$N^o$ d'ordre : **1651**

# THÈSE

présentée à

## L'UNIVERSITÉ BORDEAUX I

### Ecole doctorale de Mathématiques et Informatique

pour obtenir le grade de

**DOCTEUR en INFORMATIQUE**

par


**Luc Brun**


---

# Segmentation d'images à base Topologique

---

*Soutenue le ../12/96 devant la Commission d'Examen composée de :*

| | |
|---|---|
| R. Cori | Président |
| P. Lienart | Rapporteur |
| A. Montanvert | Rapporteur |
| J.P. Braquelaire | Examinateur |
| A. Gagalowicz | Examinateur |
| J. Shen | Examinateur |

— 1996 —

# Sommaire

# Introduction

Une **image**, dans sa conception la plus vaste, peut se définir comme la représentation planaire d'une scène tridimensionelle ou d'une idée. Cette définition englobe aussi bien les peintures rupestres de Lascau que certaines peintures modernes contemporaines. En informatique les images sont obtenues grâce à la juxtaposition d'un ensemble de points colorés appelés **pixels**. Le niveau de gris ou la couleur d'un pixel sont usuellement représentés par un entier (pour les images en niveaux de gris) ou un triplet d'entiers (pour les images couleurs). Une image informatique est donc représentée grâce à un ensemble de valeurs entières : une telle image est appelée **image numérique**. Dans la suite du présent travail, nous ne considèrerons que des images numériques, ce qualificatif sera donc omis et le terme "image" désignera implicitement une image numérique. Afin de former une image "sans trou", la juxtaposition des pixels doit former un pavage d'une partie du plan. Les pavages les plus connus sont les pavages par triangles, hexagones et carrés (voir figure 0.1). Les pavages hexagonaux ont longtemps été utilisés sur les écrans de télévisions. En informatique, les images numériques utilisent principalement les pavages carrés. Nous nous limiterons donc à ce type de pavage et une image sera considérée comme un tableau rectangulaire composé de pixels carrés.

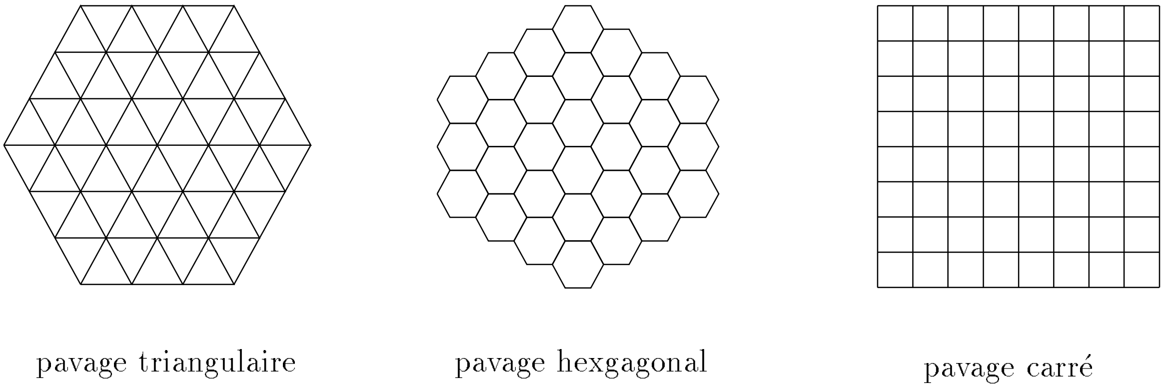



Figure 0.1: *Trois pavages du plan.*

L'apparition de l'informatique et les nouvelles capacités de traitement et de calcul qui en ont découlées ont permis d'automatiser un grand nombre d'opérations effectuées auparavent par des opérateurs humains. L'ensemble des opérations usuellement appliquées à une image sont regroupées sous le terme d'**analyse d'images**; celle-ci comprend l'extraction d'informations pertinentes sur l'image ainsi que le traitement et l'interprétation de ces informations. Dans le cadre d'un traitement informatique de l'analyse d'image on distingue usuellement deux types de traitements. Les traitements de **bas nivau** qui manipulent essentiellement les valeurs numériques des pixels et les traitements de **haut niveaux** qui manipulent des symboles. Partant d'une photographie satellite un traitement de haut niveau regroupera certains pixels de l'image en leurs affectant une étiquette identique (route, ferme, parking, etc.). Le lien entre les traitements de bas et haut niveaux est réalisé grâce à une étape fondamentale de l'analyse d'image appelée l'étape de **segmentation**.

Le but de la segmentation est de partitionner l'image en un ensemble de groupes de pixels connexes et homogènes. Un tel groupe de pixels est appelé une **région** de l'image. L'homogénéité est généralement calculée à partir des valeurs des pixels. On peut, par exemple, regrouper tous les pixels dont la distance est inférieure à un seuil. Le postulat sous-jacent de cette étape est que des groupes de pixels homogènes appartiennent au même objet. On espère donc à partir des régions obtenir les **objets** ou des parties des objets présents dans l'image. La segmentation idéale est bien sûr celle qui crée une région pour chaque objet.

Cette segmentation "idéale" est un objectif très difficile et certainement impossible à atteindre. Par exemple si l'on segmente une image représentant une chaise, doit-on avoir une seule région représentant la chaise ou plusieurs régions décomposant la chaise avec ses pieds, son dossier, etc. ? La réponse à une telle question varie en fonction de l'utilisateur et du domaine concerné. Faute d'une définition mathématique précise du terme d'objet de nombreux chercheurs ont développé des algorithmes de segmentation restreints à des domaines très précis. Par exemple, lors du développement d'un algorithme de segmentation d'images satellites on définira précisément le concept de route et l'algorithme sera construit de façon à regouper les pixels correspondant à cette définition. Afin d'élargir le domaine des algorithmes de segmentation, on les munit souvent d'un ensemble de paramètres permettant de modifier légèrement leur comportement et donc de les adapter à des types d'images légèrements différents. Toutefois, l'adaptibilité induite par ce type de procédé reste faible et les paramètres doivent souvent être adaptés à chaque image.

Les premières images traitées par ordinateur ont longtemps été des images en niveau de gris. L'évolution du matériel (augmentation de la place mémoire disponible, baisse du prix des scanners couleurs, etc.) n'a permis que récemment de manipuler "confortablement" des images couleurs. Avec l'appartition de ce nouveau type d'image de nombreux chercheurs ont tenté d'adapter leurs travaux, réalisés dans le cadre d'images en niveaux de gris, aux images couleurs. Malheureusement, une couleur est alors souvent considérée comme la juxtaposition de trois composantes plutôt que comme une entité à part entière. En particulier, les algorithmes de segmentation de ce type tiennent rarement compte de la vision humaine des couleurs et manipulent l'image comme un signal sans tenir compte de la perception que nous en avons.

Le présent travail fut initié en collaboration avec l'entreprise Archéoscopie. Cette entreprise disposait de nombreuses photos de peintures murales. Le traitement de ces images par un archéologue comprenant une étape de segmentation, cette entreprise s'est adressée à nous pour automatiser ou du moins faciliter ce travail fastidieux. L'objectif de la thèse était de présenter un logiciel résolvant au moins en partie les problèmes cités dans les paragraphes précédents. Ce logiciel devait donc :

- Traiter l'information couleur non comme la somme de trois composantes mais comme une information à part entière.

- Créer des méthodes suffisamment générales pour pouvoir s'appliquer sur tout type d'image.
- Permettre une segmentation suffisamment souple pour pouvoir guider le processus manuellement ou insérer facilement des connaissances de l'expert.

Bien que la société Archéoscopie ait déposé son bilan au début de ma thèse nous avons exploré le domaine de la couleur et de la segmentation d'images en gardant ces trois objectifs. De fait, hors du cadre de l'archéologie, ces trois critères s'ils sont respectés permettent à tout utilisateur d'obtenir rapidement la segmentation qu'il désire et ce quel que soit le type d'image.

Nous allons présenter la démarche suivie pour atteindre ces objectifs en décomposant l'exposé en deux grandes parties.

La première partie a pour objet l'étude des ensembles de couleurs. Cette partie est composée de trois chapitres :

- Le chapitre 1 étudie les ensembles de données pondérées, ensembles qui apparaissent naturellement lorsque l'on étudie l'ensemble des couleurs d'une image.
- Le chapitre 2 permet d'approfondir la notion de couleur et d'étudier différents modèles utilisés pour représenter une couleur.
- Le chapitre 3 étudie différentes méthodes permettant d'extraire les principales couleurs d'une images.Ces méthodes sont usuellement appelées des méthodes de **quantification**.

La seconde partie est consacrée à la segmentation proprement dite. Elle se compose de 5 chapitres.

- Le chapitre 4 permet de définir plus précisément les relations entre les pixels. Il permet, en particulier de présiser la notion d'adjacence entre pixels. Ceci permet de définir plus formellement la notion de région.
- Le chapitre 5 permet d'acquérir une vision d'emsemble de la plupart des méthodes de segmentation. Nous préciserons en particulier la problématique posée par la segmentation et les avantages et inconvénients des différentes méthodes. Nous étudierons en outre les différentes structures de données utilisées en segmentation et l'incidence de celles-ci sur les méthodes de segmentation.
- Le chapitre 6 décrit la structure de données que nous avons utilisée pour nos algorithmes de segmentation. Nous décrivons également dans ce chapitre un ensemble de primitives permettant de modifier cette structure de données.
- Finalement, le chapitre 7 décrit un ensemble de méthodes de segmentation basées sur notre modèle de contours.

# Partie I

# Partition d'ensembles de couleurs

# Chapitre 1

# La couleur

## 1.1 Définition de la couleur

L'étude de la couleur est très différente de celle de la lumière. Cet état de fait est résumé par Kandinsky lorsqu'il déclare : *"La couleur est la touche. L'oeil est le marteau. L'âme est le piano aux cordes nombreuses. ...Il est donc clair que l'harmonie des couleurs doit reposer uniquement sur le principe de l'entrée en contact efficace avec l'âme humaine"*. En effet, si l'impression de couleur est bien provoquée par la lumière, phénomène physique mesurable, ce signal est traité et transformé par notre oeil et notre cerveau pour aboutir à la sensation finale de couleur. L'ensemble des transformations subies par le signal initial est si complexe que les mesures de l'énergie lumineuse rentrant dans l'oeil ne rendent que très imparfaitement compte de la perception colorée. Il est donc important de saisir que toute mesure de la couleur ne permet que d'approximer la vision humaine. Il est ainsi parfois inutile de complexifier un algorithme, si l'accroissement de ses performances n'est pas perceptible visuellement.

Les études menées en neuro-sciences et en colorimétrie permettent d'apréhender plusieurs points importants. Le signal lumineux qui excite l'oeil peut être considéré comme un mélange d'ondes électromagnétiques sinusoïdales se propageant toutes à la vitesse de la lumière c ($c \approx 3\ 10^8 m/s$). Ces ondes ou composantes spectrales peuvent être caractérisées par leurs longueur d'onde $\lambda$ et leurs puissance $P_\lambda$. Un signal lumineux peut donc être décrit par une fonction $P(\lambda)$ qui associe à chaque longueur d'onde la puissance du signal associé. Ce signal lumineux excite des récepteurs photo-sensibles situés sur la rétine.

Les cellules photo-sensibles sont composées de cônes et bâtonnets. Les bâtonnets sont sensibles à de faibles niveaux de lumière mais n'atteignent le maximum de leur puissance qu'à des intensités lumineuses modérées. Les bâtonnets permettent une vision monochromatique ; ce sont eux qui interviennent lors de la vision nocturne. Les cônes ont une sensibilité assez faible et sont responsables de la vision diurne. Les cônes se répartissent en trois familles de sensibilités spectrales différentes (L pour *Long*, M pour *Middle* et S pour *Short*). Chaque type de cône est donc sensible à une gamme de longueur d'onde particulière et est associé à la perception d'une couleur. Ainsi, si l'oeil reçoit une lumière de faible longueur d'onde (aux alentours de 450 nanomètres), donc si l'on excite essentiellement les cônes S, le sujet percevra une couleur bleue. De même si l'oeil reçoit une lumière de longueur d'onde moyenne (entre 500 et 600 nm), il percevra une couleur verte. Enfin les longues longueurs d'ondes (entre 600 et 700 nm) correspondent à la perception du rouge. La combinaison des signaux émis par ces trois types de cônes permet la vision colorée. Cette décomposition du spectre lumineux en trois composantes est à la base de la colorimétrie. Le but de la colorimétrie est de décrire un ensemble de couleurs grâce à plusieurs (généralement trois) composantes réelles. Ces composantes sont habituellement déduite de la décomposition spectrale de la lumière. Nous allons dans la section suivante décrire brièvement les principaux espaces de couleur utilisés en quantification et segmentation.

## 1.2 Les principaux espaces de couleur

Un espace de couleur est établi en fixant trois couleurs (A), (B), (C) appelées couleurs primaires. Soit $(1_\lambda)$ une couleur correspondant à un signal mono-chromatique de longueur d'onde $\lambda$ et de puissance 1 Watt. Les lois de Grassman [DG80] nous indiquent que l'on peut obtenir la même sensation visuelle que celle obtenue avec la couleur $1_\lambda$ en superposant les couleurs (A), (B), (C) avec des coefficients $a_\lambda$, $b_\lambda$ et $c_\lambda$. On a alors :

$$(1_\lambda) = a_\lambda.(A) + b_\lambda.(B) + c_\lambda.(C)$$

Soit à présent une couleur (D) de décomposition spectrale $P_\lambda$. La couleur (D) peut être vue comme une somme de couleurs $(1_\lambda)$ pondérée par les coefficients $P(\lambda)$. Plus précisément on a :

$$(D) = \int_{400}^{700} P(\lambda)(1_\lambda)d\lambda$$

On a donc :

$$\begin{aligned} (D) &= \textstyle\int_{400}^{700} P(\lambda)(a_\lambda.(A) + b_\lambda.(B) + c_\lambda.(C))d\lambda \\ &= \left(\textstyle\int_{400}^{700} P(\lambda)a_\lambda d\lambda\right)(A) + \left(\textstyle\int_{400}^{700} P(\lambda)b_\lambda d\lambda\right)(B) + \left(\textstyle\int_{400}^{700} P(\lambda)c_\lambda d\lambda\right)(C) \end{aligned}$$

On pose alors :

$$\begin{aligned} A &= \int_{400}^{700} P(\lambda)a_\lambda \\ B &= \int_{400}^{700} P(\lambda)b_\lambda \\ C &= \int_{400}^{700} P(\lambda)c_\lambda \end{aligned}$$

On obtient donc :

$$(D) = A.(A) + B.(B) + C.(C)$$

La couleur (D) est obtenue grâce à la somme des couleurs (A), (B) et (C) pondérée par les coefficients A, B et C. Si les trois coefficients A, B, C sont positifs on parlera de synthèse additive. Inversement si l'un des trois coefficients est négatif, on parlera de synthèse soustractive. La synthèse soustractive est utilisée en peinture ou en imprimerie. L'informatique graphique utilise quant à elle la synthèse additive. Nous nous limiterons donc à cette dernière. Les lois de Grassman permettent de combiner les coefficients de deux couleurs pour obtenir une troisième couleur. L'ensemble des opérations définies par ces lois permet de définir un espace vectoriel réel de base $\{(A),(B),(C)\}$. Une première façon de définir un espace de couleur consiste donc à choisir les trois couleurs primaires qui formeront la base de l'espace vectoriel. On peut également creer un espace de couleur en définissant une matrice de changement de base definie sur un espace de couleur prédéfini.

### 1.2.1 Le modèle RGB

Le modèle de couleur sans doute le plus connu en informatique graphique est le modèle $RGB$. Ce modèle est basé sur le fait que l'on peut reconstituer une large gamme de couleurs selon une synthèse additive de trois primaires : le rouge (R) (pour Red), le vert (G) (pour Green) et le bleu (B) (pour Blue).

L'intérêt d'une telle démarche est qu'elle modélise ne serait-ce qu'approximativement le système visuel de perception des couleurs. On peut en effet associer la composante $R$ aux grandes longueurs d'ondes, la composante $G$ aux moyennes et la composante $B$ aux courtes longueurs d'onde.

Ce modèle a été adopté dès 1931 par la Commission internationale de l'éclairage (CIE) afin de caractériser la réponse visuelle la plus commune aux observateurs, d'où la définition d'un observateur de référence appelé l'observateur de référence colorimétrique CIE 1931. Cette notion d'observateur de référence permet d'établir une base commune à toute recherche. On a pu ainsi définir quelles étaient les courbes $r_\lambda$, $g_\lambda$ et $b_\lambda$ de mélange des couleurs primaires physiques pour cet observateur standard (voir Figure 1.1). Au vu de

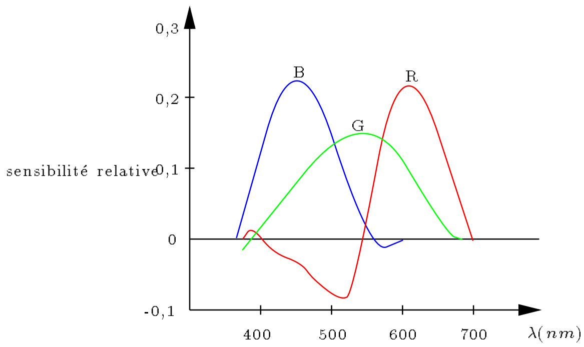



Figure 1.1: *Courbes de mélange des couleurs primaires RGB pour l'observateur standard CIE 1931.*

ces courbes il apparaît que certaines couleurs, en particulier les bleu-vert (450-550 nm), ne peuvent être reproduites par synthèse additive. En effet pour obtenir la couleur correspondant à une lumière mono-chromatique de 500 nm il faut ajouter des lumières bleu et verte et retrancher une lumière rouge. Ceci sort donc du cadre de la synthèse additive.

### 1.2.2 Le modèle XYZ

Pour pallier à cet inconvénient, la CIE a établi en 1931 un espace colorimétrique basé sur trois nouvelles primaires (X), (Y) et (Z). L'espace de couleur induit par ces trois primaires permet de représenter toutes les couleurs par synthèse additive. Si une couleur C est représentée dans l'espace $RGB$ par :

$$(C) = R.(R) + G.(G) + B.(B)$$

où $(R), (G), (B)$ représentent les trois primaires, alors cette couleur sera représentée dans l'espace $XYZ$ par :

$$(C) = X.(X) + Y.(Y) + Z.(Z)$$

avec :

$$\begin{pmatrix} X \\ Y \\ Z \end{pmatrix} = \begin{pmatrix} 0.166 & 0.125 & 0.093 \\ 0.060 & 0.327 & 0.005 \\ 0.000 & 0.004 & 0.46 \end{pmatrix} * \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

Le système $XYZ$ est peu utilisé en tant que tel et sert souvent d'interface entre le système $RGB$ et des systèmes plus évolués.

Partant du système $XYZ$ de nombreux travaux de recherche ont tenté d'établir une métrique uniforme des différences de couleur perçues. Intuitivement, définir une métrique uniforme consiste à définir une distance $d$ telle que si $d(C_1, C_2) = d(C_1, C_3)$, les couleurs $C_2$ et $C_3$ *paraissent* à la même distance de la couleur $C_1$. De même si $d(C_1, C_2) = 2d(C_1, C_3)$ la couleur $C_3$ doit *paraître* deux fois plus éloignée de $C_1$ que la couleur $C_2$. Il est rapidement apparu qu'une telle distance ne pouvait être définie dans l'espace $XYZ$(voir [Wri41]). Pour remédier à cet inconvénient de nombreuses recherches ont été menées pour établir un nouveau système de représentation qui cette fois serait uniforme. Ces recherches n'ont malheureusement pas abouti. Alain Trémeau [Tre93] a montré que la vision des couleurs obéissait plus à une géométrie Riemannienne qu'euclidienne. Or plonger un espace de Rieman dans un espace euclidien nécessite de passer de la dimension $n$ à la dimension $m$ avec :

$$m = \frac{n(n+1)}{2}$$

L'espace de couleur étant de dimension 3 on ne peut donc espérer obtenir une distance euclidienne uniforme qu'en passant en dimension 6. On peut toutefois construire des espaces de dimension 3 approximativement uniformes tel que les espaces $L^*u^*v^*$ et $L^*a^*b^*$.

### 1.2.3 Les modèles $L^*u^*v^*$ et $L^*a^*b^*$

L'espace $L^*u^*v^*$ se déduit de l'espace $XYZ$ à l'aide de la transformation suivante :

$$L^* = \begin{cases} 116\left(\frac{Y}{Y_w}\right)^{\frac{1}{3}} - 16 & \text{si } \frac{Y}{Y_w} > 0.01 \\ 903.3\frac{Y}{Y_w} & \text{si } \frac{Y}{Y_w} \leq 0.01 \end{cases}$$

$L^*$ représente la luminosité. Les coordonnées chromatiques $u^*$ et $v^*$ sont définies par les équations suivantes.

$$u^* = 13L^*(u' - u'_w) \text{ avec } u' = \frac{4X}{X+15Y+3Z}$$
$$v^* = 13L^*(v' - v'_w) \text{ avec } v' = \frac{9Y}{X+15Y+3Z}$$

Les coordonnées $X_w, Y_w, Z_w, u'_w$ et $v'_w$ sont celles du blanc de référence noté W.

La coordonnée $L^*$ du modèle $L^*a^*b^*$ est définie de la même façon que dans le modèle $L^*u^*v^*$ les deux autres variables sont définies par :

$$a^* = 500\left[f\left(\frac{X}{X_w}\right) - f\left(\frac{Y}{Y_w}\right)\right]$$
$$b^* = 500\left[f\left(\frac{Y}{Y_w}\right) - f\left(\frac{Z}{Z_w}\right)\right]$$

avec :

$$f(r) = \begin{cases} r^{\frac{1}{3}} & \text{si } r \geq 0.008856 \\ 7.787r + \frac{16}{116} & \text{si } r \leq 0.008856 \end{cases}$$

La distance entre deux couleurs $(C_1)$ et $(C_2)$ de coordonnées $(L_1^*, u_1^*, v_1^*)$, $(L_2^*, u_2^*, v_2^*)$ dans le système $L^*u^*v^*$ et $(L_1^*, a_1^*, b_1^*)$, $(L_2^*, a_2^*, b_2^*)$ dans le système $L^*a^*b^*$est alors définie comme la distance euclidienne :

$$d(C_1, C_2) = \sqrt{(L_1^* - L_2^*)^2 + (u_1^* - u_2^*)^2 + (v_1^* - v_2^*)^2}$$

dans le système $L^*u^*v^*$ et

$$d(C_1, C_2) = \sqrt{(L_1^* - L_2^*)^2 + (a_1^* - a_2^*)^2 + (b_1^* - b_2^*)^2}$$

dans le système $L^*a^*b^*$.

Suivant une étude menée sur le sujet par Pointer [Poi81], il semble qu'aucun des deux espaces CIE $L^*a^*b^*$et $L^*u^*v^*$ ne soit plus uniforme que l'autre (voir section 1.2.2). Nous avons donc arbitrairement choisi l'espace $L^*u^*v^*$ légèrement plus simple à implémenter. La conversion entre les espaces $RGB$ et $L^*u^*v^*$ impose de passer par l'espace $XYZ$, de calculer une racine cubique et d'effectuer plusieurs divisions. Cette transformation implique donc souvent un surcoût de calcul non négligeable.

### 1.2.4 Le modèle $I_1I_2I_3$

L'espace $I_1I_2I_3$ introduit par Otha et al. [OKS80] répond à une approche totalement différente. Otha, Kanade et Sakai ont cherché l'espace de couleurs présentant le plus d'intérêt pour la segmentation d'images. Ils ont constaté que l'on obtenait de bons résultats en utilisant l'espace de couleur défini par les trois axes de plus grande variance de l'ensemble de couleurs associé à l'image. Un résultat bien connu en analyse de données établit que ces axes correspondent aux vecteurs propres de la matrice de covariance de l'ensemble de couleurs associé à l'image. Ces trois axes sont représentés sur la figure 1.2.

Les tests établis par Otha et confirmés par nos propres expériences (voir Table 1.1 ) montrent que les vecteurs propres d'une image "naturelle" s'écartent très peu de trois directions constantes. Le terme "image naturelle" s'oppose ici à "image de synthèse". On désignera par "image naturelle" une image habituellement perçue par l'oeil.

On constate sur la Table 1.1 que le vecteur propre de plus grande valeur propre $V_1$ peut être approximé par le vecteur $(\frac{1}{3}, \frac{1}{3}, \frac{1}{3})$. Ce vecteur définit un axe vectoriel correspondant à la luminosité. Le second vecteur propre $V_2$ peut quant à lui être approximé par $(\frac{1}{2}, 0, -\frac{1}{2})$. Ceci correspond à l'opposition rouge-bleu. Le dernier vecteur propre $V_3$ peut

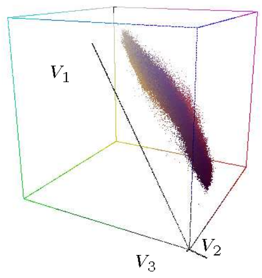



Figure 1.2: *Multi-ensemble associé à l'image Lenna. Les vecteurs propres sont notés $V_1$, $V_2$ et $V_3$ dans l'ordre décroissant de leurs valeurs propres. La longueur des vecteurs est proportionelle à leurs valeur propre.*

| | $V_1$ | | | $V_2$ | | | $V_3$ | | |
|---|---|---|---|---|---|---|---|---|---|
| | R | G | B | R | G | B | R | G | B |
| Zelda | 0.38 | 0.33 | 0.27 | 0.44 | 0.25 | -0.30 | -0.06 | 0.45 | -0.47 |
| Lenna | 0.35 | 0.40 | 0.23 | 0.44 | 0.16 | -0.39 | -0.20 | 0.40 | -0.39 |
| Fleurs | 0.41 | 0.38 | 0.20 | 0.32 | 0.01 | -0.67 | -0.36 | 0.47 | -0.16 |
| Anemone | 0.31 | 0.37 | 0.31 | 0.49 | 0.01 | -0.48 | -0.26 | 0.45 | -0.28 |
| Mandrill | 0.18 | 0.34 | 0.47 | 0.68 | -0.02 | -0.28 | -0.15 | 0.52 | -0.32 |
| Moyennes | 0.33 | 0.36 | 0.3 | 0.47 | 0.08 | -0.42 | -0.21 | 0.46 | -0.32 |
| $\sigma$ | 0.09 | 0.03 | 0.11 | 0.13 | 0.12 | 0.16 | 0.11 | 0.04 | 0.12 |

Table 1.1: Les vecteurs propres $V_1$, $V_2$ et $V_3$ sont ordonnés dans l'ordre décroissant de leurs valeurs propres. Leurs coordonnées sont exprimées dans l'espace RGB pour chacune des images tests Lenna, Zelda, Fleur, Anemone et Mandrill. Les moyenne et les écart types calculés sur chacune des coordonnées sont affichés sur la dernière et l'avant dernière ligne.

être approximé par : $(-\frac{1}{4}, \frac{1}{2}, -\frac{1}{4})$ ce qui correspond à l'opposition vert-violet. Ces trois vecteurs fournissent une nouvelle base permettant de définir un nouvel espace de couleur déduit de l'espace $RGB$ par la transformation suivante :

$$\begin{cases} I_1 & = \frac{R+G+B}{3} \\ I_2 & = R - B \\ I_3 & = \frac{2G-(R+B)}{2} \end{cases}$$

Étant donnés une image naturelle et son multi-ensemble associé, les axes $I_1$ $I_2$ et $I_3$ sont par construction proches des vecteurs propres de la matrice de covariance du multi-ensemble. Ceci a deux conséquences intéressantes en analyse d'image.

- Les variances des axes $I_1$, $I_2$ et $I_3$ seront importantes. En terme d'analyse de données un axe de forte variance correspond à un axe contenant beaucoup d'information.
- Les covariances entre les axes $I_1$, $I_2$ et $I_3$ seront faibles. Ceci signifie que chaque axe contient un seul type d'information. Par exemple l'axe $R$ du système $RGB$ contient une information de luminance (l'intensité du rouge) et une information de chrominance. Cette double information luminance/chrominance est partagée entre l'axe $I_1$ qui code la luminance et les axes $I_2$ et $I_3$ codant la chrominance (Voir [SB85] pour plus de détails sur ce point).

Le système $I_1I_2I_3$ contient donc un axe représentant la luminosité et deux axes représentant la chromaticité. Remarquons que les axes $I_2$ et $I_3$ ont été multipliés par un facteur 2. Ceci permet de renforcer l'importance de la chromaticité par rapport à la luminosité afin d'être plus en adéquation avec la vision humaine.

### 1.2.5 Le modèle $H_1H_2H_3$

Le modèle $I_1I_2I_3$ permet d'obtenir des axes de coordonnées proches des trois axes de plus grande variance d'une image naturelle. Cet espace permet donc d'expliciter trois axes importants pour ce type d'images. Ces axes ne correspondent toutefois pas forcément aux directions privilégiées de la vision humaine. Les travaux de Levine [Lev85] confirmés par de récentes découvertes en neuro-science(voir [LIF95]) montrent que les signaux émis par les cônes L, M et S sont recombinés dans le cerveau pour former trois signaux représentant les opposition “rouge/vert”, “bleu/jaune” et “blanc/noir”( voir figure 1.3).

Ces directions privilégiées sont mises en évidence dans l'espace $H_1H_2H_3$ déduit de l'espace $RGB$ par la transformation suivante :

$$\begin{cases} H_1 = R + G \\ H_2 = R - G \\ H_3 = B - \frac{R+G}{2} \end{cases}$$

Calculer les distances dans l'espace $H_1H_2H_3$ permet donc d'utiliser trois axes privilégiés de la vision humaine. De plus, la conversion entre les espaces $RGB$ et $H_1H_2H_3$ définie par le système ci-dessus est extrêmement simple et peut donc être calculée efficacement.

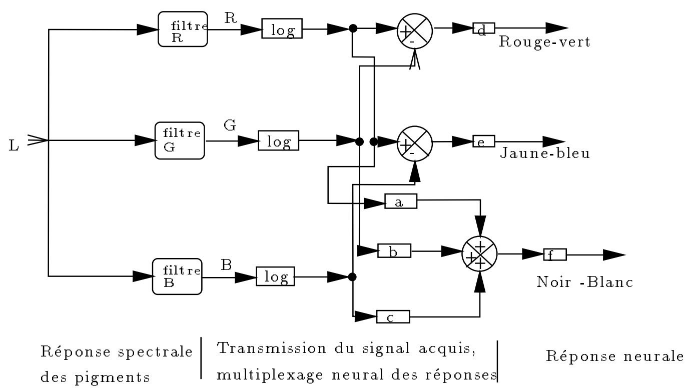



Figure 1.3: *Modèle de Lévine de la vision Humaine.*

### 1.2.6 Le modèle $YIQ$

Le modèle $YIQ$ est une variante du modèle $RGB$ établie par le NTSC[1] pour rendre plus efficace la transmission des signaux de télévision et la compatibilité avec les écrans noir et blanc. La composante $Y$ contient l'information concernant la luminosité de l'image. L'information chromatique est quant à elle codée par les axes $I$ et $Q$ correspondant respectivement aux oppositions cyan-orange et magenta-bleu. La relation entre les modèles $YIQ$ et $RGB$ est la suivante :

$$\begin{pmatrix} Y \\ I \\ Q \end{pmatrix} = \begin{pmatrix} 0.30 & 0.59 & 0.11 \\ 0.60 & -0.27 & -0.32 \\ 0.11 & -0.52 & 0.31 \end{pmatrix} * \begin{pmatrix} R \\ G \\ B \end{pmatrix}$$

## 1.3 Conclusion

Chacun des espace de couleur que nous venons de voir possède un ensemble d'avantages et d'inconvénients. Il est impossible de privilégier un espace de couleur et le choix d'un espace dépend des contraintes de la méthode envisagée. Par exemple, une application privilégiant la quantité d'information contenue dans chaque axe de coordonnée choisira des espaces tels que $YIQ$ ou $I_1I_2I_3$. Une application privilégiant l'adéquation entre l'espace utilisé et la vision humaine choisira plutôt les espaces $L^*u^*v^*$, $L^*a^*b^*$ ou $H_1H_2H_3$. Enfin, l'espace $RGB$ peut être choisi si l'application ne possède pas de contraintes particulières ou si le temps de calcul est une contrainte importante de celle-ci.

[1]National Television Standards Committee

# Chapitre 2

# L'erreur quadratique

Une image étant vue comme un tableau de valeurs, elle peut être abordée selon deux aspects : en considérant la répartition spatiale de l'ensemble des valeurs (telle valeur est située à côté de telle autre dans le tableau) ou en considérant l'ensemble des valeurs sans considération spatiale. Dans ce chapitre, nous allons étudier un ensemble de concepts et de théorèmes permettant d'associer un ensemble de mesures à un ensemble de valeurs. En particulier nous allons introduire la notion d'**erreur**. L'erreur associée à un ensemble de valeurs permet de mesurer son homogénéité. Une image dont toutes les valeurs sont identiques possèdera une erreur nulle et sera donc très homogène. Inversement une image composée d'une multitude de valeurs très différentes possèdera une erreur élevée, ce qui signifiera que l'image est très hétérogène.

## 2.1 Multi-ensemble

De façon très générale un multi-ensemble est un élément de $\mathcal{P}(\mathbb{R}^n) \times \mathcal{F}(\mathbb{R}^n, \mathbb{R})$, où $\mathcal{P}(\mathbb{R}^n)$ désigne l'ensemble des parties de $\mathbb{R}^n$ et $\mathcal{F}(\mathbb{R}^n, \mathbb{R})$ l'ensemble des applications de $\mathbb{R}^n$ dans $\mathbb{R}$. Un multi-ensemble est donc un couple $(C, f)$ où $C$ est un sous-ensemble de $\mathbb{R}^n$ et $f$ une application de $\mathbb{R}^n$ dans $\mathbb{R}$. Cette définition étant un peu trop générale pour nos besoins, nous avons adopté la définition suivante :

**Définition 1** *Soit $\mathcal{PB}(\mathbb{R}^n)$ l'ensemble des parties bornées de $\mathbb{R}^n$. Nous définissons $\mathcal{ME}_n$ l'ensemble des* **multi-ensembles** *de $\mathbb{R}^n$, comme suit :*

$$\mathcal{ME}_n = \{(C, f) \in \mathcal{PB}(\mathbb{R}^n) \times \mathcal{F}(\mathbb{R}^n, \mathbb{R}_+) / f_{|\mathbb{R}^n - C} == 0\}$$

En d'autres termes, un multi-ensemble est la donnée d'un ensemble borné et d'une application réelle positive nulle hors de cet ensemble. Sauf mention contraire, tous les ensembles considérés dans ce chapitre seront des ensembles discrets. D'un point de vue informatique, les multi-ensembles peuvent être vus comme une extension de la notion d'histogramme à des espaces de dimension $n$. La fonction associée à un ensemble est souvent appelée la **fonction de fréquence** ou de **distribution**. L'application de ce concept à l'image est assez immédiate. En effet, si nous considérons une image $I$, on peut y associer un ensemble

$C$, égal à l'ensemble de valeurs $v$ associées aux pixels de l'image. Nous pouvons alors définir $f(v)$ comme le nombre de fois où le n-uplet $(v_1, \ldots, v_n)$ apparaît dans l'image. Par exemple, dans le cas d'une image couleur, $f(0,0,0)$ représente le nombre de pixels noirs contenus dans l'image. La fonction de fréquence associe un poids à chaque élément du multi-ensemble. Ces poids peuvent être utilisés pour définir les moments, la moyenne ou la variance d'un multi-ensemble.

**Définition 2** *Soit $\mathcal{ME}_n$ l'ensemble des multi-ensembles de $\mathbb{R}^n$. On définit sur $\mathcal{ME}_n$ les fonctions $\boldsymbol{M}_0$, $\boldsymbol{M}_1$ et $\boldsymbol{M}_2$ appelées* **moments** *d'ordre 0, 1 et 2 par :*

$$\boldsymbol{M}_0 \left( \begin{array}{ccl} \mathcal{ME}_n & \rightarrow & \mathbb{R} \\ (C,f) & \mapsto & \sum_{v \in C} f(v) \end{array} \right.$$

$$\boldsymbol{M}_1 \left( \begin{array}{ccl} \mathcal{ME}_n & \rightarrow & \mathbb{R}^n \\ (C,f) & \mapsto & \sum_{v \in C} f(v)(v_1, \ldots, v_n) \end{array} \right.$$

$$\boldsymbol{M}_2 \left( \begin{array}{ccl} \mathcal{ME}_n & \rightarrow & \mathbb{R}^n \\ (C,f) & \mapsto & \sum_{v \in C} f(v)(v_1^2, \ldots, v_n^2) \end{array} \right.$$

*où $(v_1, \ldots, v_n)$ sont les $n$ coordonnées du vecteur $v$.*

Il est bien sûr possible de définir des moments d'ordres supérieurs, mais les ordres 0, 1 et 2 seront suffisants pour notre étude. Afin d'alléger les notations on omettra généralement la fonction de fréquence lors du calcul de l'image d'un multi-ensemble. L'image de $(C,f)$ par $\boldsymbol{M}_0$, $\boldsymbol{M}_0((C,f))$ sera donc notée plus simplement $\boldsymbol{M}_0(C)$. On peut remarquer que $\boldsymbol{M}_0(\mathrm{C})$ également noté $|C|$ représente un scalaire alors que $\boldsymbol{M}_1(\mathrm{C})$ et $\boldsymbol{M}_2(\mathrm{C})$ sont des vecteurs de $\mathbb{R}^n$ où $n$ est la dimension de l'espace dans lequel est plongé l'ensemble $C$. Usuellement cette dimension est égale à 1 pour les images en niveaux de gris et 3 pour les images couleurs. À partir des moments, on définit la moyenne et les variances d'un multi-ensemble de la façon suivante :

**Définition 3** *Avec les notations précédentes, les fonctions* **moyenne** *et* **variance** *sont définies sur $\mathcal{ME}_n$ par :*

$$\mu \left( \begin{array}{ccl} \mathcal{ME}_n & \rightarrow & \mathbb{R}^n \\ (C,f) & \mapsto & \dfrac{M_1(C)}{M_0(C)} \end{array} \right.$$

$$\forall i \in \{1, \ldots n\}, \quad var_i \left( \begin{array}{ccl} \mathcal{ME}_n & \rightarrow & \mathbb{R}_+ \\ (C,f) & \mapsto & \dfrac{\boldsymbol{M}_2(C)_i}{\boldsymbol{M}_0(C)} - \mu^2(C)_i \end{array} \right.$$

*où $\boldsymbol{M}_2(C)_i$ et $\mu(C)_i$ représentent respectivement la $i^{eme}$ coordonnée de $\boldsymbol{M}_2(C)$ et $\mu(C)$.*

La fonction $var_i$ nous permet de mesurer la variance d'un multi-ensemble le long d'un axe et peut servir à mesurer l'homogénéité d'un multi-ensemble. La mesure d'homogénéité que nous allons utiliser, appelée **erreur quadratique**, peut être vue comme une extension de la variance pour les dimensions supérieures à 1.

**Définition 4** *L'***erreur quadratique** *d'un multi-ensemble* $(C, f)$ *notée* $\boldsymbol{SE}(C)$ *est donnée par :*

$$\boldsymbol{SE}(C) = \sum_{v \in C} f(v) \|v - \mu(C)\|^2$$

L'erreur quadratique d'un multi-ensemble $(C, f)$ est donc une mesure de l'écart des éléments de $C$ par rapport à la moyenne $\mu(C)$. Si nous appliquons ce concept à l'image, la moyenne $\mu(C)$ représente le niveau de gris moyen (ou la couleur moyenne) de l'image et $\boldsymbol{SE}(C)$ représente l'écart de l'ensemble des couleurs de l'image par rapport à cette moyenne.

Des calculs relativement simples montrent que l'erreur quadratique peut s'exprimer en fonction des moments et des variances. De fait nous avons, pour un multi-ensemble $(C, f)$ donné :

$$\boldsymbol{SE}(C) = \sum_{i=1}^{n} \boldsymbol{M}_2(C)_i - \frac{\boldsymbol{M}_1(C)_i^2}{\boldsymbol{M}_0(C)} = \boldsymbol{M}_0(C) \sum_{i=1}^{n} var_i(C) \tag{2.1}$$

L'erreur quadratique d'un multi-ensemble $(C, f)$ peut donc également être vue comme la somme de ses variances pondérée par le cardinal de $(C, f)$. Cette pondération permet de tenir compte du fait qu'une même erreur est a priori plus importante si elle se produit sur un grand ensemble que sur un petit.

Jusqu'à présent nous avons vu la définition d'un multi-ensemble et les différentes mesures que l'on peut associer à celui-ci. Les multi-ensembles peuvent également être combinés pour créer de nouveaux multi-ensembles.

**Définition 5** *Soit* $(C_1, f_1)$ *et* $(C_2, f_2)$ *deux éléments de* $\mathcal{ME}_n$. *La somme, l'union, l'intersection et la différence de* $(C_1, f_1)$ *et* $(C_2, f_2)$ *sont définies comme suit:*

$$\begin{aligned}
(C_1, f_1) + (C_2, f_2) &= (C_1 \cup C_2, f_1 + f_2) \\
(C_1, f_1) \cup (C_2, f_2) &= (C_1 \cup C_2, \max(f_1, f_2)) \\
(C_1, f_1) \cap (C_2, f_2) &= (C_1 \cap C_2, \min(f_1, f_2)) \\
(C_1, f_1) - (C_2, f_2) &= (C_1 - C_2, \max(0, f_1 - f_2))
\end{aligned}$$

La combinaison de plusieurs multi-ensembles prend tout son intérêt en segmentation lorsque l'on manipule une image partitionnée en régions. Supposons par exemple que l'on désire fusionner deux régions $R_1$ et $R_2$ adjacentes dans l'image. Les multi-ensembles $(C_1, f_1)$ et $(C_2, f_2)$ correspondant à $R_1$ et $R_2$ ne sont a priori pas distincts. Le multi-ensemble correspondant à la fusion des régions $R_1$ et $R_2$ est alors égal à la somme des multi-ensembles $(C_1, f_1)$ et $(C_2, f_2)$. Inversement, si on désire partitionner une région $R$ en deux sous-régions $R_1$ et $R_2$, nous avons la relation : $(C, f) - (C_1, f_1) = (C_2, f_2)$ où $(C, f)$ désigne le multi-ensemble associé à $R$. Le problème est alors d'évaluer l'erreur quadratique de l'union ou de la différence de deux multi-ensembles. Un premier pas vers l'évaluation de l'erreur quadratique est donné par la proposition suivante:

**Proposition 1** *Soient* $\{(C_1, f_1), \ldots, (C_p, f_p)\}$ *un ensemble de multi-ensembles de* $\mathcal{ME}_n$, *et* $C = \sum_{i=1}^{p}(C_i, f_i)$. *On a :*

$$\forall i \in \{0,1,2\} \quad \boldsymbol{M}_i(C) = \sum_{j=1}^{p} \boldsymbol{M}_i(C_j)$$

**Preuve:**

Nous allons établir la propriété pour le cas $p = 2$, le cas général s'en déduisant trivialement par récurrence. Afin de simplifier, on notera dans cette démonstration $v^0$ la constante 1, $v^1$ le vecteur $v$ et $v^2$ le vecteur $(v_1^2, \ldots, v_n^2)$ si $v$ est égal à $(v_1, \ldots, v_n)$. Soit donc $(C_1, f_1)$ et $(C_2, f_2)$ deux multi-ensembles de $\mathcal{ME}_n$ et $C = (C_1, f_1) + (C_2, f_2)$. Nous avons pour tout $i$ dans $\{0,1,2\}$ :

$$\begin{array}{rcl}
\boldsymbol{M}_i(C) & = & \sum_{v \in C_1 \cup C_2} f_1(v)v^i + f_2(v)v^i \\
& = & \sum_{v \in C_1 - C_2} f_1(v)v^i + \sum_{v \in C_2 - C_1} f_2(v)v^i + \sum_{v \in C_1 \cap C_2} f_1(v)v^i + f_2(v)v^i \\
& = & \sum_{v \in C_1 - C_2} f_1(v)v^i + \sum_{v \in C_1 \cap C_2} f_1(v)v^i + \sum_{v \in C_2 - C_1} f_2(v)v^i + \sum_{v \in C_1 \cap C_2} f_2(v)v^i \\
& = & \sum_{v \in C_1} f_1(v)v^i + \sum_{v \in C_2} f_2(v)v^i \\
\boldsymbol{M}_i(C) & = & \boldsymbol{M}_i(C_1) + \boldsymbol{M}_i(C_2)
\end{array}$$

□

Une propriété importante de la moyenne se déduit trivialement de l'additivité des moments :

**Lemme 1** *Soit* $(C_1, f_1)$ *et* $(C_2, f_2)$ *deux éléments de* $\mathcal{ME}_n$. *On a :*

$$\mu(C_1 + C_2) = \frac{|C_1|\mu(C_1) + |C_2|\mu(C_2)}{|C_1| + |C_2|}$$

**Corollaire 1** *Soit* $(C_1, f_1)$, $(C_2, f_2)$ *et* $(C, f) = (C_1, f_1) + (C_2, f_2)$ *des éléments de* $\mathcal{ME}_n$. *On a :*

$$\mu(C_1) - \mu(C_2) = \frac{|C_1| + |C_2|}{|C_2|}(\mu(C_1) - \mu(C)) = \frac{|C_1| + |C_2|}{|C_1|}(\mu(C) - \mu(C_2))$$

**Preuve:**

En utilisant le lemme 1 on a :

$$\mu(C_1) - \mu(C_2) = \mu(C_1) - \frac{(|C_1| + |C_2|)\mu(C) - |C_1|\mu(C_1)}{|C_2|} = \frac{|C_1| + |C_2|}{|C_2|}(\mu(C_1) - \mu(C))$$

De même :

$$\mu(C_1) - \mu(C_2) = \frac{(|C_1| + |C_2|)\mu(C) - |C_2|\mu(C_2)}{|C_1|} - \mu(C_2) = \frac{|C_1| + |C_2|}{|C_2|}(\mu(C) - \mu(C_2))$$

□

L'additivité des moments et le lemme précédent nous permettent de définir un théorème important permettant d'exprimer l'erreur quadratique de la somme de deux multi-ensembles.

**Théorème 1** *Soit $(C_1, f_1)$ et $(C_2, f_2)$ deux multi-ensembles de $\mathcal{ME}_n$ L'erreur quadratique du multi-ensemble $(C, f) = (C_1, f_1) + (C_2, f_2)$ est donnée par l'équation suivante :*

$$\boldsymbol{SE}(C) = \boldsymbol{SE}(C_1) + \boldsymbol{SE}(C_2) + \frac{|C_1||C_2|}{|C_1| + |C_2|}\|\mu(C_1) - \mu(C_2)\|^2 \tag{2.2}$$

**Preuve:**

Si nous utilisons l'équation 2.1, l'erreur quadratique du multi-ensemble $(C, f)$ s'exprime de la façon suivante :

$$\begin{aligned}\boldsymbol{SE}(C) &= \sum_{i=1}^{n} \boldsymbol{M}_2(C)_i - \frac{\boldsymbol{M}_1(C)_i^2}{|C|} \\ &= \sum_{i=1}^{n} \boldsymbol{M}_2(C)_i - |C|\mu(C)_i^2\end{aligned}$$

En utilisant la proposition 1 et le lemme 1 nous obtenons :

$$\begin{aligned}\boldsymbol{SE}(C) &= \sum_{i=1}^{n} \boldsymbol{M}_2(C_1)_i + \boldsymbol{M}_2(C_2)_i - |C|\mu(C_1 + C_2)_i^2 \\ &= \sum_{i=1}^{n} \boldsymbol{M}_2(C_1)_i + \boldsymbol{M}_2(C_2)_i - (|C_1| + |C_2|)\frac{(|C_1|\mu(C_1)_i + |C_2|\mu(C_2))^2}{(|C_1| + |C_2|)^2}\end{aligned}$$

Si nous rajoutons et enlevons $\frac{1}{|C_1|}\sum_{i=1}^{n} \boldsymbol{M}_1{}^2(C_1)_i$ et $\frac{1}{|C_2|}\sum_{i=1}^{n} \boldsymbol{M}_1{}^2(C_2)_i$ à la somme, nous obtenons :

$$\boldsymbol{SE}(C) = \boldsymbol{SE}(C_1) + \boldsymbol{SE}(C_2) + \sum_{i=1}^{n} |C_1|\mu(C_1)_i^2 + |C_2|\mu(C_2)_i^2 - \frac{(|C_1|\mu(C_1)_i + |C_2|\mu(C_2)_i)^2}{|C_1| + |C_2|}$$

Le développement du carré présent en fin d'égalité donne :

$$\begin{aligned}\boldsymbol{SE}(C) &= \boldsymbol{SE}(C_1) + \boldsymbol{SE}(C_2) + \frac{\sum_{i=1}^{n} |C_1||C_2|\mu(C_1)_i^2 + |C_1||C_2|\mu(C_2)_i^2 - 2|C_1||C_2|\mu(C_1)_i\mu(C_2)_i}{|C_1| + |C_2|} \\ &= \boldsymbol{SE}(C_1) + \boldsymbol{SE}(C_2) + \frac{|C_1||C_2|}{|C_1| + |C_2|}\sum_{i=1}^{n} \mu(C_1)_i^2 + \mu(C_2)_i^2 - 2\mu(C_1)_i\mu(C_2)_i \\ &= \boldsymbol{SE}(C_1) + \boldsymbol{SE}(C_2) + \frac{|C_1||C_2|}{|C_1| + |C_2|}\sum_{i=1}^{n} (\mu(C_1)_i - \mu(C_2)_i)^2 \\ \boldsymbol{SE}(C) &= \boldsymbol{SE}(C_1) + \boldsymbol{SE}(C_2) + \frac{|C_1||C_2|}{|C_1| + |C_2|}\|\mu(C_1) - \mu(C_2)\|^2\end{aligned}$$

□

## 2.2 Étude de la partition d'un multi-ensemble

Un cas particulier important est le cas où deux multi-ensembles disjoints sont munis de la même fonction de distribution. Cette propriété permet de simplifier grandement les opérations sur les multi-ensembles.

**Proposition 2** *Soient* $(C_1, f)$ *et* $(C_2, f)$ *deux multi-ensembles munis de la même fonction de distribution. Si* $C_1 \cap C_2 = \emptyset$ *alors on a :*

$$\begin{aligned} (C_1, f) + (C_2, f) &= (C_1, f) \cup (C_2, f) = (C_1 \cup C_2, f) \\ (C_1, f) \cap (C_2, f) &= (C_1 \cap C_2, f) \\ (C_1, f) - (C_2, f) &= (C_1 - C_2, f) \end{aligned}$$

**Remarque 1** *La simplification constatée dans la proposition 2 vient essentiellement du fait que les multi-ensembles manipulés sont disjoints. Le fait qu'ils possèdent la même fonction de distribution peut être considéré comme une conséquence de leur caractère disjoint. En effet, si deux multi-ensembles disjoints* $(C_1, f_1)$ *et* $(C_2, f_2)$ *ne possèdent pas la même fonction de fréquence, on considère généralement les multi-ensembles* $(C_1, f)$ *et* $(C_2, f)$ *avec :*

$$f = \chi_{C_1} f_1 + \chi_{C_2} f_2$$

*où* $\chi$ *désigne la fonction caractéristique.*

*Les multi-ensembles* $(C_1, f_1)$ *et* $(C_2, f_2)$ *sont alors confondus avec les multi-ensembles* $(C_1, f)$ *et* $(C_2, f)$*.*

Un ensemble de multi-ensembles disjoints est obtenu notamment dans le cas où un multi-ensemble $C$ est partitionné en $n$ sous multi-ensembles $\{C_1, \ldots, C_n\}$. La fonction de distribution restant la même pour tous les ensembles formant la partition, elle est souvent omise et l'on note le multi-ensemble $(C, f)$ simplement $C$.

Comme nous l'avons vu, l'erreur quadratique $\boldsymbol{SE}(C)$ mesure l'homogénéité de l'ensemble $C$. Afin de mesurer l'homogénéité d'une partition $\{C_1, \ldots, C_n\}$ on définit **l'erreur quadratique d'une partition** de la façon suivante :

**Définition 6** *Soient* $(C_1, f), \ldots, (C_p, f)$ *des éléments de* $\mathcal{ME}_n$ *tels que :*

$$\begin{array}{l} C = \bigsqcup_{i=1}^{p} C_i \\ \forall (i,j) \in \{1, \ldots, p\}^2 \quad i \neq j \Longrightarrow C_i \cap C_j = \emptyset \end{array}$$

*On définit alors* **l'erreur quadratique de la partition** *notée* $\boldsymbol{E}(C)$ *par :*

$$\boldsymbol{E}(C) = \sum_{i=1}^{p} \boldsymbol{SE}(C_i)$$

La partition de $C$ en sous-ensembles minimisant $\boldsymbol{E}(C)$ est un problème $\mathcal{NP}$ complet pour des espaces de dimension supérieure à un [WZ91]. Dans le cas mono-dimensionnel, donc pour des images en niveaux de gris, Wong, Wan et Prusinkiewicz ont élaboré un algorithme permettant de trouver la partition optimale avec une complexité en $O(N log N)$ [WWP89]. Le cas général de $p$ multi-ensembles dans un espace de dimension $n$ étant trop complexe, on simplifie le problème en étudiant le cas $p = 2$. On peut alors générer un nombre quelconque de multi-ensembles en divisant récursivement ceux-ci en deux sous-multi-ensembles. La partition obtenue n'est généralement pas celle qui minimise l'erreur quadratique globale définie par la définition 6, toutefois cette heuristique donne généralement de bons résultats si le découpage de chaque multi-ensemble en deux sous-multi-ensembles minimise suffisamment l'erreur quadratique. De plus des méthodes itératives telles que les nuées dynamiques [Did71] permettent ensuite de faire converger la partition initiale vers la partition optimale.

Afin de simplifier un peu plus le problème nous allons tout d'abord étudier le cas monodimensionnel avec $p = 2$ et voir quels résultats établis dans ce cas particulier peuvent se généraliser à des dimensions supérieures.

### 2.2.1 Étude de la dimension 1

Dans ce cas très particulier le multi-ensemble $(C, f)$ peut se voir comme un histogramme, avec $C = \{0, 1, 2, \ldots, 255\}$ et $f$ une fonction de $C$ dans $\mathbb{R}$ dénombrant le nombre de fois où un niveau de gris donné est présent dans l'image. Le problème est alors de trouver un niveau de gris $t^0$ divisant l'intervalle $\{0, \ldots, 255\}$ en deux multi-ensembles $C_1$ et $C_2$ tels que $\boldsymbol{SE}(C_1) + \boldsymbol{SE}(C_2)$ soit minimum. Wong [WWP89] a construit un algorithme linéaire permettant de trouver cette valeur optimale de t. Dans le même article Wong donne l'énoncé d'un théorème (non démontré) dont certaines propositions restent valable en dimension 3. Ce théorème peut s'énoncer de la façon suivante:

**Théorème 2** *Soient $([0, 1], f)$ un multi-ensemble de dimension 1 avec $f$ continue et $t$ un réel de l'intervalle $[0, 1]$ partitionnant $C$ en deux multi-ensembles $C_1(t)$ et $C_2(t)$ de moyenne $\mu_1(t)$ et $\mu_2(t)$. On définit $g(t) = \frac{\mu_1(t)+\mu_2(t)}{2}$ et $y(t) = g(t) - t$. On a alors les propositions suivantes :*

1. *$\mu_1$ et $\mu_2$ sont des fonctions croissantes vérifiant pour tout $t$ de l'intervalle $[0, 1]$ : $0 \leq \mu_1(t) \leq t \leq \mu_2(t) \leq 1$.*
2. *Si $y(t) > 0$ ( respectivement $y(t) < 0$) alors $y(g(t)) \geq 0$ (respectivement $y(g(t)) \leq 0$) et il n'existe pas de point $t^0$ dans l'intervalle $[t, g(t)[$ tel que $y(t^0) = 0$.*
3. *On a $M_0(C_1(t)) = M_0(C_2(t))$ si et seulement si $g(t) = \mu$ . Où $\mu$ désigne la moyenne de $([0, 1], f)$.*
4. *La valeur optimale de $t$ appartient à l'intervalle $[\frac{\mu}{2}, \frac{\mu+1}{2}]$*
5. *Soient $t_1 < t_2$ dans l'intervalle $[0, 1]$. Si $y(t_1)y(t_2) < 0$ alors il existe au moins un $t^0$ dans $[t_1, t_2]$ tel que $y(t^0) = 0$.*

**Preuve:**

Voir annexe section 7.7 □

**Remarque 2** *Les propositions 1 à 3 peuvent être étendues au cas discret. Les propositions 4 et 5 font elles plus spécifiquement appel à la continuité de la fonction de distribution $f$.*

### 2.2.2 Extension à la dimension 3

Nous avons vu que l'erreur quadratique d'un multi-ensemble partitionné en deux sous-multi-ensembles s'exprime par la formule :

$$\boldsymbol{E}(C) = \boldsymbol{SE}(C_1) + \boldsymbol{SE}(C_2) \text{ avec } C = C_1 \sqcup C_2 \tag{2.3}$$

Il peut être intéressant d'exprimer l'erreur de la partition en fonction de l'erreur quadratique du multi-ensemble $(C, f)$. Cette nouvelle formulation, conséquence directe du théorème 1, est une extension en trois dimensions d'un théorème initialement établi par Wong [WWP89] dans le cas mono-dimensionnel.

**Lemme 2** *En utilisant les notations précédentes, l'erreur quadratique d'une partition d'un ensemble $C = C_1 \cup C_2$ peut s'exprimer sous la forme :*

$$\boldsymbol{E}(C) = \boldsymbol{SE}(C) - \frac{|C_1|}{|C_2|}\|\mu(C_1) - \mu(C)\|^2 \tag{2.4}$$

**Preuve:**

Nous avons grâce au théorème 1 la relation suivante :

$$\boldsymbol{SE}(C) = \boldsymbol{SE}(C_1) + \boldsymbol{SE}(C_2) + \frac{|C_1||C_2|}{|C_1| + |C_2|}\|\mu(C_1) - \mu(C_2)\|^2$$

Donc :

$$\boldsymbol{SE}(C) = \boldsymbol{E}(C) + \frac{|C_1||C_2|}{|C_1| + |C_2|}\|\mu(C_1) - \mu(C_2)\|^2$$

On a également par le corollaire 1 :

$$\|\mu(C_1) - \mu(C_2)\|^2 = \left(\frac{|C_1| + |C_2|}{|C_2|}\right)^2 \|\mu(C_1) - \mu(C)\|^2$$

En combinant ces deux équations on obtient :

$$\boldsymbol{SE}(C) = \boldsymbol{E}(C) + \frac{|C_1|}{|C_2|}\|\mu(C_1) - \mu(C)\|^2$$

Si nous faisons passer $\frac{|C_1|}{|C_2|}\|\mu(C_1) - \mu(C)\|^2$ au second membre, on obtient l'égalité recherchée :

$$\boldsymbol{E}(C) = \boldsymbol{SE}(C) - \frac{|C_1|}{|C_2|}\|\mu(C_1) - \mu(C)\|^2$$

□

**Remarque 3** *Ce théorème reste valable pour deux éléments quelconques de $\mathcal{ME}_n$. Ce théorème étant principalement utilisé dans le cadre d'une partition nous avons simplement jugé plus logique de le placer dans ce cadre.*

Comme nous l'avons déjà mentionné, le découpage d'un multi-ensemble s'effectue souvent en considérant un de ses sous-ensembles et en faisant croître celui-ci jusqu'à obtenir une partition qui induit une erreur quadratique minimale. Du fait de la complexité du problème on ne choisit généralement pas la partition qui minimise globalement l'erreur quadratique, mais plus simplement la partition induisant une erreur quadratique minimale sur un ensemble de partitions envisagées. La partition la plus couramment utilisée est la *partition par plans*. Le multi-ensemble est alors découpé en deux sous-ensembles de par et d'autre du plan. Le nombre de plans susceptibles de couper un multi-ensemble étant encore trop grand, on restreint les possibilités en fixant la normale à celui-ci. L'ensemble des partitionnements possibles est alors égal au nombre de plans parallèles susceptibles de couper le multi-ensemble. Ce type de découpe utilise **la projection** d'un multi-ensemble définie de la façon suivante :

**Définition 7** *Soit $(C, f)$ un élément de $\mathcal{ME}_n$. La* **projection** *de $(C, f)$ sur l'axe vectoriel défini par le vecteur $\vec{n} \in \mathbb{R}^n$ est égale à $([m, M], F) \in \mathcal{ME}_1$ avec :*

$$\begin{array}{rcl} m & = & \min_{v \in C} v.\vec{n} \\ M & = & \max_{v \in C} v.\vec{n} \\ D_t & = & \{v \in C / v.\vec{n} = t\} \\ F(t) & = & \sum_{v \in D_t} f(v) \end{array}$$

*La découpe du multi-ensemble 1D $([m, M], F)$ au point $t \in [m, M]$ induit un partitionnement de $(C, f)$ en $(C_t, f)$ et $(C, f) - (C_t, f)$ avec :*

$$C_t = \{v \in C \ / \ v.\vec{n} \leq t\}$$

où $v.\vec{n}$ désigne le produit scalaire des vecteurs $v$ et $\vec{n}$. La variable $t$ joue dans ce cadre le rôle d'une abscisse le long de la direction $\vec{n}$. L'abscisse $t_{opt}$ du plan minimisant l'erreur quadratique est alors donnée par le théorème suivant :

**Théorème 3** *Avec les notations de la définition 7, soit la fonction $g$ définie sur $[m, M]$ par :*

$$g(t) = \frac{\delta(t)}{1 - \delta(t)} ||\mu(t) - \mu||^2 \quad avec \left\{ \begin{array}{lcl} \mu & = & \mu(([m, M], F)) \\ \mu(t) & = & \mu(C_t) \\ \delta(t) & = & \frac{|C_t|}{|C|} \end{array} \right.$$

*l'abscisse curviligne du plan minimisant l'erreur quadratique de la partition est égale à:*

$$t_{opt} = \arg \max_{t \in [m, M]} g(t) \tag{2.5}$$

*où* $\arg\max g(t)$ *désigne une valeur de $t$ réalisant le maximum de $g(t)$.*

*La fonction $\delta(t)$ est une fonction croissante bornée par zéro et par un.*

**Preuve:**

Si nous réécrivons l'expression de l'erreur quadratique de la partition en fonction des nouvelles notations nous avons :

$$\boldsymbol{E}(C) = \boldsymbol{SE}(C) - \frac{|C_t|}{|C| - |C_t|}\|\mu(t) - \mu\|^2 \tag{2.6}$$

En divisant le numérateur et le dénominateur de la fraction par $|C|$ il vient :

$$\boldsymbol{E}(C) = \boldsymbol{SE}(C) - \frac{\delta(t)}{1 - \delta(t)}\|\mu(t) - \mu\|^2 = \boldsymbol{SE}(C) - g(t) \tag{2.7}$$

L'erreur quadratique $\boldsymbol{SE}(C)$ étant constante, $\boldsymbol{E}(C)$ est minimum lorsque $g(t)$ est maximum. De plus, nous avons pour tout $t$ de l'intervalle $[m, M]$ :

$$\begin{array}{lll} \forall t \in [m, M] & & C_t \subseteq C \Rightarrow |C_t| \leq |C| \\ \forall t, t' \in [m, M]^2 & t \leq t' \Rightarrow & C_t \subseteq C_{t'} \Rightarrow |C_t| \leq |C'_t| \end{array}$$

Donc $\delta$ est bien une fonction croissante à valeurs dans $[0, 1]$.
□

**Remarque 4** *Si nous faisons évoluer $t$ de $m$ à $M$, $\delta(t)$ représente la proportion de données déjà parcourues.*

La nouvelle formulation de l'erreur quadratique de la partition fournie par le lemme 2 permet de manipuler des quantitées plus réduites. De plus l'expression donnée par le théorème 3 permet de préciser le comportement de l'erreur quadratique de la partition en fonction de $t$. Le théorème suivant fournit un encadrement de $g(t)$.

