Compression et décompression d'image numérique

Alexis Vialaret et Lysandre Debut

***Abstract*— Nous avons implémenté en tant qu'exercice une compression combinant la transformée en ondelettes dicrète de Daubechies, la modulation par impulsion codée différentielle et la compression de Huffman. Nous démontrons qu'il est possible d'atteindre des taux de compression de 40.**

I. INTRODUCTION

La compression de données est une opération informatique qui sert à réduire la taille d'une suite de bits, tout en conservant l'information. L'information initiale peut-être en partie ou intégralement restaurée en effectuant une décompression des données précedemment compréssées. Les techniques de compression sont très utilisées dans différents domaines qui nécessitent une sauvegarde des données, car cela permet de stocker plus d'informations, de les transmettre plus efficacement et donc de réduire les coûts. On retrouve ces techniques dans le domaine de la vidéo, de l'image, de l'audio, ainsi que pour la majorité des types de données informatiques. La compression d'image est l'application de cette compression de données sur des images numériques. De multiples méthodes de compression d'image existent, certaines "sans pertes", qui restituent une image identique à l'image initiale après les étapes de compression puis de décompression, certaines "avec pertes", qui restituent une image altérée mais qui permettent une réduction de la taille des données plus importante. Deux grands axes de compression de l'image existent; le premier compresse l'image mais conserve sa lisabilité (exemple avec la compression JPEG), le deuxième ne conserve pas la lisabilité et l'image a besoin d'être décompressée avant de pouvoir être lue. C'est sur le deuxième axe que nous nous sommes concentrés pendant ce projet. Pendant celui-ci nous avons cherché à allier plusieurs méthodes de compression, de manière à obtenir une perte d'information minimale mais une réduction de taille maximale. Nous avons donc effectué une compression sans perte là où les informations sont importantes, et une compression avec pertes là où les informations le sont moins.

Afin de réaliser ce projet, nous avons utilisé le logiciel Matlab, qui permet de faire de la gestion de matrices et donc de la gestion d'images, de texte et de vidéo.

II. MÉTHODES

Pour obtenir une compression optimale, nous avons choisi d'appliquer plusieurs transformations sur l'image. Certaines transformations se font sans pertes : la transformée intercomposante, la transformée en ondelettes, le codage DPCM, RLE et Huffman, tandis que d'autres induisent des pertes : la quantification scalaire et le seuillage.

A. Séparation en tuiles

Les images que nous traitons sont composées de plusieurs zones distinctes. Nous cherchons à avoir un contenu le plus uniforme possible, de manière à avoir une majorité de l'information contenue dans les basses fréquences de l'image. La première étape de notre projet est de décomposer une image en tuiles qui seront ensuite compressées individuellement. Ces tuiles auront un contenu plus uniforme qui l'image initiale, et pourront être plus ou moins compressées en fonction de leur contenu, ce qui nous assure une efficacité optimale.

Nous donnons le choix à l'utilisateur du programme du nombre de tuiles sur lesquelles le programme effectuera la compression. Une quantité de tuiles élevée peut permettre une compression plus importante, mais l'algorithme aura un temps d'execution plus long à la compression comme à la décompression. L'utilisateur choisit entre 4 options de résolution de tuilage : 1, 2, 3 et 4. Le programme découpera ensuite l'image en n tuiles avec n le carré de cette résolution.

