Titre: Title:	Restauration d'image et champs markoviens : étude de deux méthodes
Auteur: Author:	Jean-Michel Durocher
Date:	1989
Type:	Mémoire ou thèse / Dissertation or Thesis
Référence: Citation:	Durocher, J.-M. (1989). Restauration d'image et champs markoviens : étude de deux méthodes [Mémoire de maîtrise, Polytechnique Montréal]. PolyPublie. https://publications.polymtl.ca/58222/

UNIVERSITE DE MONTREAL

RESTAURATION D'IMAGES ET CHAMPS MARKOVIENS: ETUDE DE DEUX METHODES

par

Jean-Michel Durocher

DEPARTEMENT DE MATHEMATIQUES APPLIQUEES

ECOLE POLYTECHNIQUE

MEMOIRE PRESENTE EN VUE DE L'OBTENTION

DU GRADE DE MAITRE ES SCIENCES APPLIQUEES (M.Sc.A.)

Septembre 1989

UNIVERSITE DE MONTREAL

ECOLE POLYTECHNIQUE

Ce mémoire intitulé:

RESTAURATION D'IMAGES ET CHAMPS MARKOVIENS:

ETUDE DE DEUX METHODES

présenté par: Jean-Michel Durocher

en vue de l'obtention du grade de: Maître es sciences appliquées

a été dûment accepté par le jury d'examen constitué de:

M. Gilles Deslauriers, Ph.D., président

M. Mario Lefebvre, Ph.D.

M. Claude Tricot, D. D'état.

Sommaire

La restauration d'images par ordinateur est un problème complexe et, pour le résoudre, on a recours à des outils mathématiques de plus en plus intéressants et puissants. Nous étudions ici deux méthodes de restauration; la première, l'ICM de Besag, est basée sur une nouvelle approche par l'analyse bayesienne, la seconde, de Derin et Elliott, repose sur une application de la programmation dynamique. Toutes les deux utilisent la notion de champs markoviens, un modèle aux propriétés locales particulièrement adaptées aux images, et se veulent des approximations du maximum *a posteriori* (MAP), qui permet de s'attaquer à la restauration d'images en présence de bruit aléatoire.

Les deux méthodes sont étudiées et expliquées en détail et ont été implantées sur ordinateur. Afin de les comparer, nous proposons de nouvelles mesures permettant de juger de la précision d'une restauration. En particulier, nous considérons le Laplacien qui permet de donner un poids plus important aux pixels des contours. Nous avons de plus modifié l'ICM de Besag pour tenir compte du bruit multiplicatif, de dégradations non-linéaires et d'une fonction d'étalement ponctuelle.

On constate que la seconde méthode est la moins intéressante. Sa performance pour des images de plus de deux couleurs est décevante, en plus d'encourir un temps de traitement assez long. L'ICM de Besag par contre est très rapide et donne d'excellents résultats. Cette technique devrait constituer un nouvel outil très intéressant.

Abstract

Digital image restoration is a complex problem that requires more and more complex and powerful mathematical tools. We present here two techniques. The first one, by J. Besag, is Bayesian in nature while the second one by Derin & Elliott is an application of dynamic programming. Both methods rely on Markov random fields, a recent model proposed for its local properties well-suited for image analysis, and both are approximations of the maximum *a posteriori* (MAP) estimate, which allows restoration with random noise.

The two methods are studied and explained in detail. The new method by Besag will be modified to take into account more complex degradations like multiplicative noise, non-linear distorsion, and the effects of a point spread function. We have written the programs to test these techniques and we discuss the results obtained. New error measures are proposed, particularly an error rate that gives a different weight to boundary points using the Laplacian.

The second method becomes less interesting and especially slow for images with more than 2 colors while giving poor restorations. On the contrary, ICM by Besag is very fast and gives excellent results. This new technique could become a major tool in computer image analysis.

Remerciements

Mes remerciements vont premièrement à mon directeur, M. Marc Moore, Ph.D., qui sut me diriger adéquatement tout au long de ma maîtrise, ainsi qu'à mon co-directeur, M. Mario Lefebvre, Ph.D.. Ils ont toujours été disponibles pour répondre à mes questions et leurs commentaires et suggestions pertinentes m'ont souvent été d'un grand secours.

Un mémoire ne s'écrit jamais en un jour ni tout à fait seul. Je voudrais remercier Sophie pour ses nombreuses corrections et lectures successives et pour sa patience. Je tiens également à remercier ma mère et mon père.

Table des matières

Sommaire iv

Abstract v

Remerciements vi

Listes des symboles et des abréviations xii

Liste des figures xviii

Liste des tableaux xxii

CHAPITRE 1: INTRODUCTION 1

CHAPITRE 2: GENERALITES SUR LA RESTAURATION D'IMAGES 6

2.1 Restauration par ordinateur 6

2.2 Formation d'images 8

2.3 Exemples de systèmes de formation d'images 9

2.4 Dégradation d'images 9

2.5 Modélisation 11

2.6 La fonction de transfert 14

2.7 Le bruit dans la restauration 15

2.8 Filtrage inverse 17

2.9 Filtre de Wiener ou des moindres carrés 19

2.10 Formulation dans le cas discret 22

2.11 Maximum a posteriori (MAP) 24

2.12 Autres méthodes 26

CHAPITRE 3: LES CHAMPS MARKOVIENS ET LA DISTRIBUTION DE GIBBS . . . 28

3.1 Processus stochastique markovien 29

3.2 Champs aléatoires 31

3.3 Notions de voisinages et de cliques 34

3.4 Champ de Markov et distribution de Gibbs 37

3.5 Modèle de Ising et autres modèles 42

CHAPITRE 4: L'ICM de Besag 50

4.1 Hypothèses de base et maximum a posteriori 50

4.2 Iterated Conditional Modes (ICM) 54

4.3 Modèles 56

4.4 Détériorations complexes 60

4.5 Reconstruction par blocs 68

4.6 Implantation sur ordinateur 72

4.6.1 Considérations techniques 72

4.6.2 Synchronisation de la mise à jour 73

4.6.3 Choix de ß et de σ^2 76

4.6.4 Modèles de dégradations 77

4.6.5 Code de couleur 78

4.6.6 Exemples et résultats 79

4.6.6.1 Mona Lisa modifiée: convergence et choix de ß 79

4.6.6.2 Chromosomes modifiés, bruit uniforme et «vote de la majorité» 82

4.6.6.3 Image TT et fonction d'étalement ponctuelle 86

4.6.6.4 Image TT et bruit gaussien multiplicatif 90

4.6.6.5 A propos de la vitesse 91

CHAPITRE 5: DERIN ET ELLIOTT ET LA PROGRAMMATION DYNAMIQUE 93

5.1 Maximum a Posteriori 93

5.2 Modèles 94

5.3 Formulation récursive 97

5.4 Algorithme 101

5.5 Programmation dynamique 103

5.6 Exemples de réalisations de champs markoviens 106

5.7 Exemples de restaurations 111

CHAPITRE 6: CRITERES DE COMPARAISON 119

6.1 Pourcentage de pixels mal assignés 124

6.2 Image des différences ou «plan d'erreur» 126

6.3 Blocs de pixels 127

6.4 Pourcentage avec Laplacien 128

6.5 Autres suggestions 135

CHAPITRE 7: PROGRAMMES ET ALGORITHMES 138

7.1 Ordinateur et langage 139

7.2 Programme «besag1.exe» 140

7.2.1 Description 140

7.2.2 Mise en oeuvre et utilisation 140

7.2.3 Algorithmes et structure interne 147

7.3 Programme «derin1.exe» 152

7.3.1 Description 152

7.3.2 Utilisation 153

7.3.3 Structure interne 157

7.4 Autres programmes et considérations techniques 157

Conclusion 159

Bibliographie 163

ANNEXES: FICHIERS SOURCES A.1

BESAG1.PAS A.1

DERIN1.PAS A.6

ICM.PAS A.11

DERIN.PAS A.21

D_MISC.PAS A.23

DF_GEN.PAS A.25

D_GEN.PAS A.28

D_ESTIM.PAS A.33

DF_REST.PAS A.37

D_REST.PAS A.40

IMAG.PAS A.47

TIMAGE.PAS A.76

TAUX.PAS A.76

Listes des symboles et des abréviations

$\in$: Appartient à.

$\notin$: N'appartient pas à.

$\cong$: Approximativement égal à.

$\otimes$: Convolution.

∂: Dérivée partielle.

$\neq$: Différent de.

$=$: Egal à.

$\exists$: Il existe.

∇: Gradient.

$\subset$: Inclus dans.

$\subseteq$: Inclus ou égal à.

$\cap$: Intersection.

∇^2: Laplacien.

$[\nabla^2]$: Matrice du Laplacien avec facteur négatif.

$[\nabla^2]$: Matrice du Laplacien avec facteur positif (habituel).

Θ: Opérateur mathématique général.

$\prod$: Produit.

$\propto$: Proportionnel à.

$\forall$: Quelque soit.

Σ: Somme.

$\sim$: Suit une loi.

$\cup$: Réunion.

$|\ |$: Valeur absolue.

α: Paramètre affecté aux cliques de 1 pixel dans les modèles de champs markoviens (ou de Gibbs).

β: Paramètre affecté aux cliques de 2 pixels dans les modèles de champs markoviens (ou de Gibbs).

Γ: Ensemble des couleurs.

Γ_p: Rapport de la densité de puissance du bruit sur celle du signal.

δ_i: Ensemble des pixels voisins du site i. Voir $\mathcal{G}_i$.

$[\theta_b]$: Matrice de covariance de *b*.

$[\theta_f]$: Matrice de covariance de *f*.

θ_t: Fonction comptant les erreurs associées au $t^{ième}$ taux d'erreur τ_t suggéré.

μ: Moyenne d'une variable aléatoire.

π: Indice de persistance.

$\pi^{q,r}$: Nombre de paires distinctes de pixels voisins entre eux de couleurs k_q et k_r différentes.

π_B: Nombre total de paires distinctes de pixels voisins entre eux dont au moins un des deux appartient au bloc *B*.

π_{dB}: Nombre de paires distinctes de pixels voisins entre eux de couleurs différentes dont au moins un des deux appartient au bloc *B*.

π_{mB}: Nombre de paires distinctes de pixels voisins entre eux de couleurs identiques dont au moins un des deux appartient au bloc *B*.

σ^2: Variance d'une variable aléatoire.

τ_t: Taux d'erreur *t*.

Φ: Fonction de distorsion non-linéaire de $\mathbb{R} \dashrightarrow \mathbb{R}$.

$[\Phi_b]$: Matrice des dérivées partielles de Φ.

Ω: Ensemble des réalisations d'images possibles.

A: Sous-région de *S*.

ASYN: Balayage asynchrone (mise à jour au fur et à mesure).

b: Fonction de $\mathbb{R}^2 \dashrightarrow \mathbb{R}$ représentant le bruit additif gaussien.

b: Champ aléatoire de *b*.

B: Selon le contexte:

- Transformée de Fourier de *b*;
- Bloc de pixels.

c: Nombre de couleurs.

C: Clique.

$\mathcal{C}$: Système de cliques.

$\mathcal{C}^j$: Ensemble de cliques formées de pixels situés uniquement dans la colonne j.

$\mathcal{C}^{j-1,j}$: Ensemble de cliques formées de pixels situés uniquement dans la colonne $j-1$ ou dans la colonne j.

$\mathcal{C}_1$: Selon le contexte:

- Ensemble de cliques contenant le pixel i;
- Ensemble des cliques de 1ère catégorie.

$\mathcal{C}_t$: Ensemble des cliques de catégorie t.

CAM: Champ aléatoire markovien.

D: Nombre de lignes considérées à la fois.

DG: Distribution de Gibbs.

E{}: Espérance mathématique.

f: Une fonction de $\mathbb{R}^2$ --> $\mathbb{R}$ représentant une image originale.

f_e: Une fonction de $\mathbb{R}^2$ --> $\mathbb{R}$ représentant l'estimé de l'image originale (l'image restaurée).

f_m: Espérance mathématique de f.

f: Fonction de densité d'une variable continue.

f_i: Fonction de densité conditionnelle au point i.

$\mathbf{f}$: Champ aléatoire dont une réalisation est f.

$\mathbf{f_e}$: Champ aléatoire dont une réalisation est f_e.

F: Transformée de Fourier de f.

F_e: Transformée de Fourier de f_e.

FEP: Fonction d'étalement ponctuelle.

g: Une fonction de $\mathbb{R}^2$ --> $\mathbb{R}$ représentant une image dégradée.

$\mathbf{g}$: Champ aléatoire dont une réalisation est g.

G: Transformée de Fourier de g.

$\mathcal{S}$: Système de voisinages.

$\mathcal{S}^n$: Système de voisinages du $n^{ième}$ ordre.

$\mathcal{S}_i$: Ensemble des pixels voisins du site i. Voir δ_i.

GA: Bruit gaussien additif.

GM: Bruit gaussien multiplicatif.

h: Une fonction de $\mathbb{R}^2$ --> $\mathbb{R}$ représentant une fonction d'étalement ponctuelle invariante dans l'espace.

H: Transformée de Fourier de h (la fonction de transfert d'entrée).

[H]: Matrice brouillante représentant l'échantillonage de la fonction d'étalement ponctuelle.

i: Généralement un pixel (site ou point d'une image).

ICM: Iterated Conditionnal Modes.

k: Généralement une couleur assignée à un point.

ln: Logarithme naturel.

$\mathcal{L}$: Loi de probabilité conjointe.

M: Fonction dans le domaine de Fourier servant de fonction de transfert de sortie (le filtre de la restauration).

MAP: Maximum *a posteriori*.

n: Nombre de sites de la grille *S*.

n_i: Nombre total de voisins du site *i*.

n^k: Nombre de pixels de couleur *k*.

n_b: Nombre de blocs dans la grille *S*.

$\mathbb{N}$: Espace des entiers naturels positifs (0,1,2,3...).

N_1: Nombre de rangées dans une image.

N_2: Nombre de colonnes dans une image.

NL: Distorsion non-linéaire.

p: Fonction de masse d'une variable discrète.

p_i: Fonction de masse conditionnelle au point *i*.

{p(x)}: Champ aléatoire (généralement markovien). Voir *X*.

P[]: Probabilité

r_i: Selon le contexte:

- Ensemble des pixels contenant le site *i* dans leurs abords;
- Etape *i* du processus récursif de programmation dynamique.

$\mathbb{R}$: Espace des réels.

S: Ensemble des sites (pixels) formant généralement une grille rectangulaire.

S_{bb}: Densité spectrale du bruit.

S_{ff}: Densité spectrale de l'image originale *f*.

S\i: Ensemble des sites à l'exclusion du site *i*.

S\B: Ensemble des sites à l'exclusion des pixels du bloc *B*.

SYN: Balayage synchrone (mise à jour simultanée).

SSYN2: Semi-synchrone (mise à jour une ligne sur deux, un pixel sur deux).

$u_i(k)$: Nombre de voisins du site *i* de couleur *k*.

U(x): Fonction d'énergie (distribution de Gibbs).

UNI: Bruit uniforme additif.

v_i: Nombre de voisins du site *i* de couleur 1.

$V_C(x)$: Potentiel associé à une clique *C* (distribution de Gibbs).

x: Réalisation de *X* (généralement une image).

$\hat{x}$: Estimé de l'image idéale.

x*: Image idéale.

x^-: Image obtenue en appliquant $[\nabla^2]$ sur *x*.

x^+: Image obtenue en appliquant $[\nabla^2]$ sur *x*.

x^d: Image des différences (plan d'erreur).

$x_{\partial i}$: Couleurs de tous les pixels des abords du site *i*.

$x_{\delta i}$: Couleurs de tous les pixels voisins du site *i*.

x_i: Couleur du pixel *i*.

$\hat{x}_i$: Estimé de la couleur du pixel *i*.

x_B: Couleurs des pixels du bloc *B*.

$x_{S\setminus i}$: Couleurs de tous les points d'une image à l'exception du site *i*.

$x_{S\setminus B}$: Couleurs de tous les points d'une image à l'exception des sites du bloc *B*.

X: Processus stochastique (généralement un champ markovien).

X_i: Variable (ou vecteur) aléatoire au point *i*.

y: Réalisation de Y (généralement une image ou des observations).

Y: Processus stochastique (généralement un champ markovien).

Z: Constante de normalisation appelée fonction de répartition (distribution de Gibbs).

$\mathbb{Z}$: Espace des entiers {...,-2,-1,0,1,2,...}.

Note:

- Nous utilisons indifféremment le terme *pixel* ou *site* pour désigner un *point* de la grille S.

Liste des figures

Figure 2.5.1 Modèle de formation d'images avec fonction de dégradation et bruit aléatoire additif. 12

Figure 2.8.1 Modèle de reconstruction d'images avec fonction de dégradation, bruit aléatoire additif et restauration. 18

Figure 3.1.1 Influence locale dans un processus de Markov d'ordre r. 30

Figure 3.3.1 Système de voisinages: (a) voisinage du 1er ordre formé des 4 pixels les plus proches (notés *) à l'horizontale et à la verticale; (b) voisinage du 2ième ordre formé des pixels du voisinage du 1er ordre plus les 4 pixels aux diagonales; (c) voisinages jusqu'à l'ordre 6, chaque ordre conservant toujours les pixels des ordres inférieurs. 36

Figure 3.3.2 Famille de cliques (a) du système du 1er ordre possèdant 1 pixel ou 2 pixels adjacents à l'horizontale ou à la verticale; (b) du système du 2ième ordre possédant de 1 à 4 pixels, notées $C_1,\ldots,C_{10}$. 37

Figure 3.5.1 Notation des pixels du voisinage du 1er ordre autour du site (i,j) pour le modèle de Ising. 42

Figure 3.5.2 Exemple de configuration $x_o \in \Omega$ pour une grille S de 9 pixels avec c=2 couleurs, $\Gamma=\{0,1\}$. On compte 6 points égaux à 1, 3 paires de pixels successivement égaux à 1 horizontalement et 2 paires verticales de pixels de valeur 1. On trouve alors que la fonction d'énergie du modèle de Ising est $U(x_o) = 6A+(3+2)\beta$. 44

Figure 3.5.3 Notation des pixels d'un voisinage autour du pixel i pour le modèle de champ markovien (3.5.9) donnant des poids différents aux pixels des diagonales. 49

Figure 4.4.1 Exemple de matrice [H] correspondant à une fonction d'étalement ponctuelle invariante par rapport à la translation et la rotation. 64

Figure 4.5.1. Bloc de 4 pixels donnant π_B = 8x4-6=26 paires distinctes de pixels: (a) chaque pixel du bloc fait partie de 8 cliques de 2 pixels; (b) les paires formés de 2 pixels appartenant au bloç - ceux marqués par un x - apparaissent 2 fois. 71

Figure 4.6.1 Ordre des visites de l'ICM lors d'un balayage classique de gauche à droite et de haut en bas. Au site intérieur (i,j), les pixels notés + ont déjà été mis à jour, les pixels notés - ne l'ont pas encore été. 74

Figure 4.6.2 Ordre de passage d'un balayage semi-synchrone. A chaque étape les voisins sont répartis symétriquement. 75

Figure 4.6.3 Restauration par l'ICM et convergence: **(a)** Image originale «Monal Lisa modifiée»; **(b)** dégradation **GA**, σ^2=0.5, donnant un taux d'erreur de 48.46%; **(c)** restauration ICM-SYN avec modèle GA, ß=1.5 constant, 10 itérations, err=9.40%; **(d)** avec ß=1.0, err=8.76%; **(e)** avec ß=2.0, err=9.82%; **(f)** courbe de convergence 81

Figure 4.6.4 Restauration par l'ICM avec UNI: **(a)** Image originale «Chromosome modifié»; **(b)** dégradation **GA**, σ^2=0.3, donnant un taux d'erreur de 36%; **(c)** restauration ICM-SYN avec modèle GA, ß=1.2 constant, 6 itérations, err=5.3%; **(d)** dégradation **UNI**, err=33%; **(e)** ICM-SYN GA, ß=1.2, 6 itérations, err=5.78% **(e)** ICM-SYN GA, ß=1.2, err=5.78% 83

Figure 4.6.5 Restauration par l'ICM et «vote de la majorité»: **(a)** Image originale «Chromosome modifié»; **(b)** dégradation **GA**, σ^2=0.3, donnant un taux d'erreur de 37%; **(c)** restauration «vote de la majorité» (ß=1000), 6 itérations, err=10.0%; **(d)** ICM-SYN GA, ß=0.5 à ß=1.5, 6 itérations, err=5.2% 85

Figure 4.6.6 Restauration par l'ICM avec FEP: **(a)** Image originale «TT»; **(b)** dégradation **FEP** err=43.24%; **(c)** superposition supplémentaire **FEP+GA** σ^2=0.1, err=31.02%; **(d)** restauration ICM-SYN GA, 6 itérations, ß=1.5, err=17.43%; **(e)** ICM-ASYN GA, 6 itérations, ß=1.5, σ^2=0.32, err=11.29%; **(f)** restauration ICM-SYN GA-FEP, 6 itérations, ß=1.5, err=0.49% 87

Figure 4.6.7 Restauration par l'ICM avec GM: **(a)** Image originale «TT»; **(b)** dégradation **GM**, σ^2=0.2, donnant un taux d'erreur de 21.56%; **(c)** restauration ICM-SYN avec modèle GA, ß=1.5 constant, 6 itérations, err=8.38%; **(d)** ICM-SYN GM, ß=1.5, 6 itérations, err=3.08% 91

Figure 5.3.1 Cliques de 1 (pixels notés *) et de deux pixels (liens horizontaux ——, verticaux | ou diagonaux \) pour (a) C^1; (b) $C^{1,2}$. 99

Figure 5.4.1 Algorithme de Derin et Elliott pour des bandes de D=2 lignes: (a) séquence d'estimation des pixels des lignes $i=i_e$ à $i=i_e+1$ (notés *), qui utilise l'information des pixels de la ligne i_e-1 (notés x) et où seule la ligne i_e sera conservée; (b) séquence suivante. 103

Figure 5.5.1 Image simple de 2 rangées par 4 colonnes. 104

Figure 5.6.1 Images markoviennes, modèle (5.2.3)-(5.2.4). **(a)** image aléatoire uniforme de 2 couleurs; **(b)** image générée après 5 itérations de relaxation stochastique sur (a), avec $ß_1=ß_2=1.0$ et $ß_3=ß_4=-1.0$; **(c)** image générée après 15 itérations avec $ß_i=0.3$ (équivalent au modèle de l'ICM); **(d)** même que (c) avec 10 itérations et $ß_i=2.0$; **(e)** image générée à partir d'une image aléatoire de 4 couleurs, après 10 itérations avec $ß_i=2.0$; **(f)** même que (e) avec $ß_1=ß_2=ß_3=1.0$ et $ß_4=-1.0$ 109

Figure 5.7.1 Restauration d'images par programmation dynamique, 2 couleurs: **(a)** image originale «DB24A» ; **(b)** image dégradée par un bruit **GA**, σ^2=0.27, err=27%; **(c)** restauration programmation dynamique, $ß_i=0.4$, D=3, err=8.01%; **(d)** restauration D&E avec $ß_i=0.3$, D=3, err=6.97%; **(e)** restauration D&E avec $ß_i=0.3$, D=2, err=9.31%; **(f)** restauration ICM-SYN GA, ß=1.5, 5 itérations, err=6.06% 113

Figure 5.7.2 Restauration d'images par D&E et ICM avec forte dégradation: **(a)** Image originale «DB24A»; **(b)** dégradation **GA**, σ^2=0.37, donnant un taux d'erreur de 37.2%; **(c)** restauration programmation dynamique $ß_i=0.3$, D=3, err=18.03%; **(d)** res-

tauration ICM-SYN avec modèle GA, ß=1.5 constant, 6 itérations, err=16.58% 115

Figure 5.7.3 Restauration d'images par D&E et ICM, 4 couleurs: **(a)** Image originale 4 couleurs «D29»; **(b)** dégradation **GA**, σ^2=0.55, donnant un taux d'erreur de 38.29%; **(c)** restauration ICM-SYN avec modèle GA, ß=0.5 augmenté par incrément de 0.2 pendant 6 itérations plus ß=1.5 constant pendant 5 it., err=9.69%; **(d)** restauration programmation dynamique $ß_i$=0.35, D=3, err=21.32% 117

Figure 6.1 Etude des taux d'erreur, premier groupe: **(a)** Image originale «TT»; **(b)** dégradation **FEP+GA** σ^2=0.1, err=31.02%; **(c)** restauration ICM-SSYN2 GA-FEP, 6 itérations, ß=1.5 constant, err=1.49%; **(d)** restauration ICM-SYN GA, 6 itérations, ß=1.5 constant, err=14.52%; **(e)** plan d'erreur de (c); **(f)** plan d'erreur de (d) 121

Figure 6.2 Etude des taux d'erreur, second groupe: **(a)** Image originale «DB24A»; **(b)** dégradation **GA** σ^2=0.27, err=27%; **(c)** restauration ICM-SYN GA, 5 itérations, ß=1.5 constant, err=6.06%; **(d)** restauration D&E, $ß_i$=0.3, D=3, err=6.97%; **(e)** plan d'erreur de (c); **(f)** plan d'erreur de (d) 123

Figure 6.4.1 Notation des pixels autour d'un site central (i,j) pour une approximation discrète du Laplacien ∇^2. 130

Figure 6.4.2 Patron des poids d'une approximation discrète du Laplacien ∇^2. 130

Figure 6.4.3 Grille de couleurs pour un exemple de calcul de $[\nabla^2_+]$. 131

Figure 6.4.4 Détection d'arêtes: illustration de l'application de l'opérateur de Laplacien ∇^2. **(a)** application de $[\nabla^2_+]$ sur «TT»; **(b)** application de $[\nabla^2_-]$ sur «TT»; **(c)** image originale «MK»; **(d)** application de $[\nabla^2_-]$ sur «MK»; **(e)** image originale «TA»; **(ff)** application de $[\nabla^2_-]$ sur «TA» 133

Liste des tableaux

Tableau 6.1 Comparaisons entre différentes mesures d'erreur sur quelques images; τ_1: pourcentage de pixels mal assignés; τ_2: pourcentage du nombre de blocs 3x3 incorrects; τ_3: pourcentage du nombre d'erreurs avec poids du Laplacien intérieur; τ_4: pourcentage du nombre d'erreurs avec poids du Laplacien extérieur; τ_5: moyenne des pourcentages du nombre d'erreurs avec poids du Laplacien intérieur et extérieur; τ_6: moyenne des valeurs normalisées des erreurs avec poids du Laplacien intérieur et extérieur. 124

CHAPITRE 1: INTRODUCTION

Pour pouvoir être traitée par un moyen automatique, une image est présentée sous la forme d'une matrice de points, des *pixels*, qui prennent des valeurs dans un ensemble représentant les niveaux de gris, les couleurs, les longueurs d'onde, ou les émissions de rayons X par exemple. Dans tous les cas, ces images numérisées se veulent le reflet d'une scène réelle sous-jacente: images de télévision transmises par satellite, radiographies, photos quelconques. Le but de la restauration d'images et de nettoyer les images corrompues par la présence d'une ou de plusieurs sources de dégradation.

Plusieurs méthodes de restauration d'image ont été développées au cours des vingt dernières années. Elles permettent souvent de très bien nettoyer des images dégradées par des distorsions déterministes, causées par exemple par une lentille hors foyer ou la réponse logarithmique des sels d'argent lors de prise de la photo. En présence de phénomènes aléatoires cependant, la plupart des techniques ne sont plus valides et donnent des restaurations de faible qualité.

La recherche du maximum *a posteriori* (MAP), une méthode statistique, offre un intérêt particulier car elle permet de tenir compte

directement du bruit aléatoire causant la dégradation. Son application engendre cependant deux problèmes majeurs.

Premièrement, elle nécessite une information *a priori* sur le type d'images considérées. Parmi les modèles possibles, celui des *champs markoviens* connaît une certaine popularité en traitement d'images depuis une dizaine d'années, grâce à ses propriétés locales, naturelles pour des images, et surtout depuis que l'on a montré son équivalence avec la *distribution de Gibbs*, facilitant ainsi l'établissement de modèles d'images à partir des lois conditionnelles de chaque pixel.

Le second problème concerne sa vitesse. Pour des images relativement simples, une application directe du MAP pourrait prendre des milliards d'années! Plusieurs méthodes ont donc été suggérées qui obtiennent comme estimé de l'image originale une *approximation* du MAP.

Le but de ce mémoire est d'analyser deux de ces méthodes, l'une basée sur l'analyse bayesienne et l'autre sur la programmation dynamique. Nous les implanterons sur ordinateur en développant les programmes informatiques nécessaires. Pour obtenir les algorithmes adéquats, une étude détaillée de leurs principes théoriques sera nécessaire.

Par la suite nous leur soumettrons des images connues, dégradées par un bruit additif gaussien selon un procédé contrôlé. Pour la méthode bayesienne, nous examinerons également l'implication de

dégradations plus complexes, notamment l'effet d'une fonction d'étalement ponctuelle.

Nous étudierons les images restaurées au niveau de leur précision et du temps de traitement requis. Bien qu'une comparaison détaillée ne sera pas effectuée, nous chercherons des critères permettant de juger de la qualité des images estimées.

La première méthode, l'ICM (*Iterated Conditional Modes*) de Besag [1986], est basée sur une approche bayesienne et une recherche itérative des couleurs maximisant la loi *a posteriori* étant données les observations et la reconstruction en cours.

La deuxième méthode, de Derin et Elliott [1987], repose sur une division du calcul du MAP grâce à un procédé de récursion. Une solution sous-optimale est proposée puis calculée par la programmation dynamique.

Par conséquent, nous viserons six objectifs principaux:

1) présenter le problème de la restauration d'image;
2) présenter le modèle des champs markoviens et de la distribution de Gibbs;
3) étudier la méthode basée sur l'approche bayesienne, et proposer de nouveaux modèles pour tenir compte de dégradations complexes;
4) étudier la méthode basée sur la programmation dynamique;

5) proposer des critères servant à mesurer l'erreur de la restauration;

6) présenter les programmes que nous avons écrits afin d'implanter les algorithmes, créer des images et mesurer la qualité des restaurations.

La structure de ce mémoire est construite de façon à répondre aux objectifs que nous nous sommes fixés. Le chapitre 2 aborde le problème de la restauration d'images et décrit les limites des méthodes déterministes avant de présenter le MAP. Le chapitre 3 expose le modèle des champs markoviens, caractérisés par leurs propriétés locales. Les notions requises de voisinages et de cliques sont expliquées. Nous présentons également la distribution de Gibbs qui nous aidera à donner plusieurs modèles de champs markoviens. Le chapitre 4 décrit en profondeur l'ICM de Besag [1986]. De plus, nous y élaborons le critère de maximisation pour tenir compte de détériorations complexes: bruit multiplicatif, distorsion non-linéaire et fonction d'étalement ponctuelle. Les problèmes de l'implantation sur ordinateur de cette méthode sont également analysés et nous présentons des exemples de restauration où différents aspects de la méthode (choix des paramètres, convergence, robustesse) sont mis en évidence. Le chapitre 5 aborde la seconde des deux méthodes de restauration d'images étudiées. Nous y expliquons comment décomposer le problème du MAP en un schéma récursif et comment appliquer la programmation dynamique pour le résoudre. A partir du modèle proposé par Derin et Elliott [1987] nous présentons des réalisations de champs markoviens puis étudions le comportement de leur

méthode de restauration sur des images soumises à un bruit additif gaussien. Au chapitre 6 nous suggérons de nouveaux critères de mesures d'erreur et de comparaison d'images. La détection d'arêtes par le Laplacien est utilisée afin de donner des poids plus importants aux pixels de contours, c'est-à-dire situés à la frontière de deux régions. Pour terminer, le chapitre 7 donne les informations nécessaires à l'utilisation des programmes et leurs algorithmes et structures générales afin de faciliter d'éventuels développements futurs.

CHAPITRE 2: GENERALITES SUR LA RESTAURATION D'IMAGES

Les premières idées qui nous viennent à l'esprit en pensant au mot "image" sont probablement relatives à la photographie, la télévision, ou encore la peinture. Ces medias, et d'autres que nous mentionnerons, ont comme point commun de servir à la représentation, réelle ou imaginaire, de quelque chose ou de quelqu'un. Lorsque pour une raison ou pour une autre l'image acquise subit une détérioration, il peut être souhaitable de lui faire subir une "restauration". Dans la plupart des cas, on cherchera en effet à obtenir une image possédant la plus grande ressemblance possible avec la "scène" ou la peinture d'origine.

2.1 Restauration par ordinateur

Bien sûr nous ne parlerons pas ici de l'art de la restauration d'un tableau d'un grand peintre de l'école flammande du XVIIe siècle, mais des développements mathématiques du problème de la restauration d'images pouvant être posés sous la forme de variables continues ou discrètes. Ainsi, plusieurs techniques de restauration d'images peuvent être réalisées par un ordinateur soit numérique, soit analogique (optique).

Si le dilemne du choix de l'une ou l'autre méthode peut être posé avec pertinence, nous parlerons uniquement ici de restauration d'images par ordinateur sous forme numérique. Nous considérerons donc le problème d'images discrètes, et référons le lecteur intéressé par le côté purement optique du choix de combinaisons de lentilles et autres obturateurs à Goodman [Goodman:1968]. Notons qu'il demeure possible que les deux branches se recoupent dans des systèmes hybrides digitaux/optiques lors de développements futurs [Casasent:1975].

La première véritable application de la restauration et de l'amélioration d'images par ordinateur date du programme spatial américain du début des années 1960 et dont le but ultime était d'envoyer des hommes sur la lune avant la fin de la décennie. Il fut tout d'abord décidé de faire alunir des engins inhabités qui renvoyaient des images télévisées de la surface de la lune. Toutefois, pour des raisons techniques (e.g. économie de poids), les appareils et donc les images transmises étaient de piètre qualité. Ce fut, semble-t-il, le "Jet Propulsion Laboratory" du "California Institute of Technology" qui eut le mandat de nettoyer le mieux possible les images dégradées reçues de la lune [Andrews et Hunt:1977].

Depuis, l'intérêt pour la restauration d'images par ordinateur n'a cessé d'augmenter et elle est appliquée aujourd'hui dans un grand nombre de champs d'activité. Que ce soit en médecine (radiologie, imagerie par résonnance magnétique ou par médecine nucléaire, échographie, biologie cellulaire et microscope à électrons), en contrôle de la

qualité et inspection industrielle, en météorologie, en exploration de ressources naturelles, ou encore en transmission et prise d'images par satellite, etc., il n'existe pratiquement plus un domaine ayant recours aux images où l'on ne fait pas, sous une forme ou une autre, de la restauration d'images par ordinateur.

2.2 Formation d'images

Sans vouloir entrer dans les détails physiques, optiques ou électroniques des systèmes d'acquisition d'images, en radiologie ou en télévision par exemple, certaines caractéristiques générales doivent être décrites.

Bien que l'on puisse vouloir analyser des figures en deux dimensions qui ne sont pas à proprement parler des images dans le sens physique du terme, on appellera image un ensemble de points lumineux formés par des rayons émanant des divers points d'un objet après réfraction ou réflexion recueillis indirectement sur un écran. Cette définition nous permet de conceptualiser le principe de la formation d'images en trois parties. L'*objet*, d'où proviennent les rayons, un *système de formation d'images*, qui intercepte l'énergie propagée et la transforme en une *image*.

Pour plus de facilité cependant, nous parlerons d'une image originale qui correspondrait à l'image *idéale* pouvant être obtenue de l'objet et d'une image observée, l'image *réelle* obtenue du système de

formation d'images. Ce modèle possède l'avantage de parler de deux choses du même type, l'image observée et l'image originale, ce qui permet de s'attaquer au problème de la *restauration d'images*, c'est-à-dire de chercher une *image* (et non un *objet*) nettoyée.

D'une manière générale dans nos discussions, et selon la définition la plus couramment rencontrée, une image sera une grille rectangulaire. Il s'agit d'une matrice de points appelés pixels possédant des valeurs réelles possiblement entières, représentant en général une intensité lumineuse.

2.3 Exemples de systèmes de formation d'images

Une carte de saisie d'image en noir et blanc standard, qui numérise un signal vidéo provenant d'une caméra de télévision, donne une image de 512 lignes par 512 colonnes où chaque pixel est un entier prenant une valeur entre 0 (noir) et 256 (blanc). Dans le cas d'une image couleur, chaque pixel est un vecteur formé des composantes du rouge, du bleu et du vert dont le mélange donnera une couleur parmi 32768 couleurs possibles.

2.4 Dégradation d'images

Chaque système possède sa propre façon de procèder mais il est possible de fournir un modèle général en tenant compte des différentes parties et de tous les intermédiaires influençant le processus de

formation de l'image finale. Il peut s'agir des lentilles de la caméra de télévision, du degré de concentration de radio-isotopes injectés ou de la fréquence d'échantillonage - qui jouent un rôle très important dans la résolution effective de l'image, de la vitesse et de l'ouverture de l'obturateur, etc. A tous ces phénomènes physiques liés au système s'ajoutent souvent des effets supplémentaires, généralement imprévus ou incontrôlables, tels que le déplacement de l'objet devant la caméra, des perturbations atmosphériques en photographie aérienne ou par satellite, des erreurs de transmission, les battements de coeur d'un sujet, une lentille hors foyer, et combien d'autres.

Ce sont ces dégradations, détériorations, bruits de toutes sortes qui font que l'image *observée* est loin d'être une image idéale. Certains types de dégradations affectent seulement les niveaux de gris des pixels individuellement, sous la forme de bruit additif. D'autres, que l'on appelle dégradations spatiales, produisent un effet de flou affectant plusieurs pixels en même temps. Certaines produisent des images possédant un contraste insuffisant ou au contraire trop marqué. On rencontre également des déformations chromatiques et des effets temporels.

Dans tous les cas, il est nécessaire de posséder un modèle expliquant le mieux possible tous ces effets sur l'image observée. On cherche alors un modèle mathématique combinant les effets *déterministes*, liés aux propriétés physiques du système de formation d'images, et les phénomènes *aléatoires* de bruit perturbant l'acquisition d'images.

2.5 Modélisation

Etablissons premièrement le concept mathématique d'une image. Nous appellerons "image" une fonction f de $\mathbb{R}^2 \rightarrow \mathbb{R}$ dans le cas continu, toutes les valeurs de $\mathbb{R}$ peuvent être prises; dans le cas discret les valeurs de f sont restreintes à $\mathbb{Z}$ ou $\mathbb{N}$.

Bien que l'on puisse vouloir incorporer un plus grand nombre de paramètres pour tenir compte de tous les effets possibles, le modèle général le plus simple est le suivant. Etant donné une image idéale $f(x,y)$ et l'image dégradée correspondante $g(x,y)$, on suppose que g et f sont reliées par:

$$g(x,y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h(x,y,x',y',f(x',y'))dx'dy' + b(x,y) \qquad (2.5.1)$$

où h est une fonction de dégradation de $\mathbb{R}^5 \rightarrow \mathbb{R}$ et $b(x,y)$ est le bruit aléatoire additif généralement présent dans l'image observée. Pour les cas où le système de formation est linéaire, l'équation devient:

$$g(x,y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h(x,y,x',y')f(x',y')dx'dy' + b(x,y). \qquad (2.5.2)$$

Ce modèle est représenté à la figure 2.5.1.

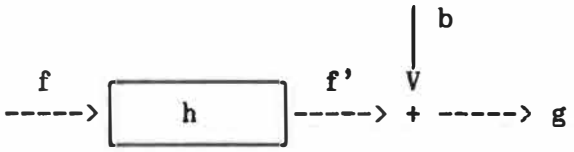


Figure 2.5.1 *Modèle de formation d'images avec fonction de dégradation et bruit aléatoire additif.*

La fonction *h* est appelée la fonction d'étalement ponctuelle et correspond à la réponse du système de formation d'images à un point d'intensité infinie - comme la réponse impulsionnelle, en théorie des communications, correspond à la réponse du système à une fonction de Dirac.

L'hypothèse selon laquelle l'image observée $g(x,y)$ est une fonction linéaire de l'image idéale $f(x,y)$ n'est pas toujours exacte. Dès lors, il faut parfois tenir compte de non-linéarités: les caractéristiques des émulsions photographiques, par exemple. Dans ce cas un modèle plus précis pourrait être:

$$g(x,y) = \Phi\left[\int_{-\infty}^{\infty}\int_{-\infty}^{\infty} h(x,y,x',y')f(x',y')dx'dy'\right] + b(x,y) \qquad (2.5.3)$$

où la fonction Φ représente la transformation non-linéaire du processus d'enregistrement.

Nous tiendrons brièvement compte de la non-linéarité, introduite par la fonction Φ, dans le chapitre 4, à l'aide de quelques exemples. Sauf mention explicite, nous supposons qu'il s'agit de la fonction identité dans tous les autres cas.

La seconde hypothèse selon laquelle le bruit aléatoire est additif peut également être mise en question: dans certains cas le bruit est multiplicatif. Toutefois, il est parfois possible de le considérer additif sur une petite région dynamique et, comme l'hypothèse d'additivité facilite le traitement mathématique, c'est celle que l'on rencontre le plus souvent. Comme pour la non-linéarité, nous étudierons ses implications au chapitre 4.

D'autre part, si l'influence d'un point (x',y') sur la dégradation en un point (x,y) ne dépend que de la position relative de ces deux points (c'est-à-dire du vecteur les joignant) alors la fonction d'étalement ponctuelle prend la forme $h(x-x',y-y')$ où h est une fonction de $\mathbb{R}^2 \rightarrow \mathbb{R}$. Dans ce cas on dit que la dégradation est <u>invariante</u> par rapport à la translation. Nous nous limiterons à ce type de dégradation. En l'absence de bruit, le modèle devient alors:

$$g(x,y) = \int_{-\infty}^{\infty} \int_{-\infty}^{\infty} h(x-x',y-y')f(x',y')dx'dy' \qquad (2.5.4)$$

ce qui en fait représente la <u>convolution</u> des fonctions h et f, notée

$$g = h \circledast f.$$

D'après le théorème de convolution qui stipule que la transformée de Fourier de la convolution de deux fonctions est égale au produit de leurs transformées de Fourier respectives, en prenant la transformée de Fourier de chaque côté de (2.5.4) on obtient:

$$G(u,v) = H(u,v)F(u,v) \qquad (2.5.5)$$

où $G(u,v)$, $F(u,v)$ et $H(u,v)$ sont respectivement les transformées de Fourier de $g(x,y)$, $f(x,y)$, $h(x,y)$.

La fonction $H(u,v)$ est appelée la fonction de transfert du système de formation d'images qui transforme l'image idéale $f(x,y)$ en une image dégradée $g(x,y)$.

Mathématiquement, la restauration d'image consiste donc, à partir du modèle (2.5.4) et de l'image observée $g(x,y)$, à obtenir un estimé $f_e(x,y)$ aussi bon que possible de la scène d'origine, l'image idéale $f(x,y)$. Il devient bien sûr nécessaire de posséder un certain nombre d'informations sur la fonction $h(x,y)$ et, si le bruit est présent, sur $b(x,y)$.

2.6 La fonction de transfert

Dans certains cas les caractéristiques physiques de la dégradation peuvent être directement utilisées pour déterminer $h(x,y)$. Par exemple, le déplacement relatif entre la caméra et l'objet [Lohmann et Paris:1965], pour un mouvement uniforme à vitesse V dans la direction x pendant un temps T, est représenté par:

$$H(u,v) = \frac{\sin(\pi uVT)}{\pi uV}. \qquad (2.6.1)$$

Toutefois il est souvent nécessaire d'estimer $h(x,y)$ directement à partir des observations $g(x,y)$, lorsque sa forme analytique est trop difficile à obtenir. Ainsi, s'il y a lieu de penser que la scène originale contient un point brillant, l'image de ce point représente la réponse ponctuelle: en astronomie on peut prendre l'image d'une petite étoile par exemple. Il est également possible d'obtenir un estimé de $h(x,y)$ si l'on sait que l'image idéale contient une fine ligne, ainsi que dans d'autres circonstances [Rosenfeld et Kak:1982].

2.7 Le bruit dans la restauration

Pour restaurer une image en présence de bruit il faut connaître, au moins en théorie, les propriétés statistiques du bruit $b(x,y)$ et la façon dont il est corrélé avec l'image. En pratique, l'hypothèse la plus communément utilisée est celle du bruit *blanc*, indépendant de l'image et dont la densité spectrale est constante. Il y a évidemment beaucoup d'exemples où ni l'une ni l'autre de ces conditions n'est remplie, notamment dans le cas du bruit dû au grain du film [O'Neil: 1963].

Les différentes techniques de restauration tenant compte du bruit nécessitent différentes informations *a priori* sur le bruit. Par exemple, parmi les méthodes de restauration d'images que nous allons mentionner rapidement ici, le filtre de Wiener requiert la caractérisation du bruit par sa densité spectrale, le procédé de déconvolution avec contraintes ne demande que la variance du bruit, tandis que le

maximum *a Posteriori* exige la connaissance des lois de probabilités conditionnelles du bruit étant donné l'image originale.

Dans la majorité des cas le modèle statistique du bruit n'est que très vaguement connu. On se contentera très souvent de supposer qu'il suit une loi normale, parfois une loi de Poisson. Toutefois il reste le problème de l'estimation des paramètres de ces lois.

A partir d'une image dégradée et de son image originale, il est possible par exemple de quantifier la variance du bruit par les différences entre les deux images si l'image contient des régions d'intensités uniformes. Si, d'autre part, la variance mesurée dans les régions claires est différente de celle des régions foncées, il est possible alors que le bruit soit de nature multiplicative (et non additive): une meilleure mesure de l'estimé de la variance consisterait à prendre le rapport entre les pixels des deux images plutôt que leur différence.

Une fois les différents "intervenants" dans le processus d'acquisition d'images bien identifiés et paramétrisés, il s'agit de restaurer l'image. Le problème, cependant, ne possède pas de solution unique et plusieurs méthodes ont été explorées. Nous en mentionnons ici quelques-unes.

2.8 Filtrage inverse

Nous avons vu qu'en l'absence de bruit, à partir du modèle (2.5.4), les transformées de Fourier de l'image originale $f(x,y)$ et de l'image dégradée $g(x,y)$ satisfont à:

$$G(u,v) = H(u,v)F(u,v),$$

ou, ce qui est équivalent, à:

$$F(u,v) = G(u,v)/H(u,v). \qquad (2.8.1)$$

Dès lors, si l'on connaît $H(u,v)$, il est théoriquement possible de récupérer $f(x,y)$ en multipliant la transformée de Fourier $G(u,v)$ de l'image dégradée par $1/H(u,v)$ et en effectuant une transformée inverse.

Dans le domaine de Fourier, la fonction appliquée à $G(u,v)$ dans le but d'obtenir un estimé de $F(u,v)$ est appelée le <u>filtre</u> du processus de restauration; il s'agit ici de $1/H(u,v)$.

En pratique cependant, il y a des régions du plan uv où $H(u,v) = 0$ et où, en l'abscence de bruit, la transformée $G(u,v)$ de l'image dégradée est également nulle à ces fréquences, conduisant à des quotients indeterminés.

En présence de bruit on obtient:

$$G(u,v) = H(u,v)F(u,v) + B(u,v)$$

où $B(u,v)$ est la transformée de Fourier du bruit $b(x,y)$, et les zéros de $G(u,v)$ et de $H(u,v)$ ne coïncident pas en général. En appliquant le filtre de la restauration on a:

$$\frac{G(u,v)}{H(u,v)} = F(u,v) + \frac{B(u,v)}{H(u,v)}. \qquad (2.8.2)$$

Dans le voisinage des zéros de la fonction de transfert $H(u,v)$, la division dans l'équation (2.8.2) résulte en de très grandes valeurs. Dans ce cas, la transformée inverse $G(u,v)/H(u,v)$ est très influencée par ces grands termes et l'image restaurée risque de ne pas ressembler beaucoup à $f(x,y)$.

Dès lors, il est nécessaire de faire un compromis, et nous cherchons une fonction $M(u,v)$ qui, en l'absence de bruit et aux endroits du plan uv où $H(u,v)$ n'a pas de zéros, est telle que:

$$M(u,v) = 1 \;/\; H(u,v) \qquad (2.8.3)$$

tandis qu'ailleurs on pourra par exemple lui donner la valeur 1.

La dégradation totale incluant le bruit et le processus de restauration peut alors être représentée comme à la figure 2.8.1.

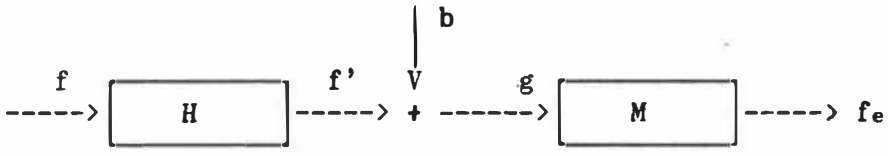


Figure 2.8.1 *Modèle de reconstruction d'images avec fonction de dégradation, bruit aléatoire additif et restauration.*

Par conséquent, la fonction de transfert globale de la dégradation et de la restauration est donnée par le produit $H(u,v)M(u,v)$. Ainsi, si $f_e(x,y)$ représente l'image restaurée et $F_e(u,v)$ sa transformée de Fourier, nous avons:

$$F_e(u,v) = (H(u,v)M(u,v))F(u,v) \qquad (2.8.4)$$

et $f_e(x,y)$, notre estimé de l'image idéale $f(x,y)$, est donné par la transformée inverse de $F_e(u,v)$.

On appelle $H(u,v)$ la "fonction de transfert d'entrée", $M(u,v)$ la "fonction de transfert de traitement" (ou "filtre de la restauration") et le produit $H(u,v)M(u,v)$ la "fonction de transfert de sortie".

Si en certaines occasions une solution adéquate est facile à obtenir et la restauration donne de bons résultats, en d'autres occasions elle peut être très pauvre. Le choix de $M(u,v)$ est généralement difficile et souvent arbitraire; il est donc préférable de chercher des méthodes plus systématiques et plus fiables.

2.9 Filtre de Wiener ou des moindres carrés

Comme dans tout problème d'estimation, pour effectuer une restauration, il est possible de choisir une image $f_e(x,y)$ qui minimise un critère d'erreur. Mais comment mesurer la différence entre l'image estimée $f_e(x,y)$ et l'image idéale $f(x,y)$? Un observateur humain peut avoir des critères subjectifs difficilement quantifiables. Sans être le critère idéal, une méthode qui possède l'avantage d'être simple

mathématiquement est celle de l'erreur quadratique moyenne. Nous décrivons brièvement ici le filtre de Wiener, l'une des techniques permettant d'obtenir la restauration qui minimise cette mesure d'erreur.

Préalablement , il est utile de rappeller brièvement la notion de *champ aléatoire* - une étude plus en détails en est donnée dans le chapitre suivant. Disons donc simplement qu'un champ aléatoire peut être considéré comme une famille de variables (ou vecteurs) aléatoires, chacune correspondant à un point du plan.

Dès lors, supposons qu'une image est la *réalisation* d'un champ aléatoire et, en particulier ici, que l'image idéale, l'image dégradée correspondante et le bruit sont respectivement des réalisations des champs aléatoires $f(x,y)$, $g(x,y)$ et $b(x,y)$. En présence de bruit, notre modèle (2.5.4) peut se réécrire par:

$$g(x,y) = \int\int h(x-x',y-y')f(x',y')dx'dy' + b(x,y) \qquad (2.9.1)$$

où $h(x,y)$ est la fonction d'étalement ponctuelle de la dégradation.