**Théorème 4** *Soient $(C, f)$ un élément de $\mathcal{ME}_n$, $\vec{n}$ un vecteur normé, non nul de $\mathbb{R}^n$. Avec les notations précédentes nous avons :*

*S'il existe $\beta \in [m, M]$ tel que $\delta(\beta) = \frac{1}{2}$ alors :*

$$\exists \Lambda \in \mathbb{R}^n, \alpha \in \mathbb{R} \ / \ \forall t \in [m, M] \quad L(t) \leq g(t) \leq U(t)$$

*avec :*

$$\forall t \in [m, M] \quad \left\{ \begin{array}{l} U(t) = \delta(t)(1 - \delta(t))\|\Lambda\|^2 \\ L(t) = \chi_{[m,\beta]}\frac{\delta(t)}{1-\delta(t)}\alpha + \chi_{]\beta,M]}\frac{1-\delta(t)}{\delta(t)}\alpha \end{array} \right.$$

*où $\chi$ représente la fonction caractéristique.*

**Preuve:**

**1)** $C$ étant borné, on peut définir deux vecteurs $\gamma^1$ et $\gamma^2$ tels que :

$$\forall v \in C \quad \forall i \in \{1, \ldots, n\} \quad \gamma_i^1 \leq v_i \leq \gamma_i^2 \tag{2.8}$$

On a de plus :

$$M_1(C_t) = \sum_{v \in C_t} v_i f(v) \tag{2.9}$$

$f$ étant une fonction positive, on a :

$$\forall i \in \{1,\ldots,n\} \quad {\gamma_i}^1|C_t| \leq M_1(C_t)_i \leq {\gamma_i}^2|C_t| \tag{2.10}$$

Où $M_1(C_t)_i$ désigne la $i^{eme}$ coordonnées de $M_1(C_t)$. Si nous désignons par $C - C_t$ le complémentaire de $C_t$ dans $C$, un raisonnement analogue nous donne l'équation :

$$\forall i \in \{1,\ldots,n\} \quad \gamma_i^1(|C|-|C_t|) \leq M_1(C-C_t)_i \leq \gamma_i^2|(|C|-|C_t|) \tag{2.11}$$

Si l'on introduit la fonction $\delta(t) = \frac{|C_t|}{|C|}$ ces deux équations s' écrivent :

$$\forall i \in \{1,\ldots,n\} \quad \begin{cases} {\gamma_i}^1\delta(t) \leq \frac{M_1(C_t)_i}{|C|} \leq {\gamma_i}^2\delta(t) \\ \\ \gamma_i^1(1-\delta(t)) \leq \frac{M_1(C-C_t)_i}{|C|} \leq \gamma_i^2(1-\delta(t)) \end{cases} \tag{2.12}$$

A présent, si nous exprimons le vecteur $\mu(t)-\mu$ en fonction de $M_1(C_t)$ et $M_1(C-C_t)$, il vient :

$$\begin{aligned}
\mu(t)-\mu &= \frac{M_1(C_t)}{|C_t|} - \frac{M_1(C)}{|C|} \\
&= \frac{1}{|C_t|}(M_1(C_t) - \frac{|C_t|}{|C|}M_1(C)) \\
&= \frac{1}{\delta(t)|C|}(M_1(C_t) - \delta(t)M_1(C)) \\
&= \frac{1}{\delta(t)|C|}((1-\delta(t))M_1(C_t) - \delta(t)M_1(C-C_t)) \\
\mu(t)-\mu &= \frac{1-\delta(t)}{\delta(t)}\frac{M_1(C_t)}{|C|} - \frac{M_1(C-C_t)}{|C|}
\end{aligned}$$

Or en utilisant les encadrement définis par les équations 2.12 on a pour tout $i$ de l'ensemble $\{1,\ldots,n\}$ :

$$\begin{array}{lcccl}
\gamma_i^1(1-\delta(t)) & \leq & \frac{(1-\delta(t))}{\delta(t)}\frac{M_1(C_t)_i}{|C|} & \leq & \gamma_i^2(1-\delta(t)) \\
{\gamma_i}^1(1-\delta(t)) & \leq & \frac{M_1(C-C_t)_i}{|C|} & \leq & \gamma_i^2(1-\delta(t))
\end{array}$$

On a donc :

$$\forall i \in \{1,\ldots,n\} \quad (1-\delta(t))(\gamma_i^1-\gamma_i^2) \leq (\mu(t)-\mu)_i \leq (1-\delta(t))(\gamma_i^2-\gamma_i^1) \tag{2.13}$$

On a donc, pour tout $i$ dans $\{1,\ldots,n\}$ :

$$|(\mu(t)-\mu)_i| \leq (1-\delta(t))(\gamma_i^2-\gamma_i^1) \tag{2.14}$$

où $|(\mu(t)-\mu)_i|$ désigne la valeur absolue de la $i^{eme}$ coordonnées du vecteur $\mu(t)-\mu$. Si nous posons $\Lambda = \gamma^2 - \gamma^1$ on obtient :

$$\|\mu(t)-\mu\|^2 \leq (1-\delta(t))^2\|\Lambda\|^2 \tag{2.15}$$

En combinant, cette dernière équation et l'expression de $g(t)$ on obtient l'inégalité recherchée, à savoir :

$$g(t) \leq \delta(t)(1-\delta(t))\|\Lambda\|^2 \tag{2.16}$$

**2)** Soit $C' = ([m,M],F)$ la projection sur l'axe $\vec{n}$ du multi-ensemble $(C,f)$ (voir définition 7). Soit $t$ un réel appartenant à l'intervalle $[m,M]$. Ce réel partitionne le multi-ensemble $[m,M]$ en deux multi-ensembles mono-dimensionnels $C_1(t) = ([m,t],F)$ et $C_2(t) = (]t,M],F)$. A chacun de ces sous-multi-ensembles correspond une moyenne définie à partir de la fonction $F$ notées respectivement $r_1(t)$ et $r_2(t)$. Nous pouvons également définir une fonction $\delta'$ telle que :

$$\delta' \left( \begin{array}{lll} [m,M] & \rightarrow & \mathbb{R} \\ t & \mapsto & \dfrac{M_1(C_1(t))}{|C_1(t)|} \end{array} \right.$$

La fonction $\delta'$ est l'analogue de la fonction $\delta$ définie précédemment. Du fait de la construction de la fonction $F$ un fort lien existe entre les partitions des multi-ensembles $C$ et $C'$. De fait nous avons les égalités suivantes :

$$\forall t \in [m,M] \quad \left\{ \begin{array}{lll} \delta'(t) & = & \delta(t) \\ r_1(t) & = & \mu_1(t).\vec{n} \\ r_2(t) & = & \mu_2(t).\vec{n} \end{array} \right. \tag{2.17}$$

En effet nous avons :

$$|C'| = \sum_{x=m}^{M} F(x) = \sum_{x=m}^{M} \sum_{v \in D_x} f(v) \tag{2.18}$$

Or, nous avons également de façon triviale $C = \bigsqcup_{x=m}^{M} D_x$, donc :

$$|C'| = \sum_{x=m}^{M} \sum_{v \in D_x} f(v) = \sum_{v \in C} f(v) = |C| \tag{2.19}$$

de même, nous avons $C_t = \bigsqcup_{x=m}^{t} D_x$, et donc $C_1(t) = |C_t|$. De plus :

$$\begin{array}{lll} M_1(C).\vec{n} & = & (\sum_{v \in C} v f(v)).\vec{n} \\ & = & \sum_{v \in C} v.\vec{n} f(v) \\ & = & \sum_m^M t(\sum_{x \in D_t} f(x)) \\ & = & \sum_m^M tF(t)dt \\ M_1(C).\vec{n} & = & M_1(C') \end{array}$$

On montre de même que $(M_1(C_t)).\vec{n} = M_1(C_1(t))$ et $(M_1(C - C_t)).\vec{n} = M_1(C_2(t))$. De simples divisions permettent alors de vérifier les égalités 2.17.

Les fonctions $\delta$ et $\delta'$ étant égales, $\beta$ est une valeur médiane de l'histogramme $C'$. Nous pouvons donc appliquer la proposition 3 du théorème 2. On a donc :

$$\frac{r_1 + r_2}{2}(\beta) = r$$

où $r$ désigne la moyenne de l'ensemble $C'$. Soit $\alpha$ défini par :

$$\sqrt{\alpha} = (r - r_1)(\beta) = (r_2 - r)(\beta) > 0$$

Nous savons grâce à la proposition 1 du même théorème 2 que $r_1$ et $r_2$ sont deux fonctions croissantes. Donc $r - r_1$ est une fonction décroissante tandis que $r_2 - r$ est croissante. Les deux fonctions se rencontrant au point $\beta$. On a donc :

$$\begin{array}{ll} \forall t \in [m,\beta] & r - r_1 \geq \sqrt{\alpha} \\ \forall t \in [\beta, M] & r_2 - r \geq \sqrt{\alpha} \end{array}$$

On montre également facilement à partir du corollaire 1 que :

$$\frac{\delta(t)}{1-\delta(t)}\|r_1 - r\|^2 = \frac{1-\delta(t)}{\delta(t)}\|r_2 - r\|^2 \tag{2.20}$$

En combinant cette égalité aux inégalités précédentes on obtient :

$$\begin{array}{lrl} \forall t \in [m,\beta] & & \frac{\delta(t)}{1-\delta(t)}\|r_1 - r\|^2 \geq \frac{\delta(t)}{1-\delta(t)}\alpha \\ \forall t \in ]\beta, M] & \frac{\delta(t)}{1-\delta(t)}\|r_1 - r\|^2 = & \frac{1-\delta(t)}{\delta(t)}\|r_2 - r\|^2 \geq \frac{1-\delta(t)}{\delta(t)}\alpha \end{array}$$

Or, on a également :

$$\forall t \in [m, M] \quad (r_1(t) - r)^2 = ((\mu(t) - \mu).\vec{n})^2 \leq \|\mu(t) - \mu\|^2$$

On obtient donc bien les inégalités recherchées, à savoir :

$$\begin{array}{ll} \forall t \in [m,\beta] & \dfrac{\delta(t)}{1-\delta(t)}\|\mu(t) - \mu\|^2 \geq \dfrac{\delta(t)}{1-\delta(t)}\alpha \\ \forall t \in [\beta, M] & \dfrac{\delta(t)}{1-\delta(t)}\|\mu(t) - \mu\|^2 \geq \dfrac{1-\delta(t)}{\delta(t)}\alpha \end{array}$$

Ces deux inégalités peuvent se factoriser de la façon suivante :

$$\begin{array}{lll} g(t) & \geq & \chi_{[m,\beta]}\frac{\delta(t)}{1-\delta(t)}\alpha + \chi_{]\beta,M]}\frac{1-\delta(t)}{\delta(t)}\alpha \\ g(t) & \geq & L(t) \end{array}$$

□

**Remarque 5** *Si on ne peut définir une valeur médiane $\beta$ les inégalités précédentes restent valables si l'on prend :*

$$\beta = \max\{t \in [m, M]/\delta(t) \leq \frac{1}{2}\} \ et \ \sqrt{\alpha} = \frac{(r - r_1)(\beta) + (r_2 - r)(\beta)}{2}$$

*De plus si $\delta$ est strictement croissante, on peut alors réaliser le changement de variable $u = \delta(t)$. A ce moment là, la fonction $G = g \circ \delta^{-1}$ est encadrée par une parabole et par une courbe composée de deux bouts d'hyperboles. Plus précisément l'on a (voir Figure 2.1) :*

$$\forall u \in [0,1] \quad \chi_{[0,\frac{1}{2}]}\frac{u}{1-u}\alpha + \chi_{]\frac{1}{2},1]}\frac{1-u}{u}\alpha \leq G(u) \leq u(1-u)\|\Lambda\|^2$$

*Cet encadrement sera utilisé dans le chapitre 3.*

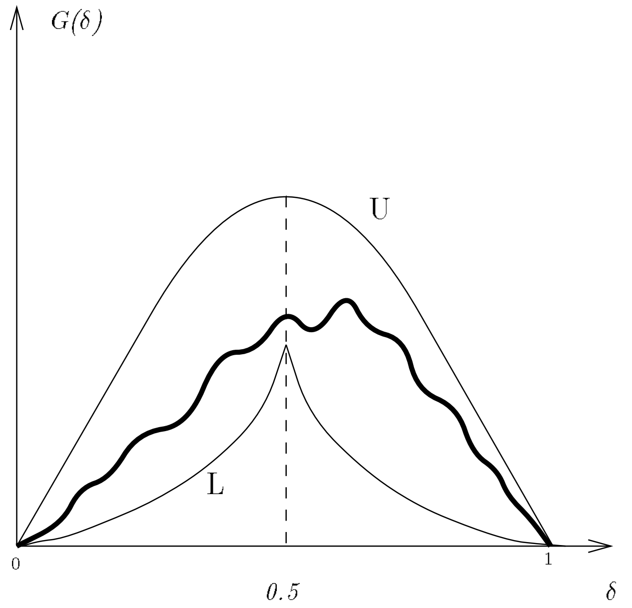



Figure 2.1: *$g(t)$ est encadrée par une parabole et une courbe composée de deux partie d'hyperboles. Toutes ces courbes sont paramétrisées en $\delta$ et atteignent leur maximum pour $\delta = \frac{1}{2}$.*

## 2.3 Mesure de l'homogénéité d'un multi-ensemble de couleurs

L'homogénéité d'un multi-ensemble $(C, f)$ est calculée gâce à l'erreur quadratique (voir définition 4) :

$$\boldsymbol{SE}(C) = \sum_{c \in C} f(c) \|c - \mu(C)\|^2$$

L'erreur quadratique d'un multi-ensemble de $\mathcal{ME}_n$ est donc relative à une base de $\mathbb{R}^n$ et à la métrique définie sur celui-ci. Si nous définissons un multi-ensemble $(C, f)$ dans un espace de couleurs $E$, on obtient un élément de $\mathcal{ME}_3$. L'erreur quadratique de $(C, f)$ calculée dans l'espace $E$ n'est a priori pas comparable avec l'erreur quadratique du même multi-ensemble calculée dans un autre espace. Si nous calculons, par exemple, l'erreur quadratique dans un espace déduit de $E$ par une matrice $A$, l'erreur quadratique dans l'espace d'arrivée sera égale à :

$$\boldsymbol{SE}(C) = \sum_{c \in C} f(c)(c - \mu(C)) A^t A (c - \mu(C))$$

où $A^t$ représente la transposé de la matrice $A$.

Donc, à moins que la matrice $A$ ne soit diagonale, les erreurs quadratiques calculées dans deux espaces de couleurs différents ne sont pas comparables.

L'espace $L^*u^*v^*$ étant approximativement uniforme, il semble logique de l'utiliser comme espace de référence et de calculer les erreurs quadratiques dans cet espace. Ceci est impossible pour deux raisons. Tout d'abord, l'expérience nous a montré que l'erreur quadratique calculée dans l'espace $L^*u^*v^*$ n'est pas plus pertinente que l'erreur quadratique calculée dans les espaces $RGB$, $I_1I_2I_3$ ou $H_1H_2H_3$. De plus, si un algorithme manipule des multi-ensembles dans un espace de couleurs donné, il utilisera la métrique de cet espace. Il est donc illogique de mesurer les performances d'un algorithme en mesurant l'erreur quadratique dans un espace donné si l'algorithme tend, par exemple, à minimiser l'erreur quadratique à l'aide d'une autre métrique. La construction d'un algorithme travaillant sur des multi-ensembles de couleurs impose donc le choix préalable d'un espace de couleurs. De plus les mesures de performances de cet algorithme devront utiliser le même espace de couleurs.

# Chapitre 3

# La quantification d'images couleurs

## 3.1 Objectifs

Le principal objectif des méthodes de quantification d'images couleurs est de réduire le nombre de couleurs de l'image originale en créant une distorsion minimale entre l'image quantifiée et l'image originale. Plus formellement, considérons une image $I$, nous pouvons y associer le multi-ensemble $(C, f)$ où C représente l'ensemble des couleurs présentes dans l'image et $f(c)$ le nombre de pixels de couleur $c$ dans $I$. La quantification de $I$ en $K$ couleurs, avec $K < |C|$ et usuellement $K << |C|$, consiste à sélectionner un ensemble de $K$ **couleurs représentatives** et à remplacer chaque pixel de l'image originale par sa couleur représentative la plus proche. L'ensemble des couleurs représentatives $\{c_1, \ldots, c_K\}$ est communément appelée la **table de couleurs**. La fonction qui associe à chaque couleur de l'image sa couleur représentative est appelée la fonction **d'inversion de table de couleurs** et est notée $\boldsymbol{Q}$.

La quantification d'image peut être utilisée pour afficher une image comportant un nombre important de couleurs, comme les images 24 bits sur des terminaux ne pouvant en afficher qu'un nombre réduit. Plus généralement la quantification de couleurs peut être vue comme un processus permettant une compression des données de l'image. Il est généralement possible de construire une image extrêmement proche de l'original avec moins de 256 couleurs. Malgré leurs faibles taux de compression les algorithmes de quantification restent très populaires. Wu [Wu92] a souligné une importante raison de cette popularité en remarquant que ces heuristiques combinées avec des tables de couleurs matérielles permettent un décodage de l'image en temps réel. L'aspect de la quantification d'images qui a motivé notre étude sur ce sujet est la compression d'information. En effet, la quantification d'une image en $K$ couleurs nous permet de trouver les $K$ couleurs les plus significatives de celle-ci. Cette propriété sera utilisée dans plusieurs algorithmes de segmentation que nous étudierons par la suite.

## 3.2 Choix d'un espace de couleurs pour la quantification

Jusqu'à présent il n'a pas été effectué de tests systématiques des avantages et inconvénients des différents espaces de couleurs pour les algorithmes de quantifications. Ceci est sans doute dû à la grande diversité de ces algorithmes. Une telle démarche a été adoptée par Otha [OKS80] dans le cadre des algorithmes de segmentation. Ceci l'a conduit à la définition de l'espace $I_1I_2I_3$ (voir section 1.2.4).

Cette absence d'avantages évidents d'un espace de couleurs vis à vis des autres est illustrée sur la figure 3.1 où l'image test Lenna a été quantifiée en 16 couleurs avec le même algorithme de quantification utilisant différents espaces de couleurs. Les images 3.1-(b), 3.1-(c), 3.1-(e) et 3.1-(f) ont respectivement été obtenues par des quantifications dans les espaces $RGB$, $YIQ$, $L^*u^*v^*$, $I_1I_2I_3$ et $H_1H_2H_3$, l'image 3.1-(a) représentant l'image originale. Bien que l'utilisation d'espaces de couleurs différents induise des images finales différentes, il est difficile d'établir des critères visuels objectifs permettant de privilégier une image.

Faute d'une étude exhaustive des avantages et inconvénients des différents espaces de couleurs pour la quantification la plupart des méthodes de quantification ne font que préconiser un espace de couleurs. Les espaces de couleurs les plus recommandés sont les espaces $RGB$, $YIQ$, $L^*u^*v^*$ et$L^*a^*b^*$( voir section 1.2). L'espace $RGB$ est souvent choisi pour sa simplicité. De plus de nombreux formats d'images codent la couleur de chaque pixel dans cet espace. Utiliser celui-ci pour la quantification évite donc de convertir les coordonnées de chaque pixel. L'espace $YIQ$ se déduit facilement de l'espace $RGB$. Il permet de plus de séparer l'information de luminance de l'information de chrominance. Les espaces uniformes $L^*u^*v^*$ et $L^*a^*b^*$ séparent également ces deux types d'informations. Ils présentent en outre l'avantage d'être plus en adéquation avec la vision humaine. Les conversions de l'espace $RGB$ aux espaces $L^*u^*v^*$ ou $L^*a^*b^*$ sont les plus coûteuses des transformations vues au chapitre 1. Ces espaces de couleurs sont donc essentiellement utilisés par des algorithmes privilégiant la qualité des images obtenues au détriment des temps de calculs.

## 3.3 Utilisation de l'erreur quadratique

Nous avons vu au chapitre 2 que l'erreur quadratique $\boldsymbol{SE}(C)$ d'un multi-ensemble $(C, f)$ permet de mesurer l'homogénéité de $(C, f)$. De même l'erreur quadratique d'une partition permet de mesurer l'homogénéité des ensembles formant la partition. L'utilisation de l'erreur quadratique pour la quantification repose sur le fait qu'une partition en un ensemble de multi-ensembles homogènes fournira une image quantifiée visuellement proche de l'original. Une des justifications de ce postulat repose sur la propriété suivante :

**Proposition 3** *Soient $I$ une image de taille $m \times n$, $(C, f)$ le multi-ensemble associé à $I$ et $I'$ l'image issue de la quantification de $I$ en $K$ couleurs. Si l'ensemble des couleurs représentatives $\{c_1, \ldots, c_K\}$ vérifie :*

$$\forall i \in \{1, \ldots, K\} \quad c_i = \mu(Q^{-1}(c_i)) \tag{3.1}$$


Figure 3.1: *Quantification de l'image test Lenna dans différents espaces de couleurs*

*alors l'erreur quadratique de la partition de* $(C, f)$ *peut s'exprimer de la façon suivante:*

$$E(C) = \sum_{i=1}^{m} \sum_{j=1}^{n} \|c_I(i,j) - c_{I'}(i,j)\|^2$$

*Où* $c_I(i,j)$ *et* $c_{I'}(i,j)$ *représentent les couleurs du pixel* $(i,j)$ *dans les images* $I$ *et* $I'$.

**Preuve:**

On a pour tout pixel $(i,j)$ de $I'$ :

$$c_{I'}(i,j) = Q(c_I(i,j))$$

Donc,

$$\begin{array}{rcl} \sum_{i=1}^{m} \sum_{j=1}^{n} \|c_I(i,j) - c_{I'}(i,j)\|^2 & = & \sum_{i=1}^{m} \sum_{j=1}^{n} \|c_I(i,j) - Q(c_I(i,j))\|^2 \\ & = & \sum_{c \in C} \|c - Q(c)\|^2 \\ & = & \sum_{i=1}^{K} \sum_{c \in Q^{-1}(c_i)} \|c - c_i\|^2 \\ & = & \sum_{i=1}^{K} \sum_{c \in Q^{-1}(c_i)} \|c - \mu(Q^{-1}(c_i)\|^2 \end{array}$$

Ce qui est la formulation de l'erreur d'une partition fournie par la définition 6.

□

La pré-condition 3.1 étant généralement vérifiée l'erreur de partition peut se voir comme la somme pixel à pixel des différences de couleurs au carré entre l'image originale et l'image quantifiée. Il est toutefois dangereux de voir l'erreur quadratique comme une fonction de distance entre l'image originale et l'image quantifiée. Ceci est confirmé par l'expérience suivante. L'image 3.3-(a) est obtenue à partir de l'image 3.2 par une quantification en 8 couleurs. L'image originale 3.2 comporte 15 738 couleurs. Si nous appliquons l'algorithme de tramage (ou dithering) de Floyd-Steinberg [FS76] sur l'image quantifiée 3.3-(a) nous obtenons l'image 3.3-(b) qui est visuellement beaucoup plus proche de l'image originale 3.2. L'erreur quadratique de l'image 3.3-(b) est pourtant plus élevée que celle de l'image 3.3-(a), la différence relative entre les deux erreurs étant d'environ 30%. Ce phénomène a priori surprenant est dû au fait que l'algorithme de tramage a brisé la partition créée par l'algorithme de quantification. La nouvelle partition créée par l'algorithme de tramage tient compte de propriétés locales de l'image ce qui n'est pas pris en compte dans l'erreur quadratique. Les multi-ensembles créés par l'étape de tramage ne sont plus forcément connexes et sont a priori moins homogènes. L'erreur quadratique de ces multi-ensembles est donc plus importante que celle produite par les multi-ensembles issus de la quantification. Orchard [OB91] a défini une erreur quadratique pondérée permettant de tenir compte des propriétées locales de l'image durant l'étape de quantification. Orchard remplace dans la définition de l'erreur quadratique (voir définition 4) la fonction de fréquence $f$ par la fonction de **fréquence pondérée** $W$, où $W(c)$ est définie comme une somme d'attributs calculés localement sur tous les pixels de couleur $c$ de l'image. Dans ce cadre, l'erreur quadratique définie au chapitre 2 peut se voir comme un cas particulier d'erreur quadratique pondérée où l'attribut d'un pixel de couleur $c$ est égal à 1. Cette fonction de pondération permet de donner plus de poids aux couleurs de l'image majoritairement situées dans des régions sensibles aux erreurs de quantification.

Figure 3.2: *Rochers : image originale*

Figure 3.3: *Application de l'algorithme de Floyd-Stenberg sur l'image de gauche*

## 3.4 Création de l'image quantifiée

Nous avons vu dans la section 3.1 que l'objectif d'un algorithme de quantification est de partitionner le multi-ensemble $(C, f)$ contenant l'ensemble des couleurs de l'image en un ensemble de multi-ensembles $\{(C_1, f), \ldots, (C_K, f)\}$. Partant de cette partition, l'on construit un ensemble de couleurs représentatives $\{c_1, \ldots, c_K\}$ défini par :

$$\forall i \in \{1, \ldots, K\} \quad c_i = \mu(C_i)$$

La fonction $\boldsymbol{Q}$ définie dans la section 3.1 associe à chaque couleur de l'image sa couleur représentative la plus proche. On a donc :

$$\boldsymbol{Q}(c) = ArgMin_{z \in \{c_1, \ldots, c_K\}} \|z - c\|$$

où $ArgMin$ renvoie l'argument réalisant le minimum.

La fonction $\boldsymbol{Q}$ peut se définir à l'aide d'un diagramme de Voronoi 3D [Ber94, Aur91, CP95]. Dans ce cas $\boldsymbol{Q}(c)$ est égal à la couleur représentative $c_i$ si $c$ appartient à la cellule de Voronoi associée à $c_i$. Une fois le diagramme de Voronoi effectué le calcul de la fonction $\boldsymbol{Q}$ s'effectue en temps constant. Thomas [Tho91] a proposé en 1991 un algorithme permettant de construire le diagramme de Voronoi 3D associé à $K$ couleurs représentatives. Cet algorithme construit une partition de l'espace de couleurs et réclame généralement de longs temps de calcul. Thomas réduit l'occupation mémoire et les temps de calcul en ne considérant que les 5 bits de poids fort des composantes $R, G, B$ de chaque couleur. Toutefois, malgré cette simplification, l'algorithme de Thomas réalise le diagramme de Voronoi correspondant à 256 couleurs représentatives en environ 24 secondes. Le temps nécessaire à un algorithme de quantification pour quantifier une image étant généralement inférieur à la minute le surcoût en temps induit par l'utilisation de l'algorithme de Thomas est important.

Une façon "naïve" de calculer $\boldsymbol{Q}(c)$ consiste à calculer pour chaque couleur représentative $\{c_1, \ldots, c_K\}$ la distance à la couleur $c$ et de prendre la couleur représentative la plus proche. Heckbert [Hec82] a amélioré cette recherche exhaustive en partitionnant l'espace des couleurs en $N$ sous-cubes. Cette partition permet, de rejeter a priori certaines couleurs représentatives de la recherche exhaustive. Les mesures expérimentales faites par Heckbert montrent que les $K$ tests nécessaires à la recherche "naïve" sont dans ce cas ramenés à $\frac{K}{23}$ tests. Toutefois, malgré cette amélioration la complexité de la fonction $\boldsymbol{Q}$ reste linéaire.

La partition de l'espace de couleurs induite par l'algorithme de quantification ne constitue généralement pas un diagramme de Voronoi. Toutefois, des expériences réalisées par Wu [WZ91] montrent que si l'algorithme de quantification produit une faible erreur de partition, les cellules de Voronoi peuvent être approximées par les multi-ensembles construit par l'algorithme de quantification. Donc si l'algorithme de quantification partitionne l'ensemble des couleurs de l'image $(C, f)$ en $K$ multi-ensembles $\{(C_1, f), \ldots, (C_K, f)\}$ la couleur représentative $\boldsymbol{Q}(c)$ d'une couleur $c$ sera égale à $\mu(C_i)$ si et seulement si $c$ appartient à $C_i$. Il est alors souvent possible d'utiliser les structures de données induites par les algorithmes de quantification pour rechercher efficacement le multi-ensemble contenant une couleur donnée. Par exemple, la structure de données induite par les algorithmes de Wu [WZ91] ou de Wan et al. [WWP88] permet de retrouver la couleur représentative $\boldsymbol{Q}(c)$ d'une couleur $c$ grâce à un algorithme de complexité $\mathcal{O}(\log_2(K))$.

## 3.5 Classification des algorithmes

Une première approche pour définir une table de couleurs consiste à fixer l'ensemble des couleurs représentatives $\{c_1, \ldots, c_K\}$ de façon à couvrir un large spectre de couleurs. On parle dans ce cas de **quantification uniforme** , l'ensemble des couleurs représentatives est prédéterminé [GAW90, Pae91] et reste identique pour chaque image à quantifier. Il est bien évident que si le nombre de couleurs finales est réduit ($K = 8$ ou 16) ce type d'algorithme donne des résultats nettement moins intéressants qu'une quantification construisant un ensemble de couleurs représentatives adapté à chaque image. Les algorithmes de ce type seront appelés algorithmes de **quantification adaptative**. Ces algorithmes partitionnent le multi-ensemble $(C, f)$ associé à l'image en un ensemble $\{(C_1, f), \ldots, (C_K, f)\}$ d'éléments de $\mathcal{ME}_3$. Ils associent ensuite à chaque multi-ensemble $(C_i, f)$ une couleur représentative $\mu(C_i)$.

Ces méthodes peuvent grossièrement se scinder en deux grandes familles. La première utilise une **approche ascendante**. Ces algorithmes sélectionnent $K$ couleurs de l'image afin d'initialiser $K$ multi-ensembles, avec $K$ le nombre de couleurs finales. Les autres couleurs de l'image sont alors lues et fusionnées aux $K$ multi-ensembles courants selon différentes heuristiques [GP90, XJ94]. De telles méthodes n'essaient généralement pas de minimiser l'erreur de partition (voir définition 6) et s'appuient sur des heuristiques qu'il est souvent difficile de justifier d'un point de vue théorique.

Les **approches descendantes** ont quant à elles été beaucoup plus explorées [Hec82, WWP88, WZ91, Wu92, BBA94]. Ces méthodes initialisent un multi-ensemble $(C, f)$ à partir de l'image et partitionnent celui-ci jusqu'à obtenir $K$ multi-ensembles formant une partition de $(C, f)$. Comme nous l'avons vu dans la section 2.2, trouver le partitionnement idéal en $K$ multi-ensembles minimisant l'équation de la définition 6 est un problème $\mathcal{NP}$ complet pour des images couleurs. Selon Anderberg [And73] le nombre de partitionnements possibles est égal à $\frac{1}{K!}\sum_{i=0}^{K}(-1)^{K-i}C_K^i i^{|C|}$ où $K$ est le nombre final de couleurs et $(C, f)$ le multi-ensemble associé à l'image. L'algorithme trivial testant chaque partitionnement n'est donc pas réaliste et des heuristiques de découpe doivent être employées. Balasubramanian [BBA94] réalise un histogramme 1-D suivant une des composantes de l'image (voir définition 7) puis partitionne $(C, f)$ par des plans perpendiculaires à cet axe en $N_1$ multi-ensembles. Il sélectionne ensuite un autre axe de découpe, et redécoupe chaque multi-ensemble perpendiculairement à cet axe grâce à des histogrammes 1D calculés sur chacun des $N_1$ multi-ensembles. Il obtient ainsi $N_2$ multi-ensembles. Une dernière itération de ce processus fournit les $N_3 = K$ multi-ensembles finaux. Cette méthode semble prometteuse mais laisse de nombreuses questions en suspend. Tout d'abord on ne connaît pas, sauf à faire une énumération exhaustive, la séquence optimale d'axes suivant lesquels on doit réaliser le partitionnement. De plus l'extension de résultats valables pour $K$ proche de l'infini à des valeurs de $K$ comprises entre 4 et 256 semble pour le moins hasardeuse. Enfin, cet algorithme travaillant avec un ensemble d'histogrammes 1D tient a priori moins compte de l'information 3D des couleurs qu'un algorithme travaillant directement sur les données 3D. Une autre méthode présentée par Wu [Wu92] utilise la programmation dynamique pour partitionner le multi-ensemble $(C, f)$ en $\kappa$ multi-ensembles ($\kappa < K$). Les multi-ensembles restants sont alors récursivement découpés en deux jusqu'à obtenir les $K$ multi-ensembles finaux. Le problème dans ce cas est de définir la valeur optimale de $\kappa$.

## 3.6 Étude détaillée de l'approche descendante

La plupart des algorithmes de quantification basés sur une approche descendante [Hec82, WWP88, WZ91, Wu92, BBA94] initialisent un multi-ensemble $(C, f)$ à partir de l'image et subdivisent $(C, f)$ récursivement en deux jusqu'à obtenir $K$ multi-ensembles formant une partition de $(C, f)$. Le découpage récursif consiste à choisir un multi-ensemble parmi les multi-ensembles déjà créés et à le découper en deux. Ce processus doit être itéré $K+1$ fois de façon à obtenir les $K$ multi-ensembles finaux. Le découpage de chaque multi-ensemble est réalisé par un plan appelé **plan de découpe** orthogonal à une direction appelée **direction de découpe**. Ce processus peut se subdiviser en 4 étapes élémentaires communes à tous les algorithmes de cette famille :

1. La sélection d'une stratégie de découpe.
2. La sélection du multi-ensemble à découper.
3. La sélection d'une direction de découpe.
4. La recherche de la position du plan de coupe orthogonal à la direction de coupe.

Les heuristiques utilisées lors de ces 4 étapes nécessitent un stockage approprié du multi-ensemble 3D associé à l'image. Nous allons à présent étudier ces heuristiques et les structures de données utilisées pour stocker le multi-ensemble associé à l'image.

### 3.6.1 Sélection de la stratégie de découpe

Une stratégie de découpe possible consiste à découper récursivement $(C, f)$ jusqu'à l'obtention des $K$ multi-ensembles finaux. Cette séquence de découpes peut être représentée par un arbre binaire complet (un arbre dont chaque noeud possède exactement deux fils). Cet arbre est appelé l'**arbre de découpe** (voir Figure 3.4).

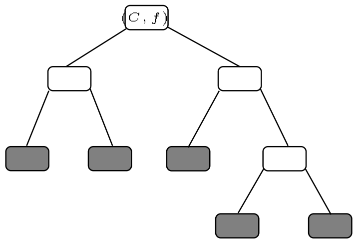



Figure 3.4: *Arbre de découpe représentant une stratégie de découpe de $(C, f)$ en 5 multi-ensembles.*

Le nombre de découpes possibles par cette stratégie est donc égal au nombre d'arbres binaires complets possédant exactement $K$ feuilles soit : $\frac{1}{K}C_{2(K-1)}^{K-1}$ [FGS90]. Ce nombre est trop important pour envisager une énumération exhaustive de toutes les stratégies. Chou et al. [CTM89], Lin et al [LSC91] et Balasubramanian et al. [BA91] ont suggéré de créer un arbre intermédiaire avec N feuilles (où $K < N << \frac{1}{K}C_{2(K-1)}^{K-1}$) et d'élaguer cet arbre de façon à ne conserver que les $K$ feuilles qui conduisent à une erreur quadratique minimale. Selon Wu [WZ91] cette stratégie implique un surcoût trop important par rapport à l'amélioration apportée. Il recommande donc de ne générer que les $K$ multi-ensembles nécessaires.

### 3.6.2 Sélection du multi-ensemble à découper

La stratégie la plus simple pour sélectionner le multi-ensemble à découper a été énoncée par Heckbert [Hec82]. Cette stratégie consiste à découper le multi-ensemble de plus grand cardinal. Cette stratégie peut être grandement améliorée en utilisant l'erreur de la partition (voir définition 6). Wan et al. [WWP88] ont proposé de sélectionner le multi-ensemble d'erreur quadratique maximale. Cette stratégie repose sur le fait que l'erreur quadratique de la partition est définie comme la somme des erreurs quadratiques des multi-ensembles formant cette partition. Donc en découpant le multi-ensemble dont la contribution à la somme est la plus importante, on espère faire diminuer celle-ci. Une stratégie légèrement différente a été proposée par Wu [WZ91]. Celui-ci découpe le multi-ensemble dont la partition va entraîner la plus grande diminution de l'erreur quadratique. Cette stratégie repose sur le raisonnement suivant :

Supposons que le multi-ensemble $(C, f)$ contenant l'ensemble des couleurs de l'image a été découpé $k$ fois en $k+1$ multi-ensembles $\{(C_0, f), \ldots, (C_k, f)\}$. L'erreur $E(C)$ associée à la partition est alors égale à $\sum_{i=0}^{k} \boldsymbol{SE}(C_i)$. Supposons à présent que le prochain multi-ensemble à découper soit $C_i$ et considérons $C_i^1$ et $C_i^2$ les deux multi-ensemble issus du découpage de $C_i$. L'erreur associée à la partition après le découpage est alors égale à :

$$E(C) = \boldsymbol{SE}(C_i^1) + \boldsymbol{SE}(C_i^2) + \sum_{j=0, j\neq i}^{k} \boldsymbol{SE}(C_j)$$

Le découpage du multi-ensemble $(C_i, f)$ a donc modifié $E(C)$ de $\boldsymbol{SE}(C_i^1) + \boldsymbol{SE}(C_i^2) - \boldsymbol{SE}(C_i)$. On peut remarquer que le théorème 1 assure que $\boldsymbol{SE}(C_i^1) + \boldsymbol{SE}(C_i^2) - \boldsymbol{SE}(C_i)$ est négatif et donc que l'erreur de la partition est effectivement décroissante. L'heuristique consistant à couper le multi-ensemble dont le découpage entraînera une décroissance maximum de l'erreur de la partition est donc a priori meilleure que celle consistant à couper le multi-ensemble d'erreur quadratique maximale. Toutefois cette heuristique impose de couper à chaque itération chaque multi-ensemble afin de détecter celui qui réduira l'erreur au maximum. Cela implique que l'on va générer plus de multi-ensembles que nécessaire. De plus, selon Wu, cette heuristique ne fait diminuer l'erreur quadratique globale que de façon marginale.

Il apparaît ainsi que découper le multi-ensemble d'erreur quadratique maximum à chaque étape soit une bonne heuristique permettant de réaliser un bon compromis entre le temps de calcul et la diminution de l'erreur de la partition.

De plus cette stratégie peut s'implémenter de manière très efficace en utilisant l'arbre de découpe. Il suffit d'associer à chaque noeud de l'arbre un pointeur sur la branche comportant la feuille d'erreur quadratique maximale. A chaque découpe de multi-ensemble, deux fils sont rajoutés au noeud correspondant et le pointeur est mis à jour le long du chemin menant de ce noeud à la racine de l'arbre. Cette méthode permet donc de retrouver rapidement, à chaque étape de l'algorithme, le multi-ensemble d'erreur maximum. Des études expérimentales nous ont montré que bien que l'arbre produit par les stratégies de Wan [WWP88] et Wu [WZ91] ne soit pas parfaitement équilibré le temps de recherche du multi-ensemble à découper est quasiment logarithmique.

### 3.6.3 Sélection de l'axe de découpe d'un multi-ensemble

Une fois que le multi-ensemble à découper a été sélectionné, il faut choisir la normale du plan qui va couper celui-ci. La décroissance de l'erreur de la partition induite par la coupe étant égale à $\boldsymbol{SE}(C_i^1) + \boldsymbol{SE}(C_i^2) - \boldsymbol{SE}(C_i)$, le meilleur plan de découpe est celui qui minimise la somme $\boldsymbol{SE}(C_i^1) + \boldsymbol{SE}(C_i^2)$. Malheureusement nous ne disposons pas d'outils mathématiques permettant de calculer le plan optimal et une énumération de tous les plans possibles est là encore irréaliste. On simplifie donc le problème en déterminant d'abord la normale au plan qui défini l'axe de coupe puis la position du plan sur cet axe.

Un choix simple de l'axe de coupe a été proposé par Heckbert [Hec82]. Celui-ci propose de découper l'axe de coordonnée sur lequel le multi-ensemble à découper est le plus étendu. Wu [WZ91] a étendu cette idée en choisissant la direction de plus grande variance du multi-ensemble. Cette direction est donnée par le vecteur propre associé à la plus grande valeur propre de la matrice de covariance du multi-ensemble à découper. Ce vecteur est plus simplement appelé **l'axe principal** du multi-ensemble. La justification de cette heuristique apparaît clairement lorsque l'on examine l'équation 2.1 que nous rappelons ci-dessous :

$$\boldsymbol{SE}(C) = \boldsymbol{M}_0(C) \sum_{i=1}^{3} var_i(C) \tag{3.2}$$

Si l'on effectue un changement de repère en prenant pour nouvelle base la base des vecteurs propres, l'équation 3.2 devient :

$$\boldsymbol{SE}(C) = \boldsymbol{M}_0(C) \sum_{i=1}^{3} \lambda_i \tag{3.3}$$

où $\lambda_i$ représente la $i^{eme}$ valeur propre et est égale à la variance du multi-ensemble le long de la direction définie par le $i^{eme}$ vecteur propre.

Si nous découpons le multi-ensemble perpendiculairement à un axe, nous allons faire décroître majoritairement la variance suivant cette axe. L'idée de base de l'heuristique de Wu consiste à supposer que l'on obtient une plus grande décroissance de l'erreur quadratique en faisant décroître en priorité la plus grande des variances, c'est à dire en coupant perpendiculairement à l'axe principal. Néanmoins, même si elle est très souvent vérifiée pour des images réelles cette heuristique peut être mise en défaut. En effet, considérons l'exemple de la figure 3.5 où le multi-ensemble $(C, f)$ est défini par:

$$\begin{array}{lcl} C & = & \{(i, \sqrt{7}), i \in \{1, 2, .., 10\}\} \cup \{(i, -\sqrt{7}), i \in \{1, 2, .., 10\}\} \\ f(x) & = & 1 \ \ \forall x \in C \end{array}$$

Dans cet exemple, la matrice de covariance de $(C, f)$ est alors égale à :

$$\begin{pmatrix} 8.25 & 0 \\ 0 & 7 \end{pmatrix}$$

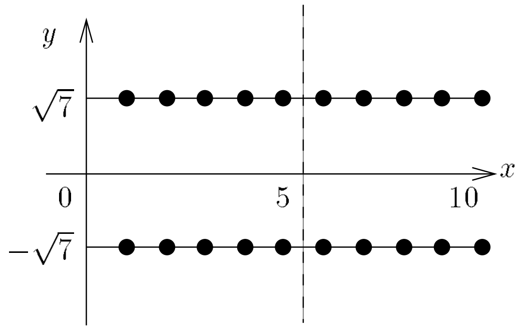



Figure 3.5: *Les points noirs sont les éléments de $(C, f)$ dont l'axe principal est le vecteur $\vec{x}$.*

Ceci signifie que l'axe principal de $(C, f)$ est le vecteur $\vec{x}$. Pourtant si nous découpons $(C, f)$ perpendiculairement au vecteur $\vec{x}$, l'erreur quadratique $\boldsymbol{SE}(C)$ ne décroîtra que de $6,25$ alors qu'un découpage perpendiculaire à $\vec{y}$ entraîne une diminution de 7. Dans ce cas, l'axe principal du multi-ensemble est donc orthogonal à la direction de coupe optimale. C'est donc la direction la plus éloignée de l'optimum.

Outre qu'elle ne conduit pas nécessairement à la meilleure solution, l'heuristique de Wu implique plusieurs contraintes algorithmiques :

- Le fait de couper perpendiculairement à une direction variable complique le traitement des données nécessaire au découpage. Par exemple, avant d'effectuer un découpage, l'algorithme de Wu doit trier les données de chaque multi-ensemble suivant leur projection sur la direction de coupe.
- Le découpage d'un multi-ensemble réclame le calcul de la matrice de covariance du multi-ensemble et le calcul de la plus grande valeur propre de cette matrice. Le calcul du vecteur propre associé s'effectue par une méthode itérative ce qui s'avère relativement coûteux.

### 3.6.4 Sélection du plan de coupe

Une fois sélectionné le multi-ensemble à découper $(C, f)$ et l'axe de coupe $A$ nous devons définir la position du plan de coupe le long de l'axe $A$. La stratégie la plus simple est encore une fois proposée par Heckbert [Hec82]. Celui-ci propose de couper $(C, f)$ par un **plan médian**, c'est à dire un plan tel que $|C_1| = |C_2|$ où $C_1$ et $C_2$ sont les deux multi-ensembles issus de la découpe de $(C, f)$.

Cette stratégie peut être grandement améliorée en tenant compte de l'erreur de la partition. En effet, soit $m$ et $M$ les deux extrémités de $(C, f)$ le long de l'axe $A$ (voir Figure 3.6 et définition 7) et $t$ un réel de l'intervalle $[m, M]$. Le plan orthogonal à $A$ et de position $t$ sur l'axe $A$ découpe $(C, f)$ en deux multi-ensembles $(C_t, f)$ et $(C - C_t, f)$. L'erreur associée à cette partition est égale à :

$$E_t(C) = \boldsymbol{SE}(C_t) + \boldsymbol{SE}(C - C_t) \quad (3.4)$$

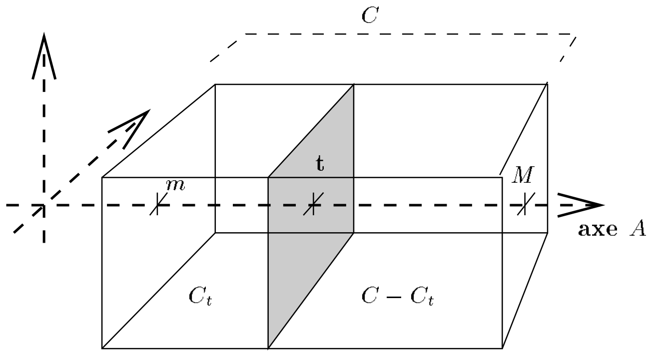



Figure 3.6: *Le multi-ensemble sélectionné est découpé orthogonalement à l'axe de coupe A positionné à la coordonnée t sur l'axe A.*

Wu [Wu92] a donné une autre formulation de l'erreur de la partition :

$$E_t(C) = \sum_{v \in C} f(v) \|v\|^2 - \left( \frac{\|M_1(C_t)\|^2}{M_0(C_t)} + \frac{\|M_1(C) - M_1(C_t)\|^2}{M_0(C) - M_0(C_t)} \right) \tag{3.5}$$

Cette nouvelle formulation est plus efficace puisqu'elle ne fait apparaître que le multi-ensemble $C_t$ que nous allons faire évoluer jusqu'à trouver la valeur $t_{opt}$ minimisant $E_t(C)$. Cette formulation présente toutefois quelques inconvénients. Tout d'abord elle impose de manipuler des grands nombres tels que les moments d'ordre un au carré. Du point de vue pratique, la présence de grands nombres oblige à stocker les variables sur des entiers ou réels longs qui ralentissent l'algorithme. De plus, cette formule se prête peu à des manipulations et peut donc difficilement être optimisée.

Wan et al. ont simplifié l'équation 3.5 en minimisant non pas l'erreur de la partition mais la somme des variances $var_A(C_t)$ et $var_A(C - C_t)$. Intuitivement cette simplification revient à approximer le multi-ensemble $(C, f)$ par sa projection sur l'axe $A$ (voir définition 7). Afin de limiter la perte d'information liée à cette simplification Wan et al. définissent $A$ comme l'axe principal de $(C, f)$. Le point 4 du théorème 2 permet alors de limiter la recherche de $t_{opt}$ à l'intervalle $[\frac{\mu+m}{2}, \frac{\mu+M}{2}]$. Cette approche permet donc de limiter l'intervalle de recherche mais ne permet pas d'obtenir la position du plan minimisant l'erreur de la partition.

Le calcul de la valeur $t_{opt}$ par les méthodes de Wu ou de Wan et al nécessite le calcul des moments $\boldsymbol{M}_0(C_t)$ et $\boldsymbol{M}_1(C_t)$ pour $t$ appartenant à l'intervalle $[m, M]$. Ces moments peuvent être calculés rapidement grâce à la propriété suivante. Supposons que le multi-ensemble $(C, f)$ défini par $C = [a_1, a_2[\times[b_1, b_2[\times[c_1, c_2[$ doit être découpé perpendiculairement au premier axe de coordonnée. Pour tout $t$ nous avons alors : $C_t = [a_1, t[\times[b_1, b_2[\times[c_1, c_2[$. Si nous définissons :

$$D_t = \{t\} \times [b_1, b_2[\times[c_1, c_2[$$

$C_t$ peut se définir comme l'union des $D_t$. Le multi-ensemble $(C, f)$ étant discret, l'intervalle $[a_1, a_2]$ est égal à $\{a_1, a_1+1, \ldots, a_2\}$ et les moments peuvent être calculés incrémentalement par les formules suivantes :

$$\forall i \in \{0, 1, 2\} \left\{ \begin{array}{l} M_i(C_{a+1}) = M_i(D_a) \\ \forall t \in ]a+1, b[, M_i(C_{t+1}) = M_i(C_t) + M_i(D_t) \end{array} \right. \tag{3.6}$$

Les moments du multi-ensemble $(C_{t_{opt}}, f)$ peuvent donc être calculés incrémentalement grâce aux équations 3.6. Les moments du multi-ensemble $(C - C_{t_{opt}}, f)$ se déduisent des moments de $(C_{t_{opt}}, f)$ par la proposition 1 sur l'additivité des moments. Une fois les moments des deux multi-ensembles calculés on peut calculer leurs moyennes, leurs variances et leurs erreurs quadratiques grâce à la définitions 3 et à l'équation 2.1. Les moments, moyennes et erreurs quadratiques de chacun des multi-ensemble sont ensuite stockés dans les feuilles de l'arbre de découpe correspondant aux multi-ensembles.

### 3.6.5 Stockage du multi-ensemble associé à l'image

Nous avons vu dans les précédentes sections que le calcul de la valeur $t_{opt}$ déterminant la position du plan de coupe nécessite le calcul des moments $M_i(D_t)$. L'efficacité de l'algorithme est donc en grande partie déterminée par la méthode permettant d'accéder à l'ensemble des couleurs de l'image contenues dans $D_t$. Une des premières méthodes envisagées pour calculer les moments de $D_t$ a été l'utilisation d'une matrice tri-dimensionelle $T$ telle que $T[i][j][k]$ soit égal au nombre de pixels de couleur $(i, j, k)$ contenus dans l'image. En utilisant une telle structure, les moments $M_i(D_t)$ sont obtenus en calculant les moments sur deux dimensions de la matrice, la troisième étant fixée à $t$. Un premier problème induit par l'utilisation de cette structure de données est la taille mémoire. En effet, stocker l'ensemble des couleurs affichables sur une image 24 bits nécessite $(256)^3 * taille(entier) = 64Mo$. Une solution à ce problème proposée par Heckbert [Hec82] consiste à effectuer une pré-quantification en ne conservant que les 5 bits de poids fort de chaque composante, la taille mémoire nécessaire est alors égale à $128Ko$. Cette pré-quantification permet de résoudre le problème de la taille mémoire mais entraîne une perte d'information sensible, notamment si on quantifie en plus de 256 couleurs. De plus, l'utilisation d'une matrice pour stocker la fréquence d'apparition de chaque couleur implique de stocker l'ensemble des couleurs affichables alors que seules les couleurs présentes dans l'image nous intéressent. Outre un gaspillage de place mémoire cette structure de données implique une perte de temps induite par la considération de couleurs qui n'appartiennent pas à l'image. Finalement l'utilisation d'une matrice impose de stocker des espaces cubiques. Une matrice est donc utilisable si l'on quantifie dans l'espace $RGB$ qui est contenu dans un cube. Elle devient en revanche inutilisable si l'on travaille dans un espace déduit de l'espace RGB par une transformation non orthogonale. De fait, la taille nécessaire pour stocker le parallélogramme englobant la transformée dans l'espace $H_1H_2H_3$ du cube RGB en 24 bits est de $512Mo$. Cette taille est réduite à $8Mo$ si l'on effectue une pré-quantification en 15 bits. De plus, au temps perdu en considérant des couleurs qui n'appartiennent pas à l'image, s'ajoute dans ce cas le temps passé à considérer des triplets ne correspondant pas à des couleurs affichables.

La méthode adoptée par Wu [WZ91, Wu92] et Wan et al. [WWP88] consiste à stocker l'ensemble des couleurs de l'image ainsi que leurs fréquences d'apparition dans un tableau uni-dimensionnel $V$. Cette méthode résoud le problème de la taille mémoire de l'histogramme. Toutefois, bien que suffisante pour les algorithmes de Wu et Wan, cette méthode ne permet pas un calcul rapide des moments des ensembles $D_t$. En effet, l'algorithme de Wu coupant chaque multi-ensemble par rapport à son axe principal, il est impossible de structurer les données lors de l'initialisation de l'algorithme. Ceci impose de trier les données suivant l'axe principal avant chaque découpe d'un multi-ensemble $(C, f)$.

## 3.7 Un algorithme de quantification basé sur une division récursive des multi-ensembles

Nous avons vu, dans la section 3.5 que les algorithmes de quantification basés sur une division récursive du multi-ensemble $(C, f)$ peuvent se décomposer en 4 étapes principales. L'ensemble des choix et heuristiques utilisés pour résoudre chacun de ces points détermine le rapport qualité/temps de calcul de chaque algorithme. Le terme qualité désignant dans ce contexte l'importance de l'erreur de partition de l'image finale. Nous allons à présent proposer un nouvel ensemble de choix et d'heuristiques permettant d'obtenir un bon rapport qualité/temps de calcul.

### 3.7.1 Choix de l'espace de couleurs

Nous avons vu dans la section 3.2 qu'il n'existait pas d'espace de couleurs de référence pour la quantification. Aucun des espaces de couleurs usuellement utilisé ($RGB$, $YIQ$,$L^*u^*v^*$) ne présentant d'avantage déterminant vis à vis des autres espaces de couleurs, nous avons choisi d'utiliser l'espace $H_1H_2H_3$. Les avantages de cet espace pour la quantification sont les suivants :

- Il se déduit facilement de l'espace $RGB$ usuellement utilisé pour coder les images.
- Les trois axes de coordonnées correspondent à trois directions privilégiées par la vision humaine.

Cet espace permet de tenir compte de notre sensibilité aux couleurs sans impliquer les coûteuses transformations induites par l'utilisation de l'espace $L^*u^*v^*$. Le choix de cet espace peut donc se concevoir comme un compromis entre l'espace $RGB$ et l'espace $L^*u^*v^*$.

### 3.7.2 Choix de la stratégie de découpe et du multi-ensemble à découper

Nous avons vu dans la section 3.5 que les deux premières heuristiques à définir pour ce type d'algorithme sont celles déterminant la stratégie de découpe et le choix du multi-ensemble à découper à chaque étape de l'algorithme. Les expériences que nous avons menées ont confirmé les observations de Wu [WZ91, Wu92] (voir section 3.6.1 et 3.6.2). Nous avons donc décidé de générer uniquement $K$ multi-ensembles et de découper à chaque étape le multi-ensemble d'erreur quadratique maximum.

### 3.7.3 Notre sélection de l'axe de découpe

Nous avons vu dans la section 3.6.3 que couper le multi-ensemble perpendiculairement à son axe principal ne conduisait pas nécessairement à la solution optimale. De plus cette stratégie impose de trier les données de chaque multi-ensemble préalablement à tout découpe. Elle nécessite également le calcul de la matrice de covariance et de l'axe principal du multi-ensemble à découper.

Ces inconvénients nous ont conduit à étudier une stratégie consistant à couper chaque multi-ensemble uniquement par rapport aux axes de coordonnées, en choisissant celui des trois fournissant la plus grande variance. Cette méthode donne généralement des résultats légèrement moins bons que ceux obtenus en coupant perpendiculairement à l'axe principal (voir Table 3.1) mais permet de supprimer le calcul du vecteur propre, de remplacer le calcul de la matrice de covariance par trois calculs de variances et enfin de trier les données suivant chacun des trois axes une fois pour toute lors de l'initialisation de l'algorithme.

| | | 1 | 2 | 3 |
|---|---|---|---|---|
| Zelda | 16 | 620 | 608 | 2% |
| | 256 | 71 | 61 | 14% |
| Lenna | 16 | 375 | 370 | 1% |
| | 256 | 62 | 56 | 9% |
| Fleurs | 16 | 1128 | 1105 | 2% |
| | 256 | 137 | 131 | 5% |
| Anemone | 16 | 944 | 909 | 4% |
| | 256 | 166 | 154 | 7% |
| Mandrill | 16 | 596 | 573 | 4% |
| | 256 | 94 | 88 | 5% |
| Moyenne | 16 | 742 | 725 | 2% |
| | 256 | 92 | 85 | 7% |

| | | |
|---|---|---|
| (1) | : | $H_1H_2H_3$ axes fixes |
| (2) | : | $H_1H_2H_3$ axe principal |
| (3) | : | différence relative des erreurs |

Table 3.1: Erreurs quadratiques calculées sur six images après une quantification en 16 et 256 couleurs à axes fixes (colonne 1) ou à axe principal (colonne 2). Les erreurs quadratiques sont calculées dans l'espace $H_1H_2H_3$. Les pourcentages affichés colonne (3) sont relatifs à la colonne 2.

### 3.7.4 Notre sélection du plan de coupe

Notre algorithme convertit donc l'ensemble des données de l'image dans l'espace $H_1H_2H_3$. Ces données sont représentées par un multi-ensemble $(C, f)$ que l'on désire partitionner en un ensemble de multi-ensembles $\{(C_1, f), \ldots, (C_K, f)\}$. L'algorithme fait évoluer cette partition en choisissant le multi-ensemble d'erreur quadratique maximum et en le découpant par un plan orthogonal à l'axe $H_1$, $H_2$ ou $H_3$ de plus grande variance. Une fois l'axe de coupe choisi, il est nécessaire de déterminer la position du plan de coupe perpendiculaire à cet axe.

Nous avons vu dans la section 3.6.4 que la formulation de l'erreur de la partition donnée par Wu [WZ91] induisait la manipulation de nombres importants et se prêtait peu à des optimisations. Nous avons donc utilisé la formulation donnée par le Théorème 3 ( voir chapitre 2) que nous rappelons ci-dessous.

$$t_{opt} = \arg \max_{t \in [m,M]} g(t) \text{ avec } \begin{cases} g(t) &= \frac{\delta(t)}{1-\delta(t)} \|\mu(t) - \mu\|^2 \\ \mu(t) &= \mu(C_t) \\ \delta(t) &= \frac{|C_t|}{|C|} \end{cases} \quad (3.7)$$

Cette nouvelle formulation permet d'utiliser des nombres plus petits stockables sur des réels courts.

Le calcul de $t_{opt}$ à l'aide des équations 3.5 ou 3.7 nécessite de parcourir l'intervalle $[m, M]$. L'équation 3.7 permet, en conjonction avec le théorème 4, d'optimiser le calcul de $t_{opt}$ :

**Théorème 5** *Avec les notations précédentes :*

$$\exists \delta_{min} \in ]0,1[ \quad / \quad \delta(t_{opt}) \in [0, \delta_{min}]$$

Nous pouvons donc arrêter la recherche de $t_{opt}$ dès que $\delta(t) > \delta_{min}$.

**Preuve:**

Nous savons par le théorème 3 que la fonction $\delta$ est croissante. Puisque nous recherchons un maximum de $g$, nous pouvons sans perte de généralité supposer $\delta$ strictement croissante.

On peut alors définir le changement de variable $x = \delta(t)$. Si nous définissons la fonction $G(x) = g \circ \delta^{-1}(x)$ le théorème 4 peut se reformuler de la façon suivante :

$$\forall x \in [0,1] \; L(x) \leq G(x) \leq U(x) \text{ avec } \begin{cases} U(x) &= x(1-x)\|\Lambda\|^2 \\ L(x) &= \chi_{[0,\frac{1}{2}]} \frac{x}{1-x} \alpha + \chi_{]\frac{1}{2},1]} \frac{1-x}{x} \alpha \end{cases}$$

où $\chi$ désigne la fonction caractéristique.

Soit $U'$ la restriction de $U$ à l'intervalle $[\frac{1}{2}, 1]$. $U'$ est strictement décroissante, donc bijective, de plus G est majorée par $U$, donc $G([0,1]) \subseteq U'([0,1])$. On peut donc définir la fonction $P$ (voir Figure 3.7) :

$$P \left( \begin{array}{ccc} [0,1] & \rightarrow & [\frac{1}{2}, 1] \\ x & \mapsto & U'^{-1} \circ G(x) \end{array} \right.$$

Soit :

$$\delta_{min} = \min_{x \in [0,1]} P(x)$$

$U'$ étant décroissante on a :

$$\delta_{min} = U'^{-1}(\max_{x \in [0,1]} G(x)) = U'^{-1}(G(\delta(t_{opt}))$$

d'où

$$U'(\delta_{min}) = G(\delta_{topt})$$

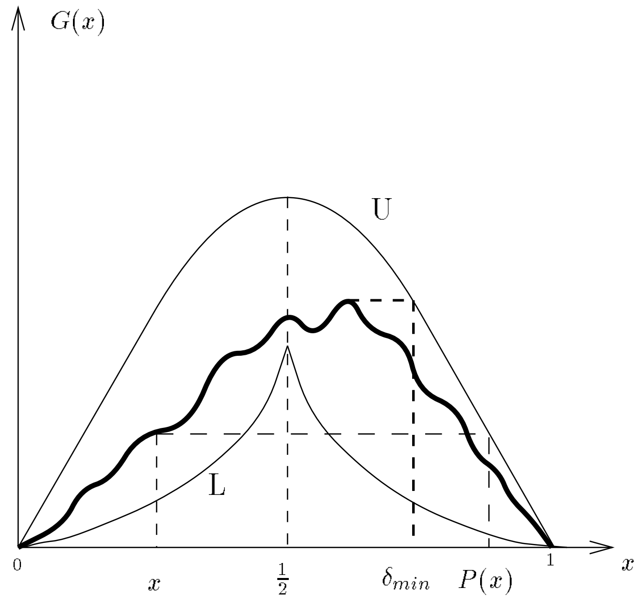



Figure 3.7: *Illustration de la fonction $P$ et du calcul de $\delta_{min}$.*

et

$$\forall x \in ]\delta_{min}, 1] \; G(x) \leq U'(x) < U'(\delta_{min}) = G(\delta(t_{opt}))$$

d'où finalement : $\delta(t_{opt}) \in [0, \delta_{min}]$

□

Cette dernière formule permet une réduction de l'intervalle de recherche toute théorique. En effet, si nous savons qu'il est inutile de continuer la recherche au delà de $\delta_{min}$, le calcul de $\delta_{min}$ impose de traverser tout l'intervalle de recherche. Ce dernier problème est résolu par la proposition suivante :

**Proposition 4** *Avec les notations précédentes :*

$$\forall x \in [0,1] \; \left(x > Min_{y\in[0,x]}P(y) \Longrightarrow x > \delta_{min}\right)$$

**Preuve:**

Si nous définissons :

$$\left\{ \begin{array}{lcl} y_0 & = & ArgMin_{y\in[0,x]}P(y) \\ x_0 & = & \min_{y\in[0,x]} P(y) \end{array} \right.$$

On a :

$$U'(x_0) = G(y_0) \leq G(\delta(t_{opt})) = U'(\delta_{min})$$

$U'$ étant décroissante on a :

$$x > x_0 \geq \delta_{min}$$

□

L'algorithme de recherche de la valeur $t_{opt}$ peut donc se ramener aux points mis en évidences dans la Figure 3.8.

```
determiner_t_opt()
Debut
    max = 0
    Pour tout t de l'intervalle [m, M]
        evaluer g(t)
        si (g(t) > max)
            max = g(t)
            t_opt = t

        mettre a jour min = min_{t' ∈ [0,t]} P(δ(t))
        si (δ(t) > min)
            retourner t_opt
    Fin Pour
Fin
```

Figure 3.8: *Algorithme de recherche de la valeur* $t_{opt}$.

La valeur $\delta_{min}$ étant supérieure à $\frac{1}{2}$ nous devons parcourir l'intervalle $[m, M]$ au moins jusqu'à ce que $\delta(t) = \frac{1}{2}$. La fonction de fréquence liée à l'image intervenant dans le calcul de la fonction $\delta$, la réduction de l'intervalle de recherche dépendra de l'image. De plus, chaque itération de l'algorithme traitant un nouveau multi-ensemble le gain variera d'une itération à l'autre. Des expériences nous ont montré que lors de quantifications en 16 couleurs, l'intervalle de recherche était réduit d'environ 10%.