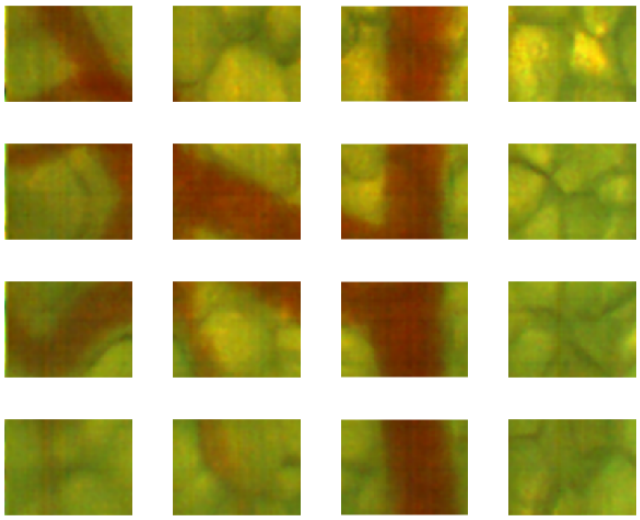

Fig. 1. Séparation en tuiles

B. Transformée intercomposante

Une banque de quatre images sont disponibles pour l'utilisateur, qui sont chacune en RGB24. L'utilisateur pourra faire le choix de rajouter ses propres images tant qu'elles sont dans le même format. Ce format permet d'avoir trois composantes pour décrire chaque couleur : une composante rouge, verte et bleue. Chaque pixel est codé sur les trois composantes, avec une valeur comprise entre 0 et 255

sur chacune des composantes (soit un octet par pixel par composante, et trois octets pour un pixel complet).

La première opération qui s'effectue sur les tuiles est la transformation de cette image aux composantes RGB vers une image aux composantes YCbCr. Cette manière de représenter l'espace colorimétrique permet ici aussi d'avoir trois composantes, la première décrivant la luminance, les deux suivantes représentant les chrominances bleues et rouges respectivement. On choisit de travailler avec cet espace colorimétrique pour effectuer nos opérations de compression car il tend à grouper la majorité des informations pertinantes en terme de perception visuelle dans sa première composante Y, contrairement à l'espace RGB dans lequel l'information est répartie uniformément. Cela implique qu'aux étapes de compression provoquant des pertes, on pourra altérer de manière plus significative les composantes Cb et Cr sans altérer de manière majeure la percéption de l'image pour l'utilisateur.

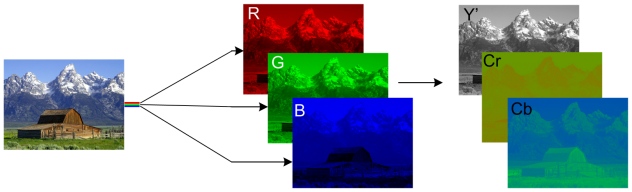

Fig. 2. Conversion RGB vers YCbCr

C. Transformée en ondelettes

La transformation en ondelettes est une transformée sans perte, qui permet de décomposer une image en quatre images de résolution inférieure. Ces images peuvent ensuite être combinées en utilisant une tranformée en ondelettes inverse, de manière à retrouver l'image originale. Ceci s'effectue en faisant une décompostion par bancs de filtres. L'image est filtrée par des filtres passe-bas et des filtres passe-haut. L'image est filtrée quatre fois, passant par les quatre combinaisons possibles de ces quatre filtres. Nous nommons les images en rapport avec les filtres par lesquels elles ont été filtrées :

- LL : l'image est successivement filtrée par deux filtres passe-bas
- LH : l'image est successivement filtrée par un filtre passe-bas puis un filtre passe-haut
- HL : l'image est successivement filtrée par un filtre passe-haut puis un filtre passe-bas
- HH : l'image est successivement filtrée par deux filtres passe-haut

Cette transformation nous permet de "trier" l'image originale en quatre types de composantes : les ondelettes basse-fréquences (LL) qui contiennent la large majorité des information perceptibles par un humain, les ondelettes de moyenne fréquences (HL et LH) qui contiennent les gradients verticaux et horizontaux de l'image, et les ondelettes

