TER : Algorithmes de compression d'images et quadtree

Lionel GUEZ

12 mai 2011

Table des matières

Le but de mon TER, est de faire un état de l'art de deux algorithmes de la compression d'image : EZW et EBCOT(ce dernier étant utilisé par JPEG2000), de comprendre leur force, leur faiblesse, et leur différence. Cet état de l'art passe par faire une petite implémentation de ces algorithmes. Ces deux algorithmes ont pour point commun d'utiliser, non pas une transformée de Fourier (ou une DCT qui est similaire), comme l'algorithme utilisé par le format JPEG, mais une transformée en ondelettes qui a de meilleurs propriétés. Ainsi mon rapport commencera par expliciter les nouveaux enjeux de la compression d'image, puis il continuera en expliquant en quoi consiste une transformée en ondelettes, puis il expliquera, en les détaillant un par un, les algorithmes EZW et EBCOT. Enfin, je concluerai par ce qui sépare ces deux algorithmes et tenterai d'expliquer quels sont leur force et leur faiblesse.

1 Nouveaux objectifs de la compression d'image

Evidemment le but de la compression d'image est de comprimer le plus possible des données de type image. Toutefois, l'évolution des besoins, fait qu'en comprimant des données, on a aussi besoin de nouvelles fonctionnalités en plus de la compression pure et simple : ainsi les formats images développés pour les algorithmes EZW et EBCOT ont été développés non seulement pour pouvoir stocker un grand nombre de données dans un faible espace mais aussi pour pouvoir garantir d'autres propriétés.

1.1 Scalabilité

Une de ces propriétés les plus importantes est la "scalabilité"(c'est un anglicisme venant de scalability). Pour comprendre cette notion, voyons comment est stocké une image dans les formats classiques comme JPEG : les images sont codés dans un ordre(en géneral de gauche à droite puis de haut en bas ou l'inverse) tel qu'il faut décoder toute l'image pour pouvoir afficher une image complète.A contrario, les images stockés dans les formats générés par EZW et EBCOT sont codés de telle maniere que si on ne décode que le début du fichier d'image, on a une image complète mais de qualité faible : si on décode la fin du fichier, la qualité de l'image s'améliore : en d'autres termes, le début du fichier code toute l'image mais avec une résolution et/ou qualité faible, puis la suite du fichier rajoute des détails au fur et a mesure. C'est cette propriété qu'on appelle scalabilité. Cette propriété est utile pour diverses raisons : par exemple elle permet à de petits écran(téléphone portable) et disposant d'un faible débit d'avoir à ne décoder qu'une faible partie du fichier. En vidéo, cette propriété permetrait de pouvoir faire fonctionner du streaming multi-résolution. Nous verrons, par la suite, que la transformée en ondelettes est trés bien adapté pour cette propriété

1.2 Autres propriétés

Une autre propriété utile (disponible avec l'algorithme EBCOT mais pas avec EZW) est la mise en place de "région d'intérêt", cela signifie qu'au lieu de tout compresser et de tout décompresser uniformément, on privéligie certaines zones lors du codage(en compresse plus de donnés correspondant à cette zone) ou lors du décodage(si le format du fichier est scalable, on se contente d'une faible résolution(on ne décode quel le début du fichier) sauf pour une zone où on va chercher jusqu'à la fin du fichier les informations). Une autre propriété est la navigabilité : la scalabilité fait que le stockage perd la notion de localisation(que faire si on ne veut qu'une partie de l'image alors que le fichier n'est pas organisé en zone comme dans JPEG, mais en couche de qualité ?). Toutefois, cette notion de localisation peut parfois être utile, c'est pourquoi il est utile de garder certains mécanismes permettant d'aller à la zone de l'image que l'on veut :cette propriété s'appelle la "navigabilité". EBCOT a cette propriété .

Enfin, une dernière propriété est la résistance aux erreurs de transmission : coupler un code compresseur et un code correcteur d'erreur permet d'avoir cette propriété, toutefois, il est plus optimal d'avoir ces deux aspects dans un même format(le format JPEG2000 a ces deux aspects).

On comprend alors que la performance en compression n'est plus le seul aspect utile lorqu'on met au point de nouveaux algorithmes de compression d'images, ce qui explique pourquoi les chercheurs ont voulu dépasser les formats existants comme JPEG pour mettre au point de nouveaux algorithmes.

2 La transformée en ondelettes

La transformée de Fourier,et les autres transformées similaires, comme la DCT (Discrete Cosinus Transform ou Transformée en cosinus discrète), a eu pendant un long moment le monopole de la compression de signaux, et particulierement de la compression d'image. Mais la transformée de Fourier n'est pas exempte de défauts. Le principal défaut est que si elle contient le contenu fréquentiel d'une image(son spectre), elle ne dit rien(dans un sens que je préciserai plus tard) sur *quand* apparait la fréquence. Evidemment, on peut toujours recomposer totalement un signal avec la transformée du signal : il faut donc comprendre le "Elle ne dit rien" par le fait que deux signaux trés différents mais dont le spectre est le même, donne une transformée trés proche. Ce défaut fait que le format "star" de la compression d'image, le JPEG, est obligé de subdiviser l'image en bloc 8x8(mais cette subdivision est aussi dû à des raisons de performance) avant de faire une DCT sur chacun des blocs, mais cette taille est arbitraire et artificielle, elle est tantôt mal adaptée, tantôt bien adaptée, selon le niveau de détail de l'image que l'on mettre en évidence. C'est pourquoi, une autre transformation s'impose, et un exemple de transformation qui évite les écueils de la transformée de Fourier, est la transformée en ondelettes. C'est pourquoi, nous allons détailler cette transformation, d'abord dans le cas continu(rapidement, car ce cas n'est pas utile pour la compression qui s'applique à des données discrète), puis dans le cas discret. Et enfin nous allons voir son application dans le cas de la compression d'image.

