage sur quelques unes des multiples facettes qui constituent l'évaluation d'une méthode de segmentation d'image, au sens le plus général du terme. Cette description préliminaire permettra de motiver les choix d'évaluation retenus par la suite, d'en souligner les caractéristiques et les faiblesses. Ensuite seulement, l'algorithme pourra être soumis à ces méthodes d'évaluation retenues.

5.2 Définition d'un protocole d'évaluation

5.2.1 Les approches dans la littérature

On distingue deux types d'approches pour l'évaluation des algorithmes de traitement d'image, les approches *quantitatives*, et les approches *qualitatives* ou *fonctionnelles*.

- Les méthodes d'évaluation quantitatives ont pour but de faire des mesures sur l'image résultat, permettant d'évaluer la pertinence du résultat. Cette mesure de qualité se rapporte donc à une modélisation de l'objectif qui était à atteindre par l'algorithme, ce qui permet de savoir *a posteriori* dans quelle mesure le programme de segmentation est proche ou non du résultat "optimal" qui était attendu. A ce niveau, une nouvelle dichotomie se présente, suivant que l'algorithme dans l'étape d'évaluation a fonctionné sur des images naturelles ou artificielles.
  - L'utilisation d'images artificielles, ou de synthèse, dans le cadre d'une évaluation d'algorithme permet de disposer d'un résultat optimal sans équivoque, car le concepteur [1] maîtrise complètement la chaîne de construction des images test. Cette chaîne passe par la construction d'une image *pure*, qui sera volontairement dégradée pour se rapprocher des conditions d'acquisition usuelles. Les images artificielles sont

[1] Le concepteur de l'algorithme, et le concepteur du protocole d'évaluation, qui bien souvent sont une même personne. Ne pourrait-on d'ailleurs pas envisager de déléguer l'étape d'évaluation à un expert extérieur un cycle de développement, qui serait apte à produire un protocole de validation non complaisant ?

obtenues à partir d'une modélisation qui se veut aussi fidèle que possible de la réalité que ces images cherchent à reproduire. Les difficultés que peuvent présenter de telles images, même si elles sont réelles, et peuvent mettre en échec un algorithme, ne constituent qu'un sous ensemble des difficultés réelles présentes dans une image naturelle. A ce titre, il semble logique qu'un algorithme destiné à fonctionner sur des images naturelles ne puisse utiliser une évaluation à base d'images artificielles que sous la forme de vérifications préliminaires.

L'étape consistant à dégrader [2] l'image optimale afin de mesurer *jusqu'à quel niveau de dégradation* un algorithme fonctionne est, selon nous, sujette à caution. En effet, à partir d'un bruitage d'une certaine ampleur, l'information que l'algorithme est censé détecter a parfois disparu complètement, ou bien s'est déplacée sensiblement. Dans le cas d'un détecteur de contours, de nouvelles transitions objectivement visibles ont pu apparaître dans l'image, toutes autant représentatives que la transition optimale initiale. Il n'est donc pas très rationnel de demander à un algorithme de retrouver dans l'image une information qui n'existe plus, ou de le sanctionner pour avoir mis en évidence une information que le processus de dégradation aura lui-même générée. Il est convenu que les outils d'acquisition (caméra CCD, scanner, etc) génèrent un bruit standard lié à l'électronique et à l'instrumentation dont l'écart-type σ_n n'excède pas 1.5 niveaux de gris pour les acquisitions standards [MCOT89] [3], alors que dire des principes d'évaluation bruitant des images, en utilisant des écart types qui atteignent parfois une valeur d'écart-type de $\sigma = 50$ [4].

L'utilisation d'un tel bruitage excessif sert généralement à simuler un effet de texture. Ces deux choses sont tout à fait distinctes, car la texture correspond à une réalité dans la scène, alors que le bruit de l'image est un artefact lié à l'électronique de l'acquisition. Ainsi, une texture n'occupe pas toute une zone de l'image, alors que la plupart des bruitages simulant la texture sont appliqués, eux, de manière uniforme. Les images de test ainsi générées occulteront donc les configurations avec une frontière entre une zone texturée et une zone homogène.

Les images artificielles doivent donc être utilisées dans un cadre limité, afin d'effectuer une première vérification "grossière" de l'algorithme. Leur conception doit demeurer réaliste par rapport aux circonstances qu'elles simulent. Les images naturelles sont cependant les seules aptes à placer l'algorithme dans des conditions de segmentation réelles, et à éprouver une méthode de façon réaliste.

Ainsi à titre d'exemple d'évaluation de méthodes classiques sur des images de synthèse, Peli et Malah [TP82] étudient divers détecteurs de contours (algorithme de *Robert*, *Rosenfeld*, *Hueckel*, algorithme du *Min-Max*, etc) sur de telles images. Ce sont deux objets, un carré et un disque, placés sur un fond homogène. Les transitions de type *marche* et de type *ligne* sont étudiées, selon un bruitage de type gaussien ou binaire [5]. Des mesures de robustesse aux différents types de bruits sont faites pour chaque

[2] Les *dégradations* classiques d'une image artificielle consistent essentiellement à lui appliquer un filtrage, par exemple pour la rendre plus *floue*, ou bien pour lui ajouter du *bruit*. Dans ce dernier cas, les types de bruits utilisés sont variés et correspondent à ceux que l'on trouve dans les images naturelles, dus aux imperfections électroniques des capteurs. Il peut s'agir d'un blanc gaussien, impulsionnel, etc.

[3] La qualité des composants depuis 1989 a encore dû diminuer cet écart-type, mais comme nombre d'images de référence manipulées en traitement d'image sont anciennes, cette valeur de σ_n reste "actuelle".

[4] Cette valeur est d'autant plus excessive que la plupart des procédés d'acquisition travaillent avec des intensités codées sur 8 bits, ce qui représente "seulement" 256 valeurs. Utiliser un bruitage ayant pour écart-type $\sigma = 50$ consiste à ne retenir approximativement que les 2 premiers bits de poids le plus fort comme étant significatifs !

[5] Le bruitage binaire est proche du bruitage impulsionnel, il consiste à passer un pourcentage donné de pixels

algorithme.

 - L'évaluation quantitative sur des images naturelles est plus délicate à effectuer, car elle doit résulter d'un consensus de la part des experts du domaine [6], sur le résultat "optimal" qui est espéré à partir d'une image naturelle. Le problème est simplifié, et le cas générique est souvent limité à un de ses aspects clairement exprimable. Les images de test sont restreintes à des objets parfaitement connus, à un environnement clairement défini, qui ne souffre pas d'effets indésirables.

   L'exemple classique de ce genre de pratique, a été l'utilisation du *monde des cubes* dans divers travaux mariant robotique, intelligence artificielle, et vision par ordinateur [Rob63]. Les conditions d'éclairage de la scène, les couleurs très contrastées des objets par rapport au fond rendent la plupart des algorithmes de traitement d'image efficaces sur ce type d'images [Bro91]. La réalité est modelée pour s'adapter aux capacités de l'algorithmique, alors que l'inverse serait plus logique.

- Les méthodes d'évaluation qualitatives ont une approche différente, dans le sens où elles disposent d'un point de vue *comportemental* ou encore *fonctionnel*. A défaut de pouvoir donner une note globale à un algorithme, qui serait une simplification de la réalité, dans la mesure où toute méthode présente des points faibles et des points forts, la validation s'attache à montrer le comportement des algorithmes sur des caractéristiques bien précises. Fleck [Fle92], par exemple, a étudié le comportement de divers algorithmes, à partir d'une image de synthèse comportant certaines difficultés classiques : des coins, des contours de type ligne, des dégradés, du bruit dans l'image, voir la figure 5.1.

  Les problèmes qualitatifs mis en évidence sur une batterie de détecteurs classiques (*Canny*, *Boie-Cox*, *Marr-Hilldreth*) sont, entre autres :

  - la délocalisation et la duplication des contours sur les transitions de type *toit* ou *ligne* ;
  - les difficultés pour segmenter les coins et les structures à angle aigu ;
  - les problèmes à proximité des jonctions, liés à la perte du critère de maximalité locale ;
  - les faux pixels contour souvent trouvés dans les dégradés.

  Fleck arrive aussi à la conclusion que les différents masques de dérivation utilisés produisent des résultats très semblables, et que le point crucial réside davantage dans les post-traitements : interpolation *sub-pixel*, chaînage, hystérésis, etc.

  Cette approche est beaucoup plus intéressante à nos yeux que la précédente, car elle peut permettre au concepteur de l'algorithme de mettre en lumière les faiblesses de son approche, sur lesquelles doivent porter ses efforts d'amélioration. Au contraire, une évaluation quantitative globale d'un résultat ne permet pas d'identifier des défauts particuliers. En considérant l'exemple d'un détecteur de contours, il est plus intéressant de découvrir qu'un algorithme trouve trop de faux contours dans des zones en dégradé, plutôt que de savoir que 75 % des pixels contours obtenus sont "bien placés" par rapport à une carte de référence, car cette valeur ne porte pas de renseignements sur le contexte local pour lequel ces pixels contours sont bien placés.

  Une approche qualitative de l'évaluation reste cependant difficile à automatiser, car l'appréciation — voire même l'intuition — humaine reste le meilleur outil pour mener

de la couleur noire à la couleur blanche.

[6] Ce point étant souvent difficilement réalisable en une durée bornée, l'utilisation d'images synthétiques prend ici tout son sens.

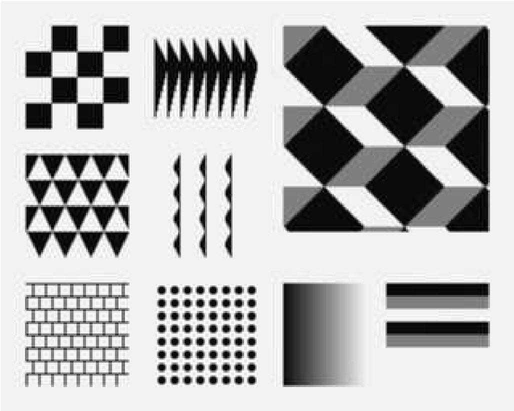


FIG. 5.1: L'image de test utilisée dans l'article de M.Fleck.

à bien cette évaluation. C'est pourquoi les expérimentations qui suivent utiliseront principalement une évaluation quantitative, en tentant cependant le plus possible d'apporter un éclairage concret sur ces chiffres bruts.

5.2.2 Des problèmes de l'évaluation

5.2.2.1 La diversité

Peu de consensus existe sur des protocoles d'évaluation en vision par ordinateur. Considérons une autre discipline informatique, celle du développement des compilateurs, pour un langage X donné. Les données en entrée sont strictement définies par la grammaire qui dicte les règles de construction du langage à compiler. Les données en sortie sont tout autant définies par les codes opérations supportés par le processeur pour lequel le compilateur va générer du code. On s'accorde enfin dans la pratique sur les critères d'évaluation classiques suivants :

- Générer un code exact [7].
- Générer un code rapide.
- Générer un code compact [8].

Un cadre de développement très contraint, notamment avec des données en entrée et en sortie très figées, assure des protocoles d'évaluation universels, entraîne un esprit d'émulation stimulant entre les équipes de recherche, et fait progresser le domaine.

Le domaine de la vision par ordinateur présente une géométrie beaucoup plus variable. Les algorithmes revendiquant l'étiquette de méthode de traitement d'image ne travaillent pas sur les mêmes types d'image. Les moyens d'acquisition, les domaines d'application divers créent autant de "niches" de développement, tentées de développer des méthodes – et donc des protocoles d'évaluation – dédiées à leur domaine. Ces algorithmes ne travaillent pas au même niveau. Ils s'inscrivent pour la plupart dans une chaîne de traitement, et une dichotomie est apparue entre les approches de bas niveau [9] et les approches de haut niveau [10].

[7] La condition la plus difficile à remplir.

[8] Ces deux derniers critères sont antagonistes.

[9] Les approches dites de *bas niveau* sont celles qui généralement manipulent directement les données de l'image sous forme de niveaux de gris, ou de niveaux des couleurs de base. Elles élaborent une première structuration de cette énorme quantité d'information, afin de rendre plus aisés les traitements ultérieurs. Cette structuration consiste à trouver les *contours* (les zones de transition) et les *régions* (les zones homogènes) de l'image.

[10] Les approches dites de *haut niveau* sont celles qui produisent un résultat directement exploitable par l'utilisateur. Les méthodes de reconnaissance d'objet, de dénombrement, de suivi dans une série d'images appartiennent

La première difficulté consiste donc à définir clairement le cadre de l'évaluation. S'agit-il d'évaluer séparément chaque niveau, ou bien une évaluation de la chaîne des traitements dans son ensemble est-elle suffisante ? Le paragraphe suivant s'intéresse plus particulièrement à l'évaluation de bas niveau.

5.2.2.2 La diversité du bas niveau

La structuration du domaine de la vision sous la forme de deux niveaux d'abstraction hiérarchiquement liés n'a pas résolu toutes les ambiguïtés sur le type des données traitées, sur les résultats produits, et par effet de bord sur les évaluations à utiliser. Nous proposons une petite liste non exhaustive montrant la diversité extrême des approches :

- Faut-il introduire de la connaissance de haut niveau dans un algorithme de bas niveau ? Comment comparer un algorithme qui travaille sans connaissance du domaine, et qui peut alors prétendre à la généricité sur différents types d'image, à un autre algorithme qui fait un usage intensif d'une connaissance du domaine, et qui le rend totalement dépendant du type d'image à traiter ? La connaissance de haut niveau consiste à connaître par exemple le type et les caractéristiques des images qui seront utilisées, et à écrire des algorithmes en conséquence.
- Faut-il travailler au niveau du pixel ou au niveau du contour ? Faut-il alors valider un chaînage effectué *a posteriori* sur l'image résultat ? L'évaluation de la qualité d'une segmentation au niveau du pixel semble naturelle, elle permet de mesurer une proportion de bonne/mauvaise détection, ainsi que de quantifier la localisation correcte des pixels contours, si une carte de référence est à disposition. Elle ne permet cependant pas de qualifier la "qualité" de ces contours, plutôt des petits contours tordus en zone texturée, ou bien de longs contours sur les fortes transitions, proportion de contours de moins de dix pixels, mesure de la force de ces contours, etc.
- Comment valider un algorithme utilisant une interaction humaine ? Est-il possible de valider un système utilisant une Interface Homme-Machine ? Lorsque l'intervention humaine est indispensable au fonctionnement d'un système de segmentation, il devient nécessaire de définir son rôle dans le cadre d'un protocole expérimental. Des interventions ayant pour but de guider ou de corriger les résultats de l'algorithme vont augmenter ses scores de performance d'une manière difficilement mesurable, car dépendante de l'utilisateur, et fluctuante d'une expérimentation à une autre.
- Comment quantifier les contraintes temporelles ? Comment comparer un algorithme devant produire un résultat en temps-réel à un algorithme classique ? Un résultat immédiatement utilisable est obtenu en appliquant une algorithmique légère, à base de traitements rapides et superficiels, difficilement comparable avec une méthode dont le but est de produire le meilleur résultat possible, sans contrainte de temps, et qui à ce titre mettra en jeu une artillerie algorithmique de plus grande ampleur.
- Comment comparer des algorithmes utilisant des données en entrée qui ne sont pas unifiées ? Les sources d'acquisition sont multiples (couleur, IRM, Infra rouge, caméra CCD, digitalisation, etc), de qualité variable (taille, focus, qualité de l'électronique, etc), présentant des types de dégradation multiples (prédictibles ou pas, bruit, etc). Selon le domaine d'acquisition, l'objectif de segmentation à atteindre est plus ou moins évident, et les traitements à mettre en œuvre s'en ressentent naturellement.

à cette classe. Elles utilisent la structuration produite par le bas niveau, et lui appliquent des raisonnements souvent davantage cognitifs (système expert, tableau noir, etc), dans lesquels l'utilisateur dispose d'une interaction plus importante que dans le bas niveau.

- Comment comparer deux algorithmes qui produisent une quantité d'information différente ? La détection de contours, dans sa version la plus simple, consiste à fournir à l'utilisateur une carte binaire, dans laquelle les pixels contours sont simplement étiquetés. Un algorithme qui génère un résultat plus *riche* [11] est raisonnablement plus pertinent pour la résolution des problèmes du haut niveau.

Il est clair que la grande diversité des approches, des méthodes, des données et des résultats pose des problèmes pour uniformiser un protocole d'évaluation, comme les quelques exemples précédents le montrent. L'utilisation d'images naturelles pour valider une approche parait indispensable. Une évaluation quantitative sur ces images naturelles est souhaitable, mais difficile à obtenir. La définition de cartes de référence permet de s'appuyer sur un résultat considéré comme “optimal”, du moins par l'expert qui a produit la carte de référence et par les utilisateurs, à défaut de pouvoir rallier l'assentiment d'un plus large public.

La construction puis l'utilisation d'une carte des contours de référence pour une image naturelle est adaptée pour résoudre certains des problèmes énoncés précédemment, et retient à ce titre notre intérêt dans le paragraphe qui suit :

- elle permet une évaluation quantitative ;
- elle permet de comparer des méthodes qui produisent une quantité de résultats différente, sur la base d'un sous-ensemble de primitives communes aux deux méthodes. Dans le cas présent, le système coopératif, bien que fournissant une carte contenant à la fois des contours et des régions, pourra être évalué et comparé à des méthodes de détection de contours ne générant pas un résultat aussi riche ;
- elle contient des difficultés qui reflètent de la réalité des problèmes de segmentation rencontrés par les algorithmes classiques, que les images artificielles ne parviennent pas à reproduire [12] ;
- elle est issue d'une expertise qui n'inclut pas de connaissances sur le domaine d'application d'un algorithme, et donc peut à ce titre servir à comparer des méthodes de segmentation différentes, dès lors qu'elles fonctionnent également au bas niveau, sans connaissances sur le domaine.

5.2.3 Une carte de référence

Un algorithme dont la finalité est d'exploiter des images naturelles doit pouvoir être évalué numériquement. L'évaluation numérique d'une détection de contour peut porter sur des critères objectifs du résultats (nombre, longueur et profil des contours trouvés), mais également sur des critères relatifs à un résultat qualifié de *référence*, comme cela a été étudié dans la thèse de Spinu [Spi97]. Ce deuxième cas utilise des images de synthèse, dont le résultat de référence est

[11] Un résultat *riche* fournit, en plus de la carte classique des contours, une liste des contours segmentés, avec leur liste de pixels respectifs, des informations photométriques et topologiques pour chaque contour (gradient, orientation, valeurs extrêmes, relations de vis à vis avec d'autres contours, etc), et éventuellement d'autres objets d'intérêt de l'image (jonctions, régions homogènes, segments, dégradés). L'objectif d'une segmentation *riche* est d'offrir le maximum d'information utilisable à un autre niveau de traitement, sans que ce niveau ait besoin de revenir travailler sur les niveaux de gris de sa propre initiative. Une segmentation *riche* contient alors, dans le cas idéal, l'ensemble des informations extraites de l'image originale, qui peuvent servir aux algorithmes de haut niveau. Une telle segmentation réalise de ce point de vue une compression efficace de l'image, ou du moins de toutes les informations pertinentes qu'elle peut contenir.

[12] La confrontation de l'expérimentateur qui construit la carte de référence avec ces problèmes de segmentation “bien réels” est souvent riche d'enseignements lorsque l'expérimentateur est lui-même le développeur d'un système de segmentation. Il peut ainsi être tenté de reproduire algorithmiquement l'expertise qu'il développe dans le marquage de ses pixels de référence.

simple à établir. Pourtant, cette simplicité est le principal défaut des images de synthèse par rapport aux images naturelles auxquelles est destiné l'algorithme. Il apparaît donc nécessaire de produire un résultat de référence également pour les images naturelles, avec toutes les difficultés et ambiguïtés que cela suppose.

La notion de carte de contours de référence établies sur des images naturelles, sans introduire de connaissances *a priori* du domaine, a été introduite dans les travaux de thèse de M. Salotti [Sal94], [SBG96]. Le principe d'une telle carte est de définir et de localiser les lieux de transition objectivement visibles dans une image naturelle en niveaux de gris. L'idée sous-jacente developpée par Salotti est que le concepteur de cette carte de référence dispose d'une certaine expertise pour le marquage de ces pixels de référence. Cette expertise porte sur la manière dont il explore visuellement l'information des niveaux de gris, et sur la structuration qu'il fait de cette information dans son esprit pour aboutir à l'étiquetage d'un pixel dans l'image. Idéalement, cette expertise devrait pouvoir être extraite et incorporée dans un algorithme de bas niveau qui produirait à terme, à partir des mêmes données d'entrée le même résultat de référence.

De telles cartes sont construites par l'expert humain, selon un protocole expérimental strict. En travaillant sur une fenêtre glissante de taille relativement réduite (par exemple 32×32 pixels), l'expert doit marquer des pixels aux endroits où il considère qu'une transition des niveaux de gris est visible, à cette résolution de travail, et sans avoir de contrôle dans l'image globale. Cela plonge l'expérimentateur dans des conditions de travail proches de celles d'un algorithme de traitement d'image, avec certes une fenêtre de travail plus importante, et le contraint en particulier à ignorer toute connaissance sur le domaine. En effet, un tel facteur de zoom retire toute la sémantique à l'objet sur lequel l'expérimentateur travaille. Une segmentation manuelle sur une fenêtre glissante de ce type met en lumière des problèmes de segmentation, qui apparaissent dans tout algorithme, mais que le concepteur n'a pas l'habitude de devoir lui-même se poser :

- Comment différencier une transition floue d'un dégradé, et à partir de quel moment l'étiquetage d'un contour devient-il légitime sur une transition de ce type ?
- Un contour est souvent représenté par une chaîne de pixels d'épaisseur unitaire, alors qu'une transition entre deux zones homogènes – comme son nom l'indique – concerne une frange de plusieurs pixels de large. Le pixel devant représenter cette transition dans l'image résultat n'a pas de position particulière dans cette frange de pixels, même s'il est peut-être plus logique de le placer au milieu de la transition. Dans le meilleur des cas, il subsiste toujours une ambiguïté de un pixel, liée à la discrétisation.
- Quelle convention d'étiquetage adopter pour des contours de type ligne ? Faut-il utiliser une étiquette particulière, ou bien les marquer sous la forme de deux contours de type marche en vis à vis. La première proposition est plus riche sémantiquement, car l'étiquetage en lui-même devient porteur de cette information de vis à vis, qui caractérise le profil d'un contour de type ligne.
- Quelle convention de visibilité faut-il utiliser ? Il est simple de corriger la balance des niveaux de gris d'une image, avec un outil de visualisation classique, ce qui a pour effet de rendre une image plus ou moins lumineuse, et de rendre plus ou moins visible une transition. Les réglages de l'écran sur lequel l'image est visualisée peuvent avoir le même effet.
- Comment caractériser l'importance d'un contour ? Pour un nombre de pixels égal, il est plus gênant qu'un algorithme n'ait pas détecté un contour fortement marqué, plutôt qu'une transition floue et ambiguë. Le facteur de confiance avec lequel l'expérimentateur marque un pixel de contour dans sa carte de référence devrait se refléter dans son résultat

produit, mais il est malaisé de quantifier numériquement un doute, ou une absence de doute.

Une image test dont on souhaite réaliser une carte des contours de référence est donc présentée à un utilisateur. Cette personne n'a pas à connaître le domaine d'application, et n'a pas à posséder de connaissances particulières en traitement d'image. En explorant l'image avec un facteur de zoom très important, de manière à ne voir à chaque instant qu'une fenêtre de pixels restreinte de l'image totale [13], il lui est demandé de marquer les pixels qui lui semblent séparer des zones d'homogénéité différente, en respectant simplement des contraintes d'épaisseur et de connexité des pixels contours ainsi placés.

Cette carte de référence contient ainsi les transitions *objectivement visibles* [14] sans introduire de connaissance du domaine [15]. L'idéal est de disposer de différentes cartes de référence, établies par des expérimentateurs différents, suivant le même protocole expérimental. La stabilité des évaluations à partir de plusieurs cartes de référence devient un gage de fiabilité de l'expérimentation. Hélas, la conception de telles cartes prend beaucoup de temps avec des images de taille habituelle 256×256, et cela limite beaucoup la bibliothèque de carte de référence à disposition.

Ainsi, grâce à de telles contraintes, une carte de référence devient indépendante de l'algorithme de bas niveau, et peut donc servir à valider différents algorithmes, d'origine diverse.

5.2.4 L'évaluation par les cartes de référence

Les cartes de références ne contiennent pas une information aussi riche que celle pouvant être produite par un algorithme de détection de contours. La seule information qu'il est raisonnable de demander à l'expérimentateur qui construit la référence est un étiquetage au niveau du pixel. Les cartes de référence utilisées contiennent uniquement une carte des pixels contours qu'il est souhaitable de détecter. Un algorithme de détection de contours produira en général des contours sous forme de listes de pixels chaînés, avec des informations photométriques supplémentaires.

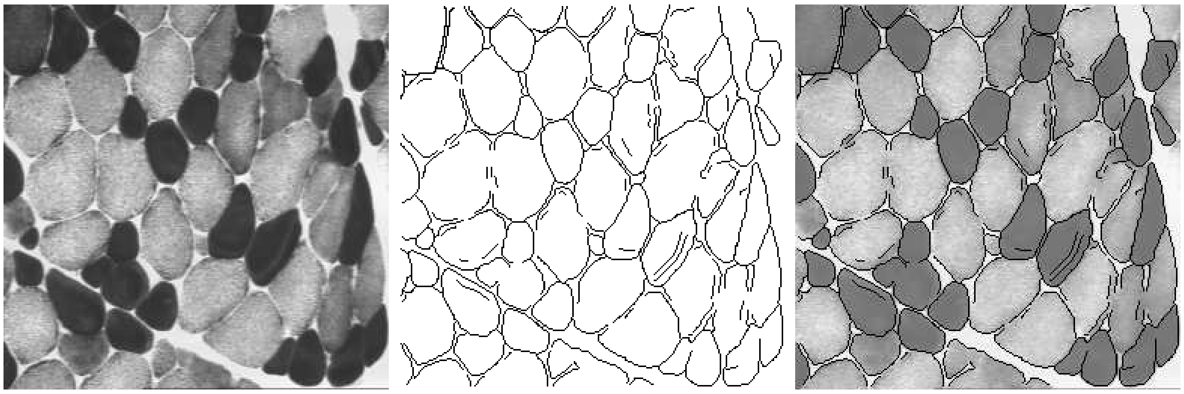
Les cartes de contours de référence à disposition sont présentées en figure 5.2(b), 5.3(b), 5.4(b), 5.5(b) et 5.7(b). Chaque figure présente l'image originale, ainsi qu'une superposition entre l'image originale et la carte de référence. L'image superposée à des niveaux de gris plus clairs (tous supérieurs à 80), afin de mettre davantage en relief les contours de référence en noir.

L'évaluation à partir de ces cartes de référence se fait à la fois au niveau du pixel et au niveau du contour. Pour ce faire, il suffit de restructurer la carte de référence, ainsi que l'image résultat sous la forme d'un ensemble de contours. Il faut faire apparaître dans la carte de référence les pixels ayant le statut particulier de *pixels de jonction* par rapport à la configuration des autres pixels de leur voisinage. Un contour est alors défini dans la référence, comme un ensemble de pixels chaînables sans équivoque, entre deux pixels de jonction. Un pixel de jonction sera un pixel contour ayant plus de deux voisins eux-mêmes pixels contour, non liés par une contrainte de voisinage, voir la figure 5.8. Des informations photométriques et topologiques peuvent être collectées sur chaque contour ainsi reconstruit, à la fois sur la carte de référence, et sur la carte des contours résultat du système de segmentation.

[13] en général une fenêtre 32×32, ce qui sera davantage que la plupart des algorithmes classiques de détection de contour, qui limitent l'exploration à de très petites fenêtres 5×5 ou 7×7.

[14] Les transitions *objectivement visibles* sont celles qui apparaissent visuellement à l'expérimentateur lorsqu'il marque les pixels de type contour dans la fenêtre glissante.

[15] Par exemple, l'expérimentateur ne cherchera pas à produire une carte de référence ne contenant que des contours fermés, sous prétexte que l'algorithme ne fonctionne que sur des images cellulaires, aux objets de forme circulaire.

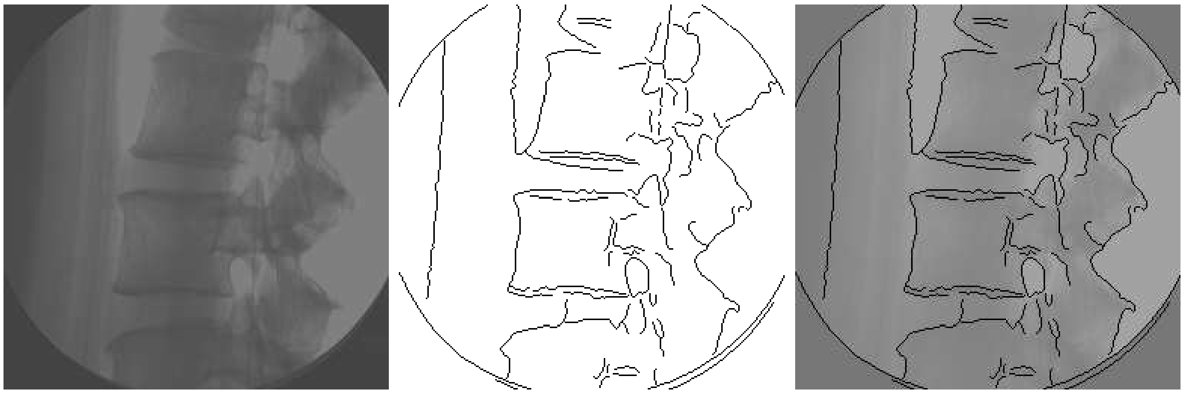


(a) Niveaux de gris. (b) Carte de référence. (c) Superposition de (a) et (b).