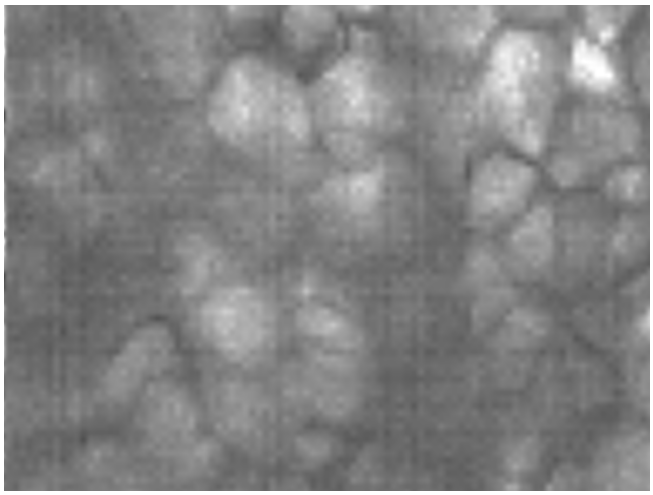

Fig. 3. Tuile de base

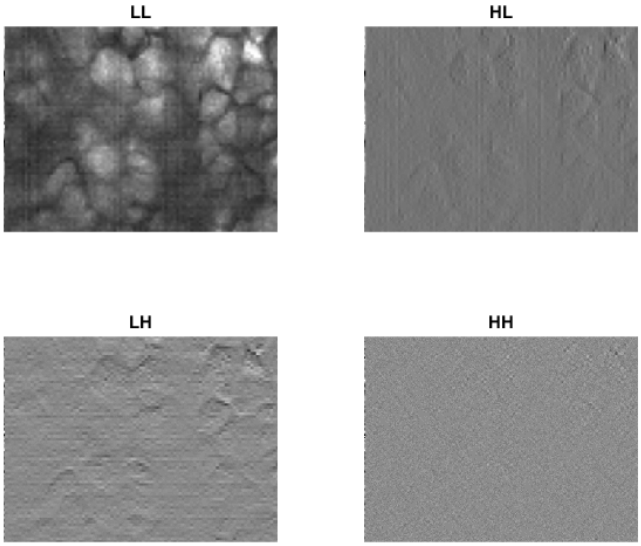

Fig. 4. Transformation en ondelettes de la tuile

haute-fréquences (HH) qui sont du bruit en terme de perception. La somme de ces quatres composantes est rigoureusement égale à l'image originale, cette transformation est donc réversible. Par conséquent, de la même manière que lorsque nous avons groupé l'information colorimétrique perceptuellement pertinante de notre image originale dans dans le canal Y, nous isolons ici la majorité de l'information fréquentielle pour laquelle l'oeil est sensible dans un quart de l'espace mémoire. Pour les étapes prochaines de compression, nous conservons au mieux cette information, et nous sacrifions une grande partie des données contenues dans les autres gammes de fréquences.

Notre programme permet à l'utilisateur de choisir le nombre d'itérations de la transformée en ondelettes. Dans la figure 5, l'utilisateur a choisi d'itérer une fois, sur une tuile. Une itération supplémentaire applique de nouveau la transformée en ondelettes sur l'image LL, comme visible sur la figure 6. Chaque itération va appliquer la DWT sur

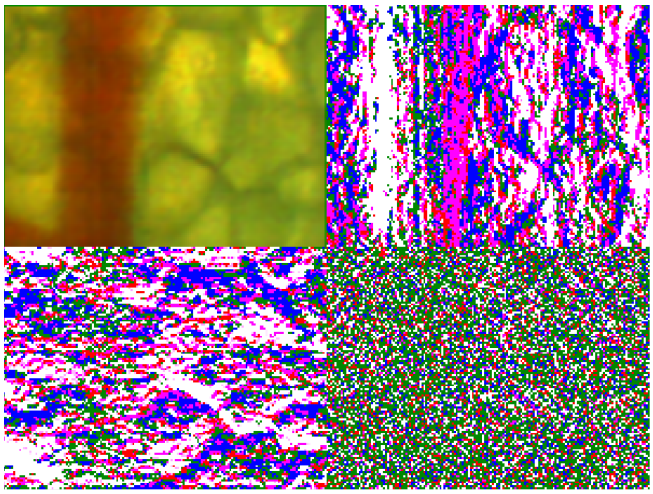

Fig. 5. Exemple de transformation en ondelettes d'une tuile de l'image en RGB (une itération)

la partie basse-fréquence du résultat de DWT précédent permettant ainsi de localiser l'information utile dans un très petit espace mémoire.

