TABLE DES MATIERES

CHAPITRE 1

NOTIONS SUR LE TRAITEMENT D'IMAGES

CHAPITRE 2

METHODES DE COMPRESSION

CHAPITRE 3

COMPRESSION DES IMAGES FIXES BASEE SUR UNE TRANSFORMATION FRACTALE

CHAPITRE 4

ARCHITECTURES PARALLELES ET TRAITEMENT D'IMAGE

CHAPITRE 5

PARALLELISATION D'UN ALGORITHME DE COMPRESSION FRACTALEET IMPLEMENTATION

PARTIE A

Parallélisation d'un algorithme de compression fractale

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES ACRONYMES

RVB	:	Mode Rouge, Ver, Bleu.
GIF	:	Graphic Interchange Format
JPEC	:	Joint Photographique Experts Group
MSE	:	Mean Square Error
SNR	:	Signal To Noise Ratio
PSNR	:	Peak Signal To Noise Ratio
DCT	:	Discrète Cosine Transform
RLC	:	Run Length Coding
LZW	:	Lempel-Ziv-Welch
QV	:	Quantification Vectorielle
QS	:	Quantification Scalaire
IFS	:	Iterated Function System
PIFS	:	Partitioned Iterated Function System
R_i	:	Bloc Destination Numéro I
D_i	:	Bloc Source Numéro I
SISD	:	Single Instruction Single Data
SIMD	:	Single Instruction Multiple Data
MISD	:	Multiple Instructions Single Data
MIMD	:	Multiple Instruction Multiple Data
VLSI	:	Very Large Scale Intergrated
FPFIC	:	Fixed-Size Partitioning Fractal Image Compression
FCM	:	Fast Comparison Module
PE	:	Processing Element
QPFIC	:	Quadtree-Based Partitioning Fractal Image Compression
MAD	:	Mean Absolute Difference

INTRODUCTION GENERALE

Récemment, on a remarqué une croissance en terme de besoin de stockage et/ou de transmission des informations visuelles. Les diverses applications telles que la télécopie, la vidéo-conférence, l'imagerie médicale et satellitaire, la télévision haute définition, la télésurveillance, et les services d'informations sur l'Internet sont basées sur la fiabilité de sauvegarder et de transmettre les images. Le stockage des images sur les disques durs des ordinateurs grand public ainsi que l'utilisation des technologies numériques pour le traitement et la retouche des images nécessitent d'acquérir les images sous format numérique (encore appelé format électronique).

Le format numérique des images est extrêmement coûteux en taille mémoire, bien qu'il soit le plus adapté aux applications citées plus haut. Pour résoudre ce problème de coût qui peut limité la faisabilité de stockage et de transmission des images ; des techniques de compression d'images ont été élaborées pour compacter leur représentation numérique. A l'aide de ces techniques de compression, le stockage et la transmission des images seront plus efficaces et plus rapides.

Comprimer les données (images) revient tout simplement à en éliminer toute information superflue. On parle alors de réduction de la redondance. La redondance peut être occasionnée par une représentation non efficace des données ; son élimination dans ce cas, n'empêche pas la restitution de ces données, et elle se fait à l'aide des méthodes de compression dites réversibles. En effet, les images comportent des informations non pertinentes dont l'élimination ne nuis pas au message perçu. Dans ce cas, la restitution exacte des données n'est pas possible, et on parle des méthodes de compression irréversibles [1]. Nous nous intéresserons dans le cadre de cette thèse à l'une des méthodes de compression irréversibles qui est la compression d'image selon une approche fractale.

La compression d'images par fractales est une technique relativement récente puisqu'elle a été proposée pour la première fois par *M. Barnsley,* en 1988. Il s'agissait au départ de coder des formes fractales à l'aide d'un nombre très réduit de coefficients. La théorie des *IFS* (*Iterated Function System*) permet en effet de générer des images fractales simplement décrites par un ensemble de transformations affines contractantes. Un théorème a en suite été démontré par *M. Barnsley,* permettant de résoudre le problème inverse [1]. Le codage des images réelles

nécessite une adaptation de la théorie des *IFS*. Une possibilité, est d'exploiter les similarités locales présente dans l'image. *A. Jacquin* fut le premier en 1989, à présenter un algorithme automatique de compression d'images réelles par fractales [2]. Celui-ci consiste à partitionner l'image, et à mettre chacun des blocs carrés obtenus en correspondance avec un autre bloc de taille supérieure, recherché n'importe où dans l'image. On parle dans ce cas *d'IFS* local, désigné sous le terme *L-IFS* (*PIFS*). Divers travaux ont montrés que cette méthode possède un potentiel lui permettant de figurer parmi les méthodes de compression efficaces, dont le principal avantage est :

- Des taux de compression élevés.
- Une représentation indépendante de l'échelle.
- Une procédure de décompression particulièrement rapide et simple.

L'inconvénient principal de cette technique est le temps très élevé pour déterminer les codes *IFS* de l'image compressée. En effet, plusieurs travaux de recherches dans ce domaine se penchés sur deux axes, l'axe de partitionnement de l'image à savoir le quadtree, *HV*, *Delaunay* et autre, ainsi que l'accélération du codage avec l'élaboration des shemas hybrides et de parallélisation d'algorithmes. Cette recherche peut être implantée naïvement, avec une complexité linéaire; auquel cas la complexité globale est quadratique en fonction de taille de l'image. Mise en œuvre de cette manière, la compression d'une image de taille moyenne peut prendre quelques heures sur un ordinateur personnel [3]. A cause de ce problème, plusieurs techniques sont utilisées pour améliorer le temps de compression.

Dans notre travail, nous allons introduire le parallélisme dans la compression fractale d'images fixes, ce qui permet de minimiser le temps de codage d'une manière intéressante en implémentant un environnement parallèle de compression avec cette méthode dont l'architecture utilisée est *MIMD* à mémoire distribuée qui est simulée par un réseau.

L'organisation générale du mémoire est décrite ci-dessous :

Le chapitre 1 est consacré à quelques concepts sur l'image, les besoins en compression d'images, et les éléments fondamentaux pour mesurer la qualité de l'image compressée.

Le chapitre 2 est consacré à la présentation des principales méthodes de compression réversibles et irréversibles des images fixes.

Le chapitre 3 introduit les notions nécessaires à la compréhension de la théorie des systèmes des fonctions itérées (*IFS*), et présente le principe de l'algorithme de compression des images naturelles selon l'approche fractale sur des blocs de pixels. Ainsi que les différents types de partitionnements déjà utilisés en compression d'images par fractales.

Le chapitre 4 introduit quelques notions concernant le parallélisme et les architectures parallèles.

Le chapitre 5 est consacré à la notion de parallélisation de l'algorithme de compression fractale. Ainsi que son implémentation suivant une nouvelle approche (proposée) et les résultats obtenus avec une discussion de ces derniers.

[1] Barnsley M.F., Hurd L.P., Fractal Image Compression, AK Peters, Ltd., Wellesley, 1993.

[2] Jacquin A.E., "Image Coding Based on a Fractal Theory of Iterated Contractive Image Transformations", IEEE transactions on image processing, Vol. 1, No. 1, pp. 18-30, 1992.

[3] C.Jean " compression fractale d'images " L'ULB 4 octobre 2001

1 INTRODUCTION

L'image constitue l'un des moyens les plus importants qu'utilise l'homme pour communiquer avec autrui. C'est un moyen de communication universel dont la richesse du contenu permet aux êtres humains de tout âge et de toute culture de se comprendre.

C'est aussi le moyen le plus efficace pour communiquer, chacun peut analyser l'image à sa manière, pour en dégager une impression et d'en extraire des informations précises.

De ce fait, le traitement d'image est un ensemble des méthodes et techniques opérant sur l'image, dont le but est d'améliorer son aspect visuel. Il se définit comme un ensemble de taches destinées à extraire de l'image des informations qualitatives et quantitatives.

Nous allons dans ce chapitre parler de quelques concepts de l'image puis nous abordons les besoins en compression d'images, et par la suite nous présentons les éléments fondamentaux pour mesurer la qualité de l'image compressée.

2 DEFINITION DE L'IMAGE

L'image est une représentation d'une personne ou d'un objet par la peinture, le dessin, la photographie, le film, etc. C'est aussi un ensemble structuré d'informations, qui après affichage sur l'écran, ont une signification pour l'oeil humain [4].

Elle peut être décrite sous la forme d'une fonction $I(x, y)$ de brillance analogique continue, définie dans un domaine borné, tel que x et y sont les coordonnées spatiales d'un point de l'image et I est une fonction d'intensité lumineuse et de couleur. Sous cet aspect, l'image est inexploitable par la machine, ce qui nécessite sa numérisation [5].

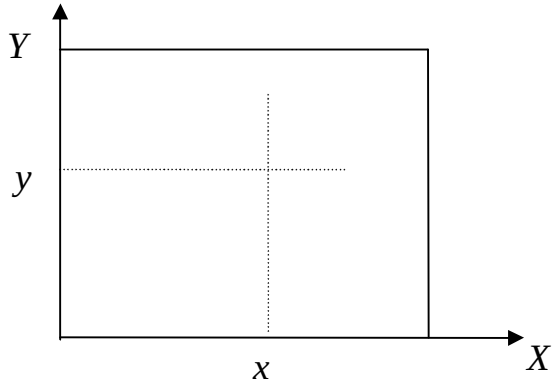


Figure 1.1 : Un point de l'image de coordonnées (*x*, *y*)

3 IMAGE NUMERIQUE

Contrairement aux images obtenues à l'aide d'un appareil photo, ou dessinées sur du papier, les images manipulées par un ordinateur sont numériques (représentées par une série de bits) [5]. L'image numérique est l'image dont la surface est divisée en éléments de tailles fixes appelés cellules ou pixels, ayant chacun comme caractéristique un niveau de gris ou de couleur prélevé à l'emplacement correspondant dans l'image réelle, ou calculé à partir d'une description interne de la scène à représenter [6].

La numérisation d'une image est la conversion de celle-ci de son état analogique (distribution continue d'intensités lumineuses dans un plan $x * y$) en une image numérique représentée par une matrice bidimensionnelle de valeurs numériques $I(x, y)$ où: x, y : coordonnées cartésiennes d'un point de l'image. $I(x, y)$: niveau de gris en ce point.

Pour des raisons de commodité de représentation pour l'affichage et l'adressage, les données images sont généralement rangées sous formes de tableau I de n lignes et p colonnes. Chaque élément $I(x, y)$ représente un pixel de l'image et à sa valeur est associé un niveau de gris codé sur m bits (2^m niveau de gris ; 0 = noir ; 2^m-1 = blanc). La valeur en chaque point exprime la mesure d'intensité lumineuse perçue par le capteur [5].

4 LES ATTRIBUTS DE LIMAGE

L'image est un ensemble structuré d'informations caractérisé par les paramètres suivants:

4.1 Pixel

Contraction de l'expression anglaise "*Picture éléments*": éléments d'image, le pixel est le plus petit point de l'image, c'est une entité calculable qui peut recevoir une structure et une quantification. Si le bit est la plus petite unité d'information que peut traiter un ordinateur, le pixel est le plus petit élément que peuvent manipuler les matériels et logiciels d'affichage ou d'impression. La lettre *A*, par exemple, peut être affichée comme un groupe de pixels dans la figure ci-dessous (figure 1.2) :

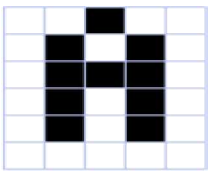


Figure 1.2 : Représentation de la lettre *A* sous la forme d'un groupe de pixels

La quantité d'information que véhicule chaque pixel donne des nuances entre images monochromes et images couleurs. Dans le cas d'une image monochrome, chaque pixel est codé sur un octet, et la taille mémoire nécessaire pour afficher une telle image est directement liée à la taille de l'image.

Dans une image couleur (*R.V.B.*), un pixel peut être représenté sur trois octets : un octet pour chacune des couleurs : rouge (*R*), vert (*V*) et bleu (*B*) [4].

4.2 Dimension (définition)

On appelle définition, le nombre de points (pixel) constituant l'image, c.a.d sa (dimension informatique). Cette dernière se présente sous forme de matrice dont les éléments sont des valeurs numériques représentatives des intensités lumineuses (pixels). Le nombre de lignes de cette matrice multiplié par le nombre de colonnes nous donne le nombre total de pixels dans une image [4].

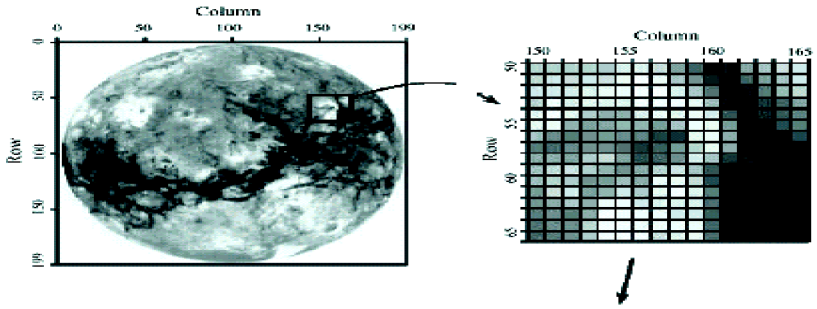


Figure 1.3 : Représentation de dimension d'une image

4.3 Bruit

C'est un signal qui lors de l'acquisition ou la transmission vient s'ajouter à l'image, Il se matérialise par la présence dans une région homogène des valeurs plus ou moins éloignées de l'intensité de la région. Le bruit est le résultat de certains défauts électroniques du capteur et de la qualité de numérisation.

Image avec bruit *Image sans bruit*

Figure 1.4 : Image avec et sans bruit

4.4 Résolution

C'est la clarté ou la finesse de détails atteinte par un moniteur dans la production d'images. Sur les moniteurs d'ordinateurs, la résolution est exprimée en nombre de pixels par unité de mesure (pouce ou centimètre). On utilise aussi le mot résolution pour désigner le nombre total de pixels affichables horizontalement ou verticalement sur un moniteur; plus grand est ce nombre, meilleure est la résolution [4].

4.5 Histogramme

L'histogramme des niveaux de gris ou des couleurs d'une image est une fonction qui donne la fréquence d'apparition de chaque niveau de gris (couleur) dans l'image (figure1.5). Pour diminuer l'erreur de quantification, pour comparer deux images obtenues sous des éclairages différents, ou encore pour mesurer certaines propriétés sur une image, on modifie souvent l'histogramme correspondant [4].

Il permet de donner un grand nombre d'informations sur la distribution des niveaux de gris (couleur) et de voir entre quelles bornes est répartie la majorité des niveaux de gris (couleur) dans les cas d'une image trop claire ou d'une image trop foncée.

Il peut être utilisé pour améliorer la qualité d'une image (Rehaussement d'image) en introduisant quelques modifications, pour pouvoir extraire les informations utiles de celle-ci.

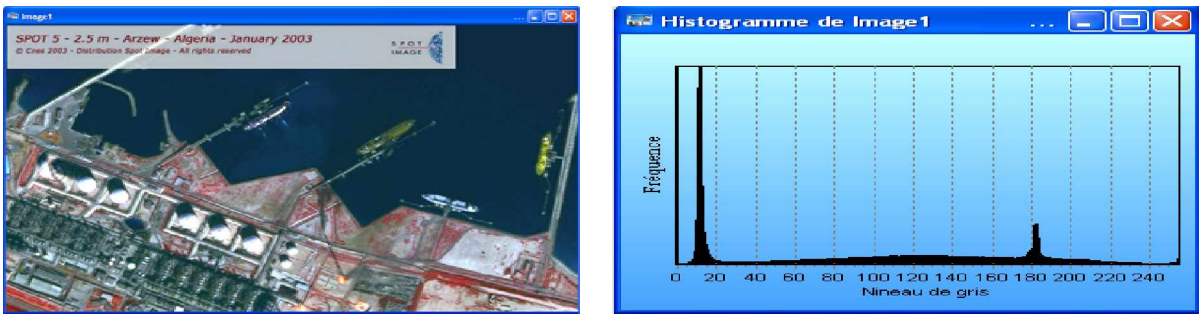


Figure 1.5 : Image et histogramme Associés

Algorithme

1. Balayer l'image

2. Incrémenter le niveau de gris pour chaque pixel.

4.6 Contours et textures

Les contours représentent la frontière entre les objets de l'image, ou la limite entre deux pixels dont les niveaux de gris représentent une différence significative [4]**.** Les textures décrivent la structure de ceux-ci. L'extraction de contour consiste à identifier dans l'image les points qui séparent deux textures différentes (figure 1.6).

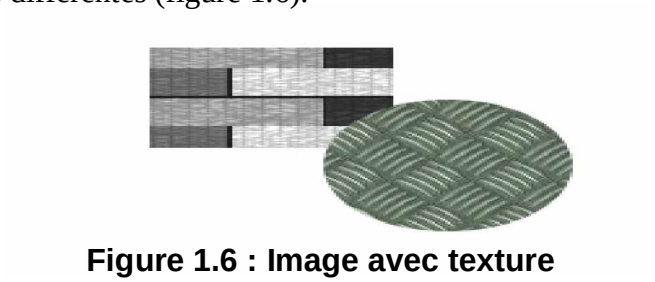


Figure 1.6 : Image avec texture

4.7 Luminance

C'est le degré de luminosité des points de l'image. Elle est définie aussi comme étant le quotient de l'intensité lumineuse d'une surface par l'aire apparente de cette surface, pour un observateur lointain, le mot luminance est substitué au mot brillance, qui correspond à l'éclat d'un objet. Une bonne luminance se caractérise par : [4]

1. Des images lumineuses (brillantes).
2. Un bon contraste : il faut éviter les images où la gamme de contraste tend vers le blanc ou le noir; ces images entraînent des pertes de détails dans les zones sombres ou lumineuses.
3. L'absence de parasites.

4.8 Contraste

C'est l'opposition marquée entre deux régions d'une image, plus précisément entre les régions sombres et les régions claires de cette image. Le contraste est défini en fonction des luminances de deux zones d'images. Si *Ll* et *L2* sont les degrés de luminosité respectivement de deux zones voisines *A*l et *A*2 d'une image, le contraste *C* est défini par le rapport :

$$C = L1 - L2 / L1 + L2 \qquad (1.1)$$

4.9 Le poids de l'image

Le poids d'une image se détermine en fonction de ces 3 paramètres : dimensions, résolution et nombre de couleurs. Il se mesure en octets.

5 LES DIFFERENTS TYPES D'IMAGES

II existe différentes catégories d'image selon le nombre de bit sur lequel est codée la valeur de chaque pixel.

5.1 Mode monochrome

Le mode monochrome (figure 1.7) est le plus simple; chaque pixel *y* est soit allumé [*Blanc*], soit éteint [*Noir*], l'image obtenue n'est pas très nuancée. Alors, pour convertir une image couleur en mode monochrome il faut d'abord passer par le mode niveaux de gris [7].

Figure 1.7 : Image en mode monochrome

5.2 L'image en niveaux de gris

Le niveau de gris est la valeur de l'intensité lumineuse en un point. La couleur du pixel peut prendre des valeurs allant du noir au blanc en passant par un nombre fini de niveaux intermédiaires. Donc pour représenter les images à niveaux de gris (figure 1.8), on peut attribuer à chaque pixel de l'image une valeur correspondante à la quantité de lumière renvoyée. Cette valeur peut être comprise par exemple entre 0 et 255. Chaque pixel n'est donc plus représenté par un bit, mais par un octet [6].

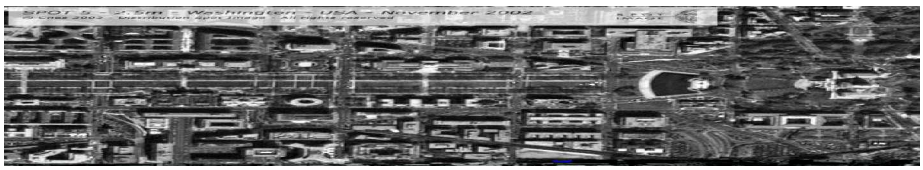


Figure 1.8 : Image en mode niveau de gris

5.3 L'image en couleurs

Même s'il est parfois utile de pouvoir représenté des images en noir et blanc ou en niveau de gris, les applications multimédias utilisent le plus souvent des images en couleurs (figure 1.9). La représentation des couleurs s'effectue de la même manière que les images monochromes avec cependant quelques particularités. En effet, il faut tout d'abord choisir un modèle de représentation.

Figure 1.9 : Image en mode couleur

Pour cela on utilise un espace de couleur à plusieurs dimensions qui consiste à donner suffisamment de composantes numériques pour décrire une couleur. Il y a des différentes représentations des images couleur :

- La représentation en couleurs réelle sur 24 bits.
- La représentation en couleurs indexées; on utilise une table appelée palette pour éviter la redondance de couleur.
- Le mode *RGB* est idéal pour l'affichage sur écran, une image *RGB* est composée de trois couches *Rouge*, *Vert* et *Bleu*. Chaque pixel est définit par une valeur possible de ces couleurs de [0 à 255]. Une fois combinées, ces couches permettent de générer toutes nuance de couleur. Le mode RGB correspond à l'affichage des moniteurs, où chaque point affiché est décomposé d'un mélange de lumière *RGB*.

6 LES FORMATS D'IMAGE

6.1 Les images bitmap

Les images affichées sur l'écran d'un *PC* sont des images matricielles, encore appelées bitmap, chaque image est en réalité une matrice de pixels.

6.1.1 Les différents formats d'Images bitmap

- ***Le format GIF (Graphic Interchange Format) :*** Créé par Compuserve, utilise aussi le codage *RVB*, mais le format *GIF* n'utilise pas toutes les 16 millions de couleurs. Il prend les 256 couleurs les plus courantes pour réaliser l'image au format *GIF*. Cela permet une bonne compression et un affichage rapide de l'image.
- ***Le format JPG ou JPEG (Joint Photographique Experts Group):*** Créé par un consortium industriel, ce format très utilisé sur Internet, permet d'afficher les images en mode 16 millions de couleurs et donc c'est le mode sans perte. Le codage d'un pixel (point de l'affichage) 3*x*1, byte (*RVB*) soit :(0<*R*<255), (0<*V*<255), (0<*B*<255). Le format *JPG* peut

aussi être utilisé pour compacter les images, il utilise un algorithme de compression qui réduit énormément la taille des images.

- ***Le format PCX (Picture Exchange Image Bitmap Zsoft)* :** A été crée à l'origine par *ZSoft* pour un programme de dessin (paint bruch)

6.1.2 Avantages et inconvénients

- **Avantages**
  - ü Format proche du hardware (sauf processeur dédié).
  - ü Adaptable aux images complexes.
- **Inconvénients**
  - ü Modification spatiales difficile, transformation par réduction => perte d'information, transformation par agrandissement => grossissement des points.
  - ü Encombrement important.

6.2 Les images vectorielles

L'image vectorielle est définie par une fonction mathématique. Pour définir par exemple la silhouette de la lettre *"d"*, on dira à l'ordinateur de dessiner un anneau, à la droite de laquelle est accolé un rectangle vertical et allongé de même couleur, positionné de manière spécifique par rapport à l'anneau. Mais pour afficher à l'écran ou imprimer, ces images vectorielles sont en fait automatiques traduites en images bitmap, car c'est le seul format directement affichable par ordinateur [4].

6.2.1 Les différents formats d'Images vectorielles

- ***Le format EPS (Postscript / Encapsulated Postscript)*** : Le Postscript est un langage de description de page conçu pour imprimer des documents indépendamment du périphérique utilisé. Il contient toutes les commandes nécessaires pour dessiner l'image sauvegardée.
- ***Le format CGM (Computer Graphics Metafile)*** : Les fichiers graphiques *CGM* restent un moyen privilégié d'échange de dessins vectoriels entre applications, mais il ne traite pas les images bitmaps.

6.2.2 Avantages et inconvénients

- **Avantages**
  - ü Adaptable à chaque périphérique (car ajustable, coordonnées relatives, adaptable aux drivers intelligents, ...).
  - ü Stockage moins encombrant (la taille ne dépend que de la complexité du document lui même).
  - ü Adapté aux objets mathématiques et leur extension (voiture, avion,...).
- **Inconvénients**
  - ü Mal adapté aux images quelconques.
  - ü La difficulté de traitement d'une partie de l'image.

7 DOMAINES D'APPLICATION

Parmi les domaines d'application de traitement d'images on peut cité :

- o ***Imagerie aérienne et spatiale:*** dans laquelle les traitements concernent l'étude des images satellites, l'analyse des ressources terrestres, la cartographie automatique, les analyses météorologiques.
- o ***L'imagerie médicale:*** on trouve des utilisations de cette technique dans l'échographie, la résonance magnétique nucléaire, ainsi que dans le domaine de la reconnaissance automatique des cellules ou de chromosomes.
- o ***La robotique:*** qui connaît actuellement le plus grand développement et dont les tâches usant de l'imagerie sont principalement l'assemblage (pièce mécanique, composants électroniques,…), le contrôle de la qualité, ainsi que la robotique mobile.
- o ***Simulation et contrôle de processus :*** on trouve des utilisations de cette technique dans les Cours de pilotage et le contrôle des Panneaux [9].

8 QUALITE DE L'IMAGE NUMERIQUE

Elle dépend, d'une part, de la qualité des images d'origine, et d'autre part, des moyens mis en œuvre pour convertir un signal analogique en signal numérique. Elle dépend aussi de :

- La qualité des périphériques de numérisation de l'image, du nombre de niveaux de gris ou de couleurs enregistrées, etc.
- La qualité de l'affichage à l'écran : définition de l'écran, nombre de teintes disponibles simultanément, calibrage de l'écran, etc.

Les critères d'appréciation de la qualité d'une image, tels que cités succinctement ci-dessus, dépendent largement de la structure même de l'image réaliste ou conceptuelle et de son mode de représentation (bitmap ou vectorielle) [4], [10].

9 POURQUOI COMPRIMER ?

La manipulation des images pose cependant des problèmes beaucoup plus complexes que celles du texte. En effet, l'image est un objet à deux dimensions, censé de représenter un espace à trois dimensions, ce qui pose deux problèmes majeurs :

- Le volume des données à traiter est beaucoup plus important (Les problèmes de stockage, de traitement et de transmission qui apparaissent rapidement avec la taille des images).
- La structure de ces données est nettement plus complexe, se heurtant à certains limitations (grâce au traitement d'image, ces contraintes sont résolues ou contournées).

Pour en savoir plus, nous présentons quelques exemples, empruntés à [10], et [11], qui donnent une idée des besoins qu'implique l'utilisation de la technologie de la numérisation, en compression comme solution aux problèmes de stockage et de transmission de l'image :

- Une image couleur représentée dans l'espace *Rouge-Vert-Bleu*, de taille 512*x*512 pixels dont chacune des composantes est codée sur 8 bits par pixel représente 786 *K* octets ;
- Une image de 800*x*600 pixels en 16 millions de couleurs (24 bits par pixels), correspondant à un fond d'écran, occupe 1 millions 400 000 octets.
- La transmission d'une séquence d'images couleur au format *QCIF* échantillonnée à 30 *Hertz* représente un débit de 9.12 *M* bits par seconde, et de 36.40 *M* bits par second au format *CIF* (Le format de *QCIF* [Quart de *CIF*] est défini par 167 pixels sur 144 lignes pour la luminance et 88*x*72 pixels pour la chrominance. Le format *CIF* [Common intermediate format] est défini par 352 pixels sur 288 lignes pou la luminance et 176*x*144 pixels pour la chrominance)

- Un film négatif 24*x*36 *mm* numérisé à 12µ*m* par point retourne une image de taille 3000*x*2000 pixels par couleurs, 8 *bpp*, 3 couleurs ce qui représente 18 *M* octets ;
- La transmission d'une séquence vidéo 512*x*512, 8 *bpp*, 3 couleurs sur une ligne téléphonique avec modem à 9600 bauds nécessite 11 minutes par image;
- Une image *LANDSAT* : 6000*x*6000 pixels par bande spectrale, 8 bits par pixel, et 6 bandes, soit $1.7x10^9$ bits.
- Une image de télévision basse résolution contient trois composantes ou couleurs, 512*x*512 pixels par couleur et 8 bits par pixel, soit un total de $6x10^6$ bits.

@ La compression et le codage consiste en la réduction de la taille physique d'un bloc d'information (on réduisant le nombre de bits par pixel à stocker ou à transmettre), en exploitant la redondance informationnelle dans l'image

@ Toute la problématique de la compression d'image consiste à satisfaire les contraintes technologiques, techniques ou financières au quelles on est confronté, tout en obtenant la qualité requise de l'image décomprimée pour l'application désirée.

10 NATURE DES IMAGES A COMPRESSER

Les images destinées à être compressées peuvent être de natures fondamentalement différentes, il convient donc de distinguer parmi toutes les classes d'images possibles celles qui sont les plus adaptées à la compression fractales en général. En effet, Les images à compresser peuvent être de déférentes natures :

1. L'image provenant de scannings de photos.
2. Les images de type «industriel».
3. Les images plus «géométriques», généralement générées par un programme ou numérisées : graphes, dessins, images de synthèse, …etc.

11 MESURES DE PERFORMANCE DE LA COMPRESSION D'IMAGE

On distingue les dimensions principales de performances ci-dessous :

11.1 Rapport et taux de compression

Le rapport de compression est l'une des caractéristiques les plus importantes de toutes les méthodes de compression, il représente le rapport entre le nombre de bits de la forme canonique au nombre de bits après codage :

$$Rapport_c = CR = \frac{nombre\ de\ bits\ de\ l'image\ avant\ compression}{nombre\ de\ bits\ de\ l'image\ aprés\ compression} = \frac{R_0}{R_c} \qquad (1.2)$$

Par conséquent le taux de compression est un pourcentage de l'espace obtenu après la compression par rapport à l'espace total requis par les données avant la compression. Il est défini par : $T_c = \left(1 - \frac{1}{rapport\ de\ compression}\right) x100$ (1.3)

11.2 Entropie

L'entropie $H(s)$ d'une source simple $[S]_N$ associée à une loi de probabilité $[P]_N$ est définie par la formule suivante :

$$H(s) = -\sum_{i=1}^{N} P(s_i)\log_2(P(s_i)) \quad \text{bits} \qquad (1.4)$$

Dans une image, L'entropie est une grandeur qui caractérise la qualité de l'information que contient cette dernière. Par exemple une image dont tout les pixels ont la même valeur contient très peut d'informations car elle est extrêmement redondante, son entropie est faible. En revanche une image dont tous les pixels ont une valeur aléatoire contient beaucoup d'information, son entropie est forte [12].

En pratique, l'entropie d'une image numérique est inversement liée à la probabilité d'apparition des niveaux de gris dans l'image. Par définition l'entropie d'ordre zéro H_0 est donnée par : $H_0 = \sum_{k=0}^{2^R-1} P(K).\log_2 P(K)$ *bpp* (1.5)

Avec : P (K) est la probabilité d'apparition des niveaux de gris dans l'image, K est la valeur de gris et R est le nombre de bits par pixels.

L'entropie H_0 d'une image originale fournit le débit minimal qu'il est possible d'atteindre par compression, pixel par pixel sans dégrader l'image, est par la même, un taux de compression sans perte maximal [12].

On peut cependant par détérioration de l'image dépassé l'entropie comme indiquée sur la courbe E (Quantité d'information, Détérioration) ci-dessous :

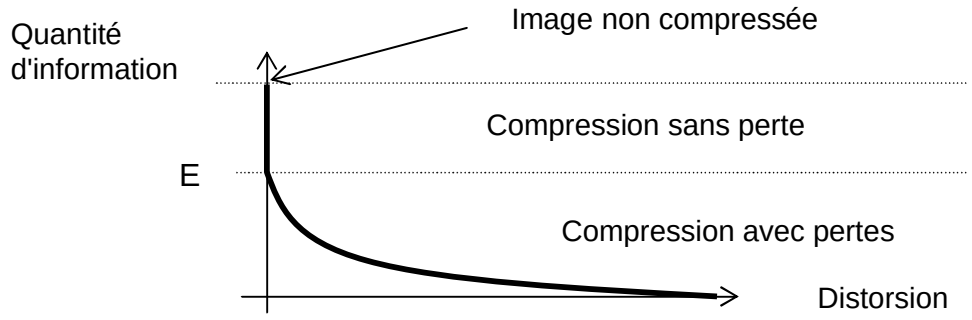


Figure 1.10 : la détérioration de l'image en fonction de l'entropie

11.3 Mesures de distorsion

La distorsion (D) est l'erreur introduite par l'opération de compression. La mesure de distorsion utilisée généralement en compression d'image, est l'erreur quadratique moyenne *MSE* (*Mean Square Error*.). Cette grandeur est définie par la moyenne des écarts au carré Entre le pixel I (m, n) de l'image originale et le pixel $\hat{I}$ (m, n) de l'image reconstruite comme suit :

$$MSE = \frac{1}{M.N}\sum_{m=0}^{M-1}\sum_{n=0}^{N-1}\left[I(m,n) - \hat{I}(m,n)\right]^2 \qquad (1.6)$$

Le rapport signal sur bruit (*Signal to Noise Ratio*) est défini par :

$$SNR = 10 \cdot \log_{10} \frac{\sum_m \sum_n \left[x^{(m,n)^2}\right]}{MSE} \qquad (1.7)$$

Le signal à bruit de crête pour une image dont le maximum est 2^R -1 dénoté par *PSNR* (*Peak Signal to Noise Ratio*) est déterminé par la formule :

$$PSNR = 10\log_{10}.\frac{\left(2^R - 1\right)^2}{MSE} DB \text{ (Décibels)} \qquad (1.8)$$

En compression d'image le *PSNR* d'une image de taille $8x(512)^2$ bits ($(512)^2$ indique une image de taille 512 par 512 pixels, chaque pixel est codé sur 8 bits) est défini plus souvent par [13].

$$PSNR = .10 \cdot \log_{10} \frac{(255)^2}{MSE} \tag{1.9}$$

Les mesures de distorsion sont très utiles pour déterminer la performance d'une méthode par rapport à d'autres méthodes.

12 CONCLUSION

Dans ce chapitre, on a essayé de faire un récapitulatif sur les notions élémentaires de traitement d'image.

On a porté aussi sur, les problèmes de taille rencontrées lors de la manipulation de l'image et le besoin de compression comme solution à ces problèmes. Nous avons abordé les paramètres de performances nécessaires pour estimer la qualité d'une image résultat d'une technique de compression. Le chapitre suivant est consacré aux différentes méthodes de compression.

[15] K. Chakib, *Généralité sur le traitement d'image*, le site www.ImgDist/LM.fr/travs/C1/ (1999).

[16] J. Fruitet, *Outils et méthodes pour le traitement des images par ordinateur*, (Université de Murne-la-vallée. France 2000).

[17] *Cours traitement d'image*, DUT Informatique (Université de Strasbourg).

[18] F. Devaux, *Filtrage d'image par réseau de neurones*, thèse de maîtrise, (Université de Paris 8, France 1997).

[19] Mathias Payan, « A hierarchical image segmentation algorithm », Rapport de Projet, 22 mai 2003.

[20] T.T. Alföldi, *Introduction aux images numériques et aux techniques d'analyse numérique*, (Centre canadien de télédétection 1996).

Azziz

[11] = [9] F. Davoine. Compression D'images par Fractales Basée sur la Triangulation de Delaunay. Thèse L'INPG ' Institut National polytechnique de grenoble ' Décembre 1995.

[27] B. Maher. Partition Arborescentes et Compression Fractale. Thèse de L'INPT, Janvier 2000.

[3] P. Beaurepaire, Epouse Beretta. " Compression D'images Appliquee Aux Angiographes Cardiaques : Aspects Algorithmiques, Evaluation de La Qualité Diagnostique " , Thèse N° 97 ISAL 0107 Institut National De Sciences Appliquees De Lyon, 1997.

[43] D. Saupe et M. Ruhl . Evolutionary Fractal Image Compression. IEEE International Conference on Image Processing (ICIP'96), Lausanne, Sept. 1996.

