ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

MÉMOIRE PRÉSENTÉ À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DE LA
MAÎTRISE EN TÉCHNOLOGIE DES SYSTÈMES
M. ING

PAR
MONSEF MEKOUAR

# COMPRESSION D'IMAGES MÉDICALES PAR ONDELETTES ET RÉGIONS D'INTERET

MONTRÉAL, 12 JUIN 2001

CE MÉMOIRE A ÉTÉ ÉVALUÉ

PAR UN JURY COMPOSÉ DE :

Mme Rita Noumeir, directrice du mémoire
Département de génie électrique à l'École de technologie supérieure

M. Christian Gargour, codirecteur
Département de génie électrique à l'École de technologie supérieure

M. Chakib Tadj, président du jury
Département de génie électrique à l'École de technologie supérieure

M. Marcel Gabrea, professeur
Département de génie électrique à l'École de technologie supérieure

IL A FAIT L'OBJET D'UNE PRÉSENTATION DEVANT JURY ET UN PUBLIC

LE 28 MAI 2001

À L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

# COMPRESSION D'IMAGES MÉDICALES PAR ONDELETTES ET RÉGIONS D'INTERET

Monsef Mekouar

## Sommaire

Un bon nombre d'images contiennent des régions qui sont plus importantes que d'autres. Les méthodes de compression qui sont capables de reconstruire sans pertes ces régions sont donc d'un grand intérêt. Dans le cas des images médicales, il arrive souvent que seule une petite portion de l'image soit utile pour établir un diagnostic. Toutefois, le coût d'une mauvaise interprétation peut être très élevé. Les méthodes de compression sans pertes dans les régions d'intérêt et avec pertes partout ailleurs, semblent être un compromis intéressant. Nous présentons et comparons ici quelques méthodes pour la compression sans pertes des régions d'intérêt . Les méthodes proposées sont basées sur la compression par ondelettes, celles-ci sont appliquées selon deux approches différentes. L'une traite l'image au complet et l'autre est appliquée à l'image subdivisée en blocs égaux.

# LOSSELESS REGION OF INTEREST COMPRESSION OF MEDICAL IMAGES USING WAVELETS

Monsef Mekouar

Abstract

Some images contain regions which are more important than other. Compression methods which are capable of delivering higher reconstruction quality for the important parts are attractive in this situation. For medical images, only small portion of the image might be diagnostically useful, but the cost of a wrong interpretation is high. Methods which deliver lossless compression within the regions of interest, and lossy compression elsewhere in the image, might be the key to providing efficient and accurate image coding to the medical community. We propose and compare some new lossless region of interest compression methods. These methods are based on the Embedded zerotree wavelet coding ,that is used according to two different approaches. One applied to the whole image and the other applied to a subdivision of the image in equal blocs.

## REMERCIEMENTS

Je tiens à remercier vivement tout ceux qui ont participé de près ou de loin au bon déroulement de cette recherche. Bien sûr, je pourrais m' en tenir à ces quelques mots qui résument de manière concise ma pensée. Cependant, il y a trop de personnes que je ne voudrais en aucun cas omettre de citer tant leur importance dans l' achèvement de ce travail est grande.

A mes parents, Amina et Abderrazac, sans lesquels je n'aurai pas atteint ce degré : Pour tous les sacrifices que vous avez fait, je vous présente par ces quelques mots mon profond respect, et amour.

A mes sœurs et frères, Ghita, Wafaa, Hatim et Adil pour leurs encouragements.

A mes codirecteurs de recherche, Mme Rita NOUMEIR, Ph.d. et M. Christian GARGOUR, Ph.D., je ne saurais vous exprimer mes remerciements et ma profonde reconnaissance pour le soutien technique et moral qui m'ont été essentiels pour le bon déroulement de cette recherche.

A tout les membres du LIVIA que j'ai côtoyé durant ces deux années, qui m'ont aidé techniquement et qui n'ont cessé de m'encourager durant toute la période de ma recherche.

A tout mes amis de CASABLANCA et MONTREAL, je ne pourrai citer des noms vu qu'ils sont nombreux à m'avoir encouragé, merci beaucoup...

Monsef MEKOUAR

# TABLE DES MATIÈRES

Page

## LISTE DES TABLEAUX

Page

## LISTE DES FIGURES

Page

## LISTE DES ABRÉVIATIONS ET DES SIGLES

| | |
|---|---|
| ÉTS | École de Technologie Supérieure |
| IEEE | Institute of Electrical and Electronics Engineers |
| ISO | International Standard Organization |
| MSE | Mean Square Error |
| PSNR | Peak Signal to Noise Ratio |
| GIF | Graphic Interchange Format |
| JPEG | Joint Photographic Experts Group |
| DCT | Discrete Cosine Transform |
| FFT | Fast Fourier Transform |
| DWT | Discrete Wavelet Transform |
| LZW | Lempel Zip Welch |
| BPP | Bit Par Pixel |
| TD | Transformée discrète |
| VLC | Variable Length Coding |
| EZW | Embedded Zerotree Wavelet |
| BPP | Bit Par Pixel |
| MB | Méga Bits |
| MO | Méga Octets |
| ZL | Zerotree local |
| ZG | Zerotree Global |
| SPIHT | Set Partitioning In Hierarchical Trees |

# CHAPITRE 1

# INTRODUCTION

## 1.1 Nécessité de la compression de données

Au début de "l' ère informatique", les ordinateurs disposaient de très peu de mémoire, aussi bien vive que morte, avec des disques durs de très faible capacité. Le problème de stockage des données s'est donc très vite posé et rapidement on a recherché des moyens de diminuer la quantité de ces données.

Aujourd'hui, les équipements informatiques sont largement plus performants qu'avant, un ordinateur de moyenne gamme dispose de 64 Méga-Octets (MO) de mémoire vive, 10 GigaOctets de disque dur et un processeur d'au moins 600 MHZ de fréquence. Cependant, malgré cette ascension de la technologie des ordinateurs, les données traitées demandent de plus en plus d'espace, on peut citer à titre d'exemple les fichiers images ou vidéo dont la qualité de numérisation ne cesse d'augmenter entraînant des quantités d'information de plus en plus importantes. Ceci dit, la compression de données s'impose quelque soit l'évolution du matériel de traitement.

Le gain en espace de stockage obtenu à travers la compression n'est pas avantageux juste pour notre disque dur, mais aussi au niveau de la transmission par réseau. En effet, moins les données occupent d'espace, plus la transmission est rapide. Cet avantage est aussi important que le premier cité vu le volume des données multimédias (image, vidéo, son) traités à travers le réseau internet.

A titre d'exemple, une image classique à niveaux de gris de taille 512x512 pixels codée sur 8 bits par pixel nécessite : 512 x 512 x 8 bits soit 2.097 Mbits . Dans le cas d' une image en couleur, le nombre de bits requis pour le stockage est de 512 x 512 x 8 x 3 (6.91 Mbits). En multipliant ce résultat par 25 images/seconde (pour les fichiers vidéo), on peut imaginer le temps que prendra une séquence d'une seconde pour être téléchargée par Internet via un modem classique.

## 1.2 Principales techniques de compression d'images

Les techniques de compression d'images sont multiples, que ce soit pour la compression avec ou sans pertes. Une méthode de compression d'image est généralement composée d'une transformation, suivie d'une quantification et d'un codage comme le montre la figure 1.1.

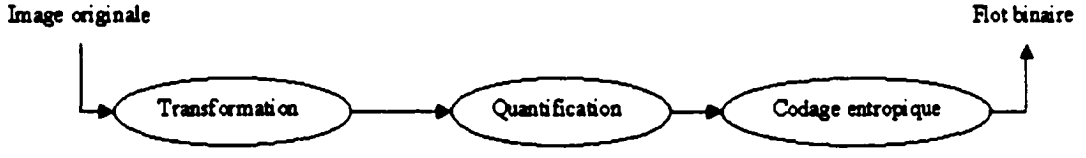


Figure 1.1 Étapes principales de compression d'images

Le but de la transformation est de décorréler les pixels, ce qui a pour effet en général de redistribuer l'énergie de l'image dans un nombre restreint des coefficients transformés. De cette façon, un grand nombre de coefficients transformés ont des très faibles valeurs, et peuvent être supprimés ou se voir allouer un nombre très faible de bits lors de l'étape suivante de quantification. Parmi les transformations utilisées en compression d'images, on distingue la transformée en cosinus discrète (DCT), celle-ci était utilisée dans le format standard de compression JPEG [16] avant que la transformée en ondelette ne soit adoptée pour le JPEG 2000 pour ses performances au niveau de la

représentation des images, La nouvelle génération de compression de données en général s'appuie sur la transformée en ondelette .

À chaque transformée, on peut adapter une méthode de quantification, cette dernière peut être classée en deux grandes catégories : quantification scalaire qui traite les coefficients indépendamment, et le quantification vectorielle qui agit sur des blocs de coefficients. L'étape de codage consiste à donner la représentation binaire la plus compacte de l'entité quantifiée, plusieurs méthodes sont utilisées dans ce but, notamment le codage de Huffman et le codage arithmétique

## 1.3 Présentation, justification et motivation du projet

La nécessité d'avoir un grand support de stockage et un médium de transmission ultra rapide est un besoin constant qui accompagne la médecine moderne depuis que ce domaine et les domaines qui lui sont connexes se sont informatisés. Ce besoin se fait surtout sentir dans la numérisation des radiographies. Par exemple, une mammographie analogique doit être numérisée à 4096/4096 *16 bpp, ce fichier tout seul requiert environ 33 MO (Méga Octets). Si l'on considère une clinique radiologique qui traite une centaine de cas par jour, l'espace disque quotidien requis se calculera en giga-octets. Il faut ajouter le temps que prendra cette même image pour être chargée si on veut la consulter ultérieurement et ce que coûtera une transmission que ce soit par Internet ou même par réseau interne.

Même avec le développement des supports de stockage, de transmission, et la puissance des microprocesseurs actuels, ces images restent 'lourdes'. La seule issue pour les traiter demeure la compression, et comme généralement les informations contenues dans ces images sont importantes, on ne se permet pas plus d'une compression sans pertes. Sur ce point, plusieurs questions se posent : quel est l'ordre de précision que peut

demander une analyse d'images médicales ? Et est ce que toute l'image est indispensable pour un bon diagnostic?

On peut répondre à ces deux questions par un simple exemple : considérons un patient qui a une fissure dans une côte. Généralement une radiographie prendra toutes les côtes, mais en faisant des agrandissements, on atteindra la partie touchée. Comme première remarque, on se rend compte que la partie intéressante se situe dans une portion qui ne dépasse pas 10 ou 15% de la surface de l'image, le reste est quasiment sans importance sinon pour indiquer l'emplacement de la côte en question ou bien pour permettre de distinguer que la partie atteinte est bien une côte et non un genou. Pour ce qui est de la précision, une fissure se mesure en centimètres voire en millimètres, et son évolution nécessite plus de précision (centièmes de mm ou microns même). On ne peut donc pas se permettre de perdre de l'information dans cette partie de l'image, qu'on peut appeler région intérêt Pour cela, une compression qui agira sans pertes d'informations dans la région touchée et avec pertes dans la partie restante, s'avère une solution permettant de gagner de l'espace de stockage et du temps de traitement tout en donnant une représentation précise de la zone atteinte.

A partir de ces remarques et la recherche effectuée sur les algorithmes de compression d'images par ondelette, on propose ici des méthodes qui réalisent la compression d'images par région d'intérêt. Ces méthodes utilisent les mêmes algorithmes de codage et décodage adaptés selon le contexte d'utilisation (région importante ou non). La quantification se base sur la méthode d'arbres de zéros (EZW) [17], et le codage entropique utilisé est le codage arithmétique.

## 1.4 Organisation de ce mémoire

Le chapitre 2 consiste en un tour d'horizon portant sur les principales méthodes de compression d'images. On y retrouve la compression avec et sans pertes, différentes

implémentations et algorithmes proposés pour l'élimination des redondances , les récents développement dans le domaine, et à la fin, une introduction à la compression par ondelettes incluant des notions de base et les algorithmes utilisant ce genre de transformée qui sera utilisée dans ce travail.

Dans le chapitre 3, on fera plus le point sur une méthode de compression par ondelette, nommée 'Embedded Zerotree of Wavelets' (EZW), une description détaillée de cet algorithme, et de son implémentation seront présentées. Des exemples montrant son efficacité y seront proposés et enfin, quelques méthodes qui sont basées sur cet algorithme pour effectuer de la compression avec ou sans pertes seront exposées.

Dans le chapitre 4, le décodage arithmétique sera couvert avec une implémentation de la méthode. Les avantages que peut apporter cet algorithme par rapport à ses semblables et son utilisation dans les codecs de compression existants seront mis en évidence ainsi que différentes améliorations proposées dans la littérature spécialisée .

Dans le chapitre 5, nous commencerons d'abord par une description des différents blocs de compression allant de la transformée en ondelette au codage arithmétique en passant par une quantification par approximations. La justification du choix de chaque bloc sera incluse dans cette première partie. La deuxième partie sera consacrée à la proposition des algorithmes de compression par région d'intérêt ainsi que différentes variantes possibles de ces méthodes. Par la suite, les résultats de chacune des méthodes proposées seront exposés suivis d'une discussion et de comparaisons entre ces techniques.

# CHAPITRE 2

# METHODES DE COMPRESSION D'IMAGES

## 2.1 Introduction

On a souvent besoin de compresser des données. On compresse aussi bien des documents pour qu'ils puissent être contenus dans une disquette, des archives pour qu'elles occupent moins d'espace, des données pour les transmettre plus rapidement, etc... Dans certains cas, notamment pour les images, il arrive qu'un document compressé requiert beaucoup moins d'espace que l'original. Il existe plusieurs méthodes de compression, chacune a ses avantages et ses inconvénients. Selon la situation, certaines méthodes sont meilleures que d'autres. Certaines méthodes diminuent la qualité des données, c'est souvent le cas pour les fichiers multimédia, tels que les images, ou la vidéo. A titre d'exemple, le format JPEG diminue de la qualité au profit de la compression.

La compression d'un signal (1-D, 2-D, 3-D) consiste à diminuer la quantité ou la redondance de l'information qui la représente, ceci en vu d'un stockage, d'une transmission ou simplement pour l'accélération des traitements ultérieurs. En traitement d'images, les applications sont nombreuses et en développement rapide. Citons par exemple: les images satellite, la vidéo-conférence, la transmission par fac-similés de documents, les images télévision, la transmission par internet. Ceci n'est certes possible que sur des images numériques qui peuvent être obtenues par la numérisation des images

analogiques à l'aide d'un numériseur ou tout autre instrument approprié, qui, de nos jours, atteignent une excellente précision .

Une image classique à niveaux de gris de taille 512x512 pixels codée sur 8 bits par pixel nécessite : 512 x 512 x 8 bits soit 2.097 Mbits . Dans le cas d' une image en couleur, le nombre de bits requis pour le stockage est de 512 x 512 x 8 x 3 (6.91 Mbits). En dehors du problème de stockage, si les données images sont transmises en utilisant un modem classique, la lenteur de la transmission peut être inacceptable pour certaines applications en temps réel.

Toutes les techniques de compression d'images peuvent être classées en deux grands groupes en fonction de la qualité de l'image reconstituée. On a d'abord le groupe des techniques de compression sans pertes d'information dites aussi réversibles ou à reconstruction exacte. Ces techniques sont idéales et conduisent à des taux de compression moyens. L'autre groupe avec perte d'information, englobe des techniques dites irréversibles, et permettent d'obtenir des images reconstruites de moyenne qualité. En réalité, le critère d'appréciation de la bonne qualité de l'image reconstruite est surtout visuel et par conséquent subjectif. Cependant les techniques de reconstruction exacte conduisent à des taux de compression moins important. Il existe des domaines, la médecine par exemple où les techniques réversibles sont presque obligatoires, ceci est du au risque important qu'entraînerait un mauvais diagnostic causé par une image de mauvaise qualité obtenue à la reconstruction.

## 2.2 Généralités sur la compression

La compression de données a pour but de réduire l'espace requis pour représenter une certaine quantité d'informations. Généralement, le fichier traité contient des

informations qui ne sont pas 'essentielles', la redondance de données est ce champ de la compression.

On peut distinguer plusieurs types de redondances de données, les plus exploitées sont les redondances de codage, spatiales et psychovisuelles [1].

Un codage typique utilise généralement le même nombre de bits pour représenter un certain niveau de gris, mais, comme les niveaux de gris ne sont pas tous distribués uniformément dans la représentation d'image, le codage à longueur variable (VLC : Variable Length Coding) donne à l'image une représentation plus compacte sans pertes d'informations. Parmi les méthodes de codage, on peut citer celui de Huffman et celui de Shannon qui visent à diminuer voire éliminer la redondance de codage.

La redondance spatiale est bien exploitée pour des images qui ne contiennent pas des passages brusques de niveaux de gris, les méthodes qui exploitent ce type de redondance s'appuient sur le fait que les pixels adjacents représentants une image sont souvent liés soit par un même niveau de gris ou bien des valeurs voisines.

La redondance psychovisuelle est exploitée par la compression en éliminant des données sans pour autant diminuer la qualité de l'image auprès de l'œil humain. En fait, la perception humaine de l'information dans une image ne permet pas de distinguer la luminance de tous les pixels d'où la possibilité d'éliminer quelques détails pour les méthode de compression avec pertes d'informations .

Étant donné que l'objectif d'une compression est de minimiser la quantité d'information nécessaire à la représentation d'une image, on définit la quantité T comme suit:

$$T = \frac{Nombre\ de\ bits\ de\ l'image\ originale}{Nombre\ de\ bits\ de\ l'image\ compressée} \tag{2.1}$$

cette quantité est le taux de compression. Il existe une autre quantité permettant de définir le taux de compression maximal ceci sans perte d'information, c'est l'entropie H, définie ci-dessous.

$$H = -\sum_{i=1}^{N} p_i \log_2 p_i \tag{2.2}$$

où $p_i$ est la probabilité de présence de niveau de gris i dans l'image qui est supposée être quantifiée sur N niveaux. En fait, il s'agit ici de l'entropie d'ordre zéro car cette formule ne tient pas compte d'une relation éventuelle entre des pixels voisins. On peut définir des entropies d'ordre supérieur.

L'importance de l'entropie se retrouve aussi dans le théorème de codage de Shannon, qui précise qu'une source image S, ayant une entropie H(S) peut être codée sans perte d'information avec des pixels ayant un nombre L de bits si $H(S) \leq L \leq H(S)+\varepsilon$ où $\varepsilon$ est une quantité relativement faible. On peut dans le cas du codage sans perte d'information, définir le taux de compression maximal par $T_{max} = b/L$ pour une image codée sur b bits. Ceci ne peut éventuellement être réalisé qu'avec des techniques sans perte d'information. Avec les autres techniques, on a $T > T_{max}$, avec d'autant plus de perte que T est grand. Cette perte qui se traduit par la dégradation de l'aspect visuel de l'image peut se quantifier par des mesures de fidélité de la qualité de la reconstruction. Il s'agit de définir des quantités permettant d'évaluer numériquement la qualité de l'image reconstruite. Si l'on note $f_0$ l'image originale de taille MxN, et $f_1$ l'image de même taille obtenue après reconstruction, on peut définir : L'erreur quadratique moyenne (MSE : Mean Square Error) et le rapport signal maximal sur bruit (PSNR : Peak Signal to Noise Ratio, pour une image représentée sur 256 niveaux de gris variant de 0 à 255.

$$MSE = \frac{1}{MN}\sum_{i=1}^{N}\sum_{j=1}^{M}(f_0(i,j) - f(i,j))^2 \quad (2.3)$$

$$PSNR = 10\log(\frac{(255)^2}{MSE}) \quad (2.4)$$

## 2.3 Compression sans pertes d'informations

Les méthodes de compression sans pertes d'informations permettent de retrouver exactement toute l'information contenue dans l'image numérique originale. Il existe plusieurs méthodes de compression sans pertes, basées sur le codage ou prédiction nous présentons ici celles que l'on retrouve souvent dans la littérature [1] [2][3] et qui sont basées sur le codage.

### 2.3.1 Codage de Shannon-Fano

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

Le procédé de Shannon-Fano [4] consiste à construire une arborescence partant de la racine, et procédant par divisions successives. Le classement des fréquences se fait par ordre décroissant, ce qui suppose une première lecture du fichier et la sauvegarde de l'en-tête .

Le principe est le suivant :

1. Classer les n fréquences non nulles par ordre décroissant.

2. Répartir la table des fréquences en deux sous tables de fréquences cumulatives égales ou proches. Poursuivre l'arborescence jusqu'à ce que tous les symboles soient isolés.
3. Attribuer dans l'arborescence le bit 0 à chaque fils droit et 1 à chaque fils gauche.
4. Attribuer aux symboles les codes binaires correspondant aux bits de description 1 de l'arborescence.

L'exemple suivant illustre cette méthode, on désire coder la chaîne de caractères 'ECOLE DE TECHNOLOGIE SUPERIEURE'. Le tableau suivant classe les symboles (caractères dans ce cas) par ordre de fréquence décroissant et contient aussi le code attribué suite au codage de Shannon. La figure 2.1 décrit l'arborescence qui a permis la distribution des codes (pour plus de clarté, le caractère 'espace' est remplacé par '/').

Tableau I

Exemple du codage de Shannon

| Symbole | Fréquence | Code attribué |
|---|---|---|
| E | 8 | 00 |
| O | 3 | 010 |
| / | 3 | 0110 |
| C | 2 | 0111 |
| L | 2 | 1000 |
| I | 2 | 1001 |
| U | 2 | 1010 |
| R | 2 | 1011 |
| D | 1 | 11000 |
| T | 1 | 11001 |
| H | 1 | 11010 |
| N | 1 | 11011 |
| G | 1 | 11100 |
| S | 1 | 11101 |
| P | 1 | 1111 |

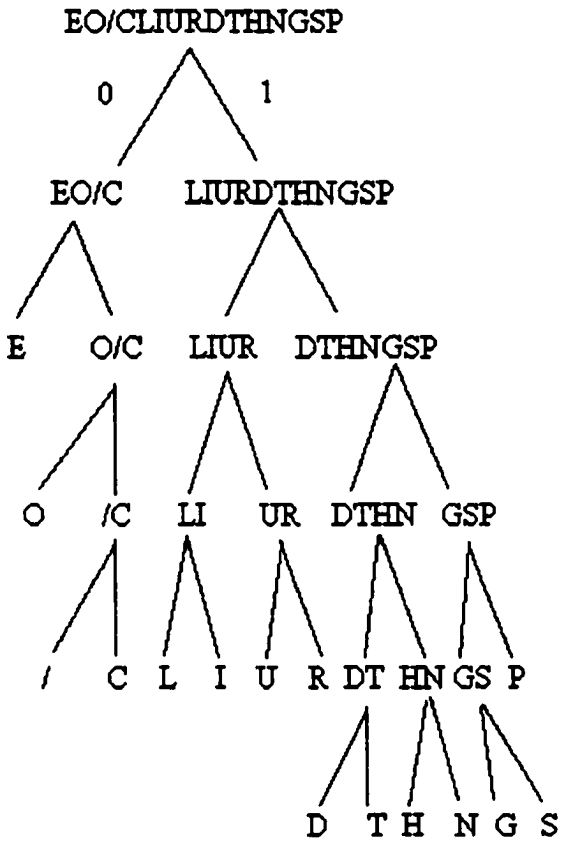


figure 2.1 Arbre d'attribution des codes de Shannon

### 2.3.2 Codage Lempel Zif Welsh (LZW)

Le codage LZW [4] est une technique de compression réversible qui peut être appliquée à tout type de fichier de données, que ce soit : texte, image, fichier informatique etc.. Elle a été adoptée pour la mise en œuvre du format de compression d'images 'GIF'.

Le principe géneral de cette méthode consiste à créer un dictionnaire contenant toutes les répétitions . Il doit être construit de la même manière, à la compression et à la décompression, et contenir les mêmes informations.

Tous les ensembles de lettres qui sont lus sont placés dans le dictionnaire, et sont numérotés. A chaque fois qu'un ensemble est lu, on regarde s'il en existe déjà un qui est identique. Si c'est le cas, on émet son numéro vers le fichier compressé. Sinon, on le rajoute à la fin du dictionnaire, et on écrit chacune des lettres dans le fichier compressé.

Quand on écrit un numéro au lieu d'écrire des lettres, il y a un gain de place. Mais pour cela, il faut déjà avoir beaucoup de chaînes dans le dictionnaire. L'apprentissage est donc nécessaire pour que la méthode doit efficace.

Les caractéristiques essentielles de LZW sont :

- Il n'existe pas de table d'en-tête : le dictionnaire est construit au fur et à mesure de la lecture du fichier tant à la compression qu'à la décompression.

- L'algorithme ne fonctionne pas sur un nombre fixe de motifs mais apprend les motifs du fichier durant la lecture du fichier à comprimer.

- Le compression se fait en une seule lecture.

### 2.3.3 Codage de Huffman

Le codage de Huffman crée des codes à longueurs variables sur un nombre entier de bits. L'algorithme considère chaque message à coder comme étant une feuille d'un arbre qui reste à construire. L'idée est d'attribuer aux deux messages

de plus faibles probabilités, les mots codés les plus longs. Ces deux mots codés ne se différencient que par leur dernier bit. Contrairement au codage de Shannon-Fano qui part de la racine des feuilles de l'arbre et, par fusions successives, remonte vers la racine.

Le principe est le suivant :

1. Répartir les fréquences f(i) des symboles .
2. Classer les symboles dans l'ordre décroissant des fréquences d'occurrence. Le résultat de l'algorithme ne change donc pas si l'on remplace les fréquences f(i) par les probabilités .
3. Regrouper par séquences les paires de symboles de plus faible probabilité, en les reclassant si nécessaire. Plus précisément : calculer s=f(n) + f(n-1), la somme des deux plus faibles fréquences.
4. Choisir le plus petit indice k tel que s soit supérieur ou égal à f(k), remplacer k par k+1.
5. Recomposer la table des fréquences en plaçant à la k ème position la valeur s et en décalant les autres d'une position vers le bas. Puis décrémenter n d'une unité, poursuivre jusqu'à ce que la table des fréquences ne comporte plus que deux éléments.
6. Coder avec retour arrière depuis le dernier groupe, en ajoutant un 0 ou un 1 pour différencier les symboles préalablement regroupés.

