Rapport de TP - SY26

TP04 - La compression JPEG

Rémi Burtin

Cyril Fougeray

29 mai 2014

Université de Technologie de Compiègne

1 Introduction

Le but de ce TP est de mettre en œuvre certaines étapes de l'algorithme de compression JPEG.

2 Transformée en cosinus discrète (DCT)

2.1 Mise en œuvre de la fonction MyDCT

Cette première fonction nous permet de calculer la transformée en cosinus discrète d'un bloc de taille 8x8.

Pour cela, nous utilisons les formules suivantes :

$D = X_m B Y_m^T$

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

en prenant $C(0) = \frac{1}{\sqrt{2}}$ et $C(k) = 1$ pour $k \in [1..7]$.

Les coefficient u et v varie de 0 à 7, donc les matrices X_m et Y_m ont 8 lignes. Par ailleurs, la matrice B avec laquelle nous travaillons est une matrice 8x8. Ainsi, les coefficients x et y sont pris entre 1 et 8. On obtient donc $X_m = Y_m$.

Nous calculons les matrices X_m et Y_m en remplaçant les 4 variables (u, v, x, y) dans les formules, nous multiplions les matrices X_m, B et Y_m et obtenons ainsi la matrice D, la transformée en cosinus discrète de B.

Le code de la fonction est en annexe A.1.

2.1.1 Résultats

Après avoir exécuté la fonction, nous nous rendons compte du résultat :

```
>> Bref=[ 139 144 149 153 155 155 155 155;
144 151 153 156 159 156 156 156;
150 155 160 163 158 156 156 156;
159 161 162 160 160 159 159 159;
159 160 161 162 162 155 155 155;
161 161 161 161 160 157 157 157;
162 162 161 163 162 157 157 157;
162 162 161 161 163 158 158 158];
```

```
>> BrefDCT = MyDCT(Bref)

BrefDCT =

   1.0e+03 *

 1.2596   -0.0010   -0.0121   -0.0052    0.0021   -0.0017   -0.0027    0.0013
-0.0226   -0.0175   -0.0062   -0.0032   -0.0029   -0.0001    0.0004   -0.0012
-0.0109   -0.0093   -0.0016    0.0015    0.0002   -0.0009   -0.0006   -0.0001
-0.0071   -0.0019    0.0002    0.0015    0.0009   -0.0001   -0.0000    0.0003
-0.0006   -0.0008    0.0015    0.0016   -0.0001   -0.0007    0.0006    0.0013
 0.0018   -0.0002    0.0016   -0.0003   -0.0008    0.0015    0.0010   -0.0010
-0.0013   -0.0004   -0.0003   -0.0015   -0.0005    0.0017    0.0011   -0.0008
-0.0026    0.0016   -0.0038   -0.0018    0.0019    0.0012   -0.0006   -0.0004
```

La matrice $BrefDCT$ correspond au résultat souhaité.

3 Quantification

Cette partie nous permet de calculer une matrice de quantification, qui nous permettra ensuite de quantifier le bloc après la DCT. Le bloc quantifié s'obtient en divisant chacun des coefficients après la transformée en cosinus discrète par le facteur de quantification situé à la même position dans la matrice de quantification QM :

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

L'oeil humain étant sensible aux fréquences basses (situées dans le coin supérieur gauche de la matrice après DCT), nous devons diviser ces coefficients par un facteur faible. Voici la matrice utilisée pour un facteur $Quality = 50$:

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

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 $Fq = 1$. Afin de trouver la fonction $f()$ prenant comme paramètre *Quality*, nous avons recherché du côté des standards JPEG qui nous dit :

$$Fq(Quality) = \begin{cases} (100 - Quality)/50 & Quality \geq 50 \\ 50/Quality & \text{sinon} \end{cases}$$

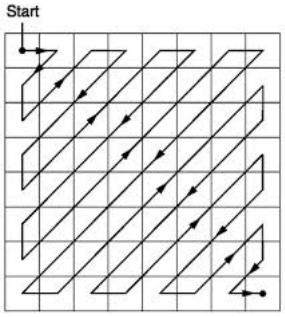
Nous obtenons ainsi Fq. La matrice de quantification est alors modifiée en conséquence. Il faut faire attention à ce que les valeurs de la matrice soit compris entre 1 et 255. Si certaines valeurs sont plus petites (*resp.* plus grandes) nous les fixons à 1 (*resp.* à 255). Pour cela, nous utilisons la fonction *find*, cf Annexe A.3.

Nous avons maintenant la matrice de quantification, nous pouvons alors calculer Dq, le bloc quantifié après la DCT :

```
Dq = round(BrefDCT./QM)
```

4 Parcours en zigzag

Afin de pouvoir compresser efficacement l'information du bloc Dq, on se propose de transformer la matrice 8x8 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.



