Université de Liège
Faculté de Sciences Appliquées
Section Informatique

Année académique 2002-2003

# Codage et segmentation d'images sur base d'une prédiction par extrapolation

Renaud Dardenne

Mémoire présenté en vue
de l'obtention du grade de
licencié en Informatique

# Table des matières


**I Modèle de prédiction et de segmentation** **6**

**1 Introduction** **7**
1.1 Généralités . . . . . 7
1.2 Objectifs . . . . . 7
1.3 Aperçu . . . . . 8

**2 La norme JPEG** **9**
2.1 Introduction . . . . . 9
2.2 Aperçu des différentes étapes . . . . . 10
2.3 Préparation de l'image . . . . . 11
2.3.1 Changement du système de couleurs . . . . . 11
2.3.2 Sous-échantillonnage . . . . . 11
2.4 Découpage et Entrelacement . . . . . 12
2.4.1 Découpage . . . . . 12
2.4.2 Images à plusieurs composantes . . . . . 13
2.4.3 Unité de données . . . . . 13
2.5 DCT . . . . . 14
2.5.1 Définition . . . . . 14
2.5.2 Pourquoi effectuer une DCT ? . . . . . 15
2.5.3 Comment effectuer la DCT ? . . . . . 15
2.6 Quantification . . . . . 17
2.7 Codage . . . . . 18
2.7.1 Coefficients DC . . . . . 18
2.7.2 Coefficients AC . . . . . 18
2.7.3 Codage de Huffmann . . . . . 19
2.8 JFIF . . . . . 20

**3 Statistique des coefficients de la DCT** **21**
3.1 Objectif . . . . . 21
3.2 Rappels de théorie de l'information . . . . . 21
3.3 Coefficients DC et DPCM . . . . . 22
3.4 Coefficients AC . . . . . 23
3.4.1 Coefficients AC de la DFT . . . . . 27

**4 Prédiction intra-bloc de coefficients AC** **30**
4.1 Prédiction des coefficients AC . . . . . 31
4.2 Prédiction des coefficients AC quantifiés . . . . . 31

**5 Prédiction inter-bloc** **33**
5.1 Algorithme de prédiction . . . . . 33
5.1.1 Modèle . . . . . 33
5.1.2 L'algorithme de déconvolution sélective . . . . . 34
5.1.3 Résultats attendus lors de l'application au codage . . . . . 35
5.2 Exemples . . . . . 36

**6 Prédiction inter-bloc et segmentation** **39**
6.1 Détermination d'un schéma de segmentation . . . . . 39
6.2 Critère de fusion des régions . . . . . 40
6.3 Résumé du fonctionnement de l'algorithme : ordre de fusion des régions 40

**II Codage** **42**

**7 Codage de la carte de segmentation** **43**
7.1 Données de la carte de segmentation . . . . . 43
7.2 Codage Arithmétique de la carte . . . . . 44
7.2.1 Rappels sur le codage arithmétique . . . . . 44
7.2.2 Résultats obtenus . . . . . 45

**8 Codage de l'image** **46**
8.1 Fonctionnement général du codeur . . . . . 46
8.2 Codage d'image par JPEG . . . . . 46
8.3 Présentation et interprétation de résultats de codage par prédiction . 49

**III Segmentation** **54**

**9 Segmentation** **55**
9.1 Introduction . . . . . 55
9.2 Résultats . . . . . 56

**IV Conclusions** **63**

**V Annexes** **65**

**A Images de tests** **66**

**B Outils de mesures** **69**

**C Codes sources** **71**
C.1 intraPrediction.c . . . . . 71
C.2 intraSegmentation.c . . . . . 78

**Bibliographie** **85**

**Remerciements** **87**

# Première partie

# Modèle de prédiction et de segmentation

# Chapitre 1

# Introduction

## 1.1 Généralités

L'informatique actuelle emploie sans cesse des images : les interfaces graphiques rendent l'utilisation d'un programme ou d'un système d'exploitation plus agréable, l'usage d'éléments graphiques dans les documents rendent ceux-ci plus attrayants, etc. Beaucoup de types d'informations complexes sont ainsi convertis en images afin de faciliter leur échange ou leur manipulation. Leur emploi dans Internet n'est plus à démontrer.
Malheureusement, le stockage de ces images est très gourmand en espace disque. Prenons un exemple : une image de $800 \times 600$ pixels, chaque pixel étant codé sur 24 bits (ce qui représente une palette de $2^{24} \simeq 16$ millions de couleurs, suffisante pour représenter une image réelle) occupe $12 \times 10^6$ bits ! La solution existe : la compression, qui permet de réduire le nombre de bits nécessaires à son stockage.
En règle générale, la compression des données est rendue possible par la présence de redondance dans l'information. Nous allons voir que dans certains cas, nous pouvons également tolérer un certain degré de perte d'information lors de la compression.
Tout l'art de la compression consiste à satisfaire les contraintes imposées (techniques, financières,...) tout en obtenant la qualité d'image requise par l'application désirée. En effet, la compression d'images peut s'employer avec des contraintes très différentes, selon l'utilisation à laquelle les images sont destinées. On peut prévoir d'utiliser l'image pour des applications de loisirs, pour du traitement d'images, pour du diagnostic médical, etc.

## 1.2 Objectifs

L'objectif du présent travail de fin d'études est l'exploitation d'une technique d'extrapolation, appelée *déconvolution sélective*, pour la prédiction de parties d'une image. Cette prédiction est exploitée dans le cadre de deux applications particulières : le codage et la segmentation.

Nous appliquerons cette méthode à JPEG, en tentant de prédire les caractéristiques de l'image, plus précisément, les valeurs des coefficients DCT des différents blocs constituant l'image. La prédiction est d'un réel intérêt : toute technique capable de réduire le débit et donc d'augmenter le gain de compression d'une image est la bienvenue.

La segmentation est un outil essentiel pour l'analyse d'images : beaucoup de techniques de compression fondées sur la notion d'objet sont basées sur le résultat d'une segmentation. Nous définirons le critère de fusion des régions extrapolées qui nous permettra d'effectuer la segmentation de l'image. L'originalité de ce travail réside donc dans la manière dont nous allons extrapoler les blocs les uns sur les autres : à notre connaissance, l'utilisation d'un critère spectral n'a jamais été réalisée.

## 1.3 Aperçu

La première partie introduira les concepts que nous utiliserons par la suite. Nous commencerons par présenter la norme JPEG en toute généralité afin de définir bon nombre de concepts que nous emploierons par la suite. Nous continuerons en étudiant certaines statistiques et propriétés de la transformée de Fourier discrète ainsi que de la transformée en cosinus discrète puisqu'elles interviendront sans cesse par la suite ; nous insisterons notamment sur leurs différents comportements. Viendra ensuite la présentation d'une méthode de prédiction présentée dans la norme JPEG, mais n'ayant jamais été implémentée à cause de ses résultats mitigés. Enfin deux chapitres présenteront notre méthode de prédiction et son emploi dans le cadre de la segmentation.

La seconde partie est entièrement consacrée à l'utilisation de la déconvolution sélective dans un schéma de codage. Nous incorporerons l'algorithme d'extrapolation dans un codeur JPEG et nous en étudierons le gain éventuel.

La troisième partie présentera quelques résultats relatifs à la segmentation par croissance de régions.

Le mémoire se terminera par une conclusion générale, suivie des annexes contenant le jeu d'images à partir desquelles nous avons effectué tous nos tests, les outils de mesures utilisés, ainsi que les plus importants codes sources développés.

# Chapitre 2

# La norme JPEG

## 2.1 Introduction

Une méthode de compression d'images est adaptée à un type d'images. JPEG est une méthode sophistiquée de compression d'images statiques en couleurs réelles ou en tons de gris, représentant des scènes réelles (pas de dessins ou de texte). JPEG, acronyme de Joint Photographic Experts Group, a été défini par un groupe mixte, au sens où ses experts venaient de différents grands organismes de normalisation (dont l'ISO et le CCITT). Leurs efforts ont commencé en juin 1987 et le premier rapport fut publié en 1991, date à laquelle il devint un standard international.

Quelles sont les raisons de son succès ?

- JPEG peut s'appliquer à tous les types d'images auxquelles il est destiné (continuous-tone) sans aucune restriction sur la taille de l'image, la complexité de son contenu ou l'espace de couleurs utilisé pour sa représentation. JPEG peut traiter des images jusqu'à 255 composantes codées sur 12 bits.
- Il possède de nombreuses options, notamment la possibilité de fixer la valeur du rapport qualité - compression désiré.
- Il permet l'obtention de taux de compression importants tout en gardant une qualité d'image excellente. Il n'est pas rare que l'œil humain ne distingue pas de différence entre l'image originale et l'image compressée, et ce, même à des rapports de compression de 10 :1.
- Il est implémenté sur une grande variété de plateformes.
- La décompression est de même complexité que la compression.
- Bien que le standard soit sophistiqué, les algorithmes utilisés ne sont pas d'une trop grande complexité algorithmique.

Il possède également plusieurs modes d'opération :

**Baseline** (parfois aussi appelé séquentiel) est le mode le plus répandu. C'est le mode qui nous intéressera par la suite : chaque composante de l'image est encodée lors d'un parcours gauche-droite, haut-bas.

**Progressive** est un mode qui encode l'image en plusieurs passages qui augmentent

progressivement le niveau de détail. L'intérêt est de pouvoir afficher très rapidement un aperçu de l'image de très mauvaise qualité, puis d'afficher progressivement l'image à de plus hautes résolutions, ce qui autorise par exemple à rejeter l'image après le premier aperçu.

**Lossless** est rarement utilisé et implémenté (une seule implémentation connue à ce jour). Ce mode est sans perte, mais il ne permet des taux de compression que de l'ordre de 2 :1. Ce mode calcule une valeur "prédite" pour chaque pixel, calcule la différence entre le pixel et la valeur prédite et encode la différence à l'aide d'un codage de Huffmann ou arithmétique.

**Hierarchical** est un mode où l'image est compressée à de multiples résolutions ce qui permet de visualiser l'image à basse résolution sans devoir d'abord décompresser toute l'image en haute résolution.