2.1 Transformée en ondelettes continue

Tentons de donner un trés rapide aperçu de ce qu'est la transformée en ondelettes dans le cas continu. Une ondelette est une fonction ψ qui vérifie certaines propriétés ($\psi \in L^1(\mathbb{R}) \cap L^2(\mathbb{R})$ *et* $\int_{-\infty}^{\infty} \left|\frac{\widehat{\psi}(\nu)}{\nu}\right| d\nu < \infty$). Soit ψ une telle fonction, on appellera cette fonction ψ une "ondelette mère". On définit alors les ondelettes "filles" $\psi_{a,b} = \frac{1}{\sqrt{a}}\psi\left(\frac{x-b}{a}\right)$. Muni de cette famille de fonctions, on peut desormais décomposer une fonction avec cette famille, exactement de la mème manière qu'on le fait avec la famille des fonctions exponentielles complexes lors d'une transformation de Fourier.

$$Wf(a,b) = \frac{1}{\sqrt{a}} \int_{-\infty}^{\infty} f(x) \overline{\psi\left(\frac{x-b}{a}\right)}$$

On obtient ainsi une fonction de 2 variables (alors qu'on avait une fonction à une variable avec une transformée de Fourier). Expliquons en quoi cette transformée est utile. La transformée de Fourier, comme on l'a dit, ne donne des informations que sur le contenu fréquentiel du signal, on ne sait pas(sauf à faire une transformation inverse, et encore, cela ne fonctionne que si le signal transformé n'a pas été altéré, même légerement) *quand* la fréquence apparaît

dans le signal.Pour donner une information temporel, on pourrait se dire qu'il suffit de faire une transformée de Fourier sur une "fenêtre glissante", c'est à dire qu'on ne fait la transformée de Fourier que sur le produit de fonction la fonction par une fonction 'fenêtre' (fonction continue approchant la fonction caractéristique d'un intervalle), puis de "déplacer" la fenêtre, on obtiendrait alors une fonction transformée à deux dimensions qui sont la fréquence et le centre de la fenêtre. Mais le problème d'une telle approche est que la taille de la fenêtre est trop déterminante : une trop grande fenêtre a tendance à oublier les phénomènes apparaissant dans des petits intervalles, alors qu'une petite fenêtre ne permet pas de remarquer les tendances longues. C'est là que la transformée en ondelettes intervient ! En effet, supposons qu'un phénomène apparaisse dans un petit intervalle de taille a autour du point b alors cela affectera la valeur W(a,b). De cette manière, on peut repérer des phénomènes de toutes les echelles et partout ! Mais, de par sa nature continue, la transformée en ondelettes continu ne peut être directement appliqué dans le cas du traitement d'image car les images étant représentés par des pixels sont intrinsequement de nature discrète.

2.2 Transformée en ondelettes discrète

2.2.1 Transformation 1D

Le but de la transformation en ondelettes discrète est la même que celle de la transformée en ondelettes continue : pouvoir mettre en évidence des phénomène de n'importe quelle échelle tout en donnant leurs localisation. Nous expliquerons ici, uniquement, les ondelettes de Haar, car ce sont les plus simples, mais il faut savoir qu'il existe d'autres ondelettes(Daubechies,...) Dans la suite, nous noterons $< .|. >$ le produit scalaire qui est tel que : $< f|g >= \int_0^1 fg$. Soit ϕ la fonction qui vaut 1 sur [0,1] et est nulle ailleurs. ϕ sera nommé, par la suite, la fonction d'échelle. Soit ψ la fonction qui vaut 1 sur [0,1/2[et -1 sur [1/2,1[et est nulle ailleurs. On définit ensuite les fonctions $\psi_{k,j} = \psi(2^j x - k)$ ou $0 \leq k < 2^j - 1$

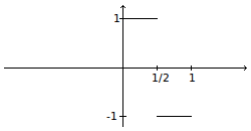

FIGURE 1 – La fonction ψ

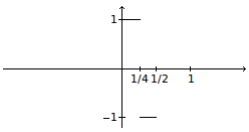

FIGURE 2 – La fonction $\psi_{0,1}$

Donc $\psi_{j,k}$ est la fonction qui à x associe 1 quand $x \in [k2^j,\ \left(k+\frac{1}{2}\right)2^j[$ et -1 quand $x \in [\left(k+\frac{1}{2}\right)2^j,\ (k+1)\,2^j[$.

On remarque que si on se fixe un niveau j maximum, on peut facilement discretiser ces fonctions sur l'ensemble $\mathbb{R}^{2^{j+1}}$. En effet, pour $j_{max} = 1$, et donc pour $\mathbb{R}^4$ les fonctions ϕ, ψ, $\psi_{0,1}$ et $\psi_{1,1}$ sont représenté par les vecteurs

$$\phi = {}^t(1\ 1\ 1\ 1)\,;\quad \psi = {}^t(1\ 1\ -1\ -1)\,;\quad \psi_{1,0} = {}^t(1\ -1\ 0\ 0)\,;\quad \psi_{1,1} = {}^t(0\ 0\ 1\ -1)$$

De même, pour $j_{max} = 2$:

$$\phi = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \\ 1 \end{pmatrix} ; \ \psi = \begin{pmatrix} 1 \\ 1 \\ 1 \\ 1 \\ 1- \\ -1 \\ -1 \\ -1 \end{pmatrix} ; \ \psi_{1,0} = \begin{pmatrix} 1 \\ 1 \\ -1 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} ; \ \psi_{1,1} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ -1 \\ -1 \end{pmatrix} ;$$

$$\psi_{2,0} = \begin{pmatrix} 1 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} ; \ \psi_{2,1} = \begin{pmatrix} 0 \\ 0 \\ 1 \\ -1 \\ 0 \\ 0 \\ 0 \\ 0 \end{pmatrix} ; \ \psi_{2,2} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ -1 \\ 0 \\ 0 \end{pmatrix} ; \ \psi_{2,3} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ -1 \end{pmatrix} ;$$

A chaque fois, on obtient des bases orthogonales, qui se nomme base de Haar. En général, on préfère normaliser ces vecteurs, mais il existe plusieurs sortes de coefficient de normalisation, et ils n'ont pas d'importance dans la dans la compréhension de la transformation, (même s'ils en ont dans la compression, comme on le verra dans la partie 3).

Ainsi la transformée en ondelettes discrète est, à l'instar de la DCT, un simple changement de base. Les coefficients d'un vecteur v , dans la base ainsi créé sont donc obtenu par produit scalaire discret dans $\mathbb{R}^{2^{j+1}}$ de v par chaque vecteur de la base. Remarquons, avant de passer à la transformation 2D, que le produit scalaire d'un vecteur par un le vecteur de la base ${}^t(1\ 1\ 1\ 1)$ donne la moyenne des coefficients du vecteur(donc la composante continue). Le vecteur ${}^t(1\ 1\ -1\ -1)$ donne les tendances des grands phénomèmes, alors que ${}^t(1\ -1\ 0\ 0)$ et ${}^t(0\ 0\ 1\ -1)$ donne les tendances de de phénomènes localisés, c'est pourquoi, on peut parler pour ${}^t(1\ 1\ 1\ 1)$ et ${}^t(1\ 1\ -1\ -1)$ de basses fréquences(car les phénomènes se déroulent sur de grandes échelles) alors que l'on parle de hautes fréquence pour ${}^t(1\ -1\ 0\ 0)$ et pour ${}^t(0\ 0\ 1\ -1)$. On remarque qu'en plus de donner une information sur la fréquence, le produit scalaire par les hautes fréquences donne aussi une information sur la localisation du phénomène.