Puisque $G$ est minorée par la fonction $L$ non nulle sur $]0, 1[$, l'erreur de la partition $E_t(C)$ est strictement inférieure à l'erreur quadratique $\boldsymbol{SE}(C)$ pour tout $t$ tel que $\delta(t) \in ]0, 1[$. Ce résultat peut s'interpréter comme la preuve qu'un multi-ensemble est mieux représenté par deux points que par un seul, ce qui est intuitivement évident. Les cas extrêmes $\delta = 0$ et $\delta = 1$ correspondent aux positions du plan de coupe telles que le multi-ensemble $C_t$ est vide ou égal à $C$. Dans ce cas le multi-ensemble $(C, f)$ n'est pas découpé. On peut également remarquer que puisque les fonctions $U$ et $L$ atteignent leur maximum pour $\delta(t) = \frac{1}{2}$ le maximum de $g(t)$ a de fortes chances d'être atteint pour $t$ proche de $\delta^{-1}(\frac{1}{2})$. Cette valeur de $t$ correspond à la position médiane du plan de coupe sélectionnée par Heckbert [Hec82] (voir section 3.6.4). De ce point de vue, l'heuristique employée par Heckbert peut être vue comme une approximation des méthodes minimisant l'erreur quadratique de la partition.

### 3.7.5 Notre structure de données

Nous avons vu dans la section 3.6.5 que le multi-ensemble 3D correspondant à l'image était usuellement stocké dans une matrice 3D T ou dans un tableau unidimensionnel $V$. L'utilisation du tableau $V$ introduite par Wu [WZ91] nécessite moins de place en mémoire que l'utilisation du tableau 3D $T$. Toutefois cette structure de données ne permet pas un accès efficace aux données des multi-ensembles $D_t$ (voir section 3.6.4) et implique un tri d'une partie du tableau $V$ avant chaque découpe.

Afin de structurer l'ensemble des couleurs de l'image nous reprenons le tableau $V$ utilisé par Wu [WZ91, Wu92] et Wan et al [WWP88]. On trie ensuite le tableau $V$ suivant chacun des axes de coordonnées $c_1$, $c_2$ et $c_3$. Le résultat de ces tris est stocké dans trois tableaux $Ind_{c_1}$, $Ind_{c_2}$ et $Ind_{c_3}$. Les tableaux $Ind_{c_i}$ ne contiennent pas les couleurs de l'image mais simplement des indices du tableau $V$ (voir Figure 3.9).

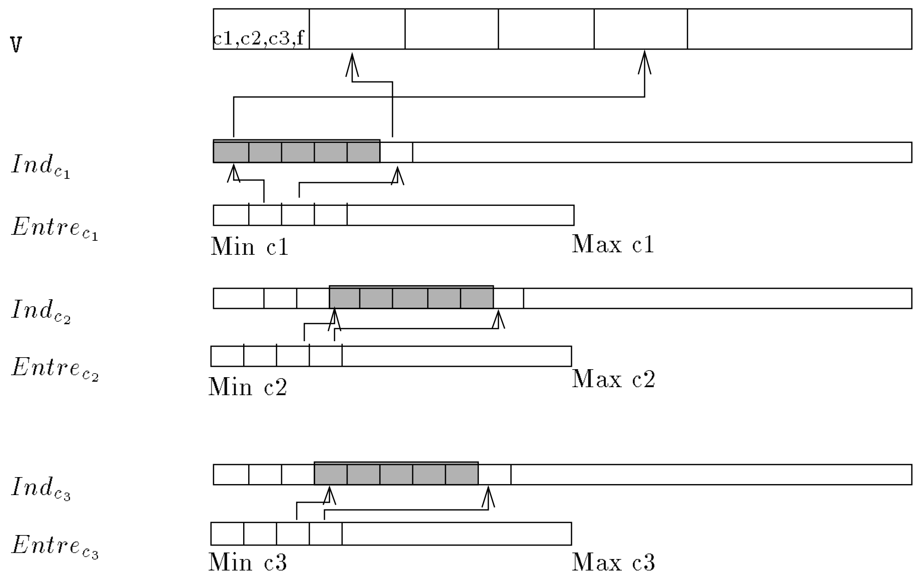



Figure 3.9: *L'ensemble des couleurs de l'image est trié suivant chaque axe.*

On a alors la propriété suivante :

$$\forall i \in \{1,2,3\}\ \forall (k,l) \in [Min_i, Max_i]^2\ k \leq l \Longrightarrow V[Ind_{c_i}[k]].c_i \leq V[Ind_{c_i}[l]].c_i$$

où $Min_i$ et $Max_i$ sont les coordonnées minimale et maximale des éléments du multi-ensemble sur l'axe $c_i$.

On obtient alors dans les tableaux $Ind_{c_1}$ des plages d'indices référençant des cellules de $V$ possédant une même valeur sur l'axe $c_1$. On trie donc chacune de ces plages suivant $c_2$. On applique le même traitement aux tableaux $Ind_{c_2}$ et $Ind_{c_3}$ dont les plages sont triées suivant $c_1$. On définit enfin trois tableaux $Entre_{c_1}$, $Entre_{c_2}$ et $Entre_{c_3}$ nous permettant d'accéder directement au début d'une plage donnée. Ces tableaux vérifient la propriété suivante :

$$\forall j \in \{1,2,3\}\ \ \forall k \in [Min_j, Max_j]\ \ \forall\ l \geq Entre_{c_j}[k]\ \ V[Ind_{c_j}[l]].c_j \geq k$$

Supposons que $D_t$ soit défini par :

$$D_t = \{t\} \times [m_2, M_2[ \times [m_3, M_3[$$

Trouver l'ensemble des couleurs de l'image contenues dans $D_t$ s'effectue alors en trois étapes :

1. Trouver l'ensemble des couleurs vérifiant $(c_1 = t)$ grâce à $Entre_{c_1}$. Cet ensemble constitue une plage codée de façon continue dans $Ind_{c_1}$. Le début et la fin de la plage ont respectivement pour indice $Entre_{c_1}[t]$ et $Entre_{c_1}[t+1]-1$.

2. Chercher par dichotomie dans la plage précédemment définie les indices $j$ vérifiant $m_2 \leq V[j].c_2 < M_2$. Ces indices sont stockés de façon continue dans $Ind_{c_1}$.

3. Parcourir l'ensemble des indices $j$ trouvés à l'étape 2 pour ne retenir que ceux vérifiant $m_3 \leq V[j].c_3 < M_3$.

Cette structure de données permet donc :

- de stocker et traiter uniquement les couleurs effectivement présentes dans l'image,
- d'accéder très rapidement aux données nécessaires à l'algorithme de partitionnement.

### 3.7.6 Évaluation de l'heuristique proposée

Nous avons vu qu'un algorithme de quantification basé sur une division récursive du multi-ensemble associé à l'image doit itérer quatre processus élémentaires jusqu'à l'obtention du nombre de multi-ensembles désirés.

1. Trouver le multi-ensemble à découper parmi les multi-ensembles créés aux itérations précédentes.
2. Sélectionner un ensemble d'axes de coupe possibles.
3. Choisir un critère pour sélectionner l'axe de coupe.
4. Déterminer le plan de coupe normal à l'axe de coupe.

En accord avec cette décomposition les principales méthodes de quantification présentées dans les sections précédentes sont résumées dans les colonnes 1 à 3 du tableau 3.2.

La première méthode est la coupe médiane classique présentée par Heckbert [Hec82]. Cette méthode résumée sur la colonne 1 du tableau 3.2 découpe le multi-ensemble initial en $K$ multi-ensembles $\{C_1, \ldots, C_K\}$ de cardinal $|C_i|$ approximativement égaux. Elle n'a donc pas à calculer les moments $\boldsymbol{M}_1$ et $\boldsymbol{M}_2$ des multi-ensembles $C_i$ et est la moins coûteuse des méthodes présentées dans le tableau 3.2. Lorsque le nombre de couleurs est supérieur ou égal à 256 la différence entre l'image quantifiée et l'image originale est peu perceptible. Ce dernier point explique en partie le fait que l'algorithme de découpage médian reste plus populaire que de nombreuse méthodes plus sophistiquées. Cette popularité s'explique également par l'aspect fondateur de cette méthode élaborée en 1982. Toutefois lorsque l'on quantifie en moins de 32 couleurs les performances de l'algorithme se dégradent et l'on obtient des images de moins bonne qualité que celles obtenues en utilisant des algorithmes plus sophistiqués. Ce fait est illustré sur la Planche 3.10 où l'image 3.10-(a) semble avoir perdu des couleurs importantes de l'image.

L'heuristique de Wan et al. [WWP88] résumée colonne 2 utilise l'erreur quadratique à chacune des étapes de l'algorithme. La principale simplification opérée par cet algorithme est effectuée lors du choix du plan de coupe où Wan et al. approximent le multi-ensemble

| | | 1 | 2 | 3 | 4 |
|---|---|---|---|---|---|
| Choix du multi-ensemble | Erreur quadratique | | ⋆ | ⋆ | ⋆ |
| | Cardinal | ⋆ | | | |
| Choix d'un ensemble d'axes | Axes de coordonnées | ⋆ | | | ⋆ |
| | Axe principal | | ⋆ | ⋆ | |
| Choix de l'axe | Plus grande longueur | ⋆ | | | |
| | Plus grande variance | | ⋆ | ⋆ | ⋆ |
| Choix du plan | Plan médian | ⋆ | | | |
| | Minimisation de la variance | | ⋆ | | |
| | Minimisation de l'erreur | | | ⋆ | ⋆ |

| | | |
|---|---|---|
| (1) | : | Méthode de Heckbert |
| (2) | : | Méthode de Wan and Wong |
| (3) | : | Méthode de Wu |
| (4) | : | Méthode proposée. |

Table 3.2: Ce tableau résume les principales heuristiques employées par les algorithmes de quantification basés sur une division récursive du multi-ensemble de départ.

par sa projection sur l'axe principal. L'ensemble des heuristiques utilisées par Wan et al. permettent d'obtenir des images quantifiées proches des images originales lorsque le nombre de couleurs représentatives est supérieur à 32 couleurs. En dessous de ce nombre l'algorithme préserve les principales couleurs de l'image. Néanmoins, des défauts visuels comme des "cassures" de dégradés montrent alors clairement l'aspect quantifié de l'image (voir Image 3.10-(b))

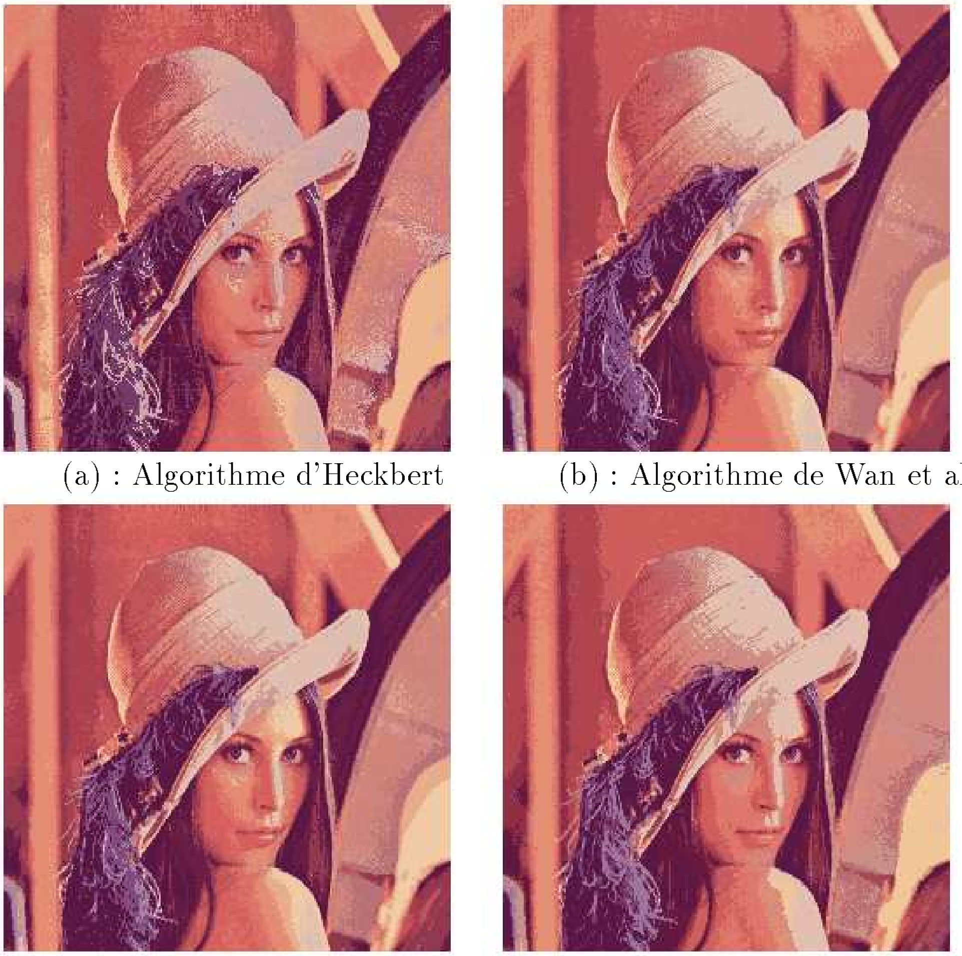
L'heuristique de Wu [WZ91] est résumée sur la colonne 3 de la table 3.2. Cette méthode donne une image quantifiée proche de l'image originale pour des quantifications utilisant au moins 64 couleurs. Si l'on utilise moins de 64 couleurs la méthode de Wu préserve les couleurs les plus importantes de l'image sans empêcher une dégradation visuelle de l'image quantifiée (voir Image 3.10-(c)). Cette méthode effectue le calcul de la matrice de covariance de chaque multi-ensemble et trie les données de celui-ci avant chaque découpe. En raison de ces traitements la méthode de Wu est la plus coûteuse des méthodes présentées dans le tableau 3.2.

Les tests résumées dans la table 3.3 montrent que l'utilisation d'axes fixes permet de diminuer les temps de calcul d'environ 35%. Ce gain est à mettre en parallèle avec l'accroissement de l'erreur quadratique provoquée par l'utilisation d'axes fixes. On constate sur la table 3.3 que l'erreur quadratique ne s'accroit que de 2% pour une quantification en 16 couleurs et de 7% pour une quantification en 256 couleurs. Ces pourcentages sont relatifs à l'erreur quadratique de la méthode qui coupe perpendiculairement à l'axe principal. L'augmentation de 7% pour une quantification en 256 couleurs est relativement peu importante, puisque pour un tel nombre de couleurs il est difficile d'apprécier visuellement la différence entre deux algorithmes de quantification.

| | | | Temps | | $E$ | | |
|---|---|---|---|---|---|---|---|
| Image | Taille | K | 1 | 2 | 3 | 4 | 5 |
| Zelda | 704⋆574 | 16 | 42 | 1'06 | 620 | 608 | 2% |
| | | 256 | 52 | 1'41 | 71 | 61 | 14% |
| Lenna | 512⋆480 | 16 | 29 | 47 | 375 | 370 | 1% |
| | | 256 | 36 | 1'15 | 62 | 56 | 9% |
| Fleurs | 462⋆416 | 16 | 43 | 1'18 | 1128 | 1105 | 2% |
| | | 256 | 49 | 2'06 | 137 | 131 | 5% |
| Anemone 3.10 | 722⋆471 | 16 | 1'13 | 2'00 | 944 | 909 | 4% |
| | | 256 | 1'26 | 3'05 | 166 | 154 | 7% |
| Mandrill | 510⋆480 | 16 | 36 | 1'00 | 596 | 573 | 4% |
| | | 256 | 41 | 1'34 | 94 | 88 | 5% |
| Moyenne | | 16 | 44 | 1'14 | 742 | 725 | 2% |
| | | 256 | 52 | 1'56 | 92 | 85 | 7% |

| | | |
|---|---|---|
| (1) | : | Temps requis par l'heuristique utilisant des axes fixes dans l'espace $H_1H_2H_3$ |
| (2) | : | Temps requis par l'heuristique utilisant les axes principaux dans l'espace $H_1H_2H_3$ |
| (3) | : | Erreur quadratique obtenue en utilisant les axes fixes dans l'espace $H_1H_2H_3$ |
| (4) | : | Erreur quadratique obtenue en utilisant les axes principaux dans l'espace $H_1H_2H_3$ |
| (5) | : | Difference relative des colonnes 3 et 4 |

Table 3.3: Cette table montre le temps requis (colonne 1 et 2) et l'erreur quadratique obtenue (colonne 3 et 4) par deux algorithmes. Le premier coupe les multi-ensembles perpendiculairement à leur axe de coordonnées de plus grande variance, le second coupe les multi-ensembles perpendiculairement à leur axe principal. Les pourcentages de la colonne (5) sont relatifs aux résultats de la colonne (3).



(a) : Algorithme d'Heckbert (b) : Algorithme de Wan et al.

(c) : Algorithme de Wu (d) : Notre algorithme

Figure 3.10: *Image Lenna quantifiée par 4 algorithmes différents en 16 couleurs dans l'espace $H_1H_2H_3$.*

## 3.8 Une méthode de quantification basée sur une approche découpe-fusion

Nous avons vu dans la section 3.5 que les méthodes de quantification adaptatives pouvaient se scinder en deux grandes familles :

1. Les méthodes de fusion [GP90, XJ94] qui analysent l'ensemble des pixels de l'image en les fusionnant aux $K$ clusters qui formeront la partition finale.
2. Les méthodes de découpe [Hec82, WWP88, WZ91, Wu92, BBA94] qui partitionnent le multi-ensemble $(C, f)$ des couleurs de l'image en $K$ multi-ensembles.

Dans cette section nous proposons une nouvelle approche basée sur une étape de découpe suivie d'une étape de fusion. Cette approche a suscité de nombreux travaux [HP75, HP76, CMVM86, Lee86] en segmentation mais n'a jusqu'à présent pas été appliquée aux problèmes de quantification.

Le principe de l'algorithme est le suivant : on effectue en premier lieu une quantification uniforme basée sur une partition implicite du cube $RGB$. Ainsi le multi-ensemble $(C, f)$ des couleurs de l'image est partitionné en $N$ multi-ensembles. Ces multi-ensembles sont alors fusionnés de façon à obtenir la partition finale en $K$ multi-ensembles. La fusion de deux multi-ensembles $(C_1, f_1)$ et $(C_2, f_2)$ est définie comme le multi-ensemble $(C, f)$ avec $C = C_1 \cup C_2$ et $f = f_1 + f_2$. L'opération de fusion correspond donc à l'opération somme définie au chapitre 2 (voir définition 5).

### 3.8.1 Description de l'algorithme

Notre méthode utilise une quantification uniforme pour produire $N$ multi-ensembles avec $N$ plus grand que le nombre de multi-ensembles finaux $K$. Elle fait ensuite appel à un algorithme de fusion utilisant une structure de données spécifique appelée **Graphe d'Adjacence de Multi-ensembles (GAM)** par référence aux **Graphe d'Adjacence de Régions** utilisés dans les algorithmes de segmentation [BGK⁺89]. Nous avons implémenté le **GAM** avec une matrice triangulaire inférieure comportant $\frac{N(N-1)}{2}$ éléments. Le **GAM** est initialement composé de $N$ noeuds, chaque noeud correspondant à un multi-ensemble. Le but de l'algorithme de fusion est de minimiser l'erreur de la partition donnée par la définition 6. Nous stockons donc dans chaque noeud des paramètres permettant une mise à jour efficace de l'erreur de la partition après la fusion de deux multi-ensembles. Ensuite, afin d'obtenir les $K$ multi-ensembles finaux, nous effectuons $N - K$ fusions. A chaque étape, nous sélectionnons les deux noeuds dont la fusion produira la plus petite augmentation de l'erreur de la partition. Pour cela, nous valuons chaque arête du **GAM** par l'augmentation de l'erreur quadratique induite par la fusion des deux noeuds partageant cette arête. L'ensemble des arêtes d'un noeud résultant de la fusion de deux noeuds est égal à l'union des arêtes des deux noeuds (voir Figure 3.11). En termes de graphes cette opération est appelée une contraction d'arêtes.

A chaque étape de fusion la détermination des deux noeuds à fusionner peut s'effectuer sur l'ensemble des couples de noeuds, auquel cas le **GAM** correspond à un graphe com-

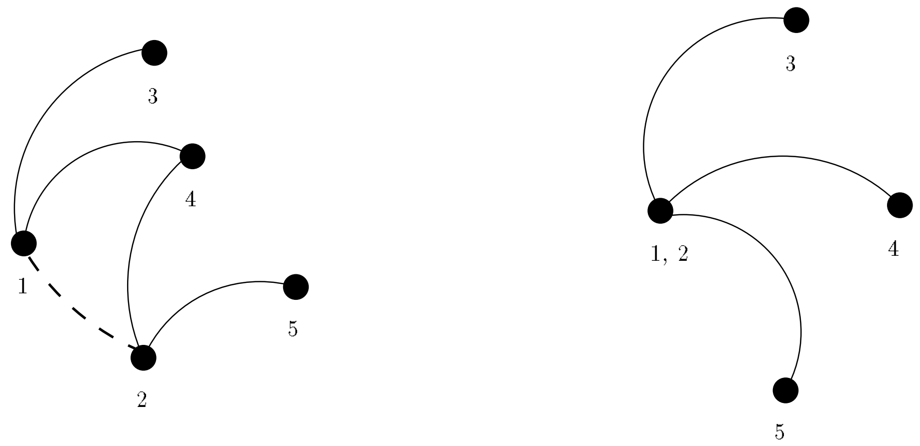



Figure 3.11: *Contraction d'une arête du* **GAM**.

plet (un graphe où tous les noeuds sont reliés par une arête) ou sur l'ensemble des noeuds correspondant à des multi-ensembles adjacents. Il est bien évident que si nous considérons les relations d'adjacence nous restreignons l'ensemble des couples considérés. L'erreur finale de la partition doit donc être plus importante. Ceci est confirmé par les expériences résumées Table 3.4. Les temps requis par notre algorithme avec un **GAM** complet ou avec un **GAM** restreint par l'adjacence des multi-ensembles étant à peu près équivalents nous avons choisi d'utiliser un **GAM** complet (voir Table 3.5). Nous avons de plus utilisé l'algorithme utilisant le **GAM** complet sur plusieurs images sans observer d'aberrations visuelles dues à la fusion de multi-ensembles non-adjacents.

**Remarque 6** *Les temps équivalents constatés Table 3.5 sont sans doute dus à l'utilisation de la même structure de données pour les deux types de fusions. Étant donné le temps moyen de quantification d'une image avec les deux approches (environ 2 secondes), nous avons préféré privilégier la diminution de l'erreur de la partition. L'utilisation d'une structure de données plus adaptée au graphe restreint induit par les relations d'adjacences permettrait sans doute d'obtenir des temps nettement inférieurs. Si ces temps étaient inférieurs au quart de seconde, il serait alors envisageable d'utiliser notre algorithme pour l'animation de séquences d'images.*

L'algorithme peut se ramener aux points suivants :

### 3.8.2 Valuation des arêtes du GAM

Nous avons vu que chaque arête du graphe est valuée par l'augmentation de l'erreur de la partition induite par la fusion des deux noeuds partageant cette arête. Afin de calculer cette augmentation il nous faut calculer l'erreur quadratique d'un multi-ensemble issu de la fusion de deux multi-ensembles. En vertu du théorème 1, l'erreur quadratique du multi-ensemble $(C, f) = (C_1, f_1) + (C_2, f_2)$ est égale à :

$$E(C) = E(C_1) + E(C_2) + \frac{|C_1||C_2|}{|C_1| + |C_2|}\|\mu(C_1) - \mu(C_2)\|^2$$

```
decoupe_et_fusionne((C, f), N, K)
{
    Generer N multi-ensembles grace a la quantification uniforme.
    Initialiser un graphe G a partir de ces multi-ensembles
    tant que(N > K)
    {
        Trouver l'arete minimale de G : (node1, node2)
        Fusionner les noeuds node1 et node2
        N = N - 1
    }
}
```

Figure 3.12: *Principales étapes de notre algorithme.*

| ***E*** | 1 | 2 |
|---|---|---|
| Lenna | 254 | 257 |
| Zelda | 402 | 396 |
| Girl | 313 | 278 |
| House | 147 | 144 |
| Mean | 279 | 268 |

| | | |
|---|---|---|
| (1) | : | Notre algorithme avec un **GAM** déduit de la 26-connexité des multi-ensembles |
| (2) | : | Notre algorithme avec un **GAM** complet |

Table 3.4: Cette table montre la diminution de l'erreur de la partition induite par l'utilisation d'un **GAM** complet. Les erreurs de cette table ont été calculées dans l'espace RGB et obtenues à partir de 2048 multi-ensembles initiaux.

| $s$ | 1 | 2 |
|---|---|---|
| Lenna | 2.4 | 2.2 |
| Zelda | 3.6 | 3.3 |
| Girl | 1.1 | 0.9 |
| House | 0.9 | 0.8 |
| Mean | 2.0 | 1.8 |

| | | |
|---|---|---|
| (1) | : | Notre algorithme avec un **GAM** déduit de la 26-connexité des multi-ensembles |
| (2) | : | Notre algorithme avec un **GAM** complet |

Table 3.5: Cette table montre les temps nécessaires à la quantification d'une image avec un **GAM** complet et un **GAM** utilisant la 26-connexité des multi-ensembles. Ces temps ont été obtenus pour 2048 multi-ensembles initiaux.

Où $\mu_1$ et $\mu_2$ sont les moyennes des multi-ensembles $(C_1, f_1)$ et $(C_2, f_2)$. Si nous définissons $E_p$ comme l'erreur de la partition après $p$ fusions, on a alors le théorème suivant :

**Théorème 6** *Soit $E_p$ l'erreur quadratique de la partition après p fusions. La suite $E_p$ avec $p \in \{1, \ldots, N-K\}$ peut être définie comme suit :*

$$E_{p+1} = E_p + \frac{|C_{i_p}||C_{j_p}|}{|C_{i_p}| + |C_{j_p}|}||\mu_{i_p} - \mu_{j_p}||^2 = E_p + \Delta(i_p, j_p)$$

*où $i_p$ et $j_p$ représentent les deux noeuds fusionnés à l'étape p.*

**Preuve:**

Évident au vu de la définition 6 et du théorème 1.
□

Donc à chaque étape de l'algorithme, la fusion des noeuds $i$ et $j$ va augmenter l'erreur de la partition de $\Delta(i,j)$. Nous valuons donc chaque arête d'extrémités $i$ et $j$ par $\Delta(i,j)$. Si le **GAM** est implémenté à l'aide d'une matrice triangulaire inférieure $G$, l'étape d'initialisation consiste à affecter à chaque cellule $G[i][j]$ la valeur $\Delta(i,j)$. Lors de la première étape de fusion les deux noeuds $n_1$ et $n_2$ qui doivent être fusionnés sont définis par :

$$(n_1, n_2) = ArgMin_{(i,j)\in\{1,\ldots,N\}\star\{1,\ldots,i-1\}} G[i][j] \tag{3.8}$$

où $ArgMin$ renvoie l'argument réalisant le minimum.

Nous avons optimisé cette recherche en ajoutant au tableau $G$ un tableau $Min$ défini par :

$$\forall i \in \{1, \ldots, N\} Min[i] = ArgMin_{j\in\{1,\ldots,i-1\}} G[i][j]$$

L'équation 3.8 devient alors:

$$(n_1, n_2) = \left(ArgMin_{i\in\{1,\ldots,N\}} G[i][Min[i]], Min[n_1]\right)$$

Si nous invalidons le noeud $n_2$, la mise à jour de la matrice $G$ nécessite d'invalider la ligne et la colonne $n_2$ et de mettre à jour la ligne et la colonne $n_1$. De plus nous devons également mettre à jour la valeur $Min[i]$ si $Min[i]$ est égal à $n_2$ ou si la mise à jour de la ligne et de la colonne $n_1$ a changé le minimum d'une ligne.

Si nous examinons le théorème 6, on constate que le calcul de la fonction de valuation $\Delta$ nécessite pour chaque noeud la moyenne et le cardinal du multi-ensemble associé. La moyenne pouvant se déduire facilement du moment d'ordre 1 et du cardinal (voir définition 3), nous avons préféré stocker dans chaque multi-ensemble ses moments d'ordre 0 et 1. Les moments d'un multi-ensemble issus de la fusion de deux multi-ensembles se calculent alors simplement par l'additivité des moments définie par la propriété 1 au chapitre 2.

La fusion de deux noeuds peut donc se décomposer comme suit :

```
fusionne_noeuds (G, noeud_1, noeud_2)
{
    |C_noeud_1| = |C_noeud_1| + |C_noeud_2|
    M_1(C_noeud_1) = M_1(C_noeud_1) + M_1(C_noeud_2)
    μ = M_1(C_noeud_1)/|C_noeud_1|
    Utiliser G, μ and |C_noeud_1| pour mettre a jour
        les aretes partagees par noeud_1 ou noeud_2
    Mettre a jour le tableau Min
}
```

Figure 3.13: *Fusion des noeuds* $node_1$ *et* $node_2$.

### 3.8.3 Inversion de la table de couleurs

Nous avons vu dans la section 3.8.1 que l'algorithme peut se décomposer en deux étapes : la génération de $N$ multi-ensembles $\{\boldsymbol{C}_1, \ldots, \boldsymbol{C}_N\}$ par l'algorithme de découpe uniforme et la fusion de ces multi-ensembles pour obtenir les $K$ **méta multi-ensembles** finaux $\{\boldsymbol{MC}_1, \ldots, \boldsymbol{MC}_K\}$ (chaque méta multi-ensemble $\boldsymbol{MC}_i$ est représenté par un multi-ensemble $\boldsymbol{C}_j$ contenant des informations de fusion suplémentaires).

Nous avons vu dans la section 3.4 que l'inversion de table de couleurs est réalisée par la fonction $\boldsymbol{Q}$ qui affecte à chaque couleur de l'image sa couleur représentative la plus proche. Si $\{c_1, \ldots, c_K\}$ représente l'ensemble des couleurs représentatives, l'image $\boldsymbol{Q}(c)$ d'une couleur $c$ est définie par :

$$\boldsymbol{Q}(c) = ArgMin_{z \in \{c_1, \ldots, c_K\}} \|z - c\|$$

Ce calcul peut être réalisé par une recherche exhaustive du minimum de $\|z - c\|$ pour tout z dans $\{c_1, \ldots, c_K\}$. On peut alléger ce calcul en utilisant un diagramme de Voronoi (voir Thomas [Tho91]) ou en rejetant a priori certaines couleurs représentatives qui ne peuvent réaliser le minimum (voir Heckbert [Hec82]). On peut également approximer ce calcul en affectant à chaque couleur $c$ la moyenne de son multi-ensemble englobant.

Dans cette section nous présentons deux algorithmes d'inversion de table de couleurs. Le premier approxime la fonction $\boldsymbol{Q}$ en affectant chaque couleur à la moyenne de son méta multi-ensemble englobant. La seconde fournit une fonction $\boldsymbol{Q}$ presque toujours égale à la solution exacte fournie par une recherche exhaustive. Les deux algorithmes sont basés sur le même prétraitement. Ce prétraitement permet de trouver le méta multi-ensemble $\boldsymbol{MC}_i$ englobant le multi-ensemble $\boldsymbol{C}_i$ avec une seule indirection.

Afin de réaliser cette indirection nous ajoutons un index canonique à chaque multi-ensemble. L'index canonique d'un multi-ensemble $\boldsymbol{C}_i$ est initialisé à son index $i$. Lorsque nous fusionnons deux multi-ensembles $C_i$ et $C_j$ toutes les informations de fusion sont stockées dans le multi-ensemble de plus grand indice. Donc, si nous fusionnons le multi-ensemble $\boldsymbol{C}_1$ avec $\boldsymbol{C}_3$ puis $\boldsymbol{C}_3$ avec $\boldsymbol{C}_5$ on a :

$$\boldsymbol{C}_1.canonique = 3, \boldsymbol{C}_3.canonique = 5, \boldsymbol{C}_5.canonique = 5$$

Donc les multi-ensembles $C_1$, $C_3$ et $C_5$ ont été fusionnés et l'information de fusion est conservée dans $C_5$. Partant du multi-ensemble $C_1$, on doit exécuter deux indirections pour retrouver le multi-ensemble conservant les informations de fusion. Ces indirections peuvent être supprimées grâce à l'algorithme suivant :

```
Fixer_canonique (N)
{
    Pour i = N a 0
    {
        j = C_i.canonique
        C_i.canonique = C_j.canonique
    }
}
```

Figure 3.14: *Suppression des indirections.*

Après l'algorithme `Fixer_canonique` tous les multi-ensembles qui ont été fusionnés dans un même méta-multi-ensemble partagent le même index canonique. Les méta-multi-ensembles peuvent donc être définis de la façon suivante :

$$\forall i \in \{1, \ldots, K\}\ \boldsymbol{MC}_i = \bigsqcup_{\{k/\boldsymbol{C}_k.canonique=i\}} \boldsymbol{C}_k$$

Notre premier algorithme d'inversion de table de couleurs décompose le calcul de l'image $\boldsymbol{Q}(c)$ d'une couleur $c$ en deux opérations. Tout d'abord il trouve le multi-ensemble $\boldsymbol{C}_i$ généré par l'algorithme de découpe uniforme qui contient la couleur $c$. Ensuite, étant donné le multi-ensemble $\boldsymbol{C}_i$ il trouve le méta multi-ensemble $\boldsymbol{MC}_i$ qui contient $\boldsymbol{C}_i$. L'index de ce méta multi-ensemble est égal à $\boldsymbol{C}_i.canonique$. Ces deux opérations conduisent à l'algorithme `inverse_colormap` (voir Figure 3.15) qui calcule l'image par la fonction $\boldsymbol{Q}$ de toutes les couleurs de l'image.

```
inverse_colormap ()
{
    Pour chaque couleur c de l'image
    {
        C_j = quantification_uniforme_donne_multi_ensemble()
        k = C_j.canonic
        Ecrire la couleur representative c_k associe a C_k
            dans l'image quantifiee.
    }
}
```

Figure 3.15: *Algorithme d'inversion de table de couleurs.*

Pour chaque couleur de l'image, le multi-ensemble englobant généré par l'algorithme de découpe uniforme est retrouvé avec seulement deux multiplications. Le calcul de $\boldsymbol{Q}(c)$ s'effectue donc à l'aide de deux multiplications et d'une indirection. La complexité de la fonction $\boldsymbol{Q}$ est donc constante. De plus l'algorithme `Fixer_canonique` (voir Figure 3.14) a une complexité égale à $\mathcal{O}(N)$. Des valeurs typiques de $N$ étant de l'ordre de 1024 ou 2048, $N$ est négligeable par rapport à la taille de l'image et le sur-coût en temps induit par l'algorithme `Fixer_canonique` est également négligeable par rapport au calcul de $\boldsymbol{Q}(c)$ pour toutes les couleurs de l'image.

Notre second algorithme d'inversion de table de couleurs peut être vu comme une adaptation à notre structure de données de l'algorithme de limitation de la recherche exhaustive décrit par Heckbert [Hec82]. L'idée de base de cet algorithme est que les couleurs mal classifiées par l'algorithme `inverse_colormap` appartiennent à des multi-ensembles situés sur la frontière des méta multi-ensembles. Donc pour chaque multi-ensemble $\boldsymbol{C}_i$ nous calculons la liste $L_i$ définie par :

$$L_i = \{C_i.canonique\} \cup \{C_j.canonique \ / \ C_j \ adj \ C_i \ \text{ et } C_j.canonique \neq C_i.canonique\}$$

où *adj* est une relation d'adjacence entre les multi-ensembles générés par l'algorithme de découpe uniforme. L'algorithme de découpe uniforme fournit une partition du multi-ensemble initial en un ensemble de cubes. Donc, les 6,18 et 26 connexités des voxels [And92] peuvent être utilisées sur les multi-ensembles générés par l'algorithme de découpe uniforme.

En d'autres mots, $L_i$ contient la liste des méta multi-ensembles adjacents au multi-ensemble $\boldsymbol{C}_i$. Nous modifions alors l'algorithme `inverse_colormap` de la façon suivante :

```
inverse_colormap_ameliore ()
{

    Pour chaque couleur c de l'image
    {
        C_j = quantification_uniforme_donne_multi_ensemble()
        k = ArgMin_{p∈L_j} ||c − c_p||
        Ecrire la couleur representative c_k associee a MC_k
        dans l'image quantifiee.
    }
}
```

Figure 3.16: *Amélioration de l'algorithme* `inverse_colormap`.

Les expériences résumées table 3.6 montrent que le second algorithme n'obtient pas les mêmes résultats que l'algorithme de recherche exhaustive. Toutefois la plus grande différence entre les erreurs obtenues par l'algorithme `inverse_colormap_ameliore` et l'algorithme de recherche exhaustive n'excède pas 0,5. Cette différence représente 0,14% de l'erreur produite par notre algorithme et est donc négligeable. L'utilisation de la 26 connexité des multi-ensembles induit de plus faibles erreurs que l'utilisation de la 18 connexité. Toutefois, une fois encore, les différences entre les erreurs produites par les deux algorithmes sont négligeables et l'utilisation de la 18 connexité induit des temps de calcul légèrement inférieurs à l'utilisation de la 26 connexité (voir Table 3.7). Nous avons donc décidé d'utiliser la 18 connexité des multi-ensembles. La taille moyenne des listes $L_i$ est affichée pour chaque image test sur la Table 3.8. Cette table montre que la taille moyenne des listes $L_i$ est égale à 1,65 si l'on utilise la 18 connexité des multi-ensembles et 16 couleurs représentatives. Notre algorithme d'inversion de table de couleurs devra donc effectuer approximativement 1,65 test par pixel pour créer l'image finale.

| $E$ | 1 | 2 | 3 |
|---|---|---|---|
| Lenna | 224.42 | 224.509 | 224.51 |
| Zelda | 347.95 | 348.42 | 348.43 |
| Girl | 250.81 | 250.91 | 250.94 |
| House | 125.27 | 125.31 | 125.36 |

| | | |
|---|---|---|
| (1) | : | Algorithme de recherche exhaustive |
| (2) | : | `inverse_colormap_ameliore` en utilisant la 26-connexité des multi-ensembles |
| (3) | : | `inverse_colormap_ameliore` en utilisant la 18-connexité des multi-ensembles |

Table 3.6: Erreur de la partition dans l'espace $RGB$ produite par notre algorithme avec trois algorithmes d'inversion de table de couleur.

| $s$ | 1 | 2 | 3 |
|---|---|---|---|
| Lenna | 8.68 | 3.92 | 3.71 |
| Zelda | 13.92 | 6.35 | 5.92 |
| Girl | 2.78 | 1.73 | 1.4 |
| House | 2.58 | 1.33 | 1.25 |
| Mean | 7.00 | 3.33 | 3.07 |

| | | |
|---|---|---|
| (1) | : | Algorithme de recherche exhaustive |
| (2) | : | `inverse_colormap_ameliore` en utilisant la 26-connexité des multi-ensembles |
| (3) | : | `inverse_colormap_ameliore` en utilisant la 18-connexité des multi-ensembles |

Table 3.7: Temps en secondes requis par notre algorithme avec trois différentes fonctions d'inversion de table de couleur.

| $\|L_i\|$ | 1 | 2 |
|---|---|---|
| Lenna | 2.3 | 1.98 |
| Zelda | 2.14 | 1.78 |
| Girl | 1.78 | 1.53 |
| House | 1.37 | 1.33 |
| Mean | 1.89 | 1.65 |

| | | |
|---|---|---|
| (1) | : | 26-connexité des multi-ensembles |
| (2) | : | 18-connexité des multi-ensembles |

Table 3.8: Taille moyenne des listes $L_i$ avec différentes relations d'adjacence. La taille des listes $L_i$ represente le nombre de tests nécessaires pour calculer l'image $\boldsymbol{Q}(c)$ d'une couleur $c$.

### 3.8.4 Expériences

Nous avons comparé nos algorithmes de quantification avec les algorithmes présentés par Wan et al. [WWP88] et Wu [WZ91]. Des mesures sur le temps, l'erreur quadratique et la qualité visuelle des images obtenues ont été effectuées sur les quatre images tests : Lenna, Zelda, Girl et House. Afin de faciliter le discours nous noterons `decoupe_fusion` notre algorithme de quantification combiné avec l'algorithme d'inversion de table de couleurs `inverse_colormap` (voir Figure 3.15). Notre algorithme combiné avec l'algorithme `inverse_colormap_ameliore` (voir Figure 3.16) sera noté `decoupe_fusion_ameliore`.

Les expériences résumées Table 3.9 montrent que le temps moyen requis par l'algorithme `decoupe_fusion` pour quantifier les quatres images tests est approximativement égal à 1,8 secondes. Ce temps est égal à 34,2 secondes pour l'algorithme de Wu [WZ91] et 14,5 secondes pour l'algorithme de Wan et al [WWP88]. Donc, l'algorithme `decoupe_fusion` est approximativement 20 fois plus rapide que l'algorithme de Wu et 8 fois plus rapide que l'algorithme de Wan et al. Le temps moyen requis par l'algorithme `decoupe_fusion_ameliore` est égal 3.1 secondes. L'algorithme `decoupe_fusion_ameliore` est donc 8 fois plus rapide que l'algorithme de Wu et 5 fois plus rapide que l'algorithme de Wan et al.

| s | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Lenna | 3.7 | 2.2 | 26.9 | 63.6 |
| Zelda | 5.9 | 3.3 | 19.0 | 42.8 |
| Girl | 1.4 | 0.9 | 6.3 | 16.7 |
| House | 1.2 | 0.8 | 5.7 | 13.8 |
| Mean | 3.1 | 1.8 | 14.5 | 34.2 |

| | | |
|---|---|---|
| (1) | : | Algorithme `decoupe_fusion_ameliore` avec $N = 2048$ |
| (2) | : | Algorithme `decoupe_fusion` avec $N = 2048$ |
| (3) | : | Algorithme de Wan et al. |
| (4) | : | Algorithme de Wu |

Table 3.9: Temps en secondes requis par nos algorithmes et ceux de Wan et Wu pour quantifier les quatres images en 16 couleurs. Le temps moyen pour les quatres images est affiché sur la dernière ligne.

Nous avons comparé l'erreur quadratique produite par nos algorithme à celles produites par les algorithme de Wu [WZ91] et Wan et al. [WWP88]. Ces mesures sont rassemblées dans la Table 3.10. L'erreur produite par l'algorithme `decoupe_fusion` se situe généralement entre l'erreur produite par l'algorithme de Wan et celle produite par l'algorithme de Wu. L'algorithme `decoupe_fusion` améliore l'erreur produite par Wan d'environ 2%. L'erreur produite par Wu est quand à elle inférieure à celle produite par notre algorithme d'environ 4%, ces pourcentages étant relatif à l'erreur produite par l'algorithme `decoupe_fusion`. Pour les quatres images tests, les plus faibles erreurs ont été obtenues avec l'algorithme `decoupe_fusion_ameliore`. L'écart entre la moyenne des erreurs produites par cet algorithme et celle produite par l'algorithme de Wu est égal à environ 8% de l'erreur produite par l'algorithme `decoupe_fusion_ameliore`. Cet écart monte à 15% lorsque l'on compare l'algorithme `decoupe_fusion_ameliore` avec l'algorithme de Wan et al.

Le ratio qualité/temps de l'algorithme `decoupe_fusion` est contrôlé par le nombre $N$ de multi-ensembles initiaux. Les expériences résumées dans les Tables 3.11 et 3.12 montrent que le temps requis par l'algorithme `decoupe_fusion` est une fonction croissante de $N$ tandis que l'erreur de la partition décroit pour des valeurs croissantes de $N$. La place nécessaire au stockage de la demit matrice codant le **GAM** est égale à $\frac{N(N-1)}{2}$. Donc le nombre $N$ de multi-ensembles initiaux permet de contrôler simultanément le ratio qualité/temps et la place mémoire occupée par l'algorithme.

Ce phénomène est un peu plus complexe avec l'algorithme `decoupe_fusion_ameliore` Les expériences résumées dans les Tables 3.13 et 3.14 montrent que l'algorithme `inverse_colormap_ameliore` (voir Figure 3.16) introduit de petites irrégularités dans la décroissance de l'erreur pour des valeurs croissantes de $N$ (voir Table 3.14). De plus, pour de petites valeurs de $N$ (typiquement 128 ou 256) le nombre de multi-ensembles totalement inclus dans un méta multi-ensemble est faible. Ce dernier point implique que pour de faibles valeurs de $N$ le nombre de méta multi-ensembles adjacents à un même multi-ensemble est élevé. Pour chaque multi-ensemble $\boldsymbol{C}_i$ nous stockons l'ensemble des index des méta multi-ensembles adjacents à $\boldsymbol{C}_i$ dans une liste $L_i$ (voir section 3.8.3), donc la taille des liste $L_i$ est une fonction décroissante de $N$ (voir table 3.15). Si nous notons $|L|$ la taille moyenne des listes $L_i$, l'algorithme `inverse_colormap_ameliore` (voir Figure 3.16) doit effectuer environ $|L|$ tests pour calculer la couleur représentative de chaque couleur de l'image. Le temps requis par l'algorithme `inverse_colormap_ameliore` est donc une fonction décroissante de $N$. On a donc deux phénomènes en compétition pour des valeurs de $N$ croissantes : la complexité croissante de l'algorithme de fusion et la complexité décroissante de l'algorithme d'inversion de table de couleurs. Ceci explique pourquoi les temps requis par l'algorithme `decoupe_fusion_ameliore` diminuent pour des valeurs de $N$ croissantes dans l'intervalle $[128, 1024]$. Après $N = 1024$ la taille des listes $L_i$ devient stable et le temps requis par l'algorithme devient une fonction croissante de $N$. Ces irrégularités interdisent un contrôle efficace du ratio qualité/temps. Nous avons donc décidé d'utiliser l'algorithme `decoupe_fusion_ameliore` avec une valeur fixe de $N$. Les expériences résumées dans les Tables 3.13 et 3.14 montrent que les valeurs $N = 1536$ et $N = 2048$ fournissent de bons ratios qualité/temps lorsque le nombre final de couleurs est inférieur à 16.

La qualité visuelle des images produites par nos algorithmes a été comparée aux images produites par les algorithmes de Wu et Wan et al. Nous avons quantifié l'image test Lenna avec 16 couleurs. Les résultats de ces expérimentations sont affichés sur la Planche 3.17. L'image 1 est l'image originale, l'image 2 a été obtenue grâce à l'algorithme de Wan et al. et l'image 3 grâce à l'algorithme de Wu. L'image 4 a été obtenue grâce à l'algorithme `decoupe_fusion` avec 2048 multi-ensembles initiaux tandis que l'image 5 a été obtenue avec l'algorithme `decoupe_fusion_ameliore` avec 2048 multi-ensembles initiaux. L'image obtenue grâce à l'algorithme de Wan (voir l'image 2) a plusieurs défauts : Les dégradés du bras, du front et de la joue ont été brisés en trois ou quatre plages de couleurs uniforme. De plus un défaut apparaît sur la poutre en haut à droite de l'image. La détérioration des dégradés a été atténuée sur les images 3, 4 et 5 respectivement obtenues avec l'algorithme de Wu, l'algorithme `decoupe_fusion` et l'algorithme `decoupe_fusion_ameliore`. Le dégradé de la joue a été légèrement mieux préservé par l'algorithme de Wu (image 3) et le défaut de la poutre a disparu sur les trois images. Finalement, la balance des couleurs semble être légèrement mieux préservée par notre algorithme (images 4 et 5) que par l'algorithme de Wu (image 3) (voir en particulier les plumes du chapeau de Lenna).

| $E$ | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| Lenna | 224 | 257 | 260 | 241 |
| Zelda | 348 | 396 | 384 | 367 |
| Girl | 250 | 278 | 293 | 285 |
| House | 125 | 144 | 156 | 132 |
| Mean | 237 | 268 | 273 | 256 |

| | | |
|---|---|---|
| (1) | : | Algorithme `decoupe_fusion_ameliore` avec $N = 2048$ |
| (2) | : | Algorithme `decoupe_fusion` avec $N = 2048$ |
| (3) | : | Algorithme de Wan et al. |
| (4) | : | Algorithme de Wu |

Table 3.10: Erreur calculée dans l'espace $RGB$ produite par nos algorithmes et ceux de Wan et Wu pour quantifier les quatre images en 16 couleurs. La moyenne des erreurs pour les quatre images est affichée sur la dernière ligne.

| | $N$ | | | | |
|---|---|---|---|---|---|
| $s$ | 128 | 256 | 512 | 1024 | 2048 |
| Lenna | 1.59 | 1.6 | 1.61 | 1.79 | 2.2 |
| Zelda | 2.65 | 2.64 | 2.66 | 2.85 | 3.3 |
| Girl | 0.42 | 0.43 | 0.45 | 0.56 | 0.9 |
| House | 0.42 | 0.43 | 0.76 | 0.51 | 0.8 |

Table 3.11: Temps en secondes requis par l'algorithme `decoupe_fusion` pour quantifier les images tests en utilisant $N$ multi-ensembles initiaux.

| | $N$ | | | | |
|---|---|---|---|---|---|
| $\boldsymbol{E}$ | 128 | 256 | 512 | 1024 | 2048 |
| Lenna | 468 | 301 | 285 | 261 | 257 |
| Zelda | 432 | 408 | 403 | 406 | 396 |
| Girl | 686 | 302 | 299 | 291 | 278 |
| House | 360 | 174 | 174 | 143 | 144 |

Table 3.12: Erreurs dans l'espace $RGB$ produites par l'algorithme `decoupe_fusion` utilisant $N$ multi-ensembles initiaux.

| $s$ | $N$ | | | | | |
|---|---|---|---|---|---|---|
| | 128 | 256 | 512 | 1024 | 1536 | 2048 |
| Lenna | 4.79 | 4.2 | 3.85 | 3.64 | 3.67 | 3.75 |
| Zelda | 7.92 | 6.86 | 6.04 | 5.72 | 5.77 | 5.93 |
| Girl | 1.2 | 1.08 | 1.16 | 1.05 | 1.19 | 1.38 |
| House | 1.16 | 1.06 | 1.21 | 1.02 | 1.14 | 1.14 |

Table 3.13: Temps en secondes requis par l'algorithme `decoupe_fusion_ameliore` pour quantifier les images tests en utilisant $N$ multi-ensembles initiaux.

### 3.8.5 Conclusion

Nous avons présenté dans cette section deux algorithmes de quantification basés sur une approche découpe-fusion. Le premier de ces algorithmes : `decoupe_fusion` produit des images dont l'erreur est généralement située entre les erreurs produites par l'algorithme de Wan [WWP88] et l'algorithme de Wu [WZ91]. Le temps requis par cet algorithme pour effectuer la quantification d'une image est généralement égal au vingtième du temps requis par l'algorithme de Wu et au huitième du temps requis par l'algorithme de Wan. De plus, le temps requis par cet algorithme peut être réduit en réduisant le nombre $N$ de multi-ensembles initiaux utilisés par l'algorithme de fusion.

Le second algorithme `decoupe_fusion_ameliore` fournit de plus faibles erreurs que les algorithmes de Wan [WWP88] et Wu [WZ91]. Nos expériences ont montré qu'avec 16 couleurs finales, l'erreur produite par notre algorithme est 8% plus faible que l'erreur obtenue par Wu et 15% plus faible que celle obtenue par Wan. Cet algorithme effectue la quantification d'une image environ 8 fois plus rapidement que l'algorithme de Wu et 5 fois plus rapidement que l'algorithme de Wan et al.

L'efficacité de ces algorithmes peut être expliquée par leurs décomposition en une étape de découpe suivie par une étape de fusion. L'étape de découpe est réalisée grâce au processus de découpe uniforme qui est extrêmement simple et efficace. Cet algorithme réduit le nombre de données à traitrer. La perte d'information induite par ce prétraitement est controlée par le nombre $N$ de multi-ensembles initiaux générés par l'algorithme de découpe uniforme. Cette réduction du nombre de données permet d'appliquer une heuristique de fusion plus complexe, et donc plus précise que les heuristiques habituellement utilisées par les algorithmes s'appliquant à l'ensemble des pixels de l'image.

| $E$ | $N$ | | | | | |
|---|---|---|---|---|---|---|
| | 128 | 256 | 512 | 1024 | 1536 | 2048 |
| Lenna | 324 | 252 | 245 | 221 | 219 | 224 |
| Zelda | 386 | 357 | 357 | 357 | 346 | 348 |
| Girl | 374 | 256 | 257 | 257 | 256 | 250 |
| House | 276 | 135 | 133 | 122 | 126 | 125 |

Table 3.14: Erreurs dans l'espace $RGB$ produites par l'algorithme `decoupe_fusion_ameliore` utilisant $N$ multi-ensembles initiaux.


Figure 3.17: *Différentes quantifications de Lenna en 16 couleurs*

| $\lvert L \rvert$ | $N$ | | | | | |
|---|---|---|---|---|---|---|
| | 128 | 256 | 512 | 1024 | 1536 | 2048 |
| Lenna | 5.4 | 4.0 | 3.1 | 2.4 | 2.1 | 1.9 |
| Zelda | 4.9 | 3.7 | 2.9 | 2.2 | 1.8 | 1.7 |
| Girl | 5.4 | 3.3 | 2.7 | 1.9 | 1.6 | 1.5 |
| House | 5.2 | 3.6 | 3.0 | 2.3 | 1.9 | 1.7 |

Table 3.15: Taille moyenne des listes $L_i$ générées par l'algorithme `inverse_colormap_ameliore` avec $N$ multi-ensembles initiaux. La taille de la table de couleur est fixée à 16.

# Partie II

# La segmentation

# Chapitre 4

# Notations et définitions de base

Dans l'introduction de ce travail nous avons défini une image comme un tableau rectangulaire composé de carrés de cotés 1 appellés pixels. Par convention nous dirons que le centre du pixel d'indice $(i, j)$ dans ce tableau à pour coordonnées $(i, j)$ dans un repère dont l'origine se situe au centre du premier pixel du tableau et dont les axes sont parallèles aux cotés des carrés (voir Figure 4.1).

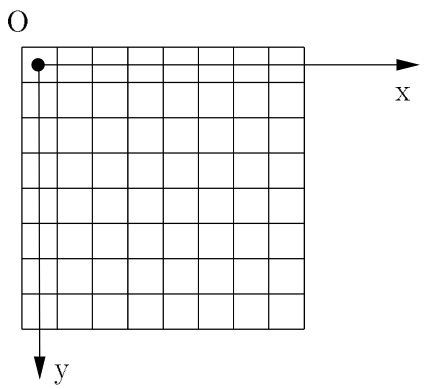



Figure 4.1: *Axes de coordonnées associés à une image.*

Nous pouvons donc associer à chaque indice $(i, j)$, un carré de côté 1 centré en $(i, j)$ et associer à chaque pixel le couple $(i, j)$ correspondant aux coordonnées de son centre. Une image peut donc se définir comme un ensemble de pixels ou un ensemble de couples d'entiers. Si l'on utilise cette dernière définition l'image est considérée comme un sous ensemble fini du plan $P_0$ défini par :

$$P_0 = \{(i, j) \in \mathbb{Z} \times \mathbb{Z}\}$$

où $\mathbb{Z}$ désigne l'ensemble des entiers relatifs.

Ces deux définitions de l'image étant duales on confond généralement les deux et un pixel sera souvent confondu avec son centre. On parlera ainsi du pixel $(i, j)$, du point $(i, j)$ ou du pixel de centre $(i, j)$.

## 4.1 La connexité

**Définition 8** *Deux pixels $p$ et $p'$ sont dits* **4-voisins** *si et seulement si:*

$$|x_p - x_{p'}| + |y_p - y_{p'}| = 1$$

**Définition 9** *Deux pixels $p$ et $p'$ sont dits* **8-voisins** *si et seulement si:*

$$\max(|x_p - x_{p'}|, |y_p - y_{p'}|) = 1$$

**Définition 10** *Soient deux points $p$, $q$ d'une image $I$. Un* **chemin** *de $p$ à $q$ est une séquence de points $p = p_0, p_1, \ldots, p_n = q$ telle que $p_i$ est un voisin de $p_{i+1}$ pour tout $i$ dans $\{1, \ldots, n\}$.*

Cette définition fournit deux types de chemins selon la relation de voisinage envisagée. On parlera ainsi de **chemins 4-connexe** (respectivement **8-connexe**) lorsque tout point est un 4-voisin (resp. 8-voisin) du point suivant.

**Définition 11** *Un chemin $C$ sera dit* **simple** *si et seulement si tout point $p$ de $C$ possède au plus deux voisins appartenant à $C$.*

**Définition 12** *Un chemin $C = p_0, p_1, \ldots, p_n$ sera dit* **fermé** *si ses deux extrémités sont égales au même point, donc si $p_0 = p_n$.*

**Définition 13** *Soit $S$ un ensemble de pixels d'une image $I$. On dira que deux pixels $p$ et $q$ sont* **connectés** *dans $S$ s'il existe un chemin de $p$ à $q$ dont tous les points appartiennent à $S$.*

On peut noter que nous avons à nouveau deux définitions en une seule. Rosenfeld [Ros79] a montré que la relation précédente est une relation d'équivalence. Cette dernière propriété permet de définir les **régions** :

**Définition 14** *Soit $S$ un ensemble de pixels d'une image $I$. L'ensemble $S$ défini une* **région** *si et seulement si tous les points de $S$ sont connectés entre eux.*

On parlera ainsi de **région 8-connexe** ou de **région 4-connexe** selon le voisinage utilisé.

**Définition 15** *Soir $S$ un ensemble de pixels d'une image $I$, les classes d'équivalences définies par la relation "est connecté dans $S$" sont appelées les* **composantes connexes de S**.

Donc, une composante connexe est forcément une région. Inversement, étant donné un ensemble $S$ de référence, une région incluse dans $S$ n'est pas forcément une composante connexe de celui-ci.

**Définition 16** *Soit $I$ une image, $(C, f)$ le multi-ensemble associé à $I$ et $P = \{(C_1, f), \ldots, (C_K, f)\}$ une partition de $(C, f)$.* **La partition de l'image** *associée à $P$ est définie comme l'ensemble $\{S_1, \ldots, S_K\}$ avec :*

$$\forall i \in \{1, \ldots, K\} \quad S_i = \{p \in I \ / \ I(p) \in C_i\}$$

*où $I(p)$ représente la valeur du pixel $p$.*

Si nous effectuons une quantification de l'image $I$ en $K$ couleurs, $S_i$ représentera donc l'ensemble des pixels de couleur représentative $\mu(C_i)$.

**Définition 17** *Soient $I$ une image, $(C, f)$ son multi-ensemble associé et $\{S_1, \ldots, S_K\}$ la partition de l'image associée à une partition de $(C, f)$.* **La partition de l'image en régions 4-connexes** *(resp. 8-connexes) associée à la partition $\{S_1, \ldots, S_K\}$ est égale a l'ensemble des composantes connexes des ensembles $S_i$.*

La notion d'adjacence est induite par la notion de voisinage utilisée. Si nous utilisons des voisinages 4-connexes, les régions de la partition seront définies comme l'ensemble des groupes de pixels 4-connexes dont la valeur appartient au même multi-ensemble.

## 4.2 Frontières de régions

La définition des régions induit la notion de **frontières** de régions. Lorsqu'on cherche à définir de façon formelle cette notion, on se heurte rapidement à plusieurs problèmes. On définit classiquement la frontière d'une région 8-connexe $\mathbf{R}$ comme l'ensemble des points de $\mathbf{R}$ dont au moins l'un des 4-voisins n'est pas élément de $\mathbf{R}$. Dans ce cas, la frontière de la région est composée d'un ou plusieurs chemins 8-connexes. De manière duale, on définit la frontière d'une région 4-connexe $\mathbf{R}$ comme l'ensemble des points de $\mathbf{R}$ dont au moins l'un des 8-voisins n'est pas élément de $\mathbf{R}$. Dans ce cas, la frontière de la région est composée d'un ou plusieurs chemins 4-connexes. Ces deux définitions posent le problème bien connu de la transposition du théorème de Jordan dans le plan discret [CM91]. De plus, aucune de ces définitions ne vérifie le fait que deux régions adjacentes partagent une portion de frontière.

Pour résoudre ces difficultés, Brice et Fennema [BF70] ont introduit la notion de **chemin inter-pixels**. Ces chemins sont définis dans un plan appelé **le plan demi-entier** dont les propriétés ont été étudiées par J.P. Domenger [Dom92].

**Définition 18** *Le plan demi-entier $P_{\frac{1}{2}}$ est l'ensemble des points du plan réel dont les coordonnées sont de la forme :*

$$x = k + \frac{1}{2}; y = k' + \frac{1}{2} \text{ avec } (k, k') \in \mathbb{Z}^2$$

On a donc :

$$P_{\frac{1}{2}} = \left\{(i+\tfrac{1}{2}, j+\tfrac{1}{2}),\ \text{avec } (i,j) \in \mathbb{Z}^2\right\}$$

Les relations de voisinage entre les points demi-entiers sont les mêmes que celle définies pour les points entiers. On parlera donc de **chemin demi-entier 4-connexe** ou **8-connexe**. On définit en plus une relation de demi-voisinage entre les points entiers et demi-entiers :

**Définition 19** *Soit $p$ un point entier et $p'$ un point demi-entier. Les points $p$ et $p'$ sont* **demi-voisins** *si :*

$$|x_p - x'_p| = |y_p - y'_p| = \frac{1}{2}$$

Cette définition nous permet de définir les frontières d'une région $R$ de l'image.

**Définition 20** *Un point demi-entier est appelé* **point frontière** *si au moins deux de ses demi-voisins appartiennent à des régions différentes.*

**Définition 21** *Soit $R$ une région de l'image $I$. La* **frontière** *de $R$ est l'ensemble des points frontières dont au moins un demi-voisin appartient à $R$ (voir Figure 4.2).*

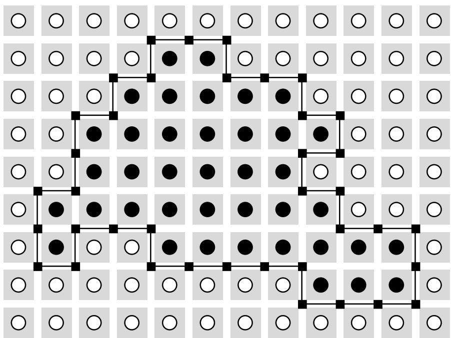


Figure 4.2: *Exemple de frontière dans le plan $P_{\frac{1}{2}}$.*

Les points demi-entiers peuvent se voir comme les coins d'un pixel (voir Figure 4.2). Deux points demi-entiers 4-voisins sont séparés par un coté de pixel. Domenger [Dom92] a montré que tout chemin simple fermé du plan $P_{\frac{1}{2}}$ sépare le plan entier $P_0$ en deux composantes discrètes. Cette propriété est l'analogue discret du théorème de Jordan. Elle permet de définir indifféremment une région à partir de l'ensemble de ses pixels ou à partir de sa frontière. Ce dernier point est fondamental pour le modèle décrit dans le chapitre 6.

# Chapitre 5

# Les algorithmes de segmentation

## 5.1 Introduction

L'image a longtemps été considérée comme une représentation planaire d'objets, de personnages ou de scènes tridimensionnels. L'apparition de l'informatique et de nouveaux capteurs (images infrarouges, ultraviolets, IRM, rayon X,...) a transformé cette notion en la notion plus vaste de collection d'informations. L'informatique, par sa capacité à traiter un grand nombre de données fut rapidement utilisée pour analyser, traiter et enfin interpréter l'ensemble des informations contenues dans une image. Le but de ces traitements étant de construire à partir des données constituant l'image une représentation symbolique des objets qui la constituent. L'ensemble de ces traitements est souvent regroupé sous le terme d'**analyse d'image**.

La reconnaissance de routes sur des images satellites constitue un exemple caractéristique d'analyse d'image. Un programme réalisant ce genre d'opérations devra, par exemple, réaliser les opérations suivantes:

- Extraire de l'image un ensemble d'objets susceptibles d'être des routes.
- Éliminer tous les objets qui ne sont pas des routes.
- Extraire des paramètres des objets restant : localisations, carrefours, longueurs, ...
- Identifier chaque route (Nationale 3, Départementale 22, ...)