Dans l'équation (2.9.1), $b(x,y)$ n'est pas connu exactement mais on suppose que certaines propriétés statistiques à son sujet le sont: la densité spectrale d'énergie du bruit (la transformée de Fourier de sa fonction d'autocorrélation). On cherche alors l'estimé de moindres carrés $f_e(x,y)$, qui minimise:

$$e^2 = E\{[f(x,y) - f_e(x,y)]^2\} \qquad (2.9.2)$$

où $E\{.\}$ est l'espérance mathématique.

Si aucune restriction n'est imposée à la solution de ce problème, on peut montrer que l'estimé de moindres carrés est en fait l'espérance conditionnelle de f étant donné g [Papoulis:1965], donc, en général, une fonction non linéaire nécessitant pour être calculée la connaissance de la probabilité conjointe de f et g, ce qui la rend analytiquement compliquée.

En imposant comme restriction à l'estimé f_e d'être une fonction linéaire des niveaux de gris de g, nous n'obtenons plus le minimum absolu de (2.9.2) mais l'estimé <u>linéaire</u> donnant la plus petite valeur de e^2. Si l'estimé est une fonction linéaire de g, on peut l'exprimer par:

$$f_e(x,y) = \int\int m(x,x',y,y')g(x',y')dx'dy'. \qquad (2.9.3)$$

Il est possible de montrer [Rosenfeld et Kak:1982] que si le bruit n'est pas corrélé avec l'image et possède une moyenne nulle, alors le filtre de la restauration (la transformée de Fourier de $m(x,y)$) est donné par:

$$M(u,v) = \frac{1}{H(u,v)} \frac{|H(u,v)|^2}{|H(u,v)|^2+[S_{bb}(u,v)/S_{ff}(u,v)]} \qquad (2.9.4)$$

où $S_{bb}(u,v)$ est la densité spectrale du bruit (la transformée de Fourier de la fonction d'autocorrélation), et S_{ff} la densité spectrale de $f(x,y)$. En l'absence de bruit, $S_{bb} = 0$ et (2.9.4) n'est plus que le filtre inverse $1/H(u,v)$.

Notons qu'en l'absence de connaissances *a priori* sur les propriétés statistiques des champs aléatoires $b(x,y)$ et $f(x,y)$, on prend souvent comme valeur approximative de l'équation (2.9.4) le filtre:

$$M(u,v) = \frac{1}{H(u,v)} \frac{|H(u,v)|^2}{|H(u,v)|^2+\Gamma_p} \qquad (2.9.5)$$

où Γ_p, représentant le rapport de la densité de puissance du bruit sur celle du signal, est remplacé par une constante appropriée.

Si d'autre part le bruit est multiplicatif, il est possible de construire des filtres qui en tiennent compte [Franks:1969].

2.10 Formulation dans le cas discret

Le problème de la restauration d'images doit souvent être transposé dans le cas discret. En effet, la plupart des systèmes d'acquisition d'images d'application courante sont discrets, échantillonant une image continue. Dans ce cas, le modèle général peut être réécrit:

$$g = \Phi\{[H]f\} + b \qquad (2.10.1)$$

où *[H]* représente la matrice "brouillante" résultant de l'échantillonage de la fonction d'étalement ponctuelle, et où Φ est la fonction non-linéaire correspondant à la réponse du détecteur [Andrews et Hunt:1977].

La notation $\Phi[W]$ signifie que chaque élément d'un vecteur W est transformé par la même fonction:

$$\Phi[W] = \Phi\begin{bmatrix} W_1 \\ W_2 \\ . \\ . \\ W_n \end{bmatrix} = \begin{bmatrix} \Phi(W_1) \\ \Phi(W_2) \\ . \\ . \\ \Phi(W_n) \end{bmatrix} \qquad (2.10.2)$$

Il est intéressant de noter que dans le cas discret, des développements analogues à ceux présentés dans les deux sections précédentes sont sensiblement les mêmes et donnent des résultats équivalents. La transformée de Fourier, en particulier, possède un équivalent dans le cas discret. Mieux, ses propriétés utiles (orthogonalité, inverse) sont présentes dans des transformées autres que la transformée de Fourier, souvent plus rapides à obtenir, et pouvant être utilisées dans

l'implantation de filtres optimaux. Mentionnons par exemple les transformées de Hadamard et de Karhunen-Loève [Rosenfeld et Kak:1982].

2.11 Maximum *a posteriori (MAP)*

Après avoir expliqué les méthodes "déterministes", nous présentons maintenant une des méthodes "statistiques" les plus connues bien qu'elle demeure probablement l'une des moins utilisées! Elle s'avère en effet pratiquement impossible à réaliser, les temps de calculs étant énormes. En suivant Rosenfeld et Kak [1982] et Andrews et Hunt [1977] qui se réfèrent tous à Hunt [1975], nous en traçons toutefois les grandes lignes parce qu'elle est à la base des deux méthodes que nous avons implantées et testées, et qui sont expliquées aux chapitres 4 et 5.

L'approche envisagée est celle de l'estimation bayesienne. Pour estimer une quantité, on utilise la valeur maximisant la loi de probabilité *a posteriori* de cette quantité. Soit $\boldsymbol{g(x,y)}$ le champ aléatoire représentant l'image observée $g(x,y)$, et soit $\boldsymbol{f(x,y)}$ le champ aléatoire représentant l'image originale (idéale) $f(x,y)$. Le problème de l'estimation bayesienne, associé à l'équation (2.10.1), consiste à trouver $f_e(x,y)$ qui maximise $P[\boldsymbol{f}=f|\boldsymbol{g}=g]$.

Par la formule de Bayes, on obtient dans le cas discret:

$$p(f|g) = \frac{p(g|f)p(f)}{p(g)}. \qquad (2.11.1)$$

En d'autres mots, f_e représente l'image originale la plus probable une fois connue l'image dégradée g (la réalisation la plus probable du champ aléatoire $\boldsymbol{f}$ étant donné que la réalisation du champ aléatoire $\boldsymbol{g}$ est g). L'estimation f_e est appelée l'estimé du **maximum** ***a posteriori***, le *MAP*.

Puisqu'il faut maximiser (2.11.1) par rapport à f, le terme $p(g)$ peut être considéré comme une constante et il n'est pas nécessaire de donner sa forme analytique. Par contre, il est nécessaire de spécifier $p(f)$ et $p(g|f)$.

Il est assez courant en estimation bayesienne de choisir un processus gaussien multivarié pour $P[\boldsymbol{f}=f]$ tel que:

$$p(f) \propto \exp\{-\tfrac{1}{2}(f-f_m)^t[\theta_f]^{-1}(f-f_m)\} \qquad (2.11.2)$$

où $[\theta_f]$ est la matrice de covariance et $f_m = E\{\boldsymbol{f}\}$ le vecteur de moyennes. Nous verrons cependant qu'il peut être plus judicieux de choisir d'autres sortes de processus stochastiques, notamment les champs aléatoires de Markov introduits au chapitre suivant.

En ce qui concerne la probabilité conditionnelle $P[\boldsymbol{g}=g|\boldsymbol{f}=f]$, on voit, d'après le modèle de formation d'image 2.10.1, qu'une fois la valeur de f donnée, la variation de g n'est due qu'au seul bruit b. En supposant que le bruit est additif gaussien, on obtient:

$$p(g|f) \propto \exp\{-\tfrac{1}{2}(g - \Phi\{[H]f\})^t[\theta_b]^{-1}(g - \Phi\{[H]f\})\} \qquad (2.11.3)$$

où $[\theta_b]$ est la matrice de covariance du bruit $\boldsymbol{b}$.

En combinant les modèles gaussiens (2.11.2) et (2.11.3), nous cherchons la solution annulant le gradient. Nous obtenons, après manipulations, que l'estimé du maximum *a posteriori* f_e est le f qui satisfait à l'équation:

$$[\theta_f]^{-1}(f-f_m) - [H]^t[\Phi_b][\theta_b]^{-1}(g-\Phi\{[H]f\}) = 0 \qquad (2.11.4)$$

où $[\Phi_b]$ est la matrice diagonale des dérivées partielles de Φ.

Cette formule matricielle donne pour une image standard un système de 512x512 = 262 144 équations non-linéaires à résoudre, sans compter qu'il faut d'abord accomplir la tâche fastidieuse consistant à obtenir la matrice $[\Phi_b]$. Il devient nécessaire de passer par une méthode itérative, mais là encore le temps requis pour obtenir le MAP exact est excessivement long.

2.12 Autres méthodes

Plusieurs autres méthodes sont mentionnées dans la littérature sur le traitement d'images. Nous en avons retenu les principales afin d'illustrer les différentes avenues existantes et la difficulté de la restauration d'images par ordinateur: déconvolution avec contraintes, filtre de Kalman, entropie maximale, filtrage récursif, sont autant de techniques possibles ayant chacune leurs avantages.

Cependant plusieurs recherches parmi les plus récentes nous mènent à l'estimé du maximum *a posteriori*, du moins à son approximation par

des algorithmes plus rapides, notamment ceux proposés par Besag (1986) et Derin et Elliott (1987).

Nous verrons qu'un autre modèle que (2.11.2), donnant la fonction de masse (ou fonction de densité dans le cas continu) de l'image non-dégradée, permet notamment une formulation simplifiée de l'évaluation du MAP. Ce modèle est celui des champs markoviens.

CHAPITRE 3: LES CHAMPS MARKOVIENS ET LA DISTRIBUTION DE GIBBS

Ce chapitre porte sur la théorie des champs markoviens et de la distribution de Gibbs, deux processus stochastiques très liés comme nous le montrerons.

Nous avons rencontré dans le chapitre précédent le problème suivant: nous cherchons à retrouver l'image originale (l'image idéale correspondant à une scène) à partir d'une image que nous savons être son observation.

Nous avons vu que la formule de Bayes fournit la loi *a posteriori* qui combine les observations (l'image dégradée) avec certaines connaissances sur le type de scènes originales (l'information *a priori*), cela par l'entremise d'un modèle représentant le type de dégradation qu'a subi l'image. Une possibilité consiste à prendre comme estimation de l'image idéale, l'image qui maximise la loi *a posteriori*, c'est-à-dire l'image ayant la plus grande probabilité étant donné les observations. C'est ce qu'on appelle le MAP, le maximum *a posteriori*.

On suppose souvent que la dégradation résulte d'un bruit additif gaussien, mais afin de trouver la loi *a posteriori*, il nous manque

encore un modèle pour les images, c'est-à-dire une façon de quantifier l'information *a priori*. Si nous avons vu que la loi gaussienne multivariée fournit un tel modèle, les champs markoviens en fournissent un autre relativement simple qui possède des propriétés extrêmement utiles. De plus, formulé sous la forme d'une distribution de Gibbs, un champ aléatoire de Markov permet de représenter assez fidèlement un nombre beaucoup plus grand d'images. Nous commençons par l'étude générale des processus markoviens.

3.1 Processus stochastique markovien

Un processus stochastique $X = \{X_k, \ k \in T\}$ est une famille de variables aléatoires [Ross:1985]. C'est-à-dire que pour chaque k, X_k est une variable aléatoire. Si T est un sous-ensemble des réels, on dit que le processus stochastique est un processus à temps continu. De la même façon, si T est ensemble dénombrable, c'est un processus à temps discret (dans ce qui suit tous les processus stochastiques seront considérés comme étant des processus à temps discret, sauf lorsqu'explicitement mentionné).

On dit que X est un processus markovien unilatéral (causal) d'ordre r si:

$$P\,[X_k=x_k \,|\, X_{k-1}=x_{k-1}, \ldots, X_{k-n}=x_{k-n}] = P\,[X_k=x_k \,|\, X_{k-1}=x_{k-1}, \ldots, X_{k-r}=x_{k-r}], \qquad (3.1.1)$$

pour chaque k et chaque $n \geq r$. Dès lors, pour un processus stochastique markovien la loi conditionnelle de X_k, étant donné les n états précédents, ne dépend que des r états les plus proches et est indépendante des autres. On appelle ceci l'influence locale (voir figure 3.1.1).

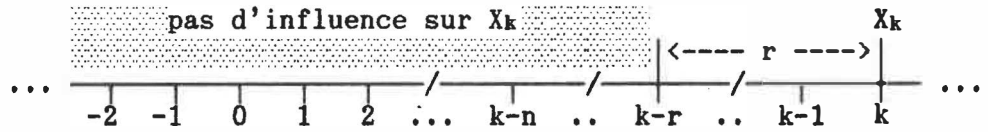


Figure 3.1.1 *Influence locale dans un processus de Markov d'ordre r.*

Considérons un exemple: soit $X = \{X_k,\ k \in \mathbb{Z}\}$ un processus markovien à temps discret du 1er ordre ($r = 1$), une chaîne de Markov, tel que X_k puisse prendre deux valeurs, 0 ou 1, avec $P[X_k=1] = p = 1-P[X_k=0]$, $0 \leq p \leq 1$, et pour lequel:

$$
\begin{aligned}
P\,[X_k=1 \mid X_{k-1}=1, X_{k-2}=i_2, \ldots, X_{k-n}=i_n] &= P\,[X_k=1 \mid X_{k-1}=1] \\
&= p + (1-p)\pi, \\
P\,[X_k=0 \mid X_{k-1}=1, X_{k-2}=i_2, \ldots, X_{k-n}=i_n] &= P\,[X_k=0 \mid X_{k-1}=1] \\
&= (1-\pi)(1-p), \\
P\,[X_k=0 \mid X_{k-1}=0, X_{k-2}=i_2, \ldots, X_{k-n}=i_n] &= P\,[X_k=0 \mid X_{k-1}=0] \\
&= (1-p) + \pi p, \\
P\,[X_k=1 \mid X_{k-1}=0, X_{k-2}=i_2, \ldots, X_{k-n}=i_n] &= P\,[X_k=1 \mid X_{k-1}=0] \\
&= (1-\pi)p,
\end{aligned}
\qquad (3.1.2)
$$

où π, $0 \leq \pi \leq 1$, est un indice de persistance. Remarquons que si $\pi = 0$ l'état X_k est indépendant de l'état précédent X_{k-1} tandis que si $\pi = 1$ le processus ne peut plus changer d'état. Faire passer π de 0 à 1 correspond donc à passer de l'indépendance à la persistance.

Cet exemple illustre comment on peut, pour les processus markoviens, utiliser des paramètres quantifiant l'influence locale d'un état sur ses "voisins" (ou l'inverse), le paramètre étant ici l'indice de persistance π. Nous verrons bientôt qu'il est possible de faire de même pour les pixels d'une image avec les modèles des champs markoviens.

3.2 Champs aléatoires

Bien que la notion de champ aléatoire ne soit pas directement reliée aux "images", nous l'utiliserons toujours dans ce contexte et établissons donc une terminologie et une notation en conséquence.

Soit S une grille, généralement rectangulaire, formée de n sites i (les points ou les *pixels*), $i = 1, 2, \ldots, n$. Nous utiliserons aussi parfois la notation en rangées et colonnes: $S = \{(i,j), i = 1, 2, \ldots, N_1, j = 1, 2, \ldots, N_2\}$, plus naturelle pour des processus bi-dimensionnels.

Une image sur S est décrite par une valeur pour chaque site. Il est à noter qu'une image ne sera considérée que sur S. De façon abrégée une image sera dénotée par $x = (x_1, x_2, \ldots, x_n)$, où x_i est la valeur de l'image au site i. On suppose que x est la réalisation d'un vecteur aléatoire X, formé de n variables alétoires: $X = (X_1, X_2, \ldots, X_n)$.

Soit y un vecteur d'observations $y = (y_1, y_2, \ldots, y_n)$, y_i étant la mesure prise au pixel i, $i = 1,..,n$. On suppose également que y est la réalisation d'un vecteur aléatoire $Y = (Y_1, Y_2, \ldots, Y_n)$.

Soit $\Gamma = \{k_1, k_2, \ldots, k_c\}$, les c couleurs (valeurs) possibles à chaque site (pixel) d'une image.

Soit $\Omega = \{(x_1, \ldots, x_n): x_i \in \Gamma, i = 1, \ldots, n\}$, l'ensemble des réalisations (configurations, coloris, images) possibles. Remarquons que $\Omega = \Gamma^n$.

Soit $\{p(x)\}$ l'ensemble des probabilités de chaque configuration ou coloris x de S. Une telle mesure est appelée un champ aléatoire défini sur S. Par la suite, on appellera aussi X un champ aléatoire dont la loi de probabilité est décrite par $\{p(x)\}$.

Une classe spécifique de champs aléatoires nous intéresse plus particulièrement. Dénotons par x_A une portion de l'image, c'est-à-dire les valeurs prises par les pixels d'une sous-région A de S, et en particulier par $x_{S\setminus i}$ l'image entière à l'exception du pixel i; bien sûr $x_S = x$ et $x_{\{i\}} = x_i$.

Considérons maintenant $P[X_i = x_i \mid x_{S\setminus i}]$ la probabilité conditionnelle associée à la couleur x_i, au pixel i, étant donné la couleur de tous les autres pixels de l'image, $x_{S\setminus i}$. Pour les champs dont la distribution conditionnelle est localement dépendante, c'est-à-dire ne dépen-

dant que de la couleur des pixels autour du pixel i, on obtient pour tout x:

$$P[X_i=x_i \mid x_{S\setminus i}] \equiv p_i(x_i \mid x_{\delta i}), \qquad (3.2.1)$$

où p_i est spécifique au pixel i et δ_i est un sous-ensemble de S\i. Les membres de δ_i sont appelés les voisins du pixel i. Un champ aléatoire possédant une telle propriété de dépendance locale est dit markovien.

A partir des probabilités conditionnelles (3.2.1) on peut en principe, en utilisant la loi de multiplication des probabilités, retrouver $p(x)$. Toutefois, à cause du caractère spatial du problème, cette multiplication peut se faire de plusieurs façons. Bien sûr, la loi conjointe $p(x)$ est unique et indépendante de la façon de considérer les sites. Par conséquent, certaines restrictions doivent être imposées sur les probabilités en (3.2.1), parmi lesquelles se trouvent une définition stricte des "voisins" d'un pixel, ainsi que la définition de "cliques", qui seront cruciales dans la construction de champs markoviens valides.

La forme la plus générale que peuvent prendre les probabilités conditionnelles est complexe et est décrite par le théorème de Hammersley-Clifford [Besag:1974].

3.3 Notions de voisinages et de clique

On dit qu'un site $j(\neq i)$ est un voisin d'un site i si et seulement si la probabilité conditionnelle $P[X_i=x_i \mid X_1=x_1, \ldots, X_{i-1}=x_{i-1}, X_{i+1}=x_{i+1}, \ldots, X_n=x_n]$ est dépendante de la variable x_j. Comme exemple très simple, considérons que $X_1, \ldots, X_n$ est une chaîne de Markov. On montre facilement que le site intérieur i ($2 \leq i \leq n-1$) possède pour voisins les sites $i-1$ et $i+1$, tandis que les sites frontières 1 et n ont comme seul voisin 2 et $n-1$ respectivement.

Il existe plusieurs systèmes de voisinages, et bien que ce soit le cas habituellement, la définition donnée n'implique pas que les pixels du voisinage d'un site soient proches, pour ce qui à trait à la distance, du pixel central.

D'une manière générale, on dit que $\mathcal{G} = \{\mathcal{G}_s, s \in S\}$ est un système de voisinages pour les sites de S si:

$$\begin{aligned} &\text{i)} \quad \mathcal{G}_s \subseteq S, \quad \forall\, s \in S \\ &\text{ii)} \quad s \notin \mathcal{G}_s, \quad \forall\, s \in S \\ &\text{iii)} \quad s \in \mathcal{G}_r \Leftrightarrow r \in \mathcal{G}_s, \quad \forall\, r,s \in S \end{aligned} \tag{3.3.1}$$

Ce qui signifie que:

i) le voisinage d'un site est un sous-ensemble de S (i.e. pixels);
ii) un pixel n'appartient pas à son propre voisinage;
iii) un système de voisinages possède une certaine symétrie, c'est-à-dire que si s est un voisin de r alors réciproquement r est un voisin de s.

On notera $x_{\delta s}$ l'ensemble des valeurs aux sites appartenant au voisinage $\mathcal{G}_s$ de s.

Le système de voisinages le plus courant est le système $\mathcal{G}^m$, appelé système du $m^{ième}$ ordre, qui prend une des formes de la figures 3.3.1 selon la valeur de m. Nous utilisons ce système de voisinages pour tous les modèles de champs markoviens dans ce mémoire.

On dénote par $\mathcal{G}^1$ le système de voisinages du 1^{er} ordre, aussi connu sous le nom de modèle du "voisin-le-plus-proche", formé des quatre pixels les plus proches du pixel central à la verticale et à l'horizontale (ceux marqués d'une étoile * à la figure 3.3.1(a)).

La figure 3.3.1 (b) montre comment le système de voisinages $\mathcal{G}^2$ du $2^{ième}$ ordre est formé. On ajoute au système du 1^{er} ordre les quatre pixels immédiatement à la diagonale du site central.

La figure 3.3.1 (c) nous donne les pixels des voisinages jusqu'à l'ordre 6, en continuant comme précédemment et en conservant toujours les pixels des voisinages des ordres inférieurs.

En pratique, nous utiliserons le modèle du $2^{ième}$ ordre. On rencontre parfois des systèmes qui vont jusqu'au $4^{ième}$ ordre en modélisation de textures par champs markoviens [Cross et Jain:1983]; il est rare cependant de voir des voisinages d'un ordre supérieur.

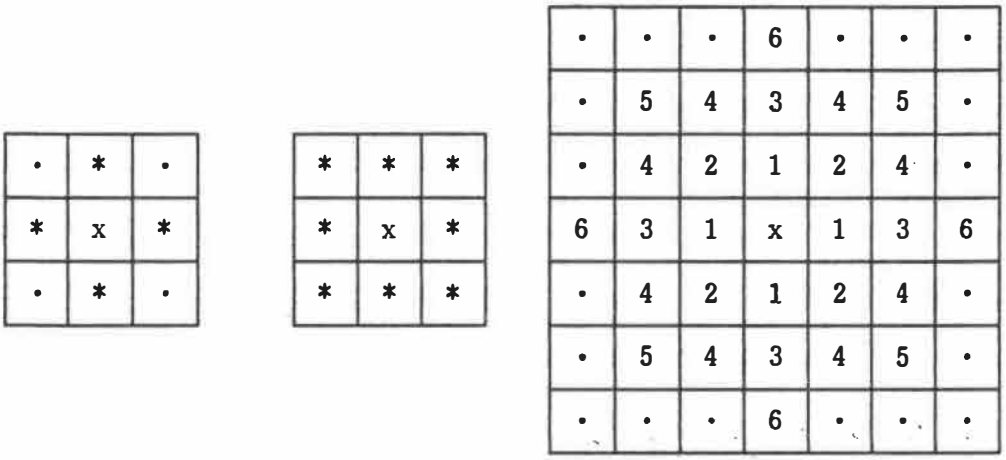


Figure 3.3.1 *Système de voisinages: (a) voisinage du 1er ordre formé des 4 pixels les plus proches (notés *) à l'horizontale et à la verticale; (b) voisinage du 2ième ordre formé des pixels du voisinage du 1er ordre plus les 4 pixels aux diagonales; (c) voisinages jusqu'à l'ordre 6, chaque ordre conservant toujours les pixels des ordres inférieurs.*

Notons que les sites situés à proximité des frontières ont moins de voisins que ceux de l'intérieur: dans un système du second ordre, un pixel sur la frontière, sans être sur un coin, n'a que 5 voisins et ceux sur les coins ont seulement 3 voisins. Bien que ceci puisse être évité en supposant que la grille soit périodique, nous conserverons ce type de frontière "libre", plus naturelle en traitement d'image.

La seconde définition concerne la notion de clique, cruciale pour une construction valide de champs aléatoires markoviens. Etant donné un sytème de voisinages $\mathcal{G}$ sur S, on dit qu'un sous-ensemble $\mathcal{C}$ de S forme une clique si:

i) $\mathcal{C}$ ne contient qu'un élément,

ou (3.3.2)

ii) toutes les paires de sites dans $\mathcal{C}$ sont voisins:
$\forall\ s,r \in S,\ r \neq s,\ r \in \mathcal{C},\ s \in \mathcal{C} \Rightarrow s \in \mathcal{G}_r$.

Pour le sytème de voisinages du 1er ordre, les cliques sont de un pixel, ou de deux pixels adjacents à l'horizontale ou à la verticale (voir figure 3.3.2 (a)). Pour celui du 2ième ordre, les cliques sont formées de 1, 2, 3 ou 4 pixels (fig. 3.3.2 (b)), avec éventuellement des ajustements possibles aux frontières.

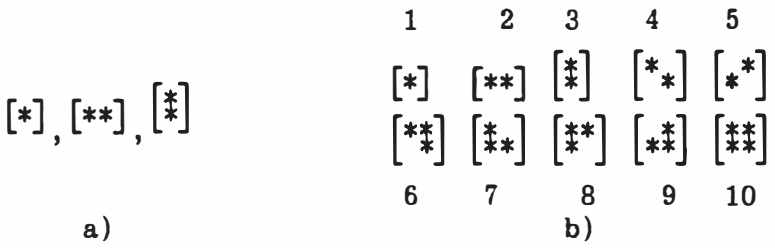


Figure 3.3.2 *Famille de cliques (a) du système du 1er ordre possèdant 1 pixel ou 2 pixels adjacents à l'horizontale ou à la verticale; (b) du système du 2ième ordre possédant de 1 à 4 pixels, notées* $\mathcal{C}_1,\ldots,\mathcal{C}_{10}$.

3.4 Champ de Markov et distribution de Gibbs

Il est maintenant possible de donner une définition plus formelle d'un champ markovien. Un champ aléatoire X est markovien par rapport à un système de voisinages $\mathcal{G}$ sur S si les deux conditions suivantes sont satisfaites:

i) $p(x) > 0,\ \forall\ x=(x_1, \ldots, x_n) \in \Omega$,

(3.4.1)

ii) $p(x_i \mid x_A) = p(x_i \mid x_{\delta i}),\ \forall\ x \in \Omega,\ \forall\ i \in S,\ \forall A$ tel que $A \subseteq S \backslash i,\ \delta_i \subseteq A$

La condition *i)* correspond à une condition de positivité (rien n'est exclu) et *ii)* correspond aux caractéristiques locales.

On ajoute habituellement la condition d'homogénéité (ou de stationnarité):

iii) $p(x_i \mid x_{\delta i})$ dépend seulement de la configuration du voisinage et est invariante par rapport à la translation (c'est-à-dire est la même à chaque site).

Ainsi une mesure ayant les propriétés (3.4.1) peut être considérée comme une généralisation à l'espace d'un processus de Markov [Kindermann et Snell:1980].

Notons que $S \backslash i$ est le seul ensemble conditionnel naturel pour des distributions spatiales, par opposition aux chaînes de Markov où le temps fournit un ordre évident [Besag:1986].

Une solution de remplacement à cet ensemble conditionnel, aujourd'hui peu rencontrée, consiste à ordonner les sites suivant un parcours de la grille de gauche à droite et de haut en bas, ce qui procure une analogie directe avec les chaînes de Markov [Abend *et al*:1965].

Comme nous l'avons mentionné plus haut, le problème avec la définition des champs markoviens est que la mesure de probabilité conjointe des X_S n'est pas facile à trouver à partir des caractéristiques locales.

Il existe heureusement un théorème montrant l'équivalence entre les champs aléatoires markoviens et une certaine distribution de probabilité empruntée à la physique statistique. Cette dernière décrit justement le comportement conjoint des variables aléatoires $X_1,\ldots,X_n$; il s'agit de la distribution de Gibbs.

On dit que $\{p(x)\}$ est un champ aléatoire de Gibbs (ou qu'il possède une distribution de Gibbs) par rapport à un système de voisinages $\mathcal{G}$ sur S si pour chaque $x\in\Omega$:

$$p(x) = \frac{1}{Z}\, e^{-U(x)/T} \qquad (3.4.2)$$

où:

T est une constante (la température en mécanique),

$U(x) = \sum_{C\in\mathcal{C}} V_C(x)$ est une fonction d'énergie, avec

$V_C(x)$ = le *potentiel* associé à la clique C,

$Z = \sum_{x\in\Omega} e^{-U(x)}$ est une constante de normalisation.

Rappelons que $\mathcal{C}$ est la classe des cliques du système de voisinages $\mathcal{G}$. Chaque $V_C(x)$ est une fonction définie sur Ω dépendant uniquement des valeurs x_i tel que $i\in C$. Dans les modèles que nous utilisons, T est considérée comme étant égale à 1.

Non seulement une mesure de Gibbs possède les propriétés (3.4.1), mais réciproquement, toute mesure ayant les propriétés (3.4.1) peut se représenter comme une mesure de Gibbs en choisissant une fonction

d'énergie appropriée. C'est ce que nous énonçons dans le théorème suivant.

Théorème 3.4.1:

> Soit $\mathcal{G}$ un système de voisinages. Alors X est un champ aléatoire markovien par rapport à $\mathcal{G}$ si et seulement si $P[X=x]$ est une distribution de Gibbs par rapport à $\mathcal{G}$.

La démonstration de ce théorème, qui peut être considéré comme une forme du théorème de Hammersley-Clifford, est donnée par Besag [1974] et Kindermann et Snell [1980].

Cette équivalence nous fournit une façon simple et pratique de définir un champ aléatoire de Markov: il suffit de spécifier les potentiels. A partir de la loi conjointe on peut facilement obtenir les caractéristiques locales (c'est-à-dire les probabilités conditionnelles). En effet:

$$p(x_i \mid x_{s\setminus i}) = \frac{p(x_i, x_{s\setminus i})}{p(x_{s\setminus i})}$$

$$= \frac{p(x_1, .., x_{i-1}, x_i, x_{i+1}, .., x_n)}{\sum_{k\in\Gamma} p(x_1, .., x_{i-1}, k, x_{i+1}, .., x_n)}$$

$$= \frac{e^{-\sum_{c\in\mathcal{C}} V_c(x_1, .., x_{i-1}, x_i, x_{i+1}, .., x_n)}}{\sum_{k\in\Gamma} e^{-\sum_{c\in\mathcal{C}} V_c(x_1, .., x_{i-1}, k, x_{i+1}, .., x_n)}}.$$

Divisons les cliques en deux classes, soit $\mathcal{C} = \mathcal{C}_1 \cup \mathcal{C}_2$ avec $\mathcal{C}_1$ la classe des cliques contenant le site i et $\mathcal{C}_2$ la classe des cliques ne contenant pas le site i, avec $\mathcal{C}_1 \cap \mathcal{C}_2 = \emptyset$. Dès lors la probabilité conditionnelle devient:

$$\frac{e^{-\sum_{c\in\mathcal{C}_1} Vc(x_1,..,x_{i-1},x_i,x_{i+1},..,x_n)} \cdot e^{-\sum_{c\in\mathcal{C}_2} Vc(x_1,..,x_{i-1},x_{i+1},..,x_n)}}{\sum_{k\in\Gamma}\left[e^{-\sum_{c\in\mathcal{C}_1} Vc(x_1,..,x_{i-1},k,x_{i+1},..,x_n)} \cdot e^{-\sum_{c\in\mathcal{C}_2} Vc(x_1,..,x_{i-1},x_{i+1},..,x_n)}\right]}$$

car si $C\in\mathcal{C}_2$, la fonction $Vc(.)$ est indépendante de la $i^{ème}$ variable. Nous pouvons donc simplifier, ce qui nous donne comme expression pour la probabilité conditionnelle:

$$p(x_i \mid x_{s\setminus i}) = \frac{e^{-\sum_{c\in\mathcal{C}_1} Vc(x_1,..,x_{i-1},x_i,x_{i+1},..,x_n)}}{\sum_{k\in\Gamma} e^{-\sum_{c\in\mathcal{C}_1} Vc(x_1,..,x_{i-1},k,x_{i+1},..,x_n)}}. \qquad (3.4.3)$$

Remarquons que l'on prouve facilement avec ce qui précède qu'une distribution de Gibbs vérifie (3.4.1):

- d'une part l'équation (3.4.3) vérifie la condition (3.4.1. *ii)*) puisque seuls les pixels membres du voisinage du pixel i feront partie des cliques de la classe $\mathcal{C}_1$; en effet, ces cliques contiennent le site i et par la définition (3.3.2. *ii)*) toutes les paires de sites dans une clique sont voisins; ainsi la couleur des autres pixels n'a aucune incidence et:

$$p(x_i \mid x_{s\setminus i}) = p(x_i \mid x_{\delta i}).$$

- d'autre part la fonction exponentielle étant toujours supérieure à zéro, la condition *i)* est remplie.

Ce qui démontre la première partie du théorème 3.4.1.

3.5 Modèle de Ising et autres modèles

Le concept des champs markoviens provient des essais de généralisation du modèle très spécifique du physicien allemand Ernst Ising. Dans sa thèse de doctorat, dont un sommaire fut publié en 1925, Ising essayait d'expliquer à l'aide de ce modèle certains résultats empiriques observés sur des matériaux ferro-magnétiques.

Ce modèle de base est formé à partir de ce que l'on appelle maintenant un voisinage du 1er ordre pour deux "couleurs"[1], par exemple 0 et 1, sur une grille carrée $S = \{(i,j): 1 \leq i \leq m, \ 1 \leq j \leq m\}$ (voir figure 3.5.1).

	i,j-1	
i-1,j	i,j	i+1,j
	i,j+1	

Figure 3.5.1 *Notation des pixels du voisinage du 1er ordre autour du site* (i,j) *pour le modèle de Ising.*

A chaque configuration x, une énergie $U(x)$ est assignée par:

$$U(x) = \alpha \sum_{(i,j)} x_{i,j} + \beta \sum_{(i,j)} x_{i,j} x_{i+1,j} + \beta \sum_{(i,j)} x_{i,j} x_{i,j+1}. \qquad (3.5.1)$$

---

[1] Il s'agissait en fait des deux états des dipôles (ou "spin") du matériau, orientés soit dans le même sens, soit dans le sens contraire d'un champ magnétique extérieur.

En établissant la correspondance avec le modèle général de la distribution de Gibbs on obtient:

$$U(x) = \sum_{(i,j)} V_1(x_{i,j}) + \sum_{(i,j)} V_2(x_{i,j}, x_{i+1,j}) + \sum_{(i,j)} V_3(x_{i,j}, x_{i,j+1}) \qquad (3.5.2)$$

où:

$$V_1(x_{i,j}) = \begin{cases} \alpha & \text{si } x_{i,j} = 1 \\ 0 & \text{si } x_{i,j} = 0 \end{cases}$$

$$V_2(x_{i,j}, x_{i+1,j}) = \begin{cases} \beta & \text{si } x_{i,j} = 1 \text{ et } x_{i+1,j} = 1 \\ 0 & \text{autrement} \end{cases}$$

$$V_3(x_{i,j}, x_{i,j+1}) = \begin{cases} \beta & \text{si } x_{i,j} = 1 \text{ et } x_{i,j+1} = 1 \\ 0 & \text{autrement} \end{cases}$$

et bien sûr (3.5.1) et (3.5.2) sont équivalents.

Remarquons que la somme sur la fonction V_1 équivaut à compter le nombre de pixels égaux à 1 en accordant un poids de α à chacun. La somme sur la fonction V_2 pour sa part, équivaut à affecter un poids β au nombre de paires de pixels succesifs à l'horizontale valant 1, de même pour la somme sur V_3 à la verticale.

Par exemple, pour le modèle étudié ci-dessus, la configuration x_o montrée à la figure 3.5.2 possède une probabilité

$$P[X=x_o] = Z^{-1} \exp(-(6\alpha+5\beta)/T).$$

1	0	0
0	1	1
1	1	1

Figure 3.5.2 *Exemple de configuration* $x_o \in \Omega$ *pour une grille* S *de 9 pixels avec c=2 couleurs,* $\Gamma=\{0,1\}$. *On compte 6 points égaux à 1, 3 paires de pixels successivement égaux à 1 horizontalement et 2 paires verticales de pixels de valeur 1. On trouve alors que la fonction d'énergie du modèle de Ising est* $U(x_o) = 6\alpha+(3+2)\beta$.

En développant l'équation générale (3.4.3) pour ce modèle, les caractéristiques locales sont données, au numérateur, par:

$$e^{-[V_1(x_{s,t}) + V_2(x_{s,t},x_{s,t+1}) + V_2(x_{s,t},x_{s,t-1}) + V_2(x_{s,t},x_{s+1,t}) + V_2(x_{s,t},x_{s-1,t})]}$$

soit:

$$e^{-[\alpha\, x_{s,t} + \beta x_{s,t}\{x_{s,t+1} + x_{s,t-1} + x_{s+1,t} + x_{s-1,t}\}]};$$

appelons $v_{s,t} = x_{s,t+1} + x_{s,t-1} + x_{s+1,t} + x_{s-1,t}$ le nombre de voisins du pixel (s,t) de couleur 1 et nous obtenons pour le numérateur:

$$e^{-x_{s,t}(\alpha + \beta v_{s,t})};$$

pour le dénominateur, il faut faire la somme pour $x_{s,t}$ valant 0 et 1, ce qui s'écrit:

$$e^{-\,0\,(\alpha + \beta v_{s,t})} + e^{-\,1\,(\alpha + \beta v_{s,t})} = 1 + e^{-(\alpha + \beta v_{s,t})}$$

Nous obtenons donc comme caractéristiques locales pour le modèle de Ising:

$$p(x_{s,t}|x_{S\setminus(s,t)}) = p(x_{s,t}|x_{\delta(s,t)}) = \frac{e^{-x_{s,t}(\alpha + \beta v_{s,t})}}{1 + e^{-(\alpha + \beta v_{s,t})}}. \quad (3.5.3)$$

Le second modèle que nous étudions, plus général, est basé sur un voisinage $\mathcal{G}^2$ du second ordre et c couleurs: $\Gamma=\{k_1,\ldots,k_c\}$. Le modèle, défini (en revenant à la notation linéaire) sur une grille $S = \{(i): 1\leq i\leq n\}$, est donné par:

$$p(x) = \frac{1}{Z} e^{\sum_{i=1}^{n} G_i(x_i) + \sum_{1\leq i<j\leq n} G_{i,j}(x_i,x_j)}. \quad (3.5.4)$$

Remarquons que le modèle n'utilise que les cliques de un ou de deux pixels. Pour les cliques de 1 pixel, on a $V_C(x) = G_i(x_i)$, pour 2 pixels $V_C(x) = G_{i,j}(x_i,x_j)$, et pour les cliques de 3 ou de 4 pixels, $V_C(.)$ est toujours égale à 0.

Comme cas particulier de ce modèle, considérons le cas où:

$$G_i(x_i) = \begin{cases} \alpha_q & \text{si } x_i = k_q \\ 0 & \text{autrement} \end{cases}$$

$$G_{i,j}(x_i,x_j) = \begin{cases} -\beta_{q,r} & \text{si } x_i=k_q,\ x_j=k_r,\ k_q\neq k_r,\ i\in\mathcal{G}_j \\ 0 & \text{autrement} \end{cases}$$

où $\beta_{q,r} > 0$ et $\beta_{q,r} = \beta_{r,q}$.

Remarquons que dans ce cas nous pouvons réécrire la probabilité conjointe par:

$$p(x) \propto e^{\sum_{q=1}^{c} \alpha_q n^q - \sum_{1 \le q < r \le c} \beta_{q,r} \pi^{q,r}}$$

où n^q est le nombre de pixels de couleur k_q et $\pi^{q,r}$ le nombre de paires distinctes de pixels voisins entre eux de couleurs k_q et k_r, $k_q \neq k_r$.

En développant comme précédemment l'équation générale (3.4.3) pour ce cas particulier de notre second modèle, on obtient que la probabilité conditionnelle que le pixel i prenne une valeur k_o, étant donné les couleurs des autres pixels, est:

$$P[X_i = k_o \mid x_{S \setminus i}) = p(x_i \mid x_{S \setminus i}) \propto e^{\alpha_o - \sum_{\substack{q=1 \\ q \neq o}}^{c} \beta_{o,q} u_i(k_q)} \qquad (3.5.5)$$

puisqu'au site i, la couleur étant k_o, il ne reste que α_o multiplié par 1 pour les cliques de 1 pixel contenant le pixel i, et où $u_i(k_q)$ est une fonction donnant le nombre de voisins du pixel i de couleur k_q, et donc la somme couvre bien toutes les cliques de 2 pixels contenant le pixel i.

En supposant que $\beta_{o,q} = \beta$, c'est-à-dire que le paramètre β soit le même pour toutes les paires de couleurs, on peut le sortir de la somme. Dans ce cas cette somme revient à compter le nombre de voisins du site i qui ne sont pas de la couleur k_o. Ce qui s'écrit:

$$\sum_{\substack{q=1 \\ q \neq o}}^{c} u_i(k_q) = n_i - u_i(k_o)$$

où n_i donne le nombre de voisins du site i (8 en général sauf aux frontières).

On obtient dans ce cas comme probabilité conditionnelle:

$$P[X_i=k_o | x_{S\setminus i}] \propto e^{\alpha_o - \beta n_i + \beta u_i(k_o)}.$$

En sortant le terme βn_i qui ne dépend pas de la couleur et peut donc rentrer dans la constante de proportionnalité, on obtient:

$$P[X_i=k_o | x_{S\setminus i}] \propto e^{\alpha_o + \beta u_i(k_o)}. \qquad (3.5.6)$$

Si nous simplifions davantage, en supposant que le paramètre α_o ne dépend pas de la couleur, soit $\alpha_o = \alpha$, nous obtenons finalement comme caractéristiques locales:

$$P[X_i=k_o | x_{S\setminus i}] \propto e^{\beta u_i(k_o)}. \qquad (3.5.7)$$

Par sa grande simplicité et sa facilité d'interprétation, le modèle (3.5.7) est des plus intéressants. En effet, on comprend que le modèle favorise la couleur assignée au plus grand nombre de voisins. Il s'agit là d'une hypothèse couramment rencontrée en traitement d'images et tout à fait naturelle pour un grand nombre d'images de tous les jours.

C'est ce modèle de champ markovien, donné à l'équation (3.5.7) par ses caractéristiques locales, que nous utilisons, en suivant Besag [1986], pour illustrer l'algorithme de restauration d'images (l'*ICM*) de Besag.

En plus du modèle ci-dessus et des différentes variantes que l'on peut trouver en remontant la simplification jusqu'à l'équation (3.5.5) en passant par (3.5.6), il existe plusieurs autres modèles. Nous en considérons deux autres.

Tout d'abord le modèle général (3.5.4) peut être développé dans le cas particulier qui suit:

$$G_i(x_i) = \begin{cases} \alpha_q & \text{si } x_i = k_q \\ 0 & \text{autrement} \end{cases}$$

$$G_{i,j}(x_i, x_j) = \begin{cases} -\beta_t & \text{si } i \text{ et } j \text{ appartiennent à } \mathcal{C}_t \\ 0 & \text{autrement} \end{cases} \qquad (3.5.8)$$

où $\mathcal{C}_t$ est l'ensemble des cliques t, t=2,...10 (voir figure 3.3.2 (b)).

Ce modèle, que nous utiliserons au chapitre 5, donne des poids différents selon l'orientation des cliques plutôt que selon la couleur des pixels.

Finalement, un voisinage du second ordre, tel que nous l'avons illustré pour une grille rectangulaire, contient des pixels qui ne sont pas vraiment à égale distance du pixel central. Il est parfois

nécessaire de changer les poids des diagonales pour en tenir compte. Ce qui nous donne comme dernier modèle:

$$p_i(k|\cdot) = \frac{e^{\alpha_k - \sum_{\substack{q=1 \\ q\neq k}}^{c} \{\beta'_{kq} u'_i(q) + \beta''_{kq} u''_i(q)\}}}{\sum_{o=1}^{c} e^{\alpha_o - \sum_{\substack{q=1 \\ q\neq o}}^{c} \{\beta'_{oq} u'_i(q) + \beta''_{oq} u''_i(q)\}}} \qquad (3.5.9)$$

où $u'_i(q)$ donne le nombre de voisins du pixel i de la première catégorie (ceux marqués d'un + à la figure 3.5.3) et de couleur q; et où $u''_i(q)$ donne le nombre de voisins du pixel i de la seconde catégorie (ceux marqués d'une étoile * à la figure 3.5.3) et de couleur q. Les β' et β'' sont les paramètres correspondants pour chaque paire de couleurs.

*	+	*
+	i	+
*	+	*

Figure 3.5.3 *Notation des pixels d'un voisinage autour du pixel* i *pour le modèle de champ markovien (3.5.9) donnant des poids différents aux pixels des diagonales.*

CHAPITRE 4: L'*ICM* de Besag

Nous étudions dans ce chapitre la première des deux méthodes de restauration d'images abordées dans ce mémoire. Appelée *ICM* pour «Iterated Conditional Modes» par son auteur, J. Besag [1986], il s'agit d'une méthode itérative fournissant une approximation à l'estimé du maximum *a posteriori* (MAP).

4.1 Hypothèses de base et maximum *a posteriori*

Soit Y et X deux champs aléatoires, et y et x leurs réalisations respectives. On considère que x est une image, et que y représente les observations prises sur une image. Soit x^* l'image "vraie", l'image idéale de la scène d'origine, "observée" après une dégradation quelconque pour donner y.

Soulignons que pour Besag, y n'est pas véritablement une image mais est plutôt formée des observations de chaque pixel. Chaque y_i possède une valeur réelle, étant ici la réalisation d'une variable alétoire Y_i continue. Cependant, et bien que nous ayons conservé l'hypothèse de Besag dans notre implantation numérique, le cas discret où chaque y_i est elle même une couleur est entièrement analogue.

Mentionnons également que dans ce qui suit les variables X_i sont discrètes et que les y_i peuvent contenir plusieurs éléments s'il existe plusieurs observations du même pixel i, au quel cas le développement de l'ICM est tout à fait semblable.

Dénotons par x_i la valeur au point (pixel) i de la grille (la région bi-dimensionnelle) S, $i=1,2,..,n$, d'une image x, et notons $x_{\delta i}$ les couleurs des pixels voisins du point i, selon la définition de voisinage donnée au chapitre précédent. On notera $x_{S\setminus i}$ la couleur de toute l'image à l'exception du pixel i.

Dans le but d'établir sa méthode, Besag pose deux hypothèses de base: la première concerne la relation entre les observations y et l'image originale x, et la seconde, l'information *a priori*.

Supposons premièrement que la distribution $f(y|x)$ est connue (hypothèse normale en restauration d'images) et que les variables aléatoires Y_i constituant Y sont conditionnellement indépendantes étant donné que $X=x$:

$$f_{Y|X}(y|x) = f(y_1, \ldots, y_n|x) = f_1(y_1|x) \ldots f_n(y_n|x)$$
$$= \prod_{i=1}^{n} f_i(y_i|x).$$

Supposons également que chaque observation y_i ne dépend que de la valeur x_i au point i: $f_i(y_i|x) = f_i(y_i|x_i)$. Ainsi, nous obtenons comme

première hypothèse que la distribution conditionnelle de y, étant donné x, est simplement:

$$f_{Y|X}(y|x) = \prod_{i=1}^{n} f_i(y_i|x_i). \qquad (4.1.1)$$

Notons que cette hypothèse, comme celle qui suit, impose des restrictions pas toujours naturelles pour les phénomènes réels expliqués. D'une part, cette hypothèse d'indépendance n'est pas toujours valide et d'autre part les y_i, subissant l'effet d'une fonction d'étalement ponctuelle h connue (voir chapitre 2), peuvent contenir de l'information provenant non seulement du pixel i mais également des pixels adjacents. Nous montrerons plus loin comment nous avons modifié l'ICM pour en tenir compte.

La deuxième hypothèse concerne l'information *a priori*. On suppose que la vraie image, dénotée x^*, est une réalisation d'un champ aléatoire *markovien* $\{p(x)\}$ connu. Ainsi, la couleur d'un point, étant donné la couleur aux autres pixels de l'image, ne dépend en fait que de la couleur des pixels membres du voisinage du point i:

$$P[X_i=x_i \,|\, x_{S\setminus i}] \equiv p_i(x_i \,|\, x_{\delta i}). \qquad (4.1.2)$$

Notons que l'ICM est une méthode indépendante de l'ordre du champ markovien. Toutefois, nous nous limiterons dans nos exemples de restauration à des champs définis sur un système de voisinages du second ordre. Ceux-ci semblent suffisants pour modéliser un très grand nombre d'images.

Par la formule de Bayes, il est possible de combiner la loi *a priori* donnant $P[X=x]$ et la loi conditionnelle donnant $f(y|x)$ dans une loi *a posteriori*:

$$P[X=x|y] = \frac{f(y|x)p(x)}{f_Y(y)}. \qquad (4.1.3)$$

où $f_Y(y)$ est la fonction de densité de Y.

Dans ce qui suit, nous noterons indifféremment $f(.)$ la fonction de densité de Y ou la fonction de densité conditionnelle de Y étant donné X, les arguments indiquant de laquelle il s'agit. La fonction de densité conditionnelle marginale de Y_i étant donné X sera notée $f_i(.)$.

La formule (4.1.3) nous fournit la loi de probabilité de l'image originale x (ce que nous cherchons) étant donné les observations y (ce que nous avons).

En cherchant à prendre comme estimé de l'image originale le $\hat{x}$ tel que $p(\hat{x}|y)$ soit maximale, on choisit l'image ayant la plus grande probabilité d'être l'image idéale étant donné les observations. Cette estimation est dite du maximum *a posteriori* (MAP). Puisque $f_Y(y)$ ne dépend pas de x, il suffit de maximiser:

$$f(y|x)p(x) \qquad (4.1.4)$$

par rapport à x.

La recherche de ce maximum s'avère très longue: pour des images simples de dimensions réduites de 50 x 100 comme celles que nous utiliserons, et même en restant dans le cas binaire (2 couleurs), il existe $2^{50 \times 100}$ images possibles parmi lesquelles il faut trouver celle qui maximise la loi *a posteriori*. Avec l'équipement dont nous disposons et une méthode directe, un calcul rapide nous montre qu'une telle recherche prendrait plusieurs milliards d'années! Dès lors, il semble utile de chercher une approximation plus rapide au MAP.

4.2 *Iterated Conditional Modes* (ICM)

Le principe utilisé par Besag [1986] est le suivant: plutôt que de chercher le maximum de la loi *a posteriori* $p(x|y)$ en une seule étape pour une image entière, il procède pixel par pixel.

Soit $\hat{x}$ un estimé provisoire de la vraie scène x^*. En ayant pour but de mettre à jour la couleur $\hat{x}_i$ au point i à partir de toute l'information disponible, Besag propose une méthode itérative.

A une étape donnée, il suggère de choisir la couleur $\hat{x}_i$ maximisant $P[X_i = x_i | y, \hat{x}_{s \setminus i}]$. Il s'agit de la probabilité d'avoir une couleur x_i au pixel i (celui traité à cette étape) étant donné toutes les observations y et la reconstruction réalisée jusqu'à ce moment, $\hat{x}_{s \setminus i}$.