Pour illustrer la méthode, on va coder la chaîne de caractères 'SUPERIEURE'

Tableau II

Exemple du codage de Huffman

| Symbole | Code | Fréquence | Étapes de codage |
|---|---|---|---|
| E | 00 | 3 | 3 3 4 6 |
| U | 10 | 2 | 2 3 3 4 |
| R | 11 | 2 | 2 2 3 |
| S | 011 | 1 | 2 2 |
| P | 0100 | 1 | 1 |
| I | 0101 | 1 | |

## 2.4 Compression avec pertes d'informations

Contrairement aux méthodes de compressions sans pertes d'informations, ce type de compression comporte une perte de données pendant le processus. Le résultat qu'on peut en obtenir est une version dégradée de l'image originale. Le but de ce type de compression est d'éliminer le plus d'information possible sans atténuer la qualité de l'image perçue par système visuel humain. On peut distinguer deux types de compression pouvant générer des pertes: le codage prédictif et le codage par transformée, ce dernier est de loin le plus utilisé dans la compression d'image.

Un codeur avec pertes idéal [1] est composé d'un module de transformée, un module de quantification et un codeur entropique (figure 2.2)

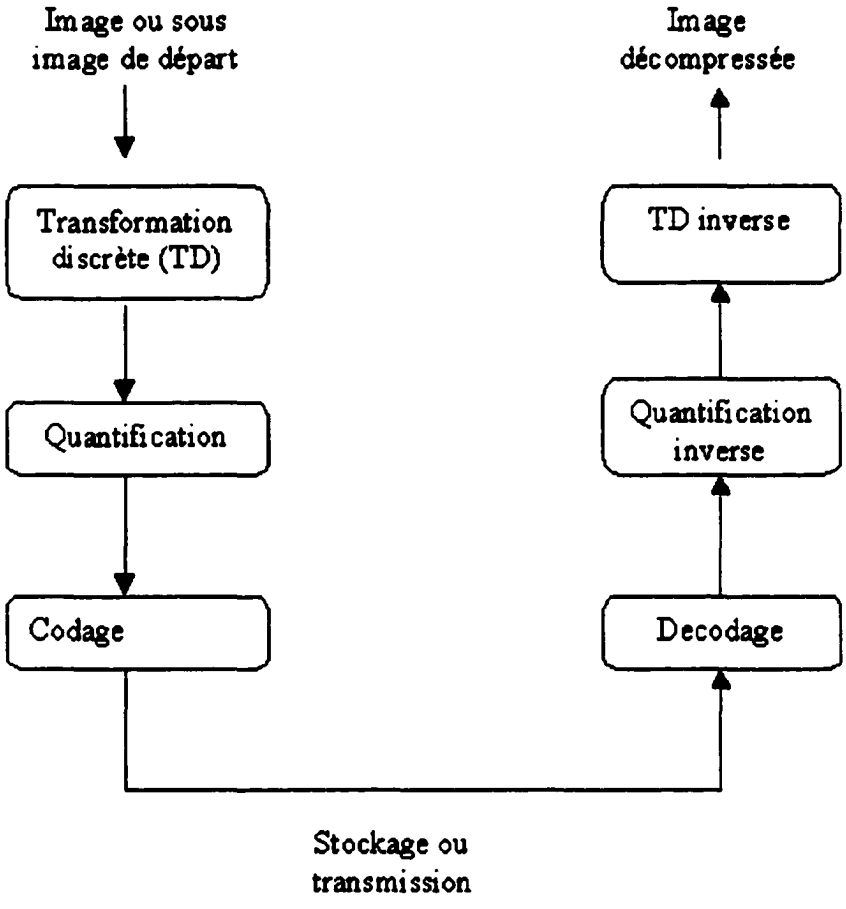


Figure 2.2 Schéma d'un codeur par transformée avec pertes d'informations

### 2.4.1 Transformée discrète

Les méthodes de compression par transformation n'agissent pas directement sur l'image numérique dans sa représentation canonique, mais sur le domaine de sa transformée. Cette transformation peut être linéaire ou non. Il est bien connu qu'une transformation peut permettre de mettre en évidence certaines propriétés de l'image que la représentation originale ou canonique ne laisse pas apparaître. En partant d'un ensemble de valeurs numériques corrélées d'une image, le but est d'obtenir un autre ensemble de valeurs le moins corrélées possible dans

l'espace transformé. En général, les schémas de codage par transformation subdivisent l'image de taille NxN en sous-images de taille plus petites avant de faire subir à chacune de ces sous images une transformation. On privilégie les transformations qui sont unitaires et qui conservent l'énergie. La transformation consiste en la décomposition de l'image selon une base adéquate de fonctions telles que :

a- les coefficients de la transformation soient indépendants

b- qu'un nombre minimum de ces coefficients contienne une proportion importante de l'énergie de l'image.

Ainsi, on pourra mettre à zéro certains d'entre eux sans nuire de manière significative ni à la quantité d'énergie, ni à l'aspect visuel de l'image reconstruite.

On peut distinguer plusieurs transformées dont [5] :

- Transformation de Karhunen-Loeve ( TKL ).
- Transformation de Fourrier discrète ( TFD ).
- Transformation de Hadamard ( TH ).
- Transformation de Haar ( THA ).
- Transformation en cosinus discrète (DCT).
- Transformation par ondelette discrète (DWT)

La transformation de Fourier et celles qui en découlent, telles la transformation en sinus, la transformation en cosinus, sont très utilisées en analyse et en filtrage du signal [6]. Ces transformations peuvent etre mises en œuvre à l'aide d'algorithmes rapides comme la FFT et ses variantes. La variable de l'espace transformé étant la fréquence, une telle décomposition permet de mieux observer la répartition fréquentielle de l'image. Étant donné que ce sont les

premières harmoniques qui contiennent la quasi-totalité de l'énergie, il est donc possible de mettre à zéro une proportion importante des coefficients et de coder l'image à moindre coût. Malgré la rapidité de la transformation de Fourier, elle décompose l'image en une partie réelle et une partie imaginaire pouvant se convertir en module et argument ce qui n'est pas facile à manipuler ou à interpréter. Les traitements de ces données peuvent s'avérer lourds, d'où la préférence accordée à la transformation en cosinus qui bénéficie de toutes les caractéristiques de la FFT. La transformée en cosinus a été choisie comme standard par JPEG (Joint Photographic Experts Group) pour le codage d'images fixes.

Actuellement, la transformée qui est sujette à multitudes recherches et qui a été choisie comme standard du JPEG2000 est la transformée en ondelette. Cette transformée a révolutionné plusieurs domaines de l'imagerie notamment la compression d'images, où elle permet un meilleur taux de compression que les autres et ceci avec une meilleure qualité d'images.

### 2.4.2 Quantification

La quantification est un processus qui permet d'associer à un nombre réel (resp. vecteur de réels) un nombre entier (resp. vecteur d'entiers). Dans un certain sens on peut considérer qu'elle réalise une compression implicite (passage des réels aux entiers) en réduisant le nombre de bits nécessaires à la représentation de l'information image. On distingue en général deux types de quantification: la quantification scalaire et la quantification vectorielle [2].

### 2.4.2.1 Quantification scalaire

Soit u une variable aléatoire de densité de probabilité p(x). Une quantification est alors une fonction Q définie sur R et à valeurs dans Z qui à u associe ũ. En général, l'image de Q est un ensemble fini $\{q_j ; j=1...L\}$ où L est le nombre de niveaux de quantification (on dit aussi de reconstruction). On choisit souvent pour Q, une fonction en escalier (Figure (2.3)).

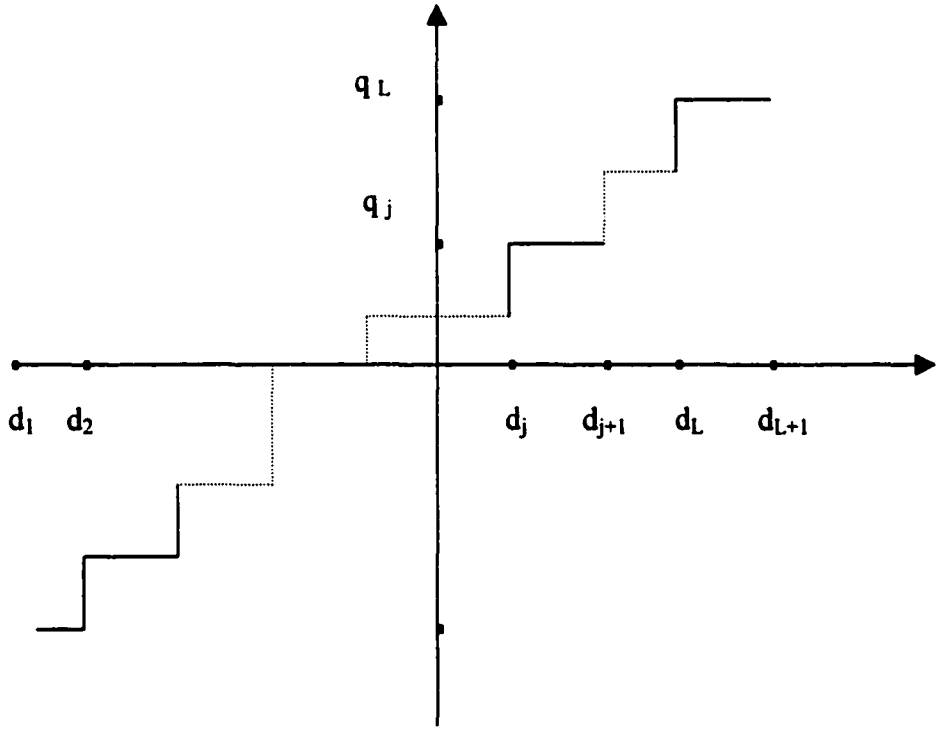


Figure 2.3: Exemple de fonction de quantification: fonction en escalier.

La règle de quantification est alors la suivante : on définit $\{d_j;j=1...L+1\}$, un ensemble croissant de niveaux de transitions ou de décisions et l'application Q est alors définie par $Q((d_j ; d_j+1]) = q_j , j = 1..L$, ceci revient à subdiviser l'ensemble des valeurs de u en L intervalles (dj ;

dj+1] et à associer à chaque u appartenant à l'intervalle (dj ; dj+1], la valeur $q_j$. L'objet de la quantification scalaire revient à déterminer les niveaux de transition $d_j$ et de quantification $q_j$ optimums, connaissant la densité de probabilité et se fixant un critère d'optimisation. De toute évidence, l'application ainsi définie introduit par conséquent une distorsion entre les valeurs de u et Q(u) qu'il faut rendre minimale. Si l'on prend pour critère d'optimisation, la minimisation de l'erreur quadratique moyenne, pour un nombre donné de niveaux de transition, on obtient un quantificateur optimal au sens des moindres carrés. C'est le quantificateur de Lloyd-Max [5].

En désignant par $d_j$; (j = 1: L+1), (resp. $q_j$; (j = 1: L)) les niveaux de transition (resp. de quantification) et par p(x) la fonction de densité de probabilité, l'erreur quadratique se définit par :

$$Erreur = \sum_{j=1}^{L} \int_{d_j}^{d_{j+1}} (x - Q(x))^2 \, p(x) dx \tag{2.5}$$

La minimisation de cette fonction revient à trouver les quantités $d_j$ et $q_j$ solutions du système d'équations suivant :

$$d_j = \frac{1}{2}(q_j + q_{j-1}) \tag{2.6}$$

$$q_j = \frac{\int_{d_j}^{d_{j+1}} x p(x) d(x)}{\int_{d_j}^{d_{j+1}} p(x) d(x)} \tag{2.7}$$

Notons que ces 2L-1 équations à 2L-1 inconnues se déduisent par annulation des dérivées obtenues par différenciation de (2.5) par rapport à

$d_j$ d'une part et $q_j$ d'autre part. Ces équations sont non linéaires et par conséquent nécessitent la mise en œuvre d'algorithmes spécifiques à des systèmes non linéaires. Elles montrent que les niveaux de décision sont les centres des intervalles délimités par les niveaux de reconstruction et que les niveaux de quantification sont les moyennes normalisées dans chaque intervalle de décision.

On modélise le plus souvent la distribution de densité des niveaux de gris par une gaussienne ou une laplacienne. Ces deux densités de probabilité sont des cas particuliers de la gaussienne généralisée dont la formule est donnée par :

$$p(x) = \alpha * exp(-|\beta(x)|^r) \tag{2.8}$$

$$\text{avec} \quad \alpha = \frac{\beta r}{2\Gamma(\frac{1}{r})} \quad \text{et} \quad \beta = \frac{1}{\sigma}\left[\frac{\Gamma(\frac{3}{r})}{\Gamma\frac{1}{r}}\right]^{\frac{1}{2}}$$

où $\sigma^2$ est la variance de l'image et $\Gamma(r)$, la fonction Gamma. Notons que pour r = 1, on a la densité de probabilité laplacienne et celle de la gaussienne pour r = 2.

Si l'on suppose que la densité de probabilité est uniforme, les niveaux de décision $d_j$ et de quantification $q_j$ du quantificateur optimal sont alors définis par :

$$d_j = d_1 + (j-1)p \tag{2.9}$$

$$q_j = d_j + \frac{p}{2} \tag{2.10}$$

$$p = (d_{L+1} - d_1)/L \qquad (2.11)$$

Ce quantificateur est souvent appelé quantificateur linéaire. Au lieu de la fonction densité de probabilité, on peut utiliser l'histogramme de l'image à quantifier. En général, les lois ne sont pas connues ou sont complexes à évaluer, ce qui complique le calcul de $d_j$ et $q_j$. Pour certaines lois classiques (loi normale, loi de laplace ...), ces valeurs sont données par des tables, ce qui facilite la mise en oeuvre de la quantification quand on utilise ces modèles.

2.4.2.2 Quantification vectorielle

Le principe général du quantificateur vectoriel consiste à quantifier une suite d'échantillons (vecteur) d'une image au lieu de quantifier chaque valeur comme c'est le cas pour la quantification scalaire. Pour réaliser ceci, on commence par créer un catalogue de vecteurs $\tilde{U}_i$; $i = 1..N_c$ de référence appelé codebook ou dictionnaire à partir d'un ensemble d'images tests par un algorithme d'apprentissage.

L'image à quantifier est décomposée en un ensemble de vecteurs $U_i$, de même taille que ceux du codebook. Ensuite, utilisant une métrique d; chaque vecteur $U_i$ de l'image à quantifier est comparé, à tous les vecteurs $\tilde{U}$ de la séquence d'apprentissage générée précédemment. Il s'agit de choisir dans le codebook le vecteur $\tilde{U}_k$ tel que $d(U_k ; \tilde{U}_k;) \geq d(U_i ; \tilde{U}_i)$ ; $i=1..N_c$. La distorsion mesurée par d doit être minimale et permettre d'obtenir une image de bonne qualité visuelle. La métrique communément utilisée pour ce type de quantification est la métrique euclidienne qui correspond à :

$$d^2(U,\tilde{U}) = \sum_{i=1}^{N} (U(i) - \tilde{U}(i))^2 \qquad (2.12)$$

On peut envisager, pour mieux tenir compte de la valeur de chaque pixel, une pondération de cette métrique :

$$d_\omega^2(U,\tilde{U}) = \sum_{i=1}^{N} \omega_i (U(i) - \tilde{U}(i))^2 \qquad (2.13)$$

où les $w_i$ sont des poids pouvant être identiques. Après avoir sélectionné parmi les vecteurs du codebook le vecteur le plus proche (au sens de la métrique choisie) du vecteur à quantifier, ce dernier est représenté par son indice qui peut alors être transmis ou stocké en vue de la reconstruction de l'image originale. En fait si l'on a une image de taille N x N = 512 x 512, on peut la décomposer en 512 vecteurs de taille 512 chacun. Son indice est alors codé sur $\log 2^N$ bits c.à.d 9 bits. On se rend ainsi compte que si au lieu de manipuler (stocker ou transmettre) l'image toute entière on ne travaille que sur les indices, on réalise un gain important au niveau de la quantité d'information à traiter. A la réception d'un processus de transmission, l'image originale est reconstruite en utilisant simplement une table de correspondance. Il est à noter que soit le codebook est transmis en même temps que les index ou bien un autre dictionnaire est construit de manière identique c.à.d en utilisant la même métrique et les mêmes images dans la séquence d'apprentissage que celles utilisées pour la quantification.

Les difficultés principales pour l'utilisation de ce type de quantification sont :

- La génération du dictionnaire ou codebook. Cette phase permettant la création du catalogue de référence est très importante et dépend de la structure et de la nature de l'image.

- La structuration du dictionnaire pour une recherche efficace et rapide des index de référence.

S'il est vrai que la mise en oeuvre de cette quantification est complexe, elle permet d'obtenir des taux de compression importants. Par exemple, si l'on considère comme indiqué ci-dessus un dictionnaire formé de vecteurs de taille 512 chacun, avec les index codés sur 9 bits, ceci donne un codage de 9/512 bit/pixel (bpp), soit un taux de compression de 512 * 8/9 pour une image ayant $2^8$ niveaux de gris. Remarquons que ce taux ne dépend pas de la taille de l'image, mais de la longueur des mots (ici vecteurs) du dictionnaire. En général, si l'on veut coder un bloc de taille n*m pixels avec un dictionnaire ayant des mots sur d bits, on a un rapport de d /n* m bit/pixel d'où un taux de compression de n * m* b/d pour une image codée sur $2^b$ bits.

Les expériences quantitatives réalisées [3] montrent que la quantification vectorielle est meilleure que la quantification scalaire parce qu'elle donne de meilleurs taux de compression.

### 2.4.3 Codage entropique

Pour la partie codage, les méthodes de compression avec pertes utilisent généralement le codage de Huffman, arithmétique ou LZW selon le type de transformée utilisée, par exemple, pour le format GIF, c'est le codage LZW qui

est utilisé, pour le JPEG c'est Huffman et pour pratiquement toutes les méthodes a base d'ondelettes c'est surtout le codage arithmétique qui est utilisé.

## 2.5 Transformée en ondelette et multirésolution

### 2.5.1 Définitions

Les ondelettes, famille de fonctions déduites d'une même fonction, appelée ondelette mère, par opérations de translations, dilatations et rotations, sont nées de l'analyse des signaux sismiques haute résolution.. Cet outil puissant a ensuite été développé, tant du point de vue pratique que du point de vue théorique, par des chercheurs tels Y. Meyer [7], I. Daubechies [8][9], S. Mallat [10], P.G. Lemarié [11], Morlet [12] etc... pour aboutir à la théorie de l'analyse par ondelettes telle qu'elle existe aujourd'hui.

L'idée de l'analyse par ondelette est de décomposer un signal sur une base de fonctions d'un sous-espace ayant des propriétés bien déterminée. En particulier, on peut chercher à analyser un signal en tentant de localiser dans le temps les irrégularités du signal c.à.d les variations brusques dans le signal qui correspondent aux hautes fréquences. Nous donnerons ci-dessous quelques définitions se rapportant de près ou de loin à l'analyse par ondelettes.

- Le concept de la transformée en ondelettes est déterminée par des fonctions de base qui sont déduites par translation, contraction et dilatation [9] d'une fonction mère $\psi(t)$. Ces fonctions de base sont de courte durée à haute fréquence et de longue durée à basse fréquence [8].

- La décimation est la réduction du taux d'échantillonnage par un facteur entier M [13].

- L'interpolation consiste en une élévation de la fréquence d'échantillonnage [13] par un facteur entier L.

- L'analyse multi-résolution est en quelque sorte la formalisation mathématique du phénomène suivant : lorsqu'on regarde un paysage ou un objet, suivant la distance à laquelle se trouve ,on observe plus ou moins des détails, mais le paysage où l'objet est le même (figure 2.4) . On peut dire que l'espace dans lequel il est représenté n'est pas le même, d'où cette présence ou absence de détails .

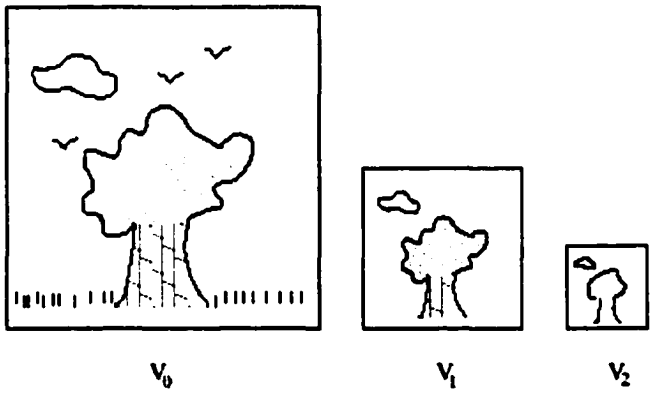


Figure 2.4: Représentation d'un paysage en analyse multi-résolution. Progressivement les détails disparaissent, d'abord les oiseaux et l'herbe, puis le tronc devient uniforme.

### 2.5.2 Transformée continue en ondelette

La transformée en ondelettes réalise une analyse à toutes les échelles. C'est une fonction S(a,b) qui associe aux paramètres 'a' et 'b' la valeur du coefficient $C_{a,b}$ de l'ondelette $\psi_{a,b}$ dans la décomposition du signal. La quantité 'b' est le paramètre de localisation temporelle, tandis que '1/a' est le paramètre de fréquence. $C_{a,b}$ est une intégrale qui mesure la somme des aires algébriques décrites par la courbe produit de S(t) et $\psi_{a,b}$ tel qu'illustré par la figure 2.5.

La transformation continue en ondelette, utilise des paramètres de translation et de dilatation variant continuellement, les fonctions utilisés sont définies par:

$$\psi_{a,b}(x) = \frac{1}{\sqrt{|a|}}\psi(\frac{x-b}{a}) \quad où\ a,b \in R,\ a \neq 0. \tag{2.14}$$

L'expression de la transformée en ondelette continue W d'une fonction f est donnée par :

$$W(a, b) = < f, \psi_{a,b} > = \int \psi_{a,b}(t) f(t) dt \tag{2.15}$$

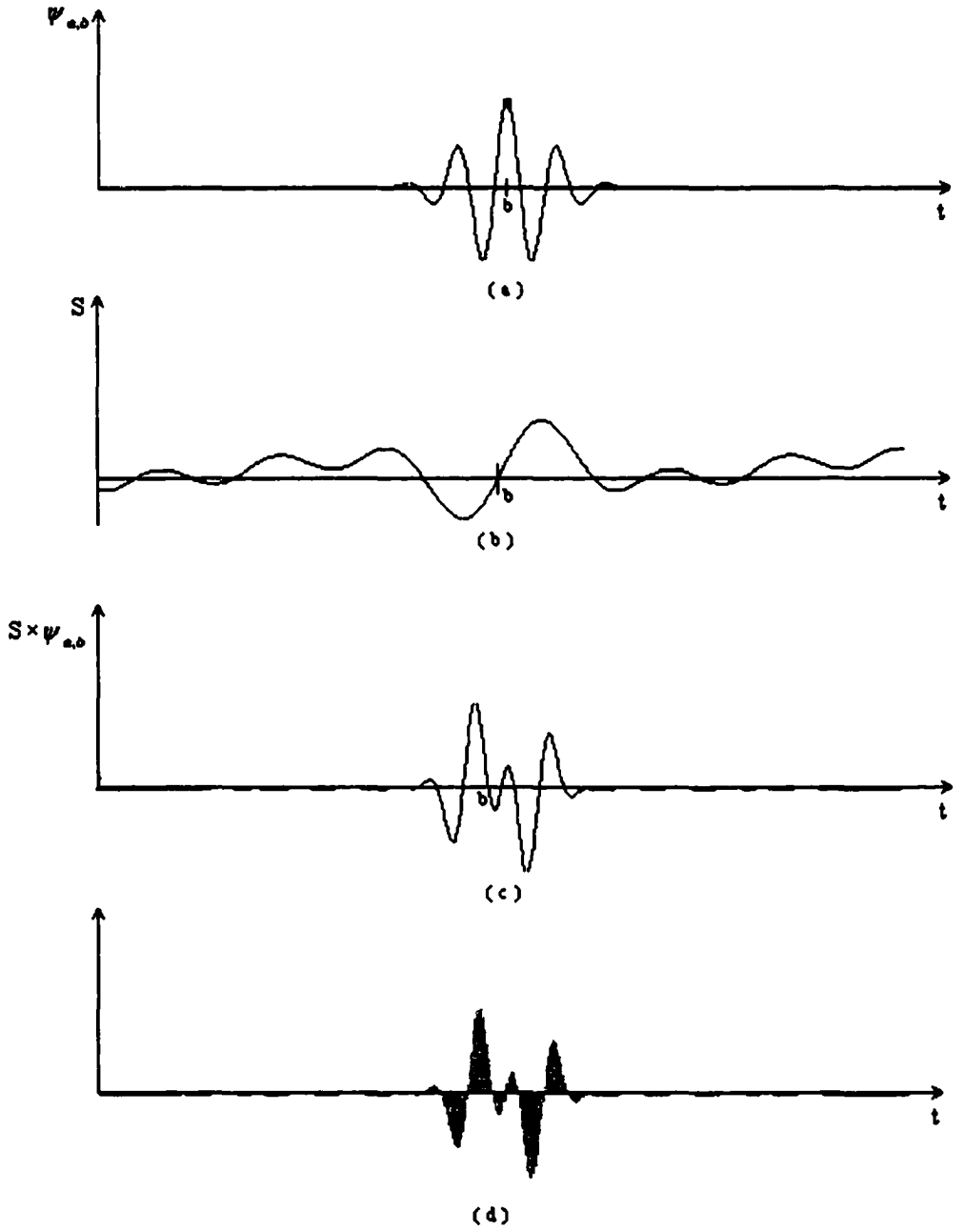


Figure 2.5 (a) Ondelette de morlet $\psi_{a,b}$ de fréquence 1/a centrée en b, (b) signal S(t), (c) produit S(t) $\psi_{a,b}$, (d) mesure du coefficient $C_{a,b}$ représenté par l'aire du signal produit.

### 2.5.3 Transformation discrète en ondeletttes

Les deux paramètres a et b peuvent être discrétisés afin d'obtenir un ensemble de fonctions d'ondelette à paramètres discrets. On peut considérer [8] : $a = a_0^i$ et $b = j.a_0^i .T$ . Où i,j sont des entiers et T une période.

