Rapport de TP - SY26

TP02 - Codage de Huffman

Rémi Burtin

Cyril Fougeray

2 avril 2014

Université de Technologie de Compiègne

1 Introduction

L'objectif de ce TP est de comparer le codage de Huffman au codage arithmétique. Pour cela, nous travaillons sur une image ('lena.bmp') en niveaux de gris, ce qui permet de fixer une seule valeur (entre 0 et 255) pour chaque pixel de l'image. Depuis les probabilités des pixels, que nous avons calculé via un histogramme (cf. TP précédent), nous avons implémenté les deux algorithmes, puis calculé le taux de compression de chacun.

2 Codeur de Huffman

L'image que nous manipulons pour réaliser l'encodage/décodage selon Huffman ne contient qu'une seule composante correspondant à l'intensité du gris. Pour convertir une image couleur en niveaux de gris, nous utilisons la fonction *rgb2gray* qui retourne une matrice à deux dimensions, de même taille que l'image.

Afin de réaliser l'encodage, il est nécessaire de connaître la distribution de probabilités des niveaux de gris de l'image. Pour cela, nous commençons par réaliser l'histogramme comptant le nombre d'occurrences de chaque valeur de gris. L'histogramme est en fait un vecteur de taille 256, ce qui correspond aux différentes valeurs de gris. Ainsi, à l'index $x \in [0, 255]$, nous obtenons le nombre d'occurences de cette intensité de gris dans l'image. Afin d'obtenir une distribution de probabilités ($\sum P(X = x_i) = 1$), nous normalisons l'histogramme en divisant chaque valeur de l'histogramme par le nombre total de pixels dans l'image.

3 Codeur arithmétique

4 Conclusion

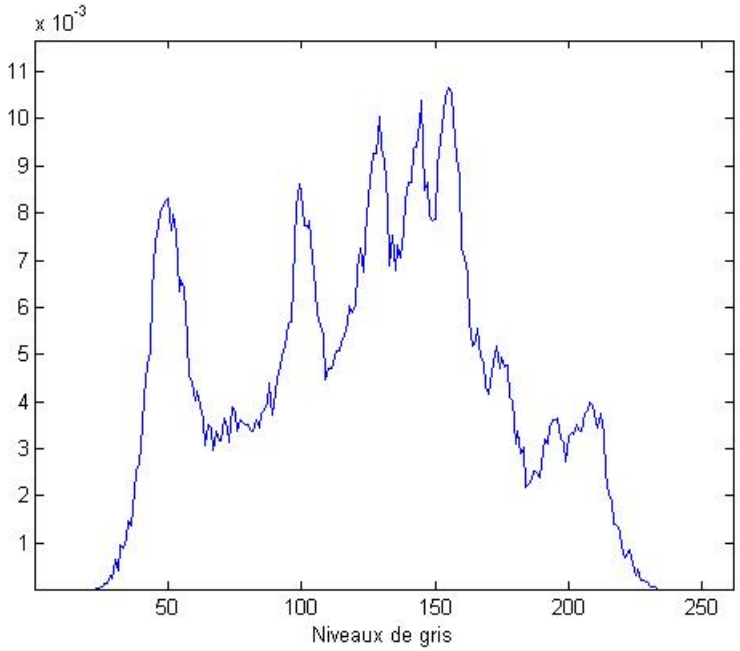

FIGURE 1 – Distribution de probabilités, niveaux de gris

A Codes source MATLAB

A.1 Algorithme - codage de Huffman

```
function compression = huffman(path)
global prob avglen dict histogramme;

% matrice de l'image
disp(['Lecture de l''image ' path]);
img = imread(path);

% initialisation vecteurs
histogramme = zeros(1,256); % occurences par niveaux de gris
prob = zeros(1,256); % probas (histo normalise)

composantes = size(img, 3); % Taille de la 3eme dimension

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

dim = size(img,1)*size(img,2); % nombre de pixels dans l'image
list=reshape(double(img), 1, dim); % images en ligne (vecteur)

% creation histogramme
for i=1:dim,
    histogramme(round(list(i))) = histogramme(round(list(i))) + 1;
end

% creation probas des niveaux de gris
disp('Creation vecteur des probalites des niveaux de gris');
for i=1:256,
    prob(i) = histogramme(i)/dim;
end

% creation vecteur de taille 256 (1, 2, 3, ... 256)
% correspond aux niveaux de gris
disp('Creation du vecteurs des differents symboles');
symbols = [1:256];

% creation du dictionnaire
disp('Creation du dictionnaire (via huffmandict())');
[dict,avglen] = huffmandict(symbols,prob);
disp(['Longueur moyenne des mots encodes : ' num2str(avglen)])

% encodage de l'image (passer un vecteur ligne)
disp('Encodage de l''image');
tic % calcul du temps d'encodage/decodage
enco = huffmanenco (list, dict);

% decodage (retourne un vecteur)
disp('Decodage de l''image');
deco = huffmandeco(enco, dict);
time = toc;
```

```
52 disp(['Duree encodage/decodage : ' num2str(time) 's.']);
53
54 % reshape vecteur -¿ image 2D
55 imagedeco = reshape(deco, size(img,1), size(img,2));
56
57 imshow(imagedeco/256);
58
59 compression = 1-length(enco)/(dim*8);
60 disp(['Taux de compression : 1 - taille finale / taille initiale = ' num2str(compression)]);
61
62 compression2 = 1-avglen/8;
63 disp(['Taux de compression : 1 - longueur moyenne mot code / 8 = ' num2str(compression2)]);
64
65 return;
66 end
```