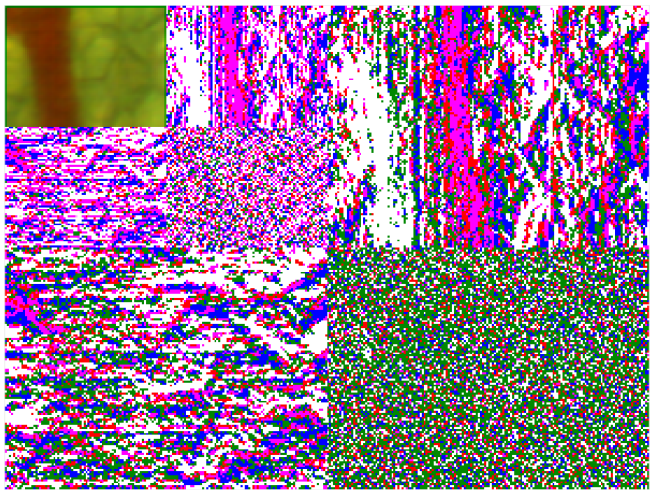

Fig. 6. Exemple de transformation en ondelettes d'une tuile de l'image en RGB (deux itérations)

Les dernières étapes de compression diffèrent en fonction du type de composante : on utilise une Differential Pulse-Code Modulation (DPCM) qui est une méthode de codage prédictif réversible de compression sans pertes si elle est utilisée sans quantification. Pour les hautes fréquences, on va simplement les quantifier et les seuiller de manière à tronquer une partie de l'information. Toutes les couleurs en dessous d'une certaine valeur de seuil seront mises à zéro, ce qui rendra la compression de Huffman utilisée ensuite très efficace.

D. Quantification des composantes hautes fréquences

La quantification est une opération surjective qui vise à placer un ensemble de valeurs de départ dans un nouvel ensemble plus petit en conservant ou non la répartition de ces valeurs. Cette opération provoque une perte d'informations, et élimine ainsi beaucoup d'informations peu pertinantes. On applique une forte quantification aux composantes hautes fréquences qui contiennent peu de valeurs visuellement intéressantes. On réduit de cette manière la diversité des mots binaires décrivant l'image, ce qui va rendre Huffman plus efficace.

E. DPCM de la composante basse fréquences

Pour sauvegarder le maximum d'informations dans les basses fréquences tout en les compressant au mieux, on utilise la DPCM. Le but ici est d'appliquer un prédicteur qui se base sur quatre pixels voisins connus afin de prédire le pixel courant. Pour ce faire, on applique à chaque pixel la formule suivante :

$$x = 1/4*[x(i-1,j-1)+x(i,j-1)+x(i+1,j-1)+x(i-1,j)] \quad (1)$$

Au lieu d'encoder la valeur du pixel, on encodera à la place la différence entre la valeur réelle et la valeur prédite x. Si le prédicteur est bon, ces valeurs d'erreur seront centrées (le plus proche possible) autour de zéro au lieu d'aller de 0 à 255 par exemple. Cela va introduire une redondance de certains mots binaires dans l'encodage de l'image, ce que l'algorithme de Huffman compresse extrèmement bien. L'efficacité du prédicteur est fortement liée à l'absence de gradients importants ou de discontinuités dans le signal, on peut supposer qu'il est donc particulièrement adapté ici puisqu'on l'applique sur les basses fréquences de l'image originale où les gradients de couleur sont faibles. le schéma bloc d'encodage est donc le suivant :

