Rapport de TP - SY26

TP02 - Codage de Huffman

Rémi Burtin

Cyril Fougeray

9 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. Nous avons donc réalisé deux scripts, chacun codant et décodant une image en utilisant un des deux algorithmes.

2 Codeur de Huffman

2.1 Mise en oeuvre

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 $x \in [1, 256]$, nous obtenons le nombre d'occurrences 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.

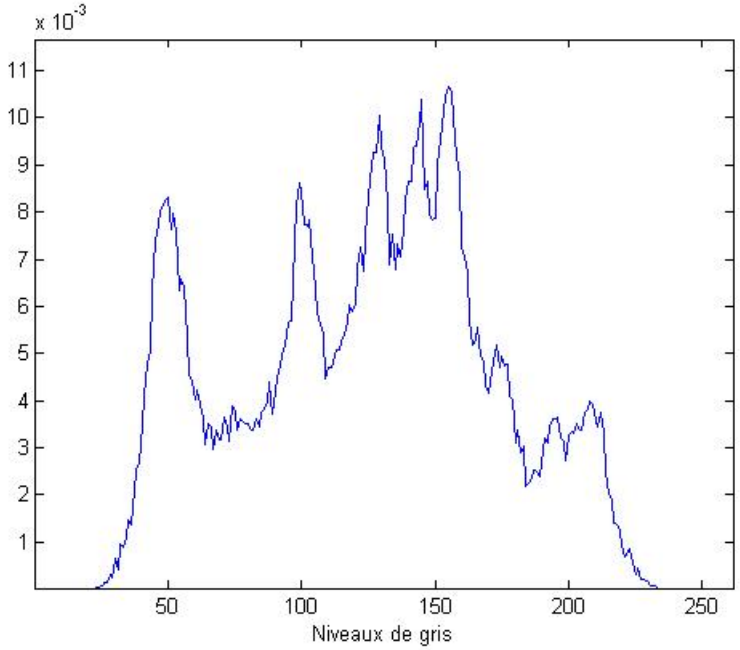

FIGURE 1 – Distribution de probabilités, niveaux de gris

L'algorithme d'Huffman fonctionne de sorte que les symboles ayant la moins grande probabilité prennent des valeurs les plus grande dans le dictionnaire, afin d'améliorer la compression. Dans le cas de notre image, beaucoup d'intensités de gris n'apparaissent pas dans l'image : elles correspondent aux indexes de l'histogramme ayant pour valeur 0. Afin de supprimer ces valeurs inutiles lors de la création du dictionnaire, nous créons un nouvel histogramme, ayant des valeurs toutes strictement positives. Ce nouvel histogramme pourra alors avoir à l'index 1, le niveaux de gris 30. (si les 30 premiers niveaux de gris n'apparaissent pas dans l'image). Afin de reconstruire l'image originale après encodage/décodage, nous devons alors établir une table de correspondance. Ainsi, chaque index de l'histogramme devra être lié à une valeur d'un niveaux de gris présent dans l'image.

Selon la longueur de l'histogramme sans zéro, nous créons un dictionnaire avec les nouveaux symboles allant de 0 à L, L étant la taille du nouvel histogramme. Dans le tableau 1 page 3, nous associons le nombre d'occurrences de chaque niveaux de gris. Afin de construire le dictionnaire via le codeur de Huffman, nous devons créer les probabilités qui en découle (en normalisant l'histogramme). Chaque effectif est alors divisé par le nombre total des effectifs, soit ici le nombre de pixels.

L'index de l'histogramme sans zéro correspond à un symbole lors de la création

Histogramme simple		Histogramme sans zéros		
Index	Effectif	Index	Effectif	Niveau de gris
1	0	1	2	3
2	0	2	6	5
3	2	3	7	6
4	0			
5	6			
6	7			
7	0			
8	0			

TABLE 1 – Exemple - Histogrammes

Index (Histogramme sans zéro)	Niveau de gris associé
1	3
2	5
3	6

TABLE 2 – Exemple - Table de correspondance

du dictionnaire par la fonction *huffmandict*.

2.2 Résultats

Nous affichons un log grâce à la fonction *disp* :

```
>> huffman('../lena.bmp')
Lecture de l'image ../lena.bmp
Image couleur : conversion en gris
Creation vecteur des probalites des niveaux de gris
Creation du vecteurs des differents symboles, nombre de symboles : 217
Creation du dictionnaire (via huffmandict())
Longueur moyenne des mots encodes : 7.4675
Encodage de l'image
Decodage de l'image
Duree encodage/decodage : 131.098s.
Taux de compression : 1 - taille finale / taille initiale = 0.066566
Taux de compression : 1 - longueur moyenne mot code / 8 = 0.066566
```

3 Codeur arithmétique

4 Conclusion

A Codes source MATLAB

A.1 Algorithme - codage de Huffman

```
function compression = huffman(path)
global prob avglen dict histogramme histo_0 corres index;

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

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

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

% Suppression des zeros
[val, index] = find (histogramme ~= 0) ;
histo_0 = histogramme(index);

prob = zeros(1,length(histo_0)); % initialisation "prob" (histo normalise)

disp('Creation vecteur des probalites des niveaux de gris');
for i=1:length(histo_0),
    prob(i) = histo_0(i)/dim;
end

disp('Creation du vecteurs des differents symboles')
disp(['(nombre de symboles : ' num2str(length(histo_0)) ')']);
symbols = [1:length(histo_0)];

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

% Creation des tables de correspondance
% pour pouvoir retrouver plus tard les bonnes
% valeurs des differents pixels
corres = zeros(1, 256);

for i=1:length(index),
    corres(index(i)) = i;
```

```
52 end
53
54 list_0 = zeros(1,dim);
55
56 for i=1:dim,
57     list_0(i) = corres(list(i));
58 end
59
60 % encodage de l'image (passer un vecteur ligne)
61 disp('Encodage de l''image');
62 tic  % calcul du temps d'encodage/decodage
63 enco = huffmanenco (list_0, dict);
64
65 % decodage (retourne un vecteur)
66 disp('Decodage de l''image');
67 deco = huffmandeco(enco, dict);
68 time = toc;
69 disp(['Duree encodage/decodage : ' num2str(time) 's.']);
70
71 % on remet les bonnes valeurs des pixels
72 deco_2 = zeros(1,dim);
73 for i=1:dim,
74     deco_2(i) = index(deco(i));
75 end
76
77 % reshape vecteur -¿ image 2D
78 imagedeco = reshape(deco_2, size(img,1), size(img,2));
79
80 imshow(imagedeco/256);
81
82 compression = 1-length(enco)/(dim*8);
83 disp(['Taux de compression : 1 - taille finale / taille initiale = ' num2str(compression)]);
84
85 compression2 = 1-avglen/8;
86 disp(['Taux de compression : 1 - longueur moyenne mot code / 8 = ' num2str(compression2)]);
87
88 return;
89 end
```