Par la suite, nous ne nous intéresserons plus qu'au mode *baseline*, qui est le plus répandu car suffisant pour la plupart des applications, bien que le mode *progressive* soit de plus en plus utilisé sur le Web.

## 2.2 Aperçu des différentes étapes

L'encodeur procède par étapes :

1. La transformation de l'image de départ de l'espace des couleurs additif RGB vers un espace de couleurs chrominance/luminance, et le sous-échantillonnage de certaines composantes. Notons que ces étapes ne sont pas nécessaires, mais, comme nous le verrons par après, elles permettent d'obtenir un taux de compression plus élevé.
2. Le découpage de chaque composante de l'image en blocs de 8×8 pixels.
3. L'application d'une transformée en cosinus discrète à chaque bloc.
4. La quantification des coefficients. C'est lors de cette étape que la perte d'information est la plus importante. C'est également ici qu'est pris en compte le paramètre de qualité sélectionné.
5. La lecture des coefficients, leur codage entropique et leur entrelacement.
6. L'ajout au résultat d'une série de headers ainsi que les informations utilisées lors de la compression afin que le décodeur puisse effectuer le processus inverse. Et enfin, placement du tout dans un fichier JPEG au format d'échange.

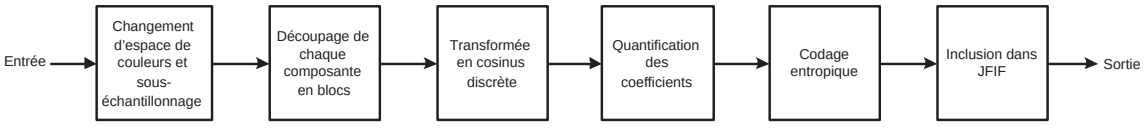



FIG. 2.1: Déroulement général de JPEG en mode Baseline.

Nous allons maintenant décrire, dans le cas le plus général, le détail de ces différentes étapes.

## 2.3 Préparation de l'image

### 2.3.1 Changement du système de couleurs

Nous commençons donc par convertir l'image, représentée initialement dans l'espace de couleurs additif RGB (Red, Green, Blue), dans un espace de couleurs luminance/chrominance (YUV, YCbCr). La composante luminance est représentée par les différents tons de gris allant du noir au blanc et permet d'éclaircir ou d'assombrir une couleur, les deux autres composantes de chrominance représentent l'information de couleur.
Pourquoi vouloir changer d'espace ? Parce que l'oeil humain n'est pas aussi sensible aux hautes fréquences chromatiques qu'aux hautes fréquences de luminance. Il en résulte que l'on peut tolérer une perte plus importante d'information sur les deux composantes chromatiques, sans réel impact visuel sur le résultat.
Ce changement de système n'est pas indispensable : le reste de l'algorithme s'effectue sur chaque composante sans se soucier de la signification de ces composantes. S'il n'est pas effectué, vu qu'aucune des différentes composantes de couleurs ne pourra être sous-échantillonnée, il en résultera une moins bonne compression.

Examinons le cas d'une transformation RGB vers YCbCr :

- La luminance Y est définie comme la puissance rayonnée, pondérée par une fonction caractéristique de la vision appelée sensitivité spectrale. La luminance est donc proportionnelle à la puissance de la source émettrice, semblable à l'intensité lumineuse, à ceci près qu'elle est relative à la sensation d'éclat de la vision humaine.
- L'œil étant très sensible à la variation de luminance, une manière très facile de représenter les composantes de chrominance Cb et Cr est de soustraire la luminance des deux composantes bleu et rouge. Cb et Cr représentent, pour une luminance donnée, la couleur en fonction de la présence ou de l'absence de bleu ou de rouge, respectivement.
  La conversion entre RGB et YCbCr est linéaire et donc très simple :

$$\begin{cases} Y = & (77/256)R + (150/256)G + (29/256)B \\ U = & -(44/256)R - (87/256)G + (131/256)B + 128 \\ V = & (131/256)R - (110/256)G - (21/256)B + 128 \end{cases} \tag{2.1}$$

### 2.3.2 Sous-échantillonnage

Certaines composantes de l'image sont sous-échantillonnées en moyennant entre eux des groupes de pixels. La composante de luminance est laissée en pleine résolution ; cette étape ne s'applique donc pas aux images en tons de gris : c'est la raison pour laquelle les images en couleurs réelles sont plus compressées que les images en tons de gris. Les deux composantes chromatiques, quant à elles, sont sous-échantillonnées à un rapport 2 :1 horizontalement et verticalement (échantillonnage 2h2v) ou à un rapport 2 :1 horizontalement et 1 :1 verticalement (échantillonnage

2h1v). Ceci diminue automatiquement le volume de données à compresser : de moitié dans le cas d'un échantillonnage 2h2v $(1/3 + 2/3 \times 1/4 = 1/2)$ , d'un tiers dans le cas 2h1v $(1/3 + 2/3 \times 1/2 = 2/3)$.
En terme numérique, cette étape implique beaucoup de pertes, mais la composante luminance étant laissée telle quelle, il n'y a pas d'impact réel sur la qualité de l'image.

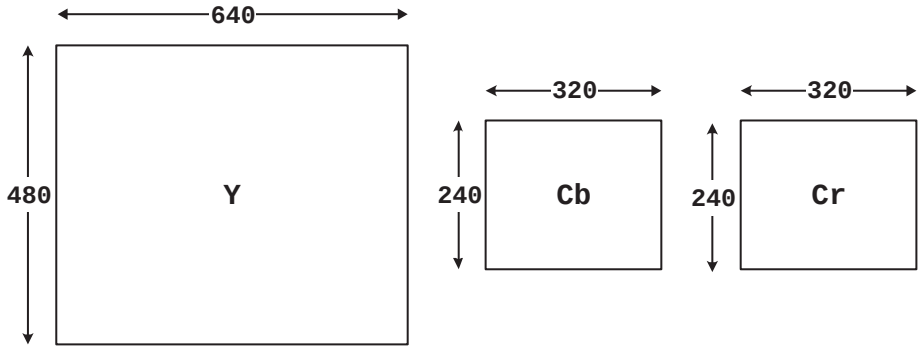



FIG. 2.2: Composantes d'une image de taille 640x480 après sous-échantillonnage 2h2v.

## 2.4 Découpage et Entrelacement

### 2.4.1 Découpage

Pour chaque composante, nous groupons ensuite les pixels en blocs de $8 \times 8$. Si le nombre de colonnes ou de lignes de l'image n'est pas un multiple de 8, la dernière ligne du bas et la dernière colonne de droite sont recopiées autant de fois que nécessaire.

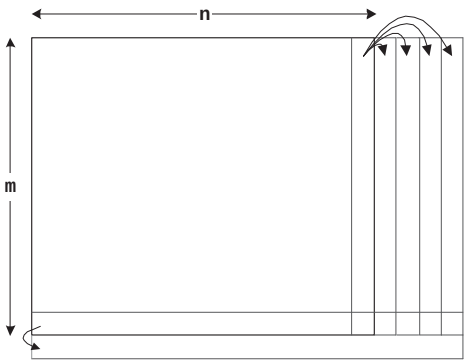



FIG. 2.3: Extension d'une composante.

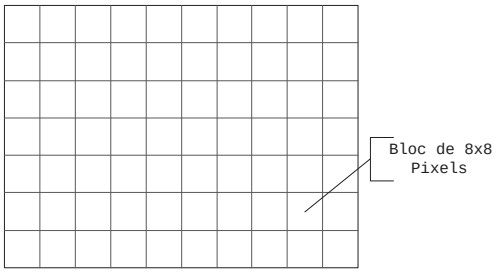



FIG. 2.4: Composante découpée en blocs 8x8.

En supposant que l'image a une taille de $m \times n$, nous recopions la colonne $m$ un nombre de fois suffisant pour arriver au premier multiple de 8 et nous faisons de même pour les colonnes. Le résultat peut ainsi être divisé en blocs de $8 \times 8$.

### 2.4.2 Images à plusieurs composantes

Dans le but de pouvoir traiter de manière parallèle le processus de décompression et l'affichage de ce qui a déjà été décompressé, les différentes composantes de d'image sont entrelacées. Mais avant de voir comment, nous allons examiner les relations existantes entre les différentes composantes.
Bien que JPEG soit très souple dans sa syntaxe (il laisse une très grande liberté de choix des différents paramètres), nous nous limiterons par la suite au cas le plus courant, à savoir une image à 3 composantes et dont chaque composante est codée sur 8 bits (même si JPEG supporte des images ayant jusqu'à 255 composantes).

Une image est donc composée de trois composantes, chacune étant visualisée comme un tableau à deux dimensions ayant ses éléments codés sur 8 bits. Les différentes composantes peuvent être de tailles différentes, mais il doit exister une relation entre elles :

- supposons que la i$^{\text{ème}}$ composante ait les dimensions $x_i$ et $y_i$.
- La taille de l'image X et Y est définie comme la plus grande valeur des $x_i$ et $y_i$.
- Chaque composante encode donc les données X, Y, $H_i$ et $V_i$.

Que sont $H_i$ et $V_i$ ? Ces nombres représentent la relation entre la composante et la taille de l'image. Ils permettent à partir de la relation suivante de retrouver la taille de chaque composante :

$$\begin{cases} x_i = & X \times (\frac{H_i}{H_{max}}) \\ y_i = & Y \times (\frac{V_i}{V_{max}}) \end{cases} \quad i = 1, 2, 3 \tag{2.2}$$

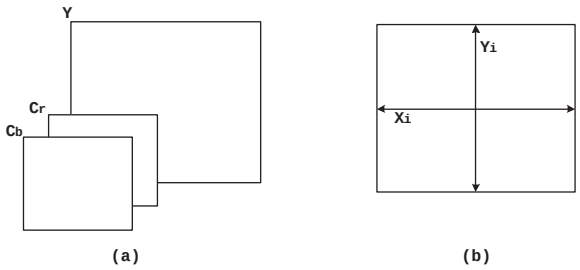



FIG. 2.5: Composantes d'une image (a) Taille d'une composante (b).