Par la formule de Bayes, les équations (4.1.1) et (4.1.2) et en notant que $p(x) = p(x_i, x_{s\setminus i})$, nous avons:

$$
\begin{aligned}
P[X_i = x_i \mid y, \hat{x}_{s\setminus i}] &= \frac{\mathcal{L}(x_i, y, \hat{x}_{s\setminus i})}{\mathcal{L}(y, \hat{x}_{s\setminus i})} \\
&= \frac{f(y \mid x_i, \hat{x}_{s\setminus i}) p(x_i, \hat{x}_{s\setminus i})}{\mathcal{L}(y, \hat{x}_{s\setminus i})} \\
&= \frac{f(y \mid x_i, \hat{x}_{s\setminus i}) p(x_i \mid \hat{x}_{s\setminus i}) p(\hat{x}_{s\setminus i})}{\mathcal{L}(y, \hat{x}_{s\setminus i})} \\
&= \frac{f(y \mid x_i, \hat{x}_{s\setminus i}) p_i(x_i \mid \hat{x}_{\delta i}) p(\hat{x}_{s\setminus i})}{\mathcal{L}(y, \hat{x}_{s\setminus i})} \\
&= \frac{f_i(y_i \mid x_i) \prod_{\substack{j=1 \\ j \neq i}}^{n} f_j(y_j \mid \hat{x}_j) p_i(x_i \mid \hat{x}_{\delta i}) p(\hat{x}_{s\setminus i})}{\mathcal{L}(y, \hat{x}_{s\setminus i})}
\end{aligned}
$$

où $\mathcal{L}(\cdot)$ signifie loi de probabilité conjointe des arguments. En ne tenant compte que des termes contenant x_i, nous obtenons:

$$P[X_i = x_i \mid y, \hat{x}_{s\setminus i}] \propto f_i(y_i \mid x_i)\ p_i(x_i \mid \hat{x}_{\delta i}). \qquad (4.2.1)$$

Par conséquent nous cherchons comme nouvelle couleur au point i le x_i qui <u>maximise</u> le produit de $f_i(y_i \mid x_i)$: la distribution conditionnelle d'une observation étant donné la valeur vraie à un point i; et de $p_i(x_i \mid \hat{x}_{\delta i})$: la distribution conditionnelle caractérisant le champ aléatoire de Markov.

L'application de ce procédé à tous les pixels de l'image représente une itération complète de l'ICM.

Besag démontre que la méthode converge théoriquement vers ce qui doit être un maximum local de $p(x|y)$. En effet, en notant que:

$$P[X=\hat{x}|y] = p(\hat{x}_i, \hat{x}_{s\setminus i}|y) = p(\hat{x}_i|y,\hat{x}_{s\setminus i})p(\hat{x}_{s\setminus i}|y), \qquad (4.2.2)$$

il est clair, puisque 1) la fonction de masse $p(\hat{x}_{s\setminus i}|y)$ n'est pas influencée par un changement de $\hat{x}_i$, et 2) $p(\hat{x}_i|y,\hat{x}_{s\setminus i})$ est justement maximisé en (4.2.1), que la probabilité $P[X=\hat{x}|y]$ ne décroît jamais. La suite $\langle P[(X=\hat{x})_t \mid y]\rangle$, où l'indice t représente l'itération, est donc une suite non décroissante bornée - par 1, il s'agit de probabilités - et forme par conséquent une suite convergente.

4.3 Modèles

Nous venons d'établir le principe de l'ICM d'une manière très générale en termes de probabilité conditionnelle et de loi *a posteriori*. Avant de discuter de sa performance en analysant certaines images, voyons comment nous avons développé l'ICM pour certains modèles bien précis de dégradations et de lois *a priori*.

Le principal modèle de champ markovien que nous avons utilisé est celui employé par Besag [1986] dans ses exemples. Le champ markovien considéré est défini sur un système de voisinages du second ordre et sa loi conditionnelle marginale est donnée par:

$$p_i(x_i|x_{\delta i}) = \frac{e^{\beta u_i(x_i)}}{\sum_{k=1}^{c} e^{\beta u_i(k)}}. \qquad (4.3.1)$$

Il s'agit d'un des modèles de champs aléatoires de Markov les plus simples (voir chapitre précédent, équation (3.5.7)) où $u_i(k)$ est une fonction donnant le nombre de voisins du pixel i ayant la couleur k et où β est un paramètre quantifiant l'importance du voisinage dans la loi *a priori*. En pratique β sera souvent choisi égal à 1.5 dans les restaurations.

La seule dégradation considérée par Besag [1986] correspond à une détérioration causée par un bruit aléatoire additif gaussien. Nous avons vu au chapitre 2 qu'il est possible d'interpréter ce bruit en considèrant l'observation y_i comme étant générée par une distribution gaussienne de moyenne x_i^*, la couleur au point i de l'image idéale.

Avec une variance de σ^2, la loi conditionnelle marginale des observations est donnée dans ce cas par:

$$f(y_i \mid x_i) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(y_i - x_i)^2}{\sigma^2}}. \qquad (4.3.2)$$

La maximisation de l'expression de droite en (4.2.1) dans le contexte de (4.3.1) et (4.3.2) équivaut à <u>maximiser</u>, à une étape donnée de la reconstruction, le produit suivant:

$$\frac{e^{\beta \hat{u}_i(x_i)}}{\sum_{k=1}^{c} e^{\beta \hat{u}_i(k)}} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}\frac{(y_i - x_i)^2}{\sigma^2}} \qquad (4.3.3)$$

par rapport à x_i, où $\hat{u}_i(x_i)$ est le nombre de voisins de i ayant la couleur x_i à cette étape de la reconstruction.

En ne tenant compte que des termes contenant x_i, nous cherchons:

$$\underset{x_i}{Max} \left[e^{\beta \hat{u}_i(x_i)} \; e^{-\frac{1}{2}\frac{(y_i - x_i)^2}{\sigma^2}} \right]$$

puis en prenant le logarithme naturel nous obtenons:

$$\underset{x_i}{Max} \left[\beta \hat{u}_i(x_i) - \frac{1}{2\sigma^2}(y_i - x_i)^2 \right]$$

et en choisissant de prendre le négatif le problème devient:

$$\underset{x_i}{Min} \left[\frac{1}{2\sigma^2}\{y_i - x_i\}^2 - \beta \hat{u}_i(x_i) \right]. \qquad (4.3.4)$$

Les x_i représentant la couleur au point i, il est possible de réécrire (4.3.4) en remplaçant x_i par $k \in \Gamma$ comme suit:

$$\underset{k \in \Gamma}{Min} \left[\frac{1}{2\sigma^2}\{y_i - k\}^2 - \beta \hat{u}_i(k) \right].$$

Pour d'autres modèles de champs markoviens on obtient facilement les équations gouvernant la recherche du maximum *a posteriori*. Bien qu'elles ne soient pas implantées dans le programme, nous donnons ci-dessous les équations correspondant aux champs étudiés dans le chapitre précédent.

Pour le modèle suivant:

$$p_i(k|\cdot) = \frac{e^{\beta u_i(k) + \alpha_k}}{\sum_{q=1}^{c} e^{\beta u_i(q) + \alpha_q}} \tag{4.3.5}$$

il faut chercher dans le cadre de l'ICM, la couleur k qui rend:

$$\frac{1}{2\sigma^2}(y_i - k)^2 - \alpha_k - \beta u_i(k) \tag{4.3.6}$$

minimale et remplacer la couleur du $i^{ème}$ pixel par celle-ci.

Pour le champ aléatoire markovien:

$$p_i(k|\cdot) = \frac{e^{\alpha_k - \sum_{\substack{q=1 \\ q\neq k}}^{c} \beta_{kq} u_i(q)}}{\sum_{o=1}^{c} e^{\alpha_o - \sum_{\substack{q=1 \\ q\neq o}}^{c} \beta_{oq} u_i(q)}} \tag{4.3.7}$$

on prend successivement:

$$\underset{k\in\Gamma}{Min} \left[\frac{1}{2\sigma^2}(y_i - k)^2 - \alpha_k + \sum_{\substack{q=1 \\ q\neq k}}^{g} \beta_{kq} u_i(q) \right]. \tag{4.3.8}$$

Finalement, en changeant les poids des diagonales, le champ markovien:

$$p_i(k|\cdot) = \frac{e^{\alpha_k - \sum_{\substack{q=1 \\ q\neq k}}^{c} \{\beta'_{kq} u'_i(q) + \beta''_{kq} u''_i(q)\}}}{\sum_{o=1}^{c} e^{\alpha_o - \sum_{\substack{q=1 \\ q\neq o}}^{c} \{\beta'_{oq} u'_i(q) + \beta''_{oq} u''_i(q)\}}} \tag{4.3.9}$$

impose comme critère de l'ICM:

$$\underset{k\in\Gamma}{Min} \left[\frac{1}{2\sigma^2}(y_i - k)^2 - \alpha_k + \sum_{\substack{q=1 \\ q\neq k}}^{c} \left[\beta'_{kq} u'_i(q) + \beta''_{kq} u''_i(q) \right] \right]. \tag{4.3.10}$$

4.4 Détériorations complexes

Nous envisageons dans cette section les détériorations additionnelles dont nous avons parlé au chapitre 2 et indiquons comment nous avons procédé pour apporter les développements et les modifications nécessaires à l'ICM.

La première de ces détériorations dont nous avons tenu compte, brièvement mentionnée par Besag, est la superposition de pixels causée par une fonction de dégradation h connue ou estimée (la fonction d'étalement ponctuelle du système de formation d'images).

Supposons en effet que la loi conditionnelle de chaque vecteur d'observation Y_i étant donné x ne dépende pas seulement de x_i mais également d'un ensemble a_i des abords du pixel i[2].

Un tel effet se produit par exemple lorsque la caméra se déplace devant la scène à saisir, ou l'inverse, ou encore quand la lentille est hors foyer. Il en résulte une image brouillée. Il est alors souhaitable de tenir compte de ce phénomène dans la formulation de l'ICM.

---

[2]Cet ensemble a_i n'est pas obligatoirement relié au voisinage $\delta_i \in \mathcal{G}$, bien qu'en pratique dans nos exemples les deux seront considérés comme identiques. La notion de voisinage est nécessaire à la définition de champ markovien et n'a rien à voir avec la détérioration causée par une fonction d'étalement ponctuelle.

Rappelons tout d'abord le modèle de dégradation d'images établi au chapitre 2. Le système de formation d'images est divisé en trois parties: 1) le bruit aléatoire, que nous avons traité jusqu'à maintenant sous une forme additive mais que nous considérerons sous une forme multiplicative, puis deux aspects pratiquement ignorés par Besag, 2) le brouillage résultant d'une fonction d'étalement ponctuelle, et 3) une dégradation non-linéaire supplémentaire.

On obtient le modèle général suivant:

$$Y = \Phi(H(X)) \odot B \qquad (4.4.1)$$

où

- B correspond au bruit aléatoire, ici gaussien, bien que la méthode s'applique pour un processus arbitraire de bruit,
- Φ est une transformation non-linéaire,
- H est la matrice de "brouillage" correspondant à une fonction d'étalement ponctuelle invariante dans l'espace
- l'opération $\odot$ représente ici soit l'addition soit la multiplication.

Nous supposerons également que les processus stochastiques X et B sont indépendants. Cette hypothèse est aussi requise pour le modèle de Besag s'il est formulé comme ci-dessus.

Puisqu'il est maintenant nécessaire de tenir compte de $x_{\partial i}$ (la couleur des pixels aux abords du pixel i) dans la formulation de l'ICM,

nous réécrivons la densité conditionnelle marginale de Y_i étant donné x sous la forme $f_i(y_i \mid x_i, x_{a_i})$.

Ainsi, considérant toujours que les Y_i sont conditionnellement indépendantes étant donné x, nous obtenons:

$$f(y|x) = \prod_{i=1}^{n} f_i(y_i|x_i, x_{a_i}) \qquad (4.4.2)$$

Pour remplacer la couleur du pixel i à une étape donnée d'une itération de l'ICM, il est alors nécessaire de choisir comme nouvel estimé $\hat{x}_i$ celui qui maximise $p(x_i \mid y_i, \hat{x}_{s\setminus i})$ par rapport à x_i, où $\hat{x}_{s\setminus i}$ dénote la couleur aux autres points de l'image à cette étape précise de la restauration.

En suivant le même développement qu'à la section 4.2, nous obtenons:

$$\begin{aligned} p(x_i|y,\hat{x}_{s\setminus i}) &= \frac{\mathcal{L}(x_i,y,\hat{x}_{s\setminus i})}{\mathcal{L}(y,\hat{x}_{s\setminus i})} \\ &= \frac{f(y|x_i,\hat{x}_{s\setminus i})p(x_i,\hat{x}_{s\setminus i})}{\mathcal{L}(y,\hat{x}_{s\setminus i})} \\ &= \frac{f(y|x_i,\hat{x}_{s\setminus i})p(x_i|\hat{x}_{s\setminus i})p(\hat{x}_{s\setminus i})}{\mathcal{L}(y,\hat{x}_{s\setminus i})} \\ &= \frac{f(y|x_i,\hat{x}_{s\setminus i})p_i(x_i|\hat{x}_{\delta i})p(\hat{x}_{s\setminus i})}{\mathcal{L}(y,\hat{x}_{s\setminus i})} \end{aligned}$$

puis en tenant compte des pixels aux abords de chaque site i, et en utilisant (4.4.2) pour obtenir la loi conditionnelle $f_i(.|.)$, nous avons:

$$= \frac{f_i(y_i|x_i,\hat{x}_{ai}) \prod_{\substack{j=1 \\ j\neq i}}^{n} f_j(y_j|\hat{x}_j,\hat{x}_{aj})\ p_i(x_i|\hat{x}_{\delta i})\ p(\hat{x}_{s\setminus i})}{\mathcal{L}(y,\hat{x}_{s\setminus i})}$$

$$= \frac{f_i(y_i|x_i,\hat{x}_{ai}) \prod_{j\in r_i} f_j(y_j|\hat{x}_j,\hat{x}_{aj}) \prod_{\substack{j=1 \\ j\neq i \\ j\notin r_i}}^{n} f_j(y_j|\hat{x}_j,\hat{x}_{aj})\ p_i(x_i|\hat{x}_{\delta i}) p(\hat{x}_{s\setminus i})}{\mathcal{L}(y,\hat{x}_{s\setminus i})}$$

où $r_i = \{j: i \in a_j, \ \forall\ i,j \in S\}$; en d'autres mots r_i représente l'ensemble des pixels dont les abords contiennent x_i.

En se limitant aux termes contenant x_i, nous cherchons comme nouveau $\hat{x}_i$ le x_i qui maximise:

$$f_i(y_i|x_i,\hat{x}_{ai}) p_i(x_i|\hat{x}_{\delta i}) \prod_{j\in r_i} f_j(y_j|\hat{x}_j,\hat{x}_{aj}). \qquad (4.4.3)$$

Les calculs supplémentaires par rapport à (4.2.1) ne sont pas énormes: à chaque étape il faut ajouter le calcul de l'influence du pixel x_i dont on estime la nouvelle couleur sur tous les pixels contenant x_i à leurs abords.

La zone d'influence a_i de la fonction d'étalement ponctuelle autour d'un point donné est considérée ici comme étant égale au voisinage du second ordre: les 8 pixels les plus proches. En pratique il n'y a aucune restriction quant au choix de cette zone, sauf celles purement physiques expliquant le phénomène de détérioration.

Appliquer la matrice H de "brouillage" de la fonction d'étalement sur une image correspond à prendre, pixel par pixel, une moyenne

pondérée sur les pixels proches de celui considéré. Dans nos exemples, la matrice H est symétrique bien qu'il puisse être souhaitable de favoriser une direction en particulier dans le modèle de H, dans le cas du déplacement relatif entre l'objet et la caméra par exemple. Cette matrice est donc invariante par rapport à la rotation en plus d'être invariante par rapport à la translation.

Nous avons choisi ici (voir fig. 4.4.1.) la matrice 3x3

$$[H] = \{h_{k,m}: |k|,|m| \leq 1\}$$

telle que:

$$h_{k,m} = \begin{cases} \frac{1}{2}, & \text{si } k = 0,\ m = 0 \\ \frac{1}{16}, & \text{si } |k|,|m| \leq 1,\ (k,m) \neq (0,0). \end{cases} \tag{4.4.4}$$

$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$
$\frac{1}{16}$	$\frac{1}{2}$	$\frac{1}{16}$
$\frac{1}{16}$	$\frac{1}{16}$	$\frac{1}{16}$

Figure 4.4.1 *Exemple de matrice* [H] *correspondant à une fonction d'étalement ponctuelle invariante par rapport à la translation et la rotation.*

Avec une telle dégradation et un changement momentané de la notation en rangées et en colonnes, nous obtenons pour chaque pixel:

$$y_{i,j} = \Phi\left(\sum_{(k,m)} h_{k,m} x_{i-k,j-m}\right) \odot b_{i,j} \tag{4.4.5}$$

où $|k|,|m| \leq 1$, *et* $b_{i,j}$ est la réalisation du processus stochastique B, le bruit aléatoire indépendant de X.

Pour les quelques exemples où nous avons utilisé une dégradation non-linéraire Φ, nous avons choisi la racine carrée: $\Phi(x) = \sqrt{x}$. Cependant, la plupart du temps nous illustrons surtout la superposition de pixels et prenons pour Φ la fonction identité, ignorant par conséquent les distorsions non-linéaires.

Dans tous les cas le bruit est modélisé par une loi gaussienne de variance σ^2 et de moyenne μ. Dès lors, dans le cas d'un bruit additif la formation d'images est représentée par:

$$Y = \Phi(H(X)) + B. \tag{4.4.6}$$

Par conséquent la loi conditionnelle $f(Y|X=x)$ est donnée par:

$$f(Y|X=x) = f(\Phi(H(x))+B|X=x)$$

et puisque B est indépendant de X, on obtient:

$$f(Y|X=x) = f(\Phi(H(x))+B). \tag{4.4.7}$$

Ainsi, nous trouvons que si chaque $b_{i,j} \sim N(\mu,\sigma^2)$, la densité conditionnelle de Y_i étant donné X_i suit une loi gaussienne de moyenne $\mu+\Phi(H(x_i))$ et de variance inchangée σ^2.

On obtient alors comme loi conditionnelle marginale:

$$f_{i,j}(y_{i,j}|x_{i,j},xa_{i,j}) =$$

$$\frac{1}{\sqrt{2\pi\sigma^2}}\, e^{\frac{-\frac{1}{2}\{y_{i,j} - \Phi(\sum_{(k,m)} h_{k,m}\, x_{i-k,j-m}) - \mu\}^2}{\sigma^2}}. \tag{4.4.8}$$

Comme il est naturel de le faire dans ce contexte, nous supposons ici que la moyenne μ du bruit est nulle. En prenant comme nouvel estimé $\hat{x}_{i,j}$ du pixel (i,j), à une étape donnée, le $x_{i,j}$ qui maximise (4.4.3) et en utilisant toujours le modèle de champ markovien (4.3.1), on cherche le $\hat{x}_{i,j}$ qui maximise:

$$\frac{1}{\sqrt{2\pi\sigma^2}}\, e^{\frac{-\frac{1}{2}\{y_{i,j} - \Phi(\sum_{(k,m)} h_{k,m}\, x_{i-k,j-m})\}^2}{\sigma^2}} \cdot \frac{e^{\beta u_{i,j}(x_{i,j})}}{\sum_{k=1}^{c} e^{\beta u_{i,j}(k)}} \cdot$$

$$\prod_{(o,q)\in r_{i,j}} \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{\frac{-\frac{1}{2}\{y_{o,q} - \Phi(\sum_{(k,m)} h_{k,m}\, x_{o-k,q-m})\}^2}{\sigma^2}}, \qquad (4.4.9)$$

par rapport à $x_{i,j}$, avec $r_{i,j} = \{(o,q):(i,j)\in a_{o,q}\}$.

En ne tenant compte que des termes contenant $x_{i,j}$ et en prenant le négatif du logarithme de (4.4.9), le nouvel estimé de la restauration d'images pour un site i est le $\hat{x}_{i,j}$ qui <u>minimise:</u>

$$\frac{1}{2\sigma^2}\Big\{ (y_{i,j} - \Phi(\sum_{(k,m)} h_{k,m}\, x_{i-k,j-m}))^2 + \sum_{(o,q)\in r_{i,j}} (y_{o,q} - \Phi(\sum_{(k,m)} h_{k,m}\, x_{o-k,q-m}))^2 \Big\} - \beta u_{i,j}(x_{i,j}) \qquad (4.4.10)$$

par rapport à $x_{i,j}$.

Dans le cas d'un bruit multiplicatif nous supposons que celui-ci suit une loi gaussienne de moyenne $\mu=1$ et de variance σ^2. Le développement est semblable et le modèle (4.4.1) de formation d'images devient:

$$Y = \Phi(H(X)) \cdot B \tag{4.4.11}$$

En suivant la même démarche que ci-haut, la fonction de densité conditionnelle $f(Y|X=x)$ est donnée par:

$$f(Y|X=x) = f(\Phi(H(x))B) \tag{4.4.12}$$

Par conséquent, si chaque $b_{i,j}$ suit une loi gaussienne de variance σ^2 et de moyenne $\mu=1$, la densité conditionnelle de $Y_{i,j}$ étant donné $X_{i,j}$ suit une loi gaussienne de moyenne $\Phi(H(x_{i,j}))$ et de variance $[\Phi(H(x_{i,j}))]^2\sigma^2$. Ainsi, nous obtenons comme loi conditionnelle marginale:

$$f_{i,j}(y_{i,j}|x_{i,j}, xa_{i,j}) =$$

$$\frac{1}{(2\pi[\Phi(H(x_{i,j}))]^2\sigma^2)^{\frac{1}{2}}} e^{\frac{-\frac{1}{2}\{y_{i,j} - \Phi(\sum_{(k,m)} h_{k,m}\, x_{i-k,j-m})\}^2}{[\Phi(H(x_{i,j}))]^2\sigma^2}}. \tag{4.4.13}$$

Nous prendrons dans ce cas comme estimé le $\hat{x}_{i,j}$ qui minimise:

$$\frac{1}{2[\Phi(H(x_{i,j}))]^2\sigma^2}\left[y_{i,j} - \Phi(\sum_{(k,m)} h_{k,m}\, x_{i-k,j-m})\right]^2 +$$

$$\sum_{(o,q)\in r_{i,j}}\left[\frac{1}{2[\Phi(H(x_{o,q}))]^2\sigma^2}\left[y_{o,q} - \Phi(\sum_{(k,m)} h_{k,m}\, x_{o-k,q-m})\right]^2\right]$$

$$- \beta u_{i,j}(x_{i,j}) \tag{4.4.14}$$

par rapport à $x_{i,j}$, et bien sûr:

$$\Phi(H(x_{i,j})) \equiv \Phi(\sum_{(k,m)} h_{k,m}\ x_{i-k,j-m}).$$

Dans le cas du seul bruit multiplicatif, sans fonction d'étalement ponctuelle, le critère se simplifie en:

$$\frac{1}{2[\Phi(x_{i,j})]^2\sigma^2}\left[y_{i,j} - \Phi(x_{i,j})\right]^2 - \beta u_{i,j}(x_{i,j}). \qquad (4.4.15)$$

4.5 Reconstruction par blocs

Nous savons que l'ICM fournit une approximation de l'estimé du maximum *a posteriori*. Il cherche, à chaque site, la couleur maximisant une loi conditionnelle dépendant de la reconstruction autour du point et de l'observation à ce point. Ainsi, on procède pixel par pixel, évitant d'avoir à considérer tous les pixels globalement.

Une solution différente que suggère Besag [1986], et qui constitue un pas vers la globalisation, consiste à estimer la configuration de couleur d'un bloc B représentant une région composée d'un nombre restreint de pixels autour du site i.

Pour cela il s'agit de prendre comme estimé le $\hat{x}_B$ qui maximise la probabilité $P[X_B=x_B \mid y, \hat{x}_{S\setminus B}]$ par rapport à x_B, où $x_{S\setminus B}$ dénote l'image

entière à l'exception de la région B. Cela revient à maximiser la loi conditionnelle d'un bloc de pixels plutôt que d'un seul pixel.

Le critère de l'ICM pour ce qui à trait à la reconstruction par blocs s'obtient comme suit. Nous cherchons le $\hat{x}_B$ qui maximise:

$$p(x_B | y, \hat{x}_{S\setminus B}) = \frac{\mathcal{L}(x_B, y, \hat{x}_{S\setminus B})}{\mathcal{L}(y, \hat{x}_{S\setminus B})} \qquad (4.5.1)$$

par rapport à x_B. Oublions tout de suite le dénominateur qui ne contient pas de termes en x_B. Nous obtenons:

$$\begin{aligned} p(x_B | y, \hat{x}_{S\setminus B}) &\propto \mathcal{L}(x_B, y, \hat{x}_{S\setminus B}) \\ &\propto f(y | x_B, \hat{x}_{S\setminus B})\, p(x_B, \hat{x}_{S\setminus B}) \\ &\propto f(y | x_B, \hat{x}_{S\setminus B})\, p(x_B | \hat{x}_{S\setminus B})\, p(\hat{x}_{S\setminus B}). \end{aligned}$$

Puisque cette expression nous intéresse en tant que fonction de x_B, nous pouvons également rentrer $p(\hat{x}_{S\setminus B})$ dans la constante de proportionnalité. Nous obtenons alors:

$$\begin{aligned} p(x_B | y, \hat{x}_{S\setminus B}) &\propto f(y | x_B, \hat{x}_{S\setminus B})\, p(x_B | \hat{x}_{S\setminus B}) \\ &\propto \prod_{i\in B} f_i(y_i | x_i) \prod_{\substack{j=1 \\ j\notin B}}^{n} f_j(y_j | \hat{x}_j)\, p_B(x_B | \hat{x}_{\delta B}) \\ &\propto \prod_{i\in B} f_i(y_i | x_i)\, p_B(x_B | \hat{x}_{\delta B}) \qquad (4.5.2) \end{aligned}$$

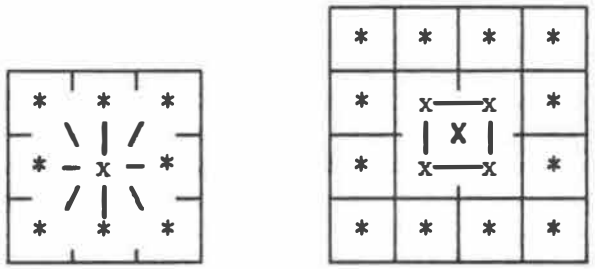
où nous conservons uniquement les termes contenant x_B; $p_B(x_B | \hat{x}_{\delta B})$ correspond à la loi conditionnelle marginale conjointe des pixels de la région B.

A chaque étape de l'ICM, nous remplaçons donc x_B par le $\hat{x}_B$ maximisant (4.5.2) par rapport à $\{x_i : i \in B\}$, ce qui exige toutefois plus de calculs.

Considérons en effet le problème du calcul de $p_B(x_B \mid \hat{x}_{\delta B})$. En suivant les mêmes développements qui donnent les caractéristiques locales pour un seul pixel au chapitre 3 (équation (3.4.3)), il faut considérer ici la somme des cliques disctinctes contenant au moins 1 pixel du bloc B ($\mathcal{C}_1 = \{C : \exists\ i \in B,\ i \in C\}$).

Or, pour un champ markovien défini sur un système de voisinages du second ordre, pour lequel on ne considère que les cliques de 1 ou de 2 pixels (modèle général (3.5.4)), et pour une région B (intérieure à la grille S) formée de 4 pixels, il existe π_B=26 paires <u>distinctes</u> de pixels à l'horizontale (−), à la verticale (|) ou en diagonale (/,\) contenant au moins un pixel du bloc B.

En effet, comme le montre la figure 4.5.1 (a), chacun des 4 pixels peut être contenu dans 8 cliques de 2 pixels, pour un total de 4x8=32. En retirant les 6 paires de pixels qui apparaissent deux fois - ce sont les paires formées de 2 pixels du bloc B, voir figure 4.5.1 (b) -, on obtient π_B=32-6=26 paires distinctes de pixels.



(a) (b)

Figure 4.5.1. *Bloc de 4 pixels donnant* $\pi_B = 8x4-6=26$ *paires distinctes de pixels: (a) chaque pixel du bloc fait partie de 8 cliques de 2 pixels; (b) les paires formés de 2 pixels appartenant au bloc - ceux marqués par un* x *- apparaissent 2 fois.*

Il faut ainsi, à chaque étape, choisir pour le bloc un coloriage parmi les c^4 possibles en utilisant les observations des 4 pixels et, pour un champ markovien défini sur un système de voisinages du second ordre, les 26 pixels directement ou diagonalement adjacents.

En suivant le même développement qui fournit, au chapitre 3 (équations (3.5.4) à (3.5.7)), le champ markovien que nous utilisons, le terme de l'exponentielle sera:

$$- \beta\pi_{dB} = - \beta(\pi\beta - \pi_{mB}) = - \beta\pi\beta + \pi_{mB}$$

où π_{dB} est le nombre de paires distinctes de pixels voisins entre eux de couleurs différentes, dont au moins un appartient à B, et où π_{mB} est le nombre de paires de pixels de couleurs identiques.

Comme π_B ne dépend pas des couleurs, nous obtenons alors comme critère de l'ICM:

$$\underset{x_B}{Max} \left[p(x_B | y, \hat{x}_{S \setminus B}) \propto \prod_{i \in B} f_i(y_i | x_i) \exp(\beta \pi_{mB}) \right]. \qquad (4.5.3)$$

4.6 Implantation sur ordinateur

Nous présentons dans cette section des images réelles connues, généralement créées de toutes pièces, préalablement dégradées puis soumises à l'ICM pour être restaurées. Plusieurs types d'images originales possédant des caractéristiques particulières sont considérés. Nous avons fait varier les dégradations auxquelles elles ont été soumises ainsi que les paramètres de l'ICM.

4.6.1 Considérations techniques

Tous les exemples montrés proviennent de l'implantation en langage Turbo-Pascal 4.0 des algorithmes que nous avons développés sur un micro-ordinateur XT compatible IBM. Le système est équipé d'un microprocesseur Intel-8088 fonctionnant à une vitesse de 8 Mhz, d'un coprocesseur mathématique Intel-8087 et d'une carte graphique Hercules monochrome. Nous donnons au chapitre 7 tous les détails concernant l'utilisation et la structure interne des programmes.

4.6.2 Synchronisation de la mise à jour

Un point technique, provenant du fait que l'ICM repose sur une méthode doublement itérative, mérite cependant d'être développé ici. L'ICM est doublement itérative puisque, premièrement, un estimé provisoire de l'image originale est obtenu en procédant par une estimation pixel après pixel, et, deuxièmement, puisque le processus est répété une seconde, une troisième fois, en fait jusqu'à la convergence.

Dès lors une question se pose d'où découle le point que nous voulons soulever: à quel moment faut-il remplacer la nouvelle couleur estimée $\hat{x}_i$ d'un pixel i de l'image? A première vue, la solution paraît simple : la mise à jour se fait dès que la nouvelle couleur est connue. Mais procéder de cette façon amène le problème suivant. Selon la direction spatiale du "balayage", c'est-à-dire la séquence d'estimations et de mises à jour des couleurs, les pixels servant au calcul du critère de l'ICM ne sont pas tous à la même itération.

Considérons en effet que le balayage s'effectue de gauche à droite et de haut en bas, procédé classique en traitement d'images, et que chaque pixel est mis à jour dès que possible. Comme le montre la figure 4.6.1, le pixel intérieur (i,j) est mis à jour avant le pixel $(i+1,j)$ et après le pixel $(i-1,j)$. Ainsi, au moment de calculer le critère de l'ICM qui nécessite la connaissance des 8 pixels voisins, 4

de ces pixels (notés +) ont déjà été estimés et remplacés, tandis que les 4 autres (notés -) n'ont pas encore été "visités".

+ i-1,j-1	+ i,j-1	+ i+1,j-1
+ i-1,j	x i,j	- i+1,j
- i-1,j+1	- i,j+1	- i+1,j+1

Figure 4.6.1 *Ordre des visites de l'ICM lors d'un balayage classique de gauche à droite et de haut en bas. Au site intérieur* (i,j), les pixels notés + *ont déjà été mis à jour, les pixels notés - ne l'ont pas encore été.*

Il est possible que ceci puisse avoir une influence sur la restauration, sous la forme d'effets directionnels visibles sur l'image estimée par exemple. Une mise à jour de ce type sera dite *asynchrone*.

Si l'on peut pallier à ce problème potentiel en variant la direction du balayage à chape itération complète, une autre solution consiste à ne plus écrire immédiatement dans l'image en reconstruction la nouvelle couleur estimée de chaque pixel mais à la conserver dans la mémoire de l'ordinateur. L'ICM est appliqué sur des pixels qui sont alors tous exactement au même niveau de restauration à chaque étape. A la fin de l'itération, une fois tous les pixels visités et leur nouvelle couleur estimée, ils sont mis à jour en même temps en une seule opération. Nous nommerons cette méthode balayage *synchrone*.

Cependant, comme le fait remarquer Besag [1986], la convergence n'est plus assurée pour un balayage de ce type. On peut supposer de plus qu'elle se produit plus lentement - bien que nos tests ne semblent pas assez significatifs sur ce point pour le confirmer en pratique-, les pixels déjà estimés aidant en quelque sorte le choix d'une nouvelle couleur lors d'un balayage asynchrone.

Une autre solution consiste à effectuer la mise à jour des pixels instantanément, comme dans le premier cas considéré, mais en parcourant l'image selon un ordre particulier. Le balayage proposé, que nous appelerons balayage *semi-synchrone*, visite une ligne sur deux, un pixel sur deux. En variant la ligne et le pixel de départ, la figure 4.6.2 nous montre que ce schéma de mise à jour requiert 4 passages pour compléter une itération.

1	3	1	3
4	2	4	2
1	3	1	3
4	2	4	2

Figure 4.6.2 *Ordre de passage d'un balayage* semi-synchrone. *A chaque étape les voisins sont répartis symétriquement.*

L'intérêt de cette dernière solution réside dans le fait qu'à chaque étape, les pixels déjà estimés sont toujours répartis symétriquement autour du site considéré. Aucune direction n'est favorisée et ce balayage semi-synchrone ne nécessite pas de mémoire supplémentaire pour conserver des résultats intermédiaires[3].

Cependant nos essais nous ont permis de constater que le choix du balayage n'a finalement pas une importance capitale. Selon les images considérées et sans qu'il soit possible de le prédire, l'une ou l'autre méthode donnera de meilleurs résultats.

Nous avons donné aux images des symboles permettant d'identifier le type de mise à jour utilisé. Ce sont les codes **ASYN**, **SYN** ou **SSYN2** selon que le balayage ait été de type asynchrone, synchrone ou semi-synchrone respectivement.

4.6.3 Choix de β et de σ^2

Nous indiquons toujours dans nos exemples la valeur de β choisie dans le modèle de champ markovien utilisé dans l'ICM. Notons ici que tous les exemples de restauration sont basés sur le modèle (4.3.1).

---

[3] Le schéma du balayage semi-synchrone nous a été suggéré par M. Sylvain Archambault.

Dans le modèle du bruit gaussien additif et donc dans le critère de l'ICM, il est nécessaire de connaître la variance σ^2 du processus de bruit aléatoire. Nous n'abordons pas ici le problème de l'estimation de paramètre et supposons toujous que σ^2 est connu. Sauf autrement mentionné, la variance utilisée dans le critère de l'ICM sera toujours celle ayant servi à la dégradation.

4.6.4 Modèles de dégradations

Nous étudions dans certains exemples le comportement de l'ICM face à des dégradations de plusieurs types. Ce sont, avec entre parenthèses le symbole utilisé pour décrire les images, une combinaison de une ou plusieurs des détériorations suivantes:

- un bruit additif gaussien (**GA**);
- un bruit multiplicatif gaussien (**GM**);
- un bruit additif uniforme (**UNI**);
- une distorsion non-linéaire (**NL**), toujours une racine carrée;
- une dégradation par fonction d'étalement ponctuelle (**FEP**), toujours le modèle (4.4.4).

Notons que l'ICM est prévu pour une détoriation aléatoire et qu'un bruit additif ou multiplicatif est toujours présent dans la dégradation, le cas purement déterministe ne nous intéressant pas ici.

Il est important de souligner également que nous avons effectué la

dégradation d'une image par une fonction d'étalement ponctuelle de façon totalement synchrone.

4.6.5 Code de couleur

Les "couleurs" utilisées dans nos images, de valeurs 1 à 4, sont représentées par des symboles formés de points de plus en plus noirs. Le caractère précis associé à chaque couleur peut cependant varier d'un exemple à l'autre.

4.6.6 Exemples et résultats

4.6.6.1 Mona Lisa modifiée: convergence et choix de β

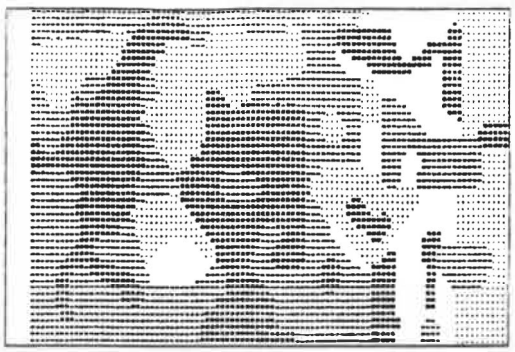
L'image originale du premier exemple est montrée à la figure 4.6.3 (a). Composée de 4 couleurs sur une grille 50x100, elle représente une Joconde légèrement retouchée à laquelle nous avons ajouté sur la droite certains motifs simples mais qui possèdent des caractéristiques intéressantes pour l'étude du bruitage puis de la restauration.

Nous avons généré des observations dégradées par un bruit gaussien selon une distribution normale *circulaire* avec $\sigma^2 = 0.5$. La première étape consiste à obtenir la reconstruction par le maximum de vraisemblance, ce qui nous donne un taux d'erreur de 48.46 % (voir la figure 4.6.3 (b) où il est difficile de reconnaître l'image originale).

Par la suite, nous avons soumis l'image dégradée à l'ICM avec différentes valeurs de β. L'objectif était de tester tout d'abord l'importance de β puis de vérifier si la méthode convergeait effectivement. Les résulats des trois restaurations sont montrés aux figures 4.6.3 (c), 4.6.3 (d) et 4.6.3 (e), après 10 itérations de l'ICM et β=1.5, β=1.0 et β=2.0 respectivement, avec une mise à jour de type synchrone. La valeur β=1.5 est celle suggérée par Besag dans son article.

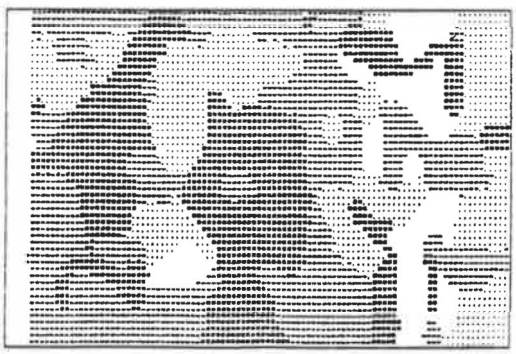
Nous avons observé des taux d'erreur de 9.4%, 8.76% et 9.82% respectivement. Bien que β=1.0 semble donner de meilleurs résultats, il est très difficile de voir des différences entre les trois images restaurées.

La figure 4.6.3 (f) montre les courbes de taux d'erreur par itération pour chacune des trois valeurs de β. Remarquons que les courbes se stabilisent rapidement et semblent converger. En effet, nous avons observé qu'après 6 itérations l'ICM produit moins de 1% de changement.

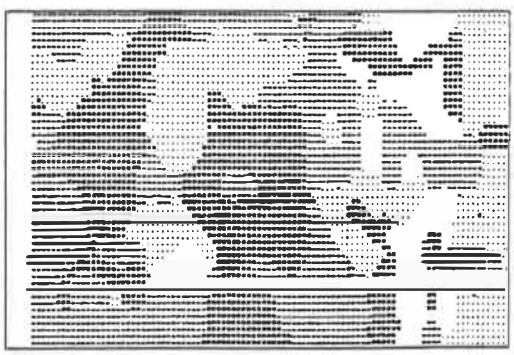


(a)

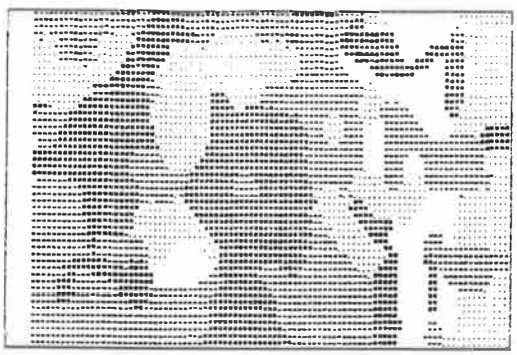
(b)



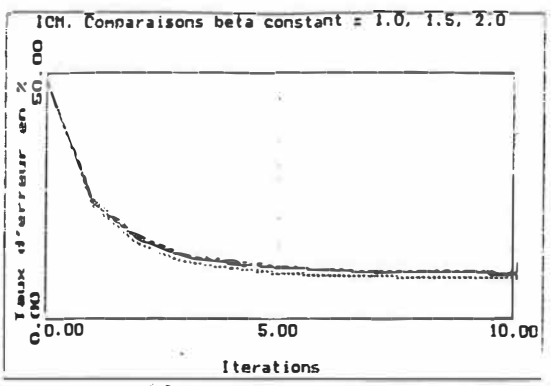
(c)



(d)



(e)



(f)

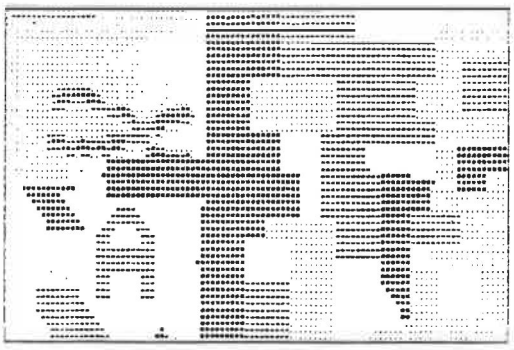
Figure 4.6.3 *Restauration par l'ICM et convergence:* ***(a)*** *Image originale «Monal Lisa modifiée»;* ***(b)*** *dégradation* ***GA****,* $\sigma^2=0.5$*, donnant un taux d'erreur de 48.46%;* ***(c)*** *restauration ICM-SYN avec modèle GA,* $\beta=1.5$ *constant, 10 itérations, err=9.40%;* ***(d)*** *avec* $\beta=1.0$*, err=8.76%;* ***(e)*** *avec* $\beta=2.0$*, err=9.82%;* ***(f)*** *courbe de convergence.*

4.6.6.2 Chromosomes modifiés, bruit uniforme et «vote de la majorité»

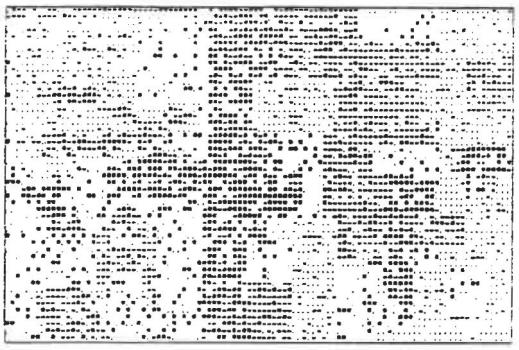
Le deuxième exemple, à la figure 4.6.4 (a), montre une autre image intéressante présentant des particularités que l'on ne retrouvait pas sur la «Mona Lisa»: figures géométriques (carrés, rectangles, cercles, diagonales) avec superposition et juxtaposition entre différentes couleurs, lettre «A», chromosome non adouci, etc.

La reconstruction (fig. 4.6.4 (c)) après 6 itérations de l'ICM avec β=1.2 constant est particulièrement bonne, donnant un taux d'erreur de 5.3%. Remarquons particulièrement le «A» qui est entièrement reconstruit, les diagonales et les figures géométriques bien restituées. La principale source d'erreur provient du chromosome. Celui-ci est sans doute formé de détails trop fins pour que l'ICM puisse les reconstruire.

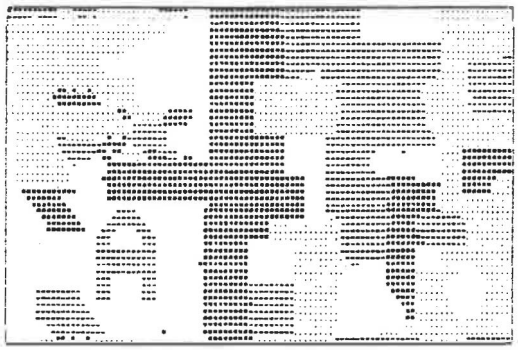
Ceci ne nous semble pas être une limite de l'ICM mais plutôt une caractéristique à laquelle il fallait s'attendre d'après le choix du modèle de champ markovien.



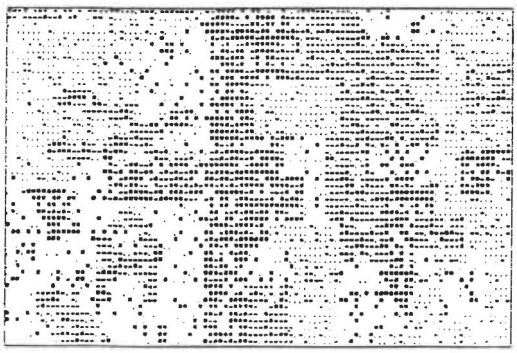
(a)



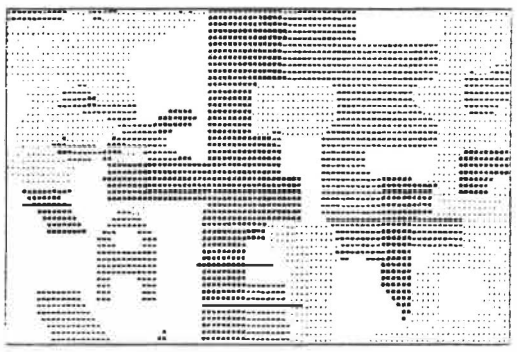
(b)



(c)



(d)



(e)

Figure 4.6.4 *Restauration par l'ICM avec UNI:* ***(a)*** *Image originale «Chromosome modifié»;* ***(b)*** *dégradation* ***GA****,* $\sigma^2=0.3$*, donnant un taux d'erreur de 36%;* ***(c)*** *restauration ICM-SYN avec modèle GA,* $\beta=1.2$ *constant, 6 itérations, err=5.3%;* ***(d)*** *dégradation* ***UNI****, err=33%;* ***(e)*** *ICM-SYN GA,* $\beta=1.2$*, 6 itérations, err=5.78%* ***(e)*** *ICM-SYN GA,* $\beta=1.2$*, err=5.78%.*

Nous présentons également un autre exemple à partir du chromosome de la figure 4.6.4 (a). Cette fois nous avons dégradé l'image avec un bruit uniforme (fig. 4.6.4 (d)), et ce sans modifier notre critère de l'équation (4.3.4), c'est-à-dire sans tenir compte de ce changement dans le modèle de la dégradation.

Ceci nous permet de vérifier la robustesse de l'ICM sur des images réelles où le modèle de dégradation n'est pas toujours parfaitement connu. On voit à la figure 4.6.4 (e) que la restauration reste très bonne, avec moins de 6% de taux d'erreur après 6 itérations.

Finalement, comme dernier exemple sur cette image, nous présentons une comparaison entre l'ICM et le «vote de la majorité». Cette méthode populaire est obtenue avec l'algorithme de l'ICM en prenant une très grande valeur pour β de sorte que seule la deuxième partie de l'équation (4.3.4) est considérée: on donne à chaque pixel la couleur que possède le plus grand nombre de ses voisins.

Nous reprenons l'image originale 4.6.5 (a) que nous bruitons avec une loi gaussienne pour laquelle σ^2=0.3, ce qui donne un taux d'erreur de 37% (fig. 4.6.5 (b)). Les figures 4.6.5 (c) et 4.6.5 (d) représentent respectivement une reconstruction avec le «vote de la majorité» et l'ICM avec un β croissant de 0.5 à 1.5 par pas égaux de 0.2 à chaque itération.

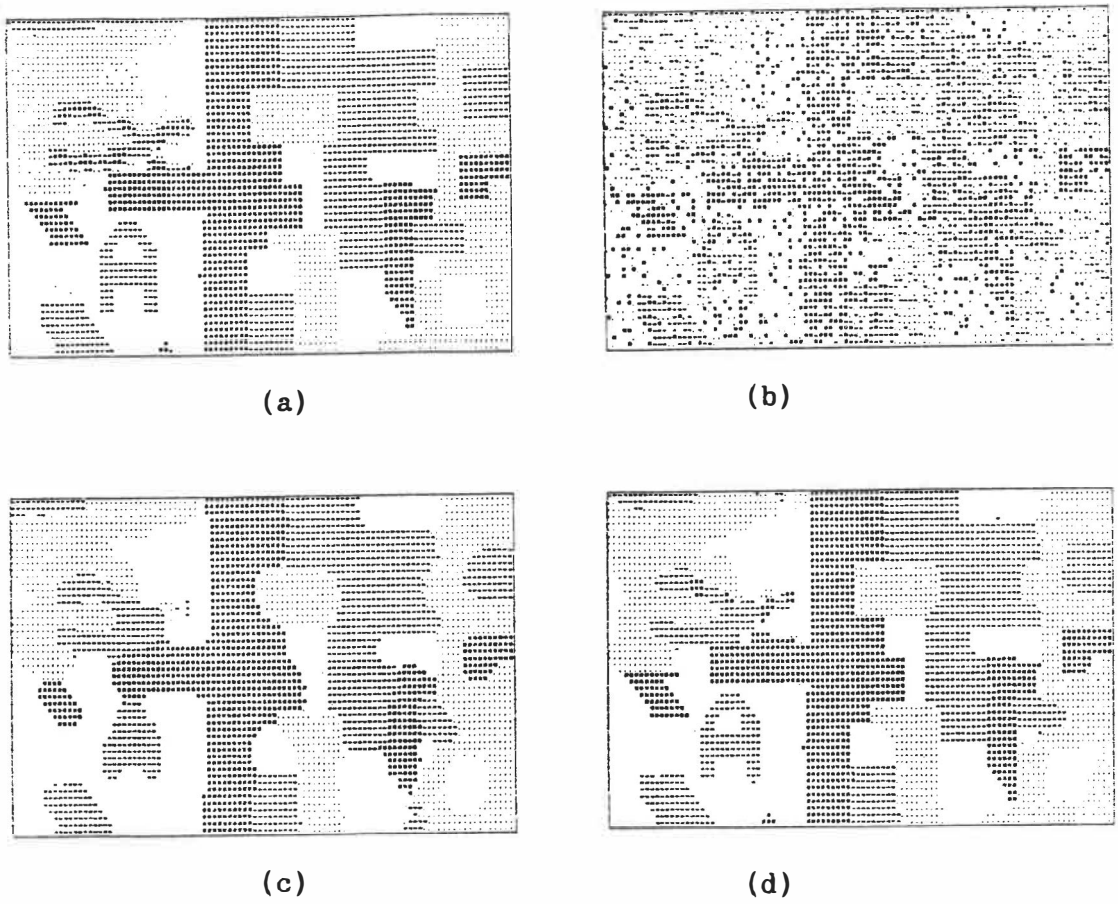


Figure 4.6.5 *Restauration par l'ICM et «vote de la majorité»: **(a)** Image originale «Chromosome modifié»; **(b)** dégradation **GA**, $\sigma^2=0.3$, donnant un taux d'erreur de 37%; **(c)** restauration «vote de la majorité» ($\beta=1000$), 6 itérations, err=10.0%; **(d)** ICM-SYN GA, $\beta=0.5$ à $\beta=1.5$, 6 itérations, err=5.2%.*

Remarquons que même si le taux d'erreur de 10% n'est pas très important pour la reconstruction avec le «vote de la majorité» (fig. 4.6.5 (c)), il est tout de même le double de celui de l'ICM (fig. 4.6.5 (d)) pour la même image. On voit où se situent les différences: le «vote de la majorité» élimine rapidement les petits groupes de pixels pour les fondre dans la masse et arrondir les angles (le «A»

n'est plus reconnaissable, tandis que l'ICM le restitue presque sans erreur).