2.2.2 Transformation 2D

La transformée en ondelettes en dimension 2 ressemble beaucoup à celle en dimension 1. Les fonctions qui prendront le rôles des fonction ϕ et ψ sont :

$$\Phi(x,y) = \phi(x)\phi(y)$$

$$\Psi^1(x,y) = \psi(x)\phi(y)$$
$$\Psi^2(x,y) = \phi(x)\psi(y)$$
$$\Psi^3(x,y) = \psi(x)\psi(y)$$

Ces fonctions ont la forme suivante :

$$\Phi : \begin{array}{|cc|}\hline 1 & 1 \\ 1 & 1 \\ \hline \end{array} ; \ \Psi^1 : \begin{array}{|c|c|}\hline 1 & -1 \\ 1 & -1 \\ \hline \end{array} ; \ \Psi^2 : \begin{array}{|cc|}\hline 1 & 1 \\ \hline -1 & -1 \\ \hline \end{array} ; \ \Psi^3 : \begin{array}{|c|c|}\hline 1 & -1 \\ \hline -1 & 1 \\ \hline \end{array} ;$$

On définit alors les fonctions suivantes :

$$\Psi^i_{j,\vec{k}} = \Psi^i\left(2^j x - k_x, 2^j y - k_y\right)$$

Donc de la même manière qu'en 1D, on peut facilement discrétiser ces fonction si on se fixe un j maximum. Ainsi pour $j_{max} = 1$, la base de Haar pour $\mathbb{R}^{16}$ est :

$$\phi = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{pmatrix};\ \psi^1 = \begin{pmatrix} 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \end{pmatrix};\ \psi^2 = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ -1 & -1 & -1 & -1 \\ -1 & -1 & -1 & -1 \end{pmatrix};$$

$$\psi^3 = \begin{pmatrix} 1 & 1 & -1 & -1 \\ 1 & 1 & -1 & -1 \\ -1 & -1 & 1 & 1 \\ -1 & -1 & 1 & 1 \end{pmatrix};\psi^1_{1,(0,0)} = \begin{pmatrix} 1 & -1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix};\psi^2_{1,(0,0)} = \begin{pmatrix} 1 & 1 & 0 & 0 \\ -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix};$$

etc..

De même qu'en 1D, pour que la transformation soit vraiment orthonormale, il faudrait des coefficients de normalisation, mais ceux-ci importe peu dans la compréhension globale des ondelettes.

Interprétons maintenant les vecteurs de la nouvelle base : les produits scalaires par des vecteurs de la forme $\begin{pmatrix} 1 & 1 \\ -1 & -1 \end{pmatrix}$ permettent de mettre en évidence les différences verticales, les produits scalaires par des vecteurs de la forme $\begin{pmatrix} 1 & -1 \\ 1 & -1 \end{pmatrix}$ permettent de mettre en évidence les différences horizontales. Les vecteurs du type, $\begin{pmatrix} 1 & -1 \\ -1 & 1 \end{pmatrix}$,eux, s'interprètent plus difficilement : on peut dire qu'ils permette de mettre en évidence la différence entre les différences verticales et horizontales. Enfin, de même qu'en 1D les produits scalaires par les vecteur de la base s'interpretent comme des basses fréquences ou des hautes fréquences selon la "quantité de zéros" que contient une matrice Une fois les coefficients d'un vecteur v calculés, (on a $v = c_0\Phi(x,y) + \sum_{j=0}^{j_{max}} \left(\sum_{k_x=1}^{2^j-1}\sum_{k_y=1}^{2^j-1} d^1_{j,\vec{k}}\Psi^1_{j,\vec{k}} + d^2_{j,\vec{k}}\Psi^2_{j,\vec{k}} + d^3_{j,\vec{k}}\Psi^3_{j,\vec{k}}\right)$)les coefficients sont placés de la maniere suivante(avec $j_{max} = 1$) :

$$\begin{pmatrix} c_0 & d^1_0 & d^1_{1,(0,0)} & d^1_{1,(0,1)} \\ d^2_0 & d^3_0 & d^1_{1,(1,0)} & d^1_{1,(1,1)} \\ d^2_{1,(0,0)} & d^2_{1,(0,1)} & d^3_{1,(0,0)} & d^1_{1,(0,1)} \\ d^2_{1,(1,0)} & d^2_{1,(1,1)} & d^3_{1,(1,0)} & d^3_{1,(1,1)} \end{pmatrix};$$

Cette organisation permet de concentrer les basses fréquences en haut à gauche. Une autre manière d'obtenir cette transformation est la suivante : On part de la matrice globale qu'on appellera LL_0. On groupe les coefficient en carré de 4. On fait alors la moyenne de ces 4 coefficients pour chaque carré (qu'on multiplie par un coefficient de normalisation), on obtient alors une matrice 4 fois plus petite qu'on appellera LL_1. A partir des mêmes carrés, on fait la différence entre les coefficients du haut et et ceux du bas, et on fait la moyenne des deux résultas obtenus, on obtient une nouvelle matrice(qu'on appellera LH_1), elle aussi 4 fois plus petite. Puis, on fait de même, la différence entre les coefficients de gauche et ceux

de droite, et on stocke les résultats dans une sous-matrice qu'on nommera HL_1. Et enfin on fait un dernier calcul, mais cette fois, on fait la difference des différence(au lieu de la moyenne) et on appellera la sous-matrice obtenue HH_1. On a donc obtenu 4 matrices 4 fois plus petite qu'on a appelé LL_1, LH_1, HL_1, HH_1. On place ensuite ces 4 matrices dans une nouvelle matrice de la manière suivante :

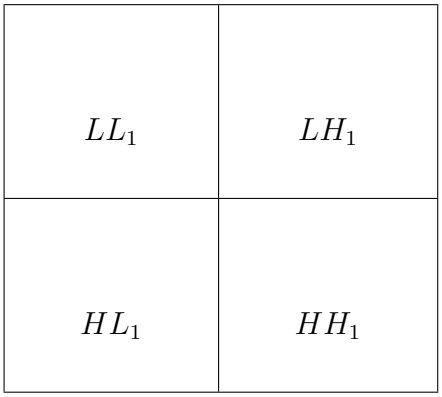

FIGURE 3 – schéma de la première étape de l'algorithme

Les sous-matrices LL_1,HH_1,LH_1,HL_1 sont appelés "sous-bandes". On fait ensuite subir exactement la même transformation à LL_1, et ainsi de suite, jusqu'à que LL_n soit réduit à un point. La matrice ainsi obtenu est exactement la même que celle obtenue avec la base de Haar.

