Compression d'images

ENSEA, 2e année

Les quatre séances de TP de majeure de Signal du semestre 8 consistent en un TP de compression d'images par ondelettes (6h) et un projet de codeur/décodeur JPEG simplifié (10h) dans l'environnement Matlab ou Octave. Les fichiers nécessaires sont accessibles sur Moodle.

L'évaluation se fera sur la base du travail en séance et d'un compte-rendu détaillé délivré à la fin de chaque partie. Ce compte-rendu devra comporter :

- un exposé clair de la problématique,
- un détail des voies explorées au cours des TPs,
- une synthèse présentant clairement les éléments validés, les éléments restant à tester, ceux à améliorer et ceux éventuellement manquants,
- une archive numérique contenant l'ensemble des codes produits.

Quelques fonctions utiles

- `I=imread('image1.bmp');` : lire un fichier image nommé `image1.bmp`
- `imagesc(I);axis image;colormap(gray);` : afficher l'image `I`
- `I1=rgb2gray(I);` : passer une image couleur `I` en une image en niveau de gris `I1`
- `I1=rgb2ycbcr(I);` : permet de passer une image couleur (RGB) en une image YC_bC_r `I1`
- `D=dctmtx(N);` : construire une matrice $N \times N$ de DCT. Pour calculer la DCT d'une image `A` de taille $N \times N$: `D*A*D'`.
- `dico=huffmandict(S,P);` : générer un dictionnaire pour un codage de Huffman pour les symboles `S` de probabilités `P`.
- `encode=huffmanenco(signal,dico);` : réaliser l'encodage de la séquence `signal` avec le dictionnaire `dico`.
- `deco=huffmandeco(encode,dico);` : réaliser le décodage de la séquence `encode` avec le dictionnaire `dico`.

TP : Compression d'images par ondelettes

L'objectif de ce TP est de réaliser la compression et la décompression d'une image à l'aide d'une décomposition en ondelettes, réalisée par un banc de filtres.

1 Introduction (source : wikipédia)

La technique de compression à base d'ondelettes offre une grande finesse au niveau de l'analyse du signal et permet de mieux s'adapter aux propriétés locales de l'image. La compression en ondelettes est en elle-même sans pertes, c'est l'introduction facultative d'une quantification ou d'un seuillage qui entraine la perte irréversible d'informations.

La première transformation par ondelettes est une technique inventée par Alfréd Haar en 1909, avec l'ondelette du même nom. En 1984, Jean Morlet, un ingénieur français, les utilise pour la prospection pétrolière et introduit le terme même d'ondelette qui fut traduit en anglais par wavelet, à partir des termes wave (onde) et le diminutif let (petite). Yves Meyer (prix Abel 2017), rassembla en 1986 toutes les découvertes précédentes puis définit les ondelettes orthogonales. La même année, Stéphane Mallat fit le lien entre les ondelettes et l'analyse multirésolution. Enfin, Ingrid Daubechies mit au point en 1987 des ondelettes orthogonales appelées ondelettes de Daubechies, et utilisées dans le standard JPEG 2000.

L'utilisation de cette transformation en imagerie consiste à décomposer une image en une myriade de sous-bandes, c'est-à-dire des images de résolution inférieure. La transformation en ondelettes provient d'une analyse multirésolution de l'image. On considère des espaces d'approximations et des espaces capturant les détails perdus entre chaque niveau d'approximation. Les bases ondelettes se situent sur les espaces "détails". Le choix de l'ondelette mère est très important et fait toujours l'objet d'expérimentations pour adapter l'analyse du signal image au système visuel humain.