De plus, nous avons réalisé la reconstruction avec l'ICM en implantant la suggestion de Besag [1986], consistant à augmenter β à chaque itération, ici de 0.5 à 1.5. En général, une telle approche réduit le taux d'erreur final. En effet, d'itération en itération, l'erreur diminuant normalement graduellement vers un minimum, il est plausible que le terme d'erreur soit de moins en moins important tandis que le modèle *a priori* prend de la valeur.

4.6.6.3 Image *TT* et fonction d'étalement ponctuelle

Nous présentons maintenant une nouvelle image originale «TT» à la figure 4.6.6 (a), construite spécialement pour faciliter l'interprétation de résultats. On ne voit que 2 couleurs, mais celles-ci sont en fait à l'extrémité d'une échelle de 4 couleurs; nous leur avons assigné les valeurs 1 et 4.

Grâce à ce procédé, il est plus facile de constater l'effet de la fonction d'étalement ponctuelle (FEP), une sorte de moyenne pondérée des pixels voisins (figure 4.6.6 (b)). La FEP a tendance à donner des valeurs intermédiaires aux couleurs et se comporte ici comme un "flou", donnant une erreur de 43%.

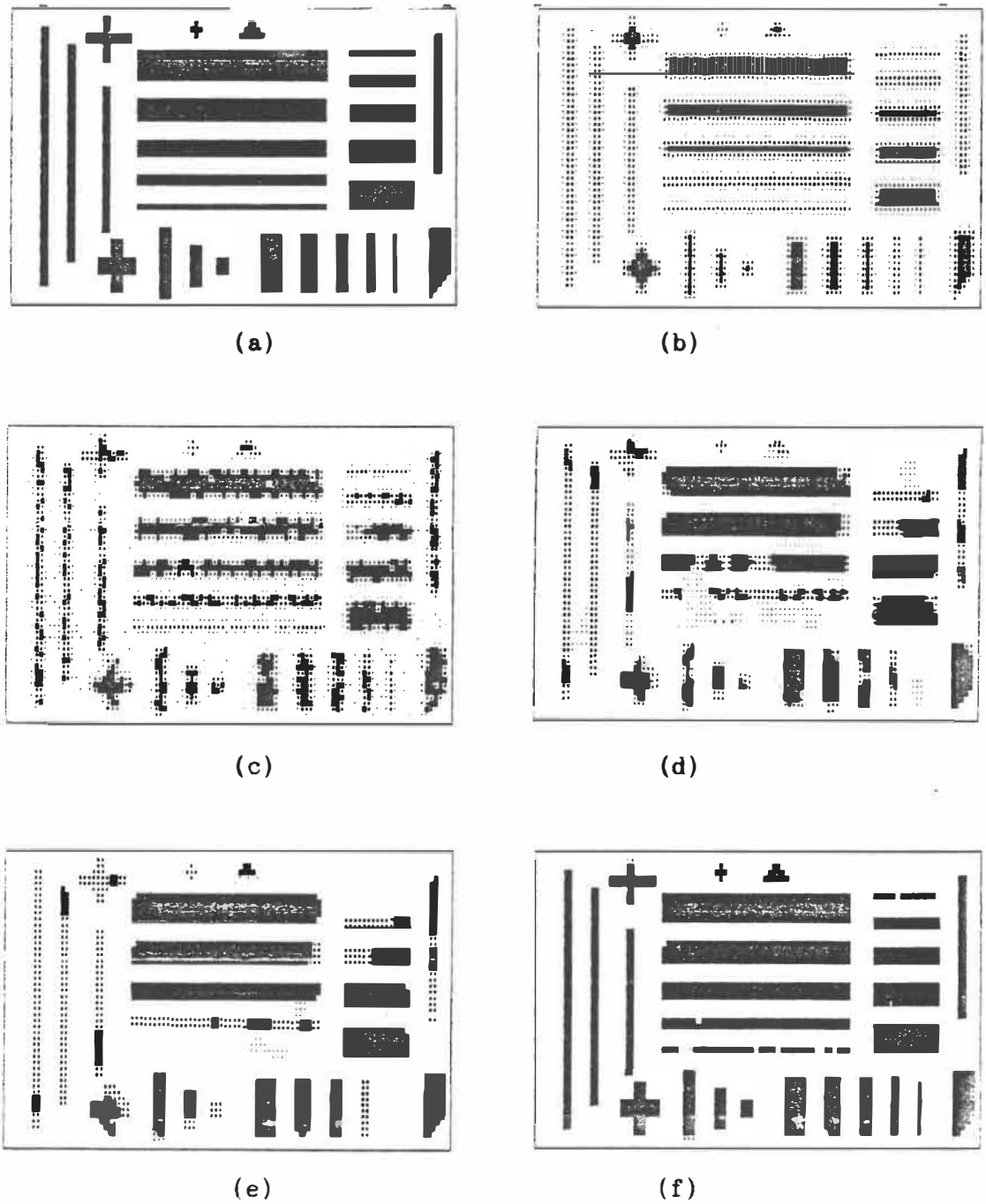


Figure 4.6.6 *Restauration par l'ICM avec FEP:* ***(a)*** *Image originale «TT»;* ***(b)*** *dégradation* ***FEP*** *err=43.24%;* ***(c)*** *superposition supplémentaire* ***FEP+GA*** σ^2*=0.1, err=31.02%;* ***(d)*** *restauration ICM-SYN GA, 6 itérations,* β*=1.5, err=17.43%;* ***(e)*** *ICM-ASYN GA, 6 itérations,* β*=1.5,* σ^2 *=0.32, err=11.29%;* ***(f)*** *restauration ICM-SYN GA-FEP, 6 itérations,* β*=1.5, err=0.49%.*

L'image à la figure 4.6.6 (c) est le résultat de l'addition d'un faible bruit gaussien (σ^2=0.1) à l'image précédente. Il est amusant de remarquer que le taux d'erreur a diminué (err=31%). Cela s'explique par le fait que le bruit a été distribué de façon égale entre les "bons" et les "mauvais" points de 4.4.6 (b) et a parfois ramené ces derniers dans le droit chemin. Le bruit étant plus important pour les couleurs intermédiaires (2,3), et puisque l'on retrouve plus de "mauvais" pixels ayant ces couleurs, le taux d'erreur global a baissé.

A la figure 4.6.6 (d) on voit le résultat de l'ICM après 6 itérations lorsque le critère ne tient pas compte de la fonction d'étalement ponctuelle, c'est-à-dire lorsqu'on utilise (4.3.4) et non (4.4.10). Bien sûr la restauration ne donne pas de très bons résultats (err=17.43%) bien qu'une partie importante de l'erreur soit due au choix de la couleur: la plupart des barres noires sont récupérées mais dans une couleur trop faible.

La figure 4.6.6 (e) a un bien meilleur taux d'erreur de 11.29%. On retrouve deux changements dans l'ICM ayant produit cette nouvelle image estimée. Premièrement, nous avons procédé à une mise à jour asynchrone au lieu de simultanée; deuxièmement, et nous pensons qu'il s'agit là du point important dans ce cas, nous avons augmenté la valeur de la variance dans le critère (4.3.4) à 0.32, alors que pour l'image précédente nous avions conservé σ^2=0.1. Ceci est important car l'image

dégradée 4.6.6 (c) représentant les observations a en fait une erreur plus grande que ne l'aurait causée un simple bruit additif de variance égale à 0.1. Une bonne valeur de σ^2 s'avère donc un facteur majeur pour la qualité de la restauration.

La dernière image à la figure 4.6.6 (f) a été obtenue en appliquant l'ICM modifié afin de tenir compte de la fonction d'étalement ponctuelle, ce nouveau critère étant donné à l'équation (4.4.10). Le taux d'erreur observé, moins de 0.5%, est excellent. En termes quantitatifs, c'est 20 fois mieux que celui de la figure 4.4.6 (e)! Remarquons en particulier la reconstruction, absente des deux autres images, des 2 lignes de 1 pixel de largeur, l'une verticale l'autre horizontale, à droite de l'image, ainsi que la petite croix et le triangle en haut, parfaitement restitués.

Ce dernier résultat n'aurait pas pu être obtenu sans tenir compte de la fonction d'étalement ponctuelle dans le critère de l'ICM, à l'équation (4.4.10). On peut penser qu'il serait judicieux d'appliquer l'inverse de la détérioration déterministe sur l'image restaurée par l'ICM classique. Toutefois ce n'est pas possible, la matrice inverse $[H]^{-1}$ n'existe pas et de plus nous avons vu au chapitre 2 qu'appliquer le filtre inverse en présence de bruit ne donne pas de bons résultats. On s'apperçoit d'ailleurs que même s'il était possible de reconstituer l'image originale 4.6.6 (a) à partir des observations 4.6.6 (b) de la dégradation déterministe, les images restaurées 4.6.6 (d) et 4.6.6 (e)

ne ressemblent pas particulièrement à 4.4.6 (b), ce qui laisse croire que le résultat ne serait pas très bon.

Mentionnons toutefois que les résultats ne sont pas toujours aussi impressionnants pour d'autres images originales. Nous avons procédé à des essais supplémentaires où la restauration, bien que meilleure lorsque l'on tient compte de la fonction d'étalement ponctuelle, donne des taux d'erreurs comparables à la méthode classique.

4.6.6.4 Image *TT* et bruit gaussien multiplicatif

Les derniers exemples que nous présentons dans cette section concernent le bruit multiplicatif gaussien. La figure 4.6.7 (a) représente l'image initiale et la figure 4.6.7 (b) le résultat de l'application d'un bruit gaussien de variance σ^2=0.2 multiplié à l'image originale. L'erreur initiale est de 21%.

Pour l'ICM classique appliquée sur cette image dégradée, le résultat est donné à la figure 4.6.7 (c), montrant un taux d'erreur de plus de 8%. Lorsque nous utilisons le modèle approprié (4.4.15) qui tient compte du bruit multiplicatif, nous observons à la figure 4.6.7 (d) un taux d'erreur de 3% après 6 itérations.

Comme pour la fonction d'étalement ponctuelle, le nouveau modèle de dégradation proposé dans le cas d'un bruit multiplicatif s'avère donner de bons résultats.

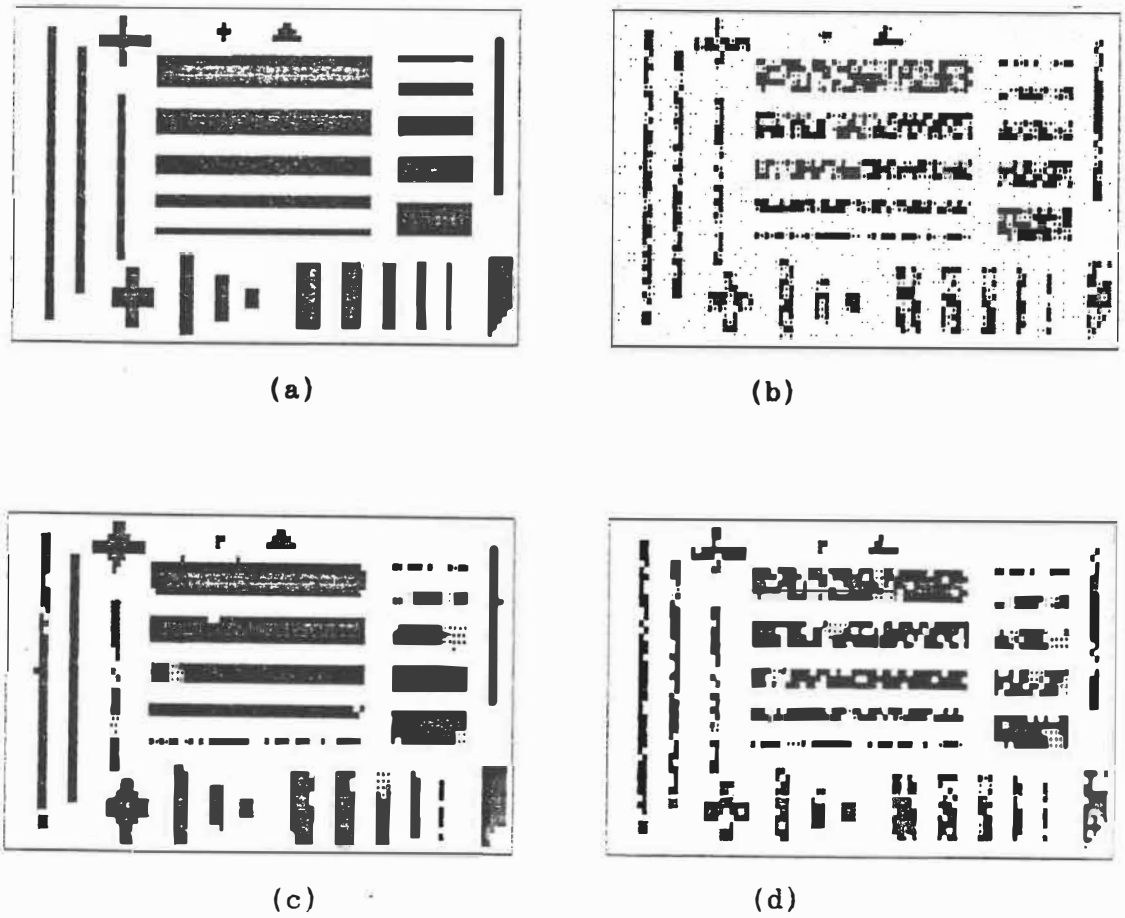


Figure 4.6.7 *Restauration par l'ICM avec GM:* ***(a)*** *Image originale «TT»;* ***(b)*** *dégradation* ***GM****,* $\sigma^2=0.2$*, donnant un taux d'erreur de 21.56%;* ***(c)*** *restauration ICM-SYN avec modèle GA,* $\beta=1.5$ *constant, 6 itérations, err=8.38%;* ***(d)*** *ICM-SYN GM,* $\beta=1.5$*, 6 itérations, err=3.08%.*

4.6.6.5 A propos de la vitesse

Il nous a paru important de noter le temps habituellement nécessaire pour restaurer une image en appliquant l'ICM. Pour une image de 50x100 comme celles sur lesquelles nous avons travaillé, et pour 4 couleurs, un cycle complet de 6 itérations de (4.3.1) prend à peu près 4 minutes sur notre équipement. Lorsqu'on veut tenir compte du bruit

multiplicatif, le temps passe à 5 minutes. Finalement pour employer (4.4.10) qui calcule l'influence d'une fonction d'étalement ponctuelle, le temps requis est de plus de 50 minutes.

En général donc, les images obtenues par l'ICM sont de bonne qualité et s'avèrent être de bons estimés de l'image originale. Le procédé de traitement pour sa part est relativement rapide. Ces facteurs, en plus du fait que la méthode peut être facilement modifiée pour tenir compte de phénomènes complexes, font que l'ICM est un outil de premier ordre pour la restauration d'images par ordinateur.

CHAPITRE 5: DERIN ET ELLIOTT ET LA PROGRAMMATION DYNAMIQUE

5.1 Maximum *a Posteriori*

Derin et Elliott [1987] travaillent à partir de formulations, d'hypothèses et d'outils mathématiques très proches de ceux de Besag. En utilisant des champs markoviens pour modéliser les images et en calculant une approximation de l'estimation par le maximum *a posteriori* (MAP), ils présentent dans leur article, en particulier, un algorithme de restauration d'images dégradées par du bruit blanc.

Les principes de base de Derin et Elliott sont à peu près les mêmes que ceux de Besag [1986]. De plus, certaines hypothèses communes, d'indépendance notamment, bien qu'elles ne soient pas explicitement décrites, sont toujours implicites dans leurs développements mathématiques. Nous conserverons les notations de Besag pour décrire des concepts identiques.

Par conséquent, soit X un champ aléatoire markovien et x sa réalisation, et soit y une observation de x (après dégradation), considérée comme la réalisation d'un champ aléatoire markovien Y.

La loi *a posteriori* $p(x|y)$ est donnée par la formule de Bayes:

$$p(x|y) = \frac{f(y|x)\ p(x)}{f(y)}. \qquad (5.1.1)$$

Dans ce contexte, la recherche du maximum *a posteriori*, c'est-à-dire l'estimé (l'image) $\hat{x}$ qui maximise (5.1.1) par rapport à x, puisque y n'affecte pas la maximisation, équivaut à maximiser le numérateur du membre de droite de (5.1.1), soit la probabilité conjointe $\mathcal{P}(x,y)$, ou encore son logarithme:

$$\ln[\ f(y|x)\ p(x)] = \ln \mathcal{P}(x,y) = \ln p(x) + \ln f(y|x). \qquad (5.1.2)$$

Bien sûr, la taille du problème de calcul demeure la même. Derin et Elliott s'y attaquent en formulant la maximisation de (5.1.2) sous une forme <u>récursive</u> et en appliquant la programmation dynamique par "morceaux", comme nous l'expliquons ci-dessous en décrivant en détail les modèles et l'algorithme.

5.2 Modèles

Soit $X = \{X_{ij}\}$ un champ aléatoire défini sur une grille S de pixels définie par:

$$S = \{(i,j): 1 \le i \le N_1,\ 1 \le j \le N_2\}.$$

Remarquons que nous utilisons la notation en rangées et en colonnes, nécessaire pour la suite du développement de la méthode.

Par rapport à un système de voisinages $\mathcal{G}$, la distribution conjointe du champ X est donnée par:

$$P(X=x) = \frac{1}{Z} e^{-U(x)}, \qquad (5.2.1)$$

où

$U(x) = \sum_{C \in \mathcal{C}} Vc(x)$ est une fonction d'énergie,

$Vc(x)$ = *potentiel* associé à la clique C,

et $Z = \sum_{x} e^{-U(x)}$ est la fonction de répartition.

Pour définir une distribution de Gibbs (DG) il suffit de spécifier son système de voisinages $\mathcal{G}$, les cliques associées et leurs potentiels, les $Vc(x)$.

La classe de DG utilisée par Derin et Elliott est la suivante. Pour un champ aléatoire X homogène (potentiels indépendants de la position des cliques dans S) et constitué de variables aléatoires discrètes $\{X_{ij}\}$ prenant des valeurs dans $\Gamma = \{k_1, k_2, \ldots, k_C\}$, la distribution est définie à partir d'un système de voisinages du second ordre $\mathcal{G}^2$ (l'extension aux autres ordres étant évidente).

Les potentiels associés aux cliques dans $\mathcal{G}^2$ sont définis comme suit: un paramètre β_t est assigné à chaque catégorie t de clique, sauf pour les cliques d'un seul pixel, qui prennent des valeurs différentes α_k pour chaque couleur k. Plus précisément, pour les cliques de 2 pixels ou plus, l'association clique-paramètre est la suivante:

$$\left[\begin{smallmatrix}**\\\end{smallmatrix},\beta_1\right],\ \left[\begin{smallmatrix}*\\*\end{smallmatrix},\beta_2\right],\ \left[\begin{smallmatrix}*&*\\\end{smallmatrix},\beta_3\right],\ \left[\begin{smallmatrix}*&\\&*\end{smallmatrix},\beta_4\right],$$

$$\left[\begin{smallmatrix}**\\*\end{smallmatrix},\beta_5\right],\ \left[\begin{smallmatrix}*\\**\end{smallmatrix},\beta_6\right],\ \left[\begin{smallmatrix}**\\&*\end{smallmatrix},\beta_7\right],\ \left[\begin{smallmatrix}*&*\\*&\end{smallmatrix},\beta_8\right],\ \left[\begin{smallmatrix}**\\**\end{smallmatrix},\beta_9\right]. \qquad (5.2.2)$$

Les potentiels associés aux cliques sont alors définis par:

$$Vc_t(x) = \begin{cases} -\beta_t & \text{si tous les } x_{ij} \text{ dans } C_t \text{ sont égaux} \\ \beta_t & \text{autrement,} \end{cases} \qquad (5.2.3)$$

pour chaque type t de clique de deux pixels ou plus, $t = 1, \ldots, 9$, et pour les cliques d'un seul pixel, on pose

$$Vc(x) = \alpha_q \text{ si } x_{ij} = k_q. \qquad (5.2.4)$$

Dans les modèles que nous utilisons dans nos exemples, tous les paramètres β sont nuls pour des cliques de plus de deux pixels, ainsi que tous les paramètres α associés aux cliques d'un pixel.

La deuxième hypothèse à considérer concerne le modèle de dégradation. Le bruit est ici encore gaussien. En supposant que l'observation à chaque point ne dépend que de ce point, et que les variables Y_{ij}, étant donné X_{ij} sont indépendantes et identiquement distribuées, la distribution conditionnelle $f(y|x)$ est donnée par:

$$f(y|x) = \prod_{i=1}^{N1} \prod_{j=1}^{N2} \frac{1}{\sqrt{2\pi\sigma^2}}\, e^{-\frac{1}{2}\frac{(y_{ij} - x_{ij})^2}{\sigma^2}}. \qquad (5.2.5)$$

La méthode que proposent Derin et Elliott cherche à <u>maximiser</u> la somme des deux termes $ln\ p(x)$ et $ln\ f(y|x)$. Or, pour une configura-

tion donnée $x=\{x_{11}, \ldots, x_{1N2}, \ldots, x_{N1N2}\}$, le logarithme de la probabilité pour le premier terme est:

$$\ln p(x) = \ln \left[\frac{1}{Z} e^{-\sum_{c \in \mathcal{C}} V_c(x)} \right]$$
$$= - \ln Z - \sum_{c \in \mathcal{C}} V_c(x), \qquad (5.2.6)$$

et pour le second:

$$\ln f(y|x) = \ln \left[\prod_{i=1}^{N1} \prod_{j=1}^{N2} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2} \frac{(y_{ij} - x_{ij})^2}{\sigma^2}} \right];$$

en sortant le terme constant $(2\pi\sigma^2)^{-\frac{1}{2}}$ du double produit, on obtient:

$$\ln f(y|x) = \ln \left[(2\pi\sigma^2)^{-\frac{1}{2}N1N2} \prod_{i=1}^{N1} \prod_{j=1}^{N2} e^{-\frac{1}{2} \frac{(y_{ij} - x_{ij})^2}{\sigma^2}} \right],$$

puis en développant le logarithme du produit des termes:

$$\ln f(y|x) = - \frac{N_1 N_2}{2} \ln(2\pi\sigma^2) - \sum_{i=1}^{N1} \sum_{j=1}^{N2} \frac{1}{2\sigma^2} (y_{ij} - x_{ij})^2. \qquad (5.2.7)$$

5.3 Formulation récursive

Il est possible de réarranger $\ln \mathcal{L}(x,y)$, c'est-à-dire la somme des équations (5.2.6) et (5.2.7), de façon récursive, en considérant l'ensemble colonne après colonne, parcourant ainsi toute la grille S.

Considérons l'évaluation des constantes comme étape initiale 0 d'un processus récursif:

$$r_0 = - \ln Z - \frac{N_1 N_2}{2} \ln(2\pi\sigma^2). \qquad (5.3.1)$$

Pour obtenir la somme des équations (5.2.6) et (5.2.7), il suffit d'ajouter à *r0* la somme:

$$- \sum_{i=1}^{N_1} \sum_{j=1}^{N_2} \frac{1}{2\sigma^2} (y_{ij} - x_{ij})^2 - \sum_{c \in \mathcal{C}} V_c(x). \qquad (5.3.2)$$

Décomposons (5.3.2) colonne par colonne et commençons par considérer la contribution de la 1ère colonne. Nous avons pour *j*=1 (en continuant le processus itératif):

$$r_1 = r_0 - \sum_{i=1}^{N_1} \frac{1}{2\sigma^2} (y_{i1} - x_{i1})^2 - \sum_{c \in \mathcal{C}^1} V_c(x) \qquad (5.3.3)$$

où $\mathcal{C}^1$ est l'ensemble des cliques formées de pixels situés uniquement dans la colonne *1*.

De la même façon, en continuant pour la colonne suivante, l'itération *j*=2 est donnée par:

$$r_2 = r_1 - \sum_{i=1}^{N_1} \frac{1}{2\sigma^2} (y_{i2} - x_{i2})^2 - \sum_{c \in \mathcal{C}^{1,2}} V_c(x) \qquad (5.3.4)$$

où cette fois $\mathcal{C}^{1,2}$ est l'ensemble des cliques formées de pixels situés dans la colonne *1* ou 2. En effet, puisque les cliques d'un système de voisinages du second ordre ne contiennent pas de pixels dans plus de 2 colonnes adjacentes, à l'étape *r2* nous avons tenu compte de toutes les cliques possibles se trouvant dans une image de *N1* lignes par *2*

colonnes. Graphiquement, les cliques de 1 ou de 2 pixels des itérations r_1 et r_2 sont représentées à la figure 5.3.1. Rappelons que les cliques doivent être distinctes.

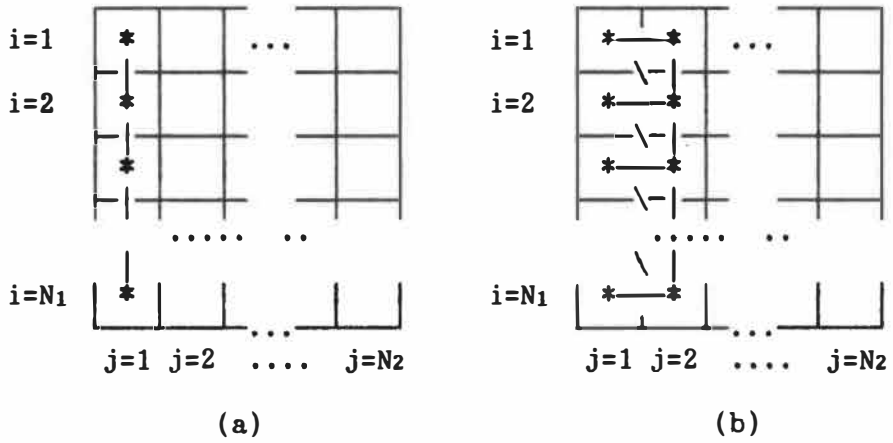


Figure 5.3.1 *Cliques de 1 (pixels notés *) et de deux pixels (liens horizontaux ——, verticaux | ou diagonaux \) pour (a)* $\mathcal{C}^1$*; (b)* $\mathcal{C}^{1,2}$.

Pour prendre en compte la contribution d'une colonne j, nous avons comme ci-dessus à considérer 2 parties. Premièrement, les potentiels des cliques dont les pixels appartiennent à la colonne j ou à la colonne précédente $j-1$, et deuxièmement le carré de la différence entre les observations y_{ij} et les valeurs estimées x_{ij} , pour toutes les rangées i, $i = 1, \ldots, N_1$. Dès lors, pour une étape j donnée, le processus de récursion ci-dessus s'écrit:

$$r_j = r_{j-1} - \sum_{i=1}^{N_1} \frac{1}{2\sigma^2} (y_{ij} - x_{ij})^2 - \sum_{c \in \mathcal{C}^{j-1,j}} V_c(x) \qquad (5.3.5)$$

où

$\mathcal{C}^{j-1,j}$ = {*C:C* est une clique contenant seulement des pixels dans la colonne *j* ou dans les colonnes *j-1* et *j*}

On vérifie facilement qu'au dernier pas de la récursion, à l'étape r_{N2}, c'est-à-dire à la dernière colonne, on a:

$$r_{N2} = \ln P(x,y), \tag{5.3.6}$$

ce qui correspond bien à l'équation (5.1.2), puisque toutes les cliques et toutes les observations ont été parcourues.

Nous venons de voir comment s'obtient la probabilité qu'une image donnée *x* soit l'image originale des observations *y*, à partir d'un modèle de dégradation gaussien et d'un champ markovien (l'information *a priori*). La restauration d'images dans le contexte établi ci-dessus consiste à choisir la configuration des x_{ij} qui maximise r_{N2}, c'est-à-dire à choisir l'image ayant la plus grande probabilité *a posteriori*.

Notons qu'il n'est pas nécessaire de chercher à effecture l'étape r_0. Celle-ci contient une constante (*Z*) particulièrement difficile à calculer ne changeant rien à la solution de la maximisation du logarithme de vraisemblance conjointe.

5.4 Algorithme

La récursion qui vient d'être développée et le principe d'optimalité permettent la formulation d'un algorithme de programmation dynamique pour trouver l'estimé $\hat{x}$ qui maximise r_{N2} rapport à x.

Cependant, plutôt que de procéder à chaque étape (à chaque colonne) en considérant tous les pixels de cette colonne, ce qui serait excessivement long, Derin et Elliott proposent de ne considérer qu'un nombre D (2 à 4) de lignes à la fois.

Le principe est le suivant. A partir du développement récursif donné ci-dessus, on cherche à maximiser r_{N2} non pas sur l'ensemble des N_1 lignes mais seulement une bande formée des D premières lignes 1 à D.

Parmi ces D lignes on ne retient que la première puis on recommence le procédé avec les lignes 2 à $D+1$. La ligne 2 est conservée et les autres ignorées.

On procède ensuite de la même manière pour toute l'image, estimant à chaque fois une bande de largeur D formée des lignes i à $i+D-1$. La ligne i est conservée et les $D-1$ autres sont reléguées aux oubliettes.

En fait, ce découpage équivaut à trouver le *MAP* sur une image de dimensions réduites de D lignes par $N2$ colonnes. Pour chaque bande estimée, la première ligne est conservée comme estimé final du *MAP* global. Puis le procédé recommence pour les D lignes suivant la ligne conservée, et ainsi de suite jusqu'aux D dernières rangées de l'image. L'estimé trouvé pour cette dernière bande est conservé intégralement pour l'image estimée finale.

Sauf pour la première bande, il faut également utiliser la ligne qui vient d'être estimée comme conditions aux frontières pour les D suivantes. Par conséquent il faut utiliser cette rangée dans le calcul de la probabilité *a posteriori* de chaque combinaison de couleurs sur les D autres lignes.

La figure 5.4.1 nous montre le processus d'estimation sur la bande i_e à i_e+D-1, pour $D=2$. La ligne i_e-1, bien qu'elle ne soit pas estimée, est nécessaire pour calculer le potentiel associé à une partie des cliques verticales ou diagonales de 2 pixels. Seule la ligne i_e est conservée, l'étape suivante estimant la bande formée des lignes i_e+1 à i_e+D.

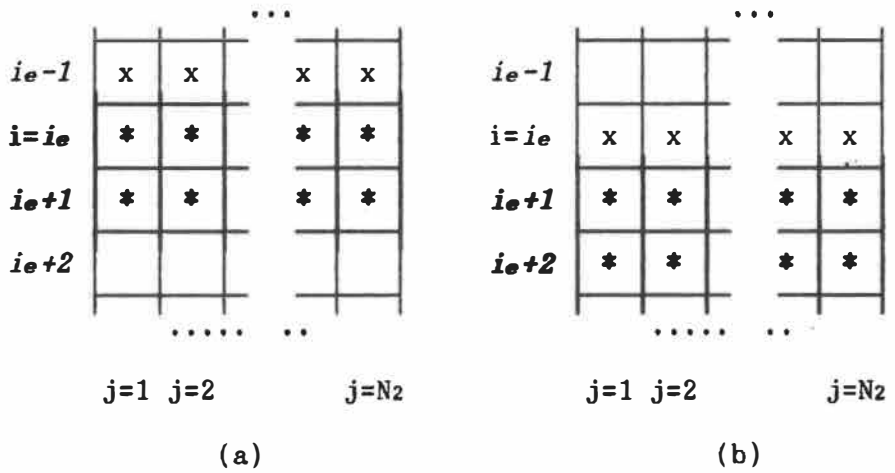


Figure 5.4.1 *Algorithme de Derin et Elliott pour des bandes de* D=*2 lignes: (a) séquence d'estimation des pixels des lignes* $i=i_e$ *à* $i=i_e+1$ *(notés *), qui utilise l'information des pixels de la ligne* i_e-1 *(notés* x*) et où seule la ligne* i_e *sera conservée; (b) séquence suivante.*

Même dans le cas simple d'une bande de 3 rangées par 100 colonnes, pour le cas binaire il y a 2^{3x100} combinaisons de couleurs possibles. En utilisant la programmation dynamique pour obtenir le MAP sur cette bande, on réduit considérablement le nombre de combinaisons à comparer.

5.5 Programmation dynamique

Pour expliquer le principe de la programmation dynamique dans le cadre du *MAP*, considérons une image de 2 rangées par 4 colonnes (figure 5.5.1) avec c=2 couleurs possibles.

rangée\colonne	1	2	3	4
1	x_{11}	x_{12}	x_{13}	x_{14}
2	x_{21}	x_{22}	x_{23}	x_{24}

Figure 5.5.1 *Image simple de 2 rangées par 4 colonnes.*

Considérons le nombre de configurations à conserver à chaque étape. A la première colonne, il y a $2^2=4$ configurations de couleurs possibles. Dès lors, il est nécessaire de calculer la valeur de r_1 pour chacune des 4 configurations à l'étape 1. Par ailleurs ces 4 valeurs doivent être conservées puisqu'il n'est pas encore possible de connaître celle, parmi les 4, qui fera partie de la configuration globale de la mince bande considérée. La configuration associée à la plus grande valeur de r_1 ne fait pas obligatoirement partie de la configuration "gagnante" qui donne la plus grande valeur de r_{N2}.

Dans la $2^{ième}$ colonne, de même, il y a 4 configurations de couleurs. Pour calculer le critère r_2 associé à chacune d'elles, nous avons besoin de la configuration de couleurs de la colonne 1, requise dans l'évaluation des potentiels des cliques de 2 pixels en diagonale ou à l'horizontale. Rappelons que r_2 utilise la valeur de r_1. Comme il y a 4 configurations de couleurs possibles à la colonne 1, il faut calculer, pour chacune des 4 configurations de couleurs à la colonne 2, les 4 valeurs correspondantes de r_2, soit 16 valeurs en tout. Toutefois il faut seulement conserver celle ayant la plus grande valeur pour chacun des choix. En effet, si nous ne savons pas encore laquelle des

4 configurations de couleurs de la colonne 2 sera la bonne, nous connaissons par contre, pour chacune d'elles, la bonne configuration pour la colonne 1: celle qui donne la plus grande valeur de r_2. Donc, même si pour la 2ième colonne il faut effectuer 16 calculs et 16 comparaisons de r_2, il n'est pas nécessaire d'en conserver plus de 4, ainsi que le choix correspondant de configuration de couleurs de la colonne 1.

A la colonne 3, il y a 4 configurations de couleurs possibles également. Cependant, il n'est pas nécessaire de calculer r_3 avec toutes les configurations de couleurs possibles aux colonnes précédentes, mais uniquement les 4 configurations de couleurs de la colonne 2. Comme pour chaque configuration les r_2 sont connues, les r_3 s'obtiennent sans problème. Ayant pris soin de conserver la meilleure configuration de couleurs à la colonne 1 pour chaque choix à la colonne 2, le procédé donne l'enchaînement de configurations maximisant le critère r_3 associé à chaque configuration de couleurs pour la colonne 3. Bien que l'on ait encore une fois 16 calculs à effectuer, on ne conserve que 4 valeurs de r_3, ainsi que les configurations de couleurs respectives des colonnes 1 et 2.

A l'étape 4, on calcule encore la valeur de r_4 pour chacune des 4 configurations de couleurs de la colonne 4, puis on effectue les 16 comparaisons. En conservant celle qui a la plus grande valeur, nous trouvons le véritable MAP sur cette mince bande. La séquence de

configurations associées à chaque colonne qui donne la plus grande valeur au critère r_4 nous donne notre estimé $\hat{x}$.

Pour l'image complète, l'approximation proposée par Derin et Elliott consiste à ne conserver que la première rangée de la bande estimée, puis à recommencer pour les D lignes suivantes, et ainsi de suite, en conservant à chaque fois uniquement la première ligne du *MAP*. Pour les autres bandes de i à $i + D - 1$, il faut également utiliser la rangée frontière $i-1$ pour les potentiels des deux cliques suivantes:

rangée $i-1$
rangée i

C'est cet algorithme que nous avons utilisé dans l'implantation de la méthode de Derin et Elliott.

5.6 Exemples de réalisations de champs markoviens

Avant de présenter les résultats de la méthode de Derin et Elliott appliquée sur des images dégradées, nous faisons une parenthèse pour illustrer les champs markoviens.

Les images que nous avons utilisées jusqu'à présent ont été construites "manuellement": à l'aide d'un programme, brièvement expliqué au chapitre 7, nous avons choisi la couleur de chaque point de l'image. Or il n'est pas du tout évident que ces images possèdent des caractéristiques markoviennes, et il pourrait être intéressant de

calculer leur probabilité de réalisation selon différents modèles afin de vérifier le degré de pertinence de l'information a priori dans les modèles bayesiens utilisés.

Nous ne savons pas si les images que nous avons présentées jusqu'à présent sont markoviennes, par contre nous pouvons regarder à quoi ressemble une image "générée" par un vrai champ markovien. Nous avons pour cela utilisé le modèle décrit par ses potentiels en (5.2.3) et (5.2.4). A mi-chemin entre le modèle simple utilisé par Besag (équation (4.3.1)) et les modèles plus complexes présentés au chapitre 3, le modèle de Derin et Elliott permet de générer un nombre assez large de réalisations aux propriétés spatiales caractérisées et variées.

Pour générer une réalisation possible d'un champ aléatoire, nous avons procédé de la façon suivante. Une image tout à fait aléatoire est générée en donnant à chaque pixel une couleur au hasard selon une loi uniforme ou gaussienne par exemple. Nous appliquons par la suite une «relaxation stochastique» sur cette image, c'est-à-dire que de façon itérative, par balayage successif de tous les points de l'image, nous générons à chaque site une nouvelle couleur selon la loi conditionnelle du champ markovien étant donné les voisins du pixel considéré. D'après Derin et Elliot, en général de 10 à 500 itérations sont nécessaires, selon le degré de complexité du modèle, le nombre de couleurs, etc.

Nous avons noté par ailleurs que le choix du balayage peut être très important à ce niveau. De ce fait, nous déconseillons tout particulièrement une mise à jour totalement synchrone (voir chapitre précédent), les changements apportés à un pixel devant servir le plus tôt possible à influencer ses voisins.

Nous donnons à la figure 5.6.1 quelques exemples à 2 ou 4 couleurs d'images générées à partir de (5.2.3)-(5.2.4).

La première image 5.6.1 (a) n'est pas la réalisation d'un champ markovien mais celle d'un champ aléatoire uniforme. Elle a servi de base aux relaxations stochastiques qui ont donné les 3 images suivantes.

L'image 5.6.1 (b) représente une réalisation avec $\beta_1=\beta_2=1.0$ et $\beta_3=\beta_4=-1.0$. Ce choix de paramètres favorise les cliques de deux pixels à l'horizontale ou à la verticale et pénalise les cliques de pixels en diagonale. Cela a pour effet d'empêcher le regroupement des pixels de même couleur en amas (un groupe contenant automatiquement des diagonales) tout en incitant la construction de lignes verticales et horizontales.

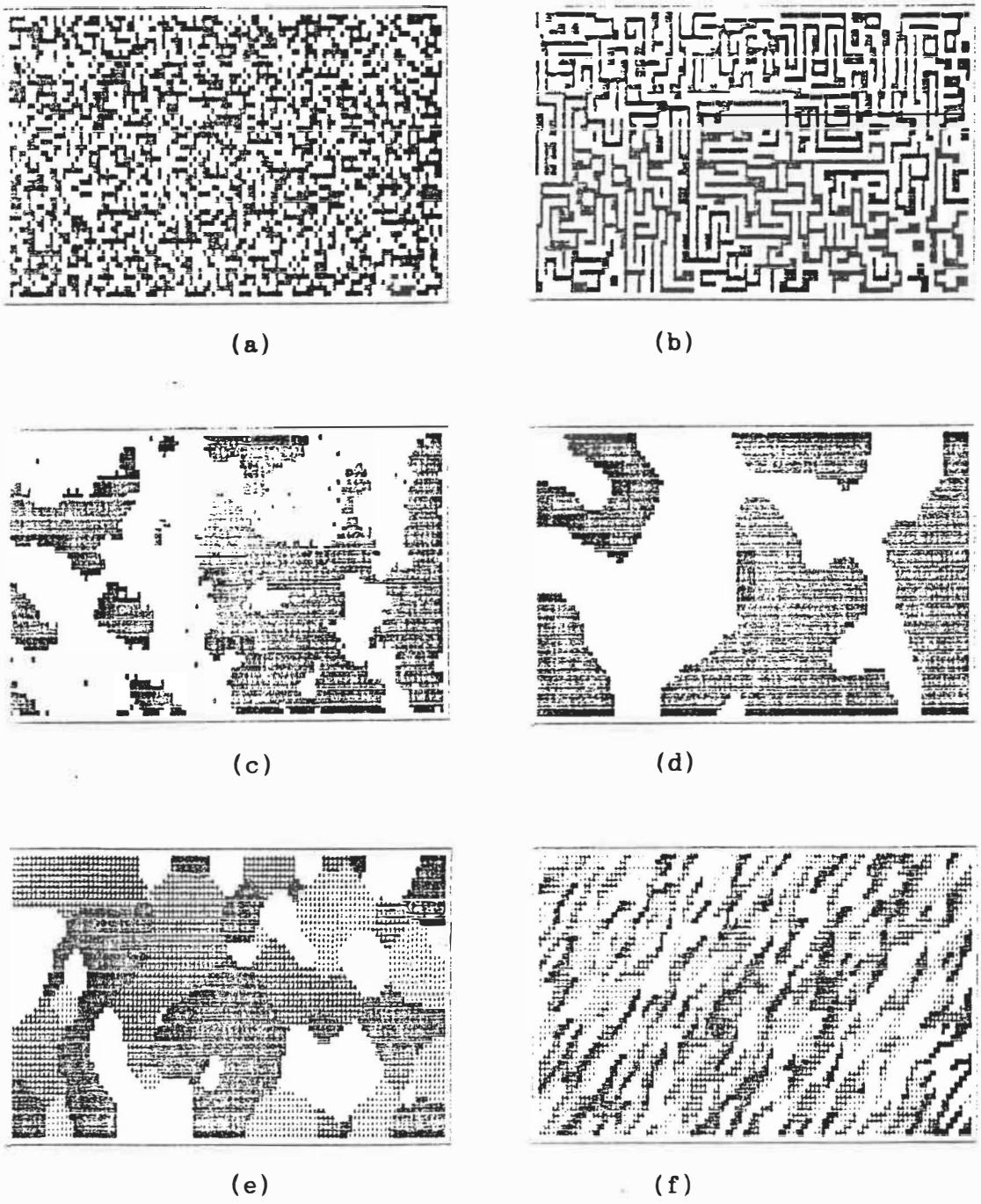


Figure 5.6.1 *Images markoviennes, modèle (5.2.3)-(5.2.4).* ***(a)*** *image aléatoire uniforme de 2 couleurs;* ***(b)*** *image générée après 5 itérations de relaxation stochastique sur (a), avec* $\beta_1=\beta_2=1.0$ *et* $\beta_3=\beta_4=-1.0$*;* ***(c)*** *image générée après 15 itérations avec* $\beta_i=0.3$ *(équivalent au modèle de l'ICM);* ***(d)*** *même que (c) avec 10 itérations et* $\beta_i=2.0$*;* ***(e)*** *image générée à partir d'une image aléatoire de 4 couleurs, après 10 itérations avec* $\beta_i=2.0$*;* ***(f)*** *même que (e) avec* $\beta_1=\beta_2=\beta_3=1.0$ *et* $\beta_4=-1.0$*.*

L'image 5.6.1 (c) représente une image générée lorsque tous les paramètres β_i sont égaux. Aucune direction n'est favorisée. Des groupes de pixels se forment dont la taille dépend de la valeur des β_i, ici 0.3. Notons que ce modèle équivaut à peu près à celui utilisé par Besag [1986] pour l'ICM (équation (4.3.1)). Cependant, la valeur de l'unique paramètre β doit être plus grande dans (4.3.1) pour obtenir le même effet.

L'image 5.6.1.(d) provient du même type de modèle sauf que dans ce cas la valeur de β (β=2.0) et les régions de couleurs uniformes sont plus grandes.

Les deux images qui suivent proviennent d'une image de départ semblable à 5.6.1 (a) mais composée de 4 couleurs. Après 10 itérations, pour des β identiques et égaux à 2.0, nous avons obtenu la figure 5.6.1 (e). Un modèle où $\beta_1=\beta_2=\beta_3=1.0$ et $\beta_4=-1.0$ défavorise la diagonale vers la gauche. Un exemple est donné à la figure 5.6.1 (f).

5.7 Exemples de restaurations

Nous donnons dans cette section quelques exemples de restaurations réalisées par l'algorithme de Derin et Elliott. Les résultats ont été obtenus en soumettant uniquement des images dégradées par du bruit gaussien additif.

Le modèle de dégradation aurait pu être modifié pour tenir compte d'un bruit multiplicatif ou d'une distorsion non-linéaire, comme nous l'avons fait au chapitre précédent. Avec plus d'efforts et de considérations pour la séparation du problème en un procédé récursif, il aurait également été possible, sans doute, de modéliser l'influence d'une fonction d'étalement ponctuelle.

La première série d'images consiste en un cas binaire où l'image originale 5.7.1 (a) a été générée par un modèle markovien avec les β_i égaux à 0.3 (il s'agit en fait de 5.6.1 (c) que nous avons légèrement nettoyée).

Le résultat d'une dégradation par bruit gaussien additif de variance σ^2=0.27 est présenté à la figure 5.7.1 (b). Nous remarquons que dans le cas binaire le taux d'erreur n'a pas besoin d'être très grand pour que l'image devienne difficile à reconnaître. Le taux d'erreur est ici de 27%.

La figure 5.7.1 (c) montre l'image après la restauration de Derin et Elliott pour laquelle nous avons procédé par bandes de D=3 lignes, et avec β_i=0.4, soit pas tout à fait l'image orginale. Le taux d'erreur observé est de 8%.

En ajustant les paramètres β à 0.3, nous avons obtenu l'image 5.7.1 (d) où l'erreur est de 7%, donc moins importante que précédemment.

L'image 5.7.1 (e) montre la restauration de 5.7.1 (b) avec des bandes de D=2 lignes. Le taux d'erreur grimpe alors à plus de 9%.

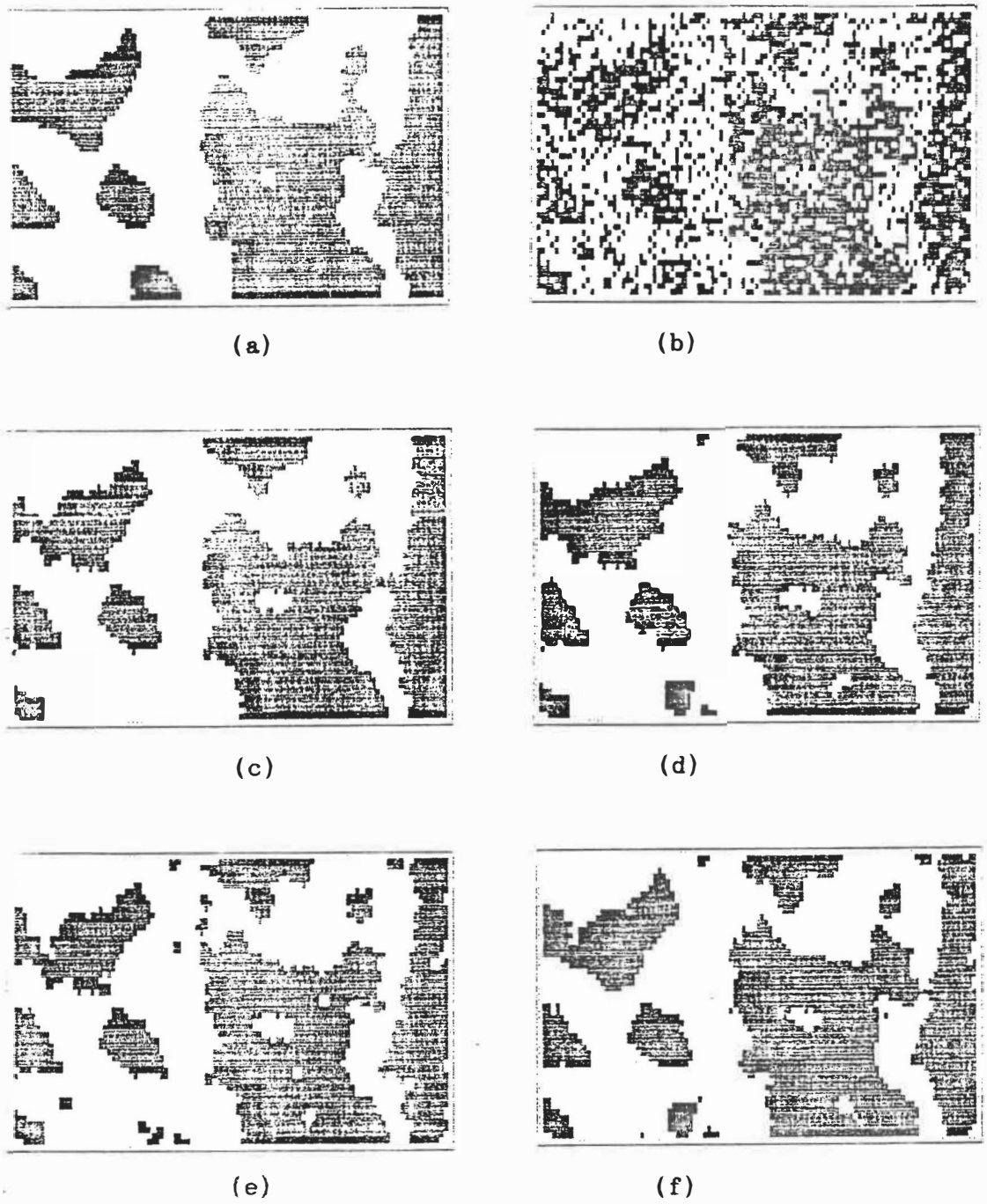


Figure 5.7.1 *Restauration d'images par programmation dynamique, 2 couleurs:* ***(a)*** *image originale «DB24A» ;* ***(b)*** *image dégradée par un bruit GA,* $\sigma^2 = 0.27$*, err=27%;* ***(c)*** *restauration programmation dynamique,* $\beta_i = 0.4$*,* $D = 3$*, err=8.01%;* ***(d)*** *restauration D&E avec* $\beta_i = 0.3$*,* $D = 3$*, err=6.97%;* ***(e)*** *restauration D&E avec* $\beta_i = 0.3$*,* $D = 2$*, err=9.31%;* ***(f)*** *restauration ICM-SYN GA,* $\beta = 1.5$*, 5 itérations, err=6.06%.*

Afin de comparer visuellement les deux méthodes, nous avons également soumis l'image dégradée à l'ICM. Le résultat de l'image 5.7.1 (f) est à peu près semblable à celui de 5.7.1 (e), donnant un taux d'erreur de 6%. Nous notons cependant que le paramètre β utilisé est de 1.5, alors qu'en appliquant l'ICM avec β=0.3, nous avons obtenu un taux d'erreur de 25%. Ce résultat semble confirmer que les champs markoviens sont équivalents mais que le modèle de Besag donne un poids plus important à l'unique β, puisque le modèle de Derin et Elliott possède 4 fonctions pour 4 paramètres.

Le deuxième exemple utilise la même image originale 5.7.2 (a) que celle de l'exemple précédent. La dégradation causée par un bruit gaussien additif avec σ^2=0.37 est plus importante, donnant une erreur de 37.2% à la figure 5.7.2 (b).

Deux restaurations sont montrées. La première, à la figure 5.7.2 (c) est celle de la programmation dynamique de Derin et Elliott avec β_i=0.3 et D=3 pour un taux d'erreur de 18%. L'image 5.7.3 (d) est le résultat de la restauration par l'ICM avec β=1.5 et 6 itérations donnant un taux d'erreur final de 17%. Bien que les deux images soient différentes l'une de l'autre, il est difficile d'avoir une préférence pour l'une ou pour l'autre: aucune des deux méthodes n'a donné de très bons résultats.