Ch1

1.1. Introduction

1.2. Définition de l'image

1.3. Image numérique

1.4. Les attributs de limage

1.4.1. Pixel

1.4.2. dimension (définition)

1.4.3. Bruit

1.4.4. Résolution

1.4.5. Histogramme

1.4.6. Contours et Textures

1.4.7. Luminance

1.4.8. Contraste

1.4.9 Le poids de l'image

1**.5. Les différents Types d'Images**

1.5.1. Mode monochrome

1.5.2. L'image en niveaux de gris

1.5.3. L'image en couleurs

1.6. Les formats d'image

1.6.1 Les images bitmap

1.6.1.1 Les différents Formats d'Images bitmap

1.6.1.2. Avantages et Inconvénients

1.6.2 Les images vectorielles

1.6.2.1. Les différents Formats d'Images vectorielles

1.6.2.2. Avantages et Inconvénients

1.7. Domaines d'application

1.8 Qualité De L'image Numérique

1.9 Les Problèmes Que Posent Les Images et Besoins en compression

1.10 Nature des images à compresser

1.11 Mesures de performance de la compression d'image

1.11.1 Rapport et Taux de compression

1.11.2 Entropie

1.11.3 Mesures de distorsion

1.12Conclusion

1 INTRODUCTION

Les méthodes de compression et de codage réduisent le nombre moyen de bits par pixel à stocker ou à transmettre, en exploitant la redondance informationnelle de l'image. Les techniques de compression se différencient par le fait quelles permettent ou non de compresser sans perte d'information, c'est-à-dire de manière réversible.

Ce chapitre se limite aux cas particuliers des images fixes, et à faire l'inventaire des méthodes disponibles, en vue de comprendre le principe et de tracer les grandes lignes prévisibles dans ce domaine.

2 L'INTERET DE COMPRESSION D'IMAGE

La compression et le codage consiste à la réduction de la taille physique d'un bloc d'information (réduction du nombre de bits par pixel à stocker ou à transmettre), en exploitant la redondance informationnelle dans l'image. Trois sortes de redondances sont exploitées dans la compression d'images :

- La redondance spatiale entre pixels ou blocs voisins dans l'image ;
- La redondance temporelle entre images successives dans une séquence vidéo ;
- La redondance spectrale entre plans de couleur ou bandes spectrales.

Les principaux critères d'évaluation de toute méthode de compression sont :

ü la mesure de qualité : la qualité d'un système se mesure par la qualité de reconstruction de l'image ;
ü La rapidité du codeur et décodeur ;
ü La réduction des débits : le taux de compression dépend de l'application**.**

3 LES DIFFERENTS TYPES DE COMPRESSION

3.1 Compression physique et logique

3.1.1 La compression physique

C'est un traitement qui agit directement sur les données, ce traitement est capable de comprimer l'information en agissant sur les données redondantes, afin qu'elles occupent un minimum de place. La lecture du fichier compressé est incompréhensible pour l'utilisateur.

3.1.2 La compression logique

C'est un traitement différent, c'est plutôt une réorganisation plus compacte des données ; l'information est mieux structurée et les données originales sont substituées par une autre information équivalente. Ce processus de substitution logique consiste à remplacer des symboles par d'autres plus petits, mais qui logiquement ont une même signification.
Exemple : "type de données abstrait" → «*TDA*» [30].

3.2 Compression symétrique et asymétrique

3.2.1 La compression symétrique

Utilise la même méthode pour compresser les données que la décompression. La quantité de travail effectué est la même dans les deux cas. Ce type de décompression est généralement utilisé dans la transmission de données.

3.2.2 La compression asymétrique

Nécessite une plus grande quantité de travail soit pour la phase de compression ou de la décompression. Tout dépend de l'application que l'on utilise. Si on prend le cas de l'archivage, la compression sera plus rapide que la décompression, car on accède rarement à ces données.

3.3 Encodage adaptif, semi adaptif et non adaptif

3.3.1 Un encodeur non adaptif

C'est basé sur un dictionnaire spécifique à un jeu de données précis. Par exemple si on utilise un dictionnaire basé sur la langue française, alors l'application de l'encodeur ne pourra se faire que sur des fichiers textes dont le contenu est en français.

3.3.2 Un encodeur adaptif

Permet de s'adapter aux données. Suivant la technique utilisée pour l'encodage des données, un dictionnaire peut-être construit au fur et à mesure de la lecture des données.

3.3.3 Un encodeur semi adaptif

C'est très proche d'un encodeur adaptif, la différence que l'on observe est que l'encodeur semi adaptif à besoin d'effectuer deux fois la lecture des données. La première lecture va servir à construire le dictionnaire et la seconde lecture permettra d'encoder les données [30].

4 MODELE GENERAL POUR L'ANALYSE DES METHODES DE COMPRESSION

La problématique de la compression peut être considérée de différents points de vue.

Le schéma général souvent utilisé pour décrire le fonctionnement des algorithmes de compression est celui présenté dans la figure 2.1 :

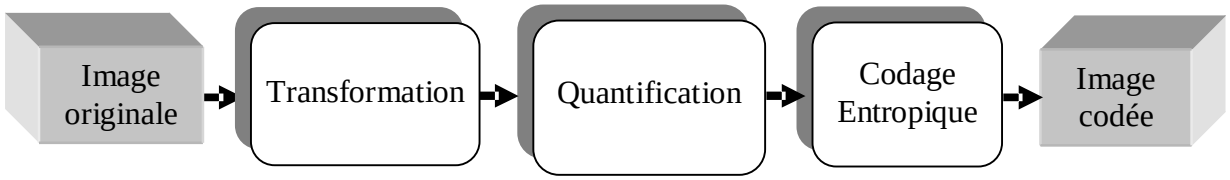


Figure 2.1 : Schéma général de la compression

4.1 Transformation ou décorrélation

La dépendance existante entre chacun des pixels et ses voisins (la luminosité varie très peu d'un pixel à un pixel voisin) traduit une corrélation très forte sur l'image. La décorrélation consiste à transformer les pixels initiaux en un ensemble de coefficients moins corrélés pour réduire le volume d'information, c'est une opération réversible.

4.2 Quantification

La quantification des coefficients a pour but de réduire le nombre de bits nécessaires pour leurs représentations. Elle représente une étape clé de la compression. Elle approxime chaque valeur d'un signal par un multiple entier d'une quantité *q*, appelée quantum élémentaire ou pas de quantification. Elle peut être scalaire ou vectorielle.

4.3 Le codage entropique

Le codage entropique effectue un codage sans perte sur les valeurs quantifiées. Cette dernière étape est nécessaire dans les méthodes sans perte, mais elle est souvent présente aussi dans les algorithmes irréversibles, puisque les valeurs transformées et quantifiées contiennent davantage de redondances [15]. Cependant, l'absence du codeur entropique peut être justifiée et nécessaire, à cause notamment des contraintes de transmission.

Cependant, pour analyser et classer les différentes approches de compression, à côté de ce modèle "*séquentiel*" [16] il nous a semblé nécessaire de définir un modèle "*vertical*" qui

répertorie la nature des informations utilisées. En fait, si on examine, quelles sont les idées de base des différentes méthodes, certaines questions se posent : l'approche est basée sur :

- quel type de redondances ?
- Quelles sont les informations exploitées durant la compression ?
- Quelle est l'unité de traitement d'image ?

Le modèle vertical présenté dans la Figure 2.2, nous présente qu'il y avait quatre réponses différentes à ces questions, quatre possibilités principales de traiter les images dans le but de la compression.

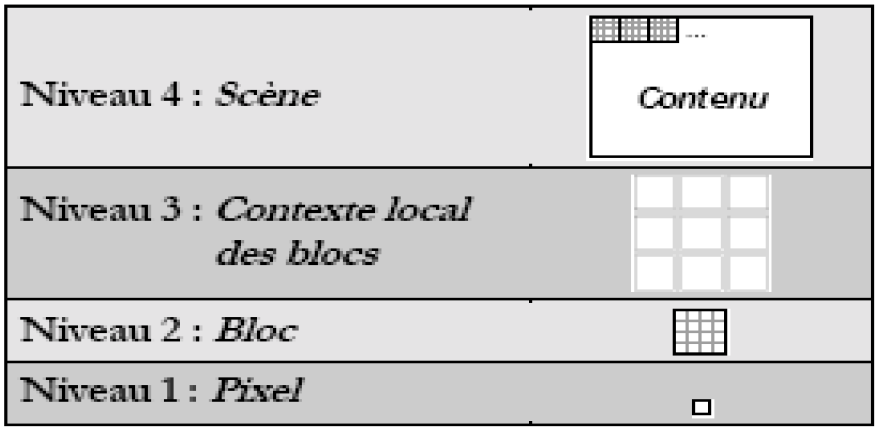


Figure 2.2 : Le modèle à quatre niveaux

Quatre niveaux ou couches se différencient donc :

- **Niveau pixel :** les pixels sont pris un par un par le codeur, et c'est directement ou indirectement la valeur de chaque pixel qui est codée et transmise au décodeur.
- **Niveau bloc :** cette stratégie est très appliquée principalement par les algorithmes irréversibles, elle consiste à regrouper les pixels en blocs, qui, en général, ne se recouvrent pas, et l'ensemble de ces blocs couvre l'image. L'unité du codage devient ainsi le bloc [16].
- **Niveau contexte local des blocs :** de la même manière que les pixels, les blocs peuvent aussi être traités soit individuellement, ou bien par rapport aux blocs adjacents. L'exploitation des dépendances inter-blocs nécessite des techniques plus complexes.
- **Niveau scène :** la totalité des pixels constitue l'image complète, les motifs des blocs forment le contenu de la scène d'image. Le contenu de l'image est l'information du plus haut niveau que l'on peut exploiter dans un système de compression [19].

@ La distinction de ces quatre niveaux ne signifie évidemment pas qu'une méthode se place entièrement à un seul niveau. Le premier niveau est toujours concerné, puisque l'étape finale de la décompression est la restitution des valeurs de pixels. Le troisième niveau est une extension du niveau *bloc*, qui tient compte des régularités entre les blocs voisins. Le niveau de la *scène* peut aussi donner des indications au niveau *bloc*, et vice versa [16].

5 CLASSIFICATION DES METHODES DE COMPRESSION

On peut regrouper les méthodes de compression en cinq classes à l'aide de différents critères. A la suite nous détaillons les méthodes les plus courantes.

5.1 Méthodes avec ou sans pertes d'information

Cette première classification s'intéresse à la présence ou non d'une distorsion ou perte d'information introduite par la compression. Notons que le taux de compression est limité par l'entropie de l'image pour les méthodes réversibles ou sans perte d'information, par contre pour les méthodes irréversibles ou avec perte d'information, le taux de compression est sensiblement supérieur à l'entropie de l'image [4].

C'est la classification que nous utiliserons par la suite pour exposer les principales méthodes de compression rencontrées.

5.2 Méthodes par pixels, bloc de pixels, ou image entière (scène)

Cette deuxième classification concerne les méthodes de compression dont leurs algorithmes touchent : le niveau pixel, blocs de pixels, ou toute l'image. On peut citer quelques exemples :

- Codage individuel des pixels : codage de *Huffman*;
- Codage de Blocs de pixels : Les normes standards *JPEG* et *MPEG* qui travaillent par blocs adjacents de 8*x*8 pixels, la quantification vectorielle, et le codage fractal;
- Codage de l'image entière : Des approches de codage par région, transformée cosinus discrète *DCT* (exposée dans les sections suivante), codage d'images sous bandes par filtrage ou décomposition sur une base d'ondelettes.

5.3 Méthodes Intra et Inter-images

Cette troisième classification s'applique aux séquences d'images (séries de coupes *3D*). Les méthodes intra-images (*intra-frame* en anglais) effectuent la compression de chaque image

indépendamment. Les méthodes inter-image (*inter-frame*) exploitent la redondance entre les images successives. Le standard *MPEG* code des séquences d'images en détectant le mouvement d'une image à l'autre [12], [17].

5.4 Méthodes spatiales et méthodes par transformation

La quatrième classification s'intéresse au domaine dans lequel s'effectuent les opérations de base de la compression. Une image peut être représentée de deux façons strictement équivalentes :

- Dans le domaine spatial, dans lequel l'image est représentée sous forme de pixels. C'est le domaine accessible visuellement à l'observateur.
- Dans le domaine fréquentiel, dans le quel l'image est représentée sous forme de coefficients de fréquences spatiales. Le passage d'un domaine à l'autre se fait par des transformations mathématiques les plus connues, telles que :
  - La transformation de *Fourier*
  - La transformation en cosinus discrète (*DCT Discrète cosine transfor*m), valable pour les images fixes. Elle traduit l'information spatiale en une information fréquentielle.

5.5 Méthodes adaptatives, non adaptatives

Cette cinquième classification indique si la méthode de compression est adaptative ou non.

- **Une méthode adaptative :** qui construit sa table de correspondance à l'analyse de l'information (modifier ses paramètres au fur et à mesure du codage, en s'adaptant aux données d'entrées) [12].
- **Une méthode non adaptative :** la table de correspondance statique et prédéfinie. Elle peut également y avoir une compression semi adaptative (mélange des deux méthodes) [18]

6 PRESENTATION DES METHODES AVEC ET SANS PERTES

On peut distinguer deux grandes familles d'algorithmes de compression : les méthodes dites *sans perte* ou *réversibles* garantissent la restitution parfaite des images, alors que les méthodes dites *avec perte* ou *irréversibles* modifient plus ou moins la valeur des pixels.

6.1 Les méthodes réversibles ou sans pertes

Comme son nom l'indique, ce type de compression n'occasionne aucune perte de données. Cette compression conservatrice est utilisée dans des applications comme l'archivage des images médicales, l'imagerie satellitaire (le coût des images est élevé et les détails sont importants), les textes, les programmes et tout autre type de donnée nécessitant une conservation à l'identique des données. Le seul critère d'évaluation des performances est dans ce cas "*Le taux de compression*". Il existe de nombreux types de compression d'image sans perte de données. Voici les plus répandus.

6.1.1 Méthodes différentielles et prédictives

La méthode prédictive est l'une des plus anciennes, c'est une méthode décorrélative dont le principe est le suivant :

L'idée du codage prédictif (Modulation par Impulsions Codées Différentielles) est de supprimer la redondance entre pixels voisins et de ne coder que la différence entre la valeur du pixel courant et sa valeur prédite à partir des pixels voisins. Ce qui permet l'élimination de la redondance et en ne codant que la nouvelle information apportée par chaque pixel. Dans des systèmes plus complexes et performants, on établit une fonction de prédiction, qui permet d'estimer la valeur d'un pixel en fonction de la valeur des pixels voisins. On code alors l'erreur de prédication, qui est l'écart entre la vraie valeur du pixel et la valeur prédite [17].

6.1.2 Méthodes par plages

v Le codage par répétition ou "Run Length Coding" (*RLC*)

Plusieurs types d'algorithmes sont utilisés pour compresser l'information vont de plus simple au plus complexe, et leur efficacité varie suivant la nature des données à comprimer. Le plus simple de ces algorithmes est le *Run Length Encoding* (*RLE*), méthode aussi appelée *Run Length Coding* (*RLC*).

C'est une technique avec mémoire ("*avec mémoire"* signifie qu'elle code les valeurs d'entrée en prenant en compte les valeurs précédentes). Son principe est de regrouper les valeurs voisines identiques et ne transmettre cette valeur qu'une seule fois, précédée par *le nombre de répétition*. Il est clair que cette approche fonctionne bien s'il y a beaucoup de répétitions dans le signal. Cet algorithme très simple, il peut aboutir à des taux de compression plus élevés. Il

existe des variantes dans lesquelles l'image est encodée par pavés de points, selon des lignes, ou bien même en zigzag.

Un exemple de traitement *RLE* est donné à la Figure 2.3 [29].

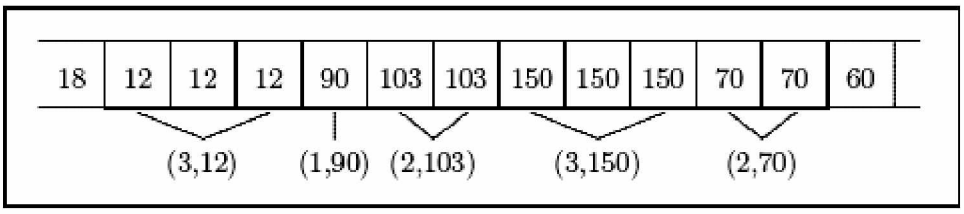


Figure 2.3 : Un exemple de codage par plage RLE

6.1.3 Codage de *SHANNON-FANO*

C. Shannon du laboratoire *Bells* et *R.M. Fano* du *MIT* ont développés à peu près en même temps une méthode de codage basée sur de simples connaissances de la probabilité d'occurrence de chaque symbole dans le message.

La procédure se décrit ainsi :

- **Procédure de codage**

ü **etape1** : Classer les n fréquences non nulles $\{f_i\}$ par ordre décroissant.

ü **etape2** : Deviser l'ensemble des messages en deux sous ensembles de fréquences aussi proches que possible.

ü **etape3** : Attribuer à chaque sous ensemble un bit 0 ou 1 ;

ü **etape4** : Rediviser chaque sous ensemble en deux nouveaux sous ensembles de fréquences équivalentes afin de réitérer l'algorithme jusqu'à ce qu'il n'y ait plus qu'un seul élément dans chaque sous ensemble.

Exemple

La chaîne que nous allons traiter est : *ACBBCDECECBCEEECCCCABCECBDBDBD*

c.à.d. $f(A)=2$, $f(B)=7$, $f(C)=11$, $f(D)=4$, $f(E)=5$.

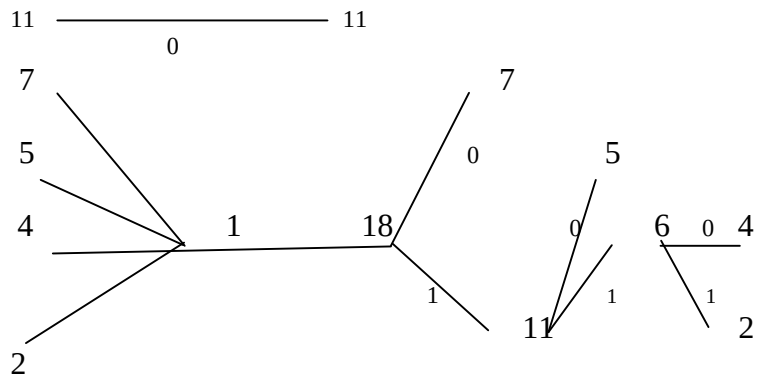
Octet	Fréquence processus	Code
C B E D A		0 10 110 1110 1111

Figure 2.4 : Codage de SHANNON -FANO

On remarque que le fichier compressé comporte 64 bits contre 196 bits pour le fichier original.

6.1.4 Le codage de *HUFFMAN*

En 1952, *David Huffman* inventa une nouvelle méthode de compression appelée compression à Arbre de *Huffman* [33]. Le codage de *Huffman* crée des codes à longueurs variables sur un nombre entier de bits [24]. L'algorithme considère chaque message à coder comme étant une feuille d'un arbre qu'il reste à construire. L'idée est d'attribuer aux deux messages de plus faible probabilité, les mots codes les plus longs. Ces deux mots codes ne se différencient que par leur dernier bit. Contrairement au codage de *Shannon-Fano* qui part de la racine d'un arbre et évolue par divisions successives, le codage de *Huffman* part des feuilles de l'arbre, et par fusions successives, redescend vers la racine [25].

o Procédure de codage

- Les probabilités d'occurrence de chaque message sont placées dans une liste dans un ordre décroissant. Nous dirons que la liste est composée d'enfants.
- Les deux probabilités les plus faibles sont identifiées en fin de liste.
- La somme des deux probabilités est placée à sa place dans la liste triée. Elle constitue un noeud parent. Les deux enfants sont retirés de la liste.
- Le chemin «enfant de plus faible probabilité, parent» est codé par un 1, l'autre par un 0
- La procédure reprend à l'étape 2 jusqu'à ce qu'il ne reste plus qu'une probabilité dans la liste.

Malgré son ancienneté, cette méthode est toujours remis au goût du jour, et offre des performances appréciables. En effet, beaucoup de recherches en algorithmiques ont permis d'améliorer les fonctionnalités de la méthode *Huffman* de base, comme avec les arbres binaires, arbres équilibrés, etc.[19].

Exemple

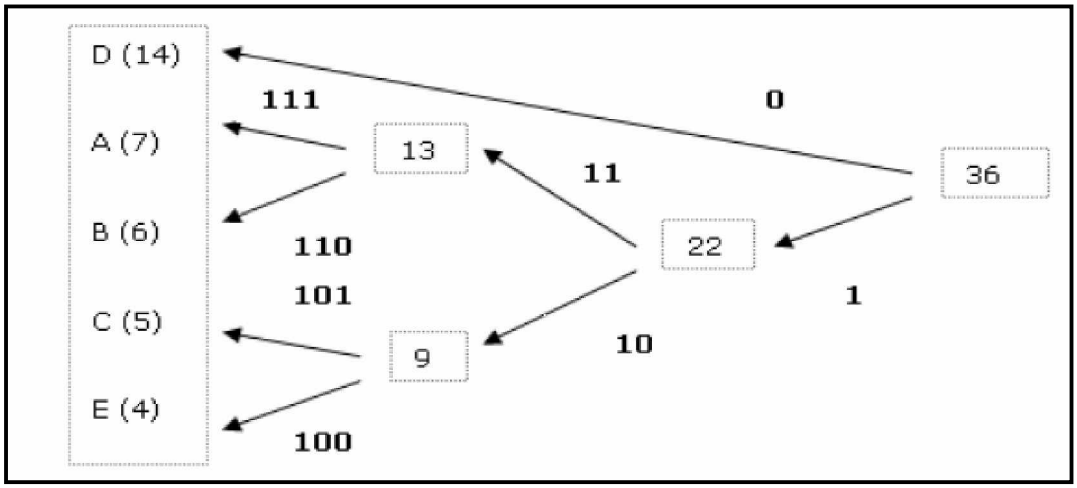


Figure2.5 : Algorithme de *Huffman*

Le codage obtenu est donc :

D (occurrence =14) : 0 ; *A* (occurrence =7) : 111 ; *B* (occurrence =6) : 110 ;

C (occurrence =5) : 101 ; *E* (occurrence =4) : 100

6.1.5 Le codage arithmétique

Le codage arithmétique est un codage récent utilisant un modèle statistique, tout comme le codeur de *Huffman*. Contrairement à ce dernier, il produit un code pour une séquence de symboles tout entière, et non pas un code par symbole. Chaque nouveau symbole lu modifie de façon incrémentale le code de sortie. Ce code de sortie est un nombre à virgule flottante compris entre 0 et 1, dont le nombre de chiffres après la virgule correspond au nombre de symboles. Contrairement à *Huffman*, il n'est pas obligatoire que chaque code ait un nombre entier de bits. Par exemple un symbole de probabilité 0.9 a pour entropie 0.15, mais *Huffman* affectera probablement un code de un bit (ou plus), et la séquence codée aura un nombre de bits plus long qu'en théorie. Le codeur arithmétique est plus performant que le codeur de *Huffman*, mais il est plus complexe à implémenter.

- **Algorithme du codage arithmétique**

Nous décrivons brièvement ci-dessous l'algorithme de codage arithmétique dans le but d'en illustrer le principe, sachant que le décodage opère de manière inverse.

- **Etape 1** : Calculer la probabilité associée à chaque symbole dans la chaîne à coder.
- **Etape 2** : Associer à chaque symbole un sous intervalle proportionnel à sa probabilité, dans l'intervalle [0,1[(l'ordre de rangement des intervalles sera mémorisé car nécessaire au décodeur).
- **Etape 3** : Initialiser la limite inférieure de l'intervalle de travail à la valeur 0 et la limite supérieure à la valeur 1.
- **Etape 4** : Tant qu'il reste un symbole dans la chaîne à coder :
  - largeur = limite supérieure - limite inférieure
  - limite inférieure = limite inférieure + largeur x (limite basse du sous intervalle du symbole)
  - limite supérieure = limite inférieure + largeur x (limite haute du sous intervalle du symbole)
- **Etape 5** : La limite inférieure code la chaîne de manière unique.

Le principal inconvénient de l'algorithme réside dans sa complexité d'implémentation. Une étude complète du codeur est donnée dans la référence [23], et différentes solutions pratiques ont été proposées [23], de manière à faciliter la compression des fichiers de grande taille sans être limité par la précision des calculs.

6.1.6 Codage par dictionnaire adaptatif (*LZW*) (*Lempel-Ziv-Welch*) ou *LZ77*

La société *Unisys* ayant déposée la méthode *LZW*, il en existe une variante non brevetée, le *LZ77*. Elle porte le nom de ses trois inventeurs : *Lempel* et *Ziv* qui l'ont conçue en 1977, et *Welch* qui l'a finalisée en 1984.

Elle utilise un dictionnaire qui n'est pas stocké dans le fichier compressé qu'elle construit dynamiquement, au cours de la compression et de la décompression. Il s'agit cette fois de repérer des motifs composés de séries variables en longueur de bits ou d'octets. Chaque motif est copié dans le dictionnaire et se voit attribué un indice. *Puis chaque motif est remplacé*

dans le fichier par son indice, sachant qu'une valeur isolée n'est pas codée. Elle a besoin d'un apprentissage pour être efficace, pour reconnaître des longues chaînes répétées. En effet, l'algorithme ne fonctionne pas sur un nombre fixe de motifs mais *apprend* les motifs du fichier durant la lecture du fichier à compacter. Elle est donc peu efficace sur des petits fichiers. Cette méthode est très rapide.

L'algorithme suivant montre son mécanisme

Le *LZW* ou *LZ77*
W ⟵ NULL
Tant que (valeur # eof)
    Lire la valeur *k*
Si W_K est dans le dictionnaire $w \longleftarrow w_k$
Sinon Ecrire le code de W
 Ajouter W_K au dictionnaire
W ⟵ *K*
Fin tant que

6.2. Les méthodes de compression avec pertes ou irréversible

Les méthodes irréversibles permettent des taux de compression assez élevés au prix d'une dégradation de la qualité de l'image. Outre, le taux de compression, une mesure de cette dégradation est nécessaire à l'évaluation des performances de ces méthodes. Plusieurs mesures existent pour l'évaluation des performances de ces méthodes comme le *SNR*, *MSE*, et *PSNR*, voir chapitre1.

Avec ces méthodes, on peut aussi distinguer :

- ***les méthodes spatiales (ou directes)*** qui agissent directement sur les échantillons d'une image dans le domaine spatial.
- ***les méthodes par transformation*** qui reposent sur une transformée (en général linéaire) de L'image originale.

6.2.1 Quantification

En général, la quantification apparaît en deuxième lieu dans un processus de compression, pour réduire la quantité d'information, de manière souvent *irréversible* [34].

Ainsi, une quantification est utilisée pour simplifier la représentation, tout en préservant l'information la plus pertinente. On remplace ainsi les valeurs initiales par un ensemble fini d'éléments qui donneront des résultats acceptables lors de la phase de décompression. Ces éléments peuvent être scalaires lorsque, on remplace l'information en chaque pixel par des nombres entiers; ou des vecteurs lorsque l'on considère des groupes de pixels représentant des configurations typiques.

Dans les deux cas, le principe est décrit par la Figure 2.6. Un élément quelconque (scalaire ou vecteur) est remplacé par l'élément du dictionnaire le plus approprié.

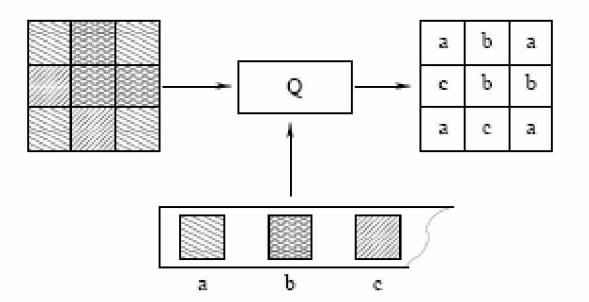


Figure 2.6 : Principe de la quantification

6.2.1.1 Quantification scalaire

La quantification scalaire [22], consiste à réduire le nombre de valeurs que peut prendre une grandeur scalaire. C.à.d. remplacer un nombre très grand de symboles par un nombre restreint de codes. C'est une opération irréversible très largement employée en compression (il est à noter que les méthodes de codage utilisant un quantificateur ne sont jamais réversibles parce que l'étape de quantification introduit inévitablement une distorsion).

Un quantificateur scalaire est un opérateur qui associe à une variable continue u une variable discrète u' pouvant prendre un nombre plus faible, et fini de valeurs ; il est généralement définie comme une fonction en escalier (Figure 2.7), l'intervalle entre chaque valeur étant appelé intervalle de décision.

Pour un nombre de niveaux de quantification fixé *L*, on peut choisir les régions de décision $\{t_k \ , k = 1....L+1\}$ ainsi que les seuils de décision $\{r_1.....r_L\}$ de façon à minimiser la distorsion entre l'entrée et la sortie. Si u se trouve dans la région (t_k, t_{k+1}), u' aura pour valeur r_k.

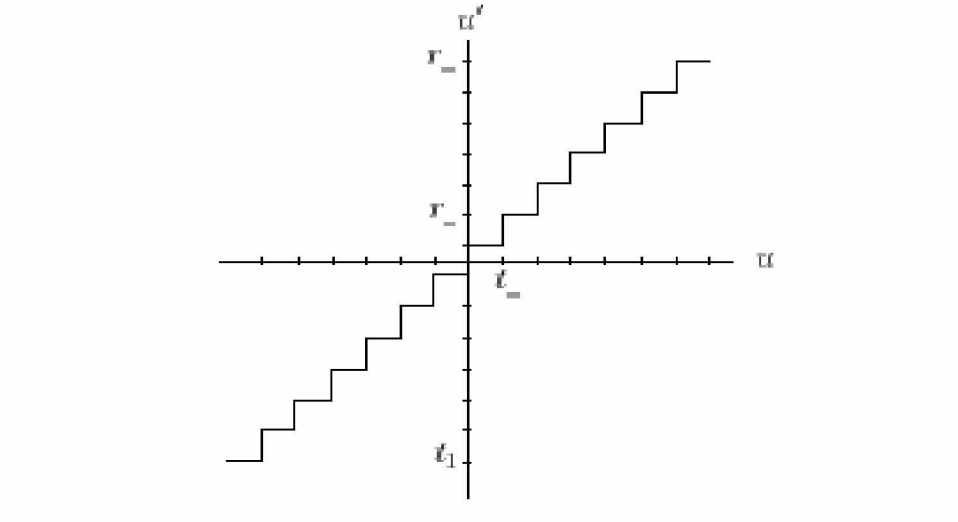


Figure 2.7 : Quantification scalaire uniforme en escalier

La quantification scalaire porte sur la grandeur physique associée à l'information de niveau de l'image. L'objectif de la quantification est de diviser la gamme dynamique ou dynamique de la grandeur physique en un nombre fini d'intervalles, et d'attribuer à toutes les valeurs du même intervalle une seule valeur, dite valeur quantifiée. Trois questions se posent alors :

- trouver la gamme dynamique ;
- choisir le nombre d'intervalles ;
- répartir ces intervalles;

Il existe plusieurs formes de quantifications scalaires dont la plus simple est la quantification scalaire uniforme. C'est la forme la plus couramment utilisée en compression d'images.

6.2.1.2 Quantification vectorielle

L'approche de la *quantification vectorielle* (*QV*) est une généralisation naturelle de la quantification scalaire (*QS*) pour des vecteurs. Par conséquent elle se place dans la deuxième phase de la chaîne de codage.

La *QV*, est un sujet principal de plusieurs recherche et publication dans le domaine de la compression d'images voir [22], [14], [16], et [17]. Nous décrivons par la suite

schématiquement le principe de la quantification vectorielle, codage et décodage (voir Figure2.8).

o Principe

L'image à coder est découpée en blocs qui ne se chevauchent pas mais qui couvrent toute l'image. Chaque bloc de taille *k* est comparé aux imagettes d'un ensemble de blocs, appelé *dictionnaire W= {w1, w2,..., wN}* $\subset R^k$. Ces blocs prédéfinis sont nommés *mots de code* ou vecteurs de reproduction. La comparaison consiste à calculer une mesure de distance entre le bloc (vecteur) à coder et les mots de code.

ü ***Le codage*** s'effectue en ignorant le bloc original et en gardant seulement l'*indice* (l'adresse) du mot de code le plus proche. La distance appliquée est en général la distance euclidienne, ce qui est équivalent à la minimisation de l'erreur quadratique moyenne.

ü ***Le décodeur*** reprend tout simplement les mots de code correspondants Aux indices reçus (transmis ou stockés), et reconstruit ainsi l'image.

Le taux de compression dépend du nombre de mots de code ainsi que de leur taille. Avec une taille du dictionnaire égale à $N=2^n$, le taux de compression s'exprime par :

$$Tc = mk/n \qquad (2.1)$$

Où *k* : est le nombre des pixels dans un bloc ;

m : désigne le nombre de bits par pixel dans l'image originale.

IL est évident que la *QV* comme la *QS* est une approche irréversible. La qualité du codage est fonction de la taille du dictionnaire, de la taille et du contenu des mots de code.

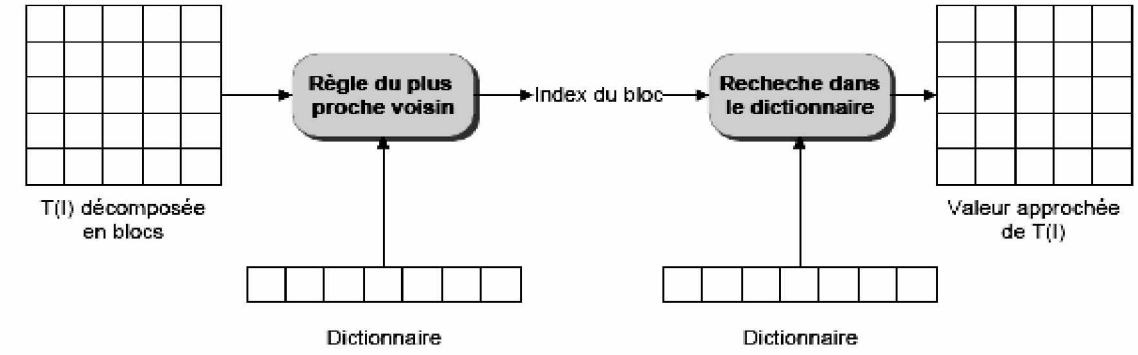


Figure 2.8 Le principe du Quantificateur Vectoriel.

6.2.2 Codage par transformée

Dans ces méthodes, l'image de dimension *NxN* est subdivisée en sous images ou blocs de taille réduite (la quantité de calcul demandée pour effectuer la transformation sur l'image entière est très élevée). Chaque bloc subit une transformation mathématique orthogonale inversible linéaire du domaine spatial vers le domaine fréquentiel, indépendamment des autres blocs (transformée en un ensemble de coefficients plus ou moins indépendants). Les coefficients obtenus sont alors quantifiés et codés en vue de leur transmission ou de leur stockage. Pour retrouver l'intensité des pixels initiaux, on applique sur ces coefficients la transformation inverse.

L'objectif de ces transformations est double : il s'agit de

- ü décorréler les données, c'est-à-dire d'obtenir des coefficients transformés moins corrélés que les pixels de l'image ;
- ü concentrer l'énergie sur un nombre réduit de coefficient, les coefficients ayant une valeur plus importante aux basses fréquences qu'aux hautes fréquences.

Dans ce cas, on obtiendra une compression effective en codant finement les coefficients des basses fréquences, et grossièrement, voire en les supprimant, les coefficients hautes fréquences. Dans ce but, plusieurs transformations linéaires ont été proposées :

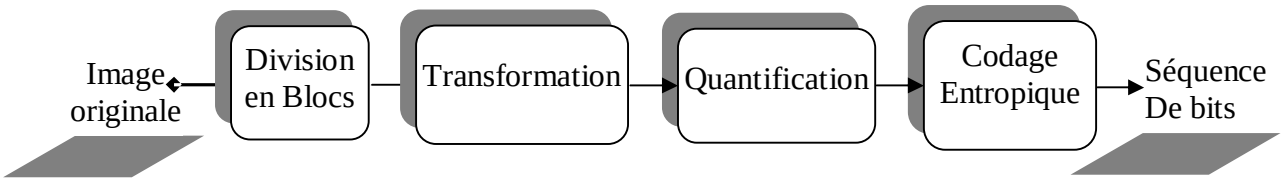
- Transformation de *Karhunen-Loeve* (*TKL*).
- Transformation de *Fourrier* discrète (*TFD*).
- Transformation de *Hadamard* (*TH*).
- Transformation de *Haar* (*THA*).
- Transformation en cosinus discrète (*TCD*).