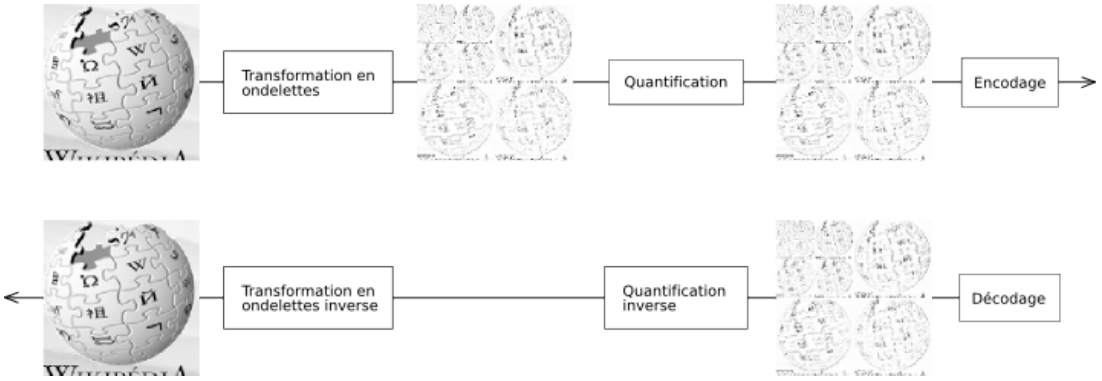


FIGURE 1 – Schéma de compression d'image par ondelettes.

La compression par ondelettes se compose des étapes suivantes (Figure 1) :
— Transformations par ondelettes.
— Quantification : les valeurs des images de détails inférieures à un certain niveau sont éliminées, en fonction de l'efficacité recherchée. C'est cette étape qui introduit des pertes.
— Codage des valeurs restantes.

2 Décomposition en ondelettes et reconstruction parfaite

Dans cette partie, nous nous intéressons à l'analyse de signaux à l'aide de leurs d'ondelettes. Commencer par :
— charger le fichier `PieceRegSig` avec l'instruction `load`. Celle-ci permet de créer un certain nombre de signaux test (sig).
— tracer la courbe correspondante.

2.1 Première décomposition dans la base de Haar

Dans cet exercice, vous allez apprendre l'adressage partiel d'un vecteur, et comment sous-échantillonner un signal.
— créer un vecteur e de longueur moitié contenant les moyennes de deux valeurs consécutives de sig, la première des deux valeurs utilisées ayant toujours un indice impair. Mathématiquement, cela s'écrit $e[k] = (sig[2k-1] + sig[2k])/2$. Utilisez pour cela des indexages partiels dans sig.

— tracer e (appelé *approximation*) sur la fenêtre 1 en exécutant `figure(1)` avant `plot(e)`.
— créer un vecteur w de longueur moitié contenant les demi variations de deux valeurs consécutives de sig, soit $w[k] = (sig[2k] - sig[2k-1])/2$;
— tracer w (appelé *details*) sur la fenêtre 2.

Voilà votre première décomposition dans la base de Haar ! (à $\sqrt{2}$ près). (le script `solution_2_1.m`)

Remarquez que e diffère peu du signal d'origine, surtout dans ses parties lisses. Par contre, w n'est important qu'au voisinage des sauts. C'est normal, puisqu'on le calcule par différences finies.

2.2 Convolution

Vous savez déjà sous-échantillonner un signal. Pour effectuer des décompositions sur des bases d'ondelettes plus générales, il faut pouvoir implémenter des convolutions. L'opérateur Matlab `conv` calcule des convolutions. La convolution de deux suites $(a_n)_{n\in Z}$ et $(b_n)_{n\in Z}$ est définie par : $(a * b)_n = \sum_{k\in Z} a_k b_{n-k}$

• Lire et lancer le script `convHaar.m` qui compare des implémentations « à la main » de la décomposition sur le base de Haar avec des implémentations utilisant l'opérateur de convolution.

Implémenter une cascade simple par un banc de filtres de Daubechies.
— charger le fichier 'PieceRegSig' avec l'instruction 'load'.
— charger le banc de filtres de Daubechies à deux moments nuls à partir du fichier 'Daub4'. Quatre variables sont alors chargées dans Matlab : h, g, rh, rg. Les deux premiers sont les filtres d'analyse (respectivement passe-bas et passe-bande), et les seconds sont les filtres de reconstruction.
— constater que g s'obtient a partir de rh en changeant un signe sur deux ; même observation pour rg et h.
— implémenter le schéma fonctionnel de décomposition dans Figure 2 (implémenter le sous-échantillonnage : `e=e_conv(1:2:length(e_conv));`).

