UV SY26

TP 4

Thème : La compression JPEG

Introduction

Le but de ce TP est de mettre en oeuvre certaines étapes de l'algorithme de compression JPEG. Cet algorithme se résume dans les étapes suivantes :

1. Soustraire 128 à chaque pixel de l'image.
2. Diviser l'image en blocs de taille 8×8 et appliquer une DCT (transformée en cosinus discrète).
3. Quantifier les coefficients des blocs obtenus après la transformation.
4. Transformer les blocs de taille 8×8 en un vecteur 1D de taille 64 en parcourant les coefficients en "zigzag".
5. Coder séparément la liste des composantes continues de chaque bloc (DCT(0,0)), et le reste des coefficients qui seront codés par une méthode de "Run Length Coding".

Partie 1 : Transformée en cosinus discrète

Pour un bloc B de taille 8×8, la DCT est donnée par

$$D(u,v) = \frac{1}{4}C(u)C(v)\sum_{x=0}^{x=7}\sum_{y=0}^{y=7} B(x,y)\cos(\frac{(2x+1)\pi u}{16})\cos(\frac{(2y+1)\pi v}{16})$$

avec $C(0) = \frac{1}{\sqrt{2}}$, $C(k) = 1$ pour $k = 1 \dots 7$.

Il est possible de considérer la DCT 2D comme deux produits matriciels. Ainsi en considérant B comme une matrice 8×8 on a :

$$D = X_m B Y_m^T$$

$$\text{avec } X_m(u,x) = \frac{1}{2}C(u)\cos(\frac{(2x+1)\pi u}{16}) \text{ et } Y_m(v,y) = \frac{1}{2}C(v)\cos(\frac{(2y+1)\pi v}{16})$$

Compléter la fonction ***D=MyDCT(B)*** *(fichier MyDCT.m) pour effectuer la DCT d'un bloc B. Essayer d'appliquer la DCT au bloc Bref dans le fichier et vérifier que le résultat est identique au bloc BrefDCT (défini en commentaire dans le fichier TP4.m).*

Partie 2 : Quantification

On se propose de quantifier le bloc après la DCT. On remarque que les coefficients les plus importants sont concentrés au coin supérieur du bloc, ces coefficients codent les basses fréquences (DC). Les composantes hautes fréquences (AC) sont situées dans le bord inférieur et le bord droit du bloc de coefficients DCT et correspondent aux détails de l'image. Le quantification effectue l'opération suivante :

$$Dq(i,j) = \lfloor \frac{D(i,j)}{QM(i,j)} + 0.5 \rfloor$$

avec QM une matrice de quantification qui spécifie le pas de quantification différent pour chaque composante de la DCT. L'œil humain étant plus sensible aux erreurs de quantification dans les basses fréquences que dans les hautes fréquences, on pourra choisir des pas de quantification fins pour les basses fréquences et plus grossiers sur les hautes fréquences. La matrice de quantification par défaut ($Quality = 50$) est la suivante :

$$QM = \begin{bmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{bmatrix} * Fq$$

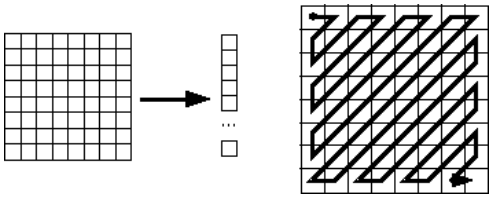
avec $Fq = f(Quality)$.

Le facteur $Quality$ fourni à la fonction en paramètre d'entrée est compris entre 1 (très mauvais) et 100 (très bien). Si $Quality = 50$ alors $Fq = 1$. Il faut donc déterminer la fonction, dans un premier temps, la fonction $f()$. Pensez aussi à vérifier que les coefficients de QM sont toujours compris entre 1 et 255.

Compléter la fonction ***QM=QuantM(Quality)*** *(fichier QuantM.m) qui calcule la valeur de la matrice de quantification.*

Partie 3 : Parcours en zigzag

Afin de pouvoir compresser efficacement l'information du bloc Dq on se propose de transformer la matrice 8×8 en un vecteur de longueur 64 en parcourant le bloc dans un ordre qui favorise la concentration des coefficients nuls à la fin du vecteur. Dans le vecteur zig, on stocke les indices des pixels parcourus suivant l'ordre indiqué dans la figure ci-dessous.



Vérifier que cette tâche est accomplie par la commande matlab suivante : ***Vzig=Dq(zig)*** *(définie dans le fichier TP4.m).*

Partie 4 : Codage

Cette partie permet de simuler le Run-Length-Encoding (**sans Huffman**). Ainsi pour chaque élément non nul du vecteur on calcule le triplet suivant (**nz, nb, Coef**) où :

- **nz** est le nombre de zéros qui le précèdent.
- **nb** est le nombre de bits nécessaires pour coder ce coefficient (qu'on va supposer constant durant ce TP et il sera fixé à 8).
- **Coef** la valeur du coefficient.

Pour le premier élément du bloc, il sera codé différemment : on ne précise pas le nombre de zéros qui le précèdent et on ne code pas la valeur même du coefficient mais une différence avec le premier élément du bloc voisin (codage différentiel). En effet, il faut noter que le coefficient DCT(0,0) est proportionnel à la moyenne des coefficients de tout le bloc B donc le coder en tant que tel demande un nombre de bits important. Mais, si on suppose que deux blocs voisins ont des moyennes proches, la différence entre les moyennes des blocs est faible et donc moins coûteuse en terme de codage. La partie restante du vecteur où il n'y a que des zéros sera codée par un double zéro. Ainsi le vecteur $[7, 0, 2, 0, 0, 5, 0, 0, 0, 0, 0, 0]$ avec le premier élément du bloc précédent $= 3$, sera codé par :

$$[8, 4, 1, 8, 2, 2, 8, 5, 0, 0]$$

Implémenter la fonction de codage d'un bloc ***A=code(Vzig,diff,nb)*** *qui prend en entrée le vecteur* ***Vzig*** *ainsi que la différence* ***diff*** *avec le premier élément du bloc précédent et le nombre de bits* ***nb*** *pour coder chaque coefficient (pour nous 8 bits). Indication : utiliser la fonction* ***find*** *de matlab qui cherche les indices des éléments non nul dans un vecteur.*

Partie 5 : Codage/décodage d'une image

Vous utiliserez les différentes images des TPs précédents. Pour le codage JPG vous définirez la fonction **codJPG.m** qui reprend de manière séquentielle toutes les étapes réalisées dans les parties précédentes, et pour le décodage la fonction **decJPG.m** (où il faudra implémenter la DCT inverse). Pour chaque image vous utiliserez comme facteur de qualité **Quality** : 1, 20, 50 et 100. Pour chaque facteur de qualité vous calculerez le SNR et le taux de compression. Vous commenterez les résultats qualitatifs et quantitatifs.