### 2.4.3 Unité de données

Une *unité de donnée* (Data Unit, dans le mode avec pertes qui nous occupe) est un bloc de $8 \times 8$ pixels. Chaque composante est divisée en *régions* rectangulaires de $H_i$ sur $V_i$ unités de données (parcourues de gauche à droite et de bas en haut). L'intérêt est d'obtenir un même nombre de blocs d'une composante à l'autre.

Une *unité de codage minimum* (MCU) est définie comme le plus petit groupe d'unités de données entrelaçées.
En mode entrelaçé, l'encodeur traite d'abord l'unité de données supérieure gauche des trois composantes ($MCU_1$), puis les trois suivantes($MCU_2$), ...

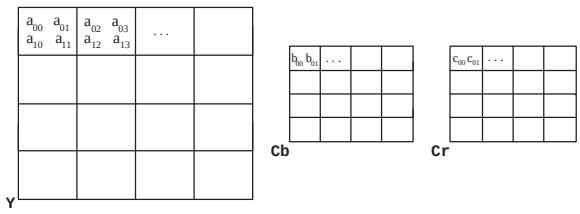



FIG. 2.6: Data Units assemblés en régions dans les différentes composantes.

$$MCU_1 = a_{00}, a_{01}, a_{10}, a_{11}, b_{00}, b_{01}, c_{00}, c_{01}$$
$$MCU_2 = a_{02}, a_{03}, a_{12}, a_{13}, b_{02}, b_{03}, c_{02}, c_{03}$$

## 2.5 DCT

### 2.5.1 Définition

La transformée en cosinus discrète de chaque bloc de chaque composante de l'image est le coeur de la compression JPEG. Un bloc de $8 \times 8$ pixels est un signal discret, fonction des deux dimensions spatiales $x$ et $y$. La FDCT (Forward Discrete Cosine Transform) à 2 dimensions prend en argument cette fonction et la décompose en fréquences sur 64 fonctions orthogonales de base. Le résultat est donc un ensemble de 64 coefficients DCT, représentant l'amplitude de ces différentes fonctions de base, ou encore la quantité relative des 64 fréquences spatiales contenues dans le bloc de départ (figure 2.9).

Il existe différentes définitions de la DCT, dépendantes des conditions imposées aux bords. Nous nous contenterons donc de la version classique en traitement d'images, à savoir celle où l'image a été miroirisée le long de ses bords.

Soit $f(m,n)$ le signal de départ, la FDCT $F(u,v)$ se définit comme suit :

$$F(u,v) = \frac{2}{\sqrt{MN}} c(u)c(v) \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} f(m,n) \cos\left(\frac{2m+1}{2M} u\pi\right) \cos\left(\frac{2n+1}{2N} v\pi\right) \quad (2.3)$$

$$\begin{aligned} u &= 0, 1, ..., M-1 \\ v &= 0, 1, ..., N-1 \\ c(k) &= \begin{cases} \frac{1}{\sqrt{2}} & \text{si k=0} \\ 1 & \text{sinon} \end{cases} \end{aligned} \quad (2.4)$$

Le processus de décompression étant totalement symétrique au processus de compression, le décodeur devra calculer la transformée inverse (IDCT) de la manière suivante :

$$f(m,n) = \frac{2}{\sqrt{MN}} \sum_{u=0}^{M-1} \sum_{v=0}^{N-1} c(u)c(v)F(u,v) \cos\left(\frac{2m+1}{2M}u\pi\right) \cos\left(\frac{2n+1}{2N}v\pi\right) \quad (2.5)$$

La DCT est donc une transformation complètement réversible[1].

### 2.5.2 Pourquoi effectuer une DCT ?

Elle permet, tout comme le changement d'espace de couleurs, de représenter l'information dans un domaine dans lequel elle pourra être compressée plus efficacement.

On pourrait se demander pourquoi on applique la FDCT aux blocs et non à l'image entière. Il y a deux raisons principales : tout d'abord l'application de la FDCT à l'image entière implique de nombreuses opérations arithmétiques, ce qui rend l'opération très lente. Ensuite, l'expérience montre que dans une image réelle, la corrélation entre pixels s'effectue sur une courte distance : une composante d'un pixel a une valeur proche de la valeur d'un pixel voisin, mais n'a rien à voir avec la valeur de pixels lointains.

Le premier coefficient $F(0,0)$ est appelé coefficient DC (Direct Component) et représente la valeur moyenne du bloc avant transformation. Les 63 autres coefficients sont appelés coefficients AC (Alternative Component).

Pourquoi effectuer une DCT et pas une Transformée de Fourier Discrète ? Parce que la discrétisation du domaine fréquentiel a eu pour effet de dupliquer l'image dans le domaine spatial, ce qui produit des discontinuités. De plus, le choix de la miroirisation dans la DCT utilisée produit une fonction paire, ce qui élimine les coefficients issus du sinus.

### 2.5.3 Comment effectuer la DCT ?

Bien que la formule ci-dessus soit mathématiquement correcte, la pratique préfère l'utilisation de transformations orthogonales. Les transformations orthogonales sont des opérations séparables, c'est-à-dire qu'une opération en deux dimensions est équivalente à deux opérations successives à une dimension.
En compression d'images, ces matrices sont réelles et orthogonales ($T^1 = T^{*T}$, où

[1] Pas tout à fait : la fonction cosinus étant transcendante, il y aura une légère perte d'information due à la précision limitée de la machine.

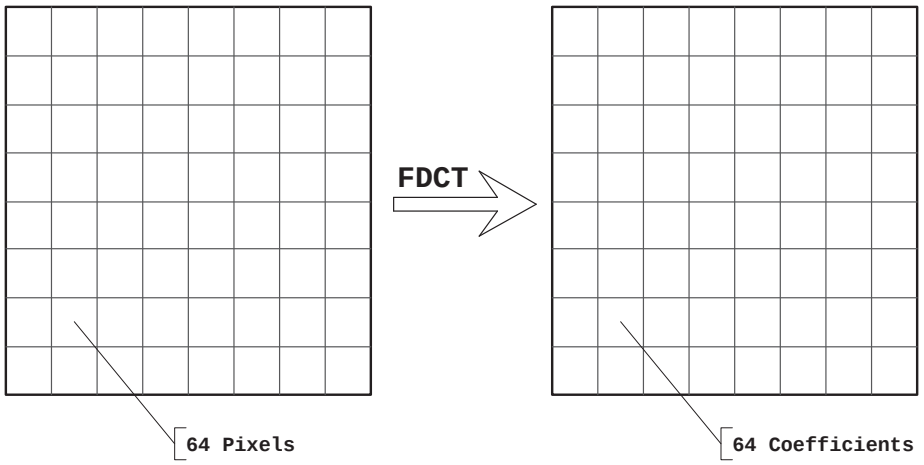



FIG. 2.7: Transformée en cosinus discrète d'un bloc 8x8.

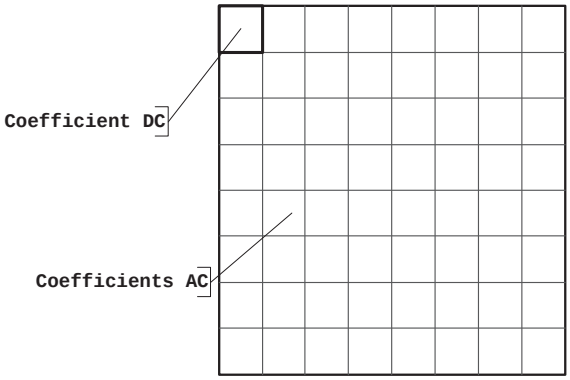



FIG. 2.8: Coefficients de la FDCT.

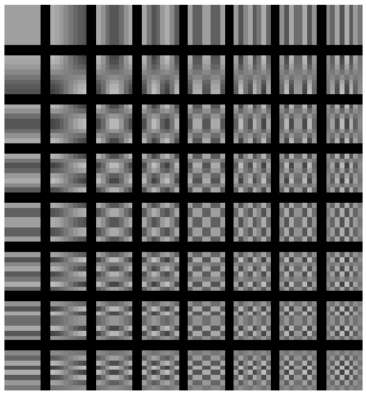


FIG. 2.9: Fonctions de base DCT.

$T^{*T}$ est la matrice conjuguée transposée de $T$). Il résulte des propriétés des matrices que $T$ est unitaire. La matrice de transformation $T$ est aussi considérée comme un ensemble de vecteurs colonnes qui forment une base dans l'espace à $n$ dimensions.
Si nous considérons que l'entrée est un bloc de $8 \times 8$ pixels $f(m,n)$, et que nous nous donnons deux matrices de transformation orthogonales $P$ et $Q$ (dans notre cas particulier de dimensions $n \times n$), nous avons :

$$F = PfQ \tag{2.6}$$

et

$$f = P^{-1}FQ^{-1} \tag{2.7}$$

Ces transformations ont donc la propriété d'utiliser le même algorithme dans un sens que dans l'autre. Dans le cas de la transformée en cosinus discrète, les matrices

(carrées : $N$=8) de transformation sont définies comme suit :

$$P = Q^T = C_N(k,l) = \begin{cases} \frac{1}{\sqrt{N}} & \text{si } l=0 \\ \sqrt{\frac{2}{N}} \cos\left[\frac{(2k+1)l\pi}{2N}\right] & \text{sinon} \end{cases} \tag{2.8}$$

## 2.6 Quantification

Jusqu'à présent, aucune transformation opérée sur l'image n'a entraîné de perte d'information. La quantification des coefficients fréquentiels est l'étape qui va permettre d'éliminer de l'information (et donc de diminuer le débit). C'est la seule étape de JPEG qui ne soit pas totalement réversible. Lorsque JPEG permet d'ajuster le rapport compression / qualité d'image, c'est lors de la quantification que ce paramètre intervient : dans chaque bloc, chacun des 64 coefficients est divisé par un coefficient de quantification ($QC$), et la valeur obtenue est arrondie à l'entier le plus proche. Plus grand est le coefficient de quantification, plus il y aura d'information perdue. Remarquons que même lorsque les coefficients sont minimum (égaux à 1) il y a une légère perte car les coefficients DCT ne sont généralement pas entiers.