- ü l'efficacité de ces transformations peut être mesurée par la prise en compte de trois facteurs : l'efficacité de décorrélation, la concentration de l'énergie, et l'existence d'algorithmes rapides pour calculer les transformations.
- ü Les méthodes de codage par transformation présentent des propriétés d'immunité au bruit de transmission bien supérieures aux méthodes précédentes. Une erreur affecte en effet seulement la valeur d'un coefficient, et sera lissée au décodage lors du calcul de la transformation inverse. L'effet sera ainsi peu visible.

o **Le schéma de compression des méthodes de transformations d'images**

Le principe d'un système de codage par transformation est le suivant :

Phase de compression



Phase de décompression

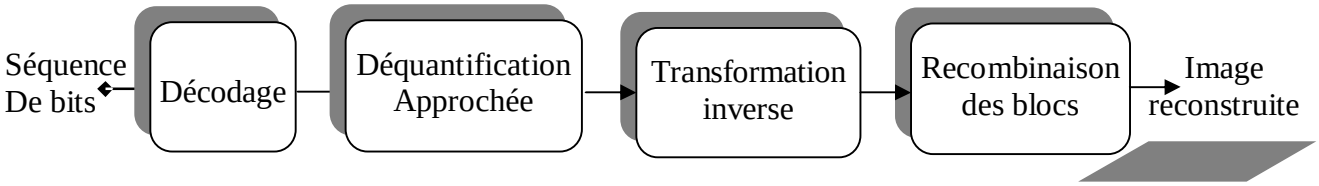


Figure 2.9 Schéma de principe de la compression / décompression par transformation

6.2.2.1 Transformée de fourrier

Elle permet simplement de passer du domaine spatial au domaine fréquentiel. Cette transformée rend donc visible les composantes en fréquence de l'image

$$F(\omega) = \frac{1}{\sqrt{2\pi}} \int_{-\infty}^{\infty} f(t) \cdot e^{-j\omega \cdot t} \cdot dt \qquad (2.2)$$

Il est intéressant de noter que nous pouvons revenir au domaine spatial via la transformée de fourrier inverse. Une application brute de cette formule étant extrêmement longue. Une autre façon d'effectuer ce calcul permet de limiter considérablement la durée de cette transformation. C'est ce que l'on appelle la *FFT* (*Fast Fourier Transform*) [27].

6.2.2.2 Transformation en cosinus discrète DCT

La *DCT* (*Discret Cosinus Transform*) est une transformée fort semblable à la *FFT*, travaillant sur un signal discret. Elle prend un ensemble de points d'un domaine spatial et les transforme en une représentation équivalente dans le domaine fréquentiel. La *DCT* transforme un signal d'amplitude (chaque valeur du signal représente *l'amplitude* d'un phénomène) discret bidimensionnel en une information bidimensionnelle de "*fréquences*".

Les équations qui suivent, donnent respectivement la transformée en cosinus discrète directe et inverse.

$$DCT(i,j) = \frac{1}{2\sqrt{2N}} C(i)C(j) \sum_{X=0}^{N-1} \sum_{Y=0}^{N-1} Pixel(X,Y) \cos\left[\frac{(2X+1)i\pi}{2N}\right] \cos\left[\frac{(2Y+1)j\pi}{2N}\right] \quad (2.3)$$

$$pixel(X,Y) = \frac{1}{2\sqrt{2N}} \sum_{X=0}^{N-1} \sum_{Y=0}^{N-1} C(i)C(j)DCT(i,j) \cos\left[\frac{(2X+1)i\pi}{2N}\right] \cos\left[\frac{(2Y+1)j\pi}{2N}\right] \quad (2.4)$$

Où $C(i) = 1/\sqrt{2}$ si $i = 0$; $C(i) = 1$ si $i \neq 0$ (2.5)

pixel (*X*, *Y*) désigne la valeur du pixel de cordonnées (*X*, *Y*) et *DCT* (*i*, *j*) le coefficient repéré par la ligne *i* et la colonne *j* dans la matrice *DCT*.

La *TCD* est effectuée sur une matrice carrée *NxN* de valeurs de pixels en donnant une matrice carrée *NxN* de coefficients de fréquence. Le temps de calcul requis pour chaque élément dans la *TCD* dépend de la taille de la matrice.

Vu la difficulté d'appliquer la *TCD* sur la matrice entière, celle-ci est décomposée en blocs de taille 8*x*8 pixels (compression *JPEG*). Concrètement, et en terme simple, cette transformation va essaye de faire correspondre des blocs de 8*x*8 de l'image en une somme de fonction basique qui sont données dans la matrice 8*x*8 de la *DCT* (*DCT* matrix).

- **Différence entre la *FFT* et la *DCT***

- la *DCT* est actuellement une version simplifier de la *FFT*.
- seule la partie réelle de la *FFT* est conservé
- beaucoup plus simple en terme de coup de programmation
- la *DCT* est efficace dans la compression de multimédia (*JPEG*)
- *DCT* Beaucoup plus utilisée.

6.2.3 Le codage en sous-bandes

Le codage en sous-bandes a été introduit pour la parole par *Crochiere et al.* En 1976, puis appliqué à l'image sous la forme que nous connaissons aujourd'hui par *Woods* et *O'Neil* en 1986 [31]. Il consiste à décomposer le signal ou l'image en différentes bandes de fréquences spatiales et à coder chacune d'entre elles de manière indépendante ou non. Dans les schémas

classiques, la bande spectrale du signal original est d'abord divisée en deux parties à l'aide de deux filtres numériques (passe-bas et passe-haut).

La procédure est ensuite répétée dans chacune des sous-bandes fréquentielles. Le résultat constitue une structure arborescente, symétrique ou non, composée d'un nombre entier de sous-bandes. Le signal est reconstruit par recombinaison des sous-bandes à l'aide de filtres d'interpolation.

La figure 2.10 représente une étape de la décomposition. Le banc de filtre d'analyse qui est généralement un banc bi-dimensionnel suivis chacun par sous-échantillonneur. Ces filtres uni-dimensionnels et ces sous-échantilonnages, sont d'abord appliqués aux lignes de l'image, et ensuit aux colonnes des images obtenues. On se retrouve avec quatre images notées. L'image centre *BF* est une sous-image de basse résolution, sous-image horizontale *HF*, sous-image verticale *HF* et sous-image diagonale *HF* correspondent à quatre sous-bandes fréquentielles directionnelles.

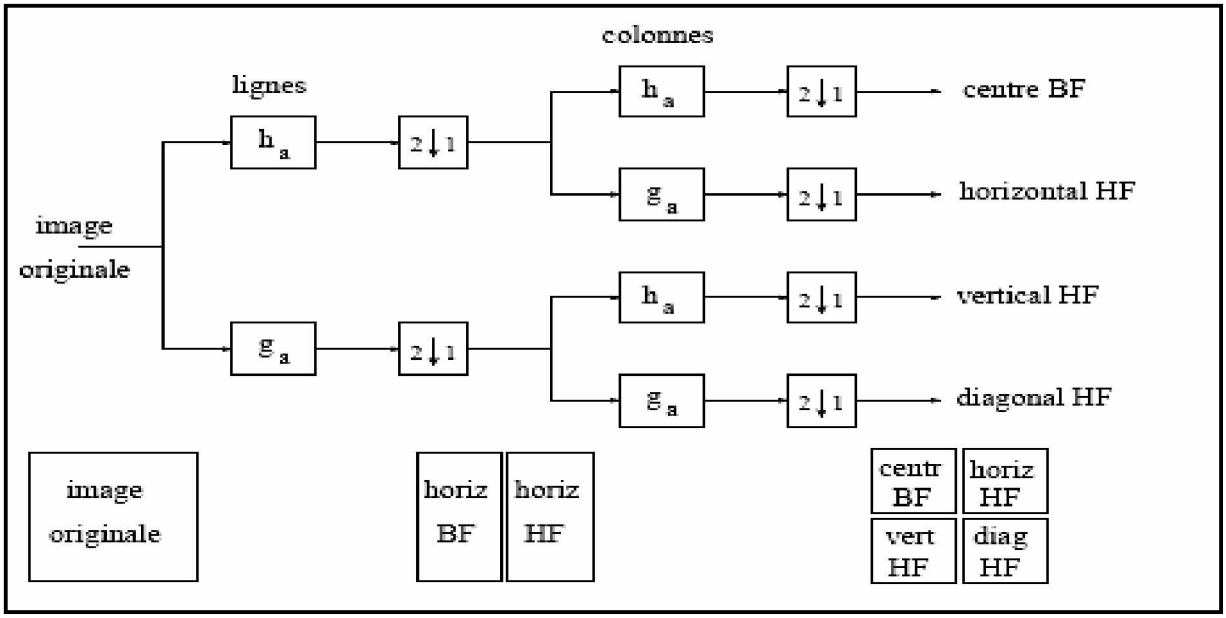


Figure 2.10 Décomposition d'une image en sous-bandes

Ces travaux ont suscité un grand intérêt pour les techniques multirésolution qui se sont traduits par les travaux sur l'utilisation des ondelettes en analyse du signal (*Rioul* et *Vetterli*) dans les années 80, et de nombreuses application au codage de la parole, de la musique et de l'image fixe et animée. C'est un domaine encore très actif aujourd'hui.

6.2.4 La compression par ondelettes

Les ondelettes c'est d'abord une théorie mathématique récente d'analyse du signal, développée dans les années 80. On peut considérer qu'il s'agit d'une extension de l'analyse de *Fourier* [28]. La technologie de compression à base d'ondelettes offre une plus grande finesse au niveau de l'analyse du signal, et permet de mieux s'adapter aux propriétés locales de l'image. La transformation par Ondelettes est une technique de compression d'image fixe très performante.

Quand on enregistre une image en utilisant les ondelettes, on divise sa résolution par deux et on code l'information perdue par des coefficients d'ondelettes. On commence donc à coder les détails les plus fins (les hautes fréquences). On recommence l'opération autant de fois que nécessaire jusqu'à ce que l'image se réduise à 1 pixel. A chaque étape l'image est «lissée» et les détails perdus sont codés en coefficients d'ondelettes.

L'intérêt de la transformation par ondelettes par rapport aux autres méthodes de compression est que celle-ci ne considère pas l'image dans son ensemble pour la coder mais, la travaille par couche, cherchant à enregistrer les détails les plus importants (ceux qui se démarquent le plus du reste du signal) à chaque résolution [26].

- **Algorithme de compression par ondelette ou waveletes**

Passons maintenant à l'algorithme pyramidal utilisé. La décomposition en coefficients d'ondelettes n'utilise pas une fonction de moyenne, mais s'appuie sur deux filtres. Un filtre passe bas (*H*) et un filtre passe haut (*L*). La combinaison de ces filtres permet d'obtenir quatre sous images *HH*, *HL*, *LH* et *LL*. Ces filtres sont nommés miroirs et quadratures.

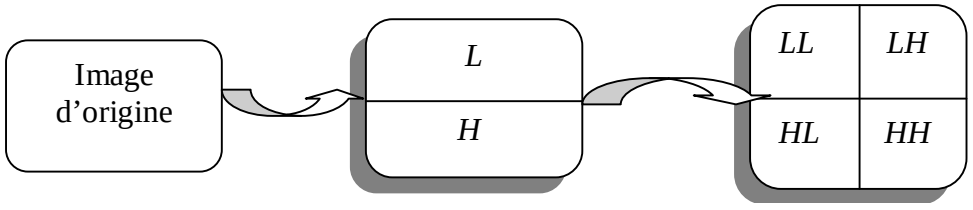


Figure 2.11 Transformation en colonnes et en lignes

Chacune des quatre images obtenues par la transformation représente des informations bien distinctes.

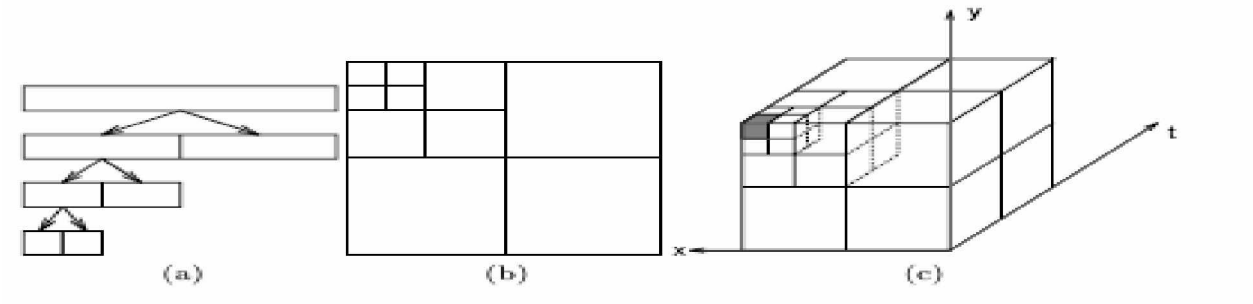


Figure 2.12 : Transformation ondolette «pyramidal» (1-D (a),2-D (b),3-D(c))

o Les étapes de compression par ondelttes

Tout d'abord, voici le principe de compression et décompression d'une image par ondelettes :

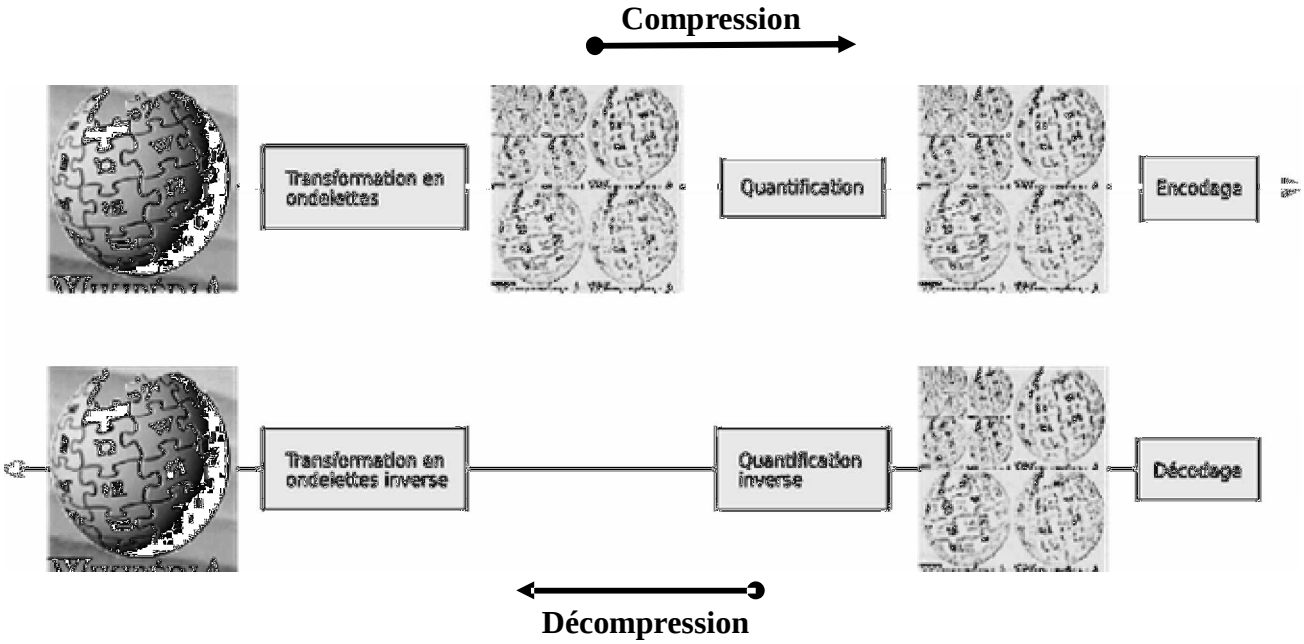


Figure 2.13 Le principe de Compression / Décompression par ondelettes

v La compression ondelettes

Les étapes pour compresser une image sont

1. Transformations par ondelettes
2. Quantification : les valeurs des images de détails inférieures à un certain niveau sont éliminées, en fonction de l'efficacité recherchée. C'est cette étape qui introduit des pertes.

3. Codage des valeurs restantes. Les données restantes sont transmises à un encodeur entropie, c'est à dire à un algorithme de compression de données (*LZW, HUFFMAN,….*).

v **Décompression ondelettes**

La transformation inverse par ondelettes reconstruit une image originale. La construction de l'image à partir des sous-bandes restitue l'image en mode progressif. L'affichage de l'image peut s'effectuer en deux modes :

ü Soit la taille de l'image augmente au fur et à mesure de la lecture du fichier compressé.

ü Soit la résolution de l'image augmente au fur et à mesure de la lecture du fichier compressé.

6.2.5 Les normes de compression des images

6.2.5.1 La norme de compression *JPEG*

La norme *JPEG* (*Joint Photographic Experts Group*) est conçue par le groupe *ISO* (*International Standards Organisation*) et le groupe *CEI* (*Commission Electronic International*).

Ce standard a vu le jour en 1992. Il est devenu le format le plus populaire pour le stockage de photographies numériques, car il offrait alors une meilleure compression que tous les formats bitmap connus à l'époque. Le standard avait pour objectif de concevoir une nouvelle norme de compression, avec les différentes contraintes suivantes [32] :

ü La norme JPEG doit être très proche des nouvelles techniques de compression, en terme de taux de compression, qualité de restitution et de temps de calcul ;

ü JPEG doit pouvoir compresser tout type d'images réelles (tailles différentes, images multi-composantes . . .) ;

ü L'algorithme doit être implémentable sans trop de problèmes sur une grande gamme de CPUs et sur des cartes plus spécialisées ;

ü Le codage doit pouvoir être séquentiel, progressif, sans perte et/ou hiérarchique,

Les techniques définies par la norme *JPEG* se divisent en deux classes : les méthodes de compression avec pertes qui sont basées sur la *TCD* suivie d'une quantification et d'un codeur

entropique. La seconde classe, concerne les processus de codage sans pertes, cette classe de codeurs n'est pas basée sur la *TCD* mais sur le codage *MICD* suivi d'un codage entropique.

o Le principe de compression de la norme *JPEG* non réversible

La figure 2.14 explique clairement les différentes étapes de l'algorithme de compression *JPEG* non réversible.

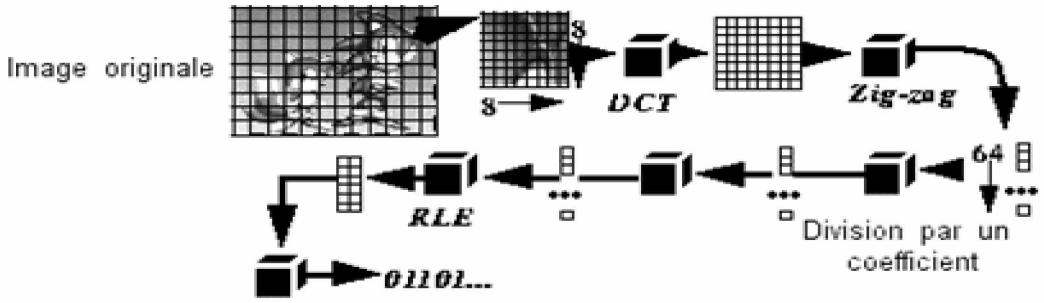


Figure 2.14 : Schéma de principe de La compression *JPEG*

o Détails de l'algorithme de compression *JPEG* non-réversible

Cette compression se décompose en six étapes:

1. *Décomposition en blocs et transformation des couleurs* : l'image est découpée en blocs de 8*x*8 pixels.
2. *Transformation DCT* : On applique la *DCT* aux blocs de 8*x*8 pixels.

$$DCT = C \times Pixels \times C^t \tag{2.6}$$

Avec C: la matrice de transformée de cosinus C et C^t : la matrice transposée de C.

Les formules déterminant la matrice de transformée de cosinus C

$$C_{i,j} = \frac{1}{\sqrt{N}} \; si \; i = 0 \tag{2.7}$$

$$C_{i,j} \sqrt{\frac{2}{N}} \cos\left[\frac{(2j+1)i\pi}{2N}\right] \; si \; i > 0 \tag{2.8}$$

3. *Seuillage et quantification des coefficients issus de la DCT* :

$$valeur\ quantifiée\,(i,j) = \frac{DCT}{Quantum\,(i,j)} \tag{2.9}$$

On voit facilement que l'opération s'inverse facilement lors de la dé-quantification:

$$DCT\,(i,j) = valeur\ quantifiée\,(i,j) \times Quantum\,(i,j) \tag{2.10}$$

4. *Lecture en zigzag des coefficients* : après la phase de quantification un grand nombre de valeurs de la matrice résultat se trouvent réduite à 0. Le fait qu'il y ait beaucoup de valeurs mises à 0 va nous permettre de compresser encore mieux grâce à la compression *RLE* que nous verrons plus tard. La lecture en zigzag est présentée dans la figure 2.15.

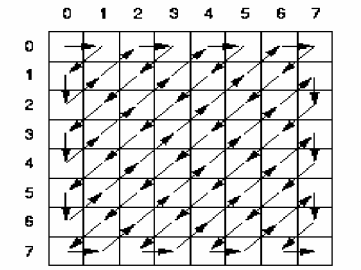


Figure 2. 15 Le parcours d'un bloc *DCT* en zigzag

5. *Codage RLC* : Ce codage est basé sur le fait que l'on va coder une grande quantité de 0.
6. Un codage entropique des coefficients de la dernière matrice termine la chaîne de traitement. *JPEG* propose deux méthodes : le codage de *Huffman*, et le codage arithmétique.

- **La décompression *JPEG*** comporte les mêmes étapes que la compression mais dans le sens inverse en commençant par la méthode de décodage statistique, puis la matrice obtenue est multipliée par la matrice de quantification que l'on reconstitue grâce au facteur de qualité et enfin on applique la *DCT* inverse (*IDCT*) pour retrouver une image plus ou moins dégradée par rapport à l'image initiale.

- **Algorithme *JPEG* réversible**

La seconde classe, concerne les processus de codage sans pertes, cette classe de codeurs n'est pas basée sur la *TCD* mais sur le codage *MICD* suivi d'un codage entropique (*huffman* ou arithmétique).le traitement se fait sur l'image entière ,et non sur des blocs .

2.6.2.5.2 La norme de compression *JPEG2000*

La version définitive du standard a pris forme en décembre 2000. Ce nouveau standard a pour objectif d'offrir de nouvelles fonctionnalités permettant de répondre à une demande croissante, à savoir :

- Obtenir des performances de compression supérieures à son prédécesseur *JPEG*, notamment pour des débits très faibles.

- Permettre d'organiser le fichier compressé de plusieurs manières, notamment en fonction de la résolution désirée ou de la qualité de reconstruction.
- Avoir un mode de compression sans perte performant
- Fournir la possibilité de coder des parties d'une image avec une qualité supérieure à d'autres parties [31].
- En *JPEG* 2000 le même algorithme de codage s'applique à une grande variété de types d'images (couleurs, niveaux de gris, multi-composantes...).

La norme *JPEG* 2000 est une norme très récente basée sur la technologie des ondelettes. Le résultat pratique de ce choix devrait être une amélioration modeste mais réelle du taux de compression (de l'ordre de 20 %) et une qualité supérieure de restitution par rapport à ce que permet le *JPEG* actuel. Les coefficients d'ondelettes sont plus faciles à stocker que les blocs résultants de l'application de la *DCT* [20].

Cette nouvelle norme est composée de plusieurs parties dont certaines sont publiées et d'autres non.

- Nous sommes donc devant une norme d'une grande ampleur dont certaines parties seulement sont publiées, il faudra plusieurs années pour terminer ce travail. D'autre part la norme n'est pas constituée de logiciels directement utilisables. Un gros effort doit être fait par les industriels pour développer les logiciels et les dispositifs matériels qui utiliseront le *JPEG* 2000.

Quel est l'algorithme utilisé ?

Cet algorithme typique de codage *JPEG* 2000 se décompose essentiellement en 5 modules:

- transformée couleur
- transformée en ondelettes discrète
- quantification
- codage entropique
- allocation de débit.

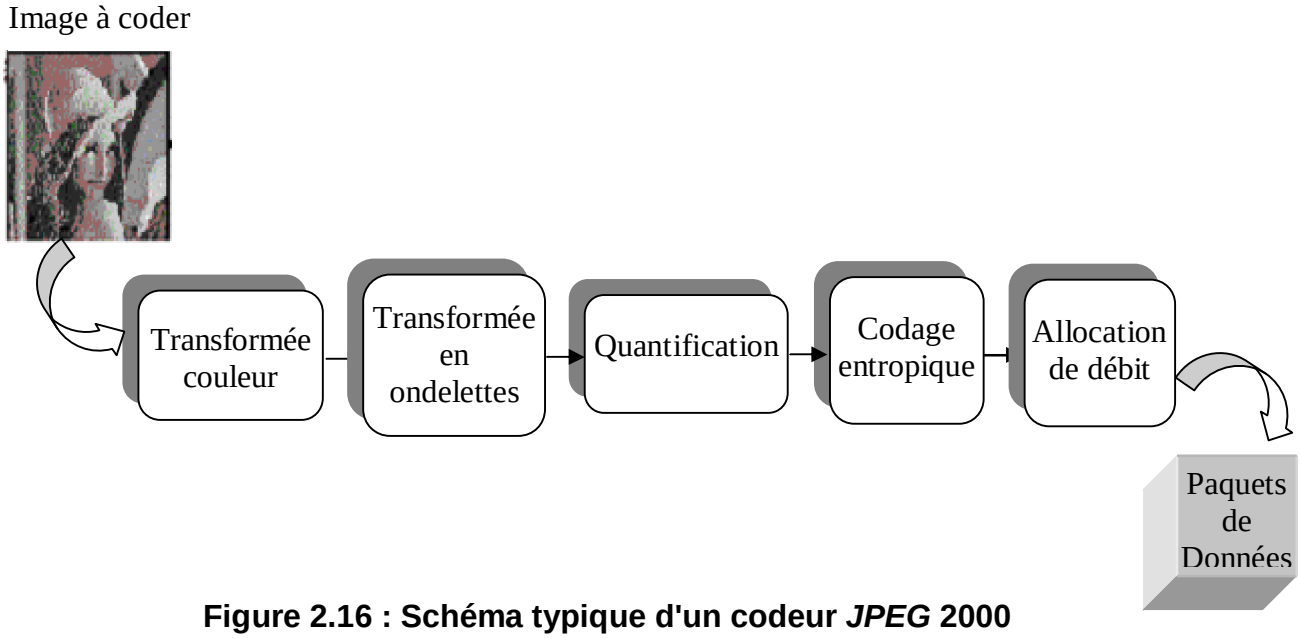


Figure 2.16 : Schéma typique d'un codeur *JPEG* 2000

Pour plus de renseignements sur le standard de compression *JPEG* 2000 vous pourrez lire l'article [21].

6.2.6 La méthode de compression fractale

Les méthodes fractales reposent sur l'idée que l'on peut identifier des objets mathématiques de type fractal dans l'image. Ceux-ci sont composés de façon récursive des copies d'eux-mêmes. Le codage par fractale consiste à repérer des zones de l'image qui peuvent être déductibles par une transformation géométrique d'une autre zone de taille différente [17].
(La méthode est bien détaillée dans le chapitre 3).

7 ETUDE COMPARATIVE DES DIFFERENTES METHODES DE COMPRESSION PRINCIPALE

la méthode / propriétés	JPEG	Ondelettes	Compression Fractale
Décomposition	Fréquentielle	Fréquentielle et Spatiale Multi résolution	Limite itérée d'une application contractante
Temps de Compression	1 s	5 s	120
Temps de décompression	1 s	1 s	1 s
Ratio pour une image de bonne qualité	8	12	9
Ratio pour une image de mauvaise qualité	30	60	40

Tableau 2.1 : Synthèse des différents algorithmes

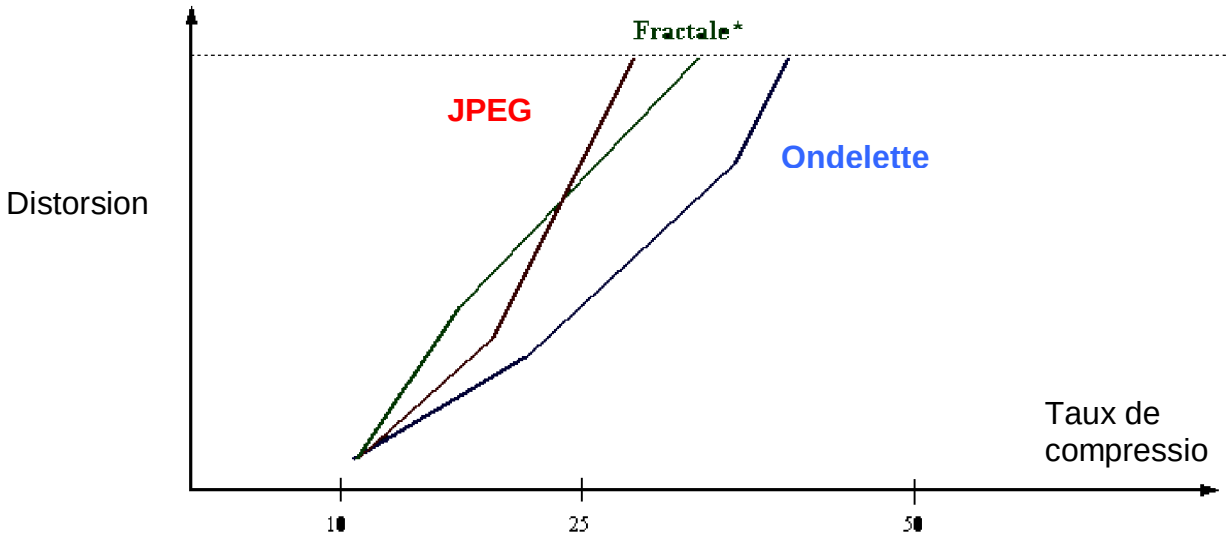


Figure 2.17: Comparaison des méthodes en fonction (d'erreur RMSE, et taux de compression)

8 CONCLUSION

La compression des données est appelée à prendre un rôle encore plus important en raison du développement des réseaux et du multimédia.

Nous ne nous sommes intéressés au cours de ce chapitre qu'aux méthodes de compression des images fixes (*Huffman*, *LZW*, …, *JPEG*). Des recherches très actives se poursuivent évidemment dans le domaine des images animées et du son. Nous avons présenté les différentes méthodes de codage de bases et de compression réversible où irréversible des images.

La compression d'images utilisant les fractales est l'une des applications les plus récentes de la géométrie fractale. Cette nouvelle méthode sera détaillée dans le chapitre suivant.

BIBLIO AZZIZ

[1] K.Belloulata , R.Stssinski et J. Konrad Region-based image compression using fractals and shape-adaptive DCT . IEEE PROCESSING , ICIP-99 Oct. 25-28 ,1999 Kobe , Japan.

[2] P. Beaurepaire, Epouse Beretta. " Compression D'images Appliquee Aux Angiographes Cardiaques : Aspects Algorithmiques, Evaluation de La Qualite Diagnostique " , Thèse N° 97 ISAL 0107 Institut National De Sciences Appliquees De Lyon, 1997.

[3=15 ch1] K. Chakib . La Compression des Images Fixes par les Approximations Fractales Basée sur la Triangulation de Delaunay et la quantification Vectorielle. Année 1999.

[4] F. Charot. Architectures Parallèles Spécialisées pour le Traitement D'image .INRIA , N° 1978, IRISA Campus Universitaire de Beaulieu France, AVRIL 1993 .

[5] A. Czihó . Quantification et Compression D'image. Application à l'imagerie médicale , Thèse N° 2142, Universite de Rennes 1, Mai 1999

[6] A.Francoise. « Mise en ligne d'images 3D traitements » UVprojet V0.1 2001.

[7] J. M. Pascal. " La Compression des Données ou le flux Multimédia sous Contrainte " DÉCOUVERTE N° 283 , Décembre 2000

[8] E. Phuc . Compression Selective et Focalisation Visuelle : Application au codage hybride de sequences d'images . Thèse N° 1434 , Universite de Rennes 1, Décembre 1995.

[9] S. Renard. "Compresser les images " INTERGRAPHIC 2000 Séminaires , Formation Les nouveaux savoir-faire Palais des Congrès - Porte Maillot – Paris Janvier 2000

[10] D.Santacruz , R.Grosbois et T.ebrahimi " JPEG 2000 - la nouvelle norme pour le codage d'images" *© FI-3-1 du 27 mars 2001*

[12] A. Gersho and R. M. Gray. Vector Quantization and Signal Compression .Kluwer Academic Publishers.1993.

[13] P. G. Howard and J. S.Vitter. Arithmetic coding for data compression. Proceedings of the IEEE, 82(6) :857-865,1994.

[14] D. A. Huffman. A method for the construction of minimum_redundancy codes.

Proc. of the IRE,40:1098-1101, September 1952.

[15] M. Nelson. La compression de données :texte, images , sons. Editions Dunod,1993.

[16] P.Cédric. F.Olivier. la compression d'image par ondelettes. Lycée Pothier MPSI 3. T.I.P.E. d'informatique 1998.

[17] J. PUGLIESI, C.PIOVANO , le tatouage d'images où "watermarking" .Université de Nice - Sophia Antipolis Licence d'Informatique Travail d'études. Juin 2004

[18] A.GERSHO et M. GRAY. *Vector quantization and signal compression*. Kluwer Academic,1992

[19] S.Renard. Compression de données. Conférence du 14 octobre 1999

[20] M. AMMAR. « Optimisation D'un Schéma De Codage D'image à Base D'une TCD. Application à Un Codeur JPEG Pour L'enregistrement Numérique à Bas Débit ». Thèse Présentée pour obtenir le grade de Docteur de l'Ecole Nationale Supérieure des télécommunications Spécialité : Signal et Images..le 14 Janvier 2002 .

[22] C. Stalder .Les algorithmes de compression sans perte. 7 janvier 2005.

[23] A. Skodras, C. Christopoulos, and T. Ebrahimi. The jpeg 2000 still image compression standard. Signal Processing Magazine, 18 :36{58, September 2001.

[24 M.Feil, R. Kutil, et A. Uhl. Parallel Wavelet Transforms on Multiprocessors: Architectures and Algorithms for Vision and other Senses .RIST++ & Department of Scientific Computing University of Salzburg, AUSTRIA.

[25] Mr. Brault. Mme. Dougherty. Les formats de compression d'image. Institut Universitaire de Technologie de Tours Département Génie Électrique et Informatique Industrielle. 2004.

[26] G.MERCIER, C.ROUX, G.MARTINEAU « Technologies du Multimédia » , ENST Bretagne, dpt ITI, BP 832, F-29280 Brest, France.15 janvier 2 003.

[27] A. FRANCOISE , Traitements Et Mise En Ligne D'images 3d , Version : 1.0, 03/05/2001 .

Chapitre 2l'étude bibliographiques des méthodes de compression

Table des matières

2.6 Présentation des méthodes avec et sans pertes :

2.7Conclusion

Table Des Figures

1 INTRODUCTION

La compression fractale des images est une technique relativement récente puisqu'elle a été proposée pour la première fois par *M.Barnsly* en 1988 « *fractals everywhere* ». En effet L'idée consiste donc à changer la représentation de l'image : au lieu de la représenter à l'aide de pixels, on le fait à l'aide des coefficients qui permettent son calcul. Ces deux représentations sont équivalentes d'un point de vue du rendu visuel, mais l'une d'elles est infiniment plus concise que l'autre.

Nous commençons ce chapitre par un bref aperçu de quelques notions mathématiques relatives à la géométrie des fractales, et qui forme la justification de cette nouvelle méthode de compression. Nous détaillerons ensuite la méthode originale de compression par une approche fractale, et toutes les améliorations faites aux niveaux des partitionnements de l'image, et ceci dans le contexte d'améliorer l'algorithme général de la compression fractale.

2 THEORIE DES SYSTEMES DE FONCTIONS ITEREES (*IFS*)

Nous rappelons dans cette section les principaux aspects de la théorie des *IFS* permettant le codage et la synthèse d'images fractales binaires et en niveaux de gris.

2.1 Les outils mathématiques

Les outils mathématiques sur lesquels a été fondée la géométrie des fractales sont :

- La théorie de l'espace,
- , La théorie d'itérations,
- *f* Les notions de topologie (géométrie de situation).

La nécessité d'évaluer quantitativement le rapprochement entre deux objets, est une raison d'introduire et d'utiliser les notions de l'espace métrique (distance) et des transformations.

2.2 Quelques définitions dans l'espace (rappels sur les espaces métriques)

Soit *X* un espace sur lequel on veut dessiner nos fractales. Qu'est-ce qu'une fractale? , jusqu'à présent, c'est un sous ensemble de l'espace.

v **Définition 1 : Métrique ou distance :** Une fonction à valeurs réelles *d* définie sur *X*x*X*, est appelée une distance ou une métrique sur *X* si elle vérifie, pour tout *a*, *b*, *c* ∈ *X*, les axiomes suivants :

- $d(a, b) \geq 0$ et $d(a, a) = 0$. (3.1)

, $d(a, b) = d(b, a)$ (symétrie). (3.2)

f $d(a, c) \leq d(a, b) + d(b, c)$ (Inégalité triangulaire). (3.3)

„ Si $a \neq b$ alors $d(a, b) > 0$. (3.4)

Le nombre réel $d(a, b)$ est appelé *la distance* de *a* à *b*.