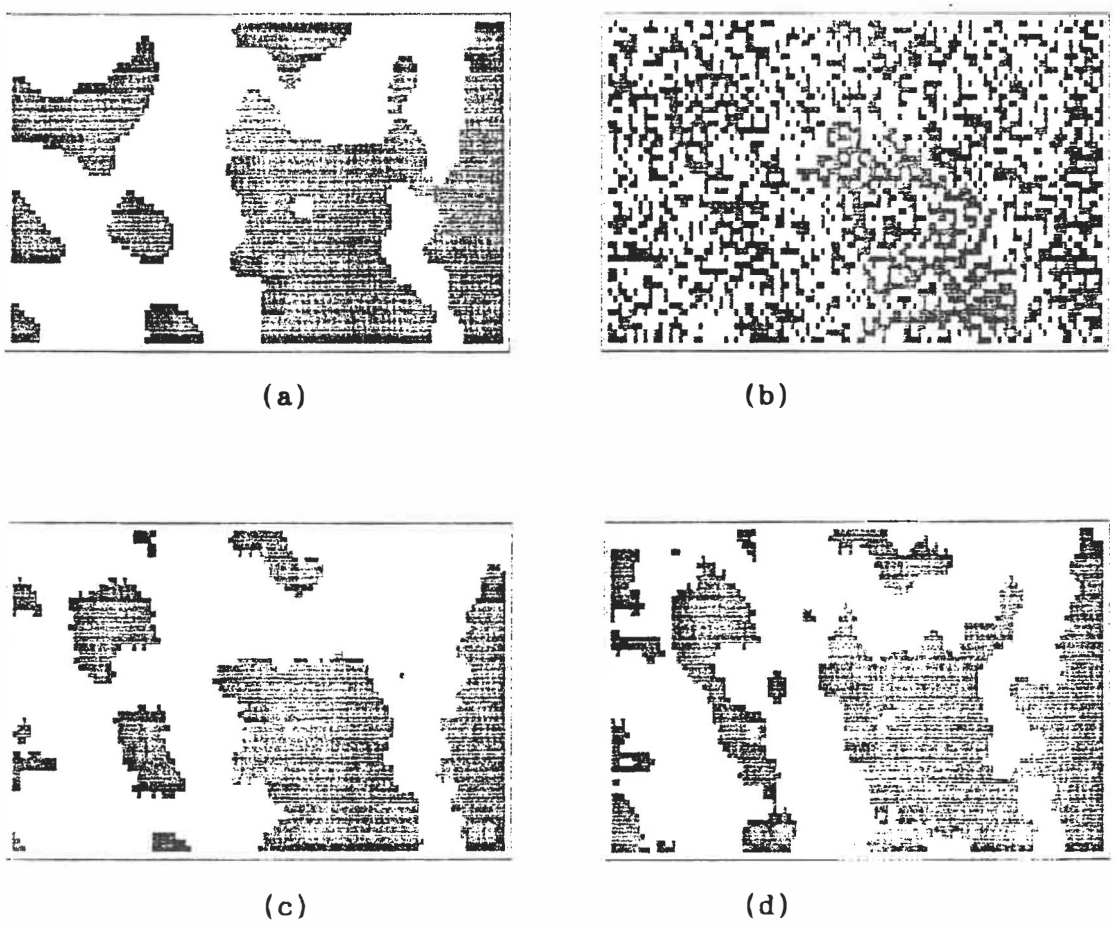


Figure 5.7.2 *Restauration d'images par D&E et ICM avec forte dégradation:* ***(a)*** *Image originale «DB24A»;* ***(b)*** *dégradation* ***GA***, $\sigma^2=0.37$, *donnant un taux d'erreur de 37.2%;* ***(c)*** *restauration programmation dynamique* $\beta_i=0.3$, $D=3$, *err=18.03%;* ***(d)*** *restauration ICM-SYN avec modèle GA,* $\beta=1.5$ *constant, 6 itérations, err=16.58%.*

Toutes les images que nous avons utilisées jusqu'à présent pour illustrer la méthode de restauration par programmation dynamique de Derin et Elliott sont binaires: elles n'ont que 2 couleurs. Nous faisons ici une petite parenthèse, avant de présenter un exemple de restauration sur une image de 4 couleurs, pour parler du temps requis pour appliquer la méthode.

Sur notre équipement (voir la discussion à la section «A propos de la vitesse» au chapitre précédent), la méthode de Derin et Elliott prend environ 25 minutes pour D=3, contrairement à l'ICM qui ne requiert que 2 minutes de temps d'ordinateur pour ces images binaires. En passant à des bandes de D=2 lignes la programmation dynamique tombe à 8 minutes mais, comme nous l'avons montré, les résultats sont moins bons dans ce cas.

Avec l'ICM, passer de 2 à 4 couleurs signifie effectuer 2 fois plus de tests et cela prend à peu près 2 fois plus de temps. Avec Derin et Elliot, on passe de 3^2=9 à 3^4=81 configurations de couleurs par colonne de D=3 lignes. Et effectivement, nos exemples de 4 couleurs ont pris près de 9 heures de traitement!

On comprendra, en regard de ce facteur de vitesse et des conclusions tirées de l'exemple qui suit, que nous ayons peu d'images de plus de 2 couleurs.

La figure 5.7.3 (a) représente l'image originale, identique à celle montrée à la figure 5.6.1 (e). Après une dégradation causée par un bruit gaussien additif de variance σ^2=0.55, nous avons obtenu l'image 5.7.3 (b) où le taux d'erreur est presque de 40%.

La restauration par l'ICM est montrée à la figure 5.7.3 (c). Nous avons effectué un balayage synchrone (comme pour toutes les restaura-

tions de ce chapitre) et 11 itérations, β choisi croissant de 0.5 à 1.5 par incrément de 0.2 pendant 5 itérations puis constant à 1.5 pendant 6 autres. Le taux d'erreur est de moins de 10% pour une image d'excellente qualité où presque toutes les régions ont été reconstruites (avec β constant à 1.5, le taux d'erreur après 6 itérations était de 11%).

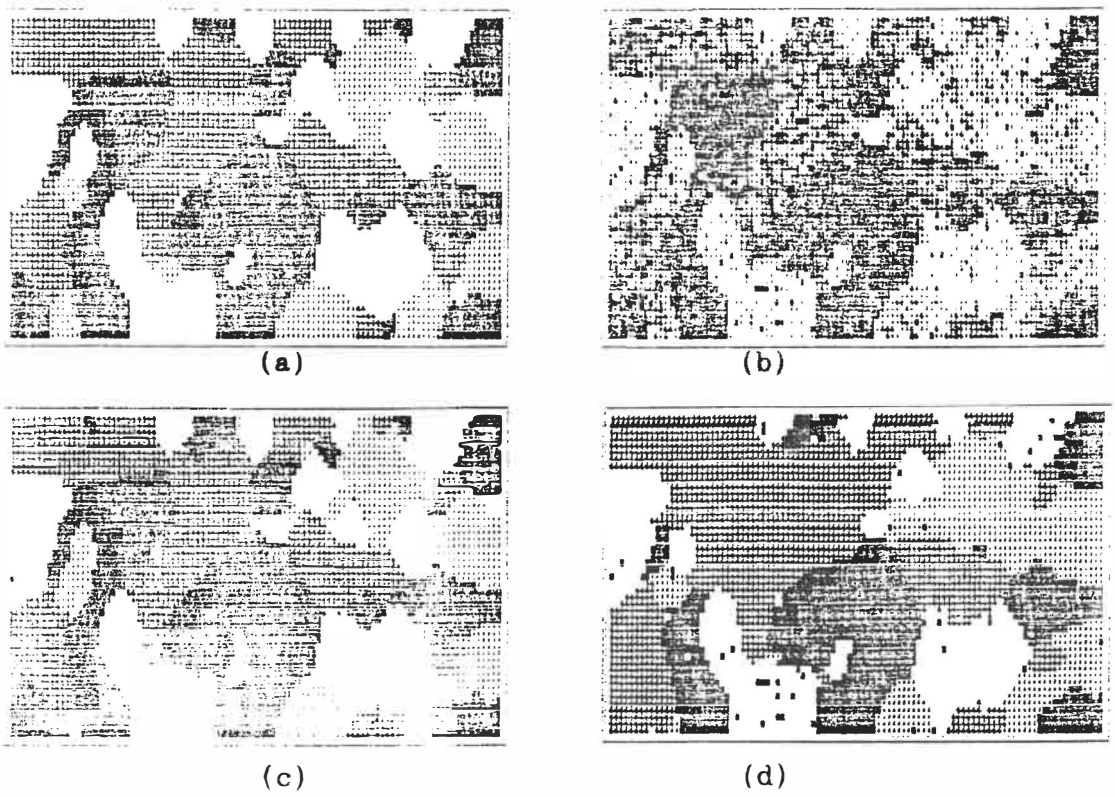


Figure 5.7.3 *Restauration d'images par D&E et ICM, 4 couleurs:* ***(a)*** *Image originale 4 couleurs «D29»;* ***(b)*** *dégradation* ***GA***, $\sigma^2=0.55$, *donnant un taux d'erreur de 38.29%;* ***(c)*** *restauration ICM-SYN avec modèle GA,* $\beta=0.5$ *augmenté par incrément de 0.2 pendant 6 itérations plus* $\beta=1.5$ constant pendant 5 it., err=9.69%; **(d)** restauration programmation dynamique $\beta_i=0.35$, D=*3, err=21.32%.*

L'image 5.7.3 (d) montre le meilleur résultat après quelques essais de restauration avec la méthode de Derin et Elliott. Le taux d'erreur est de plus de 20%, avec de grandes zones englouties dans l'estimé final.

Remarquons que les paramètres β_i, i=1,..,4 sont égaux à 0.35 et non à 2.0 comme ceux utilisés pour générer l'image. En spécifiant 2.0 dans le modèle au moment de la restauration, nous avons obtenu des images méconnaissables. Une valeur autour de 0.3 semble être à peu près standard pour ce modèle où les β_i sont égaux, comme β=1.5 semble l'être pour l'ICM, et pour un bruit additif gaussien.

Par conséquent, dans le cas binaire, les images estimées par l'ICM de Besag et la méthode de Derin et Elliott sont à peu près équivalentes, avec un léger avantage à l'ICM pour ce qui à trait au taux d'erreur. Cependant, le facteur de vitesse, ainsi que les erreurs rencontrées pour des restaurations d'images contenant plus de 2 couleurs, nous portent à considérer l'ICM comme une méthode plus avantageuse et plus souple.

CHAPITRE 6: CRITERES DE COMPARAISON

Des deux méthodes de restauration d'images que nous venons d'expliquer, l'ICM de Besag, certainement la plus rapide, semble donner de meilleurs résulats. Cependant, comment juger de la qualité de l'estimation? L'implantation de l'algorithme de restauration nous fournit une approximation de l'image ayant la plus grande probabilité étant donné les observations, mais que vaut-elle?

Il s'agit là d'un problème peu étudié mais de grande importance si l'on veut s'attaquer à la comparaison des images et à l'analyse des techniques de restauration.

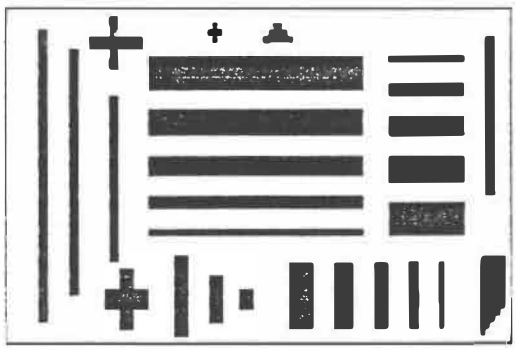
Le taux d'erreur que nous avons considéré jusqu'à présent, qui mesure la proportion de pixels mal assignés, n'est sans doute pas le seul indice de ressemblance entre deux images. Un observateur humain est parfois capable de donner une évaluation visuelle de la qualité de l'image restaurée en se posant des questions subjectives: ressemble-t-elle à l'image idéale? Sommes-nous capable de l'identifier et/ou d'en reconnaître les régions importantes?

Nous avons cherché dans ce chapitre à développer de nouveaux critères évaluant le degré de ressemblance de deux images. Comme il est

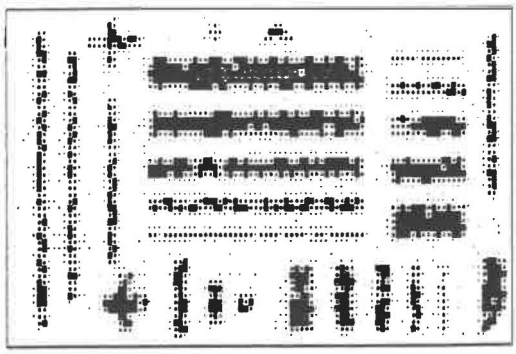
souhaitable d'obtenir une valeur numérique, nous proposons quelques techniques quantifiant la mesure de la différence entre images.

Dans un but d'étude, à la fois des techniques de restauration d'images et des mesures d'erreur, nous avons appliqué ces nouveaux taux d'erreur sur quelques images tests. Cependant, cela ne saurait être interprété comme une comparaison approfondie des méthodes de Besag et de Derin et Elliott. Nous proposons plutôt de considérer les suggestions données ci-dessous comme un premier pas dans l'analyse de la mesure d'erreur entre images, problème complexe parallèle à la restauration d'images par ordinateur.

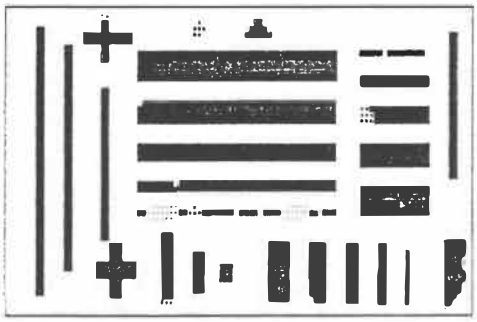
Le premier groupe d'images où des mesures d'erreur ont été prises comprend une image idéale «TT» à la figure 6.1 (a), déjà utilisée aux figures 4.6.6 et 4.6.7. L'image dégradée associée 6.1 (b) a été obtenue en appliquant une fonction d'étalement ponctuelle (FEP) et un bruit gaussien additif sur 6.1 (a).



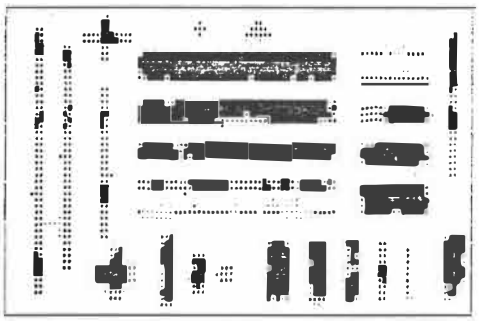
(a)



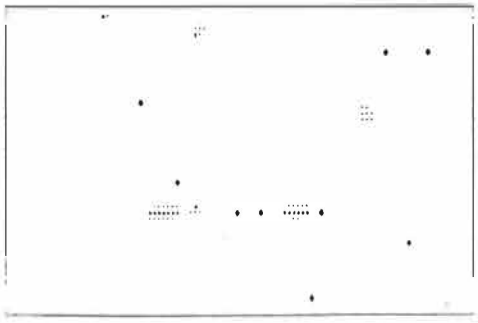
(b)



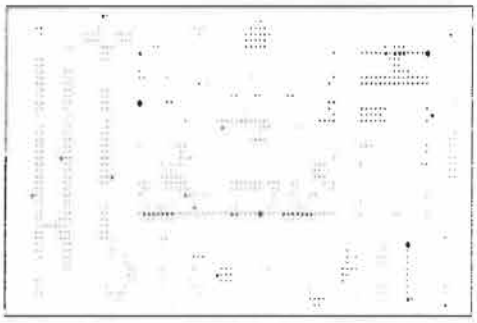
(c)



(d)



(e)



(f)

Figure 6.1 *Etude des taux d'erreur, premier groupe:* ***(a)*** *Image originale «TT»;* ***(b)*** *dégradation* ***FEP+GA*** σ^2*=0.1, err=31.02%;* ***(c)*** *restauration ICM-SSYN2 GA-FEP, 6 itérations,* ß*=1.5 constant, err=1.49%;* ***(d)*** *restauration ICM-SYN GA, 6 itérations,* ß*=1.5 constant, err=14.52%;* ***(e)*** *plan d'erreur de* (c)*;* ***(f)*** *plan d'erreur de* (d).

Deux restaurations ont été étudiées: la première, à la figure 6.1 (c), a été obtenue après avoir soumis les observations 6.1 (b) à l'ICM en tenant compte de la FEP et en effectuant un balayage semi-synchrone; la seconde pour sa part, à la figure 6.1 (d), a été obtenue sans modèle de FEP et avec une mise à jour simultanée des pixels.

Le deuxième groupe est formé autour de l'image originale «DB24A» à la figure 6.2 (a), que nous avons déjà utilisée aux figures 5.7.1 et 5.7.2. L'image 6.2 (b) représente le résultat de l'application d'un bruit additif gaussien. Deux restaurations sont étudiées: l'image 6.2 (c) est l'image estimée par l'ICM et l'image 6.2 (d) est obtenue par la programmation dynamique.

Les valeurs des différents taux étudiés obtenues sur ces images sont données au tableau 6.1. Les images considérées sont données aux figures suivantes. On trouvera également des images associées que nous expliquons dans les sections subséquentes.

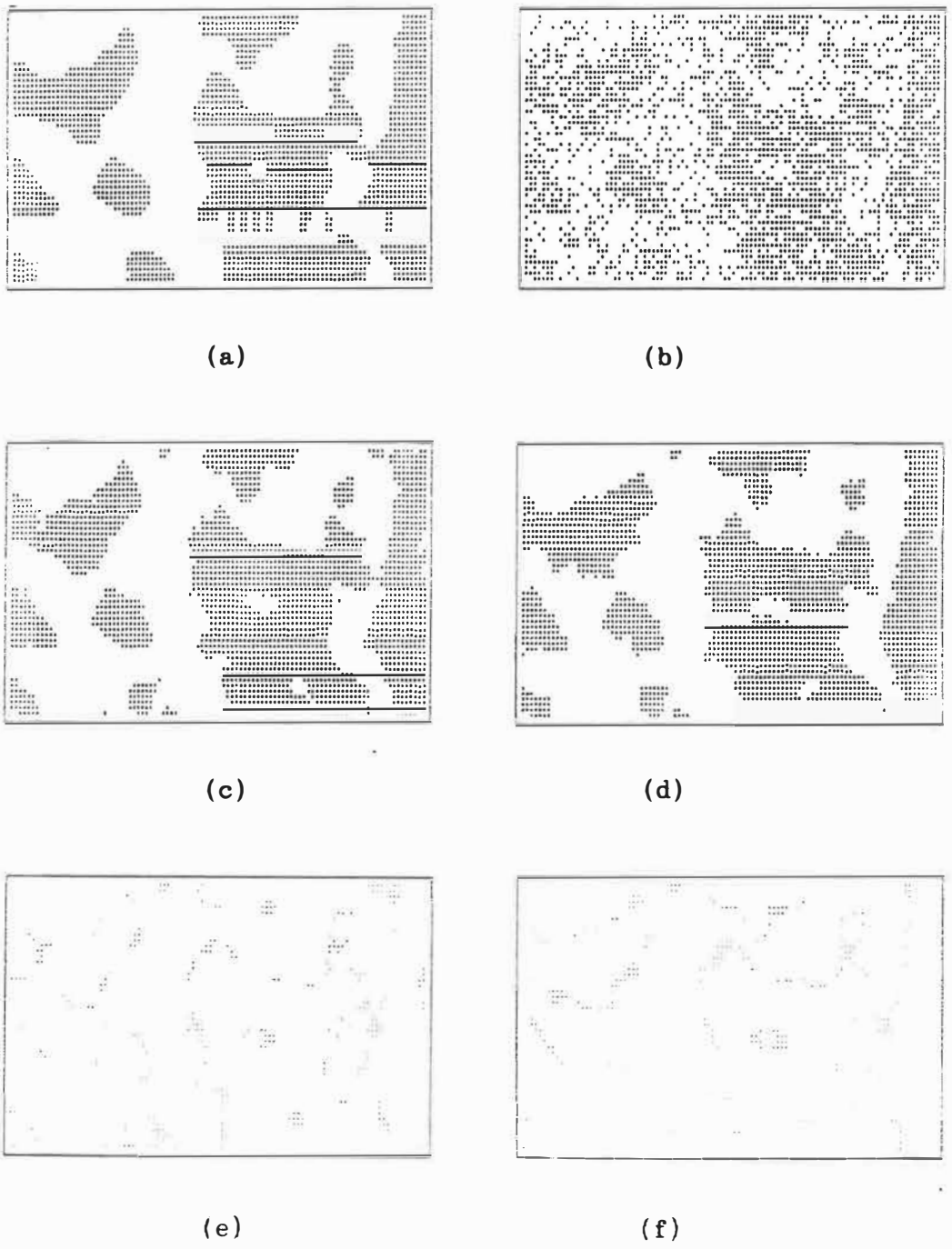


Figure 6.2 *Etude des taux d'erreur, second groupe:* ***(a)*** *Image originale «DB24A»;* ***(b)*** *dégradation* ***GA*** σ^2*=0.27, err=27%;* ***(c)*** *restauration ICM-SYN GA, 5 itérations,* β*=1.5 constant, err=6.06%;* ***(d)*** *restauration D&E,* β_i*=0.3,* D*=3, err=6.97%;* ***(e)*** *plan d'erreur de* (c)*;* ***(f)*** *plan d'erreur de* (d).

Image	τ_1	τ_2	τ_3	τ_4	τ_5	τ_6
6.1 (d)	14.52%	44.70%	27%	13%	20.28%	5.45%
6.1 (c)	1.49%	5.87%	2.8%	1.7%	2.26%	0.87%
6.1 (b)	31.02%	87.12%	42%	38%	40.74%	10.35%
6.2 (c)	6.06%	25%	7.3%	6.7%	7.02%	1.75%
6.2 (d)	6.97%	28.79%	8.2%	7.5%	7.89%	1.97%
6.2 (b)	27.10%	93.94%	27.4 %	27.3 %	27.40%	6.85%

Tableau 6.1 *Comparaisons entre différentes mesures d'erreur sur quelques images; τ_1: pourcentage de pixels mal assignés; τ_2: pourcentage du nombre de blocs 3x3 incorrects; τ_3: pourcentage du nombre d'erreurs avec poids du Laplacien intérieur; τ_4: pourcentage du nombre d'erreurs avec poids du Laplacien extérieur; τ_5: moyenne des pourcentages du nombre d'erreurs avec poids du Laplacien intérieur et extérieur; τ_6: moyenne des valeurs normalisées des erreurs avec poids du Laplacien intérieur et extérieur.*

6.1 Pourcentage de pixels mal assignés

La première méthode que nous étudions dans ce chapitre consiste à compter le nombre de pixels qui ont reçu la mauvaise couleur à la fin de la restauration. Comme la plupart des calculs d'erreur dont nous parlerons, il faut, pour l'appliquer, connaître l'image restaurée et l'image originale.

Afin d'obtenir un taux relatif que l'on puisse utiliser pour des images de dimensions différentes, le nombre de pixels mal reconstruits

est divisé par le nombre total de pixels dans l'image. Multipliée par 100, cette mesure nous donne le pourcentage de pixels incorrects dans l'image estimée par rapport à l'image souhaitée.

Comme première mesure d'erreur nous avons donc:

$$\tau_1 = \left[\sum_{i=1}^{n} \theta_1(x_i, \hat{x}_i) \; / \; n \right] * 100 \qquad (6.1.1)$$

où:

$$\theta_1(x_i, \hat{x}_i) = \begin{cases} 0 \text{ si } x_i = \hat{x}_i \\ 1 \text{ autrement} \end{cases}$$

L'avantage de cette méthode, outre sa simplicité, est de fournir un nombre relatif qui soit significatif. Même sans comparaison avec des images estimées par d'autres méthodes de restauration, la valeur obtenue représente un indice clair et facile à comprendre de la qualité de l'image reconstruite.

Toutefois deux défauts sont à souligner. Premièrement cette méthode ne fournit que le nombre (ou le pourcentage) de pixels incorrects, ce qui est peut-être insuffisant, par exemple dans le cas d'images où les couleurs sont ordonnées par des niveaux de gris. On voudra alors chercher à connaître la valeur de ces différences, ou tout au moins un paramètre quantifiant l'écart entre les deux images.

Deuxièmement - et il s'agit de son plus sérieux handicap - τ_1 ne donne aucune information qualitative sur l'image estimée. La même importance est accordée à chaque pixel, sans souci de la qualité

visuelle de l'image, de ses formes générales, de ses contours ou de sa géométrie.

A l'appui du critère ci-dessus, il faut remarquer au tableau 6.1 que tous les taux proposés respectent le choix de la "meilleure" restauration telle que décidé par τ_1. Dès lors aucune des mesures qui suivent ne permettent de croire que l'autre restauration représente un meilleur estimé de l'image originale.

Par ailleurs il faut reconnaître que c'est ce taux τ_1 que nous avons utilisé jusqu'à présent pour décrire nos images. Il s'agit en fait de l'indice le plus usuel, sans doute à défaut de solutions de remplacement adéquates.

6.2 Image des différences ou «plan d'erreur»

On retrouve souvent, associée au taux d'erreur ci-dessus, une image représentant la différence entre l'image restaurée et l'image idéale. Bien que l'on n'y retrouve pas de valeur, elle donne une idée de l'erreur de la restauration aussi bien qualitative par la répartition des erreurs, que quantitative par la valeur de l'erreur (une couleur plus ou moins foncée indique une erreur plus ou moins grande).

Cette image, que l'on appelle également «plan d'erreur» (en particulier dans le cas binaire), est fournie pour chaque restauration considérée aux figures 6.1 (e), 6.1 (f), 6.2 (e) et 6.2 (f).

Notons par x^d l'ensemble des pixels de cette image. Ceux-ci sont obtenus par:

$$x_i^d = |\hat{x}_i - x_i| \qquad (6.2.1)$$

6.3 Blocs de pixels

Le taux d'erreur τ_1 ne donne qu'une indication de la moyenne point par point des erreurs et il peut arriver que l'on reconnaisse facilement une image malgré les erreurs si celles-ci sont réparties hors des zones importantes, les contours par exemple. De même, il est possible que les erreurs ne soient pas réparties de la même façon à travers l'image.

Une façon de tenir compte plus globalement des erreurs consiste à changer d'échelle. Il s'agit de considérer des blocs de pixels et d'en mesurer le nombre parfaitement reconstruits à la fin de la restauration. On compte une erreur dès qu'au moins un pixel d'un bloc x_B donné ne correspond pas à son vis-à-vis dans l'autre image. Nous avons choisi dans nos exemples des blocs de 3x3 pixels mutuellement exclusifs (un bloc ne contient les pixels d'aucun autre bloc).

En normalisant cette mesure par le nombre de blocs trouvés dans la grille n_b, et en multipliant par 100, nous obtenons:

$$\tau_2 = \left[\sum_{i=1}^{nb} \theta_2(x_{Bi}) \; / \; n_b \right] * 100 \qquad (6.3.1)$$

où:

$$\theta_2(x_{Bi}) = \begin{cases} 1 \text{ si au moins un pixel de } x_{Bi} \textit{ est incorrect} \\ 0 \text{ autrement} \end{cases}$$

Remarquons que les valeurs trouvées pour nos images dans le tableau 6.1, bien que grimpant rapidement, semblent donner une bonne idée du degré de "dégâts" dans les images.

6.4 Pourcentage avec Laplacien

Les méthodes ci-dessus donnent la même importance à tous les pixels de l'image sans distinction. Or un critère de la qualité de la restauration pourrait être la capacité de la méthode à reconstruire les contours des régions de l'image (les frontières séparant deux zones de couleurs différentes).

Il existe en effet dans le domaine du traitement et de l'analyse d'images une importante littérature sur le problème de la détection de bordures, le contour étant à la base de la plupart des techniques de reconnaissance de formes. Cela nous permet de croire que de donner un poids plus important aux sites situés sur les contours, dans le calcul du taux d'erreur, devrait fournir une meilleur mesure de l'erreur.

L'outil que nous avons utilisé ici est le Laplacien, noté ∇^2. Sans entrer dans les détails de la recherche de contours et de la détection d'arêtes, disons que ce dernier fournit une indication sur la différence entre la couleur d'un point et celle des points autour de lui. Le Laplacien mesure l'amplitude du changement d'intensité lumineuse aux points de l'image.

Reprenons brièvement la notation mathématique du chapitre 2 pour décrire cet opérateur. Le Laplacien d'une image bi-dimensionnelle $f(x,y)$ est obtenu par:

$$\nabla^2 f(x,y) = \frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \qquad (6.4.1)$$

Dans le cas discret, une approximation de la dérivée seconde au site (i,j) du centre de la figure 6.4.1 est donnée par:

$$\frac{\partial^2 f}{\partial x^2} \cong (f_{i-1,j} - 2f_{i,j} + f_{i+1,j}),$$

$$\frac{\partial^2 f}{\partial y^2} \cong (f_{i,j-1} - 2f_{i,j} + f_{i,j+1}),$$

i-1,j-1	i,j-1	i+1,j-1
i-1,j	i,j	i+1,j
i-1,j+1	i,j+1	i+1,j+1

Figure 6.4.1 *Notation des pixels autour d'un site central* (i,j) *pour une approximation discrète du Laplacien* ∇^2.

ce qui nous donne:

$$\frac{\partial^2 f}{\partial x^2} + \frac{\partial^2 f}{\partial y^2} \cong (f_{i-1,j} + f_{i,j-1} + f_{i+1,j} + f_{i,j+1}) - 4\, f_{i,j}.$$

Cette valeur est obtenue en utilisant un patron que l'on applique sur le site *(i,j)* et dont les poids sont donnés à la figure 6.4.2.

	1	
1	-4	1
	1	

Figure 6.4.2 *Patron des poids d'une approximation discrète du Laplacien* ∇^2.

Sur une grille rectangulaire, il est difficile d'obtenir un patron qui fournit une bonne approximation au Laplacien et qui est symétrique. La matrice ci dessus par exemple oublie les pixels des coins. Un choix populaire [Horn:1986] qui donne un estimé très précis du Laplacien est:

$$[\nabla^2] = 1/6 \begin{array}{|c|c|c|}\hline 1 & 4 & 1 \\ \hline 4 & -20 & 4 \\ \hline 1 & 4 & 1 \\ \hline \end{array} \qquad (6.4.2)$$

C'est cette matrice que nous utilisons dans les exemples qui suivent. L'indice + indique le signe du facteur 1/6.

La convolution de la matrice du Laplacien $[\nabla^2]$ sur l'image originale donne des valeurs élevées aux points situés à la "frontière" de deux régions de couleurs différentes, et faibles ou nulles aux points situés dans des zones d'intensité (de couleurs) uniforme.

Considérons deux points de la figure 6.4.3, où les chiffres représentent la valeur de la couleur à chaque point. Lorsqu'on applique le patron (6.4.2) sur le site indiqué par ***4***, on obtient une valeur de -3. En appliquant $[\nabla^2]$ sur le point 1^1, on trouve +3, et sur 1^2 on obtient la valeur 0.

4	4	1	1	1
4	**4**	$\mathbf{1^1}$	$\mathbf{1^2}$	1
4	4	1	1	1

Figure 6.4.3 *Grille de couleurs pour un exemple de calcul de* $[\nabla^2]$.

On remarque donc que d'appliquer $[\nabla^2]$ sur toute l'image donne une valeur nulle à tous les points à l'intérieur d'une zone de couleurs.

uniforme, tandis qu'à la frontière entre deux régions de couleurs différentes, on trouve une valeur plus grande ou plus faible selon que le site ait une couleur, à l'inverse, faible ou élevée.

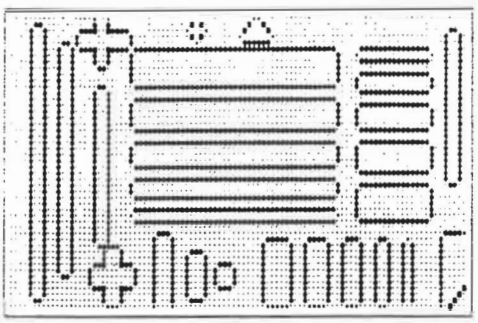
En considérant les valeurs obtenues comme des couleurs, nous obtenons des images où seules les arêtes des figures considérées sont visibles[4]. Par exemple, l'image 6.4.4 (a) représente l'application de $[\nabla^2]$ sur «TT» donnée à la figure 6.1 (a).

En prenant le négatif de la matrice (6.4.2) nous obtenons les mêmes résultats mais inversés: une valeur élevée est obtenue pour un site frontière de couleur élevée. Nous notons cette nouvelle matrice $[\nabla^2]$ pour indiquer que l'on prend le négatif du Laplacien habituel. L'image 6.4.4 (b) nous montre ce que donne l'application de $[\nabla^2]$ sur l'image 6.1 (a).

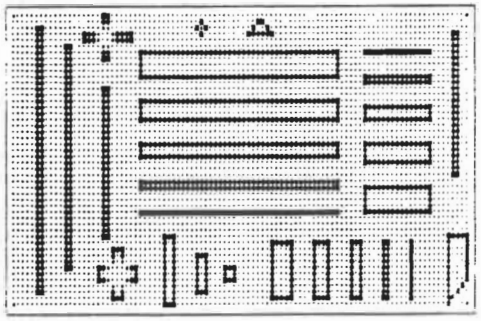
Nous donnons également deux autres exemples aux figures 6.4.4 (d) et 6.4.4 (f) pour des images originales de natures différentes, les figures 6.4.4 (c) et 6.4.4 (e), afin d'illustrer la détection d'arêtes par l'opérateur du Laplacien.

---

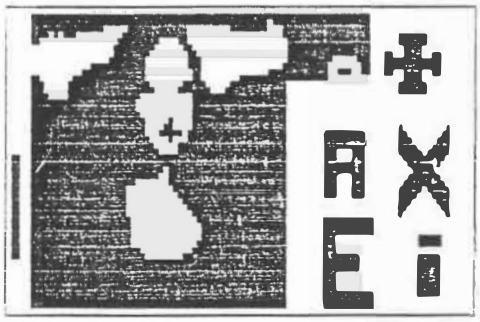
[4]Nous avons réajusté les valeurs obtenues du Laplacien en assignant aux valeurs négatives ou nulles la couleur 1.



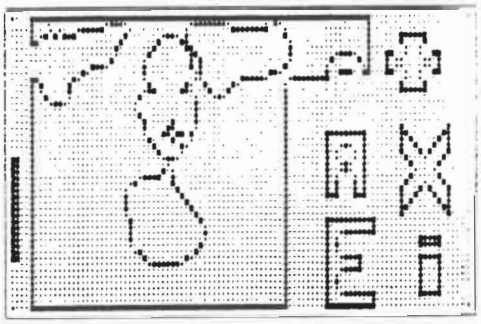
(a)



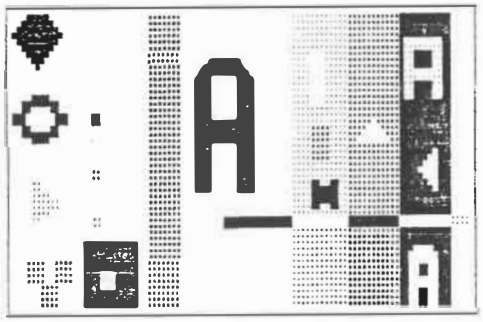
(b)



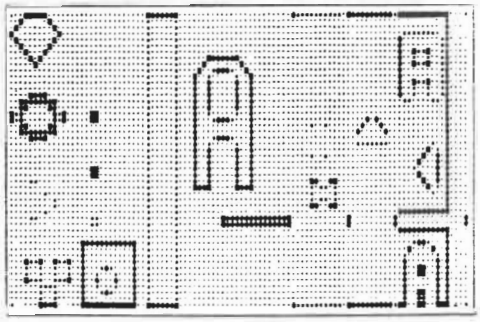
(c)



(d)



(e)



(f)

Figure 6.4.4 *Détection d'arêtes: illustration de l'application de l'opérateur de Laplacien* ∇^2. ***(a)*** *application de* $[\nabla^2]$ *sur «TT»;* ***(b)*** *application de* $[\nabla^2]$ *sur «TT»;* ***(c)*** *image originale «MK»;* ***(d)*** *application de* $[\nabla^2]$ *sur «MK»;* ***(e)*** *image originale «TA»;* ***(f)*** *application de* $[\nabla^2]$ *sur «TA».*

En considérant les couleurs de ces images comme des "poids" sur les pixels formant les contours des images, la méthode que nous proposons consiste en deux étapes.

Premièrement créer une image x^- formée par la convolution de $[\nabla^2]$ sur l'image idéale x, afin d'obtenir une grille des poids, puis appliquer le même critère qu'en (6.1.1) en donnant les poids trouvés à chaque pixel - c'est-à-dire obtenir la moyenne pondérée des pixels mal assignés par des poids donnés par le Laplacien.

Dès lors, une erreur à un point ne vaut plus 1 mais le poids correspondant dans l'image de contours. En divisant par la somme de tous les poids de l'image et en multipliant par 100, nous obtenons le critère suivant:

$$\tau_3 = \left[\sum_{i=1}^{n} \theta_3(x_i, \hat{x}_i)\ x_i^- \ \Big/ \sum_{i=1}^{n} x_i^- \right] * 100 \qquad (6.4.3)$$

où comme pour θ_1, la fonction θ_3 est donnée par:

$$\theta_3(x_i, \hat{x}_i) = \begin{cases} 0 & \text{si } x_i = \hat{x}_i \\ 1 & \text{autrement} \end{cases}$$

Les valeurs obtenues pour nos images tests semblent correspondre assez bien aux critères subjectifs et intuitifs.

Par conséquent, la méthode ci-dessus favorise les points frontières ayant la plus grande couleur. Pour inverser cette situation,

nous utilisons $[\nabla^2]$ et les taux d'erreur pour le même critère sont donnés, dans ce cas, sous la colonne τ_4 dans le tableau 6.1.

Nous avons pensé utile de concilier les deux taux ci-dessus et de faire la moyenne des valeurs obtenues pour chaque Laplacien. C'est la valeur donnée par τ_5.

D'autre part, la colonne τ_6 représente le taux τ_5 normalisé, en le divisant par l'écart entre les couleurs ayant la plus grande et la plus petite valeur.

6.5 Autres suggestions

Les critères proposés ci-dessus sont intéressants mais aucun ne couvre, à lui seul, tous les aspects de la comparaison d'images et de la mesure d'erreur. Sans les avoir implantés, nous décrivons dans les paragraphes qui suivent quelques critères supplémentaires parfois plus complets, mais souvent plus difficiles à mettre en pratique.

Mentionnons tout d'abord une mesure souvent rencontrée, dont nous n'avons pas parlé, le rapport de l'énergie du signal (l'image orginale) sur l'énergie du bruit, le "signal-to-noise ratio" en anglais.

Un second critère du même type est l'entropie. Nous avons brièvement mentionné au chapitre 2 que certaines techniques de restauration d'images reposent sur une maximisation de l'entropie. Il

pourrait être intéressant de calculer cette valeur pour les images que nous avons obtenues. Une définition heuristique de l'entropie, pour une image x, est donnée par:

$$x^t \ln x.$$

Plus proche des discussions que nous avons eues jusqu'à présent, puisque les 2 méthodes de restauration que nous avons étudiées donnent une image maximisant une certaine probabilité *a posteriori*, un critère intéressant consisterait à calculer la probabilité finale $p(\hat{x}|y)$.

Par contre il est difficile d'obtenir la constante de normalisation Z dans le modèle de la distribution de Gibbs donné à l'équation (3.4.2), mais pour comparer deux méthodes le calcul de cette dernière n'est sans doute pas nécessaire.

La meilleure restauration serait alors celle qui a la plus grande probabilité, ce qui était recherché au départ, et par conséquent permettrait une vraie comparaison.

Si l'on veut développer les critères de ce chapitre, nous présentons quelques idées supplémentaires, complexifiant en général la mesure d'erreur. Il serait intéressant de:

- utiliser un Laplacien "seuillé", c'est-à-dire qu'il n'y a qu'un poids pour les pixels dont le Laplacien dépasse un certain seuil; ceci aurait l'avantage de donner la même importance à toutes les

arêtes, quelque soit la valeur des couleurs des régions frontières;

- combiner le Laplacien et la méthode des blocs: on pourrait assigner un poids à chaque bloc afin d'obtenir une moyenne pondérée; le poids d'un bloc serait alors fonction du nombre de pixels de contour qu'il contient.

Finalement, le critère ultime de comparaison, puisque nous cherchons avant tout à reconnaître l'image originale, serait d'appliquer des techniques de reconnaissance de formes sur les restaurations à comparer: l'image ayant fourni le plus grand nombre de succès serait alors considérée comme étant la meilleure.

CHAPITRE 7: PROGRAMMES ET ALGORITHMES

Ce dernier chapitre décrit d'une manière plus technique l'implantation que nous avons faite des méthodes étudiées et les modifications que nous avons apportées pour étudier des dégradations plus complexes.

Nous avons écrit plusieurs programmes que nous fournissons sur disquettes et dont les fichiers sources imprimés sont donnés en annexe. Pour chacun nous donnerons les descriptions du rôle du programme, du mode d'utilisation et des algorithmes développés.

L'analyse et les explications les plus complètes sont fournies pour le programme «besag1.exe». Ce programme, qui effectue les restaurations par l'ICM, recouvre un grand nombre d'aspects sur les images, l'interface usager, la structure interne des fonctions et des modules. Pour éviter d'alourdir l'exposé des autres programmes, nous ferons régulièrement référence à celui-ci.

Nous terminons par un plan d'ensemble des fichiers disponibles ainsi que sur quelques considérations techniques concernant le mode graphique, la présence d'un coprocesseur mathématique et la recompilation des programmes.

7.1 Ordinateur et langage

Les programmes expliqués sont les suivants:

Besag1.exe Implantation de la méthode de Besag

Derin1.exe Implantation de la méthode de Derin et Elliott

Les programmes fonctionnent sur des ordinateurs compatibles IBM et ont été écrits en Turbo-Pascal version 4.0. Nous supposerons connues les procédures permettant de mettre en marche un ordinateur de ce type, les notions de programmes informatiques et de fichiers, ainsi que les quelques commandes simples du système d'exploitation DOS nécessaires à la copie de fichiers, au changement de lecteur de disquette ou à l'exécution d'un programme.

Les programmes peuvent être utilisés directement sur la disquette fournie avec ce mémoire (format double densité/double côté 360Ko) ou préalablement copiés sur disque dur.

Nous recommandons de faire une copie de la disquette contenant les programmes exécutables et d'utiliser uniquement cette copie comme disquette de travail, l'originale servant de double de sécurité.

7.2 Programme «besag1.exe»

7.2.1 Description

Ce programme permet d'obtenir la restauration d'une image par la méthode de l'ICM de Besag. Après avoir lu une image (l'image dégradée) du disque dur ou d'une disquette, l'usager spécifie certains paramètres (nombre d'itérations, choix de β, type de balayage, de dégradations ...) et obtient l'image estimée par l'ICM à partir d'un modèle *a priori* de champ markovien (pour le moment seul l'exemple de Besag est implanté).

Il est également possible de comparer l'image restaurée avec une image de référence qui peut également être lue du disque, et de bruiter une image à l'aide de différents modèles de dégradations à des fins de simulations.

Chaque image peut être affichée à l'écran en spécifiant un code de couleur si l'ordinateur est équipé d'une carte graphique, être imprimée ou sauvée sur disque.

7.2.2 Mise en oeuvre et utilisation

Sur la ligne de commande du DOS (après avoir rendu le lecteur **A** actif par exemple):

A:\>_

tapez les lettres:

BESAG1

puis appuyez sur la touche **RETURN** (ou **ENTER**).

L'information suivante apparaîtra alors à l'écran:

Programme de gestion de l'implantation de l'ICM

0: Fin
1: Lire une image du disque
2: Ecrire l'image sur disque
3: Affiche l'image graphiquement
4: Ecrit les valeurs d'une image
5: Bruite l'image
6: Taux d'erreur
7: Itération de l'ICM
11: Ajustement des paramètres
33: Imprime une image
44: Imprime les valeurs d'une image

Votre choix ? [4]:

Choisissez alors une des options en écrivant le numéro correspondant puis en appuyant sur la touche **RETURN.**

Notez que le numéro apparaissant entre crochets, par exemple **[4]**, représente la valeur par défaut suggérée par le programme. En appuyant immédiatement sur **ENTER** vous acceptez la valeur proposée.

Les options **1, 2, 3, 4, 33** et **44** sont d'utilisation simple et ne nécessitent pas d'explications détaillées. Un sous-menu apparaîtra pour demander plus de détails sur l'image sur laquelle s'effectuera l'opération. Lors de l'affichage graphique, appuyez sur la touche **ENTER** ou **RETURN** pour revenir au menu.

Il est important cependant de mentionner qu'il existe en tout temps deux (2) images plus une (1) matrice (une «image») de points réels. Ce sont: les images originale - également appelée image de référence, image non-dégradée ou image «OK» - et en restauration- également appelée image dégradée, ou image «ICM» -, formées de pixels (ayant des valeurs de 1 à 16 dans notre cas), et une matrice - également appelée image «Y» - représentant la valeur réelle à chaque pixel donnant l'image dégradée.

Cette dernière est normalement obtenue en appliquant une dégradation permettant de bruiter l'image originale (option **5**). Elle ne peut pas être affichée graphiquement à l'écran, mais ses valeurs peuvent toutefois être imprimées (option **44**) ou listées (option **4**).

Notez que l'image dégradée «ICM» est la seule image qui est modifiée par l'ICM. En effet, à chaque itération de l'ICM, les nouveaux pixels estimés sont mis en mémoire dans cette image et c'est elle qui est affichée graphiquement au cours du traitement. Cette image est également celle qui est utilisée, avec l'image originale «OK» pour calculer le taux d'erreur.

Certaines actions de lectures (option **1**) et de bruitage (option **5**) affectent plusieurs images:

- la lecture de l'image originale «OK» efface les deux images «Y» et «ICM» pour les remplir par les valeurs de l'image «OK»;

- la lecture d'une image en restauration «ICM» a également pour effet de remplacer les valeurs de la matrice réelle «Y» par les valeurs lues;
- inversement, la lecture d'une matrice réelle «Y» efface l'image dégradée «OK» en remplaçant chaque pixel par l'entier le plus proche de la valeur réelle correspondante dans l'image «Y»;
- l'option **5** permettant de bruiter l'image originale, ce qui crée l'image «Y» et réinitialise l'image «OK» comme ci-dessus.

L'option **5** permet de choisir parmi plusieurs types de dégradations - bruit aléatoire additif uniforme ou gaussien, multiplicatif gaussien, dégradation par racine carrée ou par fonction d'étalement - qui peuvent être appliquées l'une après l'autre dans un ordre quelconque. Pour bruiter une nouvelle fois l'image sans que l'effet ne soit cumulatif, il est nécessaire de relire l'image orginale «OK», ce qui a pour effet de "nettoyer" les images «ICM» et «Y». Mentionnons que pour l'ICM il faut absolument que l'image ait subi une dégradation aléatoire par bruit additif ou multiplicatif.

L'option **6** fournit le taux d'erreur (pourcentage de pixels mal classés) et calcule la variance de l'erreur.

Avant de parler de la restauration, nous donnons une description des paramètres et de leur importance pour l'ICM. En choisissant l'option **11**, les lignes suivantes apparaissent les unes après les autres. Vous pouvez alors appuyez sur **ENTER** pour accepter la valeur

proposée (qui représente une valeur la première fois ou la dernière réponse entrée), ou bien donner une nouvelle valeur qui sera mémorisée:

ß ? [1.50]:
La valeur de *beta* spécifiant le champ markovien.

ßinc ? [0.00]:
La valeur de l'incrément de *beta* à chaque itération.

K(σ^2) ? [0.700]:
La valeur de la variance de l'erreur. Cette variance est calculée à l'option **6** et de façon automatique lors du bruitage (option **5**).

Itération graphique (o/n) ? [TRUE]:
Répondez par 'o' (oui) si vous voulez voir graphiquement la progression de la restauration d'étape en étape, ou par 'n' (non) pour une restauration légèrement plus rapide **sans** affichage graphique. Pendant l'ICM, appuyez sur la combinaison des deux touches «ALT-G» pour passer du mode d'affichage graphique au mode texte et vice-versa.

Son (o/n) ? [TRUE]:
Répondez par 'o' (oui) si vous voulez des traits sonores ponctuant chaque étape de la restauration, par 'n' (non) dans le cas contraire.

Debug (o/n) ? [FALSE]:
Répondez par 'o' (oui) si vous voulez voir affichés certains calculs intermédiaires à chaque étape de la restauration, par 'n' (non) dans le cas contraire. Pendant l'ICM, appuyez sur la combinaison des deux touches «ALT-D» pour passer du mode d'affichage des calculs intermédiaires au mode normal et vice-versa.

ICM full (toutes les couleurs) (o/n) ? [TRUE]:
Répondez par 'o' (oui) si vous voulez que la maximisation de l'ICM se fasse sur toutes les couleurs pour chaque pixel, ou par 'n' (non) pour que le critère ne s'effectue que sur les couleurs faisant partie du voisinage du pixel considéré en plus de la couleur même du pixel. Ce dernier choix devrait permettre d'accélérer la restauration mais n'est pas tout à fait identique à l'ICM.

Nombre de couleurs ? [4]:
Nombre de couleurs dans l'image. Il est important que ce nombre concorde avec le nombre réel de couleurs dans les images lues. Un maximum de 16 couleurs (1 à 16) sont possibles, la couleur 0 étant réservée pour représenter les bordures.

Change les couleurs (o/n) ? [FALSE]:

Permet de changer la grille graphique (nombre de points) représentant chaque couleur lors de l'affichage ou de l'impression. Les couleurs par défaut vont du blanc (aucun point) pour le '0' au noir (16 points) pour la couleur '16'. En répondant 'o' (oui) vous avez la possibilité de changer cet ordre, par exemple en donnant le 'caractère graphique' 0 à la couleur 1 et le 'caractère graphique' 16 à la couleur 2 dans un cas binaire.

L'option **7** permet d'effectuer la restauration par l'ICM. Certaines questions sont posées au préalable (comme d'habitude vous pouvez appuyez sur la touche **RETURN** ou **ENTER** pour accepter la valeur par défaut proposée entre crochets **[]**):

TYPE DE BRUIT
0: Des détails SVP
1: Bruit additif Gaussien
2: Bruit multiplicatif Gaussien
Votre choix ? [1]:

Il faut ici choisir le type de bruit qui aurait corrompu l'image originale. L'option 0 donne des détails sur le pourquoi de cette question et l'effet de la réponse dans le modèle utilisé.

MINIMISATION/MAXIMISATION
0: Des détails SVP
1: ICM Normal (Gaussienne et nombre de voisins)
2: ICM + H (Gaussienne sur moyenne pondérée et nombre de voisins)
Votre choix ? [1]:

L'option **1** correspond à l'ICM standard de Besag. Choisissez l'option **2** lorque vous avez des raisons de croire qu'un bruit détermiste pouvant être modélisé par une fonction d'étalement aurait dégradé l'image. Dans ce cas on vous demandera de donner les valeurs de la matrice h de dimensions 3 x 3 dont la convolution avec l'image originale aurait donné l'image dégradée:

Choix de la matrice:
Les valeurs représentent la position des indices de la matrice h.