Pour cela, nous avons à disposition un vecteur contenant les indexes des positions successives du parcours en zigzag dans le bloc 8x8. Matlab nous permet facilement de récupérer le vecteur à partir ce des indexes. En effet, prenons cet exemple simple :

```
>> a = [5 4 3 2 1]
a =
     5     4     3     2     1
```

```
>> a([1 3 5])
ans =
     5     3     1
```

Nous voyons que pour récupérer les valeurs d'un vecteur à des indexes précis, nous pouvons passer en paramètre du vecteur un vecteur d'indexes. Nous faisons de même pour le parcours en zigzag dans un bloc 8x8 :

```
%% Le parcours en ZigZag (Indexes)
zig=[1 9 2 3 10 17 25 18 ...
     11 4 5 12 19 26 33 41 ...
     34 27 20 13 6 7 14 21 ...
     28 35 42 49 57 50 43 36 ...
     29 22 15 8 16 23 40 37 ...
     44 51 58 59 52 45 38 41 ...
     24 32 38 46 53 60 61 54 ...
     47 40 48 55 62 63 56 64];

Vzig=Dq(zig);
```

5 Codage

Pour le codage du bloc, nous appliquons l'algorithme RLE (Run-Length-Encoding). Nous avons vu que le nombre de zéros dans un bloc après DCT est important (tant que l'image a des fréquences relativement basses) et que seul le premier élément du bloc est très supérieur à zéro, ainsi, nous allons coder les valeurs non nulles du bloc, par un triplet (**nz, nb, Coef**) :

- **nz** est le nombre de zéros précédant la valeur non nulle (le coefficient).
- **nb** est le nombre de bits nécessaires pour coder ce coefficient (il sera constant et fixé à 8 dans notre cas).
- **Coef** est la valeur du coefficient.

Le premier élément du bloc est coder sur un couple (nb, diff). En effet, 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. Ainsi, pour le premier élément du bloc, il sera encodé la différence du coefficient avec le coefficient du premier élément du bloc voisin. La partie restante du vecteur où il n'y a que des zéros sera codée par un double zéro.

Nous encodons chaque bloc via l'implémentation d'une fonction **code(Vzig, diff, nb)**, prenant 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.

Dans un premier temps, nous cherchons les coefficients non nuls dans le bloc via la fonction *find*. Celui-ci retourne les indexes de ces coefficients dans le bloc. Sa longueur nous donne le nombre de coefficient non nuls.

Nous pouvons dès à présent initialiser les deux premières valeurs de ce vecteur. En effet, la première valeur prend le nombre de bits pour encoder la seconde, donné ici via le paramètre **nb**. La seconde prend le paramètre **diff**.

Ensuite, nous complétons les triplets (**nz, nb, Coef**), en s'aidant du vecteur d'index des coefficients non nuls. Ainsi, nous parcourons ce vecteur pour obtenir le triplet, en commençant par le deuxième index (le premier étant déjà encodé). Le nombre de zéros, **nz**, est donné par la différence entre deux indexes successifs. **nb** est fixé par le paramètre **nb**. Le coefficient (**Coef**) est pris dans **Vzig** à l'index courant. Voici la boucle utilisée :

```
for i=2:length(v_non_nul),
    A(3*(i-1)) = v_non_nul(i) - v_non_nul(i-1) - 1;
    A(3*(i-1)+1) = nb;
    A(3*(i-1)+2) = Vzig(v_non_nul(i));
end;
```

Les deux dernières valeurs étant déjà à zéro, puisque le vecteur a été initialisé à zéro, le bloc passé en paramètre (**Vzig**) est alors encodé dans le vecteur **A**.

L'algorithme entier est en Annexe A.4.

6 Codage/décodage d'une image

6.1 Codage

Pour le codage d'une image entière, nous devons appliquer l'ensemble des algorithmes effectués jusqu'à présent pour des blocs de taille 8x8 sur toute une image. Nous faisons l'hypothèse que cette image à une taille multiple de 8, tant en longueur qu'en largeur, ce qui permet de faire ressortir un nombre entier de blocs 8x8. Si ça n'avait pas été le cas, nous aurions dû augmenter la taille de l'image pour qu'elle atteigne un multiple de 8 en y ajoutant des pixels égaux à 0 (padding).
Pour effectuer la division de l'image en bloc de 8x8, nous utilisons la fonction mat2cell :

```
blocks = mat2cell(img,8*ones(1,size(img,1)/8),8*ones(1,size(img,2)/8));
```