Les coefficients d'ondelettes discrétisés sont donnés par :

$$C_{i,j} = \int \psi_{i,j}(t) f(t) dt \qquad (2.16)$$

Avec un choix optimal de $\Psi(t)$, $a_0$ et T , on peut reconstruire f(t) par :

$$f(t) \approx c \sum_i \sum_j c_{i,j} \psi_{i,j}(t) \qquad (2.17)$$

Dans le cas discret, on peut démontrer [14] qu'une décomposition par ondelettes dites orthogonales peut être représenté par un banc de filtres dyadiques de décomposition (tel que représenté par la figure 2.7). Les coefficients des filtres dépendent de l'ondelette utilisée .Cette représentation est donnée par [14]:

$$\sum x(n) h(2^k - n) \qquad (2.18)$$

### 2.5.4 Exemple d'ondelette

Un exemple très simple mais très utile pour illustrer les meilleurs propriétés des ondelettes, est l'ondelette de Haar, où l'on peut illustrer facilement les propriétés de la fonction d'échelle et de l'ondelette. Cette ondelette a aussi des utilisations pratiques.

- Fonction d'échelle, elle est définit par :

$$\Phi(x) = \begin{cases} 1 & si\ \ 0 \le x < 1 \\ 0 & ailleurs \end{cases} \tag{2.19}$$

Le sous espace $V_0$ est étendu par la fonction d'échelle «scaling function» $\Phi(x\text{-}k)$, qui est formée de translations entières de la même fonction. Le sous espace $V_1$ est étendu par $\Phi(2x\text{-}k)$, qui est formée de translations de k/2 de la fonction d'échelle sur un intervalle de 1/2. En général, $V_j$ est étendu par des translations de $k/2^j$ de la fonction d'échelle sur un intervalle de $1/2^j$. Les relations de double échelle «two-scale relation» de l'ondelette de Haar sont:

$$\Phi(x) = \sum_k p_k \Phi(2x-k) = \Phi(2x) + \Phi(2x-1) \tag{2.20}$$

avec $p_0 = p_1 = 1$ et $p_j = 0,\ \forall j$

- L'ondelette : l'ondelette de Haar $\psi(x)$ correspond à la fonction d'échelle de Haar "Haar scaling function" $\Phi(x)$ donnée par:

$$\Phi(x) = \begin{cases} 1 & si\ \ 0 \le x < 1/2 \\ -1 & si\ 1/2 \le x < 1 \\ 0 & ailleurs \end{cases} \tag{2.21}$$

Il est facile de construire la relation de double échelle "two-scale relation" de l'ondelette de Haar comme suit :

$$\psi(x) = \Phi(2x) - \Phi(2x - 1) \tag{2.22}$$

avec $q_0 = 1$ $q_1 = -1$.

- Relations de décomposition et de reconstruction : les relations de reconstruction sont données par:

$$\begin{bmatrix} \Phi(x) \\ \psi(x) \end{bmatrix} = \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \begin{bmatrix} \Phi(2x) \\ \psi(2x-1) \end{bmatrix} \tag{2.23}$$

Les relations de décomposition sont l'inverse des celles de reconstruction et il sont données par:

$$\begin{bmatrix} \Phi(2x) \\ \psi(2x-1) \end{bmatrix} = \begin{bmatrix} 1/2 & 1/2 \\ 1/2 & -1/2 \end{bmatrix} \begin{bmatrix} \Phi(x) \\ \psi(x) \end{bmatrix} \tag{2.24}$$

avec $a_0 = 1/2$, $b_0 = 1/2$, $a_1 = 1/2$ et $b_1 = -1/2$ (séries de décomposition non nulles).

Il existe plusieurs autres ondelettes usuelles telles que: l'ondelette de DAUBECHIES [8] , le chapeau mexicain, et l'ondelette de MORLET..

### 2.5.5 Transformée bidimentionnelle

Pour nombreuses applications, en particulier en traitement d'images, on a besoin d'utiliser la transformée en ondelette à deux dimensions, ceci peut se faire sur une image en appliquant une transformée en ondelette 1D sur les lignes puis en la réappliquant sur les colonnes de l'image résultante [8].

### 2.5.6 Bancs de filtres

En pratique, la transformée en ondelette est associée aux bancs de filtres.

La notion des bancs de filtres se base sur la séparation du signal original en plusieurs bandes de fréquence (basse fréquence et haute fréquence) pour mieux le traiter et le transmettre. Au récepteur, on reconstruit le signal en rassemblant ces diverses bandes (figure 2.6).

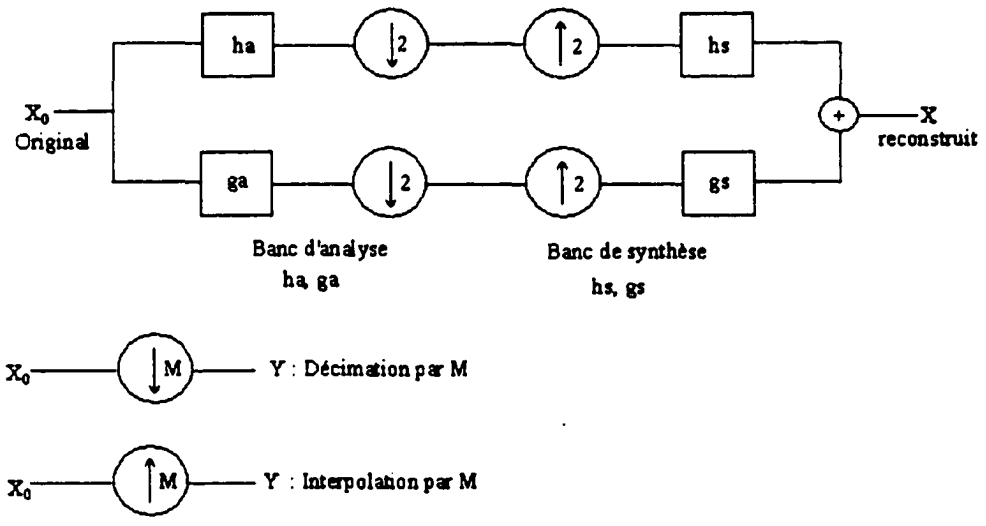


Figure 2.6 Banc de filtres (banc d'analyse/synthèse) à un étage.

Un banc de filtres est un ensemble de filtres reliés entre eux par des décimateurs et des interpolateurs [15]. Pour un banc de filtres à deux canaux, les filtres analysants sont normalement un passe-bas et un passe-haut.

A cause de ses caractéristique, la transformée en ondelette est associée à l'analyse multi-résolution. L'idée de base de l'analyse multi-résolution est de diviser l'image originale en plusieurs sous-images selon une échelle de bandes d'octaves puis de les analyser séparément.. La façon la plus commune de diviser l'image originale en plusieurs sous images selon une échelle de bandes d'octaves en se basant sur les ondelettes, est d'utiliser un banc de filtres d'ondelettes unidirectionnel. La figure 2.7 représente un tel banc de filtres d'ondelettes à deux

étages, les figures 2.8 représente la décomposition d'une image à deux étages . La figure 2.9 illustre la distribution des bandes de fréquence issue d'une transformée à deux étages, on peut distinguer la sous-bande LL qui définit les coefficients les plus importants de l'image (aussi appelée image 'grossière') et aussi, LH, HL et HH qui désignent les détails obtenus après transformée.

Ce genre de bancs de filtres d'ondelettes est adapté aux algorithmes de compression d'images qui utilisent pour la quantification des quantificateurs de type EZW, pour effectuer un codage progressif.

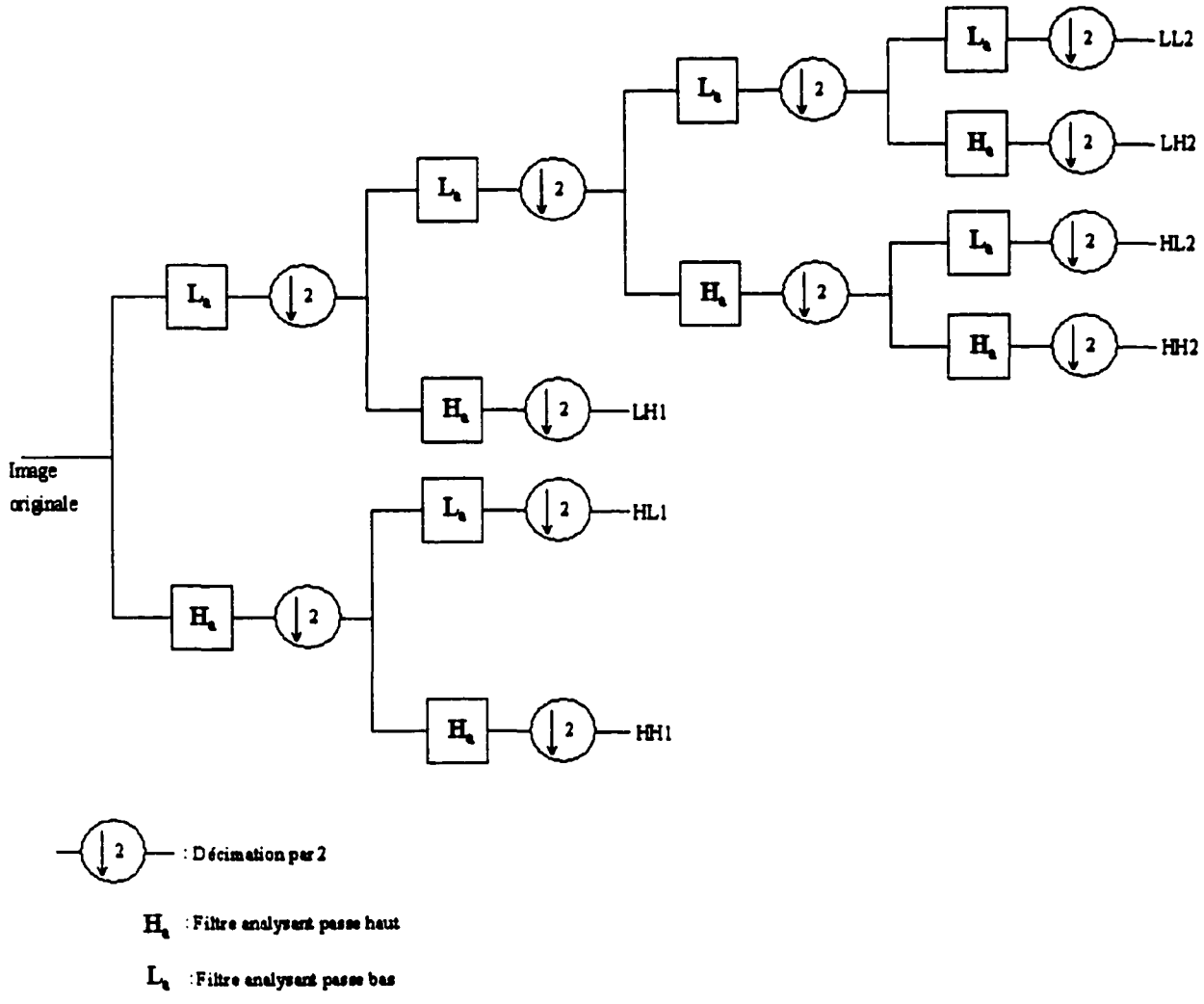


Figure 2.7 Banc de filtres d'ondelettes à deux étages de décomposition

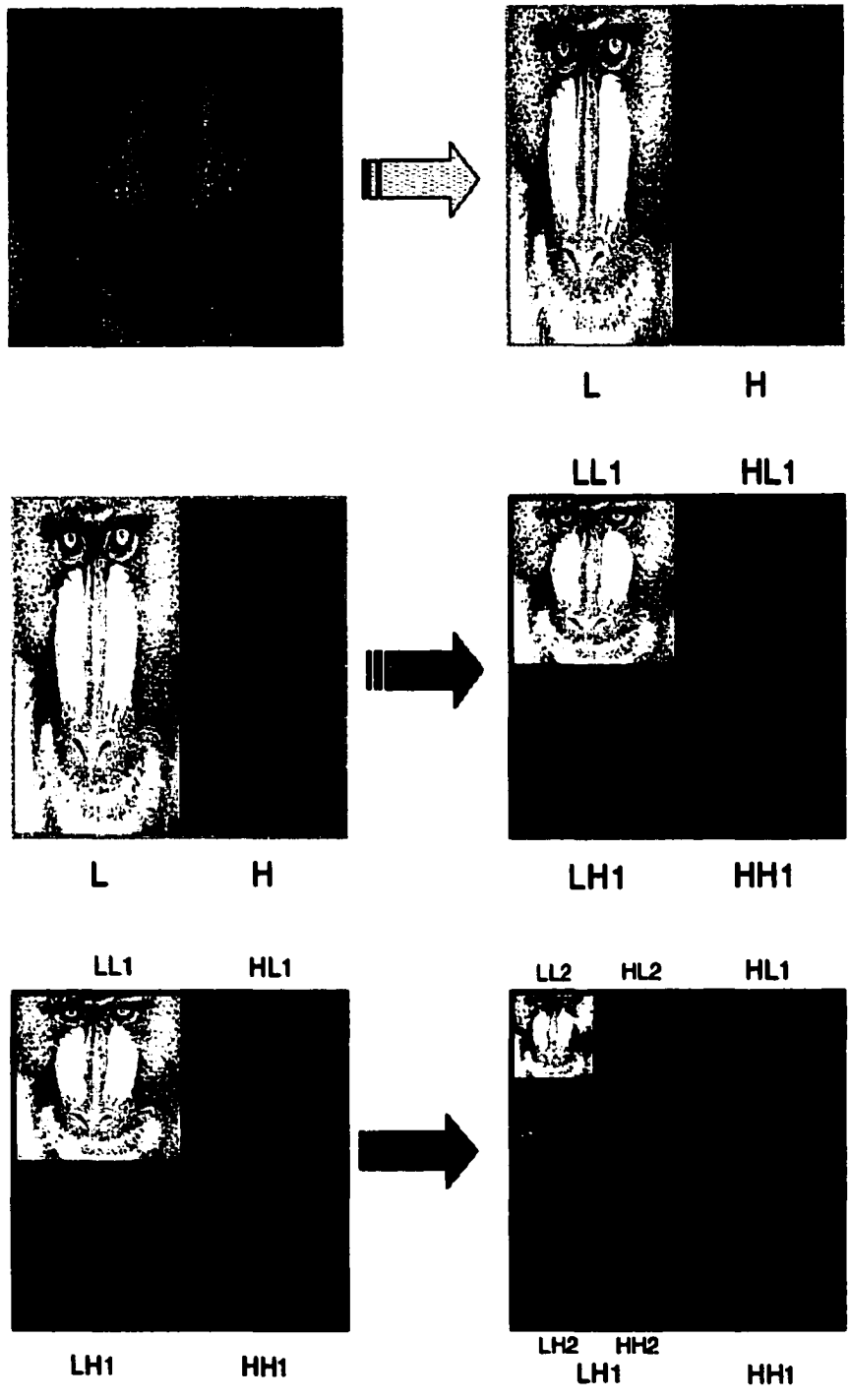


Figure 2.8 Décomposition en ondelette à deux étages de 'Mandrill'

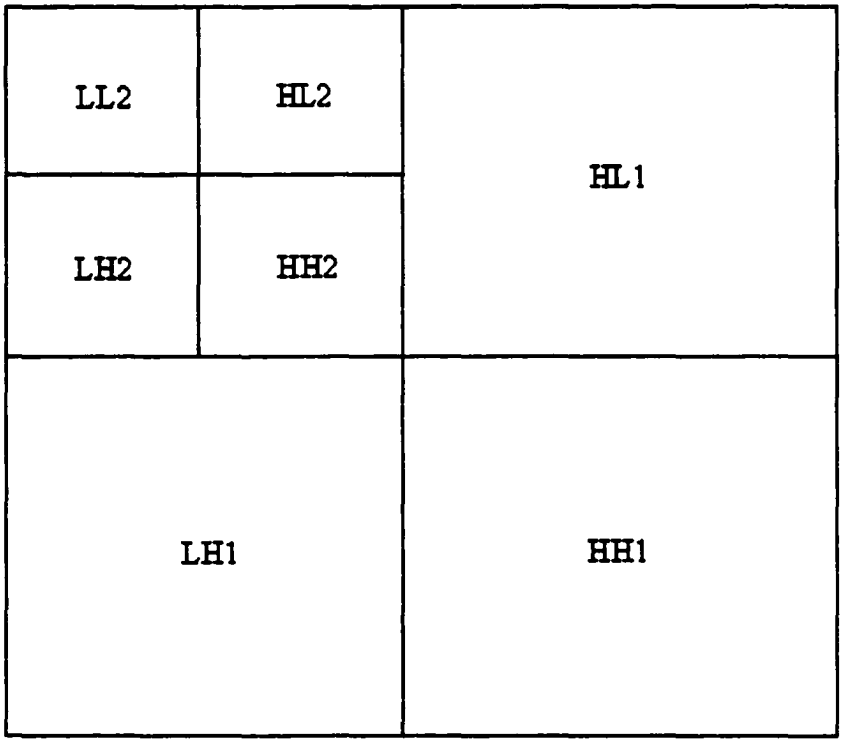


Figure 2.9 Distribution des fréquences dans une décomposition à deux étages

## 2.6 Conclusion

La compression d'image par ondelette est maintenant bien développée pour les algorithmes de compression d'images, on la trouve dans le dernier standard de compression JPEG2000 [16]. Cependant pour mettre au point une méthode complète, il faut associer à la transformée un quantificateur et codeur entropique adéquats tels ceux utilisés dans le EZW ou le SPIHT [17-18].

# CHAPITRE 3

# QUANTIFICATION ZEROTREE

## 3.1 Introduction

La transformée en ondelette permet, comme décrit dans le chapitre précédent, de représenter les images sous forme de coefficients ordonnés en bandes de fréquences. Pour la compression d'images, la transformée représente le premier maillon de la chaîne, afin de comprimer l'information, il faut compléter le cycle par la quantification et le codage.

Afin de garder le coté progressif de la transformée en ondelettes, une méthode d'organisation et de quantification des coefficients s'impose : c'est le EZW.

La méthode de codage progressif connue sous le nom de Embedded Zerotree Wavelet coding (EZW), proposée par Shapiro [17], est une méthode simple et très efficace de compression d'image par ondelettes. Elle a démontré sa puissance dans les deux formes de compression (avec et sans perte d'informations) depuis son élaboration en 1993. Plusieurs variantes de ce type de codage ont été proposé par différents chercheurs dans le domaine, ce qui fait sa force .On peut citer par exemple le SPIHT (Set Partitionning In Hierarchical Tree) réalisé par A.SAID et W.PEARLMAN [18] qui est la variante la plus populaire de l'EZW. L'intérêt de consacrer ce chapitre à cette méthode est de la mettre en évidence vu qu'elle sera à la base des algorithmes proposés dans le chapitre 5.

## 3.2 Présentation du codage EZW

En général, dans une représentation d'image par coefficients d'ondelettes, l'image obtenue est organisée de façon à représenter les principaux traits de l'image dans les bandes de basses fréquence, puis les détails dans les bandes de hautes fréquence. Le principe de l'EZW s'appuie sur cette représentation, pour coder les coefficients d'une manière progressive, ainsi, on commence par les basses fréquences L, ensuite on code les détails (hautes fréquences), l'avantage de cet algorithme, est que l'on a en tout temps un niveau de compression et que l'on peut arrêter en tout moment le codage.

Le codage EZW est basé sur deux principales observations :

- Quand une image est transformée par ondelettes, l'énergie dans les sous-bandes diminue pendant que l'échelle diminue (la basse échelle signifie la haute résolution). Ainsi les coefficients d'ondelette seront plus petits en moyenne dans les sous-bandes plus hautes que dans les sous-bandes inférieures. Ceci prouve que le codage progressif est un choix très normal pour des images transformées par ondelettes, puisque les sous-bandes plus hautes ajoutent seulement les détails fins.

- Les grands coefficients d'ondelette sont plus importants que les plus petits.

Ces deux observations sont exploitées en codant les coefficients d'ondelettes par ordre décroissant, dans plusieurs passages. Pour chaque passage on choisit un seuil par rapport auquel tous les coefficients d'ondelettes sont comparés. Si un coefficient d'ondelette est supérieur au seuil, il est codé et retiré de l'image, sinon, il est laissé pour le prochain passage. Quand tous les coefficients d'ondelettes ont été examiné, le seuil est abaissé et l'image est rebalayée pour ajouter plus de détails à l'image déjà codée. Ce processus est répété jusqu'à ce que tous les coefficients d'ondelettes soient encodés

complètement ou qu'un autre critère soit satisfait (débit binaire maximum par exemple) selon le mode de compression utilisé.

L'algorithme emploie la dépendance entre les coefficients d'ondelettes à travers différentes échelles pour coder efficacement les grandes parties de l'image qui sont au-dessous du seuil actuel, c'est ici qu'intervient le zerotree.

Dans la représentation de ' l'image ondelettes ', chaque coefficient peut être considéré en tant qu'ayant quatre descendants dans la prochaine plus haute sous-bande (figure 3.1). Ainsi que pour les quatre descendants, chacun a également quatre fils dans la prochaine plus haute sous-bande et nous voyons un arbre à quatre descendants émerger ; chaque racine a quatre branches, ce processus se poursuit jusqu'aux fréquences les plus hautes.

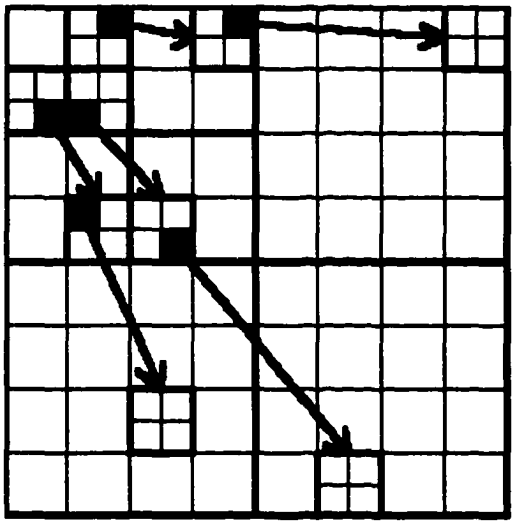

Figure 3.1: Représentation de l'organisation en arbre des coefficients d'ondelettes

Nous pouvons maintenant donner une définition du zerotree. Un zerotree est un quadruple-arbre dont tous les nœuds enfants sont égaux ou plus petits que les nœuds

parents. L'arbre est codé avec un symbole unique et reconstruit par le décodeur comme quadruple-arbre rempli de zéros. Nous devons insister sur le fait que la racine doit être plus petite que le seuil par rapport auquel les coefficients d'ondelettes sont comparés, sinon, ce coefficient ne serait pas considéré comme base de zerotree.

Le codeur d'EZW exploite le fait qu'il y 'a une probabilité très élevée que tous les coefficients dans un arbre quadruple soient plus petits qu'un certain seuil si la racine de cet arbre est plus petite que ce seuil. Ceci entraîne alors un seul code zerotree pour tout l'arbre. Ceci dit, en balayant toute la représentation des coefficients d'ondelettes, des basses aux hautes fréquences, on aura automatiquement beaucoup de zerotree ce qui constituera un gain considérable au niveau de la compression.

Une bonne approche est d'utiliser un seuil et seulement un signal au décodeur si les valeurs sont plus grandes ou plus petites que le seuil. Si nous transmettons également le seuil au décodeur, il peut reconstruire une bonne partie de l'image. Pour arriver à une reconstruction parfaite, on doit cependant répéter le processus après abaissement du seuil, jusqu'à ce que le seuil devienne inférieur au plus petit coefficient que nous avons voulu transmettre. Nous pouvons rendre ce processus beaucoup plus efficace par la soustraction du seuil du coefficient d'ondelette qui lui était supérieur. Le choix des seuils peut être optimisé en considérant un lien entre eux, si l'ordre prédéterminé est un ordre des puissances de deux, le codage est appelé codage 'bitplane', puisque les seuils correspondent dans ce cas-ci aux bits dans la représentation binaire des coefficients. Le codage d'EZW tel que décrit dans [17] utilise ce type de seuils.

L'information additionnelle minimale exigée par le décodeur, outre le code des coefficients signifiants, est le nombre de niveaux de transformées d'ondelettes utilisés et le seuil initial. Si on retranche le coefficient après son codage, il est nécessaire d'envoyer aussi la valeur moyenne de l'image, ça permet une meilleure reconstruction et l'obtention d'un meilleur PSNR .

## 3.3 Algorithme

L'algorithme peut être divisé en deux parties :une tache principale (aussi passage dominant) qui constitue le fond de tache du codeur, et une partie secondaire ou subalterne qui est plus pour le raffinement de la reconstruction de l'image pendant le decodage.

### 3.3.1 Traitement principal

Comme initialisation de l'algorithme, on choisit le seuil de départ, pour le 'bitplane coding', le seuil de départ est défini par [17]:

$$t_0 = 2^{\lfloor \log_2 \max(|im(x,y)|) \rfloor} \tag{3.1}$$

où im(x,y) désigne les coefficients d'ondelettes, x et y étant leur position dans l'espace. Le symbole 'max' correspond a la valeur maximale de tous les coefficients de la représentation en ondelettes de l'image . Ce seuil, comme le montre la définition, est un multiple de 2, en fait c'est la puissance de 2 la plus proche (inférieure ou égale) du maximum des coefficients de l 'image .Cette puissance fera partie par la suite du paquet d'informations transmis pour initialiser le décodage, cette partie sera développée plus en détail dans la description du protocole du codec EZW.

La matrice de coefficients est parcourue par l'une des méthodes suivantes : 'Raster scan' ou 'Morton scan' (figure 3.2). Ces méthodes de parcours ont été choisies de manière a préserver l'ordre d'importance des coefficients traités, ainsi, pour les deux types, on commence par parcourir les coefficients de basse fréquence, et on avance graduellement vers les détails (hautes fréquences ), la différence entre elles est la façon avec laquelle est parcourue une même sous

bande .La figure 3.2 illustre l'ordre de parcours des coefficients pour les deux méthodes sur une matrice issue de transformée en ondelette à trois niveaux de décomposition .