Soient $F(u,v)$ les coefficients DCT et $Q(u,v)$ les coefficients de quantification, Le bloc $8 \times 8$ résultant est défini comme :

$$F^Q(u,v) = \text{Integer Round}\left(\frac{F(u,v)}{Q(u,v)}\right) \tag{2.9}$$

Lors de la décompression, l'opération «inverse» sera effectuée :

$$F^{Q'}(u,v) = F^Q(u,v) \times Q(u,v) \tag{2.10}$$

La matrice résultante $F^{Q'}(u,v)$ diffère de la matrice originale $F(u,v)$, tout d'abord par l'impossibilité d'effectuer l'opération inverse de l'arrondi, ensuite parce que nombre de coefficients de hautes fréquences qui avaient de petites valeurs (de l'ordre de l'unité) ont été divisés par un nombre assez élevé et arrondis à zéro. C'est d'ailleurs cette longue suite de 0 de $F^Q(u,v)$ qui va nous permettre de coder efficacement l'image.

Quelle est la forme exacte de ces tables de quantification ? Les coefficients de basses fréquences (coin supérieur gauche) seront toujours moins quantifiés que les hautes fréquences. De même, la composante de luminance sera moins quantifiée que les composantes de chrominance. Il n'existe malheureusement pas de table idéale car elle dépend des caractéristiques de l'image initiale. En théorie, l'utilisateur a la possibilité de spécifier ces tables. En pratique, l'utilisateur pressé par le temps a le choix entre deux approches :

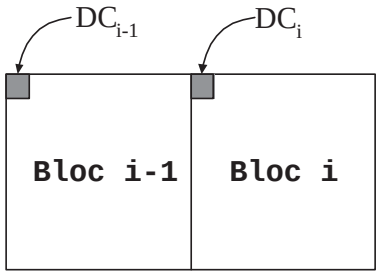
- Utiliser les tables de quantification par défaut. Ce sont deux tables définies par le comité JPEG pour chaque type de composante. Elles fournissent de bons résultats pour des taux compression moyens.

- Spécifier un paramètre unique $R$ à partir duquel une table de quantification unique est calculée : $Q_{ij} = 1 + (i + j) \times R$. On observe bien que dans cette expression les QC ont une valeur proche de l'unité dans le coin supérieur gauche et croissent lorsqu'on se dirige vers le coin inférieur droit. Notons que cette formule n'est qu'indicative : en effet, les détails de l'implémentation ne sont pas spécifiés dans la norme.

## 2.7 Codage

### 2.7.1 Coefficients DC

Le coefficient DC représente la valeur moyenne du bloc $8 \times 8$ original. Dans une image réelle, les blocs adjacents sont corrélés dans le sens où les valeurs moyennes de blocs adjacents sont proches. Cela implique donc que les valeurs des coefficients DC voisins ne diffèrent que très peu. Par un système DPCM (Differential Pulse Code Modulation), les coefficients DC sont codés en utilisant la valeur du coefficient DC du bloc précédent.




$$\text{diff} = DC_i - DC_{i-1}$$

FIG. 2.10: Codage par DCPM.

Ces valeurs sont ensuite transformées dans une forme intermédiaire afin d'être plus facilement codée par la suite :

| *Symbole* 1 | *Symbole* 2 |
|---|---|
| (TAILLE) | (AMPLITUDE) |

Où (TAILLE) représente le nombre de bits nécessaires à la représentation d'(AMPLITUDE). Une analyse numérique [8] a montré que les coefficients DC peuvent prendre des valeurs dans l'intervalle $[-2^{11}, 2^{11} - 1]$. AMPLITUDE, quant à lui, représente soit la valeur du coefficent DC du premier bloc de l'image, soit la différence *diff*.

### 2.7.2 Coefficients AC

Les 63 coefficients AC sont lus dans l'ordre zig-zag représenté sur la figure 2.11.

La quantification a eu pour effet de ne rendre que quelques coefficients AC non nuls, séparés par de longues suites de zéros. L'ordre zig-zag permet donc d'obtenir de

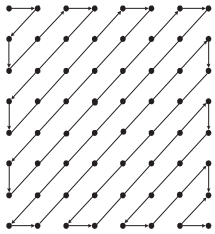


FIG. 2.11: Lecture en zigzag des coefficients.

longues séquences de zéros, ce qui va réduire la taille de la forme intermédiaire dans laquelle ils sont transformés :

| *Symbole* 1 | *Symbole* 2 |
|---|---|
| (COURSE, TAILLE) | (AMPLITUDE) |

COURSE représente le nombre de zéros consécutifs précédant le coefficient AC actuel. TAILLE représente toujours le nombre de bits d'AMPLITUDE, qui est la valeur du coefficient. COURSE est codé sur 4 bits, mais si une course de plus de quinze zéros apparaît, on accepte la représentation $(15, 0)$ suivi d'un nouveau doublet ( *Symbole 1,Symbole 2*). Le *Symbole 1* *(0,0)* (= EOB = End Of Block) signifie que l'on a atteint la dernière valeur non-nulle du bloc et qu'il ne reste plus que des zéros (ce qui suffit pour marquer la fin du bloc, puisqu'on en connaît le nombre d'éléments). Dans ce cas, les coefficient sont compris dans l'intervalle $[-2^{10}, 2^{10}-1]$. Le tableau ci-dessous reprend donc les différentes valeurs de TAILLE ainsi que les valeurs possibles d'AMPLITUDE correspondantes :

| Taille | Amplitude |
|---|---|
| 1 | $-1, 1$ |
| 2 | $-3, -2, 2, 3$ |
| 3 | $[-7, \ldots, -4] \cup [4, \ldots, 7]$ |
| 4 | $[-15, \ldots, -8] \cup [8, \ldots, 15]$ |
| 5 | $[-31, \ldots, -16] \cup [16, \ldots, 31]$ |
| 6 | $[-63, \ldots, -32] \cup [32, \ldots, 63]$ |
| 7 | $[-127, \ldots, -64] \cup [64, \ldots, 127]$ |
| 8 | $[-255, \ldots, -128] \cup [128, \ldots, 255]$ |
| 9 | $[-511, \ldots, -256] \cup [256, \ldots, 511]$ |
| 10 | $[-1023, \ldots, -512] \cup [512, \ldots, 1023]$ |

### 2.7.3 Codage de Huffmann

Maintenant que tous les coefficients sont représentés sous leur forme intermédiaire, ils peuvent être encodés. Le mode baseline n'autorise que le codage de Huffmann (dans les autres modes les symboles peuvent subir un codage arithmétique). Rappelons simplement que le codage de Huffmann attribue des codes de longueur variable (VLC

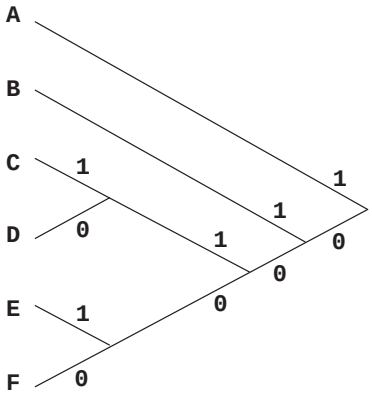
= Variable Length Coding) aux différents symboles selon leur probabilité d'apparition. Ce codage se base sur des tables qui doivent être fournies en entrée de l'encodeur.

Rappelons les grandes étapes d'un codeur de Huffmann :

- après avoir trié les symboles par ordre de probabilité croissante, nous construisons un arbre de codage : les deux symboles les moins probables sont unis pour constituer un noeud qui forme deux branches d'un arbre. Ce processus est réappliqué récursivement jusqu'à obtenir un arbre complet. Les feuilles de cet arbre sont ensuite étiquetées par la somme des probabilités des symboles des deux branches formant ce noeud. La racine a une probabilité de 1.
- pour créer le mot de code pour chaque symbole, nous assignons, en commençant par la racine, un 0 et un 1 à chaque paire de branches de l'arbre. Nous concaténons ensuite les bits assignés à chaque branche pour obtenir le code.

Illustrons ceci par un exemple simple :

| Symbole | Probabilité |
|---|---|
| $A$ | 0.5 |
| $B$ | 0.3 |
| $C$ | 0.07 |
| $D$ | 0.05 |
| $E$ | 0.05 |
| $F$ | 0.03 |




| Symbole | Code |
|---|---|
| $A$ | 1 |
| $B$ | 01 |
| $C$ | 0011 |
| $D$ | 0010 |
| $E$ | 0001 |
| $F$ | 0000 |

Le codage de Huffman a, de plus, la particularité de produire des codes de préfixe unique, ce qui permet de les décoder instantanément sans ambiguïté.

## 2.8 JFIF

JPEG est une méthode de compression, pas un format de fichier. Le flot binaire que nous obtenons en sortie du codeur de Huffmann n'est donc pas suffisant pour qu'une application puisse l'utiliser. Il faut entre autres spécifier les tables de quantification et les tables de Huffmann utilisées.
JFIF (JPEG File Interchange Format) inclut les tables nécessaires, assure la compatibilité entre plate-formes, permet l'insertion d'informations spécifiques à une application sans nuire à la décodabilité par une autre application, spécifie l'espace de couleur utilisé, le nombre et la taille des composantes, la taille de l'image, etc.

# Chapitre 3

# Statistique des coefficients de la DCT

## 3.1 Objectif

Le but de ce chapitre est de montrer que la prédiction par codage différentiel (DPCM) des coefficients DC fonctionne et d'expliquer pourquoi. Nous tenterons également d'expliquer pourquoi on ne peut prédire les coefficients AC de la même façon.

## 3.2 Rappels de théorie de l'information

Soit $X$ une source d'information qui génère une série de symboles à partir d'un alphabet $X = (x_1, x_2, \ldots, x_n)$ tel qu'à chaque symbole correspond une probabilité d'occurrence : $P = \{p(x_1), p(x_2), \ldots, p(x_n)\}$.
L'*entropie* de cette source est définie comme étant la somme des incertitudes liées à chaque symbole :

$$H(X) = -\sum_{i=1}^{n} p(x_i) \log_2 p(x_i) \tag{3.1}$$

Soit $Y$ une seconde source d'information basée sur l'alphabet $Y = (y_1, y_2, \ldots, y_m)$ ; son entropie est notée $H(Y)$. L'*entropie conjointe* est définie par :

$$H(X,Y) = -\sum_{i=1}^{n}\sum_{j=1}^{m} p(x_i, y_j) \log_2 p(x_i, y_j) \tag{3.2}$$

On définit également l'*entropie conditionnelle* :

$$H(X|Y) = -\sum_{i=1}^{n}\sum_{j=1}^{m} p(x_i, y_j) \log_2 p(x_i|y_j) \tag{3.3}$$

L'entropie conditionnelle mesure l'incertitude liée à la source $X$ si les symboles générés par $Y$ sont donnés. Parmi les propriétés les plus intéressantes, il est possible de montrer que $H(X|Y) \leqq H(X)$ c'est-à-dire qu'en connaissant les symboles de $Y$, l'incertitude liée à la source $X$ sera inférieure ou égale à celle sans préconnaissance de $Y$. (Il y a égalité lorsque les deux sources sont indépendantes.)