FIG. 5.2: La carte de référence de l'image *cells256.gif*.

(a) Niveaux de gris. (b) Carte de référence. (c) Superposition de (a) et (b).

FIG. 5.3: La carte de référence de l'image *femme256.gif*.

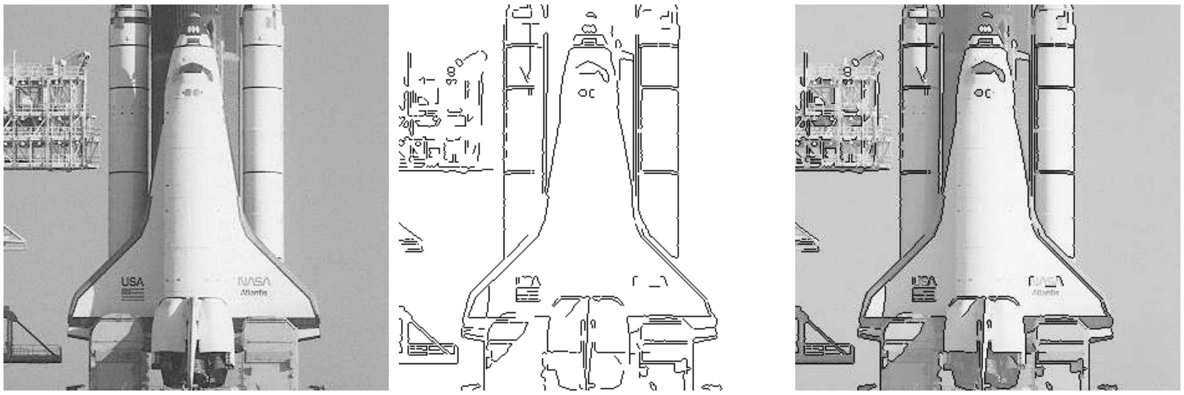


(a) Niveaux de gris.

(b) Carte de référence.

(c) Superposition de (a) et (b).

FIG. 5.4: La carte de référence de l'image *vertebre256.gif*.



(a) Niveaux de gris.

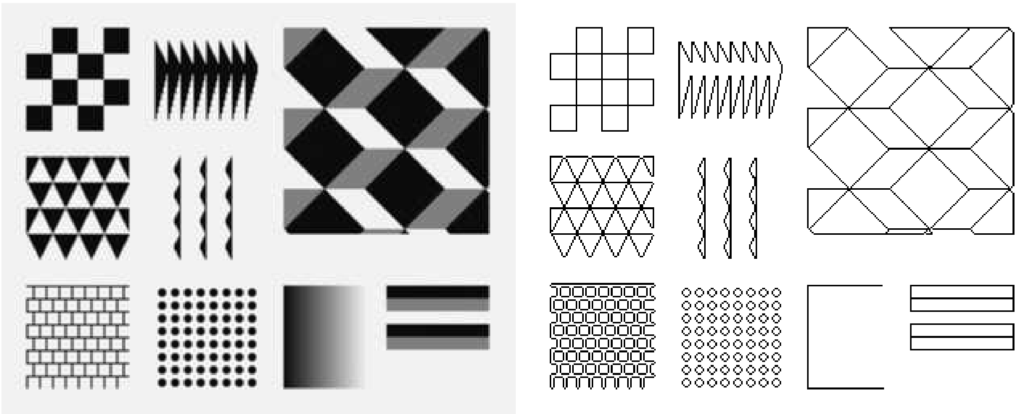
(b) Carte de référence.

(c) superposition de (a) et (b).

FIG. 5.5: La carte de référence de l'image *shuttle310.gif*.

(a) Niveaux de gris. (b) Carte de référence. (c) superposition de (a) et (b).

Fig. 5.6: La carte de référence de l'image *bureau256.gif*.



(a) Niveaux de gris. (b) Carte de référence.

Fig. 5.7: La carte de référence de l'image *fleck.gif*.

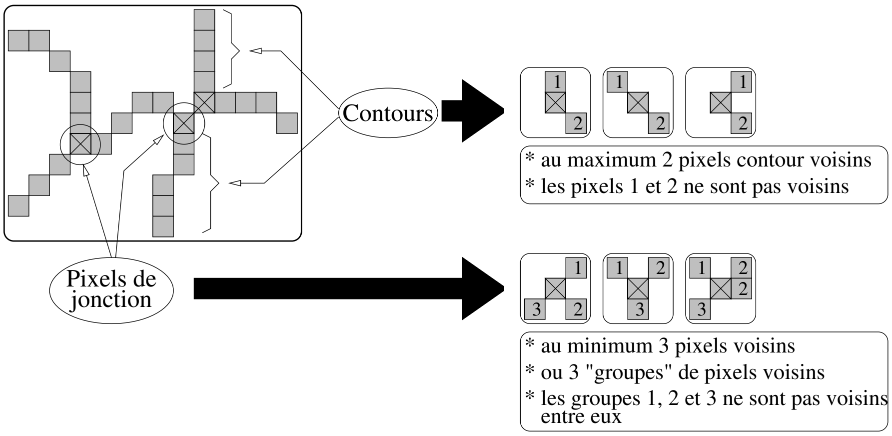


Fig. 5.8: Structuration de la carte de référence en contours.

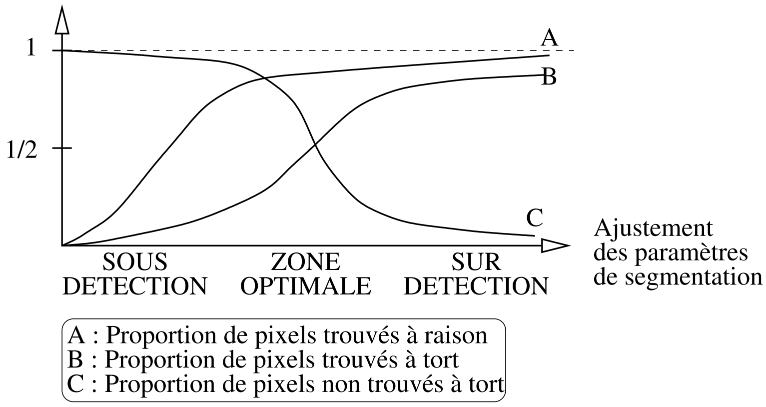


FIG. 5.9: Mesure d'un compromis entre la sous- et la sur-segmentation.

L'évaluation au niveau du pixel consiste à valider une proportion de pixels coïncidents, c'est-à-dire trouvés à raison (A), trouvés à tort (B), et non trouvés à tort (C). L'évaluation au niveau contour permet de comparer le nombre de primitives trouvées, et de quantifier éventuellement des caractéristiques de segmentation :

- Trop ou pas suffisamment de petits contours par rapport à la référence.
- Qualité photométrique particulière dépendant de la longueur des contours.

L'évaluation d'un résultat est un compromis entre la sur- et la sous-détection [16]. En ce sens, la valeur intéressante à mesurer est une combinaison linéaire entre B et C, puisque l'algorithme optimal réalise une minimisation conjointe de B et C. La sur-détection se caractérise par une forte valeur de A ainsi que de B, et la sous-détection, inversement, par une faible valeur de A, et une forte valeur de C, voir le graphique de la figure 5.9.

Il peut sembler logique d'ajouter une quatrième catégorie (D) de pixels *non trouvés à raison*, ce qui permettrait d'obtenir la somme des quantités (A), (B), (C) et (D) qui couvrirait tous les pixels de l'image. Ceci n'est pas fait, car (A), (B) et (C) ne sont pas rapportées au nombre total de pixels de l'image, mais au nombre de pixels *contour* de leurs cartes respectives. L'égalité n'est donc pas réalisée. Ainsi :

- Le nombre de pixels trouvés à tort est rapporté au nombre de pixels contours du résultat de l'algorithme pour fournir la valeur (B) ;
- Le nombre de pixels non trouvés est rapporté au nombre de pixels contours de la référence pour fournir la valeur (C) ;
- Le nombre de pixels communs peut être rapporté indifféremment aux pixels contours du résultat ou de la référence, pour fournir en fait deux valeurs (A1) et (A2).

A ces questions de dénombrement et de mise en proportion se rajoute la question de la localisation des pixels contours, puisque les mesures précédentes mesurent des correspondances au pixel près, et qu'il peut être souhaitable de relâcher un peu cette contrainte. On pourra ainsi qualifier de *communs*, un pixel contour de référence et un pixel contour du résultat, qui seraient situés à des coordonnées proches l'une de l'autre dans leur image respective.

[16]La *sous-détection* est un résultat dans lequel seul les contours fortement visibles ont été trouvés, à cause, par exemple, d'un mauvais ajustement des paramètres de segmentation. Les contours faiblement marqués ont échappé au détecteur. La *sur-détection* est le résultat inverse. Trop de contours ont été obtenus, et beaucoup ne sont pas porteur d'une information significative. Le meilleur résultat est bien-sûr celui qui réalise un habile compromis entre sur-détection et sous-détection. Ces problèmes apparaissent en général pour toutes les méthodes utilisant un seuillage de la valeur du gradient dans l'image. Les fortes valeurs de gradient correspondent aux fortes transitions, et les faibles valeurs, lorsqu'elle sont significatives marquent les transitions plus faibles.

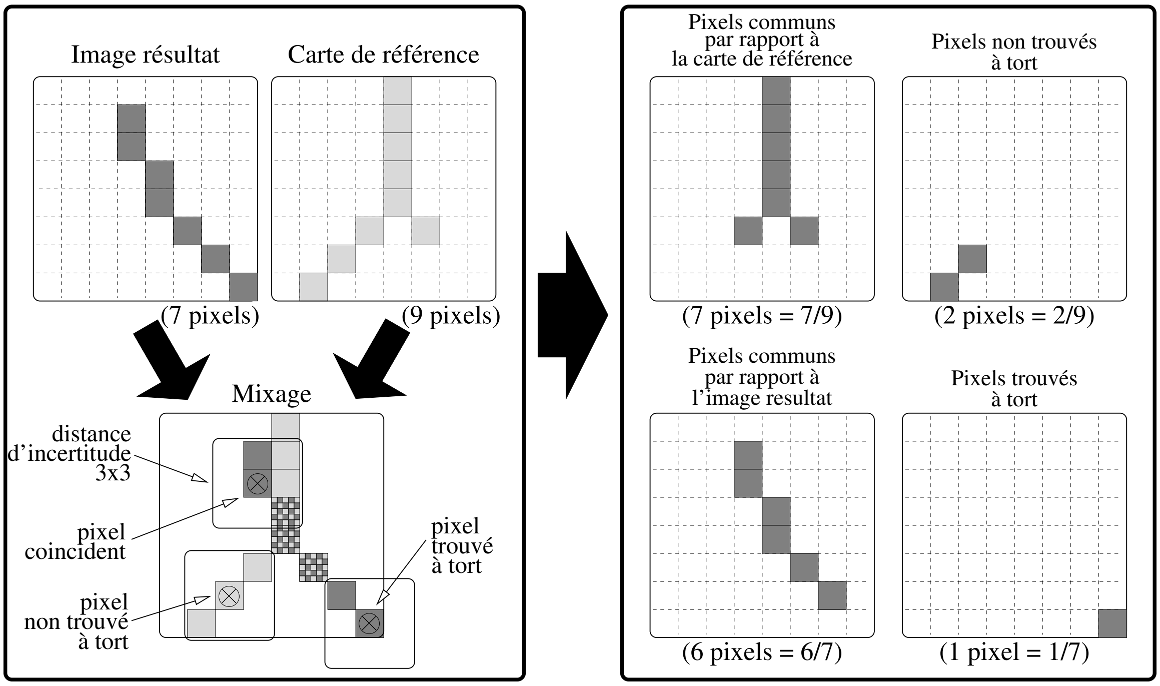


FIG. 5.10: Comparaison de niveau pixel entre un résultat de segmentation et une carte de référence, en utilisant une distance d'incertitude de un pixel.

Donc, afin de ne pas être trop restrictif sur la localisation des pixels, une distance d'incertitude est utilisée pour mesurer la coïncidence d'un pixel contour de la segmentation, avec un pixel contour de référence. Une distance d'incertitude de un pixel, consiste à utiliser une fenêtre 3×3 [17] centrée sur le pixel de la segmentation pour y rechercher un pixel dans la carte optimale. La figure 5.10 illustre sur un exemple simple, la manière dont les mesures d'évaluation A, B et C sont calculées, en utilisant une fenêtre d'incertitude 3×3. Il convient de remarquer que la quantité A peut être calculée de deux façons différentes, selon que l'on comptabilise les pixels de l'image résultat ou les pixels de la carte de référence, ce qui donne les valeurs $7/9$ et $6/7$ dans cet exemple. B vaut ainsi $1/7$ et C vaut $2/9$. L'utilisation d'une distance d'incertitude différente change évidemment ces résultats. En utilisant une distance d'incertitude nulle, on obtient $B = 4/7$ et $C = 6/9$ en comptabilisant les pixels qui n'ont pas de correspondant exact dans l'autre image.

Ce type d'évaluation se rapproche de la mesure de performance publiée par Pratt [Pra91], qui constitue une méthode très connue et souvent référencée : la "figure de mérite" [18]. Les points communs sont d'effectuer une comparaison pixel à pixel entre une carte de contours résultat et une carte de contours idéale. La caractéristique supplémentaire de la FOM de Pratt est d'introduire l'éloignement du pixel trouvé par rapport au pixel optimal au sein même de la formule calculant la performance, sous la forme :

$$FOM = \frac{1}{\max(I_I, I_A)} \sum_{i=1}^{i=I_A} \frac{1}{1 + \alpha d^2(i)}$$

avec I_A et I_I étant respectivement le nombre de pixels de l'image résultat actuelle et de l'image idéale, et $d(i)$ la distance du pixel détecté au plus proche pixel optimal.

[17] Une fenêtre de taille 3×3 correspond à une boule de rayon 1, au sens de la norme I. Une fenêtre 5×5 à une boule de rayon 2, une fenêtre de rayon $(2n + 1) \times (2n + 1)$ à une boule de rayon n.

[18] *figure of merit* ou FOM dans le texte original.

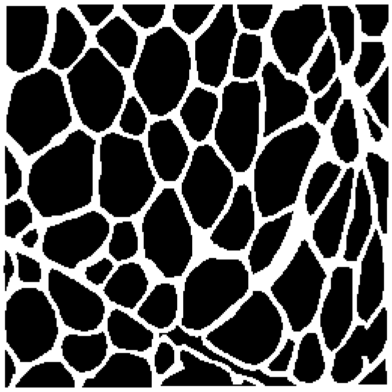


Fig. 5.11: La carte de référence des régions pour l'image *cells256.gif*.

Notre évaluation exclut tout post-traitement de la carte contour résultat produite par un algorithme à tester, autre qu'un éventuel *amincissement* des contours, ayant pour but de leur restaurer leur 8-connexité. Ce traitement est justifié, dans le sens où, pour une transition équivalente dans l'image, un contour épais contiendra davantage de pixels qu'un contour fin en 8-connexité, et cela a pour effet de fausser les mesures qui rapportent leur valeur au nombre de pixels contours total dans les images.

Une carte de référence existe également pour les régions de l'image cellulaire, permettant une évaluation en terme de nombre de primitives trouvées, et de facteur de recouvrement, voir la figure 5.11. Cette image a été obtenue manuellement par la délimitation, puis remplissage des zones homogènes de l'image.

Une telle carte des régions de référence permet de calculer des mesures de dissimilarités avec les régions résultats fournies par un système de segmentation. Nous utiliserons la mesure proposée par Vinet [Vin91] dans ses travaux de thèse. Cette mesure s'applique entre deux images d'étiquettes que l'on cherche à comparer. Dans notre cas, il suffit de réaliser un classique étiquetage des composantes connexes à partir des deux cartes à comparer : la carte de référence d'une part, comparée à chacune des images résultats d'autre part. Le principe de cette mesure consiste à déterminer les couples de région assurant un recouvrement maximum entre les deux segmentations, et à caractériser la dissimilarité par la proportion de pixels ne participant pas à ce recouvrement. Une matrice T est construite dont chaque élément $T(i,j)$ comptabilise le nombre de pixels en commun entre la région R_i^1 de la première image, et la région R_j^2 de la deuxième image. On recherche dans cette matrice le couple de régions de recouvrement maximal, dont on conserve la valeur du recouvrement $c_1 = T(i_1, j_1)$. On itère parmi les régions restantes, et on obtient la valeur $c_2 = T(i_2, j_2)$, etc. Le recouvrement global est obtenu en calculant $\sum_i c_i$. La mesure de dissimilarité est calculée par l'expression suivante, dans laquelle N est le nombre total de pixels dans l'image :

$$D = \frac{N - \sum_i c_i}{N}$$

Cette mesure est intéressante car elle ne caractérise plus seulement la pertinence d'une correspondance en nombre de région, mais en nombre de pixels, et ceci sous une forme très compacte. Elle pénalise cependant les méthodes, comme la nôtre, qui ne proposent pas une segmentation en régions complètes de l'image, puisque la valeur de D introduit le nombre de pixels de l'image. Ainsi, la dissimilarité entre deux cartes de régions identiques, mais ne partitionnant pas complètement l'image au sens de Zücker [Züc76], ne sera pas nulle. La solution est alors de considérer la zone restant non segmentée comme une région supplémentaire, tant du point de vue de la carte région à valider que du point de vue de la carte des régions de références.

5.3 Evaluation du système

L'objectif de cette partie est d'effectuer des mesures quantitatives sur le fonctionnement du système de segmentation qui a été décrit de manière extensive dans les précédents chapitres. Cette évaluation se focalisera particulièrement sur certains points, qui semblent constituer la force de l'approche par rapport aux méthodes habituelles, telles que la *robustesse*, la *justesse* des résultats par rapport à une carte de référence, la puissance de l'*adaptation*, et l'intérêt de la *coopération*.

Le système, dans sa forme actuelle, dispose de paramétrages nombreux [19]. Les plus importants sont regroupés dans un fichier de configuration, modifiable par l'utilisateur, et lu à chaque exécution. D'autres paramétrages sont directement codés dans le programme lui-même. A titre d'exemple, un fichier de configuration type contient les informations suivantes :

```
# ----------------------------------------------------------------------------
#
# Fichier de configuration des parametres de l'algorithme cooperatif
# (c) 03/98 Fabrice Bellet
#
#    Ne pas modifier les champs a gauche des ':'
#
# ----------------------------------------------------------------------------

Affiche les fenetres : yes
Utilisation de couleurs : yes
# ---------------------------------------
# le champs peut prendre comme valeur :
#    DISPLAY  : pour recuperer la valeur de la variable d'environnement
#    host:0   : pour utiliser le display precise
# ---------------------------------------
Display : DISPLAY
Echantilonnage du gradient : 256
Lissage gaussien prealable : 0.7
Gradient de Deriche : yes
# ---------------------------------------
# Ce champs peut prendre comme valeur :
#    manuel/auto ou automatique
# ---------------------------------------
Initialisation des germes : auto
# ------------------------------------------------
# Champs relatifs a l'initialisation automatique.
# Le germe peut etre de type :
#     - contour
#     - region
# Son positionnement peut etre de type :
#     - photometrique
#     - regulier sur une grille
#     - aleatoire
# La syntaxe anterieure reste valide et correspond a
#    Nombre de germes contours   : contours [photometrique]
#    Nombre de germes regions    : regions [photometrique]
#    Nombre de germes aleatoires : regions [grille]
# ------------------------------------------------
Germes contours [photometrique] : 1
Germes contours [grille] : 0
Germes contours [random] : 0
Germes regions [photometrique] : 0
Germes regions [grille] : 0
Germes regions [random] : 0
```

[19] Il est utopique et vain de tenter de masquer ce fait. Tout système complexe, ayant à traiter des données fluctuantes, se doit de disposer d'un ensemble de points de contrôle, sous la forme de constantes ajustables. Cela constitue à notre avis une richesse indéniable pour l'utilisateur, par rapport à un algorithme de type boîte noire, qui ne lui laisse aucun choix possible. Il faut cependant que ces points de contrôle de la méthode soient répertoriés et ordonnés en fonction de leur influence et de leur importance respective dans le système. Une Interface Homme-Machine telle que celle décrite dans le chapitre 4 permet une telle structuration.

```
# -------------------------------------------
# Champs relatifs a l'initialisation manuelle
# au maximum 10 germes differents
# -------------------------------------------
Germes : (39,90,LD)
# -------------------------------------------------------------
# La limitation de la charge est limitee a <N> valeurs graduelles
# On associe autant de "Frontier Sample" et de "Window Size" que
# l'on dispose de seuils pour la charge
# -------------------------------------------------------------
Limitation de la charge : 5000 9000
Load - Frontier Sample : 5 100
Load - Edge Sample : 50 300
Load - Window Size : 5 7
# -------------------------------------------------------------
# Defifinition des classes d'ecart type de type region/contour
# -------------------------------------------------------------
Nombre d'etapes de relaxation : 4
Classes d'ecarts types (R) : 1.0 2.0 4.0 6.0 10.0
Seuils d'evaluation (R) :
0.2 0.4 0.5 0.7 1.2 1.3
0.3 0.6 0.7 0.9 1.4 1.5
0.5 0.8 0.9 1.1 1.8 2.0
0.7 1.0 2.0 3.0 4.5 5.0
#
# Anciennes valeurs
#
# Classes d'ecarts types (R) : 2.0 5.0 6.0 8.0 10.0
# Seuils d'evaluation (R) :
# 0.4 0.5 0.7 1.0 1.2
# 0.6 0.7 0.9 1.2 1.4
# 0.8 0.9 1.1 1.5 1.8
# 0.9 1.0 1.2 1.6 2.0
Classes d'ecarts types (C) : 2.0 5.0 8.0 10.0 12.0
Seuils d'evaluation (C) :
3 5 7 9 11 13
5 8 12 14 20 25
7 12 17 20 23 27
9 14 20 26 30 35
11 20 23 30 35 40
13 25 27 35 40 45
# -------------------------------------------------------------
# Valeur a appliquer par defaut sur le seuillage du gradient.
# Cette valeur est appliquee pour les processus initiaux, ou
# pour les autres processus contour lorsque l'heritage du
# seuil du gradient n'est pas autorise
# -------------------------------------------------------------
Seuil gradient statique : 35.0
# -------------------------------------------------------------
# Le champ 'Autorise adaptation' a disparu au profit de deux
# parametres distincts plus explicites. La syntaxe precedente
# reste utilisable et correspond a 'Autorise seuil gradient dynamique'
# -------------------------------------------------------------
Autorise heritage du seuil gradient : no
Autorise seuil gradient dynamique : yes
# -------------------------------------------------------------
Autorise Focus delocalise : yes
Autorise validation contour : yes
Autorise validation region : yes
Autorise fusion : yes
# -------------------------------------------------------------
# Parametres relatifs a la methode de croissance de region
# par remplissage
# -------------------------------------------------------------
Autorise le region-filling : yes
Priorite pour le region-filling : 1
Seuil pour le region-filling : 1.0
Ecart-type du region-filling : (0.0,1.5)
# -------------------------------------------------------------
# Parametres relatifs a la methode de croissance de region
# par construction de rectangles adjacents
# Eval peut prendre comme valeur : local ou global
# -------------------------------------------------------------
Autorise les boxes-growing : no
```

```
Eval pour les boxes-growing : local
Priorite pour les boxes-growing : 2
Seuil pour les boxes-growing : 3.0
# -------------------------------------------------------------
Quantum de pixels contour : 500
Quantum de pixels region : 500
# --------------------------------------------
# L'ordre et la signification des ponderations
# Contour :
#     - Norme du gradient
#     - Gradient maximum local
#     - Rectitude du contour
#     - Longueur du contour
#     - Passage au mieux entre deux regions
#     - Suivi de frontiere de la plus proche region
#     - Preserver homogeneite region proche
#     - Extremum des niveaux de gris (represente
#       une transition de type ligne)
#     - Direction du gradient
# Region :
#     - Homogeneite
#     - Gravite
#     - Compacite
#     - Degrade
# --------------------------------------------
Ponderation contour : 0.7 0.3 0.0 0.0 0.0 0.0 0.0 0.0 0.0
Ponderation region : 0.4 0.0 0.6 0.0
Frequence de reevaluation : 100
Autorise synchronisation : yes
Autorise ecart masque : no
Autorise main window : yes
Autorise load window : yes
Autorise activity window : yes
Autorise link window : yes
Autorise focus window : yes
# -------------------------------------------------------------
# nouveau !
# -------------------------------------------------------------
Fichier des images internes : /tmp/pictures.dat
# -------------------------------------------------------------
# Le Backup intervient a une date choisie par l'utilisateur
# Le Recover intervient toujours au debut de l'execution
# (il est en fait lance par le processus Root)
# -------------------------------------------------------------
Autorisation du backup : no
Autorisation du recover : no
Fichier pour ecrire le backup : backup.dat
Date de lancement du backup : 10
# -------------------------------------------------------------
# Des preferences visuelles
# -------------------------------------------------------------
Initial Main Zoom : (2,2)
Initial Load Zoom : (1,1)
Initial Focus Zoom : (1,1)
Initial Activity Zoom : (1,1)
Initial Link Zoom : (1,1)
# -------------------------------------------------------------
# Des parametres relatifs a l'arborescence des processus
# -------------------------------------------------------------
Profondeur maximum de l'arbre : 100
Largeur maximum de l'arbre : 9000
Degre maximum de l'arbre : 9000
# -------------------------------------------------------------
# Parametres relatifs a la multi-resolution
# -------------------------------------------------------------
Resolution initiale region : 1
Resolution finale region : 1
# -------------------------------------------------------------
# Position relative des fenetres de focalisation
# -------------------------------------------------------------
Edge > Edge Focus Distance : 3
Edge > Edge Focus Profondeur : 3
Edge > Edge Focus Largeur : 5
Edge > Region Focus Distance : 3
```

```
Edge > Region Focus Profondeur : 3
Edge > Region Focus Largeur : 5
Region > Edge Focus Distance : 3
Region > Edge Focus Profondeur : 3
Region > Edge Focus Largeur : 5
# -------------------------------------------------------------
# Parametres et seuils des procedures de validation
# -------------------------------------------------------------
[TestFreemanDeviation] Nombre de pixels : 20
[TestFreemanDeviation] Nombre de pixels d'amorce : 1
[TestFreemanDeviation] Seuil du gradient : 15
[TestFreemanDeviation] Seuil de la variation de direction : 45
#
[TestIrregularGradient] Nombre de pixels : 20
[TestIrregularGradient] Nombre de pixels d'amorce : 1
[TestIrregularGradient] Seuil de la longueur : 16
[TestIrregularGradient] Seuil du gradient : 22
[TestIrregularGradient] Seuil de fluctuation  : 0.18
#
[TestLowGradient] Nombre de pixels : 10
[TestLowGradient] Nombre de pixels d'amorce : 1
[TestLowGradient] Seuil du gradient du pixel candidat : 25
[TestLowGradient] Seuil de comparaison au gradient moyen : 0.60
#
[TestFewMaxiLocal] Nombre de pixels : 50
[TestFewMaxiLocal] Nombre de pixels d'amorce : 1
[TestFewMaxiLocal] Seuil du gradient moyen : 15
[TestFewMaxiLocal] Nombre de classes de gradient : 2
[TestFewMaxiLocal] Classes de gradient : 5 9
[TestFewMaxiLocal] Seuils sur la proportion de maxima locaux : 0.8 0.6 0.3
#
[TestHomogeneity] Nombre de pixels : 2500
[TestHomogeneity] Seuil de difference d'ecart-type : 30
#
[ContourVisibility] Nombre de classes de gradient : 2
[ContourVisibility] Classes de gradient : 10 20
[ContourVisibility] Seuils sur la longueur du contour : 8 6 3
#
[TestHighGradient] Facteur multiplicatif : 8.0
# -------------------------------------------------------------
Coefficient de calcul du seuil gradient : 2.25
Gradient minimum pour le calcul du seuil gradient : 2.00
Longueur minimum du contour pour la validation : 8
# -------------------------------------------------------------
# L'information de debuggage est une liste de mots cles
# indiquant les modules du systeme devant fournir une
# information de trace. Les mots cles utilisable sont :
# BACKUP EDGE_EVAL EDGE_FAIL
# EDGE_MERGING EDGE_PIXELS_LIST EDGE_PROCESS
# EDGE_SELECTION FILE_ACCESS PARAMETERS
# POPUP REGION_BOXES REGION_ENVELOP
# REGION_EVAL REGION_FAIL REGION_FILL
# REGION_MAP REGION_MERGING REGION_NEIGHBOUR_MAP
# REGION_PIXELS_LIST REGION_POINTER_MAP REGION_PROCESS
# REGION_SELECTION REGION_SHAPE REGION_VISUAL_MAP
# ROOT_FOCUS ROOT_INFO SCHEDULING
# SCHEDULING_KILL SEED SELECTION_STATUS
# SENDMAIL SYNCHRO SYNCHRO_HIGHLIGHT
# VALIDATION
# -------------------------------------------------------------
Information debuggage :
PARAMETERS
end
```

Dans le cadre d'un protocole expérimental, il devient indispensable de définir une politique de modification de ces paramètres. Une étude exhaustive de l'influence réciproque du rôle de chacun n'est pas très réaliste, même si idéalement elle s'avére être le seul moyen de "prouver" un algorithme, afin de justifier du rôle de tel ou tel constante fixée arbitrairement dans le programme. Tous les paramètres disposent d'une valeur par défaut, fixée empiriquement à la conception du programme, qui sera utilisée sauf mention contraire, dans le cadre des expérimentations qui suivent. Entre deux tests, toutes les variables initiales du système conserveront la même valeur, sauf mention contraire explicite. La politique appliquée sera donc celle du *toutes choses égales par ailleurs.*

La suite de ce chapitre s'articule autour de quatre parties, chacune d'elle mettant l'accent sur un point particulier de l'approche. La robustesse, la justesse, les possibilités d'adaptation, et les modes de coopération seront successivement étudiés dans les paragraphes suivants.