Cette fonction prend en paramètres l'image à découper ainsi que deux vecteurs. Ces vecteurs décrivent le découpage de la matrice.
Par exemple si on a une matrice 32x32 et qu'on veut la découper en 16 blocs de 8x8, on passera en paramètre à la fonction mat2cell les deux vecteurs suivants :
$\begin{bmatrix}8 & 8 & 8 & 8\end{bmatrix}$ $\begin{bmatrix}8 & 8 & 8 & 8\end{bmatrix}$
Pour une matrice 32x32 qu'on veut découper en 4 blocs 16x16 : $\begin{bmatrix}16 & 16\end{bmatrix}$ $\begin{bmatrix}16 & 16\end{bmatrix}$
Une fois la matrice découpée, on parcourt l'ensemble des blocs et on leur applique l'ensemble des algorithmes que nous avons codés : DCT, Quantification, Zig-Zag et RLE.
Nous concatenons l'ensemble des résulats dans un vecteur. Ce qui nous donne notre image encodée.

	SNR				**Compression**			
Image \Qualité	**1**	**20**	**50**	**100**	**1**	**20**	**50**	**100**
Lena	8.07 dB	16.10 dB	19.33 dB	31.42 dB	91.85%	73.09%	54.60%	-149.42%
Peppers	10.68 dB	18.88 dB	21.21 dB	33.97 dB	92.50%	80.26%	65.14%	-171.95%
Harbour	5.74 dB	11.21 dB	14.23 dB	24.17 dB	92.21%	71.09%	47.26%	-154.32%
Bridge	7.66 dB	13.63 dB	16.14 dB	27.46 dB	92.17%	59.36%	27.17%	-184.53%
Boats	9.69 dB	17.93 dB	21.42 dB	38.84 dB	92.57%	77.41%	61.45%	-153.06%
Airfield	8.55 dB	14.91 dB	17.41 dB	28.46 dB	91.1%	63.34%	33.15%	-187.75%

TABLE 1 – Tableau de comparaison des taux de compression et des SNR en fonction du facteur de qualité utilisé pour le codage

On peut observer des taux de compression négatifs pour un facteur de qualité de 100. Cela peut s'expliquer par le manque de coefficients nuls dans la matrice quantifié. Or chaque valeur est ensuite codé par un triplet, ce qui augmente grandement la taille.
Il faudrait faire en sorte que lorsque le taux de compression descend sous la barre de 0, on utilise l'image originale et non celle codée.

FIGURE 1 – Comparaison visuelle : (de gauche à droite) lena.bmp encodée avec les facteurs de qualité 1, 20, 50 et 100

6.2 Décodage

Pour le décodage, il faut effectuer l'exact inverse de ce que l'on a fait pour le codage.
On commence par parcourir le vecteur obtenu par le codage de l'image. On reconstitue chaque blocs grâce aux informations contenues dans les triplets. Nous savons que nous arrivons à la fin d'un bloc lorsque nous rencontrons deux zéros à la place d'un triplet.

On a alors un vecteur que l'on remet sous forme matricielle à l'aide d'un parcours en zig-zag inverse. On multiplie ensuite la matrice obtenue par la matrice de quantification (multiplication membre à membre). On peut enfin obtenir le bloc décodé en appliquant une DCT inverse, qui est définie comme telle :

Pour un bloc D de taille 8 x 8, la DCT inverse est donnée par

$$B = X^T D X$$
$$\text{avec } X(i,j) = \frac{1}{2} C(i) cos(\frac{(2j+1)\pi i}{16}) \text{ et } C(0) = \frac{1}{\sqrt{2}}, C(k) = 1 \text{ pour } k = 1...7$$

On met alors tous les blocs ainsi décodés dans un vecteur que nous remettons sous forme de matrice (une fois que tous les blocs ont été décodés) grâce à la fonction **reshape**.

7 Conclusion

A Codes source MATLAB

A.1 Transformée en cosinus discrète d'un bloc 8x8

```
function D = MyDCT(B)
X = zeros(8,8);
Y = zeros(8,8);
C = [1/sqrt(2) 1 1 1 1 1 1 1];

for i=0:7,
    for j=0:7,
        X(i+1,j+1) = C(i+1)/2*cos(((2*(j)+1)*pi*(i))/16);
    end;
end;

Y = X;
D = X * B * Y';

end
```

A.2 Transformée en cosinus discrète inverse d'un bloc 8x8

```
function B = MyIDCT(D)
X = zeros(8,8);
C = [1/sqrt(2) 1 1 1 1 1 1 1];

for i=0:7,
    for j=0:7,
        X(i+1,j+1) = C(i+1)/2*cos(((2*(j)+1)*pi*(i))/16);
    end;
end;

B = single(X' * (D * X));

end
```

A.3 Quantification

```
function QM = QuantM(Quality)
    %Cf standards JPEG
    if Quality >= 50
        Fq = (100 - Quality)/50;
    else
        Fq = 50/Quality;
    end;

    QM = round([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] * Fq);

      %On borne les valeurs a 255
      index = find(QM > 255);
      QM(index) = 255;

      index = find(QM == 0);
      QM(index) = 1;
end
```