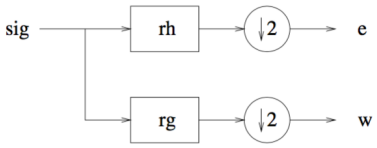


FIGURE 2 – Décomposition en ondelettes.

Le support du filtre rh de Daubechies-4 est [0,3], h est le miroir du filtre rh, son support est [-3,0] (en effet, le miroir F d'un filtre f est defini par $F(n) = f(-n)$).

Que remarque-t-on aux bords de e et w ? Quelle est la cause de ce phénomène ?

2.3 Reconstruction

Les filtres de Daubechies sont des filtres à reconstruction parfaite. Cela que le diagramme de Figure 3 suivant fonctionne :

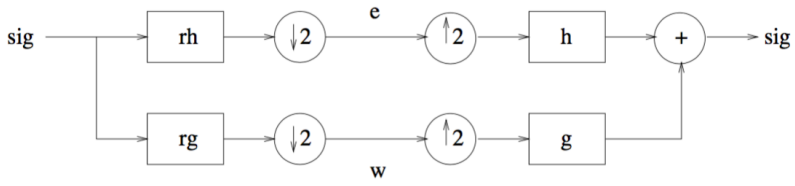


FIGURE 3 – Banc de filtres d'ordre 2.

— la partie gauche du schéma (banc d'analyse) a été implémentée dans ce source lors de l'exercice précédent ; il reste à effectuer la reconstruction (banc de synthèse). Implémenter le sur-échantillonnage par insertion de zéros, puis la convolution.
Astuce : La création d'un vecteur ligne nul de longueur n se fait par l'instruction : zeros(1,n). Une fois crée un vecteur ligne de longueur adéquate, il suffit d'y recopier les signaux e et w selon des espacements de 2 pour obtenir le sur-échantillonnage par insertion de zéros.
Calculer le support du résultat et tronquer pour retrouver sig : les filtres h et rh sont de support [0,3]. La

convolution de x avec rh s'ecrit : $y[n] = rh[0] * x[n] + rh[1] * x[n-1] + rh[2] * x[n-2] + rh[3] * x[n-3]$. y est nul si $n-3 > 1023$ ou $n < -3$; le support de y est donc [-3,1026]. Pour récupérer sig, il faut éliminer trois valeurs à gauche et trois valeurs à droite.

— tracer en figure 4 la différence entre la reconstruction et le signal d'origine.

— tracer sig en figure 1, e en figure 2 et w en figure 3.

La reconstruction n'est pas numériquement parfaite. Quelle est l'ordre de grandeur de l'erreur ? Pour vérifier l'importance de la troncature lors de la reconstruction, qui peut s'interpréter comme une synchronisation avec le signal de départ, refaite la troncature en la décalant de 1 et tracez en figure 5 la différence avec le signal d'origine.

3 Compression des signaux 1D

Afin de faciliter le reste du TP, plusieurs fonctions et fichiers sont fournies :

• `GetFiltres.m` : renvoyer les jeux de filtres associés aux différentes ondelettes utilisées

• `DownFilter.m` : réaliser la décompostion

• `UpFilter.m` : réaliser la reconstruction

• `SignauxTypiques.m` : générer des signaux de types différents.

Les ondelettes utilisées pour la compression sont des ondelettes biorthogonales, qui correspondent à des filtres de longueurs impaires. La norme JPEG2000 précise deux types :

• les ondelettes 5/3 : $rh[n]$ et $g[n]$ de longueur 5, $rg[n]$ et $h[n]$ de longueur 3.

• les ondelettes 9/7 : $rh[n]$ et $g[n]$ de longueur 9, $rg[n]$ et $h[n]$ de longueur 7.

Les ondelettes orthogonales (telles que les ondelettes de Haar ou de Daubechies dans les parties précédentes) sont associées à des filtres de même longueur, obligatoire paire. Ces ondelettes sont moins bien adaptées à la compression.