5.3.1 La robustesse du système

La robustesse de l'approche se décline en trois points, tout d'abord en montrant la stabilité par rapport aux conditions d'initialisation, puis par rapport aux données à segmenter, et enfin par rapport aux contraintes que l'utilisateur peut être amené à imposer au système.

5.3.1.1 Robustesse par rapport aux conditions initiales

La question de l'initialisation du système est un point d'intérêt de taille, car si une utilisation interactive du système autorise une mauvaise initialisation, il en est tout autrement dans le cadre d'un processus de traitement entièrement automatisé, où les cas d'échecs liés à une initialisation imparfaite doivent être minimisés.

Le système de segmentation est constitué à tout instant par une arborescence de processus. L'initialisation consiste à définir le ou les processus initiaux, leur emplacement, et leur type respectif. Cette liberté totale dans les conditions d'initialisation est paradoxalement gênante, car :

- elle ne garantit pas un résultat complet à l'issue du fonctionnement du système.
- elle ne garantit pas un résultat identique ou même comparable selon la position initiale du ou des processus initiaux.

Cette expérimentation propose d'initialiser de système de façon aléatoire, par un certain nombre de germes de type région, et d'étudier les résultats fournis par le système, en particulier :

- Quelle est la proportion des cas où le système fournit un résultat ?
- Quelle est la qualité de ce résultat par rapport aux cartes de référence ?
- Quelle est l'influence du nombre de germes initiaux ?

Initialisation par des processus de type région

Trois expérimentations ont été conduites, respectivement en plaçant 1,2 et 4 germes initiaux de type région sur l'image 128×128 des cellules musculaires *cellules256.gif*.

La carte de référence utilisée dans cet exemple comporte 19 régions identifiables en figure 5.12(b). Ces régions correspondent pour la plupart à des cellules. Trois régions correspondent au fond blanc de l'image, et une région regroupe deux cellules proches, qu'il n'a pas été jugé possible de différencier lors de la conception de la carte de référence. Sur un ensemble de 100 exécutions distinctes, la table 5.1 présente les scores de réussite du système, en fonction de l'emplacement du ou des pixels germe. Ainsi, la ligne *Nombre de résultats fournis* donne le nombre d'exécutions pour lesquelles la méthode a fourni un résultat en sortie. Ce score de 38 % s'améliore substantiellement dès que l'on augmente le nombre de germes initiaux. Effectivement, le fonctionnement du système dépend de manière cruciale du succès du germe initial, car seul ce premier processus peut créer d'autres processus de segmentation, et assurer ainsi la pérénité du système. L'augmentation du nombre de germes initiaux, même placés aléatoirement, augmente

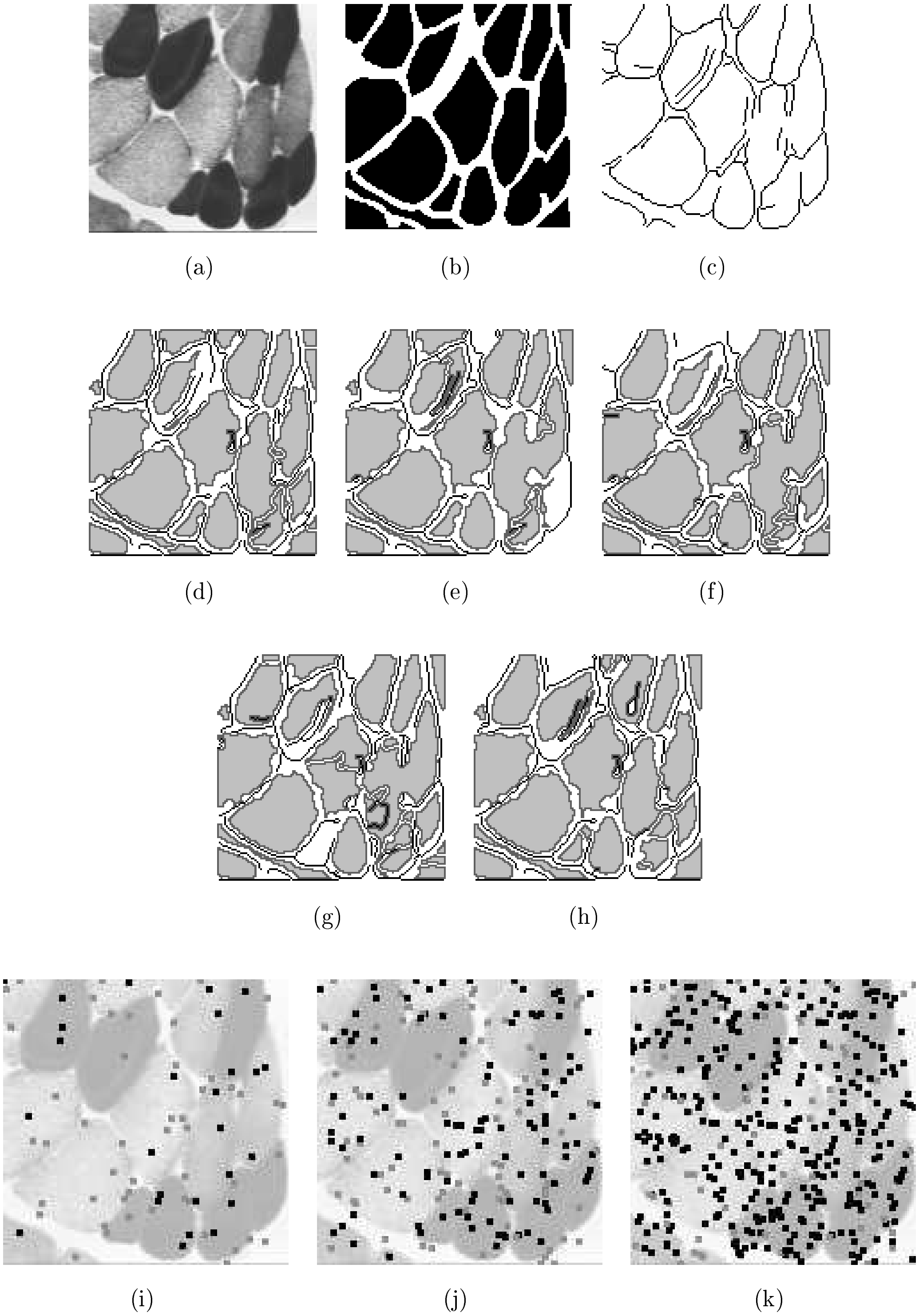


FIG. 5.12: Les images montrant la robustesse par rapport aux conditions initiales. (a) image en niveaux de gris utilisée. (b) carte des régions de référence. (c) carte des contours de référence. (d)-(h) exemples de résultats produits. (i)-(k) Carte des pixels germes de type région dans la configuration d'initialisation utilisant respectivement 1,2 ou 4 germes. Les points noirs donnent l'emplacement des germes des initialisations qui ont produit un résultat. Les points gris repèrent les germes pour lesquels l'initialisation n'a pas permis au système de démarrer.

Taille de l'image	128×128
Nombre de régions de la carte optimale	19
Nombre de pixels de la carte région optimale	11221 (68 % des pixels couverts)

	1 germe	2 germes	4 germes
Nombre d'exécutions :	100	100	100
Nombre de résultats fournis :	38	60	84
Sur les cas de succès			
Nombre moyen de régions :	22.05	23.32	23.74
Nombre moyen de pixels région :	9380	9708	10111
Par rapport à la carte région optimale			
Nombre moyen de régions communes :	10.6 ($\sigma = 3.2$)	10.6 ($\sigma = 2.6$)	11.2 ($\sigma = 1.9$)
Nombre moyen de régions non segmentées :	4.3 ($\sigma = 4.3$)	3.4 ($\sigma = 3.3$)	2.8 ($\sigma = 1.9$)
Nombre moyen de régions sur-segmentées :	2.1 ($\sigma = 2.1$)	5.0 ($\sigma = 1.9$)	5.0 ($\sigma = 1.5$)
Mesure de dissimilarité de Vinet :	0.34 ($\sigma = 0.108$)	0.33 ($\sigma = 0.089$)	0.31 ($\sigma = 0.057$)

TAB. 5.1: Statistique sur la robustesse des régions par rapport aux conditions d'initialisation dans le cas d'une initialisation par des processus de type région.

donc les chances de fonctionnement du système. On obtient ainsi un score de 84 % de résultats produits, avec seulement 4 germes dans une image 128×128.

La suite de la table 5.1 effectue des mesures sur les résultats produits par le système, lorsque celui-ci en fournit un. Il convient de noter que la proportion des pixels régions étiquetés dans l'image est particulièrement complet dans toutes les configurations d'initialisation, et se situe entre 9000 et 10000 pixels. Ce score est proche du *nombre de pixels de la carte région optimale* de 11221 pixels.

La concordance des résultats en terme de nombre de régions est finalement présentée dans la dernière partie de la table 5.1. Chaque région de la carte de référence est considérée séparément, et le nombre des régions du résultat qui l'intersectent est compté. La valeur du *Nombre de régions communes* compte le nombre des régions optimales intersectant une unique région du résultat, *Nombre de régions non segmentées* compte le nombre de régions optimales n'intersectant aucune région du résultat, et *Nombre de régions sur-segmentées* compte les régions optimales intersectant plus d'une région du résultat. Ce dernier cas constitue donc effectivement un cas de sur segmentation de l'image résultat [20]. Les résultats indiquent un bon niveau de correspondance exacte, autour de 10 régions en moyenne (par rapport aux 19 régions optimales), et qui n'est pas influencé par le nombre et l'emplacement des germes initiaux. La dernière ligne de la table 5.1 calcule la mesure de dissimilarité de Vinet [Vin91], décrite au paragraphe 5.2.4.

L'examen plus particulier de la carte des contours produite par le système amène des commentaires similaires à ceux faits concernant les régions. La table 5.2 présente quelques valeurs significatives calculées sur les images résultats fournies par le système. Cette table permet d'effectuer les remarques suivantes :

- l'augmentation du nombre de germes initiaux dans l'image contribue à augmenter les cas où le système fournit un résultat [21].
- En limitant ensuite l'étude à ces cas-là (respectivement 32, 57 et 83 % des 100 expérimentations

[20] Pour être complet, il conviendrait de réaliser le même comptage des intersections en permutant le rôle de l'image résultat et de l'image de référence, ce qui permettrait alors d'obtenir une mesure de la sous-segmentation de l'image résultat : une région du résultat qui couvre plusieurs régions de la carte de référence.

[21] Les cas de succès du système en terme de primitives de type contour et région sont d'ailleurs étroitement liés, en particulier à cause des mécanismes coopératifs mis en jeu. Il est peu probable que le système produise un résultat contenant uniquement des contours ou des régions. Respectivement 62, 40 et 16 résultats vides de contours *et* de régions sont obtenus avec 1, 2 et 4 germes, ce qui signifie que les 35 cas de succès observés pour les contours sur la table 5.2, et les 38 cas pour les régions de la table 5.1, obtenus avec un unique germe, correspondent en partie aux mêmes résultats.

Taille de l'image	128×128
Nombre de pixels contours de la carte optimale	1289

	1 germe	2 germes	4 germes
Nombre d'exécutions :	100	100	100
Nombre de résultats fournis :	35	57	83
Sur les cas de succès			
Nombre moyen de pixels contours :	1040 ($\sigma = 76.2$)	1054 ($\sigma = 64.8$)	1053 ($\sigma = 63.4$)
Par rapport à la carte optimale des contours :			
Nombre moyen de pixels strictement communs :	514 ($\sigma = 29.6$)	520 ($\sigma = 26.1$)	518 ($\sigma = 26.2$)
Pourcentage de pixels strictement communs :	39.91 %	40.32 %	40.18 %
Nombre moyen de pixels communs (dist=1) :	970 ($\sigma = 59.7$)	983 ($\sigma = 49.5$)	983 ($\sigma = 47.9$)
Pourcentage de pixels communs (dist=1) :	75.23 %	76.25 %	76.26 %

TAB. 5.2: Statistique sur la robustesse des contours par rapport aux conditions d'initialisation dans le cas d'une initialisation par des processus de type région.

effectuées avec 1, 2 et 4 germes initiaux de type région), la quantité de pixels contours obtenus est particulièrement stable, quel que soit le nombre de germes initiaux.

- Ces pixels contours réalisent une bonne couverture de la carte des contours de référence, puisque les 3/4 des pixels de cette carte de référence sont segmentés.
- L'introduction d'une faible distance d'incertitude s'avére indispensable pour localiser les correspondances entre les pixels de la carte de référence, et les pixels fournis par l'algorithme, car sur les 75 % de pixels en correspondance entre les deux images, seulement 40 % réalisent une correspondance exacte. L'autre partie est donc, dans notre exemple, à une distance de un pixel [22] du pixel de la carte de référence. Le paragraphe 5.2.4 justifie l'utilisation d'une carte de référence, ainsi que d'une distance d'incertitude pour localiser les pixels contours.

Initialisation par des processus de type contour

L'expérimentation précédente est reproduite, initialisant le système par le placement aléatoire de pixels de type contour dans l'image, et en observant la carte résultat produite à l'issue de l'exécution. Des mesures sont effectuées, à la fois sur la carte des contours et la carte des régions du résultat. La même portion d'image est utilisée. Un plus grand nombre d'expérimentations est effectué – 200 au lieu de 100 – afin d'obtenir des statistiques plus représentatives, et également parce qu'il semble plus difficile de placer correctement un pixel germe de type contour aléatoirement qu'un pixel de type région. En effet, les zones de transition de l'image, sur lesquelles les germes de processus contour sont appelés à démarrer, sont en nombre beaucoup plus restreint que les zone homogènes, sur lesquelles les processus régions peuvent être initialisés avec succès.

Les tables 5.3 et 5.4 ne confirment pas l'hypothèse précédente, puisque l'on observe pratiquement les mêmes proportions de succès qu'avec l'initialisation par les régions (respectivement environ 35, 50 et 80 % avec 1, 2 et 4 germes initiaux). La figure 5.13 indique par exemple les cas de succès de la méthode, lorsqu'un unique germe est utilisé. Ces pixels sont représentés en noir sur cette image. Ainsi ces pixels sont logiquement localisés à proximité directe d'une zone de fort contraste dans l'image. Les cas d'échec correspondent alors à une mauvaise initialisation du germe initial.

[22] Cela signifie qu'un pixel résultat de l'algorithme est contenu dans une fenêtre 3×3 centrée sur un pixel contour de la carte de référence.

Taille de l'image	128×128
Nombre de régions de la carte optimale	19
Nombre de pixels de la carte région optimale	11221 (68 % des pixels couverts)

	1 germe	2 germes	4 germes
Nombre d'exécutions :	200	200	200
Nombre de résultats fournis :	71 (35.5 %)	109 (54.5 %)	157 (78.5 %)
Sur les cas de succès			
Nombre moyen de régions :	24.03	23.69	24.64
Nombre moyen de pixels région :	10173	10139	10181
Par rapport à la carte région optimale			
Nombre moyen de régions communes :	11.56 ($\sigma = 1.91$)	11.48 ($\sigma = 1.95$)	11.24 ($\sigma = 2.00$)
Nombre moyen de régions non segmentées :	2.56 ($\sigma = 1.84$)	2.72 ($\sigma = 1.61$)	2.61 ($\sigma = 1.55$)
Nombre moyen de régions sur-segmentées :	4.87 ($\sigma = 1.74$)	4.79 ($\sigma = 1.73$)	5.14 ($\sigma = 1.76$)
Mesure de dissimilarité de Vinet :	0.316 ($\sigma = 0.064$)	0.306 ($\sigma = 0.052$)	0.312 ($\sigma = 0.054$)

TAB. 5.3: Statistique sur la robustesse des régions par rapport aux conditions d'initialisation dans le cas d'une initialisation par des processus de type contour.

Taille de l'image	128×128
Nombre de pixels contours de la carte optimale	1289

	1 germe	2 germes	4 germes
Nombre d'exécutions :	200	200	200
Nombre de résultats fournis :	76 (38.0 %)	116 (58.0 %)	163 (81.5 %)
Sur les cas de succès			
Nombre moyen de pixels contours :	976.9 ($\sigma = 287.7$)	985.0 ($\sigma = 270.5$)	1019.1 ($\sigma = 217.5$)
Par rapport à la carte optimale des contours			
Nombre moyen de pixels strictement communs :	478.9 ($\sigma = 140.9$)	485.0 ($\sigma = 132.9$)	500.0 ($\sigma = 105.7$)
Pourcentage de pixels strictement communs :	37.15 %	37.62%	38.79%
Nombre moyen de pixels communs (dist=1) :	905.5 ($\sigma = 265.1$)	917.2 ($\sigma = 250.2$)	947.0 ($\sigma = 199.5$)
Pourcentage de pixels communs (dist=1) :	70.25 %	71.16 %	73.47 %

TAB. 5.4: Statistique sur la robustesse des contours par rapport aux conditions d'initialisation dans le cas d'une initialisation par des processus de type contour.

Les autres mesures sont très analogues à celles de l'expérimentation précédente, ce qui tend à prouver que le système est relativement indépendant du mode d'initialisation, par des germes de type contour ou région, mais par contre très sensible au nombre de processus initiaux, comme cela a déjà été dit. Il convient cependant de souligner à nouveau qu'un nombre de germe extrêmement restreint permet d'obtenir des résultats complets dans la grande majorité des cas. Un pourcentage de 80 % de réussite est obtenu, avec seulement 4 pixels germes initiaux, qu'il s'agisse de processus de type contour ou de type région.

5.3.1.2 Robustesse par rapport aux données

L'objectif de cette partie est de mettre l'accent sur un point qui nous semble très important et qui justifie d'une certaine manière les bonnes performances en terme de complétude des résultats observés au paragraphe 5.3.1.1. Le terme de robustesse par rapport aux *données* s'entend dans le sens où l'algorithme proposé s'adapte facilement à la topologie de l'image sur laquelle il travaille, et présente une tolérance importante aux échecs. Cette tolérance trouve son explication dans la manière dont la population de processus se propage dans l'image.

Les expérimentations de ce paragraphe vont tenter de mettre en évidence cette tolérance. Pour cela, les points suivants sont essentiels :

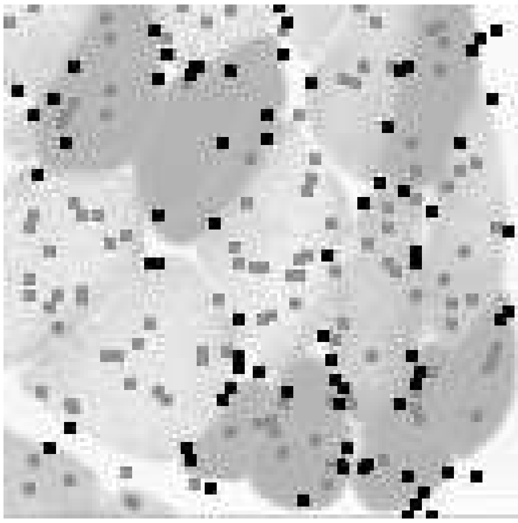


FIG. 5.13: La carte des germes de type contour pour lesquels le système produit un résultat. Ces pixels apparaissent en noir sur l'image. Les autres pixels en gris montrent les cas d'échec du système. L'initialisation se fait avec un seul processus de type contour, donc un seul germe. Chaque pixel de cette image correspond donc à une exécution distincte du système.

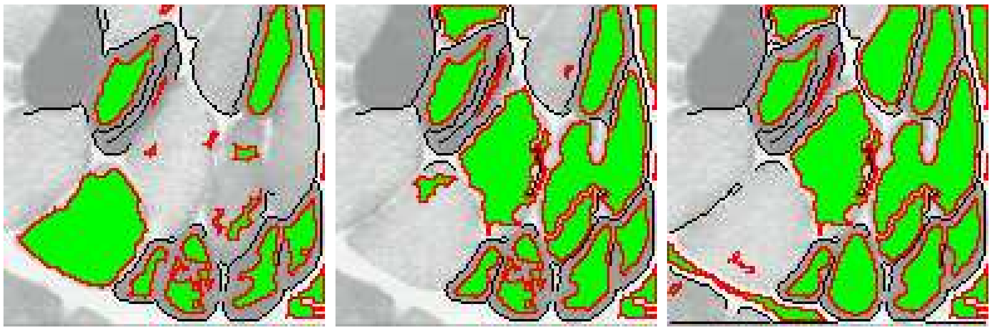
- L'échec d'une primitive ne remet pas en cause le fonctionnement global du système [23].
- L'émergence d'une primitive peut se faire par différentes *voies d'accès*, à différents moments du fonctionnement du système. Aussi, l'échec de la segmentation d'une primitive à un instant donné, ne signifie pas que cette primitive ne sera pas segmentée par la suite, lorsqu'un autre processus distinct du premier en fera à nouveau la demande, dans un tout autre contexte d'émergence d'information.

Afin d'illustrer ces deux arguments, une légère modification est apportée au fonctionnement du système, ayant pour but de *simuler* de façon radicale l'échec de la segmentation d'une primitive. Une zone rectangulaire est définie par l'utilisateur à l'intérieur d'un objet de l'image, voir la figure 5.14(a). A chaque cycle de séquencement du système, le présence d'une région dans cette zone est testée, et la primitive est immédiatement effacée, ainsi que le processus en charge de sa segmentation. Cette destruction permet de simuler l'échec de la segmentation d'une région de façon tout à fait transparente pour les autres processus. A tout instant, cette zone de *no man's land* reste donc exempte de toute primitive région.

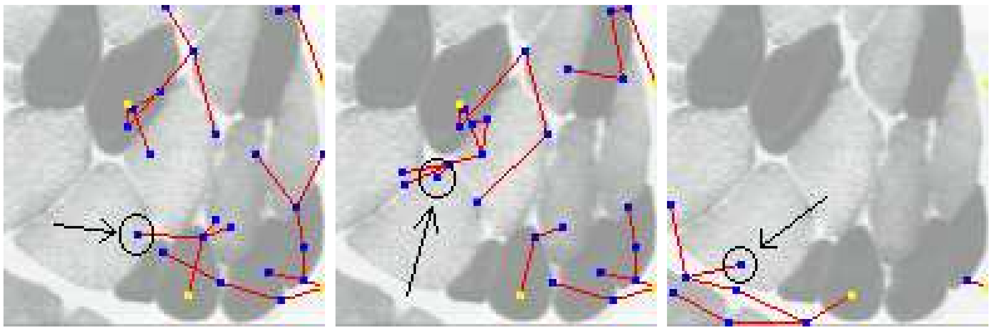
La système de segmentation est initialisé dans cette configuration, avec 5 processus initiaux de type région et contour placés dans l'image. Les images 5.14(d), (e) et (f) montrent les états intermédiaires de la segmentation dans lesquels un processus de type région a tenté de segmenter la région non autorisée. Les images 5.14(g), (h) et (i) correspondent à l'état de l'arbre des processus pour ces mêmes états de segmentation. Les nœuds de ces arbres représentent un processus, localisé dans l'image suivant la position de son pixel germe initial. Les arcs entre les processus représentent les relations de filiation. Les processus initiaux sont indiqués par une couleur jaune particulière (gris clair). Le processus correspondant à la région qui a pénétré la zone interdite est marqué par une flèche sur les images (g), (h) et (i). L'étape qui suit immédiatement la prise de ces images est l'effacement de la primitive segmentée dans les images (d), (e) et (f), ainsi que la destruction du processus de segmentation associé dans les images

[23]Cet argument parait logique. La création de nouveaux processus répond à une demande du processus père pour augmenter la quantité d'information de l'environnement. L'échec d'un fils ne produira pas l'information souhaitée, le processus père disposera donc de moins de données pour prendre sa décision, cette dernière risque d'être moins sûre, cependant le fonctionnement du processus père n'en est pas pour autant déstabilisé.

(a) zone bloquée. (b) final à t=1529 s. (c) final 2 à t=1034 s.



(d) t=346 s. (e) t=665 s. (f) t=894 s.



(g) t=346 s. (h) t=665 s. (i) t=894 s.

FIG. 5.14: Illustration de la robustesse du système par la diversité des *voies d'accès* à la segmentation d'une primitive. (a) une zone rectangulaire au milieu d'un objet de l'image définit un lieu où toute primitive segmentée sera supprimée immédiatement. (b) le résultat final avec la destruction des primitives de type région. (c) le résultat sans activer la destruction des primitives de type région. (d)-(f) des résultats partiels du système en cours de segmentation, aux moments où un processus de segmentation tente de segmenter la zone interdite. (g)-(i) arbre des processus aux instants correspondants aux résultats partiels précédents. Le processus ayant effectué *l'intrusion* est repéré par une flèche.

(g), (h) et (i) comme prévu.

A partir de ces trois instants du fonctionnement du système, il est intéressant de constater que :

- Plusieurs tentatives de segmentation, se produisant à des instants différents, ont lieu, et ne préjugent pas de l'historique des tentatives de segmentation antérieures.
- Les processus parents qui génèrent ces régions au sort funeste sont distincts pour chacun des trois essais de segmentation. Concernant la tentative à t=346 s, le processus père créateur de cette région est le contour des cellules noires à droite du rectangle. A t=665 s, il s'agit d'un morceau de contour situé dans la partie supérieure de cette région texturée [24], et enfin à la date t=894 s, la dernière tentative de segmentation provient d'un autre contour localisé en bas à gauche du rectangle d'intérêt.

Cet exemple illustre clairement que le système est particulièrement tolérant aux échecs individuels, et que cela ne remet pas en cause le fonctionnement global de l'ensemble. De plus, la délégation des tâches au niveau local présente l'intérêt de pouvoir segmenter une primitive par des *voies d'accès* multiples, sans préjuger d'un échec antérieur. La stratégie de fonctionnement consistant pour un processus à segmenter la primitive si son pixel germe est placé en zone non étiquetée, ou à simplement retourner un pointeur vers l'objet existant dans le cas contraire, permet cette tolérance. Les processus parents n'ont pas de connaissance sur la façon dont leurs fils ont fonctionné. En effet, les processus parents récupèrent un pointeur vers un objet de la part de leurs processus fils, mais ignorent la manière dont ce pointeur a été acquis soit par segmentation effective, soit par *appropriement* de la primitive d'autrui.

5.3.1.3 Robustesse par rapport aux contraintes système

L'objectif de cette partie est de montrer la robustesse du système par rapport à des contraintes internes appliquées sur le nombre de processus pouvant fonctionner dans le système. Deux approches étaient envisageables pour limiter le nombre de processus : une approche centralisée imposant des contraintes globales sur l'arbre des processus, en terme de degré, de largeur, de profondeur, ou bien une approche distribuée laissant chaque processus de segmentation moduler le nombre de processus fils qu'il décide de générer. Cette limitation distribuée est applicable :

- dans le cas des régions, où le nombre de contours fils est proportionnel au nombre de pixels de la frontière de la région. La limitation du nombre de processus se fait en ajustant ce coefficient de proportionnalité.
- dans le cas des contours, où le nombre de processus est fixe à chaque interruption de la construction. Mais il est cependant possible de paramétrer la fréquence de ces arrêts [25].

La philosophie distribuée des traitements a naturellement conduit à utiliser la deuxième possibilité [26].

---

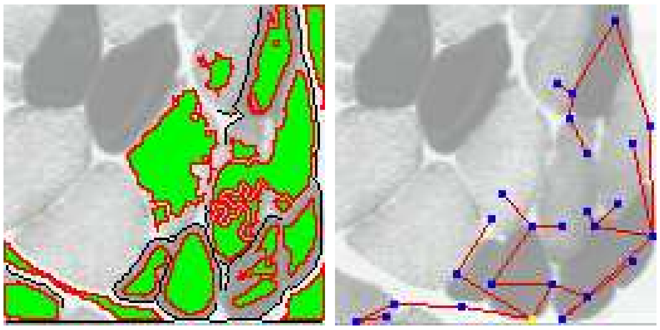
[24] Le fonctionnement de ce processus contour est exemplaire sur cet exemple précis, puisque, en se référant à l'arbre des processus de l'image 5.14(h), on constate que ce processus contour génère trois fils : deux sont situés dans les deux régions texturées, et le troisième au milieu est cadré sur la transition entre ces deux régions, laissant à penser qu'il s'agit d'un processus de type contour. Ceci correspond parfaitement au schéma coopératif décrit précédemment.

[25] Ce paramétrage a le même effet que la modification du *quantum de temps* décrit au paragraphe 3.4.2.

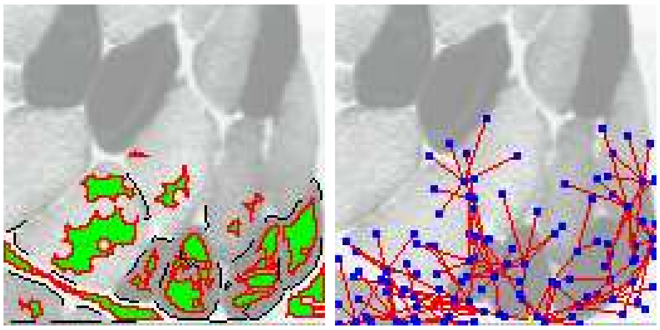
[26] La limitation distribuée, telle qu'elle est présentée n'agit que sur le degré de l'arbre des processus, à la différence d'une limitation centralisée qui pourrait aussi agir sur les paramètres globaux, tels que sa profondeur. La limitation distribuée est cependant suffisante. Par des contraintes locales, elle agit efficacement sur la valeur globale du nombre de processus de segmentation à chaque instant dans le système.