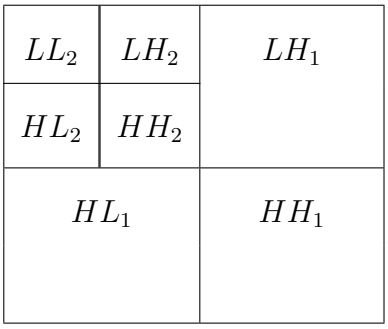

FIGURE 4 – Schéma de la deuxième étape de l'algorithme

2.3 Application au traitement d'image.

L'algorithme précédent permet en un nombre fini d'étapes d'arriver à une décomposition complète dans la base de Haar. Mais on n'est pas obligé d'aller jusqu'au bout dans la décomposition, on peut s'arrêter aprés k étapes de la décomposition. On parle alors de transformée en ondelettes de niveau k. Voyons ce qui se passe quand on fait une transformée en ondelettes incomplète sur une image(lenna) :

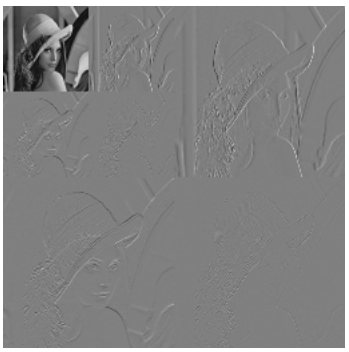

FIGURE 5 – Transformée en ondelettes incomplète

Remarquons une première chose : c'est que l'image en haut à gauche(qui correspond à la bande LLn), est la même image que l'original sauf qu'elle est en résolution plus petite. Là, on comprend un autre intéret de la transformation en ondelette : elle permet de faire des compression scalables en résolution : il suffit de coder les coefficients du haut à gauche en premier, puis les autres coefficients du carré du haut à gauche un peu plus grand, et ainsi de suite. Cela permet de coder d'abord l'image en faible résolution, puis de coder la différence entre une résolution plus forte avec celle déjà codé, et ainsi de suite.....

On remarque aussi que les formes obtenues dans chaque sous-bande se ressemblent cela indique certainement des redondances qui pourront être utilisés pour améliorer le taux de compression.

3 EZW

Le premier algorithme de compression d'image basé sur une transformation en ondelettes que je vais vous présenter dans cette partie est l'algorithme EZW(Embedded Zerotrees of Wavelet coefficients). Cet algorithme a été développé par Jérôme M. Shapiro. Cet algorithme utilise les redondances dans les différentes sous-bandes afin de compresser au maximum l'image. Il utilise de plus une transformée en ondelettes de préférence complète(au contraire d'EBCOT qui se base sur une transformée en ondelettes incomplète comme on le verra). Pour cela, elle s'appuie sur une structure de données particulierement adapté à la transformée en ondelettes : les quadtree (qui sont je le rappelle un des sujets-titres de mon TER).

3.1 Principe de l'algorithme

L'algorithme, comme je l'ai dit plus haut, est basé sur une structure de donnée nommée quadtree. Mais qu'est-ce qu'un quadtree ? Comme son nom l'indique, c'est un arbre dont les noeuds ont 4 fils. Mais en quoi cette structure de données est utile pour cet algorithme en particulier et pour la compression d'image basé sur une transformée en ondelettes en général ? La réponse est simple : si on se souvient de la manière dont on obtient une sous-bande, on remarque qu'une sous-bande LH_k(ou HL_k ou HH_k) a quatre fois plus de points qu'une sous-bande LH_{k+1} (ou HL_{k+1} ou HH_{k+1}).Vu que chaque sous-bande est une "transformée" de l'image complète, à quatre points d'une sous-bandes LH_k correspond un et un seul point d'une sous-bande LH_{k+1} : en effet un point d'une bande LH_{k+1} représente la même zone qu'un carré de 4 points de LH_k et même qu'un carré de 16 points d'une sous-bande LH_{k-1}. Ainsi, on va organiser les points de la matrice comme un arbre : chaque point d'une certaine bande LH_k(respectivement HL_k ou HH_k), pour $k \neq 1$, aura 4 fils : les quatres points qui correspondent à la même zone de l'image dans la sous-bande LH_{k+1},(respectivement HL_{k+1}, ou HH_{k+1}). Les feuilles de l'arbre seront donc les points des sous-bandes LH_1,HL_1,et HH_1(

car il n'y a pas de sous-bande LH_0, HL_0 et HH_0). En réalité seul les points dans la sous-bande LL_n (où n est le niveau de la décomposition) n'ont pas quatres fils, mais d'une part, LL_n n'est réduit qu'à un unique point car la transformation est de préférence complète, et d'autre part, il suffit de considérer que c'est le seul point qui n'aura que 3 fils qui seront les points qui constituent sous-bandes LH_n, HL_n et HH_n. Ce point LL_n devient donc la racine de l'arbre tout en étant le seul noeud n'ayant que trois fils. Même si la transformation n'est pas complète, on peut tout de même utiliser l'algorithme : il suffit de représenter la matrice image non pas en un arbre mais en plusieurs arbres : un arbre pour chaque points de LL_n, mais je ne m'étendrai pas sur ce cas, car il complique inutilement la compréhension de l'algorithme EZW. On a donc créé un arbre dont tous les noeuds(excepté la racine) ont 4 fils : c'est cela un quadtree.Le schéma suivant est extrait de l'article de Shapiro et résume trés bien la structure de donnée :

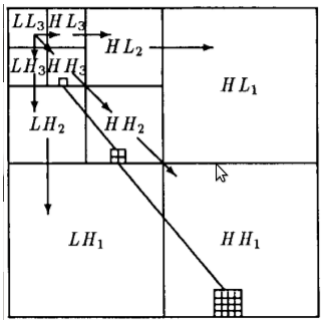

FIGURE 6 – Illustration de la structure de donnée nommée quadtree

Maintenant comment peut-on utiliser cette structure pour exploiter les redondances entre les sous-bandes ? Et bien, on sait que les hautes fréquence d'une image sont en général plus faibles que les basses fréquences. (cette propriété est d'ailleurs utilisée par JPEG qui conservent plus souvent les basses fréquences d'un bloc 8x8 que les hautes fréquences). Or, comme on l'a dit dans la première partie, les points en haut à gauche, c'est-à-dire correspondantes à des sous-bandes ayant un "grand numéro" correspondent aux basses fréquences alors que celle qui se situent aux autres éxtrémités correspondent aux hautes fréquences. Donc, vu la manière dont a été construit le quad-tree, on en déduit que plus un noeud est "profond" dans l'arbre, plus il correspond à un point représentant une haute fréquence.Ainsi, si un point de l'image a une valeur faible(au regard d'un certain seuil), alors il y a de trés fortes chances pour que tous les fils(ainsi que les petit-fils et les petit-petit-fils, etc..) soit aient aux-aussi une faible intensité.