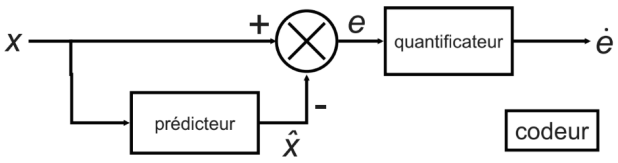

Fig. 7. Encodeur simple

L'inconvénient de cette méthode est la propagation de l'erreur au fur et à mesure du décodage. Si on balaye une image de gauche à droite et de haut en bas, le haut de l'image sera fidèle à l'original une fois décodé, alors que le bas sera altéré. Cet effet sera d'autant plus présent que l'image est grande. Pour contrer ce problème, nous devons tenir compte de l'erreur qui sera introduite par le décodeur au moment de l'encodage et le compenser. Pour connaître cette erreur, on pratique le décodage du pixel courant lors de l'encodage et on ajoute l'erreur de décodage à l'erreur du prédicteur. Avec cette simulation, le schéma bloc devient comme suit :

Le décodage de l'information obtenue est assez simple puisqu'il s'agit de l'opération inverse. Il nécessite néanmoins la conaissance du vecteur de prédiction utilisé lors de l'encodage. La valeur d'un pixel décodé est donc :

$$\dot{x} = \dot{e} + pred(x) \quad (2)$$

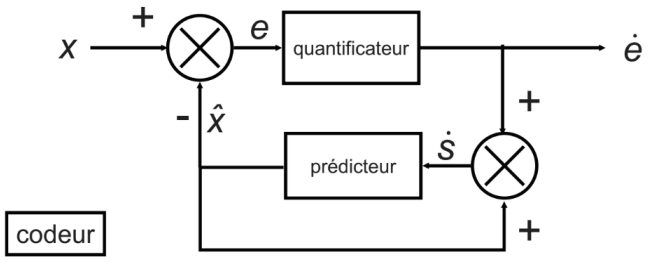

Fig. 8. Encodeur avec simulation du décodage

F. Formation du train binaire

Après avoir effectué toutes ces opérations visant à maximiser les redondances dans les valeurs encodées, il faut maintement les compresser. On utilisera ici l'algorithme de compression par arbres de Huffman. Cette méthode ne provoque pas de pertes de données supplémentaires et excelle dans la compression de fichiers comportant de nombreuses répétitions. Son fonctionnement est le suivant :

- Il prend en entrée le fichier binaire correspondant aux données à encoder. - Il compte le nombre de répétitions des mots binaires de ce fichier. - Il construit un arbre dont les feuilles sont ces mots binaires, plus le nombre de redondances pour un mot donné est important, plus la feuille sera proche de la racine. - Chaque mot binaire est ensuite encodé sous la forme du chemin menant à sa feuille (index) dans l'arbre. Comme les chemins sont courts pour les mots fréquents, la taille de leur index est nettement inférieure à celle du mot original, ils sont donc compressés.

Afin de préparer notre résultat pour l'algorithme de Huffman, il faut d'abord créer un vecteur binaire que l'on passera en paramètre à la fonction qui effectuera la compression. Ce vecteur binaire est composé de trois différents objets : une entête avec les caractéristiques de l'image et de la compression effectuée, un vecteur de deltas pour la quantification effectuée, et un train binaire contenant toutes les données de l'image créée. Le vecteur de l'entête, concaténé avec le vecteur de deltas regroupe les informations suivantes :

Longueur de la tuile
Largeur de la tuile
Nombre de tuiles
Nombre d'itérations de la transformée en ondelettes
Delta[1]
...
Delta[n]

Fig. 9. Vecteur de l'entête

Le train binaire est créé pendant que les transformations sont réalisées. Un vecteur est créé au tout début de la compression, vecteur qui fait la taille de l'image totale. Dans notre étude, l'image comprenait trois composantes de 640 colonnes et 480 lignes. Notre vecteur comprenait donc 921600 valeurs. Nous cherchons à respecter la forme imposée du vecteur binaire :

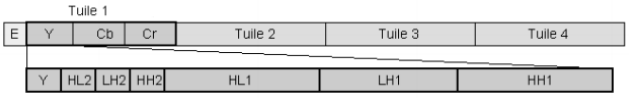

Fig. 10. Formation du train binaire

Lors de la création de tuiles, des vecteurs de la taille des tuiles sont créés pour accueillir les différentes valeurs des tuiles. A la fin de l'opération sur la tuile, le vecteur tuile est placé dans le vecteur binaire initial, qui sera ensuite renvoyé.