- Tout espace *X* muni de la métrique *d* est appelé ***espace métrique*** et noté (*X*, *d*).
- Une suite $\{X_n\}_{n=1,\infty}$ de points dans l'espace métrique (*X*, *d*) est appelée une suite de Cauchy, si : $\forall\ \varepsilon > 0$, $\exists\ N$ entier, $N > 0$ tel que $d(x_n, x_m) < \varepsilon$, $\forall\ n, m > N$. (3.5)

v **Définition 2 : espace métrique(*X*, *d*) complet :** un espace métrique(*X*, *d*) est dit complet, si chaque suite de Cauchy dans *X* a une limite $x \in X$. C'est à dire : $\{X_n\}_{n=1,\infty}$ converge vers *x*. En d'autres termes, $\exists\ N > 0$ tel que $d(x_n, x_m) < \varepsilon$, $\forall\ n > N$. (3.6)

v **Définition 3 : L'espace métrique *h*(*X*) :** C'est l'espace où l'on étudie la géométrie des fractales pour traiter les images, des dessins. Il est nécessaire d'introduire *l'espace H.* La métrique utilisée dans l'espace *H(X)* est la distance de *Haussdorf,* notée *h.*

v **Définition 4 : La distance de *Haussdorf*** entre deux ensembles *A* et *B* de *H*(*X*), noté $h_d(A, B)$, est définie par :

$h_d(A, B) = max\{d(A, B), d(B, A)\}$. (3.7)

Où : $d(A, B) = max\{d(x, B) : x \in A\}$. (3.8)

Et $d(x, B) = min\{d(x, y) : y \in B\}$. (3.9)

2.3 Transformation des espaces métriques

2.3.1 Transformation affine

- **Dans la ligne réelle (*IR*) :** Les transformations affines dans *IR* sont des transformations

  $f : R \to R$ de la forme: $f(x) = ax + b \ \forall x \in IR$, a, b constantes réelles. (3.10)

- **dans le plan euclidien (IR^2) :** Les transformations affines dans IR^2 sont des transformations de la forme :

  $$f(x,y) = (ax+by+e, cx+dy+f) \ \forall (x,y) \in IR^2. \quad (3.11)$$

  Où a, b, c, d, e, et f sont des constantes réelles. f est dite transformation affine à deux dimensions.

2.3.2 Transformation lipschitzienne

Soit *:* $W \ IR^2 \to IR^2$ une transformation définie sur l'espace métrique (IR^2, d), d représente la distance entre deux points de IR^2. La transformation W est dite lipschitzienne, avec pour facteur de lipschitz le réel s strictement positif si :

$$d(W(x),W(y)) \leq s.d(x,y) \ \forall (x,y) \in IR^2. \quad (3.12)$$

2.3.3 Transformation contractante

Soit $W : IR^2 \to IR^2$ une transformation définie sur l'espace métrique (IR^2,d). La transformation W est dite contractante, avec pour facteur de contraction le réel s, $0 < s < 1$ si :

$$d(w(x), w(y)) \leq s.d(x, y) \quad \forall x, y \in R^2 \quad (3.13)$$

2.3.4 Transformation contractante sur l'espace *h* (IR^2)

Soit une transformation contractante $W : IR^2 \to IR^2$ définie sur l'espace métrique (IR^2,d) avec le réel s pour facteur de contraction. La transformation : $W : H(IR^2) \to H(IR^2)$ définie par :

$$W(B) = \{W(x) : x \in B\}, \ \forall B \in H(IR^2), \quad (3.14)$$

Elle est contractante sur ($H(IR^2)$,h_d), avec s pour facteur de contraction, h_d désigne la distance de *Haussdorff*.

2.3.5 Point fixe

Une transformation contractante W possède un unique point fixe $x_f \in IR^2$, tel que $W(x_f) = x_f$. Pour tout point x élément de IR^2, la séquence $\{W^{0n}(x) : n=0, 1,2,...\}$ converge vers x_f.

2.4 Théorie des *IFS* et compression par fractales

La théorie des *IFS* est une extension de la géométrie classique. Elle utilise un ensemble de transformations affines pour exprimer des relations entre les parties d'une même image.

2.4.1 Définition de *L'IFS*

Un *IFS* consiste en une collection de transformations élémentaires géométriques souvent linéaires ou affines, contractantes dans un espace métrique (*X*,*d*) qui permet de transformer une image en une autre image :

$\{w_i : IR^2 \to IR^2 \mid i = 1,2,...,n\}$ Qui projettent le plan IR^2 sur lui-même. Cette collection de transformations affines définit une fonction :

$$W/W(A) = \bigcup_{i=1}^{n} w_i(A) = w_1(A) \cup w_2(A) \cup ... \cup w_n(A) \qquad (3.15)$$

Avec *A* un sous ensemble du plan $\subset X$.

- La contractivité des *Wi* se traduit par :

  $$(\forall i = 1...n)\,(\exists s \in R, o < s_i < 1)\,(\forall x, y \in X)\, d(w_i(x), w_i(y)) \le s_i d(x, y) \qquad (3.16)$$

  Une fonction W_i vérifiant la relation de contractivité avec s_i <1 est dite Lipschitzienne, et s_i le facteur de lipschitz.

- Les *Wi* peuvent être décrites comme une combinaison de:
  - ü rotation ;
  - ü de réduction d'échelle ;
  - ü de translation de cordonnées des axes dans un espace à n dimension(s).

$$w_i = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \times \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix} \qquad (3.17)$$

Où *a, b, c, d, e,* et *f* représentent des paramètres de transformation :

$a = r\cos\theta$, $b = -s\sin\varphi$, $c = r\sin\theta$, $d = s\cos\varphi$ (3.18)

- ü *r* : facteur de réduction sur *X*
- ü *S* : facteur de réduction sur *Y*
- ü θ : est l'angle de rotation sur *X*
- ü φ : est l'angle de rotation sur *Y*
- ü *e* : est la translation en *X*
- ü *f* : est la translation en *Y*

- *W* possède un unique *point fixe***,** appelé *attracteur* de l'*IFS*. il est égal à l'union de copies réduite de lui-même (figure 3.1).
- Le point fixe (image) qui est obtenu possède quelques caractéristiques spécifiques. En particulier, il effectue de copies d'image de lui-même mais modifiées par des transformations élémentaires. Dans l'exemple suivant, le système est fait de différentes transformation : une réduction, suivie d'une translation parmi trois qui assurent le repositionnement sous forme de triangle. Le point fixe, indépendant du motif de départ, est le triangle de *Sierpinski*.

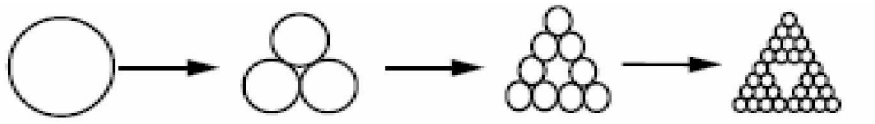


Figure3.1 : Illustre le point fixe d'une transformation affine contractive par la répétition de l'application *W*

2.4.2 Attracteur d'un *IFS*

Soit un *IFS* {IR^2; *Wi*, *i*=1,2,..., *n*}. L'opérateur $W : H(IR^2) \to H(IR^2)$ défini par:

$W(A) = \bigcup_{i=1}^{n} W_i(A), A \in H(IR^2)$ est contractant et a pour facteur de contraction celui de *l'IFS*.

L'opérateur *W* possède un unique point fixe x_f donné par :

$$x_f = W(x_f) = \lim_{n \to \infty} W^{on}(x), \forall x \in H(IR^2). \qquad (3.19)$$

L'objet x_f est appelé *"attracteur de l'IFS"* et il est égal à l'union de *n* copies de lui même transformées par W_1, W_2,…, W_n.

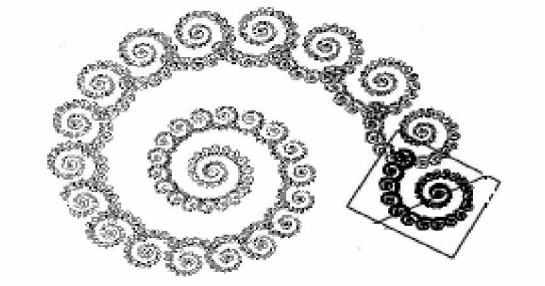


Figure 3.2 : l'attracteur de *l'IFS*

2.4.3 Théorème du collage

Soit (X,d) espace métrique complet.

Soit $I \in H(X) et \varepsilon > 0$.

Alors $\exists \{X; w_n : n = 1,2,...,N\} IFS$ avec un facteur de contraction *s*, tel que $h_d(I, W(I)) \leq \varepsilon$, h_d étant la distance de *Hausdorff*. (3.20)

Alors $h_d(I, A) \leq \dfrac{\varepsilon}{(1-s)}$, avec *A* attracteur de l'*IFS*. (3.21)

On vérifie alors la relation :

$$h_d(I, A) \leq \frac{1}{(1-s)} h_d\big(I, W(I)\big) = \frac{1}{(1-s)} h_d\left(I, \bigcup_{n=1}^{N} w_n(I)\right). \quad (3.22)$$

Ce théorème donne une borne supérieure à la distance de *Hausdorff* entre tout point de $H(X)$ et l'attracteur d'un *IFS*. De plus, il indique que l'on peut transformer tout élément I de $H(X)$ de manière à vérifier $h_d(I, W(I)) \leq \varepsilon$, c'est-à-dire $I \approx W(I)$, avec l'opérateur contractant W [60].

2.4.4 Système de fonctions itérées partitionnées (*PIFS*)

Nous rappelons que «*comprimer*» une image par fractales est l'opération par laquelle, à partir d'une image I de $H(X)$, il faut trouver un opérateur W tel que $W(I) \approx I$, avec l'espoir implicite de pouvoir écrire W d'une manière plus compacte que celle de l'image initiale. Cette opération est un problème inverse très difficile à résoudre de manière automatique pour des images. La solution la plus courante consiste à partitionner l'image de deux manières différentes pour obtenir deux familles de blocs.

Le théorème du collage donnera la distance entre *I* et le point fixe *A* de *W* à travers la relation

$$h_d(I, A) \leq \frac{1}{(1-s)} h_d\big(I, W(I)\big) \quad (3.23)$$

Dans ce cas, chaque transformation w_n est restreinte à un domaine inclus dans *X*. Nous avons donc $w_n : D_n \to X$.

- Les sous-ensembles D_n sont appelés blocs sources ou *domain blocks*.

De plus, un *IFS* (ou *IFS* partitionné —*PIFS*—) d'attracteur *A* est dit sans recouvrement si chaque transformation (bijective) $w_n : X \to X$. vérifie :

$$\forall i, j \in \{1,2,...,N\}, i \neq j \quad w_i(A) \cap w_j(A) = \phi. \tag{3.24}$$

Dans le cas des images, l'ensemble $\{w_n(A) = R_n ; n = 1,2,...,N\}$définit une partition de $H(X)$. Nous avons finalement $w_n : D_n \to R_n$.

- Les sous-ensembles R_n sont appelés blocs destinations où *range blocks*.

Les algorithmes de compression par fractales se distinguent par la stratégie du partitionnement des blocs sources et destinations, et par l'écriture des transformations w_n.

2.4.5 Problème inverse

Soit une image I de $H(X)$ connue, et une mesure de distance sur $H(X)$.

La compression d'image par fractales se présente comme la résolution d'un problème inverse sous contrainte, Alors il faut trouver un *IFS W* contractant dont le point fixe est I.

La recherche de cet *IFS* à partir d'un objet donné est un exercice complexe, de grande dimensionnalité et de minimisation de fonctions irrégulières. Plusieurs outils ont été utilisés pour résoudre ce problème (dans le cas d'un signal «fractal»), à partir d'algorithmes génétiques (sur des images binaires [61]), et de transformée en ondelettes continue (en *1D* [59], [62]). Le cas des images est encore plus complexe. C'est un signal à deux dimensions dont les éléments $f(x)$ représentent des niveaux de gris. De plus, les images naturelles ou de télédétection ne sont pas fractales. Le caractère éventuellement auto-similaire du signal sera donc étudié dans un contexte rigide défini a priori :

l'auto-similarité est étudiée à travers des transformations affines (le terme auto-affinité serait alors plus approprié), et la recherche de ces similarités est effectuée dans des familles de blocs issues de partitions précises. La solution est presque sûrement sous optimale, mais elle est toujours atteinte en un temps fini. Alors il faut trouver un *PIFS W* finalement contractant tel que son point fixe soit une approximation fidèle de I.

Différents travaux ont tentés de résoudre ce problème d'optimisation sous contraintes qui est difficile de par sa grande dimensionnalité et l'irrégularité de la fonction à minimiser. Les solutions envisagées utilisent des techniques diverses basées sur les algorithmes génétiques [48], sur les ondelettes [51], et autres : [40], [47], [49], [42], [55], [46].

2.4.6 La limite des *IFS*

Les *IFS* ont montré leur bonne performance en compression (des taux très élevés), ce qui est très important, cependant, deux aspects montrent leur insuffisance :

- les images compressées sont des figures typiquement fractales (similarité à tous les niveaux), alors que les images du monde réel tel que le visage d'un homme ou une maison ne sont pas tout à fait fractals (auto-similarité par régions).
- Ce sont des images à deux niveaux : noir et blanc, alors qu'on veut coder les images à plusieurs niveaux de gris.

Pour surmonter ce problème, on fait appel aux *PIFS* (*Partitionned Iterated Function System*) qui constituent une extension des *IFS* et qui peuvent donc coder n'importe quelle image

3 LA COMPRESSION DES IMAGES PAR FRACTALES

Pour pouvoir exploiter la similarité entre les régions d'une image quelconque, on va utiliser une méthode qui complémente la méthode des *IFS* et qui introduit le partitionnement et la génération des niveaux de gris, c'est la méthode des *PIFS*.

3.1 Qu'est–ce qu'une Fractale ?

Une fractale est un sous-ensemble *A* d'un espace métrique *X*, tel que *A* est extrêmement compliqué géométriquement. Le terme "*fractal*" est issu du latin "*fractus*" qui signifie brisé, irrégulier. Un objet fractal est une forme géométrique se répétant infiniment et dont les détails irréguliers se reproduisent avec des angles et des échelles différentes.

Un objet fractal possède au moins l'une des caractéristiques suivantes :

- il a des détails similaires à des échelles arbitrairement petites ou grandes,
- il est trop irrégulier pour être décrit efficacement en termes géométriques traditionnels,

- il est exactement ou statistiquement autosimilaire, c'est-à-dire que le tout est semblable à une de ses parties

Les fractales peuvent être réparties en trois grandes catégories :

1. *Les systèmes itérés de fonctions.* Ceux-ci ont une règle de remplacement géométrique fixe

(l'ensemble de *Cantor*, le tapis de *Sierpinski*, le triangle de *Sierpinski*, la courbe de *Peano*, le flocon de *Koch*).

2. *Les fractales définies par une relation de récurrence* en chaque point dans un espace (tel que le plan complexe). Des exemples de ce type sont les ensembles de *Mandelbrot* et le fractal de *Lyapunov*.

3. *Les fractales aléatoires*, sont les plus utilisées dans la pratique, par exemples les paysages fractals.

De toutes ces fractales, seules celles construites par des systèmes itérés de fonctions affichent habituellement la propriété d'«*autosimilitude*», propriété signifiant que leur complexité est invariante par changement d'échelle.

3.2 Pourquoi "la compression fractale d'images" ?

Les méthodes standards de compression d'images peuvent être évaluées par leur taux de compression : le quotient de la mémoire occupée par l'image comme collection de pixels par la mémoire nécessaire pour stocker l'image sous forme compressée. Pour répondre à cette question nous présentons un exemple classique des *IFS* de la fougère (la feuille) de *Barnsley* voir figure 3.3. Cette feuille est obtenue en itérant les transformations suivantes [37]:

$$
\begin{aligned}
w_1\begin{bmatrix}x\\y\end{bmatrix} &= \begin{bmatrix}0.085 & 0.04\\ -0.04 & 0.85\end{bmatrix}\times\begin{bmatrix}x\\y\end{bmatrix}+\begin{bmatrix}0.02\\0.08\end{bmatrix}\\
w_2\begin{bmatrix}x\\y\end{bmatrix} &= \begin{bmatrix}-0.13 & 0.24\\ 0.22 & 0.20\end{bmatrix}\times\begin{bmatrix}x\\y\end{bmatrix}+\begin{bmatrix}0.12\\-0.27\end{bmatrix}\\
w_3\begin{bmatrix}x\\y\end{bmatrix} &= \begin{bmatrix}0.18 & 0.24\\ 0.21 & 0.20\end{bmatrix}\times\begin{bmatrix}x\\y\end{bmatrix}+\begin{bmatrix}0.12\\-0.30\end{bmatrix}\\
w_4\begin{bmatrix}x\\y\end{bmatrix} &= \begin{bmatrix}0.0 & 0.0\\ 0.0 & 0.16\end{bmatrix}\times\begin{bmatrix}x\\y\end{bmatrix}+\begin{bmatrix}0.0\\-0.42\end{bmatrix}
\end{aligned}
\qquad (3.25)
$$

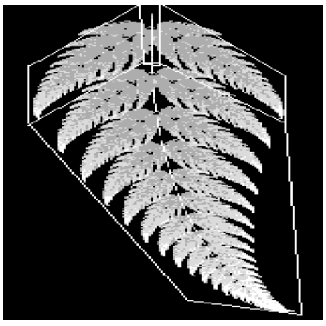


Figure 3.3 : La feuille de *Barnsley*

Sur une image de départ quelconque. Le taux d'information nécessaire au stockage de ces transformations est assez faible:

- 4 transformations *x* 6 nombres par transformation *x* 32 bits par nombre =768 bits
- l'image de fougère de taille 256*x*256 nécessite au moins 65536 bits à stocker comme collection de pixels

Le taux de compression estimé est: 65536/768=85.3.

Il est clair que la technique de compression par fractales d'images est plus fiable que la méthode *JPEG*, car la norme de compression *JPEG* ***dégrade*** l'image lorsque le taux de compression est élevé, mais la compression fractales quelque soit le taux produise une image fiable à l'originale avec une représentation indépendante de l'échelle.

3.3 La notion d'auto-similarité

Un objet fractal est dit self-similaire ou auto-similaire s'il est composé de *N* copies de lui même, chacune étant une réduction d'un facteur *"r"* selon toutes les coordonnées de l'objet global [4], [38] (auxquelles des transformations affines, comme les translations ou rotations peuvent ou non être impliquées).

La notion d'auto-similarité revient dans plusieurs contextes car les objets naturelles sont auto-similaires (ou presque) : "les montagnes sont formés de piques qui sont à leurs tours formés de pics plus petits". La figure 3.4 illustre cette notion d'auto-similarité par un exemple de triangle de *Sierpinski* : Ce triangle est constitué de trois triangles qui sont une réduction d'un facteur d'échelle de l'ensemble, et chacun des trois triangles est lui même constitué de trois autres triangles qui sont une réduction d'un facteur ...etc.

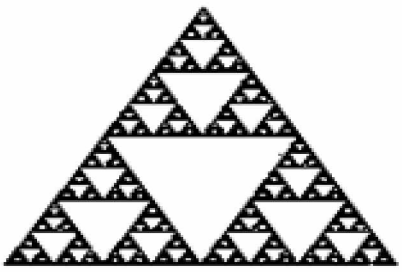


Figure 3.4 : Triangle de *Sierpinski*.

3.4 Auto-similarité dans les images

Les images réelles ne contiennent pas le même type d'auto-similarité que celui des fractales tel que le triangle de *Sierpinski* ou la fougère de *Barnsley*, où l'image est constituée de l'union des répliques réduites de l'image entière.

(La figure 3.5) montre des exemples de régions de la photo de *Lena* qui sont similaires, à différentes échelles : une portion de son épaule recouvre une région qui est presque identique, et une portion du reflet du chapeau dans le miroir est similaire (après transformation) à une autre partie de son chapeau.

Ces parties transformées ne peuvent pas être réunies, en général, pour former une copie exacte de l'image originale, et on doit donc permettre une certaine erreur à notre représentation de l'image en tant qu'ensemble de transformations. Ceci signifie que l'image que nous coderons comme ensemble de transformations ne sera pas une copie identique à l'originale, mais plutôt une approximation de celle-ci.

Figure 3.5 Parties auto similaires dans l'image de Lena

3.5 Principe de la compression *IFS* et *PIFS*

La théorie des *IFS* est une extension de la géométrie classique. Elle utilise un ensemble de transformations affines pour exprimer des relations entre les parties d'une même image. En utilisant seulement ces relations, on peut exprimer et définir des images complexes (images, feuille, arbres, …etc.).

Le principe de la méthode de compression dite par *IFS* ou fractale est l'exploitation des auto similarités des images naturelles. Où l'auto similarité désigne la propriété qui fait que certains objets peuvent être construits à partir de parties d'eux même moyennant des transformations spécifiques. En effet, au lieu de rechercher la corrélation entre les pixels adjacents, on s'intéresse à des corrélations entre parties plus au moins espacées dans l'image.

@ La problématique de la compression fractale est donc, pour une image donnée *A*, de trouver un *IFS* dont l'attracteur soit le plus proche possible de *A*.

L'algorithme de base consiste à *partitionner* d'abord *A* en *blocs destination*. Pour chacun de blocs, on cherche ensuite, toujours dans *A*, un *bloc «source»*, qui par une transformation

contractante simple (translation et rotation spatiale, changement d'échelle, modification du contraste et du niveau de gris moyen) donne une bonne approximation du bloc destination.

Quand tous les blocs destination ont été traités, on a obtenu un ensemble de fonctions contractantes qui constitue un *IFS* dit «*partitionné*» dont l'attracteur approxime *A*. On peut ainsi atteindre des taux de compression importants en conservant une bonne qualité d'image.

3.6 Le codage fractale par bloc (transformation fractale)

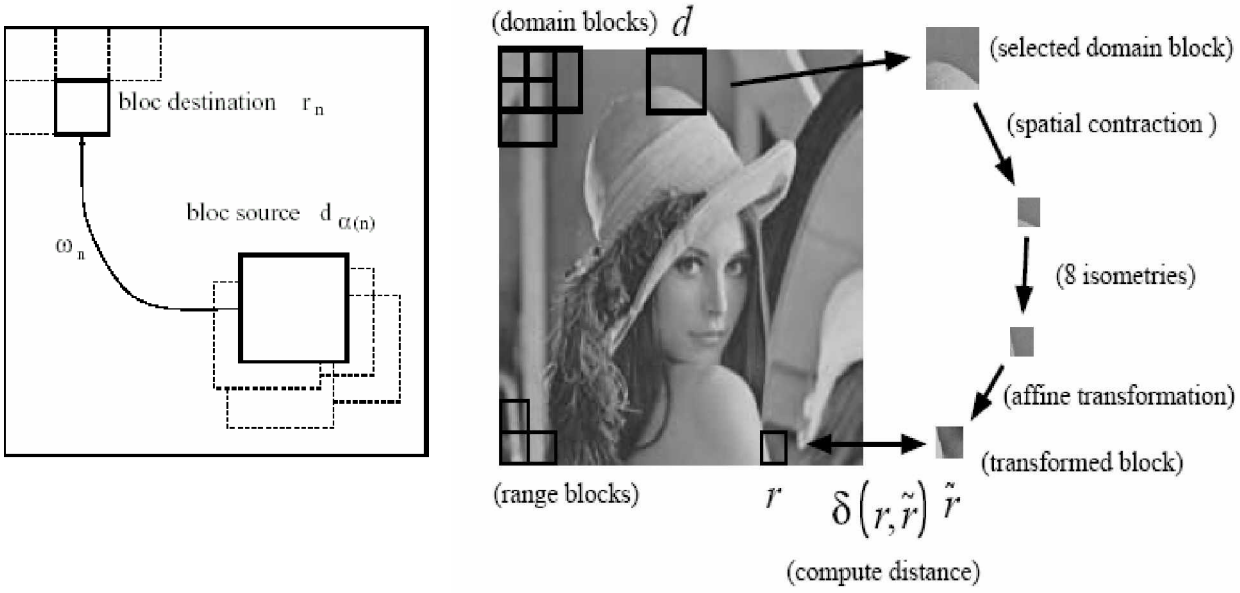


Figure 3.6 : principe de transformation fractale (application sur l'image de Lena)

La compression d'une image par fractales repose sur une transformation qui consiste à transformer l'image à l'aide d'un opérateur finalement contractant *W*, de manière à ce que son aspect visuel reste quasiment inchangé. Pour cela, la transformation de l'image est composée de *n* sous transformations élémentaires, chacune opérant sur un bloc de l'image, de la manière suivante: voir (la figure 3.6)

L'image *A* est partitionnée en *N* blocs r_n appelés blocs destination (blocs *range*).

Elle s'écrit : $A = \bigcup_{i=1}^{n} r_i$ (3.26)

Cette partition du support de l'image en blocs destination est appelée partition *R*.

Chaque bloc destination est ensuite mis en correspondance avec un autre bloc transformé $w_i(d_i)$ lui "*ressemblant*" au sens d'une mesure d'erreur sur les niveaux de gris. Le bloc d_i, appelé *bloc source* est recherché au travers d'une librairie composée de *Q* blocs appartenant à

l'image. Les *Q* blocs ne forment pas nécessairement une partition de l'image, mais ils sont représentatifs de toute l'image. La géométrie de la région d_i doit être proche de celle du bloc r_i , de façon à limiter les temps de calcul lors des comparaisons inter-blocs, puisque ceux-ci sont liés à la complexité de la transformation spatiale utilisée.

Une seconde contrainte impose que les blocs source soient en moyenne de surface supérieure à celle des blocs destination.

La transformation *W* de l'image *A* est formulée à l'aide de l'équation suivante :

$$W(A) = \bigcup_{n=1}^{N} w_n(d_{\alpha n}) = \bigcup_{n=1}^{N} r_n \approx \bigcup_{n=1}^{N} \hat{r}_n = A \tag{3.27}$$

Où $\hat{r}$ est l'approximation du bloc destination r_i obtenue en transformant le bloc source d_i par w_i (l'opération permettant d'obtenir le bloc $\hat{r}$ à partir du bloc d_i est appelée opération de *collage***).**

Chaque transformation w_i s'écrit sous la forme :

$$w_i\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a_i & b_i & 0 \\ c_i & d_i & 0 \\ 0 & 0 & s_i \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} e_i \\ f_i \\ o_i \end{bmatrix} \tag{3.28}$$

Où :

- s_i : contrôle *le contraste***.**
- o_i ***:*** *la luminance.*
- z : *Niveau de gris.*

Ce qui exige de calculer autant de fois les paramètres de la transformation affine W_i , *s*, *o* et d_{rms}:

$$s = \frac{n \cdot \left(\sum_{i=1}^{n} d_i \cdot r_i\right) - \left(\sum_{i=1}^{n} d_i\right) \cdot \left(\sum_{i=1}^{n} r_i\right)}{n \cdot \sum_{i=1}^{n} d_i^2 - \left(\sum_{i=1}^{n} d_i\right)^2} \tag{3.29}$$

$$o = \frac{1}{n}\left(\sum_{i=1}^{n} r_i - s\sum_{i=1}^{n} d_i\right) \tag{3.30}$$

$$d_{rms} = E(D_i, R_i) = \frac{1}{n}\left[\sum_{i=1}^{n} r_i^{\,2} + s\left(s\sum_{i=1}^{n} d_i^{\,2} - 2\sum_{i=1}^{n} d_i r_i + 2o\sum_{i=1}^{n} d_i\right) + o\left(o \cdot n - 2\sum_{i=1}^{n} r_i\right)\right] \tag{3.31}$$

Pour que le codage par fractales soit efficace, il doit y avoir suffisamment de *similarités locales à diverses échelles* entre les blocs de la partition *R* et les blocs source d_i. Lorsque ceux-ci sont recherchés dans toute l'image, la condition est généralement vérifiée. Si la recherche se fait dans une partition contenant un nombre restreint de blocs, le codage n'est plus optimal, mais plus rapide. Le but est de trouver le meilleur compromis entre le nombre de blocs source disponibles pour les recherches de similarités, et la ressemblance entre les blocs de la partition *R* et les blocs d_i .Bien que dans la pratique, il est plus facile d'utiliser des blocs *carrés*, il est toujours possible d'utiliser d'autres formes telles que les rectangles, triangles, polygones,...etc.

Il existe plusieurs méthodes de compression fractale (subdivision de triangles, Jacquin, Fisher, Delaunay, etc.), mais la compression par la méthode Jacquin et la plus connue.

La plupart des algorithmes de compression fractale sont des variations sur ce type d'algorithme (de Jacquin), plus d'informations sur ces algorithmes dans [99].

v Transformation d'un bloc source à un bloc destination

L'opération de collage d'un bloc source $d_{\alpha(n)}$ sur un bloc destination r_n, réalisée par la transformation *Wn*, se décompose en deux parties :

- **une transformation spatiale** ramène le bloc source $d_{\alpha(n)}$ de taille D^2 à l'échelle et au-dessus du bloc destination r_n de taille B^2, et déforme le support du bloc $d_{\alpha(n)}$;
- **une transformation "massique"** agit sur la luminance des pixels du bloc $d_{\alpha(n)}$ déformé.

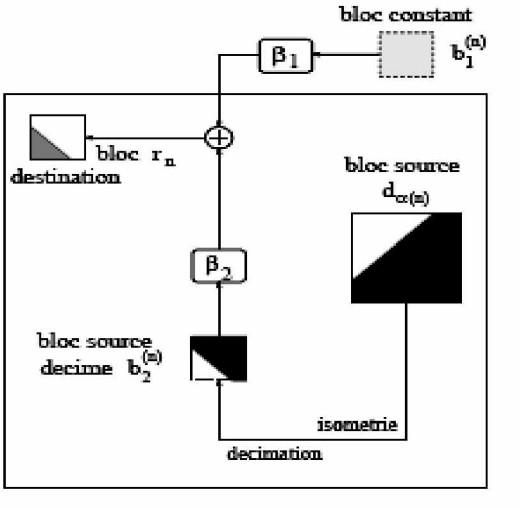


Figure 3.7: Transformation massique

L'approximation du bloc r est donnée par une combinaison linéaire de deux blocs, parmi lesquels le bloc $d_{\alpha(n)}$ est extrait de l'image elle-même et $b_1^{(\alpha)}$ est un vecteur unitaire. Elle s'écrit :

$$\hat{r} = \beta_2 b_2 + \beta_1 b_1 \qquad (3.32)$$

3.7 Schéma général d'un codeur - décodeur fractal

On peut à présent donner le schéma général (figure 3.8) et l'algorithme (pseudo-code) d'un codeur et décodeur fractal générique utilisant [98] :

- Une partition *R* où les blocs destinations $r_n = \{r_i,\ i = 1, n\}$
- Un dictionnaire où les blocs sources $d_{\alpha(n)}$

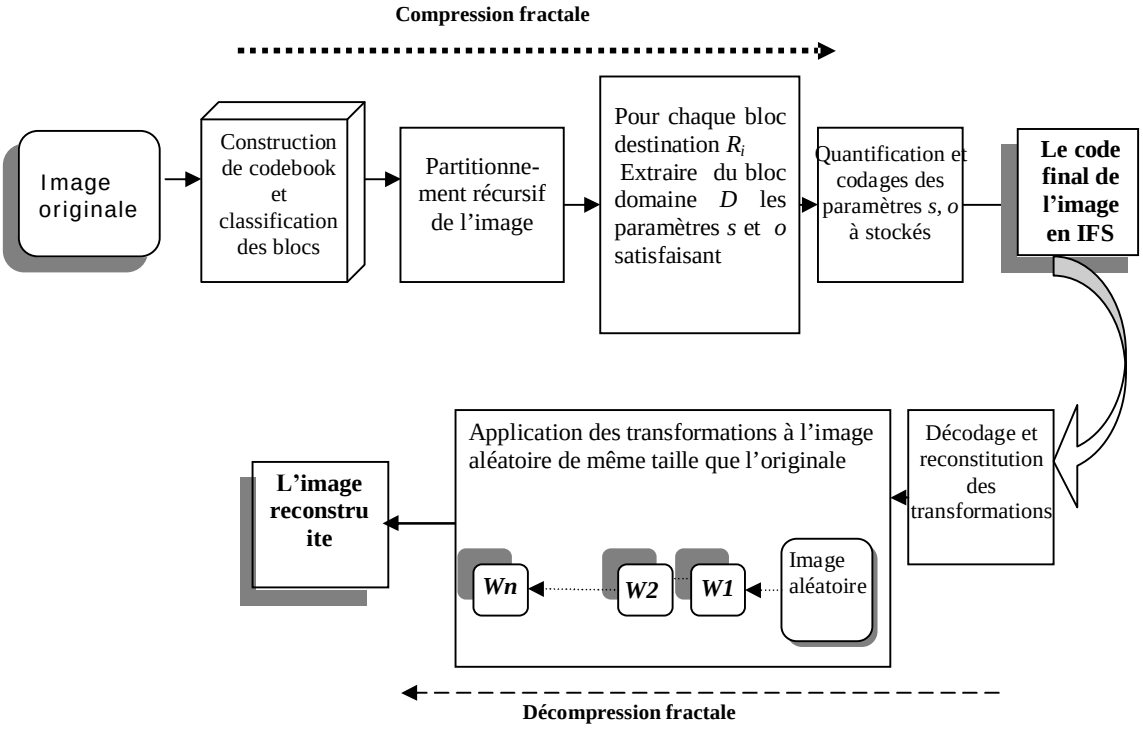


Figure 3.8 Schéma Général d'un Codeur et Décodeur Fractale.

v L'algorithme de Compression

Etape 1 : Lire l'image originale

Etape 2 : *Fixé* le seuil (l'erreur *RMS*)

Etape 3 : *Donner Tmin, Tmax* (taille min et Max du bloc à partitionner)

Etape4 : Construction du dictionnaire des blocs sources $d_{\alpha(n)}$ (domaines) et leurs classifications.

Etape 5 : Partitionnement (quadtree) et récursif de l'image.

Etape 6 : Pour chaque bloc destination (range r_i (i=1……n))

- Classifier le bloc destination r_i
- Rechercher dans la classe du bloc destination le bloc source $d_{\alpha(n)}$ qui satisfait la distorsion calculée < seuil fixé *RMS* (l'erreur quadratique)
- Si oui stocker les paramètres de la transformation W_i du bloc destination luminance : o, contraste : s, les cordonnées du bloc source,…
- Si non (aucun bloc ne satisfait la distorsion calculée < seuil *RMS*)
  - Si (la taille du bloc > T_{min} alors) décomposer le bloc destination a nouveau
  - Si non la taille du bloc destination inférieure à la taille T_{min}. En prend le bloc source (domaine) avec la plus faible erreur possible et ses paramètre sera stocker (*Wi* du bloc destination luminance: o, contraste: s, les cordonnées du bloc source)

Etape 7 : Quantification des paramètres stockés et codage

Etape 8 : Fermer la bibliothèque domaine

Etape 9 : Terminer le processus de compression (résultat est un fichier *IFS*).

∨ L'algorithme de décompression

Etape 1 : Lire ou ouvrir le fichier *IFS*

Etape 2 : *Décodage* et reconstitution des transformations

Etape 3 : *Charger* une image *A* aléatoire de même taille que l'image originale à compresser

Etape 4 : Pour chaque transformation W_i (i=1, N)

- Appliquer la transformation W_i à l'image *A*
- Reconstruire l'image *A*
- Passer à la transformation suivante (aller à l'étape 4)

Etape 5 : Traitement et Affichage de l'image résultante.

4 PARTITIONNEMENT DE L'IMAGE

Différentes partitionnements de l'image ont été étudiés dans [35], [11], [26], [39], [47], [53] de façon à minimiser le nombre de transformations locales, et à coder le mieux les similarités inter-blocs dans la compression à bases des fractales. Nous distinguerons trois classes de partitionnement:

G les partitionnements rigides

A les partitionnements semi-rigides

B les partitionnements souples

Nous limiterons ci-après à présenter quelques partitionnements selon une graduation allant du plus rigide au plus souple, (les détails sur les classes de partitionnement dans [40]).