Donc le principe de l'algorithme est le suivant : on choisit un seuil, ensuite et on code l'arbre noeud aprés noeud, mais dés qu'on trouve un noeud(donc un point) dont l'intensité est faible, on vérifie que toute sa descendance a, elle aussi une faible intensité, et si c'est le cas(et c'est souvent le cas, d'aprés la propiété évoqué précedemment), on élague l'arbre du noeud(qui est alors appelé un "zerotree") et de toute sa descendance, ainsi on ne code pas une grosse partie de l'image, qui n'est pas signifiante. Et donc on compresse. ET on comprend alors comment EZW utilise les redondances entre les sous-bandes : elle suppose que si une zone d'une image est peu signifiante, elle sera faible pour toutes les sous-bandes correspondantes à des fréquences supérieures.

3.2 Explication

3.2.1 L'algorithme en détail

Aprés avoir expliqué le principe général de l'algorithme, entrons dans le détail de l'algorithme.

L'algorithme se déroulent en un nombre fini d'étapes chacune composé de deux passes. On choisit un seuil initial T_{init} qui est la plus grande puissance de 2 inférieur à la plus grande valeur absolu de l'intensité des points, et on pose $T = T_{init}$. Ensuite on effectue un certain nombre de phases chacune composé de deux passes que l'on va décrire

Premiere passe : la passe dominante

On parcourt l'arbre dans un ordre qui est tel que les parents soit toujours parcourus avant leurs fils. Quatre cas sont possible :

- Soit le noeud-point considéré a une intensité qui est supérieur à T, on code alors le point par un symbole nommé POS(pour positif)
- Soit le noeud-point considéré a une intensité qui est inférieur à -T, on code alors le point par le symbole NEG(pour négatif)
- Soit le noeud-point a une intensité qui est inférieur, en valeur absolue, au seuil T, mais il existe un noeud de sa descendance dont l'intensité est supérieure au seuil T, on code alors ce point par le symbole IZ(pour IZOLATED ZERO).
- Soit le noeud-point a une intensité qui est inférieur, en valeur absolue, au seuil T, et toute sa descendance est dans le même cas que lui, alors on code le symbole ZTR(pour zero-tree) et on ne codera pas les noeud de sa descendance lors du codage des symboles de cette passe.

Deuxième passe : la passe de raffinage

On ne code dans cette passe que les points qui ont été codés par POS ou NEG lors d'une des précédentes passes dominantes. Pour chaque point codé, on code 0 si le bit de l'écriture binaire de l'intensité correspondant à T/2 vaut 0 ,et on code 1 sinon. Dit autrement, on code 0 pour chaque point qui est signifiant vis-à-vis du seuil T, le résultat de l'opération "et binaire" entre T/2 et la valeur absolue de l'intensité est nulle et 1 sinon.

Ensuite, on recommence en mettant à jour le seuil T = T/2. Sauf que pendant la passe dominante, on considerera les points qui ont déja été codés comme valant 0(car on sait déja qu'ils sont supérieures à T, et donc, à fortiori, à T/2). Cela permet d'élaguer plus de noeud. Et ainsi de suite, jusqu'à qu'on ait la précision voulu ou qu'on a atteint la taille maximale que l'on s'était fixé.

Dernier détail, l'ordre dans lequel sont codés les points est un ordre spécial appelé ordre de Morton, qui permet de coder d'abord les sous-bandes LH_k, HL_k et HH_k avant de commencer de coder les sous-bandes LH_{k-1}, HL_{k-1} et HH_{k-1}, de telle sorte qu'on est sûr de coder les parents avant les enfants. Voici un schéma de l'ordre de codage :

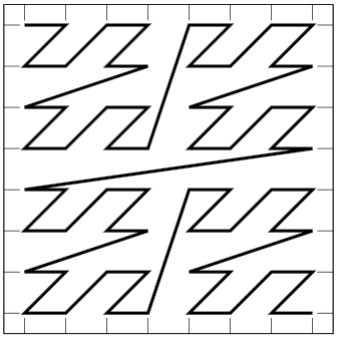

FIGURE 7 – Ordre de Morton

3.2.2 Exemple

Prenons un exemple : considérons la matrice suivante qui est le résultat d'une transformée en ondelettes complètes :

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

63 étant la plus grande intensité en valeur absolue, on choisit comme seuil 32(qui est la plus grande puissance de 2 inférieure à 63). On a alors les symboles suivants :

POS	NEG	POS	ZTR	ZTR	ZTR		
IZ	ZTR	ZTR	ZTR	ZTR	ZTR		
ZTR	IZ						
ZTR	ZTR						
		ZTR	POS				
		ZTR	ZTR				

En suivant l'ordre de Morton, on code donc :

POS NEG IZ ZTR POS ZTR ZTR ZTR ZTR IZ ZTR ZTR

ZTR ZTR ZTR ZTR ZTR POS ZTR ZTR

Puis on passe à l'ordre de raffinage, puisque le résultat du ET binaire entre 63 et 16 n'est pas nul(il vaut 16 car $63 =_2 111111$ ET $16 =_2 001000$ donc $63\ ET\ 16 =_2 111111\ ET\ 001000 =$

001000) donc on code 1. A contrario, 34 ET 16 $=_2$ 100010 ET 001000 = 000000 $=_{10}$ 0 , donc on code 0. On code donc,au final pour cette passe :

1010

Puis on passe à la seconde phase avec un seuil 16.
Cette fois, on a les symboles suivants :

*	*	*					
NEG	POS						
ZTR	ZTR	ZTR	ZTR				
ZTR	ZTR	ZTR	ZTR				
			*				

Donc on le code :

IZ ZTR POS NEG ZTR ZTR ZTR ZTR ZTR ZTR ZTR ZTR

Enfin les coefficients déja codés étant (dans l'ordre où ils ont été codés : 63, -34, 49, 47, -31, et 23, la passe de refinement donne :

100110

Et on continue, avec les autres passes.

3.2.3 Conclusion : la scalabilité de EZW

Remarquons pour conclure cette sous-partie que la compression est scalable : en effet si on ne décode que la premiere phase(qui est au début du fichier), on décode une version trés grossière de l'image, mais on décode l'image entierement, puis plus on décode de phase, plus on ajoute des détails à l'image. Cette scalabilité est encore renforcé par l'ordre de Morton car cette ordre commence par les basses fréquences et donc par ce qui est en général signifiant, donc même si on arrête de décoder en plein fichier, on aura les coefficients les plus importants.