	Temps CPU (s)	Nb de cycles de séquencement	Nb de processus	Nb de contours	Nb de régions	Charge maxi (actifs)	Charge maxi (attente)
ER=150 EC=100	12.95	1546	196	50	51	38	24
ER=100 EC=50	13.12	2063	300	84	76	30	33
ER=50 EC=25	16.64	2703	633	172	146	68	48
ER=25 EC=10	21.59	4463	1360	336	254	120	84
ER=25 EC=10 sans fusion	19.14	3426	1289	342	238	112	84

TAB. 5.5: Effet d'une contrainte locale sur le nombre de processus fils du système de segmentation. Les expérimentations font varier la valeur de ER, échantillonnage de la frontière de la région, exprimée en pixels, ainsi que EC, échantillonnage de la longueur du contour, exprimée en pixels. La valeur *Nb de cycles de séquencement* indique le nombre de passages dans la boucle du séquenceur pour exécuter un des états de l'automate d'un des processus de segmentation. Les deux dernières colonnes indiquent les pointes de charge du système au cours de l'exécution.



(a) résultat (b) processus



(c) résultat (d) processus

FIG. 5.15: Résultats intermédiaires illustrant les effets de la limitation de la prolifération des processus. (a) Etat courant de la segmentation pour le couple $(ER, EC) = (150, 100)$. (b) Arbre des processus correspondant à l'état de segmentation (a). (c) Etat courant de la segmentation pour le couple $(ER, EC) = (25, 10)$. (d) Arbre des processus correspondant à l'état de segmentation (c).

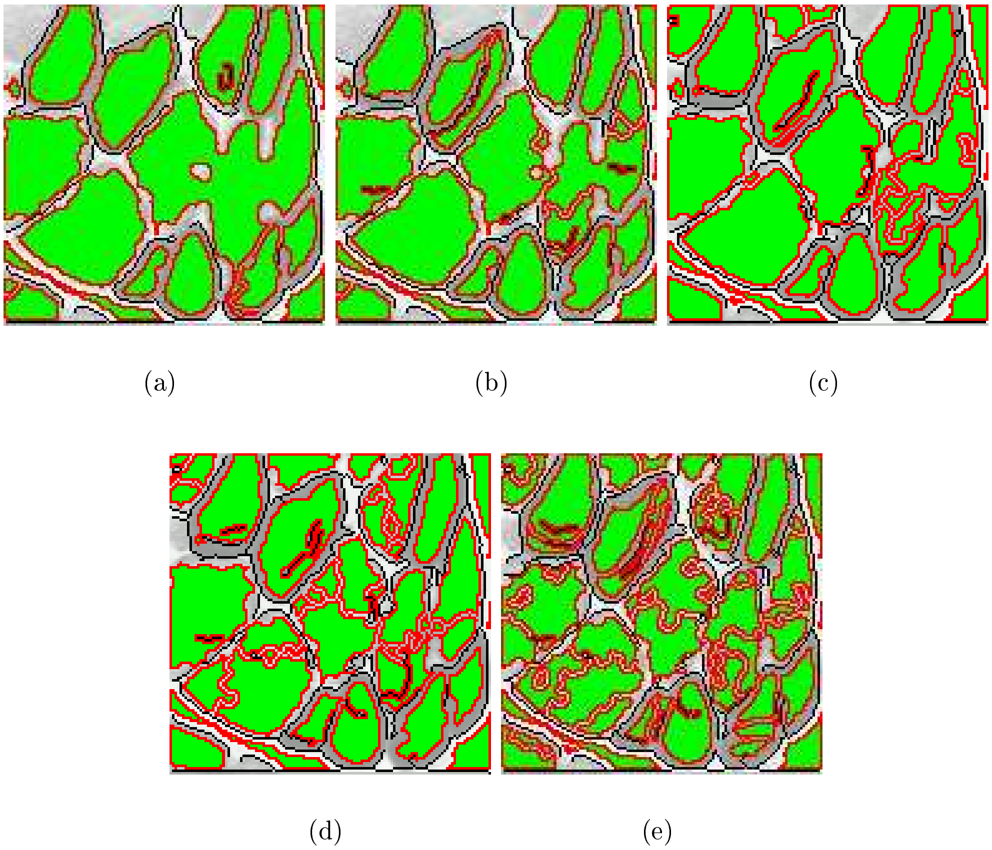


Fig. 5.16: Résultats de segmentation obtenus avec différentes limitations de la prolifération locale des processus de segmentation. Les résultats correspondent aux valeurs données en table 5.5. (a) $ER = 150$ et $EC = 100$. (b) $ER = 100$ et $EC = 50$. (c) $ER = 50$ et $EC = 25$. (d) $ER = 25$ et $EC = 10$. (e) Même configuration que (d), dans laquelle le protocole de fusion de primitives de type région et contour n'est plus activé.

La prolifération des processus est liée au nombre de processus fils générés par un processus père. Pour une région, ce nombre de contours fils est déduit d'un échantillonage de la frontière de la région, consistant à initialiser un contour tous les ER pixels de la frontière. Pour le contour, la prolifération se traduit par le nombre maximum de pixels aggrégés au contour EC entre deux requêtes de focalisation de type région consécutives.

L'expérimentation consiste à faire fonctionner le système avec différentes limitations de la prolifération des processus, afin de valider l'image résultat obtenue dans chacun des cas (figure 5.16, ainsi que les conditions de fonctionnement observées dans la table 5.5). Cette table permet tout d'abord de constater que le temps CPU utilisé augmente lorsque le nombre de processus de segmentation augmente dans l'image. Il en est de même des différentes structures mises en jeu dans le système : structures de processus, de contours, de régions.

Ces valeurs indiquent que le nombre de primitives de type région augmentent significativement lorsque le couple (ER, EC) diminue. Cependant, les objets à segmenter dans l'image ne varient pas en nombre. Donc, en l'absence de tout protocole de fusion de primitives, le système s'achemine vers une sur-segmentation importante [27]. Les deux dernières lignes de la table 5.5, ainsi que les images de résultat correspondantes en figure 5.16(d) et (e) présentent une petite comparaison, selon que le protocole de fusion de primitives est activé ou pas. La table n'indique pas une grande différence dans le nombre de primitives région crées tout au long du fonctionnement du système (254 contre 238), pourtant l'image 5.16(d) dispose de beaucoup moins d'objets de type région, que l'image (e), indiquant ainsi que de nombreuses régions initialement créées, ont peu à peu été incorporées à d'autres régions existantes.

La figure 5.15 présente les résultats partiels obtenus par le système de segmentation. Les images (b) et (d) donnent une idée de la différence de population de processus entre les deux cas extrêmes définis par les couples de valeurs $(ER, EC) = (150, 100)$ et $(50, 25)$.

Les résultats de la figure 5.16, montrent que des résultats complets sont obtenus, quel que soit le couple de valeurs (ER, EC) utilisé pour limiter le nombre de processus. Les images (b), (c) et (d) montrent cependant une sur-segmentation qui tend à augmenter, lorsque le nombre de processus augmente. La stratégie de fusion de primitives mise en jeu semble arriver à saturation lorsque le nombre de processus, et donc le nombre d'objets fusionable est très important. Inversement l'image (a) présente une certaine sous-segmentation, puisque trois cellules texturées importantes ont été segmentées par la même primitive région. Ce comportement est logique, puisqu'en deçà d'une certaine population de processus fils, les informations locale ne sont plus segmentées en quantité suffisamment importante (exemple de contours pour cerner la frontière des régions) pour garantir un fonctionnement sûr des processus parents.

Ces constatations justifient le comportement graduel instauré dans notre système, consistant pour un processus à adapter le couple (ER, EC), en fonction de la charge courante du système. Lorsque peu de processus travaillent dans l'image, chacun peut créer davantage de processus fils, alors que lorsque le nombre de processus dépasse un certain plafond, une limitation graduelle du nombre de processus intervient, ayant pour effet de faire redescendre le nombre total de processus courant en deçà du plafond. Cette contre-réaction permet d'implanter un mécanisme distribué d'équilibrage dynamique de la charge du système. La table 5.6 donne un exemple de valeurs utilisées pour ER et EC en fonction de la charge du système (nombre de processus dans le système, en cours d'exécution, ou en attente).

[27] Plusieurs primitives de type région se partagent les pixels d'un même objet. Parallèlement, les primitives de type contour sont plus courtes, bien que cela n'apparaisse pas visuellement, car de tels contours sont bien souvent jointifs à leur extrémité.

	Charge < 20	20 < Charge < 30	30 < Charge < 40	40 < Charge
ER (en pixels)	50	60	80	100
EC (en pixels)	100	150	200	300

TAB. 5.6: Exemple de limitation dynamique de la charge. *Charge* indique le nombre de processus instantané du système, tant actifs qu'en attente.

5.3.2 La justesse du système

Cette partie se décompose en six expérimentations, menées sur des images distinctes, synthétiques et naturelles, issues de domaines divers (biomédical, scène d'intérieur, d'extérieur, portrait). Pour chacune de ces images, une carte des contours de référence a été construite. L'objectif est de mesurer la qualité de la carte contours produite, ainsi que de comparer ce résultat avec une méthode classique. La méthode classique retenue est le détecteur de Deriche [Der87].

Le protocole expérimental suivi consiste à :

- rechercher les meilleurs paramètres du détecteur de Deriche pour l'image étudiée. Ces paramètres sont ceux qui offrent le meilleur compromis entre sur- et sous-détection par rapport aux contours de référence [28] [29].
- rechercher un jeu de paramètres initiaux pour le système coopératif assurant également le meilleur compromis entre sur- et sous-détection. La table 5.25 du paragraphe 5.3.2 présente un aperçu des variantes de configuration obtenues pour les différentes images testées.
- effectuer des comparaisons quantitatives entre le *meilleur* résultat de Deriche et la carte de référence.
- effectuer des comparaisons quantitatives entre le *meilleur* résultat du système coopératif et la carte de référence.
- dresser des comparaisons qualitatives et quantitatives entre le résultat du système coopératif et le résultat de Deriche.

Les paragraphes suivants décrivent les résultats obtenus avec les différentes images utilisées. Le dernier paragraphe donne des tableaux synthétisant ces divers résultats.

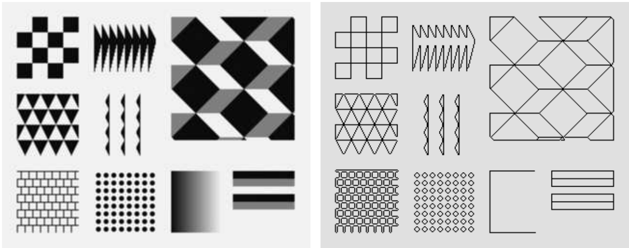
Comparaison avec la carte de référence - fleck320

L'image synthétique utilisée est tirée des travaux de Fleck [Fle92]. Elle ne présente pas de difficultés de détection particulières dans le sens où tous les objets sont très contrastés. Elle met spécialement l'accent sur la gestion des petites structures géométriques, sur les jonctions, les angles et les intersections. A ce titre, la partie la plus problématique de cette image, en figure 5.17, est le petit quadrillage en bas à gauche, car les transitions qu'il met en jeu sont de type *ligne*, et un détecteur de contour de type *marche* devra donc étiqueter deux rangées de pixels contours, de chaque côté de la transition de type *ligne*. Cet étiquetage pose des problèmes topologiques et de localisation *sub-pixel* [30] lorsque l'on impose comme c'est le cas ici qu'un pixel

[28] la somme des pixels détectés *à tort* et *en trop* est minimisée.

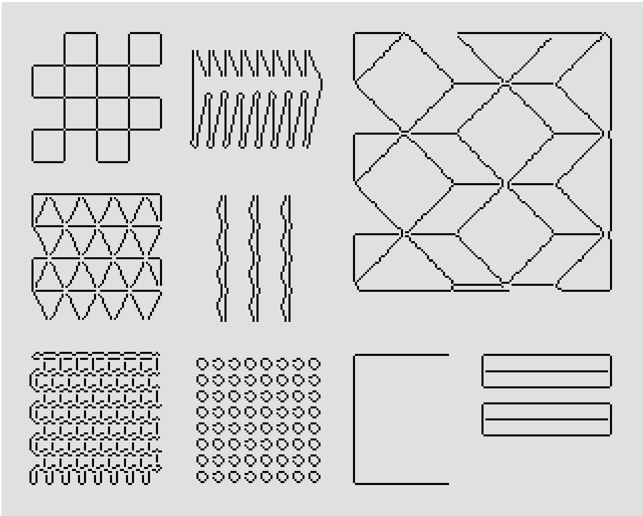
[29] La recherche des trois paramètres "optimaux" du Deriche – α et les deux seuils de l'hystérésis – s'obtient en minimisant la somme de la proportion de pixels non trouvés et de la proportion de pixels trouvés en trop. Cette recherche se fait dans un espace de dimension trois. L'approche choisie consiste ici à minimiser la fonction en ne faisant varier qu'un seul paramètre à la fois, c'est-à-dire en minimisant la fonction successivement sur ses trois dimensions.

[30] Idéalement un contour devrait pouvoir être localisé entre deux pixels.

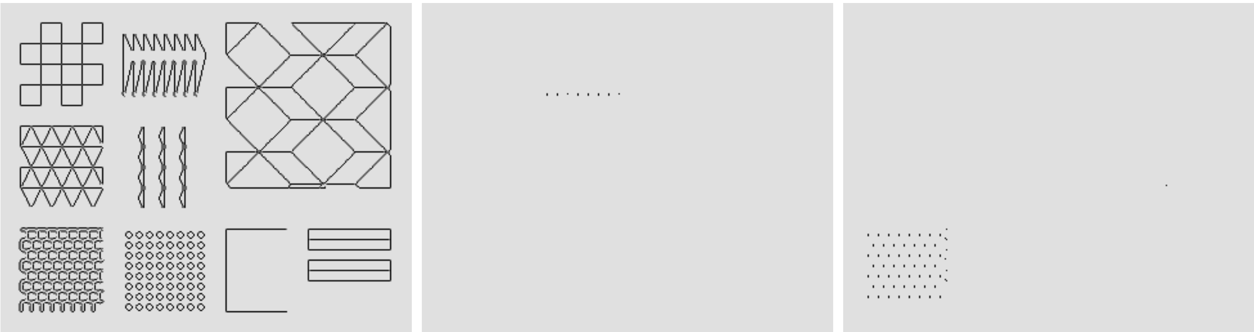


(a) Niveaux de gris. (b) Contours de référence.

Fig. 5.17: L'image en niveaux de gris *fleck320.gif* et la carte des contours de référence correspondante.



(a) Résultat de Deriche.



(b) Contours communs. (c) Contours en trop. (d) Contours manquants.

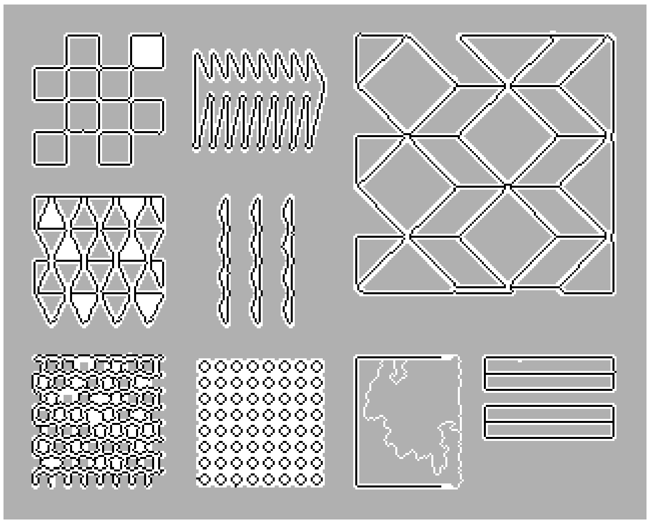
Fig. 5.18: Les résultats de la méthode de Deriche sur l'image *fleck320.gif* obtenus avec une recherche de paramètres optimaux, ici $\alpha = 2.44$, $sb = 17$ et $sh = 17$, respectivement pour le paramètre de lissage, et les seuils bas et haut de l'hystérésis.

Nombre de pixels de l'image de référence	6334	
Nombre de pixels de l'image résultat	5990	
PIXELS COMMUNS		
Evaluation stricte (dist=1)	4582	
par rapport a l'image de référence		72.3 %
par rapport a l'image résultat		76.5 %
Evaluation large (dist=3)		
par rapport a l'image de référence	6219	98.2 %
par rapport a l'image résultat	5976	99.8 %
PIXELS NON TROUVES		
Evaluation stricte (dist=1)	1752	27.7 %
Evaluation large (dist=3)	115	1.8 %
PIXELS TROUVES A TORT		
Evaluation stricte (dist=1)	1408	23.5 %
Evaluation large (dist=3)	14	0.2 %
PIXELS COMMUNS (des 2 images), dist=3, (7613)		
Moyenne du gradient	54.6	
Variance du gradient	16.7	
PIXELS NON TROUVES (115)		
Moyenne du gradient	36.5	
Variance du gradient	5.5	
PIXELS TROUVES A TORT (14)		
Moyenne du gradient	35.5	
Variance du gradient	2.8	

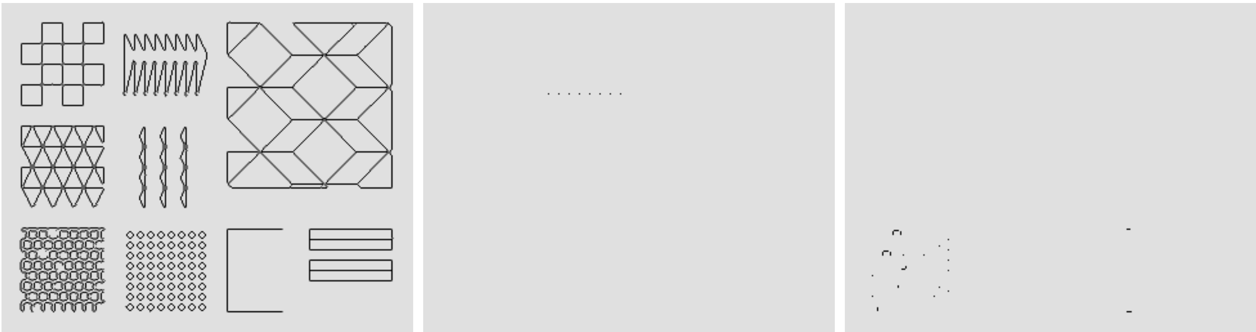
TAB. 5.7: Comparaison entre le résultat du détecteur de Deriche et la carte des contours de référence de l'image *fleck320.gif*.

Nombre de pixels de l'image de référence	6334	
Nombre de pixels de l'image résultat	6178	
PIXELS COMMUNS		
Evaluation stricte (dist=1)	4708	
par rapport a l'image de référence		74.3 %
par rapport a l'image résultat		76.2 %
Evaluation large (dist=3)		
par rapport a l'image de référence	6285	99.2 %
par rapport a l'image résultat	6170	99.9 %
PIXELS NON TROUVES		
Evaluation stricte (dist=1)	1626	25.7 %
Evaluation large (dist=3)	49	0.8 %
PIXELS TROUVES A TORT		
Evaluation stricte (dist=1)	1470	23.8 %
Evaluation large (dist=3)	8	0.1 %
PIXELS COMMUNS (des 2 images), dist=3, (7747)		
Moyenne du gradient	54.0	
Variance du gradient	16.7	
PIXELS NON TROUVES (49)		
Moyenne du gradient	32.7	
Variance du gradient	12.7	
PIXELS TROUVES A TORT (8)		
Moyenne du gradient	37.9	
Variance du gradient	0.5	

TAB. 5.8: Comparaison entre le résultat de l'algorithme de segmentation et la carte des contours de référence de l'image *fleck320.gif*.



(a) Résultat de la méthode coopérative.



(b) Contours communs. (c) Contours en trop. (d) Contours manquants.

FIG. 5.19: Les résultats du système de segmentation sur l'image *fleck320.gif*.

contour réalise le maximum local du gradient : deux lignes de pixels de contours ne peuvent pas se toucher pour des raisons de maximalité du gradient.

Les deux résultats proposés en figure 5.18 et 5.19 sont complets, et présentent des défauts comparables. Les jonctions sont assez mal appréhendées, spécialement dans la structure de gauche à petits triangles. Il s'agit toutefois d'un cas particulièrement délicat, puisque six contours doivent en théorie partir de chaque pixel de jonction. La structure en forme de peigne a posé également des difficultés de segmentation, puisque l'on remarque que les changements de direction formant des angles importants sont très arrondis. Ce motif a provoqué la détection de pixels *en trop* pour le Deriche, voir la figure 5.18(c), ce qui signifie que les quelques pixels détectés ici sont à une distance de plus de un de leur position *optimale*. La délocalisation est donc flagrante ici. La forte valeur de α utilisée ici implique pourtant un faible lissage de l'image dans l'algorithme du Deriche, ce qui n'est pas suffisant pour empêcher la délocalisation. A titre de curiosité, un test du Deriche avec un fort lissage ($\alpha = 0.6$) ne trouve aucune des extrémités de cette structure, voir la figure 5.20.

La carte de résultats de l'algorithme coopératif (figure 5.19) présente pour sa part une information capitale, et qui ne pouvait être obtenue par aucun détecteur de contours, mais qui est pourtant exploitable par le résultat fourni. Le motif en dégradé présente clairement une transition sur sa partie droite avec le fond de l'image : il y a une séparation entre l'intérieur du motif, qui est une zone en dégradé homogène, et le fond de l'image, qui est une zone non-texturée sans dégradé. Le problème réside ici dans le fait que le gradient n'est pas porteur de l'information permettant de discriminer ces deux zones. Seule la construction des deux régions, et leur non-fusion (pour raison d'homogénéité différente) permet de faire apparaître cette transition particulière. Le résultat proposé dispose donc de deux régions définissant le dégradé (qui auraient dû fusionner), et d'une région unique pour le fond. Ces deux types de régions apportent donc une information plus riche qu'une carte contour seule ne peut intrinsèquement pas produire. Le cas se représentera dans les autres images naturelles utilisées.

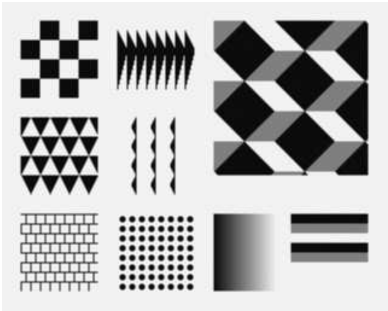
Comparaison avec la carte de référence - cellules256

L'image de cellules musculaires en coupe (voir la figure 5.21) présente des difficultés classiques. Des zones homogènes sombres sont séparées par des contours de type *trait* faiblement marqués. Des zones plus texturées présentent des délimitations également délicates à obtenir. La segmentation optimale fournie par le détecteur de Deriche est présentée en figure 5.22, ainsi que sa mesure de qualité par rapport à l'image de référence dans la table 5.9. Le résultat du système de segmentation coopératif est fourni en figure 5.23, ainsi que la mesure de qualité dans la table 5.10.

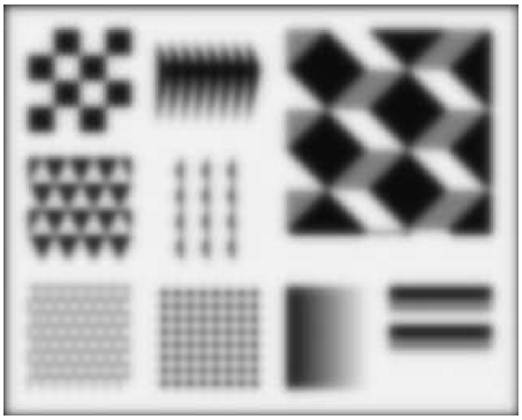
La carte des contours de Deriche est un compromis en sur- et sous-détection. Ce compromis est particulièrement difficile à obtenir pour les frontières entre deux cellules texturées. Elles présentent peu de différences photométriques avec des artefacts de contours situés au cœur de la texture. La différence se fait visuellement très bien, grâce à l'alignement et au nombre des pixels du contour marquant une frontière, particularité à laquelle le détecteur de Deriche ne sera pas sensible.

Le résultat de Deriche se place donc à la limite, où peu de contours invalides sont placés en texture, et où cependant les frontières entre ces mêmes textures sont détectées. Elles possèdent un gradient moyen un peu plus significatif que la gradient moyen de la texture. Ce résultat manque cependant des contours faibles mais très visibles entre les zones homogènes noires de l'image. Ce résultat est visuellement très bon selon nous, et réalise un bon score (18 % de pixels non trouvés, et 7 % de pixels trouvés en trop).

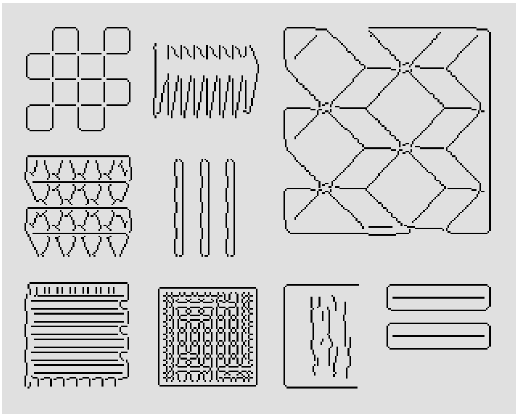
Le résultat du système coopératif obtient un score comparable (17 % de pixels non trouvés,



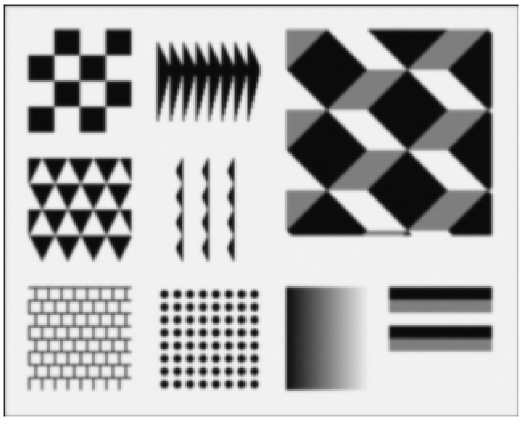
(a) Original.



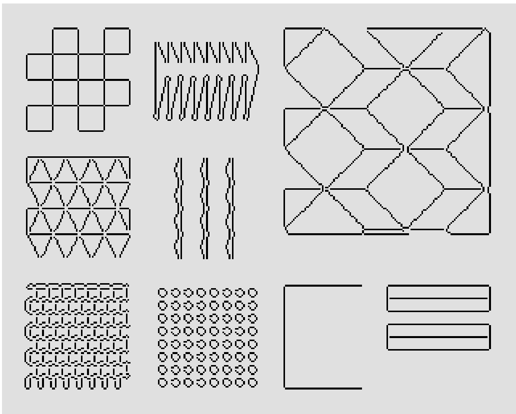
(b) Lissage ($\alpha = 0.60$).



(c) Contours ($\alpha = 0.60$).

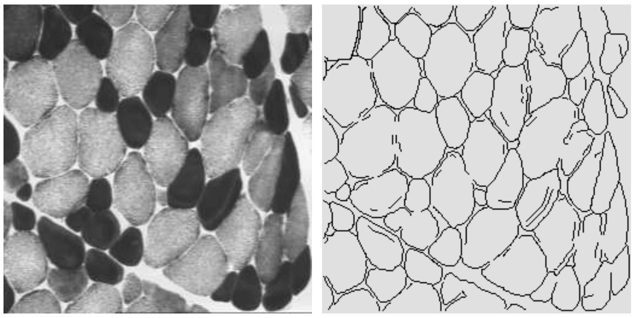


(d) Lissage ($\alpha = 2.44$).



(e) Contours ($\alpha = 2.44$).

FIG. 5.20: Le résultat obtenu par la méthode de Deriche sur l'image *fleck320.gif* avec deux valeurs différentes pour α. (a) Image originale en niveaux de gris. (b) Image lissée par l'algorithme de Deriche avec $\alpha = 0.60$. (c) Contours obtenus par Deriche avec $\alpha = 0.60$. (d) Image lissée par l'algorithme de Deriche avec $\alpha = 2.44$. (e) Contours obtenus par Deriche avec $\alpha = 2.44$.

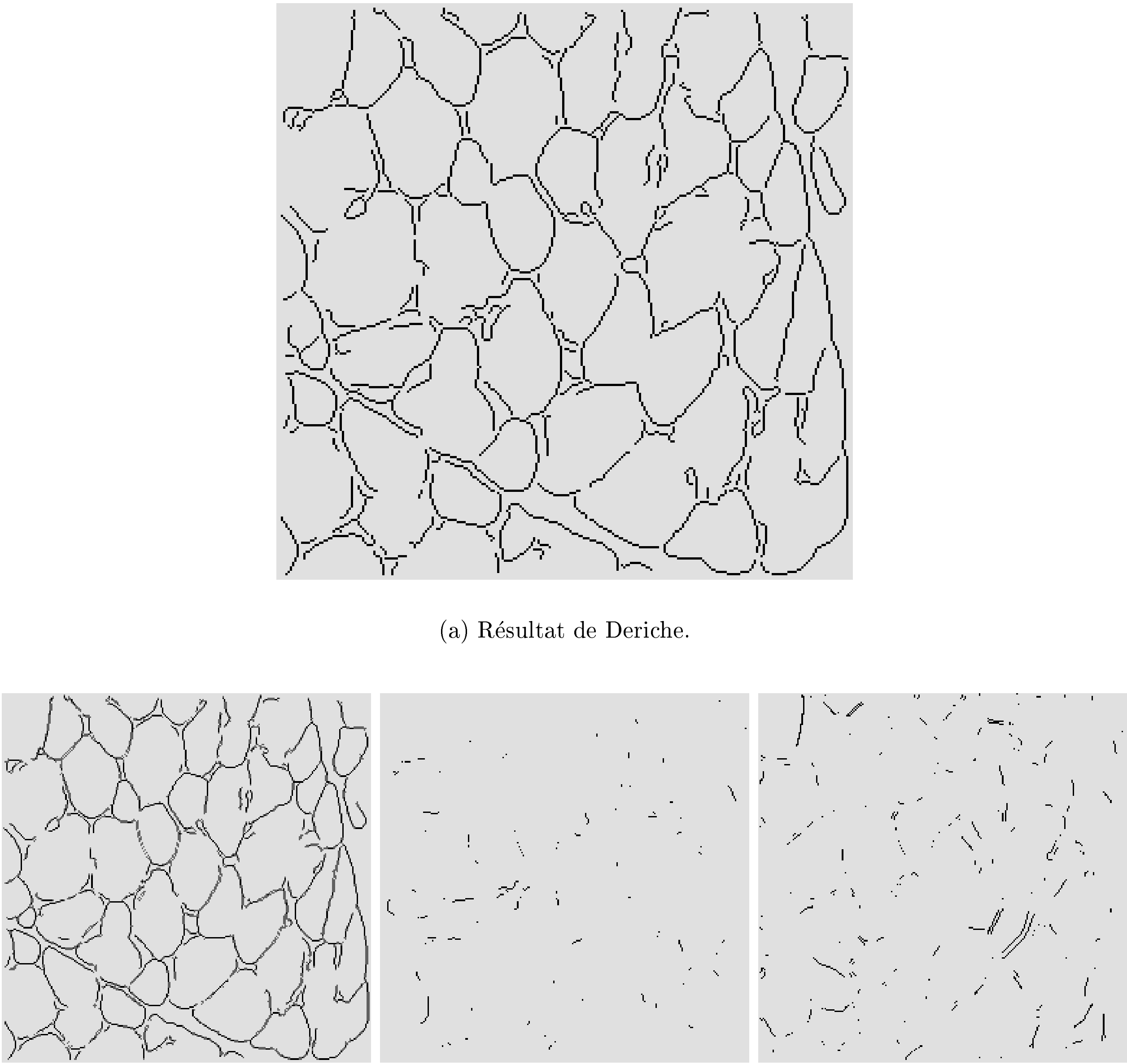


(a) Niveaux de gris. (b) Contours de référence.