4.1 Rôle du partitionnement pour la compression par fractales

Le but de la compression par fractales est de «*saisir*» la redondance visuelle locale à l'intérieur de l'image à l'aide de transformations contractantes. La transformation fractale est directement calculée sur une partition *R* de l'image, dont les propriétés recherchées sont les suivantes :

- la partition *R* doit être adaptée à l'image, de façon à minimiser le nombre de blocs ;
- la localisation et la manipulation informatique des blocs dans la partition doivent être facile ;
- l'information nécessaire à son codage doit être minimale.

Chacun des blocs destination r_i de la partition *R* est mis en correspondance avec une région source d_j de l'image, lui étant proche au sens des moindres carrés dans l'espace des niveaux de gris.

La géométrie de la région d_j doit être proche de celle du bloc r_i , de façon à limiter les temps de calcul lors des comparaisons inter-blocs, puisque ceux-ci sont liés à la complexité des transformations spatiales utilisées.

Une contrainte supplémentaire impose que les blocs source soient en moyenne de surface supérieure à celle des blocs destination. De cette manière, un certain nombre de transformations spatiales contractantes sont mises en jeu et assurent la contraction finale de la

transformation fractale. Dans la suite de ce chapitre, nous présenterons des partitionnements adaptatifs, rigides (quadtree), et semi-rigides (horizontal_vertical), ainsi que les partitionnements souples et leur utilisation pour la compression des images par fractales.

4.2 Partitionnement rigide (Quadtree)

Un partitionnement est qualifié de rigide s'il ne s'adapte pas à moindre coût aux formes des objets présents dans l'image.

Quadtree

Le quadtree ou *arbre quaternaire* est une structure initialement utilisée pour représenter des images binaires [53], [45]. La représentation est exacte lorsque le processus récursif de sous division des blocs carrés descend jusqu'à la taille du pixel.

Chaque bloc est alors composé de valeurs toutes égales à 1, ou toutes égales à 0.

Le quadtree peut aussi être utilisé pour la représentation des images en niveaux de gris. C'est ce dernier cas que nous considérerons dans la suite de cette section. Des études détaillées du quadtree sont données dans [52], [54].

v Phase de division

Considérons une image initiale de taille $2^m x 2^m$, que l'on note ${}^0A$. La construction du haut vers le bas (*top, down*) du quadtree consiste à diviser récursivement tout bloc ${}^lA_i$ non homogène selon un prédicat donné, en considérant le bloc ${}^0A$ comme une seule région de départ. *l* est le niveau de la représentation pyramidale au codage du quadtree.

L'indice *i* (*i*=1,… ,4) dénote le numéro du sous bloc.

La division d'un bloc ${}^lA_i$ de taille $2^{m-l} \times 2^{m-l}$ crée quatre sous blocs carrés ${}^{l+1}A_j\,(j = 1,...,4)$ de dimension 2^{m-l-1}. A chaque nouvelle division, les attributs des quatre blocs créés sont recalculés pour être à nouveau soumis au prédicat d'homogénéité.

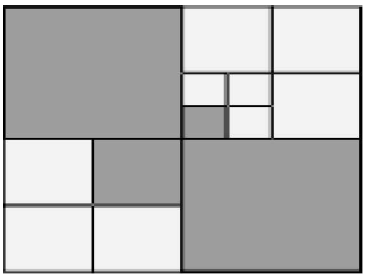


Figure 3.9 : Principe de la division récursive dune image binaire.

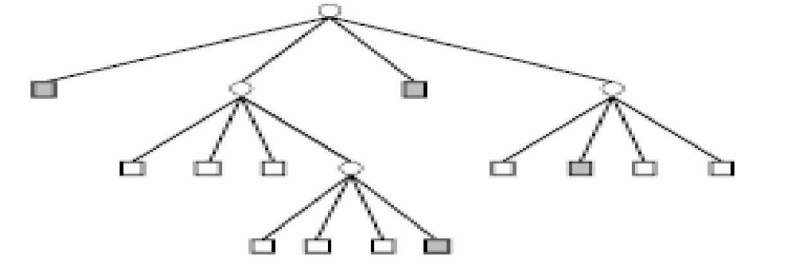


Figure 3.10 : Représentation arborescente du quadtree

- Les cercles sont appelés *sommets* de l'arbre quaternaire et les carrés gris et blancs sont appelés feuilles.
- Ce type de partitionnement est certainement le plus répandu dans les programmes de compression par fractale .il est simple à utiliser et fournit des résultats corrects.

v Phase de fusion

La phase de fusion est utilisée en segmentation d'images. Elle consiste à regrouper les blocs adjacents égaux selon le prédicat d'homogénéité considéré. L'intérêt de cette étape est *de réduire* le nombre total de régions. Les bords des régions sont aussi beaucoup plus proches des contours réels de l'image.

L'étape de fusion n'est cependant pas utilisée dans le cas de la compression par fractales, car les blocs obtenus sont de formes trop complexes.

v Propriétés du quadtree

Le partitionnement obtenu est rigide puisqu'il est guidé par le processus de découpage récursif en blocs carrés. *Il n'est pas adapté* aux formes des objets présents dans l'image. Même s'il s'adapte au contenu de celle-ci. Le partitionnement contient un nombre important de blocs, si on le compare aux autres partitionnements, aussi la construction récursive du quadtree n'est pas une solution optimale en terme de temps de calculs puisque chaque pixel est visité un nombre de fois égal à sa profondeur finale dans l'arborescence.

v Utilisation du quadtree pour la compression par fractales

Le quadtree est largement utilisé dans la littérature [43], [41], [44], [50], pour la compression des images selon une approche fractale.

v Procédure de codage

La profondeur du quadtree est fixée à l'avance, ce qui impose la taille minimale B_{min} des blocs destination. La taille maximale B_{max} est aussi imposée. Pour chacun des N blocs destination r_n, de taille B^2, l'algorithme recherche un bloc source $d_{\alpha(n)}$ de taille D^2 avec $D = 2B$.

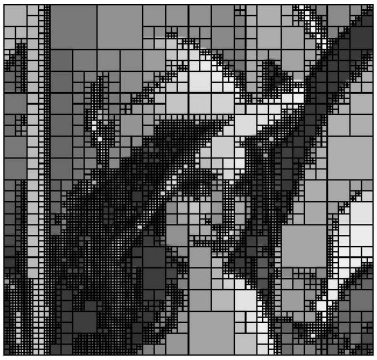


Figure 3.11 : Illustration d'un partitionnement quadtree sur limage Lena 512*512 ,composé de 5704 carrés

Le codage d'un bloc destination r se fait par recherche du bloc source d_n décimé qui permet de minimiser l'erreur entre r et l'approximation $\hat{r}$ donnée par :

$$\min d(r,\hat{r}) = \min_{\beta_1,\beta_2} d(r - \beta_1 b_1 - \beta_2 b_2) \qquad (3.33)$$

Où B_1 *et* B_2 sont des coefficients réels. Si la distance minimale demeure supérieure à un seuil prédéfini, et si le niveau de r dans le quadtree demeure inférieur a la profondeur maximale de ce dernier, alors le bloc destination est redivisé et la procédure de codage est relancée sur chacun des quatres sous blocs créés. Si la distance minimale est inférieure au seuil fixé, le bloc r est codé par les coefficients B_1 *et* B_2 de la *transformation massique* retournant $\hat{r}$ et par la position dans le dictionnaire du bloc source d associé. Ces informations codent la transformation élémentaire w.

4.3 Partitionnement semi_rigide (*H/V*)

Le défaut de la partition quadtree est qu'elle choisit l'ensemble des domaines D de Façon indépendante du contenu. L'ensemble doit être rendu très grand pour qu'on puisse y trouver une bonne correspondance pour un range donné. Une façon de remédier à cela, tout en augmentant la flexibilité de la partition en ranges, est d'utiliser une *partition H_V.*

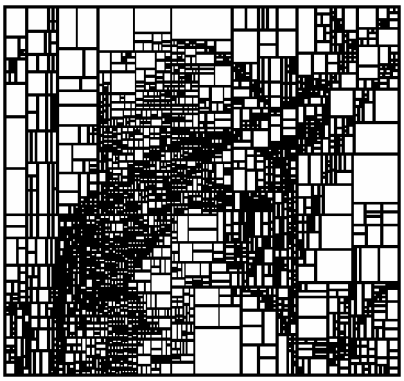


Figure3.12 : partitionnement H_V calculé sur l'image Lena et composé de 2910 rectangles.

v Partitionnement horizontal_ vertical

Fisher et *Menlove,* proposent un partitionnement rectangulaire *H_V* pour le codage des images.

Un processus récursif de subdivision des blocs en deux sous_rectangles conduit à une partition adaptée au contenu de l'image. La partition n'est pas rigide comme le quadtree puisque la division d'un bloc, qui tient compte de sa texture, ne crée pas nécessairement deux blocs d'égales surfaces. La partition est semi_rigide dans le sens où les arêtes des blocs demeurent obligatoirement soit horizontales, soit verticales.

v Intérêt du partitionnement pour la compression par fractales

La règle de construction de la partition *H_V* est proche de celle utilisée pour le calcul du quadtree dans le sens où la subdivision horizontale ou verticale d'un rectangle qui ne vérifié pas le critère d'homogénéité crée de nouveaux rectangles (en l'occurrence deux). *Fisher* propose deux méthodes de division qu'il utilise en fonction de la nature du rectangle non homogène .

1. si le bloc est parcouru par une frontière oblique de l'image, la séparation est choisie de manière à ce que l'un des sous blocs soit parcouru diagonalement par la frontière et que s'il existe un autre sous bloc, celui-ci soit homogène.
2. si le bloc inclut une frontière horizontale ou verticale, la séparation est choisie de manière à créer deux sous-blocs homogènes.

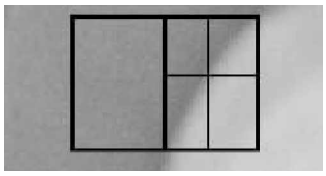


Figure3.13 : partition *H_V* méthodes de division (1)

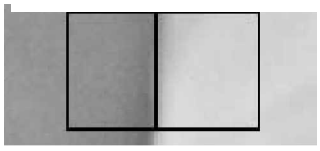


Figure3.14:partition *H_V* méthodes de division (2)

4.4 Partitionnement souple (triangulaire)

Beaucoup d'études ont été faites sur l'utilisation de partitions bien adaptées au contenu des images. Même si l'on trouve dans les diverses publications un grand nombre d'études de compression par segmentation d'images à travers des partitions adaptées et hiérarchiques [63], [64], [65], [66], [67], [68], [69], [70], [71]. C'est la partition de *Delaunay* (dual du graphe de *Voronoï*) qui a été la plus étudiée pour le codage fractal, notamment par *Davoine* [72], [73], [10].

Nous proposons ici un partitionnement de l'image en triangles de *Delaunay*. Celui-ci a l'avantage de ne pas être rigide (*souples*) puisqu'ils sont calculés sur un ensemble de points pouvant être positionnés à peu prés n'importe où sur le support de l'image. De plus, le diagramme est construit dans un environnement de type division et fusion [56], [57], [58], ce qui lui permet d'être adapté à l'image.

v Triangle de *Delaunay*

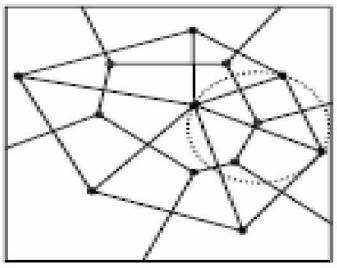


Figure 3.15: Triangles de Delaunay, et Polygones de Voronoi.

Un triangle $T = (P_i, P_j, P_k)$ où P_i, P_j, P_k sont dans P avec ($P = \{P_i \in IR^2, i=1,\ldots, n\}$ les n points appelés germes ou sites) est un triangle de *Delaunay* si seulement si, l'intérieur du cercle circonscrit à T ne contient aucun point de *:*

$$B(Pi, Pj, Pk) \cap (P\text{-}pi\text{-}pj\text{-}pk) = \phi \qquad (3.34)$$

Une propriété intéressante des triangles de *Delaunay* est qu'ils sont tous bornés

- Le dual géométrique est appelé le diagramme de *Voronoi*, composé de polygones (Figure3.16).

Les sommets du polygone constituent les centres des cercles circonscrits aux triangles. Les sommets des triangles constituent l'ensemble *T*.

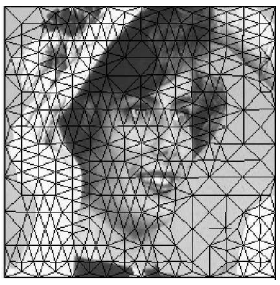
v Codage d'une image

L'objectif est d'avoir un maximum de ressemblance entre les différents blocs des partitions *R* et *D*. Pour cela, la triangulation de *Delaunay* est construite dans un environnement de Division et Fusion.

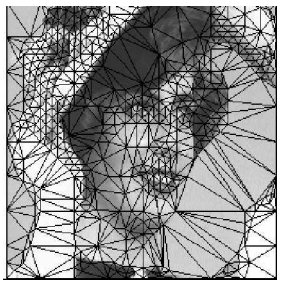
ü **La phase de division :** Consiste à rajouter des points sur le barycentre des triangles dont le niveau de gris n'est pas homogène (*critère de variance*).

ü **La phase de fusion :** Consiste à supprimer les triangles inutiles, pour lesquels les voisins ont une valeur moyenne de niveaux de gris très proche. Cette étape diminue sensiblement le nombre de triangles dans la partition *R*. Ceci aura pour effet de diminuer le temps de calcul pour coder l'image ainsi que d'augmenter le taux de compression.

Image initiale



Division



Fusion

Figure 3.16: Illustration de construction de l'initialisation de la triangulation de *Delaunay* à gauche est suivie des division et fusion à droite

Dans le cas de la compression d'images par fractales basée sur la triangulation de *Delaunay*, il est nécessaire de calculer d'abord une triangulation générique pour construire la famille des *domain blocks* et une triangulation plus fine, pour la famille des *range blocks* [10], [74], [75].

4.5 Autres types de partitionnements

De nombreux autres types de partitionnements plus complexes ont déjà été utilisés dans [39], [11] : polygonaux (ou quadrilatères), arborescente, mixtes, …etc.

- *Dans la partitionnement polygonaux de Vornoi*, l'approche de dévision et de fusion a été retenue pour la construction d'une partition de *Vornoi* adaptée au contenue d'une image (constituées de polygones homogènes).
- L'introduction des partitions à structure arborescente en compression fractale est due à *maher baraket* en l'an 2000 [11], mais la première méthode de compression utilisant un arbre binaire de plus de deux directions fut proposée par *Wu* et *Yao* (*IEEE* en 1991) .

L'approche adoptée consiste à construire des partitions arborescentes (les arbres sont complet voir figure 3.17) avec la racine de l'arbre qui correspond à l'image entière et les feuilles aux blocs de la partition.

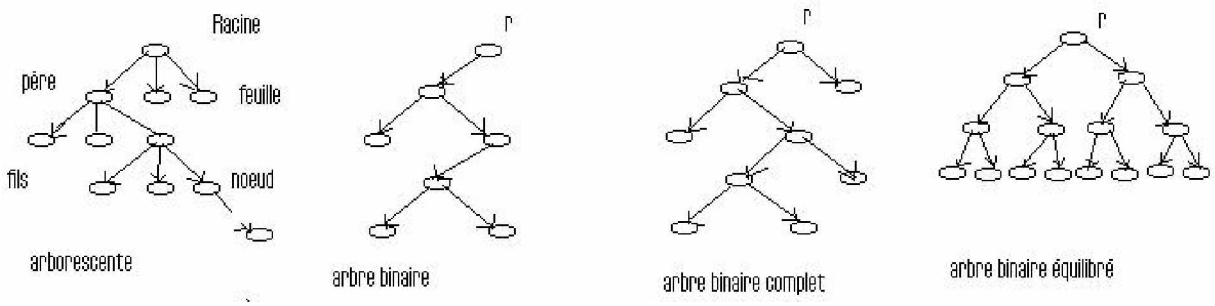


Figure3.17 Les types d'arbres

5 OPTIMISATION FRACTALE ET TRANSFORMATIONS HYBRIDES

5.1 Codage par *TCD* et fractales

La méthode proposée par *Barthel* et *Al*. Dans (*IEEE* 91) améliore sensiblement les résultats de la norme *JPEG*, à taux de compression élevé. Elle permet en outre de reconstruire des images de meilleure qualité que celles obtenues avec les schémas classiques de compression par fractales dans l'espace des luminances, à taux de compression égaux.

L'approche, basée sur un partitionnement carré fournissant des blocs destination est *hybride*. Elle combine la transformation en cosinus discrète (*codage de la redondance intra-bloc*) et la transformation fractale (*codage des similarités inter-blocs*).

5.2 *QV* et compression fractales

Dans [4], [53], un schéma de compression *hybride*, combinant quantification de vecteurs et compression fractale, est décrit. Chaque bloc est encodé soit par une référence dans l'image elle-même, soit par une référence dans un *codebook*. Les avantages sont nombreux : la recherche est plus rapide pour les blocs encodés par un vecteur du *codebook*, et on s'attend de plus à ce que la convergence au décodage soit également plus rapide, puisque ces mêmes blocs seront reconstruits en une seule itération [98].

5.3 Ondelettes et compression fractales

Dans [36], [10], [37], [26], le codage d'une image par fractales tel que nous l'avons vu jusqu'à présent est fondé sur des transformations locales qui opèrent sur des blocs de l'image. Les effets de blocs sont dans ce cas difficile à masquer lorsque le taux de compression est élevé. Récemment, différents chercheurs ont montrés que le schéma de codage par fractal peut être utilisé pour définir des relations entre les différents niveaux d'une décomposition multi résolution de l'image (décomposition en sous bandes ou par ondelettes).

Dans [10**]**, l'image originale est décomposée en n sous bandes notées i, (i=0,…, n-1). L'image basse résolution (niveau n-1) ainsi que les trois sous bandes du même niveau sont codées de manière exacte. Chacune des trois sous bandes de niveau supérieur (niveau n-2) sont ensuite codées par une approche fractale (la recherche de similarités au sein des sous bandes de fréquences spatiales se fait entre blocs source et destination de même taille) à partir des trois sous bandes dans le niveau n-1. Le processus continue ainsi jusqu'à atteindre la résolution 0 de l'image originale[98].

L'avantage de ces méthodes est qu'elles réduisent ou annulent les effets de blocs dans les images reconstruites .

5.4 L'avantages et inconvénients du compression fractale

@ Avantages

Les principaux avantages de cette technique fractale sont

- ü des taux de compression élevés.
- ü une représentation indépendante de l'échelle.
- ü une procédure de décompression particulièrement simple, qui consiste simplement à interpréter la représentation à une échelle donnée.

@ Inconvénients

- ü La compression reste lente malgré toutes les améliorations (coûteuse en temps).
- ü elle peut nécessiter un hardware spécialisé.
- ü Ce type de compression n'a pas encore fait l'objet de l'édition d'une norme, son utilisation n'est donc pas encore totalement démocratisée.

5.5 Optimisations

La notion d'optimisation doit être comprise de différentes manières :

Le travail initial de *Jacquin*, basé sur l'utilisation d'un système de fonctions itérées locales et contractantes, a donné lieu au démarrage de nombreuses autres recherches sur la compression des signaux réels *1D, 2D* et *3D* par fractales.

Elles concernent principalement :

- ü la construction d'une partition optimale pour calculer la transformation fractale ;
- ü l'accélération de l'étape de codage ;
- ü l'accélération du décodage : itératif, non-itératif, hiérarchique ;
- ü l'extension de la méthode au codage des images vidéo ;
- ü l'utilisation des fractales dans les schémas de codage-décodage hybrides.

6 CONCLUSION

Nous avons introduit dans ce chapitre les bases mathématiques nécessaires à la Compréhension de la théorie des systèmes de fonctions itérées. Nous avons présenté le théorème de point fixe et les algorithmes de compression / décompression des images naturelles selon l'approche fractale d'une image fixe en se basant sur l'auto similarité contenue à l'intérieure de toute l'image.

Nous avons introduit les différents modèles géométriques de partitionnement pour la compression par fractales à savoir le partitionnement quadtrée, le partitionnement *HV*, le partitionnement triangulaire, ainsi que les différentes améliorations usuelles en terme d'hybridation.

Malgré les performances et la qualité de l'image reconstruite à des taux de compression élevés, la compression fractale souffre de sa lenteur en terme de temps de compression par rapport à d'autres méthodes; pour cela, plusieurs recherches ont tentées d'améliorer les performances en temps de compression et d'introduire la notion de parallélisme pour résoudre le problème de la vitesse des algorithmes de compression par fractale. Ce sera l'objet des chapitres suivants et le cœur de notre travail.

CHAPITRE III

Compression des images fixes basée sur une transformation fractale

3.1.Introduction

3.2. Théorie des Systèmes de Fonctions Itérées (IFS)

3.2.1 Les outils mathématiques

3.2.2 Quelques définitions dans l'espace (Rappels sur les espaces métriques)

3.2.3. Transformation des espaces métriques

3.2.3.1 Transformation affine

3.2.3.2 Transformation lipschitzienne

3.2.3.3 Transformation contractante

3.2.3.4 Transformation contractante sur l'espace h(IR^2)

3.2.3.5 Point fixe

3.2.4. Théorie des ifs et compression par fractales

3.2.4.1 Définition de L'IFS

3.2.4.2Attracteur d'un IFS

3.2.4.3Théorème du collage

3. 2.4.4 Système de Fonctions Itérées Partitionnées (PIFS)

3.2.4.5 Problème inverse

3.2.4.6 La limite des ifs

3.3. La compression des images par fractales

3.3.1 Qu'est –ce qu'une Fractales ?

3.3.2 Pourquoi " la Compression Fractale d'Images" ?

3.3.3 La notion de L'auto-similarite

3.3.4 Auto-similarite dans les images

3.3.5 Principe de la compression IFS et PIFS

3.3.6 Le codage fractale par bloc (transformation fractale)

3.3.7 Schéma général d'un codeur - décodeur fractal

3.4. Partitionnement de l'image

3.4.1 Rôle du partitionnement pour la compression par fractales :

3.4.2 Partitionnement rigide (*Quadtree)*

3.4.3 Partitionnement Semi_Rigide *(Horizontal _ Vertical*)

Table des figures

Bibliographie

Azziz :

[1] A. Ali-Pacha et N. Hadj-Said . Compression des Images Fixes par Fractale : Partitionnement Quadtree. Proceeding, FRACTALES ‘ 2000, pages 12-19, Université Mentouri Constantine , Algérie , Novembre 2000.

[2] kada =15

[3] F. Davoine et J. Marc Chassery . "Compression D’images par F ractales " laboratoire TIMAG-IMAG, GRETSI CORSA 1996.

[4] F. Davoine. Compression D’images par Fractales Basée sur la Triangulation de Delaunay. Thèse L’INPG ‘ Institut National polytechnique de grenoble ‘ Décembre 1995

[5] S. Derr. "Compression F ractale à Partir Du Partitionnement de Delaunay" . IMAG , Mars 1998 .

[6] Y. Fisher D. Rogovin et T.P. Shen. " A Comparaison of Fractal Methods with DCT and Wavelets " , In Neural And Stochastic ethods In Image And Signal Processing III, Volume Proc. SPIE 2304-16, San Diego

[7] C.Jean « compression fractale d’image « L4ULB 4octobre 2001.

[8] H. Krupnik, D. Malah and E. Karnin. “fractal representation of images via the discretewavelet transform’. appears in *IEEE* 18th Conv. of EE in Israel, Tel-Aviv. Department of Electrical Engineering Technion - Israel Institute of Technology.Haifa 32000, Israel *March 1995.*

[9] E. Lutton . Algorithmes Génitiques et Fractales. Rapport D’habilitation Universite de Paris XI Orsay , 11 Février 1999.

[10] B. Maher . Partition Arborescentes et Compression Fractale . Thèse de L’INPT , Janvier 2000.

[11] F. Olivier et P. Cédric. La compression d’images par Ondelettes T.I.P.E.d’informatique 1998 Lycée Pothier MPSI 3.

[12] G.Robert , F.davoine et J.M .chssery. « Compression dimage par fractales à base de maillages polygonaux de voronoi » Laboratoire TIMC-IMAG Projet Accord Entre GDR-PRC ISIS , Le CCETT Le CENT 1997.

[13] B. Wohlberg et G. de Jager . " A review of the fractal image coding literature ", *IEEE* Transactions on Image Processing. Vol 8 n° 12 PP 1716-1729 December 1999.

Delauny

[14] M. F. Barnsley, V. Ervin , D.Hardin, and J. Lancaster. Solution of an inverse problem for fractals and other sets. Proc. Natl. Acad. Sci. USA,83:1975-1977, April 1986.

[15] T. J. Bedford, F. M. Dekking , M. Breeuwer, M. S. Kean, and D. Van Schoone-Veld. Fractal coding of monochrome images. Signal Processing: Image Communication,6:405-419,1994.

[16] C. A. Cabrelli, B. Forte, U. M. Molter, and E. R. Vrscay. Iterated fuzzy set Systems : A new approach to the inverse problem for fractals and other sets Journal of Mathematical Analysis and Applications ,171(1) :79-100; November 1992.

[17] Y. Fisher. Fractal Image Compression with Quadtrees, 1995.

[18] B. Hurtgen, F. Muller, and C. Stiller. Adaptive fractal coding of still pictures. Picture Coding Symposium,1993.

[19] Klinger. Data structures and pattern recognition. In Proc. Int. Joint Conf.on Pattern Recognition, pages 497-498, 1973.

[20] . W. G. Kropatsch, M. A. Neuhausser, I. J. Leitgeb, and H. Bischof. Combining pyramidal and fractal image coding . In Proc.11th ICPR, The Hague, The Netherlands, volume 3, pages 61-64,1992.

[21] J. Lévy_Véhel and A . Gagalowicz. Fractal approximation of 2-D object. Technical Report 1187, INRIA – Rocquencourt, France , 1990.

[22] E.Lutton and J. Lévy_Véhel. Optimization of fractal functions using genetic Algorithms. In Fractal'93. Londres,1993.

[23] G. Mantica and A.Sloan. Chaotic optimization and the construction of fractals: solution of an inverse problem. Complex Systems,3:37-62,1989.

[24**]** G. E. oien and S. Lepsoy. Fractal-based image coding with fast decoder convergence, Signal Processing ,40:105-117, October 1994.

[25] R. Rinaldo and A. Zakhor. Inverse and approximation problem for two dimensional fractal sets. IEEE Transactions on Image Processing,3(6):802-820, November 1994.

[26] A. Rosenfeld . Quadtrees and pyramids for pattern recognition and image Processing . In Proc. 5th. Int. Conf . on Pattern Recognition . pages 802 – 811, University of Maryland ,1980

[27] H.Samet. Region representation: quadtrees from binary arrays. CVGIP, 13:88-93 , 1980.

[28] H. Samet. The quadtree and related hierarchical data structures. ACM Comput. Surv.16:178-260,1984.

[29] R . Vrscay . Moment and collage methods for the inverse problem of fractal construction with iterated function systems . In Fractal 90 conference, June 1990.

Leila

[30] Bertin E., Parazza F., J-M. Chassery, "Segmentation and Measurement based on 3D Voronoi Diagram: Application to Confocal Microscopy", Computerized Medical Imaging and Graphics, Vol. 17, pp. 000-008, 1993.

[31] Chassery J.-M., Melkemi M., "Diagramme de Voronoi applique a la segmentation d'images et a la detection d'evenements en imagerie multi-sources", Traitement du Signal, Vol. 8, No. 3, pp. 155-164, 1991

[32] Chen X., Schmitt F., "Split-and-merge image segmentation based on Delaunay triangulation", Proc. of The 7th Scandinavian Conf. on Image Analysis, Aalborg, Denmark, Vol. 2, pp. 910-917, aug.13-16 1991

Cour+contif

[33] A. ARNEDO, F. ARGOUL, E. BARCY, J. ELEZGARAY and J.-F. MUZY, *Ondelettes, Multifractales et Turbulences*, Diderot Eds - Arts et sciences, 1995.

[34] M. BARNSLEY, *Fractals Everywhere*, Academic Press, Inc., 1988.

[35] E. LUTTON, J. LÉVY-VÉHEL, G. CRETIN, P. GLEVAREC and C. ROLL, *Mixed IFS : resolution of the inverse problem using Genetic Programming*, Technical Report 2631, INRIA, August 1995, Accepted for publication in Complex Systems.

[36] K. BERKNER***, a wavelet-based solution to the inverse problem for fractal interpolation functions,*** in Lévy Véhel , pages 81–92.

[37] H. RADHA, R. LEONARDI and M. VETTERLI, *A multiresolution approach to binary tree*

representations of images, in *ICASSP*, pages 2653–2656, 1991.

[38] Y. COHEN, M. LANDY and M. PAVEL, *Hierarchical coding of binary images*, IEEE transactions on Pattern Analysis and Machine Intelligence, vol. 7, no. 3, pages 284–298, May 1985.

[39] P. CHOU, T. LOOKABAUGH and R. GRAY, *Entropy-constrained Vector quantization*, IEEE transactions on acoustics, speech and signal processing, vol. 37, no. 1, pages 31–42, January 1989.

[40] X. WU, *Image coding by adaptative Tree-structured segmentation*, IEEE transactions on information theory, vol. 38, no. 6, pages 1755–1767, November 1992.

[41] X. WU and Y. FANG, *A segmentation-based predictive multiresolution image coder*, IEEE transactions on image processing, vol. 4, no. 1, pages 34–47, January 1995.

[42] F. DENATALE, G. DESOLI and D. GIUSTO, *Segmentation-based hybrid-coding of colo images*, in *ICASSP*, pages 2757–2759, 1991.

[43] R. DANSEREAU and W. KINSNER, *Perceptual Image Compression Through Fractal Surface Interpolation*, in *CCECE*, 1996.

[44] R. DISTASI, M. NAPPI and S. VITULANO, *Image Compression by _ -Tree Triangular Coding*, IEEE transactions on communications, vol. 45, no. 9, pages 1095–1100, september 1997.

[45] R. SEQUEIRA and F. PRÊTEUX, *Discrete Voronoï Diagrams and the SKIZ operator : A Dynamic Algorithm*, IEEE transactions on Pattern Analysis and Machine Intelligence, vol. 19, no. 10, pages 1165–1170, 1997, October.

[46] F. DAVOINE and J.-M. CHASSERY, *Compression d'Images par Fractales*, in *Journées sur les"Nouvelles téchniques pour la compression et la représentation des signaux audiovisuels"*, Janvier 1995.

[47] F. DAVOINE and J.-M. CHASSERY, *Compression d'Images par Fractales*, in *CORESA*, 1996.

[48] F. DAVOINE, *Compression d'images par fractales basée sur la triangulation de Delaunay*,

Thèse de doctorat, Institut Polytechique de Grenoble, France, 1995. repeter

[50] E. BERTIN, *Diagrammes de Voronoï 2D et 3D : Application en analyse d'images*, Thèse de

doctorat, Université J. Fourier, Grenoble, France, 1994.

[51] V. RAMASUBRAMANIAN and K. PALIWAL, *Voronoï projection-based fast nearest-neighbor search algorithms : mapping table-based search techniques*, Digital Signal Processing, vol. 7, pages 260–277, 1997.

Donc la compression fractale peut être résumé par cette équation : $\hat{r}_{i,j} = s_{i,j} wS(d_{(i,j)}) + o_{i,j}$ /(S : transformation contractant spatiale , w 1 des 8 transformations isométrique montrée dans le tableau suivant)

PARTIE A : PARALLELISATION D'UN ALGORITHME DE COMPRESSION FRACTALE

1 INTRODUCTION

Ces dernières années les techniques de compression par fractales d'images (*IFS*), ont gagnées plus d'intérêt en raison de leur capacité de réaliser des taux de compression très élevés tout en mettant à jour la qualité très bonne de l'image reconstruite. L'inconvénient principal de telles techniques est le temps très élevé pour déterminer les codes *d'IFS* de l'image compressée.

Pour résoudre ce problème, plusieurs techniques sont utilisées. Dans ce chapitre nous allons introduire la technique du parallélisme dans la compression fractale pour augmenter les performances pour un bon rendement en utilisant une architecture parallèle *MIMD* à mémoire distribuée.

Deux approches différentes peuvent être distinguées pour la compression d'image fractale selon les recherches effectuées par [88], [89].

1- Techniques séquentielles

2- Calcul à haute performance (les algorithmes parallèles).

2 L'APPROCHE SEQUENTIELLE

La compression d'image par fractale exploite les similarités au sein des images. Ces similarités sont décrites par une transformation affine contractante W de l'image dont le point fixe G est proche de l'image elle même $W(G)=G$. La compression d'image consiste en des blocs de transformation qui approche les petites portions de l'image par d'autres plus larges en utilisant des transformations affines contractantes. Les petites portions sont appelées ″bloc range R_i″ et les plus larges sont appelées ″blocs domaines D_i″. Toutes les blocs ranges forment une partition de l'image. Les blocs domaines peuvent être choisis librement au sein de l'image et peuvent s'imbriquer. Pour chaque range R_i un domaine D_i approprié doit être trouvé. Pour juger de la qualité d'un seul domaine, une erreur carrée moyenne (*rms*) de la distance entre le bloc range R_i et $W_i(D_i)$ est calculée et doit être inférieure à une tolérance Δ [89]. Le bloc domaine le plus similaire pour un bloc range spécifique peut être trouvé par une recherche à travers tous les domaines et leurs isométries (réflexion, rotation,…).

Dans un algorithme non adaptatif, pour chaque bloc range le bloc de transformation avec la plus (*rms*) devient une partie de la transformation d'image sans se soucier comment la rangée peut être couverte. Dans un algorithme adaptatif l'erreur d'un seul bloc de transformation est comparée à une erreur maximale prédéfinie appelée erreur de collage Δ. Si l'erreur calculée est plus grande le bloc range spécifique est divisé en des blocs qui sont alors couverts indépendamment. Donc par rapport à l'algorithme non adaptatif, une qualité d'image peut être garantie. La compression fractale est non symétrique.

L'étape de codage a un coût de programmation énorme qui dépend du nombre des blocs domaines à comparer avec chaque bloc range, par exemple une image de taille $n \times n$ avec des domaines carrés de taille $h \times h$ nécessitent $(n-h+1)^2$ domaines. En outre chaque bloc domaine à 8 isométries. Donc chaque bloc range doit être comparé à $8 \times (n-h+1)^2$ blocs domaines. Pour une image de taille 256×256, les blocs ranges ont une taille de 8×8 et les blocs domaines ont la taille 16×16, l'algorithme effectue 516104192 opérations de calcules de *rms*. Ce qui rend l'étape de codage plus lente par rapport à l'étape de décodage.

Cette énorme complexité de calcul montre clairement le besoin pour accélérer la vitesse d'exécution. Plusieurs approches de parallélisation d'algorithmes sont présentées dans les sections suivantes.

2.1 Evaluation de la complexité de l'algorithme des *IFS* (inconvénient de l'approche séquentielle)

Comme nous avons vu dans le chapitre 3, que la détermination du code fractal (*IFS*) d'une image fixe par l'algorithme séquentiel de la compression fractale, nécessite le passage plusieurs fois par les mêmes étapes (la boucle principale de l'algorithme fractal). Autrement dit, l'algorithme des *IFS* doit être réitéré plusieurs fois.

Dans cette section nous déterminons la complexité du temps de codage de l'algorithme de compression par *IFS*. Les étapes de l'algorithme doivent être répétées pour toute la partition R d'une image numérique A de taille $N \times N$. Pour chaque bloc range R_i comparé à tout les blocs domaines D_i du "domain pool" D. Le nombre de répétitions de comparaison sera donné par [90]:

$$N_{etap} = 8 \cdot \left(\frac{N}{n} \right)^2 \cdot \left(\frac{N-(2n-1)}{P} \right)^2, \text{ avec} \qquad (5.1)$$

Le nombre de transformations isométriques est $8 \cdot \left(\frac{N}{n} \right)^2$: Est le nombre de blocs ranges R_i, P est un facteur de recouvrement entre *"domain pool"* et blocs $\left(\frac{N-(2n-1)}{P} \right)^2$ domaines.