## 3.3 Coefficients DC et DPCM

Les coefficients DC de chaque bloc sont proportionnels à la valeur moyenne du bloc original. Le codage différentiel (DPCM) exploite la redondance entre un symbole et le symbole suivant. Les coefficients DC étant corrélés (en effet, la valeur moyenne d'un bloc de pixels est assez proche de la valeur moyenne d'un bloc voisin), il semble naturel d'employer ce codage : on code le premier symbole, on calcule la différence avec le second symbole et on code cette différence. Celle dernière nécessite moins de bits que les symboles eux-mêmes car cette différence est souvent faible. On code ensuite la différence entre le deuxième symbole et le troisième, etc.

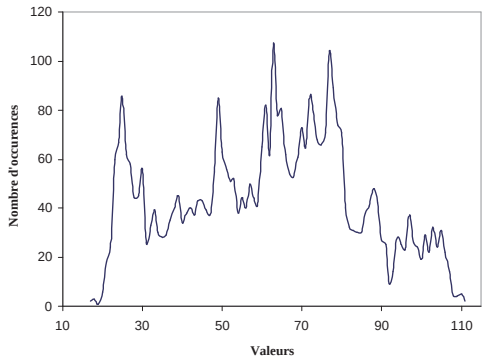



FIG. 3.1: Distribution des coefficients *DC* pour Lena avant DPCM

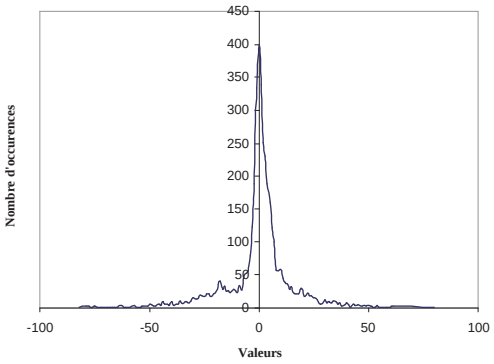



FIG. 3.2: Distribution des coefficients *DC* pour Lena après DPCM

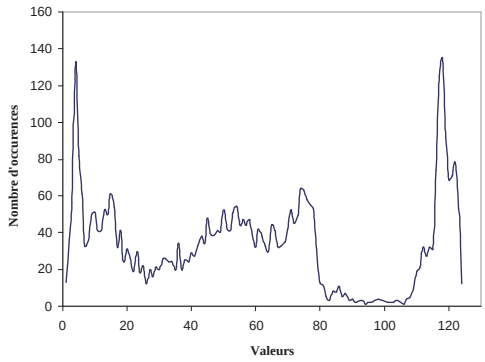



FIG. 3.3: Distribution des coefficients *DC* de San Francisco avant DPCM

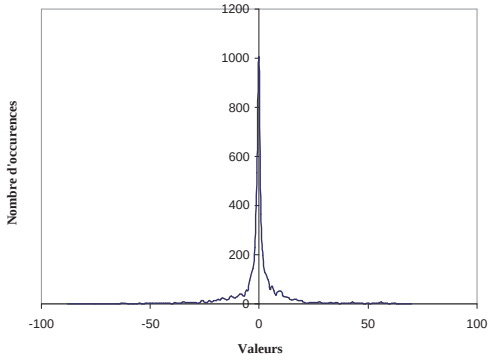



FIG. 3.4: Distribution des coefficients *DC* de San Francisco après DPCM

Dans des systèmes plus complexes et performants, on établit une fonction de prédiction qui permet d'estimer la valeur d'un symbole en fonction de la valeur des symboles voisins. On code alors l'erreur de prédiction, qui est l'écart entre la vraie valeur du symbole et la valeur prédite. La fonction de prédiction peut être plus ou moins

complexe selon l'ordre de prédiction (nombre de symboles impliqués dans le calcul de prédiction), la topologie (position des symboles voisins utilisés dans le calcul), l'utilisation de pondération (poids affectés aux symboles du calcul de prédiction selon leur position relative par rapport au symbole prédit ou selon les propriétés statistiques de l'ensemble de symboles). L'efficacité d'une méthode prédictive est fortement liée à l'adéquation de la fonction de prédiction aux symboles à coder afin que les valeurs d'erreur de prédiction soient faibles.

La théorie de l'information affirme que le nombre moyen de bits nécessaires à la représentation d'un symbole provenant d'une source ne peut être inférieur à l'entropie de cette source. Nous avons donc appliqué les formules 3.1 et 3.3 aux distributions des figures 3.1, 3.2, 3.3 et 3.4, et comme le montre le tableau ci-dessous, l'entropie des coefficients $DC$ (les symboles) a diminué :

| | avant DPCM | après DPCM |
|---|---|---|
| Image Lena | $H_0(DC) = 4.38$ | $H_1(DC) = 3.77$ |
| Image San-Francisco | $H_0(DC) = 4.51$ | $H_1(DC) = 3.2$ |

En effet, l'efficacité du codage par DPCM, est due à la corrélation existante entre les valeurs des coefficients $DC$. Après DPCM, nous ne considérons plus l'entropie des coefficients seuls (entropie d'ordre 0) mais bien l'entropie des coefficients $DC$ *conditionnellement* à la connaissance du coefficient précédent (entropie conditionnelle = entropie d'ordre 1).

Ci-dessous, un modèle schématique du codeur et du décodeur DPCM :

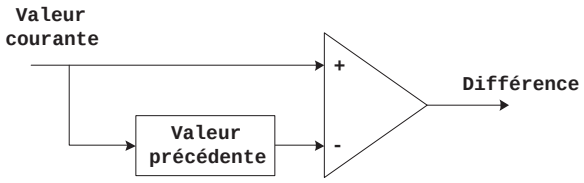



FIG. 3.5: encodeur DPCM

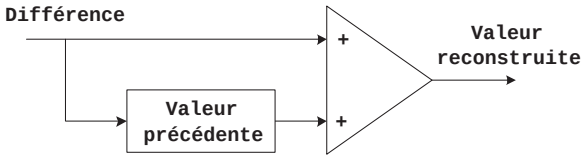



FIG. 3.6: décodeur DPCM

## 3.4 Coefficients AC

Dans cette section nous allons discuter des différentes propriétés des statistiques des coefficients AC. Nous nous attacherons également à illustrer les différences entre DCT et DFT. Plus précisément, bien que ce soit la DCT qui ait été choisie comme transformée dans JPEG, nous illustrerons le meilleur comportement de la DFT dans certaines situations.

Cela fait longtemps que l'on cherche à déterminer la loi de distribution des coefficients AC. Plusieurs méthodes de *fit* ont été testées sur plusieurs types de distributions ; il est maintenant généralement considéré [1] que la distribution des coefficients AC est une Laplacienne. La distribution laplacienne : $p(x) = \frac{\lambda}{2} e^{-\lambda|x|}$ (figure 3.7) est symétrique par rapport à l'axe $x = 0$. La figure 3.8 montre l'ajustement d'une Laplacienne de paramètre $\lambda = 0.7$ sur le premier coefficient AC de l'image Lena.

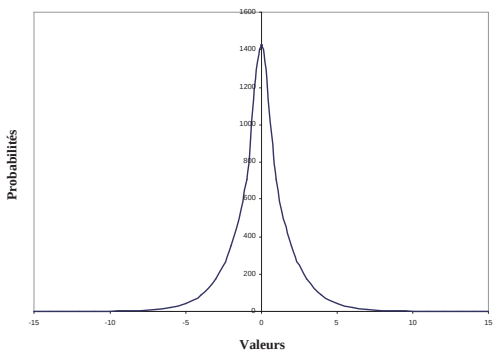



FIG. 3.7: Distribution laplacienne

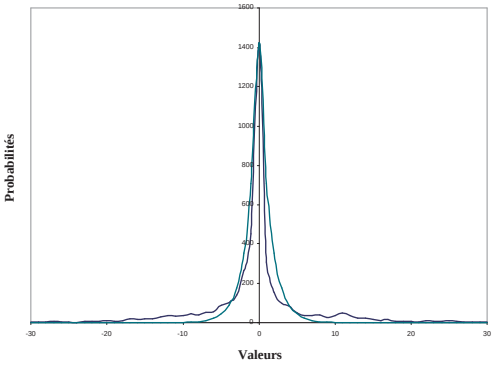



FIG. 3.8: Fit sur $AC_0$ de Lena

Les figures 3.9 à 3.14 montrent les distributions des 3 premiers coefficents horizontaux ($AC_0$, $AC_4$ et $AC_5$) des images Lena et San-Francisco :

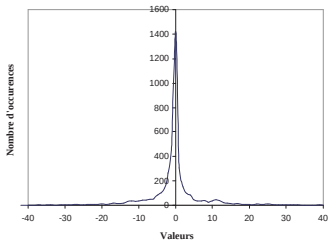



FIG. 3.9: Distribution du coefficient $AC_0$ de Lena

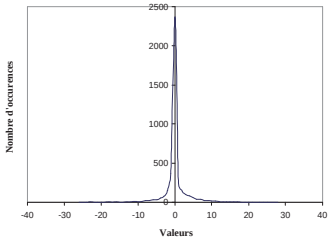



FIG. 3.10: Distribution du coefficient $AC_4$ de Lena

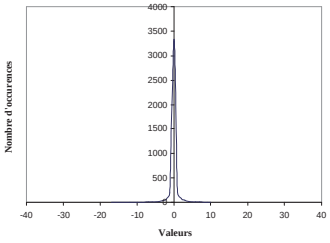



FIG. 3.11: Distribution du coefficient $AC_5$ de Lena

Ces différentes distributions de coefficients confirment la justesse du modèle [1]. Celui-ci justifie également la diminution de largeur des distributions lorsqu'on se dirige vers les hautes fréquences, ce que nous observons expérimentalement : les images naturelles possédant une certaine corrélation spatiale, leur énergie est principalement concentrée dans les basses fréquences ; il est donc normal de trouver moins d'énergie, et donc une variance moindre, dans les composantes de hautes fréquences.