FIG. 5.21: L'image en niveaux de gris *cellules256.gif* et la carte des contours de référence correspondante.

Nombre de pixels de l'image de référence	5004	
Nombre de pixels de l'image résultat	4077	
PIXELS COMMUNS		
Evaluation stricte (dist=1)	2299	
par rapport a l'image de référence		45.9 %
par rapport a l'image résultat		56.4 %
Evaluation large (dist=3)		
par rapport a l'image de référence	4097	81.9 %
par rapport a l'image résultat	3785	92.8 %
PIXELS NON TROUVES		
Evaluation stricte (dist=1)	2705	54.1 %
Evaluation large (dist=3)	907	18.1 %
PIXELS TROUVES A TORT		
Evaluation stricte (dist=1)	1778	43.6 %
Evaluation large (dist=3)	292	7.2 %
PIXELS COMMUNS (des 2 images), dist=3, (5583)		
Moyenne du gradient	23.3	
Variance du gradient	11.2	
PIXELS NON TROUVES (907)		
Moyenne du gradient	9.1	
Variance du gradient	6.5	
PIXELS TROUVES A TORT (292)		
Moyenne du gradient	11.5	
Variance du gradient	5.0	

TAB. 5.9: Comparaisons entre le résultat du détecteur de Deriche et la carte des contours de référence de l'image *cellules256.gif*.



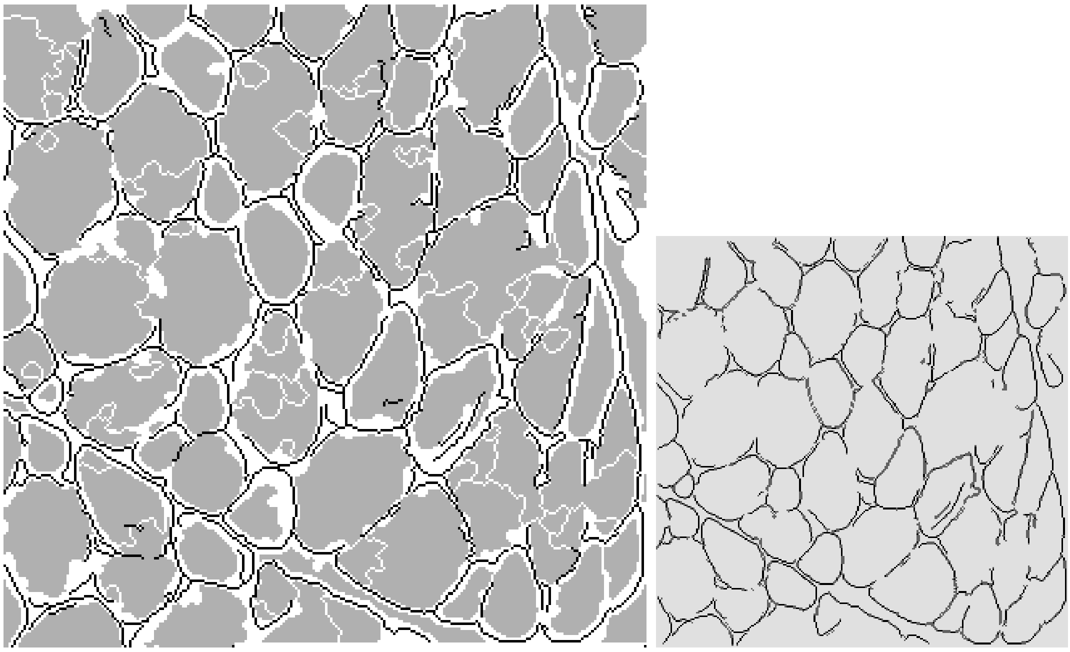
(a) Résultat de Deriche.

(b) Contours communs.

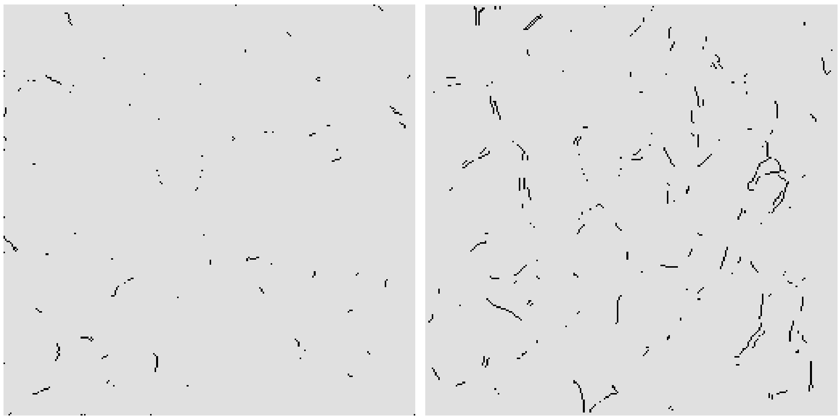
(c) Contours en trop.

(d) Contours manquants.

FIG. 5.22: Les résultats de la méthode de Deriche sur l'image *cellules256.gif* obtenus avec une recherche de paramètres optimaux, ici $\alpha = 1.16$, $sb = 16$ et $sh = 44$, respectivement pour le paramètre de lissage, et les seuils bas et haut de l'hystérésis.



(a) Résultat de la méthode coopérative. (b) Contours communs.



(c) Contours en trop. (d) Contours manquants.

FIG. 5.23: Les résultats du système de segmentation sur l'image *cellules256.gif*.

Nombre de pixels de l'image de référence	5004	
Nombre de pixels de l'image résultat	4148	
PIXELS COMMUNS		
Evaluation stricte (dist=1)	2411	
par rapport a l'image de référence		48.2 %
par rapport a l'image résultat		58.1 %
Evaluation large (dist=3)		
par rapport a l'image de référence	4132	82.6 %
par rapport a l'image résultat	3903	94.1 %
PIXELS NON TROUVES		
Evaluation stricte (dist=1)	2593	51.8 %
Evaluation large (dist=3)	872	17.4 %
PIXELS TROUVES A TORT		
Evaluation stricte (dist=1)	1737	41.9 %
Evaluation large (dist=3)	245	5.9 %
PIXELS COMMUNS (des 2 images), dist=3, (5624)		
Moyenne du gradient	23.9	
Variance du gradient	11.6	
PIXELS NON TROUVES (872)		
Moyenne du gradient	10.0	
Variance du gradient	6.3	
PIXELS TROUVES A TORT (245)		
Moyenne du gradient	10.8	
Variance du gradient	7.5	

TAB. 5.10: Comparaison entre le résultat de l'algorithme de segmentation et la carte des contours de référence de l'image *cellules256.gif*.

et 6% de pixels en trop), mais il ne saurait être réduit à ces valeurs. On peut visuellement noter que le résultat ne présente quasiment aucun faux contour en zone texturée, et par voie de conséquence, il est logiquement plus restrictif sur les contours entre des cellules texturées. Par contre, la plupart des contours entre les cellules noires sont ici présents. Cela peut s'évaluer numériquement en comparant la *moyenne du gradient* dans la catégorie des *pixels non trouvés* entre les deux tables 5.9 et 5.10. Ces contours entre les objets homogènes contribuent à la moyenne de cette catégorie, comme le prouve l'image 5.22(d), et ne se retrouvent plus dans l'image 5.23(d), pour laquelle la moyenne du gradient a augmenté d'environ 10 %.

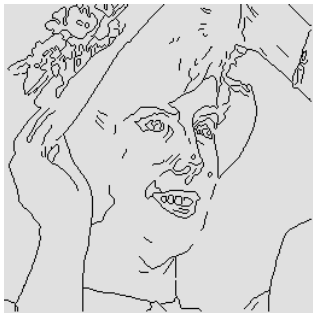
Comparaison avec la carte de référence - femme256

L'image utilisée ici est un classique du traitement d'image. Ses difficultés de segmentation sont les très petites structures à fort gradient que constituent les fleurs dans le chapeau, ainsi que le tracé des yeux et de la bouche. Des zones de dégradé sur son visage, ainsi que des ombres peuvent introduire des contours parasites. La limite entre son visage et le fond de l'image est à certains endroits indétectable. Enfin de manière générale, cette image est porteuse d'une forte sémantique, car un visage est quelque chose de très structuré, répondant à des codes visuels très stricts, auxquels un détecteur de contours ne pourra être sensible. Le résultat de la détection sera donc la plupart du temps particulièrement en deçà des espérances de son utilisateur.

L'image de la femme, et la carte de référence sont données en figure 5.24, le résultat du Deriche avec une recherche de paramètres optimaux en figure 5.25, des statistiques de détection du Deriche dans la table 5.11, le résultat du système coopératif en figure 5.26, ainsi que ses statistiques associées dans la table 5.12.

Le Deriche apparaît visuellement en légère sous-segmentation, du moins sur toute la partie du visage. Les pixels contours en trop sont regroupés dans les structures du chapeau, figure 5.25(c), pour lesquelles une forte moyenne du gradient est à signaler, comme l'indique l'avant-dernière ligne de la table 5.11. Un manque de sensibilité l'empêche de détecter la transition entre son visage et le fond de l'image, même au niveau de son cou, en figure 5.25(d).

(a) Niveaux de gris.

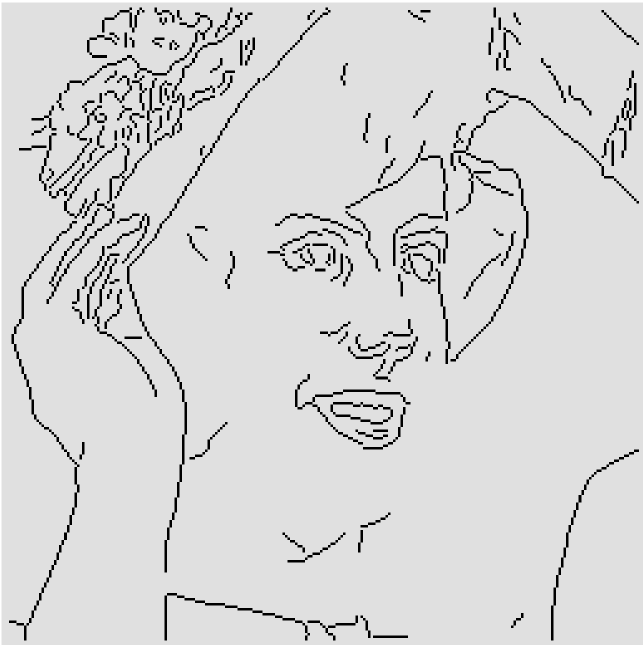


(b) Contours de référence.

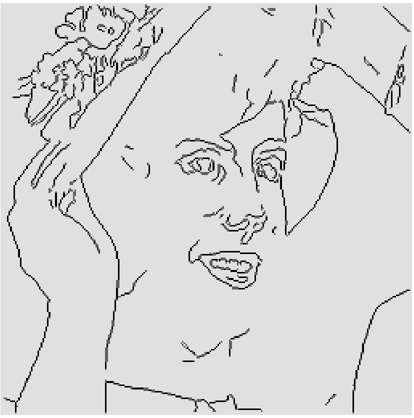
FIG. 5.24: L'image en niveaux de gris *femme256.gif* et la carte des contours de référence correspondante.

Nombre de pixels de l'image de référence	3339	
Nombre de pixels de l'image résultat	3038	
PIXELS COMMUNS		
Evaluation stricte (dist=1)	1861	
par rapport a l'image de référence		55.7 %
par rapport a l'image résultat		61.3 %
Evaluation large (dist=3)		
par rapport a l'image de référence	2840	85.1 %
par rapport a l'image résultat	2694	88.7 %
PIXELS NON TROUVES		
Evaluation stricte (dist=1)	1478	44.3 %
Evaluation large (dist=3)	499	14.9 %
PIXELS TROUVES A TORT		
Evaluation stricte (dist=1)	1177	38.7 %
Evaluation large (dist=3)	344	11.3 %
PIXELS COMMUNS (des 2 images), dist=3, (3673)		
Moyenne du gradient	20.5	
Variance du gradient	11.3	
PIXELS NON TROUVES (499)		
Moyenne du gradient	7.9	
Variance du gradient	6.2	
PIXELS TROUVES A TORT (344)		
Moyenne du gradient	17.9	
Variance du gradient	9.7	

TAB. 5.11: Comparaison entre le résultat du détecteur de Deriche et la carte des contours de référence de l'image *femme256.gif*.



(a) Résultat de Deriche.



(b) Contours communs.

(c) Contours en trop.

(d) Contours manquants.

FIG. 5.25: Les résultats de la méthode de Deriche sur l'image *femme256.gif* obtenus avec une recherche de paramètres optimaux, ici $\alpha = 1.67$, $sb = 25$ et $sh = 33$, respectivement pour le paramètre de lissage, et les seuils bas et haut de l'hystérésis.

(a) Résultat de la méthode coopérative.

(b) Contours communs. (c) Contours en trop. (d) Contours manquants.

Fig. 5.26: Les résultats du système de segmentation sur l'image *femme256.gif*.

Nombre de pixels de l'image de référence	3339	
Nombre de pixels de l'image résultat	3368	
PIXELS COMMUNS		
Evaluation stricte (dist=1)	1836	
par rapport a l'image de référence		55.0 %
par rapport a l'image résultat		54.5 %
Evaluation large (dist=3)		
par rapport a l'image de référence	2703	81.0 %
par rapport a l'image résultat	2730	81.1 %
PIXELS NON TROUVES		
Evaluation stricte (dist=1)	1503	45.0 %
Evaluation large (dist=3)	636	19.0 %
PIXELS TROUVES A TORT		
Evaluation stricte (dist=1)	1532	45.5 %
Evaluation large (dist=3)	638	18.9 %
PIXELS COMMUNS (des 2 images), dist=3, (3597)		
Moyenne du gradient	20.7	
Variance du gradient	11.8	
PIXELS NON TROUVES (636)		
Moyenne du gradient	9.1	
Variance du gradient	4.9	
PIXELS TROUVES A TORT (638)		
Moyenne du gradient	12.8	
Variance du gradient	9.8	

TAB. 5.12: Comparaison entre le résultat de l'algorithme de segmentation et la carte des contours de référence de l'image *femme256.gif*.

Notre résultat présente des défauts assez similaires, voir la figure 5.26. La structure des narines a complètement été ignorée. Les pixels contours trouvés *à tort* sont environ deux fois plus nombreux par rapport au Deriche (638 contre 344), mais leur gradient moyen est nettement inférieur (12.8 contre 17.9), ce qui confirme la caractéristique de sous-détection du Deriche sur cette image. La qualité de notre segmentation est ici inférieure à celle du Deriche.

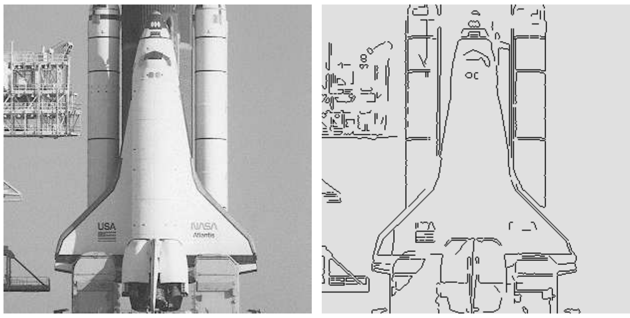
Comparaison avec la carte de référence - shuttle310

L'image de la figure 5.27(a) est assez difficile à segmenter car elle présente un bruit important. Le dos de la navette est en dégradé très net par rapport à la source d'éclairage qui est sur le coté. Les structures tubulaires situées à gauche sont clairement contrastées, mais les éléments qui les composent sont petits, et de courte taille, si bien qu'il est difficile de dégager des contours clairs dans cette zone, voir la carte de référence dans l'image 5.27(a).

Le résultat de Deriche sur cette image, voir la figure 5.28(a), est visuellement en sur-segmentation. La table 5.13 indique que si seulement 9.1 % des pixels de référence n'ont pas été trouvés, 22.8 % des pixels du résultats sont en trop. Ceux-ci se situent essentiellement dans la structure tubulaire de gauche, et leur moyenne de gradient est assez importante. Les paramètres optimaux contiennent pour cette image un facteur de lissage moins important que pour les précédentes images naturelles ($\alpha = 1.72$). Il peut se justifier par le besoin de ne pas trop perdre de détails dans la zone tubulaire de gauche.

Le résultat de l'algorithme coopératif présente un meilleur compromis entre la sur- et la sous-détection, respectivement 18.0 et 13.8 % des pixels dans la table 5.14. Des éléments de la structure tubulaire ont été segmentés avec difficulté, car parmi les pixels trouvés, une proportion comparable de pixels est évaluée *non trouvé* et *en trop*. La détection de l'ombre sur le réservoir droit de la navette, qui est une transition particulièrement large, presque assimilable à une zone de dégradé, apporte un nombre significatif de pixels *en trop*.

Il est intéressant de noter que le fond a été segmenté en un seul morceau (en fait une région de chaque coté de la navette), malgré son caractère très bruité.

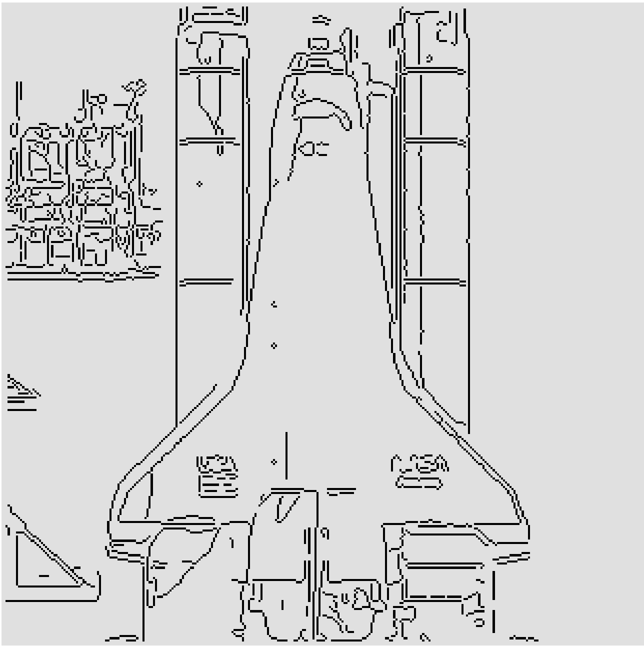


(a) Niveaux de gris. (b) Contours de référence.

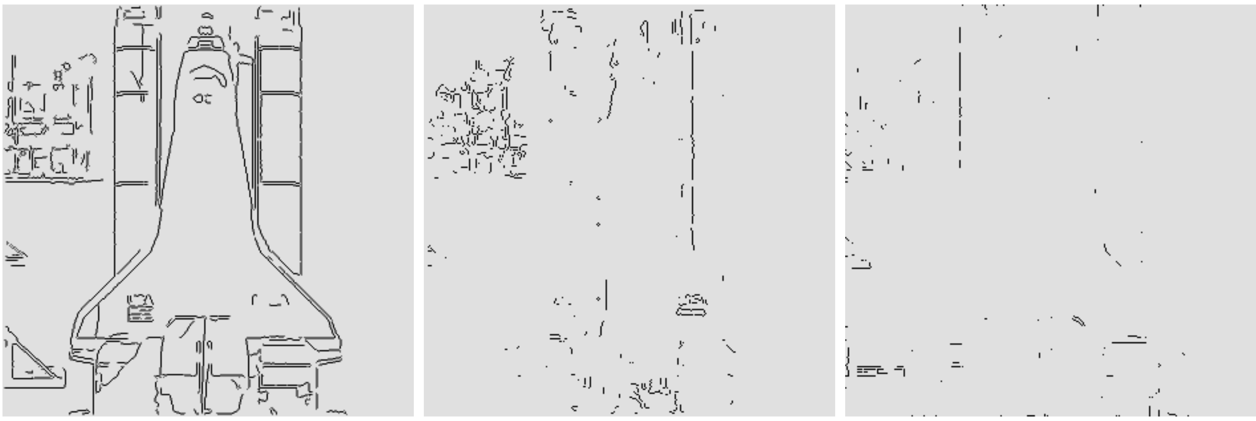
FIG. 5.27: L'image en niveaux de gris *shuttle310* et la carte des contours de référence correspondante.

Nombre de pixels de l'image de référence	5518	
Nombre de pixels de l'image résultat	6470	
PIXELS COMMUNS		
Evaluation stricte (dist=1)	3319	
par rapport a l'image de référence		60.1 %
par rapport a l'image résultat		51.3 %
Evaluation large (dist=3)		
par rapport a l'image de référence	5016	90.9 %
par rapport a l'image résultat	4992	77.2 %
PIXELS NON TROUVES		
Evaluation stricte (dist=1)	2199	39.9 %
Evaluation large (dist=3)	502	9.1 %
PIXELS TROUVES A TORT		
Evaluation stricte (dist=1)	3151	48.7 %
Evaluation large (dist=3)	1478	22.8 %
PIXELS COMMUNS (des 2 images), dist=3, (6689)		
Moyenne du gradient	28.6	
Variance du gradient	13.1	
PIXELS NON TROUVES (502)		
Moyenne du gradient	11.5	
Variance du gradient	8.4	
PIXELS TROUVES A TORT (1478)		
Moyenne du gradient	21.4	
Variance du gradient	10.3	

TAB. 5.13: Comparaison entre le résultat du détecteur de Deriche et la carte des contours de référence de l'image *shuttle310.gif*.

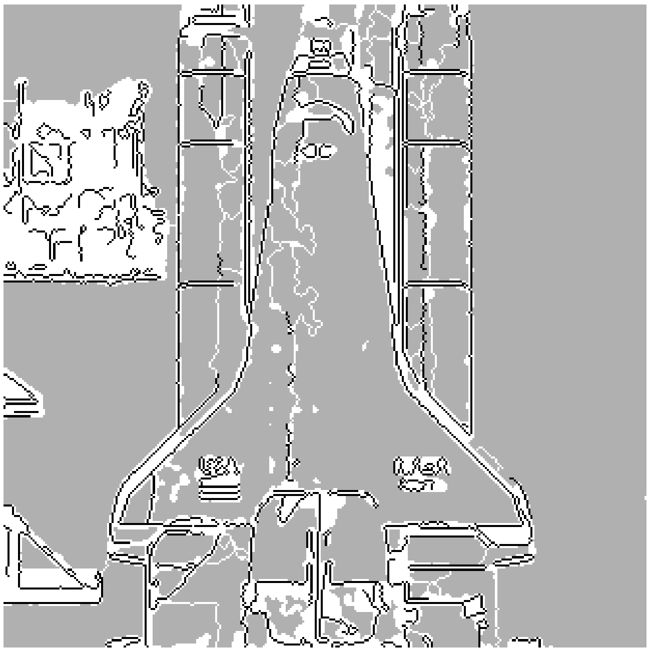


(a) Résultat de Deriche.



(b) Contours communs. (c) Contours en trop. (d) Contours manquants.

FIG. 5.28: Les résultats de la méthode de Deriche sur l'image *shuttle310.gif* obtenus avec une recherche de paramètres optimaux, ici $\alpha = 1.72$, $sb = 28$ et $sh = 32$, respectivement pour le paramètre de lissage, et les seuils bas et haut de l'hystérésis.



(a) Résultat de la méthode coopérative.

(b) Contours communs. (c) Contours en trop. (d) Contours manquants.

FIG. 5.29: Les résultats du système de segmentation sur l'image *shuttle310.gif*.

Nombre de pixels de l'image de référence	5518	
Nombre de pixels de l'image résultat	5752	
PIXELS COMMUNS		
Evaluation stricte (dist=1)	3081	
par rapport a l'image de référence		55.8 %
par rapport a l'image résultat		53.6 %
Evaluation large (dist=3)		
par rapport a l'image de référence	4757	86.2 %
par rapport a l'image résultat	4714	82.0 %
PIXELS NON TROUVES		
Evaluation stricte (dist=1)	2437	44.2 %
Evaluation large (dist=3)	761	13.8 %
PIXELS TROUVES A TORT		
Evaluation stricte (dist=1)	2671	46.4 %
Evaluation large (dist=3)	1038	18.0 %
PIXELS COMMUNS (des 2 images), dist=3, (6390)		
Moyenne du gradient	28.9	
Variance du gradient	13.2	
PIXELS NON TROUVES (761)		
Moyenne du gradient	16.1	
Variance du gradient	10.9	
PIXELS TROUVES A TORT (1038)		
Moyenne du gradient	20.2	
Variance du gradient	11.6	

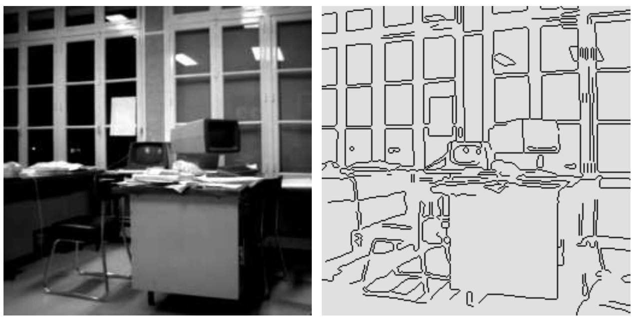
TAB. 5.14: Comparaison entre le résultat de l'algorithme de segmentation et la carte des contours de référence de l'image *shuttle310.gif*.

Comparaison avec la carte de référence - bureau256

L'image de la figure 5.30(a) est elle-aussi très classique dans le domaine du traitement d'image. Il s'agit d'une image très peu bruitée [31], présentant de nombreuses structures géométriques, aux contours rectilignes. Des ombres viennent ajouter des contours aux transitions très larges. De nombreux reflets existent dans cette image, sur les vitres, et sur l'écran d'un des terminaux, mais ils sont bien contrasté, et ne posent pas de problème de détection [32]. Une zone particulièrement sombre en bas à droite de l'image masque partiellement la chaise qui se trouve à cet endroit. Seuls ses barreaux métalliques présentent un faible caractère lumineux.

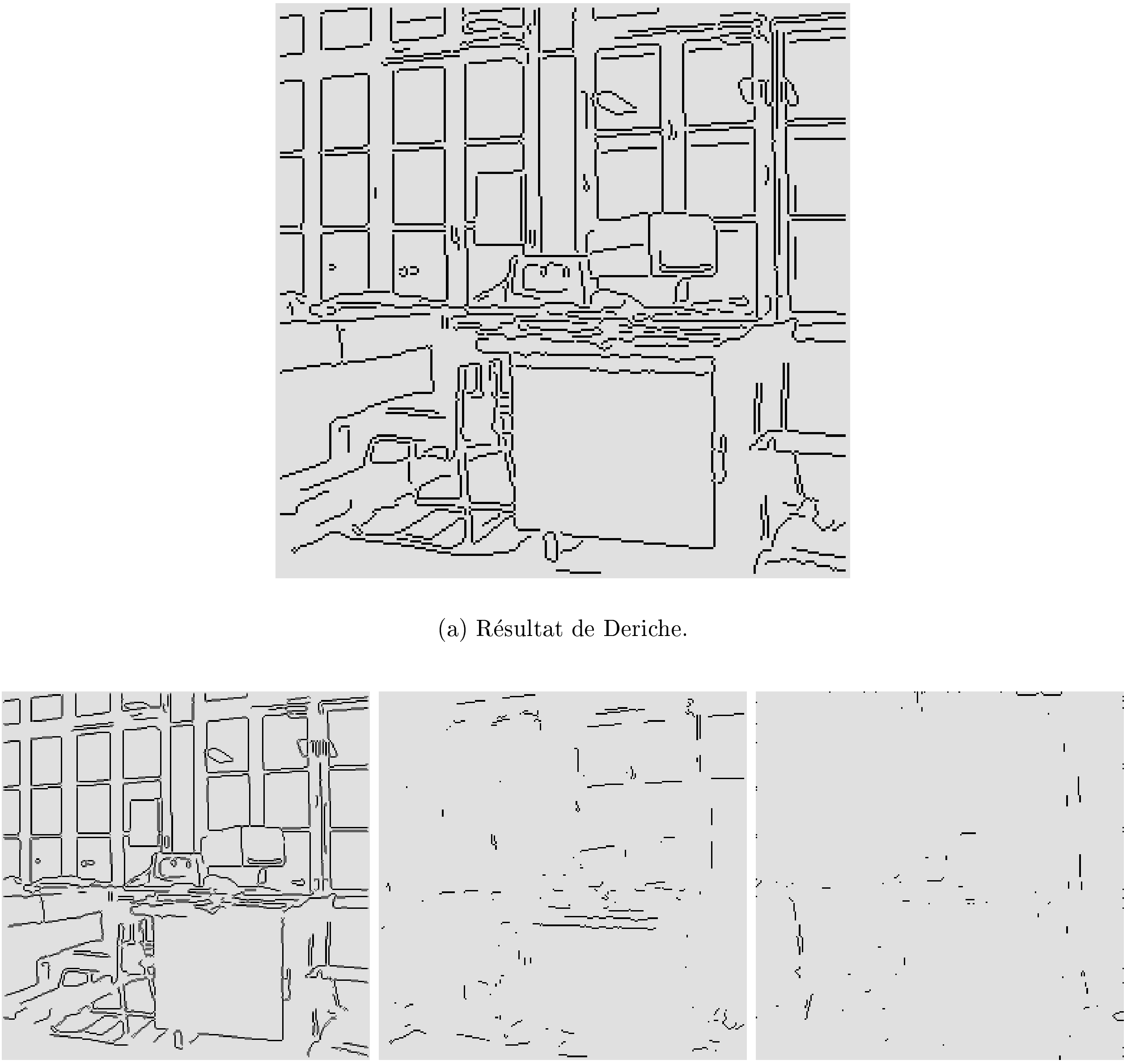
[31] Ce caractère très peu bruité se traduira par la suite par la valeur élevée de α des paramètres optimaux de la méthode de Deriche. $\alpha = 2.22$ correspond à un lissage faible, rendu ici possible par l'aspect peu bruité de l'image.