Nous avons vu dans le chapitre 4 que l'algorithme de compression fractale, exigera de calculer autant de fois la dérivée de l'expression qui donne les paramètres de la transformation affine W_i ,s, o et d_{rms}:

$$s = \frac{n \cdot \left(\sum_{i=1}^{n} d_i \cdot r_i \right) - \left(\sum_{i=1}^{n} d_i \right) \cdot \left(\sum_{i=1}^{n} r_i \right)}{n \cdot \sum_{i=1}^{n} d_i^2 - \left(\sum_{i=1}^{n} d_i \right)^2} \tag{5.2}$$

$$o = \frac{1}{n} \left(\sum_{i=1}^{n} r_i - s \sum_{i=1}^{n} d_i \right) \tag{5.3}$$

$$d_{rms} = E(D_i, R_i) = \frac{1}{n} \left[\sum_{i=1}^{n} r_i^2 + s \left(s \sum_{i=1}^{n} d_i^2 - 2 \sum_{i=1}^{n} d_i r_i + 2o \sum_{i=1}^{n} d_i \right) + o \left(o \cdot n - 2 \sum_{i=1}^{n} r_i \right) \right] \tag{5.4}$$

Cette équation comporte 13 exécutions de virgule flottante. En conséquence, l'algorithme de compression par *IFS* exige un certain nombre de virgule flottante flops donné par la formule suivante:

$$N_{flops} = 8 \cdot \left(\frac{N}{n} \right)^2 \left(\frac{N-(2n-1)}{P} \right) (2n^2 + 21) \tag{5.5}$$

La valeur de N_{flop} est généralement très grande, ce qui rend le temps de codage trop lourd. Par exemple, si nous choisissons n=8 et p=4, la compression d'une image de taille 512×512 exige le nombre 75×10 Flops c-à-d une demi-heure sur un processeur de 200 *M*flops avec une efficacité h=0.2 [90].

@ Cette énorme de complexité de calcul montre clairement le besoin pour accélérer la vitesse d'exécution. Comme on a mentionné précédemment qu'il existe plusieurs techniques Pour résoudre ce problème, parmi ces techniques :

1) ***La classification des blocs*** : cette méthode est un moyen efficace pour réduire le nombre de comparaisons lors de la recherche de transformations. A la création de «domain pool», chaque domaine est rangé dans une classe. Lors de la recherche, le range est à son tour

classé et les comparaisons ne se feront qu'avec les domaines qui se trouve dans la même classe. La difficulté bien entendue, est de trouver le critère de classification raisonnable, tel que la probabilité que le meilleur domaine ne se trouve pas dans la même classe que le range, soit minimisée. Si leur nombre est assez grand, plusieurs classes peuvent être examinées pour chaque range. Parmi les méthodes de classification qui sont très répandues : la méthode de classification de *ficher* et la méthode de *Jaquin*.

2) ***la parallélisation de la compression fractale* :** La complexité liée à l'approche séquentielle, et aux problèmes de classification justifie l'utilisation d'une autre approche qui est la parallélisation d'algorithmes de compression d'image par fractale. L'algorithme d'*IFS* permet l'exploitation du parallélisme massive, où le traitement massivement parallèle *MPP* (*Massively Parallel Processing*) qui semble être une réponse raisonnable au problème de codage de l'*IFS*, permettant d'augmenter la vitesse de compression et de converger vers une image reconstruite de haute qualité.

3 L'APPROCHE PARALLELE DE LA COMPRESSION FRACTALE

Il existe deux principales classes d'algorithmes pour la compression d'image fractale sur une architecture parallèle. La première classe de ces algorithmes permet de stocker la totalité de l'image sur chaque *PE* (*Processing Element*), à savoir quand tous les *PEs* ont suffisamment de mémoire pour avoir une copie locale de toute l'image. Donc chaque *PE* a l'entier domain pool à sa disposition. À chaque *PE* un sous-ensemble des blocs ranges est assigné, lequel peut être fait statiquement ou dynamiquement. La deuxième classe de ces algorithmes permet de distribuer le domain pool sur l'ensemble des *PEs*, à cause de l'espace insuffisant de la mémoire ou à d'autres raisons. Dans [91], plus de détails peuvent être trouvés [94].

Jackson, et *Tinney* [92], [93], ont présenté trois schémas génériques pour l'encodage fractal parallèle de l'image, ils seront présentés dans la suite.

3.1 Allocation statique de la charge (à travers les ranges)

La façon la plus simple de distribuer le travail à travers l'ensemble des p processeurs est de placer $1/p$ du nombre total des blocs ranges sur chacun d'eux. Les processeurs ne communiquent pas entre eux jusqu'à ce qu'ils finissent le traitement et rendent le résultat, d'où ils ont besoin d'une copie locale de l'image. Les actions exécutées par les processeurs sont exactement les mêmes comme dans l'algorithme séquentiel, seulement la quantité de travail

fait par chacun d'eux est approximativement 1/*p* du travail fait dans l'implémentation séquentielle. Donc, la vitesse anticipée est *p*. Cependant, les expériences portées par *Jackson* et *Al* ont montré des différences considérables dans le temps d'exécution sur différents processeurs ; ce qui a causé la diminution de la vitesse d'exécution. Cela est expliqué par trois facteurs:

1) ***Incomplétude de la recherche du domaine*** : l'algorithme a testé les comparaisons range-domaines jusqu'à ce que le premier bloc domaine satisfaisant soit trouvé. Par conséquence, un bloc domaine peut être trouvé très rapidement pour un blocs range, pendant que pour les autres, seulement après avoir cherché un très grand volume des blocs domaines.

2) ***Schéma de classification*** : une image peut avoir des proportions irrégulières des différentes classes des domaines et par conséquent, la recherche des domaines correspond au bloc range qui soit traité plus rapide, ou plus lentement suivant la classe des domaines qui lui correspond.

3) ***partitionnement Quadtree*** : pour quelques blocs ranges un bon bloc domaine ne sera pas trouvé, donc il devra être divisé en plus petits blocs. Cela prendra beaucoup plus de temps si la totalité de ces blocs est encodée immédiatement.

Pour ces raisons quelques processeurs effectuent plus de calcul que d'autres, ce qui permet de dégrader la performance totale de ce système. Donc, la compression par l'allocation de la charge statique est bien économique dans la totalité de la communication, cependant elle n'est pas efficace quand nous ne pouvons pas garantir une distribution équilibrée sur le système.

3.2 Allocation dynamique de la charge

La méthode convenable pour résoudre les problèmes mentionnés au-dessus est d'introduire l'équilibrage de la charge dans le système. Cette implémentation est connue sous le nom d'allocation dynamique de la charge. Dans cette implémentation, il y a un processus maître spécifique, qui permet de distribuer les tâches sur l'ensemble des processeurs (esclaves). Ce processus maître a deux files:

- File des tâches (blocs ranges, attendre pour être traité)
- File d'esclaves (processeurs libres).

Quand une tâche est assignée à un esclave ; la tâche et l'esclave, sont enlevés de leurs files. Après q'un esclave a traité sa tâche, il rend le résultat au maître et entre dans la file d'esclaves libres et attend une nouvelle tâche.

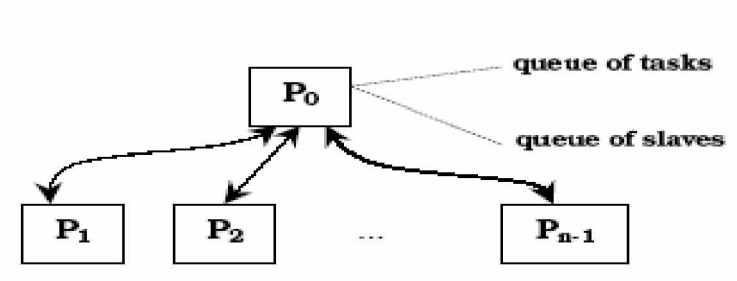


Figure 5.1 : Allocation dynamique de la charge

Dans ce schéma tous les processeurs sont occupés pendant tout le temps, à l'exception des courtes périodes quand ils attendent dans la file. Comme *Jackson* et *Al* ont représenté dans [5], cette implémentation est plus effective et généralement mieux convenu pour ce genre d'algorithme de compression d'images fractales que l'allocation statique de la charge. Cependant, il n'élimine pas complètement le déséquilibre de la charge.

3.3 Allocation dynamique de la charge avec un processus de circulation pipeline (à travers les domaines)

Une solution qui adresse les problèmes précédents est une configuration de la canalisation pipeline, comme elle se présente dans Figure 5.2.

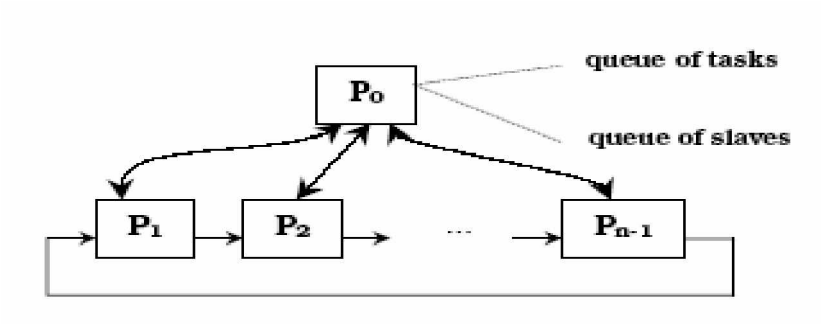


Figure 5.2 : Allocation dynamique de la charge avec un processus de circulation pipeline

Contrairement aux deux schémas précédent, ce qui est distribué ici, ce n'est pas seulement les bloques ranges, mais aussi les blocs domaines. Un bloc range est traité en étant traversé par la technique de pipeline, et comparé avec les blocs domaine stockés sur chaque processeur. Si

un domaine adéquat est trouvé, le bloc range abandonne le pipeline et rend le résultat au processus maître (P_0). Si aucun résultat n'est trouvé, le bloc range revient à P_0 en tout cas, et il le divise en quatre sous blocs avant de les placer en arrière dans le pipeline. Notons aussi, que cette implémentation exige moins de mémoire, comme les processeurs esclaves n'ont pas besoin de stocker l'image entière. Ils ont besoin seulement de stocker localement des parties de cette image qui consiste aux sous ensembles des blocs domaines. L'implémentation a prouvée des résultats très efficace et très rapide avec un nombre croissant des processeurs esclaves.

@ Après une brève étude des deux méthodes de parallélisation via les ranges et les domaines (statique et dynamique), nous avons proposé dans notre travail une nouvelle méthode de parallélisation de la méthode de compression des images à l'aide des fractales. Cette méthode permet d'introduire le parallélisme à travers des portions de l'image qui sont envoyées par le maître vers les différents esclaves. Les actions exécutées par les processeurs sont exactement les mêmes comme dans l'algorithme séquentiel, seulement la quantité de travail fait par chacun d'eux est approximativement 1/*p* du travail fait dans l'implémentation séquentielle. Donc, la vitesse anticipée est *p*.

@ Cette méthode permet de combiner les avantages des deux méthodes précédentes, ainsi que d'éviter leurs inconvénients (détails de la méthode dans la partie *Implémentation*).

@ Avec les architectures précédentes le processeur maître possède un fichier *Tompan* pour stocker les résultats retenus à travers les différents esclaves.

4 LE PARALLELISME DE COMPRESSION FRACTALE ET L'ARCHITECTURE MATERIEL DEDIEE

4.1. Introduction

Le premier pas de *FIC (Fractal Image Coding*) est de diviser l'image originale en blocs ranges et blocs domaines. Le choix du type de partitionnement est très important pour la compression fractale de l'image puisque il permet de réduire le nombre total des ranges d'une image ce qui permet d'augmenter le taux de compression de l'image.

Différents partitionnements de l'image ont été étudiés dans le chapitre (3) (Fixed-size partitionnement, et partitionnement quadtree, partitionnement *H*/*V*, arbre binaire, …). Avec la compression fractale en parallèle, le choix du type de partitionnement est très important. Nous

nous limiterons ci après à présenter quelques partitionnements et leurs utilisations en compression d'images parallèles.

4.2 Architecture parallèle pour fixed-size partitionnement *(FPFIC)*

Dans ce type de partitoinnement les tailles des blocs ranges sont équivalentes. Dans ce cas, on va étudier une architecture parallèle efficace *VLSI* (*Very Large Scale Intergrated*) pour la compression d'image par Fixed-size partitionnement *(FPFIC)* qui utilise seulement la communication locale. Le principe de cette architecture est qu'elle est capable de compresser l'image à l'aide des fractales en parallele sans besoin de la mémoire externe pour les blocs domaines [95].

On va étudier le module de la comparaison rapide (*FCM*) dans chaque processeur qui peut calculer les huit transformations isométriques. Ce module est capable de calculer les distorsions entre les blocs domaines transformés et les blocs ranges. Ce module pourrait être utilisé dans beaucoup d'applications de traitement d'image qui ont besoin des transformations isométriques.

4.2.1 L'architecture *FPFIC*

L'architecture *FPFIC* proposée est une collection des *PEs* (*Processing Element*) qui est arrangée en espace bidimensionnel par les mêmes processeurs élémentaires (*PEs*) qui sont reliés entre eux et capables de calculer la distorsion *domaine–range*. Chaque bloc range envoyé à chaque *PE* est comparé avec le bloc domaine en parallèle. Après le processus de comparaison, tous les blocs domaines sont changés et le même processus sera répété. La procédure de codage de l'architecture proposée consiste en deux étapes.

4.2.2 L'extraction du domain pool : Étape 1

Dans la première étape, le *Domain Pool* est extrait des blocs ranges. La figure 5.3 représente la connexion des *PEs* pendant l'étape 1.

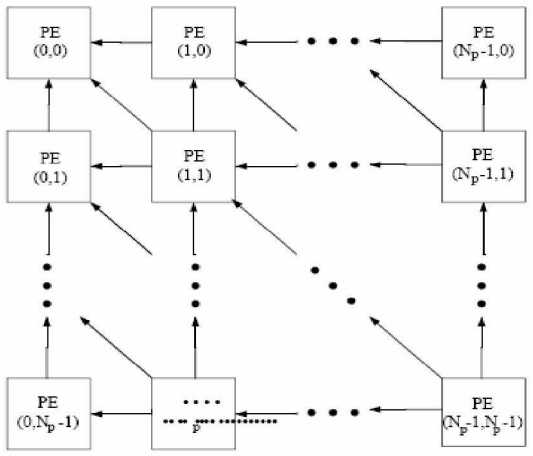


Figure 5.3 : la connexion des *PEs* pendant l'étape 1

La structure du *PE* de l'architecture étudiée est montrée dans la figure 5.4. Chaque *PE* à trois modules de mémoire, module de la comparaison rapide (*FCM*), module *CM* (*Contractive Mapping*), 1 à 2 décodeurs, et 1 à 2 multiplexeurs. Quand *CTRL* = 0, *PEs* opèrent l'étape 1. Cependant, quand *CTRL* = 1, *PEs* opèrent l'étape 2.

Trois modules de mémoire consistent en mémoire des blocs ranges, mémoire des blocs domaines, et mémoire du code qui est utilisé pour stocker les résultats obtenue. Chaque bloc range est chargé dans chaque mémoire des blocs ranges.

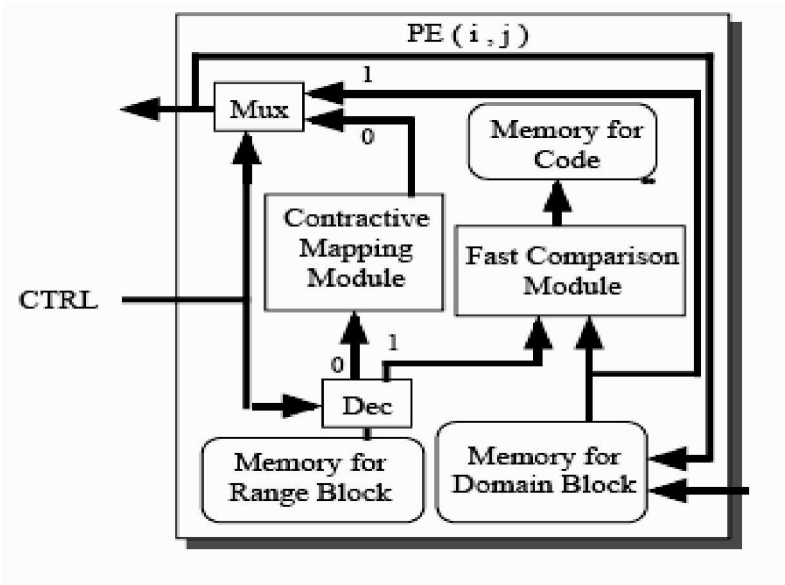


Figure 5.4 : La structure du *PE* de l'architecture *FPFIC*

4.2.3 Détermination du meilleure bloc domaine: Étape 2

La deuxième étape concerne le choix du meilleur bloc domaine pour chaque bloc range. Le Nombre total de comparaisons entre le *domain pool* et le *range pool* est : $(N_R)^2 \cdot (N_D)^2 \cdot 8$. Dans cette architecture, chaque bloc range est comparé en parallèle avec un seul bloc domaine et tous les blocs domaine sont échangés aux prochains *PEs*. Chaque *PE* ayant un bloc range, calcul la distorsion puis échange le bloc domaine au autres *PEs* par une alerte pour établir la connexion comme elle se présente dans la figure 5.5.

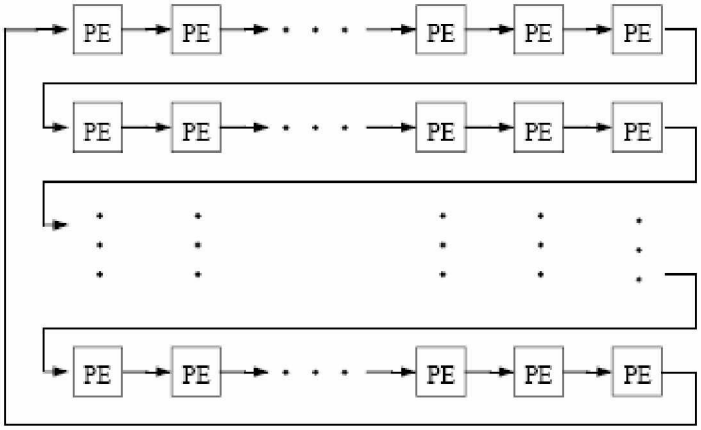


Figure 5.5: L'établissement de la connexion pour échanger les blocs domaines entre *PEs*.

4.2.4 le Module de comparaison rapide (*FCM*)

L'autonomie des données autorise le calcul de huit comparaisons entre le bloc domaine transformé par les huit transformations isométriques et le bloc range en *parallèle* au moyen de l'architecture du matériel dédié. Cependant, le coût de comparaison est considérablement augmenté dû à l'augmentation du nombre des blocs domaines et le circuit exigé pour le traitement en parallèle.

Le module de comparaison rapide (*FCM*) présente une architecture efficace pour les huit transformations isométriques pour les blocs domaines. Pour implémenter rapidement les transformations isométriques, cette architecture exécute une rotation très rapide pour l'ensemble des blocs domaines qui utilise l'échange vers le prochain *PE* sans le besoin d'une mémoire externe [95].

L'algorithme de la comparaison rapide est:

```
Algorithme de comparaison rapide (R, D1);
{
While  (les degrés de rotation sont moins que 360)
 {
Calculez les huit Transformations isométriques
Calculez la distorsion (R, D1/D2/D3/D4/ D5/D6/D7/D8);
Comparez 8 résultats pour les domaines (D1- 8) et sélectionnez le meilleur;
}
}
```

@ Le module de la comparaison rapide (*FCM*) est capable d'exécuter les huit transformations isométriques sur les blocs domaines et de sélectionner la meilleure transformation parmi eux.

4.3 Architecture parallèle pour le partitionnement Quadtree (*QPFIC*)

Nous avons étudié une architecture du traitement en parallèle pour *FPFIC* dans la partie précédente. Cette architecture parallèle pour le partitionnement *Fixed-size (FPFIC*) ne donne pas une meilleure performance que le partitionnement *flexible-size*. Le partitionnement *quadtree* est le plus communément utilisé, il est basé sur les blocs ranges à taille flexible. Cependant, il été difficile pour réaliser une architecture *VLSI* basée sur le partitionnement *flexible-size*.

Dans cette partie, nous étudions une architecture du traitement parallèle pour la compression d'image par fractale basée sur le partitionnement Quadtree (*QPFIC*) Qui utilise le partitionnement *flexible-size*. Cette architecture est conçue en modifiant le modèle *FPFIC* (vu précédemment) qui utilise seulement la communication des données locale et le module *FCM*. L'idée principale de cette architecture est que l'encodage par le partitionnement Quadtree peut être exécuté en utilisant les résultats d'encodage par le partitionnement *Fixed-size*.

L'architecture proposée exécute le *MAD* (*Moyenne Difference Absolu*). Elle permet de calculer *MAD* pour le *domain pool* à une profondeur maximale *p* puis calcule les *MADs* pour autres *domain pool* en ajoutant les *MADs* pour *domain pool* de profondeur maximale. Cette

architecture à un module de comparaison rapide de l'architecture *FPFIC* qui est capable de comparer le bloc range avec les huit transformations isométriques du bloc domaine par une rotation complète autour du centre.

4.2.1 L'architecture *QPFIC*

L'algorithme *QPFIC* à différents niveaux de *range pool* et *domain pool*, R^d, D^d, respectivement comme se présente dans la figure 5.6, où *d* est le niveau ou la profondeur du domaine ou range tel que:

$$d_{min} <= d <= d_{max.} \tag{5.6}$$

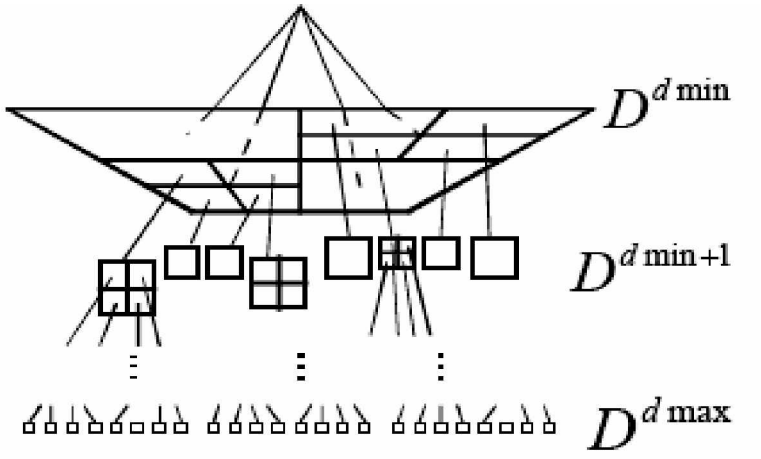


Figure 5.6 : *QPFIC* à différents niveaux

Pour réduire le calcul de différents *domain pool*, cet algorithme considère que le *MAD* de plus haute profondeur du domain pool D^{dmax} est un composant du *MAD* pour les autres *domain pool* parents. Cette architecture est capable de calculer les *MADs* pour D^{dmax} qui utilise le module de la comparaison rapide vu précédemment. Les résultats obtenus dans *PEs* sont propagés à la mémoire de D^{dmax-1}, puis à la mémoire pour les autres *domains pool* de différentes profondeurs.

La structure de base de *QPFIC* est semblable à l'architecture *FPFIC* à l'exception du module de mémoire. Le module de mémoire de *PE* consiste de trois parties, une partie contenant l'ensemble des blocs domaines et ranges, une autre partie pour la meilleure valeur *MAD* de D^{dmax}, et la dernière partie pour les *MADs* des autres *domains pool*.

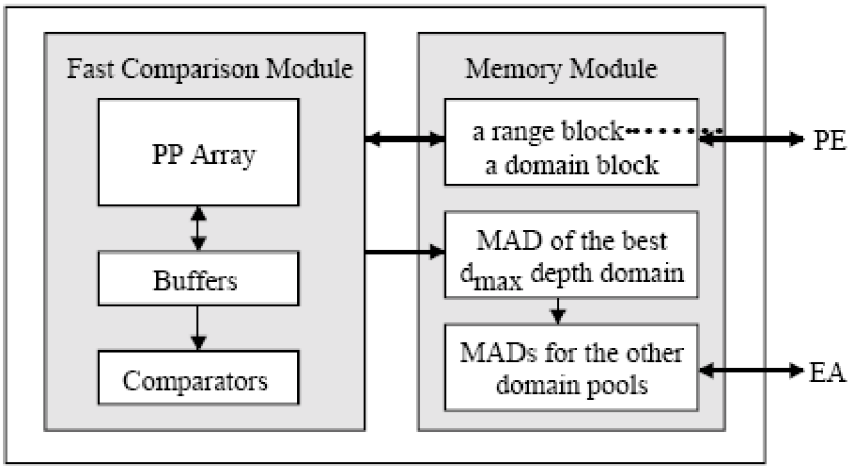


Figure 5.7: Le schéma fonctionnel de *PE*.

La figure 5.7 présente un schéma fonctionnel d'un *PE* qui a un module de la comparaison rapide et un module de mémoire. La procédure du codage de l'architecture proposée consiste en trois phases présentées ci-dessous:

- Phase 1: Génération de D^{dmax} et calcul de *MAD* pour D^{dmax}, et le calcul de *MAD* pour D^{dmax-1} qui utilise les résultats de D^{dmax}.
- Phase 2: Le calcul de *MAD* pour les autres *domain pool*.
- Phase 3: l'encodage Quadtree par les résultats de calcul du *MAD*.

PARTIE B : IMPLEMENTATION PARALLELE DE LA COMPRESSION D'IMAGE FRACTALE ET RESULTATS

Après une étude générale de l'architecture parallèle pour la compression d'image fractale, la deuxième partie de ce chapitre concerne la description d'une implémentation réelle de la technique présentée. Nous présentons tous d'abord l'architecture matérielle nécessaire pour cette réalisation, puis nous décrivons le modèle d'exécution proposé pour la réalisation de l'application.

Nous présenterons par la suite les résultats pratiques et les conditions d'expérimentation. On s'intéresse aux temps de compression parallèle et aux accélérations obtenues par rapport à l'algorithme séquentiel et l'algorithme statique en utilisant différentes images et les différentes configurations matérielles possibles.

Enfin, on peut conclure à partir de ces résultats de la validité du principe, l'avantage et l'importance d'introduire cette technique de parallélisation pour la compression d'image par fractale.

5 IMPLEMENTATION PARALLELE DU CODAGE FRACTALE

Comme on a vue préalablement (partie1) qu'il existe différentes méthodes de parallélisation d'algorithme de compression d'image selon l'approche fractale, nous avons proposé une nouvelle technique qui est basée sur la division d'image en plusieurs portions.

La parallélisation par cette méthode « portions de l'image » est l'approche que nous avons adopté pour réaliser notre application. Chaque portion de l'image est associée à un *PC* client. Chaque client permet d'effectuer un traitement autonome. Il permet l'extraction des blocs ranges et domaines concernant la portion d'image qui lui est associée ; puis effectue la comparaison entre eux pour trouver la meilleure transformation (le code *IFS*) d'une manière totalement indépendante des autres clients. Ceci, élimine la synchronisation et la communication entre les processus qui codent des parties différentes de l'image sur les autres *PC* esclaves.

6 ARCHITECTURE MATERIELLE

Notre implémentation est destinée à être exécutée sur une architecture *MIMD* à mémoire distribuée et plus précisément sur un réseau de stations de travail. Les noeuds du réseau sont

monoprocesseurs. Chaque nœud est un *PC* de type *Pentium IV* (2.4 *GHz*) et il dispose d'une mémoire vive de taille 256 *MO*.

Il faut que l'implémentation soit portable sur différents types *de réseaux de station*s de travail et différentes configurations (indépendante de la topologie et du nombre des noeuds).

7 LES RESEAUX DE STATIONS DE TRAVAIL

Les réseaux de stations de travail individuelles (*RS*) représentent l'architecture distribuée la plus familière et la plus répandue actuellement. Les performances toujours croissantes des stations individuelles et les faibles performances des supports de communication locaux ont en pratique, limité très sensiblement leur utilisation en tant qu'architectures parallèles distribuées. La disponibilité très récente de supports de communication rapides à faible coût comme *Myrinet* ou *SCI* (*Scalable Coherent Interface*) remet en cause cette situation et ouvre le marché des supercalculateurs et des serveurs aux réseaux de stations de travail.

Le rapport coût/performance de *RS* est nettement favorable quand on le compare à celui des architectures parallèles classiques, Actuellement les réseaux de stations sont les machines les plus utilisées dans le monde.

8 PRINCIPES DE BASE

- Toutes les stations sont égales vis-à-vis du réseau : il n'y a pas d'équipement maître de contrôle du réseau.
- La méthode d'accès employée est distribuée entre tous les équipements connectés.
- Le mode de transmission est de type bidirectionnel : les signaux transitent dans les deux sens, mais pas simultanément.
- On peut relier ou retirer une machine du réseau sans perturber le fonctionnement de l'ensemble.

9 CONFIGURATION DU RESEAU

La communication dans notre système est basée sur l'utilisation des routines de la norme *TCP/IP*. *TCP/IP* est issue de recherches effectuées par *DARPA* (*Defense Advanced Projects Agency*) pour réaliser un réseau fournissant des connexions sur une bande passante large entre les principaux sites informatiques gouvernementaux (*USA*).

Dans notre réseau, chaque machine doit avoir une adresse différente. De manière à ce que l'information qui lui est destinée lui soit effectivement livrée. C'est l'*Internet Protocol* (*IP*) qui contrôle ce schéma d'adressage.

10 OUTILS LOGICIELS

Comme nous l'avons évoqué dans le paragraphe précédent, les méthodes de communication de notre logiciel sont basées sur l'utilisation des protocoles du système *TCP*/*IP*. *TCP*/*IP* est indépendant de toute plate forme logicielle et matérielle. C'est à dire, nous pouvons utiliser n'importe quel système d'exploitation (*Unix, Linux, Windows,…*). Dans notre implémentation, nous avons installé sur chaque poste un système *Windows XP*.

Nous avons choisi Borland Delphi (6) comme un langage de programmation pour le développement de notre application parallèle ; c'est un environnement de programmation visuelle orienté objet permettant le développement rapide d'application Windows avec un minimum de programmation ; il offre aussi les moyens, les composants et les méthodes nécessaires et efficaces pour l'envoi et la réception des données et l'établissement des connexions entre le serveur et les clients , et il nous permet de développer une interface graphique puissante et simple à utiliser .

Borland Delphi offre les composants Sockets qui permettent d'établir des connections et d'échanger les données entre plusieurs postes de travail dans un réseau.

Une Socket supporte une communication bidirectionnelle entre un client et un serveur selon un protocole donné. Il faut associer aux Sockets une adresse sur la machine hôte .cette extrémité de connexion (l'adresse) est composée de deux champs:

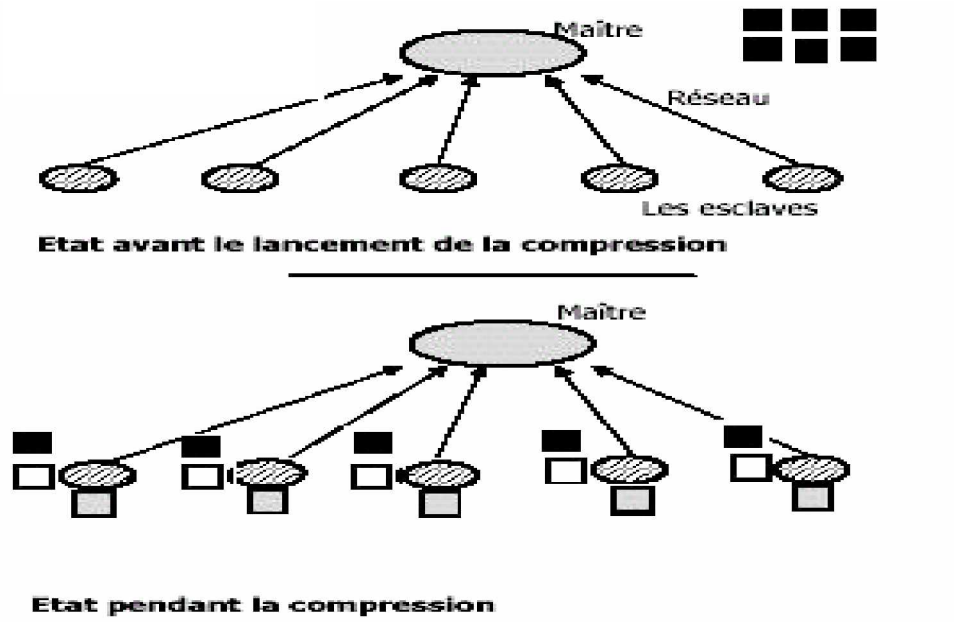
- une adresse IP
- un numéro de Port.

11 MODELE D'EXECUTION

La parallélisation des algorithmes de compression par blocs d'image sur architecture *MIMD* à mémoire distribuée convient bien, sur l'organisation *Maître*/*Esclaves* [96]. Le maître tient l'état général de l'application et possède une vue globale sur la progression de la compression [97]. C'est lui qui annonce le début et la fin du codage. Son rôle consiste à effectuer les tâches séquentielles de l'algorithme de compression et de distribuer les portions de l'image sur les

esclaves. Ces derniers codent les parties d'image qu'ils ont reçue et renvoient les résultats au maître.

De cette façon les esclaves possèdent seulement une vue partielle sur la progression de la compression, puisque chaque esclave ne voit que les tâches qui lui ont été assigné (figure 5.8). Après la réception de tous les résultats, le maître commence à construire le *Codestream*, le fichier final de l'image compressée. Cette tache est effectuée séquentiellement.



: Portion de limage avant traitement
: En cours de traitement
: Résultat (le code *IFS*)

Figure 5.8 : Modèle d'Exécution Maître/Esclave.

12 DEROULEMENT D'UNE COMMUNICATION

La communication est effectuée par l'envoi des messages. Nous allons examiner le déroulement d'une communication point-à-point, c'est à dire entre deux nœuds. Cette communication se déroule en deux temps :

- ***Envoi d'un message*** : le processus détenant des données à communiquer, les envois explicitement en précisant le destinataire. Les données sont placées dans un tampon d'émission, et sont ensuite acheminées sur le réseau ;
- ***Réception du message*** : le processus destinataire reçoit le message.

13 FONCTIONNEMENT DE L'APPLICATION

Nous avons développé deux applications, une de type serveur et l'autre de type client. Une copie de l'application client (esclave) est installée sur chaque machine de réseau. L'application serveur (maître) est installée sur une seule machine.

- **L'algorithme de compression**

Serveur (Host): pour un réseau Ethernet
Etablir la connexion avec les clients.
Charger l'image.
Envoyer les données initiales (portions de l'image) aux clients.
Terminer = faux
Tant Que non (Termine) Faire
    Bloquer jusqu'à un client envoi un signale de fin de codage.
    Marquer le client *i* comme terminé.
    Nbt =*Nbt* + 1 ;
    Si *Nbt* = Nombre de clients Alors
    Termine = vrai
    FinSi
    Fin T.Q
Pour chaque client Faire
    Collecter les résultats à partir du client
    et terminer le client
Fin Pour
Ecrire le résultat complet du codage fractal dans un fichier
FIN

Client (Node): pour un réseau Ethernet
Recevoir les données initiales (portion de l'image à coder).
Calculer le sous-ensemble des ranges correspondant.
Calculer le sous-ensemble des domaines correspondants.
Pour (chaque range) Faire
Pour (chaque domaine) Faire

Pour (chaque isométrie du domaine D_j) Faire
Calculer *rms* (range R_i, domaine D_j).
Si *rms* < meilleure_*rms* jusqu'à maintenant Alors
Meilleure_*rms* =*rms*
Sinon
Rediviser le bloc range en sous blocs ranges et les ajouter à l'ensemble des blocs Ranges et répété l'opération.
Finsi
Fin pour
Fin pour
Sauvegarder la transformation correspondante au meilleur *rms* range_domaine
Fin pour
Envoyer les résultats obtenus vers le maître.
FIN

14 EXPERIMENTATION ET DISCUSSION

Comme on a cité précédemment, notre application a été implémentée et testée sur un réseau de stations de travail dont la configuration est la suivante :

- Les noeuds du réseau sont des machines *Pentium IV* (2.4 *GHz*, 256 *MO RAM*).
- Système d'exploitation : *Windows XP*.
- langage de programmation *Borland Delphi*.
- Réseaux : *Ethernet* (*10BaseT*).

Nous avons testé notre application sur les images de différentes tailles en jouant sur le nombre d'esclaves (*nombre de clients*).

Les tableaux suivant contiennent les résultats détaillés des mesures des temps de compression que nous avons fait (résultats de la méthode proposée), et la comparaison avec la méthode statique avec des machines AMD de (1.3 GHz et 128 Mo de RAM) [3]. Les tests ont été effectués pour l'image (*LENA et SAN-FRANSISCO*) de différentes tailles (64 *ko*, 128 *ko*, 256 *ko*) en jouant sur le nombre de postes (*clients*) (1, 2, 4, 8).