Pour nous en assurer, nous avons donc étudié des images plus simples : un signal à 1 dimension périodique (un sinus dont nous avons fait varier la période). Nous

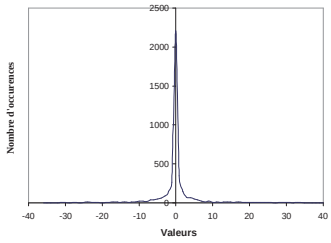



FIG. 3.12: Distribution du coefficient $AC_0$ de San Francisco

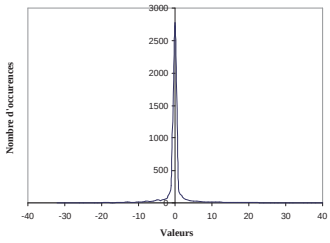



FIG. 3.13: Distribution du coefficient $AC_4$ de San Francisco

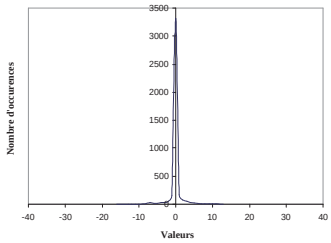



FIG. 3.14: Distribution du coefficient $AC_5$ de San Francisco

avons fait subir à ces images de taille $512 \times 512$ un découpage en blocs $8 \times 8$ suivi d'une transformée en cosinus discrète. Nous avons ensuite extrait les probabilités du coefficient $DC$ et des 3 premiers coefficients $AC$ horizontaux, et pour ces distributions nous avons finalement calculé la variance. Ce processus a été appliqué à des images dont la période varie de 0 à 8. Les figure 3.15, 3.16, 3.17 et 3.18 montrent les résultats :

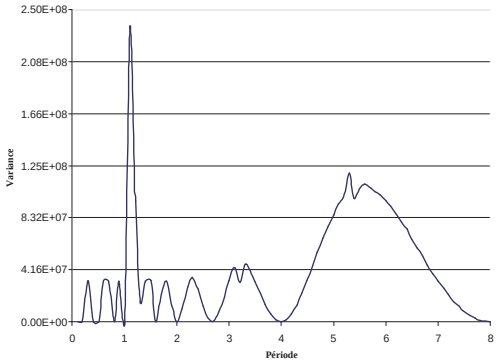



FIG. 3.15: Variance du coefficient $DC$ sur images périodiques de période $\tau \in [0 \dots 8]$

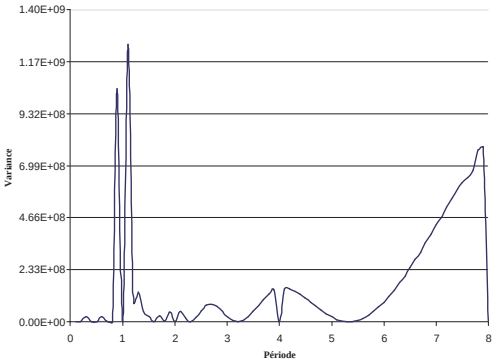



FIG. 3.16: Variance du coefficient $AC_0$ sur images périodiques de période $\tau \in [0 \dots 8]$

Ces résultats confirment bien ce à quoi nous nous attendions :

- La variance augmente avec la période (elle est donc plus élevée dans les basses fréquences) à partir d'une période de 4 correspondant à la fréquence de NYQUIST.
- La variance est nulle pour les sous-multiples entiers de 8 (1, 2 et 4) ; de même que, pour le coefficient $AC_0$, les périodes 3.2 et 5.3 correspondantes aux sous-multiples non-entiers de 16 (la variance du $AC_4$ s'annule elle aussi en 1.6 et en 2.6, sous-mutiples de 32 et la variance du $AC_5$ s'annule en 3.2 sous-multiple de 64).

Remarquons qu'il est également possible de déterminer analytiquement la valeur de certains coefficients pour un signal simple. Si nous reprenons l'expression suivante

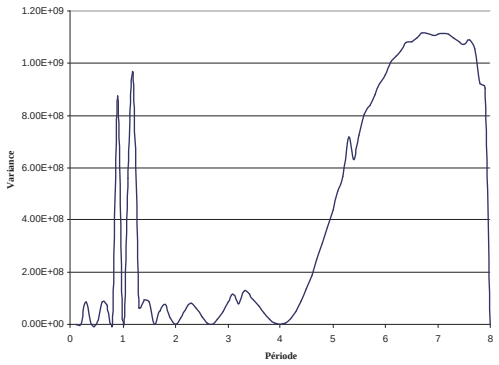



FIG. 3.17: Variance du coefficient $AC_4$ sur images périodiques de période $\tau \in [0 \dots 8]$

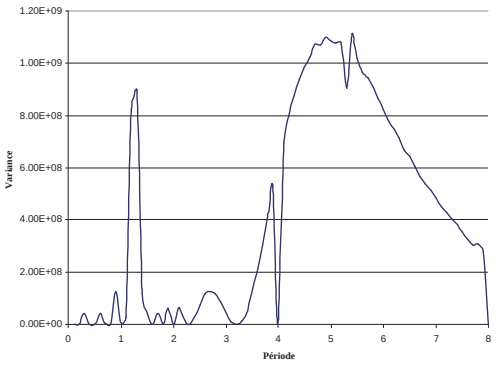



FIG. 3.18: Variance du coefficient $AC_5$ sur images périodiques de période $\tau \in [0 \dots 8]$

pour définir les coefficients DCT d'un signal $x(n)$ :

$$C(0) = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} x(n) \tag{3.4}$$

$$C(k) = \sqrt{\frac{2}{N}} \sum_{n=0}^{N-1} x(n) \cos \frac{(2n+1)\, k\, \pi}{2N} \tag{3.5}$$

et que nous considérons comme signal $x(n) = \cos(an + b)$, nous obtenons :

$$C(0) = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} \cos(an + b) \tag{3.6}$$

$$\begin{aligned} C(k) &= \sqrt{\frac{2}{N}} \sum_{n=0}^{N-1} \cos(an+b) \cos \frac{(2n+1)\, k\, \pi}{2N} \\ &= \sqrt{\frac{1}{2N}} \sum_{n=0}^{N-1} \cos\left(an + b - \frac{(2n+1)\, k\, \pi}{2N}\right) \cos\left(an + b + \frac{(2n+1)\, k\, \pi}{2N}\right) \end{aligned} \tag{3.7}$$

expression qui nous permet de calculer les valeurs des coefficients analytiquement. En particulier pour $k = 4$, si l'on prend $x(n) = \cos(\frac{2\pi n}{8})$, nous obtenons :

$$C(4) = \sqrt{\frac{1}{2N}} \sum_{n=0}^{N-1} \cos\left(\frac{2\pi n}{8} - \frac{(2n+1)\, 4\, \pi}{2N}\right) \cos\left(\frac{2\pi n}{8} + \frac{(2n+1)\, 4\, \pi}{2N}\right) = 0 \tag{3.8}$$

Reste maintenant à comprendre l'effet de la phase sur la variance des différentes distributions :
Les graphiques 3.19 à 3.22 montrent que non seulement la période, mais aussi la phase d'un signal influencent la variance des coefficients DCT. Puisque la phase modifie la variance, la prédiction du signal en sera fonction elle aussi.

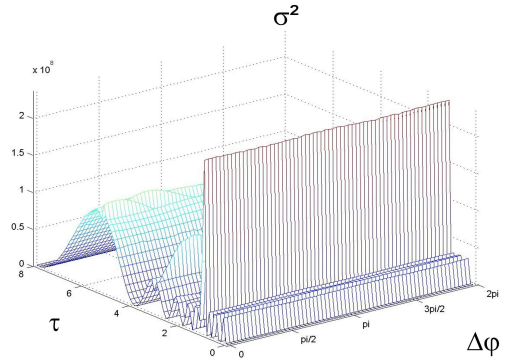



FIG. 3.19: Variance du coefficient $DC$ en fonction de la période et du décalage de phase

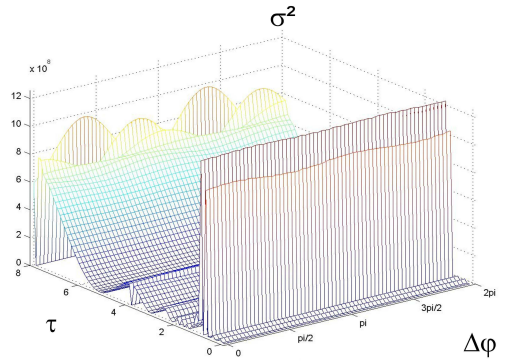



FIG. 3.20: Variance du coefficient $AC_0$ en fonction de la période et du décalage de phase

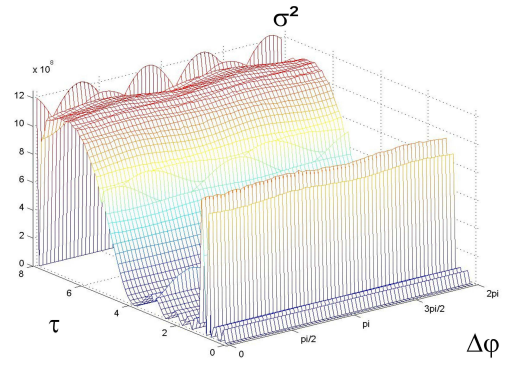



FIG. 3.21: Variance du coefficient $AC_4$ en fonction de la période et du décalage de phase

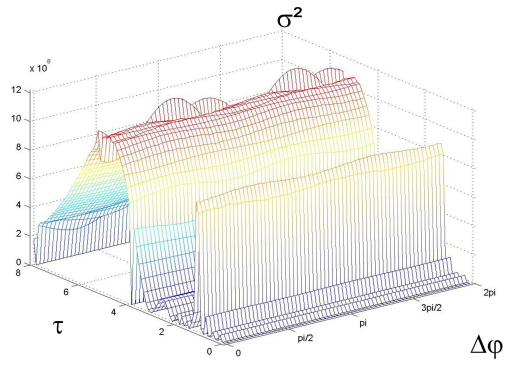



FIG. 3.22: Variance du coefficient $AC_5$ en fonction de la période et du décalage de phase

### 3.4.1 Coefficients AC de la DFT

Nous allons maintenant comparer les différences existant entre la DCT et la DFT. Les figures 3.23 à 3.28 montrent les distributions des coefficients AC de Lena et San-Francisco où l'usage de la DCT a été remplacée par une DFT :
Ces distributions ne nous apprennent pas grand-chose. Une fois encore, nous allons étudier des signaux synthétiques, plus simples, dans le but de pouvoir en tirer des conclusions : nous avons calculé les valeurs des deux transformées sur deux blocs adjacents appartenant à une même texture pour des images de différentes périodes. Les résultats sont présentés sur les figures 3.29 à 3.34.