3.3 Mes "experimentations"

Une fois que j'ai compris l'algorithme, j'ai décidé de le coder. Je l'ai donc implémenté en CAML(le codeur aussi bien que le decodeur). Et, j'ai obtenu des taux de compression que je trouvais assez satisfaisant pour un certain PSNR. Ensuite, mon encadrant, Mr Sylvain Meignen, m'a fourni un programme MATLAB implémenté par des chercheurs. J'ai alors comparé les résultats des deux programmes, et je me suis rendu compte que ma version était beaucoup moins performante que la leur. J'ai donc éssayé de comprendre les différences entre mon programme et le leur. La compréhension de mes échecs a été riche d'enseignement. Il y avait en fait deux types de différences(mais qui sont liés) : des différences sur la facon de coder les symboles et des différences dans la normalisation de la transformée en ondelettes.

3.3.1 Différences sur la façon de coder

Pour coder les symboles POS, NEG, IZ, et ZTR, j'ai choisi une version naïve : puisqu'il n'y a que 4 symboles, 2 bits suffisent donc pour coder le bon symbole. J'ai donc choisi de coder les symboles de la façon suivante : POS $\rightarrow$ 00 , NEG $\rightarrow$ 01 , ZTR $\rightarrow$ 10 , IZ $\rightarrow$ 11. Mais pour différencier cette passe des autres, j'ai coder la longueur sur quelques bit *avant* de coder la passe, afin de savoir quand elle se termine (et donc permettre au décodeur de savoir quand commence la passe de raffinage). Or, les chercheurs ayant implémenté l'algorithme ont choisi un autre codage qui est plus "entropique"(c'est-à-dire qui attribue moins de bits aux symboles dont la probabilité d'occurence est forte, et réciproquement) : NEG $\rightarrow$ 1110, POS $\rightarrow$ 110, IZ $\rightarrow$ 01 et ZTR $\rightarrow$ 0. Et au lieu de coder la longueur comme je l'ai fait, ils ont choisi de coder un séparateur '1111'. On déduit facilement que leur choix est plus optimal que le mien si et seulement si il y a véritablement une trés grande majorité de zerotree. Et c'est le cas ! Mais, en vérité, d'aprés mes tests, la différence n'est pas tellement importante si la normalisation est celle que j'avais pris au départ, comme nous allons le voir immédiatement.

3.3.2 Différences sur la normalisation

Comme je l'ai précisé dans la partie précédente, la normalisation que l'on choisit quand on fait une transformation en ondelettes importe peu dans la compréhension globale des ondelettes mais influe beaucoup sur le taux de compression. En effet, j'ai choisi naïvement la première normalisation que j'ai vu lors de ma recherche documentaire : ainsi pour passer de LL_k à LL_{k+1}, LH_{k+1}, HL_{k+1} et HH_{k+1}, je prenais la moyenne simple d'un carré de 4 points (pour LL_{k+1}) ce qui correspond au quart de la somme, je prenais le quart de la moyenne des différences horizontales (pour HL_{k+1}), enfin bref, je normalisais par $\frac{1}{4}$. Alors qu'eux, normalisent par $\frac{1}{2}$. En fait, mathématiquement, je faisais une normalisation $\mathcal{L}^1$ (la somme des valeurs absolus des coefficients de normalisation vaut 1, en effet $\left|\frac{1}{4}\right|+\left|\frac{1}{4}\right|+\left|\frac{1}{4}\right|+\left|\frac{1}{4}\right| = 1$), alors qu'eux faisaient une normalisation $\mathcal{L}^2$ (la somme des carrés des coefficients de normalisation vaut 1, en effet $\left(\frac{1}{2}\right)^2+\left(\frac{1}{2}\right)^2+\left(\frac{1}{2}\right)^2+\left(\frac{1}{2}\right)^2 = 1$). C'est cette dernière base qui permet de rendre la base de Haar orthonormale et non uniquement orthogonale. Alors on peut se demander ce que change véritablement cette normalisation. La réponse est assez subtile, car elle est trés fortement lié à l'algorithme EZW et à la façon dont ils ont codé les symboles. En effet, quand je prend le quart de la somme,(et donc la moyenne simple), les coefficients des différentes sous-bandes restents du même ordre, si on excepte la décroissance "naturelle" des coefficients lors de l'augmentation des fréquences. En effet sur un carré de type $\begin{matrix} 1 & 1 \\ 1 & 1 \end{matrix}$, on obtient 1 comme coefficient avec $\frac{1}{4}$ comme coefficient de normalisation. Alors qu'avec leur choix, quand on passe d'une sous-bande à une autre de niveau superieure, l'ordre de grandeur des coefficients est doublé. En effet, en prenant la moitié de la somme des coefficients, on obtient, par exemple, 2 comme coefficient sur un carré de type $\begin{matrix} 1 & 1 \\ 1 & 1 \end{matrix}$. Donc d'une sous-bande à l'autre, les coefficients sont, en gros, doublés. Or, apres chaque phase de l'algorithme, on divise le seuil par 2, donc la phase code essentiellement les éléments d'un seul niveau de sous-bandes. Donc, les fils des points de ses sous-bandes seront trés certainement plus petit que le seuil courant(encore plus que pour la normalisation $\mathcal{L}^1$). Et donc cela maximise le nombre de zerotree ! ! Et donc, cela rend encore plus optimale leur codage des symboles. En effet mes tests prouvent que la différence entre les deux façons de coder n'est pas trés grande pour une normalisation $\mathcal{L}^1$, mais l'est pour une optimisation $\mathcal{L}^2$! ! Aprés coup, je me suis dit que j'aurais dû me douter qu'il fallait que je change de normalisation, car il parait normal qu'une décomposition dans une base orthonormale est meilleure qu'une décomposition dans une base seulement orthogonal.

En guise de conclusion, j'ai décidé de présenter une image(lenna) compressée à l'aide de cet algorithme : le taux de compression 0.16 bits par pixels pour un PSNR de 22.84 dB.