[32] Leur interprétation serait d'une toute autre difficulté.



(a) (b)

FIG. 5.30: L'image en niveaux de gris *bureau256.gif* et la carte des contours de référence correspondante.



(a) Résultat de Deriche.

(b) Contours communs.

(c) Contours en trop.

(d) Contours manquants.

FIG. 5.31: Les résultats de la méthode de Deriche sur l'image *bureau256.gif* obtenus avec une recherche de paramètres optimaux, ici $\alpha = 2.22$, $sb = 9$ et $sh = 25$, respectivement pour le paramètre de lissage, et les seuils bas et haut de l'hystérésis.

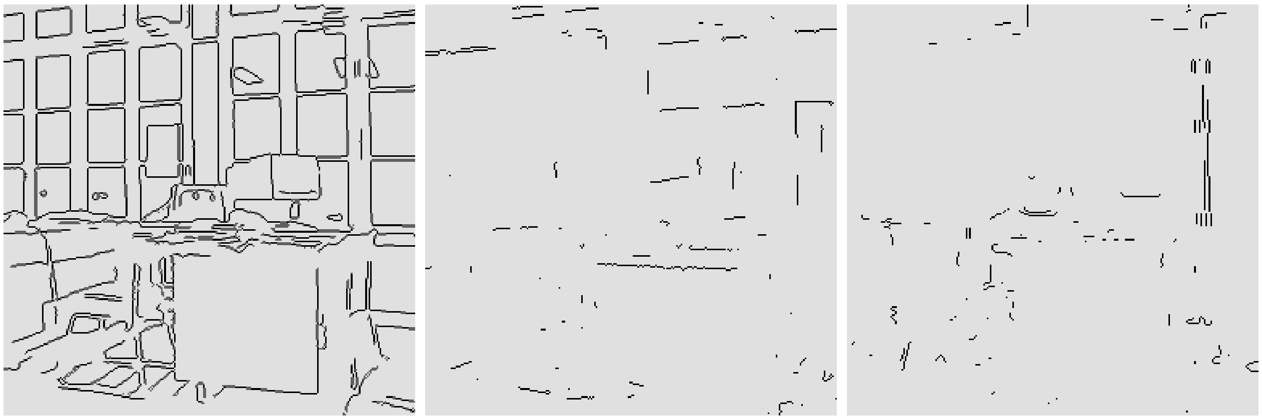
Nombre de pixels de l'image de référence	6319	
Nombre de pixels de l'image résultat	6936	
PIXELS COMMUNS		
Evaluation stricte (dist=1)	4291	
par rapport a l'image de référence		67.9 %
par rapport a l'image résultat		61.9 %
Evaluation large (dist=3)		
par rapport a l'image de référence	5961	94.3 %
par rapport a l'image résultat	5886	84.9 %
PIXELS NON TROUVES		
Evaluation stricte (dist=1)	2028	32.1 %
Evaluation large (dist=3)	358	5.7 %
PIXELS TROUVES A TORT		
Evaluation stricte (dist=1)	2645	38.1 %
Evaluation large (dist=3)	1050	15.1 %
PIXELS COMMUNS (des 2 images), dist=3, (7556)		
Moyenne du gradient	26.3	
Variance du gradient	16.6	
PIXELS NON TROUVES (358)		
Moyenne du gradient	8.0	
Variance du gradient	7.4	
PIXELS TROUVES A TORT (1050)		
Moyenne du gradient	9.4	
Variance du gradient	4.7	

TAB. 5.15: Comparaison entre le résultat du détecteur de Deriche et la carte des contours de référence de l'image *bureau256.gif*.

Nombre de pixels de l'image de référence	6319	
Nombre de pixels de l'image résultat	6406	
PIXELS COMMUNS		
Evaluation stricte (dist=1)	4106	
par rapport a l'image de référence		65.0 %
par rapport a l'image résultat		64.1 %
Evaluation large (dist=3)		
par rapport a l'image de référence	5627	89.0 %
par rapport a l'image résultat	5632	87.9 %
PIXELS NON TROUVES		
Evaluation stricte (dist=1)	2213	35.0 %
Evaluation large (dist=3)	692	11.0 %
PIXELS TROUVES A TORT		
Evaluation stricte (dist=1)	2300	35.9 %
Evaluation large (dist=3)	774	12.1 %
PIXELS COMMUNS (des 2 images), dist=3, (7153)		
Moyenne du gradient	27.5	
Variance du gradient	16.5	
PIXELS NON TROUVES (692)		
Moyenne du gradient	9.1	
Variance du gradient	6.8	
PIXELS TROUVES A TORT (774)		
Moyenne du gradient	9.4	
Variance du gradient	5.5	

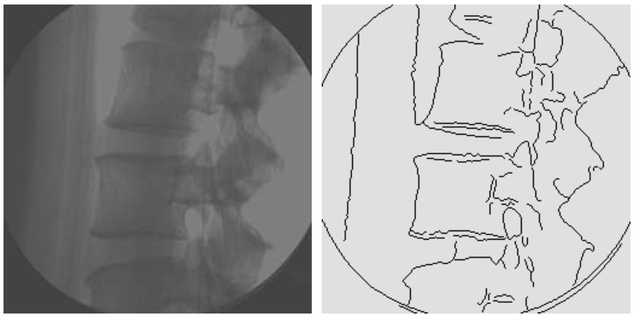
TAB. 5.16: Comparaison entre le résultat de l'algorithme de segmentation et la carte des contours de référence de l'image *bureau256.gif*.

(a) Résultat de la méthode coopérative.



(b) Contours communs. (c) Contours en trop. (d) Contours manquants.

FIG. 5.32: Les résultats du système de segmentation sur l'image *bureau256.gif*.



(a) (b)

FIG. 5.33: L'image en niveaux de gris *vertebre256.gif* et la carte des contours de référence correspondante.

L'algorithme de Deriche (figure 5.31) ainsi que de l'algorithme coopératif (figure 5.32) ont détecté de nombreuses transitions en trop correspondant à des ombres portées. Ces transitions ne figurent en effet pas sur la carte des contours de référence, car leur largeur rend leur identification incertaine, et l'information à faire figurer à ces endroits s'apparente davantage à la notion de dégradé, qu'à celle de point de contour [33].

Le résultat de Deriche présente, d'après le tableau 5.15, une sur-segmentation importante, tandis que le résultat du système coopératif présente un nombre comparable de pixels *non trouvés* et *trouvés en trop*.

Comparaison avec la carte de référence - vertebre256

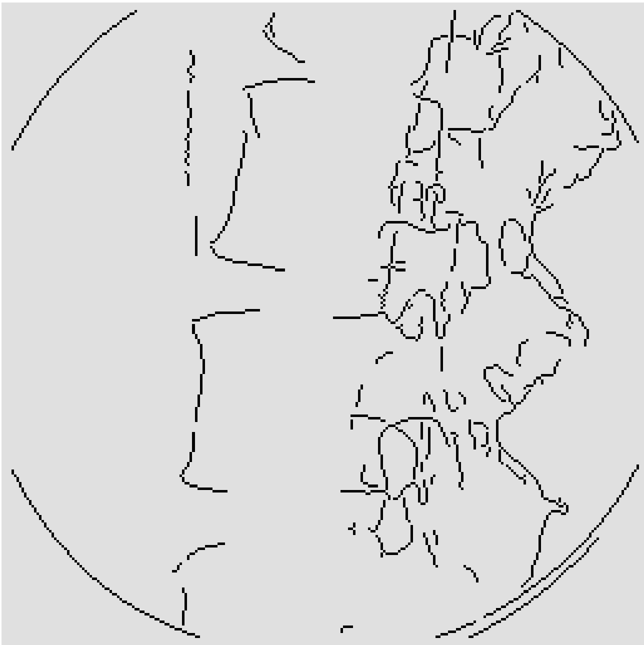
Le dernier exemple est particulièrement difficile. L'image est une radiographie de colonne vertébrale, dont la caractéristique est d'avoir une dynamique restreinte à 68 niveaux de gris, soit environ 4 fois moins que la dynamique disponible sur 8 bits. L'image n'est par ailleurs pratiquement pas bruitée. Elle dispose de grandes zones homogènes ayant exactement la même valeur de niveau de gris. Ces niveaux s'étendent entre 60 et 128, ce qui rend les transitions particulièrement visibles [34].

Les résultats des figures 5.34 et 5.35 sont assez décevants, car de nombreux contours sont ignorés par les deux algorithmes. Le système coopératif se situe clairement en sous-segmentation, tandis que le résultat de Deriche fournit de nombreux contours en trop (figure 5.34(c)) dans toute la partie droite de l'image. Il est difficile pourtant d'accuser les deux méthodes de manquer de sensibilité, car les pixels non trouvés disposent d'une moyenne du gradient inférieure à 2 selon les tables 5.17 et 5.18.

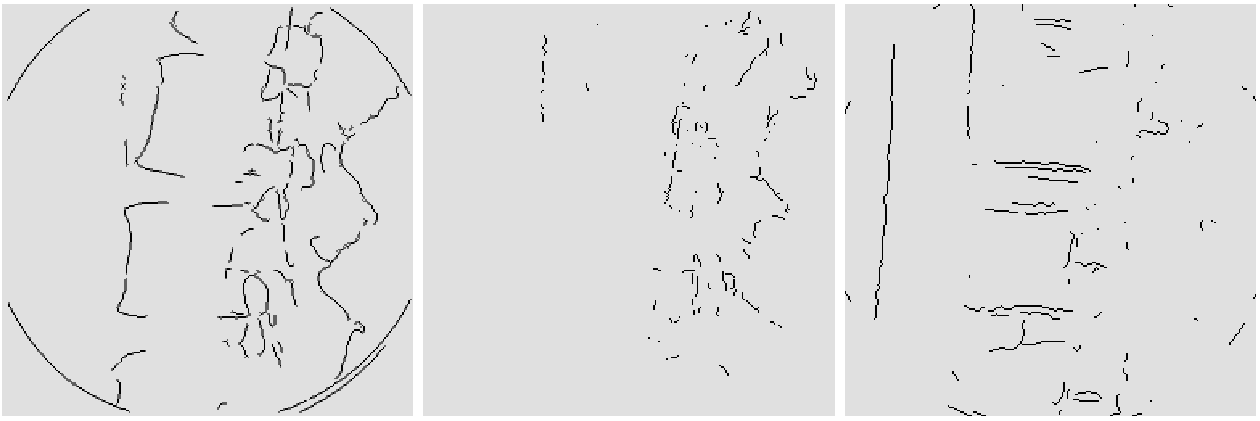
Pour une image naturelle, un gradient de 2 n'est en effet pas significatif, car il peut être imputé au bruitage du système d'acquisition de l'image. Cette image rend un tel gradient significatif, du fait de sa faible dynamique, et de son absence quasi totale de bruit.

[33] Lorsqu'une transition fait plus de 5 pixels de large, le placement du pixel censé marquer le contour devient incertain. Il convient cependant idéalement de le localiser au milieu de la transition.

[34] L'oeil est en effet plus sensible aux faibles transitions lorsqu'elles se situent dans une plage centrale des niveaux de gris de l'image.



(a) Résultat de Deriche.



(b) Contours communs. (c) Contours en trop. (d) Contours manquants.

FIG. 5.34: Les résultats de la méthode de Deriche sur l'image *vertebre256.gif* obtenus avec une recherche de paramètres optimaux, ici $\alpha = 2.18$, $sb = 8$ et $sh = 11$, respectivement pour le paramètre de lissage, et les seuils bas et haut de l'hystérésis.

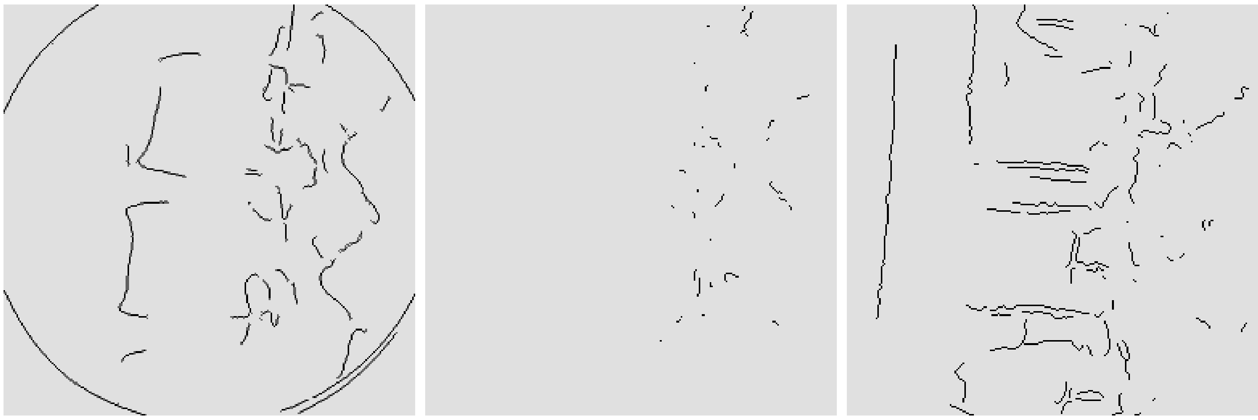
Nombre de pixels de l'image de référence	2674	
Nombre de pixels de l'image résultat	2149	
PIXELS COMMUNS		
Evaluation stricte (dist=1)	982	
par rapport a l'image de référence		36.7 %
par rapport a l'image résultat		45.7 %
Evaluation large (dist=3)		
par rapport a l'image de référence	1624	60.7 %
par rapport a l'image résultat	1615	75.2 %
PIXELS NON TROUVES		
Evaluation stricte (dist=1)	1692	63.3 %
Evaluation large (dist=3)	1050	39.3 %
PIXELS TROUVES A TORT		
Evaluation stricte (dist=1)	1167	54.3 %
Evaluation large (dist=3)	534	24.8 %
PIXELS COMMUNS (des 2 images), dist=3, (2257)		
Moyenne du gradient	4.5	
Variance du gradient	3.2	
PIXELS NON TROUVES (1050)		
Moyenne du gradient	1.2	
Variance du gradient	1.5	
PIXELS TROUVES A TORT (534)		
Moyenne du gradient	3.2	
Variance du gradient	1.1	

TAB. 5.17: Comparaison entre le résultat du détecteur de Deriche et la carte des contours de référence de l'image *vertebre256.gif*.

Nombre de pixels de l'image de référence	2674	
Nombre de pixels de l'image résultat	1385	
PIXELS COMMUNS		
Evaluation stricte (dist=1)	770	
par rapport a l'image de référence		28.8 %
par rapport a l'image résultat		55.6 %
Evaluation large (dist=3)		
par rapport a l'image de référence	1261	47.2 %
par rapport a l'image résultat	1240	89.5 %
PIXELS NON TROUVES		
Evaluation stricte (dist=1)	1904	71.2 %
Evaluation large (dist=3)	1413	52.8 %
PIXELS TROUVES A TORT		
Evaluation stricte (dist=1)	615	44.4 %
Evaluation large (dist=3)	145	10.5 %
PIXELS COMMUNS (des 2 images), dist=3, (1731)		
Moyenne du gradient	5.0	
Variance du gradient	3.6	
PIXELS NON TROUVES (1413)		
Moyenne du gradient	1.6	
Variance du gradient	1.2	
PIXELS TROUVES A TORT (145)		
Moyenne du gradient	3.7	
Variance du gradient	1.3	

TAB. 5.18: Comparaison entre le résultat de l'algorithme de segmentation et la carte des contours de référence de l'image *vertebre256.gif*.

(a) Résultat de la méthode coopérative.



(b) Contours communs. (c) Contours en trop. (d) Contours manquants.

FIG. 5.35: Les résultats du système de segmentation sur l'image *vertebre256.gif*.

Seuil Grad=$f(\sigma_1,\sigma 2)$					
	2	5	8	10	plus
2	3	3	3	3	3
5	3	3	4	4	5
8	3	4	6	9	10
10	3	4	9	13	15
plus	3	5	10	15	20

Seuil écart-type=f(σ_r,Etape de relaxation)					
	1	2	4	6	plus
1	0.2	0.4	0.5	0.7	1.2
2	0.3	0.6	0.7	0.9	1.4
3	0.3	0.7	0.8	1.0	1.6
4	0.4	0.8	1.0	1.3	2.0
5	0.4	0.8	1.3	1.6	3.0
6	0.5	0.9	1.6	2.0	3.5
7	0.5	0.9	1.8	2.5	4.0
8	0.5	1.0	2.0	3.0	4.5

TAB. 5.19: Seuils des fonctions d'évaluation pour l'image *fleck320.gif*.

Seuil Grad=$f(\sigma_1,\sigma 2)$					
	2	5	8	10	plus
2	9	11	11	11	15
5	11	12	15	15	19
8	11	15	20	20	22
10	11	15	25	25	28
plus	15	19	28	28	32

Seuil écart-type=f(σ_r,Etape de relaxation)						
	1	2	4	6	10	plus
1	0.2	0.4	0.5	0.7	1.2	1.3
2	0.3	0.6	0.7	0.9	1.4	1.5
3	0.5	0.8	0.9	1.1	1.8	2.0
4	0.7	1.0	2.0	3.0	4.5	5.0

TAB. 5.20: Seuils des fonctions d'évaluation pour l'image *bureau256.gif*.

Synthèse

La table 5.25 donne les éléments de la configuration du système coopératif qui ont été ajustés au cours des six expérimentations précédentes. Les tables 5.19, 5.20, 5.21, 5.22, 5.23 et 5.24 fournissent la table des seuils utilisés pour les fonctions d'évaluation. La table 5.26 synthétise les résultats de la comparaison avec la carte de référence, la figure 5.36 en donne une représentation graphique. Enfin, la table 5.27 apporte une synthèse des résultats précédents sous la forme de matrices de confusion.

Conclusions

Les six expérimentations précédentes permettent de conclure en la justesse de notre approche sur des types d'images très variés, issus de domaines différents (images biomédicales, naturelles, de synthèse). Les résultats fournis, en les restreignant à la seule information concernant les contours, est généralement d'une qualité comparable à celle du détecteur de Deriche, pour lequel une recherche des paramètres optimaux a été faite.

5.3.3 L'adaptation

Comparaison avec un Deriche [Der87]

Cette partie de l'étude a pour but de situer le système proposé par rapport à des approches classiques en détection de contours, telles que le détecteur de Deriche par exemple. Dans le paragraphe 5.3.2, le Deriche et le système coopératif étaient tout les deux comparés à une

Seuil Grad=$f(\sigma_1,\sigma 2)$					
	2	5	8	10	plus
2	3	5	7	9	11
5	5	8	12	14	19
8	7	12	15	17	20
10	9	14	17	20	25
plus	11	19	20	25	28

Seuil écart-type=f(σ_r,Etape de relaxation)						
	1	2	4	6	10	plus
1	0.2	0.4	0.5	0.7	1.2	1.3
2	0.3	0.6	0.7	0.9	1.4	1.5
3	0.5	0.8	0.9	1.1	1.8	2.0
4	0.7	1.0	2.0	3.0	4.5	5.0

TAB. 5.21: Seuils des fonctions d'évaluation pour l'image *cellules256.gif*.

Seuil Grad= $f(\sigma_1, \sigma 2)$					
	2	5	8	10	plus
2	9	11	11	11	15
5	11	12	15	15	19
8	11	15	18	20	22
10	11	15	20	25	28
plus	15	19	22	28	32

Seuil écart-type=f(σ_r ,Etape de relaxation)						
	1	2	4	6	10	plus
1	0.2	0.4	0.5	0.7	1.2	1.3
2	0.3	0.6	0.7	0.9	1.4	1.5
3	0.5	0.8	0.9	1.1	1.8	2.0
4	0.7	1.0	2.0	3.0	4.5	5.0

TAB. 5.22: Seuils des fonctions d'évaluation pour l'image *shuttle310.gif*.

Seuil Grad= $f(\sigma_1, \sigma 2)$			
	2	5	plus
2	7	9	11
5	9	11	13
plus	11	13	13

Seuil écart-type=f(σ_r ,Etape de relaxation)						
	1	2	4	6	10	plus
1	0.2	0.4	0.5	0.7	1.2	1.3
2	0.3	0.6	0.7	0.9	1.4	1.5
3	0.5	0.8	0.9	1.1	1.8	2.0
4	0.7	1.0	2.0	3.0	4.5	5.0

TAB. 5.23: Seuils des fonctions d'évaluation pour l'image *femme256.gif*.

Seuil Grad= $f(\sigma_1, \sigma 2)$					
	2	5	8	10	plus
2	3	5	9	10	15
5	5	6	10	12	16
8	9	10	13	14	18
10	10	12	14	16	20
plus	15	16	18	20	24

Seuil écart-type=f(σ_r ,Etape de relaxation)			
	1	2	plus
1	0.2	0.4	0.5
2	0.3	0.5	0.6
3	0.4	0.6	0.7
4	0.5	0.8	0.9

TAB. 5.24: Seuils des fonctions d'évaluation pour l'image *vertebre256.gif*.

<table>
<tr><th></th><th>FLECK</th><th>BUREAU</th><th>CELLULES</th><th>SHUTTLE</th><th>FEMME</th><th>VERTEBRE</th></tr>
<tr><td>nb germes contour</td><td>0</td><td colspan="3">64</td><td>512</td><td>1</td></tr>
<tr><td>nb germes région</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td></tr>
<tr><td>nb process maxi</td><td>20000</td><td colspan="3">5000</td><td>20000</td><td>5000</td></tr>
<tr><td>seuil grad statique</td><td colspan="4">30</td><td>21</td><td>4</td></tr>
<tr><td>nb classes d'écart-type pour seuil contour</td><td colspan="4">2/5/8/10</td><td>2/5</td><td>1/2/4/6</td></tr>
<tr><td>nb classes d'écart-type pour seuil région</td><td>1/2/4/6</td><td colspan="4">1/2/4/6/10</td><td>1/2</td></tr>
<tr><td>échantillonage frontière pour focus</td><td>10 pix</td><td colspan="5">5 pix</td></tr>
<tr><td>taille fenêtre focus</td><td>7</td><td colspan="2">5</td><td colspan="3">7</td></tr>
<tr><td>nb étapes de relaxation pour région</td><td>8</td><td colspan="5">4</td></tr>
<tr><td>quantum pixels région</td><td>1000</td><td colspan="5">500</td></tr>
<tr><td>quantum pixels contour</td><td>200</td><td colspan="5">500</td></tr>
<tr><td>éval contour norme grad/max local</td><td>50/50</td><td colspan="4">60/40</td><td>90/10</td></tr>
<tr><td>éval région homogénéité/compacité</td><td>80/20</td><td colspan="5">40/60</td></tr>
</table>

TAB. 5.25: Configuration du système.

	FLECK320 (Synthèse)				
	Strict		Large		
	NT	TT	NT	TT	(NT+TT)/2
Deriche	27.7 %	23.5 %	1.8 %	0.2 %	1.00
Coopératif	25.7 %	23.8 %	0.8 %	0.1 %	0.45
	BUREAU256				
	Strict		Large		
	NT	TT	NT	TT	(NT+TT)/2
Deriche	32.1 %	38.1 %	5.7 %	15.1 %	10.4
Coopératif	35.0 %	35.9 %	11.0 %	12.1 %	11.55
	CELLULES256				
	Strict		Large		
	NT	TT	NT	TT	(NT+TT)/2
Deriche	54.1 %	43.6 %	18.1 %	7.2 %	12.65
Coopératif	51.8 %	41.9 %	17.4 %	5.9 %	11.65
	SHUTTLE310				
	Strict		Large		
	NT	TT	NT	TT	(NT+TT)/2
Deriche	39.9 %	48.7 %	9.1 %	22.8 %	15.95
Coopératif	44.2 %	46.4 %	13.8 %	18.0 %	15.9
	FEMME256				
	Strict		Large		
	NT	TT	NT	TT	(NT+TT)/2
Deriche	44.3 %	38.7 %	14.9 %	11.3 %	13.1
Coopératif	45.0 %	45.5 %	19.0 %	18.9 %	18.95
	VERTEBRE256				
	Strict		Large		
	NT	TT	NT	TT	(NT+TT)/2
Deriche	63.3 %	54.3 %	39.3 %	24.8 %	32.05
Coopératif	71.2 %	44.4 %	52.8 %	10.5 %	31.65

TAB. 5.26: Synthèse des proportions de pixels *non-trouvés* (NT) et *trouvés à tort* (TT) dans les précédentes expérimentations. $(TT + NT)/2$ exprime une mesure de qualité, résultat de la minimisation conjointe de TT et de NT.

	FLECK320 (Synthèse)			
	Deriche		Coopératif	
	Der C	Der NC	Coop C	Coop NC
Ref C	20077 (24.5 %)	877 (1.1 %)	20198 (24.7 %)	756 (0.9 %)
Ref NC	1239 (1.5 %)	59727 (72.9 %)	1113 (1.4 %)	59853 (73.1 %)
	BUREAU256			
	Deriche		Coopératif	
	Der C	Der NC	Coop C	Coop NC
	C	NC	C	NC
Ref C	17695 (27.0 %)	2247 (3.4 %)	16795 (25.6 %)	3147 (4.8 %)
Ref NC	4559 (7.0 %)	41035 (62.6 %)	3671 (5.6 %)	41923 (64.0 %)
	CELLULES256			
	Deriche		Coopératif	
	Der C	Der NC	Coop C	Coop NC
Ref C	13208 (20.2 %)	4558 (7.0 %)	13247 (20.2 %)	4519 (6.9 %)
Ref NC	2251 (3.4 %)	45519 (69.5 %)	1984 (3.0 %)	45786 (69.9 %)
	SHUTTLE310			
	Deriche		Coopératif	
	Der C	Der NC	Coop C	Coop NC
Ref C	14929 (15.5 %)	2600 (2.7 %)	14056 (14.6 %)	3473 (3.6 %)
Ref NC	6162 (6.4 %)	72409 (75.3 %)	4725 (4.9 %)	73846 (76.8 %)
	FEMME256			
	Deriche		Coopératif	
	Der C	Der NC	Coop C	Coop NC
Ref C	9889 (15.1 %)	2751 (4.2 %)	9556 (14.6 %)	3094 (4.7 %)
Ref NC	1983 (3.0 %)	50913 (77.7 %)	3110 (4.7 %)	49786 (76.0 %)
	VERTEBRE256			
	Deriche		Coopératif	
	Der C	Der NC	Coop C	Coop NC
Ref C	5793 (8.8 %)	4239 (6.5 %)	4497 (6.9 %)	5535 (8.4 %)
Ref NC	2598 (4.0 %)	52906 (80.7 %)	1015 (1.5 %)	54489 (83.1 %)

TAB. 5.27: Synthèse des expérimentations précédentes sous la forme de matrices de confusion. Afin de tenir compte de la distance d'incertitude (1 pixel), les deux cartes de pixels contours qui composent chaque composante d'une matrice de confusion ont subi une dilatation de un pixel également. La matrice de confusion distingue les pixels *contours* (C) des pixels *non-contours* (NC).