Le programme itère ensuite à travers les couches, ce qui implique la création d'un vecteur binaire pour chaque couche. Ce vecteur binaire sera rempli en partant de la fin, comme suit :

Lors de la première itération, la DWT a lieu. Grâce à une fonction "flatten" que nous avons programmée, les images quantifiées (HH, LH et HH) sont converties de matrice vers un vecteur ligne. Nous avons créé la fonction "unflatten" qui permet de revenir à la matrice à partir de ce vecteur ligne.

Le vecteur sera remplie par les vecteurs de HH quantifié, puis LH quantifié, puis HL quantifié. S'il n'y a qu'une seule itération, le vecteur ligne représentant LL après la DPCM remplira la dernière case disponible.

Cependant, s'il y a d'autres itérations, le LL sera redécomposé en d'autres images : HH2, HL2, LH2 et LL2. Le processus précédent sera réappliqué, remplissant ainsi le vecteur. Etant donné que la taille de l'image et que la somme des tailles des quatre images créées sont égales, le remplissage du vecteur se fera toujours correctement.

A la fin de ce processus, le vecteur binaire sera rempli et prêt à être traité par l'algorithme de Huffman.

G. Evaluation des résultats

Dans le but d'évaluer la performance du codage et de la compression, on se base sur deux métriques :

- Le taux de compression
- La différence de qualité entre l'image originale et l'image décompressée et décodée.

Le calcul du taux de compression est simplement :

$$TDC = \frac{taille\_binaire\_originale}{taille\_binaire\_compressee} \tag{3}$$

Mesurer la différence de qualité entre les deux images n'est pas aussi évident, car on peut difficilement modéliser précisément la vision humaine par des formules, ce qui fait de l'évaluation de la dégradation d'une image une tâche compliquée. Cependant, afin d'avoir quand même quelques éléments de réponse, on pourra utiliser les formules suivantes :

Erreur maximum :

$$MaxErr = max(\dot{x}(i,j), x(i,j)) \tag{4}$$

Erreur quadratique moyenne :

$$MeanErr = \frac{1}{MN} \sum_{i=1}^{M} \sum_{j=1}^{N} [x(i,j) - \dot{x}(i,j)]^2 \tag{5}$$

Rapport signal sur bruit crête :

$$PSNR = 10 Log_(10) \frac{(2^8)^2}{MSE} \tag{6}$$

On calculera aussi le score de corrélation entre l'image originale et l'image traitée

III. Résultats

Nous avons procédé par étapes lors de l'implémentation des différentes fonctions nécessaires à la réalisation du projet, nous allons donc présenter ces résultats de la même manière. Pour chaque étape nous présenteront la fonction implémentée ainsi que son opération inverse.

A. Séparation en tuiles

Nous avons crée une fonction qui demande à l'utilisateur le nombre de colonnes/lignes de tuiles qu'il souhaite avoir lors de la compression. Nous avons également la fonction inverse qui permet de reconstituer une image à partir des tuiles.

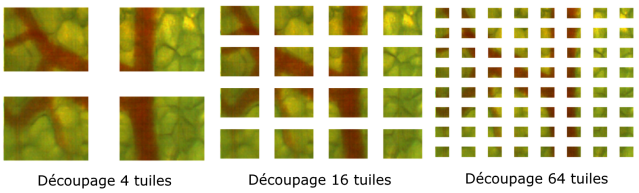

Fig. 11. Séparation en tuiles

B. RGB vers YCbCr et réciproque

A des fins pédagogiques, nous avons nous-même implémenté les conversions RGB vers YCbCr et YCbCr vers RGB.