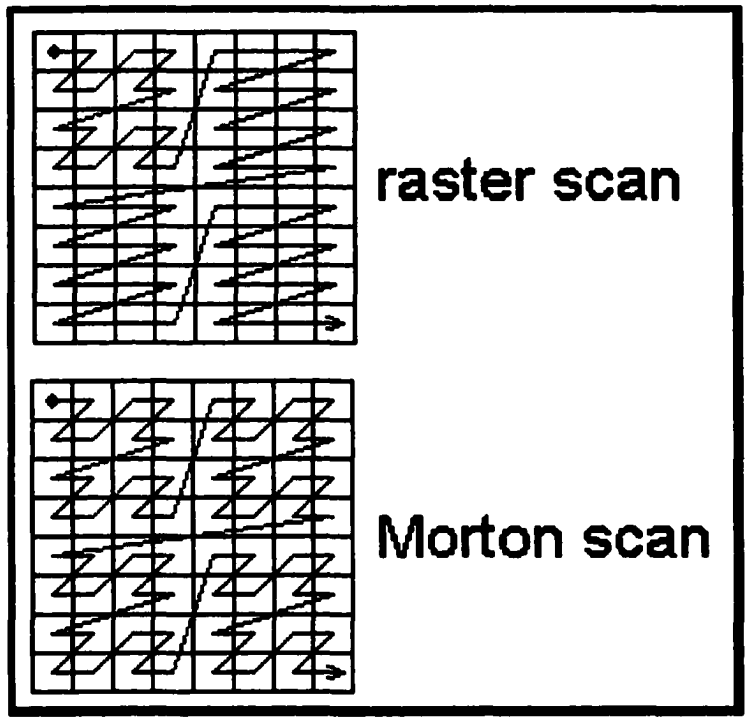


Figure 3.2 Méthodes de parcours des coefficients

Chacun des coefficients parcourus est comparé (en valeur absolue) au seuil $t_0$, si le coefficient est supérieur au seuil il est codé 'Positif' ou 'Négatif', sinon il est soit 'Zero isolé' ou bien ' Zerotree', on se ramène ainsi de X (selon niveaux de gris, résolution de l'image...) symboles à coder à un dictionnaire de quatre symboles :

- Positif (P) : indique que la valeur absolue du coefficient traité est supérieure au seuil et que son signe est positif.

- Négatif (N) : indique que la valeur absolue du coefficient traité est supérieure au seuil et que son signe est négatif.

- Zéro isolé (Z) : indique que la valeur absolue du coefficient traité est inférieure au seuil et qu'il existe parmi ses descendants (selon l'arborescence présentée dans la section 3.2) ceux qui sont signifiant (c.à.d. supérieurs au seuil)

- Zerotree (R) : indique que la valeur absolue du coefficient traité est insignifiante par rapport au seuil considéré ainsi que tous les coefficients qui lui succèdent dans l'arbre de descendance.

Après le parcours de tous les coefficients, le seuil est divisé par 2, et l'opération est refaite selon le nouveau seuil, cette méthode est appelée 'quantification par approximations successives' (SAQ : Successive Approximation Quantization) et peut être refaite tant que le seuil $t_n$ est supérieur ou égal à 1 , sachant que :

$$t_n = t_{n-1} / 2 \tag{3.2}$$

### 3.3.2 Traitement secondaire

Le traitement secondaire (ou subalterne) sert à faire des raffinement sur les valeurs des coefficients, ainsi, après chaque passage dominant, chaque coefficient codé 'Positif' ou 'Négatif' subit une autre comparaison sur un autre seuil $t_s$ qui est proportionnel au seuil du passage dominant $t=2^n$ [17],

$$t_s = 2^n + \frac{2^{n+1} - 2^n}{2}$$

Ceci peut être ramené à :

$$t_s = 3 * 2^{n-1} \tag{3.3}$$

où n est la puissance du seuil du passage dominant.

Ce passage secondaire permet au décodeur dans le cas d'une compression avec pertes d'informations, d'avoir plus de précision sur la valeur du coefficient codé, certes, la reconstruction ne sera pas parfaite (sauf dans peu de cas ), mais, elle sera de loin meilleure que si on ne code qu'avec le passage dominant.

Dans le cas ou l'on veut effectuer une compression sans pertes d'informations, le passage subalterne devient inutile

### 3.3.3 Protocole de codage

Pour que le codage soit parfait, un protocole entre codeur et décodeur est établi, ainsi, le décodeur doit connaître le dictionnaire de codage (dans ce cas les 4 symboles utilisés au codage), et le type de parcours des coefficients effectué (Raster ou Morton scan). Pour sa part, le codeur doit transmettre au moins le seuil de départ, de préférence la puissance associée à ce seuil (une puissance de 2 puisque le codage est bitplane), et le nombre de niveaux de décomposition par ondelettes. On peut aussi trouver dans certains cas une condition d'arrêt si l'on effectue une compression sans pertes d'informations.

### 3.3.4 Décodage

En premier lieu, le décodeur crée une matrice de dimension égale à l'image traitée à partir du nombre de niveaux de décomposition, ensuite, comme le codeur, il calcule le premier seuil dont la puissance lui a été transmise. Le parcours de la matrice 'image' commence alors et selon les symboles lus par le décodeur un traitement est effectué :

- Si le symbole est 'Positif ' (P), la valeur du seuil est additionnée au contenu de la case en cours.

- Si le symbole est 'Négatif' (N), Le seuil est retranché du coefficient parcouru.

- Si le symbole est 'Zerotree' (R), tout l'arbre associé à ce coefficient sera ignoré par rapport au seuil courant.

- Si le symbole est 'Zero isolé' (Z) : cela veut dire qu'il existe au moins un coefficient appartenant à l'arborescence du coefficient étudié qui est signifiant par rapport au seuil courant d'où, aucun coefficient ne sera ignoré dans cette arborescence.

A la fin du parcours, le seuil est divisé par 2 et l'algorithme reprend. Si le seuil atteint la valeur 1, la reconstruction sera parfaite sans aucune perte, mais au cas où l'on désire arrêter avant le décodage idéal, on peut avoir recours au traitement secondaire qui permettra plus de précision au niveau de la compression avec pertes d'informations.

Le principe général de la méthode est illustré par l'organigramme suivant :

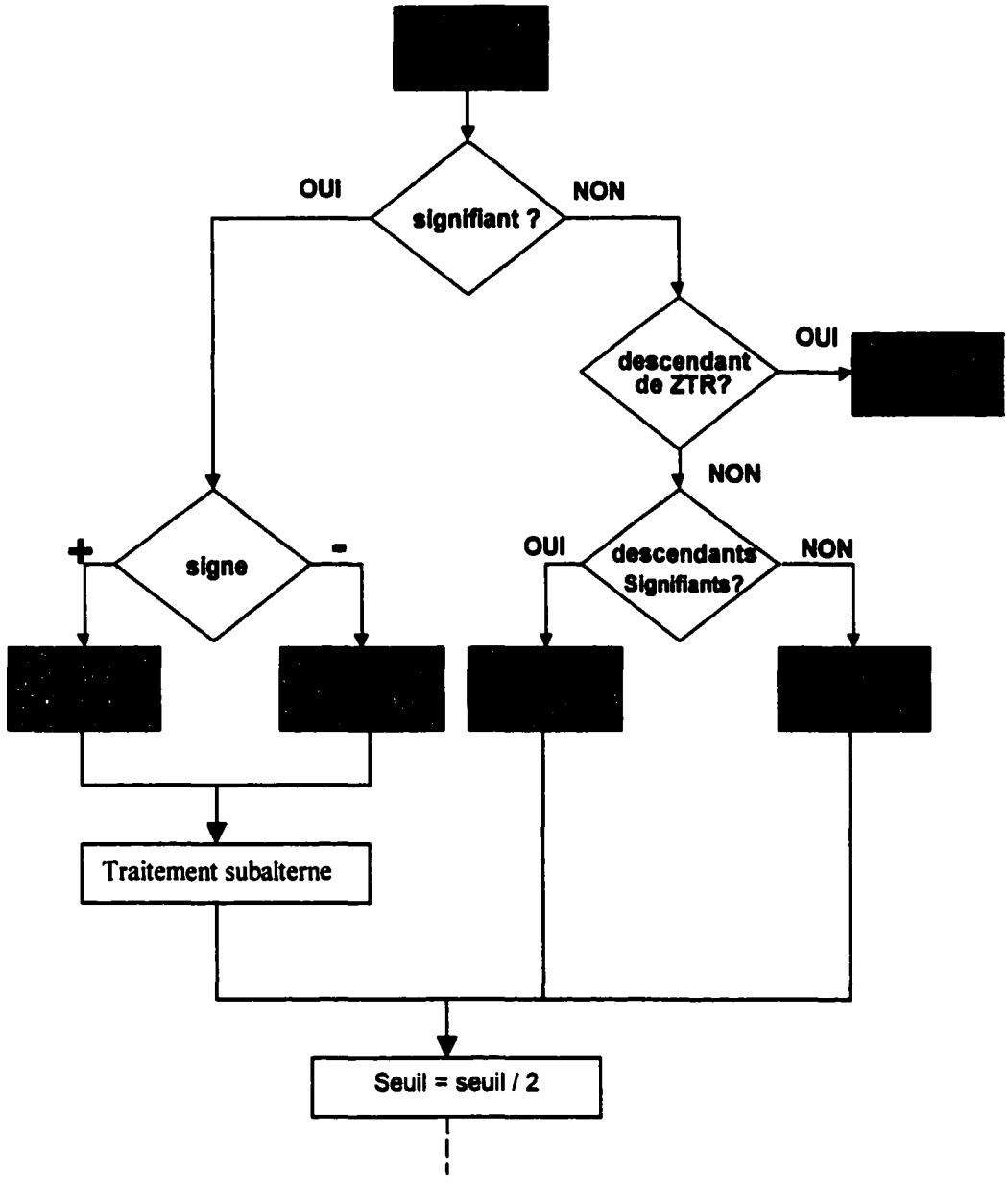


Figure 3.3 Principe du codage zerotree

## 3.4 Exemple:

La méthode est illustrée par l'exemple suivant , le codage a été appliqué sur la matrice de coefficients à trois niveaux de décomposition suivante :

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 63 | -34 | 49 | 10 | 7 | 13 | -12 | 7 |
| -31 | 23 | 14 | -13 | 3 | 4 | 6 | -1 |
| 15 | 14 | 3 | -12 | 5 | -7 | 3 | 9 |
| -9 | -7 | -14 | 8 | 4 | -2 | 3 | 2 |
| -5 | 9 | -1 | 47 | 4 | 6 | -2 | 2 |
| 3 | 0 | -3 | 2 | 3 | -2 | 0 | 4 |
| 2 | -3 | 6 | -4 | 3 | 6 | 3 | 6 |
| 5 | 11 | 5 | 6 | 0 | 3 | -4 | 4 |

Seuil initial

$$t_0 = 2^{\lfloor \log_2 63 \rfloor} = 32$$

Type de parcours choisi : Raster scan.

Résultats obtenus :

Premier passage dominant :

Le tableau suivant montre les coefficients parcourus pour t=32, la sous-bande à laquelle ils appartiennent, ainsi le code associé pour le passage dominant (code pr) et le passage secondaire (code sec.) :

Tableau IV

Exemple de codage EZW

| Sous-bande | coefficient | Code pr | Code sec | Commentaire |
|---|---|---|---|---|
| LL3 | 63 | P | 1 | (1) |
| HL3 | -34 | N | 0 | (2) |
| LH3 | -31 | Z | | (3) |
| HH3 | 23 | R | | (4) |
| HL2 | 49 | P | 1 | |
| HL2 | 10 | R | | |
| HL2 | 14 | R | | |
| LH2 | -13 | R | | |
| LH2 | 15 | R | | |
| LH2 | 14 | Z | | |
| LH2 | -9 | R | | |
| HL1 | -7 | R | | (5) |
| HL1 | 7 | R | | |
| HL1 | 3 | R | | |
| HL1 | 4 | R | | |
| LH1 | -1 | R | | |
| LH1 | 47 | P | 0 | |
| LH1 | -3 | R | | |
| LH1 | -2 | R | | (5) |

Commentaires :

1 - Le coefficient '63' est supérieur au seuil, et comme il est positif, il est codé 'P', il est aussi supérieur au seuil secondaire donc, son second code est '1', ce coefficient va changer de valeur pour le prochain passage dominant, on aura dans cette position 31 (eq.. à 63-32).

2 - On a ici un coefficient négatif '-34' dont la valeur absolue est supérieur au seuil actuel, il est donc codé 'N', cependant, dans le passage secondaire, il est inférieur au seuil secondaire, donc il est codé '0' pour le passage subalterne.

3 - Le coefficient '-31' est inférieur au seuil (en valeur absolue), et comme il possède un descendant dont la valeur est supérieure au seuil (47 dans la sous-bande LH1), il est codé 'Z' (Zero isolé). Le traitement secondaire n'est pas effectué dans ce cas vu que le coefficient n'est codé ni positif ni négatif.

4 - On remarque que le coefficient '23' ainsi que tout ses descendants sont insignifiants par rapport au seuil considéré, d'où, le coefficient actuel sera codé 'Zerotree' (R), et tous ses descendant ne seront pas traités pendant ce passage.

5 - Les principales remarques c'est que les sous bandes HH1 et HH2 ne figure pas dans la liste, et ceci parce qu'ils sont descendant d' un arbre de zéros (zerotree), c'est pourquoi on a sur 64 coefficients dans la matrice, juste 20 d'entre eux sont codés.

Les résultats complets de cet exemple donnent

$t_0$=32 $t_{s0}$=48

D1 : PNZRPRRRRZRRRRRRRPRR

S1 : 1010

$t_1$=16 $t_{s1}$=24

D2: ZRNPRRRRRRRR

S2: 100110

$t_2$=8 $t_{s2}$=12

D3 : ZZZZZPPNPPNRRNNPRPRRNRRRRRRRRPRRRPRRRRRRRRRPRRRRRRRRRRRR

S3: 10011101111011011000

$t_3$=4 $t_{s3}$=6

D4: ZZZZZZZRZRZNZZZZPRRPRPPRPNPRNRRRRRPRPNPPPPRRRRRPRPRRRPNP

S4: 1101111101100100000111011010001001010110 0

$t_4$= 2 $t_{s4}$=3

D5: ZZZZZRZZZZZRPZZZRRPRRRRNPRPPRRPRRRNPPNRRRRPNNPRRPRRP PRRR

S5: 10111100110100010111110101101100100000000110110110011000111 1

$t_5$=1

D6: ZZZRRZRRRZRRRRRNNRRR

Où :

P : Positif

N : Négatif

Z : Zéro isolé

R : Zerotree

D : passage dominant.

S : passage subalterne.

On remarque que pour t=1, on a effectué juste le passage dominant, ceci s'explique par le fait que si on arrive à ce stade, c'est que la reconstruction est parfaite, donc on n'a pas besoin de raffinement, d'où la non utilisation du passage secondaire.

## 3.5 Variantes du 'EZW coding'

Depuis son élaboration, le codage EZW a fait l'objet de nombreuses recherches, et associé à la plupart des méthodes de compression par ondelette proposées dans la littérature.

### 3.5.1 Set Partitionnig in Hierarchical Trees (SPIHT)

SPIHT est la variante du codage zerotree la plus utilisée jusqu'à date, elle se base sur les principes du codeur EZW et un développement proposé dans [19].

Cette méthode proposée par A.Said et Pearlman [18] diffère de ses antécédentes par la manière avec laquelle les coefficients sont classés ainsi que pour le traitement des coefficients significatifs. Un autre point important est que cette méthode, sans passer par le codage arithmétique, génère un flot binaire intéressant (au niveau de la compression) sans passer par un codage entropique

contrairement au codage zerotree dont les résultats n'apparaissent qu'une fois associé au codage arithmétique.

Le principal avantage de cet algorithme est qu'il est plus rapide en exécution que ses prédécesseurs, et peut donner de meilleurs résultats.

Le principe du SPIHT s'appuie sur l'organisation des coefficients en 3 listes : LIP(i,j) (List of Insignifiant Pixels) LSP (i,j) (List of Signifiant Pixels) et LIS (i,j)(List of Insignifiant Sets) , pour les deux premières, les coordonnées (i,j) désignent un coefficient, pour la troisième, elles représentent tout un arbre de coefficients. Le codage s'effectue selon un passage dominant et un passage secondaire pour raffinement, la méthode est bien détaillée dans [18].

### 3.5.2 Autres méthodes

Parmi les développements proposés pour le codage zerotree, on peut citer ceux qui ont optimisé le traitement des coefficients.

Hsung et autres [20] ont proposé une méthode de classification des coefficients d'ondelette selon leur régularité, en considérant que les coefficients non réguliers sont du bruit et les eliminent, ils proposent aussi une méthode pour distinguer irrégularités des coefficients.

Hisakazu et autres [21] ont défini une nouvelle méthode plus flexible pour déterminer les descendants des coefficients dans une représentations en sous-bandes, changeant ainsi la définition classique des arbres de coefficients.

Zhong et Leung [22] ont revu la manière de traiter les coefficients

Significatifs. Normalement, avec le codage zerotree, si un coefficient est supérieur au seuil, ses descendant sont automatiquement traités même s'ils sont non significatifs. L'algorithme proposé élimine cet inconvénient en ajoutant deux symboles 'IT_sig ' et 'LK_sig' au code zerotree. Ils désignent respectivement un coefficient significatif dont les descendants ne le sont pas, et n coefficient significatif dont un descendant au moins l'est aussi.

Shapiro [23] a proposé une méthode plus rapide de détection des arbres de zeros dans le codage EZW, cette méthode utilise des matrices de même dimension que l'image source appelées 'carte de signification' pour optimiser la détection des zerotrees.

## 3.6 Conclusion

Malgré toutes les recherches qui ont suivi la mise en œuvre du codage zerotree, il demeure une des méthodes de codage les plus utilisés et la plus souvent citée dans les revues spécialisées.

L'avantage de cette méthode est de classer les coefficients par ordre d'importance et de permettre un codage progressif qui permet par la suite d'avoir une bonne représentation de l'image selon un taux de compression désiré. Cependant, le codage EZW en lui même n'effectue pas de la compression, il doit être associé au codage arithmétique, qui jusqu'à ce jour demeure le codage entropique le plus efficace associé au EZW coding .

# CHAPITRE 4

# CODAGE ARITHMÉTIQUE

## 4.1 Introduction

La dernière étape de la compression est le codage entropique. Dans le premier chapitre, une description de différentes méthodes de ce type de codage a été présentée. Dans ce chapitre, on fera le tour de la méthode la plus adaptée aux algorithmes de compression de type EZW (chapitre 3), c'est le codage arithmétique.

Le codage arithmétique est une méthode qui permet d'obtenir une bonne compression de données sans pertes d'informations selon un modèle bien développé. Sa force est que le code est proche de l'entropie, ce qui la classe parmi les meilleurs algorithmes de codage entropique.

Contrairement aux algorithmes de Huffman [4] et de Shannon-Fano [4] qui associent à des symboles des motifs binaires dont la taille dépend de leur distribution, le codeur arithmétique traite le fichier dans son ensemble en lui associant un nombre décimal rationnel unique. Ce nombre est compris entre 0 et 1 et possède d'autant moins de chiffres après la virgule que le nombre de symboles de la trame dont il est issu. Ces chiffres décimaux dépendent non seulement des symboles du fichier dans l'ordre ou ils apparaissent, mais aussi de leur distribution statistique.

Pour comparer le codage arithmétique au codage de Huffman, on suppose que l'on va coder des symboles de probabilités 99% et 1%. L'information contenant le symboles dont la probabilité est de Ps est de –log(Ps) bits près, c.à.d. le symbole qui a une probabilité de 99% sera codé sur 0.015 bits, tandis que le codage de Huffman utilise au minimum 1 bit/symbole [24].

Les avantages obtenus par ce type de codage sont plus apparents quand un symbole revient souvent dans la trame à coder. L'un des avantages du codage arithmétique est le fait qu'il peut opérer sans connaissance préalable des probabilités de chaque symbole de son dictionnaire de données, cette propriété lui donne un gain proportionnel à la lettrine du dictionnaire et au nombre de bits utilisés pour chaque symbole.

Les inconvénients de ce type par rapport à Huffman sont la lenteur d'un côté, et l'impossibilité de décoder une trame d'information au milieu, ce qui est possible dans le codage de Huffman où l'on peut indexer un point de départ dans un symbole à décoder.

Depuis sa découverte, le codage arithmétique a été sujet de nombreuses recherches où l'on a optimisé la méthode sans pour autant obtenir de meilleurs résultats qu'avec la première version proposée par Witten et Moffat [25], ceux ci se sont inspirés de la méthode de Langdon [26]. Parmi les recherches qui ont suivis, on peut citer Howard et Vitter [27] [28], et Witten et Moffat [29].

## 4.2 Description du codage arithmétique

Comme condition de départ pour la mise en œuvre du codage arithmétique, on considère un intervalle [H,L[ dynamique qui sera modifié durant tout le processus de codage et qui contiendra à la fin le code final de sortie. Cet intervalle est initialisé à [0,1[, et sa modification se fait par rapprochement de ses bornes, ce qui nous donne à la fin un

intervalle entre 0 et 1. Tout nombre appartenant à cet intervalle final peut désigner le code arithmétique de tout le message étudié, contrairement au codage de Huffman qui code chaque symbole indépendamment. On admet aussi l'hypothèse qu'on connaît tous les symboles possibles qui peuvent être codés (exemple lettres de 'a' à 'j') dont le nombre est n.

L'algorithme de codage de base de la méthode arithmétique peut être décrit par les étapes suivantes :

a. On subdivise l'intervalle d'étude [H,L[ en sous intervalles égaux chacun d'entre eux correspond à un des n symboles du dictionnaire, la longueur de chaque sous intervalle constitue l'estimation de la probabilité du symbole dans le message à coder; Dans cette première étape, on obtiendra n intervalle de longueur 1/n.

b. On parcourt le message à coder, et selon chaque symbole s rencontré, on change l'intervalle d'étude [H,L[ par son sous-intervalle $[H_s,L_s[$ , et sa probabilité est aussi changée vu qu'on a une nouvelle distribution des symboles

c. On redistribue les nouvelles probabilités sur l'intervalle calculé.

d. On reprend les mêmes étapes jusqu'à la fin du message à coder.

La figure 4.1 illustre le choix des sous-intervalles

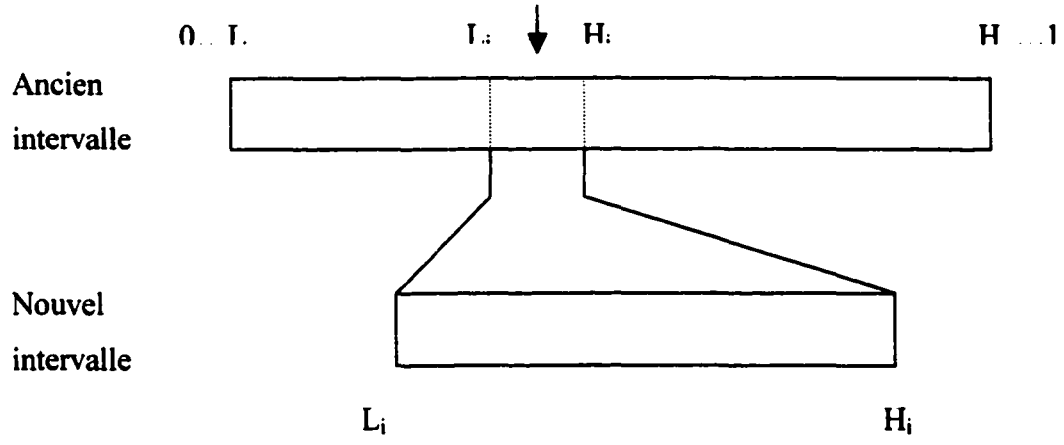


Figure 4.1 Subdivision d'intervalle après le codage d'un symbole $a_i$

A la fin du codage, on obtient un intervalle final $[H_x,L_x[$ . Tout nombre inclus dans cet intervalle peut correspondre à un code de tout le message considéré .

En ce qui concerne le décodage, l'hypothèse de départ est qu'on connaît déjà le dictionnaire de symboles .On initialise l'intervalle de départ [H,L[ par [0,1[, et on teste sur le mot à décoder x, les étapes de décodage sont :

a. On subdivise l'intervalle d'étude [H,L[ en sous intervalles égales chacun d'entre eux correspond à un des n symboles du dictionnaire, la longueur de chaque sous intervalle constitue l'estimation de la probabilité du symbole dans le message à décoder; dans cette première étape, on obtiendra n intervalle de longueur 1/n.

b. On teste à quel sous-intervalle appartient le code, le symbole $a_i$ associé a ce sous-intervalle $[L_i,H_i[$ est ajouté au message décodé qui devient le nouvel intervalle d'étude.

c. On redistribue les nouvelles probabilités sur l'intervalle calculé.

d. On reprend les mêmes étapes jusqu'à ce qu'on trouve un indice de fin de fichier.

## 4.3 Exemple de codage

Le codage arithmétique peut être intégré avec toute méthode de compression d'image, de texte pour assurer la partie codage, comme tout codeur, le codeur arithmétique a une entrée sous forme de symboles et sortie en bits, en fait la sortie est un nombre unique entre 0 et 1 , mais c'est juste la partie après la virgule qui est transmise.

Pour illustrer la méthode, on considère un mot 'bccb' à coder la liste des symboles utilisés contient les lettres {'a','b','c'}. Au début, le codeur attribue des probabilités égales à tous les symboles du dictionnaire, dans l'exemple considéré, on aura P(a) = 1/3 , P(b) = 1/3 , P(c) = 1/3 , et les distribue uniformément entre 0 et 1, l'intervalle $]0,1]$ distribué ainsi :

$Ia = [0,1/3[$

$Ib = [1/3,2/3[$

$Ic = [2/3,1[$

La première lettre à coder étant 'b', le nouvel intervalle d'étude devient $]1/3,2/3]$, et les probabilité de chacun des symboles devient :

P (a) = 1/4 ;

P (b) = 2/4 ;

P (c) = 1/4 .

Ces probabilités seront de nouveau distribuées selon le nouvel intervalle d'étude, on aura alors :

Ia = [0.3333,0.4167[ ;
Ib = [0.4167,0.5834[ ;
Ic = [0.5834,0.6667[.

Le deuxième symbole est 'c', le nouvel intervalle d'étude deviendra Ic et les probabilités :

P (a) = 1/5 ;
P (b) = 2/5 ;
P (c) = 2/5 .

L'exemple au complet est illustré par la figure 4.2.
A la fin, l'intervalle d'étude devient : I = [0.6390,0.6501[.

Tout nombre appartenant à cet intervalle peut constituer le code à transmettre, mais, le code le plus optimal serait celui contenant le moins de chiffres, pour ce cas, 0.64 ou 0.65 serait un code idéal.