(-1,-1) (-1, 0) (-1, 1)
(0,-1) (0, 0) (0, 1)
(1,-1) (1, 0) (1, 1)
h[-1,-1] ? [0.06]: répondez à chaque question
h[-1,0] ? [0.06]:
h[-1,1] ? [0.06]:
h[0,-1] ? [0.06]:

h[0,0] ? [0.50]:
h[0,1] ? [0.06]:
h[1,-1] ? [0.06]:
h[1,0] ? [0.06]:
h[1,1] ? [0.06]:

Φ: Distorsion non linéaire (racine carrée) (o/n) ? [FALSE]:
Répondez 'o' (oui) si vous souhaitez incorporer une dégradation de type racine carrée dans le modèle, par 'n' dans le cas contraire.

TYPE D'ITERATIONS
0: Des détails SVP
1: SYNC
2: RASTER
3: SEMI2
Votre choix ? [1]:
Choisissez le balayage qui sera effectué en donnant la valeur correspondante.

Nombre d'itérations ? [6]:
Choisissez le nombre d'itérations qui seront effectuées par l'ICM.

Fichier de Sortie (NIL pour aucun) ? [EE.RES]:
Vous pouvez donnez ici le nom d'un fichier qui contiendra les résultats intermédiaires à chaque itération (taux d'erreur, valeurs de β).

Pendant le traitement de l'ICM, plusieurs options sont possibles. En appuyant sur la combinaison de deux touches «ALT-G» vous passez du mode de l'affichage graphique au mode texte (ou vice-versa si vous êtes en mode texte) et en appuyant sur «ALT-D» vous pouvez voir écrire ou non les calculs intermédiaires. Ceci vous permet en fait de modifier la réponse aux questions **«Itération graphique»** et **«Debug»** de l'option **11** à n'importe quel moment du traitement.

Un message sonore avertit de la fin du traitement. Appuyez sur **ENTER** pour passer à la suite: le calcul automatique du taux d'erreur.

En résumé, les étapes à suivre pour une analyse de l'ICM sont les suivantes:

- lire une image originale, option **1**;
- ajuster les paramètres, option **11**, afin de spécifier le nombre de couleurs dans l'image, ainsi que la variance du bruit qui dégradera l'image;
- bruiter l'image originale, option **5**;
- si on le souhaite pour pouvoir répéter exactement la même expérience, sauver l'image réelle, option **2**;
- spécifier les paramètres d'opération, option **11**, y compris la variance qui a été recalculée;
- effectuer l'ICM en spécifiant la complexité du modèle de dégradation et le nombre d'itérations, option **7**.

7.2.3 Algorithmes et structure interne

Le programme besag1 est construit autour de trois grands axes: le programme principal et deux modules.

Premièrement le programme principal («besag1.pas»), regroupant la déclaration et l'initialisation des variables, la boucle principale affichant les menus et posant des questions à l'utilisateur ou l'utilisatrice, s'occupe également des appels aux fonctions regroupées dans deux modules.

Ces deux modules forment les deux autres grandes parties. Construits selon le format des «units» de Turbo Pascal 4.0, les fichiers «imag.pas» et «icm.pas» contiennent la plupart des fonctions de traitement d'images. Tandis que le module «ICM» est spécifique à la restauration par la méthode de l'ICM de Besag (explications, paramètres, itérations, calculs), «IMAG» contient des fonctions et procédures utilisées par plusieurs programmes (lecture, écriture, affichage, impression, édition, bruitage, etc.). C'est également dans ce dernier module qu'une personne désirant connaître le format des images sur disques, la structure des images en mémoires, les couleurs, etc., trouvera tous les renseignements nécessaires et toutes les déclarations de "types" et de "constantes" relatives aux images.

Les fonctions et procédures les plus importantes que nous expliquons sont toutes contenues dans le module «ICM». Premièrement, la charpente du programme, la fonction «ITER» qui effectue les itérations, peut se réduire, au plus simple, à l'algorithme suivant:

- *répéter pour chaque itération:*
  - *répéter pour chaque rangée* y *de l'image «ICM»:*
    - *répéter pour chaque colonne* x*:*
      - *remplacer la couleur à la position* (x,y) *par celle qui minimise le critère (4.3.4), (4.4.10) ou (4.4.14), selon la complexité du modèle de dégradation souhaitée.*

. *fin.*

La fonction est bien sûr plus compliquée, pour les raisons suivantes:

- il est nécessaire d'ouvrir le fichier de résultats s'il a été requis et, dans ce cas, d'effectuer les calculs de taux d'erreur à chaque itération;

- trois types de balayages étant disponibles, il est nécessaire d'effectuer les tests sur les indices de colonnes et de rangées, pour les cas «RASTER» (**ASYN**) et «SEMI2» (**SSYN2**), ou de conserver les résultats intermédiaires de deux lignes , pour le cas «SYNC» (**SYN**);

- un test supplémentaire doit être fait selon le type de minimisation, c'est-à-dire avec ou sans fonction d'étalement;

- le balayage ne s'effectue pas sur toute l'image, mais commence au second et se termine à l'avant-dernier pixel de chaque rangée et de chaque colonne; ceci permet de traiter les cas frontières simplement en y mettant une valeur de 0 qui est neutre (ne peut pas être une couleur dans l'image);

- une gestion du clavier est nécessaire pour tester si l'utilisateur a appuyé sur une combinaison «ALT-G» ou «ALT-D»;

On obtient donc en tenant compte de tout ceci l'algorithme complet de la fonction «ITER» (nous donnons en caractère **gras** certaines variables pour faciliter la consultation et la compréhension du programme en Turbo Pascal):

. *calculer une fois le facteur* 1 / (2 σ^2) *afin d'accélérer les calculs du critère de minimisation;*

. *si requis, ouvrir le fichier de résultats pour l'écriture et y écrire le type d'itération, la valeur de* β *et de* σ^2*, le taux d'erreur initial, ainsi que la matrice* h *de la fonction d'étalement si nécessaire;*

. *afficher l'image de départ si en mode graphique;*

. *obtenir le temps de départ pour calcul du temps de traitement;*

- *tant que l'itération en cours* **L** *est inférieur au nombre d'itérations souhaité* **nb_iter** *et que l'utilisateur n'a pas terminé le traitement, répéter:*

| . *si requis, écrire au fichier l'indice de l'itération et la valeur de ß;*

| . *obtenir la valeur de départ de l'indice de rangée* ***i*** *et d'incrément de rangée* ***i_inc*** *selon le type de balayage;*

| - *tant que l'indice de rangée est inférieur au nombre total de rangées* ***DIMIY*** *dans l'image et que l'utilisateur n'a pas terminé le traitement, répéter:*

| | . *obtenir la valeur de départ de l'indice de colonne* ***j*** *et d'incrément de colonne* ***j_inc;***

| | - *tant que l'indice de colonne est inférieur au nombre total de colonnes* ***DIMIX*** *dans l'image et que l'utilisateur n'a pas terminé le traitement, répéter:*

| | | . *trouver la nouvelle valeur au point* ***j, i;***

| | | . *si le balayage est de type "synchrone", conserver la valeur dans un tampon pour cette ligne;*

| | | . *dans le cas contraire (cas "asynchrone" et "semi-synchrone"), remplacer le pixel dans l'image «ICM» par la nouvelle valeur et montrer le pixel, si en mode graphique;*

| | | . *incrémenter l'indice de colonne* ***j***

| | | . *vérifier l'état du clavier et indiquer que tout est terminé si l'utilisateur appuie sur la touche* ***ENTER*** *ou* ***RETURN;***

| | . *si le balayage est de type "synchrone" et que deux lignes au moins ont été traitées (rangées numérotées 2 et 3), recopier la rangée précédant celle qui vient d'être traitée dans l'image «ICM» et l'afficher si en mode graphique, puis copier le tampon de la rangée courante dans le tampon de la ligne précédente;*

| | . *incrémenter l'indice de rangée* ***i;***

| | . *si requis, émettre un trait sonore pour indiquer que le traitement d'une ligne (rangée) est terminé;*

| . *si le balayage est de type "synchrone", recopier la dernière ligne tampon dans l'image «ICM»;*

| . *incrémenter l'indice d'itération* ***L*** *et la valeur de ß;*

| . *si requis, calculer le taux d'erreur et le sauver dans le fichier de résultats;*

. *obtenir l'heure de fin du traitement;*

. *si requis, émettre un message sonore annonçant la fin du traitement;*

. *attendre que l'utilisateur appuie sur la touche* ***RETURN*** *ou* ***ENTER****;*

. *remettre l'écran en mode texte s'il était en mode graphique;*

. *afficher le nombre d'itérations terminées, calculer le temps de traitement, et si requis l'écrire au fichier de résultats;*

. *fin.*

Cette fonction utilise l'une ou l'autre des deux fonctions «minICM» ou «minICMV» qui calculent la nouvelle valeur à mettre à chaque pixel à la position **x,y**. La différence entre les deux est que «minICMV» tient compte de la matrice *h* de la fonction d'étalement ponctuelle.

L'algorithme de la fonction «minICM» est:

. *mettre dans un tableau indexé sur la couleur* ***icm_futab*** *le nombre de pixels de chaque couleur membre du voisinage et, si requis (DEBUG), afficher les valeurs trouvées;*

. *si requis (DEBUG), afficher la grille des 9 pixels centrée sur le pixel considéré à la position* ***x,y****;*

- *pour chaque valeur de* ***c*** *allant de 1 au nombre de couleurs* ***nb_coul*** *possibles dans l'image, répéter:*

| . *obtenir le nombre de voisins* ***nu*** *ayant cette couleur grâce au tableau indexé* ***icm_futab****;*

- *si la couleur considérée* **c** *doit être considérée, c'est-à-dire soit si toutes les couleurs sont admissibles (voir* **icm full** *de l'option* **11***), soit, dans le cas contraire, si le nombre de voisins de cette couleur est non nul, ou bien si cette couleur* **c** *est celle du pixel central, alors:*

  - *calculer la valeur du critère* **valf** *en tenant compte du cas additif ou multiplicatif, et de la présence ou non d'une distorsion non-linéaire (racine carrée);*

  - *vérifier si la valeur calculée est la plus petite pour le moment et, si oui, mettre à jour les variables permettant de conserver cette valeur* **minf** *et la couleur pour laquelle elle fut obtenue* **cOK**, *puis, si requis (DEBUG), afficher une étoile (*);*

  - *si requis (DEBUG), afficher la couleur et la valeur calculée;*

- *retourner comme valeur de la fonction la couleur pour laquelle le plus petit critère fut obtenu* **cOK**.

Pour «minIMV» la fonction est semblable sauf qu'il est nécessaire de calculer l'effet des autres pixels sur le critère.

7.3 Programme «derin1.exe»

7.3.1 Description

Ce programme permet de tester la méthode de Derin et Elliott en effectuant la restauration par la programmation dynamique. Certaines options disponibles dans le programme précédent ne le sont pas ici, et quelques autres sont différentes ou nouvelles.

7.3.2 Utilisation

Le principe de mise en oeuvre est le même que pour le programme «besag1» et n'est pas expliqué ici. Le menu suivant apparaît à l'écran lors de l'exécution du programme.

Programme de gestion de l'implantation de la segmentation de Derin & Elliott

0: Fin
1: Lire une image
2: Ecrire l'image
3: Affiche l'image
5: Bruite l'image
6: Taux d'erreur
7: Itération RESTAURATION program. dynamique
9: Génèrer une image buffer
11: Ajustement des paramètres
12: Courbe des exponentielles (2 couleurs)
13: Histogramme de la réalisation (2 couleurs)
20: Imprime une image

Votre choix ? [11]:

Il est nécessaire d'expliquer les différentes images que l'on retrouve dans ce programme et qui ne sont pas les mêmes que dans le programme précédent. Par exemple, en choisissant l'option **1**, on obtient le sous-menu suivant:

LECTURE
1: Image originale
2: Image SEG
3: Image Rest

Les images proposées correspondent respectivement à l'image originale, bien sûr, l'image bruitée, et l'image restaurée. La lecture

de l'image originale a pour effet de remplacer l'image bruitée par l'image originale. Les options **5** et **9** créent l'image «SEG», tandis que l'option **7** crée l'image «REST». Notons que les trois images sont du même type, et que contrairement à l'ICM, il n'existe pas de matrice à valeurs réelles.

L'option **9**, qui génère une image à partir du modèle de champ markovien de Derin et Elliott, propose plusieurs types de balayage, correspondant respectivement aux balayages «synchrone» (avec buffer), «asynchrone» (sans buffer) et «semi-synchrone» (de Sylvain). Généralement il est préférable de prendre le balayage semi-synchrone pour ce type d'itérations afin d'obtenir l'image la plus proche d'une véritable réalisation markovienne. Appuyez sur **g** pour passer du mode graphique au mode texte, sur **s** pour voir défiler d'étape en étape, sur **c** pour revenir en déroulement continu.

Les paramètres de l'option **11** sont les suivants:

Nombre de couleurs ? [2]:
Nombre de couleurs dans l'image. Voir la discussion à la section précédente (option **11** de l'ICM).

Change les couleurs (o/n) ? [FALSE]:
'o' (oui) pour modifier le code de couleurs, 'n' (non) dans le cas contraire.

ß1 ? [1.00]:
Rentrez la valeur du paramètre *ß1*, puis des 3 autres de la même façon.
ß2 ? [1.00]:
ß3 ? [1.00]:
ß4 ? [1.00]:

Variance K ? [0.50]:
Comme pour l'ICM, cette méthode de restauration requiert la connaissance de la variance σ^2.

Graphique (o/n) ? [TRUE]:
Répondez par 'o' (oui) pour avoir un affichage graphique du traitement, par 'n' (non) dans le cas contraire.

Random (o/n) ? [TRUE]:
Répondez par 'o' (oui) pour que la couleur choisie lorsque l'on génère une image le soit selon une loi de probabilité fixée par le modèle markovien, ou par 'n' (non) dans le cas contraire, c'est-à-dire si vous souhaitez que la nouvelle couleur soit simplement celle qu maximise cette distribution. Pour une véritable relaxation stochastique, répondez oui.

SOUND (o/n) ? [TRUE]:
Répondez par 'o' (oui) pour avoir un message sonore à chaque fin d'étape, par 'n' (non) dans le cas contraire.

Les options **12** et **13** ne sont là que pour faciliter la compréhension lors d'une étude approfondie de la méthode de Derin et Elliott, et offrent respectivement d'afficher l'allure de la distribution de probabilité de couleur selon le nombre de voisins dans un cas de 2 couleurs, et d'afficher un histogramme pouvant servir à l'estimation des paramètres. Dans ce dernier cas, pendant le temps de calcul, appuyez sur **s** si vous désirez voir les calculs intermédiaires.

Dans les deux cas, après un temps court ou long, un cadre apparaît à l'écran, qui définit la zone dans laquelle sera dessiné le graphique. Utilisez les touches **PgUp** ou **PgDn** pour déplacer les lignes supérieure ou inférieure, **Home** ou **End** pour déplacer les lignes gauche ou droite, à chaque fois vers l'extérieur. Appuyez sur **Ins** pour passer au sens contraire, c'est-à-dire pour déplacer les lignes vers l'intérieur. Les

flèches vous permettent de déplacer la boîte entière, et les chiffres **1** à **9** donnent l'incrément lors des déplacements. Appuyez sur **ENTER** pour accepter les dimensions. Le graphique sera produit. Appuyez sur **p** si vous désirez imprimer ce graphique, ou sur n'importe quelle autre touche pour revenir au menu.

Finalement, l'option **7**, la plus importante, effectue la restauration en posant au préalable les questions suivantes:

Nouvelle image Restaurée (o/n) ? [TRUE]:
Vous pouvez continuer à partir d'une image précédente en répondant par 'n' (non).

Affiche calculs de départ (o/n) ? [FALSE]:
Répondez par 'o' (oui) pour observer le déroulement de certains calculs d'initialisation.

Nombre de lignes (max. 3) ? [3]:
Nombres de lignes à maximiser. Répondez par 2 ou 3.

Pendant le traitement, vous pouvez voir le détail des calculs. Appuyez sur **g** pour passer du mode graphique au mode texte (ou vice-versa), sur **s** pour voir défiler une étape à la fois, sur **c** pour faire défiler les calculs, sur **L** en mode graphique pour voir les combinaisons de couleurs conservées jusqu'à ce point, sur **ENTER** pour arrêter le traitement.

7.3.3 Structure interne

Le programme est divisé, comme pour «Besag1», en un programme principal «derin1.pas» et un module «derin.pas» contenant les fonctions spécifiques à cette restauration. Le module «IMAG» est bien sûr également utilisé.

La fonction principale est celle qui effectue la restauration, «RESTSEG» (fichier «D_REST.PAS»). Un grand nombre de fonctions sont utilisées que nous n'expliquerons pas en détails. Le code source se trouve comme toujours en appendice.

7.4 Autres programmes et considérations techniques

Deux autres programmes sont fournis, «timage.exe» qui permet de créer et d'éditer une image, et «taux.exe» qui permet de calculer tous les taux d'erreur (chapitre 6). De nombreuses explications tout au long des options permettent une utilisation facile de ces programmes simples.

Les programmes sont fournis avec les codes sources et avec les versions exécutables. Ces dernières ont été compilées avec l'option nécessitant un co-processeur mathématique. Si vous devez tester les programmes sur une machine n'en possédant pas, recompilez tous les programmes en ajustant cette option. Plusieurs modules ("units") ont été écrits au cours du développement de ce travail, nous les donnons

avec les autres fichiers. Ce sont des recueils de fonctions mathématiques, graphiques ou standards.

Les programmes devraient pouvoir fonctionner dans un mode graphique quelconque et nous donnons les "drivers" fournis par Borland.

Conclusion

Nous avons vu que le problème de la restauration d'images par ordinateur n'est pas simple, notamment lorsque vient le temps de tenir compte de dégradations aléatoires. L'approche du maximum *a posteriori*, bien que difficile à utiliser telle quelle, nous offre de mettre en évidence deux aspects de la question. Il devient nécessaire d'avoir un modèle mathématique pour la dégradation stochastique et pour le type d'images recherchées.

Le modèle gaussien fut employé pour décrire un bruit additif dégradant l'image, tandis que l'information *a priori* fut modélisée par un champ markovien. Nous avons vu pourquoi cette généralisation des chaînes de Markov au plan est intéressante en traitement d'images. Grâce aux distributions de Gibbs, il devient possible d'utiliser les propriétés locales des champs markoviens pour simuler un grand nombre d'images de toutes sortes.

Après avoir étudié en profondeur deux méthodes de restauration d'images basées sur un modèle *a priori* de champs markoviens, nous avons pu les implanter sur ordinateur grâce à des logiciels que nous avons spécifiquement développés. Sans entrer dans une comparaison détaillée des deux méthodes, il semble que la méthode de Besag donne généralement

de meilleurs résultats et est à tout le moins beaucoup plus rapide que celle de Derin et Elliott.

Basée sur une approche bayesienne, l'ICM de Besag est une méthode itérative qui maximise un critère à chaque pixel de l'image. Relativement simple à programmer, cette méthode nous a de plus permis, après des modifications que nous avons apporté au modèle de base, de tenir compte de dégradations plus complexes, notamment des combinaisons de bruits aléatoires additif ou multiplicatif, de fonctions non-linéaires et de fonctions d'étalement ponctuelle. En étudiant l'efficacité de l'ICM sur des images tests, nous avons pu observer la versalité et la puissance de cette technique de restauration.

La seconde méthode nous a permis, au travers du modèle markovien proposé par Derin et Elliott, de nous familiariser avec différents champs markoviens générés directement par ordinateur. Mais l'approche de restauration proposée n'a pas donné des résultats très intéressants lorsque l'on dépassait le cas de 2 couleurs, à la fois en ce qui a trait à la vitesse et à la qualité de l'estimé final.

Lorsque venait le temps de juger de la qualité d'une restauration, nous nous sommes vite rendus compte que le pourcentage de pixels mal classés n'était peut-être pas le critère le plus intéressant. Nous avons effectué une première étude de ce problème et proposé et testé différents critères alternatifs. Le plus attrayant semble être un de

ceux permettant de donner un poids plus important aux pixels des contours, ce que nous avons fait avec le Laplacien.

Bien sûr, après cette analyse des fondements et de la structure des méthodes étudiées, une comparaison complète serait nécessaire pour vraiment juger les méthodes. Mais d'autres points sont à soulever.

Tout d'abord, les champs markoviens que nous avons utilisés, notamment le plus simple proposé par Besag, peuvent-ils vraiment être considérés comme des modèles adéquats? Il est parfois difficile de voir en quoi certaines images utilisées sont des réalisations de ces modèles. Et pourtant les résultats sont là. Il serait intéresant d'étudier l'importance des champs markoviens. Nous avons fait un premier pas en donnant les développements mathématiques de l'ICM pour certains modèles.

Ceci amène un autre point. Si les images utilisées ne sont pas vraiment des réalisations des champs proposés, est-il possible de croire qu'elles sont des réalisations d'un autre modèle de champ markovien, et si oui duquel? Il devient alors primordial de s'attaquer au problème de l'estimation des paramètres. Tant Derin et Elliott que Besag en parlent dans leurs articles et proposent chacun une méthode différente; sans aucun doute pourrait-on s'y atteler dans une prochaine étape.

Finalement, si Derin et Elliott proposent une technique reconnue (la programmation dynamique) pour résoudre le problème du MAP, Besag propose une nouvelle approche, l'ICM. Or il est possible de croire que cette méthode, lorsqu'on s'y attarde quelque peu, pourrait être appliquée à d'autres problèmes de maximisation en traitement d'images. Nous pensons par exemple à l'analyse de textures, la reconnaissance de formes, l'estimation du mouvement, et bien d'autres encore.

Comme les champs markoviens qui se sont avérés des outils de premier ordre en analyse d'images, l'ICM de Besag devrait connaître un grand succès.

Bibliographie

ABEND, K., HARLEY, T.J. and Kanal, L.N. (1965), «Classification of Binary Random Patterns,» *IEEE Trans. Inform. Theory*, IT-11, 538-544.

ANDREWS, H. C. and HUNT, B. R. (1977), *Digital Image Restoration.* Englewood Cliffs, New Jersey: Prentice-Hall.

BESAG, J. (1974), «Spatial Interaction and the Statistical Analysis of Lattice Systems (with discussion),» *J. R. Statist. Soc.*, **B 36**, 192-326.

BESAG, J. (1986), «On the Statistical Analysis of Dirty Pictures (with discussion),» *J. R. Statist. Soc.*, **B 48**, 259-302.

CASASENT, D.P. (1975), «Optical Digital Radar Signal Processing,» *International Optical Computing Conference*, Washington, D.C., 23 avril, 1975.

DERIN, H. and ELLIOTT, H. (1987), «Modeling and Segmentation of Noisy and Textured Images Using Gibbs Random Fields,» *IEEE Trans. Pattern Anal. Machine Intell.*, **9**, 39-55.

FRANKS, L.E. (1969), *Signal Theory.* Prentice-Hall, Englewood CLiffs, New Jersey.

GEMAN, S. and GEMAN, D. (1984), «Stochastic Relaxation, Gibbs Distributions, and the Bayesian Restoration of Images,» *IEEE Trans. Pattern Anal. Machine Intell.*, **6**, 721-741.

GOODMAN, J.W. (1968), *Introduction to Fourier Optics*, McGraw-Hill, New York, New York.

HORN, B. K. P. (1986), *Robot Vision*, MIT Press, Cambridge, Massachusetts.

HUNT, B. R. (1975), «Digital Image Processing,» IEEE Proc., **63**, 693-708.

KINDERMANN, S. and SNELL, J.L. (1980), *Markov Random Fields and their Applications*. Providence: American Mathematical Society.

O'NEIL, E. L. (1963), *Introduction to Statistical Optics*. Addison-Wesley, Reading, Massachusetts.

PAPOULIS, A. (1965), *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill, New York.

LOHMANN, A.W., and PARIS, D.P. (1965), «Influence of Longitudinal Vibrations on Image Quality,» *Appl. Opt.*, **4**, 393-397.

ROSENFELD, A. and KAK, A.C. (1982), *Digital Picture Processing*. 2nd ed., Academic Press, New York.

ROSS, S.M. (1985), *Introduction To Probability Models*. Academic Press, San Diego.

ANNEXE 1: FICHIERS SOURCES

BESAG1.PAS

```
Program Besag1;
{
NAME: BESAG1 _ Programme test pour article de Besag 1986 in JRSS

DECLARATION:
   besag1

DESCRITPION:
   Realise l'ICM sur des images simples.
   Voir exemple page 266.

Jean-Michel Durocher, 1988-1989
}

USES
   CRT, Std, Graph, IMAG, ICM;

{--------- DECLARATION DES VARIABLES GLOBALES -------- }

Var
    val_tpi : T_iter;
    choix_min, choix_bruit, choix_tpi: integer;
    i,j, choix, choix2, nb_iter: integer;
    imOK,           { Image originale }
    imICM: image;     { Image en restauration }
    imY : imageR;     { Image bruitée }
    beta,     { Poid accorde aux voisins }
    beta_inc,   { Increment a chaque iteration }
    k: real;   { K est la variance des observations }
    Taux, variance, moyenne:real;
    mess, mess2, fich_res, nomICM, nomOK, nomY : string;
    DOfinishWait, DofinishSound, DoIterGraph, DoSound : boolean;
    dist_nlin : boolean;
    nb_sec : longint;


{---------- PROGRAMME PRINCIPAL ----------}

Begin
   InitImage( imOK);
   initImage( imICM);
   initRimage( imY);
```

```
nb_coul := 4;
beta_inc := 0;
beta := 1.5;
choix := 4;
choix2 := 1;
choix_min := 1;
choix_tpi := 1;
choix_bruit := 1;
dist_nlin := FALSE;
DoIterGraph := TRUE;
DoFinishSound := TRUE;
DoFinishWait := TRUE;
DoSonnd := FALSE;
nb_iter := 6;
k := 0.7;
RestoreCrtMode;
NomImage := 'EE';
NomICM := 'EE2';
nomOK := NomImage;
fich_res := nomOK + '.RES';
nomY := NomImage + 'Y';
while choix <> 0 do begin
   writeln;
   writeln( 'Programme de gestion de l''implantation de l''ICM');
   writeln;
   writeln('   0: Fin');
   writeln('   1: Lire une image du disque');
   writeln('   2: Ecrire l''image sur disque');
   writeln('   3: Affiche l''image graphiquement');
   writeln('   4: Ecrit les valeurs d''une image');
   writeln('   5: Bruite l''image');
   writeln('   6: Taux d''erreur');
   writeln('   7: Iteration de l''ICM ');
   writeln('  11: Ajustement des paramètres');
   writeln('  33: Imprime une image');
   writeln('  44: Imprime les valeurs d''une image');
   writeln;

   AskInt( 'Votre choix', choix);
   if choix <> 0 then begin
      case choix of
         1: Begin
               writeln( 'LECTURE');
               writeln( '1: Image originale');
               writeln( '2: Image ICM');
               writeln( '3: Image Observations (bruitée)');
               writeln;
               AskInt( 'Votre choix', choix2);
               case choix2 of
                  1: begin
                     NomImage := NomOK;
```

```
            LitUneImage(imOK, TRUE);
            BordeImage( ImOK, 0);
            CopieImage( imOK, imICM);
            CopieImageR( imOK, imY);
            NomOK := NomImage;
         end;
         2: Begin
            NomImage := NomICM;
            LitUneImage(imICM, TRUE);
            BordeImage( ImICM, 0);
            CopieImageR( imICM, imY);
            NomICM := NomImage;
         end;
         3: Begin
            NomImage := NomY;
            LitUneImageR(imY, TRUE);
            CopieRImage( imY, imICM);
            BordeImage( ImICM, 0);
            NomY := NomImage;
         end;
      end; { case }
   end;
2: Begin
      writeln( 'ECRITURE');
      writeln( '2: Image ICM');
      writeln( '3: Image Observations (bruitée)');
      writeln;
      AskInt( 'Votre choix', choix2);
      case choix2 of
         2: Begin
            NomImage := NomICM;
            EcritUneImage(imICM, TRUE);
            NomICM := NomImage;
         end;
         3: Begin
            NomImage := NomY;
            EcritUneImageR(imY, TRUE);
            NomY := NomImage;
         end;
      end; { case }
   end;
3: begin
      writeln( 'AFFICHAGE');
      writeln( '1: Image originale');
      writeln( '2: Image ICM');
      writeln;
      AskInt( 'Votre choix', choix2);
      nomImage := NomOK;
      case choix2 of
         1: Begin
            NomImage := NomOK;
```

```
            MontreUneImage(imOK);
         end;
         2: Begin
            NomImage := NomICM;
             MontreUneImage(imICM);
         end;
      end; { case }
      readln;
      RestoreCrtMode;
   end;
4: Begin
      writeln( 'ECRITURE des VALEURS');
      writeln( '1: Image originale');
      writeln( '2: Image ICM');
      writeln( '3: Image Observations (bruitée)');
      writeln;
      AskInt( 'Votre choix', choix2);
      case choix2 of
         1: printImg( imOK);
         2: printImg( imICM);
         3: printRImg( imY);
      end; { case }
   end;
5: begin
      BruiteImageR(imY, k);
      writeln( 'Bruite ok ...');
      CopieRImage(imY, imICM);
      writeln( 'Copie ok ...');
      taux := TauxErreur(imOK, imICM, k);
      writeln( 'Taux d''erreur: ', taux:6:2,' %');
      writeln( 'σ²: ', k:0:3);
   end;

6: Begin
      taux := TauxErreur(imOK, imICM, k);
      writeln( 'Taux d''erreur: ', taux:6:2,' %');
      writeln( 'σ²: ', k:0:3);
   end;
7:
   Begin
      repeat
         writeln( 'TYPE DE BRUIT');
         writeln( '0: Des détails SVP');
         writeln( '1: Bruit additif Gaussien');
         writeln( '2: Bruit multiplicatif Gaussien');
         AskInt( 'Votre choix', choix_bruit);
         If choix_bruit = 0 then
            detail_bruit;
      until choix_bruit <> 0;
      repeat
         writeln( 'MINIMISATION/MAXIMISATION');
```

```
   writeln( '0: Des détails SVP');
   writeln( '1: ICM Normal (Gaussienne et nombre d voisins)');
   writeln( '2: ICM + H (Gaussienne sur moyenne pondérée et nombre de voisins)');
   AskInt( 'Votre choix', choix_min);
   If choix_min = 0 then
      detail_min;
until choix_min <> 0;
if choix_min = 2 then
begin
   writeln( 'Choix de la matrice: ');
   For i := -1 to 1 do
   Begin
      For j := -1 to 1 do
         write( '(',i:2,',',j:2,')  ');
       writeln;
   end;
   For i := -1 to 1 do
   begin
      For j := -1 to 1 do
      Begin
         str( i, mess2);
         mess := 'h['+mess2+',';
         str( j, mess2);
         mess := mess + mess2+']';
         AskReal( mess, h[i,j], 2);
      End;
   End;
end;
AskBool( 'Φ: Distorsion non linéaire (racine carrée)', dist_nlin);
repeat
   writeln( 'TYPE D''ITERATIONS');
   writeln( '0: Des détails SVP');
   writeln( '1: SYNC');
   writeln( '2: RASTER');
   writeln( '3: SEMI2');
   AskInt( 'Votre choix', choix_tpi);
   if choix_tpi = 0 then
      detail_iter;
until choix_tpi <> 0;
AskInt( 'Nombre d''itérations', nb_iter);
IF (choix_min IN [1..max_min]) AND (choix_tpi IN [1..max_tpi]) AND
   (choix_bruit IN [1..max_bruit]) AND (nb_iter > 0) then
begin
   AskText( 'Fichier de Sortie (NIL pour aucun)', fich_res);
   val_tpi := tab_tpi[choix_tpi];
   nb_sec := Iter( imOK, imY, imICM, val_tpi, choix_min,
                   choix_bruit, dist_nlin, beta, beta_inc, k,
                   nb_iter, DoIterGraph, DoFinishSound,
                   DoFinishWait, fich_res);
   writeln( (nb_sec/100):0:2, ' secondes');
   taux := TauxErreur(imOK, imICM, k);
```

```
                writeln( 'Taux d''erreur: ', taux:6:2,' %');
                writeln( 'σ²: ', k:0:3);
             end;
          end;
       11: AjusteParam( beta, beta_inc, k, DoIterGraph, DoFinishSound, fich_res);
       33: Begin
             writeln( 'IMPRESSION');
             writeln( '1: Image originale');
             writeln( '2: Image ICM');
             writeln;
             AskInt( 'Votre choix', choix2);
             case choix2 of
                1: ImprimeUneImage( ImOK);
                2: ImprimeUneImage( ImICM);
             end; { case }
          end;
       44: Begin
             writeln( 'IMPRESSION des VALEURS');
             writeln( '1: Image originale');
             writeln( '2: Image ICM');
             writeln( '3: Image Observations (bruitée)');
             writeln;
             AskInt( 'Votre choix', choix2);
             case choix2 of
                1: pprinting( imOK);
                2: pprinting( imICM);
                3: pprintRing( imY);
             end; { case }
          end;
      End; { case }
    end;   { If }
  End;   { while }
end.
```

DERIN1.PAS

```
Program Derin1;
{
 Implantation de la méthode de Derin & Elliott.
 Pour M.Sc.A, JMD, le 6 mars 1989.
}
Uses
   crt, Std, Graph, imag, derin;


{---------- PROGRAMME PRINCIPAL ------------}
Var
   nomOK,
   nomSEG,
   oldImage
```

```
                :string;
  imOK,                 { Originale }
  imSEG,                { Bruitée }
  imRest                { Restaurée }
                :image;
  mchoix, choix,
  nb_level,
  nb_iter,
  onb_iter,
  i
                :integer;
  k,
  taux
                :real;
  nb_sec
                :longint;
  ch : char;
Begin
  nb_coul := 2;
  InitImage( imOK);
  InitImage( imSEG);
  nb_level := 2;
  mchoix := 11;
  k := 0.5;
  SEGgraph := TRUE;
  SEGDeb := FALSE;
  RestoreCrtMode;
  NomImage := 'Mona';
  nomOK := NomImage + 'A';
  nomSEG := NomImage + 'B';
  NomImage := NomSEG;
  while mchoix <> 0 do begin
    writeln;
    writeln( 'Programme de gestion de l''implantation de la segmentation de Derin & Elliott');
    writeln;
    writeln('  0: Fin');
    writeln('  1: Lire une image');
    writeln('  2: Ecrire l''image');
    writeln('  3: Affiche l''image');
    writeln('  5: Bruite l''image');
    writeln('  6: Taux d''erreur');
    writeln('  7: Iteration RESTAURATION program. dynamique');
    writeln('  9: Générer une image buffer');
    writeln(' 11: Ajustement des paramètres');
    writeln(' 12: Courbe des exponentielles (2 couleurs)');
    writeln(' 13: Histogramme de la réalisation (2 couleurs)');
    writeln(' 20: Imprime une image');
    writeln;
    mchoix := 11;
    AskInt( 'Votre choix', mchoix);
    case mchoix of
```

```
1: Begin
   writeln( 'LECTURE');
   writeln( '1: Image originale');
   writeln( '2: Image SEG');
   writeln( '3: Image Rest');
   writeln;
   choix := 1;
   AskInt( 'Votre choix', choix);
   case choix of
      1: begin
         NomImage := NomOk;
         LitUneImage(imOK, TRUE);
         BordeImage( ImOK, 0);
         NomOk := NomImage;
         CopieImage( imOK, imSEG);
         NomImage := NomSEG;
      end;
      2: Begin
         LitUneImage(imSEG, TRUE);
         BordeImage( ImSEG, 0);
         NomSEG := NomImage;
      end;
      3: Begin
         LitUneImage(imRest, TRUE);
         BordeImage( ImRest, 0);
         NomSEG := NomImage;
      end;
   end; { case }
end;
2: Begin
   writeln( 'ECRITURE');
   writeln( '1: Image SEG');
   writeln( '2: Image REST');
   writeln;
   choix := 1;
   AskInt( 'Votre choix', choix);
   case choix of
      1: EcritUneImage(imSEG, TRUE);
      2: EcritUneImage(imRest, TRUE);
   end; { Case }
   NomSEG := NomImage;
end;
3: begin
   writeln( 'AFFICHAGE');
   writeln( '1: Image originale');
   writeln( '2: Image SEG');
   writeln( '3: Image REST');
   writeln;
   choix := 2;
   AskInt( 'Votre choix', choix);
   case choix of
```

```
      1: Begin
         nomImage := NomOK;
         MontreUneImage(imOK);
         nomImage := NomSEG;
      end;
      2: MontreUneImage(imSEG);
      3: MontreUneImage(imRest);
   end; { case }
   readln;
   RestoreCrtMode;
end;
5: begin
   BruiteImage(imSEG, k);
   taux := TauxErreur(imOK, imSEG, k);
   writeln( 'Taux d''erreur: ', taux:6:2,' %');
end;

6: Begin
   writeln( 'TAUX D''ERREUR');
   writeln( '1: Image SEG');
   writeln( '2: Image REST');
   writeln;
   choix := 2;
   AskInt( 'Votre choix', choix);
   case choix of
      1: taux := TauxErreur(imOK, imSEG, k);
      2: taux := TauxErreur(imOK, imRest, k);
   end; { case }
   writeln( 'Taux d''erreur: ', taux:6:2,' %');
end;
7: Begin
   writeln( 'ITERATIONS RESTAURATION: ');
   writeln;
   SEGGNew := TRUE;
   AskBool( 'Nouvelle image Restaurée', SEGGNew);
   nb_sec := RestSeg( imSEG, imRest, k);
   writeln;
   writeln( (nb_sec/100):0:2, ' secondes');
   i := 1;
   while SEGSound AND (i < 200) AND ( Not KeyPressed) do
   begin
      sound( 100 + i * 20);
      delay( 90);
      sound( 100 + 4000 - i * 20);
      delay( 90);
      i := i + 1;
   End;
   nosound;
   if KeyPressed then
      ch := readkey;
   OldImage := NomImage;
```

```
      NomImage := 'TEST';
      EcritUneImage( imRest, TRUE);
      NomImage := OldImage;
   End;
   9: Begin
      writeln( 'GENERE: ');
      writeln( '1: Avec buffer      B');
      writeln( '2: Sans buffer      N');
      writeln( '3: Balayage Sylvain S');
      writeln;
      choix := 3;
      AskInt('Votre choix', choix);
      if choix IN [1,2,3] then
      begin
         nb_iter := 10;
         AskInt('Nombre d''itérations', nb_iter);
         onb_iter := nb_iter;
         SEGGNew := TRUE;
         AskBool( 'Nouvelle image de départ', SEGGNew);
         case choix of
            1: nb_sec := BGenereSEG( nb_iter, imSEG);
            2: nb_sec := NGenereSEG( nb_iter, imSEG);
            3: nb_sec := SGenereSEG( nb_iter, imSEG);
         end; { Case }
         writeln( (nb_sec/100):0:2, ' secondes');
         writeln( nb_iter,'/',onb_iter,' itérations complétées');
         i := 1;
         while SEGSound AND (i < 200) AND ( Not KeyPressed) do
         begin
            sound( 100 + i * 20);
            delay( 90);
            sound( 100 + 4000 - i * 20);
            delay( 90);
            i := i + 1;
         End;
         nosound;
         if KeyPressed then
            ch := readkey;
      end; { IF IN }
      OldImage := NomImage;
      NomImage := 'TEST';
      EcritUneImage( imSeg, TRUE);
      NomImage := OldImage;
   End;
   11: AjusteParam( k);
   12: CourbExp;
   13: Histog2( imSEG);
   20: Begin
       writeln( 'IMPRESSION');
       writeln( '1: Image originale');
       writeln( '2: Image SEG');
```

```
            writeln( '3: Image Rest');
            writeln;
            choix := 2;
            AskInt( 'Votre choix', choix);
            case choix of
               1: ImprimeUneImage( ImOK);
               2: ImprimeUneImage( ImSEG);
               3: ImprimeUneImage( ImRest);
            end; { case }
            choix := 20;
         end;
      End; { case }
   End;   { while }
end.
```

ICM.PAS

```
Unit ICM;
{
NOM:   ICM  _ Unité pour Implantation ICM

AUTEUR:  Jean-Michel Durocher

DATE:    28 février 1989

DESCRITPION:
    Contient les fonctions et procédures spécifiques à l'implantation de
    l'ICM.
}
Interface
Uses
   Graph, Crt, STD, IMAG, printer, Dos;
Type
   BufLigne = array[1..2] of ligne;
   t_iter = (SYNC, RASTER, SEMI2);
Const
   max_tpi = 3;
   max_min = 2;
   max_bruit = 2;
   tab_tpi : array[ 1..max_tpi] of t_iter = (SYNC, RASTER, SEMI2);
   tab_ctpi : array[SYNC..SEMI2] of string[6] = ('SYNC', 'RASTER', 'SEMI2');
   tab_cmin : array[ 1..2] of string[20] = ('NORM', 'NORM+PSF');
   tab_cbruit : array[ 1..2] of string[20] = ('ADDITIF', 'MULTIPLICATIF');

Var
   icm_full : boolean;
   un2k : real;

Function  u( Var img:image; x,y,c:integer):integer:
Procedure fu_init( Var img:image; x,y:integer);
```

```
Function  minICM( Var imgY:imageR; Var imgICM: image; x, y:integer; beta:real;
                  choix_bruit:integer; dist_nlin:boolean):integer;
Function  minICMV( Var imgY:imageR; Var imgICM: image; x, y:integer; beta:real;
                  choix_bruit:integer; dist_nlin:boolean):integer;
Procedure Detail_min;
Procedure Detail_bruit;
Procedure Detail_iter;

Function Iter( Var imgOK:image; Var imgY:imageR; Var imgICM:image;
                  tpi: t_iter; CMin, choix_bruit: integer; dist_nlin:boolean;
                  beta, beta_inc, k:real; nb_iter:integer;
                  DoIterGraph, DoFinishSound, DoFinishWait: boolean;
                  fich_res:string):LongInt;
Procedure AjusteParam( Var beta, beta_inc, k: real;
                       Var DoIterGraph, DoSound: boolean;
                       Var fich_res: string);


Implementation
Var
   icm_futabi, icm_futab : array[0..NCOULEUR] of integer;

{ ---------------- }
Procedure fu_init;
{ ---------------- }
{ CETTE FONCTION RETOURNE LE NOMBRE DE VOISIN AUTOUR DE (X,Y) QUI ONT LA
  MEME COULEUR POUR CHACUNE DES COULEURS }
Var
   i, j: integer;
Begin
   icm_futab := icm_futabi;
   for i := -1 to 1 do
   begin
      for j := -1 to 1 do
         if ((i <> 0) OR (j <> 0)) then
         begin
            inc( icm_futab[img[x+i,y+j]]);
            if do_debug then
            begin
               writeln( 'x:',x, ' y:',y,' i:',i,' j:',j);
               writeln( 'img[',x+i,',',y+j,']:',img[x+i,y+j],
               ' icm_futab[',img[x+i,y+j],']:',icm_futab[img[x+i,y+j]]);
            end;
         end;
   end;
end;   { FU_INIT -------- }


{ ---------------- }
Function u;
{ ---------------- }
```

```
{ CETTE FONCTION RETOURNE LE NOMBRE DE VOISIN AUTOUR DE (X,Y) QUI ONT LA
  MEME COULEUR 'C' }
Var
   i, j, compte: integer;
Begin
   compte := 0;
   for i := -1 to 1 do
   begin
      for j := -1 to 1 do
         if ((i <> 0) OR (j <> 0)) then
            if img[x+i,y+j] = c then
               compte := compte+1;
   end;
   u := compte;
end;   { U -------- }

{ ---------------- }
Function minICM;
{ ---------------- }
{ CETTE FONCTION RETOURNE LA COULEUR QUI MINISE LE CRITERE DE L'ICM AU
  POINT (X,Y) }
Var
   c, i,j: integer;      { Indice de la couleur }
   minf,           { Le minimum de la fonction }
   valf : real;    { La valeur de la fonction }
   cOK : integer;  { couleur qu minimise le critere; }
   nu: integer;    { Nombre de voisins ayant meme couleur }
   diff: integer;  { Difference entre observation et la couleur }
   cy : integer;   { Couleur au point }
   mess: string;
   mess2: string;
   rcy : real;
   mk : real;

Begin
   minf := 1000;
   fu_init( imgICM, x, y);
   rcy := imgY[x,y];
   IF do_debug then
   begin
      writeln( '(',x:2,',',y:2,')');
      for j := -1 to 1 do
      begin
         for i := -1 to 1 do
           write( ' ',imgICM[x+i,y+j]);
         writeln;
      end;
   end;
   if do_debng then
      writeln( 'imgY[x,y] :', imgY[x,y]:0:3);
   for c := 1 to nb_coul do
```

```
   begin
      nu := icm_futab[c];
      if (icm_full) OR (nu <> 0) OR (c = cy) then
      begin
         if choix_bruit = 1 then {ADDITIF}
         begin
            if NOT dist_nlin then
               valf := un2k * sqr( rcy-c) - beta * nu
            else
               valf := un2k * sqr( rcy-SQRT(c)) - beta * nu
         end
         else {MULTIPLICATIF}
         begin
            if NOT dist_nlin then
               mk := c
            else
               mk := sqrt(c);
            valf := un2k / sqr(mk) * sqr( rcy - mk) - beta * nu
         end;
         if valf < minf then
         begin
            minf := valf;
            cOK := c;
            if do_debug then
               write( '*');
         end;
      end;
      if do_debng then
         writeln('C:', c:2, '  NU:', nu:2, ' VALF:', valf:0:2);
   end;
   minICM := cOK;
end;   { MinICM ---------- }


{ ---------------- }
Function minICMV;
{ ---------------- }
{ CETTE FONCTION RETOURNE LA COULEUR QUI MINISE LE CRITERE DE L'ICM AU
  POINT (X,Y) en tenant compte des voisins (PSF) }
Var
   c: integer;      { Indice de la couleur }
   minf,            { Le minimum de la fonction }
   valf : real;     { La valeur de la fonction }
   cOK : integer;   { couleur qu minimise le critere; }
   nu: integer;     { Nombre de voisins ayant meme couleur }
   diff: real;   { Difference entre observation et la couleur }
   somme : real;
   cy,           { Couleur au point }
   mk : real;    { Variance dependant du point }
   mess: string;
   mess2: string;
```

```
  oc, i, j, ck : integer; { Couleur au point }
  hbuf : array[1..9] of real;
Begin
  If NOT ((x > 2) AND (x < DIMIX-1) AND (y > 2) AND (y < DIMIY-1)) then
    cOK := minICM( imgY, imgICM, x,y, beta, choix_bruit, dist_nlin)
  else
  Begin
    CheckDebug;
    minf := 100000;
    fu_init( imgICM, x, y);
    oc := imgICM[x,y];
    { Petit calcul avant }
    somme := 0;
    IF do_debug then
    begin
      writeln( '(',x:2,',',y:2,')');
      for j := -1 to 1 do
      begin
        for i := -1 to 1 do
          write( ' ',imgICM[x+i,y+j]);
        writeln;
      end;
    end;
    ck := 1; { numero du pixel de 1 a 9 }
    for j := -1 to 1 do
    Begin
      for i := -1 to 1 do
      begin
        { Calcul en enlevant couleur du milieu }
        hbuf[ck] := hval( imgICM, x+i, y+j) - h[-i,-j]*oc;
        if do_debug then
          write( ' ', hbuf[ck]:6:2);
        inc( ck);
      end;
      if do_debug then
        writeln;
    end;
    for c := 1 to nb_coul do
    begin
      nu := icm_futab[c];
      if do_debug then
        writeln( 'C:',c,' NU:', nu);
      { Σ (Y(o,q) - Σ H(k,m) X(o-k,y-q))² }
      somme := 0;
      ck := 1;
      for i := -1 to 1 do
        for j := -1 to 1 do
        begin
          cy := imgY[x+i,y+j];
          if choix_bruit = 1 then {ADDITIF}
          begin
```

```
                if dist_nlin then
                   somme := somme + sqr( cy - SQRT((hbuf[ck] + h[-i,-j]*c )))
                else
                   somme := somme + sqr( cy - (hbuf[ck] + h[-i,-j]*c ))
             end
             else {MULTIPLICATIF}
             begin
                if dist_nlin then
                   mk := SQRT(hbuf[ck] + h[-i,-j]*c )
                else
                   mk := hbuf[ck] + h[-i,-j]*c;
                somme := somme + 1 / SQR(mk) * SQR( cy - mk);
             end;
             inc( ck);
          end;
        valf := un2k * somme - beta*nu;
        if valf < minf then
        begin
           minf := valf;
           cOK := c;
           if do_debug then
              write( '*');
        end;
        if do_debug then
           writeln( 'VALF: ', valf:0:2);
      end;
      if do_debug then
      begin
        writeln( 'Best:', cOK);
        writeln;
      end;
   End;
   minICNV := cOK;
end;   { MinICNV ---------- }

{ ---------- }
Function Iter;
{ ---------- }
Var
   i, j, ik, ik_max, i_inc, j_inc, l, new_ji: integer;
   taux, k2, varr, n, nc: real;
   buf : BufLigne;
   mess, mess2, mss_iter, mss_chang, mss_taux:string[80];
   fhour, lhour, fmin, lmin, fsec, lsec, fsec100, lsec100: word;
   tot_t, li : LongInt;
   stop : boolean;
   f: TEXT;
Begin
   un2k := 1 / 2 / k;
   IF Fich_res <> 'NIL' then
   begin
```

```
   assign( f, fich_res);
   if fexist( fich_res) then
      append( f)
   else
      rewrite( f);
   writeln( f);
   writeln( f, nb_iter, ' itérations de type ',tab_ctpi[tpi],
            ' minimisation ', tab_cmin[Cmin]);
   taux := TauxErreur( imgOK, imgICM, k2);
   writeln( f, '   Erreur initiale: ', Taux:6:2, ' %   σ²:', k2:0:3);
   writeln( f, ' icm_full:', icm_full);
   if CMin = 1 then
      writeln( f, 'β=', beta:0:3, ' K=', k:0:3)
   else if Cmin = 2 then
   begin
      writeln( f, 'β=', beta:0:3, ' σ²=', k:0:3);
      writeln( f, 'H:');
      For i := -1 to 1 do
      begin
         For j := -1 to 1 do
            write( f, h[i,j]:6:3, ' ');
         writeln( f);
      end;
   end; { CMin }
end; { Fich_res }

stop := FALSE;
n := DIMIX * DIMIY;
nc := n / 100;
If DoIterGraph then
   MontreUneImage(imgICM);
l := 0;
GetTime( fhour, fmin, fsec, fsec100);
while (l < nb_iter) AND Not stop do
begin
   IF Fich_res <> 'NIL' then
      writeln( f, 'ITER:', l, ' β:', beta:0:2);
   ik := 1;
   ik_max := 1;
   If tpi = SEMI2 then
      ik_max := 4;
   While ik <= ik_max do
   Begin
      i := 2;
      i_inc := 1;
      If tpi = SEMI2 then
      Begin
         Case ik of
            1,4: i := 2;
            2,3: i := 3;
         end; { Case }
```

```
   i_inc := 2;
End;
while (i < DIMIY) AND Not stop do
begin
   j := 2;
   j_inc : =1;
   IF tpi = SEMI2 then
   begin
      Case ik of
         1,3: j : =2;
         2,4: j : =3;
      end; { Case }
      j_inc := 2;
   end;
   While (j <= DIMIX - 1) AND Not Stop do
   begin
      {
       SELON LE TYPE DE Minimisation (Loi conditionnelle et loi a priori)
      }
      case CMin of
         1: { Normal }
            new_ji := minICM( imgY, imgICM, j,i, beta, choix_bruit, dist_nlin);
         2: { Normal + PSF }
            new_ji : =minICMV( imgY, imgICM, j,i, beta, choix_bruit, dist_nlin);
      end; { Case }
      CASE tpi of
         SYNC:
            buf[2,j] := new_ji;
         RASTER, SEMI2:
         Begin
            imgICM[j,i] := new_ji;
            If DoIterGraph then
               MontrePixel( j, i, imgICM[j,i]);
         End;
      End; { Case }
      j := j + j_inc;
      if Not TestDebug( DoIterGraph, imgICM) then
         stop := TRUE;
   end; { j }
   IF tpi = SYNC then
   Begin
      if i >= 3 then
         BufImg( imgICM, buf[1], i-1, DoIterGraph);
      buf[1] : =buf[2];
   end;
   i := i + i_inc;
   if DoFinishSound then
   begin
      sound( 300);
      delay( 50);
      nosound;
```

```
            End;
          end;   { While i }
          If (tpi = SYNC) AND Not stop then
            BufImg( imgICM, buf[1], DIMIY - 1, DoIterGraph);
          ik := ik + 1;
        End; { IK }
        l := l + 1;
        beta := beta + beta_inc;
        IF Fich_res <> 'NIL' then
        begin
          taux := TauxErreur( imgOK, imgICM, k2);
          writeln( f, '   Erreur: ', Taux:6:2, ' %   σ²:', k2:0:3);
        end;
      end; { For Nb_iter }
      GetTime( lhour, lmin, lsec, lsec100);

      if DoFinishSound then
        for i := 1 to 10 do
        begin
          sound( 100 + i * 100);
          delay( 200);
        End;
      nosound;
      if DofinishWait then
        Readln;
      If DoIterGraph then
        RestoreCrtMode;
      writeln( l, ' itération(s) terminée(s)', ' sur ', nb_iter);
      li := 1;
      tot_t := (((lhour-fhour)*li*60 + lmin-fmin)*60 + lsec-fsec) * 100 + lsec100-fsec100;
      IF Fich_res <> 'NIL' then
      begin
        writeln( f, 'Temps:', (tot_t/100):8:2, ' s');
        close( f);
      end;
      Iter := tot_t;
    end;   { -------- Iter ------ }

    Procedure Detail_bruit;
    Begin
      HighVideo;
      writeln;
      writeln( '1: Bruit ADDITIF Gaussien: b ~ N(0,σ²) (MOYENNE 0)');
      writeln;
      writeln( '      y = Φ(H(x)) + b');
      writeln;
      writeln( '   et donc, puisqu''alors f(y|x) = f{b+Φ(H(x))}, on a:');
      writeln;
      writeln( '                          -½     - (y - Φ(H(x)) - 0}²/(2σ²)');
      writeln( '      f(y|x)  = (2πσ²)      e');
      writeln;
```

```
  writeln( '2: Bruit MULTIPLICATIF Gaussien: b ~ N(1 ,σ²) (MOYENNE 1)');
  writeln;
  writeln( '      y = Φ(H(x)) . b');
  writeln;
  writeln( '   et donc, puisqu''alors f(y|x) = P{ bΦ(H(x))}, on a:');
  writeln( '   que Y|x ~ N(Φ(H(x)),[Φ(H(x))]²)');
  writeln;
  writeln( '                                    -½    - (y - Φ(H(x))}²/(2[Φ(H(x))]²σ²)');
  writeln( '      f(y |x)  = (2π[Φ(H(x))]²σ²)     e');
  writeln;
  NormVideo;
End;

Procedure detail_min;
Begin
  HighVideo;
  writeln;
  writeln( '1: NORMAL (montré pour bruit additif)');
  writeln( '                    -½    - (y - μ(x)}²/(2σ²)');
  writeln( '   f(y|x)  = (2πσ²)     e');
  writeln;
  writeln( '              Su(μ(x))      Su(μ(x))');
  writeln( '   p(x|xδ) = e          / Σ(e         )');
  writeln;
  writeln( '2: NORMAL+PSF (montré pour bruit additif)');
  writeln( '                    -½    - (y - H(x)}²/(2σ²)');
  writeln( '   f(y|x)  = (2πσ²)     e');
  writeln;
  writeln( '  où:  H(x) est une réponse impulsionnelle de type');
  writeln( '           moyenne pondérée des voisins');
  writeln;
  writeln( '              Su(μ(x))      Su(μ(x))');
  writeln( '   p(x|xδ) = e          / Σ(e         )');
  writeln;
  NormVideo;
End;

Procedure Detail_iter;
Begin
  HighVideo;
  writeln( '1: SYNC: Synchronisé:');
  writeln( '   Tous les pixels sont changés en même temps');
  writeln( '2: RASTER: Balayage:');
  writeln( '   Les pixels sont changés l''un après l''autre,');
  writeln( '   de gauche à droite et de bas en haut');
  writeln( '3: SEMI2: Semi-Synchronisé:');
  writeln( '   Les pixels sont changés l''un après l''autre,');
  writeln( '   mais 1 colonne sur 2, 1 ligne sur 2');
  NormVideo;
End;
```

```
{--------------} Procedure AjusteParam; {------------}
Var
  ch_coul:boolean;
  i:integer;
Begin
   AskReal( 'β',beta,2);
   AskReal('βinc',beta_inc,2);
   AskReal('K(σ²)', k, 3);
   AskBool( 'Iteration graphique', DoIterGraph);
   AskBool( 'Son', DoSound);
   AskBool('Debug', do_debug);
   AskBool('ICM full (toutes les couleurs)', icm_full);
   repeat
      AskInt( 'nombre de couleur',nb_coul);
   until nb_coul IN [0..NCOULEUR];
   ch_coul := FALSE;
   AskBool( 'Change les couleurs', ch_coul);
   if ch_coul then
      for i := 0 to nb_coul do
      begin
         write( i:2);
         AskInt( ' ', choixcouleur[i]);
         couleur[i] := oldcouleur[choixcouleur[i]];
      end;

end;    ( -------------- AjusteParam ------------ }

Var
   i : integer;
Begin
   icm_full := TRUE;
   for i := 0 TO NCOULEUR DO
       icm_futabi[i] := 0;
end.
```