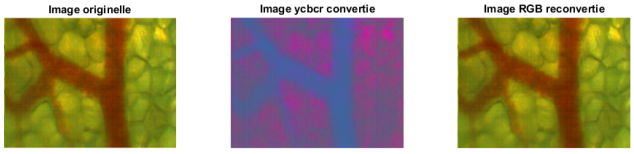

Fig. 12. Allers-retours colorimétriques

C. Transformée en ondelettes et réciproque

L'algorithme que nous avons utilisé pour la transformée en ondelettes est celui de Daubechi, que nous nommons DWT on obtient les résultats suivants :

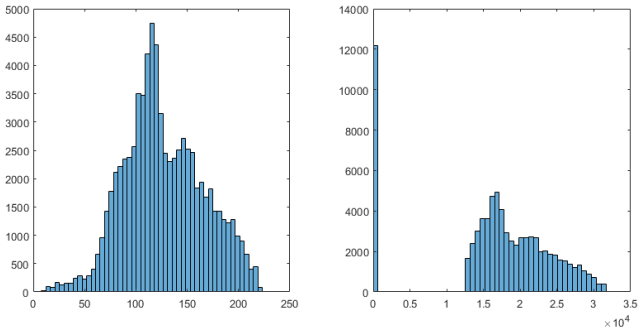

Fig. 13. Histogrammes avant et après quantification

D. Quantification et seuillage

L'evaluation de la quantification se fait en comptant le nombre de valeurs différentes dans l'image de départ et sa version quantifiée (fig.11):

On passe de 0% 16% de zéros après quantification et seuillage, avec un seuil de 50 sur 255

E. DPCM sur la composante basse fréquence

Les valeurs de sortie de notre fonction DPCM sont bien centrées autour de zéro, conséquence du codage différentiel. La variance de ces valeurs est faible, la compression sera efficace.

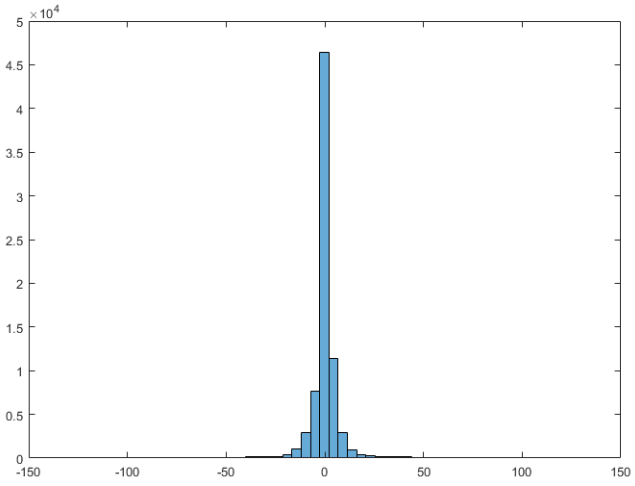

Fig. 14. Histogrammes de la sortie de DPCM

Nous avons relevé un problème sur l'impémentation de la DPCM inverse que nous n'avons malhereusement pas eu le temps de régler.

F. Evaluation de la performance de compression

Nous avons réussi à faire fonctionner la compression, ou du moins à générer un fichier. Nous obtenons les taux de compressions suivants :

Nous n'avons cependant pas eu le temps de coder la décompression qui nous permettrait de valider l'étape de compression. Nous ne pouvons donc pas appliquer les formules donnée dans la partie méthodes pour comparer l'image

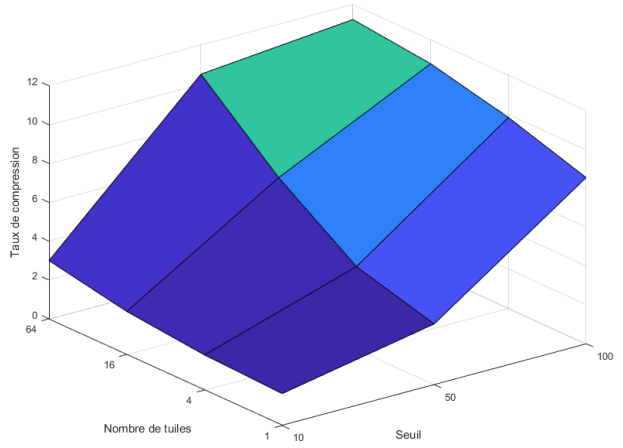

Fig. 15. Taux de compression pour une itération de DWT et des nombres de tuiles et de seuils différents