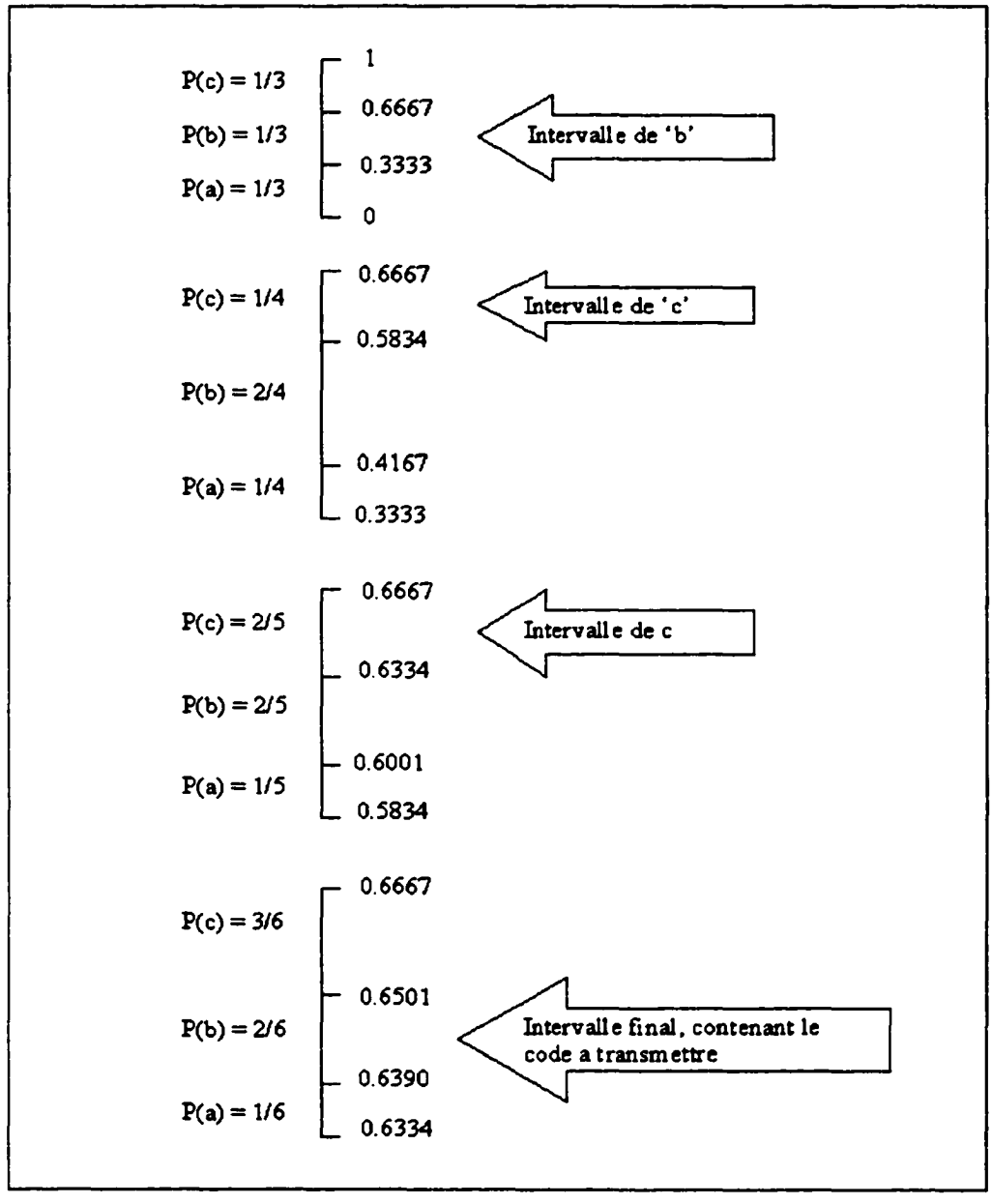


figure 4.2 Exemple de codage arithmétique du mot 'bccb'

Le décodeur fonctionne de la même façon que le codeur, ainsi, il attribue à ses symboles des probabilités égales et les distribue uniformément sur l'intervalle de départ [0,1[ :

Ia = [0,1/3[

Ib = [1/3,2/3[

Ic = [2/3,1[

Le code transmis étant 0.64 appartient à l'intervalle Ib, donc la première lettre décodée est 'b', les probabilités sont réajustées et l'intervalle d'étude devient Ib :

P (a) = 1/4 ,

P (b) = 2/4 ,

P (c) = 1/4 .

L'exemple du décodage est illustré par la figure 4.3.

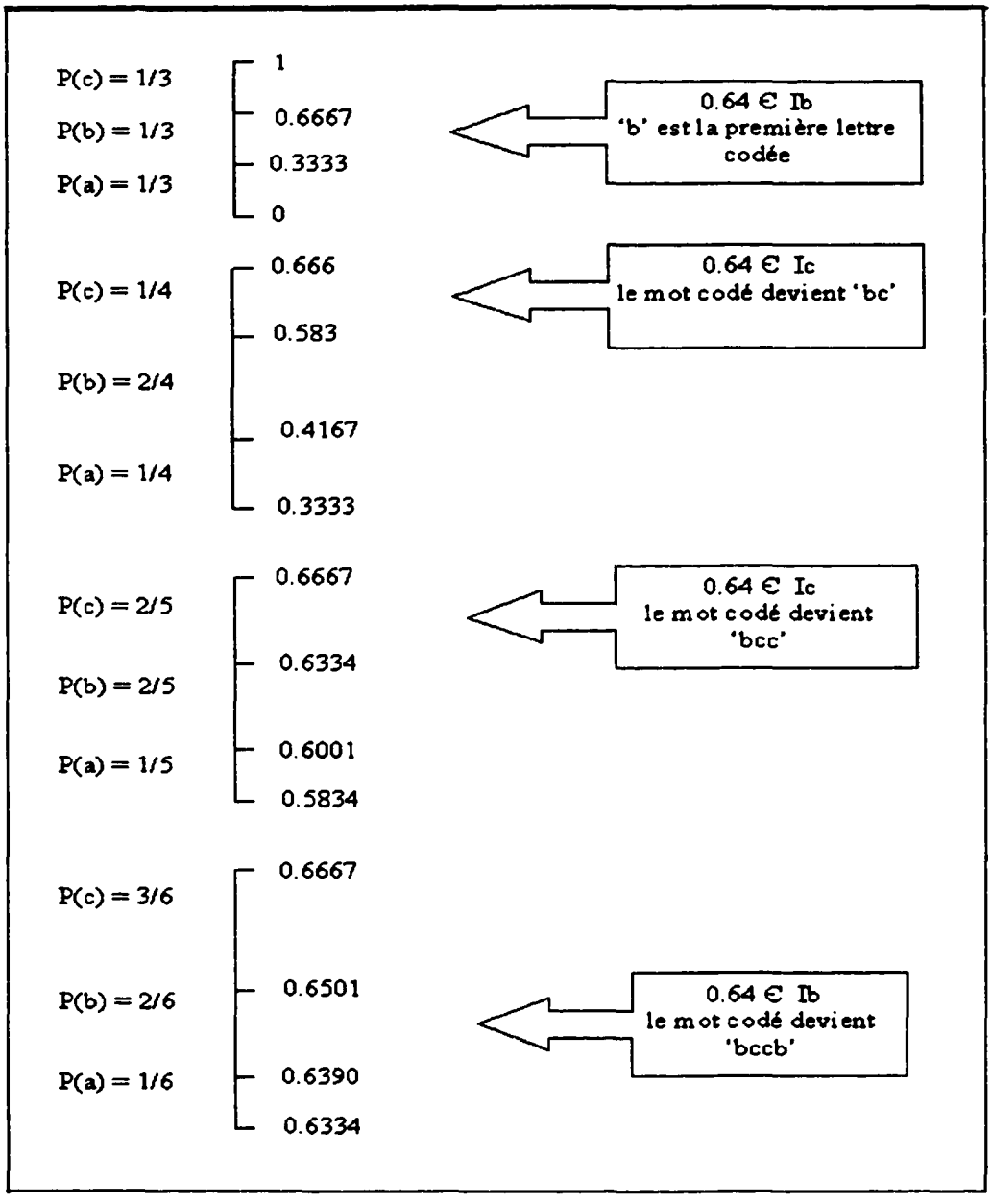


Figure 4.3 Exemple de décodage arithmétique

## 4.4 Mise en oeuvre du codage arithmétique :

On peut résumer le codage du symbole s du mot à coder par les étapes suivantes, sachant qu'on a N symboles de probabilité P(i) à coder:

1 . borne_inf $\leftarrow \sum_{i=1}^{s-1} P(i)$

2 . borne_sup $\leftarrow \sum_{i=1}^{s} P(i)$

3 . larg $\leftarrow$ borne_sup – borne_inf .

4 . borne_sup $\leftarrow$ borne_inf + (larg * borne_sup) .

5 . borne_inf $\leftarrow$ borne_inf + (larg * borne_inf) .

En ce qui concerne le décodage:

1 . Trouver

$$\sum_{i=1}^{s-1} P(i) \leq (\text{code} - \text{borne\_inf}) / (\text{borne\_sup} - \text{borne\_inf}) < \sum_{i=1}^{s} P(i)$$

2 . Adapter les intervalles comme pour le codage.

3 . Déduire les symboles codés.

Avec :

borne_sup : Borne superieure de l'intervalle d'étude

borne_inf : Borne inferieure de l'intervalle d'étude

Larg : largeur de l'intervalle d'étude

## 4.5 Remarques :

Le nombre transmis au décodeur est juste l'entier après la virgule du code final.

Pour compléter l'information du codage, il faut transmettre une condition d'arrêt au décodeur qui peut être la longueur de la chaîne codée, cette condition peut être aussi intégrée au codage en désignant un sous intervalle de EOF (End Of File), qui, une fois détecté au décodage, déclenchera l'arrêt du processus .

Le nombre de chiffres nécessaires pour coder un mot est proportionnel à :
– log(borne_sup-borne_inf) de l'intervalle final, exemple pour : borne_sup=0.354268 , borne_inf= 0.35427 on a –log(borne_sup-borne_inf) = 5.69, d'où la nécessité de 6 chiffres pour coder le mot correspondant.

Comme précisé auparavant, le codage n'est complet qu'après avoir balayé tous les symboles, cependant, vu le nombre de caractères qui peuvent être traités, on obtient des chiffres énormes. Pour remédier à cela, on peut 'alléger les bornes de l'intervalle d'étude' en transmettant les chiffres de même rang qui sont égaux, dans notre exemple, l'intervalle obtenu après avoir codé la troisième lettre est : I = [0.6334,0.6667[, on remarque que les deux bornes ont un 6 au même rang qui sera forcement présent dans le code final, donc, on peut le sauvegarder et travailler avec l'intervalle [0.334,0.667[ , ce qui nous permet d'éviter d'éventuelles erreurs de calculateurs.

## 4.6 Conclusion

Malgré que le codage arithmetique ainsi que celui de Huffman sont des méthodes de codage entropique assez anciennes, elles demeurent les premiers choix dans ce type de codage avec une supériorité au codage arithmetique qu'on retrouve pratiquement dans toutes les recherches de compression de cette décennie, cependant l'handicap que peut constituer sa lenteur a généré plusieurs recherches qui visent son optimisation.

# CHAPITRE 5

# COMPRESSION PAR RÉGIONS D'INTÉRÊT

## 5.1 Introduction

Tout en étant très utile, la compression peut s'avérer limitée, surtout quand il s'agit de compression sans pertes de données ou les taux de compression sont assez faibles.

Dans le domaine biomédical, ce problème se fait sentir énormément vu que les images traités sont volumineuses et on ne se permet pas plus d'une compression sans pertes. A titre d'exemple, une mammographie analogique numérisée à 4096/4096 *16 bpp, requiert environ 33 MO. Une compression sans pertes de cette image ne permettra sûrement pas de gagner le maximum d'espace possible . Cependant, comme l'image n'est pas analysée en totalité, il sera utile de la compresser par région d'intérêt c.à.d. de traiter la région importante sans pertes d'informations et de compresser avec pertes ailleurs.

Dans ce chapitre on va proposer des méthodes de compression appliquées à des images en considérant des régions d'intérêt, ces méthodes s'appuient sur les standards de compression décrits dans les chapitres précédents.

## 5.2 Méthodologie

Une compression de données efficace peut se résumer en tROIs principales étapes illustrés dans la figure 5.1, pour chacun des bloc de cette figure, une méthode à été implémentée et développé pour être compatible avec les méthodes de compression proposées.

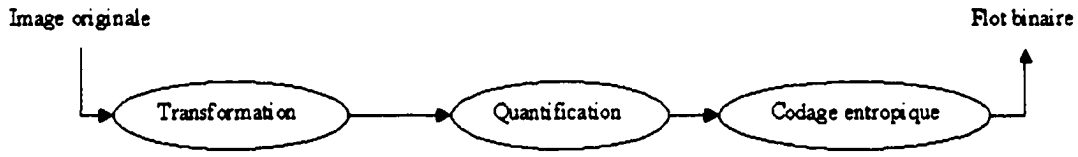


Figure 5.1 Étapes principales de compression

### 5.2.1 Transformée en ondelette

Comme transformation, la transformée en ondelette a été choisie pour les résultats satisfaisants qu'elle fournit au niveau de la compression ainsi que pour son côté progressif. Le fait que plus que 90% des méthodes de compression proposés dans ces dernières années sont à base d'ondelettes, démontre l'importance de cette transformée dans la compression de données.

En ce qui concerne la base d'ondelette choisie, il a été démontré [30] que la famille d'ondelette 'daubechies' est la plus efficace en compression d'image.

### 5.2.2 Quantification de type EZW

Un quantificateur comme celui utilisé pour EZW se présente comme premier choix, vu qu'il permet de garder le coté progressif de la transformée en

ondelette et permet le passage d'une compression avec pertes à une compression sans pertes. Aussi, vu le nombre élevé des 'zéros' constaté, une modélisation a été introduite : un nouveau symbole 'S' a été ajouté aux quatre symboles du zerotree ,ce symbole détermine une fréquence des zéros. Les essais ont démontré les avantages de cette modélisation qui donne de meilleurs résultats que ceux obtenus par la méthode normale de quantification par approximation successive (SAQ) .

### 5.2.3 Codage arithmétique

L'utilisation de ce type de codage est optimal pour les méthodes à base de zerotree. Il faut signaler que le code obtenu est précédé d'une 'entête' qui est constituée du nombre de symboles engendrés par la quantification ainsi que du seuil maximum considéré (la quantité log2(seuil_max) est suffisante vu qu'on utilise des puissances de 2 comme seuils). Durant la déquantification, le seuil maximal constituera un point de départ pour cette étape, quand au nombre de symboles, il servira au décodage arithmétique comme condition d'arrêt, autrement, si l'on ne spécifie pas cette condition, le décodage continuera indéfiniment .

## 5.3 1ère méthode : Codage zerotree local (ZL)

Pour effectuer une compression par région d'intérêt, une première approche consiste à segmenter l'image en sous images et à coder selon la région d'intérêt choisie. Cette zone sera délimitée par un nombre de sous images qui seront compressées sans pertes, Les autres sous-images seront codées avec pertes.

Dans cette méthode, on peut distinguer deux types de régions intérêt qu'on nommera :

- Région intérêt utilisateur ($ROI_u$) : c'est la ROI choisie par l'utilisateur sur l'image numérisée, soit en la délimitant par la souris ou bien par saisie des coordonnées des extrémités de cette ROI

- Région d'intérêt système ($ROI_s$) : c'est la ROI qui va être prise en considération par l'algorithme pour effectuer le codage ZL, une fois la $ROI_u$ connue, l'algorithme sélectionne toutes les sous-images issues de la segmentation qui contiennent une partie de cette $ROI_u$ (même si cette partie représente un pixel). Ces blocs ainsi réunis constituerons la $ROI_s$ .

Pour illustrer cette méthode, on propose dans la figure 5.2 une segmentation de l'image 'woman' en sous-images de 16 pixels par 16 .Arbitrairement, on a choisi l'œil gauche comme région d'intérêt (figure 5.3), c'est la $ROI_u$ .La $ROI_s$ correspondante est représentée par la figure 5.4, dans l'exemple considéré, elle est constituée de quatre sous images.

Pour cette méthode, tout l'algorithme de compression se fait à un niveau local (sur une sous image) (figure 5.5) comme décrit dans [31], chacune des sous images subit sa propre transformation en ondelette, quantification et codage indépendamment des autres régions. La région d'intérêt étant délimitée par des sous images, sera traitée sans pertes , toutes les autres parties qui restent seront codés selon le taux de compression désiré.

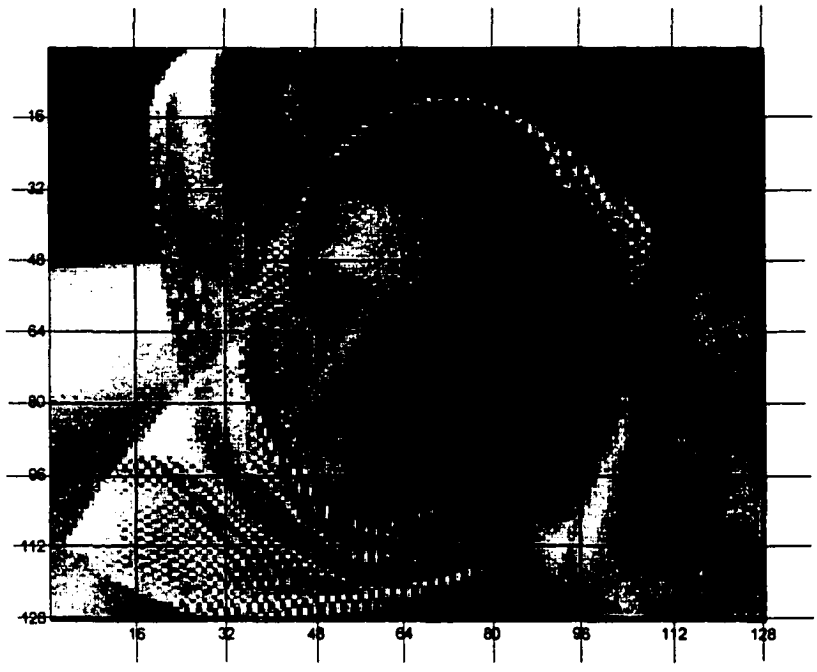


Figure 5.2 **Segmentation de l'image 'woman' en sous images de 16x16**

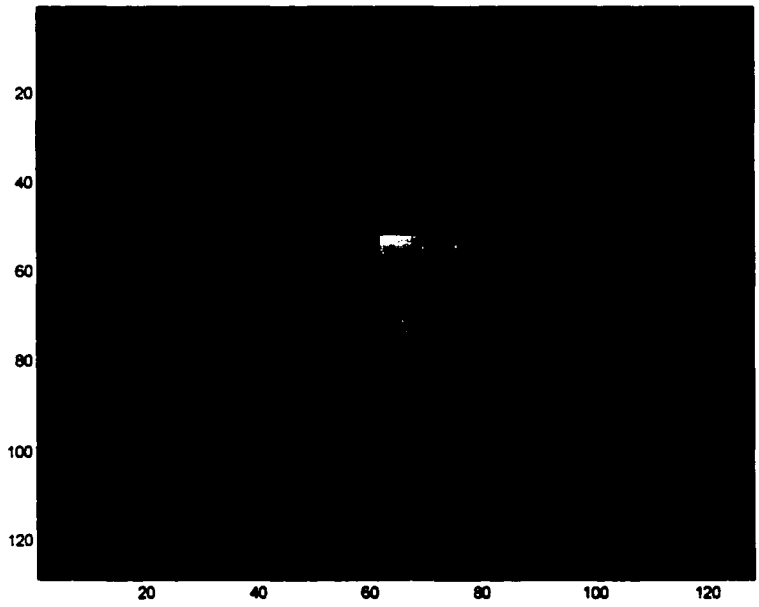


Figure 5.3 Région sélectionnée de l'image (ROI)

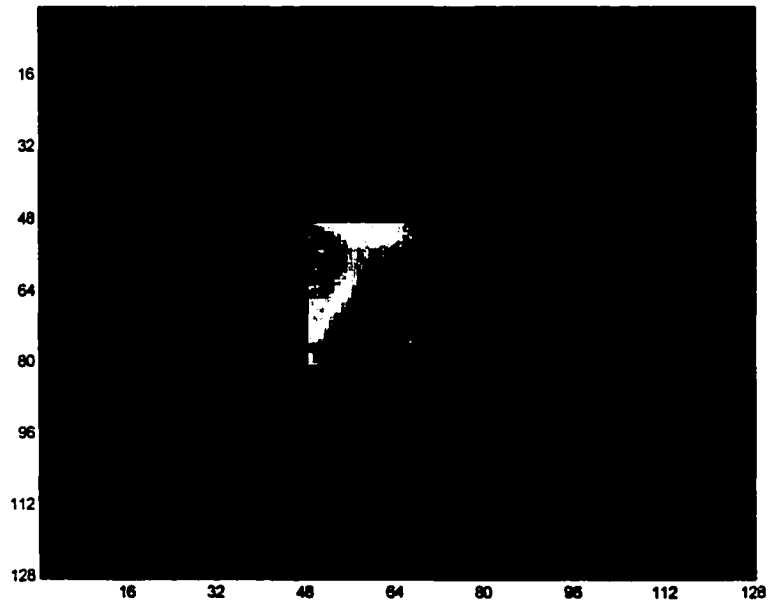


Figure 5.4 Sous images correspondantes à la ROI sélectionnée

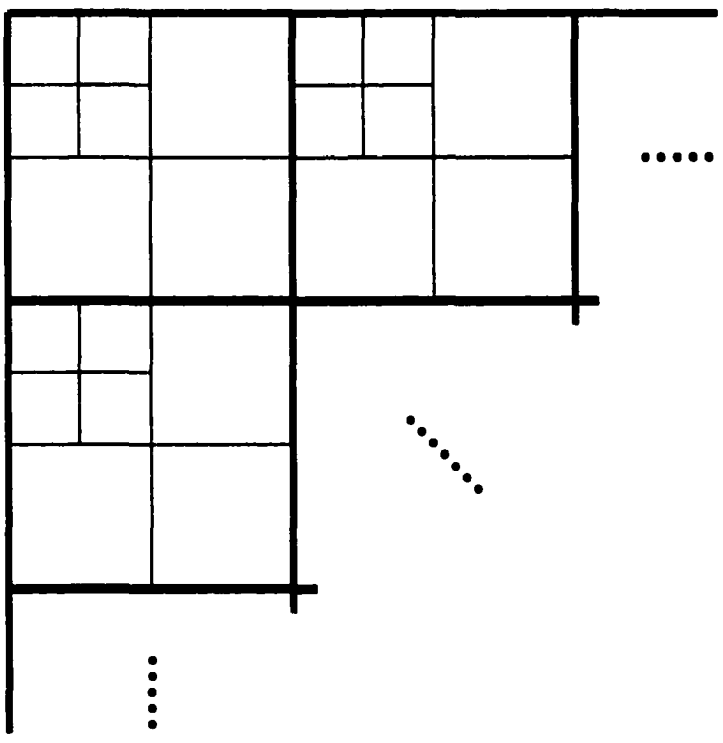

Figure 5.5 illustration de l'application du zerotree coding sur des régions d'une image et le parcours des coefficients localement

## 5.4 2 eme méthode : Zerotree global (ZG)

Cette méthode s'appuie sur la représentation de la transformation en ondelette qui permet de détecter une région d'une image à partir de sa transformée [32], pour ce faire, on détecte les coefficients d'ondelette correspondants à la région sélectionnée, ce qui nous permet de créer un masque pour le traitement de ces coefficients .

On peut résumer les étapes de cette méthode par:

- Sélection de la région d'intérêt
- Création d'un masque correspondant à cette ROI.

- Trouver la partie correspondante à cette ROI dans la représentation des coefficients.
- Traitement des coefficients de la ROI.
- Effectuer la quantification.
- Coder par le codage arithmétique

Les principales étapes sont la création du masque et le traitement des coefficients de la ROI

Pour ce qui est de la création du masque, deux approches ont été adoptées, la première (nommée 'mask1') se base sur une recherche effectuée par D. Nister et C.Christopoulos [33] , qui proposent un algorithme effectuant de simples décimations dans les bandes de hautes fréquences et donnant plus de détails dans les basses fréquences, cette approche est décrite par un algorithme (figure 5.5) et la figure 5.6.

Pour tout n $\in$ [1,N/2]

$H_m(n) = 1$ , si $X_m(2n) = 1$ ou $X_m(2n+1) = 1$

sinon $H_m(n) = 0$

$L_m(n) = 1$ , si $X_m(2n-2) = 1$ ou $X_m(2n-1) = 1$

ou $X_m(2n) = 1$ ou $X_m(2n+1) = 1$

ou $X_m(2n+2) = 1$ ou $X_m(2n+3) = 1$

Figure 5.6 Algorithme de la création du masque dans le domaine des coefficients.

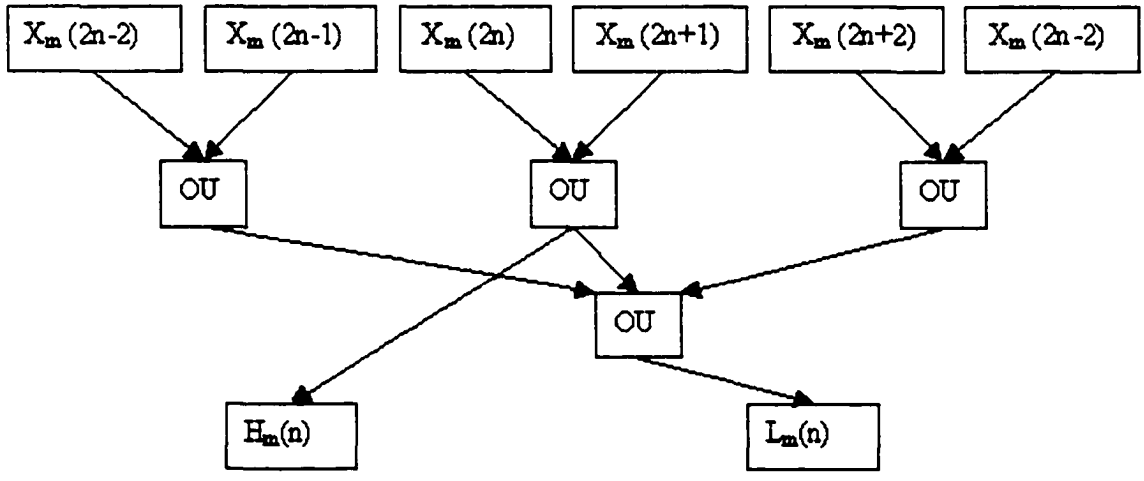


Figure 5.7 Schématisation de la transformée du masque

Où

$X_m$ Masque de la ROI.