FIGURE 8 – Image de lenna compréssé via EZW

4 EBCOT

L'algorithme EBCOT, bien que basé sur une transformée en ondelettes comme l'algorithme EZW, diffère énormément de ce dernier par sa logique :pas de quadtree, de seuil qui diminue uniformément, etc... Cet algorithme est considéré comme un des plus performants algorithmes de compression d'image aussi bien en termes de compression pure qu'en termes de fonctionalités. C'est pourquoi, c'est cet algorithme qui a été choisi pour devenir le coeur du format JPEG2000. Il a été conçu par David Taubman, un chercheur australien. Je vais ainsi tenter d'expliquer les grandes lignes de cet algorithme. Mais je tiens à préciser, dés maintenant,que contrairement à ce que j'ai fait pour EZW, je ne vais pas décrire l'algorithme avec une trop grande précision, tout simplement car, EBCOT est beaucoup plus complexe que EZW, et que même l'auteur a mis plus d'une dizaine de pages pour expliquer son algorithme, et il n'est même pas rentré dans tous les détails ! Je présenterai globalement cet algorithme, avant d'expliquer ma tentative d'implémenter cet algorithme en MATLAB.

4.1 Le principe

L'algorithme EBCOT commence, comme EZW par une décomposition en ondelettes, mais cette fois, celle-ci doit être incomplète(nous verrons pourquoi). Ensuite, on effectue une quantification des coefficients obtenues sachant que le taux de quantification dépend de la sous-bande. Ensuite, on divise l'image ainsi obtenu en plusieurs code-blocs(codeblock en anglais). Le code-bloc est le concept central de EBCOT (un peu comme le quad-tree pour EZW). Un code-bloc est, en fait, une sous-matrice carré de la matrice image et qui est de taille 32x32 ou 64x64. De plus, un code-bloc doit faire partie d'une sous-bande, donc, un code-bloc étant de taille fixe(32x32 ou 64x64), si la transformée en ondelette serait complète, alors LL_n serait réduit à un simple point, et donc un code-bloc contiendrait plusieurs sous-bande, alors que justement, un code-bloc doit faire parti d'une sous-bande. On comprend alors pourquoi la transformation en ondelette ne peut être complête, le niveau m maximum de la transformation est celui pour lequel la sous-bande LL_m a exactement la même taille qu'un code-bloc. Une fois la division en sous-blocs effectué, on entre dans le coeur de l'algorithme que je vais maintenant tenter d'expliquer.

L'algorithme se décompose en deux grandes phases que je vais décrire successivement :

La premiere phase consiste à compresser chaque code-bloc de manière totalement indépendante : chaque code-bloc est compressé sans regarder les autres code-blocs, même les code-blocs adjacents. De plus, chaque code-bloc est codé de manière scalable. En effet, la compression de chaque code-bloc, donne un flux binaire qui peut-être décomposé en plusieurs parties. Ainsi, en décodant, uniquement la première partie, on a une représentation grossière du code-bloc entier, en décodant les deux premieres parties, on a une représentation un peu moins grossière,etc... . Entrons un peu plus dans le détail de cette passe : comment cette passe arrive-t-elle à créer un flux scalaire ? En codant par plan de bit. Un plan de bit étant un tableau représentant les bits d'un certain niveau. Par exemple, on peut décomposer le vecteur (7,3,4) en trois plan de bits : (1,0,1) , (1,1,0) et (1,1,0) car $7 =_2 111$, $3 =_2 011$ et $4 =_2 100$. Ainsi, la compression se fait plan de bit par plan de bit, du plus signifiant au moins signifiant. Ainsi, garder que la première partie du flux binaire engendré par la compression du code-bloc, revient à ne garder que le premier plan de bit (le plus significatif). Mais, l'algorithme va encore plus loin, vu qu'il code tous les plan de bits (excepté le premier) en trois passes, ainsi chaque partie du flux binaire représentant une passe, cela augmente encore plus la scalabilité du flux binaire. Ainsi, si on ne garde que deux parties du flux binaires, alors on ne garde que le premier plan de bit et une partie du deuxième(la partie, en général, la plus utile) : il faut garder 4 parties si on veut avoir les deux plans de bits en entier.

On a donc expliqué, pourquoi le flux binaire engendré est scalable, mais pas pourquoi, il est comprimé, c'est pourquoi je vais desormais tenter de répondre à cette question. Pour comprimer les fichiers, l'algorithme EBCOT utilise un codage arithmétique, qui est un codage entropique comme celui de Huffman, mais est plus performant que ce dernier. Le codage arithmétique(car il en existe plusieurs) utilisé par JPEG2000 est le codage MQ. Ce codage est adaptatif, dans le sens où il adapte son modèle de probabilité pendant le codage. De plus, c'est aussi un codage par contexte. Mais que signifie, ici , contexte ? Imaginons qu'on veuille compresser un texte composé de parties en anglais et d'autres en français, il serait judicieux de ne pas utiliser le même modèle de probabilité pour les parties en anglais que pour les parties en français : la lettre w, par exemple, ayant beaucoup moins de chance d'être utilisée dans la partie en français que dans la partie en anglais. Ainsi, pour un codage optimal, on utilisera différents modèles de propriété selon l'endroit du texte où l'on se trouve : c'est le contexte. Dans le cas qui nous interesse, les contextes sont, par exemple, la quantité de points voisins significatifs, la quantité de points voisins positifs, etc.... Ces contextes permettent de nombreuses petites optimisation, qui, en s'additionnant, donne un flux trés compréssé. L'établissement de tous ces contextes(EBCOT en définit 18 !) est basé sur le fait que les points significatifs sont en général, "en grappe", donc les points voisins des points de forte intensité ont de fortes chance, d'être eux-même des points de forte intensité.

Donc, la première phase engendre pour chaque code-bloc, un flux binaire comprimé décomposable en plusieurs parties.

La seconde phase va consister à choisir les parties de chaque flux binaires qu'on garde. Cette phase est, contrairement à la première, trés paramétrable. Toutefois, on peut donner une idée générale de ce qui est fait dans cette phase. On a donc, au départ, un certain nombre de flux binaire décomposables, de manière scalable, en plusieurs parties. L'algorithme

va consister à choisir quelle partie garder de chaque code-bloc. Ainsi chaque partie qu'on ne garde pas engendre une certaine distorsion(c'est-à-dire une certaine déformation de l'image, et cette déformation est quantifiable par exemple grâce au PSNR). Supposons qu'on ait une taille limite à ne pas depasser, notre but sera de minimiser la distorsion sous la contrainte d'une taille à ne pas dépasser, or ce problême est un problème classique d'optimisation sous contraintes(résoluble par la méthode des multiplicateurs de Lagrange). Ainsi, la seconde phase consiste à optimiser le choix des parties qu'on garde des différents code-blocs. Mais, EBCOT introduit encore un niveau de complexité : les couches de qualité. Les couches de qualité(qui sont un mécanisme optionnel d'EBCOT) consistent à l'idée suivante : on commence par optimiser le choix des parties à garder pour une contrainte sur la taille trés forte. et on place les parties gardées au début au fichier JPEG2000. Puis on refait l'optimisation mais cette fois avec une contrainte un peu plus faible, et on code les différences avec ce qui a déja été codé à la suite dans le fichier, et ainsi de suite. Les différentes couches ainsi ajouté au fichier sont les couches de qualité. C'est ce mécanisme qui permet à EBCOT d'être scalable, à l'instar d'EZW. Prenons un exemple, supposons qu'on s'impose une contrainte de 100 ko, et on décide de garder les trois premieres parties du code-bloc A et les deux premières du code-bloc B. On peut ensuite relâcher la contrainte, et rajouter à la fin du fichier les parties 4 et 5 du code-bloc A , et la partie 3 du code-bloc B.