IMAGE *LENA*							
			la méthode statique	la méthode proposée (par portions d'image)		temps séquentiel	l'efficacité client
Nb client	**N0 client**	**taille**	**Temps par client**	**temps par serveur**	**temps par client**		
2	0	**64ko**	88s	125s	59s	160s	1,49
	1		116s		59s		1,97
4	0		50s	40s	11s		4,55
	1		52s		12s		4,33
	2		44s		10s		4,40
	3		51s		20s		2,55
8	0		26s	31s	4s		6,50
	1		22s		2s		11,00
	2		23s		3s		7,67
	3		26s		2s		13,00
	4		23s		2s		11,50
	5		27s		3s		9,00
	6		24s		5s		4,80
	7		28s		4s		7,00
2	0	**128ko**	361s	203s	126s	232s	2,87
	1		459		118s		3,89
4	0		216s	116s	43s		5,02
	1		216s		39s		5,54
	2		191s		44s		4,34
	3		214s		41s		5,22
8	0		109s	75s	17s		6,41
	1		109s		15s		7,27
	2		90s		17s		5,29
	3		107s		18s		5,94
	4		114s		17s		6,71
	5		95s		16s		5,94
	6		103s		18s		5,72
	7		111s		17s		6,53
2	0	**256ko**	1559s	723s	421s	876s	3,70
	1		1826s		435s		4,20
4	0		885s	375s	269s		3,29
	1		884s		259s		3,41
	2		748s		259s		2,89
	3		880s		150s		5,87
8	0		395s	223s	58s		6,81
	1		374s		58s		6,45
	2		371s		57s		6,51
	3		447s		59s		7,58
	4		449s		58s		7,74
	5		453s		60s		7,55
	6		393s		58s		6,78
	7		440s		31s		14,19

Tableau 5.1 : Temps de la compression parallèle et séquentielle pour l'image *LENA* (64, 128, 256 ko) sur différents nombre de clients (1, 2, 4, 8 clients)

IMAGE SAN FRANSISCO							
			la méthode statique	la méthode proposée (par portions d'image)		temps séquentiel	l'efficacité client
Nb client	**N0 client**	**taille**	**Temps par client**	**temps par serveur**	**temps par client**		
2	0	64ko	88s	122s	60s	172s	1,47
	1		111s		59s		1,88
4	0		53s	47s	11s		4,82
	1		52s		11s		4,73
	2		44s		9s		4,89
	3		52s		20s		2,60
8	0		26s	29s	4s		6,50
	1		22s		2s		11,00
	2		23s		2s		11,50
	3		25s		3s		8,33
	4		23s		2s		11,50
	5		26s		4s		6,50
	6		24s		4s		6,00
	7		27s		5s		5,40
2	0	128ko	361s	207s	123s	241s	2,93
	1		433S		118s		3,67
4	0		217s	112s	45s		4,82
	1		215s		40s		5,38
	2		182s		44s		4,14
	3		214s		41s		5,22
8	0		109s	81s	18s		6,06
	1		109s		15s		7,27
	2		90s		17s		5,29
	3		107s		19s		5,63
	4		115s		17s		6,76
	5		95s		17s		5,59
	6		103s		15s		6,87
	7		111s		17s		6,53
2	0	256ko	1787s	719s	420s	902s	4,25
	1		1888s		435s		4,34
4	0		779s	389s	271s		2,87
	1		893s		260s		3,43
	2		945s		251s		3,76
	3		1036s		147s		7,05
8	0		394s	235s	60s		6,57
	1		454s		58s		7,83
	2		372s		59s		6,31
	3		371s		59s		6,29
	4		450s		57s		7,89
	5		476s		61s		7,80
	6		393s		55s		7,15
	7		434s		29s		14,97

Tableau 5.2 : Temps de la compression parallèle et séquentielle pour l'image SAN FRANSISCO (64, 128, 256 ko) sur différents nombre de clients (1, 2, 4, 8 clients)

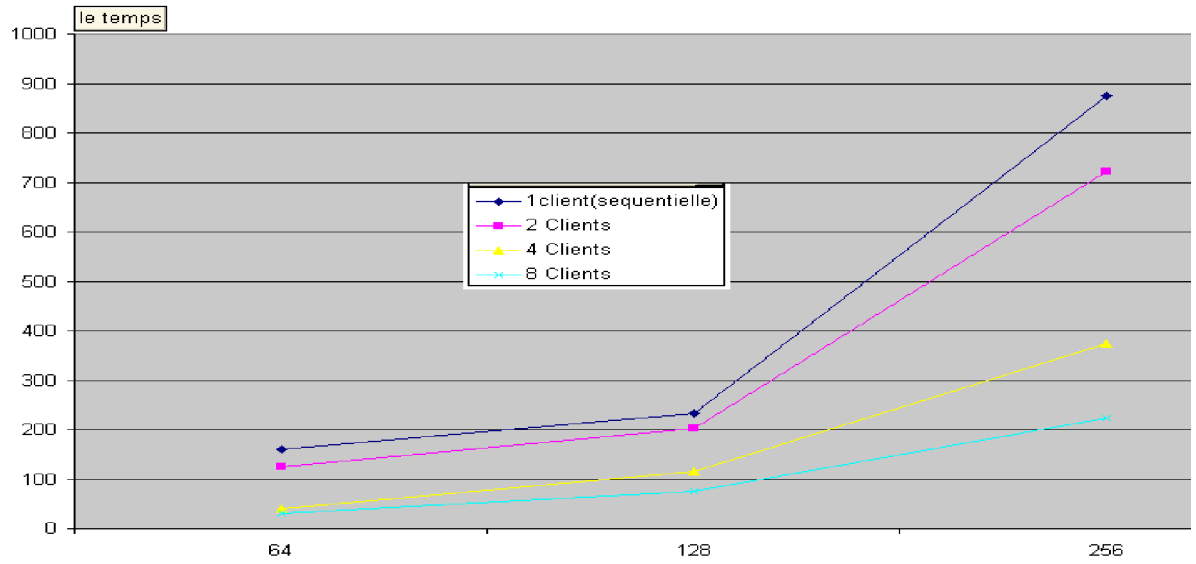


Figure 5.9: Temps de la compression serveur : en séquentielle et parallèle (1,2 ,4 ,8 postes) sur l'image de LENA de différentes taille (en prendre en considération le temps de communication)

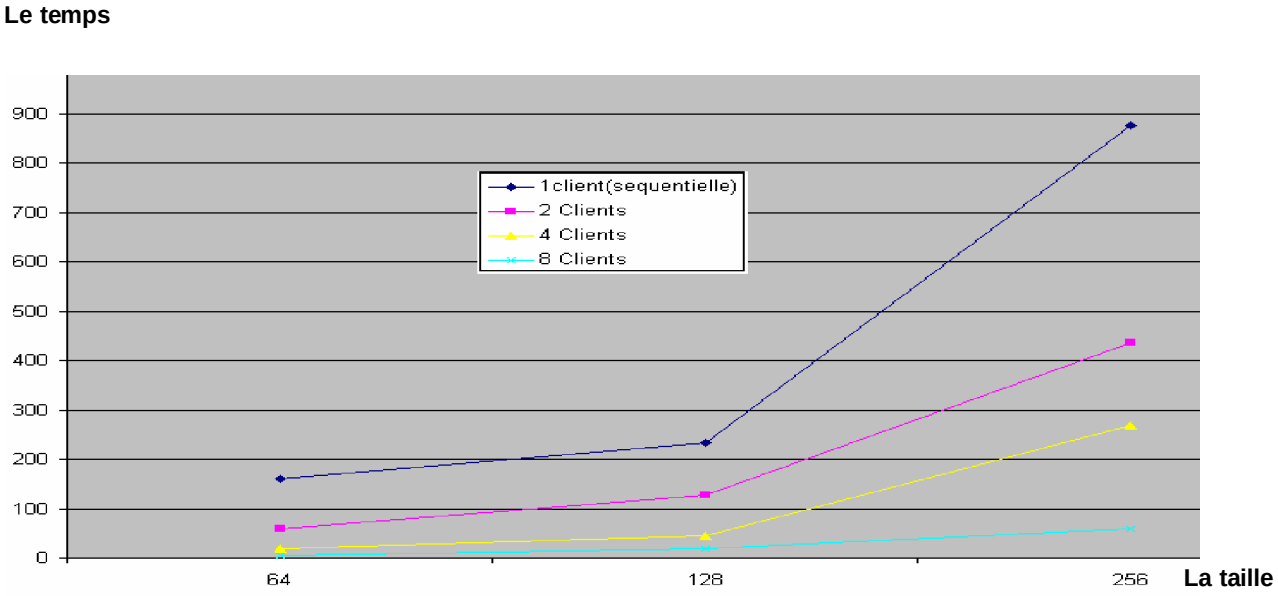


Figure 5.10 : Temps de la compression clients : en séquentielle et en parallèle (1,2 ,4 ,8 postes) sure l'image de *LENA* de différentes tailles (sans prendre en considération le temps de communication)

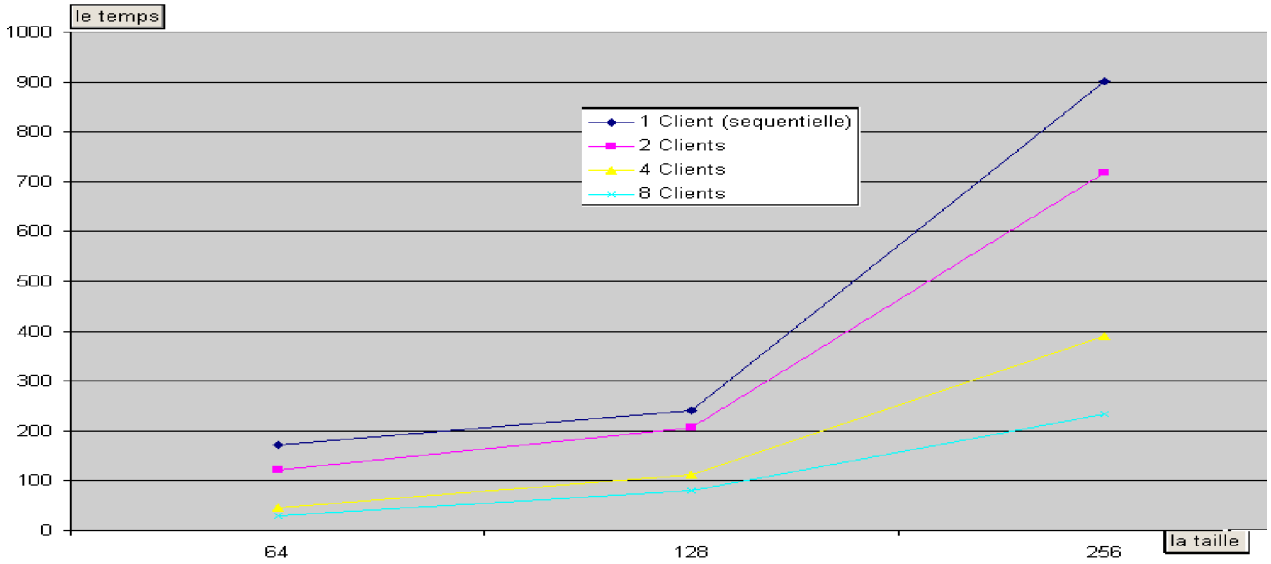


Figure 5.11: Temps de la compression serveur : en séquentielle et parallèle (1,2 ,4 ,8 postes) sur l'image de SAN FRANSISCO de différentes taille (en prendre en considération le temps de communication)

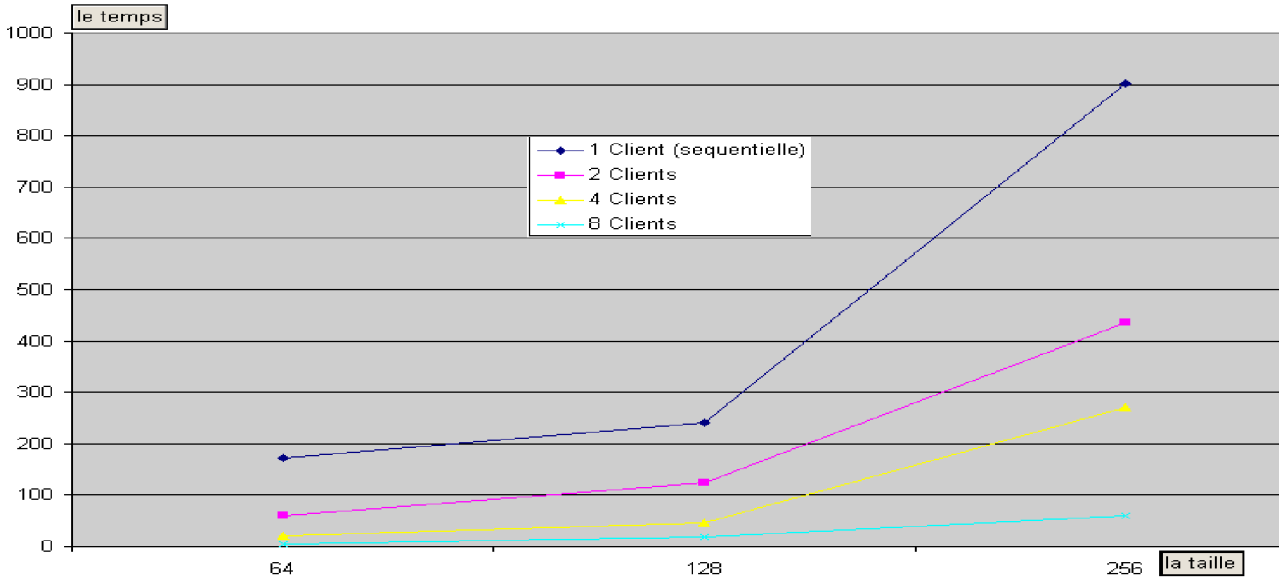


Figure 5.12 : Temps de la compression clients : en séquentielle et parallèle (1,2 ,4 ,8 postes) sure l'image de SAN FRANSISCO de différentes tailles (sans prendre en considération le temps de communication)

@ Discussion des résultats

Les courbes précédentes montrent les résultats obtenus après les tests effectués sur les images de «*LENA, SAN FRANSISCO* » de différentes tailles pour les différents nombres de postes : 1 poste (*séquentiel*), 2, 4, 8 postes dans des conditions différentes :

1- On prend en considération le temps de communication (*temps serveur*) (figures5.9, 5.11).

2- On néglige le temps de la communication (*temps clients*) (figures 5.10, 5.12).

Les résultats obtenus concernent le temps de la compression en parallèle et en séquentielle. On remarque (figures5.9, 5.11), que le temps du codage en parallèle et celui en séquentiel ont une différence minime dans le cas de 2 esclaves (*temps serveur*), et que les temps de la compression en parallèle diminuent en augmentant le nombre d'esclaves. A l'aide d'une analyse détaillée des temps de compression on peut déduire que la communication est ***la cause*** de la perte de performances, elle constitue la plus part du temps de calcul global. Le temps de communication inclus le temps d'attente de connexion avec le maître, le temps de sérialisation/désérialisation, et le temps de transmission.

Pour une configuration, la comparaison range–domaine est effectuée en même moment pour tous les esclaves (clients) pourtant que le temps de l'opération peut être différent. Mais les temps de transmission ne le sont pas car le maître est séquentiel et ne répond qu'à un esclave à la fois et les autres doivent attendre leur tour. Les performances de compression s'améliorent en augmentant le nombre des esclaves mais l'augmentation est médiocre, parce que le temps de compression sur les noeuds esclaves diminue mais pas le temps de communication qui reste considérable.

Le protocole *TCP*/*IP* est inadapté pour les applications de hautes performances sur réseaux de stations de travail parce qu'il n'est pas optimisé pour la communication dans un réseau local. En effet, la communication entre deux postes est assurée, qu'elle soit dans la même salle ou dans deux continents différents, et ceci de la même manière. C'est la raison principale de l'émergence des nouveaux protocoles de communications spécialisées pour les réseaux de stations de travail (*ATM*,…).

Les performances crêtes correspondent au temps de compression en négligeant le temps perdus en communication (*Overhead*). Dans notre cas si on néglige le temps de communication, qui constitue la source la plus importante de perte de performance, alors les

temps de compression seront tels que représentés dans (figures 5.10, 5.12). On remarque que les performances obtenues en négligeant le temps de Communication sont :

$$TCP <= TPS / NP \quad (5.7)$$

TCP = le temps d'esclave le plus lent

TCP : Temps de Compression Parallèle,

TPS : Temps de Compression Séquentielle,

NP : Nombre de Processeurs.

Ce résultat montre l'impact de l'utilisation des mécanismes de communication non adéquats (*TCP/IP, Ethernet 10BaseT*…), mais nous assure aussi que l'utilisation des mécanismes de communication plus rapides améliorerait considérablement les performances de notre application.

@ Comparaison de notre méthode avec la méthode statique

Les résultats que nous avons obtenus dans les tableaux (5.1, 5.2) montrent *l'accélération qui croit au nivaux de chaque client par l'application de notre méthode par rapport à la méthode statique*. Ce qui prouve que l'accélération est pratiquement liée au nombre réduit des blocs domaines d'une portion de l'image par rapport au nombre des blocs domaines d'une image entière (la méthode statique). Ceci permet de réduire le nombre de comparaisons ranges_domaines dans chaque client.

15 CONCLUSION

Dans ce chapitre nous avons montré que l'inconvénient principal de la compression d'image par fractale est le temps de calcul très élevé nécessaire pour déterminer le code *IFS* de l'image. Plusieurs approches ont été proposées pour résoudre ce problème (classification, parallélisation).

Dans notre travail nous avons choisi la technique de parallélisation par portions d'image comme une solution à ce problème en utilisant l'architecture *MIMD* à mémoire distribuée comme support pour l'application de notre algorithme. Nous avons montré les résultats des tests effectués sur différents nombres de postes esclaves et différentes tailles de l'image en exécutant notre application sur un réseau *Ethernet*. Nous avons conclue que la parallélisation de l'algorithme de compression d'image par fractale par l'approche proposée améliore considérablement la vitesse de codage.

Chapitre 5

Partie (A) : parallélisation d'un algorithme de compression fractale

PARTIE (B) : Implémentation parallèle de la compression d'image fractale et résultats

[88] Ch. Hufnagl , J. Hämmerle , A. Pommer ,A Uhl , et M. Vajtersic. Fractal Image Compression On Massively Parallel Arrays RIST++, Salzburg University , Austria 1999.

[89]] C.Hufnagl et A. Uhl . Algorithms for Fractal Image Compression On Massively Parallel SIMD Arrays . PCS' 97 International Picture Coding Symposium , Berlin , Germany 1997.

[90]: P.Paol , C. Moreno et L.Gugliemo , " Massively parallel processing approach to fractal image compression with near-optimal coefficient quantization " University La Sapienza , ENEA – HPCN Project –C.R Casaccia ,1997.

[91] J. HÄammerle, A. Uhl, *Parallel Algorithms for Fractal Image Coding on MIMD Architectures*, in Proceedings of the First International Conference on Visual Information Systems (Visual'96), Melbourne, February 1996, pp. 182-191.

[92] D. J. Jackson, G. S. Tinney, *Fractal Image Compression Using a Circulating Pipeline Computation Model*, Technical Report UA-CARL-95-DJJ-01, Computer Architecture Research Laboratory, The University of Alabama, March 1995.

[93] D. J. Jackson, G. S. Tinney, ***Performance Analysis of Distributed Implementations of a Fractal Image Compression*** *Algorithm*, Concurrency: Practice and Experience, 8(5) (June 1996), pp. 357-380.

[94] B. ZALAN , PETER . maximal processor utilization in parallel quadtree-based fractal image compression on MIMD architectures. STUDIA UNIV. BABES-BOLYAI, INFORMATICA, Volume XLIX, Number 2, 2004.

[95] L.Shinhaeng . "Parallel Processing Architecture for Fractal Image Compression". Doctor of Engineering Thesis. Department of Electrical and Communication Engineering Graduate School of Engineering Tohoku University, JAPAN. annee …. .

[96] A. Uhl et J. Hammerle, "Issues in implementing block-based image compression techniques on parallel MIMD architectures.", in J. Biemond and E.J. Delp, editors, Visual Communications and Image Processing '97, volume 3024 of SPIE Proceedings, pages 494-501, San Jose, Fevrier 1997.

[97] A. Aouar 1et M. Benmohammed 2.Algorithme Parallèle pour Compression d'Images Fixes**.** *1 Département d'Informatique, Université de Jijel, Jijel, Algérie. 2 Département d'Informatique, Université de Constantine, 25000 Constantine, Algérie.* SETIT 2004. International

Conference: **S**ciences of **E**lectronic, **T**echnologies of **I**nformation and **T**elecommunications March 15-20, 2004 – TUNISIA

+ IEEE de doctor

Serveur (Host): pour un réseau Ethernet

Etablir la connexion avec les clients.

Charger l'image.

Envoyer les données initiales (image, taille de l'image) aux clients.

Terminer = faux

Tant Que non(Termine) Faire

Bloquer jusqu'à un client envoi un signale de fin de codage.

Marquer le client i comme terminé.

Nbt =Nbt + 1 ;

Si Nbt = Nombre de clients Alors

Termine = vrai

FinSi

Fin T.Q

Pour chaque client Faire

Collecter les résultats à partir du client

et terminer le client

Fin Pour

Ecrire le résultat complet du codage fractal dans un fichier

FIN

CONCLUSION GENERALE

Ce mémoire est consacré à l'implémentation parallèle de la méthode fractale pour la compression des images fixes dont l'objectif est l'amélioration de la vitesse de codage. Cette méthode de compression constitue une voie de recherche d'actualité.

La représentation des formes fractales très complexe, par des transformations *IFS*, constitue la base de la compression fractale. Le but de la compression est la réduction du nombre moyen de bits nécessaire à la représentation des images pour des applications de stockage, ou de transmission.

Le début de ce mémoire est consacré à quelques concepts de l'image, et les besoins en compression d'images. En suite, nous traitons des différentes méthodes de codage des données, et de compression des images fixes. Nous avons ensuite introduit les notions liées à la théorie des *IFS* avant de montrer comment les utilisées pour compresser les images naturelles. Nous avons pour cela détaillé la méthode fractale qui consiste à coder les similarités entre blocs de l'image à l'aide des transformations affines contractantes.

La compression fractale est une méthode extrêmement performante pour stocker le maximum d'information dans le minimum de place. Ceci est expliqué par la nature du code *IFS* qui donne des taux de compression performants. Cependant la lenteur de cette méthode et sa complexité en terme de conception et de réalisation ne permet pas assez facilement de mener une telle amélioration.

Pour résoudre le problème de temps nous avons proposé une nouvelle implémentation parallèle de cette méthode en utilisant une architecture *MIMD* à mémoire distribuée qui a été simulée par un réseau de stations de travail dont le modèle utilisé est "*client /serveur*". Les noeuds du réseau sont monoprocesseurs. Chaque nœud est un *PC* de type *Pentium IV* (2.4 *GHz*) et il dispose d'une mémoire vive de taille 256 *MO*.

Notre expérimentation a été effectuée sur des images de différentes tailles. Les tests ont été effectués sur 2, 4 et 8 postes clients pour chaque image. Nous avons fait un prélèvement des temps de compression sur chaque poste ainsi que le temps global sur le serveur pour chaque test. Nous avons remarqué des différences entre les résultats obtenus. Ces différences sont dues au système informatique utilisé (*logiciel* et *matériel*).

Les résultats que nous avons obtenus aussi montrent l'efficacité *de la méthode proposée par rapport à la méthode statique*. Ce qui prouve que l'accélération est pratiquement liée au nombre réduit des blocs domaines d'une portion d'image par rapport au nombre des blocs domaines d'une image entière (la méthode statique).

Pour atteindre des performances meilleures et plus proches de la performance crête, nous proposons comme perspectives à notre travail d'utiliser un réseau de station de travail qui utilise un vrai support de communication de hautes performances (*ATM, SCI, FDDI,…*) et un maître *multi-Thread* sur une machine multiprocesseur (biprocesseur ou quadri-processeur).

Les recherches en compression fractale semblent s'ouvrir selon les voies:

@ Soucis d'optimisations et d'adaptabilité du partitionnement d'images par fractale.

@ Ouverture vers autres approches, ceci dans un contexte de méthodes *hybrides*

@ Optimisation des techniques de classification et de stockage du code *IFS*.

@ Une autre voie possible concerne l'ouverture sur les méthodes *Orientées Objets*, les *algorithmes génétique et les réseaux de neurones.*

BIBLIOGRAPHIE

[1] M. F. Barnsley, L.P. Hurd.," *Fractal Image Compression*", AK Petersbooks in Computer Science and Related Areas, 1993.

[2] A.E. Jacquin, "Image Coding Based on a Fractal Theory of Iterated Contractive Image Transformations", IEEE transactions on image processing,Vol.1, No. 1, pp. 18-30, 1992.

[3] C. Taouche,"*implémentation d'un environnement parallèle pour la compression d'images à l'aide des fractales* ", thèse magister, faculté des sciences de l'ingénieur, Département Informatique, Université de Mentouri Constantine, année 2005.

[4] K. Chakib, "*La Compression des Images Fixes par les Approximations Fractales Basée sur la Triangulation de Delaunay et la quantification Vectorielle"*, mémoire de fin d'étude, 1999.

[5] J. Fruitet, *"Outils et méthodes pour le traitement des images par ordinateur*", Université de Murne-la-vallée, France 2000.

[6] A.d'hardancourt, "fou du multimédia", Sybex, 1995

[7] F. Devaux, "*Filtrage d'image par réseau de neurones"*, thèse de maîtrise, Université de Paris 8, France 1997.

[8] M. Payan, "*A hierarchical image segmentation algorithm*", Rapport de Projet, 22 mai 2003.

[9] T.T. Alföldi, *"Introduction aux images numériques et aux techniques d'analyse numérique*", Centre canadien de télédétection, 1996.

[10] F. Davoine, "*Compression D'images par Fractales Basée sur la Triangulation de Delaunay"*, Thèse L'INPG, Institut National polytechnique de Grenoble, Décembre 1995.

[11] B. Maher, "*Partition Arborescentes et Compression Fractale*", Thèse de L'INPT, Toulouse, France, Janvier 2000.

[12] P. Beaurepaire, E. Beretta, "*Compression D'images Appliquée Aux Angiographies Cardiaques : Aspects Algorithmiques, Evaluation de La Qualité Diagnostique*" , Thèse N° 97 ISAL 0107, Institut National De Sciences Appliquées De Lyon, 1997.

[13] D. Saupe et M. Ruhl, "*Evolutionary Fractal Image Compression*", IEEE International Conference on Image Processing (ICIP-96), Lausanne, Sept. 1996.

[14] K. Belloulata , R.Stssinski et J. Konrad , "*Region-based image compression using fractals and shape-adaptive DCT"*, IEEE Processing , ICIP-99, Kobe , Japan Oct. 25-28 1999.

[15] F. Charot, "*Architectures Parallèles Spécialisées pour le Traitement D'image*", INRIA, N°1978, IRISA Campus Universitaire de Beaulieu, France, Avril 1993.

[16] A. Czihó, *"Quantification et Compression D'image. Application à l'imagerie Médicale"*, Thèse N°2142, Universite de Rennes 1, France, Mai 1999.

[17] A. Francoise, "*Mise en ligne d'images 3D traitements*", UV projet V0.1 2001.

[18] J.M. Pascal, "*La Compression des Données ou le flux Multimédia sous Contrainte*", DÉCOUVERTE N°283, Décembre 2000.

[19] E. Phuc, "*Compression Sélective et Focalisation Visuelle : Application au codage hybride de séquences d'images"*, Thèse N°1434, Université de Rennes 1, France, Décembre 1995.

[20] S. Renard, "*Compresser les images*", INTERGRAPHIC 2000 Séminaires, Formation Les nouveaux savoir-faire, Palais des Congrès - Porte Maillot, Paris Janvier 2000.

[21] D. Santacruz, R. Grosbois et T. ebrahimi, "*JPEG 2000 - la nouvelle norme pour le codage d'images*", © FI-3-1 du 27 mars 2001.

[22] A. Gersho et R.M. Gray, "*Vector Quantization and Signal Compression*", Kluwer Academic Publishers.1993.

[23] P.G. Howard et J.S. Vitter, "*Arithmetic coding for data compression*", Proceedings of the IEEE, 82(6):857-865, 1994.

[24] D.A. Huffman, "*A method for the construction of minimum redundancy codes*", Proc. of the IRE, 40:1098-1101, September 1952.

[25] M. Nelson, La compression de données : texte, images, sons, Editions Dunod, 1993.

[26] P. Cédric et F. Olivier, "*la compression d'image par ondelettes*", Lycée Pothier MPSI 3, T.I.P.E. d'informatique 1998.

[27] J. Pugliesi et C. Piovano, "*le tatouage d'images où: watermarking*" .Université de Nice, Licence d'Informatique Travail d'études, Juin 2004.

[28] S. Renard, "*Compression de données*", Conférence du 14 octobre 1999.

[29] M. Ammar, "*Optimisation D'un Schéma De Codage D'image à Base D'une TCD, Application à Un Codeur JPEG Pour L'enregistrement Numérique à Bas Débit*", Thèse Doctorat, Ecole Nationale Supérieure des télécommunications, 14 Janvier 2002.

[30] C. Stalder, "*Les algorithmes de compression sans perte*", Ecole spécialisée de suisse, 7 janvier 2005.

[31] A. Skodras, C. Christopoulos, and T. Ebrahimi, "*The jpeg 2000 still image compression standard*", Signal Processing Magazine, 18 :36{58, September 2001.

[32] C. Delgorge,"*Proposition et Evaluation de techniques de compression d'images ultrasonores dans le cadre d'une télé-échographie robotisée*", Thèse de Doctorat, Université D'orleans , 10 décembre 2005

[33] Mr. Brault, Mme. Dougherty, "*Les formats de compression d'image*", Institut Universitaire de Technologie de Tours, Département Génie Électrique et Informatique Industrielle. 2004.

[34] G. Mercier, C. Roux, et G. Martineau, "*Technologies du Multimédia*", ENST Bretagne, dpt ITI, BP 832, F-29280 Brest, France.15 janvier 2003.

[35] A. Ali-Pacha et N. Hadj-Said, "*Compression des Images Fixes par Fractale : Partitionnement Quadtree*", Proceeding, FRACTALES 2000, pages 12-19, Université Mentouri Constantine, Algérie, Novembre 2000.

[36] Y. Fisher, D. Rogovin, et T.P. Shen, "*A Comparaison of Fractal Methods with DCT and Wavelet*s", In Neural And Stochastic Methods In Image And Signal Processing III, Volume Proc. SPIE 2304-16, San Diego.

[37] H. Krupnik, D. Malah, and E. Karnin, "*Fractal Representation Of Images Via The Discrete wavelet Transform"*, appears in *IEEE* 18th Conv. of EE in Israel, Tel-Aviv, Department of Electrical Engineering Technion-Israel Institute of Technology, Haifa 32000, Israel March 1995.

[38] E. Lutton, "*Algorithmes Génitiques et Fractales*", Rapport D'habilitation Université de Paris XI Orsay, 11 Février 1999.

[39] G. Robert, F. Davoine et J.M. Chssery, "*Compression D'image Par Fractales A Base De Maillages Polygonaux De Voronoi*", Laboratoire TIMC-IMAG Projet Accord Entre GDR-PRC ISIS, Le CCETT Le CENT 1997.

[40] M.F. Barnsley, V. Ervin, D. Hardin, and J. Lancaster, "*Solution Of An Inverse Problem For Fractals And Other Sets*", Proc. Natl. Acad. Sci. USA,83:1975-1977, April 1986.

[41] T.J. Bedford, F.M. Dekking, M. Breeuwer, M.S. Kean, and D. Van Schoone-Veld, "*Fractal coding of monochrome images*", Signal Processing: Image Communication, 6:405-419,1994.

[42] C.A. Cabrelli, B. Forte, U.M. Molter, and E.R. Vrscay, "*Iterated Fuzzy Set Systems: A New Approach To The Inverse Problem For Fractals And Other Sets*", Journal of Mathematical Analysis and Applications, 171(1):79-100; November 1992.

[43] Y. Fisher, "*Fractal Image Compression* : theory and application to digital image ",Springer Verlag, New York, 1995.

[44] B. Hurtgen, F. Muller, and C. Stiller, "*Adaptive Fractal Coding Of Still Pictures*", Picture Coding Symposium, 1993.

[45] Klinger, "*Data Structures And Pattern Recognition*", In Proc. Int, Joint Conf. on Pattern Recognition, pages 497-498, 1973.

[46] W.G. Kropatsch, M.A. Neuhausser, I.J. Leitgeb, and H. Bischof, "*Combining Pyramidal And Fractal Image Coding*", In Proc.11th ICPR, The Hague, The Netherlands, volume 3, pages 61-64, 1992.

[47] J. Lévy_Véhel, and A. Gagalowicz, "*Fractal Approximation Of 2-D Object*", Technical Report 1187, INRIA–Rocquencourt, France, 1990.

[48] E. Lutton and J. Lévy_Véhel, "*Optimization Of Fractal Functions Using Genetic Algorithms*", In Fractal 93, Londres, 1993.

[49] G. Mantica, and A. Sloan, "*Chaotic Optimization And The Construction Of Fractals: Solution Of An Inverse Problem*", Complex Systems, 3:37-62, 1989.

[50] G.E. Oien, and S. Lepsoy, "*Fractal-Based Image Coding With Fast Decoder Convergence*", Signal Processing, 40:105-117, October 1994.

[51] R. Rinaldo, and A. Zakhor, "*Inverse And Approximation Problem For Two Dimensional Fractal Sets*", IEEE Transactions on Image Processing, 3(6):802-820, November 1994.

[52] A. Rosenfeld, "*Quadtrees And Pyramids For Pattern Recognition And Image Processing*", In Proc. 5th. Int. Conf. on Pattern Recognition, Pages 802–811, University of Maryland, 1980

[53] H. Samet, "*Region Representation: Quadtrees From Binary Arrays"*, CVGIP, 13:88-93, 1980.

[54] H. Samet, "*The Quadtree And Related Hierarchical Data Structures*", ACM Comput. Surv.16:178-260, 1984.

[55] R. Vrscay, "*Moment And Collage Methods For The Inverse Problem Of Fractal Construction With Iterated Function Systems*", In Fractal 90 conference, June 1990.

[56] E. Bertin, F. Parazza, and J-M. Chassery, "*Segmentation And Measurement Based On 3d Voronoi Diagram: Application To Confocal Microscopy*", Computerized Medical Imaging and Graphics, Vol. 17, pp. 000-008, 1993.

[57] J.M. Chassery, M. Melkemi, "*Diagramme De Voronoi Applique A La Segmentation D'images Et A La Detection D'evenements En Imagerie Multi-Sources*", Traitement du Signal, Vol. 8, No. 3, pp. 155-164, 1991.

[58] X. Chen, F. Schmitt, "*Split-And-Merge Image Segmentation Based On Delaunay Triangulation*", Proc. of The 7th Scandinavian Conf. on Image Analysis, Aalborg, Denmark, Vol. 2, pp. 910-917, aug.13-16 1991.

[59] A. Arnedo, F. Argoul, E. Barcy, J. Elezgaray and J.F. Muzy, "*Ondelettes, Multi fractales ET Turbulences"*, Diderot Eds-Arts et sciences, 1995.

[60] M. Barnsley, "*Fractals Everywhere"*, Academic Press, Inc., 1988.

[61] E. Lutton, J. Lévy-Véhel, G. Cretin, P. Glevarec and C. Roll, "*Mixed IFS : resolution of the inverse problem using Genetic Programming*", Technical Report 2631, INRIA, August 1995, Accepted for publication in Complex Systems.

[62] K. Berkner, "*A Wavelet-Based Solution To The Inverse Problem For Fractal Interpolation Functions"*, in Lévy Véhel , pages 81–92.

[63] H. Radha, R. Leonardi, and M. Vetterli, "*A Multiresolution Approach To Binary Tree Representations Of Images"*, in *ICASSP*, Pages 2653–2656, 1991.

[64] Y. Cohen, M. Landy, and M. Pavel, "*Hierarchical Coding Of Binary Images"*, IEEE transactions on Pattern Analysis and Machine Intelligence, vol. 7, no. 3, Pages 284–298, May 1985.

[65] P. Chou, T. Lookabaugh, and R. Gray, "*Entropy-constrained Vector Quantization"*, IEEE transactions on acoustics, speech and signal processing, vol. 37, no. 1, Pages 31–42, January 1989.

[66] X. Wu, *"Image Coding By Adaptative Tree-Structured Segmentation"*, IEEE transactions on information theory, vol. 38, no. 6, Pages 1755–1767, November 1992.