$H_m$ Bande de hautes fréquences dans le domaine de la transformée.

$L_m$ Bande de basses fréquences dans le domaine de la transformée.

N longueur de l'image traitée.

La deuxième approche utilisée pour générer le masque (mask2) effectue sur le masque choisi la même décomposition par ondelette subie par l'image étudiée, donnant ainsi une matrice de coefficient qui, une fois binarisée à un seuil de 0 (par valeur absolue) engendre le masque d'étude, l'organigramme représenté par la figure 5.8 illustre la méthode de création de ce masque.

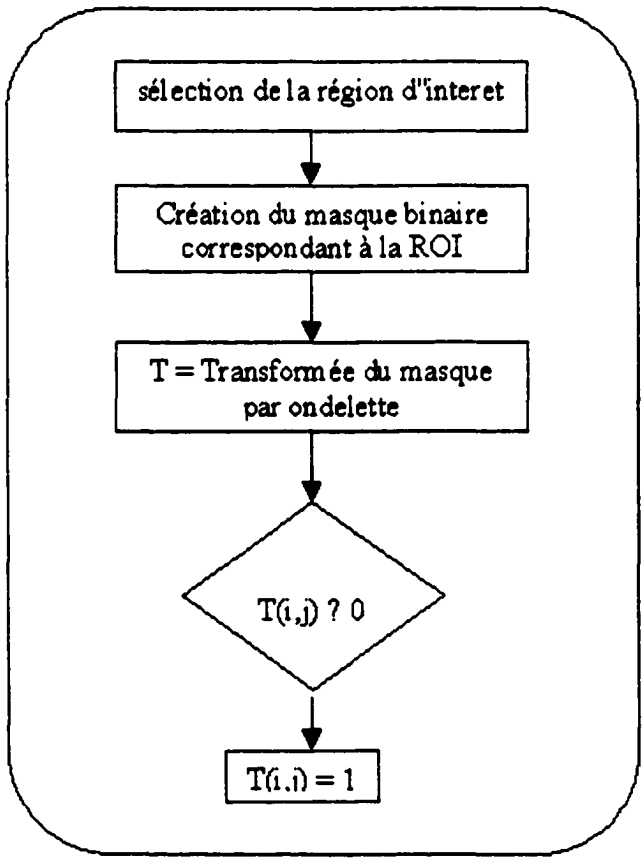


Figure 5.8 Masque issu de la transformée directe

Une fois les coefficients de la région d'intérêt localisés dans le domaine de la transformée, on procède à leur traitement : chacun de ces coefficients est décalé binairement d'autant de bits que l'équivalent du seuil final qui sera utilisé dans le codage EZW, par exemple, si l'on désire s'arrêter de coder tout les coefficients inférieurs à 64, on effectue un décalage binaire de 6 ($2^6$=64) sur les coefficients de la transformée correspondants à la région d'intérêt. Durant le décodage du zerotree , dès qu'on atteindra le seuil final (le seuil final est le seuil de pertes désiré ), les coefficients correspondants à la ROI seront déjà présents sans pertes , il suffit alors de faire le décalage binaire à dROIte (division par $2^{seuil\_final}$) de ces coefficients de $\log_2$ (seuil final) .

Soit l'exemple suivant pour illustrer l'intérêt de ce décalage; On considère la matrice 2x2 suivante $\begin{bmatrix} 22 & 5 \\ 3 & 1 \end{bmatrix}$, on désire appliquer l'algorithme pour un seuil de 4 et en considérant le coefficient '5' comme étant le plus important de la matrice (équivalent à une ROI dans une image). Normalement, si l'on effectue une quantification puis une déquantification selon la méthode zerotree sans prendre en considération la ROI (en gardant seuil=4), la reconstruction des coefficients donnera la matrice suivante (on ne considère ici que le passage principal de la quantification) : $\begin{bmatrix} 20 & 4 \\ 0 & 0 \end{bmatrix}$.

Si on applique la méthode telle qu'elle est décrite , on aura :

Seuil=4=$2^2$, donc on effectue un décalage binaire à gauche de 2 sur les coefficients les plus importants, d'ou les coefficients de la matrice considérée deviennent $\begin{bmatrix} 22 & 20 \\ 3 & 1 \end{bmatrix}$, le coefficient '5' étant considéré le plus important, il a été décalé de 2 bits , autrement (multiplié par la valeur du seuil). La quantification puis déquantification de cette dernière matrice donne $\begin{bmatrix} 20 & 20 \\ 0 & 0 \end{bmatrix}$ . En décalant le coefficient important à dROIte

(toujours selon la valeur du seuil), on obtient $\begin{bmatrix} 20 & 5 \\ 0 & 0 \end{bmatrix}$, d'où, la restitution du coefficient '5' qu'on a considéré comme ROI dans cet exemple.

## 5.6 Résultats et discussion de la méthode 'Zerotree Local'

Les méthodes proposées ont étés appliquées à l'image 'woman', une image 128x128 à 256 niveaux de gris.

Ci dessous, les résultats de la première méthode : Zerotree local , les sous images considérées sont respectivement de dimension 8x8, 16x16 et 32x32. Ces tROIs paramètres s'avèrent l'idéal pour cette méthode vu qu'au delà de 32x32 la sous image devient volumineuse, et n'est plus exploitable en région d'intérêt. Le tableaux IV, V et VI représentent respectivement les résultats obtenus pour des sous images de 8x8, 16x16 et 32x32, les figures 5.9, 5.10, 5.11, 5.12 et 5.13), représentent l'image source et différentes images (selon différents seuils) issues de cette compression pour des sous images 16x16. Le critère d'évaluation utilisé pour la région non intéressante est subjectif vu que c'est surtout la reconstruction parfaite de la ROI qui est visée.

TABLEAU IV

Résultats pour sous-image de 8x8

| Seuil | Taille (Kbits) | Nombre de symboles ($x10^3$) | Nombre de '0'($x10^3$) | Taille de ROI (pixel/pixel) | Qualité visuelle de la région hors ROI |
|---|---|---|---|---|---|
| 64 | 21.5 | 11.5 | 14.5 | 24/24 | Moyenne |
| 32 | 36.5 | 20.5 | 12.4 | 24/24 | Assez.bonne |
| 16 | 58 | 32 | 9.3 | 24/24 | Bonne |
| 8 | 84 | 47 | 6 | 24/24 | Très bonne |

TABLEAU V

Résultats pour sous-image de 16x16

| Seuil | Taille (Kbits) | Nombre de symboles($x10^3$) | Nombre de '0'($x10^3$) | Taille de ROI (pixel/pixel) | Qualité visuelle de la région hors ROI |
|---|---|---|---|---|---|
| 64 | 19.5 | 12 | 14 | 32/32 | Moyenne |
| 32 | 33 | 20.5 | 12 | 32/32 | Assez.bonne |
| 16 | 58 | 31 | 9.5 | 32/32 | Bonne |
| 8 | 84 | 45 | 6.5 | 32/32 | Très bonne |

TABLEAU VI

Résultats pour sous-image de 32x32

| Seuil | Taille (Kbits) | Nombre de symboles ($x10^3$) | Nombre de '0'($x10^3$) | Taille de ROI (pixel/pixel) | Qualité visuelle de la région hors ROI |
|---|---|---|---|---|---|
| 64 | 44 | 26 | 11.6 | 64/64 | Moyenne |
| 32 | 55.1 | 32.8 | 10.1 | 64/64 | Assez.bonne |
| 16 | 70 | 41.5 | 8 | 64/64 | Bonne |
| 8 | 90 | 52 | 5.5 | 64/64 | Très bonne |

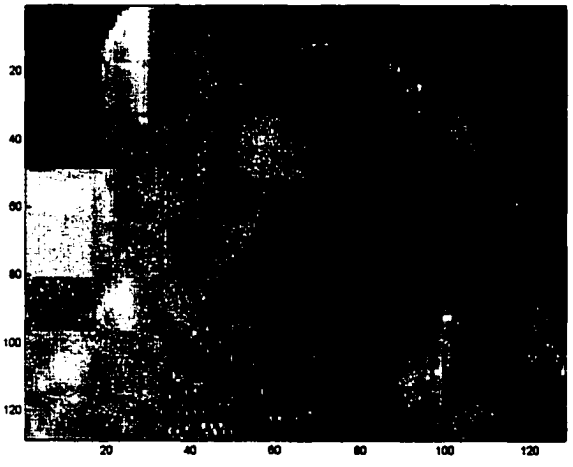


Figure 5.9 L'image 'woman' compressée par ZL à un seuil de 64

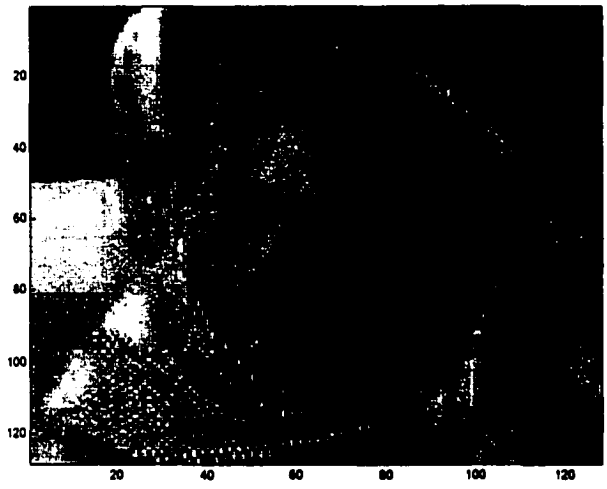


Figure 5.10 Image 'woman' compressée par ZL à un seuil de 32

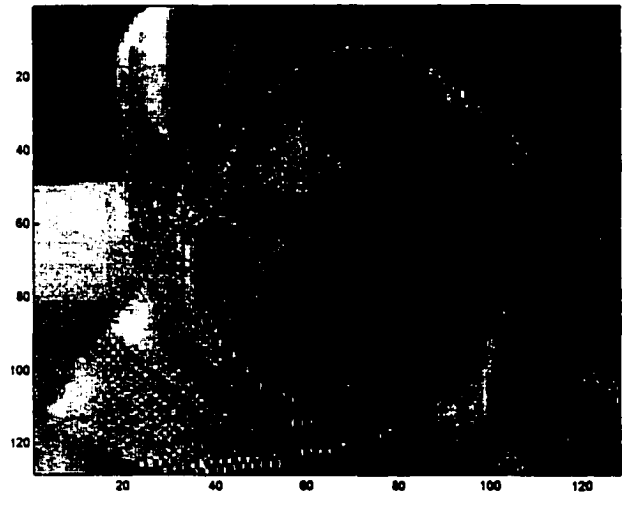


Figure 5.11 l'image 'woman' compressée par ZL à un seuil de 16

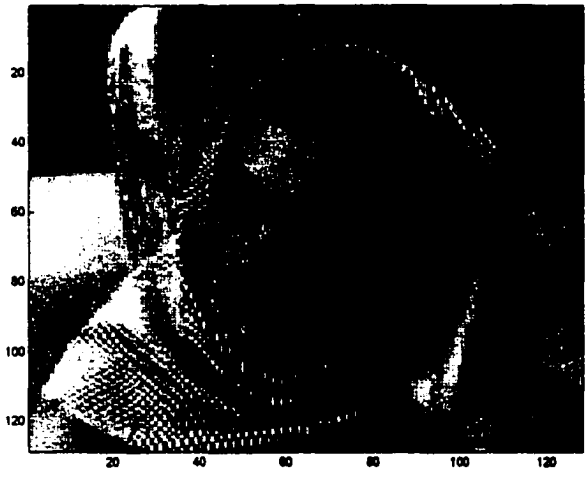


Figure 5.12 l'image 'woman' compressée par ZL à un seuil de 8

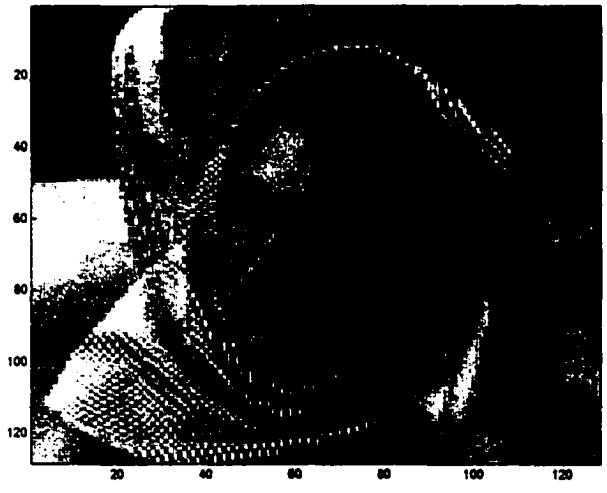


Figure 5.13 l'image originale 'woman'

Sur les tests effectués sur des sous images de 8x8, 16x16 et 32x32, chacune présente des avantages et des inconvénients. Pour éviter l'ambiguïté, on considère deux types de ROI Le premier type est celui choisi par l'utilisateur en délimitant par la souris la région sur laquelle l'étude se focalise, on nomme cette région $ROI_u$. Le deuxième type est la région considérée par l'algorithme $ROI_s$, la dimension de cette dernière sera ,bien sur, multiple de la taille de sous-image considérée.

Au niveau de la précision, il est évident que les fenêtres de dimensions 8x8 seront plus proches que les autres suppositions (16x16 et 32x32) de la région d'intérêt choisie, on peut d'ailleurs le remarquer dans les tableaux IV, V et VI , où pour une même région (choisie arbitrairement dans l'image test), la première supposition considère une $ROI_a$ de 24x24, la deuxième prend quand à elle une $ROI_s$ de 32x32, tandis que la tROIsième considère une $ROI_s$ de 64x64. Certes, les dimensions obtenues ici, peuvent représenter une des possibilités, car on peut aussi avoir, selon certains choix de $ROI_u$ , des $ROI_s$ qui seront de même dimension pour deux voire les tROIs tailles de sous-images considérées, cependant, tant que la dimension de la sous-image considérée est la plus petite, on est sur que $ROI_a$ délimite avec les moindre pertes la $ROI_u$ .

Si on se fie à cette première analyse, on aura tendance à choisir des sous images de 1x1 (le minimum possible) pour ne pas compresser des parties inutiles de l'image sans pertes 'gratuitement', cependant, il faut noter que pour chaque sous image (ou bloc) codée correspond un flot de bits 'entête' qui désigne le seuil maximal utilisé pour la quantification ($\log_2$(seuil_max) suffit pour retrouver le seuil à la reconstruction vu que le seuil est une puissance de 2), ainsi que le nombre de symboles obtenu après quantification, ce qui constitue en moyenne vingt bits supplémentaires au code de chaque bloc. Ce supplément au niveau de toute l'image devient volumineux si les sous images considérées sont de petites tailles, le tableau VII illustre le nombre de bits supplémentaires au code final pour les sous images considérés.

Tableau VII

Suppléments de code pour une image 128x128

| Taille de sous images | Nombre de sous images | Bits supplémentaires |
|---|---|---|
| 8x8 | 256 | 5120 |
| 16x16 | 64 | 1280 |
| 32x32 | 16 | 320 |

Le tableau 5.4 illustre le gain qu'on peut avoir en passant d'une taille de bloc à une autre supérieure, ceci pour une image 128x128 (vu que c'est la dimension de l'image test), mais sachant que les images médicales sont de plus grande résolution, ce supplément et encore plus volumineux pour ce genre d'images.

Pour ce qui est du taux de compression, il est difficile de juger sur les images comprimées vu que la région $ROI_a$ n'est pas la même pour les tROIs suppositions et qu'aucun critère n'est défini pour accepter la région sans pertes, on peut dire qu'une simple silhouette suffit pour déterminer l'emplacement de la ROI , ou bien qu'il nous faut une représentation avec une assez bonne qualité pour avoir une continuité dans l'image. C'est à dire que la région avec pertes peut être compressée à 30 :1 ou 40 :1 comme elle peut l'être à 5 :1.

Donc, pour optimiser l'utilisation de cette méthode, il faut trouver une solution au dilemme 'précision vs entête ', d'après les tests illustrés par les tableaux IV,V,VI, les sous images de 16x16 donnent de meilleurs résultats que les autres blocs dans le cas de cette image.

## 5.7 Résultats et discussion de la méthode 'Zerotree global'

Cette méthode a été appliquée sur la même image test 'woman', la région d'intérêt choisie est de 40x40 pixels. Le masque de la ROI a été calculé de deux manières (comme expliqué dans paragraphe 5.4) nommées mask1 et mask2. Dans le codage zerotree, vu le nombre de symbole 'R' (zéro), on a utilisé une modélisation en ajoutant un symbole 'S' qui correspond a une suite n de 'R', il est vrai que cette modélisation donne moins de symbole après quantification, cependant il faut choisir le n idéal pour ne pas alourdir la tache du codeur arithmétique qui devra gérer cinq symboles au lieu de quatre (autrement on optimise la valeur : entropie x nombre_de_symboles ) . Les figures 5.13, 5.14, 5.15 illustrent l'application de cet algorithme pour des seuils d'arrêt respectifs de 128, 64, 32, ces images ont été obtenues avec mask2.

Figure 5.14 Image 'woman' compressée par ZG pour un seuil de 128

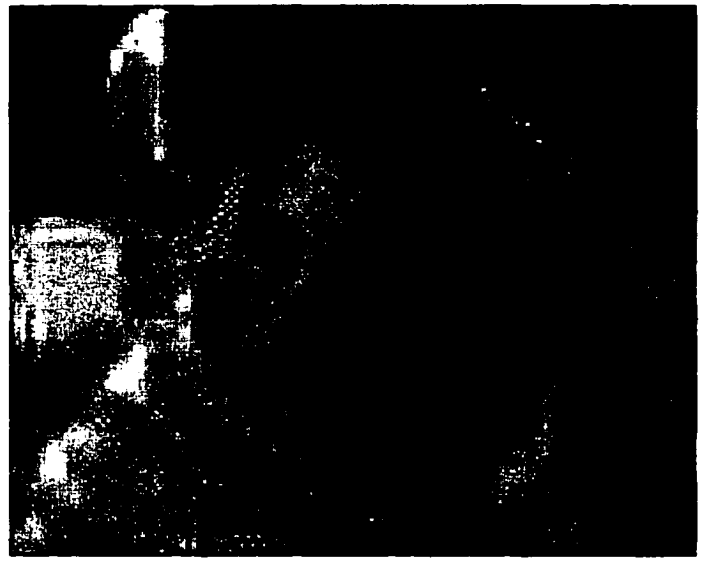

Figure 5.15 Image 'woman' compressée par ZG pour un seuil de 64

Figure 5.16 Image 'woman' compressée par ZG pour un seuil de 32

Tableau VIII

Résultats obtenus avec mask1

| Seuil | Taille ($x10^3$ bits) | Nombre de symboles ($x10^3$) | Nombre de '0' ($x10^3$) | Taille de ROI (pixel/pixel) | Qualité visuelle de la région hors ROI |
|---|---|---|---|---|---|
| 128 | 25.5 | 21 | 14.9 | 40/40 | Moyenne |
| 64 | 29.7 | 24.1 | 17 | 40/40 | Assez.bonne |
| 32 | 40.5 | 30.5 | 21 | 40/40 | Bonne |
| 16 | 56.2 | 39.2 | 25.7 | 40/40 | Très bonne |

Tableau IX

Résultats obtenus avec mask2

| Seuil | Taille ($x10^3$ bits) | Nombre de symboles ($x10^3$) | Nombre de '0' ($x10^3$) | Taille de ROI (pixel/pixel) | Qualité visuelle de la région hors ROI |
|---|---|---|---|---|---|
| 128 | 29.4 | 23.7 | 16.5 | 40/40 | Moyenne |
| 64 | 34 | 26.8 | 18.7 | 40/40 | Assez.bonne |
| 32 | 44 | 33 | 22 | 40/40 | Bonne |
| 16 | 59.2 | 41 | 27 | 40/40 | Très bonne |

Les résultats obtenus montrent qu'avec mask1 (tableau VIII), on obtient une meilleure compression qu'avec mask2 (tableau IX) si on utilise les deux masques dans le même contexte, aussi, au niveau de la qualité visuelle de l'image , on obtient le même résultat, on ne décèle pas de différence à l'œil nu. Cependant, il existe une différence au niveau des bords de la ROI de l'image résultante où l'utilisation de 'mask1' ne permet pas de reconstruire exactement cette partie de la ROI, ce qui n'est pas le cas pour mask2 qui n'a pas de pertes au niveau de la ROI Donc l'utilisation de 'mask1' dans l'algorithme peut causer des problèmes de précision si le choix des régions d'intérêt est très minutieux.

## 5.8 Résultats sur des images médicales

Les images médicales utilisées sont :

- Image 'Cerveau ' : c'est une image de 256x256 pixels, nous représentons ici les résultats obtenus en lui appliquant le ZG avec des seuils d'arrêt respectifs de 32,128,256 et 512. La ROI considérée est de 50x80 pixels (figures 5.17 à 5.22).

- Image 'poumons' : c'est une image de 512x512 pixels, nous représentons ici les résultats obtenus en lui appliquant le ZL considérant des sous-images de 16x16 , et des seuils d'arrêt respectifs de 1024, 256 et 64. La ROI considérée est de 130/150 pixels (figures 5.23 À 5.28).