La première étape de ce type de processus est appelée l'étape de **segmentation**. Cette étape est une étape fondamentale de tout processus d'analyse d'image puisqu'elle permet de constituer des objets à partir des données brutes de l'image. La segmentation est donc le premier niveau d'abstraction permettant à l'ordinateur de manipuler des relations entre des objets plutôt que les donnés brutes de l'image. Une définition formelle d'un algorithme de segmentation a été donné par Horowitz et Pavlidis [HP76, HP75].

**Définition 22** *Soit $X$ le domaine de l'image et $f$ la fonction qui associe à chaque pixel une valeur $f(x,y)$. Si nous définissons un prédicat $P$ sur l'ensemble des parties de $X$. La*

**segmentation** *de $X$ est définie comme une partition de $X$ en $n$ sous-ensembles $\{S_1, ..., S_n\}$ tels que :*

1. $X = \bigsqcup_{i=1}^{m} S_i$
2. $\forall i \in \{1, \ldots, n\}$ $S_i$ *est connexe.*
3. $\forall i \in \{1, \ldots, n\} P(S_i) = vrai$
4. $\forall i, j \in \{1, \ldots, n\}^2$ / $S_i$ *est adjacent à* $S_j$ *et* $i \neq j \Rightarrow P(S_i \cup S_j) = faux$

Le prédicat $P$ est utilisé pour tester l'homogénéité des ensembles $S_i$. Ces sous-ensembles $S_i$ constituent les régions de l'image $X$. Une segmentation de l'image est donc sa décomposition en un ensemble de régions homogènes, le critère d'homogénéité $P$ restant à déterminer. Zucker [Zuc76] a résumé les conditions de la définition 22 comme suit : la première condition implique que tout pixel de l'image appartienne à une région. Cela signifie que l'algorithme de segmentation ne doit pas se terminer avant d'avoir traité tous les points. La seconde condition implique que toute région doit être connexe, donc les pixels d'une région doivent être contigus. La contiguïté des pixels étant induite par le voisinage défini sur l'image. La troisième condition implique que chaque région doit être homogène. Enfin, la quatrième condition est une condition de maximalité indiquant que la fusion de deux régions ne doit pas être homogène. Il est important de remarquer que le nombre $n$ de régions formant la partition de l'image reste indéterminé. Il peut donc exister plusieurs segmentations possibles pour un prédicat $P$ donné.

La définition 22 étant un peu trop restrictive, beaucoup d'algorithmes de segmentation ne retiennent que l'idée sous-jacente à la définition, c'est à dire la décomposition en régions homogènes.

Extraire des régions homogènes d'une image peut s'effectuer par deux approches duales.

- Approche **Régions** : les méthodes utilisant ce type d'approche essaient de fixer un critère de similarité entre les pixels de façon à pouvoir les rassembler en régions homogènes. Les variations entre les différentes méthodes utilisant cette approche se situent essentiellement dans la définition du prédicat $P$ à partir du critère de similarité et dans l'utilisation de $P$ pour construire les régions finales.
- Approche **Frontière** : plutôt que de chercher un critère de similarité entre pixels, ces méthodes cherchent plutôt la frontière des régions, c'est à dire l'ensemble des pixels adjacents et dissemblables, d'où le nom usuel de **Détection de contours**. Ces méthodes ne fournissent généralement pas de contours fermés ce qui nécessite une étape supplémentaire dite de **fermeture** de façon à obtenir des frontières fermées définissant les régions.

Suivant la méthode employée, il est souvent plus intéressant de considérer l'image dans un formalisme particulier, comme par exemple :

- Un processus stochastique.

- Un vecteur aléatoire $(I[0,0], I[0,1], \ldots, I[n,m])$ si la taille de l'image $I$ est $n \star m$.
- Une surface 3D (dans le cas d'un signal mono-dimensionnel).
- Un ensemble de données liées par des contraintes géométriques.
- La discrétisation d'un signal continu.

Nous allons à présent étudier les principales familles de méthodes utilisées en segmentation. Nous étudieront par la suite les structures de données utilisées par ces algorithmes ainsi que l'incidence du choix d'une structure de données sur une méthode de segmentation.

## 5.2 Les principales méthodes de segmentation

### 5.2.1 Les méthodes de détection de contours

Ces méthodes tentent de trouver les frontières entre les régions. Le postulat commun à toutes les méthodes de cette famille est que deux pixels adjacents et de valeurs très différentes n'appartiennent pas à la même région. Les différences entre les diverses méthodes se situent sur la quantification du “différent”, du “très”, et bien évidemment sur la manière de trouver deux pixels très différents.

#### 5.2.1.1 Les opérateurs différentiels

Ces méthodes considèrent généralement l'image comme la discrétisation d'un signal continu 2D. On a donc :

$$I = Q \circ f$$

où $I$ est l'image discrète, $f$ le signal continu, et $Q$ un opérateur d'échantillonage. La recherche de contours dans $I$ (donc de sauts de valeurs) se caractérise dans $f$ par des maxima de la dérivé. On introduit donc l'opérateur différentiel $D$. Nous verrons dans la section 5.2.6 comment l'information couleur peut être traitée par ce type d'approche. La plupart des méthodes de détection de contours étant plus spécifiquement dédiées au traitement d'images mono-dimensionnelles, nous allons dans cette section nous limiter à ce cas. Nous supposerons de plus que $f$ appartient à $C^2(\mathbb{R}^2, \mathbb{R})$, l'ensemble des fonctions continuement deux fois différentiables de $\mathbb{R}^2$dans $\mathbb{R}$. L'ensemble des fonctions susceptibles de donner une même discrétisation $I$ étant important cette dernière condition n'est pas très restrictive.

La différentielle de $f$ peut donc s'exprimer de la façon suivante :

$$Df(p).n = \frac{\partial f}{\partial x}(p).n_x + \frac{\partial f}{\partial y}(p).n_y = \lim_{h \to 0} \frac{f(p) - f(p + (hn_x, hn_y))}{h} \tag{5.1}$$

$Df(p).n$ est donc la dérivée de $f$ dans la direction $n$ (comme seule la direction de n nous intéresse, on peut prendre $\|n\| = 1$). Si nous introduisons l'opérateur gradient défini par :

$$\nabla f = \begin{pmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{pmatrix}$$

L'équation 5.1 s'écrit :

$$Df(p).n = \bigtriangledown f(p) \bullet n \tag{5.2}$$

Où $\bullet$ désigne le produit scalaire.

On a, de plus, par l'équation 5.2 :

$$\max_{\|n\|=1} |Df(p).n| = \max_{\|n\|=1} |\bigtriangledown f(p) \bullet n| = \|\bigtriangledown f(p)\|$$

Donc, en tout point la norme du gradient permet de connaître la variation maximum de la différentielle. De plus le maximum étant atteint pour $n$ colinéaire à $\bigtriangledown f(p)$ la direction du gradient donne la direction de plus grande variation de la fonction $f$.

Les directions de variation de $f$ correspondant à des maxima de la différentielle de $f$, on peut également chercher ces directions par les zéros de la différentielle seconde de $f$. Si $D_2 f$ désigne cette différentielle seconde, on a :

$$D_2 f(p).n = \frac{\partial^2 f}{\partial x^2}(p).n_x.n_x + \frac{\partial^2 f}{\partial y^2}(p).n_y.n_y + 2\frac{\partial^2 f}{\partial x \partial y}(p).n_x n_y \tag{5.3}$$

La fonction $f$ présentera une variation locale suivant une direction $n$, si à $n$ fixé la fonction $Df(p).n$ présente un maximum local. Ce maximum local se traduira, toujours à $n$ fixé, par un zéro de la fonction $D_2 f(p).n$ en ce point.

Afin d'alléger les calculs on cherche les zéros du laplacien plutôt que les zéros de la différentielle seconde. Le laplacien de la fonction $f$, noté $\triangle f$ est défini par :

$$\triangle f(p) = \frac{\partial^2 f}{\partial x^2}(p) + \frac{\partial^2 f}{\partial y^2}(p)$$

Le laplacien étant invariant par rotation, le calcul des zéros du laplacien ne fait pas intervenir la direction $n$ de plus grande variation de $Df(p)$. L'utilisation du laplacien évite donc de calculer le gradient en plus de la différentielle seconde.

Les zéros du laplacien ne correspondent généralement pas aux zéros de $D_2 f(p).n$, toutefois Marr [MH80] a montré qu'il y avait coïncidence entre les zéros des deux fonctions au point $p$ si les variations d'intensité étaient linéaires sur la ligne de passage par zéro et sur les lignes parallèles dans un voisinage de $p$. Le laplacien peut donc être vu comme une bonne approximation de la différentielle seconde.

#### 5.2.1.2 Les opérateurs de convolution

Dans les cas usuels, la fonction $f$ comporte souvent du bruit. Ce bruit peut parasiter la détection des contours de $f$ en créant des sauts de valeurs artificiels. Un moyen simple de diminuer le bruit est de convoluer la fonction $f$ avec un filtre passe-bas. La convolution de deux fonctions $f$ et $g$ de $\mathbb{R}^2$dans $\mathbb{R}^2$définit la fonction $f * g$ :

$$f * g(x,y) = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} f(u,v)g(x-u,y-v)dt = \int_{-\infty}^{+\infty}\int_{-\infty}^{+\infty} f(x-u,x-v)g(u,v)dt$$

Le filtre passe-bas le plus connu est sans doute la gaussienne notée $G$ et définie par :

$$G(x) = \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}}$$

Le paramètre $\sigma$ contrôle l'aplatissement de la gaussienne et permet donc de contrôler la puissance du filtre. Une forte valeur de $\sigma$ induit un fort lissage et vice-versa. La convolution d'un signal avec la gaussienne s'effectue en limitant celle-ci à un support fini $[-M_\epsilon, M_\epsilon]$ avec :

$$\forall x \in [-M_\epsilon, M_\epsilon] \quad G(x) < \epsilon$$

On peut remarquer que la valeur $M_\epsilon$ est une fonction croissante de $\sigma$, donc plus l'on voudra lisser le signal plus l'on devra agrandir le support $[-M_\epsilon, M_\epsilon]$ de la gaussienne.

Une des principales raisons de la popularité de la gaussienne est sa **séparabilité**. Si $G(x, y)$ représente une gaussiennne 2D on a :

$$\begin{aligned} G(x,y) &= \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2+y^2}{2\sigma^2}} \\ &= \frac{1}{\sigma\sqrt{2\pi}} e^{-\frac{x^2}{2\sigma^2}} e^{-\frac{y^2}{2\sigma^2}} \\ G(x,y) &= \frac{1}{\sigma\sqrt{2\pi}} g(x) g(y) \end{aligned}$$

Donc si nous convoluons une fonction 2D $f$ avec $G$, on a :

$$\begin{aligned} f * G(x,y) &= \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(x-u, y-v) G(u,v) du dv \\ &= \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{+\infty} \int_{-\infty}^{+\infty} f(x-u, y-v) g(u) g(v) du dv \\ &= \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{+\infty} g(u) \left( \int_{-\infty}^{+\infty} f(x-u, y-v) g(v) dv \right) du \\ &= \frac{1}{\sigma\sqrt{2\pi}} \int_{-\infty}^{+\infty} g(u) f *_y g(x-u, y) du \\ f * G(x,y) &= \frac{1}{\sigma\sqrt{2\pi}} g *_x f *_y g(x,y) \end{aligned}$$

où $*_x$ et $*_y$ représentent les convolutions par rapport aux variables $x$ et $y$.

La convolution de $f$ avec la gaussienne 2D $G$ peut donc être réalisée grâce à deux convolutions de signaux 1D. Ceci permet d'utiliser deux convolutions avec des masques 1D $[-M_\epsilon, M_\epsilon]$ plutôt qu'avec un masque 2D de taille $[-M_\epsilon, M_\epsilon]^2$. Convoluer une image comportant $m$ pixels avec une gaussienne dont la taille du masque est égal à $n$ réclame donc $\mathcal{O}(2mn)$ opérations, l'utilisation d'un masque 2D réclamant quant à elle $\mathcal{O}(mn^2)$ opérations.

Une des propriétés essentielles du produit de convolution, en segmentation, est la possibilité de placer alternativement la dérivée sur un membre ou sur l'autre. Plus précisément on a :

**Propriété 1** *Soient $f$ et $g$ deux fonctions de $\mathbb{R}$ dans $\mathbb{R}$, si $f$ ou $g$ est $k$ fois différentiable alors $f*g$ l'est également. De plus si $f$ et $g$ sont toutes deux au moins $k$ fois différentiables on a :*

$$(f*g)^{(k)} = (f^{(k)}) * g = f * (g^{(k)})$$

Donc si $G*f$ représente l'image filtrée par une gaussienne, le gradient de l'image filtrée peut tout simplement s'effectuer en calculant la convolution de $f$ avec le gradient de G,

$\bigtriangledown G$ qui peut être précalculé une fois pour toutes. De même le filtrage d'une image par une gaussienne $G$, puis le calcul de son laplacien peuvent s'effectuer grâce à une seule convolution $\triangle G * f$.

#### 5.2.1.3 Passage au discret

La transposition de ces méthodes au modèle discret se fait par l'approximation des différentielles partielles de la fonction $f$. Si l'on utilise les différences finies l'on obtient par exemple :

$$\begin{aligned} \tfrac{\partial f}{\partial x}(i,j) &\approx \frac{\Delta I}{\Delta i}(i,j) = I(i+1,j) - I(i,j) \\ \tfrac{\partial^2 f}{\partial x^2}(i,j) &\approx 2I(i,j) - I(i+1,j) - I(i-1,j) \end{aligned}$$

Des méthodes plus évoluées [Pre70, Kir71] approximent le gradient par une convolution avec un masque $3\star 3$. Dans le cas discret et pour un masque $M$ à support fini, la convolution de $M$ avec $I$ est définie par :

$$M * I(i,j) = \sum_{k=-p}^{p} \sum_{l=-q}^{q} M[k][l] I[i-k][j-l]$$

Marr et al. [MH80] et Huertas et al. [HM86] approximent les zéros de la différentielle seconde en convoluant l'image avec le laplacien de la gaussienne. Une approche légèrement différente introduite par Cocquerez [CD85] consiste à utiliser un filtre passe-bas non linéaire [NM79] suivi d'une dérivation.

#### 5.2.1.4 Les opérateurs optimaux

Canny [Can86] modélise un contour $C$ comme la superposition d'un saut d'amplitude $S$ et d'un bruit blanc $B$.

$$C(x) = S_0 S(x) + B(x)$$

La fonction $S$ est modélisée par la fonction $\chi_{[0,+\infty)}$ où $\chi$ est la fonction caractéristique.

Suivant ce formalisme, le détecteur de contour idéal $h$ est celui qui, convolué à $C$ présente un maximum en 0. Cette contrainte n'étant pas suffisante pour déterminer $h$, Canny impose trois conditions supplémentaires à son détecteur de contour :

- Une bonne détection
- Une bonne localisation
- Une faible multiplicité des maxima dûs au bruit.

La formalisation de ces trois conditions impose à la fonction $h$ de respecter l'équation différentielle suivante :

$$2h(x) - 2\lambda_1 h''(x) + 2\lambda_2 h^{(3)}(x) + \lambda_3 = 0$$

Les conditions aux limites imposées à la fonction $h$ permettent de fixer les paramètres $\lambda_1, \lambda_2, \lambda_3$. Les conditions imposées par Canny permettent d'obtenir une fonction à support fini $[-M, M]$ :

$$h(0) = 0 \;\; ; \;\; h(M) = 0 \;\; ; \;\; h'(0) = S \;\; ; \;\; h'(M) = 0$$

Malheureusement, ces conditions donnent une fonction $h$ très coûteuse à implémenter. Partant de ce constat Deriche [Der87] a défini d'autres conditions aux limites donnant à la fonction $h$ un support infini :

$$h(0) = 0 \;\; ; \;\; h(+\infty) = 0 \;\; ; \;\; h'(0) = S \;\; ; \;\; h'(+\infty) = 0$$

Ces conditions initiales donnent la fonction :

$$h \left( \begin{array}{ccc} \mathbb{R} & \rightarrow & \mathbb{R} \\ x & \mapsto & ce^{-\alpha|x|} sin\omega x \end{array} \right.$$

L'analyse des critères donnés par Canny [Can86] montre que les meilleures performances du filtre sont obtenues pour $\omega$ tendant vers 0. On obtient à ce moment là $h(x) = Cxe^{-\alpha|x|}$ avec $C = \omega c$. Le paramètre $C$ est un paramètre de normalisation, le paramètre $\alpha$ contrôle quant à lui la sensibilité de l'opérateur de détection de contour. Une augmentation de $\alpha$ favorise la détection au détriment de la localisation et vice-versa. Le paramètre $\alpha$ joue ici, exactement le même rôle que le paramètre $\sigma$ dans la définition du gradient de la gaussienne.

Deriche construit à partir de $h$ une fonction de lissage 1D $l$ définie par :

$$l(x) = \int_0^x h(x)dx = b(\alpha|x| + 1)e^{-\alpha|x|}$$

On définit alors la fonction de lissage 2D $L(x, y)$ par :

$$L(x, y) = l(x)l(y)$$

La fonction $L$ joue ici le rôle de la gaussienne, et le filtrage de l'image s'obtient en convoluant celle-ci avec $L$. Une fois l'image lissée, on peut calculer son gradient ou son laplacien. Ces deux opérations s'effectuent en convoluant l'image avec le gradient ou le laplacien de $L$.

L'avantage des opérateurs de Deriche par rapport à ceux de Canny vient du fait que les techniques de la transformée en $z$ [Der87], appliquées à la fonction $L$ et à ses dérivées permettent de calculer de façon récursive la convolution des opérateurs de Deriche avec l'image. Si, par exemple, nous effectuons la convolution d'un signal mono-dimensionnel $x(m)$ avec la fonction échantillonnée $l(m)$, le signal final $y(m)$ se déduit des équations suivantes [DC87] :

$$\begin{array}{lcll} y^+(m) & = & a_0x(m) + a_1x(m-1) - b_1y^+(m-1) - b_2y^+(m-2) & m = 1, \ldots, M \\ y^-(m) & = & a_2x(m+1) + a_3x(m+2) - b_1y^-(m+1) - b_2y^-(m+2) & m = M, \ldots, 1 \\ y(m) & = & y^+(m) + y^-(m) & m = 1, \ldots, M \end{array}$$

où $M$ représente la taille du signal et où les coefficients $a_i$ et $b_i$ se déduisent du paramètre $\alpha$.

L'application de l'algorithme décrit par les équations précédentes nécessite 8 opérations par point quel que soit la valeur de $\alpha$. Les filtres de Deriche présentent donc les avantages suivants :

- Ils s'appuient sur une étude théorique permettant de juger et comparer différents détecteurs de contours.
- Leur expression, plus simple que la gaussienne, permet, grâce à la transformée en $z$, d'obtenir une définition récursive de la convolution. Cette récursivité permet d'effectuer l'opération de convolution en un nombre fixe d'opérations par point de l'image, indépendamment du paramètre $\alpha$.

Shen et Castan [SC92, CZS89] ont adopté une démarche similaire à celle de Canny [Can86]. Les différences entre les deux méthodes portent sur la formalisation des critères décrivant un détecteur de contour optimal. La fonction obtenue par Shen est égale à $\beta e^{-\beta|x|}$. Cette fonction permet de définir des opérateurs de lissage, de gradient et de laplacien et d'effectuer une transformée en $z$ [Mon90] permettant une implémentation récursive.

#### 5.2.1.5 Mise en oeuvre

Les recherches des points de contours à partir du gradient, des zéros de la dérivée seconde ou des zéros du laplacien sont très similaires. La démarche générale de ces trois méthodes peut se décomposer comme suit [CP95] :

*La recherche à partir du gradient*

- Calculer le gradient en chaque point de l'image
- Créer l'image de la norme du gradient.
- Extraire les maxima locaux dans la direction du gradient.
- Effectuer un seuillage à effet hystérésis (voir ci-dessous) de l'image des maxima locaux.

*La recherche à partir de la différentielle seconde*

- Calculer $D_2 f(p).n$ pour tout point $p$ de l'image ($n$ représente la direction du gradient, donc la direction de plus grande variation de la différentielle première).
- Rechercher les passages par zéros de $D_2 f(p).n$ dans la direction $n$.
- Créer l'image des passages par zéros et de la norme du gradient.
- Effectuer un seuillage à effet hystérésis de l'image des maxima locaux.

*La recherche à partir du laplacien*

- Calculer le laplacien.
- Rechercher les passages par zéros du laplacien.

- Créer l'image des passages par zéros et de la norme du gradient.

- Effectuer un seuillage à effet hystérésis de l'image des maxima locaux.

Le seuillage à effet hystérésis consiste à ne conserver que :

- Les points dont la norme du gradient est supérieure à un seuil haut ($sh$).

- Les points dont la norme du gradient est supérieure à un seuil bas ($sb$ avec $sb < sh$) et appartenant à un bout de contour dont au moins un point possède une norme du gradient supérieure à $sh$.

Ce type de seuillage permet de diminuer le nombre de bouts de contour non fermés.

#### 5.2.1.6 Conclusion

Les méthodes de détection de contours permettent d'obtenir très rapidement (de l'ordre de la seconde) un ensemble de contours qui serviront de base à des algorithmes de fermeture de contours. En raison de leur caractère local, ces méthodes doivent diminuer le bruit en appliquant un filtre passe-bas comme la gaussienne ou les filtres de Deriche [Der87] et Shen [SC92]. L'utilisation de ces filtres pose le problème du réglage des paramètres. Si l'on filtre trop, on risque de "rater" des bouts de contours importants alors que si on ne filtre pas assez, l'on risque de se retrouver avec une quantité de contours non significatifs. De plus un même contour peut avoir deux localisations légèrement différentes pour des filtrages voisins. Un début de solution à ce dernier problème a été apporté par Witkin [Wit84] dans le cas de signaux mono-dimensionnels mais à notre connaissance, il n'a pas pu être étendu aux images 2D. Williams [WS90] a toutefois utilisé une partie des travaux de Witkin pour calculer les déplacements d'un contour de type "escalier" en fonction du filtrage. Cette étude lui fournit des informations supplémentaires utilisées lors de l'étape de fermeture de contours.

De plus le caractère local de ces méthodes rend difficile l'insertion du concept de régions. Il est par exemple délicat d'insérer dans des algorithmes de détection de contours des règles telles que :

- Cette région homogène contient des détails significatifs, il faut donc baisser le paramètre de lissage à l'intérieur de celle-ci.

- On est en train de créer la frontière entre deux régions de couleurs moyennes données. Ce pixel possédant une couleur éloignée de ces moyennes ne peut correspondre qu'à du bruit.

### 5.2.2 Les méthodes d'agrégation de pixels

#### 5.2.2.1 Introduction

Le méthodes d'agrégation de pixels (ou croissance de régions) sélectionnent des groupes de pixels de l'image appelés **germes**. Ces germes, éventuellement réduits à un pixel sont généralement placés dans les zones les plus homogènes de l'image. Les algorithmes d'agrégation font ensuite croître les germes en agrégeant les pixels de l'image aux germes les plus proches géométriquement ou photométriquement.

Ces algorithmes produisent donc une partition de l'image en régions et se décomposent en deux étapes [CG84] :

- La création des germes initiaux qui peut se faire à l'aide du gradient, de la variance ou de tout autre critère.
- La croissance des germes qui s'effectue en affectant chaque pixel de l'image à un germe.

La croissance des germes s'effectue généralement grâce à un prédicat à deux arguments appelé **Oracle**. Ce prédicat permet de décider si un pixel $p$ doit ou non appartenir à un germe $g$. A une étape $k$ de l'algorithme, l'oracle peut décider de l'agrégation d'un pixel à un germe $g$ en fonction des attributs du germe $g$ ou en fonction des attributs des pixels agrégés à $g$ à l'étape $k-1$. Cette dernière heuristique permet d'obtenir des régions qui présentent un changement continu d'homogénéité. Un cas typique de ce genre de phénomènes est obtenu avec les dégradés, la région contenant le dégradé n'est pas homogène mais toute découpe de cette région apparaîtrait artificielle.

La méthode de partage des eaux par immersion [DL78, VS91] constitue un bon exemple d'algorithme d'agrégation. Cette méthode peut se décomposer comme suit :

1. Obtenir une image des gradients à partir de l'image à segmenter.
2. Initialiser les germes correspondant aux pixels de plus bas gradient dans l'image des gradients. Marquer ces pixels.
3. A l'étape $i$ :
   - Sélectionner les pixels non marqués de plus bas gradient.
   - Marquer les pixels sélectionnés.
   - Agréger à un germe tout pixel sélectionné et adjacent à celui-ci.
   - Initialiser de nouveaux germes à partir des pixels sélectionnés restant.
4. Répéter le point 3 tant qu'il reste des pixels non marqués.

#### 5.2.2.2 Les algorithmes d'Unir-Trouver

Dillencourt et al. [DST92] ont remarqué en 1992 que les problèmes d'agrégation pouvaient se ramener à un problème d'union d'ensembles disjoints. En effet, l'étude d'ensembles disjoints pose deux problèmes :

1. Si on se donne un élément et un ensemble d'ensembles disjoints, *trouver* l'ensemble qui contient l'élément.
2. Si on se donne deux ensembles disjoints *unir* ces deux ensembles en un seul.

La complexité théorique des meilleurs algorithmes de ce type est en $\mathcal{O}(\alpha(n,m)m)$ [Tar79] où $\alpha$ est une fonction à croissance très lente et où $n$ et $m$ $(n < m)$ représentent respectivement le nombre d'opérations Trouver et Unir.

Ces algorithmes choisissent arbitrairement un élément d'un ensemble et le définissent comme représentant canonique de l'ensemble. Les autres éléments sont alors rattachés à l'élément canonique par une structure arborescente. L'agrégation d'un élément à un ensemble ou l'union de deux ensembles s'effectue alors simplement en rattachant la racine d'un des deux ensembles à un noeud de l'autre ensemble. Cette opération est effectuée par l'algorithme *Uni*. Partant d'une feuille d'un arbre de hauteur $h$ on doit effectuer $h$ indirections avant d'obtenir le représentant canonique de l'ensemble. Ces indirections sont supprimées au moyen de l'algorithme 5.1 (voir également la Figure 5.2).

```
TrouveCompresse(p)
{
    si ( racine(p) )
        return p;
    sinon
    {
        p.parent = TrouveCompress(p.parent)
        return p.parent
    }
}
```

Figure 5.1: *Algorithme de compression de chemins.*

Cet algorithme rattache directement à la racine tous les noeuds situés entre une feuille et la racine. Cette compression de chemin peut être étendue à un arbre entier grâce à l'algorithme *Aplatit* qui remonte tous les noeuds d'un l'arbre au niveau de la racine.

Les algorithmes *Uni, TrouveCompresse, Aplatit* et le critère Oracle permettent de définir des algorithmes efficaces de segmentation par agrégation. L'algorithme proposé par Fiorio [FG94, FG96] illustre une utilisation possible de ces primitives (voir Algorithme 5.3)

Un des principaux avantages de cet algorithme est sa rapidité. En effet d'après Fiorio [FG94, FG96] l'algorithme 5.3 permet d'obtenir la segmentation d'une image $256 \times 256$ en $0,8$ secondes sur une station de travail courante.

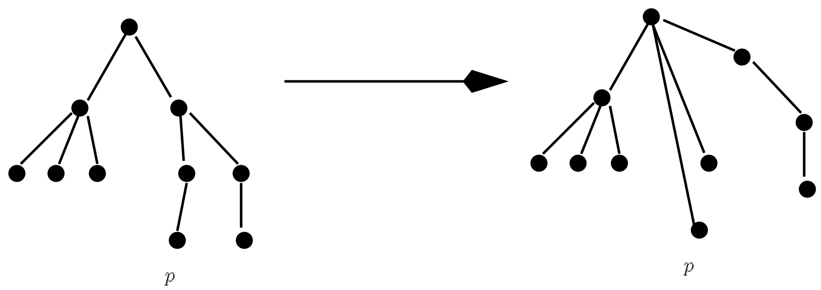



Figure 5.2: *Compression du chemin de p à la racine.*

```
Segmente(Image I)
{
    Initialiser chaque pixel comme la racine d'un arbre (un germe)
    Adapter le traitement pour la premiere ligne.
    Pour i = 2 a hauteur(I)
    {
        traitement special pour la premiere colonne
        Pour j=2 a largeur(I)
        {
            gauche = TrouveCompresse(I[i][j-1])
            haut = TrouveCompresse(I[i-1][j])
            courant =TrouveCompresse(I[i][j])
            si (Oracle(gauche,courrant))
                Uni(gauche,courant)
            si (Oracle(haut,courant))
                Uni(haut,courant)
        }
        Aplatit (ligne i)
    }
}
```

Figure 5.3: *Algorithme de type Unir-Trouver.*

#### 5.2.2.3 Conclusion

Les algorithmes d'agrégation de pixels permettent de partitionner rapidement une image en un ensemble de régions. Les contours définis implicitement sont donc naturellement fermés. Le problème majeur de ces méthodes vient de l'initialisation. En effet lors des premières itérations de ce type d'algorithme les germes sont composés de très peu de pixels. Les caractéristiques que le prédicat Oracle extraira de ces régions sont peu fiables. De plus les premières agrégations de pixels basées sur ces informations peuvent influencer les agrégations suivantes. Fiorio [FG94, FG96] limite ce problème en imposant à tous les germes d'avoir approximativement le même cardinal.

### 5.2.3 Les méthodes de fusion

#### 5.2.3.1 Introduction

Une variante des algorithmes d'agrégation consiste à créer une partition de l'image en régions puis à fusionner les régions adjacentes en fonction d'un critère d'homogénéité. Les principales différences entre les méthodes d'agrégation et les méthodes de fusion sont les suivantes :

- L'ensemble des germes initiaux forme une partition de l'image.
- Chaque région a une taille supérieure au pixel.

Ces différences en apparence mineures distinguent deux méthodologies :

- Les méthodes d'agrégation de pixels qui considèrent chaque pixel afin de l'agréger à un germe.
- Les méthodes de fusion qui considèrent un ensemble de couples de régions et fusionnent les régions adjacentes en fonction d'un critère d'homogénéité.

Les méthodes de fusion utilisent donc un algorithme de segmentation permettant d'obtenir un ensemble de régions homogènes partitionnant l'image. Partant de cette partition, ils fusionnent les régions adjacentes jusqu'à ce qu'aucun couple de régions ne vérifie le critère d'homogénéité. Ce critère permet de définir un prédicat **Fusionne** indiquant si deux régions doivent être fusionnées ou non. A la différence du prédicat Oracle vu dans la section 5.2.2, le prédicat Fusionne prend en paramètres deux régions et renvoie un booléen.

#### 5.2.3.2 Le Prédicat de Beveridge

Le prédicat Fusionne de Beveridge [BGK+89] est basé sur un critère d'homogénéité utilisant beaucoup de paramètres des régions. Il permet donc d'avoir un aperçu des critères usuellement utilisés par les algorithmes de fusion. La définition du critère d'homogénéité de Beveridge utilise les données suivantes :

- La moyenne $\mu_i$ du multi-ensemble associé à une région $i$ (sa couleur ou son niveau de gris moyen).
- l'écart type $\sigma_i$ d'une région $i$.
- Le nombre de pixels $|R_i|$ d'une région $R_i$.
- le contour $\delta R_i$ d'une région $R_i$.
- Le périmètre $|\delta R_i|$ d'une région
- La longueur de la frontière commune à deux régions $i$ et $j$ : $|\delta R_i \cap \delta R_i|$.

Le critère de Beveridge peut se décomposer en trois fonctions $S_{sim}$, $S_{size}$ et $S_{conn}$. Le critère $S_{sim}$ permet de mesurer la similitude de deux régions. Dans le cas mono-dimensionnel celui-ci est défini par :

$$S_{sim}(R_i, R_j) = \frac{|\mu_i - \mu_j|}{\max(1, \sigma_i + \sigma_j)}$$

Beveridge étend ce critère aux images couleurs par la formule suivante :

$$S_{sim}(R_i, R_j) = \frac{max_{k\in\{1,\dots,3\}}|\mu_{i_k} - \mu_{j_k}|}{\max(1, \sigma_i + \sigma_j)}$$

où $\mu_{i_k}$ et $\mu_{j_k}$ représentent la $k^{ieme}$ composante des vecteurs $\mu_i$ et $\mu_j$. Les écarts-types $\sigma_i$ et $\sigma_j$ sont dans ce cas calculés le long de l'axe d'intensité.

La mesure de similarité $S_{sim}$ représente le rapport entre la différence des deux moyenne et la somme des écart-types. Si $S_{sim}(R_i, R_j) = 0$, cela indique une très forte propension à la fusion. Ce cas se produit si les deux régions ont exactement la même moyenne. Si $S_{sim}(R_i, R_j) = 1$ on se trouve en présence de deux régions ayant une attitude neutre vis à vis de la fusion. Dans ce cas les deux moyennes sont éloignées de deux fois $\frac{\sigma_i+\sigma_j}{2}$. Des valeurs de $S_{sim}$ supérieures à 1 contrindiquent la fusion des deux régions.

Le second critère $S_{size}$ permet d'indiquer la taille approximative des régions finales. Si nous notons la taille optimum $T_{opt}$, la fonction $S_{size}$ est définie par :

$$S_{size}(R_i, R_j) = \min\left(2.0, \frac{\min(|R_i|, |R_j|)}{T_{opt}}\right)$$

Une valeur typique de $T_{opt}$ pour des images $256 \times 256$ est de l'ordre de 50. Encore une fois une valeur de $S_{size}$ proche de 0 signifie qu'une des deux régions à une taille très inférieure à la taille optimum. La fusion est alors recommandée. Une valeur de $S_{size}$ proche de 1 indique que la plus petite des deux faces est proche de la taille optimum. Ce genre de valeur indique une neutralité du paramètre $S_{size}$ vis à vis de la fusion. Des valeurs de $S_{zize}$ proche de 2 ou égale à 2 indique que les deux faces ont une surface supérieure à la surface optimum. Dans ce cas la fusion est déconseillée.

Le dernier paramètre $S_{conn}$ permet de mesurer la longueur de l'intersection de deux régions. Cette mesure correspond à l'idée intuitive que deux régions partageant un long contour sont de bons candidats à la fusion. $S_{conn}$ est défini comme suit :

$$S_{conn}(R_i, R_j) = \begin{cases} c(R_i, R_j) & \text{si } \frac{1}{2} \leq c(R_i, R_j) \leq 2 \\ \frac{1}{2} & \text{si } c(R_i, R_j) < \frac{1}{2} \\ 2 & \text{sinon} \end{cases}$$

où $c(R_i, R_j)$ est défini par :

$$c(R_i, R_j) = \frac{\min(|\delta R_i|, |\delta R_j|)}{4|\delta R_i \cap \delta R_j|}$$

La fonction $S_{conn}$ aura pour valeur 1 si elle est appliquée à deux carrés adjacents par un côté. Elle décroit au fur et à mesure que la frontière commune s'agrandit et croit si la frontière entre les deux régions se réduit.

Beveridge unit les trois critères $S_{sim}$, $S_{size}$ et $S_{conn}$ en un seul critère $S$ par la formule suivante :

$$S(R_i, R_j) = S_{sim}(R_i, R_j)\sqrt{S_{size}(R_i, R_j)}S_{conn}(R_i, R_j)$$

La racine carrée appliquée sur la mesure $S_{size}$ exprime l'importance relative de ce critère par rapport aux deux autres. Le prédicat Fusionne peut alors être construit à partir du critère $S$ de la façon suivante :

$$\text{Fusionne}(R_i, R_j) = \text{vrai} \Longleftrightarrow S(R_i, R_j) = \min_{(k,l)\in\{1,\ldots,N\}^2} S(R_k, R_l)$$

où $N$ représente le nombre de régions.

Il est bien évident que les critères $S$, $S_{sim}$, $S_{size}$ et $S_{conn}$ sont basés sur des heuristiques et ne tentent pas de fournir un résultat optimum. Toutefois, les expériences menées par Beveridge semblent montrer que le critère $S$ et son prédicat associé Fusionne fournissent de bons résultats sur une grande variété d'images.

#### 5.2.3.3 L'algorithme de Brice et Fennema

Brice et Fennema [BF70] ont été les premiers à introduire les algorithmes de fusion. Leur algorithme appelé *algorithme phagocyte* est basé sur le potentiel des points de contour. Le potentiel d'un point peut par exemple se définir comme le gradient calculé en ce point. On peut alors définir la force d'un segment définissant la frontière entre deux régions comme la somme des potentiels des points du segment. Ceci nous permet d'attacher deux attributs à un segment $S$ :

- Sa longueur $|S|$ égale au nombre de points du segment.
- Sa faiblesse $|S|_w$ égale au nombre de points du segment dont le potentiel est inférieur à une constante $\sigma$.

Si nous définissons $|\delta R_i|$ comme le périmètre de la région $i$, donc comme la somme des longueurs des segments définissant $R_i$, l'algorithme phagocyte peut se décomposer comme suit :

Cet algorithme fusionne donc toutes les régions dont la frontière possède un faible potentiel. Le paramètre $\frac{1}{\min(|\delta R_i|,|\delta R_j|)}$ permet de comparer la faiblesse du segment au périmètre des deux régions $R_i$ et $R_j$. Ce paramètre normalisateur permet de ne pas fusionner en priorité les petites régions au détriment de régions plus importantes possédant des frontières de potentiel faible relativement à leur taille. Le paramètre $\theta_1$ permet de contrôler

```
phagocyte()
{
    Pour toutes regions R_i, R_j
    {
        S_ij = δR_i ∩ δR_j
        Si (|S_ij|_w / min(|δR_i|,|δR_j|) < θ_1)
        {
            fusionner R_i et R_j en supprimant S_ij
        }
    }
}
```

Figure 5.4: *Algorithme phagocyte de Brice et Fennema.*

l'algorithme phagocyte. Si $\theta_1$ est trop faible de nombreuses régions très similaires restent non fusionnées, tandis que si $\theta_1$ est trop important l'algorithme risque de fusionner trop de régions et de supprimer des contours importants de l'image.

#### 5.2.3.4 Conclusion

Les algorithmes de fusion sont très similaires aux algorithmes d'agrégation de pixels. La différence principale se situe dans la définition de prédicat utilisé. Les méthodes d'agrégation définissent l'oracle comme un prédicat qui prend en paramètre une région et un pixel et indique si le pixel doit être agrégé ou non à la région. Le prédicat Oracle des algorithmes de fusion prend en paramètre deux régions et indique si celles-ci doivent être fusionnées ou non. Les méthodes de fusion sont souvent utilisées comme post-traitement [HP75, HP76, CP79, PRW82] afin de supprimer des régions insignifiantes créées par l'algorithme de segmentation.

Si nous reprenons le formalisme de la définition 22, la démarche d'un algorithme de fusion peut se décrire de la façon suivante :

- Pour un ensemble de régions $\{S_1, \ldots, S_n\}$ vérifiant les points 1 à 3 de la définition 22,
- fusionner toutes les régions $S_i$, $S_j$ qui ne vérifient pas le point 4.

Les régions résultant de l'algorithme ne vérifient plus forcément le point 3. Un partitionnement vérifiant simultanément les points 3 et 4 peut être obtenu en alternant le découpage des régions ne vérifiant pas 3 et la fusion des régions ne vérifiant pas 4. Ce type de méthode sera étudié dans la section 5.2.5.

### 5.2.4 Les méthodes de division

#### 5.2.4.1 Introduction

Les méthodes de division utilisent le fait que de nombreuses informations contenues dans l'image restent présentes dans le multi-ensemble associé à l'image. Par exemple, l'image couleurs d'un coucher de soleil sur la mer induira un multi-ensemble 3D $(C, f)$ avec de fortes fréquences dans le rouge (pour le soleil) et le vert sombre (pour la mer). Si nous partitionnons $(C, f)$ en deux multi-ensembles regroupant les rouges et les verts, le regroupement dans une même région des pixels dont la couleur appartient au même multi-ensemble (voir définition 17) donnera une segmentation correcte de l'image.

Un grand nombre d'algorithmes de segmentation étant basés sur le partitionnement de multi-ensembles 1D (ou histogrammes) nous allons tout d'abord étudier les principales méthodes utilisée pour le partitionnement d'histogramme. Les méthodes de division de multi-ensembles 3D seront étudiées dans la section 5.2.6.

Le partitionnement d'un multi-ensemble 1D consiste à définir un ensemble de seuils $\{T_0, \ldots, T_N\}$ découpant l'intervalle $[m, M]$ du multi-ensemble en $N$ intervalles. Le partitionnement de l'intervalle $[m, M]$ induit naturellement une partition du multi-ensemble $([m, M], f)$ en $N$ multi-ensembles $\{([T_0, T_1], f), \ldots, [T_{N-1}, T_N], f)\}$. La définition des seuils peut être basée uniquement sur l'histogramme ou utiliser le voisinage des pixels ou encore utiliser simultanément l'histogramme, le voisinage et la position des pixels. Weszka [Wes78] a défini 3 types de seuillages :

- Les seuillages globaux basés uniquement sur l'histogramme.
- Les seuillages locaux basés sur l'histogramme et sur un voisinage de chaque point de l'image.
- Les seuillages dynamiques qui tiennent simultanément compte de l'histogramme, du voisinage d'un point et de sa position dans l'image.

#### 5.2.4.2 Les seuillages globaux

Ce type de seuillage basé uniquement sur l'histogramme peut à nouveau se décomposer en deux grandes familles.

- Les méthodes basées sur l'homogénéité des multi-ensembles formant la partition
- Les méthodes décomposant l'histogramme en pics et vallées.

La décomposition d'un histogramme en pics et vallées repose sur l'assomption que la distribution des niveaux de gris d'une région suit approximativement une loi gaussienne. Si par exemple l'on construit l'histogramme d'un désert en milieu de journée, l'on obtient un ensemble de niveaux de gris "proches" du niveau de gris moyen du sable et un autre

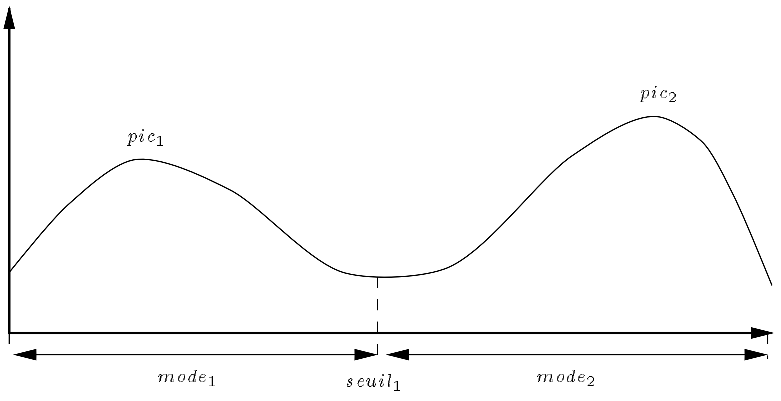



Figure 5.5: *Un histogramme bi-modal.*

ensemble centré sur le niveau de gris du ciel. On obtiendra donc l'histogramme de la Figure 5.5.

On observe que cet histogramme présente deux pics correspondant aux deux objets de la scène et un seuil entre ces pics. L'ensemble des niveaux de gris situés entre deux seuils est appelé un mode. Un seuil correspond intuitivement au centre de la vallée entre deux pics. Il peut se caractériser plus formellement de la façon suivante :

**Définition 23** *Soit* $([m, M], f)$ *un multi-ensemble avec* $f$ *deux fois dérivable. La valeur* $s$ *appartenant à* $[m, M]$ *sera appelé un seuil si :*

$$\frac{\partial f}{\partial x}(s) = 0 \ \ et \ \ \frac{\partial^2 f}{\partial x^2}(s) > 0$$

Le multi-ensemble étant généralement discret et bruité, la définition 23 ne peut être appliquée immédiatement. Bhattacharya [Bha67] définit les seuils comme les limites supérieures de décroissance de la suite $ln(f(n)) - ln(f(n-1))$ avec $n \in [m, M]$ (voir Figure 5.6).

Bartels [Bar79] a amélioré cette méthode en approximant les modes compris entre chaque seuil par des fonctions gaussiennes. Cette méthode lui permet d'améliorer itérativement la partition obtenue par Bhattacharya. Enfin, la méthode adoptée par Papamarkos [PG94] peut être vue comme le symétrique de celle adoptée par Bhattacharya et Bartels. Papamarkos détermine d'abord les pics de l'histogramme puis approxime les vallées entre les pics par des fonctions rationnelles. Le minimum de ces fonctions rationnelles entre chaque pic fournit ensuite les différents seuils.

Une autre solution consiste à lisser le signal par un filtre passe-bas. Cette méthode permet de diminuer le bruit présent dans la fonction $f$. De plus, la convolution avec un filtre passe-bas nous donne un signal d'une classe au moins égale à celle du filtre (voir proposition 1). Si l'on effectue le lissage en convoluant le signal avec une gaussienne on peut utiliser l'étude de Witkin [Wit84] pour définir le paramètre de lissage optimum. La donnée de ce paramètre permet alors d'utiliser la caractérisation des seuils fournie par la définition 23. Cette approche a été utilisée par Lim [LL90] et Celenk [Cel90] pour la segmentation d'images couleurs.

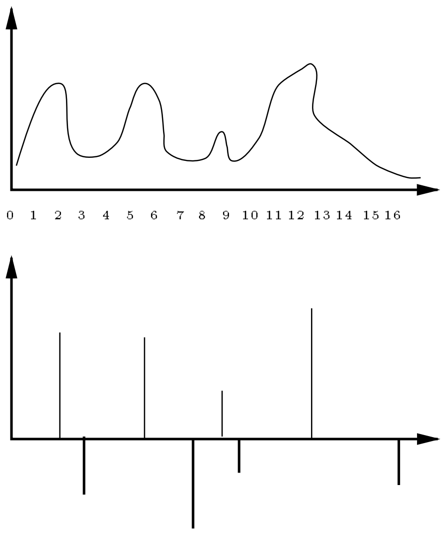



Figure 5.6: *Définition des seuils par la méthode de Bhattacharya.*

Une autre approche consiste à partitionner le multi-ensemble en $N$ multi-ensembles tels que l'erreur de la partition (voir définition 6) soit minimale. Minimiser l'erreur de la partition impose de définir les multi-ensembles les plus homogènes possible. Nous avons vu au chapitre 2 que la partition optimale d'un multi-ensemble pouvait être trouvée avec un algorithme de complexité $\mathcal{O}(NlogN)$ où $N$ est le nombre de multi-ensembles formant la partition. Ce type de méthodes développées par Fisher [Fis58] et Wong [WWP89] imposent de fournir à l'algorithme le nombre de multi-ensembles formant la partition. Ce dernier point impose, soit de demander à l'utilisateur de fournir cette information, soit de restreindre ces algorithmes à des classes d'images où $N$ est connu a priori. Fukunaga [Fuk72], Nagy [Nag68] et Andrews [And72] ont défini des critères basés sur la maximisation d'un paramètre $\beta$ permettant de minimiser l'erreur de la partition tout en minimisant l'écart des multi-ensembles les uns par rapport aux autres. Le caractère $\beta$ permet de mesurer la qualité d'une partition. Le nombre optimal de multi-ensembles $N$ est alors défini comme celui qui maximise $\beta$. La définition de $\beta$ la plus couramment utilisée est donnée par l'équation suivante :

$$\beta(N) = E_N(C)\Delta(N) \text{ avec } \Delta(N) = \sum_{i=1}^{N} \|\mu_k - \mu_0\|^2$$

où $E_N(C)$ représente l'erreur de la partition du multi-ensemble $(C, f)$ en $N$ multi-ensembles, $\mu_0$ la moyenne de $(C, f)$ et $\mu_k$ la moyenne de chacun des multi-ensembles formant la partition.

Le paramètre $\beta$ est toujours positif, de plus si la partition est réduite à un seul multi-ensemble, on a $\Delta(1) = 0$, donc $\beta(1) = 0$. De même si l'on partitionne le multi-ensemble $(C, f)$ en M multi-ensembles où $M$ est le nombre de points distincts de $C$, chaque multi-ensemble de la partition est réduit à un point. Son erreur quadratique est donc nulle et l'erreur de la partition (somme des erreurs quadratiques des multi-ensembles) est également nulle. On a donc :

$$\beta(M) = E_M(C).\Delta(M) = 0.\Delta(M) = 0$$

La fonction $\beta$ est donc positive ou nulle et s'annule en 1 et en $M$. Elle passe donc au moins par un maximum. On peut montrer [CA79] que $\beta$ atteint un maximum lorsque :

$$E_N(C) = \Delta(N)$$

La valeur de $N$ réalisant cette dernière égalité est prise comme valeur optimale.

#### 5.2.4.3 Les seuillages locaux

Nous avons vu précédemment que le bruit présent dans l'image perturbe les méthodes globales. Ces méthodes diminuent les effets du bruit en lissant l'histogramme. Il est également possible de lisser l'image puis de travailler avec l'histogramme de l'image lissée. On peut enfin incorporer la notion de voisinage dans le calcul de l'histogramme. C'est cette dernière démarche qui a été adoptée par Mardia et Hainsworth [Mar88]. Ceux ci approximent l'histogramme de l'image par la somme de gaussiennes de même écart type $\sigma$ et de moyenne $\mu_i$. Les seuils sont alors définis par :

$$s_{ij} = \frac{\mu_i + \mu_j}{2} + \frac{\sigma^2}{\mu_i - \mu_j} ln \frac{|C_j|}{|C_i|}$$

où $\mu_i$ et $\mu_j$ sont les moyennes de deux gaussiennes consécutives et $C_i$ est le $i^{eme}$ multi-ensemble de la partition. On a donc $\mu_i = \mu(C_i)$. Le caractère local de la méthode de Mardia et Hainsworth vient de l'amélioration de ce premier partitionnement de l'histogramme par l'introduction d'un paramètre $Y$ défini en chaque point par :

$$Y[i,j] = \sum_{k=-V_x}^{V_x} \sum_{l=V_y}^{V_y} \gamma[k][l] I[i-k][j-l]$$

La variable $Y$ est donc définie en chaque point comme la convolution de l'image avec un masque $\gamma$. Ce masque étant un filtre passe-bas, la distribution $Y$ se rapproche plus d'une distribution gaussienne que l'image originale $I$.

Mardia et Hainsworth définissent alors une matrice de covariance $R$ permettant d'approximer par une gaussienne la distribution de $Y(x)$ pour x appartenant à $C_i$. La méthode est basée sur le fait que $Y(C_i)$ est plus proche d'une gaussienne que la distribution initiale de $C_i$. Partant de cette approximation on définit de nouveaux seuils $s^*_{ij}$ par :

$$s^*_{ij} = \frac{\mu_i + \mu_j}{2} + \frac{\sigma^2 \gamma'^t R \gamma}{\mu_i - \mu_j} ln \frac{|C_j|}{|C_i|}$$

où $\gamma'$ est le vecteur unidimensionnel défini par : $\gamma' = (\gamma[-V_x][-V_y], \gamma[-V_x][-V_y+1], \ldots, \gamma[V_x, V_y])$ On obtient alors une nouvelle partition et l'on peut itérer le processus jusqu'à stabilisation.

La méthode de relaxation-gradient de Bhanu [BF82, BP87] permet de partitionner un multi-ensemble en deux lorsque celui-ci ne présente pas de caractère bi-modal marqué. Cette méthode est basée sur le raisonnement suivant : si l'image est composée de régions comportant plus de huit pixels, la partition optimale du multi-ensemble $(C, f)$ en $(C_1, f)$ et $(C_2, f)$ doit vérifier :

$$\forall p \begin{cases} I(p) \in C_1 \Rightarrow |I(V_8(p)) \cap C_1| > |I(V_8(p)) \cap C_2| \\ I(p) \in C_2 \Rightarrow |I(V_8(p)) \cap C_2| > |I(V_8(p)) \cap C_1| \end{cases}$$

où $V_8$ désigne le voisinage 8-connexe d'un point.

Énoncé de façon moins formelle, cela signifie que si un point appartient à un multi-ensemble $C_i$, la majorité de ses voisins appartient à $C_i$. Bhanu définit donc deux vecteurs de probabilité. Un vecteur $P$ indiquant la probabilité pour un point $s$ d'appartenir à $C_1$ ou $C_2$. Ce vecteur est défini par :

$$P(s) = \begin{pmatrix} P_{C_1}(s) \\ P_{C_2}(s) \end{pmatrix} \text{ avec } P_{C_1}(s) + P_{C_2}(s) = 1$$

Et un vecteur $Q$ appelé vecteur de compatibilité est défini par :

$$Q(s) = \begin{pmatrix} Q_{C_1}(s) \\ Q_{C_2}(s) \end{pmatrix} \text{ avec } Q_{C_i}(s) = \frac{1}{8} \sum_{t \in V_8(s)} P_{C_i}(t)$$

Pour chaque pixel $s$ la valeur : ${}^tP(s)Q(s) = P_{C_1}Q_{C_1}(s) + P_{C_2}Q_{C_2}(s)$ indique la probabilité que le pixel $s$ appartienne à l'un ou l'autre des multi-ensembles et la compatibilité de ce choix avec les pixels voisins. Si nous numérotons les pixels de l'image de 1 à M, le critère à maximiser est égal à :

$$J = \sum_{i=1}^{M} {}^tP(s_i)Q(s_i) \tag{5.4}$$

Bhanu déduit ensuite de l'équation 5.4 un système itératif permettant de faire converger la suite $P^n(s)$ vers le vecteur de probabilité maximisant $J$.

#### 5.2.4.4 Les seuillage dynamiques

Les méthodes de seuillage global et local permettent d'obtenir de bons résultats lorsque l'image est composée d'un nombre réduit de régions de niveaux de gris moyen suffisamment éloignés. Lorsque l'image comporte un nombre important de régions de niveaux de gris moyen très proches le multi-ensemble associé à l'image présente des pics et des vallées peu marqués. Ceci peut rendre la détection des seuils très difficile voire impossible.

Une solution à ce problème consiste à combiner les caractéristiques (position, gradient, laplacien) des points de l'image avec le multi-ensemble. Watanabe et al. [Wat74] utilisent l'information de gradient pour détecter les seuils. En effet à chaque niveau de gris $z$ nous pouvons faire correspondre l'ensemble $S_z$ des pixels dont le niveau de gris est égal à $z$. Les valeurs de $z$ correspondant à des vallées proviennent souvent des zones frontières entre les régions. Ces zones correspondent dans l'image à des zones de fort gradient. Donc, le niveau de gris $z$ tel que la somme des gradients calculés sur $S_z$ est maximum doit correspondre au centre d'une vallée, donc à un seuil. Watanabe définit donc la fonction :

$$d(z) = \sum_{s \in S_z} \|\nabla I(s)\|$$

où $\|\nabla I(s)\|$ représente la norme du gradient au point $s$.

Dans le cas d'un histogramme bi-modal Watanabe définit le seuil comme le $z$ maximisant $d(z)$.

Hertz [HS88] utilise l'information gradient d'une façon différente. Il partitionne l'image en un ensemble de blocs puis calcule un histogramme sur chaque bloc. Il associe ensuite à chacun de ces blocs un ensemble de seuils et partitionne chaque bloc en régions à l'aide de ces seuils. Cette partition en régions à l'intérieur de chaque bloc lui permet de comparer deux images de contours : l'image $IS$ issue de la partition en région et l'image $IG$ obtenue à partir de l'image $I$ par un opérateur gradient. Les pixels des deux images correspondant à des contours sont marqués à $+1$. L'image différence $ID = IG - IS$ est alors utilisée pour modifier les seuils à l'intérieur de chaque bloc. Les règles utilisées pour ces modifications sont les suivantes :

- Si un bloc de $ID$ comporte beaucoup de $-1$ et de $+1$ cela signifie que nous avons trouvé le bon nombre de seuils mais que ceux-ci sont mal placés. On utilise alors le bloc correspondant dans $IG$ pour modifier les seuils.

- Si un bloc contient beaucoup de pixels à $-1$ et peu à $+1$ cela signifie qu'un faux seuil a été défini dans l'histogramme du bloc. On élimine alors le seuil le moins significatif de la liste des seuils.

- Si un bloc comporte un beaucoup de pixels à $+1$ et peu à $-1$ cela signifie qu'un seuil nécessaire est absent. On introduit donc un nouveau seuil en utilisant $IG$.

L'aspect dynamique de la méthode de Hertz vient de la décomposition de l'image en blocs. On peut remarquer que tout l'algorithme présuppose que l'image des contours $IG$ est correcte, donc que le gradient n'a pas introduit de contours insignifiants et n'a pas "raté" de contours importants.

Chow, Kaneko [CK72] et Nakagawa [NR79] utilisent également le partitionnement de l'image en blocs. Leur méthode consiste à calculer un histogramme sur chaque bloc de l'image puis à approximer celui-ci par la somme de deux (méthode de Chow) ou trois (méthode de Nakagawa) gaussiennes. Dans le cas d'une approximation bi-modale, donc à deux gaussiennes le seuil de l'histogramme est défini comme la valeur $s$ minimisant l'erreur de classification. Le seuil $s$ vérifie l'équation :

$$(\frac{1}{\sigma_1^2} - \frac{1}{\sigma_2^2})s^2 + 2(\frac{\mu_2}{\sigma_1^2} - \frac{\mu_1}{\sigma_2^2})s - \frac{\mu_2}{\sigma_2^2} + \frac{\mu_1}{\sigma_1^2} + 2ln(\frac{p_2\sigma_1}{p1\sigma_2}) = 0 \qquad (5.5)$$

où $p_1$ et $p_2$ sont deux paramètres issus de l'approximation de l'histogramme par deux gaussiennes et $\mu_1$,$\mu_2$ ,$\sigma_1$ et $\sigma_2$ désignent les moyennes et écart-types des deux gaussiennes. Le seuil $s$ défini par l'équation 5.5 est affecté au centre de chaque bloc. Les blocs pour lesquels il est impossible de définir une approximation bi-modale reçoivent un seuil égal à la moyenne des seuils des blocs voisins. Chaque pixel de l'image reçoit alors un seuil défini par interpolation bilinéaire à partir des seuils des blocs avoisinant. Si $I$ désigne l'image originale et $IT$ l'image des seuils, l'image $IB$ issue de la partition est définie par :

$$IB[i,j] = \begin{cases} 0 \text{ si } I[i,j] > IT[i,j] \\ 1 \text{ sinon} \end{cases}$$

Si l'on approxime l'histogramme de chaque bloc par trois gaussiennes il faut alors définir une seconde image $IT2$ correspondant au seuil entre la deuxième et troisième gaussienne.

#### 5.2.4.5 Les méthodes de division récursive

L'ensemble des méthodes vues dans cette section permettent de décomposer une image en régions à partir du multi-ensemble associé à celle-ci. Nous avons vu que lorsque l'image est trop complexe l'information contenue dans le multi-ensemble est trop pauvre pour permettre une segmentation efficace de l'image. L'intégration de caractéristiques locales de l'image par les méthodes de seuillage dynamique permet de résoudre en partie ce problème. Toutefois la décomposition de l'image en blocs coupe quelquefois artificiellement une région (voir Figure 5.7).

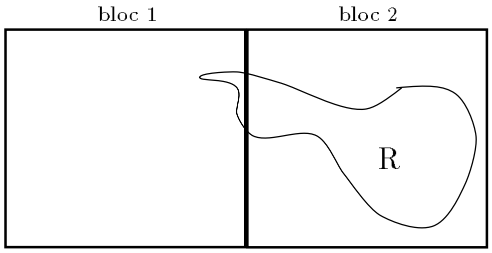



Figure 5.7: *Une région à cheval sur deux blocs.*

Dans l'exemple de la Figure 5.7 le bout de la région **R** appartenant au bloc 1 est trop minoritaire dans le bloc pour créer une valeur de seuil ce bout de région disparaîtra donc de la région **R**. Ce genre de phénomènes crée des artéfacts en forme de rectangle dans l'image segmentée, laissant ainsi deviner la méthode utilisée pour obtenir l'image des contours. Ce problème est partiellement résolu par la méthode de Hertz [HS88]. Toutefois cette méthode suppose que l'on possède déjà une bonne segmentation en gradient ce qui pose d'autre problèmes. Beveridge [BGK+89] diminue ce problème en faisant se recouvrir partiellement les blocs. Beveridge utilise des blocs de $16 \times 16$ pixels, chaque bloc recouvrant 25% de ses voisins. Ce recouvrement permet de tenir compte dans chaque bloc des bouts de régions se trouvant dans les blocs voisins. Cette méthode ne supprime toutefois pas le problème et Beveridge doit combiner la partition des blocs avec un post-traitement [Koh84] (voir figure 5.8).

La méthode de division récursive permet de s'affranchir des problèmes liés au manque d'information contenue dans le multi-ensemble global. De plus ces méthodes peuvent s'appliquer à d'autres partitions que les partitions par blocs. Elles permettent donc de s'affranchir des problèmes liés à ce genre de partitionnement. Le principe général d'un algorithme de division récursive est le suivant :

1. Calculer le multi-ensemble de l'image.
2. Partitionner le multi-ensemble.
3. Calculer la partition de l'image induite par la partition du multi-ensemble.
4. Pour chaque région ainsi définie calculer son multi-ensemble.
5. Si un multi-ensemble généré en 4 est partitionnable, revenir en 2.

Figure 5.8: *Segmentation d'un paysage par la méthode de Beveridge. Les artéfacts en forme de rectangle sont partiellement corrigés*

Les méthodes de division récursive ont été abondamment explorées [OPR78, CA79, Lee86, LL90, Cel90]. Ce type de méthode peut se voir comme une focalisation de l'attention. La première partition du multi-ensemble découpe l'image en ses zones principales. Par exemple, les zones claires et sombres, dans le cas d'un multi-ensemble 1D bi-modal. Les itérations suivantes se concentrent sur les régions définies aux étapes précédentes et enrichissent leurs description en les partitionnant en plusieurs sous-régions. Les principales différences entre ces méthodes s'établissent sur les deux points suivants :

- La façon de partitionner les multi-ensembles.
- La structure de données utilisée pour gérer les régions issues des découpages successifs.

#### 5.2.4.6 Conclusion

Les méthodes de division partitionnent le multi-ensemble associé à l'image et déduisent de cette partition une partition de l'image en régions. Lorsque la classe d'images à segmenter est simple, des méthodes de seuillage global ou local permettent d'obtenir des résultats satisfaisants. Pour des images plus complexes l'information contenue dans le multi-ensemble associé à l'image n'est plus suffisante pour permettre une bonne segmentation de l'image. Les techniques de seuillage dynamique permettent de rajouter des informations plus locales comme le gradient ou le laplacien. Les techniques de seuillage dynamique par bloc permettent de définir des multi-ensembles locaux plus exploitables que le multi-ensemble global. L'utilisation de blocs induit toutefois des problèmes de coupure de régions coûteux à résoudre. Les méthodes de division récursive permettent de passer d'une description globale de l'image à une description plus locale et plus riche. L'inconvénient majeur de ce type de méthode est son irréversibilité. En effet, une région partitionnée à tort lors des premières étapes de l'algorithme ne sera jamais refusionnée en une seule région. Cet inconvénient sera supprimé par les méthodes de division et fusion.

### 5.2.5 Les méthodes de division et fusion

#### 5.2.5.1 Introduction

Les méthodes de division et fusion introduites par Horowitz et Pavlidis [HP75, HP76] combinent des méthodes de division récursive et de fusion. Si nous nous référons à la définition 22, les méthodes de division partent d'une partition de l'image et découpent ses régions jusqu'à ce que toutes les régions vérifient le critère d'homogénéité :

$$\forall i \in \{1, \ldots, n\} \quad P(S_i) = vrai \tag{5.6}$$

où $n$ représente le nombre de régions. Ces méthodes ne testent pas la maximalité des régions définie par le point 4 de la définition 22. Ces méthodes produisent donc souvent une sur-segmentation de l'image en créant de nombreuses petites régions sans signification.

Inversement, les méthodes de fusion partent d'une image sur-segmentée et fusionnent les régions jusqu'à ce que toutes les régions vérifient le critère de maximalité, a savoir :

$$\forall i, j \in \{1, \ldots, n\}^2 \quad i \neq j \Rightarrow P(S_i \cup S_j) = faux \tag{5.7}$$

Ces méthodes ne testant pas le point 3, risquent de fusionner abusivement plusieurs régions et donc de perdre des contours importants de l'image. On parle dans ce cas de sous-segmentation.