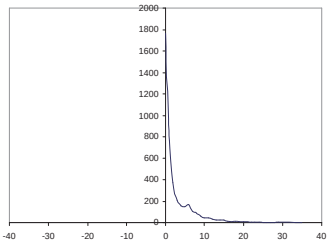



FIG. 3.23: Distribution du coefficient $AC_0$ de Lena

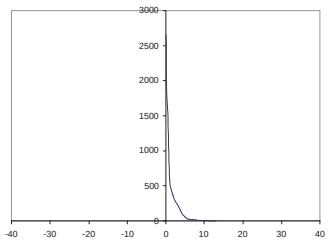



FIG. 3.24: Distribution du coefficient $AC_4$ de Lena

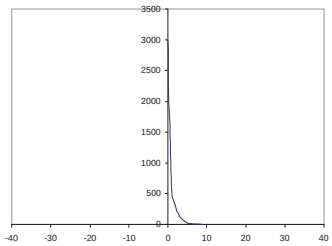



FIG. 3.25: Distribution du coefficient $AC_5$ de Lena

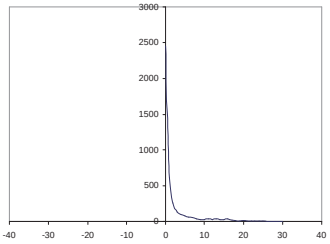



FIG. 3.26: Distribution du coefficient $AC_0$ de San Francisco

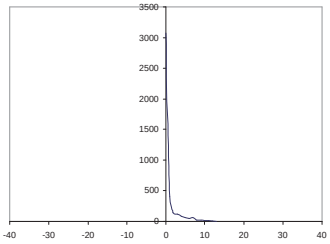



FIG. 3.27: Distribution du coefficient $AC_4$ de San Francisco

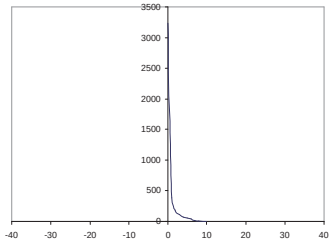



FIG. 3.28: Distribution du coefficient $AC_5$ de San Francisco

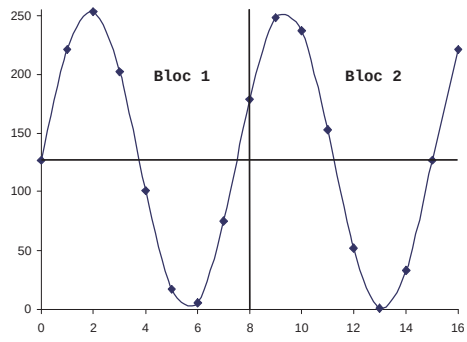



FIG. 3.29: Deux blocs adjacents d'un signal synthétique de période 7.5

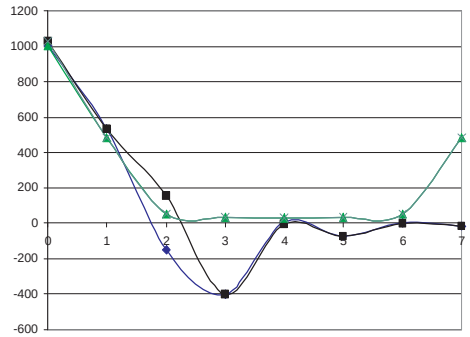



FIG. 3.30: Les DCT (bleu-noir) et DFT (vert) correspondant

Quelles conclusions pouvons-nous en tirer ?

- Les DFT de blocs voisins sont identiques : en effet, soit $g(t)$ un signal non périodique et $G(f)$ sa transformée de Fourier (notation $g(t) \leftrightharpoons G(f)$), la propriété de translation temporelle de la transformée de Fourier nous dit :

$$g(t-\tau) \leftrightharpoons G(f)\, e^{-2\pi j f \tau} \tag{3.9}$$

  Translater la fonction $g(t)$ de $\tau$ ne modifie donc pas l'amplitude de sa transformée (la propriété d'invariance n'est effectivement valable qu'en l'absence de

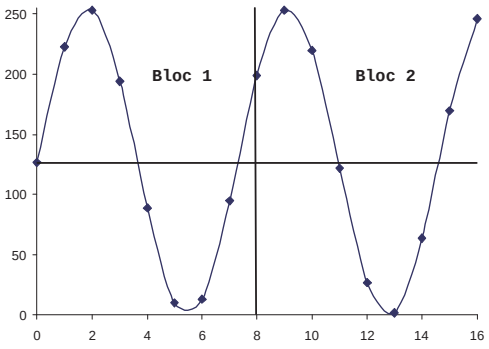



FIG. 3.31: Deux blocs adjacents d'un signal synthétique de période 7.3

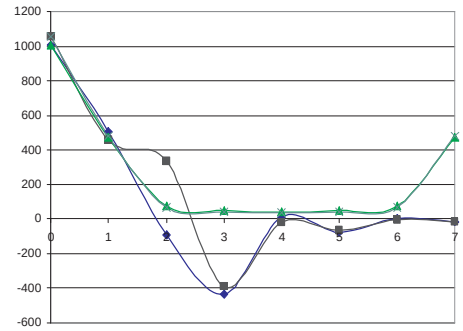


FIG. 3.32: Les DCT (bleu-noir) et DFT (vert) correspondant

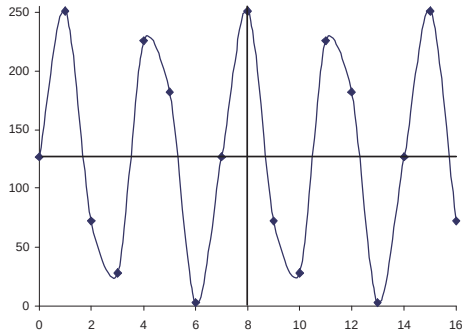


FIG. 3.33: Deux blocs adjacents d'un signal synthétique de période 3.7

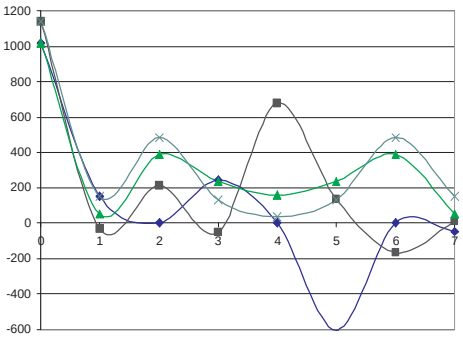


FIG. 3.34: Les DCT (bleu-noir) et DFT (vert) correspondant

fenêtre ; bien qu'elle ne le soit pas dans le cas contraire, elle le sera d'autant plus que la durée d'observation du signal périodique sera longue), par contre sa phase est modifiée d'un facteur linéaire $-2\pi jf\tau$ (remarquons que les DFT de l'image de période 3.7 diffèrent quelque peu, cela est simplement dû à un mauvais échantillonnage).

- Plus la période est élevée, plus la DCT est proche de la DFT. Cela provient du fait que, plus la période est grande, plus les sauts de phase entre blocs sont moindres.

Nous pouvons donc conclure que, en gardant la partie imaginaire (et donc l'information de phase), la DFT a un meilleur comportement que la DCT. En effet, la DFT ayant une variance plus petite que la DCT, il sera plus facile de prédire un coefficient d'un bloc voisin à partir du coefficient du bloc courant. L'idéal serait d'avoir un coefficient non-nul ayant une variance nulle. Cette étude des coefficients tend donc à démontrer que la DFT convient mieux au codage des changements de phase entre blocs voisins.

# Chapitre 4

# Prédiction intra-bloc de coefficients AC

Dans ce chapitre, nous allons exposer une technique de prédiction des coefficients AC, proposée dans la norme [3], et envisagée dans le but de réduire les discontinuités entre blocs adjacents présentes dans les images compressées à l'aide de transformées telles que la DCT. Bien que cette technique ne soit pas imposée dans le standard JPEG, elle peut y être intégrée.

FIG. 4.1: Illustration de l'effet de bloc dans l'image Lena

Le problème de l'effet de bloc apparaît lorsque le débit devient trop faible : l'utilisateur a, en effet, la possibilité de fixer le taux de compression en ajustant un paramètre influant sur la quantification. Le cas limite est bien sûr lorsque la quantification est telle que seul le coefficient DC subsiste, auquel cas les frontières des blocs deviennent parfaitement visibles, comme lors du premier passage du codeur dans le cas du mode progressif.

## 4.1 Prédiction des coefficients AC

La technique [3] propose de prédire les coefficients AC de basse fréquence en utilisant les variations des coefficients DC dans un tableau de $3 \times 3$ blocs centrés sur le bloc considéré. Ce tableau $3 \times 3$ est modélisé par un polynôme bidimensionnel du second degré de la forme :

$$P(x,y) = A_1x^2y^2 + A_2x^2y + A_3xy^2 + A_4x^2 + A_5xy + A_6y^2 + A_7x + A_8y + A_9 \quad (4.1)$$

qui est utilisé pour prédire les 64 coefficients du bloc central. Les coefficients $A_1, \ldots, A_9$ sont uniquement déterminés en imposant la contrainte que les valeurs moyennes calculées pour la surface quadratique doivent correspondre aux valeurs DC : $DC_1, \ldots, DC_9$.