DERIN.PAS

```
Unit DERIN;
{
NOM:   DERIN  _ Unité pour Implantation de l'artice de Derin & Elliott

AUTEUR:  Jean-Michel Durocher

DATE:    6 mars 1989

DESCRITPION:
    Contient les fonctions et procédures spécifiques à l'implantation de
    la méthode de programmation dynamique de Derin & Elliott
```

```
MODIFICATIONS:
    21 mars 1989. JMD, Prob. exponentielles.
}
Interface
Uses
   Graph, Crt, STD, IMAG, printer, Dos, Plot, Math;
Const
   BETAMAX = 4;
Type
   AlphaT = Array[1..NCOULEUR] of real;
   BetaT = Array[1..BETAMAX] of real;
   BufLigne = array[1..2] of ligne;
   TPot = Array[0..63, 0..63] of real;    { Somme des potentiels pour toutes les comb. possibles }
Var
   SEGGNew, SEGRand, SEGgraph, SEGDeb, SEGSound: boolean;
   oldcouleur : array[0..NCOULEUR] of pointer;
   choixcouleur : array[0..NCOULEUR] of integer;
   { histogramme 2 expos. 9 = 512 differents, codés S,U1,...,U4,V1,..,V4 }
   histo2: array[1..512] of integer;
   Betas: BetaT;
   Alphas: AlphaT;
   Pot: Tpot;     { Potentiels et differences }


Procedure AjusteParam( Var k:real);
Function  Vc( Var img:image; coul, xp, yp, c:integer; beta: real): real;
Function  pSEG( Var img: image; coul, xp, yp: integer):real;
Function  genSEG( Var img: image; xp, yp: integer):integer;
Procedure ShowGenMess( Bmess:string);
Function  GenCheck( Bmess:string; Var DoStep:boolean; Var Img:image):char;
Procedure InfoGen( i, j, k, nt, nb_iter:integer; changed:longint; Var img:image);
Function  BGenereSEG( Var nb_iter: integer; Var img: image):LongInt;
Function  NGenereSEG( Var nb_iter: integer; Var img: image):LongInt;
Function  SGenereSEG( Var nb_iter: integer; Var img: image):LongInt;
Procedure CourbExp;
Procedure Histog2( Var img:image);
Function  RestSeg( Var imB, imR: Image; k: real):longint;

Implementation
{$I \tpu\d_misc}     { MISCELANIOUS }
{$I \tpu\df_gen}     { Fonctions generales pour GENERATION }
{$I \tpu\d_gen}      { GENERATION }
{$I \tpu\d_estim}    { ESTIMATION }
{$I \tpu\df_rest}    { Fonctions generales pour RESTAURATION }
{$I \tpu\d_rest}     { RESTAURATION }
{-------------- INITIALISATION DU MODULE --- -------}
Var
   i: integer;
Begin
   SEGGraph := TRUE;
   SEGDeb := FALSE;
```

```
   SEGRand := TRUE;
   SEGGNew := TRUE;
   SEGSonnd := TRUE;
   for i := 0 to NCOULEUR do
   begin
      oldcouleur[i] := couleur[i];
      choixcouleur[i] := i;
   end;
   for i := 1 to BETAMAX do
      betas[i] := 1;
   for i := 1 to nb_coul do
      alphas[i] := 1;
end.
```

D_MISC.PAS

```
{
       MISCELANIOUS procedures and functions for UNIT 'DERIN'.
}


Procedure AjusteParam;
Var
   mess, mess2:string;
   code, cc, i : integer;
   ch_coul : boolean;
Begin
   AskInt( 'Nombre de couleurs',nb_coul);
   ch_coul := TRUE;
   AskBool( 'Change les couleurs', ch_coul);
   if ch_coul then
      for i := 0 to nb_coul do
      begin
         write( i:2);
         AskInt( ' ', choixcouleur[i]);
         couleur[i] := oldcouleur[choixcouleur[i]];
      end;

   FOR i := 1 to BETAMAX do
   BEGIN
      str(i, mess2);
      mess := 'B' + mess2;
      AskReal( mess, betas[i], 2);
   END; { For }

   AskReal( 'Variance K', k, 2);

   AskBool( 'Graphique', SEGGraph);
   SEGDeb := FALSE;
   if NOT SEGGraph then
   begin
```

```
      SEGDeb := TRUE;
      AskBool( 'Debug', SEGDeb);
   end;

   AskBool( 'Random ', SEGRand);
   AskBool( 'SOUND ', SEGSound);
End;    { ------ AjusteParam ------ }

Function GenCheck;
Var
   stop : boolean;
   ch: char;
Begin
   ch := #0;
   stop := FALSE;
   if keypressed OR DoStep then
   begin
      ch := UpCase( readkey);
      case ch of
         'S': DoStep := TRUE;
         'C': Dostep := FALSE;
         'G': if SEGGraph then
              begin
                 SEGGraph := FALSE;
                 SEGDeb := TRUE;
                 RestoreCrtMode;
                 AskBool( 'Debug', SEGDeb);
                 IF Bmess <> 'REST' then
                    ShowGenMess( Bmess)
                 else
                 begin
                    ClrScr;
                 end;
              end
              else
              Begin
                 SEGGraph := TRUE;
                 SEGDeb := FALSE;
                 MontreUneImage( Img);
              end;
         #13: stop := TRUE;
      end;
   end;
   if stop then
      ch := #1;
  GenCheck := ch;
End;  { ------- GenCheck ------- }
```

```
{
      Fonctions et procedures associees a D_GEN pour UNIT 'DERIN'
}

Function Vc;    { POTENTIEL ASSOCIE A UNE CLIQUE }
Var
   somme : real;
Begin
   somme := 0;
   case c of
      1:  { ** }
      Begin
         if coul = img[xp+1,yp] then somme := -beta
         else somme := beta;
         if coul = img[xp-1,yp] then somme := somme - beta
         else somme := somme + beta;
      end;
      2:  { * }
          { * }
      Begin
         if coul = img[xp,yp+1] then somme := -beta
         else somme := beta;
         if coul = img[xp,yp-1] then somme := somme - beta
         else somme := somme + beta;
      end;
      3:  {  * }
          {  *  }
      Begin
         if coul = img[xp+1,yp-1] then somme := -beta
         else somme := beta;
         if coul = img[xp-1,yp+1] then somme := somme - beta
         else somme := somme + beta;
      end;
      4:  {  *  }
          {  * }
      Begin
         if coul = img[xp+1,yp+1] then somme := -beta
         else somme := beta;
         if coul = img[xp-1,yp-1] then somme := somme - beta
         else somme := somme + beta;
      end;
   end; { Case }
   IF SEGDeb then
      write( ' Vc(',c,'):', somme:5:2);
   Vc := somme;
End; { --------- Vc --------- }

Function pSEG;  { PROB. d'une couleur selon la dist. de Gibbs (non normalisée) }
Var
   somme: real;
```

```
   c: integer;
Begin
   somme:= 0;
   for c := 1 to BETAMAX do
      somme := somme + Vc( img, coul, xp, yp, c, betas[c]);
   IF SEGDeb then
      writeln( ' PSEG(',coul:2,'):', exp(-somme):10:2);
   pSEG := exp(-somme);
End; { ------- pSEG ------ }


Function genSEG;   { Couleur genere avec grande PROB. }
Var
  i, bestcoul:integer;
  pv: real;
  pp: array[0..NCOULEUR] of real;
  ppmax, somme : real;
Begin
   somme := 0;
   for i:= 1 to nb_coul do
   begin
      pp[i] := pSEG( img, i, xp, yp);
      somme := somme + pp[i];
   end;
   pp[0] := 0;
   IF SEGRand then
   begin
      for i := 1 to nb_coul do
      begin
         pp[i] := pp[i] / somme;
         IF SEGDeb then
         begin
            write( ' pp[',i:2,']:', pp[i]:5:2);
            if (i mod 6) = 0 then
               writeln;
         end;
         pp[i] := pp[i] + pp[i-1];
      end;
      if (SEGDeb AND ((nb_coul mod 6) <> 0)) then
         writeln;
      pv := random;
   end;
   IF SEGDeb then
   begin
      for i := 1 to nb_coul do
      begin
         write( ' pp[',i:2,']:', pp[i]:5:2);
         if (i mod 6) = 0 then
            writeln;
      end;
      if (nb_coul mod 6) <> 0 then
```

```
        writeln;
      IF SEGRand then
        write( '     Random: ', pv:0:2)
      else
        write( '     Random: ---');
   end; { SEGDeb }

   IF SEGRand then
   begin
      for i := nb_coul DOWNTO 1 do
        if pp[i] > pv then
          bestcoul := i;
   end
   else
   begin
      ppmax := pp[1];
      bestcoul := 1;
      for i := 2 to nb_coul do
      begin
        if pp[i] > ppmax then
        begin
          bestcoul := i;
          ppmax := pp[i];
        end;
      end;
   end;
   IF SEGDeb then
   begin
      write( '     GenSEG:', bestcoul:3);
      IF NOT SEGRand then
        write( '   ppmax : ', ppmax:5:2);
   end;
   genSEG := bestcoul;
End; { ------- GenSEG ------- }

Procedure ShowGenMess;
Var
   i : integer;
Begin
   ClrScr;
   GotoXY( 1, 1);
   write( bmess, ' ');
   writeln( 'GENERATION D''IMAGE. S:Step, C:Continue, G:Graph(o/n), <RETURN>:Stop');
   GOTOXY( 1, 2);
   for i := 1 to BETAMAX do
      write( ' B',i,': ', betas[i]:5:2);
   GotoXY( 1, 3);
   writeln( '          ',',     *    ',',    *     ',',   *      ');
   writeln( '    **    ',',     *    ',',   *      ',',    *     ');

End; { ----- ShowGenMess ----- }
```

```
Procedure InfoGen;
Var
   px, py : integer;
Begin
   if NOT SEGGraph then
   begin
      IF SEGDeb then
      begin
         GotoXY( 45, 2);
         write( 'X:',j:2);
      end;
      GotoXY( 49, 2);
      write( ' Y:',i:2, ' K:',k,' It:',nt:2,'/', nb_iter:2, ' Ch:', changed:6);
      IF SEGDeb then
      begin
         For px := -1 to 1 do
         begin
            for py := -1 to 1 do
            begin
               GotoXY( 47 + px*4, py+4);
               write( img[j+px,i+py]:3);
             end;
         end;
         writeln;
      end;
   end;
end; { ------ InfoGen ----- }
```

D_GEN.PAS

```
{
        Procedures pour Generer des IMAGES (UNIT 'DERIN')
}

{ *** AVEC BUFFER *** }
Function BGenereSEG;
Var
   i, j, nt : integer;
   buf: BufLigne;
   DoStep, Stop: boolean;
   pBeta, sommeB : real;
   ch : char;
   cg : byte;
   changed : longint;
   fhour, fmin, fsec, fsec100: word;
   lhour, lmin, lsec, lsec100: word;
   li: LongInt;
```

```
Begin
   GetTime( fhour, fmin, fsec, fsec100);
   sommeB := 0;
   pBeta := abs( betas[1]);
   for i := 1 to BETAMAX do
   begin
      sommeB := sommeB + 2 * abs( betas[i]);
      if abs( betas[i]) < pBeta then
         pBeta := abs( betas[i]);
   end;
   IF SEGGNew then
   begin
      InitImage( img);
      BordeImage( img, 0);
   end;
   IF SEGGraph then
      MontreUneImage( img)
   else
      ShowGenMess( 'BUF');
   DoStep := FALSE;
   nt := 1;
   Stop := FALSE;
   randomize;
   changed := 1;
   while (nt <= nb_iter) AND ( NOT Stop) AND (changed <> 0) do
   begin
      changed := 0;
      i := 2;
      while (i <= (DIMIY-1)) AND (NOT Stop) do
      begin
         ImgBuf( img, buf[2], i);
         j := 2;
         while (j <= (DIMIX-1)) AND (NOT Stop) do
         begin
            ch := GenCheck( 'BUF', DoStep, Img);
            stop := ch = #1;
            InfoGen( i, j, 0, nt, nb_iter, changed, img);
            cg := genSEG( img, j, i);
            IF img[j,i] <> cg then
               inc( changed);
            buf[2,j] := cg;
            inc( j);
         end; { While j }
         if i > 2 then
            BufImg( img, buf[1], i-1, SEGGraph);
         buf[1] := buf[2];
         inc( i);
      end; { While i }
      BufImg( img, buf[1], i-1, SEGGraph);
      inc( nt);
   end; { While iter }
```

```
   IF SEGGraph then
      RestoreCrtMode;
   nb_iter := nt - 1;

   GetTime( lhour, lmin, lsec, lsec100);
   li := 1;
   BGenereSEG := (((lhour-fhour)*li*60 + lmin-fmin)*60 + lsec-fsec) * 100 + lsec100-fsec100;
End; { -------- BGenereSEG ------- }

{ *** SANS BUFFER (RASTER SCAN) *** }
Function NGenereSEG;
Var
   i, j, nt : integer;
   DoStep, Stop: boolean;
   pBeta, sommeB : real;
   ch : char;
   cg : byte;
   changed : longint;
   fhour, fmin, fsec, fsec100: word;
   lhour, lmin, lsec, lsec100: word;
   li: LongInt;
Begin
   GetTime( fhour, fmin, fsec, fsec100);
   sommeB := 0;
   pBeta := abs( betas[1]);
   for i := 1 to BETAMAX do
   begin
      sommeB := sommeB + 2 * abs( betas[i]);
      if abs( betas[i]) < pBeta then
         pBeta := abs( betas[i]);
   end;
   IF SEGGNew then
   begin
      InitImage( img);
      BordeImage( img, 0);
   end;
   IF SEGGraph then
      MontreUneImage( img)
   else
      ShowGenMess( '');
   DoStep := FALSE;
   nt := 1;
   Stop := FALSE;
   randomize;
   changed := 1;
   while (nt <= nb_iter) AND ( NOT Stop) AND (changed <> 0) do
   begin
      changed := 0;
      i := 2;
      while (i <= (DIMIY-1)) AND (NOT Stop) do
      begin
```

```
        j := 2;
        while (j <= (DIMIX-1)) AND (NOT Stop) do
        begin
          ch := GenCheck( '', DoStep, Img);
          stop := ch = #1;
          InfoGen( i, j, 0, nt, nb_iter, changed, img);
          cg := genSEG( img, j, i);
          IF img[j,i] <> cg then
            inc( changed);
          img[j,i] := cg;
          IF SEGGraph then
            MontrePixel( j, i, img[j,i]);
          inc( j);
        end; { While j }
        inc( i);
      end; { While i }
      inc( nt);
    end; { While iter }
    IF SEGGraph then
      RestoreCrtMode;
    nb_iter := nt - 1;
    GetTime( lhour, lmin, lsec, lsec100);
    li := 1;
    NGenereSEG := (((lhour-fhour)*li*60 + lmin-fmin)*60 + lsec-fsec) * 100 + lsec100-fsec100;
End; { -------- NGenereSEG ------- }

{ *** BALAYAGE DE SYLVAIN (1 ligne sur 2, 1 colonne sur 2) *** }
Function SGenereSEG;
Var
    i, j, k, nt : integer;
    DoStep, Stop: boolean;
    pBeta, sommeB : real;
    ch : char;
    cg: byte;
    changed : longint;
    fhour, fmin, fsec, fsec100: word;
    lhour, lmin, lsec, lsec100: word;
    li: Longint;
Begin
    GetTime( fhour, fmin, fsec, fsec100);
    sommeB := 0;
    pBeta := abs( betas[1]);
    for i := 1 to BETAMAX do
    begin
      sommeB := sommeB + 2 * abs( betas[i]);
      if abs( betas[i]) < pBeta then
        pBeta := abs( betas[i]);
    end;
    IF SEGGNew then
    begin
      InitImage( img);
```

```
   BordeImage( img, 0);
end;
IF SEGGraph then
   MontreUneImage( img)
else
   ShowGenMess( 'S');
DoStep := FALSE;
nt := 1;
Stop := FALSE;
randomize;
changed := 1;
while (nt <= nb_iter) AND ( NOT Stop) AND (changed <> 0) do
begin
   changed := 0;
   k := 1;
   while ( k <= 4) AND ( NOT Stop) do
   begin
      case k of
         1,3: i := 2;
         2,4: i := 3;
      end; { Case }
      while (i <= (DIMIY-1)) AND (NOT Stop) do
      begin
         case k of
            1,4: j := 2;
            2,3: j := 3;
         end; { Case }
         while (j <= (DIMIX-1)) AND (NOT Stop) do
         begin
            ch := GenCheck( 'S', DoStep, Img);
            stop := ch = #1;
            InfoGen( i, j, k, nt, nb_iter, changed, img);
            cg := genSEG( img, j, i);
            IF img[j,i] <> cg then
               inc( changed);
            img[j,i] := cg;
            IF SEGGraph then
               MontrePixel( j, i, img[j,i]);
            j := j + 2;
         end; { While j }
         i := i + 2;
      end; { While i }
      inc( k);
   end; { While k }
   inc( nt);
end; { While iter }
IF SEGGraph then
   RestoreCrtMode;
nb_iter := nt - 1;
GetTime( lhour, lmin, lsec, lsec100);
li := 1;
```

```
   SGenereSEG := (((lhour-fhour)*li*60 + lmin-fmin)*60 + lsec-fsec) * 100 + lsec100-fsec100;
End; { -------- SGenereSEG ------- }
```

D_ESTIM.PAS

```
{
      Fonctions et procedures pour Estimation des parametres.
            UNIT 'DERIN'
}

Procedure CourbExp;
Var
   sld: P_SLD;
   i, automa : integer;
   retour : boolean;
   betai, betaf: real;
   list : P_LISTE;
   ch : char;
   sb : string;
   step : integer;
Begin
   betai := 1;
   AskReal('B (sera le même pour les quatres)', betai, 2);
   p_defaut( sld);          { Valeur par defaut pour une structure SLD };
   sld.format.text_style := 5;
   str( betai:0:2, sb);
   sld.labels.title :=  'D&E. Dist. de Gibbs avec 4 Betas = '+ sb;
   sld.labels.ylabel := 'Probabilite';
   sld.labels.xlabel := 'Nb. de pixels differents (2 couleurs)';
   sld.extremes.xmaxo := 8;
   sld.extremes.ymaxo := 1;
   sld.extremes.xmino := 0;
   sld.extremes.ymino := 0;
   WITH list Do
   begin
      n_pts := 9;
      for i := 0 to 8 do
      begin
         pts[i+1,x] := i;
         pts[i+1,y] := exp( - (i-4)*2.0*betai) / (exp( - (i-4)*2.0*betai) +
                                                  exp( + (i-4)*2.0*betai));
      end;
   end;
   automa := MAXIM;
   SetGraphMode( p_mode);
   step := 50;
   With sld.box Do
      movebox( xmin, ymin, xmax, ymax, step, sld.colors.cline, sld.colors.cbackground);
   retour := showlist( sld, list, automa);
   ch := UpCase(readkey);
   if (ch = 'P') then
```

```
   With sld.box Do
      ImprimeUneRegion( xmin, ymin, xmax, ymax);
   RestoreCrtMode;

End;  { ---------- CourbEXP --------- }


{ Retourne une valeur de 1 a 512 selon la valeur de la grid }
Function tplus( Var img: image; xx, yy: integer):integer;
Var
   ret, i: integer;

   Function uv( pos:integer):integer;
   Begin
      case pos of
         1:  uv := img[xx  ,yy  ]-1; { S  }
         2:  uv := img[xx-1,yy  ]-1; { u1 }
         3:  uv := img[xx  ,yy-1]-1; { u2 }
         4:  uv := img[xx+1,yy  ]-1; { u3 }
         5:  uv := img[xx  ,yy+1]-1; { u4 }
         6:  uv := img[xx-1,yy-1]-1; { v1 }
         7:  uv := img[xx+1,yy-1]-1; { v2 }
         8:  uv := img[xx+1,yy+1]-1; { v3 }
         9:  uv := img[xx-1,yy+1]-1; { v4 }
      end; { case }
   End;

Begin
   ret := 1;
   for i := 1 to 9 do
      ret := ret + uv(i) * ipower( 2, (i-1));
   tplus := ret;
End;

Procedure Histog2;
Var
   dohinit, retour: boolean;
   sld: P_SLD;
   i, j, k, automa, vh: integer;
   list : P_LISTE;
   ch : char;
   step,px,py : integer;
   mess, mess2: string;
   ok: boolean;
   tot_eq, nb_vh: integer;
   max_found1 : SET of 0..255;
   max_found2 : SET of 0..255;
   mmax : integer;
   max_v : integer;
Begin
   dohinit := TRUE;
```

```
AskBool( 'Remet histogramme à zéro', dohinit);
CLrScr;
IF dohinit then
   for i := 1 to 512 do
      histo2[i] := 0;
ch := 'C';
for i := 3 to (DIMIY-2) do
   for j := 3 to (DIMIX-2) do
   begin
      vh := tplus( img, j, i);
      inc(histo2[ vh ]);
      IF ch = 'S' then
      begin
         For px := -1 to 1 do
         begin
            for py := -1 to 1 do
            begin
                GotoXY( 47 + px*4, py+4);
                write( img[j+px,i+py]:3);
             end;
         end;
         writeln;
         writeln( 'X:',j:3,' Y:',i:3,' Indice de histo2: ', vh:3);
      end;
      if keypressed OR (ch = 'S') then
         ch := UpCase( readkey);
   end;

{ GRAPHIQUE }
p_defaut( sld);          { Valeur par defaut pour une structure SLD };
sld.format.text_style := 5;
sld.labels.title :=  'D&E. Histogramme 2 couleurs';
sld.labels.ylabel := 'Quantite';
sld.labels.xlabel := 'vecteur S,U1,..U4,V1,..V4';
sld.extremes.xmaxo := 513;
sld.extremes.ymaxo := 0.5;
sld.extremes.xmino := 0;
sld.extremes.ymino := 0;
sld.format.line_mode := TO_X;
sld.format.line_style := 0;
sld.format.line_thick := 3;
sld.format.targn := ' ';
WITH list Do
begin
   n_pts := 512;
   for i := 1 to 512 do
   begin
      pts[i,x] := i;
      pts[i,y] := histo2[i];
   end;
end;
```

```
automa := INPUT;
SetGraphMode( p_mode);
step := 50;
With sld.box Do
   movebox( xmin, ymin, xmax, ymax, step, sld.colors.cline, sld.colors.cbackground);
retour := showlist( sld, list, automa);
ch := UpCase(readkey);
if (ch = 'P') then
   With sld.box Do
      ImprimeUneRegion( xmin, ymin, xmax, ymax);
RestoreCrtMode;
repeat
   nb_vh := 5;
   askint( 'Combien de valeurs d''histo.', nb_vh);
   ok := FALSE;
   tot_eq :=  (nb_vh * (nb_vh-1)) div 2;
   if tot_eq > MAXDIM then
      writeln( 'Impossible !!!, ',tot_eq, ' équations seront générées, (MAXDIM=',MAXDIM,')')
   else
   begin
      ok := TRUE;
      str( tot_eq, mess2);
      mess := mess2 + ' équations seront générées. OK';
      AskBool( mess, ok);
   end;
until ok;
writeln( 'Trouve les maximums et Constructions de la matrice.');
max_found1 := [];
max_found2 := [];
writeln( nb_vh);
for k := 1 to nb_vh do
begin
   mmax := 1;
   max_v := -1000;
   for i := 1 to 256 do
      if (histo2[i] > max_v) AND (NOT ((i-1) IN max_found1)) then
      begin
         mmax := i;
         max_v := histo2[i];
      end;
   for i := 257 to 512 do
      if (histo2[i] > max_v) AND (NOT ((i-257) IN max_found2)) then
      begin
         mmax := i;
         max_v := histo2[i];
      end;
   if mmax < 257 then
      max_found1 := max_found1 + [mmax-1]
   else
      max_found2 := max_found2 + [mmax-257]
```

```
   end;
   writeln( 'Valeurs conservées: ');
   for i := 0 to 255 do
      if i IN max_found1 then
      begin
         write( i+1, ',·');
      end;
   for i := 0 to 255 do
      if i IN max_found2 then
      begin
         write( i+257, ',  ');
      end;
   writeln;

End;  { -------- Histog2 ------- }
```

DF_REST.PAS

```
{
       Fonctions et procedures associees a D_REST
              UNIT 'DERIN'
}

Function  ikVc( Var col1, col2:colonne; d, c:integer; beta: real; doOver, doBelow, doLeft:boolean): real;
Var
   somme : real;
   coul : byte;
Begin
   somme := 0;
   coul := col2[d];
   case c of
      1:  { ** }
      Begin
         if doleft then
            if coul = col1[d] then somme := somme - beta
            else somme := somme + beta;
      end;
      2:  { * }
          { * }
      Begin
         if doBelow then
            if coul = col2[d+1] then somme := -beta
            else somme := beta;
      end;
      3:  {  * }
          { *  }
      Begin
         if doBelow AND doleft then
            if coul = col1[d+1] then somme := somme - beta
            else somme := somme + beta;
      end;
```

```
      4:  { * }
          {  * }
      Begin
        if doleft AND doOver then
          if coul = coll[d-1] then somme := somme - beta
          else somme := somme + beta;
      end;
   end; { Case }
   IF SEGDeb then
     write( ' ikVc(',c,'):', somme:5:2);
   ikVc := somme;
End; { --------- ikVc --------- }

Function pikSEG2( Var img:image;  ix, iy:integer; DoBelow, doleft:boolean):real;
Begin
End;

Function  pikSEG( Var coll, col2:colonne; d:integer; doOver, doBelow, doLeft:boolean):real;
Var
   somme: real;
   c: integer;
Begin
   somme:= 0;
   for c := 1 to BETAMAX do
     somme := somme + ikVc( coll, col2, d, c, betas[c], doOver, doBelow, doLeft);
   IF SEGDeb then
     writeln( ' PikSEG:', (-somme):10:2);
   pikSEG := (-somme);
End; { ------- pikSEG ------ }


{ Calcul de IK pour chaque comparaison }
Function  compIK( Var imR:image; DD, ix, iy: integer; doleft:boolean):real;
Var
   tot : real;
   i: integer;
Begin
   tot := 0;
   for i := 0 to (DD-1) do
     tot := tot + pikSEG2( imR, ix, iy+i, (i < (dd-1)), doleft);
   IF SEGDeb then
     writeln( ' compIK:', tot:10:2);
   compIK := tot;
End;

Function VPot( Var coll, col2: colonne; DD:integer; doLeft:boolean):real;
Var
  i: integer;
  somme : real;
Begin
  somme := 0;
```

```
  if SEGDeb then
  begin
     write( 'C1:');
     for i := 0 to (DD-1) do
        write( col1[i+1]);
     write(' C2:');
     for i := 0 to (DD-1) do
        write( col2[i+1]);
     writeln;
  end;
  for i := 0 to (DD-1) do
     somme := somme + pikSEG( col1, col2, i+1, i <> 0, i <> (DD-1), doLeft);
   IF SEGDeb then
      writeln( ' VPot:', somme:10:2);
   Vpot := somme;
End;

{ Calcul intermediaire de IK valide pour les 8 comparaison d'un choix }

Function  prevIK2( Var imB, imR:image; DD, ix, iy:integer; un2k:real):real;
Var
   tot : real;
   i: integer;
Begin
   tot := 0;
   for i := 0 to (DD-1) do
      tot := tot + un2k * sqr( imB[ix,iy+i] - imR[ix,iy+i]);
   prevIK2 := -tot;
End;

{ Calcul intermediaire de IK valide pour les 8 comparaison d'un choix }
Function  prevIK( Var col1, col2:colonne; DD:integer; un2k:real):real;
Var
   tot : real;
   i: integer;
Begin
   if SEGDeb then
   begin
      write( 'C1:');
      for i := 0 to (DD-1) do
         write( col1[i+1]);
      write(' C2:');
      for i := 0 to (DD-1) do
         write( col2[i+1]);
      writeln;
   end;
   tot := 0;
   for i := 0 to (DD-1) do
      tot := tot + un2k * sqr( col1[1+i] - col2[1+i]);
   prevIK := -tot;
End;
```

D_REST.PAS

```
{
       Fonctions et procedures pour RESTAURATION
              UNIT 'DERIN'
}


Function CimComb( Var imB:image; DD, ix, iy:integer):byte;
Var
  ir, tot : integer;
Begin
   tot := 0;
   for ir := 0 to (DD-1) do
      tot := tot + (imB[ix,iy+ir]-1)*ipower( nb_coul, ir);
   CimComb := tot;
End;




{ Restauration selon programmation dynamique.
  Maximum de 3 lignes a la fois, 4 couleurs }
Function RestSeg;
Type
   TPot02 = Array[1..4, 1..4, 1..4, 1..4] of real;  { Pour l'info. au dessus }
   TBuf = Array[0..63, 1..DIMIX] of byte; { Choix precedent }
   Tik  = Array[0..63] of real;           { Valeur de IK a l'iteration prec. }
   TCCoul = Array[0..63, 0..3] of byte;   { Combinaisons de couleurs }
Var
   fhour, fmin, fsec, fsec100: word;
   lhour, lmin, lsec, lsec100: word;
   col1, col2: colonne;
   buf1, buf2 : Tbuf;
   Ccoul : TCCoul;
   DIk : Tpot;
   Pot2: TPot02;        { Potentiels avec info. au dessus (1,1)(1,2)(2,1)(2,2)}
   ik1, ik2: Tik;
   MM, DD: integer;     { Nombre de couleurs (niveaux) et dimension des STRIPs }
   md, nb, ir, ic, ict, iik, ix, iy, id, i, j: integer;
   td : array[0..2] of integer;    { Diviseurs }
   i1,i2,i3,i4,icmax: integer;
   ikm, pik, un2k, ikmax : real;
   Go, DoStep, DoInt : boolean;
   cc: byte;
   li: LongInt;
   ick : byte;
   ch : char;
 Procedure InfoBuf;
 Var
```

```
   mx, iik, i, j: integer;
Begin
  IF SEGDeb then
  begin
     GotoXY( 1,1);
     mx := 4 - MM+1;
     writeln( 'X: ', ix:3, ' Y:', iy:3);
     for iik := 0 to (DD-1) do
     begin
        write( ' ');
        for i:= 0 to (md-1) do
        begin
           for j := imax( 2, ix-mx) to (ix-1) do
              write( CCoul[ buf1[ i,j], iik]:1);
           write( '  ');
        end; { I }
        ClrEol;
        writeln;
     end; { iik }
     for i := 0 to (md-1) do
        write( ik1[i]:5:1, ' ');
     ClrEol;
  end;
End; { ---- Info Buf ---- }
Procedure InfoB2( oicc, icc:integer);
Var
   i, j, mx:integer;
Begin
   if SEGDeb then
   begin
      mx := 4 - MM+1;
      GotoXY( 1+icc*(mx+1), DD+7); write( oicc); ClrEol;
      for i := 0 to (DD-1) do
      begin
         GotoXY( 1+icc*6, DD+8+i);
         for j := imax( 2, ix-mx) to ix do
            write( CCoul[ buf2[icc, j], i]:1);
         ClrEol;
      end;
   end;
End;   { ---- Infob2 ---- }
Procedure InfoC;
Var
   i: integer;
Begin
   IF SEGDeb then
   begin
      GotoXY( 70, 1); write( 'ICT IC');
      for i := 0 to (DD-1) do
      begin
         GotoXY( 70, i+2);
```

```
        write( Ccoul[ict,i]:2,Ccoul[ic,i]:2);
      end;
      GotoXY( 1, DD+6);
      writeln( 'IC:', ic:2, ' ICT:', ict:2);
   end;
end; { ---- InfoC ---- }
Procedure InfoLigne;
Var
   i,j,ic: integer;
   ch : char;
Begin
   if SEGGraph then
   begin
      for ic := 0 to (md-1) do
      begin
          for i := 2 to (ix-1) do
            for j := 0 to (DD-1) do
            begin
               cc := CCoul[buf1[ic,i],j];
               montrePixel( i, iy+j, cc);
            end;
         sound( 1000 + 100 * ic);
         delay( 100);
         nosound;
         ch := readkey;
      end;
      sound( 1000);
      delay( 500);
      nosound;
      for i := 2 to (ix-1) do
         for j:= 0 to (DD-1) do
         begin
            cc := imB[i,iy+j];
            montrePixel( i, iy+j, cc);
         end;
   end;
end; { ---- InfoLigne ---- }
Function IkUp( i1,i2,i3,i4:integer):real;
Var
   tot : real;
Begin
   tot := 0;
   if i1 = i2 then
      tot := -betas[1]
   else
      tot :=  betas[1];
   if i4 = i2 then
      tot := tot - betas[2]
   else
      tot := tot + betas[2];
   if i3 = i2 then
```

```
      tot := tot - betas[3]
    else
      tot := tot + betas[3];
    if i4 = i1 then
      tot := tot - betas[4]
    else
      tot := tot + betas[4];
    IkUp := -tot;
 End; { ---- IkUp ---- }
Begin
   DoInt := FALSE;
   AskBool( 'Affiche calculs de départ', DoInt);
   GetTime( fhour, fmin, fsec, fsec100);
   un2k := 1 / 2 / k;

   MM := nb_coul;
   repeat
      DD := 3;
      AskInt( 'Nombre de lignes (max. 3)', DD);
   until DD <= 3;
   { Construit les DIVISEURS pour le remplacement des couleurs }
   writeln( 'DIVISEURS');
   md := ipower( MM, DD);
   for i := 0 to (DD-1) DO
   begin
      td[i] := md div ( ipower( MM, (DD-i)));
      IF DoInt then
         writeln( i:2,': ', td[i]);
   end;
   { Construit les COULEURS }
   writeln( 'COULEURS');
   For ic := 0 to (md-1) do
      For ir := 0 to (DD-1) do
      begin
         cc := ((ic div td[ir]) mod MM) + 1;
         Ccoul[ic,ir] := cc;
         If DoInt then
            writeln( 'CRC:', ic:2, ir:2, cc:2);
      end;
   { Calcul des POTENTIELS pour toutes les combinaisons possibles }
   writeln('POTENTIELS');
   for i := 0 to (md-1) do
      for j := 0 to (md-1) do
      begin
         { remplace des deux cotes }
         for ir := 0 to (DD-1) do
         begin
            col1[1+ir] := Ccoul[i,ir];
            col2[1+ir] := Ccoul[j,ir];
         end;
         { calcul potentiel }
```

```
      Pot[i,j] := VPot( col1, col2, DD, TRUE);
      If DoInt then
         writeln( 'IJP: ' , i:3, j:3, ' ', pot[i,j]:0:2);
   end;
{ Calcul des DIFFERENCES au carre pour toutes les combinaisons possibles }
writeln('DIFFERENCES');
for i := 0 to (md-1) do
   for j := 0 to (md-1) do
   begin
      { remplace des deux cotes }
      for ir := 0 to (DD-1) do
      begin
         col1[1+ir] := Ccoul[i,ir];
         col2[1+ir] := Ccoul[j,ir];
      end;
      { calcul difference }
      Dik[i,j] := prevIK( col1, col2, DD, un2k);
      If DoInt then
         writeln( 'IJD: ' , i:3, j:3, ' ',Dik[i,j]:0:2);
   end;
{ POTENTIELS AU DESSUS }
for i1 := 1 to MM do
for i2 := 1 to MM do
for i3 := 1 to MM do
for i4 := 1 to MM do
begin
   Pot2[i1,i2,i3,i4] := IkUp( i1,i2,i3,i4);
   writeln( i1:2, i2:2, i3:2, i4:2, ': ', Pot2[i1,i2,i3,i4]:0:2);
end;
{ Initialise les BUFFERS AUX LIMITES }
for i := 0 to (md-1) do
begin
   ik1[i] := 0;
   ik2[i] := 0;
   buf1[i,1] := 0;
   buf2[i,1] := 0;
   buf1[i,DIMIX] := 0;
   buf2[i,DIMIX] := 0;
end;
{ Commence en copiant image }
IF SEGGNew then
   imR := imB;
IF SEGGraph then
   MontreUneImage( imR);
{ ITERATION SUR TOUTE L'IMAGE }
iy := 2;
Go := TRUE;
DoStep := FALSE;
while (iy <= (DIMIY-DD)) AND Go Do
Begin
   ix := 2;
```

```
While (ix <= (DIMIX-1)) AND Go do
begin
   { Pour chaque combinaison possible }
   InfoBuf;
   ic := 0;
   While (ic <= (md-1)) AND Go do
   begin
      IF SEGDeb then
         GotoXY( 1, 1);
      IF SEGGraph then
         MontrePixel( ix, iy, 0);
      { Trouve a quelle combinaison de couleur l'original correspond }
      ick := CimComb( imB, DD, ix, iy);
      { Conserve sans changement la 1ere colonne }
      { Et calcul les IK restreints }
      if ix = 2 then
      begin
         buf1[ic,ix] := ic;
         ik1[ic] := Dik[ick, ic];
         for i := 1 to (DD-1) do
            if Ccoul[ic,i-1] = Ccoul[ic,i] then
               ik1[ic] := ik1[ic] + Betas[2]
            else
               ik1[ic] := ik1[ic] - Betas[2]
      end;

      { Test avec les 8 combinaisons precedentes }
      if ix > 2 then
      begin
         { Calcul intermediaire (diff. de valeurs) }
         pik := Dik[ick, ic];
         icmax := 0;
         ikmax := -10000;
         ict := 0;
         While (ict <= (md-1)) AND Go do
         begin
            InfoC;
            { Calcul critere }
            ikm := ik1[ict] + pik + Pot[ict, ic];
            if iy > 2 then
            begin
               i1 := imR[ix-1,iy-1]; i2 := imR[ix,iy-1];
               i3 := Ccoul[ict, 0];  i4 := Ccoul[ic,0];
               ikm := ikm + Pot2[i1,i2,i3,i4];
            end;
            IF SEGDeb then
               writeln( 'IKM[',ict:2,']: ', ikm:6:2);
            ch := GenCheck( 'REST', DoStep, ImR);
            GO := NOT (ch = £1);
            if ch = 'L' then
               infoLigne;
```

```
            { Test si maximum }
            if ikm >= ikmax then
            begin
               ikmax := ikm;
               icmax := ict;
            end;
            inc( ict);
         end; { ICT }
         { Conserve le bon choix pour cette combinaison }
         ik2[ic] := ikmax;
         buf2[ic] := buf1[icmax];
         buf2[ic,ix] := ic;
         infoB2( icmax, ic);
         IF Go then
         begin
            ch := GenCheck( 'REST', DoStep, ImR);
            GO := NOT (ch = #1);
            if ch = 'L' then
               infoLigne;
         end;
      end; { IX > 2 }
      inc( ic);
   end; { IC }
   { Conserve les nouveaux buffer represantant les choix precedent }
   if ix > 2 then
   begin
      buf1 := buf2;
      ik1 := ik2;
   end;
   if SEGGraph then
      MontrePixel( ix,iy, imB[ix,iy]);
   inc( ix);
end; { IX }
{ Choisir la meilleur et afficher une ligne }
ikmax := ik1[0];
icmax := 0;
for i := 1 to (md-1) do
   if ik1[i] > ikmax then
   begin
      icmax := i;
      ikmax := ik1[i];
   end;
if SEGDeb then
begin
   writeln;
   writeln('Meilleur: ',icmax);
end;
for i := 2 to (DIMIX-1) do
begin
   cc := CCoul[buf1[icmax,i],0];
   imR[i,iy] := cc;
```

```
        if SEGGraph then
          montrePixel( i, iy, cc);
      end;
      { SI fini, recopie tout }
      IF iy = dimiy-dd then
      begin
        for i := 2 to (DIMIX-1) do
          for ir := 1 to (DD-1) do
          begin
            cc := CCoul[buf1[icmax,i],ir];
            imR[i, iy+ir] := cc;
            if SEGGraph then
              montrePixel( i, iy+ir, cc);
          end;
      end;
      inc( iy);
    end; { IY }
    GetTime( lhour, lmin, lsec, lsec100);
    If SegGraph then
      RestoreCrtMode;
    li := 1;
    RestSeg := (((lhour-fhour)*li*60 + lmin-fmin)*60 + lsec-fsec) * 100 + lsec100-fsec100;
End;   { ------- RESTSEG ----- }
```