Remarque : les filtres orthogonaux étant toujours de longueurs paire, le fichier `GetFiltres.m` leur ajoute un zéro en fin de réponse pour les rendre compatibles avec les fichiers `DownFilter.m` et `UpFilter.m`. Pour réaliser le banc de filtres, vous serez confronté(e)s au problème du retard introduit par les filtre, qui doit être compensé pour effectuer la reconstruction. Un paramètre (nommé type) est prévu pour cela dans les fonctions `DownFilter` et `UpFilter`. Le script `solution_3_todo.m` explique comment renseigner ce paramètre ('a_o' : 'approximation + orthogonal', 'd_o' : detail + orthogonal', 'a_b' : 'approximation + biorthogonal', 'd_b' : 'detail + biorthogonal').

Compléter le script `solution_3_todo.m` pour réaliser le banc de filtres (Figure 3) :

— Utiliser de différents signaux et vérifier la reconstruction parfaite, pour les ondelettes Haar et 5/3 (tracer les "approximation" et "detail")

— Effectuer ensuite la reconstruction après une compression brutale obtenue par suppression de la composante w.

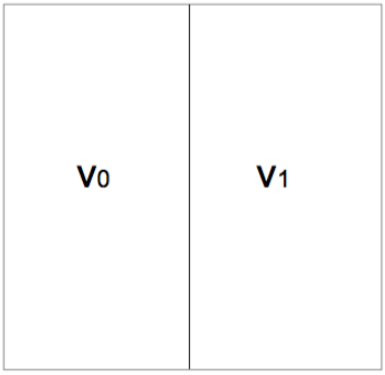
— Comparer les amplitudes des erreurs de reconstruction pour les ondelettes 5/3 et 9/7, pour le signal 'ligneLena'.

Conseil : pour de différentes ondelettes, verifier la capacité à concentrer l'énergie des signaux sur peu de coefficients, la complexité de l'implémentation.

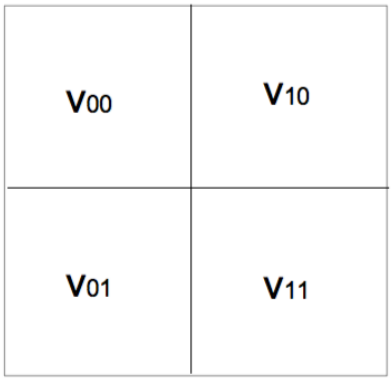
4 Compression d'une image

Une image est un tableau de taille $[M, N]$ avec M le nombre de lignes de l'image et N le nombre de colonnes. Dans ce TP, on suppose que M et N sont pairs. Nous allons travailler avec les images en niveau de gris où chaque pixel est codé sur un octet (valeurs comprises entre 0 et 255). La décompression d'une image se fait en appliquant une décomposition mono-dimensionnelle sur chacune des ses lignes, suivi d'une décomposition sur chacune de ses colonnes. La recomposition se fait de la même façon. L'ordre des opérations (lignes puis colonnes ou colonnes puis lignes) est indifférent.

La décomposition d'une ligne de N points donne deux signaux $v_0[n]$ et $v_1[n]$ de $N/2$ points chacun, que l'on concatène dans cet ordre, pour retrouver un signal de N points qui peut remplacer la ligne d'origine. On obtient alors une image composée de deux parties :



On peut effectuer la même chose sur les colonnes, ce qui donne une image composée de quatre parties :



Si M et N sont multiples de quatre, la décomposition dyadique peut être effectuée sur deux niveaux, en décomposant la composante v_{00}.

- Écrire un script qui effectue la décomposition et la reconstruction parfaite d'une image, sur un ou deux niveaux. Afficher les différentes images comme indiqué ci-dessus (voir `imread` et `imagesc`).
- Effectuer une compression en supprimant la composante v_{11}. Interpréter les défauts de reconstruction observés en fonction du type des ondelettes utilisées.
- Effectuer une compression en mettant à 0 un certain pourcentage des coefficients dont l'amplitude est la plus faible. Tester avec de différents nombres des valeurs supprimées.