Figure 5.17 Image 'Cerveau' originale

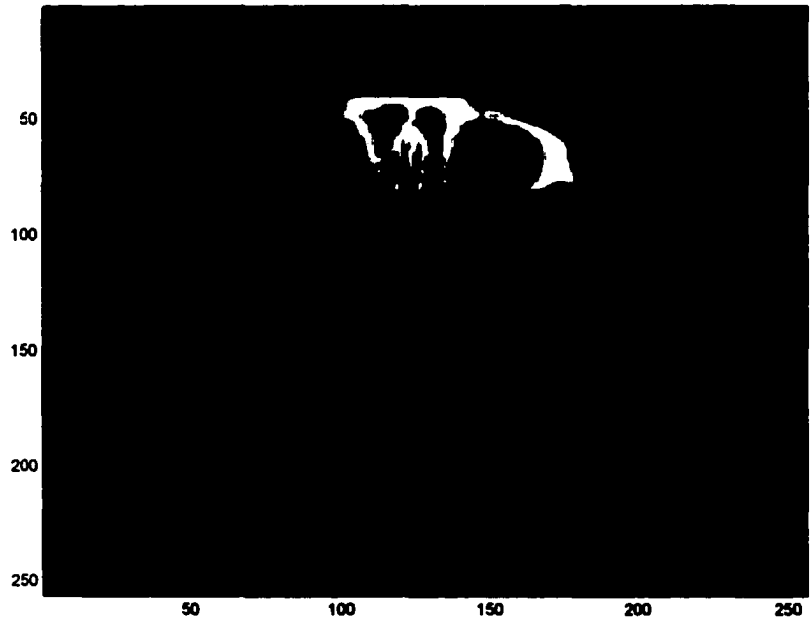


Figure 5.18 ROI considérée


Figure 5.19 Image reconstituée pour un seuil de 32 avec ZG

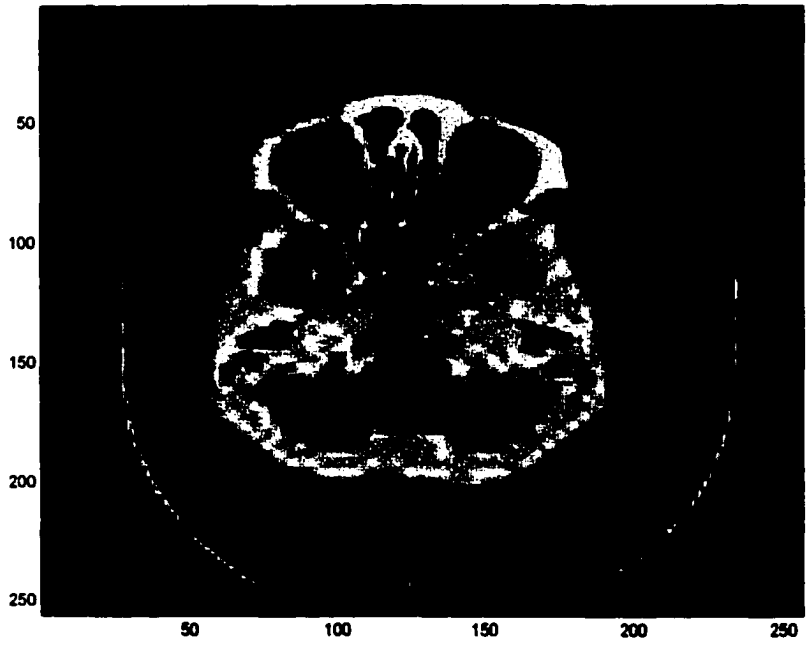


Figure 5. 20 Image reconstituée pour un seuil de 128 avec ZG

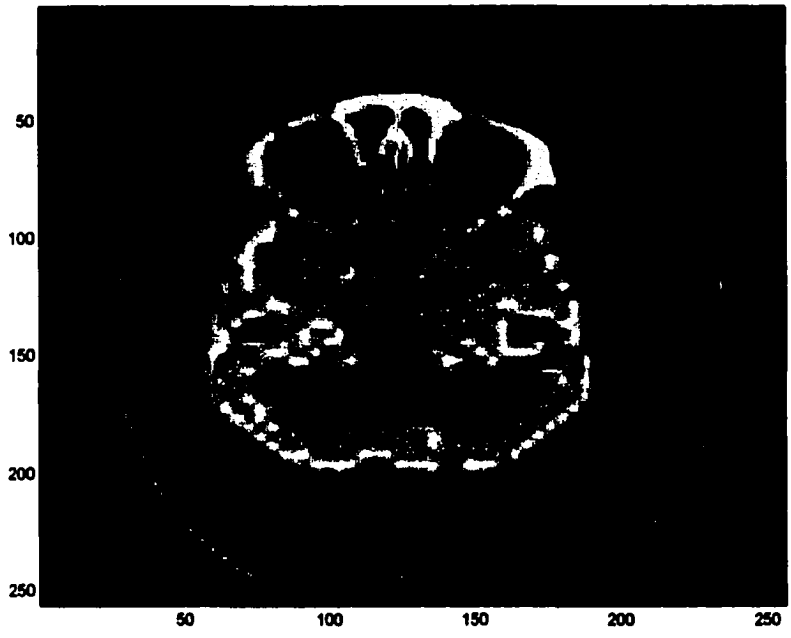


Figure 5. 21 Image reconstituée pour un seuil de 256 avec ZG

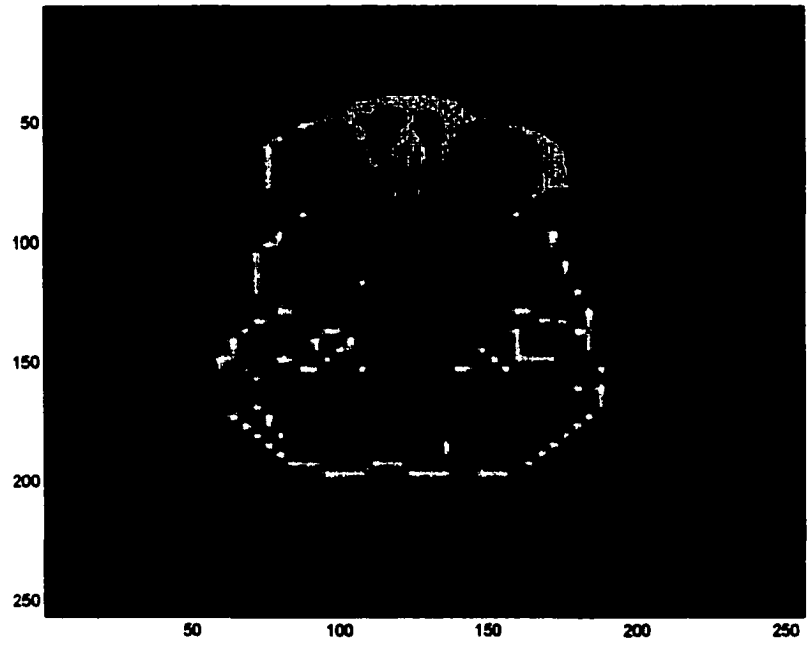


Figure 5. 22 Image reconstituée pour un seuil de 512 avec ZG

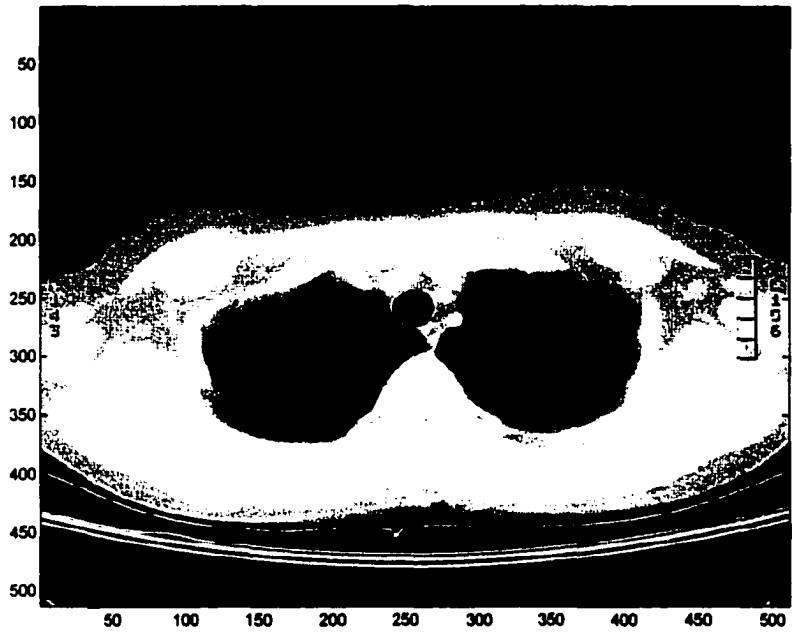


Figure 5. 23 Image 'Poumons' originale

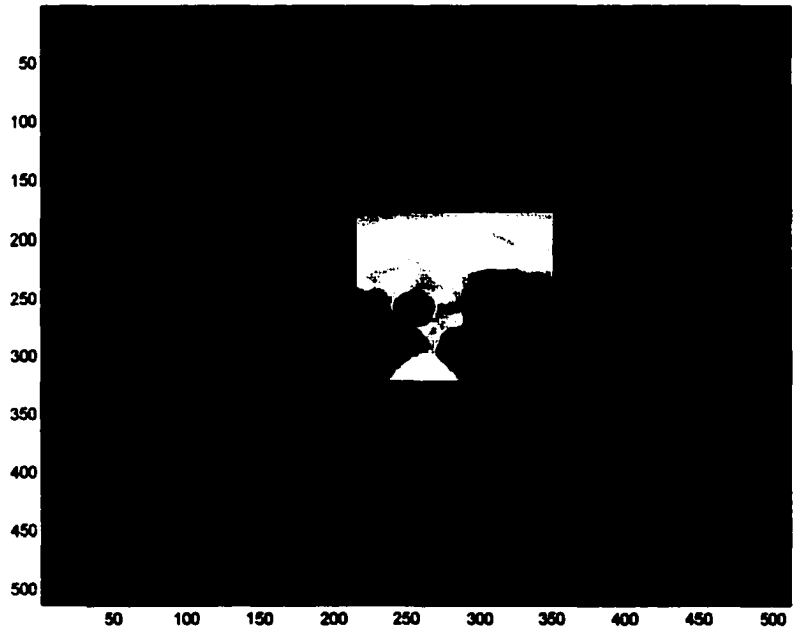


Figure 5. 24 ROI considerée

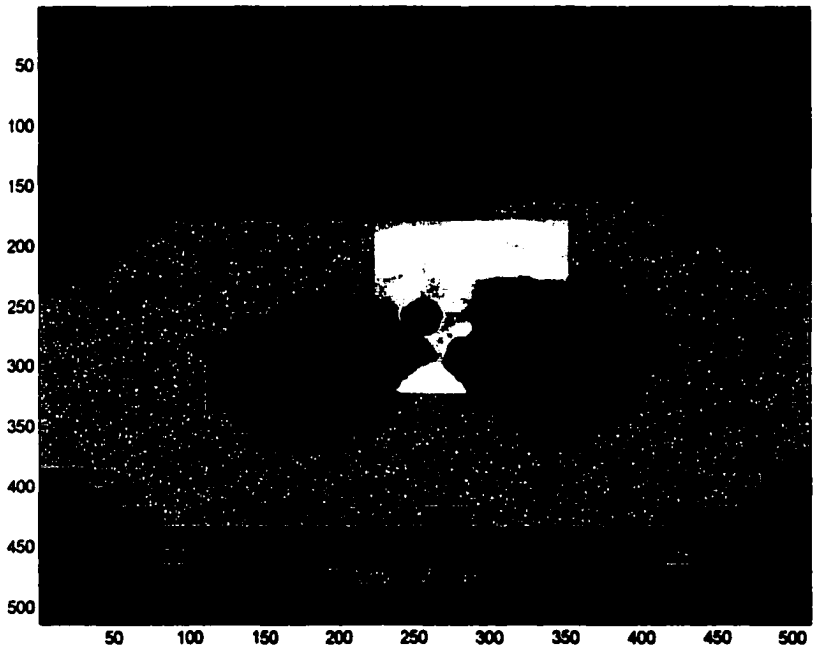


Figure 5. 25 Image reconstituée pour un seuil de 1024 avec ZL

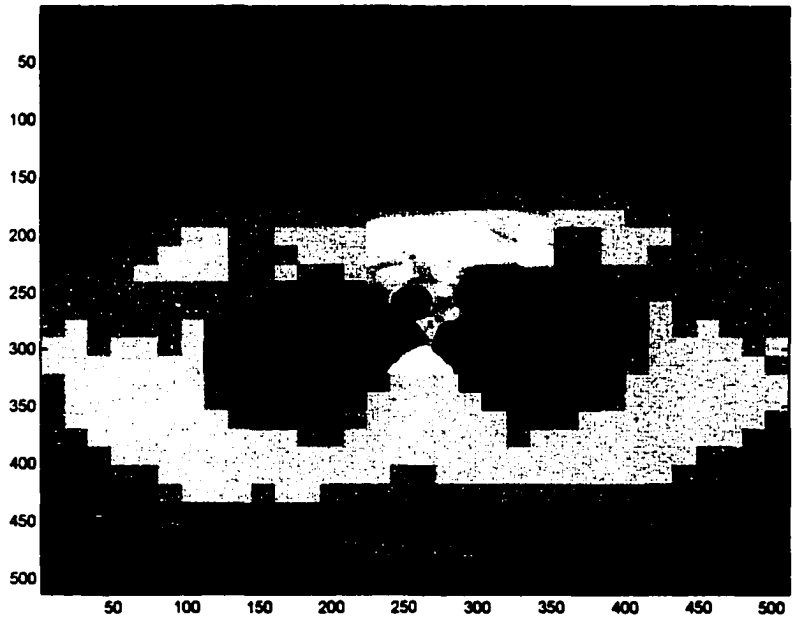


Figure 5. 26 Image reconstituée pour un seuil de 512 avec ZL

Figure 5. 27 Image reconstituée pour un seuil de 256 avec ZL

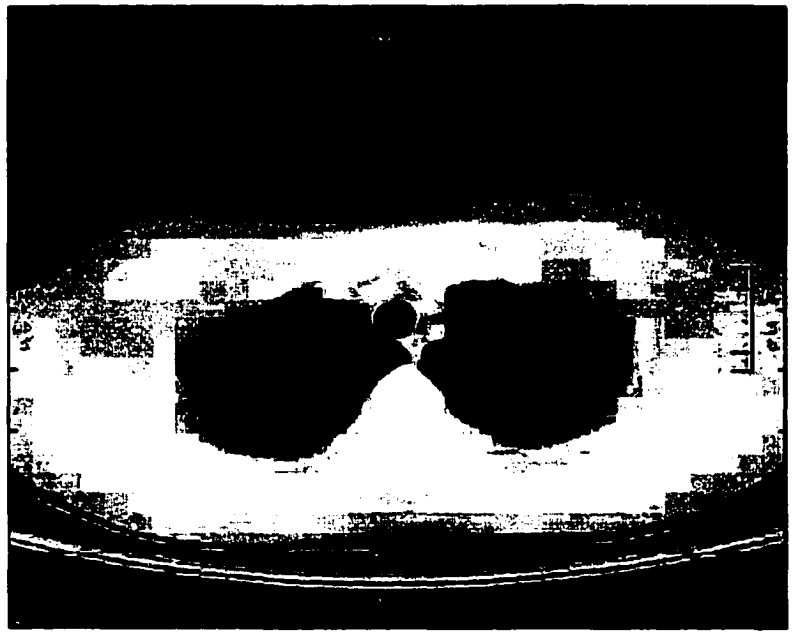

Figure 5. 28 Image reconstituée pour un seuil de 64 avec ZL

## 5.9 Comparaison entre ZL et ZG

Chacune des deux méthodes proposées a des avantages et inconvénients, qui peuvent être classés selon les critères suivants.

- La précision dans la reconstruction de la ROI; Il est évident que la méthode ZG est plus précise que ZL car elle permet de reconstruire la région choisie par l'utilisateur telle que définie au départ, contrairement au ZL où l'on reconstruit forcément des parties en plus sans pertes inutilement. Par exemple, une région d'intérêt de dimension 20x20, va être reconstruite selon ZG avec la même dimension, tandis qu'en utilisant ZL, la région qui sera reconstruite sans pertes peut atteindre 64x64 (en utilisant des blocs de 32x32, et que la ROI touche quatre blocs ), soit 3096 pixels seront codés sans pertes tout en appartenant à une région non intéressante.

- Le taux de compression ou la taille de l'image compressée ; il dépend de la précision de la reconstruction de la ROI , le seuil de quantification choisi et aussi la dimension des sous images dans le cas du ZL . Pour le premier et le troisième critère cités, le ZG est avantageux par rapport au ZL sachant qu'il est précis dans la reconstruction de la ROI et qu'il ne fait pas appel a une segmentation qui demande au ZL vingt bits par bloc d'entête. Cependant le deuxième critère est à l'avantage du ZL, vu qu'il démarre la quantification au seuil normal contrairement au ZG qui commence par un seuil décalé de $\log_2$ (seuil_min) bits et que les coefficients dans les sous bandes inférieures sont assez grands, ce qui génère des symboles de plus.

# CONCLUSION

Dans plusieurs contextes, la compression d'images par région d'intérêt peut s'appliquer, on peut citer à titre d'exemple la vidéotélephonie où l'on peut exiger que les visages soient plus précis que les autres régions, ou les images provenant de satellites qui sont tellement grandes qu'une compression avec beaucoup de pertes peut donner des résultats qu'on ne peut exploiter.

Les professionnels du domaine médical et paramédical ainsi que des chercheurs en imagerie insistent sur le fait qu'une image médicale de bonne qualité est totalement sans pertes. Dans ce but, on propose des méthodes de compression qui agissent selon les caractéristiques de l'image. Les régions les plus importantes sont compressées sans pertes et sont mises en évidence (si l'on compresse à un taux très élevé). Ce dernier avantage est aussi important que le premier cité car il n'est pas facile pour un médecin de se rappeler exactement des micro fissures subies par ses patients, et permet un gain considérable d'espace de stockage .

Deux méthodes de compression par ondelettes par région d'intérêt ont été étudiées dans le cadre de ce travail. La première, ZL, est basée sur une application d'un codeur de type 'Zerotree' sur une image segmentée. Elle agit, selon la région, sans pertes dans les sous images touchées par la ROI et avec pertes partout ailleurs (l'utilisateur peut définir le degré de pertes tolérées). Les résultats de simulation obtenus au niveau de la mise en évidence de la ROI aussi bien à celui de l'exactitude de la partie reconstruite sans pertes présentent un bon compromis entre l'information contenue dans l'image compressée et le taux de compression de l'image obtenue. La deuxième méthode

proposée ZG, qui est aussi à base de zerotree, traite l'image au complet en privilégiant les coefficients d'ondelette appartenant à la ROI. Ce traitement est effectué par le biais de masques issus de la représentation binaire de la ROI : le premier est obtenu par des décimations sur cette image binaire, le deuxième n'est autre que la transformée en ondelette de cette représentation . Cette méthode a l'avantage d'une grande précision de reconstruction des frontières de la ROI.

Par ailleurs, chacune de ces deux méthodes présentent des avantages et inconvénients qui lui sont propres. Le choix de l'une ou de l'autre dépend de la région d'intérêt considérée et du seuil de compression choisi.

Cependant, ce type de compression ouvre plusieurs portes, et quelques suggestions de travaux de recherche futures peuvent être effectués parmi lesquelles:

- L'utilisation du SPIHT (ou même d'autres méthodes à base du zerotree) : au niveau de la quantification. Cette méthode, qui est basée sur le zerotree donne d'assez bons résultats qui peuvent être comparés au méthodes utilisées dans cette recherche (en l'utilisant dans le même contexte).

- La Modélisation du codage arithmétique, par l'estimation des probabilité d'occurrence des symboles et des chaînes de symboles (par exemple, après une suite de 'RRR' un autre symbole 'R' est le plus probable dans le codage zerotree).

- L'utilisation du lifting schème pour la transformée en ondelettes[34-35], cette transformée permet l'obtention de coefficients d'ondelettes entiers.

- L'utilisation des algorithmes proposés pour pouvoir gérer plusieurs régions d'intérêts dans une même image, comme par exemple une radio contenant une

double fractures de cotes différentes . Ceci en donnant plusieurs choix de ROI à l'utilisateur, le ZL gérera automatiquement les ROI choisies ,cependant pour le ZG, il faut créer, pour chaque ROI un masque et faire le 'OU logique' de tous les masques obtenus.

- L'optimisation du temps d'exécution des algorithmes proposés en les codant en C++ ou Java.

## BIBLIOGRAPHIE


1. Gonzalez, Rafael C., and Richard E. Woods (1997), *Digital Image Processing*: Addison-Wesley.

2. Guillois, Jean Paul, (1996), *Techniques de compression des images*: Paris, Hermès.

3. Moreau, Nicolas, (1994) , *Techniques de compression des signaux* : Masson, Paris et CNET ENST.

4. Sayood, K. (1996) *Introduction to data compression*: Morgan Kaufman Publishers .

5. Netravali, A., Haskel, B., (1988) *Digital pictures : Representation and compression.* New York : plenum.

6. Weeks, Arthur R. (1996), *Fundamentals of electronic image processing.* New York: IEEE press.

7. Meyer, Y (1990) *Ondelette, in ondelettes et opérateurs*, Hermann.

8. Antonini, M., Barlaud, M., Mathieu, P. et Daubechies, I.(1992), Image coding using wavelet tranform. *IEEE Trans. on image processing*,1(2), pp.205-220.

9. Daubechies, I. (1992), *Ten lectures on wavelets*, Society for industrial and applied mathematics, Philadelphia, Pennsylvania.

10. Mallat, S (1989), A theory for multiresolution signal decomposition : the wavelet representation. *IEEE Transaction on Pattern analysis and machine intelligence*, 11,pp.674-693.

11. Lemarié, P.G.(1988), Ondelettes à localisation exponentielle. *Journal de mathematiques pures et appl.*, 67(3) pp227-2360.

12. Grossmann, A., Morlet, J. (1987) Math & Phys., *Lectures on recent results*, L.streit, World scientific.

13. Strang G., Nguyen , T., (1996) *Wavelets and filter banks* ,Wellesley Cambridge Press, Wellesley, MA.

14. Chan, Y.T., (1995), *wavelet basics* : Kluwert academic publishers.

15. Vetterli, M. Kovacevic, E. (1995), *Wavelets and filter banks*, Englewood Cliffs, NJ Prentice Hall.

16. *Home site of the JPEG and JBIG* 'http:// www.jpeg.org'.

17. Shapiro, J.M.(1993), Embedded image coding using zerotrees of wavelet coefficients, *IEEE Trans. On signal processing*, 41(12), pp.3445-3456.

18. Said, A., Pearlman, W. (1996) New, fast and efficient image codec based on set partitioning hierarchical trees, *IEEE Trans. Circ.& Sys. Video Tech*, vol.6 no.3, pp. 243-249.

19. Said, A., Pearlman ,W.(1996), An image multiresolution representation for lossless and lossy compression. *IEEE Trans. On Image Proc.*5(9) :1303-1310.

20. Hsung ,T., Chan,T., Lun, P., Feng, D. (1999), Embedded singularity detection zerotree wavelet coding, *IEEE Int. Conf. On image processing*, pp:274-278 vol. 2.

*21.* Hisakazu, K., Shigenobo, S., Jaeho, S., (1998), A flexible zerotree coding with low entropy, *International Technical Conference on Circuits / Systems, Computers and Communications (ITC-CSCC '98)* Vol.E82-A No.6 p.1117.

22. Zhong, J., Leung, C.H. and Tang, Y.Y. (1998), An improved zerotree wavelet image coder based on significance checking in wavelet trees. *Proceedings of IEEE International Conference on Systems*, Man, and Cybernetics, San Diego, USA, 5: 4567-4571.

23. Shapiro, J.M. (1995) A fast technique for identifying zerotrees in the ezw algorithm. P*roceedings ICASSP-95 (IEEE International Conference on Acoustics, Speech and Signal Processing)*, volume~3, pages 1455-1458.

24. Witten, I.H., Moffat, A., and Bell, T.C. (1999) *Managing gigabytes: compressing and indexing documents and images.*(second edition) Morgan Kaufmann, San Francisco, CA. no. 6 .

25. Witten, I. H., Neal, R. M., and Cleary, J. G. (1987), Arithmetic coding for data compression, *Communications of the ACM*, vol. 30, no. 6 .

26. Rissanen, J.J., and Langdon, G., (1979), Arithmetic Coding, *IBM J. Res. Develpo.*, 23, no. 2, pp.156-162.

27. Howard, P.G., and Vitter, J.S., (1992)*Practical Implementations of Arithmetic Coding, in Image and Text Compression,* MA: Kluwer Academic Publishers, pp. 85--112, 1992.

28. Howard and Vitter, (1994) , Arithmetic coding for data compression, *Proceedings of the IEEE*, vol.82, no.6.

29. Moffat, A., Neal, R. M., and Witten, I. H. (1995), Arithmetic coding revisited, *ACM Transactions on Information Systems*, vol. 16, pp. 256-294.

30. Mandal, M.K. , Panchanathan, S. et Aboulnasr, T.(1996), Choice of Wavelets for Image Compression, *Lecture Notes in Computer Science*, Vol. 1133, pp. 239-249.

31. Topiwala, P.N., and Tran, T.D., (1999) Local zerotree coding, *IEEE Int. Conf. on Image Processing*, Kobe, Japon.

32. Nister D. and Christopoulos C., (1998), Lossless Region of Interest with a naturally progressive still image coding algorithm, *Proceedings of IEEE International Conference on Image Processing (ICIP 98),* pp. 856-860, 4-7, Chicago, Illinois.

33. Nister, D. and Christopoulos, C., (1999), Lossless region of interest coding, *Signal Processing*, Vol. 78, No. 1, pp. 1-17.

*34.* Calderbank,A., Daubechies,I., Sweldens,W., Yeo,B,(1997) wavelet transform that map integers to *integers, Proceedings of the IEEE Conf. on Image Processing.*

35. Daubechies, I. and Sweldens, W., (1998) factoring wavelet transform into lifting steps, *J. Fourier Anal. Appl.*, Vol. 4, Nr. 3.