[67] X. Wu, and Y. Fang, "*A segmentation-based predictive multiresolution image coder"*, IEEE transactions on image processing, vol. 4, no. 1, Pages 34–47, January 1995.

[68] F. Denatale, G. Desoli, and D. Giusto, *"Segmentation-Based Hybrid-Coding Of Colo Images"*, in *ICASSP*, Pages 2757–2759, 1991.

[69] R. Dansereau, and W. Kinsner, "*Perceptual Image Compression Through Fractal Surface Interpolation*", in CCECE, 1996.

[70] R. Distasi, M. Nappi, and S. Vitulano, "*Image Compression By Tree Triangular Coding*", IEEE transactions on communications, vol. 45, no. 9, Pages 1095–1100, Sept. 1997.

[71] R. Sequeira, and F. Prêteux, *"Discrete Voronoï Diagrams and the SKIZ operator : A Dynamic Algorithm*", IEEE transactions on Pattern Analysis and Machine Intelligence, vol. 19, no. 10, Pages 1165–1170, 1997, October.

[72] F. Davoine, and J.M. Chassery, "*Compression d'Images par Fractales"*, une Journées sur les Nouvelles techniques pour la compression et la représentation des signaux audiovisuels, Janvier 1995.

[73] F. Davoine, and J.M. Chassery, *"Compression d'Images par Fractales"*, in CORESA, 1996.

[74] E. Bertin, *"Diagrammes de Voronoï 2D et 3D : Application en analyse d'images"*, Thèse de doctorat, Université J. Fourier, Grenoble, France, 1994.

[75] V. Ramasubramanian, and K. Paliwal, *"Voronoï Projection-Based Fast Nearest-Neighbor Search Algorithms: Mapping Table-Based Search Techniques"*, Digital Signal Processing, vol. 7, Pages 260–277, 1997.

[76] E. Goubault, "*Informatique Parallèle Et Distribution*", Cours de parallélisme, Ecole Polytechnique, France, 1999.

[77] C. Laurent, *"Adéquation Algorithmes et Architectures Parallèles pour la Reconstruction 3D en Tomographie X*", Thèse De Doctorat, Université De Claude Bernard Lyon 1, France, Janvier 1996.

[78] P. Lalevee, *"Algorithmes Parallèles Par Flux Dans Les Graphes Des Fondements Aux Applications"*, Thèse De Doctorat, Université De Paris 6, France, Décembre 1995.

[79] J. Sansonnet, "*Architecture des Machines Parallèles*", Cours en ligne, CNRS, Orsay, France, 2002.

[80] J. Brown & J. Terzopoulos, "*Real-time computer vision*", Cambridge University Press, 1994.

[81] V. Cantoni & A. Petrosino, "*Neural Recognition in a Pyramidal Structure*", IEEE Transactions on Neural Networks, vol. 13(2), Pages 472–480, 2002.

[82] H. Li & M. Maresca, "*Polymorphic-torus network"*, IEEE Transactions on Computers, vol. 38(9), Pages 1345–51, Sept. 1989.

[83] A. El gamal, J. Rose & A. Sangiovanni-Vincentelli, "*Proceedings of IEEE, 81*", In Architecture of Field Programmable Gate Arrays, Pages 1013–1029, 1993.

[84] L.H. Jamieson, E.J. Delp, J. Wang, C. Li & F.J. Weil, "*A Software Environment for Parallel Computer Vision"*. IEEE Computer, vol. 25(2), Pages 73–75, 1992.

[85] J. Sérot & D. Ginhac, "*Skeletons for parallel image processing: an overview of the SKiPPER project"*, Parallel Computing, vol. 28, no. 12, Pages 1785–1808, Dec 2002.

[86] J. Seinstra & D. Koelma, *"User Transparency: a Fully Sequential Programming Model for Efficient Data Parallel Image Processing"*, Concurrency and Computation: Practice and Experience, 2004.

[87] N. Sicard, "*ANET : un environnement à parallélisme dedonnées pour l'analyse d'image*", Thèse de Doctorat en sciences de l'université Paris XI orsay, 12 juillet 2004.

[88] Ch. Hufnagl, J. Hämmerle, A. Pommer, A. Uhl, et M. Vajtersic, "*Fractal Image Compression On Massively Parallel Arrays RIST++"*, Salzburg University , Austria 1999.

[89] C. Hufnagl, et A. Uhl, "*Algorithms for Fractal Image Compression On Massively Parallel SIMD Arrays*", PCS'97 International Picture Coding Symposium, Berlin, Germany 1997.

[90] P. Paol, C. Moreno, et L. Gugliemo, "*Massively parallel processing approach to fractal image compression with near-optimal coefficient quantization*", University La Sapienza, ENEA–HPCN Project–C.R Casaccia, 1997.

[91] J. HÄammerle, A. Uhl, "*Parallel Algorithms for Fractal Image Coding on MIMD Architectures*", In Proceedings of the First International Conference on Visual Information Systems (Visual'96), Melbourne, pp. 182-191, February 1996.

[92] D.J. Jackson, G.S. Tinney, "*Fractal Image Compression Using a Circulating Pipeline Computation Model*", Technical Report UA-CARL-95-DJJ-01, Computer Architecture Research Laboratory, The University of Alabama, March 1995.

[93] D.J. Jackson, G.S. Tinney, "*Performance Analysis of Distributed Implementations of a Fractal Image Compression Algorithm*", Concurrency: Practice and Experienc), pp. 357-380e, 8(5) June 1996.

[94] B. Zalan, Peter, "*Maximal Processor Utilization in Parallel Quadtree-Based Fractal Image Compression on MIMD Architectures"*, STUDIA Univ, Babes-Bolyai, INFORMATICA, Volume XLIX, Number 2, 2004.

[95] L. Shinhaeng, "*Parallel Processing Architecture for Fractal Image Compression*". Doctor of Engineering Thesis, Department of Electrical and Communication Engineering Graduate School of Engineering Tohoku University, JAPAN, 2000.

[96] A. Uhl et J. Hammerle, "*Issues in implementing block-based image compression techniques on parallel MIMD architectures*", In J. Biemond and E.J. Delp, editors, Visual Communications and Image Processing '97, volume 3024 of SPIE Proceedings, pages 494-501, San Jose, Fevrier 1997.

[97] A. Aouar et M. Benmohammed, "*Algorithme Parallèle pour Compression d'Images Fixes*", SETIT 2004, International Conference: **S**ciences of **E**lectronic, **T**echnologies of **I**nformation and **T**elecommunications, March 15-20, 2004, Tunisie.

[98] A. Ameziane,"*compression d'images par fractales*", thèse magister, faculté des sciences de l'ingénieur, département informatique, université de Batna, année 2002.

[99] F.Agen, J.Michot , " *Projet de C : La Compression Fractale, Méthodes de Jacquin, Subdivisions de triangles et Delaunay*", Cours en ligne, Polytech'Tours-D´epartement Informatique, Juin 2005.

ANNEXE A

L'ALGORITHME DE *JACQUIN*

L'algorithme de *A.Jacquin* est fondé sur une partition *R* régulière à géométrie carrée. L'image est partitionnée en blocs destination (ou blocs parents) carrés de taille fixe égale à B^2 pixels ($B = 8$). L'algorithme recherche, pour chacun des blocs destination r_n, le bloc source d_{on} de taille D^2 ($D = 2B$) qui minimise l'erreur $d(r_n, \hat{r}_n)$ où $\hat{r}_n$ est l'approximation de r_n calculée à partir du bloc source $d_{\alpha(n)}$. La mesure d'erreur *d* est donnée par :

$$d\,(r_n, \hat{r}_n) = \sum_{j=1}^{B^2} (r_{nj} - \hat{r}_{nj})^2 \qquad \text{(A.1)}$$

Où r_n et $\hat{r}_n$ sont respectivement les valeurs des pixels d'indice *j* à l'intérieur du bloc original r_n et du bloc collé $\hat{r}_n$. L'opération de collage, appelée collage parent, est détaillée dans le paragraphe suivant.

COLLAGE D'UN BLOC SOURCE SUR UN BLOC DESTINATION

L'opération de collage d'un bloc source $d_{\alpha(n)}$ sur un bloc destination r_n, réalisée par la transformation *Wn*, se décompose en deux parties :

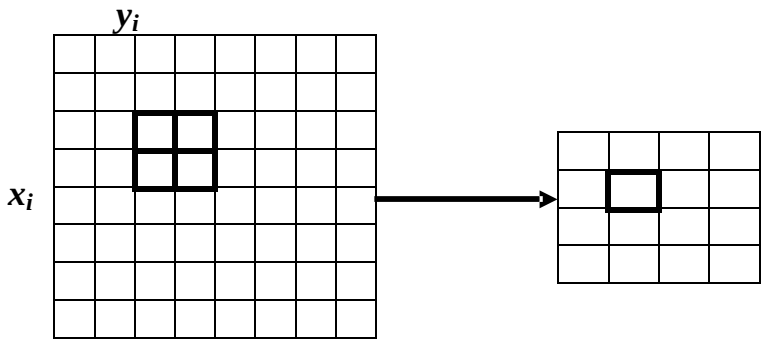
- ***une transformation spatiale*** **:** déforme le support du bloc $d_{\alpha(n)}$;
- ***une transformation massique*** **:** agit sur la luminance des pixels du bloc $d_{\alpha(n)}$ déformé.

Ces deux points sont détaillés dans la suite de ce paragraphe.

- **La transformation spatiale (géométrique) :** ramène le bloc source $d_{\alpha(n)}$ de taille D^2 au-dessus du bloc destination r_n de taille B^2. Le bloc ainsi transformé, noté $b_2^{(n)}$, est obtenu par décimation des pixels du bloc source. un pixel de coordonnées (x_i, y_j) dans $b_2^{(n)}$ est donné par l'équation suivante:

$$b_2^{(n)}(x_i, y_j) = \frac{1}{4}\left[d_{\alpha(n)}(x_k, y_l) + d_{\alpha(n)}(x_k, y_{l+1}) + d_{\alpha(n)}(x_{k+1}, y_l) + d_{\alpha(n)}(x_{k+1}, y_{l+1})\right] \qquad \text{(A.2)}$$

Dans laquelle (x_k, y_l) (sont les coordonnés d'un pixel de niveau de gris noté t_i à l'intérieur du bloc $d_{\alpha(n)}$.



ü **La transformation massique**

Elle agit sur la luminance des pixels du bloc $b_2^{(i)}$ pour approximer le bloc destination r_i. Les transformations massiques utilisées appartiennent à deux classes :

1ere classe : Elles agissent seulement sur les valeurs des pixels :

- *Absorption par g_0 :* La valeur du pixel (i, j) est affectée (forcée) par la valeur de g_0.

$$M_0(b_2^{(i)})_{i,j} = g_0,\ g_0 \in \{0,...,255\} \quad (A.3)$$

, *Translation de luminance :* On ajoute à la valeur de pixel (i, j) une constante Δg.

$$M_1(b_2^{(i)})_{i,j} = b_2^{(i)}{}_{i,j} + \Delta g,\ \Delta g \in \{-255,...,0,...,255\} \quad (A.4)$$

ƒ *Echelonnage par α ou modification de contraste (contrast scaling) :*

$$M_2(b_2^{(i)})_{i,j} = \alpha\, b_2^{(i)}{}_{i,j},\ \alpha \in [0,1] \quad (A.5)$$

„ *Inverse de couleurs :* On inverse la couleur du pixel (i, j) en la substituant à la valeur maximale des niveaux de gris g_{max}.

$$M_3(b_2^{(i)})_{i,j} = g_{max} - b_2^{(i)}{}_{i,j} \quad (A.6)$$

Où $M_k(b_2^{(i)})_{i,j}$ est la transformation massique numéro k.

2eme classe : Elles ne modifient pas les valeurs des pixels, mais les mélangent ou les déplacent dans les blocs, elles sont appelées les isométries, elles sont au nombre de huit :

1°) Identité : $$ISO_0(b_2^{(i)})_{i,j} = b_2^{(i)}{}_{i,j.} \quad (A.7)$$

2°) Réflexion orthogonale par rapport à l'axe de symétrie horizontal (i = (B-1)/2) :

$$ISO_1(b_2^{(i)})_{i,j} = b_2^{(i)}{}_{B\text{-}1\text{-}i,j.} \quad (A.8)$$

3°) Réflexion orthogonale par rapport à l'axe de symétrie vertical (*j* =(*B*-1)/2) :

$$ISO_2(b_2^{(i)})_{i,j} = b_2^{(i)}{}_{i,B-1-j.} \quad \text{(A.9)}$$

4°) Réflexion orthogonale par rapport à la première diagonale (*i* = *j*) :

$$ISO_3(b_2^{(i)})_{i,j} = b_2^{(i)}{}_{j,i.} \quad \text{(A.10)}$$

5°) Réflexion orthogonale par rapport à la seconde diagonale (*i*+*j* =*B*-1) :

$$ISO_4(b_2^{(i)})_{i,j} = b_2^{(i)}{}_{B-1-j,\,B-1-i.} \quad \text{(A.11)}$$

6°) Rotation à +90° : $ISO_5(b_2^{(i)})_{i,j} = b_2^{(i)}{}_{j,\,B-1-i.}$ (A.12)

7°) Rotation à +180°: $ISO_6(b_2^{(i)})_{i,j} = b_2^{(i)}{}_{B-1-i,\,B-1-j.}$ (A.13)

8°) Rotation à -90° : $ISO_7(b_2^{(i)})_{i,j} = b_2^{(i)}{}_{B-1-j,\,i.}$ (A.14)

ü Classification des blocs

La complexité de cette transformation (massique) dépend de la nature du bloc r_n considéré. Jacquin propose pour cela de classifier les blocs carrés à l'aide de la méthode développée par *Ramamurthi* et *Gersho* **:** trois classes regroupent :

- ***Les blocs homogènes (blocs shade)*** *:* blocs presque uniformes, qui présentent des variations de luminance négligeables ;
- ***Les blocs texturés (blocs midrange)*** *:* se caractérisent par des gradients moyens, mais ne définissent aucun contour ;
- ***Les blocs avec contours (blocs edge)*** *:* simples et divisés, ils présentent une brusque variation de niveau de gris le long de la limite d'un objet.

Selon la classe à laquelle appartient le bloc destination r_n, une transformation massique plus ou moins complexe lui est associée. Celle-ci dépend du bloc décimé $b_2^{(n)}$ et/ou d'un bloc constant $b_1^{(n)}$ formé de pixels tous égaux à un. Au bloc $b_2^{(n)}$ sera associé un coefficient d'échelle noté $\beta_2^{(n)}$ et au bloc $b_1^{(n)}$ un coefficient de décalage noté $\beta_1^{(n)}$. Le choix du type de transformation est fonction de la procédure suivante :

- **Si le bloc r_n est homogène**: absorption des niveaux de gris de r_n. Aucune recherche de blocs source $d_{\alpha(n)}$ n'est effectuée. La transformation de r_n, codée sur I_m bits, est donnée par : $\hat{r}_n = \beta_1^{(n)} b_1^{(n)}$ où l'entier $\beta_1^{(n)} \in [0, 255]$. (A.15)

- **Si le bloc r_n est texturé** : recherche du bloc source $d_{\alpha(n)}$, puis modification de contraste et décalage. La transformation de $d_{\alpha(n)}$, codée sur I_m bits, est donnée par :

$$\hat{r}_n = \beta_2^{(n)} b_2^{(n)} + \beta_1^{(n)} b_1^{(n)} \quad \text{(A.16)}$$

Où $\beta_2^{(n)}$ appartient à l'ensemble {0.7, 0.8, 0.9, 1.0} et l'entier $\beta_1^{(n)} \in$ [-255, 255].

- **Si le bloc r_n contient des contours** : recherche du bloc source $d_{\alpha(n)}$, puis modification de contraste, décalage et isométrie discrète τ_n (rotations de 0, +90, -90 et +180 degrés, réflexions suivant les axes de symétrie verticaux et horizontaux, et réflexions suivant les deux axes diagonaux). La transformation de $d_{\alpha(n)}$, codée sur I_m bits, est donnée par :

$$\hat{r}_n = \tau_n (\beta_2^{(n)} b_2^{(n)} + \beta_1^{(n)} b_1^{(n)}) \quad \text{(A.17)}$$

Où $\beta_2^{(n)}$ appartient à l'ensemble {0.5, 0.6, 0.7, 0.8, 0.9, 1.0} et $\beta_1^{(n)}$ est compris entre –255 et 255.

Lorsque le bloc destination est texturé ou recouvre des contours, le coefficient d'échelle β_2 est calculé de manière à rendre égaux les écarts types des deux blocs $b_2^{(n)}$ et r_n. Il est ensuite approximé par le coefficient appartenant à un ensemble de valeurs prédéfinies réelles positives, et inférieures à un. Le coefficient de décalage β_1 est calculé de manière à ce que les moyennes des pixels des deux blocs $b_2^{(n)}$ et r_n soient égales. Il n'est pas quantifié.

GENERATION D'UN DOMAINE POOL

La recherche exhaustive du bloc source $d_{\alpha(n)}$ est effectuée en déplaçant sur le support de l'image un bloc carré d'un pas de $\delta_h = \delta_v = 4$ pixels dans les directions horizontales et verticales.

Lorsque deux blocs sont comparés, les huit isométries discrètes sont considérées. Pour une image de taille 256*x*256, une telle recherche est ainsi effectuée au travers d'une librairie composée de Q blocs source où :

$$Q = k\left(\frac{m - D}{h} + 1\right)^2 \quad \text{Avec} \quad \text{(A.18)}$$

K =8 isométries, m = coté de l'image en pixels (256), D = coté du bloc source (16), h = 4

Nous aurons $Q = 8(\frac{256-16}{4}+1)^2$ = 29768 blocs sources.

Dans le cas d'une image de taille 512*x*512 pixels, le nombre *Q* de blocs source s'élève à 125000.

Les blocs source dans le domaine pool sont classifiés selon leurs caractéristiques géométriques perceptuels, utilisant l'étude de la classification de bloc d'image donnée par *Ramamurthi* et *Gersho*.

CODAGE DE L'OPERATION DE COLLAGE SUR UN BLOC DESTINATION

La mémorisation du collage d'un bloc source $d_{\alpha(n)}$ sur un bloc destination r_n comprend :

- l'indice du bloc source $d_{\alpha(n)}$ retenu parmi les *Q* blocs de la librairie à condition que ceux-ci soient rangés dans une liste de blocs et que leur organisation sur le support de l'image soit connue.
- l'isométrie utilisée lors du collage (une parmi huit)
- les coefficients β_1 et β_2 de la transformation massique

Cette information est associée à chacun des *N* blocs destination de la partition *R*.

Elle est codée sur un nombre variable de bits.

ANNEXE B

Dans ce qui va suivre, nous allons donner une brève description du coté visuel de notre application sans oublier de présenter quelques routines *Delphi* pour une implémentation parallèle de la méthode proposée.

PARALLELISATION

Pour l'implémentation parallèle appliquée à la compression fractale d'images, nous utilisons dans notre application, un seul maître pour plusieurs esclaves.

Le maître et les esclaves communiquent grâce au mécanisme de protocole *TCP/IP* qui est un point fort dans *Delphi* qui contient des composants, qui permet d'envoyer et de recevoir des messages sous réseau.

Pour pouvoir envoyer un message, nous devons d'abord affecter l'adresse *IP* hôte distante auquel nous souhaitons envoyer ce message puis l'envoyé en appelant la méthode *PostIt*.

Exemple

Dans notre exemple nous allons envoyer un message simple entre le maître et l'esclave.

Maître:

Initialisez IP de l'hôte

Insérez le code :

```
void __fastcall TForm1::Edit1KeyPress(TObject *Sender, char &Key)

{
  if (Key == 0x0D)
    {
      NMMsg1->FromName = Edit2->Text;

      NMMsg1->PostIt(Edit1->Text);
    }
}
void __fastcall TForm1::NMMsg1MessageSent(TObject *Sender)
{
  Memo2->Lines->Add("Message expédié");
}
```

Esclave:

Insérez le code:

```
void __fastcall TForm1::NMMSGServ1MSG(TComponent *Sender,
    const AnsiString sFrom, const AnsiString sMsg)
{
  Memo2->Lines->Add("Message arrivé de "+sFrom);
  Memo1->Lines->Add("["+sFrom+"] "+sMsg);
}
```

INSTALLATION

Notre application se divise en deux parties :

- La première partie, présentée dans la figure B.1, doit être installée sur la machine maîtresse.

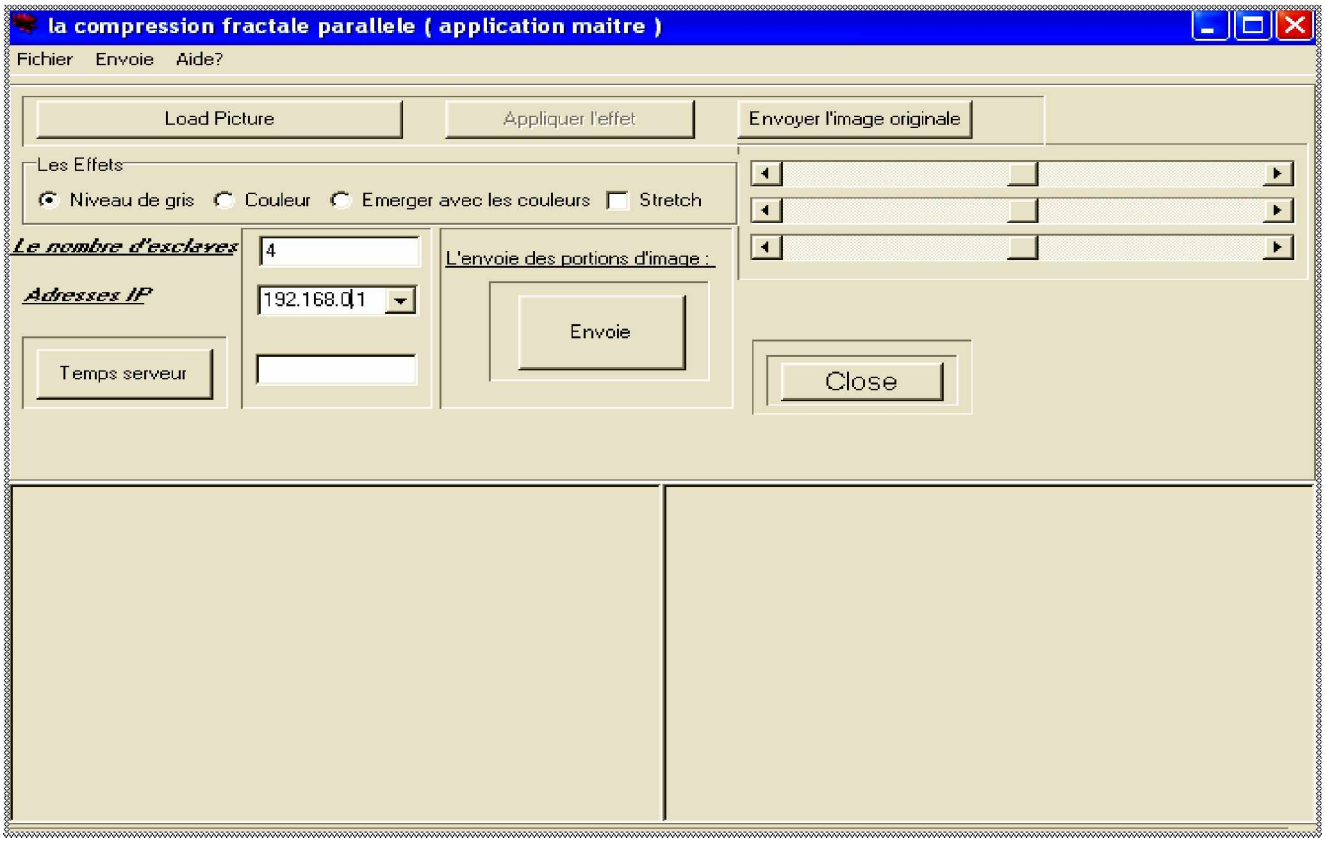


Figure B.1 : L'interface générale de la partie maître

Cette fenêtre regroupe toutes les fonctions réalisées dans le poste maître, on trouve les champs suivants : menu ***Fichier***, menu ***Envoie*** et un menu ***Aide.***

Le menu *Fichier* : comporte la fonction ***Ouvrir*** et la fonction ***Quitter***.

- La fonction ***Ouvrir*** : permet d'ouvrir des fichiers de type image. Cette fonction peut être invoquée par le raccourci ***Ctrl+O***, ou par le Bouton de commande ***"load picture"***. Un clic sur ces options fait apparaître une boite de dialogue (voir figure B.2) permettant de parcourir les répertoires et de sélectionner l'image voulue.

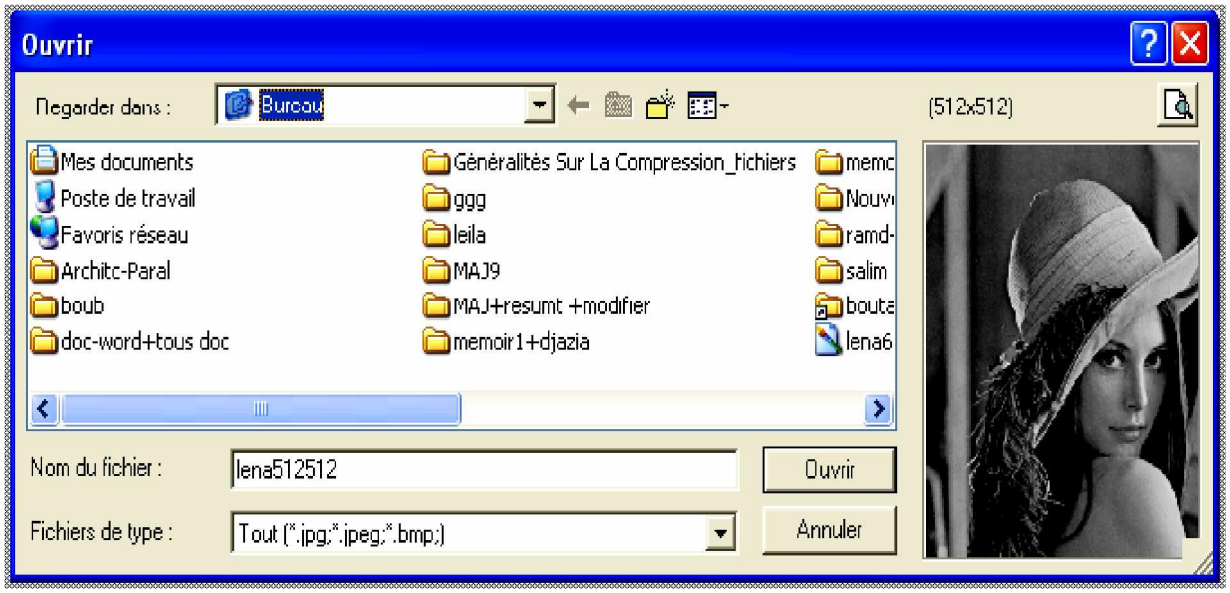


Figure B.2 : La boite de dialogue «*Ouvrir*»

- La fonction ***Quitter*** : Permet de quitter l'application à n'importe quel moment. Cette fonction peut être invoquée par le raccourci ***Ctrl + Q***, ou par le Bouton de commande ***"close"***.

Le menu *Envoie* : Comporte la fonction ***Découpage et Envoie*** et la fonction ***Envoie de l'image complète*** (*option facultative*).

- La fonction ***Découpage et Envoie*** : Permet de découper l'image sélectionnée en plusieurs portions suivant le nombre de clients présentés dans le champ « *le nombre d'esclaves* ». Chaque client a son adresse ***"IP"*** propre. cette fonction peut être invoquée par le raccourci ***F2*** ou par le Bouton de commande ***"envoie"***.

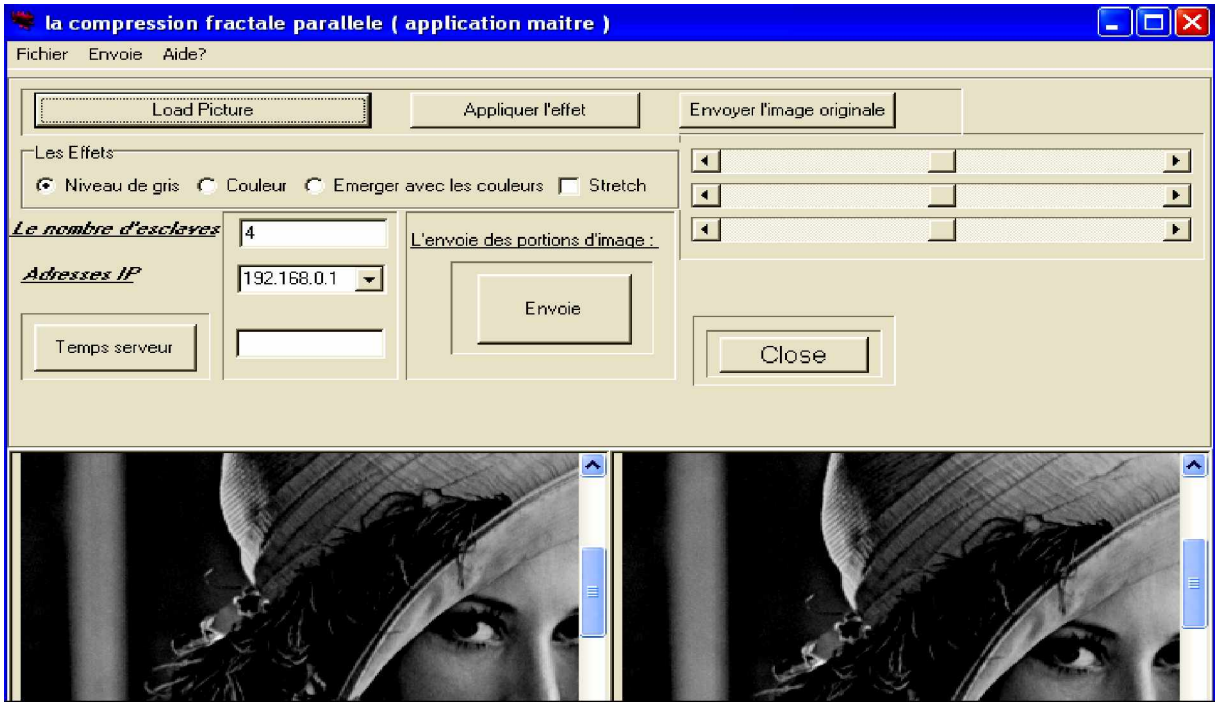


Figure B.3 : Découpage et Envoie de l'image

- Une copie de la deuxième partie doit être installée sur les autres machines (*esclaves*).

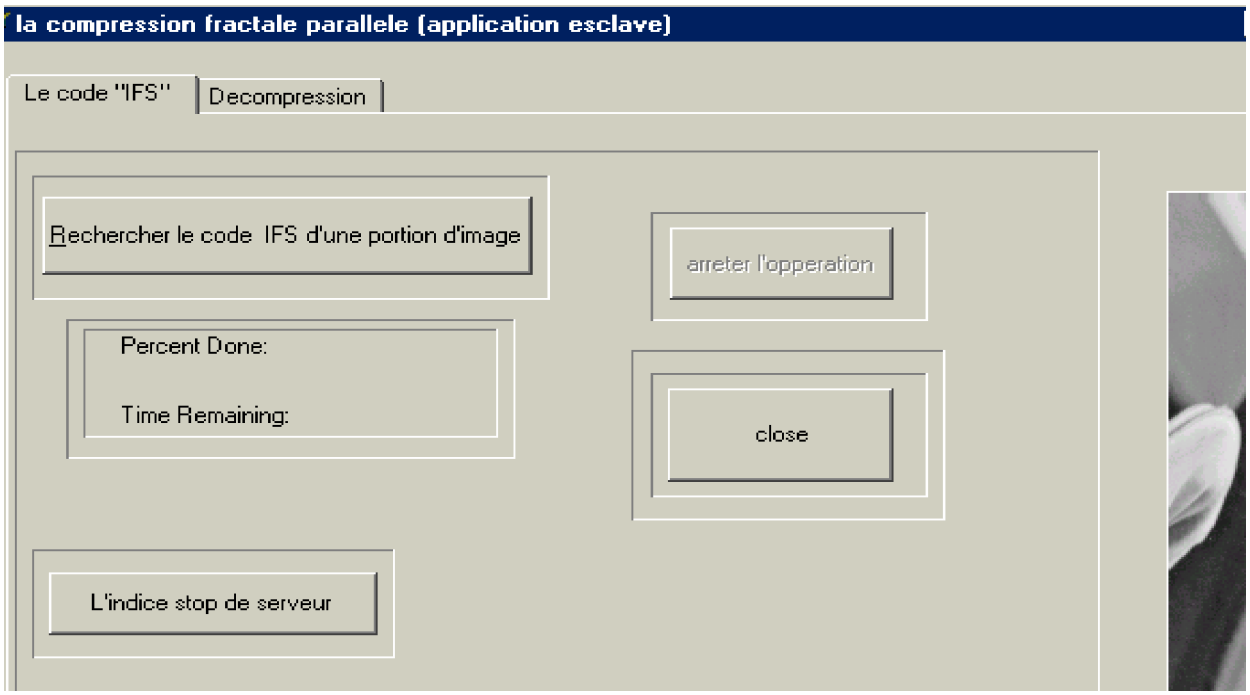


Figure B.4 : Rechercher le code IFS

Cette partie est constituée de deux fonctions. La fonction de ***Compression*** et la fonction de ***Décompression***.

La fenêtre de ***Compression*** est constituée de plusieurs opérations suivant ces boutons de commande :

- Rechercher le code IFS : permet de trouver le code *IFS* de la portion de l'image concernée.
- ***Arrêter l'opération*** : permet d'arrêter l'opération de codage.
- ***Envoyer un indice d'arrêt au serveur*** : pour calculer le temps écoulé par le serveur.
- ***Le temps client*** (time remaining): pour calculer le temps client.
- ***Close*** : permet de quitter l'application à n'importe quel moment.

RESUME

La compression d'images est une question importante en informatique multimédia, que ce soit pour les aspects transmission en réseaux ou pour les aspects stockage en bases de données. C'est toujours une question d'actualité, en recherche aussi bien qu'en développement industriel.

Ce mémoire traite de la compression des images fixes par fractales, fondée sur la théorie des systèmes de fonctions itérées (*IFS*). Après quelques rappels sur les principales méthodes de compression réversibles et irréversibles des images nous introduisons les notions nécessaires à la compréhension de la théorie des (*IFS*). La théorie des *IFS* (*Iterated Function Systems*) permet en effet de générer des images fractales, décrites par un ensemble de transformations affines contractantes.

L'inconvénient principal de cette technique est le temps très élevé pour déterminer les codes d'*IFS* de l'image compressée. Divers travaux de recherches ont été consacrés pour l'amélioration de la vitesse de compression tel que la classification des blocs et l'intégration du parallélisme dans l'algorithme de compression fractale.

C'est dans ce contexte que s'inscrit notre travail. Nous essayons dans ce mémoire d'accélérer la vitesse d'algorithme de compression .L'existence de la propriété de parallélisme dans l'algorithme de compression fractale nous a permis de proposer une nouvelle implémentation parallèle.

Les résultats que nous avons obtenus ont prouvé que l'intégration du parallélisme proposé dans l'algorithme permet d'accélérer la vitesse de codage d'une manière importante

Mots clés: Compression d'Images, Compression Fractale, *IFS,* Architectures *MIMD,* Algorithme Parallèle.

ABSTRACT

The compression of images is a significant question in multimedia data processing that it is for the transmission aspects in network or the storage aspects in data bases. It is always a topical question, in research as well as under development industrial.

This thesis deals with fractal compression of fixed images, based on the theory of iterated function systems (*IFS*). After an overview of the main lossy and lossless still image compression methods, we introduce the *IFS* theory. The theory of the *IFS* (*Iterated Function Systems*) indeed makes it possible to generate fractal images, described by contracting transformations.

The principal disadvantage of this technique is the very big time to determine codes of *IFS* of compressed image. Various works of researches have been devoted for the improvement of the speed of compression such that the classification of the blocks and the integration of parallelism in the algorithm of fractal compression.

It is in this context that enters our work. We try in this memory to accelerate the speed of algorithm of compression. The existence of the property of parallelism in the algorithm of fractal compression has allowed us to propose a new parallel implementation.

Results that we have obtained have proven that the integration of the proposed parallelism in the algorithm allows to accelerate considerably the speed of coding.

Key words: Image Compression, Fractal Compression, *IFS*, *MIMD* Architectures, Parallel Algorithm.