Les méthodes de division et fusion permettent de tenir compte simultanément des équations 5.6 et 5.7 et donc de produire une image segmentée respectant les quatre conditions de la définition 22 définissant une segmentation.

#### 5.2.5.2 Description de la méthode

Le processus de découpe-fusion peut être exprimé de façon plus visuelle en définissant **un arbre de partition**. Un arbre de partition est associé à une partition de l'image et doit vérifier les points suivants :

- L'image $I$ est la racine de l'arbre.
- Les successeurs d'un noeud correspondant à une région $R$ de l'image sont associés aux régions partitionnant $R$.
- Les feuilles de l'arbre correspondent aux plus petites régions pouvant être considérées (dans le pire des cas, les pixels).

La figure 5.9 représente un arbre de partition et la partition de l'image associée. Une

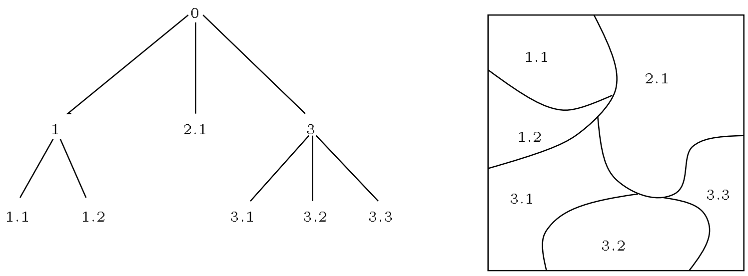



Figure 5.9: *L'arbre de partitionnement correspondant à une partition dl'image.*

segmentation correspond donc à la sélection d'un ensemble de noeuds tels qu'il existe un et un seul noeud sélectioné par branche. Une telle sélection définit une coupure [Pav77, HS79] (voir figure 5.10).

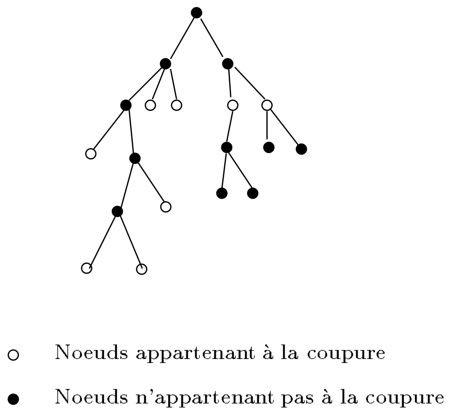
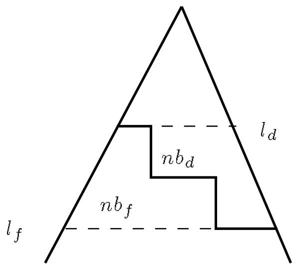



Figure 5.10: *Coupures de l'arbre vérifiant les point 3 et 4 de la définition 22.*

La méthode générale de division-fusion peut se décomposer de la façon suivante :

1. Partitionner l'image en un ensemble de régions de tailles fixes, créer l'arbre de partition associé.
2. Découper toutes les régions $R_i$ telles que $P(R_i) = faux$ en sous-régions. Ces régions sont considérées non homogènes (voir définition 22).
3. Fusionner les régions $R_i$, $R_j$ telles que $P(R_i \cup R_j) = vrai$ (voir définition 22). Les régions fusionnées doivent avoir un père commun dans l'arbre de partition.
4. Revenir au point 2 jusqu'à ce qu'aucune région ne puisse être découpée ou fusionnée.
5. Fusionner toutes les régions adjacentes $R_i$, $R_j$ telles que $P(R_i \cup R_j) = vrai$ (voir définition 22). Ces fusions ne sont plus restreintes par la structure d'arbre.

La partition initiale de l'image effectuée au point 1 est construite sans analyser l'image. Elle peut donc être effectuée très rapidement. L'analyse de l'image n'est effectuée qu'à partir du point 2, pour décider quelles régions doivent être découpées ou fusionnées. L'intérêt de cette partition initiale apparaît clairement si nous examinons l'arbre de partition. Soit $l_d$ et $l_f$ les niveaux de partitionnements nécessaires pour obtenir le partitionnement final en n'exécutant que des découpes ou que des fusions. Ces niveaux correspondent à des hauteurs dans l'arbre de partition (voir Figure 5.11). Un algorithme de découpe pur devra



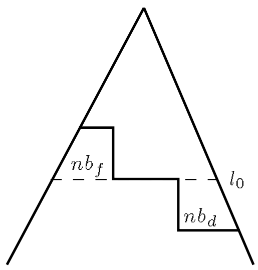

—— coupure correspondant à la segmentation finale

$nb_d$ nombre de découpes

$nb_f$ nombre de fusions

Figure 5.11: *Niveaux de partitionnements initiaux des algorithmes de découpes ou de fusion.*

sélectionner une partition initiale de niveau au plus égal à $l_d$. De même un algorithme de fusion devra partir d'un niveau au moins égal à $l_f$. Un algorithme de découpe-fusion peut partir d'un niveau intermédiare $l_0$ ($l_d < l_0 < l_m$). Cet algorithme devra exécuter moins d'opérations de découpes et de fusions que les algorithmes de découpe pure ou de fusion pure (voir Figure 5.12).




—— coupure correspondant à la segmentation finale

$nb_d$ nombre de découpes

$nb_f$ nombre de fusions

Figure 5.12: *Nombre de découpes et de fusions nécessaires à un algorithme de type découpe-fusion.*

L'utilisation du prédicat $P$ interdit aux algorithmes de division-fusion de reformer des faces fusionnées à une étape précédente. De même, il est impossible de fusionner des faces créées par une découpe. Le nombre de régions pouvant être générées par l'algorithme de découpe ou fusionnées par l'algorithme de fusion étant borné par la taille maximum de l'arbre de partition, les méthodes de division-fusion ne peuvent entrer dans des boucles infinies.

Les points (3) et (5) d'un algorithme de découpe-fusion distinguent les fusions restreintes par l'arbre de partitionnement et les fusions non restreintes. Cette distinction entre deux types de fusions est liée à des contraintes algorithmiques. Si l'arbre de partitionnement possède une structure "simple", il est souvent facile de déterminer rapidement un sous-ensemble de l'ensemble des faces voisines d'une région. Cette distinction permet donc de distinguer un algorithme de fusion restreint mais plus rapide d'un algorithme de fusion plus général mais plus coûteux. Celui-ci est donc appliqué en fin de traitement. Il est important de remarquer que l'arbre de partition ne peut rester simple que si les découpes successives ne brisent pas sa structure. L'algorithme de découpe doit donc également produire des partitionnements simples de chaque région.

C'est cette dernière option qui a été retenue par Horowitz. Celui-ci crée la partition initiale définie au point 1 en découpant l'image en quatre blocs puis chaque blocs en quatre sous-blocs jusqu'à obtenir un ensemble de blocs de taille voulue. L'arbre de partitionnement associé à une telle partition est un arbre quaternaire, appelé **quadtree** (voir Figure 5.13).

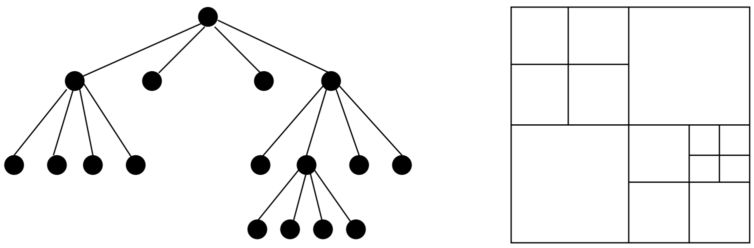


Figure 5.13: *Un arbre de partitionnement de type quadtree.*

Nous verrons plus en détail ce type de données au chapitre 6. L'utilisation d'un quadtree plutôt qu'un arbre de partitionnement général induit les simplifications suivantes sur l'algorithme de découpe-fusion :

1. Découper l'image en $N^2$ blocs carrés.
2. Découper tout bloc non homogène en quatre sous-blocs.
3. Si quatre blocs $B_1$,$B_2$,$B_3$,$B_4$ possédent le même père dans le quadtree et vérifient $P(B_1 \cup B_2 \cup B_3 \cup B_4) = vrai$, fusionner les quatre blocs en un seul. Les informations (taille, variance,...) de ces blocs sont stockées dans leur père.
4. Revenir au point 2 tant que l'on peut appliquer les points 2 ou 3.
5. Fusionner tous les couples de blocs adjacents $(B_1, B_2)$ vérifiant $P(B_1 \cup B_2) = vrai$.

Un des problèmes majeur de la technique d'Horowitz vient de l'utilisation de blocs. Comme nous l'avons mentionné dans la section 5.2.4, le partitionnement par blocs crée des artéfacts en forme de rectangle (voir Figure 5.8) dans l'image segmentée. Chen et Pavlidis [CP79] ont légèrement corrigé ce problème en découpant l'image en blocs carrés, chaque bloc chevauchant 50% des blocs voisins. Comme nous l'avons vu dans la section 5.2.4, cette solution diminue les artéfacts de l'image segmentée sans toutefois résoudre complètement le problème.

#### 5.2.5.3 Conclusion

Les méthodes de division et fusion ont été abondamment explorées [HP75, HP76, CP79, PRW82, Bro82, CMVM86]. Ces méthodes alternent des découpes et des fusions de régions de façon à partitionner l'image en un ensemble de régions vérifiant les quatre points de la définition 22. Les avantages de cette méthode sur les méthodes de découpe pure ou de fusion pure sont les suivants :

- Les algorithmes de découpe-fusion initialisent généralement un ensemble de régions grâce à une découpe uniforme de l'image. Ceci permet d'obtenir la partition finale plus rapidement qu'un algorithme de découpe pure ou de fusion pure.
- Le fait de pouvoir alterner des découpes et des fusions crée un degré de liberté suplémentaire permettant, a priori, d'obtenir une meilleure segmentation qu'une méthode n'utilisant que des découpes ou que des fusions.

Ces qualités des algorithmes de découpe-fusion sont toutefois limitées par des contraintes algorithmiques. De fait, la partition initiale utilisée par les algorithmes de découpe-fusion est généralement une partition par blocs. Les algorithmes de découpe et de fusion ne remettant pas en cause cet aspect bloc des régions initiales, la segmentation finale présente souvent des artéfacts en forme de rectangle. Un autre problème de cette méthode est la non-unicité de la solution. En effet pour un bloc $B_i$ donné, il existe plusieur blocs $B_j$ tels que $P(B_i \cup B_j) = vrai$. L'ordre dans lequel on effectue les fusions a une grande influence sur le résultat final. Cheevasuvit et al. [CMVM86] réduisent ce problème en calculant pour chaque bloc quatre regroupements (voir Figure 5.14) et en effectuant celui qui crée le bloc le plus homogène. Cette méthode ne résout toutefois pas les problèmes liés à la partition par bloc.

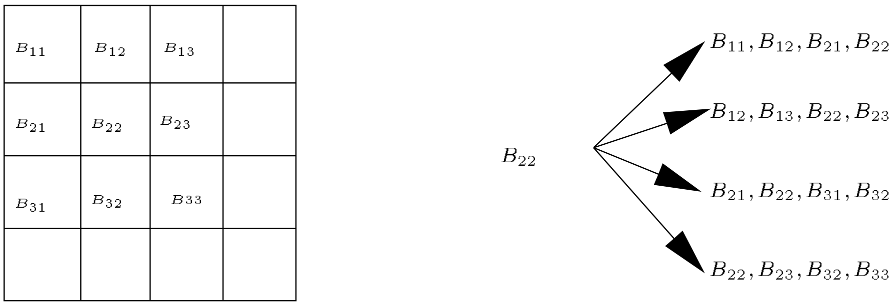



Figure 5.14: *Regroupements de blocs envisagés par Cheevasuvit.*

### 5.2.6 Méthodes pour traiter l'information couleur

#### 5.2.6.1 Introduction

Nous avons vu, dans les sections précédentes, différentes méthodes utilisées pour segmenter une image. La plupart de ces méthodes ont été conçues pour des images en niveaux de gris.

Les méthodes d'agrégations ou de fusions s'adaptent généralement très bien au passage en trois dimensions induit par la couleur. En effet ces méthodes utilisent les attributs des régions (moyenne, taille,...) et des pixels (couleur ou niveau de gris) qui se transposent très facilement aux images couleurs (voir par exemple la section 5.2.3.2). Les méthodes de détection de contours et de découpe s'adaptent en revanche plus difficilement. Les principales difficultés liées à l'introduction de la couleur sont les suivantes :

- La disparition de la notion d'ordre dans $\mathbb{R}$ que l'on ne retrouve pas dans $\mathbb{R}^3$.
- Le nombre de partitionnements possibles d'un sous-ensemble de $\mathbb{R}^3$ qui est beaucoup plus important que le nombre de partitionnements d'un segment de $\mathbb{R}$.

Nous allons voir dans les sections suivantes les différentes approches employées pour résoudre ces problèmes.

#### 5.2.6.2 La couleur et les algorithmes de détection de contours

Nous avons vu dans la section 5.2.1 qu'une méthode classique de segmentation d'image consiste à considérer une image en niveau de gris comme la discrétisation d'une fonction $f$ de $\mathbb{R}^2$ dans $\mathbb{R}$. À ce moment là les contours de l'image sont définis comme les maxima de la différentielle première $Df$ ou les zéros de la différentielle seconde $D_2f$.

Si nous appliquons la même démarche à l'analyse d'images couleurs, une image $I$ est considérée comme la discrétisation d'une fonction $f$ de $\mathbb{R}^2$ dans $\mathbb{R}^3$. Nous avons alors :

$$f\left(\begin{array}{ccc} \mathbb{R}^2 & \rightarrow & \mathbb{R}^3 \\ (x,y) & \mapsto & (f_1(x,y), f_2(x,y), f_3(x,y)) \end{array}\right.$$

La différentielle première de $f$ en $p$ dans la direction $n$ s'écrit alors :

$$Df(p).(n_x,n_y) = \begin{pmatrix} \frac{\partial f_1}{\partial x}(p).nx + \frac{\partial f_1}{\partial y}(p).n_y \\ \frac{\partial f_2}{\partial x}(p).nx + \frac{\partial f_2}{\partial y}(p).n_y \\ \frac{\partial f_3}{\partial x}(p).nx + \frac{\partial f_3}{\partial y}(p).n_y \end{pmatrix}$$

Comme dans le cas mono-dimensionnel, nous recherchons les fortes valeurs de la différentielle première. Cette différentielle étant cette fois-ci un vecteur de $\mathbb{R}^3$, nous calculons le carré de sa norme :

$$S(p,n) = \|Df(p).(n_x,n_y)\|^2 = E{n_x}^2 + 2Fn_xn_y + G{n_y}^2 \tag{5.8}$$

avec :

$$E = \sum_{i=1}^{3}\left(\frac{\partial f_i}{\partial x}\right)^2 \quad F = \sum_{i=1}^{3}\frac{\partial f_i}{\partial x}\frac{\partial f_i}{\partial y} \quad G = \sum_{i=1}^{3}\left(\frac{\partial f_i}{\partial y}\right)^2$$

L'équation 5.8 nous donne la norme de la différentielle première dans la direction $n$. Cette mesure nous indique s'il existe ou non un contour au point $p$ dans la direction $n$. L'existence d'un contour en $p$ indépendamment de $n$ s'évalue en cherchant le maximum de l'équation 5.8 pour $n$ appartenant à la boule unité (seule la direction de $n$ nous intéresse). Si nous posons $n = (cos(\theta), sin(\theta))$ les extrema de l'équation 5.8 sont atteints pour :

$$\theta_0 = \frac{1}{2}arctan\left(\frac{2F}{E-G}\right)$$

Le maximum correspondant est égal à :

$$\lambda(x,y) = \frac{E + G + \sqrt{(E-G)^2 + 4F^2}}{2} = S(p, (cos(\theta_0), sin(\theta_0)))$$

La valeur $\lambda$ peut être vue comme l'extension du concept de gradient aux images couleurs. En effet, on constate facilement que dans le cas mono-dimensionnel, $\lambda$ est égal au carré de la norme du gradient. Il est donc naturel de considérer les passages par zéro de la différentielle de la fonctions $S$, au point $p$, dans la direction $n$. Cette dernière valeur est donnée par l'expression suivante :

$$D_S(p).n = E_x(p){n_x}^3 + (2F_x(p) + E_y(p))n_x^2 n_y + (G_x(p) + 2F_y(p))n_x n_y^2 + G_y(p)n_y^3$$

où $E_x$, $E_y$,$F_x$, $F_y$ et $G_x$, $G_y$ représentent les dérivées partielles de $E$, $F$ et $G$ par rapport à $x$ et $y$.

La valeur $D_S(p).n$ représente l'extension de l'opérateur laplacien aux images multi-dimensionnelles.

L'ensemble des techniques que nous venons de présenter semblent avoir été découvertes en parallèle par Zenzo [Zen86] et Cumani [Cum89, Cum91, CGG91]. Zenzo approxime l'image en chaque pixel par trois fonctions linéaires (une pour chaque composante). Ces approximations lui permettent de calculer les dérivées partielles et d'en déduire la valeur de $\lambda$ et l'angle pour lequel le maximum est réalisé. Chapron [Cha92] utilise les filtres de Deriche [Der87] pour calculer les dérivées partielles de l'image. Il combine ensuite ces dérivées pour obtenir la fonction $\lambda$. Cumani [Cum89, Cum91, CGG91] donne un ensemble d'outils théoriques permettant d'exploiter les passages par zéros de la fonction $D_S(p)$.

#### 5.2.6.3 Les méthodes de division basées sur les multi-ensembles 1D

Comme nous l'avons vu, les méthodes de découpes ont d'abords été étudiées pour des multi-ensembles mono-dimensionnels (voir section 5.2.4). De nombreux auteurs ont tenté d'adapter ces méthodes aux espaces à plusieurs dimensions. Il s'agit alors de projeter un multi-ensemble 3D sur un axe (voir définition 7 chapitre 2). Le multi-ensemble 1D construit à partir de cette projection permet de définir un ensembles de seuils appartenant à l'axe. Partant de ces seuils, on définit une partition du multi-ensemble 3D en coupant celui-ci par les plans orthogonaux à l'axe et passant par les seuils. Deux approches ont étés proposées pour effectuer cette projection.

Ohlander [Ohl75] projette le multi-ensemble 3D sur son axe principal. Il utilise ensuite une approche pics-vallées (voir section 5.2.4.2) avec une approximation bi-modale pour découper le multi-ensemble 1D en deux modes. Les plans passant par les seuils définissent ensuite la partition du multi-ensemble 3D. L'axe principal d'un multi-ensemble est celui qui porte le plus d'information, il est donc logique de calculer les seuils par projection sur celui-ci. Toutefois, comme nous l'avons montré dans la section 3.6.3, une coupe orthogonale à l'axe principal ne conduit pas nécessairement au résultat optimal en terme d'erreur de partition.

Une autre méthode consiste à couper orthogonalement aux axes de coordonnées. Cette technique a été utilisée par Lim [LL90]. Celui-ci défini un ensemble de seuils indépendament sur les trois axes. Le partitionnement 3D est alors défini par l'ensemble des plans orthogonaux aux axes de coordonnées et passant par les seuils. Ohlander et al. [OPR78] calculent un histogramme suivant chaque axe de coordonnée, puis coupent le multi-ensemble perpendiculairement à l'axe dont l'histogramme associé possède un seuil très caractéristique. Ohlander fournit sept critères permettant de mesurer la qualité d'un seuil. Le critère le plus important étant la différence de hauteur entre le seuil et les pics avoisinants. Il crée ainsi un partitionnement de l'image et applique récursivement son algorithme sur les régions créées jusqu'à ce qu'aucun histogramme ne présente de seuil caractéristique. Une approche légèrement différente à été adoptée par Celenk [Cel90]. Celui-ci coupe le multi-ensemble 3D successivement suivant chacun des trois axes en recalculant les projections après la première découpe. La partition d'un multi-ensemble $(C, f)$ peut, par exemple se dérouler de la façon suivante :

1. Sélectionner l'axe de coordonnée $X_1$ présentant les deux meilleurs seuils. En déduire deux modes sur l'histogramme 1D. La coupure de $(C, f)$ par deux plans orthogonaux à $X_1$ définie une partition de $(C, f)$ en $(C_1, f)$ et $(C_2, f)$.
2. Calculer les projections de $(C_1, f)$ sur $X_2$ et $X_3$. En déduire des valeurs de seuil et un partitionnement de $(C_1, f)$. Le multi-ensemble $(C_1, f)$ est alors redéfini comme le produit cartésien de l'intervalle défini à l'étape 1 et des deux meilleurs modes sur les axes $X_2$ et $X_3$. Faire de même avec $(C_2, f)$.
3. Soit $(C_3, f) = (C, f) - (C_1, f) - (C_2, f)$. Si une des projections de $(C_3, f)$ a un aspect bimodal alors revenir au point 1.

Beveridge [BGK+89] effectue une segmentation sur chaque composante, l'image segmentée est alors définie par la somme des contours trouvés lors des trois segmentations. La sur-segmentation produite par ce procédé est supprimée par une étape de fusion.

La méthode de Balasubramanian [BBA94] décrite dans la section 3.5 développe l'idée de recalcul d'histogramme introduite par Celenk. Cette méthode est basée sur la minimisation de l'erreur de la partition et recalcule les projections après chaque découpe.

L'ensemble des méthodes descendantes vues dans le chapitre 3 peuvent également être vues comme des adaptations de la méthode de Wong [WWP89] (voir chapite 2) aux multi-ensembles 3D. Nous retrouvons la distinction entre les méthodes coupant perpendiculairement à l'axe principal [WWP88, WZ91, Wu92] et les méthodes comme celles décrites dans la section 3.7 qui découpent orthogonalement aux axes de coordonnées. Ces méthodes nécessitent de connaitre a priori le nombre de multi-ensembles devant former la partition. Toutefois les méthodes de divisions récursives [WWP88, WZ91, Wu92, BB95] permettent de faire évoluer le nombre de multi-ensembles formant la partition. On peut donc arrêter le partitionnement lorsqu'un critère comme $\beta$ (voir section 5.2.4.2) indique que l'on a atteint le nombre optimal de multi-ensembles.

#### 5.2.6.4 Les méthodes de division basées sur les multi-ensembles 3D

Les méthodes de division ne faisant pas appel à des techniques de projection sur des axes sont peu nombreuses et généralement basées sur des techniques issues de l'analyse de données. La plus populaire de ces méthodes est sans doute la méthode des nuées dynamiques mentionée dans la section 3.5. Cette méthode nécessite de connaître le nombre $K$ de multi-ensembles formant la partition. Elle peut se décomposer de la façon suivante :

1. Choisir au hasard $K$ éléments $(v_1, \ldots, v_K)$ du multi-ensemble $(C, f)$.
2. Affecter chaque élément de $(C, f)$ à l'élément $v_i$ le plus proche. On crée ainsi une partition en $K$ multi-ensembles $((C_1, f), \ldots, (C_K, f))$. Cette étape revient à créer la fonction $\boldsymbol{Q}$ définie dans la section 3.1 à partir des éléments $(v_1, \ldots, v_K)$.
3. Calculer la moyenne $\mu(C_i)$ de chaque multi-ensemble $(C_i, f)$.
4. Définir l'ensemble $(w_1, \ldots, w_K)$ tel que $w_i$ est l'élément de $(C, f)$ le plus proche de $\mu(C_i)$.
5. S'il existe un élément $v_i$ tel que $v_i \neq w_i$, pendre comme nouvel ensemble de départ $(w_1, \ldots, w_K)$ et revenir au point 1.

Cette méthode fut appliquée pour la première fois aux problèmes d'analyse d'image par Diday [Did71, DLPT82]. Comme nous l'avons vu dans la section 3.1, cette méthode converge vers le minimum local le plus proche du vecteur initial $(v_1, \ldots, v_K)$. L'égalité entre les éléments $v_i$ et $w_i$ pouvant être longue à obtenir, on remplace généralement le test d'arrêt par :

$$\max_{i \in \{1, \ldots, K\}} \|v_i - w_i\| < \epsilon$$

Le paramètre $\epsilon$ permettant de contrôler le rapport temps/qualité de l'algorithme.

Les nuées dynamiques ont été utilisées par Coleman [CA79]. Celui-ci associe à chaque pixel un ensemble d'attributs tel que l'intensité, la couleur, la texture, etc. Il applique ensuite une méthode d'analyse en composantes principales pour supprimer les attributs redondants. Il applique ensuite les nuées dynamiques avec un nombre de multi-ensembles $K$ croissant. Le paramètre $\beta$ vu dans la section 5.2.4.2 lui permet de déterminer la valeur optimale de $K$.

Les méthodes de nuées dynamiques convergeant vers un minimum local, il peut être utile d'utiliser une heuristique permettant de trouver un vecteur initial $(v_1, \ldots, v_K)$ proche de la solution optimale. L'ensemble des méthodes de quantification vues dans le chapitre 3 permettent d'obtenir un ensemble $(v_1, \ldots, v_K)$ de couleurs représentatives. Ces heuristiques ne permettent pas d'obtenir le minimum de l'erreur de la partition, qui peut d'ailleurs être atteint pour plusieurs ensembles de couleurs représentatives. Elle permettent toutefois d'obtenir une solution initiale proche d'un "bon" minimum. Cette solution peut ensuite être améliorée par les nuées dynamiques.

Nous avons vu dans la section 3.8 une méthode de partition basée sur une approche découpe-fusion du multi-ensemble 3D. Cette méthode permet d'obtenir la partition finale très rapidement (entre deux et six secondes). De plus les expériences résumées dans la section 3.8.3 montrent que la solution obtenue est très proche de la solution optimale. Il est donc généralement inutile d'affiner la solution en itérant la méthode des nuées dynamiques.

D'autres méthodes de découpe telle que la classification ascendante hiérarchique [DLPT82] ou la méthode des K-plus proches voisins [PS88] sont plus spécifiquement dédiées à des multi-ensembles de taille plus réduite que les multi-ensembles associés aux images couleurs. Ces méthodes sont en conséquence peu utilisées en analyse d'image.

### 5.2.7 Conclusion

Nous avons vu dans ce chapitre un ensemble de méthodes de segmentation et les extensions de ces méthodes au traitement des images couleurs. Il ressort de cette étude qu'il est impossible de privilégier une approche au détriment des autres. En effet, les méthodes de détection de contours utilisant des critères locaux fourniront de bons résultats lorsque les frontières entre les régions sont franches. En revanche, ce type de méthodes donne des résultats beaucoup moins intéressants lorsque les transitions entre les régions deviennent plus floues. Inversement, l'approche régions fournit de bons résultats lorsque l'image est composée de régions de caractéristiques assez différentes. L'approche régions pourra donc séparer deux régions de paramètres différents même si la frontière entre celles-ci est assez floue. En revanche, cette approche peut découper artificiellement des régions si celles-ci présentent un aspect dégradé.

Une solution a ce problème se trouve sans doute dans les méthodes faisant collaborer l'approche régions et l'approche contours. On peut par exemple imaginer un algorithme de découpe-fusion utilisant un critère de fusion similaire à celui de Beveridge [BGK+89] (voir section 5.2.3.2), donc faisant intervenir des paramètres des régions et de leurs frontières. L'algorithme de découpe de cette méthode pourrait alterner des méthodes de division (voir section 5.2.4) et des méthodes de détection de contour lorsque les régions présentent des détails significatifs tout en étant trop homogènes pour une approche basée sur le multi-ensemble associé à la région. Nous verrons dans le chapitre 6 une structure de données suffisamment souple pour permettre la collaboration entre différentes méthodes de segmentation.

## 5.3 Les structures de données habituellement utilisées en segmentation

### 5.3.1 Structures de base

Une des première structure de données utilisée en segmentation est le **tableau de label** [FV92, Suk83, Nic95]. Cette structure est définie par un tableau d'entiers appelés **labels** tels que tous les pixels appartenant à une même région de l'image partagent le même label. Inversement, deux pixels appartenant à des régions différentes ont obligatoirement des labels différents. Cette structure est bien adaptée aux opérateurs s'appliquant à l'image entière tels que les opérateurs morphologiques. Toutefois, cette structure de données est peu adaptée aux algorithmes de découpes récursives qui doivent changer les labels des régions nouvellement créées à chaque itération. De plus les frontières entre régions sont implicitements définies comme les côtés des pixels de label different. Les paramètres attachés aux frontières des régions (voir sous-section 5.2.3.2) sont donc coûteux à calculer puisqu'ils nécessitent le recalcul de la frontière des régions.

Les différentes régions composant une image peuvent également être définies à l'aide d'une **transformation par axe médian** (MAT en abrégé) [Lev70, RP68, RK82]. À chaque pixel $p$, nous pouvons associer l'ensemble des carrés de taille paire centrés en $P$. Ces carrés ont leurs cotés parallèles aux axes et sont inclus dans la région contenant $P$. Soit $S_P$ le carré de taille maximale de cet ensemble. Le carré $S_P$ est appelé carré maximum si il n'existe pas de point $Q$ dans $S_P$ tel que $S_P$ est contenu dans $S_Q$. Chaque région de l'image peut alors être représentée par une union de blocs maximum (voir Figure 5.15).

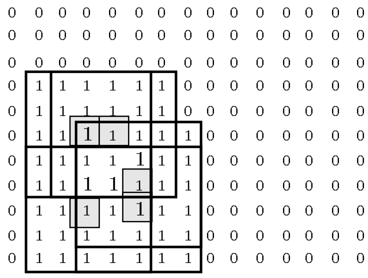


Figure 5.15: *Définition de la région 1 par des blocs maximaux.*

Pour une région $R$ représentée par un ensemble de carrés $\{S_1, \ldots, S_n\}$, la frontière de $R$ est définie comme l'ensemble des parties de côtés de carrés $\{b_1, \ldots, b_n\}$ tels que chaque $b_i$ appartient à un et un seul des carrés de l'ensemble $\{S_1, \ldots, S_n\}$. Les recouvrements à partir de carrés maximum doivent être mis à jour après chaque opération de découpe ou de fusion. On peut remarquer que si nous ne tentons pas de préserver l'aspect maximal du recouvrement de chaque région, le recouvrement de deux régions adjacentes peut être simplement défini comme l'union des carrés recouvrant chaque région. Les transformations par axe médian permettent donc des calculs efficaces sur une partition fixe de l'image et peuvent être utilisés pour réaliser des opérations de fusions. Cependant cette structure de données est peu adaptée aux algorithmes de découpes récursives ou aux algorithmes de type découpe-fusion qui doivent réaliser un nombre important de découpes.

Les tableaux de labels ou les MAT ne fournissent pas une représentation explicite des frontières des régions. Donc un traitement s'effectuant sur la frontière d'une région devra reconstruire celle-ci. Il est possible de coder explicitement les contours des régions, en stockant par exemple la séquence de pixels d'une région adjacents à au moins un pixel d'une autre région. Un tel contour peut être repésenté par un de ses points et une séquence de déplacements [Fre61]. (voir Figure 5.16).

Un important défaut des frontières définies par une séquence de pixels provient de leur inconsistence topologique (voir chapitre 4). Ce problème peut être résolu en utilisant **les frontières inter-pixels** (voir Figure 5.17).

Cette approche à été introduite par Brice et Fennema [BF70] lors de la définition de leur algorithme phagocyte (voir sous-section 5.2.3.3). Plusieurs topologies discrètes permettant d'étudier cette représentation ont été proposées [KKM90, Kov89, Fio95]. La frontière d'une région est définie comme un chemins fermé 4-connexe de points demi-entiers [Dom92]. Ces points appartiennent au plan $P_{\frac{1}{2}}$ (voir chapitre 4). On peu montrer qu'un chemin fermé de $P_{\frac{1}{2}}$ vérifie le Théorème de Jordan (voir chapitre 4). De plus tout chemin fermé peut être

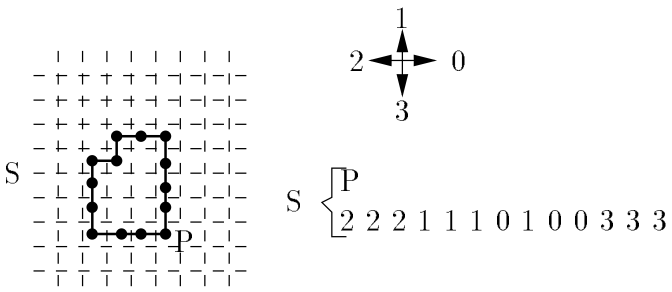



Figure 5.16: *Définition d'une région avec un chemin 4-connexe dans le plan entier.*

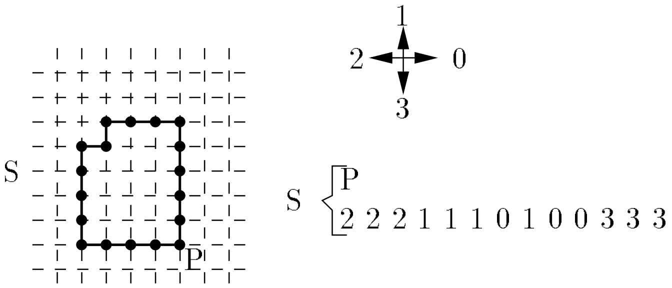



Figure 5.17: *Même région que dans la Figure 5.16 mais définie par des chemins inter-pixels.*

décomposé en un ensemble de **segments** définis comme les chemins de longeur maximum séparant deux régions. Les extrémités des segments sont appelés des **noeuds** [BD96a]. Il est possible de coder la géométrie des frontières avec un tableau $\boldsymbol{B}$ de taille $(M+1)\times(N+1)$ où $M \times N$ est la taille de l'image. Chaque entrée dans le tableau code le voisinage d'un point frontière. Sur l'exemple de la figure 5.17 nous avons $B[P] = \{1, 2\}$. Un segment peut donc être codé en stockant uniquement un de ses noeuds et le déplacement initial. Les autres points du segment sont retrouvés en lisant les déplacements successifs dans le tableau $\boldsymbol{B}$. La représentation inter-pixels permet des calculs rapides des paramètres des frontières des régions où des intersections de régions. Toutefois, cette représentation ne fournie qu'une définition implicite des régions et l'extraction de paramètres de celle-ci peut être coûteux. Par exemple, calculer l'ensemble des régions contenues dans une région $R$ nécessite de parcourir l'ensemble des pixels de $R$.

### 5.3.2 Structures hiérarchiques

Les structures de données décrites dans la sous-section précédente peuvent souvent être combinées pour produire de nouvelles structures. Ces structures permettent de concevoir des algorithmes manipulant des concepts plus complexes. En particulier il peut être intéressant de traiter l'image à différents niveaux de résolution. Les structures de données permettant cette multi-résolution sont appelées des **structures hiérarchiques**. Partant d'un niveau de résolution assez bas, la partition de l'image est affinée de niveau en niveau jusqu'à atteindre le niveau de résolution de l'image. Ces structures sont surtout utilisées par les algorithmes de découpes récursives ou de découpe-fusion (voir section 5). De plus, lorsqu'elles sont implémentées sur des machines parallèles, ces méthodes permettent de traiter simultanément plusieurs niveaux de résolution.

Les pyramides de matrices [Bro82, TK80, JM92] (voir Figure 5.18) ou M-pyramides permettent d'obtenir une description multi-résolution de l'image. Une M-pyramide peut être définie comme suit :

**Définition 24** *Une M-pyramide est une séquence* $(M(L), M(L-1), \ldots, M(0))$ *de tableaux tels que* $M(L)$ *représente l'image originale. M(L-1) est une version de* $M(L)$ *de résolution deux fois moindre et ainsi de suite jusqu'à* $M(0)$ *qui n'est constitué que d'un pixel.*

*Le passage de* $M(L)$ *à* $M(L-1)$ *est généralement effectué en affectant à chaque pixel de* $M(L-1)$ *la moyenne de quatre pixels de* $M(L)$*.*

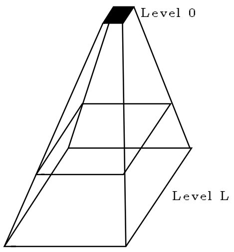



Figure 5.18: *M-pyramide associée à une image originale de taille* $2^{\cdot} \star 2^L$*.*

Si l'algorithme de segmentation doit accéder fréquemment à différents niveaux de résolution, il est plus facile de définir la pyramide en terme d'arbre plutôt qu'en terme de tableaux. Les arbres-pyramides notés également T-pyramides correspondent à la transposition en termes d'arbres des données contenues dans une M-pyramide. Leur définition peut être formulée de la façon suivante :

**Définition 25** *Une T-pyramide est un ensemble de p-noeuds* $(k, i, j)$ *avec* $0 \leq k \leq L$, $0 \leq i \leq 2^L - 1$, *et* $0 \leq j \leq 2^L - 1$. *Chaque p-noeud correspond à un pixel de la M-pyramide. Le noeud* $(k, i, j)$ *correspond au pixel* $(i, j)$ *de l'image* $M(k)$ *dans la M-pyramide* $(M(L), M(L-1), \ldots, M(0))$. *Le père d'un p-noeud* $(k, i, j)$ *est égal à* $(k-1, [i/2], [j/2])$ *où* $[x]$ *représente le plus grand entier inférieur ou égal à* $x$.

Chaque feuille de la T-pyramide représente un pixel de l'image originale. Si nous associons à chaque feuille l'indice de la région qui contient le pixel associé, nous pouvons avoir un noeud dont tous les fils possèdent le même indice. Ces fils peuvent donc être représentés par leur père sans perte d'information. Donc la T-pyramide peut être élaguée de façon à ce qu'aucun noeud ne possède de fils partageant un même indice. La T-pyramide n'est alors plus équilibrée et l'arbre correspondant est appelé un quadtree [RK82, Sam80, DRH80]. Les rotations de 90 degrés et les opérations bouléennes telles que le complément, l'union, l'intersection peuvent être effectuées avec des quadtrees en des temps proportionnels au nombre de noeuds du quadtree.

Les M-pyramides, T-pyramides et quadtrees, sont principalements dévolus aux algorithmes de découpes récursives (voir section 5). Ils permettent donc de calculer facilement les fils ou les ancêtres d'une région définie à un niveau de récursion. De plus les paramètres d'une région à un niveau donné peuvent être déduis des paramètres du niveau inférieur. Cela ne nécessite alors qu'un seul parcours de l'ensemble des pixels de la région. Toutefois, le calcul des frontières d'une région ou la détermination de l'ensemble de ses voisins nécessite des algorithmes coûteux. Hunter et Steiglitz [HS79] ont conçu une structure dérivée des quadtree appelée quadtree-lié qui permet de calculer efficacement l'ensemble des feuilles adjacentes à un bloc donné.

### 5.3.3 Le graphe d'adjacence de régions

Comme nous l'avons vu précédement, les structures hiérarchiques sont principalement dévolues aux algorithmes de découpes récursives. Ils ne permettent donc pas une implémentation efficace des algorithmes de fusion (voir chapite 5). De fait, le calcul de l'adjacence des régions avec une structure arborescente implique l'utilisation d'algorithmes complexes. Les algorithmes de découpe-fusion utilisent donc une structure hiérarchique pour les étapes de découpe et de fusion-restreinte (voir sous-section 5.2.5) puis créent une nouvelle structure pour effectuer les fusions non restreintes. L'incompatibilité entre ces deux structures est la raison pour laquelle les algorithmes de découpe-fusion ne peuvent alterner des étapes de découpe avec des étapes de fusion non restreintes.

La structure de données habituellement utilisée pour effectuer les fusions non restreintes est le **Graphe d'Adjacence de Régions** noté RAG. Chaque noeud de ce graphe représente une région de l'image segmentée et deux régions sont reliées par une arête si elles sont adjacentes (voir figure 5.19).

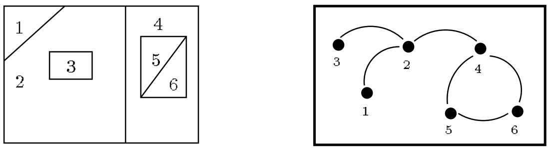



Figure 5.19: *Graphe d'adjacence d'un ensemble de régions.*

La fusions de deux régions $r_1$ et $r_2$ nécessite la mise à jour des noeuds correspondant dans le RAG en contractant l'arête reliant les deux noeuds. Les deux noeuds $n_1$ et $n_2$ sont fusionnés est le noeud résultant est relié aux noeuds qui étaient adjacents à $n_1$ ou $n_2$. Les arêtes multiples (arêtes reliant deux fois les mêmes noeuds) sont suprimées (voir Figure 5.20).


Figure 5.20: *Fusions de deux régions dans le graphe d'adjacence.*

Le graphe d'adjacence de régions peut être implémenté grâce à des tas [M.w84], des listes d'adjacence ou des demi-matrices de bouléens. Si nous implémentons le RAG au moyen d'un tableau de bouléens, la fusion de deux noeuds $i$ et $j$ nécessite d'invalider la ligne et la colonne $i$ et de mettre à jour la ligne et la colonne $j$.

Le graphe d'adjacence peut être utilisé en conjonction avec une tableau de description de régions. Chaque entrée de ce tableau contient des paramètres des régions. Ces paramètres sont mis à jour lors des fusions successives et sont utilisés pour affecter un poids à chaque arête du graphe. Ce tableau peut contenir, par exemple la couleur moyenne de chaque région. Dans ce cas l'algorithme de fusion fusionnera deux régions $r_1$ et $r_2$ si et seulement si :

- Les régions $r_1$ et $r_2$ sont adjacentes dans le graphe.
- La distance entre les deux couleurs moyennes des régions $r_1$ et $r_2$ est minimale.

Les paramètres tels que la moyenne ou la variance peuvent être efficacement mis à jour durant les fusions successives.

Le graphe d'adjacence de régions permet de coder efficacement les relations d'adjacence entre régions. De plus les relations d'adjacence et la plupart des paramètres de régions peuvent être mis à jour efficacement durant les fusions successives. Toutefois la découpe d'une région $r$ en un ensemble de régions $\{r_1, \ldots, r_n\}$ implique :

- De supprimer la région $r$ dans le RAG.
- De parcourir les régions $\{r_1, \ldots, r_n\}$ afin d'insérer les noeuds et les arêtes correspondant aux adjacences entre les régions.

La mise à jour de RAG après une opération de découpe implique trop de calculs. Cette structure est donc réduite aux opérations de fusions.

# Chapitre 6

# Représentation d'images segmentées par Frontière inter-pixels et cartes planaires

Nous avons vu dans le chapitre précédent que les structures de données utilisées en segmentation doivent gérer deux problèmes : l'aspect géométrique qui décrit la forme des régions et l'aspect topologique qui décrit les relations de voisinage ou d'inclusion des régions. Les modèles vus précédement sont essentiellement dédiés à l'un ou l'autre de ces problèmes. Le modèle que nous allons présenter dans ce chapitre est basé sur un niveau géométrique et un niveau topologique coopérant pour donner une représentation unifiée de l'image segmentée. Les bases de ce modèle ont été introduites par différents travaux dûs à J.P Braquelaire, J.P. Domenger et P. Guitton [BP91, Dom92, BD96a, BD96b] dans une perspective d'édition d'images. Après avoir décrit les fondements du modèle, nous décrirons les fonctionnalités que nous avons ajoutées afin de l'étendre à la problématique de la segmentation.

## 6.1 Définition du modèle

### 6.1.1 Le niveau géométrique

Nous avons vu dans le chapitre 4 qu'une région peut être indifféremment définie par l'ensemble de ses pixels ou par sa frontière. Si l'image comporte plusieurs régions, il est nécessaire de décomposer leur frontière en **noeuds imposés** et en **segments**. Intuitivement, les segments représentent les intersections entre les régions tandis que les noeuds sont les points de jonction entre plusieurs régions (voir Figure 6.1).

**Définition 26** *Un* **noeud imposé** *est un point de* $P_{\frac{1}{2}}$ *possédant au moins trois demi-voisins dans des régions distinctes.*

**Définition 27** *Soit* **R** *l'ensemble des points-frontières de l'image segmentée. Un point-frontière n sera appelé un* **noeud** *s'il vérifie une des conditions suivantes :*

- *$n$ est un noeud imposé.*
- *$n$ est un point-frontière. Tout point-frontière $p \neq n$ connecté à $n$ dans* **R** *(voir definition 13) n'est pas un noeud imposé.*

En d'autres termes, un noeud est soit un noeud imposé, soit un point-frontière situé sur un contour qui ne possède pas de noeud imposé.

**Définition 28** *Un chemin 4-connexe $C = (P_1, \ldots, P_n)$ de $P_{\frac{1}{2}}$ est appelé* **segment** *si :*

- *Tous les points de $C$ sont des point- frontière.*
- *Le chemin $C$ comprend au plus deux noeuds situés à ses extrémités.*
- *Tous les points du chemin excepté les extrémités sont distincts.*

$$\forall (i,j) \in \{1, \ldots, n\}^2 \; i < j \; et \; P_i = P_j \Rightarrow i = 1 \; et \; j = n$$

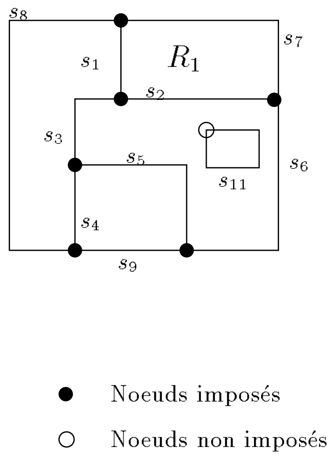



Figure 6.1: *Noeuds imposés et segments d'une image.*

Les segments sont habituellement utilisés pour définir la frontière d'une image. On dira qu'une frontière est la concaténation d'un ensemble de segments. Les noeuds peuvent être vus comme des aiguillages permettant de passer d'un segment à l'autre. Dans l'exemple de la figure 6.1, la frontière de la région $R_1$ est définie comme l'ensemble des points-frontières appartenant aux segments $s_1$, $s_2$ et $s_7$. Les trois noeuds situés aux extrémités de ces segments permettent de nous aiguiller de $s_1$ à $s_2$ puis de $s_2$ à $s_7$. Les noeuds qui ne sont pas imposés peuvent être vus comme des raccords entre deux segments Ainsi une frontière définie à partir d'un seul segment comportera obligatoirement un noeud symbolisant le lien entre le dernier et le premier point du segment (voir par exemple le segment $s_{11}$ sur la figure 6.1)

**Définition 29** *Un ensemble de segments est dit* **complet**, *si tout point-frontière appartient à au moins un segment.*

Un ensemble complet de segments permet donc de représenter l'ensemble des frontières d'une partition.

Le niveau géométrique peut être défini comme un couple $(\boldsymbol{\mathcal{N}}, \boldsymbol{\mathcal{S}})$ où $\boldsymbol{\mathcal{S}}$ représente un ensemble complet de segments et $\boldsymbol{\mathcal{N}}$ l'ensemble de noeuds. Chaque segment contient ses deux extrémités ainsi que le premier et dernier déplacement du segment (notés respectivement $\boldsymbol{Fm}(\mathrm{s})$ et $\boldsymbol{Lm}(\mathrm{s})$). Les autres points du segment sont retrouvés à l'aide du tableau $\boldsymbol{B}$ (voir section 5.3.1 et Figure 5.17). Coder les deux extrémités plutôt qu'une seule permet de parcourir le segment dans un sens ou dans l'autre. De son côté, un noeud comprend ses coordonnées ainsi que l'identificateur d'un des segments dont une extrémité est égale au noeud (ceci sera explicité dans la section 6.1.3).

### 6.1.2 Le niveau topologique

Le niveau topologique est basé sur les cartes planaires [Tut63] :

**Définition 30** *Une carte planaire est la décomposition du plan euclidien en un ensemble fini de points $V$ et un ensemble fini $E$ de courbes ouvertes de Jordan. Les extrémités de chaque courbe appartiennent à $V$. Les ensembles $E$ et $V$ définissent un ensemble de régions simplement connexes dont les frontières sont incluses dans l'union des éléments de $E$ et $V$ (voir Figure 6.2).*

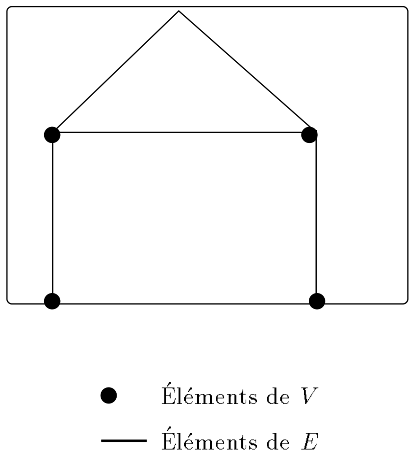


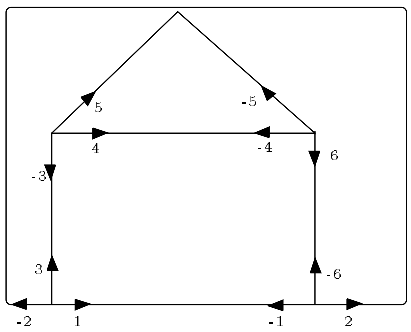

Figure 6.2: *Un exemple de carte planaire.*

Les cartes planaires peuvent être efficacement représentées par des **cartes combinatoires** [Cor75]. Une carte combinatoire est un couple de permutations $(\sigma, \alpha)$ définies sur un ensemble d'étiquettes appelées **brins**. L'ensemble des brins est noté $\boldsymbol{\mathcal{B}}$. La permutation $\alpha$ est une involution sans point fixe. Chaque brin représente une demi-arête de la carte planaire. Chaque cycle de $\alpha$ $(b, \alpha(b))$ représente une arête de la carte. Chaque cycle de $\sigma$

est la séquence de brins rencontrés en tournant dans le sens positif (inverse de celui des aiguilles d'une montre) autour d'un sommet (voir Figure 6.3). Les cycles de $\sigma$ représentent donc les sommets de la carte planaire. Nous confondrons désormais ces deux notions et le terme "sommet" représentera indifféremment le sommet de la carte planaire et le cycle de $\sigma$ associé. Une solution bien adaptée au codage des brins consiste à représenter ceux-ci par des entiers positifs et négatifs. On définit alors la permutation $\alpha$ par :

$$\forall b \in \boldsymbol{\mathcal{B}} \;\; \alpha(b) = -b$$

Ces conventions ont été utilisées dans l'exemple de la figure 6.3.




$\sigma$

$\sigma = (3,-2,1)(-6,-1,2)(-5,-4,6)(5,-3,4)$

$\forall b \in \{1,\ldots,6\} \;\; \alpha(b) = -b$ et $\alpha(-b) = b$

Figure 6.3: *Carte combinatoire associé à une carte planaire.*

Le cycle d'un brin $b$ par une permutation $\pi$ est noté $\pi^*(b)$. Par exemple, dans l'exemple 6.3 le cycle du brin $-4$, noté $\sigma^*(-4)$ est égal à $(-4,6,-5)$. On a donc $\sigma^*(-4) = \sigma^*(6) = \sigma^*(-5)$.

**Définition 31** *Le* **cardinal** *d'un sommet $\sigma^*(b)$ noté $|\sigma^*(b)|$ est défini comme le nombre de brins composant le cycle $\sigma^*(b)$.*

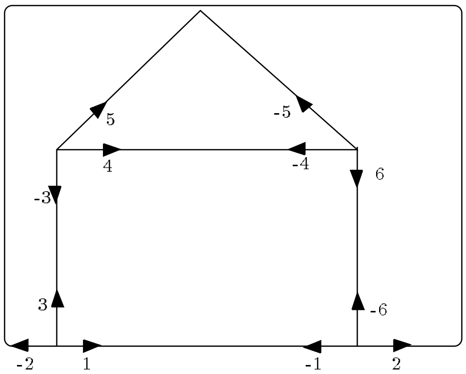
Les **faces** d'une carte planaire sont définies par l'ensemble des arêtes qui les délimitent. Dans le cas d'une carte combinatoire une face est déterminée si nous connaissons un brin de chaque arête. La permutation $\alpha$ nous permet ensuite de déterminer l'autre brin de l'arête. Les faces d'une carte combinatoire correspondent aux cycles de la permutation $\varphi = \sigma \circ \alpha$. Nous confondrons désormais ces deux notions et le terme "face" désignera indiférement les faces de la carte planaire et les cycles de la permutation $\varphi$.

**Remarque 7** *Le choix $\alpha(b) = -b$ induit la relation suivante entre $\varphi$ et $\sigma$ :*

$$\forall b \in \boldsymbol{\mathcal{B}} \;\; \varphi(b) = \sigma(-b)$$

Dans l'exemple de la figure 6.4, le toit est défini par les arêtes $(4,-4)$ et $(5,-5)$. Le cycle de $\varphi$ correspondant au toit est égal à $\varphi^*(-4) = \varphi^*(5) = (-4,5)$. Les cycles $\varphi^*(4)$ et $\varphi^*(-5)$ correspondent au mur et à l'extérieur de la maison.

**Définition 32** *Le* **cardinal d'un sommet $\sigma^*(b)$ relativement à une face $\varphi^*(b')$** *est noté $|\sigma^*(b)|_{\varphi^*(b')}$ et est défini comme le nombre de brins appartenant simultanément aux deux cycles.*





$\sigma = (3, -2, 1)(-6, -1, 2)(-5, -4, 6)(5, -3, 4)$

$\varphi = (-4, 5)(4, 6, -1, 3)(1, 2)(-5, -3, -2, -6)$

$\varphi(-4) = \sigma \circ \alpha(-4) = \sigma(4) = 5$

Figure 6.4: *Ajout de la permutation $\varphi$.*

Dans l'exemple de la figure 6.4 nous avons :

$$|\sigma^*(-4)|_{\varphi^*(5)} = |(-4, 6, -5)|_{(5,-4)} = 1$$

Afin de pouvoir désigner les faces d'une carte combinatoire nous associons une étiquette à chaque brin. Nous créons donc **une fonction d'étiquetage de face** $\lambda$ constante sur chaque cycle de $\varphi$.

**Définition 33** *Une fonction $\lambda$ définie sur un ensemble de brins $\mathcal{B}$, sera appelée une* **fonction d'étiquetage de face** *si et seulement si elle vérifie la propriétée suivante :*

$$\forall b, b' \in \mathcal{B} \quad \lambda(b) = \lambda(b') \Leftrightarrow \varphi^*(b) = \varphi^*(b')$$

**Définition 34** *Étant donné une fonction d'étiquetage de faces $\lambda$ , l'image de $\mathcal{B}$ par $\lambda$ notée $\mathcal{F}$ est appelé l'ensemble des* **etiquettes de faces**

**Remarque 8** *Si l'on définit la relation d'équivalence sur $\mathcal{B}$ :*

$$b \sim b' \Leftrightarrow \varphi^*(b) = \varphi^*(b')$$

*l'ensemble des cycles de $\mathcal{B}$ peuvent se voir comme des classes d'équivalence modulo cette relation. La fonction d'étiquetage de face $\lambda$ peut alors se voir comme une projection canonique de $\mathcal{B}$ sur l'ensemble des classes d'équivalences.*

Afin d'accéder rapidement au cycle d'une face à partir de son étiquette, nous définissons également une fonction **d'étiquetage inverse** $\lambda^{-1}$ :

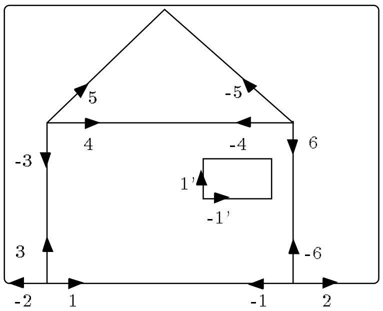
**Définition 35** *Soit $\lambda$ une fonction d'étiquetage de face, on dira que $\lambda^{-1}$ est la fonction* **d'étiquetage inverse** *associée à $\lambda$ si et seulement si :*

$$\forall f \in \mathcal{F} \; b = \lambda^{-1}(f) \Rightarrow \forall b' \in \varphi^*(b) \; \lambda(b') = f$$

La fonction $\lambda^{-1}$ associe à chaque étiquette de face un brin du cycle correspondant. Il est important de noter que la fonction $\lambda^{-1}$ n'est pas l'inverse de la fonction $\lambda$ qui n'est pas bijective. Plus précisément, nous avons les relations suivantes :

$$\left\{\begin{array}{lll} \forall\, f \;\in \boldsymbol{\mathcal{F}}\; \lambda \circ \lambda^{-1}(f) & = & f \\ \text{et} & & \\ \forall\, b \;\in \boldsymbol{\mathcal{B}}\; \lambda^{-1} \circ \lambda(b) & \in & \varphi^{*}(d) \end{array}\right.$$

**Remarque 9** *Le cycle d'une face d'étiquette $f$ est égal à $\varphi^{*}(\lambda^{-1}(f))$*




$\phi = (-4, 5)(4, 6, -1, 3)(1, 2)(-5, -3, -2, -6)$
$\phi' = (1')(-1')$

$f^{\infty}_{\phi} = \lambda(1)$
$f^{\infty}_{\phi'} = \lambda(-1')$

$\textbf{mère}(\lambda(-1')) = \lambda(3)$
$\textbf{filles}(\lambda(3)) = \{\lambda(-1')\}$

$\sigma$

Figure 6.5: *Introduction des faces incluses.*

Si nous utilisons une orientation positive pour la permutation $\sigma$ tous les cycles de $\varphi$, excepté un sont orientés dans le sens des aiguilles d'une montre [Cor75]. Les faces orientées dans le sens des aiguilles d'une montre sont appelées des **faces finies**. L'ensemble des faces finies d'une carte $(\varphi, \alpha)$ est noté $\boldsymbol{\mathcal{F}i(\varphi)}$. La face orientée dans le sens positif est appelée la **face infinie** et est notée $f^{\infty}_{\varphi}$. La face infinie de l'exemple 6.4 correspond au cycle $\varphi^{*}(1) = (1, 2)$. Un inconvénient du modèle des cartes combinatoires est qu'il ne permet pas de gérer les faces incluses. En effet pour une carte $(\varphi, \alpha)$, si nous insérons une arête $(b, -b)$ qui ne coupe aucune arête de la carte $(\varphi, \alpha)$ nous créons une nouvelle carte $(\varphi', \alpha')$ (voir Figure 6.5). Le modèle des cartes planaires ne nous permet pas de connaitre dans quelle face de la carte $(\varphi, \alpha)$ se situe la nouvelle carte $(\varphi', \alpha')$. Par exemple dans la figure 6.5, les substitutions $\varphi, \sigma, \alpha$ et les fonctions $\lambda, \lambda^{-1}$ ne nous permettent pas de décider si la fenêtre est située sur un mur, sur le toit ou à l'extérieur de la maison. La position d'une carte par rapport à une autre réclame des informations géométriques. Ce genre d'information ne peut donc être ajouté au modèle qu'en liant la géométrie et la topologie.

### 6.1.3 Lien entre la géométrie et la topologie

Nous avons vu dans la section 6.1.1 que l'ensemble des points-frontières d'une image segmentée pouvait être structuré en noeuds et segments. Les segments décrivant la frontière entre les régions et les noeuds étant définis comme les extrémités des segments. D'un autre coté, une carte planaire est constituée d'un ensemble d'arêtes reliées par des sommets. L'idée fondamentale du modèle consiste à faire correspondre :

- Les noeuds et les sommets.

- Les segments et les arêtes.
- Les régions et les faces.

Partant d'une partition d'une image, on construit sa représentation géométrique en créant $p$ noeuds et $q$ segments. On crée alors un sommet pour chaque noeud et une arête pour chaque segment. Une arête relie deux sommets si et seulement si les noeuds correspondants sont les extrémités du segment correspondant à l'arête. On peut montrer [Dom92] que les graphes ainsi constitués sont bien des cartes planaires, donc des cartes combinatoires. La figure 6.6 représente l'ensemble des cartes combinatoires associées à une partition de l'image.

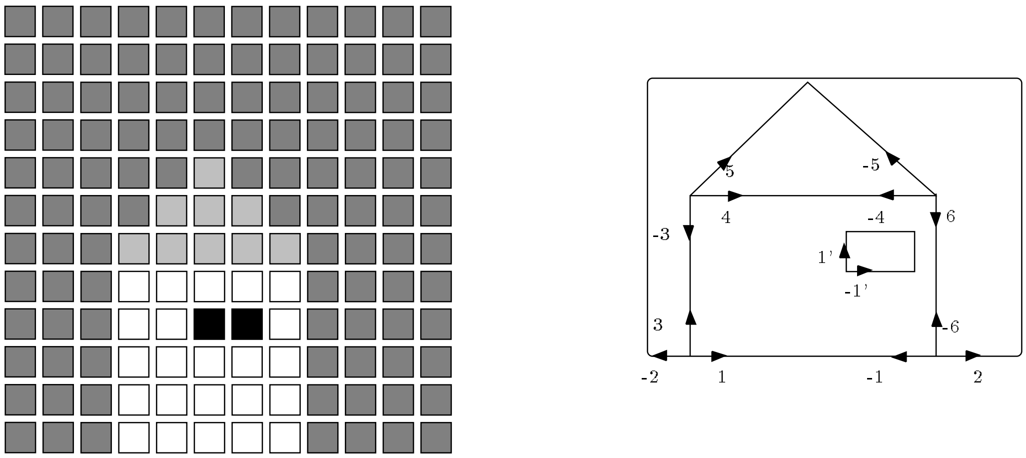



Figure 6.6: *Cartes planaires associées à une partition de l'image.*

L'association entre les brins et les segments est faite en stockant l'ensemble des segments dans un tableau indexé par les brins positifs. Le segment d'indice 4 correspond donc à l'arête $(4, -4)$ de la carte planaire. Inversement, on stocke dans chaque segment le brin positif de son arête. On peut donc trouver le segment $s_b$ à partir de son brin positif $b$ et retrouver le brin $b$ à partir du segment $s_b$.

**Remarque 10** *Par convention, nous dirons que le segment $s_b$ associé au brin $b$ relie le noeud associé au sommet $\sigma^*(b)$ au noeud associé à $\sigma^*(-b)$. Le segment non-orienté associé à l'arête $(b, -b)$ sera noté $s_{(b,-b)}$. Par exemple, sur la figure 6.6, nous dirons que le segment $s_4$ part du noeud associé au sommet $(4, 5, -3)$ et atteint le noeud associé au sommet $(-4, 6, -5)$.*

Un noeud contiendra ses coordonnées et un brin appartenant au cycle du sommet correspondant au noeud. Dans l'exemple de la figure 6.4 le noeud associé au sommet $(-4, 6, -5)$ contiendra, par exemple, le brin 6. Ce numéro de brin correspond à l'identificateur mentionné dans la section 6.1.1.

Nous avons vu que chaque face d'une carte planaire était associée à un cycle de $\varphi$. Pour une face $f$, le cycle $\varphi^*(\lambda^{-1}(f)) = (b_1, \ldots, b_n)$ défini l'ensemble des brins de la face. Par construction, pour tout $i$ dans $\{1, \ldots, n-1\}$ le dernier point de $s_{b_i}$ est égal au premier point de $s_{b_{i+1}}$ tandis que le dernier point de $s_{b_n}$ est égal au premier point de $s_{b_1}$. On peut donc construire le chemin $C_f = s_{b_1} \oplus s_{b_2} \oplus \ldots \oplus s_{b_n}$ ou $\oplus$ représente l'opération de concaténation de segments. Le chemin $C_f$ est un chemin fermé, composé uniquement de points-frontières. On vérifie facilement [Dom92] qu'il existe une région $R_f$ de l'image segmentée telle que tout point de $C_f$ a au moins un demi-voisin dans $R_f$. Le chemin $C_f$ constitue donc une frontière de la région $R_f$ (voir définition 21). La région $R_f$ est appelée la région associée à la face $f$. Un algorithme relativement simple permet de retrouver la face correspondant à une région à partir d'un point de la région. On peut donc associer une région à chaque face, parcourir l'ensemble des pixels d'une région associée à une face, retrouver une face à partir d'un des pixels de la région associée.