Conseil :

- Pour de différentes ondelettes, vérifier la capacité à concentrer l'énergie des images, la complexité de l'implémentation.
- Pour chaque image compressée, calculer l'image d'erreur et le PSNR qui mesure la qualité de reconstruction de l'image compressée par rapport à l'image originale (un PSNR entre 30 et 50 dB pour la compression d'images est très bien pour la perception humaine). Rappel : $PSNR = 10 \log_{10}(\frac{255^2}{MSE})$ avec $MSE = \frac{1}{MN} \sum_{i=1}^{M} \sum_{j=1}^{N} (f_{i,j} - \tilde{f}_{i,j})^2$. Comparer la qualité de différentes images compressées.

En pratique, la compression est bien évidemment faite de façon plus élaborée, en quantifiant différemment les différents signaux v_{00}, v_{10}, v_{01} et v_{11}, afin de rendre les défauts imperceptibles.

Projet : réalisation d'un codeur/décodeur JPEG simplifié

1 Structure d'une image (non compressée)

L'image de départ est un tableau de M sur N pixels. Pour une image en niveau de gris, chaque pixel est codé sur un octet (valeurs comprises entre 0 et 255).

Pour une image couleur, chaque pixel est codé sur trois octets, qui représentent les intensités des trois composantes couleur : le rouge, le vert et le bleu.

Conversion en luminance/chrominance

Une autre représentation que le RGB est souvent utilisé : le YC_bC_r, qui correspond à la luminance Y (intensité du pixel en niveau de gris) et deux chrominances (une rouge C_r et une bleue C_b). L'image est alors constitué par trois tableaux d'octets, associés respectivement aux trois grandeurs Y, C_r, C_b. Les formules permettant de passer d'une représentation à l'autre sont les suivantes :

$$\begin{aligned} Y &= 0.299R + 0.587G + 0.114B \\ C_b &= -0.1687R - 0.3313G + 0.5B + 128 \\ C_r &= 0.5R - 0.4187G - 0.0813B + 128 \\ R &= Y + 1.14020(C_r - 128) \\ G &= Y - 0.34414(C_b - 128) - 0.71414(C_r - 128) \\ B &= Y + 1.77200(C_b - 128) \end{aligned}$$

Remarque : R, G, B sont compris entre 0 et 255. Ces formules peuvent donner les valeurs YC_bC_r qui sortent de cet intervalle, et qui devront être saturées si nécessaire afin de les garder comprises entre 0 et 255.

Deux formats différents sont fréquemment utilisés :

• dans le format le plus simple et le moins compressé, les trois tableaux Y, C_r, C_b ont les mêmes dimensions, M et N, égales aux dimensions de l'image originale.

• un format plus efficace en terme de compression (et donc plus souvent utilisé) consiste à sous-échantillonner les deux signaux de chrominance d'un facteur 2 (l'œil humain est moins sensible à la chrominance). Plus précisément, avec le facteur 2, le tableau Y reste de taille $[M, N]$ mais les tableaux C_r et C_b sont de dimension $[M/2, N/2]$ (la valeur de chrominance est associée à quatre pixels voisins).

2 Encodage d'une data unit

Le codeur JPEG travaille sur des data unit qui sont des blocs de taille 8×8 d'une image. Chaque composante est découpée en carrés de 8×8 pixels. Si les dimensions de la composante ne sont pas des multiples de 8, l'image est complétée par duplication de la dernière ligne (ou colonne) jusqu'à obtenir le multiple de 8 immédiatement supérieur. Chaque carré 8×8 est ensuite traité indépendamment, en décrivant l'image de gauche à droite et de haut en bas.

2.1 Centrage

Les échantillons des carrés 8×8 sont des nombres compris entre 0 et 255. La première opération à réaliser est un centrage afin d'obtenir des valeurs comprises entre -128 et 127 (en retirant 128 à chaque valeur). Les valeurs centrées seront notées $f(i, j)$ avec $i, j \in [0, 7]$ (ou $i, j \in [1, 8]$ en Matlab).