# ANNEXE A

# PROGRAMME SOURCE

```
function matrice = script16


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%  script16
%%
%%         cette fonction est le script de la méthode Zerotree Local
%%         elle fait le lien entre les différentes fonctions, elle
%%         s'occupe de la définition de la région d'intérêt choisie
%%         par l'utilisateur, et transmettre les paramètres
%%         aux modules de transformée, quantification et codage.
%%         L'application présentée a été effectuée sur l'image 'woman'
%%         pour des sous images de 16x16.
%%
%%         auteur: Monsef Mekouar
%%
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


load woman2;
I=ind2gray(X,map);
L=length(X);
b=0;
b2=0;
bloc=L/16;
seuil=32
matrice_de_roi=zeros(bloc);

[x1,x2]=region(I,16); % Selection de la ROI par l'utilisateur

x1i=floor(x1/bloc)+1; % Détermination de la ROI
x1j=mod(x1,bloc)+1;   % de traitement
x2i=floor(x2/bloc)+1; % ces 4 points représentent l'emplacement de la
x2j=mod(x2,bloc)+1;   %  ROI dans la matrice des blocs(ou sous images)

%% Création du masque

for(i=x1i:x2i)
   for(j=x1j:x2j)
      matrice_de_roi(i,j)=1
   end
end

mask=zeros(128);

for(i=1:8)
   for(j=1:8)
      if (matrice_de_roi(i,j)==1)
        for(i2= 16*(i-1)+1 : 16*(i))
          for(j2=16*(j-1)+1:16*(j))
            mask(i2,j2)=1;
          end
```

```
      end
     end
    end
end

figure
imagesc(mask.*X)   % affichage de l'image masquée
colormap(gray)
pause

a=ondelette16(X);

%% Traitement de la région avec pertes

for(i=1:bloc)
   for(j=1:bloc)
      if(matrice_de_roi(i,j)==0)

      % Codage

         im=a(1+16*(i-1):16*i,1+16*(j-1):16*j);
       [code_zerotree,puissance]=par16(im,seuil);
         code_bin=arith16(code_zerotree,puissance);

       % Décodage

         b2=b2+length(code_bin);
         b=b+length(code_zerotree);
         code_zerotree2=decode_arith16(code_bin);
       [code_zerotree2,puissance2]=decode_arith16(code_bin);
         m1=dec16(code_zerotree,puissance,seuil);
       a2(1+16*(i-1):16*i,1+16*(j-1):16*j)=m1;

      end
    end
 end

 %% Traitement de la région sans pertes

 for(i=x1i:x2i)
   for(j=x1j:x2j)

     % Codage

       im=a(1+16*(i-1):16*i,1+16*(j-1):16*j);
         [code_zerotree,puissance]=par16(im,1);
         code_bin=arith16(code_zerotree,puissance);

       % Décodage

         b2=b2+length(code_bin);
        b=b+length(code_zerotree);
         code_zerotree2=decode_arith16(code_bin);
      [code_zerotree2,puissance2]=decode_arith16(code_bin);
```

```
          m1=dec16(code_zerotree,puissance,1);
            a2(1+16*(i-1):16*i,1+16*(j-1):16*j)=m1;

   end
 end

%% Affichage de nombre de zéros dans le code 'Zerotree'

   b3=length(find(a2==0))

%% Transformée inverse

   compose16(a2);

%% Résultats obtenus b2: b3:  b:
%% b2 : Taille de l'image compressée en bits
%% b3 : nombre de zéros dans le code zerotree
%% b  : nombre de symboles du code zerotree

b2
b
b3
```

```
function [x1,x2]=region(a,taille)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%  [x1,x2]=region(a,taille)
%%
%%         cette fonction s'occupe de l'acquisition de la région
%%         d'intérêt utilisateur x et la transformer en région
%%         d'intérêt système selon la taille de sous image considéré.
%%         Valable juste pour le ZL.
%%
%%         paramètres d'entrée :
%%              a      : Image étudiée
%%              taille : Taille de sous image considérée
%%
%%         parametres de sortie:
%%              x1     : Pointeur sur le premier bloc de la ROI systeme
%%              x2     : Pointeur sur le dernier bloc de la ROI systeme
%%
%%         auteur: Monsef Mekouar
%%
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

fact=length(a)/taille
imagesc(a)
colormap(gray)
x=getrect; % choix de la ROI utilisateur

% Calcul des bornes de la ROI systeme

x1=floor(x(1)/taille)+floor(x(2)/taille)*fact
x2=floor((x(1)+x(3))/taille)+floor((x(2)+x(4))/taille)*fact
```

```
function res=ondelette16(X)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%  ondelette(X)
%%
%%          cette fonction effectue la transformée en ondelette 2D
%%          l'image représentée par la matrice X, le base d'ondelettes
%%          utilisé est Daubechies, et la décomposition est faite sur
%%          des sous images de dimension 'si x si'. Elle retourne une
%%          matrice de coefficients res de la meme dimension que l'image
%%          originale
%%
%%          auteur: Monsef Mekouar
%%
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


%%
%%   La matrice résultante de chaque sous image est organisée ainsi:
%%
%%
%%             ------------------------------------
%%            |         |         |                |
%%            |  ca2    |  ch2    |                |
%%            |         |         |                |
%%             -----------------         ch1
%%            |         |         |                |
%%            |  cv2    |  cd2    |                |
%%            |         |         |                |
%%             ------------------------------------
%%            |                   |                |
%%            |                   |                |
%%            |        cv1        |       cd1      |
%%            |                   |                |
%%            |                   |                |
%%            |                   |                |
%%            -------------------------------------
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Initialisations%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

filtre='db3';
dec=3;
si=16 % taille de sous_image;
nb=ceil(length(X)/si)
dwtmode('ppd');

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%  Traitement  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%


for(i=1:nb)
   for(j=1:nb)
```

```
            % extraction de la sous image d'étude

            x=X(1+(i-1)*si:i*si,1+(j-1)*si:j*si);

            % Décomposition

            [ca1,ch1,cv1,cd1] = dwt2(x,filtre);

            ca1=ca1(dec:length(ca1),dec:length(ca1));
            ch1=ch1(dec:length(ch1),dec:length(ch1));
            cv1=cv1(dec:length(cv1),dec:length(cv1));
            cd1=cd1(dec:length(cd1),dec:length(cd1));

            [ca2,ch2,cv2,cd2] = dwt2(ca1,filtre);

            ca2=ca2(dec:length(ca2),dec:length(ca2));
            ch2=ch2(dec:length(ch2),dec:length(ch2));
            cv2=cv2(dec:length(cv2),dec:length(cv2));
            cd2=cd2(dec:length(cd2),dec:length(cd2));

            [ca3,ch3,cv3,cd3] = dwt2(ca2,filtre);

            ca3=ca3(dec:length(ca3),dec:length(ca3));
            ch3=ch3(dec:length(ch3),dec:length(ch3));
            cv3=cv3(dec:length(cv3),dec:length(cv3));
            cd3=cd3(dec:length(cd3),dec:length(cd3));

            [ca4,ch4,cv4,cd4] = dwt2(ca3,filtre);

            ca4=ca4(dec:length(ca4),dec:length(ca4))
            ch4=ch4(dec:length(ch4),dec:length(ch4))
            cv4=cv4(dec:length(cv4),dec:length(cv4))
            cd4=cd4(dec:length(cd4),dec:length(cd4))

            clear ca1;
            clear ca2;
            clear ca3;

            ca3=[ca4,ch4;cv4,cd4]
            ca2=[ca3,ch3;cv3,cd3];
            ca1=[ca2,ch2;cv2,cd2];

            r=[ca1,ch1;cv1,cd1];

            % organisation de la matrice des coefficients :
            % chaque sous image est remplacée par la matrice
            % des coefficients issus de sa transformée

            X2(1+(i-1)*si:i*si,1+(j-1)*si:j*si)=r ;

      end
   end
```

```
function [resf,puissance]= par16(im,seuil_final)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%  [resf,puissance]= par16(im,seuil_final)
%%
%%         cette fonction s'occupe de l'organisation et quantification
%%         des coefficients issus de la transformée en ondelettes, elle
%%         effectue le codage zerotree. Outre le resultat du codage,
%%         elle donne en sortie aussile seuil de départ de
%%         quantification qui sera utile lors de la déquantification
%%
%%         Parametres d'entrée
%%               im          : matrice de coefficients
%%               seuil_final : Seuil d'arret de quantification
%%
%%         Parametres de sortie
%%               resf        : Liste des symboles obtenus par le EZW
%%               puissance   : log a base 2 du seuil de départ
%%
%%         auteur: Monsef Mekouar
%%
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%



%%%%%%%%%%%%%%%%%%%%%%% initialisations%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

puissance=fix(log2(max(max(im))))
t=2^(puissance)
taille=16; % taille de la sous image
kmax=4;
niv_dec=4; %niveau de decomposition
miroir=zeros(taille);
res_sub0=0; % Passage secondaire
resf=''; % Passage dominant

%%%%%%%%%%%%%%%%%%%%%%%%%%%%Traitement%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% premiers elements de la matrice (sous bande LL)

while(t >= seuil_final)

miroir=zeros(taille);
k=kmax-niv_dec;
res0='';

 for i=1:2^k
      for j=1:2^k
         if (miroir(i,j)==0);
             if (abs(im(i,j))>= t)
```

```
                if (im(i,j)>0)
                            res0 = strcat(res0,'P');
                            else
                           res0 = strcat(res0,'N');
                      end
              res_sub0 = strcat(res_sub0,sub(t,abs(im(i,j))));
              im(i,j)=abs(im(i,j))-t ;

           else
               res0 = strcat(res0,'R');
           end
        end
     end
  end

% ....Autres sous bandes

% Parcours Horizontal

while(k < kmax)

  for i=1:2^k
     for j=(2^k)+1:2^(k+1)
        if (miroir(i,j)==0)
           if (abs(im(i,j))>= t)
                if (im(i,j)>0)
                          res0 = strcat(res0,'P');
                         else
                          res0 = strcat(res0,'N');
                    end
              res_sub0 = strcat(res_sub0,sub(t,abs(im(i,j))));
              im(i,j)=abs(im(i,j))-t ;

           else
            [boolzh,miroir]=zerotree(t,i,j,im,miroir);

            if (boolzh==1)
               res0 = strcat(res0,'Z');
            else
               res0 = strcat(res0,'R');
            end
         end
      end
    end
  end

  % Parcours Vertical

  for i=(2^k)+1:2^(k+1)
     for  j=1:2^k
        if ( miroir(i,j)==0)
          if (abs(im(i,j))>= t)
                if (im(i,j)>0)
                   res0 = strcat(res0,'P');
```

```
                    else
                              res0 = strcat(res0,'N');
                        end
                    res_sub0 = strcat(res_sub0,sub(t,abs(im(i,j))));
                    im(i,j)=abs(im(i,j))-t ;

               else
                   [boolzh,miroir]=zerotree(t,i,j,im,miroir);
                   if (boolzh==1)
                       res0 = strcat(res0,'Z');
                   else
                       res0 = strcat(res0,'R');
                   end

               end
             end
             end
          end

      % Parcours Diagonal

      for i=(2^k)+1:2^(k+1)
        for j=(2^k)+1:2^(k+1)
          if (miroir(i,j)==0)
            if (abs(im(i,j))>= t)
                    if (im(i,j)>0)
                              res0 = strcat(res0,'P');
                       else
                    res0 = strcat(res0,'N');
               end
               res_sub0 = strcat(res_sub0,sub(t,abs(im(i,j))));
               im(i,j)=abs(im(i,j))-t ;
               else
               [boolzh,miroir]=zerotree(t,i,j,im,miroir);
               if (boolzh==1)
                    res0 = strcat(res0,'Z');
               else
                    res0 = strcat(res0,'R');
               end
           end
          end
        end;
         end
      k=k+1;
end

res_sub=res_sub0;
t=t/2
resf = strcat(resf,res0);
end
```

```
function m = sub(t,i)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%  m = sub(t,i)
%%
%%          Cette fonction effectue le parcours secondaire de la méthode
%%          EZW coding.
%%
%%          Parametres d'entrée
%%                i           : coefficient étudié
%%                t           : Seuil actuel
%%
%%          Parametres de sortie
%%                m           : symbole secondaire
%%
%%
%%          auteur: Monsef Mekouar
%%
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

if (im > 3*t/2)
      m='A';
   else
      m='O';
end
```

```
function [bool,miroir]= zerotree(t,x,y,im,miroir)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%  [bool,miroir]= zerotree(t,x,y,im,miroir)
%%
%%         cette fonction étudie pour un coefficient nul par rapport
%%         a un seuil t s'il est racine d'un arbre de zéros ou non, si
%%         oui la matrice binaire retournée indique l'emplacement de
%%         tous les coefficients qui seront ignorés durant le parcours
%%         de la matrice d'étude .
%%
%%         Parametres d'entrée
%%               t          : Seuil actuel
%%               x,y        : Coordonnées du coefficient étudié
%%               im         : matrice de coefficients
%%               miroir     : Matrice binaire indiquant les
%%                            coefficients 'nuls' par rapport a ce seuil
%%
%%         Parametres de sortie
%%               bool       : Variable booleenne indiquant si l'arbre
%%                            parcouru est un arbre de zéros
%%               miroir     : Matrice binaire modifiée
%%
%%         auteur: Monsef Mekouar
%%
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%  Initialisations  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

bool=0;
miroir1=miroir;
xmin=2*x-1;
ymin=2*y-1;
xmax=2*x;
ymax=2*y;

%%%%%%%%%%%%%%%%%%%%%%%%%%    Traitement     %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

while ((xmax <= length(im)) & (ymax <= length(im)))

for i = xmin : xmax,
   for j = ymin : ymax ,

      miroir(i,j)=1;

      if (abs(im(i,j))>= t)
          bool=1;
      end

   end
end
```

```
    xmin=2*xmin-1;
    ymin=2*ymin-1;
    xmax=2*xmax;
    ymax=2*ymax;

end

% cas ou il existe un coefficient superieur a t dans l'arborescence

if(bool==1)
   miroir=miroir1;
end
```

```
function m = arith16(a,puissance)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%   m = arith16(a,puissance)
%%
%%         Cette fonction effectue le codage arithmétique sur les
%%         symboles obtenus apres quantification représentés par
%%         a. Elle retourne le flot binaire corespondant.
%%
%%         Parametres d'entrée
%%               a           : Code zerotree
%%               puissance   : Log_2 du seuil maximum de quantification
%%
%%         Parametres de sortie
%%               m           : code binaire
%%
%%         % exp : a='PNPPNNZZZZPPPNNRRRRRRRRRRRRRRRRRRR'
%%
%%         auteur: Monsef Mekouar
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%% Initialisations %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

format long;
pn=1; % numerateur de la probabilité du symbole N
pp=1; % numerateur de la probabilité du symbole P
pr=1; %numerateur de la probabilité du symbole R
pz=1; %numerateur de la probabilité du symbole Z
tot=4; % denominateur des probabilités
borne_inf=0; % borne inf de l intervalle d' etude
borne_sup=1; % borne sup de l intervalle d' etude
cmpt=1;
temp=0; % valeur temporaire pour stockage de chiffres a coder
compte=0;% compteur de chiffres à coder
code=''; % variable contenant le code final binaire
paq=10;
total=length(a);

% Calcul du nombre de symboles du code zerotree qui
% constituera une condition d'arret pour le décodage
% arithmétique, il est stocké sur 16 bits

bintotal=dec2bin(total);
while(length(bintotal)< 16)
   bintotal=strcat('0',bintotal);
end

% Conversion de la puissance du seuil maximum, qui fera
% aussi partie du code binaire final

p=dec2bin(puissance);
```

```
while(length(p)< 4)
       p=strcat('0',p);
    end

%%%%%%%%%%%%%%%%%%%%%%%%%%%% Codage %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for cmpt=1:length(a)
 larg= borne_sup - borne_inf;
 c=a(cmpt);

switch c
              % modification des bornes de l'intervalle et des
probabilités
              % associées selon le symbole lu

        case 'N'
             borne_sup=borne_inf+ larg*(pn/tot);
             pn=pn+1;

         case 'P'
             borne_inf=borne_inf+ larg*(pn/tot);
             borne_sup=borne_inf+ larg*(pp/tot);
             pp=pp+1;

         case 'R'
             borne_sup=borne_sup - larg*(pz/tot);
             borne_inf=borne_sup - larg*(pr/tot);
             pr=pr+1;

         case 'Z'
              borne_inf=borne_sup - larg*(pz/tot);
              pz=pz+1;
         end

         % traitement des similitudes entre borne sup et borne inf

         while(floor(10*borne_sup) - floor(10*borne_inf) < 10^(-15))
            temp= 10*temp + floor(10*borne_inf);
            borne_sup=10*borne_sup - floor(10*borne_sup);
            borne_inf=10*borne_inf - floor(10*borne_inf);
            compte=compte+1;

            % codage si le nombre de chiffres stockés sur temp a
atteint le
                  %paquet désiré

            if(mod(compte,paq)==0)
               temp
               bin= dec2bin(temp);

               while(length(bin)<34) % cette boucle nous permet de ne
pas avoir
                   bin=strcat('0',bin); %un depassement de bits
               end
```

```
            code=strcat(code,bin);
            temp=0;
    end;
 end
 tot=tot+1;
end

% cas où le nombre de chiffres traité n'est pas
% multiple de la longueur du paquet de codage

 if(mod(compte,paq)~= 0)
    if (borne_sup ~= 1)
      temp =10*temp+floor(10*borne_sup)
      bin= dec2bin(temp);
      code=strcat(code,bin);
    else
       %etude du cas de borne_sup=1
       code=borne_inf;
    end
 end;

 m=strcat(bintotal,p,code);
```

```
function [m,puissance] = decode_arith16(cod)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%  [m,puissance] = decode_arith16(cod)
%%
%%          Cette fonction effectue le décodage arithmétique sur
%%          'cod' et donne en résultat la liste des symboles zerotree.
%%
%%          Parametres d'entrée
%%                cod          : code binaire
%%
%%          Parametres de sortie
%%                m            : Liste des symboles zerotree
%%                puissance    : log a base 2 du seuil de départ
%%
%%
%%          auteur: Monsef Mekouar
%%
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%Initialisations%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

format long;
pn=1;
pp=1;
pr=1;
pz=1;
tot=4;
borne_inf=0;
borne_sup=1;
m='';
code=0;
paq=10;
bool=1;
cmpt=0;
total=bin2dec(cod(1:16)); % nombre de symboles zerotree
puissance=bin2dec(cod(17:20)); % puissance du seuil de départ
total2=total;
nbre_it=fix(total/34);
reste=mod(total,34);
cod=cod(21:length(cod));
pl=0;

%%%%%%%%%%%%%%%%%%%% Traitement pré-décodage %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

   if (length(cod)>= 34)
      temp= bin2dec(cod(1:34))
      cod=cod(35:length(cod));  %
      temp=temp/10000000000;
      cmpt=10;
   else
```

```
      temp= bin2dec(cod);
          code2=temp;
          while(temp > 1)
         temp=temp/10;
         cmpt=cmpt+1;
      end
   end
      code=code+temp;

while(bool)

   if(cmpt<=4)
      code2=vpa(code,5);
      code=double(code2);

    if (length(cod)> 34) %
      temp= bin2dec(cod(1:34));% var temporaire servant a stocker le
               % mot binaire en decimal avant de le transmettre a code;
      cod=cod(35:length(cod));
      cmpt=cmpt+10;

    else
      temp2= bin2dec(cod);
      temp=temp2;
      bool=0;
           while(temp2 > 1) %
            temp2=temp2/10;
            cmpt=cmpt+1;
           end % while(temp..
    end  % if (length(cod)> 34)..

    code=code+ (temp/(10^cmpt))
    if(length(cod)==0)
        pl=1;
    end

   end % if cmpt<=2

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%Décodage%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

while (cmpt > 4 & bool)

  larg= borne_sup - borne_inf;

  if (code > (borne_inf+(pn+pp)*larg/tot))
     if (code > (borne_inf+((pn+pp+pr)*larg/tot)))
        m = strcat(m,'Z');
        borne_inf=borne_sup - larg*(pz/tot);
        pz=pz+1;
     else
       m = strcat(m,'R');
       borne_sup=borne_sup - larg*(pz/tot);
       borne_inf=borne_sup - larg*(pr/tot);
       pr=pr+1;
```

```
     end

      else

   if (code > (borne_inf+(pn*larg/tot)))
       m = strcat(m,'P');
       borne_inf=borne_inf+ larg*(pn/tot);
       borne_sup=borne_inf+ larg*(pp/tot);
       pp=pp+1;
    else
       m = strcat(m,'N');
       borne_sup=borne_inf+ larg*(pn/tot);
       pn=pn+1;
    end

 end;


% traitement des similitudes entre borne sup et borne inf
         code2=code;

         while(floor(10*borne_sup) - floor(10*borne_inf) < 10^(-15))

           if (pl==1)
               pl=0;    %pb de debordement
           end

           borne_sup=10*borne_sup - floor(10*borne_sup);
           borne_inf=10*borne_inf - floor(10*borne_inf);
           code2=vpa(10*code2,15) ;
           code2=code2 -floor(double(code2));
           cmpt=cmpt-1;
         end

   code=double(code2);
   tot=tot+1;

  end
end

if(length(m)<total)
    bool=1;
end;

% Cas ou le nombre de bits traités est inférieur au code binaire

while (length(cod)<=34 & bool)      % boucle temporaire
  while((code>borne_sup) | (code< borne_inf))
    code=code/10
  end
    larg= borne_sup - borne_inf;
  if (code > (borne_inf+(pn+pp)*larg/tot))
    if (code > (borne_inf+((pn+pp+pr)*larg/tot)))
       m = strcat(m,'Z');
```

```
        borne_inf=borne_sup - larg*(pz/tot);
            pz=pz+1;
         else
        m = strcat(m,'R');
        borne_sup=borne_sup - larg*(pz/tot);
        borne_inf=borne_sup - larg*(pr/tot);
        pr=pr+1;
      end
    else
      if (code > (borne_inf+(pn*larg/tot)))
        m = strcat(m,'P');
        borne_inf=borne_inf+ larg*(pn/tot);
        borne_sup=borne_inf+ larg*(pp/tot);
        pp=pp+1;
      else
        m = strcat(m,'N');
        borne_sup=borne_inf+ larg*(pn/tot);
        pn=pn+1;
      end
   end;
    tot=tot+1;
    if(total==length(m))
       bool=0;
    end
  end
```

```
function im = dec16(code,puissance,seuil_min)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%  im = dec16(code,puissance,seuil_min)
%%
%%          Cette fonction effectue la déquantification des symboles
%%          obtenus lors du décodage arithmétique, selon le seuil
utilisé
%%          en quantification et retourne la matrice de coefficients
%%          d'ondelettes.
%%
%%          Parametres d'entrée
%%                code        : Code zerotree
%%                puissance   : log a base 2 du seuil de départ
%%                seuil_min   : Seuil d'arret de déquantification
%%
%%          Parametres de sortie
%%                im          : matrice de coefficients
%%
%%
%%          auteur: Monsef Mekouar
%%
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%% initialisations %%%%%%%%%%%%%%%%%%%%%%%%%%%%%

cmpt=1;
t=2^puissance;
taille=16; % dimension de la matrice
kmax=4;
im=zeros(taille);
niv_dec=4;

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% premiers elements de la matrice

while(t >= seuil_min)

   miroir=zeros(taille);
   k=kmax-niv_dec;

   for i=1:2^k
      for j=1:2^k
         t2=t;
         if (miroir(i,j)==0)
          c=code(cmpt);
          switch c

            case 'P'
             if (im(i,j)<0) t2=-t;
```

```
            end
                im(i,j)= im(i,j)+t2;
                miroir(i,j)=1;

            case 'N'
               im(i,j)=im(i,j)-t;
               miroir(i,j)=1;

            case 'R'
               miroir(i,j)=1;

          end
          cmpt=cmpt+1;
        end
      end
    end

while(k < kmax)

    t2=t;

    % Parcours horizontal

    for i=1:2^k
       for j=(2^k)+1:2^(k+1)

          t2=t;

          if (miroir(i,j)==0)

           c=code(cmpt);
           switch c

            case 'P'
              if (im(i,j)<0) t2=-t;
              end
              im(i,j)= im(i,j)+t2;
              miroir(i,j)=1;

            case 'N'
              im(i,j)=im(i,j)-t;
              miroir(i,j)=1;

            case 'R'
                miroir=zerotree2(miroir,i,j);

            case 'Z'
                miroir(i,j)=1;
            end

                cmpt=cmpt+1;
          end
      end
    end
```

```
    %Parcours vertical

for i=(2^k)+1:2^(k+1)
   for  j=1:2^k
      t2=t;
      if (miroir(i,j)==0)
         c=code(cmpt);
         switch c

           case 'P'
              if (im(i,j)<0) t2=-t;
              end
              im(i,j)= im(i,j)+t2;
              miroir(i,j)=1;

           case 'N'
              im(i,j)=im(i,j)-t;
              miroir(i,j)=1;

           case 'R'
              miroir=zerotree2(miroir,i,j);

           case 'Z'
              miroir(i,j)=1;
              end;

         cmpt=cmpt+1;
      end
    end
   end

% Parcours diagonal

for i=(2^k)+1:2^(k+1)
   for j=(2^k)+1:2^(k+1)
      t2=t;
      if (miroir(i,j)==0)
         c=code(cmpt);

         switch c

            case 'P'
              if (im(i,j)<0) t2=-t;
              end
              im(i,j)= im(i,j)+t2;
              miroir(i,j)=1;

            case 'N'
               im(i,j)=im(i,j)-t;
               miroir(i,j)=1;

            case 'R'
               miroir=zerotree2(miroir,i,j);
```

```
                    case 'Z'
                       miroir(i,j)=1;

            end

            cmpt=cmpt+1;
         end
      end
   end

 k=k+1;

end   % while(k<3)

if (miroir == ones(taille))
   t=t/2;
   miroir=zeros(taille);
end;

end   % while (t>=1)
```

```
function miroir = zerotree2(miroir,x,y)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%  miroir = zerotree2(miroir,x,y)
%%
%%         Cette fonction utilisée dans la déquantification permet
%%         de gérer la matrice d'état 'miroir', en définissant les
%%         arbres de zéros.
%%
%%         Parametres d'entrée
%%               miroir      : matrice binaire d'état
%%               x,y         : coordonnées du coefficient étudié
%%
%%         Parametres de sortie
%%               miroir      : matrice binaire d'état
%%
%%
%%         auteur: Monsef Mekouar
%%
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

xmin=2*x-1;
ymin=2*y-1;
xmax=2*x;
ymax=2*y;
miroir(x,y)=1;

while ((xmax <= length(miroir)) & (ymax <= length(miroir)))

 for i = xmin : xmax,
   for j = ymin : ymax ,
         miroir(i,j)=1;
     end
 end

 xmin=2*xmin-1;
 ymin=2*ymin-1;
 xmax=2*xmax;
 ymax=2*ymax;

end
```

```
function compose16(res)

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%  compose16(res)
%%
%%          Cette fonction effectue la transformation inverse en
%%          ondelettes. Comme entrée, il y'a la matrice de coefficients.
%%
%%          auteur: Monsef Mekouar
%%
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%%%%% Initialisations %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

filtre='db3';
si=16; % taille de sous_image;
l=length(res);
nb=l/si;
X2=zeros(l);
dwtmode('ppd');

%%%%%%%%%%%%%%%%%%%%%%%%%%%% Traitement %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

for(i=1:nb)
   for(j=1:nb)

    x=res(1+(i-1)*si:i*si,1+(j-1)*si:j*si);

    ca4=x(1,1);
    cv4=x(2,1);
    ch4=x(1,2);
    cd4=x(2,2);

    ch3=x(1:2,3:4);
    cv3=x(3:4,1:2);
    cd3=x(3:4,3:4);

    ch2=x(1:4,5:8);
    cv2=x(5:8,1:4);
    cd2=x(5:8,5:8);

    ch1=x(1:8,9:16);
    cv1=x(9:16,1:8);
    cd1=x(9:16,9:16);

    ca4=ajout(ca4);
    ch4=ajout(ch4);
    cv4=ajout(cv4);
    cd4=ajout(cd4);

    ch3=ajout(ch3);
```

```
    cv3=ajout(cv3);
    cd3=ajout(cd3);

    ch2=ajout(ch2);
    cv2=ajout(cv2);
    cd2=ajout(cd2);

    ch1=ajout(ch1);
    cv1=ajout(cv1);
    cd1=ajout(cd1);

    ca3= idwt2(ca4,ch4,cv4,cd4,filtre);
    ca3=ajout(ca3);

    ca2= idwt2(ca3,ch3,cv3,cd3,filtre);
    ca2=ajout(ca2);

    ca1= idwt2(ca2,ch2,cv2,cd2,filtre);
    ca1=ajout(ca1);

    r= idwt2(ca1,ch1,cv1,cd1,filtre);

    X2(1+(i-1)*si:i*si,1+(j-1)*si:j*si)=r;

   end
end

imagesc(X2),colormap(gray)
```

```
function x=ajout(y)


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%  x=ajout(y)
%%
%%          Cette fonction effectue une modiication de la matrice
%%          de coefficients 'y', pour qu'elle soit conforme a la
%%          transformée en ondelette existante sur matlab
%%
%%          Parametres d'entrée
%%                y       : Matrice initiale
%%
%%          Parametres de sortie
%%                x       : Matrice modifiée
%%
%%
%%          auteur: Monsef Mekouar
%%
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

l=length(y);

if l>1
 plus1=y(length(y)-1:l,1:l);
 y=[plus1;y]; %ajout de lignes
 plus2=y(1:l+2,l-1:l);
 x=[plus2,y];  %ajout de colonnes

else
    x=y*ones(3);
end
```

```
function matrice = script18


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%
%%  script18
%%
%%          cette fonction est le script de la méthode Zerotree Global
%%          elle fait le lien entre les differentes fonctions, elle
%%          s'occupe de la définition de la région d'interet,choisie
%%          par l'utilisateur, et transmettre les parametres
%%          respectivement aux modules de transformée, quantification et
%%          codage.L'application présentée a été effectuée sur l'image
%%          'woman'.
%%
%%          auteur: Monsef Mekouar
%%
%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%%%%%%%%%%%%%%%%%%%%%%%%% Initialisations %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

load woman2;
I=ind2gray(X,map);
L=length(X);
bw=zeros(L); % masque
seuil=32
decalage=log2(seuil); % décalage qui sera effectué sur les coef de la
roi

bw(40:80,40:80)=1; % masque arbitraire

%affichage de la région masquée
imagesc(X.*bw);
colormap(gray);

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% Compression %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

%Transformée en ondelettes sur l'image et le masque
a=ondelette(bw);
a2=ondelette(X);

% décalage des coefficients de la roi
for(i=1:L)
   for(j=1:L)
      if(a(i,j)~=0)
         a2(i,j)=bitshift(a2(i,j),decalage);
      end
   end
end

% Quantification
[a3,a4]=par16(a2,seuil);
```

```
% Codage arithmétique
    bin=arith16(a3,a4);

%%%%%%%%%%%%%%%%%%%%%%%%%% Décompression %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Décodage arithmétique
a5=dec16(a3,a4,seuil);

% décalage des coefficients de la roi
for(i=1:L)
   for(j=1:L)
      if(a(i,j)~=0)
         a5(i,j)=bitshift(a5(i,j),-decalage);
      end
   end
end

%Transformée inverse en ondelettes

b=desondelette(a5);

length(a3)
length(bin)
```