**Remarque 11** *En raison de l'orientation de $\sigma$ et de la convention définie dans la remarque 10, tous les points de la région $R_f$ associée à la face $f$, se trouveront à droite des segments orientés $S_{b_i}$ et donc à droite de la frontière $C_f$.*

Nous avons vu qu'une carte $(\varphi, \alpha)$ définit un ensemble de faces finies $\{f_1, \ldots, f_n\}$ et une face infinie $f_\varphi^\infty$. La frontière $C_{f_\varphi^\infty}$ voit tous les pixels des régions $R_{f_i}$ à sa gauche. Cette région est donc définie comme le complémentaire dans le plan entier $P_0$ de l'union des région $R_{f_i}$. On a donc :

$$R_{f_\varphi^\infty} = P_0 - \bigcup_{i=1}^{n} R_{f_i} \tag{6.1}$$

**Définition 36** *On dira qu'une région finie $R$ inclut la région infinie $R_{f_\varphi^\infty}$ (on notera $R_{f_\varphi^\infty} \subset R$) si :*

$$\bigcup_{i=1}^{n} R_{f_i} \subset R$$

Dans l'exemple de la figure 6.5, la région infinie correspondant à l'extérieur de la fenêtre est incluse dans la région correspondant au mur.

Par extension, on dira qu'une face finie $f$ de la carte planaire contient une face infinie $f_\varphi^\infty$ si et seulement si :

$$R_{f_\varphi^\infty} \subset R_f$$

On définit alors les fonctions **mère** et **filles** comme suit :

**Définition 37** *Soit $\{(\varphi_1, \alpha_1), \ldots, (\varphi_n, \alpha_n)\}$ un ensemble de cartes planaires liées au partitionnement d'une image.*

- *La* **mère** *d'une face infinie $f_\varphi^\infty$ est égale à la face $f$ qui l'inclut.*
- *Les* **filles** *d'une face finie $f$ sont les faces infinies $\{f_{\varphi_1}^\infty, \ldots, f_{\varphi_p}^\infty\}$ incluses dans $f$.*

Les deux fonctions **mère** et **filles** permettent de structurer un ensemble de cartes planaires. Elles nous indiquent en particulier si une face inclut ou non des face infinies. Ceci nous

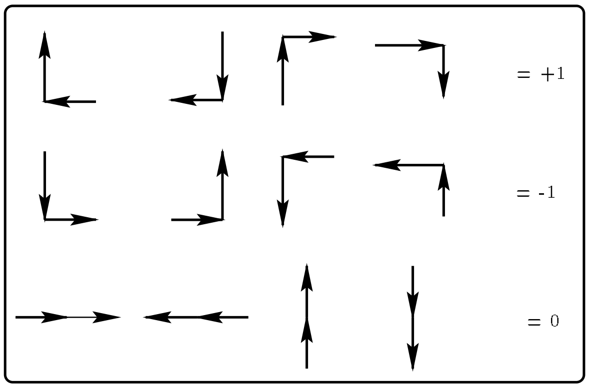



Figure 6.7: *Angle entre deux déplacements.*

permet, par exemple, de stocker le fait que la carte représentant la fenêtre est contenue dans la face représentant le mur (voir Figure 6.5).

Nous avons vu également qu'une face finie était parcourue dans le sens des aiguilles d'une montre alors qu'une face infinie était parcourue dans le sens positif. Cette propriété peut être utilisée pour caractériser les régions finies ou infinies à partir du niveau géométrique. On compte pour cela le nombre de "virages" à gauche ou à droite le long de la frontière d'une région $R$. On définit donc une fonction d'**angle** sur les codes de Freeman [Fre61] (voir figure 6.7). L'angle entre deux directions $d_1$ et $d_2$ sera noté $(\widehat{d_1, d_2})$. On peut constater sur la figure 6.7 que la fonction d'angle vérifie les propriétés suivantes :

**Proposition 5** *Soient $d_1$ et $d_2$ deux déplacements du code de Freeman, on a :*

$$(\widehat{d_1, d_2}) = -(\widehat{d_2, d_1})$$
$$(\widehat{d_1, d_2}) = -(\widehat{{d_1}^{-1}, d_2})$$
$$(\widehat{d_1, d_2}) = -(\widehat{d_1, {d_2}^{-1}})$$

*où $d_1^{-1}$ représente le déplacement opposé à la direction $d_1$.*

Cette fonction d'angle permet d'associer un **index de courbure** à chaque segment grâce à la définition suivante :

**Définition 38** *L'index de courbure d'un segment orienté $s$ reliant le noeud $n$ au noeud $n'$ par une suite de déplacements $(d_1, \ldots, d_n)$ est égal à :*

$$\boldsymbol{or}(s) = \sum_{i=2}^{n} (\widehat{d_{i-1}, d_i}) \qquad si\ n \neq n'$$

$$\boldsymbol{or}(s) = \sum_{i=2}^{n} (\widehat{d_{i-1}, d_i}) + (\widehat{d_n, d_1}) \qquad si\ n = n'$$

L'index de courbure d'un segment peut s'étendre à la frontière d'une région $R$ par la définition suivante :

**Définition 39** *Soit $C$ la frontière d'une région $R$ définie comme la concaténation des segments $(s_1, \ldots, s_n)$. L'index de courbure de $C$ est défini par la formule suivante :*

$$\boldsymbol{or}(C) = \left( \sum_{i=1}^{n} \boldsymbol{or}(s_i) + (\boldsymbol{Lm}(s_i), \widehat{\boldsymbol{Fm}}(s_{i+1})) \right) + (\boldsymbol{Lm}(s_n), \widehat{\boldsymbol{Fm}}(s_1))$$

*où $\boldsymbol{Fm}(s)$ et $\boldsymbol{Lm}(s)$ désignent les premiers et derniers déplacements du segment (voir section 6.1.1).*

Par construction, nous avons un brin pour chaque segment orienté et une frontière de région pour chaque face. On parlera donc de l'index de courbure d'un brin $b$ notée $\boldsymbol{or}(b)$, et de l'index de courbure d'une face $f$ notée $\boldsymbol{or}(f)$. On confondra donc $\boldsymbol{or}(b)$ et $\boldsymbol{or}(s_b)$, $\boldsymbol{Fm}(b)$ et $\boldsymbol{Fm}(s_b)$, $\boldsymbol{Lm}(b)$ et $\boldsymbol{Lm}(s_b)$, $\boldsymbol{or}(f)$ et $\boldsymbol{or}(C_f)$.

Le caractère fini ou infini d'une région est alors déterminé par le théorème suivant :

**Théorème 7** *Soit $(\varphi, \alpha)$ une carte liée à un partitionnement d'une image, on a :*

$$\begin{aligned} \forall f \in \boldsymbol{\mathcal{F}i}(\varphi)\ \boldsymbol{or}(f) &= 4 \\ \boldsymbol{or}(f_\varphi^\infty) &= -4 \end{aligned}$$

**Preuve:**

Voir [Dom92] □

**Corollaire 2** *Soit $s_b$ un segment fermé associé au brin $b$. L'orientation de $s_b$ est égale à 4 ou $-4$.*

**Preuve:**

$s_b$ étant un segment fermé, il définit une région. On a donc par construction :

$$\varphi(b) = b \text{ ou } \varphi(-b) = -b$$

Si, par exemple, $\varphi(b) = b$, on a :

$$\boldsymbol{or}(\lambda(b)) = \boldsymbol{or}(s_b) \in \{4, -4\}$$

L'index de courbure de $s_b$ est donc égal à 4 ou $-4$. □

**Définition 40** *Soient un noeud $n$ et un déplacement $d$ tels que le voisin $p_d$ de $n$ dans la direction $d$ soit un point-frontière. La fonction $\boldsymbol{\Sigma}$ appliquée au couple $(n, d)$ renvoie le déplacement $d'$ tel que le voisin $p_{d'}$ de $n$ dans la direction $d'$ est un point-frontière. Le point $p_{d'}$ est le premier point-frontière rencontré en tournant autour de $n$ dans le sens positif à partir de $p_d$ (voir Figure 6.8).*

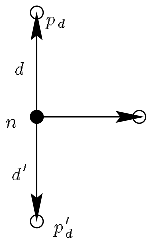



Figure 6.8: *L'image du couple* $(n, d)$ *par la fonction* $\boldsymbol{\Sigma}$ *est égal à* $d'$.

La fonction $\boldsymbol{\Sigma}$ est l'analogue au niveau géométrique de la permutation $\sigma$. De fait, on a par construction :

$$\forall n = (p, b) \in \boldsymbol{\mathcal{N}} \quad \forall b' \in \sigma^*(b) \quad \boldsymbol{\Sigma}(p, \boldsymbol{Fm}(s_{b'})) = \boldsymbol{Fm}(s_{\sigma(b')})$$

La donnée d'un segment induisant la donnée de ses noeuds-extrémités, on notera souvent $\boldsymbol{\Sigma}(\boldsymbol{Fm}(s_b))$ plutôt que $\boldsymbol{\Sigma}(n, \boldsymbol{Fm}(s_b))$ où $n$ est le noeud associé au sommet $\sigma^*(b)$.

### 6.1.4 Conclusion

Nous avons vu dans cette section les bases du modèle définies par J.P Domenger [Dom92]. Ce modèle permet de structurer l'ensemble des régions issues d'un algorithme de segmentation grâce à leurs frontières définies dans le plan $P_{\frac{1}{2}}$ et à un ensemble de cartes planaires. Si l'on désire integrer ce modèle à un algorithme de segmentation il nous faut rajouter un ensemble de méthodes permettant de découper une région en sous régions ou au contraire de fusionner deux régions. Dans le cadre d'une représentation par frontière ceci revient à inserer ou supprimer des segments. Nous nous sommes donc intéressé à l'évolution de la structure de données lors d'insertions ou de suppressions de segments. Une partie des résultats concernant l'insertion de segments ayant été traitée par J.P Domenger [Dom92], nous ne reprendrons ici que les insertions de noeuds décrites dans la section suivante. Nous avons en revanche dévelopé l'étude de certains cas d'insertion très fréquents en segmentation et conçu l'ensemble des méthodes permettant la suppression des segments. Ces méthodes seront décrites dans les sections suivantes.

## 6.2 Insertion et suppression de noeuds et sommets

### 6.2.1 Introduction

Nous avons vu dans la section précédente que notre modèle définit implicitement chaque région de l'image segmentée par sa frontière. La frontière d'une région est obtenue par la concaténation des segments qui la délimite. Si nous ajoutons un noeud $n$ à un segment $s$ nous coupons le segment $s$ en deux sous-segment $s'$ et $s''$. Cette opération ne modifie pas la frontière des régions associées à $s$. Donc, la coupure du segment $s$ a complexifié la structure de données sans modifier la partition de l'image segmentée, le noeud $n$ réalisant cette coupure est appelé un noeud **artificiel**.

**Définition 41** *Un noeud $n$ est dit* **artificiel**, *si son cardinal est égal à 2 et si il existe un autre noeud connecté à $n$ dans l'ensemble des points-frontières.*

D'un point de vue topologique, un noeud artificiel est un noeud dont le sommet associé $\sigma^*(b)$ est différent de $(b, -b)$.

**Définition 42** *La structure de données est dit* **minimale** *si elle ne comporte pas de noeuds artificiels.*

Si la structure de données est minimale, la frontière de deux régions adjacentes est définie par un seul segment. Toutefois, cette minimalité bien que souhaitable est trop restrictive pour certains algorithmes. En effet, les algorithmes de suppression ou d'insertion de segments peuvent rendre la structure temporairement non-minimale. Par exemple, l'insertion d'un nouveau segment $s$ d'extrémités $p$ et $p'$ commence par la création de deux nouveaux noeuds aux points $p$ et $p'$. Avant l'insertion de $s$ la structure ne sera pas minimale puisque les noeuds $n_p$ et $n_{p'}$ n'ont pas nécessairement un cardinal plus grand que 2. Plutôt que de travailler sur une structure de données minimale, les algorithmes vont supposer une propriété plus faible de la structure de données nommée la **cohérence**.

**Définition 43** *La structure de données sera dite* **consistante** *si la permutation $\sigma$ ne possède pas de point fixe. Donc si :*

$$\forall b \in \boldsymbol{\mathcal{B}} \;\; \sigma(b) \neq b$$

La cohérence de la structure de données interdit des arêtes $(b, -b)$ telles que $b$ ou $-b$ sont des points fixes de $\sigma$. Ces arêtes sont appelées des **arêtes pendantes**.

**Définition 44** *Une arête $(b, -b)$ est appelée une* **arête pendante** *si et seulement si :*

$$\sigma(b) = b \;\; ou \;\; \sigma(-b) = -b$$

Les arêtes pendantes apparaissent typiquement avec des méthodes de segmentation de type detection de contour (voir chapitre 5). Ces arêtes ne définissent pas des régions et sont donc automatiquement rejetées par le modèle.

**Remarque 12** *Un algorithme de détection de contour doit donc opérer une fermeture de contours avant d'inserer les segments dans le modèle. Une autre solution consiste à refermer les segments autours de leurs pixels extrémité. Dans ce cas, la fermeture de contour peut s'effecuter à l'intérieur de notre modèle.*

Il existe un autre type d'arête qui ne définit pas de régions. Il sagit des arêtes qui connectent soit deux faces infinies de même face mère, soit une face mère et une de ses filles (voir Figure 6.9). Ces arêtes sont appelées des **arêtes de connexion**.

**Définition 45** *Une arête* $(b, -b)$ *est appelée une* **arête de connexion** *si et seulement si :*

$$\lambda(b) = \lambda(-b)$$

*Les brins* $b$ *et* $-b$ *sont dans ce cas tous deux éléments de* $\varphi(\lambda^{-1}(b))$. *Les brins* $b$ *et* $-b$ *sont alors appelés des* **brins doubles**

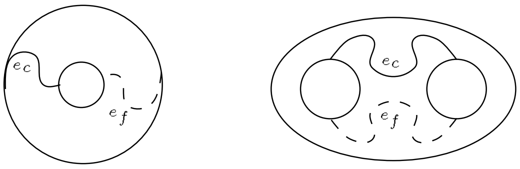



Figure 6.9: *L'insertion de l'arête de connexion* $e_c$ *suivi par l'insertion de l'arête* $e_f$ *permet de découper la région trouée.*

Ces arêtes de connexion sont nécessaires pour découper une région comportant un trou. La figure 6.9 représente les deux type d'arêtes de connexion. L' arête de connexion $e_c$ à tout d'abord été insérée dans la structure de données. Aprés l'insertion de l'arête $e_f$ une nouvelle face est crée et $e_c$ n'est plus une arête de connexion.

### 6.2.2 Insertion de noeuds

Afin d'éviter des arêtes pendantes, l'insertion d'un nouveau segment dans la structure de données implique que les deux extrémités soient des noeuds. Il faut donc créer les noeuds avant l'insertion du segment.

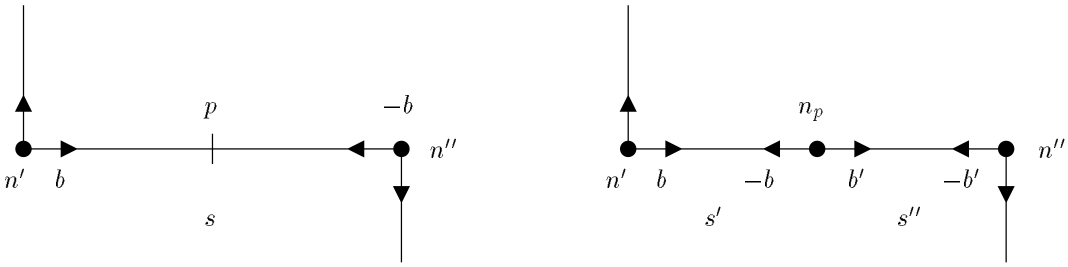



Figure 6.10: *Cette figure repésente l'insertion du noeud* $n_p$ *au point* $p$. *La création du noeud découpe le segment* $s$ *en* $s'$ *et* $s''$. *Les brins* $b$ *et* $-b$ *associés à* $s$ *sont utilisés par* $s'$ *et la nouvelle arête* $(b', -b')$ *est utilisée par* $s''$.

La création d'un nouveau noeud au point $p$ appartenant au segment $s$ découpe $s$ en deux nouveaux segments $s'$ et $s''$. Si le segment $s$ est associé à l'arête $(b, -b)$ nous associons à $s'$ l'arête $(b, -b)$. Ceci permet de limiter les modifications de la structure de données : seule une nouvelle arête correspondant à $s''$ est alors insérée.

La découpe de $s$ s'effectue de la façon suivante. Nous ajoutons un drapeau à l'entré $p$ du tableau $\boldsymbol{B}$ de façon à pouvoir l'identifié comme un noeud. Nous ajoutons ensuite le noeud $n_p = (p, b)$ dans l'ensemble des noeuds. Nous transformons alors la permutation $\sigma$ et la fonction $\lambda$ définies sur $\boldsymbol{\mathcal{B}}$ respectivement en $\sigma'$ et $\lambda'$ :

$$\begin{array}{ll|ll} \sigma'(-b') & = \sigma(-b) & \sigma'(\sigma^{-1}(-b)) & = -b' \\ \sigma'(-b) & = b' & \lambda'(-b') & = \lambda(-b) \\ \sigma'(b') & = -b & \lambda'(b') & = \lambda(b) \end{array}$$

La découpe du segment $s$ n'induit pas de modification de la subdivision. Le noeud $n_p$ permettant de découper $s$ est un noeud artificiel (voir définition 41). Les noeuds artificiels apparaissent naturellement après la supression d'un segment ou avant l'insertion d'un nouveau segment. Les noeuds artificiels créés avant l'insertion d'un segment deviennent imposés (voir définition 26) aprés l'insertion effective du segment. Inversement, les noeuds artificiels créés par la suppression d'un segment doivent être explicitement supprimés.

### 6.2.3 Suppression de noeuds

Nous avons vu que les arêtes de connexion et les noeuds artificiels n'avaient pas d'intéret pour la représentation de la subdivision. Donc, plusieurs structures de données peuvent représenter la même partition mais toutes ces structures peuvent être ramenées à la structure minimale en enlevant les arêtes de connexion et les noeuds artificiels. La suppression d'une arête de connexion sera traitée dans la section 6.4. Nous présentons à présent deux opérations utilisées pour réduire la taille de la structure de données : la supression de noeuds et l'extension d'un segment.

Soit $n$ un noeud artificiel associé au sommet $\sigma^*(b')$, ce noeud peut être supprimé de la structure de données sans modifier la partition. La suppression de noeud artificiel, implique la concaténation des segments $s_{(b',-b')}$ et $s_{(\sigma(b'),-\sigma(b'))}$. Si nous posons $b'' = -\sigma(b')$ le segment issue de la concaténation $s_{b''}$ est égal à : $s_{(b',-b')} \oplus s_{(b,-b)}$. On peut remarquer que cette concaténation contient la concaténation des deux segments orientés $s_{b''}$ et $s_{-b''}$. Il faut donc concaténer les couples de segments de même orientation. On aura par exemple $s_{b''} = s_b \oplus s_{b'}$ dans ce cas on a : $s_{-b''} = s_{-b'} \oplus s_{-b}$. Afin de réduire les modifications de la structure de données nous associons à $s_{(b'',-b'')}$ l'arête pré-existante $(b, -b)$. Les modifications de la permutation $\sigma$ et de la fonction $\lambda^{-1}$ sont alors les suivantes (voir Figure 6.10) :

$$\begin{array}{ll|ll} \sigma'(\sigma^{-1}(-b')) & = -b & \lambda^{-1'}(\lambda(-b')) & = -b \\ \sigma'(-b) & = \sigma(-b') & \lambda^{-1'}(\lambda(b')) & = b \end{array}$$

L'extension d'un segment consiste à étendre le segment en supprimant ces noeuds extrémités s'ils sont artificiels. La suppression de noeuds est itérée tant que les deux noeuds extrémités sont artificiels. L'extension du segment $s_{(b,-b)}$ crée un nouveau segment $s'_{(b,-b)}$ tel que :

$$s'_{(b,-b)} = s_{(b_{-n},-b_{-n})} \oplus \cdots \oplus s_{(-b,b)} \oplus \cdots \oplus s_{(b_m,-b_m)} \;\; avec \left\{ \begin{array}{lcll} b_i & = & \varphi^i(b), & 0 < i \leq m \\ b_{-i} & = & \varphi^i(-b), & -n \leq -i < 0 \\ \mid \sigma^*(b_i) \mid & = & 2, & -m \leq i \leq n \end{array} \right.$$

L'extension d'un segment peut être effectuée plus rapidement que l'itération des suppressions de noeuds artificiels en mettant à jour directement les sommets $\sigma^*(-b_m)$ et $\sigma^*(-b_{-n})$ de la façon suivante :

$$\begin{array}{ll} \sigma'(\sigma^{-1}(-b_{-n})) & = -b \\ \sigma'(-b) & = \sigma(-b_{-n}) \end{array} \;\Bigg\|\; \begin{array}{ll} \sigma'(\sigma^{-1}(-b_{-m}) & = b \\ \sigma'(b) & = \sigma(-b_m) \end{array}$$

Les noeuds artificiels sont simplement supprimés et leurs brins libérés. L'extension d'un segment préserve la cohérence de la structure de données.

## 6.3 Insertion de segments

Le modèle conçu par J.P. Domenger et J. P. Braquelaire [Dom92] était initialement dédié à la syntèse d'images 2D. Nous l'avons donc adapté à la segmentation. Comme nous l'avons mentioné dans la section 6.1.4 la plupart des méthodes d'insertion de contours ont été dévelopées par J.P Braquelaire et J.P Domenger [Dom92] (voir également [BP91, BD96b]). La plus grosse partie de notre travail sur les insertions de segments à porté sur les liaisons 8-connexe entre deux régions (voir Figure 6.11). Ce cas peut fréquent en édition d'image se produit fréquement lorsque l'on traite des segments issues d'une segmentation . Au niveau topologique ce cas correspond à deux faces possédant un seul sommet et aucune arête en commun. De telles faces sont appelées des **faces tangentes**.

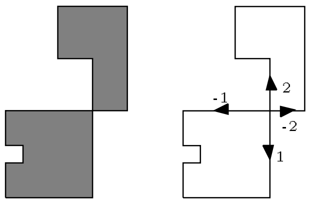



Figure 6.11: *Liaison 8-connexe entre deux régions et carte associée.*

### 6.3.1 Outils théoriques pour l'insertion de faces tangentes

Une liaison 8-connexe entre deux régions est caratérisée par l'existence d'un sommet, appelé **sommet d'étranglement** d'une face f.

**Définition 46** *On dira qu'un sommet $S = \sigma^*(b)$ est un* **sommet d'étranglement** *pour $f$ si et seulement si :*

$$|S| = 4 \;\; et \;\; |S|_f \in \{2, 3\}$$

La figure 6.11, nous permet d'illustrer le premier cas : si nous prenons $S = (-1, 1, -2, 2)$ et $f = (-1, -2)$, nous avons bien $|S| = 4$ et $|S|_f = 2$. Le cas $|S| = 4$ et $|S|_f = 3$ permet de tenir compte de la configuration illustré figure 6.12.

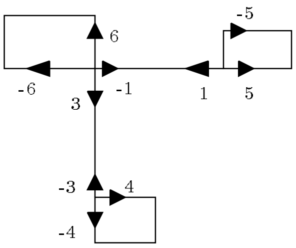



Figure 6.12: *Un exemple de cas tangent.*

On peut vérifier sur cet exemple que les brins $1, -1$ et $3, -3$ appartiennent à une même face. Ces brins doubles ne définissent pas une partitionnement en régions (voir définition 45).

Si l'on considère une carte sans brin double, donc si l'on suppose que l'ensemble des contours formant la partition d'une image ont été rajoutés la définition d'un noeud d'étranglement se simplifie de la façon suivante :

**Théorème 8** *Soit $(\varphi, \alpha)$ une carte sans brin double, $S$ un sommet de la carte $(\varphi, \alpha)$ et $f$ une face de cette carte. $S$ est un noeud d'étranglement pour $f$ si et seulement si $|S|_f = 2$.*

**Preuve:**

voir annexe section 7.8 □

**Proposition 6** *Soit $b$ un brin appartenant à $\boldsymbol{\mathcal{B}}$. Si $|\sigma^*(b)| = 4$ et $\sigma(b) = -b$ ou $\sigma(-b) = b$ alors $\sigma^*(b)$ est un sommet d'étranglement pour $\varphi^*(b)$.*

**Preuve:**

Suposons que $\sigma(b) = -b$. Le cardinal du sommet $\sigma^*(b)$ étant égal à 4 il existe deux brins $b_1$ et $-b_n$ tels que :

$$\sigma^*(b) = (b, -b, b_1, -b_n)$$

On a :

$$\varphi(-b) = \sigma(b) = -b$$

Donc $-b$ définit une face et $-b \notin \varphi^*(b)$. On a donc :

$$|\sigma^*(b)|_{\varphi^*(b)} \leq 3$$

De plus :

$$\left.\begin{array}{l} \varphi(b_n) = b \Rightarrow b \in \varphi^*(b_n) \\ \varphi(b) = b_1 \Rightarrow b_1 \in \varphi^*(b) = \varphi^*(b_n) \end{array}\right\} \Longrightarrow |\sigma^*(b)|_{\varphi^*(b)} \geq 2$$

On a donc :

$$|\sigma^*(b)| = 4 \text{ et } |\sigma^*(b)|_{\varphi^*(b)} \in \{2, 3\}$$

Le sommet $\sigma^*(b)$ est donc bien un sommet d'étranglement pour $\varphi^*(b)$. □

La définition 46 nous permet de caractériser un sommet d'étranglement, le théorème 8 nous donne une caractérisation plus simple si la carte ne comporte pas de brin double. Nous sommes donc en mesure de détecter la création d'un sommet d'étranglement au fur et à mesure de l'insertion de segments dans le modèle. La définition 46 et le théorème 8 ne nous donnent toutefois pas la liste des modifications du modèle entrainée par la création d'un noeud d'étranglement. D'importants éléments de réponse à cette question sont apportés par le lemme suivant :

**Lemme 3** *Soient $(\sigma, \alpha)$ une carte combinatoire et $f = (b_1, \dots, b_n)$ une face de cette carte. Soit $f' = (b, b_1, \dots, b_n)$ la face défine dans la carte $(\sigma', \alpha')$ déduite de la carte $(\sigma, \alpha)$ par l'ajout du brin $b$. On a alors :*

- $\varphi'(-b) = -b$
- $\sigma'^*(b)$ *est un noeud d'étranglement pour* $f'$.
- $\boldsymbol{or}(f') = \boldsymbol{or}(f) + \boldsymbol{or}(b) + 4$

**Preuve:**

On a $|\sigma^*(b_1)| \geq 2$, l'ajout d'un nouveau brin, ne pouvant enlever de brins à un noeud, on a : $|\sigma'^*(b_1)| \geq 2$ et $\{b_1, -b_n\} \subset \sigma'^*(b_1)$. De plus :

$$\varphi'(b_n) = b \Rightarrow \sigma'(-b_n) = b \Rightarrow b \in \sigma'^*(b_1)$$
$$\varphi'(b) = b_1 \Rightarrow \sigma'(-b) = b_1 \Rightarrow -b \in \sigma'^*(b_1)$$

Donc, $|\sigma'^*(b_1)| \geq 4$. les cycles étant de taille au plus 4, on a forcément :$|\sigma'^*(b_1)| = 4$. De plus les quatre éléments de $\sigma'^*(b_1)$ sont : $\{b, -b, b_1, -b_n\}$. On a donc :

$$\sigma'^*(b_1) = \sigma'^*(b) = \sigma'^*(-b) = \sigma'^*(-b_n)$$

Déterminons à présent le cycle $\sigma'^*(b_1)$. L'on sait déja que $\sigma'(-b) = b_1$ et $\sigma'(-b_n) = b$. Il nous faut donc calculer $\sigma'(b_1)$ et $\sigma'(b)$. $\sigma'$ étant bijective on a :

$$\sigma'(b) \notin \{b_1, b\} \Rightarrow \sigma'(b) \in \{-b, -b_n\}$$

Si $\sigma'(b) = -b_n$ on a :

$$\sigma'^2(b) = \sigma'(-b_n) = b$$

ce qui n'est pas possible, car on aurait alors :

$$\sigma'^*(b) = (b, -b_n) \neq \sigma'^*(b_1).$$

On a donc $\sigma'(b) = -b$. La bijectivité de $\sigma'$ nous impose alors : $\sigma'(b_1) = -b_n$. Le cycle de $b_1$ est donc égal à :

$$\sigma'^*(b_1) = \sigma'^*(b) = (b, -b, b_1, -b_n)$$

On a donc :

$$\varphi'(-b) = \sigma'(b) = -b$$

Le cyle $(-b)$ définit une face, le brin $-b$ n'appartient donc pas à $f'$ et $|\sigma'^*(b)|_{f'} \leq 3$. De plus les brins $b$ et $b_1$ appartiennent simultanément à $f'$ et au sommet $\sigma'^*(b)$. On à donc :

$$|\sigma'^*(b)| = 4 \text{ et } |\sigma'^*(b)|_{f'} \in \{2, 3\}$$

Le sommet $\sigma'^*(b)$ est donc bien un sommet d'étranglement.

Nous avons déjà établi les égalités suivantes (voir Figure 6.13) :

$$\begin{cases} |\sigma'^*(b)| = 4 \\ \sigma'^2(-b_n) = -b \\ \sigma'^2(b) = b_1 \end{cases}$$

On a donc (voir définition 40) :

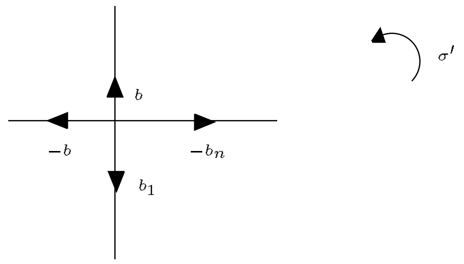



Figure 6.13:

$$\boldsymbol{Fm}(-b) = \boldsymbol{Fm}(\sigma'^2(-b_n)) = \boldsymbol{\Sigma}(\boldsymbol{Fm}(\sigma(-b_n)) = \boldsymbol{\Sigma}^2(\boldsymbol{Fm}(-b_n))$$

Le cardinal du cycle $|\sigma^*(b)|$ étant égal à 4 on a :

$$\boldsymbol{Fm}(-b) = \boldsymbol{\Sigma}^2(\boldsymbol{Fm}(-b_n)) = \boldsymbol{Fm}(-b_n)^{-1} = \boldsymbol{Lm}(b_n) \tag{6.2}$$

La notation $\boldsymbol{Fm}(-b_n)^{-1}$ représente ici le déplacement opposé au déplacement $\boldsymbol{Fm}(-b_n)$.

On obtient de même :

$$\boldsymbol{Fm}(b_1) = \boldsymbol{Lm}(-b) = \boldsymbol{Fm}(b)^{-1} \tag{6.3}$$

Or, par définition de l'index de courbure (voir définition 39)

$$\begin{aligned} \boldsymbol{or}(f') &= \boldsymbol{or}(b) + (\widehat{\boldsymbol{Lm}(b), \boldsymbol{Fm}}(b_1)) + \left(\textstyle\sum_{i=2}^{n} \boldsymbol{or}(b_i) + (\widehat{\boldsymbol{Lm}(b_{i-1}), \boldsymbol{Fm}}(b_i))\right) \\ &\quad + (\widehat{\boldsymbol{Lm}(b_n), \boldsymbol{Fm}}(b)) - (\widehat{\boldsymbol{Lm}(b), \boldsymbol{Fm}}(b)) \end{aligned}$$

On obtient donc :

$$\begin{aligned} \boldsymbol{or}(f') &= \boldsymbol{or}(f) + \boldsymbol{or}(b) - (\widehat{\boldsymbol{Lm}(b_n), \boldsymbol{Fm}}(b_1)) + (\widehat{\boldsymbol{Lm}(b_n), \boldsymbol{Fm}}(b)) \\ &\quad + (\widehat{\boldsymbol{Lm}(b), \boldsymbol{Fm}}(b_1)) - (\widehat{\boldsymbol{Lm}(b), \boldsymbol{Fm}}(b)) \end{aligned}$$

En tenant compte des équation 6.2 et 6.3 on obtient :

$$\begin{aligned} \boldsymbol{or}(f') &= \boldsymbol{or}(f) + \boldsymbol{or}(b) - (\widehat{\boldsymbol{Fm}(-b), \boldsymbol{Fm}}(b)^{-1}) + (\widehat{\boldsymbol{Fm}(-b), \boldsymbol{Fm}}(b)) \\ &\quad + (\widehat{\boldsymbol{Fm}(-b)^{-1}, \boldsymbol{Fm}}(b)^{-1}) + (\widehat{\boldsymbol{Fm}(-b)^{-1}, \boldsymbol{Fm}}(b)) \end{aligned}$$

$$\begin{aligned} \boldsymbol{or}(f') &= \boldsymbol{or}(f) + \boldsymbol{or}(b) + (\widehat{\boldsymbol{Fm}(-b), \boldsymbol{Fm}}(b)) - (\widehat{\boldsymbol{Fm}(b), \boldsymbol{Fm}}(-b)) \\ &\quad + (\widehat{\boldsymbol{Fm}(-b), Fm}(b)) - (\widehat{\boldsymbol{Fm}(-b), \boldsymbol{Fm}}(b)) \end{aligned}$$

$$\boldsymbol{or}(f') = \boldsymbol{or}(f) + \boldsymbol{or}(b) - 4(\widehat{\boldsymbol{Fm}(b), \boldsymbol{Fm}}(-b))$$

Or $|\sigma'^{*}(b)| = 4$ et $\sigma(b) = -b$ donc :

$$(\widehat{\boldsymbol{Fm}(b), \boldsymbol{Fm}}(-b)) = -1$$

On a donc :

$$\boldsymbol{or}(f') = \boldsymbol{or}(f) + \boldsymbol{or}(b) + 4$$

□

Le lemme 3 va nous permettre de décrire les transformation à apporter au modèle lors de l'insersion d'un sommet d'étranglement. En effet, soit $(\varphi_1, \alpha)$,$(\varphi_2, \alpha)$, $(\varphi, \alpha)$ trois cartes planaires.

- $(\varphi_1, \alpha)$ une carte réduite à une seule arête, donc telle que : $\varphi_1(b) = b; \varphi_1(-b) = -b$
- $(\varphi_2, \alpha)$ une carte quelquonque.
- $\varphi$ la composée de $\varphi_1$, $\varphi_2$ telle que l'on rajoute le brin $b$ dans une face $f_2 = (b_1, \ldots, b_n)$ de $(\varphi_2, \alpha)$ de façon à obtenir la face $f_3 = (b, b_1, \ldots, b_n)$.

Soit $f_1 = (-b)$ l'une des deux cartes de la carte $(\varphi_1, \alpha)$. La connaissance de l'orientation de $f_1$ permet de décomposer la création d'un noeud d'étranglement en 2 cas :

**Théorème 9** *Avec les conditions et notations précédentes :*

1. *Si* $f_1 \in \boldsymbol{\mathcal{F}i}(\boldsymbol{\varphi}_{1})$
   (a) $f_2 \in \boldsymbol{\mathcal{F}i}(\boldsymbol{\varphi}_{2}) \Rightarrow f_3 \in \boldsymbol{\mathcal{F}i}(\boldsymbol{\varphi})$
   (b) $f_2 = f^{\infty}_{\varphi_2} \Rightarrow f_3 = f^{\infty}_{\varphi}$
2. *Si* $f_1 = f^{\infty}_{\varphi_1}$ *alors :*
   $$f_2 = f^{\infty}_{\varphi_2} \text{ et } f_3 \in \boldsymbol{\mathcal{F}i}(\boldsymbol{\varphi})$$

**Preuve:**

On a : $\forall f \in \boldsymbol{\mathcal{F}}(\boldsymbol{\varphi})$ $or(f) \in \{4, -4\}$
De plus d'aprés le lemme 3 :

$$\boldsymbol{or}(f_3) = \boldsymbol{or}(f_2) + \boldsymbol{or}(b) + 4$$

1. Si $f_1 \in \boldsymbol{\mathcal{F}i}(\varphi_1)$
   on a : $\boldsymbol{or}(f_1) = \boldsymbol{or}(-b) = 4 \Rightarrow \boldsymbol{or}(b) = -4$
   Donc :
   $$\boldsymbol{or}(f_3) = \boldsymbol{or}(f_2)$$
   Les faces $f_2$ et $f_3$ ont la même orientation. Donc, si $f_2$ est finie $f_3$ le sera également. Inversement si $f_2$ est infinie, $f_3$ est infinie. On dit dans ce cas que $f_3$ hérite de l'orientation de $f_2$.

2. Si $f_1 = f^{\infty}_{\varphi_1}$. On a :
   $$\boldsymbol{or}(f_1) = \boldsymbol{or}(-b) = -4 \Rightarrow \boldsymbol{or}(b) = 4$$
   On a donc :
   $$\boldsymbol{or}(f_3) = 8 + \boldsymbol{or}(f_2)$$
   L'orientation d'une face ne pouvant être égale qu'a 4 ou $-4$, on a :
   $$\boldsymbol{or}(f_2) = -4 \Rightarrow f_2 = f^{\infty}_{\varphi_2} \text{ et}$$
   $$\boldsymbol{or}(f_3) = 4 \Rightarrow f_3 \in \boldsymbol{\mathcal{F}i}(\varphi)$$

□

Ce théorème et le lemme précédent permettent de déterminer quelles faces deviennent finies ou infinies lors de la création d'un noeud d'étranglement. Un représentation graphique des trois cas du théorème 9 est donnée figure 6.14.

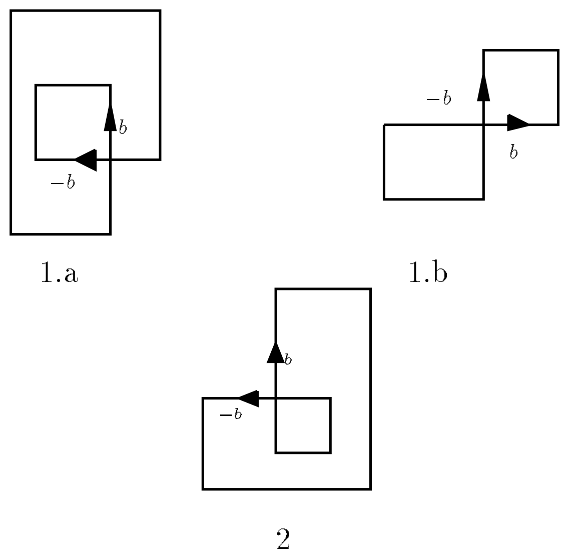



Figure 6.14: *Illustration des trois cas du téorème 9.*

**Remarque 13** *La décomposition de l'insertion d'un sommet d'étranglement en trois cas peut être obtenue en remarquant qu'une carte possède une seule face infinie. On a alors :*

- *$f_1$ et $f_3$ sont finies.*
- *$f_1$ est finie et $f_3$ infinie.*
- *$f_3$ est finie et $f_1$ infinie.*

*L'information suplémentaire fournie par le théorème 9 et la possibilité de décider dans quel cas l'on se trouve en fonction des paramètres des deux cartes à combiner.*

### 6.3.2 L'insertion de faces tangentes

Si nous reprenons les notations de la section précédente, l'insertion d'un noeud d'étranglement va transformer le noeud $(b_1, -b_n)$ en $(b, -b, b_1, -b_n)$ (voir Figure 6.13). Pour un algorithme de segmentation, cette insertion correspond au cas où l'on a inséré un ensemble faces et où on désire ajouter une autre face $f_1 = (-b)$ tangente à une face déja insérée $f_2$. Nous supposons implicitement qu'il y a correspondance entre la géométrie et la topologie, donc que nous avons :

$$\boldsymbol{\Sigma}(\boldsymbol{Fm}(s_b)) = \boldsymbol{Fm}(s_{\sigma(b)}) = \boldsymbol{Fm}(s_{-b}) \tag{6.4}$$

Si ce n'est pas le cas, nous devons inverser le rôle de $b$ et $-b$. Nous suposerons dans la suite de cette section que l'équation 6.4 est vérifiée.

Nous transformons alors $\sigma$ en $\sigma'$ de la façon suivante :

$$\sigma'(b) = -b \ ; \ \sigma'(-b) = b_1 \ ; \ \sigma'(-b_n) = b \tag{6.5}$$

La fonction $\sigma$ restant inchangée pour tous les autres brins. La transformation de $\varphi$ est induite par celle de $\sigma$.

Les fonctions $\lambda$ et $\lambda^{-1}$ sont alors mises à jour en rajoutant une nouvelle étiquette $l$ pour la face $f_1 = (-b)$. On a alors :

$$\begin{array}{rcl} \lambda'(-b) & = & l \\ \lambda'(b) & = & \lambda(b_n) \\ \lambda^{-1}(l) & = & -b \end{array}$$

La mise à jour des fonctions **mère** et **filles** nécessite d'utiliser la décomposition en trois cas définie dans le théorème 9. Le caractère fini ou infini de $f_1$ est testé simplement par le signe de $\boldsymbol{or}(b)$. La finitude de $f_2$ est supposée connue. De plus l'étiquette de $f_3$ est égal à celle de $f_2$.

### 6.3.3 Cas 1.a : $f_1$ et $f_2$ sont finies

Dans ce cas nous savons par le théorème 9 que $f_3$ est finie. Il nous faut alors tester si $f_1$ n'englobe pas des faces qui étaient auparavant incluses dans $f_2$. Il nous faut donc


1.a

Figure 6.15: *Découpe d'une face finie.*

parcourir l'ensemble **filles**$(f_2)$ et tester pour chacune de ces faces la face englobante. Soit $\{f^\infty_{\varphi_1}, \ldots, f^\infty_{\varphi_n}\}$ la liste des faces de **filles**$(f_2)$ incluses dans $f_1$. On met à jour les fonctions d'inclusion de la façon suivante :

$$\begin{array}{lcl} \textbf{mère}(f^\infty_{\varphi_i}) & = & f_1 \quad \forall i \in \{1, \ldots, n\} \\ \textbf{filles}(\lambda(-b)) & = & \{f^\infty_{\varphi_1}, \ldots, f^\infty_{\varphi_n}\} \\ \textbf{filles}(\lambda(b)) & = & \textbf{filles}(\lambda(b)) - \{f^\infty_{\varphi_1}, \ldots, f^\infty_{\varphi_n}\} \end{array}$$

Il est intéressant de remarquer que si le segment $s_b$ à une longueur inférieure à 12 on peut être certain que $R_{f_1}$ ne peut inclure d'autres régions. Dans ce cas l'on n'a pas à tester **filles**$(f_2)$. Cette remarque permet de limiter le nombre de tests d'inclusion. Ceci est particulièrement utile dans le cas d'images trés bruitées ou de nombreuses régions d'un pixel sont rajoutées.

### 6.3.4 Cas 1.b : $f_1$ est finie et $f_2$ infinie


1.b

Figure 6.16: *Découpe d'une face infinie.*

Dans ce cas la nouvelle face $f_3$ est également infinie. La face finie $f_1$ peut englober des faces précédement englobées par la mère de $f_2$. Soient $\{f^\infty_{\varphi_1}, \ldots, f^\infty_{\varphi_n}\}$ l'ensemble des faces incluses dans $f_1$ et $m$ la mère de la face $f_2$ ($m$ = **mère**$(\lambda(b_1))$). On met à jour les fonctions mère, filles de la façon suivante :

$$\begin{array}{lcl} \textbf{mère}(f^\infty_{\varphi_i}) & = & f_1 \ \forall i \in \{1, \ldots, n\} \\ \textbf{filles}(\lambda(-b)) & = & \{f^\infty_{\varphi_1}, \ldots, f^\infty_{\varphi_n}\} \\ \textbf{filles}(m) & = & \textbf{filles}(m) - \{f^\infty_{\varphi_1}, \ldots, f^\infty_{\varphi_n}\} \end{array}$$

Comme dans le cas 1.a, on peut assurer que l'ensemble $\{f^\infty_{\varphi_1}, \ldots, f^\infty_{\varphi_n}\}$ est vide si la longueur du segment est inférieure ou égale à 12.

### 6.3.5 Cas 2 : $f_1$ est infinie


2

Figure 6.17: *Création d'une nouvelle face infinie.*

Nous savons grâce au théorème 9 que $f_2$ était forcément infinie et que $f_3$ est une face finie. Ce cas correspond à la supression d'une face infinie $f_2$ et à la création d'une nouvelle face infinie $f_1$. Ceci se traduit dans la mise à jour des fonctions **mère** et **filles** de la façon suivante.

Soient

- $m = \textbf{mère}(\lambda(b_1))$
- $\{f^{\infty}_{\varphi_1}, \ldots, f^{\infty}_{\varphi_n}\}$ les faces infinies contenues dans $\textbf{filles}(m)$ et inclues dans $f_3$.

on a :

$$\begin{array}{lcl}
\textbf{mère}(\lambda(-b)) & = & m \\
\textbf{mère}(\lambda(b)) & = & NULL \\
\textbf{mère}(f^{\infty}_{\varphi_i}) & = & \lambda(b)\ \forall i \in \{1, \ldots, n\} \\
\textbf{filles}(\lambda(b)) & = & \{f^{\infty}_{\varphi_1}, \ldots, f^{\infty}_{\varphi_n}\} \\
\textbf{filles}(m) & = & \textbf{filles}(m) \cup \{\lambda(-b)\} - \{\lambda(b), f^{\infty}_{\varphi_1}, \ldots, f^{\infty}_{\varphi_n}\}
\end{array}$$

Le positionement de $\textbf{mère}(\lambda(b))$ à $NULL$ signifie que $\lambda(b)$ n'a pas de mère. C'est donc une face finie. Cette fois ci, la face $\lambda(b)$ ne peut contenir de faces incluses si la longueur du segment $s_b$ est inférieure ou égale à 16.

## 6.4 Effacement de segments

### 6.4.1 Introduction

Nous avons vu dans la section précédente un ensemble de méthodes permettant de modifier une partition en rafinant des régions déjà créées. Une autre façon de modifier la partition d'une image, consiste à supprimer des segments de la structure de données. La suppression de segments peut être utilisée pour fusionner un ensemble de régions ou pour supprimer une région existante. La suppression d'un segment doit preserver la cohérence de la structure de données (voir définition 43), et donc ne pas créer d'arêtes pendantes.

### 6.4.2 L'algorithme SUPPRESS_EDGE

Soit $s_b$ un segment dont les moeuds extrémités $n$ et $n'$ sont respectivement associés aux sommets $\sigma^*(b)$ et $\sigma^*(-b)$. La suppression de $s_b$ décroit de 1 le cardinal des noeuds $n$ et $n'$ si ils sont différents et de 2 s'ils sont égaux. La suppression du segment $s_b$ crée donc une structure inconsistante dans deux cas :

Si $n$ et $n'$ sont différents et si l'un des deux est un noeud artificiel. Supposons que $n$ soit artificiel, dans ce cas la suppression de l'arête $(b, -b)$ crée l'arête pendante $(\sigma(b), -\sigma(b))$. Afin de preserver la cohérence de la structure de données nous ne supprimons donc que les segments dont les extrémitées ne sont pas des noeuds artificiels. La validation de cette précondition impose un pré-traitement pour étendre le segment (voir section 6.2 et Figure 6.19-a).

La structure de données peut également devenir inconsistante aprés la suppression de $s_b$ si $n$ et $n'$ représentent le même noeud de cardinal 3. Dans ce cas, l'on a :

$$|\sigma^*(b)| = 3 \text{ et } \left(\sigma^2(b) = -b \text{ ou } \sigma^2(-b) = b\right)$$

Supposons, par exemple, que le noeud $n$ soit associé au sommet $\sigma^*(b) = (b, \sigma(b), -b)$. Alors la suppression de l'arête $(b, -b)$ crée l'arête pendante $(\sigma(b), -\sigma(b))$. Pour éviter les arêtes pendantes l'arête $(\sigma(b), -\sigma(b))$ doit être effacé avant l'arête $(b, -b)$ (voir Figure 6.19-b). Ces traitements sont résumés dans l'algorithme SUPPRESS_EDGE défini dans la figure 6.18.

```
SUPPRESS_EDGE(brin b)
{

    Traitements preliminaires
    si(| σ*(b) |= 2 ou | σ*(−b) |= 2) alors
        etendre s_b
    si(σ²(b) = −b et | σ*(b) |= 3 ) alors
        SUPPRESS_EDGE(σ(b))
    si(σ²(−b) = b et | σ*(b) |= 3 ) alors
        SUPPRESS_EDGE(σ(−b))
    Fin des pretraitements
    Enlever effectivement (b, −b)
}
```

Figure 6.18: *L'algorithme* SUPPRESS_EDGE *enlève l'arête* $(b, -b)$. *Pour éviter des arêtes pendantes des traitements préliminaires peuvent être effectués.*

**Proposition 7** *L' algorithme* SUPPRES_EDGE *a une récursivité d'au plus un.*

**Preuve:**

Soit $b$ le brin passé en argument de SUPPRESS_EDGE. Supposons que $\sigma^2(b) = -b$ et $|\sigma^*(b)| = 3$. L'algorithme SUPPRESS_EDGE a une récursivité plus grande que un si et seule-

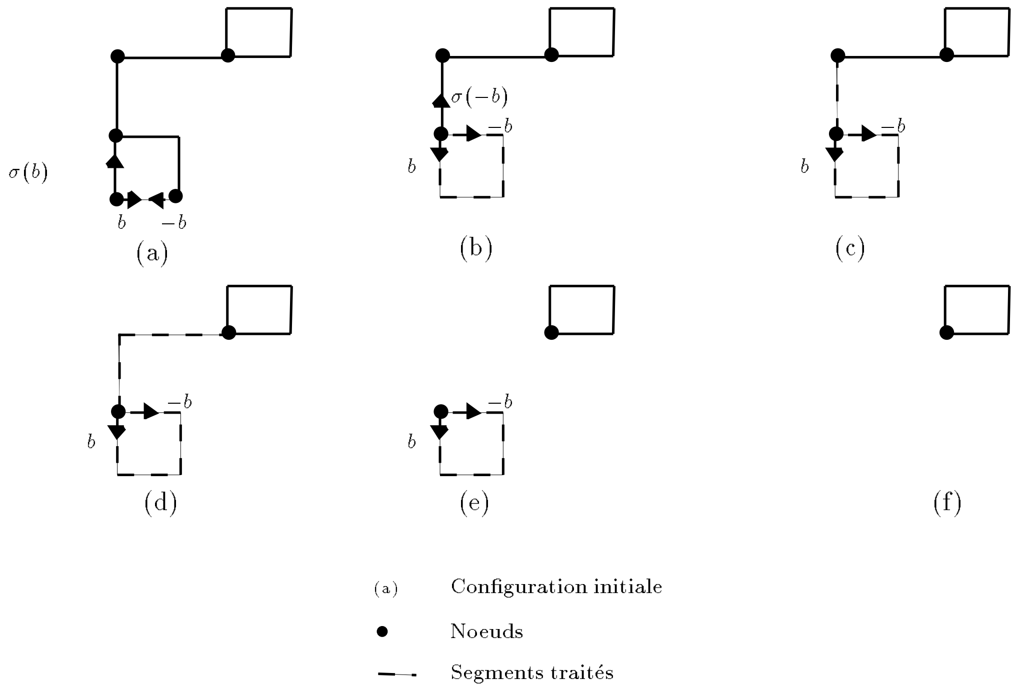



Figure 6.19: *Cette figure montre le déroulement de l'algorithme* SUPPRESS_EDGE *sur une configuration particulière. L'algorithme est lancé sur l'arête* $(b, -b)$.

ment si :

$$\sigma^2(-\sigma(b)) = \sigma(b) \text{ ou } \sigma^3(b) = -\sigma(b)$$

On a alors :

$$\begin{array}{lcll||lcll}
\sigma^2(-\sigma(b)) & = & \sigma(b) & (|\sigma^*(\sigma(b))| = 3) & \sigma^2(\sigma(b)) & = & -\sigma(b) & (|\sigma^*(b)| = 3) \\
-\sigma(b) & = & \sigma^2(b) & (\sigma^2(b) = -b) & b & = & -\sigma(b) & \\
-\sigma(b) & = & -b & & -b & = & \sigma(b) & (-b = \sigma^2(b)) \\
\sigma(b) & = & b & & \sigma(b) & = & b &
\end{array}$$

Puisque $(b, -b)$ n'est pas une arête pendante et que l'extension du segment ne produit pas d'arêtes pendantes on a : $\sigma(b) \neq b$. Donc la récursivité de l'algorithme est au plus d'un. □

**Remarque 14** *Il est également possible de supprimer le test $\sigma^*(b) = 3$ dans l'algorithme* SUPPRESS_EDGE. *Dans ce cas l'algorithme supprimera également des arêtes de connexion inutiles pour un partitionnement. On peut montrer que cette nouvelle version de l'algorithme à également une récursivité d'au plus un.*

L'algorithme SUPPRESS_EDGE préserve la cohérence de la structure de données puisqu'il supprime effectivement une arête que si les deux sommets extrémités ont un cardinal nul ou au moins égal à 2 aprés la suppression. Nous allons à présent présenter les modifications de la structure de données induites par la suppression effective d'une arête. Les méthodes que nous allons présenter permettent de réaliser la dernière ligne de l'algorithme SUPPRESS_EDGE, elles doivent donc être utilisées en conjonction avec celui-ci.

### 6.4.3 Suppression effective de segments

Nous allons dans cette section étudier l'effacement effectif d'un segment $s_{(b,-b)}$. Les noeuds extrémités sont notés $n$ et $n'$. Le noeud $n$ est associé au sommet $\sigma^*(b)$ tandis que le noeud $n'$ est associé au sommet $\sigma^*(-b)$. Dans tous les algorithmes présentés dans cette section nous supposerons que l'ordre d'effacement effectif du segment viens de l'algorithme `SUPPRESS_EDGE`. L'effacement du segment conservera donc la cohérence de la structure de données.

#### 6.4.3.1 Cas : $n = n'$ et $|\sigma^*(b)| = 2$

Si nous avons $n = n'$ alors le sommet associé à $n$ contient les brins $b$ et $-b$. Puisque l'algorithme `SUPPRESS_EDGE` nous à autorisé à effacé le brin $b$, le noeud $n$ aura un cardinal égal à 0 ou 2 aprés l'effacement. Le noeud $n$ à donc un cardinal égal à 2 ou 4.

Si le cardinal de $n$ est égal à 2 sont rang deviendra égal à 0 aprés la suppression et $n$ est supprimé. Dans ce cas nous avons $\sigma(b) = -b$ et $\sigma(-b) = b$. Donc nous pouvons simplement enlever le cycle $(b, -b)$ de la structure de données en libérant les entrées $b$ et $-b$ du tableau $\sigma$. Le noeud $n$ est également libéré. Supposons que $\lambda(b)$ est la face finie et $\lambda(-b)$ la face infinie. La relation d'inclusion se met alors à jour de la façon suivante :

$$\begin{array}{lcl} \textbf{filles}'(\textbf{mère}(\lambda(-b))) & = & \textbf{filles}(\textbf{mère}\lambda(-b)) \cup \textbf{filles}(\lambda(b)) - \{\lambda(-b)\} \\ \textbf{mère}'(l) & = & \textbf{mère}(\lambda(-b)),\ \forall l \in \textbf{filles}(\lambda(b)) \end{array}$$

#### 6.4.3.2 Cas : $n = n'$ et $|\sigma^*(b)| = 4$

Si le cardinal de $n$ est égal à 4. Les brins $b$ et $-b$ sont ou ne sont pas consécutifs dans le cycle $\sigma^*(b)$.

**Si $b$ et $-b$ sont consécutifs dans $\sigma^*(b)$.** Alors le sommet $\sigma^*(b)$ est un sommet d'étranglement (voir définition 46 et proposition 6). Si nous reprenons les notations du lemme 3 le cycle $\sigma^*(b)$ s' écrit :

$$\sigma^*(b) = (b, -b, b_1, -b_n)$$

Nous avons déja $\sigma(b_1) = -b_n$. Les modifications à apporter à la permutation $\sigma$ se limitent donc à (voir Figure 6.13) :

$$\sigma'(-b_n) = b_1$$

De plus nous avons, $\varphi(b_n) = b$, donc $\lambda(b_n) = \lambda(b)$. Si $\lambda^{-1}(\lambda(b))$ est égal à $b$ il suffit de positionner $\lambda^{-1}(\lambda(b))$ à $b_n$. La mise à jour des relations d'inclusions nécessite d'utiliser la décomposition en trois cas définie dans la remarque 13.

- Si $\lambda(b)$ et $\lambda(-b)$ sont des faces finies (cas 1.a, voir Figure 6.15).

On libère l'étiquette $\lambda(-b)$. Si $\lambda^{-1}(\lambda(b)) = b$ on positionne $\lambda^{-1}(\lambda(b))$ à $\varphi(b)$. On met ensuite à jour les relations d'inclusions de la façon suivante :

$$\begin{array}{lcl} \mathbf{filles}'(\lambda(b)) & = & \mathbf{filles}(\lambda(b)) \cup \mathbf{filles}(\lambda(-b)) \\ \mathbf{m\grave{e}re}'(l) & = & \lambda(b),\ \forall l \in \mathbf{filles}(\lambda(-b)) \end{array}$$

- Si $\lambda(b)$ est infinie et $\lambda(-b)$ est finie (cas 1.b, voir Figure 6.16).

  On supprime à nouveau $\lambda(-b)$ de la structure de données. On repositionne également $\lambda^{-1}(\lambda(b))$ si $\lambda^{-1}(\lambda(b)) = b$. On met ensuite à jour les relations d'inclusions :

$$\begin{array}{lcl} \mathbf{filles}'(\mathbf{m\grave{e}re}(\lambda(b))) & = & \mathbf{filles}(\mathbf{m\grave{e}re}(\lambda(b))) \cup \mathbf{filles}(\lambda(-b)) \\ \mathbf{m\grave{e}re}'(l) & = & \mathbf{m\grave{e}re}(\lambda(b)),\ \forall l \in \mathbf{filles}(\lambda(-b)) \end{array}$$

- Si $\lambda(b)$ est finie et $\lambda(-b)$ infinie (cas 2, voir Figure 6.17).

  L'étiquette $\lambda(-b)$ n'est pas libéré mais affecté au cycle de $\lambda(b)$. On a donc :

$$\forall b' \in \varphi^*(b)\ \lambda(b') = \lambda(-b)$$

  De plus si $\lambda^{-1}(\lambda(-b))$ est égal à $-b$, on positionne $\lambda^{-1}(\lambda(b))$ à $\varphi(b)$. On libère alors l'étiquette $\lambda(b)$. Les relations d'inclusions se mettent à jour de la façon suivante :

$$\begin{array}{lcl} \mathbf{filles}'(\mathbf{m\grave{e}re}(\lambda(-b))) & = & \mathbf{filles}(\mathbf{m\grave{e}re}(\lambda(-b))) \cup \mathbf{filles}(\lambda(b)) \\ \mathbf{m\grave{e}re}'(l) & = & \mathbf{m\grave{e}re}(\lambda(-b)),\ \forall l \in \mathbf{filles}(\lambda(b)) \end{array}$$

**Si $|\sigma^*(b)| = 4$ et $b$ et $-b$ ne sont pas consécutifs dans $\sigma^*(b)$.** Ce cas est représenté sur la figure 6.20. La supression de l'arête $(b, -b)$ va supprimer une face finie et modifier le cycle de la face infinie. Supposons que $\varphi^*(b)$ correspond à la face finie et $\varphi^*(-b)$ à la face infinie. Les brins appartenant au cycle de $\varphi^*(b)$, vont devenir des brins de la face infinie $\lambda(-b)$. On parcourt donc, le cycle $\varphi^*(b)$ afin de ré-étiqueter ses brins à $\lambda(-b)$. La mise à jour de la permutation $\sigma$ et de la fonction $\lambda^{-1}$ est alors effectué grace aux affectations suivantes :

$$\begin{array}{c} \sigma'(\sigma^{-1}(b)) = \sigma(b) \\ \sigma'(\sigma^{-1}(-b)) = \sigma(-b) \\ \lambda^{-1\prime}(\lambda(-b)) = \varphi(-b) \end{array}$$

On libère ensuite l'étiquette $\lambda(b)$ et on met à jour les relations d'inclusions :

$$\left.\begin{array}{lcl} \mathbf{filles}'(m) & = & \mathbf{filles}(m) \cup \mathbf{filles}(\lambda(b)) \\ \mathbf{m\grave{e}re}'(l) & = & m,\ \forall l \in \mathbf{filles}(\lambda(b)) \end{array}\right\} \text{ avec } m = \mathbf{m\grave{e}re}(\lambda(-b))$$

#### 6.4.3.3 Cas : $n \neq n'$

Supposons à présent que $n \neq n'$. La suppression de $(b, -b)$ dans la structure de données va modifier la permutation $\sigma$ de la façon suivante :

$$\sigma'(\sigma^{-1}(b)) = \sigma(b)\ \ ;\ \ \sigma'(\sigma^{-1}(-b)) = \sigma(-b)$$

La mise à jour des relations d'inclusions peut être décomposée en deux cas selon que l'arête $(b, -b)$ est ou non une arête de connexion (voir définition 45).

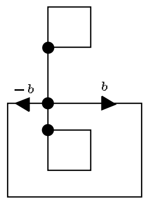



Figure 6.20: *Les brins $b$ et $-b$ ne sont pas consécutifs dans $\sigma^*(b)$.*

**Si l'arête $(b, -b)$ est une arête de connexion.** La suppression de l'arête $(b, -b)$ découpe le cycle $\varphi^*(b)$ en deux cycles $\varphi'^*(\sigma(b))$ et $\varphi'^*(\sigma(-b))$.

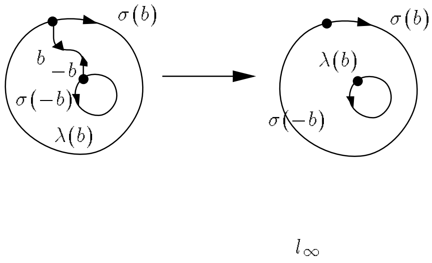



Figure 6.21: *L'arête $(b, -b)$ connecte une face à sa mère.*

Si $\lambda(b)$ est une face finie, alors $(b, -b)$ connecte une face mère à une de ses filles (voir Figure 6.21). Après la suppression de $(b, -b)$ la face mère est égale à $\varphi'^*(\sigma(b))$ ou $\varphi'^*(\sigma(-b))$. La détermination de la face mère se fait grâce à l'index de courbure (voir définition 39). Supposons que la face d'index de courbure positif soit $\varphi'^*(\sigma(b))$. Dans ce cas $\varphi'^*(\sigma(b))$ est une face finie et les brins de la face $\varphi'^*(\sigma(-b))$ sont indexés avec une nouvelle étiquette $l_\infty$. Cette étiquette correspond à la nouvelle face infinie. Les relations d'inclusions sont mises à jour de la façon suivante :

$$\begin{array}{lcl}
\mathbf{filles}'(\lambda(b)) & = & \mathbf{filles}(\lambda(d)) \cup \{l_\infty\} \\
\mathbf{m\grave{e}re}'(l_\infty) & = & \lambda(b) \\
\lambda^{-1'}(\lambda(b)) & = & \sigma(b) \\
\lambda^{-1'}(l_\infty) & = & \sigma(-b)
\end{array}$$

Si $\lambda(b)$ est une face infinie, alors $(b, -b)$ connecte deux faces de même mère (voir Figure 6.22).

Les deux cycles $\varphi'^*(\sigma(b))$ et $\varphi'^*(\sigma(-b))$ définissent des faces infinies. Les brins appartenant à $\varphi'^*(\sigma(b))$ restent arbitrairement étiquetés avec $\lambda(b)$ tandis que les brins du cycle $\varphi^*(\sigma(-b))$ sont associés à la nouvelle étiquette $l_\infty$. Les relations d'inclusions sont alors mises à jour de la façon suivante :

$$\begin{array}{lcl}
\mathbf{filles}'(\mathbf{m\grave{e}re}(\lambda(b))) & = & \mathbf{filles}(\mathbf{m\grave{e}re}(\lambda(b))) \cup \{l_\infty\} \\
\mathbf{m\grave{e}re}'(l_\infty) & = & \mathbf{m\grave{e}re}(\lambda(b)) \\
\lambda^{-1'}(\lambda(b)) & = & \sigma(b) \\
\lambda^{-1'}(l) & = & \sigma(-b)
\end{array}$$

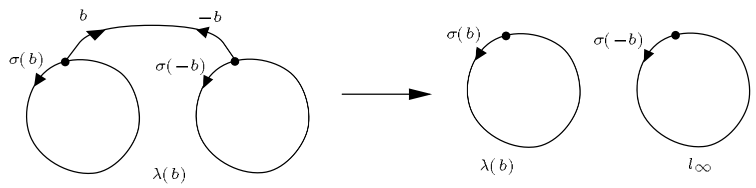



Figure 6.22: *Dans ce cas l'arête* $(b, -b)$ *connecte deux faces soeurs.*

**Si l'arête** $(b, -b)$ **n'est pas une arête de connexion.** La suppression de $(b, -b)$ induit la suppression d'une face finie. Dans ce cas les cycles $\varphi^*(b)$ et $\varphi^*(-b)$ sont fusionnés. Nous distinguons à nouveau deux sous-cas en fonction du caractère finie ou infinie des cycles $\varphi^*(b)$ et $\varphi^*(-b)$.