A.4 Codage d'un bloc

```
function A= code(Vzig,diff,nb)
% Indexes des coefficients non nuls.
v_non_nul = find(Vzig ~= 0);

% Initialisation des deux premieres valeurs deu bloc encode.
A(1) = nb;
A(2) = diff;

% Ecriture des triplets
for i=2:length(v_non_nul),
    A(3*(i-1)) = v_non_nul(i) - v_non_nul(i-1) - 1;
    A(3*(i-1)+1) = nb;
    A(3*(i-1)+2) = Vzig(v_non_nul(i));
end;

% Fin de bloc
A = [A 0 0];

end
```

A.5 Codage d'une image

```
function imgEnco= codJPG(img, quality)

zig=[1 9 2 3 10 17 25 18 ...
     11 4 5 12 19 26 33 41 ...
     34 27 20 13 6 7 14 21 ...
     28 35 42 49 57 50 43 36 ...
     29 22 15 8 16 23 40 37 ...
     44 51 58 59 52 45 38 41 ...
     24 32 38 46 53 60 61 54 ...
     47 40 48 55 62 63 56 64];

if (strcmp(class(img),'char'))
    disp(['Lecture de l''image ' img]);
    img = imread(img);
end

if (size(img, 3) == 3) % conversion en gris si image couleur
    disp('Image couleur : conversion en gris');
    img = rgb2gray(img);
end

blocks = mat2cell(img,8*ones(1,size(img,1)/8),8*ones(1,size(img,2)/8));
oldFirst = 0;

imgEnco = [];

for i=1:size(blocks,1)*size(blocks,2),
    blockDCT = MyDCT(double(cell2mat(blocks(i))));
    QM=QuantM(quality);
    Dq= round(blockDCT./QM);
    Vzig=Dq(zig);

    A= code(Vzig,Vzig(1) - oldFirst,8);
    oldFirst = Vzig(1);
    imgEnco = [imgEnco A];
end;

%Calcul du taux de compression
size_img = size(img,1)*size(img,2)*8;
size_enco = length(imgEnco)*8;
compression = (1 - (size_enco / size_img))*100

%Decodage de l'image pour pouvoir calculer le SNR
imgDeco = decJPG(imgEnco, quality);

%Mise de l'image originale et quantifie sous forme de liste
list=reshape(double(img), 1, size_img/8);
list_quant=reshape(double(imgDeco), 1, size_img/8);

%Calcul distortion
distortion = mean((list-list_quant).^2)
%Calcul NMSE
nmse = distortion/var(list)
```

```
54 %Calcul du rapport signal sur bruit
55 snr = -10*log10(nmse)
56
57 end
```

A.6 Décodage d'une image

```
function imgDeco= decJPG(code, quality)

parcoursZig=[1 9 2 3 10 17 25 18 ...
             11 4 5 12 19 26 33 41 ...
             34 27 20 13 6 7 14 21 ...
             28 35 42 49 57 50 43 36 ...
             29 22 15 8 16 23 40 37 ...
             44 51 58 59 52 45 38 41 ...
             24 32 38 46 53 60 61 54 ...
             47 40 48 55 62 63 56 64];

matQuant = [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];

iCode = 1;
iZig = 1;
oldFirst = 0;
zig = zeros(1,64);
vBlock = [];

while iCode ~= length(code) + 1,

    %Prise en compte du premier element du bloc
    zig(iZig) = code(iCode+1) + oldFirst;
    oldFirst = zig(iZig);

    iZig = iZig + 1;
    iCode = iCode + 2;
    while true
        if code(iCode) == 0 && code(iCode + 1) == 0 %Si on est a la fin d'un bloc
            iCode = iCode + 2;
            iZig = 1;
            break;
        else % C'est un triplet normal
            iZig = iZig + code(iCode); %On met le bon nombre de 0

            zig(iZig) = code(iCode + 2); %on ajoute le coefficient
            iZig = iZig + 1;
            iCode = iCode + 3; %on passe au triplet suivant
        end
    end

    block = zeros(8,8);

    %zig zag inverse
    block(parcoursZig) = zig;

```

```
54      %quantification inverse
55      block = block.*QuantM(quality);
56
57      %calcul DCT inverse
58      block = round(MyIDCT(block));
59
60      %transformation du bloc en cell
61      block = mat2cell(block, 8,8);
62
63      %on met les blocs dans un vecteur
64      vBlock = [vBlock block];
65      zig = zeros(1,64);
66  end;
67
68  %on remets les blocs sous la forme d'une matrice
69  imgDeco = cell2mat(reshape(vBlock, sqrt(length(vBlock)), sqrt(length(vBlock))));
70
71  end
```