Cette partie est extrêmement paramétrable, car on optimise comme on veut. Ainsi, on peut choisir de faire beaucoup de petites couches de qualités, et ainsi rendre le fichier trés scalable, ou au contraire tout coder en une seule grosse couche, et faire perdre au fichier toute scalabilité. On peut aussi choisir de prendre pour mesurer la distorsion, d'autres mesure que le PSNR. On peut aussi choisir de coder certains code-blocs plus que les autres même si c'est sous-optimal afin d'insister sur certaines zones : ainsi EBCOT dispose de la fonctionnalité de mise en place de "région d'interêt".

4.2 Tentative d'implémentation MATLAB

Conformément à l'objectif fixé dans le sujet du TER, j'ai tenté d'implémenter l'algorithme EBCOT en MATLAB. Mais, par manque de temps (consommé par la compréhension de l'algorithme), je n'ai pas eu le temps de finir mon programme : toutefois, il est quasiment terminé (la phase 1 est terminé, et je suis en train de deboguer la seconde) je ne désepere pas de finir ce programme avant ma présentation orale. Enfin, je précise que j'ai effectué certains choix simplificateur pour coder, comme par exemple j'ai remplacer le codeur MQ par un codeur arithmétique binaire adptatif avec contexte de mon cru, car je n'avais pas le temps de comprendre comment il fonctionnait.

5 Comparaison entre EZW et EBCOT

Concluons ce TER par la comparaison des deux algorithmes étudiées.

Première constation : EZW utilise les redondances entre sous-bande pour compresser, ce qui n'est pas le cas de EBCOT car les codes-blocs(et donc les sous-bandes) sont codés indépendamment les uns des autres. Mais cette absence d'exploitation de redondance est largement compensé par l'optimisation de la deuxieme phase.

Deuxième constation : les deux algorithmes sont scalables, mais EBCOT permet de choisir trés finement son niveau de scalabitité en choisisant comment on répartit les données entre les différentes couches de qualité, alors que la scalabilité de EZW est imposé par les propriétés de l'image.

Troisième constatation : EZW n'a pas la propriété de navigabilite : si on veut une zone précise de l'image, on n'est obigé de tout décoder, alors, que pour EBCOT,on peut choisir

de décoder uniquement les codes-blocs qui sont utile pour le décodage d'une zone précise, ainsi EBCOT, a un niveau de scalabilité plus fin que celui de EZW et, évite en même temps, le principal ecueil des algorithmes scalables(qui est de perdre la notion de navigabilité). On peut même organiser les couches de qualité afin de faciliter encore plus la navigabilité (on peut faire que la premiere couche ne code que la droite de l'image, la deuxième couche le milieu, etc...).

On déduit de toutes ces constatations que EBCOT est bien plus performant qu'EZW. Toutefois EBCOT a aussi des défauts, comme par exemple la taille arbitraire du code-bloc. Un autre défaut est son extrême complexité, qui fait qu'il existe relativement de code source implémentant l'algorithme : les seuls que j'ai trouvé étaient soit écrit par le comité JPEG2000 en tant que code de référence, soit par Taubmann, lui même. Enfin, il y a aussi des défauts à la décomposition en ondelettes et qui sont donc communes aux deux algorithmes : la décomposition en ondelettes est trop sensible, par sa contruction, aux différences verticales et horizontale, par rapport aux différences obliques.

J'ai donc, éssayé, au cours de ce TER de comprendre comment fonctionnent les algorithmes modernes de compression d'image, et je pense avoir réussi cet objectif, toutefois, je n'ai malheureusement pas fini d'implémenter EBCOT, qui fait que tous les objectifs de mon TER n'ont pas été remplis.

6 Bibliographie

- Au sujet de la transformée en ondelettes :
  - LA TRANSFORMATION EN ONDELETTES
    http ://perso.telecom-paristech.fr/ bloch/P6Image/ondelettestrsp.pdf
    Rene Alt

  - Ondelettes et application au traitement d'image
    https ://intranet.ensimag.fr/KIOSK/Matieres/5MMOATI/deafinal.ps.zip
    Valerie Perrier

- Au sujet de EZW :
  - RMF Based EZW Algorithm
    www.cs.ucf.edu/courses/cap5015/ezw_rmf.ppt
    School of Computer Science, University of Central Florida, VLSI and M-5 Research Group

  - Embedded Image Coding Using Zerotrees of Wavelet Coefficients
    http ://www.stanford.edu/class/ee398a/handouts/papers/Shapiro
    Jerome M. Shapiro

  - Compression d'images par ondelettes
    http ://perso.telecom-paristech.fr/ cagnazzo/doc/Compression%20images%20par%20ondelettes.pdf
    Frédéric Dufaux

- Au sujet de EBCOT :
  - High Performance Scalable Image Compression with EBCOT
    http ://citeseerx.ist.psu.edu/viewdoc/download ?doi=10.1.1.88.2909&rep=rep1&type=pdf
    David Taubman

  - Embedded block coding in JPEG 2000
    http ://www.infinitealgorithms.com/Docs/VideoLinks/jpeg2000_taubman.pdf
    David Taubman, Erik Ordentlich, Marcelo Weinberger, Gadiel Seroussi

  - Compression d'images par ondelettes
    http ://perso.telecom-paristech.fr/ cagnazzo/doc/Compression%20images%20par%20ondelettes.pdf
    Frédéric Dufaux

- Sur le codage arithmétique et le MQ-Coding :
  - Optimization of Arithmetic and MQ Coding
    http ://homepages.cae.wisc.edu/ ece734/project/s09/kolluru_rpt.pdf
    Krishna Bharath Kolluru

  - Embedded block coding in JPEG 2000
    http ://www.infinitealgorithms.com/Docs/VideoLinks/jpeg2000_taubman.pdf
    David Taubman, Erik Ordentlich, Marcelo Weinberger, Gadiel Seroussi