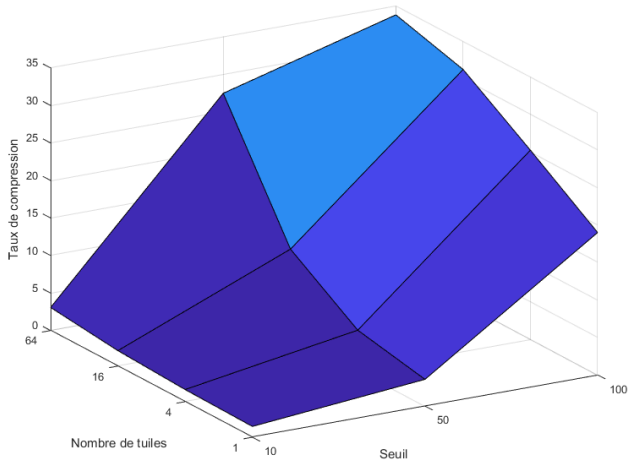

Fig. 16. Taux de compression pour deux itération de DWT et des nombres de tuiles et de seuils différents

originale et l'image décompressée, mais nous pouvons nous pencher sur l'impact des paramètres de compression comme le nombre de tuiles, le nombre d'itérations de DWT ou encore le seuillage choisi lors de la quantification.

IV. DISCUSSION

Nous sommes arrivés à implémenter cette méthode de compression sous MatLab dans sa quasi-totalité. Il nous manque seulement la DPCM inverse et la reformation de l'image codée à partir du train binaire.

Nous avons au fur et à mesure de notre avancement relevé plusieurs points sur lesquels nous pourrions agir afin de rendre la compression plus performante :

- Facteurs de compression différents pour les chrominances : nous avons choisi d'utiliser l'espace YCbCr du fait de sa densité d'information plus importante en Y. Nous pourrions exploiter ceci d'avantage pour notre compression en appliquant par exemple une quantification plus dure et un seuil plus important pour les chrominances que pour la luminosité.

- Prédicteur DPCM : nous avons choisi de manière arbitraire un vecteur de prédiction [0.25, 0.25, 0.25, 0.25]. Nous pourrions donner le choix à l'utilisateur de saisir son propre vecteur, ou mieux encore, calculer automatiquement le meilleur prédicteur pour une image donnée.

Nous avions une erreur dans le script de compression liée à notre utilisation généralisée des doubles. Nous l'avons résolue en utilisant un arrondi, mais nous ne sommes pas sûrs que cela n'altère pas certaines données, et aurions besoin d'un peu plus de temps pour s'assurer du contraire.

Comme nous n'avons pas pu produire la reconstitution de l'image à partir du train binaire, nous ne pouvons pas faire l'analyse qualitative visuelle de cette méthode ni de mesures quantitatives, ce qui est vraiment dommage car nous aurions pu comparer la méthode en terme de qualité avec d'autres compressions comme le JPEG à compression égale.

Il serait également intéressant de comparer le coût de la compression en temps, avec là aussi la comparaison avec d'autres algorithmes au but similaire.

Du point de vue personnel, nous avons trouvé le projet très intéressant et complet. Il faisait appel à de nombreuses conaissances et nous a parmi d'en apprendre beaucoup de nouvelles. Nous avons fait l'erreur au début du projet de démarrer un peu trop vite et d'implémenter les fonctions proposées sans vraiment savoir pourquoi. Nous aurions du nous poser, et prendre du temps pour comprendre la méthode dans sa globalité dès le début cela nous aurait permis d'avoir une vision d'ensemble et d'éviter des erreur bêtes. Arrivés à la fin du projet, nous avons vraiment le sentiment d'avoir compris et intégré tous les concepts théoriques nouveaux que nous avons exploités pour mettre en place la compression. Le principal frein et source de blocage ne réside pas dans la difficulté théorique des outils de théorie du signal, mais plutôt dans leur implémentation dans MatLab que nous ne maîtrisons pas encore bien.