Pourquoi est il nécessaire cet étape ?

2.2 Transformée en cosinus

Après avoir réalisé le centrage, il faut calculer la transformée en cosinus discrète (DCT) de chaque carré 8×8. Cette transformation donne une nouvelle matrice 8×8 de coefficients, appelée $F(u, v)$, $u, v \in [0, 7]$. Cette transformée est une variante de la transformée de Fourier. Elle décompose un bloc, considéré comme une fonction numérique à

deux variables, en une somme de fonctions cosinus oscillant à des fréquences différentes. Chaque bloc est ainsi décrit en une carte de fréquences et en amplitudes plutôt qu'en pixels et coefficients de couleur. Pour le décodage, on peux retrouver les $f(i,j)$ à partir de la transformée 2D inverse des $F(u,v)$. Les équations de cette transformée ainsi que de la transformée inverse :

$$F(u,v) = C_u C_v \sum_{i=0}^{7} \sum_{j=0}^{7} f(i,j) \cos\left(\frac{(2i+1)u\pi}{16}\right) \cos\left(\frac{(2j+1)v\pi}{16}\right) \tag{1}$$

$$f(i,j) = \sum_{u=0}^{7} \sum_{v=0}^{7} C_u C_v F(u,v) \cos\left(\frac{(2i+1)u\pi}{16}\right) \cos\left(\frac{(2j+1)v\pi}{16}\right) \tag{2}$$

avec $C_0 = 1/\sqrt{8}$ et $C_u = 1/2$ si $u \neq 0$. L'application de la DCT est une opération théoriquement sans perte d'informations : les coefficients initiaux peuvent être retrouvés en appliquant la DCT inverse au résultat de la DCT.

Ces transformation en deux dimensions sont séparables car elles peuvent être réalisées en appliquant des transformations en une dimension, successivement sur les lignes et les colonnes.

En Matlab, la fonction `D = dctmtx(8)` permet de calculer le DCT d'un bloc de 8×8, pour calculer la DCT d'une imagette `A` de taille 8×8, utiliser : `D*A*D'`.

Conseil : afficher les DCT de différents blocs et interpréter les résultats.

2.3 Quantification

La quantification est l'étape qui permet de gagner le plus de place (la DCT n'effectue aucune compression). La DCT a retourné, pour chaque bloc, une matrice de 8×8 nombres. La quantification consiste à diviser point à point cette matrice par une matrice de quantification également 8×8. Soit Q la matrice de quantification. Le bloc 8×8 après compression sera obtenu par :

$$\hat{F}(u,v) = round\left(\frac{F(u,v)}{Q(u,v)}\right) \tag{3}$$

Le but est d'atténuer les hautes fréquences car l'œil humain y est très peu sensible. Ces fréquences ont des amplitudes faibles, et elles sont souvent ramenées à 0 après la quantification. Ces coefficients sont situés dans la matrice en bas à droite. Le but va être de ne garder que quelques informations essentielles (concentrées dans le coin en haut à gauche) pour représenter le bloc. Le reste de la matrice sera essentiellement composée de 0, ce qui va permettre d'utiliser un codage RunLength afin de gagner de la place.

Les matrices de quantification sont les éléments centraux de la compression avec pertes parce que c'est la quantification qui permet de régler les pertes, et donc le taux de compression. En toute rigueur, ces matrices devraient être calculées pour chaque image, en tenant compte du taux de compression désiré et des propriétés de l'image et de l'œil. En pratique, ce calcul est complexe, et l'on pourra dans ce projet utiliser les matrices pré-calculées.

Pour la quantification d'une composante RGB ou de la composante luminance, la matrice Q est :

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Pour la quantification d'une chrominance, la matrice Q est :

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

Remarque : dans un premier temps du projet, faire la quantification en utilisant ces matrices de quantification. Ensuite, tester la compression en modifiant les matrices de quantification avec un facteur de qualité. Soit fq le facteur de qualité (entre 1 et 100), définir s : si $fq < 50$, $s = 5000/fq$, sinon $s = 200 - 2 * fq$ et puis calculer la nouvelle matrice de quantification : $Q2 = floor((s * Q + 50)/100)$.