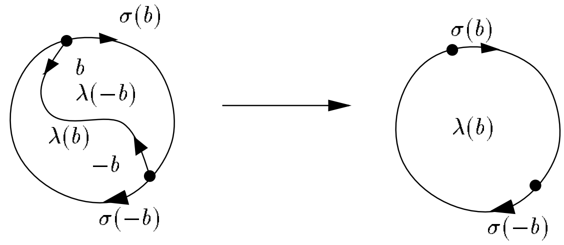



Figure 6.23: *L'arête* $(b, -b)$ *est partagée par deux faces finies.*

Si $\lambda(b)$ et $\lambda(-b)$ sont deux faces finies, la suppression de l'arête $(b, -b)$ étend la région associée à une des deux faces en la fusionnant à la région associé à l'autre face. Supposons, par exemple que nous étendions la région associé à létiquette $\lambda(b)$. Dans ce cas, la face $\lambda(-b)$ est libérée et l'on obtient une nouvelle région associé à $\lambda(b)$ et de domaine l'union des domaines des deux régions. Nous changeons, donc les étiquettes des brins du cyle $\varphi^*(-b)$ en les affectant à $\lambda(b)$ et les filles de $\lambda(-b)$ deviennent les filles de $\lambda(b)$. Ceci se traduit par les équations suivantes :

$$
\begin{array}{lcl}
\lambda(b') & = & \lambda(b) \ \ \forall b' \in \varphi^*(\lambda^{-1}(\lambda(-b))) \\
\lambda^{'-1}(\lambda(b)) & = & \sigma(b) \\
\mathbf{filles}'(\lambda(b)) & = & \mathbf{filles}(\lambda(b)) \cup \mathbf{filles}(\lambda(-b)) \\
\mathbf{m\grave{e}re}'(l) & = & \lambda(d) \ ; \ \forall l \in \mathbf{filles}(\lambda(-b))
\end{array}
$$

Si une des face $\lambda(-b)$ ou $\lambda(b)$ est infinie. Nous utilisons la fonction **mère** pour déterminer la face infinie. Supposons que $\lambda(b)$ soit infinie et $\lambda(-b)$ finie. La face $\lambda(-b)$ sera supprimée de la structure de données et les brins de $\varphi^*(-b)$ seront ré-étiquetés avec $\lambda(b)$. Soit $m$ là mère de la face $\lambda(b)$. La structure de données est mise à jour comme suit :

$$
\begin{array}{lcl}
\mathbf{filles}'(m) & = & \mathbf{filles}(m) \cup \mathbf{filles}(\lambda(-b)) \\
\mathbf{m\grave{e}re}'(l) & = & m, \ \forall l \in \mathbf{filles}(\lambda(-b)) \\
\lambda^{-1'}(\lambda(b)) & = & \sigma(b)
\end{array}
$$

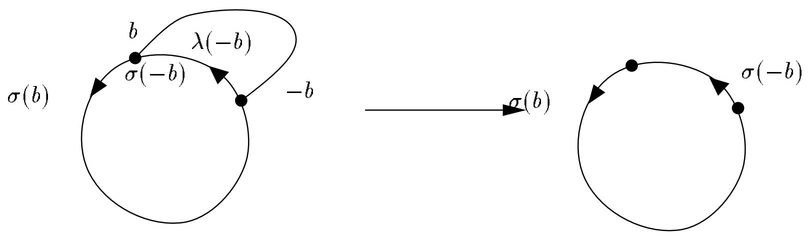



Figure 6.24: *L'arête $(b, -b)$ sépare une face finie et une face infinie.*

## 6.5 Conclusion

Nous avons présenté dans ce chapitre une struture de données permettant de représenter la partition d'une image en régions. Afin de pouvoir intégrer des algorithmes de segmentation à cette structure de données nous avons complété les méthodes d'insertion de segments définies par J.P Domenger [Dom92] et défini un ensemble de méthodes permettant la suppression de segments dans la structure de données. Ces deux fonctionalités : insertion et suppression de segments, permettent de découper ou de fusionner un ensemble de régions et sont donc à la base des algorithmes de segmentation (voir chapitre 5). Nous allons dans le chapitre suivant présenter un ensemble de méthodes de segmentation utilisant ce modèle ainsi que plusieurs fonctionalités permettant de définir facilement des algorithmes de segmentation à l'aide de cette structure de données.

# Chapitre 7

# Utilisation du modèle pour la segmentation

## 7.1 Architecture de l'application

L'application que nous avons développée se compose d'un noyau codant le modèle de contour décrit dans le chapitre précédent et d'un ensemble d'applications structurées en couches autour du noyau (Voir Figure 7.1).

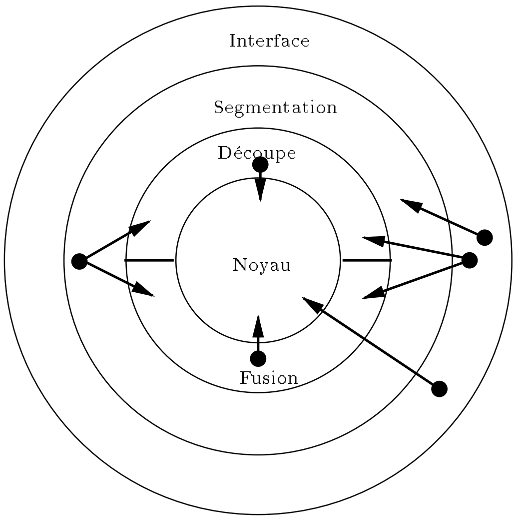



Figure 7.1: *Structure de l'application.*

La couche "noyau" gère la structure de données décrite dans la section 6.1. Elle gère également les insertions et suppressions de segments ( voir sections 6.2, 6.3 et 6.4).

Un certain nombre de fonctionnalités ont été prévues dans et autour du noyau afin de faciliter la conception et le développement d'algorithmes de segmentation. Nous allons étudier ces fonctionnalités en précisant les conventions et notations utilisées dans notre application.

### 7.1.1 La couche "découpe"

Les algorithmes de découpe prennent une face $f$ en paramètre et retournent un ensemble de faces correspondant à la découpe de la région $R_f$. Un algorithme de découpe se décompose en trois phases : initialisation de l'algorithme de découpe, génération des segments, fermeture de l'algorithme.

#### 7.1.1.1 Initialisation de l'algorithme de découpe

Cette première étape consiste à extraire un ensemble d'attributs de la région $R_f$. On peut par exemple calculer le multi-ensemble associé à la région, la couleur moyenne de la région ou tout autre paramètre utile à la segmentation. Cette opération est réalisée grâce à la fonction du noyau : `contour_set_apply_function`. Cette fonction prend une fonction fournie par l'utilisateur en paramètre et l'applique à l'ensemble des pixels de la région $R_f$.

#### 7.1.1.2 L'algorithme de découpe

Cette étape utilise les paramètres définis à l'étape précédente pour générer un ensemble de segments partitionant la région $R_f$. Si les paramètres issus de l'étape d'initialisation permettent de définir une partition implicite de la région $R_f$ une fonctionnalité du noyau permet de générer automatiquement l'ensemble des segments correspondant à cette partition. Cette fonctionnalité est réalisée grâce à la fonction `contour_set_set_same_cluster_criterion`. Cette fonction utilise une fonction booléenne fournie par l'utilisateur. Cette fonction booléenne prend en paramètre la position et la couleur de deux pixels adjacents et renvoie un booléen indiquant si les deux pixels appartiennent ou non à la même région. L'ensemble des segments est donc généré automatiquement à partir de cette fonction.

#### 7.1.1.3 Fermeture d'un algorithme de découpe

Cette étape permet de libérer l'ensemble des données allouées lors de l'étape d'initialisation. En effet, les données devant être utilisées lors de l'étape de découpe proprement dite, ne peuvent être libérées en fin d'initialisation. L'étape de fermeture qui suit la génération des segments correspondant à la partition implicite de $R_f$ permet de libérer ces données.

#### 7.1.1.4 Conclusion

Les algorithmes de découpe se décomposent donc généralement en trois parties. Cette décomposition et les fonctionnalités du logiciel qui l'accompagnent s'adaptent particulièrement bien aux algorithmes de découpe basés sur une partition du multi-ensemble. Dans ce cas l'on dira que deux pixels appartiennent à des régions différentes si leurs couleurs appartiennent à des multi-ensembles différents. Notre expérience nous a montré que l'ensemble des fonctionnalités fournies par le logiciel permettent d'intégrer un algorithme de découpe globale (voir chapitre 5) en quelques heures voire quelques minutes. Les fonctions du noyau `contour_set_memorize_inserted_faces` et `contour_set_give_inserted_faces` permettent alors d'obtenir l'ensemble des faces créées par un algorithme de découpe.

### 7.1.2 La couche "fusion"

Les algorithmes de fusion travaillent sur un ensemble de faces $\{f_1, \ldots, f_n\}$. A chaque étape d'un algorithme de fusion deux faces $f_i$ et $f_j$ sont fusionnées. Dans sa forme la plus générale l'algorithme de fusion impose qu'au moins une des deux faces $f_i$ ou $f_j$ appartienne à $\{f_1, \ldots, f_n\}$. Généralement les fusions sont restreintes à l'ensemble $\{f_1, \ldots, f_n\}$. Les deux faces $f_i$ et $f_j$ appartiennent dans ce cas à $\{f_1, \ldots, f_n\}$. Nous verrons dans la section 7.4 l'intérêt de cette généralisation. Les algorithmes de fusion utilisent deux types de voisinage définis à partir des cartes planaires :

**Définition 47** *Soit deux faces $f_1$ et $f_2$ d'étiquettes $l_1$ et $l_2$, on dira que $f_1$ est un* **voisin extérieur** *de $f_2$ si et seulement si :*

$$\exists b \in \varphi^*(\lambda^{-1}(l_2)) \ / \ -b \in \varphi^*(\lambda^{-1}(l_1))$$

*La relation de voisinage extérieur notée $\mathcal{R}_{ext}$ permet de définir le voisinage exterieur d'une face $f$. Ce voisinage noté $\mathcal{V}_{ext}$ est défini par :*

$$\mathcal{V}_{ext}(f) = \{f' \in \boldsymbol{\mathcal{F}} \ / \ f' \ \mathcal{R}_{ext} \ f\}$$

Autrement dit, deux faces sont voisines extérieures si elles partagent une arête $(b, -b)$. Un exemple de voisinage extérieur est représenté sur la figure 7.2.

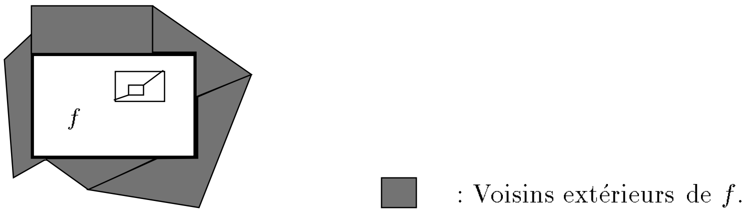



Figure 7.2: *Un exemple de voisinage extérieur.*

La définition 47 ne permet pas de tenir compte des faces incluses. Ainsi dans l'exemple de la figure 7.2 aucune des faces définissant le carré inclu dans la face $f$ n'est voisine de $f$. La définition suivante généralise la notion de voisinage et permet de résoudre ce problème.

**Définition 48** *Soit deux faces $f_1$ et $f_2$ appartenant à deux cartes $(\varphi_1, \alpha_1)$ et $(\varphi_2, \alpha_2)$, $f_1$ sera dit* **voisin** *de $f_2$ si et seulement si :*

- *$(\varphi_1, \alpha_1) = (\varphi_2, \alpha 2)$ et $f_1$ est un voisin extérieur de $f_2$ ou*

- *$f_1$ est un voisin extérieur de $f^{\infty}_{\varphi_1}$ et* **mère**$(f^{\infty}_{\varphi_1}) = f_2$ *ou*
- *$f_2$ est un voisin extérieur de $f^{\infty}_{\varphi_2}$ et* **mère**$(f^{\infty}_{\varphi_2}) = f_1$.

*Le voisinage d'une face $f$ induit par cette relation sera appelé le voisinage général de $f$ et noté $\mathcal{V}_g(f)$. La relation associée est notée $\mathcal{R}_g$ .*

Le voisinage général d'une face $f$ est donc égal à son voisinage extérieur auquel on rajoute l'ensemble des voisinages extérieurs de ses filles. Le voisinage extérieur et le voisinage général sont obtenus grâce aux fonctions `contour_set_complete_face_adjacence` et `contour_set_give_included_faces`.

Un exemple de voisinage général est représenté sur la figure 7.3.

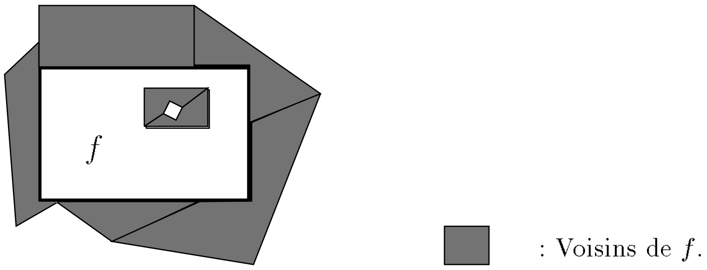



Figure 7.3: *Une exemple de voisinage général.*

**Remarque 15** *Les voisinages extérieurs et généralisés ne suffisent pas à eux seul à définir une topologie sur $\mathcal{F}$. Toutefois, si nous associons à chaque face $f$, les ensembles d'ensemble :*

$$\begin{cases} \mathcal{B}_{ext}(f) &= \{\{f\}, \{W_f \cup \{f\} \text{ avec } \forall f' \in W_f \ f' \ \mathcal{R}_{ext} \ f\}\} \\ \mathcal{B}_g(f) &= \{\{f\}, \{W_f \cup \{f\} \text{ avec } \forall f' \in W_f \ f' \ \mathcal{R}_g \ f\}\} \end{cases}$$

*nous définissons deux ensembles de voisinages fondamentaux, induisant deux topologies sur $f$. On a alors :*

$$\forall f \in \mathcal{F} \begin{cases} \forall\ U \in \mathcal{B}_{ext}(f) & U - \{f\} \ \subset \ \mathcal{V}_{ext}(f) \\ \forall\ U \in \mathcal{B}_g(f) & U - \{f\} \ \subset \ \mathcal{V}_g(f) \end{cases}$$

*Les voisinages extérieurs et généralisés sont en quelque sorte les plus gros voisinages de leurs topologies respectives.*

Si nous désignons par $S(f_1, f_2)$ un critère de similarité entre les faces $f_1$ et $f_2$ (voir section 5.2.3) et par $V(f)$ le voisinage (extérieur ou généralisé) d'une face $f$ le schéma général d'un algorithme de fusion est donné par l'algorithme 7.4.

L'algorithme `fusionne_deux_faces` mentionné dans l'algorithme `fusion` (voir Figure 7.4) réalise la fusion effective des faces $f$ et $f'$ en enlevant les segments qui définissent la frontière entre les régions $R_f$ et $R_{f'}$. Si $V(f)$ désigne le voisinage général de $f$, ces segments

```
fusion(liste de faces L)
{

    faire
    {
        trouver f et f' telles que :
            S(f,f') = Min_{f1∈L} Min_{f2∈V(f1)} S(f1,f2)

        si(S(f,f') > ε)
            terminer l'algorithme

        f∪f' = fusionne_deux_faces(f,f')
        enlever f et f' de L
        inserer f∪f' dans L
    }
    tant que (vrai)
}
```

Figure 7.4: *Schéma général d'un algorithme de fusion.*

correspondent aux brins $b$ tels que :

$$\begin{cases} \lambda(b) = l \text{ et } \lambda(-b) = l' \text{ ou} \\ \lambda(b) = l \text{ et } \mathbf{m\grave{e}re}(\lambda(-b)) = l' \text{ ou} \\ \lambda(b) = l' \text{ et } \mathbf{m\grave{e}re}(\lambda(-b)) = l \end{cases} \tag{7.1}$$

où $l$ et $l'$ correspondent aux étiquettes des faces $f$ et $f'$.

Une fonctionnalité de notre logiciel permet de déterminer cet ensemble de brins. Ceux-ci sont ensuite supprimés grâce aux méthodes vues dans la section 6.4. Si $f$ et $f'$ sont les deux faces fusionnées, l'étiquette de la face correspondant à la nouvelle région $R_f \cup R_{f'}$ se déduit très facilement des étiquettes de $f$ et $f'$ grâce à la proposition suivante.

**Proposition 8** *Soit deux faces finies et voisines d'étiquettes $l$ et $l'$. La suppression des brins vérifiant l'équation 7.1 crée une nouvelle face finie d'étiquette $l''$ avec :*

$$l'' \in \{l, l'\}$$

*De plus, si $l'' = l$ alors*

- *soit $l'$ ne correspond plus à un cycle de $\varphi$, on dit dans ce cas qu'il est indéfini,*
- *soit $l'$ correspond au cycle d'une face infinie, on dit dans ce cas qu'il est infini.*

*Inversement, nous dirons que $l''$ est fini et défini.*

Cette proposition se vérifie facilement en examinant la mise à jour de la fonction $\lambda$ lors des effacements de brins détaillés dans la section 6.4. On constate en effet que chaque fois que la suppression d'un brin implique la fusion de deux faces l'étiquette de la nouvelle face est égale à une des étiquettes des deux faces fusionnées.

L'étiquette de la nouvelle face est donc calculé rapidement en déterminant la face d'étiquette infini ou indéfini parmis les deux faces fusionnées.

**Remarque 16** *Cette proposition s'étend très facilement par récurrence à la fusion d'un ensemble de faces finies d'étiquettes* $\{l_1, \ldots, l_n\}$. *Dans ce cas seule une des étiquettes* $l_i$ *reste définie et finie.*

Du fait de la définition des voisinages extérieur et général, nous avons la propriété suivante :

$$\forall (f, f') \in \boldsymbol{\mathcal{F}}^2 \quad f \in V(f') \Leftrightarrow f' \in V(f)$$

où $V(f)$ désigne le voisinage général ou extérieur de la face $f$.

Bien qu'intellectuellement satisfaisante cette symétrie des voisinages présente un inconvénient pour les algorithmes de fusions. En effet si l'algorithme de fusion est lancé sur l'ensemble $L = \{f_1, f_2, \ldots, f_n\}$ et si $f_1 \in V(f_2)$ le couple $f_1$, $f_2$ sera considéré deux fois. Une fois lorsque l'on considèrera les voisins de $f_1$ et une autre fois lorsque l'on déterminera les voisins de $f_2$. On peut remédier à ce problème en considérant la définition de voisinage suivante :

**Définition 49** *Soient deux faces* $f_1$ *et* $f_2$ *d'étiquettes* $l_1$ *et* $l_2$, *on dira que* $f_1$ *est le voisin restreint de* $f_2$ *si et seulement si :*

$$\exists b \in \varphi^*(\lambda^{-1}(l_1)) \ / \ \left\{ \begin{array}{l} b > 0 \ et \ -b \in \varphi^*(\lambda^{-1}(l_2)) \\ ou \\ \textbf{mère}(\lambda(-b)) = \lambda(l_2) \end{array} \right.$$

*Le voisinage restreint d'une face* $f$ *est noté* $\mathcal{V}_{res}(f)$

Un exemple de voisinage restreint est représenté sur la figure 7.5.

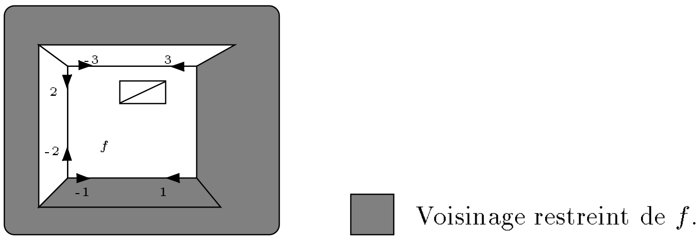



Figure 7.5: *Un exemple de voisinage restreint.*

On vérifie facilement que le voisinage restreint d'une face $f$ vérifie les propriétés suivante :

$$\begin{array}{llll} \forall f \in \boldsymbol{\mathcal{F}} & f' \in \mathcal{V}_{res}(f) & \Rightarrow & f \notin \mathcal{V}_{res}(f') \\ \forall f, f' \in \boldsymbol{\mathcal{F}} & f \ \mathcal{R}_g \ f' & \Leftrightarrow & f' \in V_{res}(f) \text{ ou } f \in V_{res}(f') \end{array}$$

La propriété 7.1.2 nous assure qu'un algorithme de fusion utilisant le voisinage restreint n'envisagera pas deux fois le même couple de faces. La propriété 7.1.2 permet d'assurer qu'un algorithme de fusion avec voisinage restreint se restreignant à des fusions dans $\{f_1, \ldots, f_n\}$ examinera tous les couples de faces $(f_i, f_j)$ tels que $f_i \mathcal{R}_g f_j$. En effet, si l'on a $f_i \mathcal{R}_g f_j$ on a alors $f_i \in V_{res}(f_j)$ ou $f_j \in V_{res}(f_i)$ le couple $(f_i, f_j)$ sera donc examiné lors de l'évaluation des voisins restreints de $f_i$ ou de $f_j$. Cet algorithme obtiendra donc les même résultat qu'avec le voisinage général tout en examinant chaque couple $(f_i, f_j)$ vérifiant $f_i \mathcal{R}_g f_j$ une fois et une seule. En revanche si l'on ne restreint pas les fusions à $\{f_1, \ldots, f_n\}$ l'algorithme ne considèrera pas les faces $f$ vérifiant la propriété suivante:

$$\left\{ \begin{array}{ll} \exists i \in \{1, \ldots, n\} & / \; f_i \in \mathcal{V}_{res}(f) \\ \text{et} & \\ \forall j \in \{1, \ldots, n\} & f \notin \mathcal{V}_{res}(f_j) \end{array} \right.$$

Ceci peut amener l'algorithme de fusion à conserver des faces non significatives, qui ne pourront être fusionnées avec aucune des faces $\{f_1, \ldots, f_n\}$. Une telle face sera considérée si l'algorithme de fusion utilise le voisinage général.

### 7.1.3 La couche "segmentation"

Dans le contexte de notre application les algorithmes de segmentation désignent des algorithmes de plus haut niveau que les algorithmes de découpe ou de fusion. Un algorithme de découpe se limite à générer un ensemble de faces à partir d'une face passée en paramètre. Inversement, un algorithme de fusion fusionne un ensemble de faces $\{f_1, \ldots, f_n\}$ pour générer $p$ faces avec $p \leq n$. Par contre un algorithme de segmentation a pour objectif d'enchainer de façon adaptative une suite de découpe et/ou de fusions. Par exemple, un algorithme de division récursive (voir chapitre 5) peut être considéré comme un algorithme de segmentation puisqu'il a pour objectif d'enchainer de façon récursive une suite de découpes. Un algorithme de segmentation doit être capable de sélectionner les faces à découper ou fusionner ainsi que de choisir les algorithmes de découpe ou de fusion les mieux adaptés. La plupart des algorithmes de segmentation existant utilisent un seul type d'algorithme de division et de fusion. Notre structure de données permet par sa souplesse d'utiliser plusieurs algorithmes de découpe et fusion permettant de s'adapter aux différentes situations. Ce "pilotage" des algorithmes de découpe ou de fusion peut être effectué par la couche segmentation ou directement par l'utilisateur à partir de la couche "interface".

### 7.1.4 La couche "interface"

La couche interface permet de diriger les algorithmes de segmentation, de découpe et de fusion. On peut, par exemple, désigner interactivement une région de l'image, la segmenter, la fusionner avec une région adjacente, supprimer un ensemble de faces, etc. L'intérêt de cette couche est double. Tout d'abord le fait de diriger la segmentation permet de concentrer les algorithmes de segmentation sur les régions intéressantes pour l'utilisateur. Ceci permet un gain de temps appréciable sur de grosses images. De plus, la fusion et la découpe interactive des faces permettent à l'utilisateur d'obtenir la segmentation désirée.

En effet, pour une même image deux utilisateurs différents peuvent s'intéresser à des détails différents et un algorithme de segmentation fournissant tous les détails suceptibles d'être demandés par les utilisateurs produira obligatoirement une sur-segmentation. Éditer une image sur-segmentée pour enlever les détails non désirés est un travail fastidieux. Notre application permet de réduire ou de supprimer cette étape.

### 7.1.5 Attributs attachés à une face et un segment

L'ensemble des attributs affectés aux régions d'une image peut se décomposer en trois grandes catégories (voir [CP95] pour plus de détails) :

- Les attributs géométriques de la région (périmètre, courbures, surface, parallélograme englobant, ...).
- Les attributs du multi-ensemble associé à la région (moyenne, variance, texture, ...).
- Les attributs faisant intervenir la valeur et la position des pixels (gradient moyen le long d'un segment, courbure surfacique d'une région, ...).

Les attributs géométriques sont souvent utilisés dans des étapes postérieures à la segmentation pour reconnaitre des régions de forme caractéristique. Les informations de surface et de courbure peuvent être utilisées pour identifier une région, par exemple la région correspondant au coeur sur une image IRM.

Les attributs du multi-ensemble associé à une région sont utilisés pour la reconnaissance de forme et pour la segmentation. On peut par exemple chercher une région correspondant à un objet de couleur caractéristique, fusionner deux régions de couleur moyenne proche ou découper une région si son erreur quadratique est trop importante.

Les attributs faisant intervenir les valeurs des pixels et leurs positions sont souvent utilisés pour la segmentation. Le gradient moyen le long d'un segment peut par exemple être utilisé pour fusionner des régions possédant un faible gradient le long de leurs frontières communes (voir par exemple Beveridge [BGK+89] section 5.2.3.2, ou Brice et Fennema [BF70] section 5.2.3.3).

L'ensemble des attributs pouvant être attachés à une région ou à un segment est donc très important. Chaque application sélectionne un ensemble d'attributs en fonction de ses objectifs. Nous avons donc attaché aux régions et segments les attributs les plus couramment utilisés tout en prévoyant des fonctionnalités permettant de calculer facilement les attributs d'une région ou d'un segment.

Les attributs attachés à une région sont les moments $\boldsymbol{M}_0$, $\boldsymbol{M}_1$, $\boldsymbol{M}_2$ du multi-ensemble associé à la région. Le moment $\boldsymbol{M}_0$ permet de connaitre le nombre de pixels de la région. Les moments d'ordre un et deux permettent de calculer la couleur moyenne, la variance ou l'erreur quadratique du multi-ensemble associé à une région (voir définitions 3 et 4). Soit $(C_{R_f}, f_{R_f})$ le multi-ensemble associé à la région $R_f$, afin de simplifier les notations nous adopterons les conventions suivantes.

- Les moments $\boldsymbol{M}_i((C_{R_f}, f_{R_f}))$ de la région $R_f$ seront notés $\boldsymbol{M}_i(R_f)$.
- La couleur moyenne de la région $\mu((C_{R_f}, f_{R_f}))$ sera notée $\mu(R_f)$
- L'erreur quadratique $\boldsymbol{SE}((C_{R_f}, f_{R_f}))$ sera notée $\boldsymbol{SE}(R_f)$

Les attributs attachés à un segment sont :

- Sa longueur définie par le nombre de déplacements du segment (nombre de points du segment moins un). La longueur d'un segment $s$ est notée $|s|$.
- La somme des distances entre les couleurs des pixels de part et d'autre du segment.

Cette somme de distances entre couleurs peut se formaliser comme suit :

**Définition 50** *Soit $s$ un segment défini par un point $P_0$ et une suite de déplacements $(d_0, \ldots, d_n)$. On défini la suite de points $(P_0, P_1, \ldots, P_{n+1})$ telle que $P_{i+1}$ est le voisin de $P_i$ dans la direction $d_i$. A chaque couple $(P_i, d_i)$ correspondent deux pixels $C_i^1$, $C_i^2$ demi-voisins de $P_i$ et $P_{i+1}$ (voir Figure 7.6). Les suites $(C_0^1, \ldots, C_n^1)$ et $(C_0^2, \ldots, C_n^2)$ permettent de définir le gradient d'un segment de la façon suivante :*

$$grad(s) = \sum_{i=1}^{n} d(I(C_i^1), I(C_i^2))$$

*où $I(C_i^j)$ représente la valeur du pixel $C_i^j$ et $d(.,.)$ la distance euclidienne définie dans le multi-ensemble associé à l'image.*

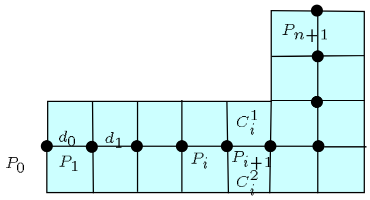



Figure 7.6: *Calcul de la somme des distances entre les pixels de part et d'autre du segment.*

Le gradient d'un segment et sa longueur permettent de calculer son gradient moyen défini par :

$$\overline{grad}(s) = \frac{grad(s)}{|s|}$$

Afin de limiter le nombre de calculs, les attributs d'un segment ou d'une région ne sont calculés qu'en réponse à une requête explicite de l'utilisateur ou d'un algorithme. L'attribut résultant de ce calcul est alors conservé dans la face ou le segment afin d'éviter une redondance de calcul lors des prochaines requêtes. Cependant si un segment ou une région conservant des attributs déjà calculés est découpé, les attributs des nouvelles régions ou des nouveaux segments engendrés ne seront calculés qu'à la suite d'une requête. En revanche la concaténation de deux segments (voir section 6.2) ou la fusion de deux régions (voir section 7.1.2) entraine une mise à jour immédiate des attributs si ceux-ci avaient déjà été calculés.

Si nous désignons par $s_1$ et $s_2$ les deux segments concaténés et par $s_1 \oplus s_2$ le segment résultant, on a :

$$|s_1 \oplus s_2| = |s_1| + |s_2|$$
$$grad(s_1 \oplus s_2) = grad(s_1) + grad(s_2)$$

De même, si $R_{f_1 \cup f_2}$ désigne la région résultant de la fusion des régions $R_{f_1}$ et $R_{f_2}$ on a :

$$\forall i \in \{0,1,2\} \quad \boldsymbol{M}_i(R_{f_1 \cup f_2}) = \boldsymbol{M}_i(R_{f_1}) + \boldsymbol{M}_i(R_{f_2})$$

Ce dernier résultat correspond à la transcription de la proposition 1 avec nos nouvelles notations.

La mise à jour des attributs des segments et des régions est donc extrêmement rapide. Cette mise à jour est effectuée seulement si les attributs des deux segments concaténés ou des deux faces fusionnées sont valides. Dans le cas contraire, si un des deux segments ou une des deux faces possède des attributs invalides, les attributs du segment ou de la face résultante seront également marqués invalides.

Les attributs que nous avons attachés aux faces et segments ne permettent d'obtenir qu'un nombre restreint d'attributs. Si l'utilisateur désire calculer ses propres attributs de région ou de segment il peut :

- Passer une fonction au noyau qui sera appliquée à tous les pixels d'une face (voir `contour_set_apply_function` section 7.1.1)
- Obtenir l'ensemble des points composant un segment.

Ces deux fonctionnalités permettent à l'utilisateur de calculer facilement ses propres attributs de segments ou de régions.

## 7.2 Algorithme de découpe

Notre algorithme de découpe utilise la décomposition en trois phases décrite dans la section 7.1.1. Il est basé sur l'algorithme de quantification `découpe_fusion` vu dans la section 3.8. Cet algorithme réalise une partition uniforme du multi-ensemle $(C_{R_f}, f_{R_f})$ associé à la région $R_f$ puis fusionne ces multi-ensembles pour obtenir les $K$ multi-ensembles finaux.

### 7.2.1 Choix du cardinal de la partition

Dans le cadre de la quantification la valeur $K$ est fixée par l'utilisateur. En revanche pour un algorithme de segmentation cette valeur doit être déterminée automatiquement. Intuitivement la valeur optimum de $K$ est la valeur nous permettant de générer le plus grand nombre de régions valides, une région valide étant une région uniforme de taille maximale (voir définition 22). Chaque multi-ensemble formant la partition induit la création de une ou plusieurs régions, donc si nous augmentons le nombre de multi-ensembles finaux $K$ nous augmentons le nombre de régions. En revanche, plus nous augmentons le nombre de multi-ensembles, plus nous augmentons le risque de créer un multi-ensemble induisant des faces non significatives. La valeur optimum de $K$ établit donc un compromis entre le nombre de régions significatives et le risque de création de faces non-significatives. Nous avons vu dans la section 5.2.4.2 que la fonction $\beta$ permettait d'obtenir une valeur de $K$ réalisant un tel compromis. On a en effet :

$$\beta(p) = E_p(C_{R_f})\Delta(p) \text{ avec } \Delta(p) = \sum_{i=1}^{p} \|\mu_k - \mu_0\|^2$$

Maximiser l'erreur de partition $E_p(C_{R_f})$ tend à créer peu de multi-ensembles. Inversement, maximiser $\Delta(p)$ tend à maximiser le nombre de multi-ensembles. La valeur optimum de $\beta$ fournit un compromis entre ces deux tendances contradictoires.

L'approche que nous avons choisie est légèrement différente. Plutôt que d'utiliser un seul critère nous créons tout d'abord la plus grande partition dont l'erreur est inférieure à un certain pourcentage $\alpha_1$ de l'erreur quadratique du multi-ensemble de départ. Cette première partition vérifie donc :

$$\boldsymbol{E}_p(C_{R_f}) < \alpha_1 \boldsymbol{SE}(C_{R_f}) \tag{7.2}$$

Partant de cette partition nous fusionnons ensuite les multi-ensembles $C_i$ et $C_j$ tels que :

$$\|\mu(C_i) - \mu(C_j)\|^2 \leq \alpha_2.var$$

où $\alpha_2$ représente un pourcentage fixé par l'utilisateur et $var$ la variance des moyennes $\mu(C_i)$ définie par :

$$var = \frac{1}{p}\sum_{i=1}^{p} \|\mu(C_i) - \mu_0\|^2 \text{ avec } \mu_0 = \frac{1}{p}\sum_{i=1}^{p} \mu(C_i)$$

La partition finale $\{(C_1, f), \ldots, (C_K, f)\}$ vérifie donc :

$$\forall i, j \in \{1, \ldots, K\}^2 \quad \| \mu(C_i) - \mu(C_j)\|^2 > \alpha_2.var \tag{7.3}$$

La fusion des multi-ensembles de moyenne proche est effectuée après la création de la partition. On ne peut donc, en toute rigueur, supposer que l'on a simultanément les conditions 7.2 et 7.3. Toutefois, du fait du choix des paramètres $\alpha_1$ et $\alpha_2$, la fusion des multi-ensembles de moyennes proches est rarement effectuée. De plus cette fusion modifie généralement assez peu l'erreur de partition finale. On supposera donc que les deux conditions 7.2 et 7.3 sont simultanément vraies.

La condition 7.3 exprime une notion de maximalité similaire à la condition (4) de la définition 22. Elle permet de ne pas générer de faces voisines de couleurs moyennes ou de niveaux de gris trop proches. Ceci permet de traiter quelques cas simples de texture [Gag83, Gag87]. En effet les textures d'une image naturelle de type mur ou herbe sont souvent composées de couleurs proches. La condition 7.3 permet de les regrouper en un seul multi-ensemble.

Les paramètres $\alpha_1$ et $\alpha_2$ permettent à l'utilisateur de contrôler le nombre de faces générées par l'algorithme de découpe. On souhaite dans un soucis de simplicité que l'utilisateur n'ait à manipuler qu'un seul paramètre. L'expérience nous a conduit à poser :

$$\alpha = \alpha_1 = \frac{K-1}{2K}\alpha_2$$

où $K$ désigne le nombre de multi-ensembles formant la partition. La valeur par défaut de $\alpha$ est égale à 30%. Cette valeur a donnée de bons résultats sur nos images tests. Sur quelques images des valeurs de $\alpha$ inférieures ( de l'ordre de 15 à 20%) permettent d'obtenir un nombre plus important de détails tout en conservant des faces valides.

Nous avons comparé la valeur de $K$ fournie par les conditions 7.2 et 7.3 à celle fournie par le critère $\beta$. Les deux critères ont fourni des valeurs approximativement équivalentes sur l'ensemble des tests effectués. Les valeurs de $K$ fournies par ces deux critères sont généralement égale à 2,3 ou 4.

### 7.2.2 Déroulement de l'algorithme de découpe

Notre algorithme utilise donc la fonction `contour_set_apply_fonction` pour calculer le multi-ensemble associé à la région et découper celui-ci en $N$ multi-ensembles. La valeur de $N$ est fixée à 2048. Nous avons vu dans la section 3.8 que cette valeur permet d'obtenir une bonne partition lorsque le cardinal de celle ci est peu élevé ( inférieur à 16).

La création du multi-ensemble, sa partition et les fusions de multi-ensembles sont effectuées durant la phase d'initialisation de l'algorithme de découpe. La partition résultant de cette première étape permet de définir la fonction d'inversion de table de couleur $\boldsymbol{Q}$ (voir sections 3.4 et 3.8.3). On utilise ensuite la fonction `contour_set_same_cluster_criterion` en définissant une fonction qui renvoie vrai si les couleur des deux pixels passés en paramètre possèdent la même image par la fonction $\boldsymbol{Q}$, donc si :

$$\boldsymbol{Q}(I(P_1)) = \boldsymbol{Q}(I(P_2))$$

où $P_1$ et $P_2$ sont les deux pixels passés en paramètre et $I(P_i)$ la couleur du pixel $P_i$. L'ensemble des segments correspondant à la partition implicite définie par la fonction $\boldsymbol{Q}$ est alors généré automatiquement.

La phase de fermeture de l'algorithme de découpe permet de libérer le tableau de multi-ensembles généré lors de l'initialisation. La matrice stockant le **GAM** (voir section 3.8.1) a quant à elle été libérée à la fin de l'étape d'initialisation.

### 7.2.3 Caractérisation des régions générées par l'algorithme de découpe

Soit $\{(C_1, f), \ldots, (C_K, f)\}$ la partition du multi ensemble $(C_{R_f}, f_{R_f})$ associée à la région $R_f$. Nous pouvons associer à chaque multi-ensemble $(C_i, f)$ les régions $\{R_{f_i^1}, \ldots, R_{f_i^p}\}$ issues de la décomposition 4-connexe de $\boldsymbol{Q}^{-1}((C_i, f))$ (voir définition 17). Si $(C_{R_{f_i^j}}, f_{R_{f_i^j}})$

désigne le multi-ensemble associé à la région $R_{f_i^j}$, on a assez trivialement :

$$(C_i, f) = \sum_{j=1}^{p} (C_{R_{f_i^j}}, f_{R_{f_i^j}})$$

Donc :

$$\forall j \in \{1, \ldots, p\} \ \boldsymbol{SE}((C_{R_{f_i^j}}, f_{R_{f_i^j}})) \leq \boldsymbol{SE}((C_i, f))$$

Si de plus l'on a (voir équation 7.2) :

$$\boldsymbol{E}_K(C) = \sum_{i=1}^{K} \boldsymbol{SE}((C_i, f)) \leq \alpha_1 \boldsymbol{SE}((C_{R_f}, f_{R_f}))$$

On a alors :

$$\forall j \in \{1, \ldots, p\} \ \boldsymbol{SE}((C_{R_{f_i^j}}, f_{R_{f_i^j}})) \leq \alpha_1 \boldsymbol{SE}((C_{R_f}, f)) \tag{7.4}$$

L'erreur quadratique des faces générées est donc inférieure à $\alpha_1 \boldsymbol{SE}((C_{R_f}, f_{R_f}))$. Ceci correspond intuitivement au fait que les régions découpées sont plus homogène que la région initiale. Le paramètre $\alpha_1$ permet de contrôler cette homogénéité.

Un exemple de découpe est représenté sur la figure 7.7 où l'image Lenna à été découpée en un ensemble de faces vérifiant l'équation 7.4. On peut remarquer l'existence de nombreuses petites faces non significatives. Celles-ci seront supprimées par l'algorithme de fusion.

## 7.3 Algorithme de fusion

### 7.3.1 Un premier critère de fusion

Notre algorithme de fusion est basé sur l'erreur quadratique. Partant d'un ensemble de faces $\{f_1, \ldots, f_n\}$ nous fusionnons à chaque étape les deux faces $f$, $f'$ dont la fusion créera la face la plus homogène parmi toutes les fusions envisagées. Une première approche consiste à définir $f$ et $f'$ de la façon suivante :

$$(f, f') = ArgMin_{i \in \{1, \ldots, n\}} Min_{f'' \in \mathcal{V}_g(f_i)} \boldsymbol{SE}(R_{f_i} \cup R_{f''}) \tag{7.5}$$

où $ArgMin$ est un argument réalisant le minimum.

Il est important de remarquer que seule une des deux faces fusionnées appartient obligatoirement à $\{f_1, \ldots, f_n\}$. Si par exemple, $f$ appartient à $\{f_1, \ldots, f_n\}$ alors que $f'$ n'y appartient pas la face fusionnée $f \cup f'$ est substituée à $f$ dans l'ensemble $\{f_1, \ldots, f_n\}$. La face $f \cup f'$ peut donc être à nouveau fusionnée avec les faces de son voisinage. Ce phénomène agrandi le domaine $\cup_{i=1}^{n} R_{f_i}$. Ce domaine peut se voir comme le domaine "d'intervention" de l'algorithme de fusion. On peut limiter celui-ci en marquant l'ensemble des faces suceptibles d'êtres fusionnés. L'équation 7.5 devient alors :

$$(f, f') = ArgMin_{i \in \{1, \ldots, n\}} Min_{f'' \in \mathcal{V}_g(f_i) \ / \ \text{marqué}(f'')} \boldsymbol{SE}(R_{f_i} \cup R_{f''})$$

On dit dans ce cas que l'algorithme de fusion est **restreint** à l'ensemble des faces marquées.

On a d'après le théorème 1 :

$$\boldsymbol{SE}(R_{f\cup f'}) = \boldsymbol{SE}(R_f)+\boldsymbol{SE}(R_{f'})+\Delta(f,f') \text{ avec } \Delta(f,f') = \frac{|R_f||R_{f'}|}{|R_f|+|R_{f'}|}\|\mu(R_f)-\mu(R_{f'})\|^2$$

Donc,

$$\boldsymbol{SE}(R_{f\cup f'}) \geq \boldsymbol{SE}(R_f) \text{ et } \boldsymbol{SE}(R_{f\cup f'}) \geq \boldsymbol{SE}(R_{f'})$$

Donc la fusion augmente l'inhomogénéité des faces. Le processus de fusion se poursuit jusqu'à ce que :

$$Min_{i\in\{1,\dots,n\}} Min_{f''\in\mathcal{V}_g(f_i)} \boldsymbol{SE}(R_{f_i}\cup R_{f''}) \geq \alpha' \boldsymbol{SE}(R_{f_0})$$

où $\alpha'$ est une constante fixée par l'utilisateur et $R_{f_0}$ une face de référence. Nous détaillerons le choix de cette face dans la section 7.4.

Intuitivement, le processus de fusion augmente l'inhomogénéité des faces jusqu'a ce que celle ci dépasse une certaine limite.

### 7.3.2 Amélioration du critère

La méthode ci-dessus présente un inconvénient lorsque toutes les fusions créent des faces d'erreur quadratique proche de la limite $\alpha' \boldsymbol{SE}(R_{f_0})$. En effet, si nous considérons une petite face homogène mais non significative inclue dans une face d'erreur quadratique proche de la limite. La fusion entre ces deux faces ne pourra être effectuée si l'augmentation de l'erreur, bien que minime, dépasse la limite. L'algorithme de fusion conservera donc des faces non significatives. Intuitivement, l'on voudrait pouvoir dire que si une des deux faces contribue peu à l'erreur quadratique de la face fusionnée, on doit fusionner ces deux faces même si l'erreur quadratique finale dépasse la limite. L'information pertinente est donc plus la contribution de chaque face à l'erreur quadratique finale que l'erreur en elle même. Nous avons donc remplacé le critère $\boldsymbol{SE}(R_{f\cup f'}) = \boldsymbol{SE}(R_f) + \boldsymbol{SE}(R_{f'}) + \Delta(f,f')$ par :

$$E'(R_f, R'_f) = Min(\boldsymbol{SE}(R_f), \boldsymbol{SE}(R_{f'})) + \Delta(f,f')$$

En effet, la valeur $\boldsymbol{SE}(R_f) + \Delta(f,f')$ peut se voir comme l'augmentation de l'erreur quadratique de $R_{f'}$ provoquée par la fusion avec $R_f$. Inversement $\boldsymbol{SE}(R_{f'}) + \Delta(f,f')$ définit l'augmentation de l'erreur de $R_f$ provoquée par la fusion avec $R_{f'}$. La fusion sera donc effectuée si la contribution d'une des deux régions à l'erreur finale est marginale.

Les deux faces fusionnées à chaque étape sont donc définie par :

$$(f,f') = ArgMin_{i\in\{1,\dots,n\}} Min_{f''\in\mathcal{V}_g(f_i)} E'(R_{f_i}, R_{f''})$$

L'ensemble du processus de fusion se déroule alors comme décrit précédement avec le nouveau critère $E'(R_f, R_{f'})$.

Un exemple de fusion utilisant ce nouveau critère est présenté sur la figure 7.8. L'ensemble des faces fusionnées et représenté sur la figure 7.7.

### 7.3.3 Comparaison de notre critère et de celui de Beveridge

Le critère de Beveridge [BGK+89] (voir section 5.2.3.2) utilise la taille, la couleur moyenne et la frontière de chaque région. Le critère $E'$ permet de tenir compte de la taille, de la couleur moyenne et de l'homogénéité des régions fusionnées. En effet, le paramètre $\Delta$ peut se voir comme la distance entre les couleurs moyennes des deux régions pondérée par la taille de celles ci. Le paramètre $Min(\boldsymbol{SE}(R_f), \boldsymbol{SE}(R_{f'}))$ permet d'assurer que les deux régions fusionnées resteront homogènes. Nous avons préféré ne pas intégrer la frontière des régions dans le critère de fusion. En effet, la taille, l'homogénéité, ou la couleur moyenne d'une région sont des paramètres globaux faisant intervenir l'ensemble des pixels de la région. La longueur ou le gradient moyen de la frontière séparant deux régions est en revanche un paramètre plus local. Nous avons donc décidé de séparer le traitement de ces deux types d'information. Les paramètres globaux sont utilisés dans le critère de fusion. Les paramètres locaux comme le gradient moyen de deux régions peuvent être utilisés après l'algorithme de fusion pour fusionner les faces dont la frontière commune possède un faible gradient.

### 7.3.4 Mise à jour des paramètres lors de la fusion

Nous avons vu dans la section 7.1.5 que les moments $\boldsymbol{M}_0$, $\boldsymbol{M}_1$ et $\boldsymbol{M}_2$ étaient attachés aux faces et automatiquement mis à jour lors de la fusion de deux régions. Ces paramètres seront donc calculés pour chaque face lors de la première étape de l'algorithme. Ils seront ensuite automatiquement mis à jour au cours des différentes fusions. Le critère utilisé par notre algorithme de fusion est basé sur le cardinal, la couleur moyenne et l'erreur quadratique de chaque région, ces paramètres se déduisant facilement des moments d'ordre zéro, un et deux (voir définitions 3 et 4). Le surcoût imposé par le calcul de ces paramètres est donc extrêmement faible.

## 7.4 Algorithme de découpe-fusion

### 7.4.1 Algorithme de découpe-fusion simple

Notre algorithme de découpe-fusion prend une face $f_0$ et découpe celle-ci grâce à l'algorithme de découpe décrit dans la section 7.2. Il génère ainsi $n$ faces $\{f_1, \ldots, f_n\}$ qui sont alors fusionnées grâce à l'algorithme décrit dans la section 7.3. L'algorithme de fusion est dans ce cas restreint aux faces $\{f_1, \ldots, f_n\}$. Toutes les faces fusionnées appartiendront donc à $\{f_1, \ldots, f_n\}$. Soit $\{f_1, \ldots, f_p\}$ l'ensemble de faces produites par l'algorithme, ces faces vérifient (voir sections 7.2.3 et 7.3) :

$$\forall i \in \{1, \ldots, p\} \quad \begin{cases} \boldsymbol{SE}(R_{f_i}) \leq \alpha \boldsymbol{SE}(R_{f_0}) & (1) \\ \boldsymbol{SE}(R_{f_i}) \geq \alpha' \boldsymbol{SE}(R_{f_0}) & (2) \end{cases}$$

L'algorithme de découpe-fusion décompose la région de départ $R_{f_0}$ en régions plus homogènes. L'homogénéité des régions produites par l'algorithme est contrôlée par les paramètres $\alpha$ et $\alpha'$. Il peut sembler a priori anormal de borner l'homogénéité des faces produites par l'algorithme. Ce choix peut se justifier de deux façon. Tout d'abord une

(a) (b)

Figure 7.7: Découpe de l'image Lenna

Figure 7.8: *Fusion des faces de la figure 7.7-b*

région significative n'est pas forcément très homogène. De plus l'expérience nous a montré que l'application directe de l'algorithme découpe-fusion avec un critère d'homogénéité trop important produit de nombreuses faces non significatives. Le respect des conditions (1) et (2) assure que les régions produites par l'algorithme seront significatives.

L'étape de découpe qui nous assure la condition (1) est suivie par l'étape de fusion qui fusionne toutes les faces ne respectant pas la condition (2). On ne peut donc pas assurer qu'une fusion ne produira pas une région violant la condition (1). Toutefois, l'écart important entre $\alpha'$ et $\alpha$ permet de limiter fortement ce risque. Nous supposerons donc que les conditions (1) et (2) sont vrais simultanément.

Afin de simplifier l'utilisation de notre algorithme nous ne conservons qu'un seul paramètre. Des mesures expérimentales nous ont conduits à poser :

$$\alpha' = \frac{1}{120}\alpha$$

Le paramètre $\alpha$ est le paramètre contrôlé par l'utilisateur et fixé par défaut à 30%.

### 7.4.2 Algorithme de découpe fusion itératif

L'algorithme de découpe-fusion simple permet de trouver les principales régions d'une image. L'algorithme permettra par exemple de détourer un personnage par rapport au fond. Une idée assez naturelle consiste à itérer cet algorithme de façon à obtenir plus de détails. Dans notre exemple précédent, on peut souhaiter que l'algorithme raffine la région représentant le personnage de façon à obtenir le visage, le corps, les vêtements, etc.... Ce processus est réalisé par l'algorithme de **découpe-fusion itératif.** Celui-ci raffine un ensemble de faces jusqu'à ce que leurs homogénéitées satisfassent un certain critère. Il applique donc l'algorithme de découpe fusion simple sur une face $f_0$ passée en paramètre. Ceci induit la création de $p_1$ faces $\{f_1,\ldots,f_{p_1}\}$. L'algorithme cherche alors la face d'erreur quadratique maximum parmi les faces $\{f_1,\ldots,f_{p_1}\}$ et réapplique l'algorithme de découpe-fusion simple sur celle ci. Cette seconde itération crée $p_2$ faces qui se rajoutent aux faces déjà générées pour former l'ensemble $\{f_1,\ldots,f_{p_1+p_2-1}\}$. Ce processus est itéré jusqu'à ce que l'erreur quadratique de toutes les faces soit inférieure à $\frac{\alpha}{3}\boldsymbol{SE}(R_{f_0})$. Si nous utilisons la valeur de $\alpha$ par défaut les faces générées par l'algorithme ont une erreur quadratique inférieure à 10% de l'erreur quadratique de la face initiale. Ceci permet généralement d'obtenir la majorité des régions présentes dans l'image. Des itérations de l'algorithme de découpe-fusion simple permettent ensuite à l'utilisateur de détailler certaines régions.

Supposons qu'à l'étape $i$ de l'algorithme $p$ faces aient été générées (on a donc $p = \sum_{k=1}^{i-1} p_k - i + 1$). L'algorithme de découpe produit alors $p_i$ faces $\{f_1,\ldots,f_{p_i}\}$. Nous généralisons l'étape de fusion en autorisant les fusions entre les faces $f_i \in \{f_1,\ldots,f_{p_i}\}$ et les $p$ faces générées par l'algorithme de découpe-fusion itératif. Soit $\{f_1,\ldots,f_{p+p_i-1}\}$ l'ensemble des faces générées aux étapes précédentes concaténé aux faces générées par l'algorithme de découpe. A chaque étape de l'algorithme de fusion on fusionnera les deux faces $f$ et $f'$ telles que :

$$(f,f') = ArgMin_{k\in\{1,\ldots,p_i\}} Min_{f''\in\mathcal{V}_g(f_k)\cap\{f_1,\ldots,f_{p+p_i-1}\}} E'(R_{f_k},R_{f''})$$

Intuitivement l'algorithme de découpe fusion itératif peut se comparer à un processus de focalisation d'attention. L'algorithme détermine les principales zones composant la région à segmenter puis se focalise sur chacune d'entre elles afin de la raffiner. Si par exemple l'on examine un personnage, on commencera par extraire le personnage du fond, on dira ensuite qu'il porte un jean bleu et une chemise verte et on ne s'intéressera aux détails du visage ou des vêtements qu'en fin de processus.

La possibilité de fusionner les faces que l'on vient de générer avec l'ensemble des faces déja créées permet de corriger des faces erronées générées lors des premières étapes de l'algorithme. Si nous reprenons le parallèle avec la focalisation d'attention se concentrer sur une face permet de la raffiner et de corriger des détails qui avaient été initialement mal perçus. Le fait de restreindre ces fusions aux faces générées par l'algorithme permet de limiter les corrections aux faces en cours de traitement. Les faces suceptibles d'être fusionnées qui ne sont pas en cours de traitement sont les faces qui étaient voisines de la face initiale $f_0$. Ces faces sont supposées correctes et n'ont donc pas à être modifiées. Si nous reprenons l'exemple précédent, l'algorithme de découpe fusion itératif lancé sur la chemise du personnage s'autorisera à fusionner des faces représentant des détails de la chemise, amis ne supprimera pas de frontières entre le personnage et le fond, celles-ci étant supposées correctes. De plus, les critères relatifs à $R_{f_0}$ n'auraient plus de sens si l'on incorporait à l'algorithme de fusion des faces extérieures à cette région.

```
decoupe_fusion(face f_0)
{
    ensemble de faces L',L = ∅
    face f_max = f_0
    faire
    {
        decoupe(f_max)
        L' ← faces crees par l'algorithme de decoupe
        fusionne(L')
        mettre a jour L et ajouter L' a L
        f_max = ArgMax_{f∈L} SE(R_f)
    }
    tant que(SE(R_{f_max}) / SE(R_{f_0}) > α/3)
}
```

Figure 7.9: *Notre algorithme de découpe-fusion.*

### 7.4.3 Comparaison avec les algorithmes de découpe-fusion classique

Notre algorithme de découpe fusion itératif raffine donc une face passée en paramètre jusqu'à ce que toutes les faces générées soient suffisament homogènes par rapport à la face initiale. Notre algorithme se distingue des algorithmes de découpe-fusion classiques sur les points suivants :

- Les faces générées par l'algorithme de découpe sont de forme quelconque. Cela évite tous les problèmes mentionnés dans la section 5.2.4.5 liés à une partition uniforme.
- L'étape de fusion utilisée par notre algorithme n'est plus restreinte par la structure de données. La distinction entre les fusions restreintes et générales introduite par Ho-

rowitz [HP75, HP76] n'a donc pas lieu d'être dans notre contexte (voir section 5.2.5).

- L'algorithme de découpe-fusion n'est plus une étape finale de l'algorithme de segmentation. En effet les algorithmes utilisant une partition uniforme doivent être appliqués sur une image rectangulaire ou de taille $2^L \star 2^L$ (voir sections 5.2.4.5 et 5.2.5). Notre algorithme peut être appliqué sur une face de forme quelquonque. On peut donc l'appliquer sur des faces générées par d'autres algorithmes ou raffiner avec d'autres algorithmes des faces générées par celui-ci.

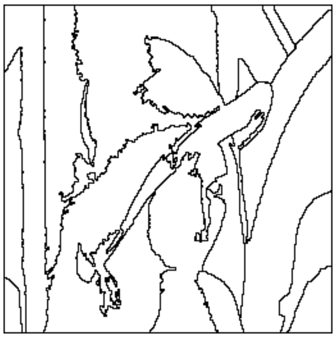


Figure 7.10: *Résultat de l'algorithme de découpe fusion itératif sur Lenna (voir figure 7.7-a)*

## 7.5 Segmentation et chemins euclidiens

Nous avons vu dans la section 6.1.1 que l'ensemble des segments délimitant les régions sont définis dans le plan $P_{\frac{1}{2}}$. Un segment est donc composé de points aux coordonnées demi-entières et ne peut être affiché tel quel à l'échelle 1. Il faut le translater par exemple de $(-\frac{1}{2}, -\frac{1}{2})$ pour pouvoir le représenter dans l'espace image. Comme c'est un chemin 4-connexe, il apparaît crénelé à l'affichage (voir Figure 7.11a).

Pour améliorer la visualisation du contour d'une région, nous avons utilisé les chemins euclidiens développés par Vialard [BV95, VB95, Via96, BLV96]. La construction du chemin euclidien associé à un contour discret s'effectue en associant un point réel à chaque point discret du contour. Chaque point réel appartient au carré de coté 1 centré sur le point discret associé (voir Figure 7.12). Ce carré est appelé la **cellule** associé au point discret.

Le chemin euclidien associé à un chemin de $P_{\frac{1}{2}}$ devient ainsi un polygone dont les sommets ont des coordonnées réelles. En visualisant ce polygone ( par une discrétisation de type Bresenham), on obtient un affichage 8-connexe plus agréable à l'oeil (comparer les figures 7.11a et 7.11b).

La visualisation d'une région relativement petite peut nécessiter d'agrandir la partie correspondante de l'image. Dans ce cas on peut tracer les contours discrets comme des pas horizontaux et verticaux séparant les pixels (voir la partie gauche de la figure figure 7.13). Cependant l'effet d'escalier est accentué et la forme des contours devient difficile à percevoir. Le passage au chemin euclidien associé permet de bien visualiser un contour même fortement zoomé. Pratiquement, il s'agit simplement de multiplier les coordonnées des

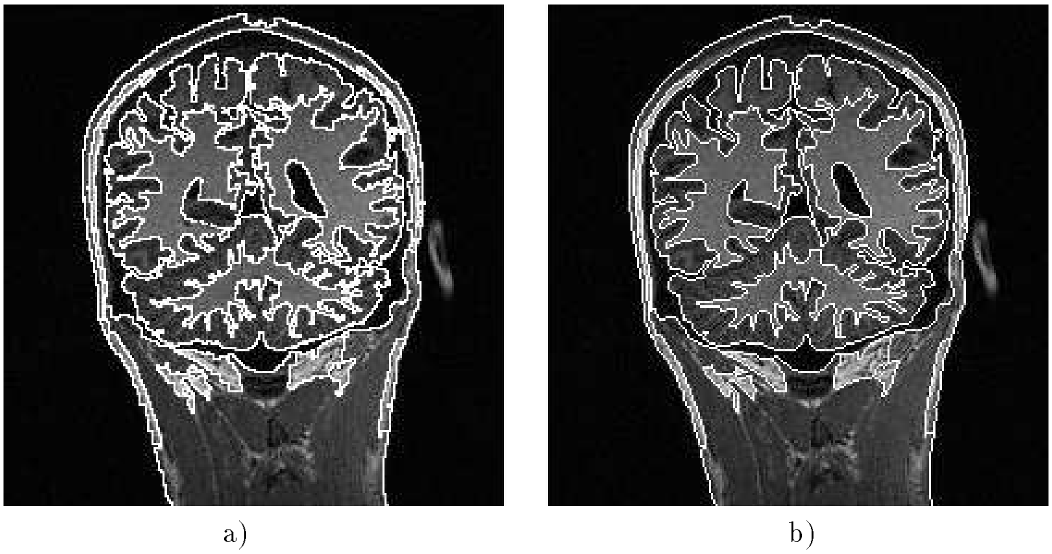


Figure 7.11: Sur l'image de gauche sont superposés à l'image, les contours demi-entiers obtenus par segmentation translatés dans le plan de l'image. Sur l'image de droite ont été tracés les chemins euclidiens correspondants.

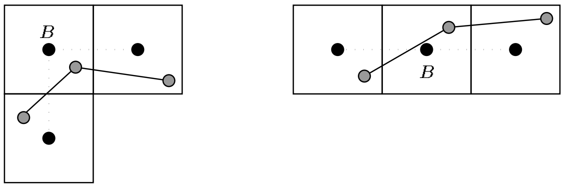



Figure 7.12: *Points euclidiens associés à trois points discrets.*

points euclidiens par le coefficient d'échelle avant de tracer les segments de droites reliant deux à deux ces points. La figure 7.13 montre les contours discrets et euclidiens à l'échelle 10 d'une des régions du cerveau de la figure 7.11.

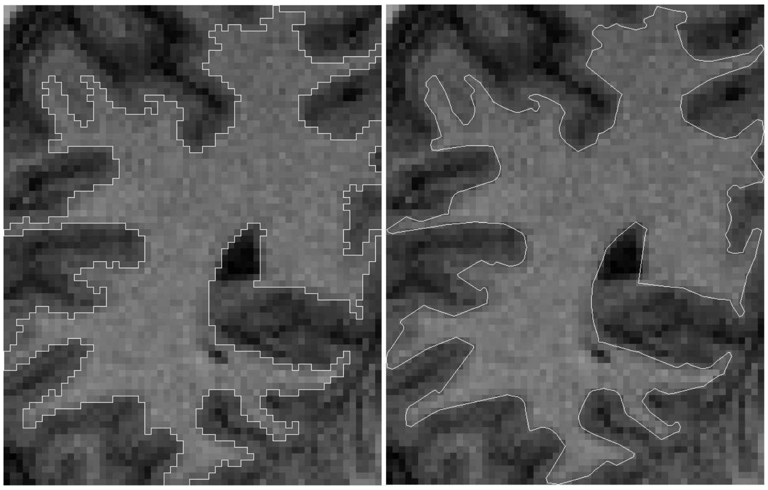


Figure 7.13: *Agrandissement d'une des régions de la figure figure 7.11 à l'échelle* 10.

Le problème est maintenant de trouver comment associer un ensemble de chemins euclidiens à l'ensemble des contours d'une image.

La première idée est tout naturellement d'associer un chemin euclidien $CE_f$ à chaque frontière $C_f$ de la partition. L'inconvénient de cette méthode est qu'elle ne préserve pas le fait que deux régions adjacentes ont une portion de frontière commune. Les contours discrets de deux régions adjacentes partagent en effet un ou plusieurs segments. Soit deux régions adjacentes $R_{f_i}$ et $R_{f_j}$ partageant le segment $s_{ij}$. Soit $CE_{f_i}$ le chemin euclidien associé au contour $C_{f_i}$ de la région $R_{f_i}$ et $CE_{f_j}$ le chemin euclidien associé au contour $C_{f_j}$ de la région $R_{f_j}$. Le segment $s_{ij}$ est parcouru deux fois pour calculer $CE_{f_i}$ et $CE_{f_j}$. Il y a donc deux points euclidiens associés à chaque point discret de $s_{ij}$. Si les points euclidiens associés au segment $s_{ij}$ lors du parcours de $C_{f_i}$ sont tous distincts des points euclidiens associés au même segment lors du parcours de $C_{f_j}$, on a $CE_{f_i} \cap CE_{f_j} = \emptyset$ alors que $C_{f_i} \cap C_{f_j} \neq \emptyset$ (voir figure 7.14).