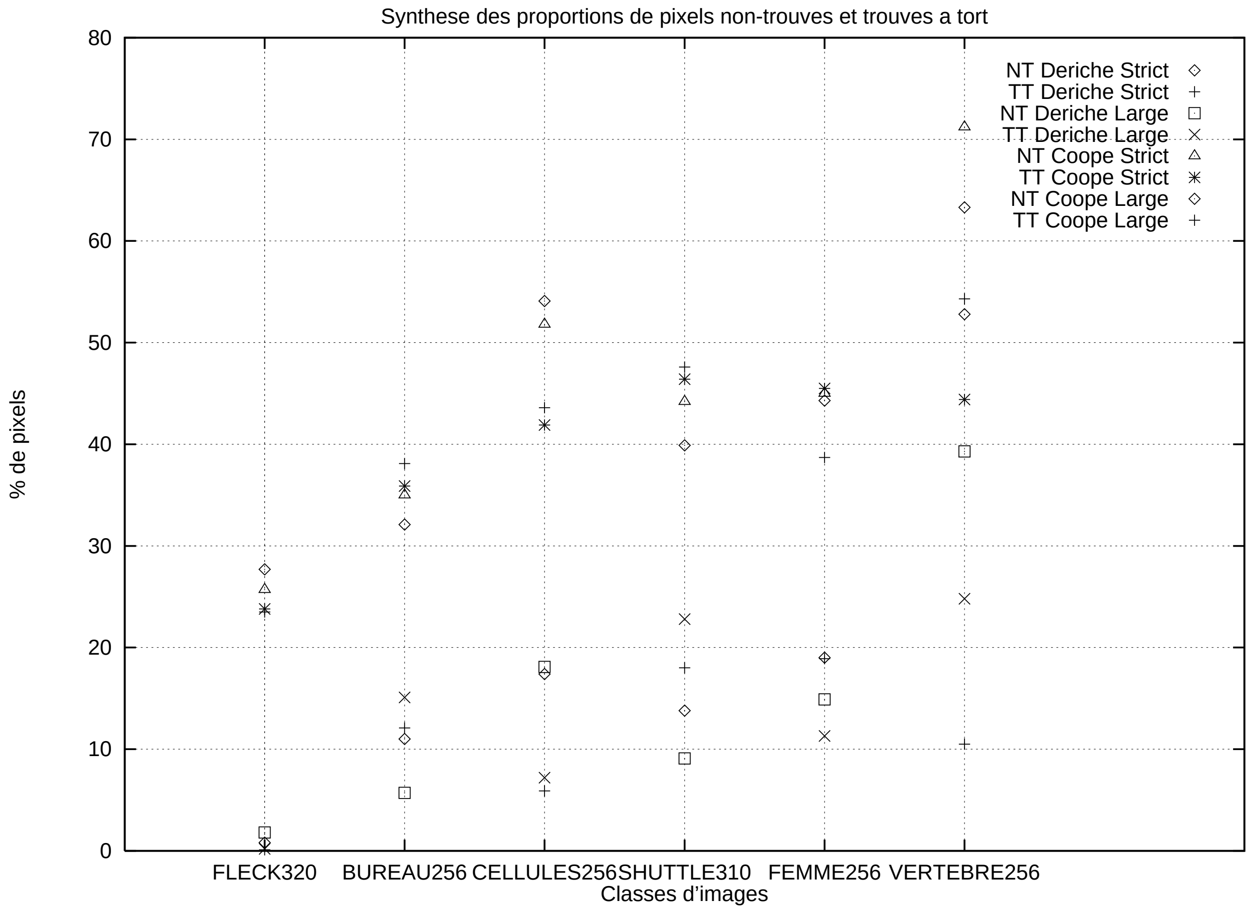


FIG. 5.36: Synthèse des expérimentations précédentes sous forme graphique. Les différentes images utilisées sont placées en abscisse. Les valeurs portées sur le graphique sont la proportion de pixels *non trouvés* (NT) et *trouvés à tort*, pour la méthode de Deriche ainsi que l'algorithme coopératif, dans le cas d'un mesure stricte (distance=0), et large (distance=1). Les images sont classées dans l'ordre inverse du succès des méthodes appliquées.

-1	0	1
-2	0	2
-1	0	1

-1	-2	-1
0	0	0
1	2	1

FIG. 5.37: Les masques de convolution utilisés pour le calcul du gradient.

même carte de référence. Ce paragraphe se propose de réaliser une comparaison directe entre les deux méthodes.

Dans ce but, et afin de situer les deux méthodes l'une par rapport à l'autre, l'expérimentation suivante va tenter de réaliser une *mutation* du système, afin de le placer dans les mêmes conditions de fonctionnement que le détecteur de Deriche. Cette mutation ressemble d'ailleurs davantage à une extension du Deriche, dans le sens où l'essentiel de l'algorithme de Deriche – le calcul de l'image du gradient – est incorporé au système, pour que celle-ci soit utilisée à la place de la carte de gradient par défaut.

A partir des mêmes données en entrée, la comparaison se fait alors entre le seuillage par hystérésis de Deriche, et la méthode de suivi de contour du système coopératif [35], dans l'objectif de montrer que les mécanismes d'interaction locale mis en jeu dans notre approche contribuent à obtenir une meilleure carte de contours que le simple seuillage par hystérésis de Deriche.

Le détecteur de contours de Deriche est une méthode élégante pour calculer une carte de gradient, en un nombre borné d'opérations, quel que soit le lissage appliqué sur l'image [36]. L'étape suivant le calcul du gradient est très classique. Elle consiste à marquer les pixels réalisant le maximum de la norme du gradient dans la direction du gradient, puis à effectuer un chaînage de ces points, en utilisant un couple de seuils [37], le seuil le plus haut étant appliqué sur le premier pixel du chaînage, et l'autre seuil, plus restrictif sur les autres pixels.

Notre approche peut aisément "intégrer" le calcul d'un gradient de Deriche en lieu et place de la carte de gradient actuellement pré-calculée au lancement du système, qui utilise deux masques directionnels donnés en figure 5.37. L'ensemble du système peut donc utiliser la même information que celle utilisée pour réaliser l'étape de chaînage de Deriche, tant pour les niveaux de gris que pour la norme et la direction du gradient.

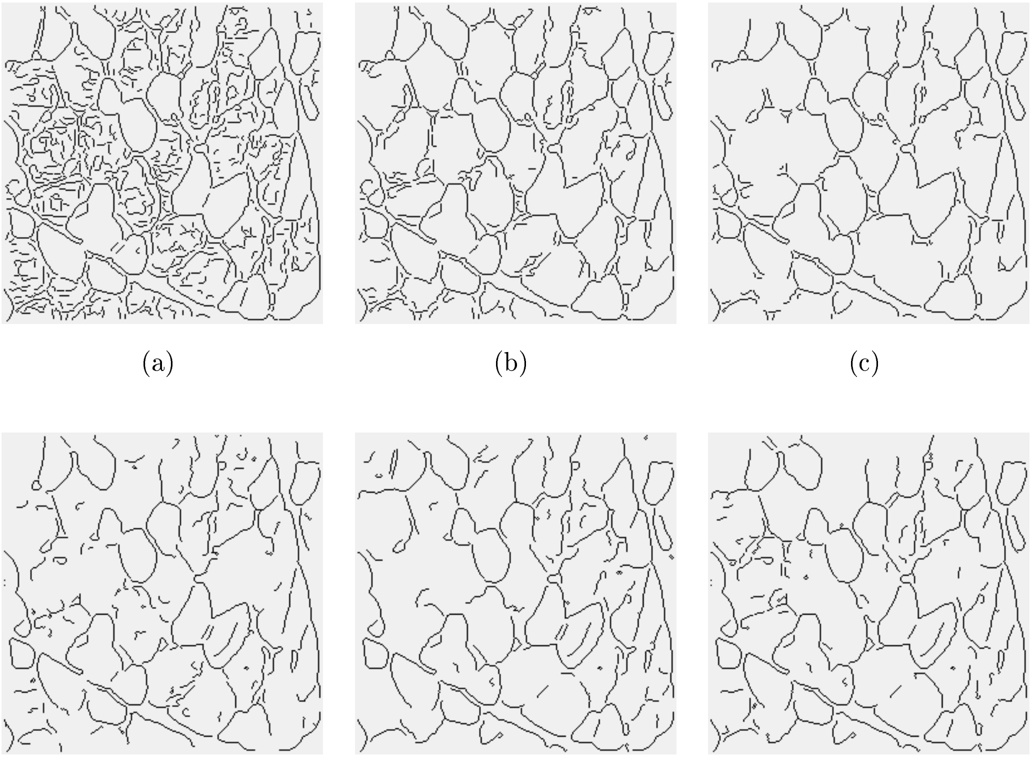
La méthode de détection de contour doit alors être configurée de manière à ce que sa fonction d'évaluation n'inclue que la contribution relative à la maximalité du gradient. Ainsi les processus de construction des contours étiquèteront les mêmes points que ceux validés par le seuillage par hystérésis du Deriche. La différence réside alors dans le choix des seuils. Dans le cas du Deriche, ce seuillage est réalisé par hystérésis, avec deux seuils fixes sur toute l'image, tandis que notre système permet lui d'adapter dynamiquement les valeurs du seuil de la fonction d'évaluation, grâce à la coopération avec les autres méthodes de détection, décrite précédemment.

La table 5.28 présente quelques éléments de mesure entre la carte de contours obtenue par l'algorithme de Deriche, en faisant varier les deux valeurs du seuillage par hystérésis, et par le système de segmentation étudié. Ce dernier est placé dans une configuration où il exploite la même carte de gradient que celle du Deriche, et où la fonction d'évaluation du processus

[35] L'étape de chaînage de Deriche consiste à regrouper les pixels connexes ayant la caractéristique d'être des maxima locaux de la norme du gradient dans la direction du gradient. La méthode de suivi de contour du système coopératif peut reflèter le même comportement : l'examen des pixels candidats voisins des extrémités du contour assure la connexité du contour, il suffit ensuite que le critère de sélection du meilleur candidat ne prenne en compte que la maximalité locale du gradient. Le même type de chaînage que celui de la méthode de Deriche est alors obtenu.

[36] En effet, la méthode classique consistant à convoluer l'image avec des masques de lissage, et des masques de dérivation ne peut pas se faire en un nombre fixe d'opérations lorsque l'amplitude du lissage varie. La taille du masque approximant un lissage gaussien est directement liée à la valeur σ caractérisant la gaussienne utilisée.

[37] Ce type de seuillage est appelé *seuillage par hystérésis.*



(a) (b) (c)

(d) (e) (f)

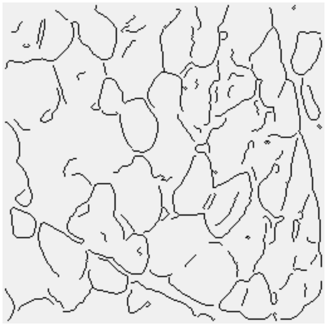
FIG. 5.38: Comparaison entre le détecteur de Deriche, et le système de segmentation dans une configuration où il utilise la même carte de gradient que celle produite par le détecteur de Deriche. La valeur $\alpha = 1.0$ est utilisée pour tous ces exemples. Le seuillage par hystérésis utilise respectivement les couples de valeurs $(5, 15)$ pour le résultat (a), $(15, 30)$ pour le résultat (b) et $(25, 50)$ pour le résultat (c). Les trois images suivantes montrent le résultat obtenu par le système de segmentation, respectivement avec 1, 128 et 256 germes de type contour pour les images (d), (e) et (f).

Pixels communs avec le Deriche	1 germe (3508 pixels)	128 germes (3478 pixels)	256 germes (3580 pixels)
$s = (5, 15)$ (6870 pixels)	3367 (96 %)	**3325 (96 %)**	3457 (97 %)
$s = (15, 30)$ (4591 pixels)	3131 (89 %)	**3162 (91 %)**	3313 (93 %)
$s = (25, 50)$ (3528 pixels)	2853 (81 %)	**2981 (86 %)**	2980 (83 %)

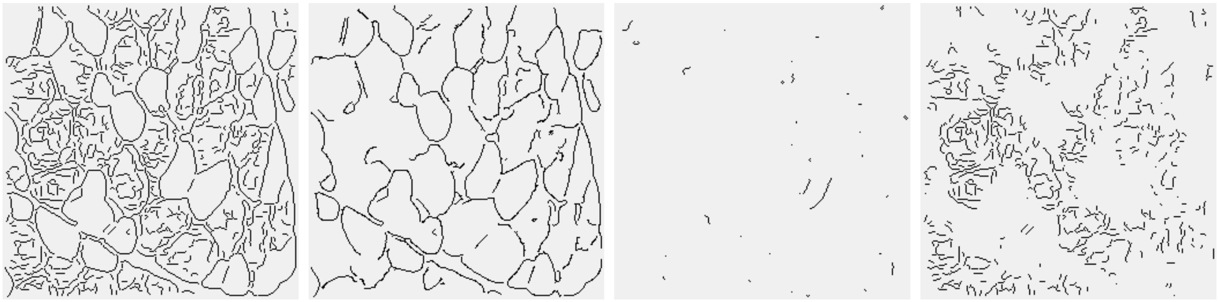
Pixels en moins par rapport au Deriche	1 germe (4310 pixels)	128 germes (4386 pixels)	256 germes (4475 pixels)
$s = (5, 15)$ (6870 pixels)	3407 (50 %)	**3463 (50 %)**	3339 (49 %)
$s = (15, 30)$ (4591 pixels)	1394 (30 %)	**1357 (30 %)**	1223 (27 %)
$s = (25, 50)$ (3528 pixels)	656 (19 %)	**515 (15 %)**	536 (15 %)

Pixels en plus par rapport au Deriche	1 germe (4310 pixels)	128 germes (4386 pixels)	256 germes (4475 pixels)
$s = (5, 15)$ (6870 pixels)	141 (4 %)	**153 (4 %)**	123 (3 %)
$s = (15, 30)$ (4591 pixels)	377 (11 %)	**316 (9 %)**	267 (7 %)
$s = (25, 50)$ (3528 pixels)	655 (19 %)	**497 (14 %)**	600 (17 %)

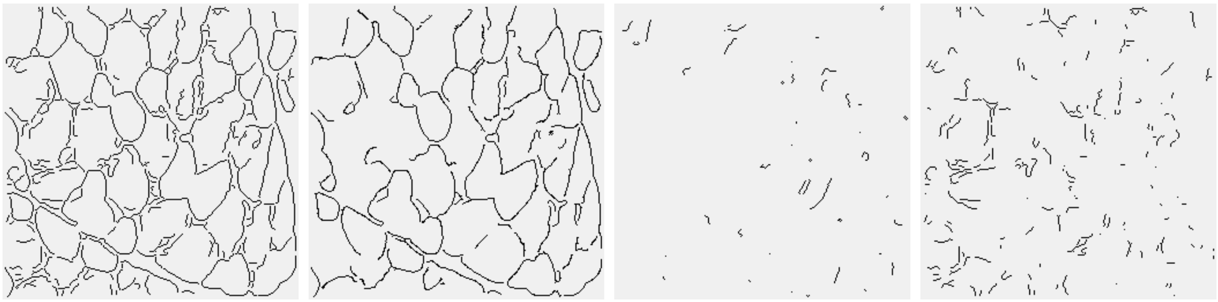
TAB. 5.28: Comparaison des cartes de contours en prenant pour chaque mesure la carte de Deriche comme référence pour les pourcentages indiqués.



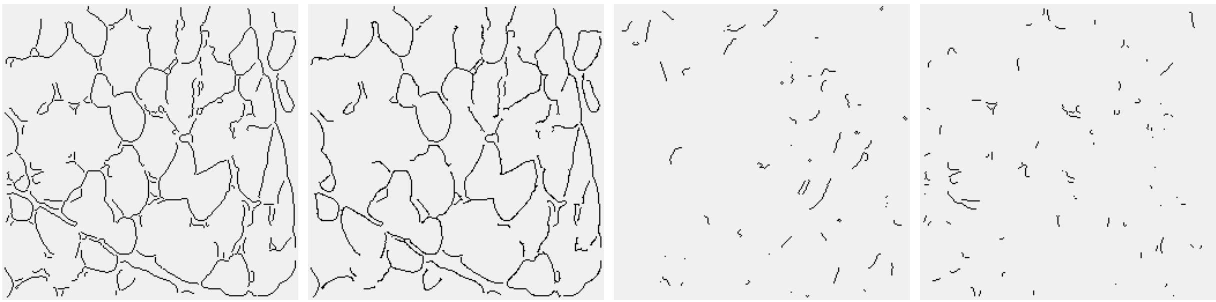
(a)



(b) $(5, 15)$ (c) communs (d) en plus (e) en moins



(f) $(15, 30)$ (g) communs (h) en plus (i) en moins



(j) $(25, 50)$ (k) communs (l) en plus (m) en moins

Fig. 5.39: Comparaison avec la carte contour de Deriche, en utilisant différents seuillages par hystérésis. (a) Résultat obtenu par notre système en l'initialisant avec 128 germes. (b),(f),(j) Deriche avec pour seuils $(5, 15)$, $(15, 30)$ et $(25, 50)$. (c),(g),(k) Pixels communs avec le deriche. (d), (h), (l) Pixels détectés en plus par rapport au Deriche. (e), (i), (m) Pixels détectés en moins par rapport au Deriche.

contour reproduit un comportement de chaînage des pixels maxima locaux du gradient. Trois conditions d'initialisation du système sont testées, en faisant varier le nombre de processus germe initiaux, respectivement avec 1, 128 et 256 germes de type contour [38] [39]. Les primitives de type région ont été retirées des résultats obtenus pour clarifier la lecture.

La figure 5.38 présente les cartes de contours obtenues par le Deriche et par le système de segmentation. Les images 5.38(a), (b) et (c) montrent l'influence classique du choix crucial des valeurs du seuillage par hystérésis. Le résultat (a) est clairement sur-segmenté. Des transitions faiblement marquées sont obtenues au prix d'un ensemble de contours parfaitement inexploitables à l'emplacement de chaque cellule texturée. Il faut sensiblement augmenter les seuils pour échapper à la texture, comme le résultat (c) le montre [40]. La contrepartie est alors la perte des transitions faibles. Les trois résultats (d), (e) et (f) montrent des cartes de contours relativement comparables en qualité, et en nombre de pixels obtenus, prouvant encore une fois la robustesse par rapport aux conditions d'initialisation.

La table 5.28 traduit en chiffres les constatations visuelles précédentes.

- La comparaison colonne par colonne donne des résultats très similaires, ce qui indique que les trois exécutions du système de segmentation ont produit des résultats comparables.
- La disparition de la sur-segmentation lorsque les seuils de l'hystérésis augmentent apparaît dans la table des *Pixels en moins par rapport au Deriche*, où les valeurs passent de 50 % jusqu'à 15 % environ en utilisant le couple de seuils $(25, 50)$.
- Le nombre de pixels en plus par rapport au Deriche est très faible lorsque les seuils du Deriche sont faibles, autour de 5 % en utilisant le couple $(5, 15)$. Les figures 5.39 (d), (h) et (l) montrent que la plupart de ces pixels trouvés en plus correspondent à des transitions significatives de l'image.

La figure 5.39 compare graphiquement les résultats du Deriche, avec un des résultats de notre système. Son choix importe peu, car ils sont tous très comparables. Le choix s'est porté sur l'exécution avec 128 germes initiaux, et correspond à la colonne en gras dans la table 5.28. L'image 5.39(a) montre ce résultat. Les quatre bandes d'images suivantes comparent ce résultat aux trois exécutions du Deriche précédemment décrites. Dans chacune de ces bandes, la première image présente le résultat brut, la deuxième est le sous-ensemble des pixels communs entre le Deriche et le résultat (a), la troisième montre les pixels trouvés *en plus* par notre système, et la quatrième montre enfin les pixels trouvés *en plus* par le Deriche. Les résultats (e), (i) et (m) confirment que le système de segmentation s'adapte bien à la texture, puisqu'il ne marque aucun des contours trouvés *à tort* par le Deriche dans les zones texturées. Les résultats (d), (h), (l) montrent les pixels trouvés en plus par le système de segmentation, qui correspondent pour la plupart à de faibles transitions. Le résultat (d) présente à ce niveau une qualité de détection très similaire au Deriche, puisque d'après la table 5.28, le nombre de pixels de cette image (d) ne représente que 4 % du nombre de pixels total détectés (image (a)).

Le système de segmentation proposé se place dont à un niveau différent des approches

[38] Le système aurait d'ailleurs pu être initialisé avec des processus de type région de façon similaire.

[39] Le nombre important de processus initiaux de type contour, répartis dans l'image selon les plus fortes valeurs du gradient, voir le paragraphe 2.4.1, avait pour objectif de rapprocher notre système des conditions de fonctionnement du Deriche. Dans ce dernier, l'étape de chaînage des maxima locaux du gradient explore systématiquement tous les pixels de l'image, ce qui peut s'assimiler à autant de germes initiaux potentiels dans ce cas. Les résultats montrent d'ailleurs que ce nombre de germes initiaux est peu influent sur la complétude du résultat.

[40] Une action aurait été possible sur le paramètre α du lissage pour limiter les fausses détections en texture. Cependant, le lissage entraîne l'apparition d'autres effets indésirables dans l'image. Les contours sont de plus en plus délocalisés de leur position optimale, et les contours s'interrompent de plus en plus loin au niveau des points de jonction.

(a) $c = 4$ (b) $c = 2$ (c) $c = 1$

FIG. 5.40: Expérimentation montrant l'influence de l'adaptation de la fonction d'évaluation du processus contour, lorsque celle-ci intervient une seule fois au moment de sa création. Le seuil appliqué dépend linéairement (coefficient c) du gradient moyen de la fenêtre de focalisation. Les fortes valeurs de c produisent un seuil du gradient plus grand, le processus est donc plus sélectif.

classiques utilisant un filtrage optimal de l'image, telles que le Deriche sommairement présenté ici. L'approche de Deriche trouve sa force dans l'implémentation efficace qui est faite pour calculer une carte de gradient de l'image. Les traitements ultérieurs effectués sur cette carte de gradient, tels la détection des maxima locaux ou le chaînage sont, eux, très banals.

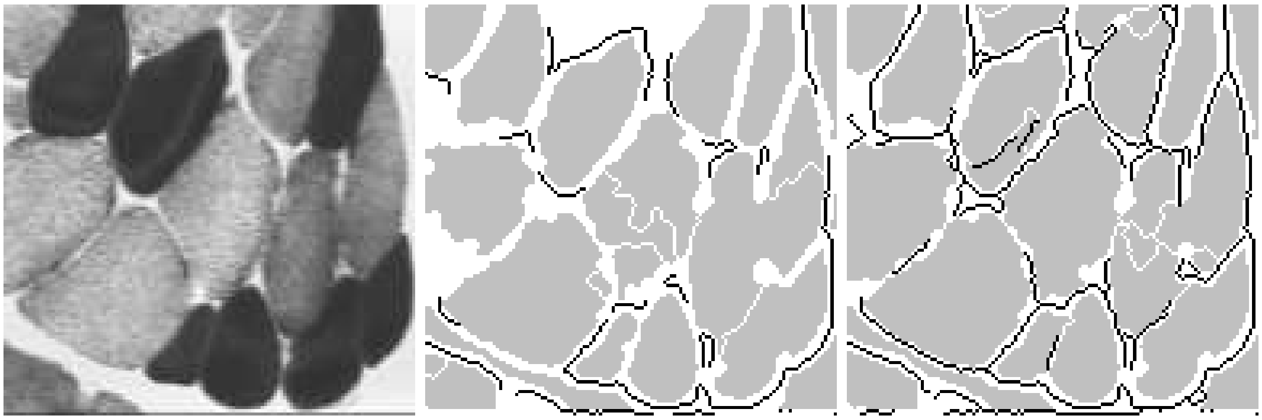
Notre approche ne se situe donc pas au même niveau, puisque le calcul du gradient est à nos yeux une étape particulièrement préliminaire des traitements. A ce titre, l'intégration du gradient de Deriche à la place du gradient classique permet aux processus de segmentation de fonctionner sans autre formalité. Les résultats obtenus indiquent clairement qu'à partir des mêmes informations photométriques que le Deriche, il est possible d'obtenir des résultats plus convaincants, puisque la coopération entre les processus assure une adaptation puissante, que ne permet par l'utilisation d'un seuillage uniforme par hystérésis.

Adaptation du seuil de l'évaluation

L'expérimentation suivante a pour objectif de montrer l'intérêt de l'adaptation des seuils de la fonction d'évaluation des processus de segmentation.

La figure 5.40 présente quelques résultats, dans lesquelles le mode d'adaptation utilisé est l'affectation d'une valeur locale, de manière définitive lors de la création du processus. Ce seuil dépend linéairement du gradient moyen de la fenêtre de focalisation qui va servir à placer le germe. La figure 5.41(b) présente le résultat obtenu lorsque cette adaptation n'a pas lieu. Tous les processus utilisent le même seuil statique sur le gradient, fixé arbitrairement à la valeur de 30. Le résultat obtenu n'est pas très satisfaisant, car seuls les plus forts pixels contours sont segmentés.

Les images de la figure 5.40(a),(b) et (c) présentent des résultats dans lesquels le seuil initial du processus contour dépend linéairement (coefficient c) du gradient moyen de la fenêtre de focalisation. Les résultats sont plus complets, mais les trois essais montrent que l'ajustement de cette constante risque d'être délicat : seul le résultat (c) segmente les transitions entre les cellules noires, mais au détriment de nombreux autres contours non significatifs en texture. Ce



(a) Niveaux de gris. (b) Sans adaptation. (c) Avec adaptation.

FIG. 5.41: Expérimentation montrant l'influence de l'adaptation dynamique des seuils du processus de type contour en fonction de l'homogénéité des régions avoisinantes. (a) L'image en niveaux de gris est une partie de l'image *cellules256.gif*. (b) Chaque processus utilise le même seuil statique sur le gradient (30). (c) Les processus calculent le seuil à appliquer sur le gradient en fonction de l'homogénéité des régions adjacentes à l'extrémité.

résultat montre donc qu'une seule adaptation du seuil du gradient ne suffit pas, il faut pouvoir discriminer les cas où le processus travaille en zone homogène ou en texture.

La figure 5.41(c) présente un résultat obtenu dans lequel le seuil du gradient s'adapte dynamiquement, selon l'homogénéité du voisinage de part et d'autre de l'extrémité : pour chaque pixel situé de chaque coté de l'extrémité :

- Soit ce pixel est situé dans une région déjà segmentée, auquel cas le système considère l'écart-type de ladite région.
- Sinon, une mesure d'écart-type local sur une fenêtre 3×3 centrée sur le pixel est effectuée.

Muni de ce couple d'écarts-types (σ_1, σ_2), le seuil du gradient est choisi dans la table suivante :

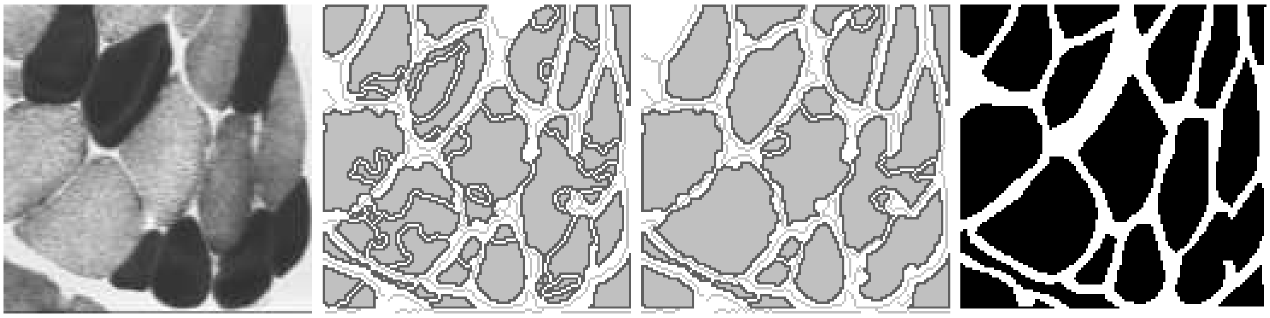
Seuil Grad	$\sigma_1 < 2$	$2 \leq \sigma_1 < 5$	$5 \leq \sigma_1 < 8$	$8 \leq \sigma_1 < 10$	$10 \leq \sigma_1$
$\sigma_2 < 2$	3	5	7	9	11
$2 \leq \sigma_2 < 5$	5	8	12	14	19
$5 \leq \sigma_2 < 8$	7	12	15	17	20
$8 \leq \sigma_2 < 10$	9	14	17	20	25
$10 \leq \sigma_2$	11	19	20	25	28

Le dernier résultat de la figure 5.41 est le plus intéressant, car il détecte de faibles contours, sans pour autant trouver de faux pixels contour dans la texture. Par ailleurs, la table de seuils précédente étant paramétrée, on peut envisager de modifier sélectivement les seuils seulement pour les couples (σ_1, σ_2) précis, si le résultat présente des défauts spécifiques à ces couples d'intervalles.

5.3.4 La coopération

Guider les contours par les régions

L'expérimentation correspondante a été produite dans le chapitre 1, voir la figure 1.21. Une pondération judicieuse entre le critère du plus fort gradient, et celui privilégiant une distance



(a) Niveaux de gris. (b) Sans fusion. (c) Avec fusion. (d) Référence.

FIG. 5.42: Influence de la fusion des primitives région sur le résultat final. (a) L'image en niveaux de gris est une partie de l'image *cellules256.gif*. (b) Les primitives régions ne fusionnent pas entre elles. La mesure de Vinet de ce résultat vaut $V_1 = 0.334$. (c) Les primitives régions fusionnent dynamiquement au cours de leur construction. La mesure de Vinet de ce résultat de ce résultat vaut $V_2 = 0.184$. (d) La carte des régions de référence.

constante avec la plus proche région peut permettre de modifier le comportement de croissance par défaut pour résoudre des problèmes de segmentation délicats (dans cet exemple, le passage d'une transition forte à une transition faible mais rattachée cependant au même objet).

Cerner les régions par les contours

L'expérimentation a été effectuée dans le chapitre 2, voir la figure 2.14. La construction *au bon moment* de primitives contours permet de cerner la région en construction, et l'empêcher de déborder sur des objets voisins. Le contour réalise à ce titre davantage qu'une simple émergence d'information photométrique, il contribue par sa présence à augmenter la qualité de la primitive région voisine.

Fusionner les primitives (cas région)

Cette expérimentation a été effectuée en inhibant le mécanisme consistant à fusionner les primitives régions au cours de leur construction : les deux types de fusion ont été inhibés indifféremment :

- la fusion avec une région en cours de construction, nécessitant la synchronisation de deux processus de segmentation.
- la fusion avec une région qui n'est plus rattachée à un processus (*région morte*).