IMAG.PAS

```
Unit Imag;
{
NOM:   Imag  _ Unité de gestion d'images.

AUTEUR:  Jean-Michel Durocher

DATE:    15 septembre 1988

DESCRITPION:
   Unité de gestion des Images pour un affichage en graphique monochrome.
}
Interface
Uses
   Graph, Crt, STD, printer, MATH;
Const
   DIMPX = 4;      { Dimension d'un point (nombre de pixels) en X }
   DIMPY = 4;      { Dimension d'un point (nombre de pixels) en Y }
   DIMIX = 100;   { Nombre de points en X }
   DIMIY = 50;      { Nombre de points en Y }
   TOTX = 400;      { DIMIX * DIMPX }
   TOTY = 200;      { DIMIY * DIMPY }
   NCOULEUR = 16;   { DIMPX*DIMPY }
   ExtImage : string[4] = '.IMA';
Type
```

```
   ligne = array[1..DIMIX] of byte;
   ligneR = array[1..DIMIX] of real;
   colonne = array[1..DIMIY] of byte;
   image = array[1..DIMIX, 1..DIMIY] of byte;
   imageR = array[1..DIMIX, 1..DIMIY] of real;
   PSF = array[-1..1, -1..1] of real; { Fonction d'etalement }
Var
   couleur: array[0..NCOULEUR] of pointer;
   oldconlenr : array[0..NCOULEUR] of pointer;
   choixcouleur : array[0..NCOULEUR] of integer;
   Nomlmage: string;              { Nom de l'image }
   mode, driver : integer;        { Mode et driver utilisés }
   nb_coul : integer;             { Nombre de couleurs effectivement ntilisées }
   h : PSF;
   do_debug : boolean;

Procedure PrintImg( var img:image);
Procedure PrintRImg( var img:imageR);
Procedure PPrintImg( var img:image);
Procedure PPrintRImg( var img:imageR);
Function  TestDebug( Var doit:boolean; Var img:image):boolean;
Procedure CheckDebug;
Procedure MarqueurOn( x, y: integer);
Procedure MarqueurOff( x, y: integer);
Procedure CopieImage(Var imDE, imA:image);
Procedure CopieImageR(Var imDE:image; Var imA:imageR);
Procedure CopieRImage(Var imDE:imageR; Var imA:image);
Procedure MontrePixel(x, y, code_couleur: integer);
Procedure MontreUneImage(Var forme:image);
Procedure MontreUneLigne(Var forme:image; Var buf:ligne; pos:integer);
Procedure LitUneImage(Var forme:image; Demande: boolean);
Procedure LitUneImageR(Var forme:imageR; Demande: boolean);
Procedure LitUneImageASCII(Var forme:image);
Procedure EcritUneImage(Var forme:image; Demande: boolean);
Procedure EcritUneImageR(Var forme:imageR; Demande: boolean);
Procedure EditeUneImage(Var forme:image);
Procedure LitUneImageTGA(Var forme:image);
Procedure ImprimeGraphique;  { Imprime tout les graphiques }
Procedure ImprimeUneImage( Var forme:image);  { Imprime l'image }
Procedure ImprimeUneRegion( x1, y1, x2, y2: integer);
Procedure GereUneImage(Var forme:image);   { Permet d'appeler les procedures ci-dessus }
Function  hval( Var img:image; xp, yp: integer):real;
Function  hRval( Var img:imageR; xp, yp: integer):real;
Procedure BruitUnif(Var img:imageR; Var k:real);
Procedure BruitConv(Var img:imageR; Var k:real; demande:boolean);
Procedure BruitRacine(Var img:imageR; Var k:real);
Procedure BruitGaussAdd(Var img:imageR; Var k:real);
Procedure BruitGaussMul(Var img:imageR; Var k:real);
Procedure BruiteImageR(Var img:imageR; Var k:real);
Procedure BruiteImage(Var img:image; Var k:real);
Function  FTauxErreur( Var imOK, imN, imERR: image; doerr:boolean; Var k2:real):real;
```

```
Function  TauxErreur( Var imOK, imN: image; Var k2:real):real;
Procedure ImgBuf( Var img:image; Var buf:ligne; pos:integer);
Procedure BufImg( Var img:image; Var buf:ligne; pos:integer; DoIterGraph:boolean);
Procedure BordeImage( Var im:image; coul:integer);
Procedure InitImage( Var im: image);
Procedure InitRImage( Var im: imageR);
Procedure ShowImage( nng: pointer; Var img: image);
Procedure SaveImage( Var nng: pointer);

Implementation
Const
   xcode_pix : array[1..NCOULEUR] of integer =
           (2,3,3,2,3,1,2,4,2,1,3,4,1,1,4,4);
   ycode_pix : array[1..NCOULEUR] of integer =
           (2,3,2,3,1,2,4,3,1,3,4,2,1,4,4,1);
Var
   XPos, YPos, LastPosX, LastPosY: word;
   Haut, xinc, yinc, err_code: integer;
   startx, starty, endx, endy, i, j: word;
   OnPeutAfficher : Boolean;
   ch : char;
   choice : string[100];
   marqueur: pointer;

Procedure PrintImg;
Const
   MCOL = 23;
Var
   vcol, i, j, k:integer;
Begin
   vcol := MCOL;
   AskInt( 'Nombre de colonnes', vcol);
   i := 1;
   while i <= DIMIX do
   begin
      write( 'Y\X');
      k := 0;
      while (k<=vcol-1) AND ((i+k)<=DIMIX) do
      begin
         write( (i+k):3);
         k := k+1;
      end;
      writeln;
      for j := 1 to DIMIY do
      begin
         write( j:3);
         k := 0;
         while (k<=vcol-1) AND ((i+k)<=DIMIX) do
         begin
            write( img[i+k,j]:3);
            k := k+1;
```

```
        end;
        writeln;
      end;
      writeln;
      i := i + vcol;
   end;
End;

Procedure PrintRImg;
Const
   MCOL = 12;
Var
   vcol, i, j, k:integer;
Begin
   vcol := MCOL;
   AskInt( 'Nombre de colonnes', vcol);
   i := 1;
   while i <= DIMIX do
   begin
      write( 'Y\X');
      k := 0;
      while (k<=vcol-1) AND ((i+k)<=DIMIX) do
      begin
         write( (i+k):6);
         k := k+1;
      end;
      writeln;
      for j := 1 to DIMIY do
      begin
         write( j:3);
         k := 0;
         while (k<=vcol-1) AND ((i+k)<=DIMIX) do
         begin
            write( img[i+k,j]:6:2);
            k := k+1;
         end;
         writeln;
      end;
      writeln;
      i := i + vcol;
   end;
End;

Procedure PPrintImg;
Const
   MCOL = 23;
Var
   vcol, i, j, k:integer;
Begin
   vcol := MCOL;
   AskInt( 'Nombre de colonnes', vcol);
```

```
   i := 1;
   while i <= DIMIX do
   begin
      write( lst, 'Y\X');
      k := 0;
      while (k<=vcol-1) AND ((i+k)<=DIMIX) do
      begin
         write( lst, (i+k):3);
         k := k+1;
      end;
      writeln( lst);
      for j := 1 to DIMIY do
      begin
         write( lst, j:3);
         k := 0;
         while (k<=vcol-1) AND ((i+k)<=DIMIX) do
         begin
            write( lst, img[i+k,j]:3);
            k := k+1;
         end;
         writeln( lst);
      end;
      writeln( lst);
      i := i + vcol;
   end;
End;

Procedure PPrintRImg;
Const
   MCOL = 12;
Var
   vcol, i, j, k:integer;
Begin
   vcol := MCOL;
   AskInt( 'Nombre de colonnes', vcol);
   i := 1;
   while i <= DIMIX do
   begin
      write( lst, 'Y\X');
      k := 0;
      while (k<=vcol-1) AND ((i+k)<=DIMIX) do
      begin
         write( lst, (i+k):6);
         k := k+1;
      end;
      writeln( lst);
      for j := 1 to DIMIY do
      begin
         write( lst, j:3);
         k := 0;
         while (k<=vcol-1) AND ((i+k)<=DIMIX) do
```

```
        begin
           write( lst, img[i+k,j]:6:2);
           k := k+1;
        end;
        writeln( lst);
     end;
     writeln( lst);
     i := i + vcol;
  end;
End;


Function TestDebug;
{ ------ Si 'Alt-D', change etat du DEBUG ------ }
{ ------ Si 'Alt-G', affiche graphique --------- }
Var
   ret_bool: boolean;
Begin
   ret_bool := TRUE;
   IF keypressed then
   begin
      ret_bool := FALSE;
      IF Readkey = #0 then
         Case Readkey of
            #32:
               begin
                  do_debug := NOT do_debug;
                  IF do_debug AND DoIt then
                  begin
                     DoIt := FALSE;
                     RestoreCrtMode;
                  end;
                  ret_bool := TRUE;
               end;
            #34:
               begin
                  DoIt := NOT DoIt;
                  If DoIt then
                  begin
                     Do_debug := FALSE;
                     MontreUneImage(img);
                  end
                  else
                     RestoreCrtMode;
                  ret_bool := TRUE;
               end;
         end; { Case}
   end;
   TestDebug := ret_bool;
End;
```

```
Procedure CheckDebug;
{ ------ Si 'Alt-D', change etat du DEBUG ------ }
Var
   ret_key: char;
Begin
   IF keypressed then
      IF Readkey = £0 then
         IF Readkey = £32 then
            do_debug := NOT do_debug;
End;

Procedure InitImage;
Var
  i, j: integer;
Begin
   for i := 1 to DIMIX do
      for j:=1 to DIMIY do
         im[i,j] := 1;
End;

Procedure InitRImage;
Var
  i, j: integer;
Begin
   for i := 1 to DIMIX do
      for j:=1 to DIMIY do
         im[i,j] := 1;
End;

Procedure CopieImage;
Var
   i, j:integer;
begin
   for i := 1 to DIMIX do
      for j:=1 to DIMIY do
         imA[i,j] := imDE[i,j];
end;

Procedure CopieImageR;
Var
   i, j:integer;
begin
   for i := 1 to DIMIX do
      for j:=1 to DIMIY do
         imA[i,j] := imDE[i,j];
end;

Procedure CopieRImage;
Var
   i, j:integer;
begin
```

```
   for j:= 2 to (DIMIY-1) do
      for i := 2 to (DIMIX-1) do
         imA[i,j] := imin( nb_coul, imax( Round( imDE[i,j]), 1));
end;

Procedure MontrePixel;
Begin
   PutImage( startx + (x-1)*DIMPX, starty + (y-1)*DIMPY, couleur[ code_couleur]^, 0);
End;

Procedure MontreUneLigne;
Var
   j:integer;
Begin
   For j := 2 to DIMIX-1 do
   begin
      If buf[j] > NCOULEUR then
      begin
         RestoreCrtMode;
         Writeln( 'Erreur buf[',j,'] (=',buf[j],') > NCOULEUR (=', NCOULEUR,')');
         Halt( 1);
      end
      else
         MontrePixel(j, pos, buf[j]);
   end;
end;

Procedure MontreUneImage;
Var
   mess, mess2 : string[200];
   nc, x1, y1 : integer;
Begin
   SetGraphMode( mode);
   nc := 6;                              { Nombre de couleur par colonne }
   OutTextXY( 0, GetMaxY - (nc+2)*Haut, 'Couleurs :');
   SetTextJustify( LeftText, CenterText);
   For i := 0 to nb_coul do
   begin
      str( i, mess2);
      x1 := 35 * ( i div nc);
      y1 := GetMaxY - nc*Haut + (i mod nc) * Haut;
      OutTextXY(  x1, y1, mess2);
      PutImage( x1 + 24, y1, couleur[i]^, 0);
   end;
   Rectangle( startx - 1, starty - 1, endx + 1, endy + 1);
   for i := 1 to DIMIX do
      for j := 1 to DIMIY do
         PutImage( startx + (i-1)*DIMPX, starty + (j-1)*DIMPY, couleur[ forme[i,j]]^, 0);
End;
```

```
Procedure MarqueurOn;
Begin
   If onpeutafficher then
    PutImage( startx + (x-1)*DIMPX, starty + (y-1)*DIMPY, marqueur^,  1);
End;

Procedure MarqueurOff;
Begin
  If onpeutafficher then
     PutImage( startx + (x-1)*DIMPX, starty + (y-1)*DIMPY, marqueur^,  1);
End;

Procedure AfficheChangementPosition;
Var
   mess, mess2 : string[200];
Begin
   If OnPeutAfficher Then
   Begin
      SetTextJustify( LeftText, CenterText);
      SetColor( Black);
      Str( LastPosX, mess);
      OutTextXY( 50, GetMaxY - 9*Haut, mess);
      SetColor( Blue);
      Str( xpos, mess);
      OutTextXY( 50, GetMaxY - 9*Haut, mess);
      SetColor( Black);
      Str( LastPosY, mess);
      OutTextXY( 50, GetMaxY - 8*Haut, mess);
      SetColor( Blue);
      Str( ypos, mess);
      OutTextXY( 50, GetMaxY - 8*Haut, mess);
      LastPosX := xpos;
      LastPosY := ypos;
   End
End;

Procedure AffichePosition;
Begin
   SetTextJustify( LeftText, CenterText);
   OutTextXY( 0, GetMaxY-10*Haut, 'Position :');
   OutTextXY( 0, GetMaxY-9*Haut, 'X : ');
   OutTextXY( 0, GetMaxY-8*Haut, 'Y : ');
   AfficheChangementPosition;
End;

Procedure EcritUneImage;
VAR
   filename: string[200];
   F: File;
   rec_count, count : word;
Begin
```

```
   AskText( 'Nom du fichier où sera sauvé l''image',NomImage);
   filename := NomImage + ExtImage;
   writeln( 'NOM: ', filename);
   Assign( F, filename);
   Rewrite( F, sizeof(byte));

   writeln('Sizeof(image): ', SizeOf( forme));
   count := DIMIX * DIMIY;
   writeln( 'count: ', count);
   BlockWrite( F, forme, count, rec_count);
   writeln('rec_count: ', rec_count);
   Close( F);
End;

Procedure EcritUneImageR;
VAR
   filename: string[200];
   F: File;
   rec_count, count : word;
Begin
{$I-}
   if demande then
   repeat
      AskText( 'Nom du fichier où sera sauvé l''image',NomImage);
      filename := NomImage + ExtImage;
      writeln( 'NOM: ', filename);
      Assign( F, filename);
      Rewrite( F, sizeof(real));
   until IOResult = 0
{$I+}
   else
   begin
      filename := NomImage + ExtImage;
      writeln( 'NOM: ', filename);
      Assign( F, filename);
      Rewrite( F, sizeof(real));
   end;
   writeln('Sizeof(imageR): ', SizeOf( forme));
   writeln('Sizeof(real): ', SizeOf(real));
   count := DIMIX * DIMIY;
   writeln( 'count:  ', count);
   BlockWrite( F, forme, count, rec_count);
   writeln('rec_count: ', rec_count);
   Close( F);
End;

Procedure LitUneImage;
VAR
   filename: string[200];
   F: File;
   rec_count, count : word;
```

```
Begin
{$I-}
   if demande then
   repeat
      AskText('Nom du fichier où lire l''image',NomImage);
      filename := NomImage + ExtImage;
      writeln( 'NOM: ', filename);
      Assign( F, filename);
      Reset( F, sizeof(byte));
   until (IOResult = 0) OR (NomImage = 'NIL')
{$I+}
   else
   begin
      filename := NomImage + ExtImage;
      writeln( 'NOM: ', filename);
      Assign( F, filename);
      Reset( F, sizeof(byte));
   end;

   if NomImage <> 'NIL' then
   begin
      writeln('Sizeof(IMAGE): ', SizeOf( forme));
      count := DIMIX * DIMIY;
      writeln( 'count: ', count);
      BlockRead( F, forme, count, rec_count);
      writeln('rec_count: ', rec_count);
      Close( F);
   end;
End;

Procedure LitUneImageR;
VAR
   filename: string[200];
   F: File;
   rec_count, count : word;
Begin
{$I-}
   if demande then
   repeat
      AskText( 'Nom du fichier où lire l''image',NomImage);
      filename := NomImage + ExtImage;
      writeln( 'NOM: ', filename);
      Assign( F, filename);
      Reset( F, sizeof(real));
   until IOResult = 0
{$I+}
   else
   begin
      filename := NomImage + ExtImage;
      writeln( 'NOM: ', filename);
      Assign( F, filename);
```

```
      Reset( F, sizeof(real));
   end;
   writeln('Sizeof(IMAGE): ', SizeOf( forme));
   writeln('Sizeof(real): ', SizeOf(real));
   count := DIMIX * DIMIY;
   writeln( 'connt:  ', count);
   BlockRead( F, forme, count, rec_count);
   writeln('rec_count: ', rec_count);
   Close( F);
End;

{ ---------------------- --ASCII ------------------}
Procedure LitUneImageASCII;
VAR
   filename: string;
   F: Text;
   entete: string;
   i,j,k : integer;
   let, inv : char;
Begin
   {$I-}
   repeat
      AskText( 'Nom du fichier où lire l''forme ASCII',NomImage);
      filename := NomImage + '.DAT';
      writeln( 'NOM: ', filename);
      Assign( F, filename);
      Reset( F);
   until IOResult = 0; {$I+}
   repeat
      writeln('(N)ormale ou (I)nversée: ');
      inv := UpCase(Readkey);
   until inv in ['N','I'];

   i := 1;
   if Not Eof( F) then
      readln( F, entete);
   While (Not Eof( F)) AND (i <= DIMIX) do
   begin
      j := 1;
      While (Not Eoln( F)) AND ( j <= DIMIY) do
      begin
         read( F, let);
         case let of
            '0'..'9': forme[i,j] := ord( let) - ord( '0') ;
            'A'..'V': forme[i,j] := ord( let) - ord( 'A') + 9;
         end;   { case }
         if (forme[i,j] > 31) OR (forme[i,j] < 0) then
         begin
            writeln( 'ERREUR ENTREE forme[',i,',',j,']: ',forme[i,j]);
            writeln(' Lettre: ',let);
         end;
```

```
        forme[i,j] := (((forme[i,j]) * nb_coul ) div 32) + 1;
        if inv = 'N' then
           forme[i,j] := nb_coul - forme[i,j] + 1;
        if (forme[i,j] > nb_coul) OR (forme[i,j] < 1) then
           writeln( 'ERREUR CONVERSION forme[',i,',',j,']: ',forme[i,j]);
        j := j+1;
      end;
      readln(F);
      i := i + 1;
   end;
   Close( F);
   for j := 1 to DIMIY do
      for k := 65 to DIMIX do
         forme[k,j]:= 1;
   writeln( 'Fin de lecture de la forme <', entete,'>');
End;


Procedure Unpackc( VAR color, r, g, b : integer);
Begin
   b := color AND $1F;
   g := (color SHR 5) AND $1F;
   r := (color SHR 10) AND $1F;
End;

Procedure LitUneImageTGA;
VAR
   filename: string[200];
   F: File;
   rec_count, count, i, j , k, width, height,
   ximag, lastyimag, yimag, color, r, g, b, intens, Lintens, Hintens,
   diffS, lowS, highS : integer;
   n_id, Entete, entete2: byte;
   LigneTGA : array[1..512] of integer;
Begin
   { ------------------ FICHIER ------------- }
   {$I-}
   repeat
      AskText( 'Nom du fichier où lire l''forme TGA',NomImage);
      filename := NomImage + '.TGA';
      writeln( 'NOM: ', filename);
      Assign( F, filename);
      Reset( F, sizeof(byte));
   until IOResult = 0;
   {$I+}
   { ------------------ SEUILS -------------- }
   err_code := 0;
   lowS := 0;
   HighS := 31;
   write( 'Limite inférieure d''intensité [',lowS,'..',highS,']: ');
   AskInt( '', lowS);
```

```
write( 'Limite Supérieure d''intensité [',lowS,'..',highS,']: ');
AskInt( '', highS);
diffS := highS - lowS + 1;
writeln( lowS:4, highS:4, diffS:4);
if diffs < 0 then
   writeln( 'ERROR');
{ ------------------- ENTETE --------------- }
{ Field 1:   1 byte   : size of field 6 }
BlockRead(F, n_id, 1, rec_count);
if rec_count <> 1 then
   writeln( 'Erreur de lecture');
Writeln('Field 1, n_id: ', n_id);
{ Field 2:   1 byte   : ignored }
BlockRead(F, entete, 1, rec_count);
if rec_count <> 1 then
   writeln( 'Erreur de lecture');
{ Field 3:   1 byte   : ignored }
BlockRead(F, entete, 1, rec_count);
if rec_count <> 1 then
   writeln( 'Erreur de lecture');
{ Field 4:   5 bytes  : ignored }
For i := 1 to 5 do
Begin
   BlockRead(F, entete, 1, rec_count);
   if rec_count <> 1 then
      writeln( 'Erreur de lecture');
End;
{ Field 5:   10 bytes : }
Begin
   { Field 5.1: 2 bytes  : X origin: ignored }
   For i := 1 to 2 do
   Begin
      BlockRead(F, entete, 1, rec_count);
      if rec_count <> 1 then
         writeln( 'Erreur de lecture');
   End;
   { Field 5.2: 2 bytes  : Y origin: ignored }
   For i := 1 to 2 do
   Begin
      BlockRead(F, entete, 1, rec_count);
      if rec_count <> 1 then
         writeln( 'Erreur de lecture');
   End;
   { Field 5.3: 2 bytes  : Width }
   BlockRead(F, entete, 1, rec_count);
   if rec_count <> 1 then
      writeln( 'Erreur de lecture');
   BlockRead(F, entete2, 1, rec_count);
   if rec_count <> 1 then
      writeln( 'Erreur de lecture');
   Width := entete + entete2 * 256;
```

```
   writeln( 'Width: ', width);
   { Field 5.4: 2 bytes  : Heigth }
   BlockRead(F, entete, 1, rec_count);
   if rec_count <> 1 then
      writeln( 'Erreur de lecture');
   BlockRead(F, entete2, 1, rec_count);
   if rec_count <> 1 then
      writeln( 'Erreur de lecture');
   Height := entete + entete2 * 256;
   writeln( 'Height: ', height);
   { Field 5.5: 1 byte  : Pixel Size : Ignored }
   BlockRead(F, entete, 1, rec_count);
   if rec_count <> 1 then
      writeln( 'Erreur de lecture');
   { Field 5.6: 1 byte  : Image Descriptor : Ignored }
   BlockRead(F, entete, 1, rec_count);
   if rec_count <> 1 then
      writeln( 'Erreur de lecture');
End;
{ Field 6: ?? bytes  : Image ID: Ignored }
for i:= 1 to n_id do
Begin
   BlockRead(F, entete, 1, rec_count);
   if rec_count <> 1 then
      writeln( 'Erreur de lecture');
End;
{ Field 7: 0: Ignored }
{ ------------------ IMAGE ----------------- }
{ Field 8: Width * height * 2 bytes: Image }
yimag := 1;
lastyimag := 0;
for i:= DIMIY Downto 1 do
Begin
   yimag := Round( (DIMIY - i) / DIMIY * height);
   writeln( 'I: ',i:3, ' ', yimag);
   for k := lastyimag to yimag do
   Begin
      count := width * sizeof( integer);
      BlockRead( F, LigneTGA, count, rec_count);
      If rec_count <> count then
         writeln( 'Erreur de Lecture');
   End;
   lastyimag := yimag + 1;
   for j := 1 to DIMIX do
   Begin
      ximag := Round( j / DIMIX * width);
      color := LigneTGA[width - ximag];
      color := bswapint( color);
      unpackc( color, r, g, b);
      Hintens := imax( imax( r, g), b);
      Lintens := imin( imin( r, g), b);
```

```
        intens := (Hintens + Lintens + 1) SHR 1;
        if intens < lowS then
           intens := lowS
        else if intens > highS then
           intens := highS;
        color := Round( int( (intens - lowS) / diffS * nb_coul + 1));
        if (color < 1) OR ( color > nb_coul) then
           writeln( 'Erreur Couleur intens:', intens, ' color:', color);
        forme[j,i] := color;
      End;   { For j }
   End;   { For i }
   Close( F);
End;


Procedure EditeUneImage;
Var
   Last_coul, pval, err : word;
   Cle, Cle2 : char;
   messn : string[5];
   messa : string;
Begin
   MontreUneImage(forme);

   SetTextJustify( LeftText, CenterText);

   messa :='Utilisez les cles pour deplacer les marqueurs';
   messa := messa + ' Rentrez le code numerique de la couleur (0-9)';
   OutTextXY( 0, 10 + haut * 0, messa);
   messa := ' Ou <Esc> suivit de l''unite (10-19)  <Enter> pour la meme valeur';
   OutTextXY( 0, 10 + haut * 1, messa);
   messa := 'PgUp PgDn Home et End pour changer la direction, Insert pour la position ';
   OutTextXY( 0, 10 + haut * 2, messa);
   messa := ' Appuyez sur ''q'' pour terminer';
   OutTextXY( 0, 10 + haut * 3, messa);
   SetTextJustify( CenterText, CenterText);
   MarqueurOn( xpos, ypos);
   AffichePosition;
   while NOT keyPressed do
   begin
      MarqueurOff( xpos, ypos);
      MarqueurON( xpos, ypos);
   end;
   last_coul := 0;
   cle := ReadKey;
   while cle <> 'q' do begin
      MarqueurOff( xpos, ypos);
      case cle of
         #0:
            Begin
               cle2 := ReadKey;
```

```
      case cle2 of
        {Insert}
        £82:
        OnPeutAfficher := Not OnPeutAfficher;
        {Home}
          £71:
        Begin
           xinc := -1;
           yinc := 0;
        End;
        {End}
        £79:
        Begin
           xinc := 1;
           yinc := 0;
        End;
        {PgDn}
        £81:
        Begin
           xinc := 0;
           yinc := 1;
        End;
        {PgUp}
        £73:
        Begin
           xinc := 0;
           yinc := -1;
        End;

        {Left}
        £75: xpos := xpos - 1;
        {Right}
        £77: xpos := xpos + 1;
        {Down}
        £80: ypos := ypos + 1;
        {Up}
        £72: ypos := ypos - 1;
      End;
   End;

Else
Begin
   pval := 0;
   if cle = £27 then
   begin
      cle := readkey;
      pval := 10;
   end;
   if cle = £13 then
      j := last_coul
   else
```

```
          begin
            val( cle, j, err);
            j := j + pval;
          end;
          last_coul := j;
          if ((j >= 0) AND (j <= NCOULEUR)) then
          Begin
            MontrePixel(xpos, ypos, j);
            forme[xpos, ypos] := j;
            xpos := xpos + xinc;
            ypos := ypos + yinc;
          end;
        end;
      end;
      if xpos < 1 then begin
        xpos := DIMIX; ypos := ypos - 1;
      End;
      if xpos > DIMIX then begin
        xpos := 1;
        ypos := ypos + 1;
      End;
      if ypos < 1 then begin
        ypos := DIMIY;
        xpos := xpos - 1;
      End;
      if ypos > DIMIY then begin
        ypos := 1;
        xpos := xpos + 1;
      End;
      If xpos < 1 then begin
        xpos := DIMIX;
        ypos := DIMIY;
      END;
      if xpos > DIMIX then begin
        xpos := 1;
        ypos := 1;
      End;
      MarqueurOn( xpos, ypos);
      if (NOT KeyPressed) AND (OnPeutAfficher) Then begin
        AfficheChangementPosition;
        MarqueurOff( xpos, ypos);
        MarqueurOn( xpos, ypos);
      end;
      while NOT keyPressed do
      begin
        MarqueurOff( xpos, ypos);
        MarqueurON( xpos, ypos);
      end;
      Cle := ReadKey;
    End; { WHILE <> £ 13 }
{   ImprimeGraphique;}
```

```
   RestoreCrtMode;
End;

Procedure ImprimeUneRegion;
Var
   r, tx, ty, i, j, d: integer;
   send : byte;
   n1, n2, nby, nbx : integer;
Begin
   writeln( lst);
   writeln( lst);
   write( lst, chr(27), 'A', chr( 8));
   { 12 POINTS D'UN COUP }
   if x2 < x1 then
   begin
      tx := x1;
      x1 := x2;
      x2 := tx;
   end;
   if y2 < y1 then
   begin
      ty := y1;
      y1 := y2;
      y2 := ty;
   end;
   nby := ( y2 - y1 + 1) div 12;
   nbx := ( x2 - x1 + 1);
   n1 := nbx mod 256;
   n2 := nbx div 256;
   for r := 0 to nby do
   begin
      write( lst, CHR(27), '*', CHR(33), CHR( n1), CHR( n2));
      for i := 0 to (nbx - 1) do
      begin
         for j := 0 to 2 do
         begin
            send := 0;
            if GetPixel( x1 + i, y1 + r*12 + j*4 + 3) <> 0 then
               send := 3;
            if GetPixel( x1 + i, y1 + r*12 + j*4 + 2) <> 0 then
               send := send OR 12;
            if GetPixel( x1 + i, y1 + r*12 + j*4 + 1) <> 0 then
               send := send OR 48;
            if GetPixel( x1 + i, y1 + r*12 + j*4) <> 0 then
               send := send OR 192;
            write( lst, chr( send));
         end;
      end;
      writeln( lst);
   end;
end;
```

```
Procedure ImprimeUneImage;
Var
   r, i, j, d: integer;
   send : byte;
   n1, n2, nby, nbx : integer;
Begin
   MontreUneImage( forme);
   writeln( lst);
   writeln( lst);
   write( lst, chr(27), 'A', chr( 8));
   { 12 POINTS D'UN COUP }
   nby := (endy + 3 - starty) div 12;
   nbx := endx + 3 - startx;
   n1 := nbx mod 256;
   n2 := nbx div 256;
   for r := 0 to nby do
   begin
      write( lst, CHR(27), '*', CHR(33), CHR( n1), CHR( n2));
      for i := -1 to (nbx - 2) do
      begin
         for j := 0 to 2 do
         begin
            send := 0;
            if GetPixel( startx + i, starty - 1 + r*12 + j*4 + 3) <> 0 then
               send := 3;
            if GetPixel( startx + i, starty - 1 + r*12 + j*4 + 2) <> 0 then
               send := send OR 12;
            if GetPixel( startx + i, starty - 1 + r*12 + j*4 + 1) <> 0 then
               send := send OR 48;
            if GetPixel( startx + i, starty - 1 + r*12 + j*4) <> 0 then
               send := send OR 192;
            write( lst, chr( send));
         end;
      end;
      writeln( lst);
   end;
   RestoreCrtMode;
end;



Procedure ImprimeGraphique;
Begin
   ImprimeUneRegion( 0, 0, GetMaxX + 1, GetMaxY + 1);
end;


Procedure GereUneImage;
```

```
Var
    choix, i, j, c_in, c_out: integer;
Begin
   choix := 4;
   c_in := 1;
   c_out := 1;
   RestoreCrtMode;
   while choix <> 0 do begin
      writeln;
      writeln( 'Programme d''édition et de création d''formes');
      writeln;
      writeln('   0: Fin');
      writeln('   1: Lire une forme ASCII 64X64 par colonnes');
      writeln('   2: Lire une forme');
      writeln('   3: Ecrire une forme');
      writeln('   4: Editer une forme');
      writeln('   5: Afficher une forme');
      writeln('   6: Imprimer une forme');
      writeln('  11: Remplacer une couleur par une autre');
      writeln;
      choix := 2;
      AskInt( 'Votre choix', choix);
      case choix of
        1: LitUneImageASCII(forme);
        2: LitUneImage(forme, TRUE);
        3: EcritUneImage(forme, TRUE);
        4: EditeUneImage(forme);
        5: Begin
             MontreUneImage(forme);
             Readln;
             RestoreCrtMode;
           end;
        6: ImprimeUneImage(forme);
       11: Begin
             Askint( 'Couleur à remplacer', c_in);
             Askint( 'Par                ', c_out);
             For i := 1 to DIMIX do
               For j := 1 to DIMIY do
                 If forme[i,j] = c_in then
                   forme[i,j] := c_out;
           End;

      End; { case }
   End;   { while }
End;   [ Procedure GereUneImage}

Function hval;
Var
   i, j : integer;
   somme : real;
Begin
```

```
   somme := 0.0;
   for i := -1 to 1 do
      for j := -1 to 1 do
         somme := somme + img[i+xp,j+yp]*h[i,j];
   hval := somme;
End;


Function hRval;
Var
   i, j : integer;
   somme : real;
Begin
   somme := 0.0;
   CheckDebug;
   for i := -1 to 1 do
      for j := -1 to 1 do
      begin
         if do_debug then
            write( '(',i:2,',',j:2,')  I:', img[i+xp,j+yp]:0:2, '  H:',h[i,j]:0:2);
         somme := somme + img[i+xp,j+yp]*h[i,j];
         if do_debug then
            writeln( '  S:', somme:0:2);
      end;
    if do_debug then
      writeln;
   hRval := somme;
End;

   { ------------------- }
   Procedure BruitGaussAdd;
   { ------------------- }
   Var
      i, j, fact, fact2: integer;
      n, diff, som_x, som_x2, ks: real;
   Begin
      ks := sqrt( k);
      randomize;
      for i := 2 to  (DIMIX - 1) do
      begin
         writeln( i);
         for j := 2 to (DIMIY - 1) do
         begin
            diff := rand_norm * ks;
            img[i,j] := img[i,j] + diff;
         end;
      end;
   end;  { BruitGauss Add}

   { ------------------- }
   Procedure BruitGaussMul;
```

```
{ ------------------- }
Var
   i, j, fact, fact2: integer;
   n, diff, som_x, som_x2, ks: real;
Begin
   ks := sqrt( k);
   randomize;
   for i := 2 to  (DIMIX - 1) do
   begin
      writeln( i);
      for j := 2 to (DIMIY - 1) do
      begin
         diff := rand_norm * ks + 1;
         img[i,j] := img[i,j] * diff;
      end;
   end;
end;  { BruitGauss Mul}


{ ------------------ }
Procedure BruitRacine;
{ ------------------ }
Var
   i, j, fact, fact2: integer;
   n, diff, som_x, som_x2, ks: real;
Begin
   ks := sqrt( k);
   randomize;
   for i := 2 to  (DIMIX - 1) do
   begin
      writeln( i);
      for j := 2 to (DIMIY - 1) do
         img[i,j] := SQRT( img[i,j]);
   end;
end;  { Bruit Racine Carrée }

{ ---------------- }
Procedure BruitConv;
{ ---------------- }
Var
  i, j, n: integer;
  mess2, mess: string;
  lr1, lr2: ligneR;          { Pour buffer de lignes }

Begin
   If Demande then
   Begin
      writeln( 'Choix de la matrice: ');
      For i := -1 to 1 do
      Begin
         For j := -1 to 1 do
```

```
            write( '(',i:2,',',j:2,')  ');
          writeln;
        End;
        For i := -1 to 1 do
        begin
          For j := -1 to 1 do
          Begin
            str( i, mess2);
            mess := 'h['+mess2+',';
            str( j, mess2);
            mess := mess + mess2+']';
            AskReal( mess, h[i,j], 2);
          End;
        end;
      End; ( Demande )
      for j := 2 to (DIMIY - 1) do
      begin
        write( j, CHR(13));
        for i := 2 to (DIMIX - 1) do
        begin
          lr1[i] := hRval( img, i, j);
          if do_debug then
            writeln( 'LR1[',i:2,'] := ', lr1[i]:0:2);
        end;
        IF j > 2 then
          FOR m := 2 to DIMIX - 1 DO
            img[m,j-1] := lr2[m];
        lr2 := lr1;
      end;
      writeln;
      FOR m := 2 to DIMIX - 1 DO
        img[m,DIMIY-1] := lr2[m];
    End;

    { ---------------- }
    Procedure BruitUnif;
    { ---------------- }
    Var
      i, j, fact, fact2: integer;
      n, diff, som_x, som_x2, ks: real;
    Begin
      ks := sqrt( k);
      randomize;
      for i := 2 to  (DIMIX - 1) do
        for j := 2 to (DIMIY - 1) do
        begin
          img[i,j] := img[i,j] + (0.5 - random)*ks;
        end;
    end; {BruitUnif}

Procedure BruiteImageR;
```

```
{  Variance = ( n * Σx² - (Σx)² ) / ( n * (n-1) )  }
Var
   choix: integer;
   ks : real;

Begin
   writeln( '1: Bruit uniforme');
   writeln( '2: Bruit Gaussien additif');
   writeln( '3: Bruit Gaussien multiplicatif');
   writeln( '4: Convolution');
   writeln( '5: Racine carrée');
   writeln;
   choix := 2;
   AskInt( 'Votre choix', choix);
   case choix of
      1: BruitUnif( img, k);
      2: BruitGaussAdd( img, k);
      3: BruitGaussMul( img, k);
      4: BruitConv( img, k, TRUE);
      5: BruitRacine( img, k);
   end; { Case }
end;   { Bruite R ---------- }

Procedure BruiteImage;
Var
   choix: integer;
   ks : real;

   Procedure BruitGauss;
   Var
      i, j, fact, fact2: integer;
      n, diff, som_x, som_x2: real;
   Begin
      ks := sqrt( k);
      randomize;
      for i := 2 to  (DIMIX - 1) do
      begin
         writeln( i);
         for j := 2 to (DIMIY - 1) do
         begin
            diff := rand_norm * ks;
            img[i,j] := imax( 1, imin( Round( img[i,j] + diff), nb_coul));
         end;
      end;
   end;  { BruitGauss }

   Procedure BruitUnif;
   Var
      i, j, fact, fact2: integer;
      n, diff, som_x, som_x2: real;
   Begin
```

```
      ks := sqrt( k);
      randomize;
      for i := 2 to  (DIMIX - 1) do
         for j := 2 to (DIMIY - 1) do
         begin
            img[i,j] := imax( 1, imin( Round( img[i,j] + (0.5 - random)*ks), nb_coul));
         end;
   end;  {BruitUnif}
Begin
   writeln( '1: Bruit uniforme');
   writeln( '2: Bruit Gaussien');
   writeln;
   choix := 2;
   Askint( 'Votre choix', choix);
   case choix of
      1: BruitUnif;
      2: BruitGauss;
   end; { Case }
end;   { Bruite ---------- }

Function FTauxErreur;
{ TROUVE LE TAUX D'ERREUR  }
{  Variance = ( n * Σx² - (Σx)² ) / ( n * (n-1) )  }
Var
   diff, i, j: integer;
   n : real;
   faux : real;
   som_x, som_x2: real;
Begin
   som_x := 0; som_x2 := 0;
   taux := 0;
   for i := 2 to (DIMIX-1) do
   begin
      for j:=2 to (DIMIY-1) do
      begin
         diff := imN[i,j] - imOK[i,j];
         if doerr then
            imERR[i,j] := ABS( diff);
         if diff <> 0 then
            taux := taux + 1;
         som_x := som_x + diff;
         som_x2 := som_x2 + SQR(diff);
      end;
   end;
   n := 1.0 * (DIMIX-2) * (DIMIY-2);
   k2 := (n * som_x2 - som_x * som_x) / ( n * (n-1));
   FTauxErreur := taux / n * 100;
End;    { FTaux ..... }

Function TauxErreur;
{ TROUVE LE TAUX D'ERREUR  }
```

```
{  Variance = ( n * Σx² - (Σx)² ) / ( n * (n-1) )  }
Var
   i, j: integer;
   n : real;
   Taux : real;
   diff, som_x, som_x2: real;
Begin
   som_x := 0; som_x2 := 0;
   taux := 0;
   for i := 2 to (DIMIX-1) do
   begin
      for j:=2 to (DIMIY-1) do
      begin
         diff := imN[i,j] - imOK[i,j];
         if diff <> 0 then
            taux := taux + 1;
         som_x := som_x + diff;
         som_x2 := som_x2 + diff * diff;
      end;
   end;
   n := 1.0 * (DIMIX-2) * (DIMIY-2);
   k2 := (n * som_x2 - som_x * som_x) / ( n * (n-1));
   TauxErreur := taux / n * 100;
End;    { Taux ..... }

Procedure ImgBuf;
{ COPIE UNE LIGNE D'IMAGE DANS UN BUFFER }
Var
   i:integer;
Begin
   For i := 2 to DIMIX -1 do
      buf[i]:=img[i,pos];
end; { IMGBUF }

Procedure BufImg;
{ COPIE UNE LIGNE DE BUFFER DANS UNE IMAGE }
Var
   j:integer;
Begin
   If DoIterGraph then
        MontreUneLigne( img, buf, pos);
   For j := 2 to DIMIX-1 do
      img[j,pos]:=Buf[j];
end; { BUFIMG }

Procedure BordeImage;
Var
   i:integer;
Begin
   For i := 1 to DIMIX do
      im[i,1] := coul;
```

```
   For i := 1 to DIMIX do
      im[i,DIMIY] := coul;
   For i := 1 to DIMIY do
      im[1,i] := coul;
   For i := 1 to DIMIY do
      im[DIMIX,i] := coul;
end;    { ------ BordelImage ---------}


Procedure ShowImage;
Begin
   if img <> NIL then
   begin
      SetGraphMode( mode);
      PutImage( 0, 0, img^, 0);
   end
   else
      MontreUneImage( img);
End;

Procedure SaveImage;
Begin
   if img = NIL then
      GetMem( img, ImageSize( 0, 0, GetMaxX, GetMaxY));
   GetImage( 0, 0, GetMaxX, GetMaxY, img^)
End;


{-----------------------------------------
  Partie Initialisation du module (UNIT) Imag
-----------------------------------------}
var
   path: string[20];
   Error, k,l,c, x1, x2, y1, y2: integer;
Begin
   nb_coul := NCOULEUR;
   NomImage := 'Monala';
   Driver := Detect;
   path := 'c:\turbop';
   Initgraph ( Driver, mode, path);
   Error := GraphResult;
   if Error <> grOK then
   begin
      writeln('Erreur ',Error,' d''initialisation graphique');
      writeln('Path: ', path);
      halt( 1);
   end;
   SetGraphMode( mode);
   { Dessine les couleurs }
   x1 := 100 - (DIMPX div 2) + 1;
   x2 := 100 + (DIMPX div 2);
```

```
y1 := 100 - (DIMPY div 2) + 1;
y2 := 100 + (DIMPY div 2);
for i := 0 to NCOULEUR DO
   GetMem( couleur[i], ImageSize( x1, y1, x2, y2));
marqueur := couleur[NCOULEUR];
GetImage( x1, y1, x2, y2, couleur[0]^);
FOR c := 1 to NCOULEUR DO
Begin
   PutPixel( x1 -1 + xcode_pix[c], y1 - 1 + ycode_pix[c], Blue);
   GetImage( x1, y1, x2, y2, couleur[c]^);
END;

SetUserCharSize( 1,2,5,16);
SetTextStyle( 1, 0, 0);
haut := TextHeight( 'H') + 1;
RestoreCrtMode;
xpos := DIMIX DIV 2;
ypos := DIMIY DIV 2;
LastPosX := xpos;
LastPosY := ypos;
xinc := 1;
yinc := 0;
startx := (GetMaxX - TOTX) div 2;
endx := startx + TOTX - 1;
starty := (GetMaxY - TOTY) div 2;
endy := starty + TOTY - 1;
OnPeutAfficher := True;
for l := -1 to 1 do
   for k := -1 to 1 do
      h[l,k] := 1.0/16;
h[0,0] := 1/2;
do_debug := FALSE;
{
writeln( 'TOTX:    ', TOTX);
writeln( 'TOTY:    ', TOTY);
writeln( 'DIMIX:   ', DIMIX);
writeln( 'DIMIY:   ', DIMIY);
writeln( 'DIMPX:   ', DIMPX);
writeln( 'DIMPY:   ', DIMPY);
writeln( 'xpos:    ', xpos);
writeln( 'ypos:    ', ypos);
writeln( 'LastPosx:', Lastposx);
writeln( 'LastPosy:', Lastposy);
writeln( 'startx:  ', startx);
writeln( 'starty:  ', starty);
writeln( 'GetMaxX: ', GetMaxX);
writeln( 'GetMaxY: ', GetMaxY);
writeln( 'endx:    ', endx);
writeln( 'endy:    ', endy);
}
for i := 0 to NCOULEUR do
```

```
   begin
      oldcouleur[i] := couleur[i];
      choixcouleur[i] := i;
   end;
   choixcouleur[0] := 0;
   couleur[0] := oldcouleur[choixcouleur[0]];
   choixcouleur[1] := 0;
   couleur[1] := oldcouleur[choixcouleur[1]];
   choixcouleur[2] := 1;
   couleur[2] := oldcouleur[choixcouleur[2]];
   choixcouleur[3] := 4;
   couleur[3] := oldcouleur[choixcouleur[3]];
   choixcouleur[4] := 16;
   couleur[4] := oldcouleur[choixcouleur[4]];
End.
```

TIMAGE.PAS

```
Program TestImage;
USES
   Graph, imag;
Var
   x:image;
Begin
   write( 'Nombre de couleur [0..',NCOULEUR,']: ');
   readln( nb_coul);
   InitImage( x);
   GereUneImage(x);
   CloseGraph;
End.
```

TAUX.PAS

```
Program Taux_d_erreurs;
{
        Jean-Michel Durocher, 29 juillet 1989.
        Essais de test pour définir des taux d'erreurs.
}
USES
   Crt, Graph, Std, Imag;
Type
   P_image = ^image;
Var
   imOK, imDEG, imFIL1, imFIL2, imERR:image;
   imY : imageR;
   i,j,choix, choix2,ichoix: integer;
   nomOK, nom2, nomFIL1, nomFIL2, nomERR, nomDEG:string;
   im_p, im_p2 : P_image;
   k, err1, err2, taux : real;
```

```
  doerr, has_convol: boolean;
  h1, h2: PSF;

  FUNCTION ErreurPoids( VAR imDEG, imOK, imFIL:image; nombre, affiche:boolean):real;
  Var
    tot_poids, err : real;
    i, j : integer;
    nb_err : longint;
  Begin
    nb_err := 0;
    err := 0;
    tot_poids := 0;
    For i := 2 to (DIMIX-1) DO
    Begin
      For j := 2 to (DIMIY-1) DO
      Begin
        tot_poids := tot_poids + imFIL[i,j];
        IF imDEG[i,j] <> imOK[i,j] then
        begin
          nb_err := nb_err + 1;
          if nombre then
            err := err+imFIL[i,j]
          else
            err := err + abs(imDEG[i,j] - imOK[i,j])*imFIL[i,j];
          If Affiche then
            writeln( 'Erreur: I:',i, ' J:', j,' DEG:',imDEG[i,j], ' OK:', imOK[i,j],' POIDS:',
imFIL[i,j]);
        end;
      End;
    End;
    if Not Nombre then
      err := err / nb_coul;
    writeln( 'Tot_poids:', tot_poids:0:1,' Tot_erreur:', err:0:1,' Nerreur:', nb_err);
    err := err / tot_poids * 100;
    ErreurPoids := err;
  End; { ERREURPOIDS }

  Procedure Taux1;
  Var
    Doerr:boolean;
  Begin
    doerr := TRUE;
    AskBool( 'Création image d''erreurs', doerr);
    taux := FTauxErreur( imOK, imDEG, imERR, doerr, k);
    writeln( 'Taux d''erreur: ', taux:6:2,' %');
    writeln( 'σ²: ', k:0:3,' -> ', (k/nb_coul*100):0:2,' %');
  End; { TAUX1 }

  Function all_eq( Var imOK, imDEG:image; x, y:integer):boolean;
  Var
    i, j: integer;
```

```
   OK : boolean;
Begin
   OK := TRUE;
   For i:= -1 to 1 do
      For j := -1 to 1 do
      begin
         if imOK[x+i,y+j] <> imDEG[x+i,y+j] then
            OK := FALSE;
      end;
   all_eq := OK;
End;


Procedure Taux2;
Var
   i, j:integer;
   tot_case, nok_count : longint;
   taux : real;
Begin
   i := 3;
   tot_case := 0;
   nok_count := 0;
   repeat
      j := 3;
      repeat
         if NOT all_eq( imOK, imDEG, i, j) then
            nok_count := nok_count + 1;
         tot_case := tot_case + 1;
         j := j + 3;
      until j >= DIMIY;
      i := i + 3;
   until i >= DIMIX;
   taux := 1.0 * nok_count / tot_case * 100;
   writeln( '$total de cases:', tot_case,'  $cases pas OK: ', nok_count);
   writeln( 'Taux:', taux:0:2, ' %');
End; { TAUX2 }

Procedure taux3;
Var
   nombre, affiche:boolean;
Begin
   nombre := TRUE;
   Affiche := FALSE;
   AskBool('NOMBRE (par opposition à valeurs)', nombre);
   AskBool( 'Affiche les valeurs fausses', affiche);
   IF NOT has_convol then
   begin
      writeln( 'Copie image pour 1ère convolution');
      CopieImageR( imOK, imY);
      writeln( 'Convolution avec les poids:');
      h := h1;
```

```
      For i := -1 to 1 do
      Begin
         For j := -1 to 1 do
            write( h[i,j]:7:3);
         writeln;
      End;
      BruitConv( imY, k, FALSE);
      CopieRImage(imY, imFIL1);
   End;
   err1 := ErreurPoids( imDEG, imOK, imFIL1, nombre, affiche);
   writeln( 'lère erreur (poids sur les pixels intérieurs):', err1:0:3, ' %');
   writeln;

   IF NOT has_convol then
   Begin
      writeln( 'Copie image pour 2ème convolution');
      CopieImageR( imOK, imY);
      writeln( 'Convolution avec les poids:');
      h := h2;
      For i := -1 to 1 do
      Begin
         For j := -1 to 1 do
            write( h[i,j]:7:3);
         writeln;
      End;
      BruitConv( imY, k, FALSE);
      CopieRImage(imY, imFIL2);
      has_convol := TRUE;
   end;
   err2 := ErreurPoids( imDEG, imOK, imFIL2, nombre, affiche);
   writeln( '2ème erreur (poids sur les pixels extérieurs):', err2:0:3, ' %');

   writeln( 'moyenne erreur:', ((err1+err2)/2):0:3,' %');
   writeln;
End; { TAUX3 }

Procedure ChangeParam;
Begin
   repeat
      AskInt( 'nombre de couleur',nb_coul);
   until nb_coul IN [0..NCOULEUR];
   for i := 0 to nb_coul do
   begin
      write( 'couleur ', i:2);
      AskInt( ' ', choixcouleur[i]);
      couleur[i] := oldcouleur[choixcouleur[i]];
   end;
End; { CHANGEPARAM }

Procedure Menu6;
Begin
```

```
   writeln( '       1: % de pixels différents et variance (plus image ERREUR)');
   writeln( '       2: % de cases de dimensions 3x3 reconstituées');
   writeln( '       3: % de pixels différents avec poids pour les contours');
   writeln( '          donnés par Laplacien');
End; { MENU6 }

Procedure Menu;
Begin
   writeln;
   writeln( 'Programme de comparaisons pour taux d''erreurs');
   writeln;
   writeln( '   0: Fin');
   writeln( '   1: Lire une image du disque');
   writeln( '   2: Ecrire une image sur disque');
   writeln( '   3: Affiche une image');
   writeln( '   4: Ecrit les valeurs d''une image');
   writeln( '   5: Convolution sur image');
   writeln( '   6: taux d''erreurs');
   menu6;
   writeln( '  11: Ajustement des paramètres');
   writeln( '  33: Imprime une image');
   writeln( '  44: Imprime les valeurs d''une image');
   writeln( '  99: MENU');
End; { MENU }

FUNCTION choix_im( Var ichoix:integer):P_image;
Var
   im_ch:P_image;
Begin
   writeln( '1: Image originale');
   writeln( '2: Image DEGRADEE');
   writeln( '3: Image FILTRE 1');
   writeln( '4: Image FILTRE 2');
   writeln( '5: Image ERREUR');
   writeln;
   AskInt( 'Votre choix d''image', ichoix);
   im_ch := NIL;
   case ichoix of
      1:
      begin
         Nom2 := NomOK;
         im_ch := @imOK;
      end;
      2:
      begin
         Nom2 := NomDEG;
         im_ch := @imDEG;
      end;
      3:
      begin
         Nom2 := NomFIL1;
```

```
            im_ch := @imFIL1;
          end;
          4:
          begin
            Nom2 := NomFIL2;
            im_ch := @imFIL2;
          end;
          5:
          begin
            Nom2 := NomERR;
            im_ch := @imERR;
          end;
      end; { CASE ichoix }
      NomImage := Nom2;
      choix_im := im_ch;
   End; { CHOIX_IM }

   Procedure Nouveau_nom;
   Begin
      If Nom2 = NomFIL1 then
         nomFIL1 := NomImage
      else If Nom2 = NomFIL2 then
         nomFIL2 := NomImage
      else If Nom2 = NomDEG then
         nomDEG := NomImage
      else If Nom2 = NomOK then
         nomOK := NomImage
      else If Nom2 = NomERR then
         nomERR := NomImage;
   End; { NOUVEAU_NOM }

Begin
   h1[-1,-1] := -1/6;
   h1[-1, 0] := -4/6;
   h1[-1, 1] := -1/6;
   h1[ 0,-1] := -4/6;
   h1[ 0, 0] :=  20/6;
   h1[ 0, 1] := -4/6;
   h1[ 1,-1] := -1/6;
   h1[ 1, 0] := -4/6;
   h1[ 1, 1] := -1/6;
   For i := -1 to 1 do
      For j := -1 to 1 do
         h2[i,j] := -h1[i,j];
   has_convol := FALSE;
   write( 'Nombre de couleur <0..',NCOULEUR,'>');
   nb_coul := 4;
   Askint( ' ', nb_coul);
   nomOK := 'BB';
   nomDEG := 'BBD';
   nomFIL1 := nomOK+'F1';
```

```
nomFIL2 := nomOK+'F2';
nomERR := nomOK+'%';
InitImage( imOK);
InitImage( imDEG);
InitImage( imFIL1);
InitImage( imFIL2);
InitImage( imERR);
BordeImage( ImERR, 0);
choix := 1;
menu;
while choix <> 0 do
begin
  writeln;
  AskInt( 'Votre choix', choix);
  if choix <> 0 then begin
    case choix of
      99: menu;
      1:
      Begin
        writeln( 'LECTURE');
        ichoix := 1;
        im_p := choix_im( ichoix);
        if im_p <> NIL then
        Begin
          LitUneImage( im_p^, TRUE);
          BordeImage( Im_p^, 0);
          nouveau_nom;
          if ichoix = 1 then
          begin
            has_convol := FALSE;
            nomFIL1 := nomOK+'F1';
            nomFIL2 := nomOK+'F2';
            nomERR := nomOK+'%';
          end;
          if ichoix IN [3,4] then
            has_convol := TRUE;
        end;
      end; { 1 }
      2:
      Begin
        writeln( 'ECRITURE');
        ichoix := 1;
        im_p := choix_im( ichoix);
        if im_p <> NIL then
        begin
          EcritUneImage( im_p^, TRUE);
          nouveau_nom;
        end;
      end; { 2 }
      3:
      Begin
```

```
   writeln( 'AFFICHAGE');
   ichoix := 1;
   im_p := choix_im( ichoix);
   if im_p <> NIL then
   Begin
      MontreUneImage( im_P^);
      readln;
      RestoreCrtMode;
      If ichoix = 5 then
         writeln( 'Attention aux codes de couleurs pour l''erreur ... Au départ, 1 = 0');
   end;
end; { 3 }
4:
Begin
   writeln( 'AFFICHE VALEURS');
   ichoix := 1;
   im_p := choix_im( ichoix);
   if im_p <> NIL then
      printimg( im_P^);
end; { 4 }
5:
Begin
   writeln( 'CONVOLUTION, image de départ ?');
   ichoix := 1;
   im_p := choix_im( ichoix);
   writeln( 'CONVOLUTION, image d''arivée ?');
   ichoix := 3;
   im_p2 := choix_im( ichoix);
   IF (im_p <> NIL) AND (im_p2 <> NIL) then
   Begin
      if ichoix = 3 then
         h := h1
      else if ichoix = 4 then
         h := h2;
      CopieImageR( im_p^, imY);
      BruitConv( imY, k, TRUE);
      CopieRImage(imY, im_p2^);
      if ichoix = 3 then
         h1 := h
      else if ichoix = 4 then
         h2 := h;
      If ichoix IN [3,4] then
         has_convol := TRUE;
   end;
end; { 5 }
6:
Begin
   writeln( 'TAUX D''ERREUR');
   menu6;
   choix2 := 3;
   AskInt( 'Votre choix', choix2);
```

```
            Case choix2 of
               1: taux1;
               2: taux2;
               3: taux3;
            End; { Case }
         End; { 6 }
         11: ChangeParam;
         33:
         Begin
            writeln( 'IMPRESSION');
            ichoix := 1;
            im_p := choix_im( ichoix);
            if im_p <> NIL then
               ImprimeUneImage( im_p^);
         end; { 33 }
         44:
         Begin
            writeln( 'IMPRIME VALEURS');
            ichoix := 1;
            im_p := choix_im( ichoix);
            if im_p <> NIL then
               pprinting( im_P^);
         end; { 4 }

      end; { CASE choix }
   end;  { Choix <> 0 }
end; { while }
CloseGraph;
End.
```