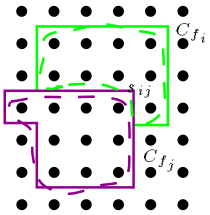



Figure 7.14: *Les chemins euclidiens associés aux deux régions adjacentes ne s'intersectent pas.*

Pour résoudre ce problème, nous associons à chaque point de contour la moyenne des points euclidiens qui lui sont associés au cours des différents parcours. Les points résultants sont bien des points euclidiens associés aux points discrets. En effet, un point euclidien est par définition situé dans la cellule du point discret correspondant. La moyenne de plusieurs points euclidiens associés à un point discret est donc aussi située dans la cellule du point discret. On associe ainsi un ensemble de chemins euclidiens à l'ensemble des contours de l'image.

La définition d'un chemin euclidien assure qu'un contour discret et son chemin euclidien associé traversent exactement la même suite de cellules. Donc si deux portions de contours ont une intersection vide, les chemins euclidiens associés ont aussi une intersection vide. Réciproquement, la technique de moyennage que nous venons de décrire garantit que les intersections entre contours sont préservées lors du passage aux chemins euclidiens. Il n'y aura donc pas d'incohérences lors de l'affichage des contours d'une partition : la frontière commune à deux régions sera représentée par une seule courbe.

La méthode retenue permet de donner une visualisation lissée des contours résultant de la partition d'une image en tenant compte des interactions entre contours adjacents. Elle a été intégrée à notre logiciel afin d'améliorer la visualisation des contours des différentes régions.

Nous avons obtenu le dernier exemple (voir figure figure 7.15) à partir d'une image d'insecte. L'image segmentée a été éditée pour ne conserver que les contours définissant l'insecte. L'image en haut à droite correspond à l'affichage des contours discrets à l'échelle 1. Les contours lissés sont obtenus en zoomant à l'échelle 4 les chemin euclidiens associés aux contours discrets des différentes régions (la technique de moyennage n'a donc pas été utilisée pour cette figure). La figure 7.16 montre une antenne de l'insecte à l'échelle 10 ainsi que la portion de l'image lui correspondant à la même échelle.

## 7.6 Exemple d'utilisation du logiciel

Notre application permet de sélectionner un segment ou une face. La sélection d'une face permet de raffiner celle ci à l'aide des algorithmes de découpe, découpe-fusion simple ou découpe-fusion itératif. L'algorithme de découpe est rarement utilisé au niveau de l'interface, on utilise plutôt les algorithmes de découpe-fusion plus évolués. Le choix entre l'algorithme de découpe-fusion simple et itératif dépend essentiellement du nombre de détails présents dans la face. Ainsi on utilise généralement l'algorithme de découpe-fusion itératif pour obtenir une première segmentation. Celle-ci est ensuite raffinée à l'aide de l'algorithme de découpe-fusion simple. La sélection des segments est essentiellement utilisée pour fusionner interactivement des régions. De plus, cette édition des segments peut être alternée avec des opérations de raffinage de régions.

Les planches 7.17,7.18 et 7.19 représentent une session de notre logiciel. Afin de bien mettre en évidence l'affinage progressif de l'image tous les affinages de faces ont été explorés grâce à l'algorithme de découpe-fusion simple. On obtient tout d'abord une première segmentation de Lenna (voir image 7.17-(a)) à l'aide de cet algorithme. Le résultat de cette première étape est affiché sur l'image 7.17-(c), l'image 7.17-(b) représentant le résultat de l'étape de découpe. On sélectionne ensuite la face contenant le chapeau (voir image 7.17-(d) afin d'affiner celle-ci grâce à l'algorithme de découpe fusion simple. Ceci permet de compléter celui-ci (voir image 7.17-(f)). L'image 7.17-(e) représente encore une fois le résultat de l'algorithme de découpe. Nous sélectionnons ensuite l'image contenant la partie gauche de l'image (voir image 7.18-(a)) afin de la rafiner (voir image 7.18-(b)).

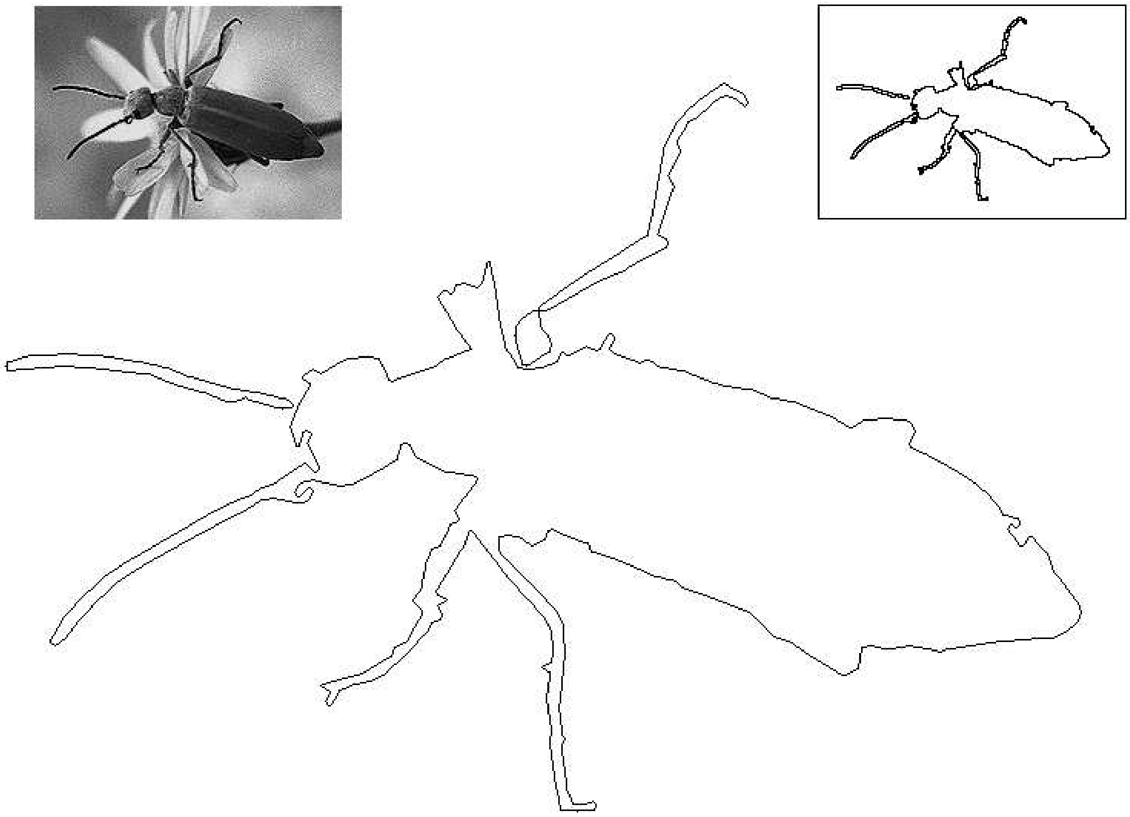


Figure 7.15: *Visualisation des contours d'un insecte. L'image initiale est de taille* $139 \times 200$ *pixels.*

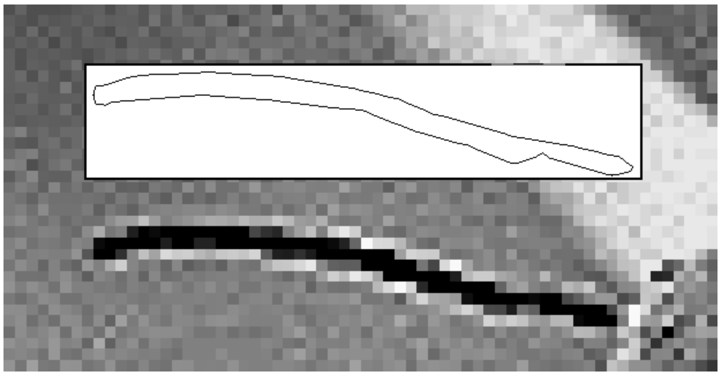


Figure 7.16: *Détail de la figure figure 7.15 : agrandissement à l'échelle* 10 *de l'antenne supérieure de l'insecte et affichage à la même échelle du chemin euclidien associé à son contour.*

L'image 7.18-(c) est issue de l'image 7.18-(b) en enlevant tous les segments composant les objets extérieurs à Lenna. Afin d'affiner le haut du contour du chapeau, nous supprimons le segment correspondant et relançons l'algorithme de découpe fusion sur la face créée par cette suppression (voir image 7.18-(d)). L'image résultante est affichée image 7.18-(f). Là encore l'image 7.18-(e) représente le résultat de l'algorithme de découpe. La planche 7.19 correspond à des affinages successifs du visage à l'aide de fusion et raffinages de faces. L'image 7.19-(a) est obtenue à partir de l'image 7.18-(f) en rafinant la face représentant le visage. L'image 7.19-(b) est obtenue à partir de l'image 7.19-(a) par suppression des segments qui découpe le visage en plusieurs régions d'intensité moyenne différente. L'image 7.19-(c) est obtenue à partir de l'image 7.19-(b) en affinant la face correspondant au toupet de lenna. L'image 7.19-(d) est obtenue à partir de l'image 7.19-(c) en affinant la chevelure et l'oeil droit de Lenna. L'image 7.19-(e) est obtenue à partir de l'image précédente en affinant le chapeau. Cette image est enfin éditée afin de supprimer certains segments non significatifs (voir image 7.19-(f).

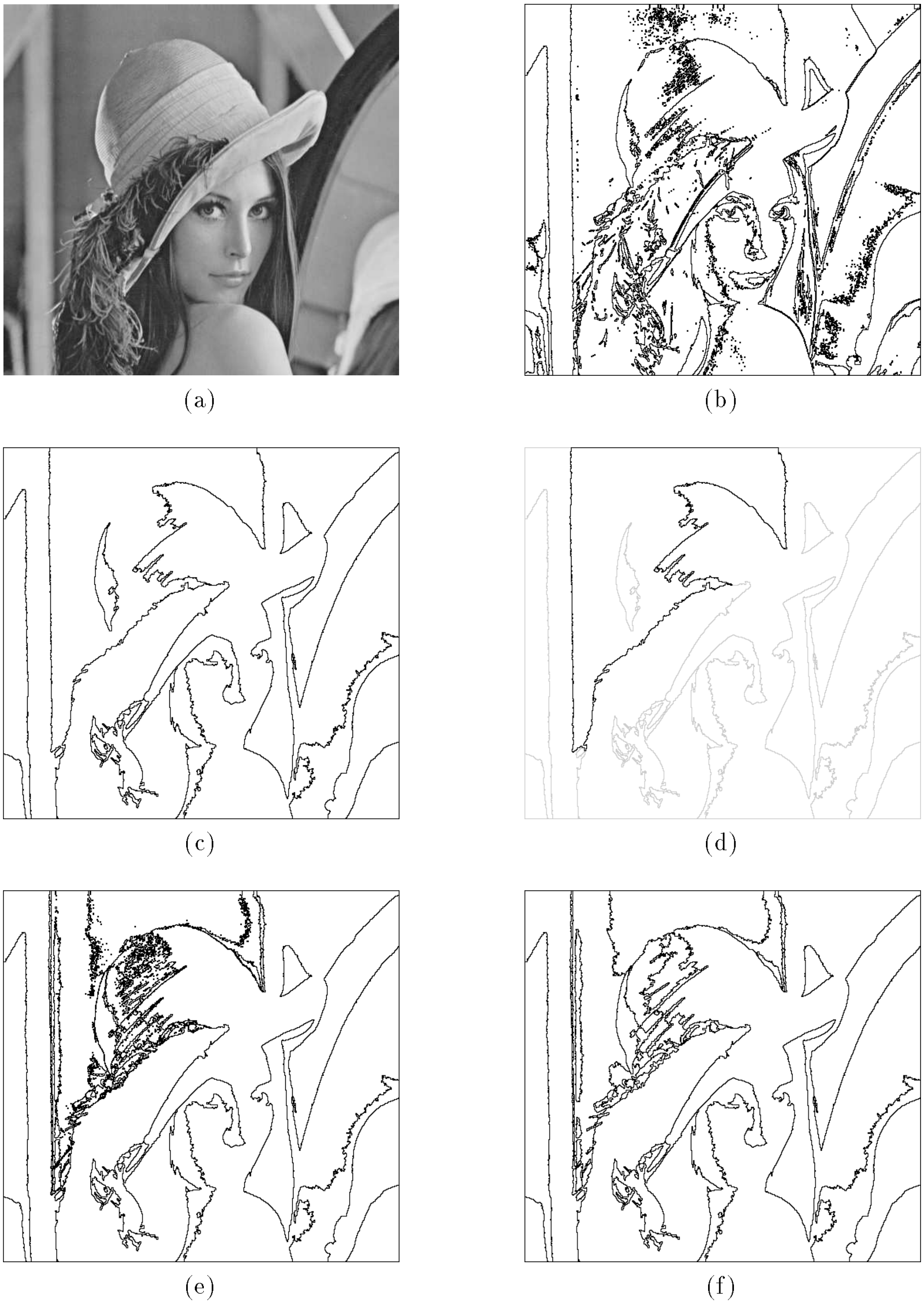


Figure 7.17: Un exemple de session de notre application

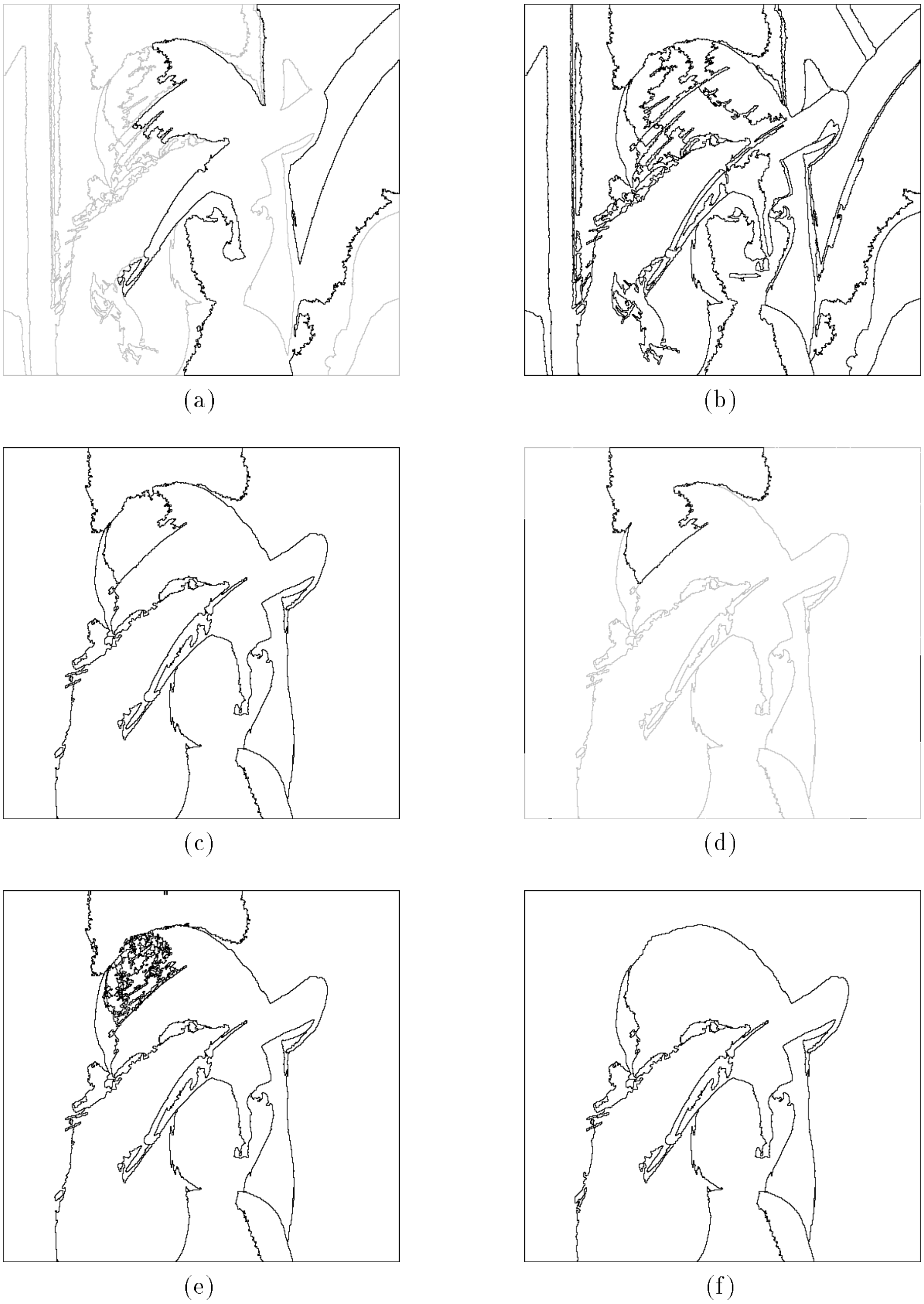



Figure 7.18: Un exemple de session de notre application, suite.

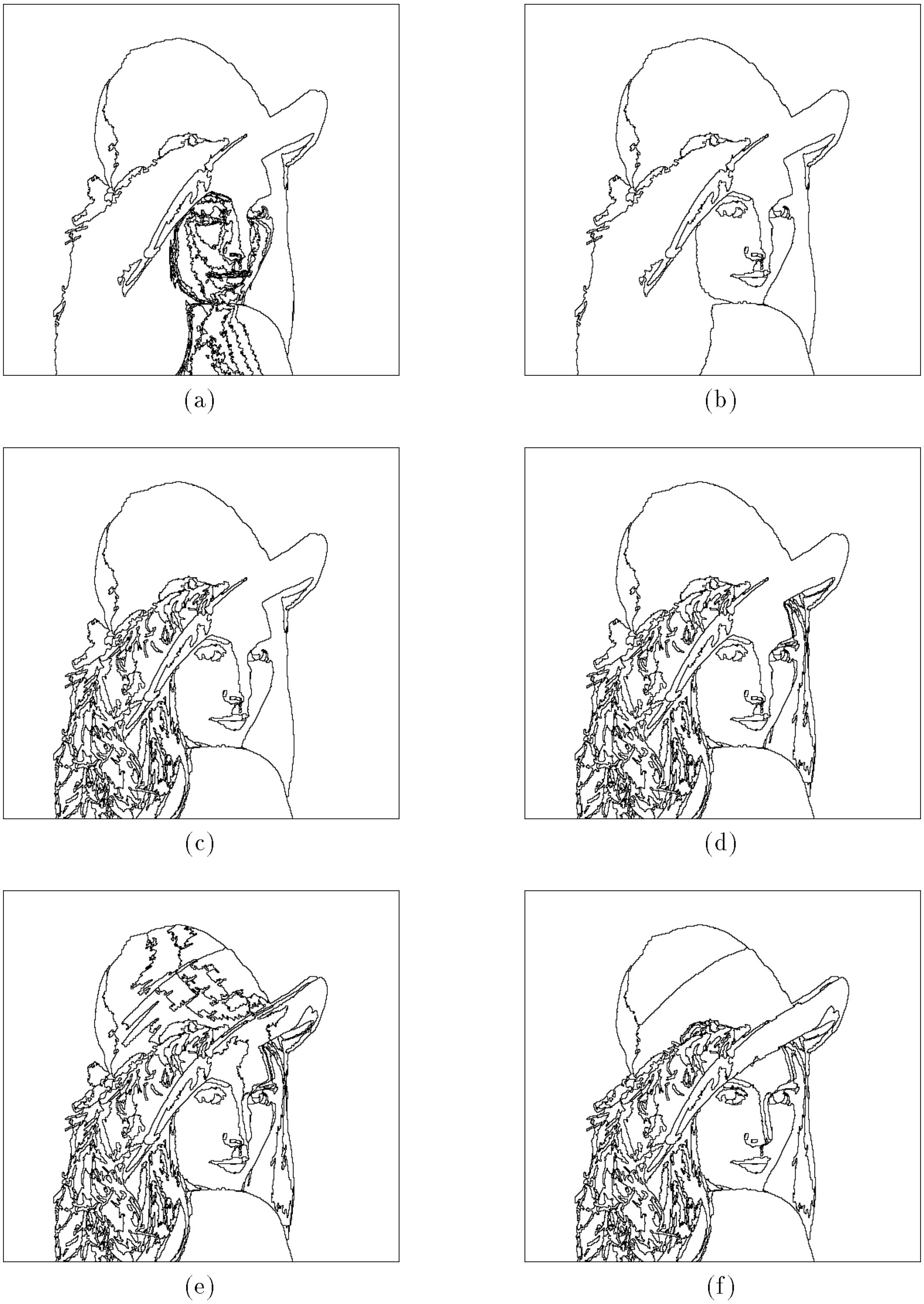



Figure 7.19: Un exemple de session de notre application : suite et fin

# Conclusion

Nous avons vu dans l'introduction qu'une image est composée de pixels, chaque pixel étant associé à 1 (respectivement 3) valeurs entières permettant de coder son niveau de gris (resp. sa couleur). Le sujet de notre travail étant la segmentation d'image couleurs, nous avons naturellement décomposé celui-ci en deux parties :

La première partie nous a permis d'établir un certain nombre de propriétées des ensembles de données ( voir chapitre 2). Le chapitre 1 nous a permis de définir plus précisément le concept de couleur et d'utiliser les notions du chapitre 2 pour les ensembles de couleurs. Le chapitre 3 a utilisé ces résultats pour définir deux algorithmes de quantification permettant de réduire le nombre de couleurs d'une image tout en préservant au maximum l'information qu'elle contient. Ces algorithmes ont les caractéristiques suivantes :

- Le premier algorithme décrit dans la section 3.7 établit un compromis entre différent algorithmes de quantification. Il permet d'obtenir des résultats proches de ceux obtenus avec les meilleurs algorithmes en beaucoup moins de temps (environs 1,5 fois plus vite).
- Le second algorithme (voir section 3.8) est basé sur une approche découpe-fusionne qui jusqu'ici a été peu utilisé en quantification. Les deux versions de cet algorithme permettent soit d'obtenir de bon résultats avec des temps interactifs, soit d'obtenir des résultats meilleurs que ceux obtenus avec les meilleurs algorithmes de quantification en des temps presque interactifs ( de l'ordre de la dizaine de seconde).

La seconde partie est consacrée à la segmentation. Après avoir rappelé quelques notions de base pour la segmentation (voir chapitre 4), nous avons proposée une classification des algorithmes de segmentation (voir chapitre 5) et des structures de données usuellement utilisées par ceux-ci (voir chapitre 5.3). Cette étude de "l'état de l'art" nous à amené à utiliser une structure de données initialement définie par J.P Domenger [Dom92] pour l'édition d'images 2D. Les principales modifications que nous avons dû apporter à cette structure de données pour l'adapter aux problèmes de segmentation sont les suivantes :

- Nous avons développé certains cas d'insertion de segments peu fréquents en édition d'image mais inévitables en segmentation. La complétions des méthodes d'insertion de segments permet de partitionner une image en région et de rafinner itérativement une région.
- Nous avons défini un ensemble de primitives permettant de supprimer un ensemble

quelconque de segments de la partition. Ces primitives permettent de supprimer une région ou de fusionner deux régions.

Grâce à cette structure de données et aux primitives qui lui sont associées nous avons construit une ensemble de fonctionnalités permettant de définir rapidement un algorithme de segmentation (voir chapitre 7). Nous avons ensuite utilisé ces fonctionnalités afin de définir plusieurs algorithmes de segmentation basés sur les méthodes de quantification vues au chapitre 3 et les résultats du chapitre 2.

Les résultats que nous avons obtenu permettent de dessiner deux axes de recherche. Comme nous l'avons vu l'approche découpe-fusionne a été peu exploré jusqu'à présent. Cette approche donnant des résultats prométeurs nous nous proposons de la développer suivant deux axes :

- Essayer d'autres heuristiques de découpe du multi-ensemble associé à l'image (par exemple une montée des eaux 3D).
- Raffiner l'heuristique de fusion des multi-ensembles issus de l'étape de découpe (voir section 3.8.1).

Nous avons jusqu'à présent peu explorer ces deux voies. Le seul résultat que nous ayons obtenue est l'inadéquation de ce type d'algorithme avec les méthodes de découpe vues dans la section 3.6. Il est en effet apparu que l'heuristique de fusion est alors trop simple pour améliorer sensiblement les résultats de ces algorithmes. De plus le temps de fusion se rajoute alors au temps de découpe rendant l'algorithme excessivement lent.

Nous avons vu dans le chapitre 7 que le noyau et l'ensemble des couches développées autour de celui-ci permettent :

- de structurer les résultats d'une découpe,
- de définir rapidement des algorithmes de découpe ou de fusion,
- d'extraire rapidement des paramètres pertinents des régions.

Toutes ces fonctionnalités permettent de définir facilement des algorithmes de segmentation. Nous avons vu que dans sa formulation la plus générale un algorithme de segmentation extrait un ensemble d'attibuts de chaque face afin de sélectionner l'algorithme de découpe ou de fusion à appliquer. Un algorithme de segmentation peut donc s'exprimer à l'aide d'une suite de conditions de type :

$Si(caractrisation[i](\{f_1, \ldots, f_n\}))$
Appliquer Traitement[i]($\{f_1, \ldots, f_n\}$)

où $caractrisation[i](\{f_1, \ldots, f_n\})$ désigne un test sur les paramètres des faces $\{f_1, \ldots, f_n\}$. Le test de caractérisation peut se décomposer en un ensemble de conjonctions ou disjonctions chaque test pouvant prendre une ou plusieurs faces en paramètres.

On voit donc apparaître le concept de **règles de segmentation**. Une règle prend en paramètre une ou plusieurs faces et leur applique un traitement adéquat. Le but de la précondition est de caractériser la ou les faces à traiter. Par exemple, une face homogène présentant des détails significatif peut être caractérisé par :

$$\max_{P,P' \in R_f} \|I(P) - I(P')\|^2 > \epsilon \text{ et } \boldsymbol{SE}(R_f) \leq \alpha$$

On pourra donc utiliser pour découper cette face un algorithme de découpe qui exploite le fait que la face à découper est homogène. Nous avons par exemple, développé un algorithme de découpe qui approxime le multi-ensemble $(C_{R_f}, f_{R_f})$ d'une région $R_f$ par une gaussienne 3D. Intuitivement, cela revient à postuler que la majorité des couleurs de $(C_{R_f}, f_{R_f})$ sont "proches" de la moyenne $\mu$ de $(C_{R_f}, f_{R_f})$. On partitionne donc $(C_{R_f}, f_{R_f})$ en deux multi-ensembles $(C_1, f)$ et $(C_2, f)$ tels que :

$$C_1 = \{c \in C_{R_f} \ / \ e^{-\frac{\|c-\mu\|^2}{\sigma^2}} < \alpha\}$$
$$C_2 = \{c \in C_{R_f} \ / \ e^{-\frac{\|c-\mu\|^2}{\sigma^2}} \geq \alpha\}$$

où $\sigma$ désigne la l'écart type du multi-ensemble $(C_{R_f}, f_{R_f})$.

Inversement, une face inhomogène peut être caractérisée par :

$$\boldsymbol{SE}(R_f) \geq \alpha$$

On peut dans ce cas lui appliquer un algorithme de découpe plus général comme celui décrit dans la section 7.2.

L'ensemble des caractéristiques pouvant être attaché à une face est très important, on peut décrire les faces homogènes sans détails, les faces homogènes contenant des détails les faces inhomogène, etc.... Il nous a semblé peut judicieux d'intégrer un ensemble de caractéristiques à l'application. En effet cet ensemble ne pourrait contenir toutes les caractérisations souhaitables par un utilisateur. De plus la modification de certaines caractéristiques comme la définition du bruit ou d'une face représentant un détail nécessite d'intervenir dans le code de l'application ce qui peut rapidement devenir extrêmement lourd si l'on manipule de nombreuses caractéristiques.

Nous prévoyons donc, dans les développements futurs de notre application, de laisser à l'utilisateur la possibilité de définir un ensemble de règles dans un fichier texte. Ce fichier sera interprété lors du lancement de notre application. L'ensemble des règles définies à partir des caractéristiques de régions constitue ce que nous appelons un algorithme de **méta-segmentation**. L'ensemble des règles peut alors être utilisé pour la segmentation d'une image.

# Bibliographie


[And72] H. C. Andrews. *Introduction to mathematical techniques in pattern recognition.* Willey, New York, 1972.

[And73] M.R. Anderberg. *Cluster analysis for applications.* Academic Press, New York, 1973.

[And92] Éric Andres. *Cercles discrets et rotations discrètes.* PhD thesis, Université Louis Pasteur, Strasbourg, 1992.

[Aur91] F. Aurenhammer. Voronoi diagrams. a survey of fundamental geometric data structure. *ACM Computing surveys*, 33(3):345–405, 1991.

[BA91] R. Balasubramanian and J. Allebach. A new approach to palette selection for color images. *Journal of imaging technology*, 17(6):284–290, december 1991.

[Bar79] P. H. Bartels. Numerical evaluation of cytologic data : Description of profiles. *Analytical and Quantitative Cytology*, 1(1):20–28, 1979.

[BB95] J.P. Braquelaire and L. Brun. Comparison and optimization of methods of color image quantization. Submitted to IEEE Image Processing, 1995.

[BBA94] Raja Balasubramanian, Charles A. Bouman, and Jan P. Allebach. Sequential scalar quantization of color images. *Journal of Electronic Imaging*, 3(1):45–59, January 1994.

[BD96a] J.P. Braquelaire and J.P. Domenger. Representation of region segmented images with discrete maps. Submitted, 1996.

[BD96b] L. Brun and J.P. Domenger. Incremental modifications on segmented image defined by discrete maps. Submitted, 1996.

[Ber94] E. Bertin. *Diagrammes de Voronoi 2D et 3D : application en analyse d'images.* PhD thesis, Université J. Fourier, Grenoble, 1994.

[BF70] R. Brice and C.L. Fennema. Scene analysis using regions. *Artificial intelligence*, 1:205–226, 1970.

[BF82] B. Bhanu and O. D. Faugeras. Segmentation of images having unimodal distributions. *IEEE Trans. PAMI*, 4:408–419, 1982.

[BGK+89] J.R Beveridge, J. Griffith, R. Kohler, A.R. Hanson, and E.M. Riseman. Segmenting images using localized histograms and region merging. *International Journal of Computer Vision*, 2:311–347, 1989.

[Bha67] C. G. Bhattacharya. A simple method of resolution of a distribution into gaussian components. *Biometrics*, 23:115–135, 1967.

[BLV96] J.P. Braquelaire, L.Brun, and A. Vialard. Inter-pixel euclidean paths for image analysis. In *Discrete Geometry for Computer Imagery*, Lyon, November 1996.

[BP87] B. Bhanu and B. A. Parvin. Segmentation of natural scenes. *Pattern Recognition*, 20(5):487–496, 1987.

[BP91] J.P. Braquelaire and Guitton P. $2\frac{1}{2}$ scene update by insertion of contour. *Computer and Graphics*, 15(1):41–48, 1991.

[Bro82] J.D. Browning. Segmentation of pictures into regions with tile-by-tile method. *Pattern Recognition*, 15(1):1–10, 1982.

[BV95] J.P. Braquelaire and A. Vialard. Euclidean paths : a new representation of boundary of discrete regions. Technical Report 1088-95, LaBRI, UBX1, 1995. Submitted.

[CA79] Guy. B. Coleman and Harry C. Andrews. Image segmentation by clustering. *Proceedings of the IEEE*, 67(5):773–785, May 1979.

[Can86] J.F. Canny. A computational approach to edge detection. *IEEE Trans. PAMI*, 8(6):679–698, 1986.

[CD85] J.-P. Cocquerez and J. Devars. Détection du contour dans les image aériennes : nouveaux opérateurs. *Traitement du signal*, 2(1), 1985.

[Cel90] Mehmet Celenk. A color clustering technique for image segmentation. *Computer Vision, Gaphics, and Image Processing*, 52:145–170, 1990.

[CG84] J.M. Chassery and C. Garbay. An iterative segmentation method based on a contextual color and shape criterion. *IEEE Trans. PAMI*, 6(6):794–800, 1984.

[CGG91] A. Cumani, P. Grattoni, and A. Guiducci. An edge-based description of color images. *Computer Vision, Graphics, and Image Processing*, 53(4):313–323, July 1991.

[Cha92] M. Chapron. A new chromatic edge detector used for color image segmentation, 1992.

[CK72] C. K. Chow and T. Kaneko. Automatic boundary detection of the left ventricle from cineangiograms. *Comput. Biomed. Res.*, 5:388–410, 1972.

[CM91] J.M. Chassery and A. Montanvert. *Géométrie dicrète en analyse d'images*, chapter 1, page 35. Hermès, 1991.

[CMVM86] F. Cheevasuvut, H. Maitre, and D. Vidal-Madjar. A robust method for picture segmentation based on a split and merge procedure. *CVGIP*, 34:268–281, 1986.

[Cor75] R. Cori. *Un code pour les graphes planaires et ses applications*. PhD thesis, Université Paris VII, 1975.

[CP79] P. Chen and T. Pavlidis. Segmentation by texture using a co-occurence matrix and a split and merge algorithm. *Computer Graphics, and Image Processing*, 10:172–182, 1979.

[CP95] J.-P. Cocquerez and S. Philipp. *Analyse d'images : Filtrage et segmentation.* Masson, 1995.

[CTM89] P. A. Chou, Lookabaugh T., and Gray R. M. Optimal pruning with applications to tree-structured source coding and modeling. *IEEE Trans. Inf. Theory*, 2:299–315, 1989.

[Cum89] A. Cumani. A second order differential operator for multispectral edge detection. In *5th Int. Conf. Image Anal. Process.*, pages 54–58, Positano, 1989.

[Cum91] A. Cumani. Edge detection in multispectral images. *Computer Vision, Graphics, and Image Processing*, 53:40–51, 1991.

[CZS89] S. Castan, J. Zhao, and J. Shen. Une famille de détecteurs de contours basée sur le filtre exponentiel optimal. In $7^{eme}$ *congrès AFCET-RFIA*, Paris, 1989.

[DC87] R. Deriche and J.-P. Cocquerez. Extraction de composantes connexes basée sur une détection optimale des contours. In *MAR87*, Paris, la vilette, mai 1987.

[Der87] R. Deriche. Using canny's criteria to derive a recursively implemented optimal edge detector. *International Journal of Computer Vision*, 2(1):167–187, 1987.

[DG80] A. Dubec and L. Goussot. Télévision en couleur, photométrie, colorimétrie, le tube image. Technical Report 1.80, Radiodiffusion Télévision, 1980.

[Did71] E. Diday. Une nouvelle méthode en classification automatique et reconnaissance des formes : La méthode des nuées dynamiques. *INRIA, revue de statistique appliquée*, 19(2):19–33, 1971.

[DL78] H. Digabel and Ch. Lantuéjoul. Iterative algorithms. In J-L. Chermant, editor, $2^{eme}$ *European Symposium on Quantitativew Analysis of Microstructures in Material Science, Biologie and Medecine*, pages 85–99, Stuttgart, 1978.

[DLPT82] E. Diday, J. Lemaire, J. Pouget, and F. Testu. *Éléments d'analyse de données.* Dunod, 1982.

[Dom92] J.P. Domenger. *Conception et implémentation du noyau graphique d'un environnement $2D\frac{1}{2}$ d'édition d'images discrètes.* PhD thesis, Labri Université Bordeaux I, 351 cours de la libération 33405 Talence, avril 1992.

[DRH80] R.C. Dyer, A Rosenfeld, and S Hanan. Region representation: Boundary codes from quadtrees. *ACM: Graphics and Image Processing*, 23(3):171–179, March 1980.

[DST92] M. B. Dillencourt, H. Samet, and M. Tamminen. A general approach to connected-component labeling for arbitrary image representations. *J. Assoc. Comput. Mach.*, 39(2):253–280, 1992. Corr. p. 985-986.

[FG94] C. Fiorio and J. Gustedt. Two linear time union-find strategies for image processing. Technical Report 375/1994, Technische Universitat Berlin, 1994.

[FG96] Christophe Fiorio and Jens Gustedt. Two linear time union-find strategies for imae processing. *Theoretical Computer Science*, 154:165–181, 1996.

[FGS90] Chritine Froidevaux, Marie-Claude Gaudel, and Michèle Soria. *Types de données et algorithmes*. Mc Graw-Hill, 1990.

[Fio95] C. Fiorio. *Approche interpixel en analyse d'images : une topologie et des algorithmes de segmentation*. Thèse de doctorat, Université Montpellier II, 24 novembre 1995.

[Fis58] W. D. Fisher. On grouping for maximum homogeneitu. *JASA*, 53:789–798, 1958.

[Fre61] H. Freeman. On the encoding of arbitrary geometric configurations. *IRE Trans. Electr. Compu.*, 10:260–268, June 1961.

[FS76] R.W. Floyd and L. Steinberg. An adaptative algorithm for spatial greyscale. *Proc. SID*, 17(2):75–77, 1976.

[Fuk72] K. Fukunaga. *Introduction to statistical pattern recognition*. Academic Press, New York, 1972.

[FV92] F. Ferri and E. Vidal. Colour image segmentation and labeling through multiedit-condensing. *PRL*, 13:561–568, 1992.

[Gag83] Andre Gagalowicz. *Vers un modele de textures*. Ph.d. thesis, Universite Pierre et Marie Curie, Paris VI, 1983.

[Gag87] Andre Gagalowicz. Texture modelling applications. *The Visual Computer*, 3(4):186–200, December 1987.

[GAW90] Ronald S. Gentile, Jan P. Allebach, and Eric Walowit. Quantization of color images bases on uniform color spaces. *Journal of imaging technology*, 16:11–21, 1990.

[GP90] Michael Gervautz and Werner Purgathofer. *A simple method for color quantization: Octree quantization*. Glassner editor, 1990.

[Hec82] Paul Heckbert. Color image quantization for frame buffer display. In *Proceedings of SIG-GRAPH'82*, pages 297–307, 1982.

[HM86] A. Huertas and G. Medioni. Detection of intensity changes with subpixel accuracy using laplacian-gaussian masks. *IEEE Trans. PAMI*, 8(5):651–661, 1986.

[HP75] S.L. Horowitz and T. Pavlidis. Picture segmentation by a directed split-and-merge procedure. Technical report, Departement of Electrical Engineering, Princeton University , N. J. 08540, 1975.

[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, April 1976.

[HS79] G.M. Hunter and K. Steiglitz. Operations on images using quad trees. *IEEE Trans. On Pattern Analysis and Machine Intelligence*, 1(2):145–153, April 1979.

[HS88] L. Hertz and R. W. Schafer. Multilevel thresholding using edge matching. *Computer Vision, Graphics, and Image Processing*, 44:279–295, 1988.

[JM92] J.M. Jolion and A. Montanvert. The adaptative pyramid : A framework for 2d image analysis. *Computer Vision, Graphics, and Image Processing*, 55(3):339–348, May 1992.

[Kir71] R. Kirsch. Computer determination of the constituent structure of biological images. *Computer Biomedical Research*, 4:315–328, 1971.

[KKM90] E. Khalimsky, R. Kopperman, and P.R. Meyer. Computer graphics and connected topologies on finite ordered sets. *Topology and its Applications*, 36:1–17, 1990.

[Koh84] R. R. Kohler. *Integrating non-semantic knowledge into image segmentation processes.* PhD thesis, Univ. of Massachusetts, 1984.

[Kov89] V.A. Kovalevsky. Finite topology as applied to image analysis. *Computer Vision, Graphics, and Image Processing*, 46:141–161, 1989.

[Lee86] C.-H. Lee. Recursive region splitting at hierarchical scope views. *Computer Vision, Graphics, and Image Processing*, 33:237–258, 1986.

[Lev70] U. Levi, G. Montanari. A grey-weighted skeleton. *Information Control*, 17:62–91, 1970.

[Lev85] M.D. Levine. *Vision in man and machine.* Mc Graw-Hill Book Compagny, 1985.

[LIF95] LIFL-ENIC. *Perception colorée et traitements visuels*, Lille, mars 1995.

[LL90] Y. W. Lim and S. U. Lee. On the color image segmentation algorithm based on the thresholding and the fuzzy c-means techniques. *Pattern Recognition*, 23(9):935–952, 1990.

[LSC91] J. Lin, J. Storer, and M. Cohn. On the complexity of the optimal tree pruning for source coding. In *Proc. of data compression conference*, pages 63–72. IEEE computer society Press Los Angeles, 1991.

[Mar88] K. V. Mardia. A spacial thresholding method for image segmentation. *IEEE Trans. PAMI*, 10(6):919–927, 1988.

[MH80] D. Marr and E. Hildreth. Theory of edge detection. *Proccedings of Royal Society of London*, 207:187–217, 1980.

[Mon90] O. Monga. *Segmentation d'images : Où en sommes nous ?* INRIA, Domaine de voluceau Rocquancourt B.P. 105 78153 Le chesnay Cedex France, Avril 1990. Rapports de Recherche.

[M.w84] M.williams. Algorithm 232 heapsort. *Graphic and Image Processing Communication of the ACM*, 7:347–348, June 1984.

[Nag68] G. Nagy. State of the art in pattern recognition. In *Proc. IEEE*, volume 56. IEEE, may 1968.

[Nic95] C. J. Nicol. A systolic approach for real time connected component labeling. *Computeer vision and Image understanding*, 61(1):17–31, January 1995.

[NM79] M. Ngao and T. Matsuyama. Edge preserving smoothing. *Computer Vision, Graphics, and Image Processing*, 9:394–407, 1979.

[NR79] Y. Nakagawa and A. Rosenfeld. Some experiments on variable thresholding. *Pattern Recognition*, 11:191–204, 1979.

[OB91] Michael T. Orchard and Charles A. Bouman. Color quantization of images. *IEEE transactions on signal processing*, 39(12):2677–2690, 1991.

[Ohl75] R. Ohlander. *Analysis of natural scenes*. PhD thesis, Department of Computer science, Carnegie-Mellon University, 1975.

[OKS80] Yu-Ichi Ohta, Takeo Kanade, and Toshiyuki Sakai. Color information for region segmentation. *Computer Vision, Graphics, and Image Processing*, 13:222–241, 1980.

[OPR78] R. Ohlander, K. Price, and D.R. Reddy. Picture segmentation using a recursive region splitting method. *Computer Graphics, and Image Processing*, 8:313–333, 1978.

[Pae91] Alan Paeth. Mapping rgb triples onto 16 distinct values. In James Arvo, editor, *Graphics Gems II*, volume II, chapter 3, pages 143–146. Academic Press, 91.

[Pav77] T. Pavlidis. *Structural Pattern recognition*. Springer-Verlag, Heidelberg, 1977.

[PG94] N. Papamarkos and B. Gatos. A new approach for multilevel threshold selection. *Computer Vision, Graphics, and Image Processing*, 56(5):357–370, September 1994.

[Poi81] M.R. Pointer. A comparison of the cie 1976 color spaces. *COLOR research and application*, 6(2):108–118, 1981.

[Pre70] J.M.S. Prewitt. *Picture Processing and Psychopictorics*, chapter Object Enhancement and Extraction, pages 75–149. Academic Press, London, 1970.

[PRW82] M. Pietikainen, A. Rosenfeld, and I. Walter. Split and link algorithms for image segmentation. *Pattern Recognition*, 15(4):287–298, 1982.

[PS88] F. Preparata and M. I. Shamos. *Computational geometry, an introduction*. Springer-Verlag, Heildeberg, 1988.

[RK82] A. Rosenfeld and A. Kak. *Digital Picture Processing*, volume 2, chapter 11, pages 191–277. Harcourt and Brace and Jovanovich, 1982.

[Ros79] A. Rosenfeld. Digital topology. *Amer. math. monthly*, 86:621–630, 1979.

[RP68] A. Rutovitz and J. L. Pfaltz. Data structures for operations on digital images. *Pictorial Pattern Recognition*, pages 105–133, 1968.

[Sam80] H. Samet. Region representation: Quadtrees from bondary codes. *ACM : Graphics and Image Processing*, 23(3):163–170, March 1980.

[SB85] M.H. Savoji and R.E. Burge. Note : on differents methods based on the karhunen-loeve expansion and used in image analysis. *Computer Vision, Graphics, and Image Processing*, 29:259–269, 1985.

[SC92] Jun Shen and Serge Castan. An optimal linear operator for step edge detection. *Computer Vision, Graphics, and Image Processing*, 54(2):112–133, march 1992.

[Suk83] M. Suk. A new segmentation technique based on partition mode test. *PaRe*, 16(5):469–480, 1983.

[Tar79] R. E. Tarjan. A class of algorithms which require non-linear time to maintain disjoint sets. *J. Comput. System Sci.*, 18:110–127, 1979.

[Tho91] Spencer W. Thomas. *Efficient inverse color map computation.* Glassner editor, 1991.

[TK80] S. Tanimoto and A. Klinger, editors. *Structured computer vision*, volume 1. Academic Press, 1980.

[Tre93] A. Tremeau. *Contribution des modèles de la perception visuelle à l'analyse d'image couleur.* PhD thesis, Université Jean Monnetde Saint Etienne, Octobre 1993.

[Tut63] W.T. Tutte. A census of planar maps. Canad.J.Math., 15:249–271, 1963.

[VB95] A. Vialard and J.P. Braquelaire. Transformations géométriques de courbes discrètes 2d. In *3èmes journées de l'AFIG (Marseille)*, 1995.

[Via96] A. Viallard. Geometrical parameters extraction from discrete paths. Submitted to the 6th DGCI conference, june 1996.

[VS91] L. Vincent and P. Soille. Watersheds in digital spaces : and efficient algorithm based on immersion simulations. *IEEE Trans. PAMI*, 13(6):583–598, 1991.

[Wat74] S. Watanabe. An automated apparatus for cancer prescreening : Cybest. *Computer Graphics, and Image Processing*, 3:350–358, 1974.

[Wes78] J. S. Weszka. A survey of threshold selection techniques. *Computer Graphics, and Image Processing*, 7:259–265, 1978.

[Wit84] A. P. Witkin. *Image Understanding*, chapter Scale Space Filtering : a new approach to multi-scale description, chapter 3, pages 79–95. Ablex, 1984.

[Wri41] W.D. Wright. The sensitivity of the eye to small colour differences. *Proc. Phys. Soc.*, 53(296):93–112, 1941. Part 2.

[WS90] D. J. Williams and Mubarak Shah. Edge contours using multiple scales. *Computer Vision, Graphics, and Image Processing*, 51:256–274, 1990.

[Wu92] Xiaoling Wu. Color quantization by dynamic programing and principal analysis. *ACM Transaction on Graphics*, 11(4):348–372, 1992.

[WWP88] S.J. Wan, S.K.M Wong, and P. Prusinkiewicz. An algorithm for multidimensional data clustering. *ACM Trans. Math. Softw.*, 14(4):153–162, 1988.

[WWP89] S.K.M. Wong, S.J. Wan, and P. Prusinkiewicz. Monochrome image quantization. In *Canadian conference on electrical and computer engineering*, pages 17–20, September 1989.

[WZ91] Xiaolin Wu and K. Zhang. A better tree-structured vector quantizer. In *Proceedings of the IEEE Data Compression Conference*, pages 392–401. IEEE Computer Society Press, 1991.

[XJ94] Zhigang Xiang and Gregory Joy. Color image quantization by agglomerative clustering. *IEEE Computer Graphics and Applications*, 272-17-16:44–48, 1994.

[Zen86] S.D. Zenzo. A note on the gradient of a multi-image. *Computer vision, Graphics and Image Processing*, 33:116–125, 1986.

[Zuc76] S. W. Zucker. Region growing : Childhood and adolescence. *Computer Graphics, and Image Processing*, 5:382–399, 1976.

# Annexe

## 7.7 Preuve du théorème 2

**1)** Montrons tout d'abord que $\mu_1$ et $\mu_2$ sont deux fonctions croissantes. On a :

$$\mu_1(t) = \frac{\int_0^t xf(x)dx}{\int_0^t f(x)} \tag{7.6}$$

On a donc :

$$\mu'_1(t) = \frac{tf(t)\int_0^t f(x)dx - f(t)\int_0^t xf(x)dx}{(\int_0^t f(x))^2}$$
$$\mu'_1(t) = f(t)\frac{\int_0^t (t-x)f(x)dx}{(\int_0^t f(x))^2} \geq 0$$

On a de même :

$$\mu'_2(t) = f(t)\frac{\int_t^1 (x-t)f(x)dx}{(\int_t^1 f(x))^2} \geq 0 \tag{7.7}$$

Les fonctions $\mu_1$ et $\mu_2$ sont donc toutes deux croissantes. Donc la fonction $g$ moyenne de deux fonctions croissantes est une fonction croissante. De plus :

$$0 \leq \mu_1(t) \leq t\frac{\int_0^t f(x)dx}{\int_0^t f(x)} = t$$

De même:

$$0 \leq \mu_2(t) = \frac{\int_t^1 xf(x)dx}{\int_t^1 f(x)dx} \geq t$$

**2)** Supposons que l'on ait $y(t) > 0$, ou c'est équivalent $g(t) > t$, on à alors $g^2(t) \geq g(t)$ soit $y(g(t)) \geq 0$. De même :

$$g(t) < t \Rightarrow g^2(t) \leq g(t)$$
$$g^2(t) \leq g(t) \Rightarrow y(g(t)) \leq 0$$

Supposons à présent que :

$$\exists t_0 \in [t, g(t)[ \ / \ g(t_0) = t_0 \tag{7.8}$$

$g$ étant croissante, on a donc :

$$\begin{aligned} t \leq t_0 &\leq g(t) \\ g(t) \leq g(t_0) &\leq g^2(t) \\ g(t) \leq t_0 &\leq g^2(t) \end{aligned}$$

Or l'on sait que :

$$t \leq t_0 < g(t) \tag{7.9}$$

On obtient donc :

$$g(t) \leq t_0 < g(t) \tag{7.10}$$

Ce qui constitue une contradiction évidente. Donc en résumé:

$$\forall t \in [0,1] \quad y(t) \neq 0 \Rightarrow y(t)y(g(t)) \geq 0 \text{ et } \forall t' \in [t, g(t)] \quad y(t') \neq 0 \tag{7.11}$$

**3)** Supposons qu'il existe $t_1$ dans l'intervalle $[0,1]$ tel que :

$$M_0(C_1(t_1)) = M_0(C_2(t_1)) \tag{7.12}$$

Posons pour simplifier les notations :

$$\begin{aligned} q_1 &= M_0(C_1(t_1)) \\ q_2 &= M_0(C_2(t_1)) \\ q &= M_0([0,1]) \end{aligned}$$

On a par la proposition 1 $q_1 + q_2 = q$ et donc $q_1 = q_2 = \frac{q}{2}$. De plus :

$$\begin{aligned} \mu_1(t_1) &= \frac{2}{q}\int_0^{t_1} xf(x)dx \\ \mu_2(t_1) &= \frac{2}{q}\int_{t_1}^{1} xf(x)dx \end{aligned}$$

Donc :

$$g(t_1) = \frac{\mu_1(t_1) + \mu_2(t_1)}{2} = \frac{1}{q}\int_0^1 x(fx) = \mu \tag{7.13}$$

Réciproquement, une transposition des résultats du corollaire 1 aux notations utilisées dans cette démonstration donne :

$$\begin{aligned} q^2(\mu - \mu_1(t_1))^2 &= q_2^2(\mu_2(t_1) - \mu_1(t_1))^2 \\ q^2(\mu - \mu_2(t_1))^2 &= q_1^2(\mu_1(t_1) - \mu_2(t_1))^2 \end{aligned}$$

Donc,

$$\frac{(\mu - \mu_2(t_1))^2}{q_1^2} = \frac{(\mu - \mu_1(t_1))^2}{q_2^2} \tag{7.14}$$

Or par hypothèse, $\mu$ est à égale distance de $\mu_1$ et $\mu_2$, donc l'on obtient $q_1 = q_2$ ce qui constitue l'égalité recherchée.

**4)** Soit $\boldsymbol{E}(t)$ l'erreur quadratique de la partition (voir définition 6), on a :

$$\boldsymbol{E}(t) = \int_0^t (x-\mu_1(t))^2 f(x)dx + \int_t^1 (x-\mu_2(t))^2 f(x)dx \tag{7.15}$$

l'erreur quadratique définie par une fonction de fréquence continue est dérivable et admet un minimum sur l'intervalle $[0,1]$. Un simple calcul de dérivée nous donne :

$$E'(t) = 2f(t)(\mu_1(t)-\mu_2(t))(\frac{\mu_1+\mu_2}{2}-t)$$
$$E'(t) = 2f(t)(\mu_1(t)-\mu_2(t))y(t)$$

La fonction $f$ est toujours positive ou nulle, de plus on a par la proposition 1 :

$$\forall t \in [0,1] \quad 0 \leq \mu_1(t) \leq t \leq \mu_2(t) \tag{7.16}$$

Donc, un minimum de $\boldsymbol{E}(t)$ ne peut être atteint que pour un zéro de $y(t)$ accompagné d'un changement de signe. Or, un prolongement par continuité des fonctions $\mu_1$ et $\mu_2$ par l'inégalité 7.16 permet d'établir que:

$$\mu_1(0) = 0 \text{ et } \mu_2(0) = \mu$$
$$\mu_1(1) = \mu \text{ et } \mu_2(1) = 1$$

Donc, en utilisant la proposition 2, il vient :

$$\begin{array}{llll} y(0) = \frac{\mu}{2} > 0 & \Rightarrow \; y(\frac{\mu}{2}) \geq 0 & \text{et } \forall t' \in [0, \frac{\mu}{2}[ & y(t') \neq 0 \\ y(1) = \frac{\mu-1}{2} < 0 & \Rightarrow \; y(\frac{\mu+1}{2}) \leq 0 & \text{et } \forall t' \in ]\frac{\mu+1}{2}, 1] & y(t') \neq 0 \end{array}$$

Donc, $y$ ne peut s'annuler que sur l'intervalle $[\frac{\mu}{2}, \frac{\mu+1}{2}]$. Comme l'on sait par ailleurs que l'erreur quadratique admet au moins un minimum sur $[0,1]$, on peut affirmer que la valeur optimale de $t$ appartient à l'intervalle $[\frac{\mu}{2}, \frac{\mu+1}{2}]$.

**5)** évident, puisque l'on utilise une distribution continue, et que par conséquent $g$ et $y$ sont deux fonctions continues.

## 7.8 Éléments de théorie des cartes

Nous allons dans cette section donné quelques lemmes, théorèmes et propositions utiles pour comprendre et manipuler des cartes combinatoires. Afin d'alléger les notations nous noterons $\sigma^*(b) \cap f$ l'ensemble des brins appartenant simultanément au sommet $\sigma^*(b)$ et à la face $f$.

Nous allons commencé par démontrer un théorème très général permettant de caractériser les brins doubles.

**Théorème 10** *Soit $(\varphi, \alpha)$ une carte combinatoire, $b$ un brin de cette carte et $f$ une face de $\boldsymbol{\mathcal{F}}(\varphi)$, alors :*

$$|\sigma^*(b)|_f \geq 3 \Rightarrow \exists\{b_1, b_2\} \subset \sigma^*(b) \cap f \;/\; b_1 \neq b_2 \textit{ et } \{-b_1, -b_2\} \subset f$$

**Preuve:**

On a :

$$|\sigma^*(b)| \leq 4$$
$$|\sigma^*(b)| \geq |\sigma^*(b)|_f \geq 3$$

Donc : $|\sigma^*(b)| \in \{3, 4\}$.

- Si $|\sigma^*(b)| = 3$.

  $$\sigma^*(b) = (b_1, b_2, b_3) \text{ avec } \{b_1, b_2, b_3\} \subset f$$

  Donc :
  $$\left.\begin{array}{l} \varphi^{-1}(b_2) = -\sigma^{-1}(b_2) = -b_1 \\ \varphi^{-1}(b_3) = -\sigma^{-1}(b_3) = -b_2 \end{array}\right) \Rightarrow \left( \begin{array}{l} -b_1 \in \varphi^*(b_2) = f \\ -b_2 \in \varphi^*(b_3) = f \end{array}\right.$$

  On a donc : $b_1 \neq b_2$ et $\{-b_1, -b_2\} \subset f$

- Si $|\sigma^*(b)| = 4$.
  Puisque $|\sigma^*(b)|_f \geq 3$, les éléments de $\sigma^*(b) \cap f$ seront forcément consécutifs dans le cycle $\sigma^*(b)$, on a donc :

  $$\sigma^*(b) = (b_1, b_2, b_3, b_4) \text{ avec } \{b_1, b_2, b_3\} \subset f$$

  On applique alors le même raisonement que pour $|\sigma^*(b)| = 3$ ce qui nous fournit le même résultat.

□

**Corollaire 3** *Soit $(\varphi, \alpha)$ une carte et $f$ une face de cette carte. Si $f$ ne possède pas de brins double, on a :*
$$\forall b \in f \ |\sigma^*(b)|_f \leq 2$$

**Preuve:**

Si la face ne comporte pas de brins double on ne peut pas trouver deux brins distincts $b_1$ et $b_2$ tels que $\{b_1, b_2\} \subset \sigma^*(b) \cap f$ et $\{-b_1, -b_2\} \in f$. La contraposé du téorème 10 nous fournit alors le résultat, a savoir :
$$|\sigma^*(b)|_f \leq 2$$

□

**Lemme 4** *Soit $(\varphi, \alpha)$ une carte combinatoire, $b$ un brin de la carte et $f$ une face de $(\varphi, \alpha)$. Si $f$ ne possède pas de brins doubles on a :*

$$\{b_1, b_2\} \subset \sigma^*(b) \cap f \Rightarrow \sigma(b_1) \neq b_2 \text{ et } \sigma(b_2) \neq b_1$$

*Autrement dit, si l'on supprime les brins doubles d'une face, les brins de celle ci ne peuvent être consécutifs sur un sommet.*

**Preuve:**

Si $\sigma(b_1) = b_2$ alors $\varphi^{-1}(b_2) = -\sigma_{-1}(b_2) = -b_1$. Le brin $b_1$ est donc un brin double de $f$ ce qui est rejeté par hypothèse. □

**Corollaire 4** *Sous les même hypothèse :*

$$\{b_1, b_2\} \subset \sigma^*(b) \cap f \Rightarrow |\sigma^*(b)| = 4$$

**Preuve:**

Puisque $b_1$ et $b_2$ ne sont pas consécutifs dans le cycle $\sigma^*(b)$, il existe forcément deux brins s'intercalant entre $b_1$ et $b_2$. On a par exemple :

$$\sigma^*(b) = (b, b_1, b', b_2)$$

Le sommet $\sigma^*(b)$ comporte donc bien quatre brins. □

### 7.8.1 Preuve du théorème 8

Il sagit de démontrer la proposition suivante :

**Proposition 9** *Soit $(\varphi, \alpha)$ un carte, $f$ une face de cette carte et $b$ un brin de $f$. Si $f$ ne comporte pas de brins double on a :*

$$|\sigma^*(b)| = 4 \textit{ et } |\sigma^*(b)|_f \in \{2, 3\} \Leftrightarrow |\sigma^*(b)|_f = 2$$

**Preuve:**

On sait, par le corollaire 3 que si $f$ ne comporte pas de brins double on a : $|\sigma^*(b')|_f \leq 2$ pour tout brin $b'$ de $f$. Il suffit donc de démontrer que :

$$|\sigma^*(b)| = 4 \text{ et } |\sigma^*(b)|_f = 2 \Leftrightarrow |\sigma^*(b)|_f = 2$$

La première implication est évidente, il nous suffit donc de montrer que :

$$|\sigma^*(b)|_f = 2 \Rightarrow |\sigma^*(b)| = 4$$

Or cette dernière implication nous est donnée par le corollaire 4. □

### 7.8.2 Plusieurs propositions ayant trait aux cartes combinatoires

**Lemme 5** *Soit* $(\varphi, \alpha)$, *une carte combinatoire et* $b$ *un brin de cette carte. Si le sommet* $\sigma^*(b)$ *ne comporte aucun brin double alors :*

$$-b \in \sigma^*(b) \Rightarrow \sigma(b) = -b \text{ ou } \sigma(-b) = b$$

*Autrement dit, si l'on exlue les brins doubles et si deux brins opposés appartiennent aux même sommet alors ils sont consécutifs dans le cycle du sommet.*

**Preuve:**

Supposons que $\sigma(b) \neq -b$ et $\sigma(-b) \neq b$. On a alors :

$$\sigma^*(b) = (b, b_1, -b, b_2)$$

Donc :

$$b_1 = \varphi(-b) = \sigma(b) \Rightarrow b_1 \in \varphi^*(-b)$$

De même :

$$\varphi(-b_1) = \sigma(b_1) = -b \Rightarrow -b_1 \in \varphi^*(-b)$$

$b_1$ est donc un brin double de $\sigma^*(b)$ ce qui est contraire à l'hypothèse. $\Box$

**Corollaire 5** *Sous les mêmes hypothèses :*

$$-b \in \sigma^*(b) \Rightarrow |\varphi^*(b)| = 1 \text{ ou } |\varphi^*(-b)| = 1$$

*Donc un des brins* $b$ *ou* $-b$ *défini une face.*

**Preuve:**

Si $\sigma^*(b) = -b$ on a :

$$\varphi(-b) = \sigma(b) = -b \Rightarrow \varphi(-b) = (-b)$$

$\Box$

**Lemme 6** *Soit* $(\varphi, \alpha)$ *une carte,* $b$ *un brin de cette carte et* $f$ *une face de* $\boldsymbol{\mathcal{F}}(\boldsymbol{\varphi})$. *On a :*

$$|\sigma^*(b)|_f = |-\sigma^*(b)|_f$$

$-\sigma^*(b)$ *désigne ici, l'opposé de l'ensemble des brins de* $\sigma^*(b)$.

**Preuve:**

- Montrons tout d'abord que : $\varphi^{-1}(\sigma^*(b) \cap f) \subset (-\sigma^*(b)) \cap f$.
  Soit $b \in \sigma^*(b) \cap f$, on a : $\varphi^{-1}(b) = -\sigma^{-1}(b)$. Donc :

  $$\varphi^{-1}(b) \in -\sigma^*(b) \cap \varphi^*(b) = (-\sigma^*(b)) \cap f$$

  On a donc :

  $$\varphi^{-1}(\sigma^*(b) \cap f) \subset (-\sigma^*(b)) \cap f$$

- Réciproquement, montrons que $\varphi((-\sigma^*(b)) \cap f) \subset \sigma^*(b) \cap f$
  Soit $b' \in (-\sigma^*(b)) \cap f$, on a $b' = -b''$ avec $b'' \in \sigma^*(b)$. De plus :
  $$\varphi(b') = \sigma(-b') = \sigma(b'') \Rightarrow \varphi(b') = \sigma(b'') \in \sigma^*(b) \cap f$$
  On a donc :
  $$\varphi((-\sigma^*(b)) \cap f) \subset \sigma^*(b) \cap f$$

$\varphi$ est une fonction bijective. Les deux inclusions ci dessus montrent que sa restriction à $(-\sigma^*(b)) \cap f$ définie une bijection de $(-\sigma^*(b)) \cap f$ sur $\sigma^*(b) \cap f$. On a donc :

$$|\sigma^*(b) \cap f| = |(-\sigma^*(b)) \cap f|$$

Autrement dit :

$$|\sigma^*(b)|_f = |-\sigma^*(b)|_f$$

□

**Théorème 11** *Soit $(\varphi, \alpha)$ une carte, $f$ une face de cette carte et $b$ un brin de $f$. Si $f$ ne comporte pas de brins doubles on a :*

$$|\sigma^*(b)|_f = 2 \Rightarrow \forall b' \in \sigma^*(b) \cap f - \sigma(b') \in f$$

*Dis autrement, si une face "part" d'un sommet avec un brin $b'$ elle y reviendra par le brin $-\sigma(b')$.*

**Preuve:**

Soient $b_1$ et $b_2$ les deux brins de $\sigma^*(b) \cap f$, on a d'aprés le corollaire 4 deux autres brins $b_3$ et $b_4$ tels que :

$$\sigma^*(b) = (b_1, b_3, b_2, b_4)$$

De plus d'aprés le lemme 6 : $|(-\sigma(b)) \cap f| = 2$. La face $f$ ne comportant pas de brins double on a $-b_1 \notin f$ et $-b_2 \notin f$. Donc :

$$(-\sigma^*(b)) \cap f = \{-b_3, -b_4\} = \{-\sigma(b_1), -\sigma(b_2)\}$$

Les deux brins $-\sigma(b_1)$ et $-\sigma(b_2)$ appartiennent donc à f. □