2.4 Parcours en zigzag, RLC, et codage de Huffman

Comme dit dans la section précédente, après la quantification, beaucoup de coefficients de la matrice $\hat{F}$ sont nuls et ils localisés en bas à droite de cette matrice. Le nombre de bits moyen de ces coefficients est donc très réduit. Afin d'exploiter cette propriété, le bloc 8×8 sera lu avec un parcours zigzag (Figure 4) afin de construire de longues plages de 0 (afin d'optimiser au mieux l'utilisation d'un RLC sur le symbole 0).

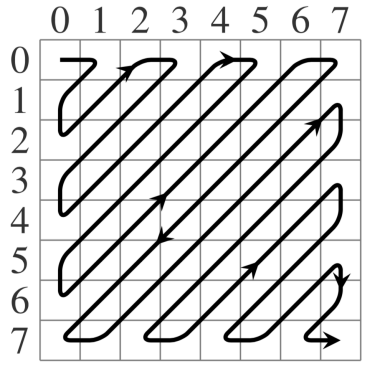


FIGURE 4 – Ré-ordonnancement Zigzag.

Une fois le vecteur contenant les 64 valeurs du bloc 8×8 construit, il faut effectuer un codage RunLength pour obtenir le vecteur Vrlc : dès qu'une plage de 0 est détectée, il faut préciser la longueur de cette plage, précédée du nombre 257.

Une fois les vecteurs codant le RLC codés, il faut effectuer un codage de Huffman pour les Vrlc (un dictionnaire pour tous les Vrlc de tous les blocs).

Remarque : En pratique, dans un bloc 8×8, on distingue deux types de coefficient : le coefficient $\hat{F}(0,0)$ est appelé coefficient DC, et les autres 63 coefficients sont appelés coefficients AC (qui sont très faibles ou nuls après quantification). Les coefficients DCs et ACs sont codés différemment.

DC : (DPCM Differential Pulse Code Modulation) la valeur DC d'un bloc correspond à sa valeur moyenne. En faisant l'hypothèse qu'elle est généralement voisine de celles des blocs voisins (vrai sauf en cas de changement brusque de couleurs ou de zones), la différence entre deux valeurs DC de blocs voisins de la même composante est plutôt faible. On encode donc les DCs par le codage différentiel et le code de Huffman.

AC : l'étape de quantification et le ré-ordonnancement ont eu pour but de mettre à 0 les hautes fréquences. On encode les ACs par le codage RLE et le code de Huffman.

3 Décodage d'une data unit

Le décodage d'une data unit va défaire tous les étapes précédentes les unes après les autres. La première étape consistera à décoder le code de Huffman de la data unit afin de reconstruire le vecteur Vrlc. De là, on peut reconstruire les 64 valeurs de bloc 8×8, puis inverser l'opération de quantification et de DCT.

4 Implémentation d'un codeur/décodeur JPEG simplifié

Les blocs nécessaires à la mise en œuvre du codeur/décodeur simplifié détaillé précédemment sont les suivants :

- Découpage en bloc de taille 8×8
- Centrage, DCT et quantification de chaque bloc
- RunLength coding
- Codage de Huffman
- Les fonctions inverses de chacun des points précédents

Dans un premier temps, commencer par travailler avec des images en noir et blanc. Par la suite, si tous les blocs fonctionnent correctement, passer à une image couleur en vérifiant d'abord avec la représentation RGB puis la représentation luminance/chrominances sans/avec les sous-échantillonnages . Ensuite, améliorer le système avec le facteur de qualité, coder les DCs et ACs différemment.

Conseil : Calculer, afficher et interpréter le résultat de différentes étapes (par exemple, le DCT ou la quantification de différents blocs).

Calculer le taux de compression, les erreurs, le PSNR.

Découper le programme en de différentes fonctions.

Le script `zigzag\_ex.m` présente une exemple pour le ré-ordonnancement Zigzag.