La figure 5.42 présente les résultats obtenus selon l'activation ou pas du protocole de fusion. L'image (c) a résulté de 37 fusions entre régions sur une condition de synchronisation, et de 29 fusions avec une région *morte*, c'est-à-dire qui n'est plus rattachée à un processus actif. Le résultat de l'image (c) est visuellement plus satisfaisant. Les deux importantes régions texturées situées en bas à gauche sont ainsi segmentée en deux primitives uniquement. En contrepartie, une "erreur" de fusion a été commise sur les deux cellules noires en bas à droite. Le terme d'erreur n'est pas tout à fait exact, car s'il est vrai que les primitives fusionnées correspondent à deux cellules distinctes, les niveaux de gris ne permettent pas de distinguer une frontière complète, ce qui explique que la carte de référence de l'image (d) propose elle-aussi une unique primitive pour ces deux cellules. La mesure de Vinet est explicite, puisqu'elle fournit une valeur

STATISTIQUES GENERALES		
Mesure de Vinet	0.334	
Nombre de régions	48	
Nombre de pixels moyen par région	231	
Taux de remplissage de l'image		67.7 %
Rapport nb pixels frontière/nb pixels total		35.8 %
COMPARAISON AVEC LA REFERENCE		
Nombre de régions de référence	19	
Régions de référence non segmentées (commun=0)	1	
Régions de référence segmentée (commun=1)	4	
Régions de référence sur-segmentées (commun¿1)	14	
Nombre de régions intersectantes	2.78	

TAB. 5.29: Statistiques sur la comparaison entre le résultat obtenu **sans** fusion et la carte de référence.

STATISTIQUES GENERALES		
Mesure de Vinet	0.184	
Nombre de régions	27	
Nombre de pixels moyen par région	419	
Taux de remplissage de l'image		69.0 %
Rapport nb pixels frontière/nb pixels total		24.6 %
COMPARAISON AVEC LA REFERENCE		
Nombre de régions de référence	19	
Régions de référence non segmentées (commun=0)	1	
Régions de référence segmentée (commun=1)	11	
Régions de référence sur-segmentées (commun¿1)	7	
Nombre de régions intersectantes	1.72	

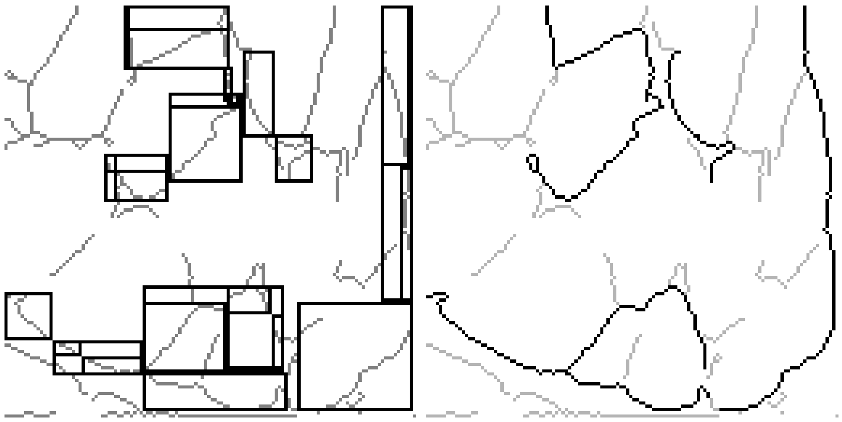
TAB. 5.30: Statistiques sur la comparaison entre le résultat obtenu **avec** fusion et la carte de référence.

significativement meilleure ($V_1 = 0.184$) dans le cas de l'image (c) lorsque l'on compare le résultat à la carte des régions de référence. La mesure de Vinet calcule le meilleur recouvrement entre des couples de régions issues des deux cartes à comparer. Une sur-segmentation ne fournira pas une mesure de Vinet faible, car la méthode n'autorise pas d'apparier plus d'une unique région par région de référence.

Le résultat produit en autorisant les fusions est donc plus intéressant dans une optique de post-traitement, car le nombre d'objets de type région à manipuler sera plus faible avec l'image (c) qu'avec l'image (b). Les tables 5.29 et 5.30 permettent d'effectuer quelques comparaisons. Le résultat (b) comporte 48 régions contre 27 pour le résultat (c), sachant que la carte de référence n'en compte que 19. La valeur qui semble la plus significative de ces deux tables est certainement la moyenne du *Nombre de régions intersectantes* (sic). Pour chaque région de référence, on compte le nombre de régions du résultat qui sont en intersection avec ladite région, puis on fait la moyenne sur toutes les régions de référence. Le résultat (b) présente pour caractéristique d'avoir en moyenne 2.78 régions en intersection par région de référence, alors que ce chiffre est de 1.72 pour le résultat (c).

Fusionner les primitives (cas contours)

La fusion des contours n'est pas visuellement explicite, car la fusion des primitives porte pour la plupart des cas sur des objets, dont les extrémités se touchent, ou sont situées à un ou deux pixels de distance. Les fusions de contours sont pourtant d'un intérêt crucial pour réduire, là encore, le nombre de structures manipulées. Ainsi, dans l'expérimentation précédente, figure 5.42, la fusion des contours était activée dans les mêmes conditions que pour les régions, et elle a donné lieu à 13 fusions de contours. La figure 5.43 tente de visualiser ces 13 fusions, en marquant les structures de contours ayant fusionné au cours de l'exécution. Toutes les fusions



(a) Boîtes englobantes. (b) Contours fusionnés.

FIG. 5.43: Mise en évidence des primitives contour ayant fait l'objet de fusion entre-elles dans l'expérimentation du précédent paragraphe. (a) Boîtes englobantes des contours concernés par la fusion. Les points de contact entre les extrémités de ces boîtes indiquent les extrémités concernées par la fusion. (b) Les contours foncés indiquent ceux qui ont effectué une fusion.

observées ici concernent des extrémités directement en contact. Aucun pixel n'a donc été rajouté au cours de l'étape de fusion, pour permettre un raccordement entre les deux primitives.

5.4 Conclusion

Ce chapitre s'est attaché à démontrer expérimentalement l'intérêt et le bien-fondé de l'approche coopérative et adaptative décrite tout au long de ce document. Un premier temps a servi à la présentation et à l'argumentation autour d'un protocole expérimental pertinent pour un système de segmentation :

- travaillant sans connaissances *a priori* sur un domaine du traitement d'image ;
- opérant sur des images naturelles, pour lesquelles un résultat optimal sera rarement consensuel entre plusieurs experts du domaine ;
- générant enfin un résultat constitué de primitives hétérogènes régions et contours, pouvant chacune prétendre à être évaluée séparément.

Le protocole d'évaluation a ainsi mis en exergue la notion de carte de primitives contours et régions de référence, établies manuellement par l'expert humain, en insistant sur le fait que la référence sera le résultat des choix d'un individu à un instant donné, avec leurs qualités (objectivité, absence d'*a priori* sur l'image), et leurs défauts (zones équivoques, pixels oubliés, sensibilité de détection variables). Cependant, ces cartes de références semblent être un outil précieux, car elles représentent un premier pas vers une validation quantitative, et moins arbitraire des méthodes de segmentation, et car leur utilisation ne nécessite pas de restreindre le problème de l'évaluation à des images synthétiques, qui ne parviennent pas à reproduire les difficultés de segmentation multiples et variées des images naturelles.

L'évaluation du système coopératif et adaptatif s'est orientée autour de plusieurs axes, visant chacun à éclairer des particularités et des fonctionalités originales de l'approche.

Il nous a semblé tout d'abord intéressant de montrer que la philosophie de construction distribuée d'une solution, ainsi que la seule définition d'interaction locale entre des processus, apporte une robustesse particulièrement appréciable du système face à des conditions d'utilisation fluctuantes selon l'utilisateur :

- Le système s'avére stable par rapport aux conditions dans lesquelles il est initialisé. Quels que soient le nombre, la nature, et l'endroit où des processus initiaux sont placés dans l'image, la dynamique d'exploration du territoire, définie encore une fois par de simples contraintes topologiques locales, assure peu à peu une couverture complète de l'image. La présentation de résultats sous la forme d'images statiques est à ce niveau particulièrement frustrante, car elle ne permet pas visualiser cet aspect dynamique et réactif, qui tient une grande place dans la conception de l'approche.
- Le système s'avére de même robuste par rapport aux interactions que l'utilisateur peut avoir sur son fonctionnement : limitations de ressources système utilisées, modifications dynamiques par le biais d'une interface graphique, etc.
- Le système s'avére enfin robuste par rapport aux données elles-mêmes, puisqu'un même objet pourra généralement être segmenté par plusieurs *voies d'approche* différentes au cours de l'exécution. L'échec local d'un processus ne remet en aucun cas en cause la viabilité globale du système.

La mesure de la qualité des résultats a porté essentiellement sur la qualité de la carte contour, par rapport à un détecteur classique de Deriche. Les résultats obtenus sont pour la plupart comparables à ceux de Deriche, en insistant sur le fait que le Deriche a fonctionné avec des paramètres optimaux, adaptés à chaque image testée pour offrir le meilleur résultat. Une comparaison avec une méthode de détection de région classique n'a pas été effectuée pour des raison d'incompatibilité du formalisme de représentation : les méthodes de détections de régions construisent généralement une partition complète de l'image, alors que le système coopératif permet à des pixels de demeurer sans étiquette [41].

Les dernières expérimentations portant sur l'adaptation et la coopération démontrent enfin la grande réactivité et l'extrême configurabilité (sic) de ce système de segmentation. Une méthode disposant de trop de paramètres ajustables est souvent considérée à tort avec défiance. La multiplicité des paramètres la rend difficilement exploitable par un utilisateur novice, qui ne saura pas sur quel paramètre agir pour orienter le fonctionnement de la méthode dans le sens qu'il souhaite. Mais vaut-il mieux exhiber les paramètres d'une méthode, ou bien tenter de les dissimuler au mieux [42] ? Quelles sont les possibilités de l'utilisateur face à un algorithme de type boîte noire disposant de trois paramètres [43] uniquement, et qui ne parvient pas cependant à obtenir le résultat escompté ? Se tourner vers une autre méthode de type boîte noire radicalement différente, ou bien tenter de programmer lui-même un algorithme pour son problème de traitement d'image ? N'est-il pas plus intéressant pour lui de disposer d'une plateforme de segmentation largement configurable, avec laquelle il a plus de chance de trouver "**L**e **B**on **J**eu **D**e **P**aramètres". Cela nécessitera certainement davantage d'investissement initial de sa part que de faire fonctionner un algorithme de type boîte noire, mais il peut espérer tout de même gagner du temps en évitant de développer son propre algorithme spécifique à son problème. Seulement à cette condition, l'utilisation d'un système hautement configurable permet un gain de temps.

---

[41] Il nous semble plus judicieux de n'étiqueter que des pixels dont l'appartenance à une région est certaine, en limitant ainsi les risques de mauvais étiquetage.

[42] Un paramètre peut être dissimulé à l'utilisateur en codant explicitement sa valeur au cœur de l'algorithme, ou plus subtilement en calculant dynamiquement sa valeur à partir de considérations sur l'image. Le premier cas rend généralement l'algorithme spécifique à un type d'images pour lesquelles le paramètre est adapté, le deuxième cas se veut plus généraliste, mais dans les deux cas on retire à l'utilisateur la possibilité d'agir lui-même sur la valeur de ce paramètre.

[43] Simple cas d'école.

Conclusion

Nous avons tenté de présenter dans les pages qui ont précédé un système de vision utilisant des processus de segmentation de bas niveau, et mettant l'accent plus particulièrement sur la gestion efficace de l'information.

La coopération entre les méthodes qui y est présentée est une partie intégrante du mécanisme de décision de chaque processus, et contribue en ce sens à son originalité. Elle permet en particulier d'avoir des décisions adaptées localement, de différer les décisions plus délicates, et de résoudre les problèmes de segmentation par la sollicitation et l'accumulation d'information, de manière opportuniste.

Dans cette optique, des processus de segmentation de bas niveau, de type contour et région, réalisant une construction incrémentale de la solution, présentent un comportement décisionnel adaptatif et réactif aux données de l'image, tout en disposant de la capacité de création de processus *fils* pour aider à la résolution de problèmes complexes. Les processus sont contrôlés par un séquenceur de tâches, et fonctionnent conjointement dans un pseudo-parallélisme inspiré des techniques de fonctionnement des systèmes d'exploitation d'ordinateurs. Selon ce modèle, des possibilités de communication dédiées ont été développées.

La résolution de problème apparaît alors comme une fonctionnalité émergente du système, à la différence des approches classiques, qui requièrent une planification de la stratégie d'exploration de l'image. Dans le cas étudié, cette dernière est le résultat d'une interaction réactive, et se caractérise par un effet commun d'enrichissement progressif lié à l'élaboration d'une solution globale.

Les cinq précédents chapitres ont tenté de donner une vision harmonieuse et cohérente à un système qui n'en demeure pas moins selon nous anticonformiste et pluridisciplinaire. Les perspectives d'évolution future sont nombreuses, tant un nombre important de portes ont été ouvertes au cours de ce mémoire, sous la forme de questions posées au détour d'un paragraphe, de voies non explorées, de configurations extrêmes, etc.

La perspective qui vient le plus aisément à l'idée est la réalisation d'une parallélisation réelle de l'approche. La voie de recherche demeure séduisante, car en l'état actuel de l'implémentation, peu de difficultés techniques insurmontables empêchent de faire fonctionner ce principe sur un réseau de processeurs parallèles. Au contraire même, l'implantation sous la forme d'un système séquençant des tâches anonymes rend la parallélisation plus simple. Au lieu que le séquenceur n'exécute qu'une tâche à la fois, il lui suffit de vider sa liste de tâches actives sur autant de processeurs qu'il en existe de disponibles, et de libres. La difficulté majeure serait de résoudre en exclusion mutuelle les accès aux données –image et résultat–, très volumineux actuellement.

Une autre perspective consiste à enrichir davantage les primitives fournies par le système, en y introduisant des régions de type dégradé qui font défaut pour caractériser les zones où le changement de d'intensité est trop rapide pour être assimilé à une région homogène, et trop lent pour un contour. Des contours de type ligne pourraient également être incorporés, ainsi que des détecteurs de structures particulières (détecteurs de coins, de jonctions, etc).

Une perspective supplémentaire consiste enfin à envisager les moyens à mettre en œuvre pour "piloter" le système en partie depuis une autre méthode de haut niveau, dont la connaissance particulière sur un problème à résoudre permettrait d'orienter et de guider le fonctionnement du système actuel. Un protocole minimum pour les échanges d'information entre ces deux niveaux rendrait possible l'adaptabilité de ce système de bas niveau avec différents systèmes de résolution de problèmes de vision de haut niveau, sur la base du même système de segmentation coopérative et adaptative.

La vision de bas niveau reste plus que jamais un domaine d'actualité, dans lequel l'expérimentation personnelle constitue le meilleur moyen pour appréhender les difficultés réelles des tâches à accomplir. De plus, il nous donne régulièrement une bonne leçon d'humilité, à nous autres chercheurs en informatique, en nous montrant que la technologie du matériel informatique actuelle, alliée à l'état de l'art en algorithmique de traitement d'image ne sont pas encore sur le point d'inquiéter la capacité de l'œil humain. Mais pour combien de temps encore ?

Bibliographie

[AB94] Adams and Bishof. Seeded region growing. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 16(6) :641–647, 1994.

[ABM87] H.L. Anderson, R. Bajcsy, and M. Mintz. A modular feedback system for image segmentation. Technical report, University of Pennsylvania, GRASP Lab., 1987.

[BB92] J. Benois and D. Barba. Image segmentation by region-contour cooperation for image coding. In *Proceedings of the 11th International Conference on Pattern Recognition*, volume 3, pages 331–334, 1992.

[BBMP93] N. Bianchi, P. Bottoni, P. Mussio, and M. Protti. Cooperative visual environments for the design of effective visual systems. *Journal of Visual Language Computing*, 1993.

[BCW93] P.F. Bertrand, W. Cowan, and M. Wein. A window architecture providing predictable temporal performance. In *Graphics Interface '93*, 1993.

[BF70] C.R. Brice and C.L. Fennema. Scene analysis using regions. *Artificial Intelligence*, 1 :205–226, 1970.

[BG93] O. Baujard and C. Garbay. Kiss : a multiagent segmentation system. *Optical Engineering*, 32(6) :1235–1249, 1993.

[BG96] F. Bellet and C. Garbay. Des processus adaptatifs et coopératifs pour la vision de bas niveau. In *Actes du 10ème Congrès "Reconnaissance des Formes et Intelligence Artificielle"*, volume 2, pages 516–526, 1996.

[BMPR94] N. Bianchi, P. Mussio, M. Padula, and G. Rubbia Rinaldi. Anthropocentric data management : A medical case. In Graeme Shanks and David Arnott, editors, *Proceedings of the 5th Australasian Conference on Information Systems*, volume 1, pages 323–331, Merbourne, Australia, September 1994.

[Bon91] P. Bonnin. *Méthode Systématique de conception et de réalisation d'applications en vision par ordinateur*. Thèse de doctorat, Université de Paris VII, France, 1991.

[Bou83] S.R. Bourne. *The Unix System*. Addison-Wesley, 1983.

[BPG91] O. Baujard, S. Pesty, and C. Garbay. A programming environment for distributed vision system design. In *Proceedings of the 6th International Conference on Image Analysis and Processing*, pages 380–384, 1991. Como (Italy).

[Bro90] R.A. Brooks. Integrated systems based on behaviors. *SIGART Bulletin*, 2(4) :46–50, 1990.

[Bro91] R.A. Brooks. Intelligence without reason. Technical Report AI Memo No 1293, Massachussets Institute of Technology, AI Lab., April 1991.

[BSG94] F. Bellet, M. Salotti, and C. Garbay. Low level vision as the opportunist scheduling of incremental edge and region detection processes. In *Proceedings of the 12th International Conference on Pattern Recognition (ICPR 94)*, volume 1, pages 517–519, 1994.

[BSG95] F. Bellet, M. Salotti, and C. Garbay. Une approche opportuniste et coopérative pour la vision de bas niveau. *Traitement du Signal*, 12(5) :479–494, 1995.

[CA93] C.C. Chu and J.K. Aggarwal. The integration of image segmentation maps using region and edge information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(12) :1241–1252, 1993.

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

[Cap95] Olivier Capdevielle. *Formulation d'Objectifs de Traitement d'Images : une Exploitation Interactive d'un Système de Planification Automatique de Chaînes d'Opérateurs*. Thèse de doctorat, Université Paul Sabatier, Toulouse, 1995.

[CJC96] E. Chiarello, L.M. Jolion, and C.Amoros. Regions growing with the stochastic pyramid : Application in landscape ecology. *Pattern Recognition*, 29(1) :61–75, 1996.

[CL83] D.D. Corkill and V.R Lesser. The use of meta-level control for coordination in a distributed problem-solving network. In *Proceedings of the 8th International Joint Conference on Artificial Intelligence*, pages 748–756, 1983.

[CM91] J.M. Chassery and A. Montanvert. Géométrie discrète en analyse d'images. In *Traité des Nouvelles Technologies - série Images*. Editions HERMES, 1991.

[CS87] B.D. Chen and P. Siy. Forward/backward contour tracing with feedback. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 9(3) :438–446, 1987.

[DCB+89] B. Draper, R. Collins, J. Brolio, A. Hanson, and E. Riseman. The schema system. *International Journal of Computer Vision*, 2 :209–250, 1989.

[Der87] R. Deriche. Using Canny's criteria to derive a recursively implemented optimal edge detector. *International Journal of Computer Vision*, pages 167–187, 1987.

[FBC94] R.K. Falah, P. Bolon, and J.P. Cocquerez. A region-region and region-edge cooperative approach of image segmentation. In *Proceedings of the IEEE ICIP Conference*, volume 3, pages 470–474, Nov 1994.

[Fer95] J. Ferber. *Les Systèmes Multi-Agents. Vers une Intelligence Collective*. InterEditions, 1995.

[FG88] J. Ferber and M. Ghallab. Problématiques des univers multi-agents intelligents. In *Acte des journées nationales PRC-GRECO Intelligence Artificielle*, pages 295–320, Toulouse, 1988.

[Fle92] M.M. Fleck. Some defects in finite-difference edge finders. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 14(3) :337–345, 1992.

[FS75] H. Freeman and R. Shapira. Determining the minimum area encasing rectangle for an arbitrary closed curve. *Communications of the ACM*, 18(7) :409–413, 1975.

[GBB98] C. Garbay, F. Bellet, and A. Boucher. Des agents situés pour l'interprétation de scènes. *Revue d'Intelligence Artificielle*, 12(1) :11–36, 1998. Numéro spécial "Intelligence Artificielle Distribuée et Systèmes Multi-Agents".

[Gir93] S. Giroux. *Agents et systêmes, une nécessaire unité*. Thèse de doctorat, Département d'informatique et de recherche opérationnelle, Faculté des Arts et Sciences, Université de Montréal, 1993.

[GM85] A. Gagalowicz and O. Monga. Un algorithme de segmentation hiérarchique. In *Proceedings du 5ème Congrès Reconnaissance des Formes et Intelligence Artificielle (RFIA 85)*, volume 1, pages 163–177, 1985.

[GSL94] S. Giroux, A. Senteni, and G. Lapalme. ReActalk, du monde réel à des systèmes informatiques ouverts et adaptatifs. In *Proceedings du 9ème Congrès Reconnaissance des Formes et Intelligence Artificielle (RFIA 94)*, volume 2, pages 259–280, 1994.

[HB90] J.F. Haddon and J.F. Boyce. Image segmentation by unifying region and boundary information. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(10) :929–948, 1990.

[Hew77] C.E. Hewitt. Viewing control structure as patterns of passing messages. *Artificial Intelligence*, 8 :323–364, 1977.

[HP76] S.L. Horowitz and T. Pavlidis. Picture segmentation by a tree traversal algorithm. *Journal of the Association for Computing Machinery*, 23(2) :368–388, 1976.

[HR85] B. Hayes-Roth. A blackboard architecture for control. *Artificial Intelligence*, 26(3) :251–321, 1985.

[HS88] L. Hertz and R.W. Schafer. Multilevel thresholding using edge matching. *Graphicals Models and Image Processing*, 44 :279–295, 1988.

[Kim82] C.E. Kim. Digital convexity, straitghtness, and convex polygons. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 4(6) :618–626, 1982.

[KR80] C.E. Kim and A. Rosenfeld. On the convexity of digital regions. In IEEE Comp. Soc. Press, editor, *Proceedings of the 5th JCPR*, pages 1010–1015, 1980.

[Kra87] S. Krakowiak. *Principes des systèmes d'exploitation des ordinateurs*. Dunod Informatique, 1987.

[Liu77] H.K. Liu. Two- and three-dimensional boundary detection. *Computer Graphics and Images Processing*, 6 :123–134, 1977.

[Lou95] F. Loubiat. Dea d'informatique : Ergonomie d'un système de vision. Technical report, TIMC-IMAG, Informatique et Mathématiques Appliqués de Grenoble, June 1995.

[Mar76] A. Martelli. An application of heuristic search methods to edge and contour detection. *Communications of the ACM*, 19(2) :73–83, 1976.

[Mar82] D. Marr. *Vision*. Freeman, 1982.

[Mat89] T. Matsuyama. Expert systems for image processing : knowledge-based composition of image analysis processes. *Graphicals Models and Image Processing*, 48 :22–49, 1989.

[MC92] M. Melkemi and J.M. Chassery. Edge-region segmentation process based on generalized voronoï diagram representation. In *Proceedings of the 11th International Conference on Pattern Recognition*, volume 3, pages 323–326, 1992. The Hague.

[MCOT89] E. De Micheli, B. Caprile, P. Ottonello, and P. Torre. Localization and noise in edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 11 :1106–1117, 1989.

[MFGC92] P. Mussio, M. Finadri, P. Gentini, and F. Colombo. A bootstrap approach to visual user-interface design and development. *The Visual Computer*, 8 :75–93, 1992.

[Mil79] D.L. Milgram. Region extraction using convergent evidence. *Computer Graphics and Images Processing*, 11 :1–12, 1979.

[Mér81] L. Mérö. An optimal line following algorithm. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 3(5) :593–598, 1981.

[Mra97] M.C. Mraghni. *Détection de chaînes de contours dans une image numérique par approche symbolique et par grammaire de formes.* Thèse de doctorat, spécialité informatique, Université de Tours, March 1997.

[NL84] A.M. Nazif and M.D. Levine. Low level image segmentation : An expert system. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(5) :555–577, March 1984.

[PL90] T. Pavlidis and Y.T. Liow. Integrating region growing and edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 12(3) :225–233, 1990.

[Pra91] W.K. Pratt. *Digital image processing, second edition.* J. Wiley & Sons, 1991.

[PSWH84] T.C. Pong, L.G. Shapiro, L.T. Watson, and R.M. Haralick. Experiments in segmentation using a facet model region grower. *Graphicals Models and Image Processing*, 25 :1–23, 1984.

[RD83] R.J. Smith R. Davis. Negociation as a metaphor for distributed problem solving. *Artificial Intelligence*, 20(1) :63–109, 1983.

[RM89] R. Reiter and A.K. Mackworth. A logical framework for depiction and image interpretation. *Artificial Intelligence*, 41 :125–155, 1989.

[Rob63] L.G. Roberts. Machine perception of three-dimensional solids. Technical Report Technical Report No. 315, MIT Lincoln Laboratory, May 1963.

[Rus97] D.A. Rusling. The linux kernel. Technical Report The Linux Document Project, GNU General Public Licence, 1997. Available on http ://sunsite.unc.edu/mdw/linux.html.

[Sal94] J.M. Salotti. *La gestion des informations dans les premières étapes de la vision par ordinateur.* Thèse de doctorat, Institut National Polytechnique de Grenoble, France, 1994.

[SBG96] M. Salotti, F. Bellet, and C. Garbay. Evaluation of edge detectors : critics and proposal. In W. Furstner H.I. Christensen and Eds C.B Madsen, editors, *ECCV Workshop on Performance Characterization of Vision Algorithms*, pages 81–97, Cambridge UK, April 1996.

[SG92] M. Salotti and C. Garbay. A new paradigm for segmentation. In *Proceedings of the 11th International Conference on Pattern Recognition*, volume 3, pages 611–614, 1992. The Hague.

[SG94] M. Salotti and C. Garbay. Détection de contours : Les heuristiques remplacent avantageusement les modèles. In *Proceedings du 9ème Congrès Reconnaissance des Formes et Intelligence Artificielle (RFIA 94)*, volume 1, pages 721–726, 1994.

[Spi97] C. Spinu. *Une approche Multi-Agents pour la Segmentation d'Images Associant Estimation et Evaluation.* Thèse de doctorat, spécialité informatique, Université Joseph Fourier, Grenoble, 1997.

[Ste90] L. Steels. Cooperation between distributed agents through self-organisation. In Werner and Demazeau, editors, *Decentralized Artificial Intelligence*, volume 1. Elsevier Science, 1990.

[TP82] D. Malah T. Peli. A study of edge detection algorithms. *Computer Graphics and Images Processing*, 20 :1–21, 1982.

[TSY+89] H. Tamura, H. Sato, H. Yamamoto, H. Okazaki, and T. Kawai. The software architecture of an image processing workstation. In *Proceedings of SCIA*, pages 763–769, 1989.

[Vin91] L. Vinet. *Segmentation et mise en correspondance de régions de paires d'images stéréoscopiques.* Thèse de doctorat, Université de Dauphine, Paris IX, 1991.

[WM87] B. Wrobel and O. Monga. Segmentation d'images naturelles : coopération entre un détecteur-contours et un détecteur-région. In *11ème Colloque "Traitement du signal et des images" (GRETSI)*, pages 539–542, 1987. Nice (France).

[XYJH+92] Y. Xiaohan, J. Ylä-Jääski, O. Huttunen, T. Vehkomäki, O. Sipilä, and T. Katila. Image segmentation combining region growing and edge detection. In *Proceedings of the 11th International Conference on Pattern Recognition*, volume 3, pages 481–484, 1992.

[Züc76] S.W. Zücker. Region growing : childhood and adolescence. *Computer Graphics and Images Processing*, 5 :382–399, 1976.

Résumé

Un des objectifs de la vision par ordinateur consiste à extraire à partir d'un important volume de données brutes issues des images, celles qui s'avéreront pertinentes pour une exploitation ultérieure. Les primitives extraites de l'image sont communément de type contour, correspondant à des zones de transition objectivement visibles, et de type région, correspondant à des regroupements de pixels de l'image avec des caractéristiques d'homogénéité communes.

Une nécessaire gestion de l'information est obtenue par la répartition de la tâche de segmentation au sein d'entités indépendantes, localisées de façon précise dans l'image, possédant chacune une primitive particulière à segmenter de type contour ou région, et construisant ces objets de manière incrémentale, c'est-à-dire pixel par pixel.

L'originalité de l'approche réside dans la coopération instaurée entre la construction des contours et des régions. Les deux types de segmenteurs fonctionnent conjointement à l'étiquetage des pixels de l'image, sous une forme pseudo-parallèle, en tirant avantage de leurs atouts réciproques. Un détecteur de contour instancie de nouveaux détecteurs de régions de part et d'autre de son extrémité en construction, afin de valider son existence, tandis qu'un détecteur de régions instancie des détecteurs de contours à sa frontière, afin de borner son expansion. L'ensemble constitue un arbre d'entités de segmentation coopérantes, dépendant chacune les unes des autres, par filiation.

Une telle approche permet une forte adaptation locale, puisque chaque primitive est détectée par une instance d'un détecteur générique, pouvant modifier ses paramètres internes indépendamment des autres instances. La coopération est réelle, puisqu'elle est intégrée au mécanisme de décision.

L'implantation d'un séquenceur de tâches anonymes, permet enfin de simuler le pseudo-parallélisme, et repose grandement sur des mécanismes classiques réservés généralement au domaine des systèmes d'exploitation.