TP2 - Compression JPEG

Axel REMACK - Stacy VARLOTEAUX

12 octobre 2021

1 Partie 1 - Transformée et quantification dans la norme JPEG

1.1 Algorithme d'encodage - *Fonction JPEGEncoder*

Trois étapes sont nécessaires dans cet algorithme :

- **Décalage de niveau** : pour chaque pixel de l'image à compresser, on va soustraire 128 à sa valeur.
- **Transformée en cosinus discrète (DCT)** : on va découper l'image à compresser en bloc 8x8 puis appliquer un calcul visant à regrouper l'énergie (en transformant un niveau de gris en un ensemble de coefficients) pour que la compression soit plus facile par la suite (*Fonction DCT*).
- **Quantification** : c'est l'étape qui va assurer la compression des données, en divisant la valeur de chaque pixel par le coefficient correspondant d'une matrice de quantification puis en arrondissant le résultat.

Suite à cela et pour les questions ultérieures, nous avons isolées les coefficients DC (celui de coordonnée (0,0) dans chaque bloc 8x8) et AC (les 63 autres coefficients du bloc 8x8) de chaque bloc dans des vecteurs. Les coefficients DC sont ajoutés au vecteur sous forme de différence (DIFF = DC(i) - DC(i-1)) (*Fonction IsolateACDC*).

Le résultat à la sortie de cette fonction est assez surprenant au premier abord : les informations importantes sont toutes isolées dans les coins haut-gauche de chaque bloc 8x8 et tout le reste est gris (car peu d'informations contenues dans ces pixels). On peut tout de même reconnaître vaguement la silhouette de Léna sur cette image compressée.

1.2 Algorithme de décodage - *Fonction JPEGDecoder*

Pour inverser le traitement fait précédemment et obtenir une image décompressée, on effectue l'inverse des opérations faites lors de l'encodage :

- **De-quantification** : on multiplie les pixels de notre image compressée par le coefficient correspondant de la matrice de quantification, toujours en arrondissant la valeur.
- **Transformée en cosinus discrète inverse** : on régénère des niveaux de gris grâce aux coefficients présents dans l'image compressée (*Fonction InverseDCT*).
- **Décalage de niveau inverse** : pour chaque pixel de l'image compressée, on va ajouter 128 à sa valeur.

1.3 Évolution selon le facteur de qualité - *Fonctions DistorsionTest et CalculateDistortion*

En sortie de la fonction *JPEGEncoder*, on obtient une copie plus ou moins pixelisée (selon le facteur de qualité entré) de l'image d'origine. Plus le facteur de qualité entré est haut, plus l'image sera de mauvaise qualité. Déjà à partir d'un facteur de qualité égal à 10, l'image est d'assez mauvaise qualité.

Pour le test quantitatif, nous avons réalisé un graphe de l'évolution du taux de distorsion en fonction du facteur de qualité utilisé pour la compression de l'image. Les résultats corroborent les tests visuels effectués : la distorsion augmente quand le facteur de qualité augmente. L'évolution de la distorsion se ralenti quand le facteur de qualité devient assez haut : de moins en moins d'informations sont perdues plus la qualité de l'image se détériore, ce qui parait intuitif.

Le taux de distorsion est calculé en calculant l'erreur quadratique entre l'image d'origine (après un décalage de niveau) et l'image compressée.

2 Partie 2 - Heuristiques et codage entropique dans la norme JPEG

Pour le codage entropique, nous avons juste eu le temps de réaliser des traitements préliminaires.

D'abord, nous avons fait une fonction pour réaliser le parcours en zig-zag des coefficients (*Fonction ZIGZAGParcours*). L'exécution de cette fonction est testée dans le programme et l'ordre de parcours est bien le bon.

Nous avons également mis en place deux structures afin de stocker les coefficients AC et DC de l'image, avant leur encodage. Ces coefficients sont extraits de chaque blocs 8x8 de l'image avec la fonction *IsolateACDC*. Les coefficients DC sont entrée sous forme de différence avec le coefficient DC du bloc précédent et les coefficients AC sont parcourus en zig-zag pour isoler les "0" au bout du vecteur. Afin de pouvoir encoder correctement ces valeurs par la suite, nous avons également réalisé une fonction de lecture du fichier "H.txt" afin d'en extraire les catégories, les longueurs de code et les codes binaires associés (*Fonctions getACDCCategories et filleCategories*).

Nous nous sommes ensuite essentiellement concentrées sur l'encodage des coefficients DC car la méthode nous paraissait plus intuitive. Nous avons donc fait des fonctions *findDCCategory, findDCIndex, BinaryConversion et EncodeDC* afin de traiter ces coefficients. Sur la base de la valeur de DIFF, nous trouvons la catégorie correspondante. Cette catégorie correspond a un mot binaire qui va constituer la première partie de la valeur binaire complète du coefficient. La seconde partie de cette valeur est la traduction en binaire de l'index de la valeur DIFF dans la catégorie trouvée juste avant. A la suite de l'exécution de ces fonctions, nous avons des coefficients DC en binaire.

Les étapes suivantes, que nous n'avons pas eu le temps de réaliser, étaient :

- La compression des coefficients AC avec RLE
- L'encodage des coefficients AC : à la place de la catégorie c'est un symbole RRRR/SSSS qui est conservé
- L'application de l'algorithme de Huffman pour compresser les données : les fonctions réalisées au TP1 sont présente dans le code source et auraient été utilisée pour cette étape (possibilité de générer un graphe).
- L'écriture du résultat binaire dans un fichier de sortie
- Le calcul du taux de compression en fonction du facteur de qualité donné en entrée.