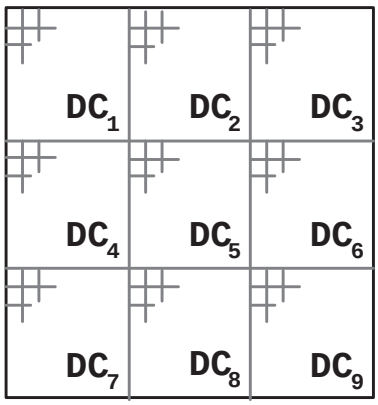



FIG. 4.2: Macrobloc utilisé pour la prédiction

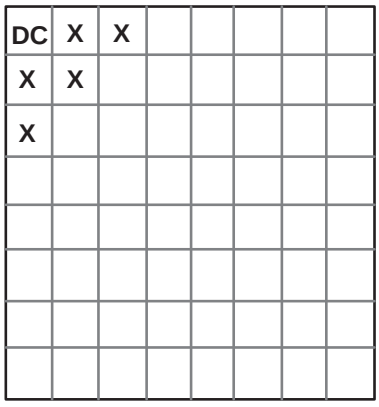



FIG. 4.3: Coefficients $AC$ prédits

Les valeurs des coefficients DC sont donc utilisées pour prédire les cinq premiers coefficients AC du bloc central dans l'ordre zig-zag :

$$\begin{aligned}
AC_0 &= (1.113885/8) \times (DC_4 - DC_6) \\
AC_1 &= (1.113885/8) \times (DC_2 - DC_8) \\
AC_2 &= (0.27881/8) \times (DC_2 + DC_8 - 2 \times DC_2) \\
AC_3 &= (0.16213/8) \times [(DC_1 - DC_3) - (DC_7 - DC_9)] \\
AC_4 &= (0.27881/8) \times (DC_4 + DC_6 - 2 \times DC_5)
\end{aligned} \quad (4.2)$$

## 4.2 Prédiction des coefficients AC quantifiés

Dans la section précédente, nous effectuions la prédiction avant quantification. Ces équations peuvent être réécrites sous une forme utilisant les coefficients DC quantifiés afin de prédire les coefficients AC quantifiés, ce qui permet de ne plus effectuer de calculs que sur des entiers. Les termes multiplicatifs des équations ci-dessus sont

multipliés par 32 et arrondis à l'entier le plus proche. Ces nouveaux facteurs sont ensuite échelonnés par le rapport des éléments de quantification des coefficients DC et AC et sont à nouveau arrondis. De plus, la normalisation de la DCT introduit un facteur 8 supplémentaire sur les coefficients DC non-quantifiés. Finalement, nous avons :

$$\begin{aligned}
\mathsf{Q}AC_0 &= \frac{R\,Q_0 + 36\,Q_{DC} \times (DC_4 - DC_6)}{256 Q_0} \\
\mathsf{Q}AC_1 &= \frac{R\,Q_1 + 36\,Q_{DC} \times (DC_2 - DC_8)}{256 Q_1} \\
\mathsf{Q}AC_2 &= \frac{R\,Q_2 + 9\,Q_{DC} \times (DC_2 + DC_8 - 2 \times DC_2)}{256 Q_2} \\
\mathsf{Q}AC_3 &= \frac{R\,Q_3 + 5\,Q_{DC} \times [(DC_1 - DC_3) - (DC_7 - DC_9)]}{256 Q_3} \\
\mathsf{Q}AC_4 &= \frac{R\,Q_4 + 9\,Q_{DC} \times (DC_4 + DC_6 - 2 \times DC_5)}{256 Q_4}
\end{aligned} \tag{4.3}$$

où $\mathsf{Q}AC_n$ est la prédiction du $n^{ème}$ coefficient AC quantifié dans l'ordre zig-zag, $R$ est une constante égale à $\pm 128$ selon que le numérateur est positif ou négatif, $Q_n$ est le rapport des éléments de quantification du $n^{ème}$ élément et $Q_n$ est la valeur quantifiée du coefficient DC du $n^{ème}$ bloc adjacent.

La prédiction des coefficients AC permet de supprimer une grande partie des artefacts liés à l'effet de bloc en lissant certaines parties de l'image. Cependant, la prédiction est souvent fausse près des bords. Les valeurs prédites doivent donc être limitées afin de ne pas dépasser la plus petite amplitude quantifiée que le décodeur peut décoder, sous peine d'introduire de grands artefacts.

Originellement, la prédiction des coefficients AC a été présentée comme un codec supplémentaire, améliorant la compression de 2%. Il a été ensuite relégué au rang de simple option par le comité JPEG afin de simplifier le système.

# Chapitre 5

# Prédiction inter-bloc

Ce chapitre présente de manière générale une technique de prédiction, basée sur la notion de texture, que nous tenterons d'appliquer au codage et à la segmentation d'images.

## 5.1 Algorithme de prédiction

Bien que la prédiction intra-images ait été étudiée lors du processus de standardisation de JPEG et de MPEG, aucun schéma de prédiction entre blocs n'a été inclus, excepté pour le coefficient DC. Il en résulte une incapacité à prédire les coefficients AC de blocs voisins. Ce chapitre discute de la possibilité de prédiction inter-bloc à l'aide d'un algorithme d'extrapolation dans le cadre d'une application au codage.

Pourquoi est-il si compliqué de prédire les coefficients AC ? Dans le cadre de JPEG, l'utilisation de la DCT supprime la partie imaginaire de la transformée de Fourier, et donc l'information de phase nécessaire à la prédiction des coefficients : les distributions spectrales de deux blocs voisins appartenant à la même texture ne sont plus corrélées. Dès lors, la prédiction des coefficients AC de deux blocs voisins est inefficace.

### 5.1.1 Modèle

Les algorithmes d'extrapolation tentent généralement de sélectionner les positions et les amplitudes de certaines fréquences. Malheureusement, la région spatiale utilisée pour calculer le spectre est étendue à un support rectangulaire. Dès lors, le spectre obtenu contient l'information propre au motif (ce qui nous intéresse), mais aussi l'information conçernant le contour de ce motif. La dissociation de ces deux informations n'est pas aisée : il faudrait parvenir à sélectionner uniquement les "bonnes" fréquences. Cette imbrication des deux spectres est connu sous le nom d'*effet de fenêtre*. Une manière de supprimer cet effet de fenêtre est l'extrapolation de la texture hors de la fenêtre jusqu'à occupation complète d'un bloc rectangulaire, auquel cas nous aurons supprimé l'information de contour.

Notre algorithme d'extrapolation travaille dans l'espace fréquentiel, c'est pourquoi nous utiliserons indifféremment les termes *d'extrapolation* et de *déconvolution*.

Exprimons tout cela dans un formalisme plus mathématique. Pour une meilleure lisibilité, nous utiliserons une notation unidimensionnelle, le passage à deux dimensions ne présentant aucune difficulté : Soit $y(n)$ le signal observé. $y(n)$ s'obtient par *fenêtrage* de la fonction $f(n)$ originale à extrapoler :

$$y(n) = w(n)\, f(n) \qquad n = 0, 1, \ldots, N-1 \tag{5.1}$$

avec

$$w(n) = \begin{cases} 1 & \text{pour} \quad n \in D_w \\ 0 & \text{si} \quad n \notin D_w \end{cases} \tag{5.2}$$

où $D_w$ est la partie du domaine de la fonction $w(n)$ sur laquelle la valeur de la fenêtre est 1. Le problème de l'extrapolation consiste à chercher les valeurs de la fonction $f(n)$, en dehors du domaine $D_w$, en se basant sur $y(n)$. Tenter de calculer $f(n) = \frac{y(n)}{w(n)}$ n'est bien sûr pas une solution intéressante puisque le seul domaine sur lequel le rapport a un sens est $D_w$ ($w(n) \neq 0$), c'est-à-dire justement lorsque $f(n)$ est connu !

Déterminer $f(n)$ hors de $D_w$ requiert une meilleure modélisation du problème : nous pouvons réécrire l'équation $y(n) = w(n)\, f(n)$ sous la forme du produit de convolution des transformées respectives dans le domaine spectral (Théorème de multiplication) :

$$Y(k) = W(k) \otimes F(k) \qquad k = 0, 1, \ldots, N-1 \tag{5.3}$$

Ceci montre clairement que le spectre de la fonction à extrapoler est modifié par le spectre de la fenêtre $W(k)$. Sous la condition que le contenu spectral soit bien marqué, ce problème a une solution. Reste ensuite à effectuer la transformée de Fourier inverse afin d'obtenir $f(n)$.

La démarche habituelle d'extrapolation consiste à construire d'abord une extrapolation du signal sur son support à partir d'un modèle. Les fonctions de décomposition mises en oeuvre lors de l'approximation sont ensuite étendues hors support en conservant les coefficients des fonctions exprimées à l'intérieur du domaine de départ.

### 5.1.2 L'algorithme de déconvolution sélective

L'algorithme de déconvolution sélective qui résoud le problème de l'extrapolation [5] est un processus qui sélectionne les plus grandes raies spectrales à chaque itération. En effet, nous émettons l'hypothèse selon laquelle le signal contient des régions texturées ayant une grande cohérence spectrale. Donc, le spectre d'une région suffisamment grande contient une série de raies dominantes, malgré la perturbation introduite par le spectre de la fenêtre.

Décrivons le fonctionnement général de cet algorithme :

- la transformation utilisée doit pouvoir traiter chaque direction ; la transformée de Fourier s'est donc imposée, la DCT représentant mal les directions diagonales à cause de son inévitable effet de miroir. De plus, le chapitre 3 tend à démontrer le meilleur comportement de la DFT face aux sauts de phase entre blocs adjacents.
- au cœur de l'algorithme se trouve une matrice $S$, modifiée à chaque itération. Cette matrice permet de sélectionner les raies spectrales. La matrice $S_{i+1}(k)$ sélectionne une nouvelle fréquence non-sélectionnée dans $S_i(k)$ (la raie de plus grande intensité encore non choisie).

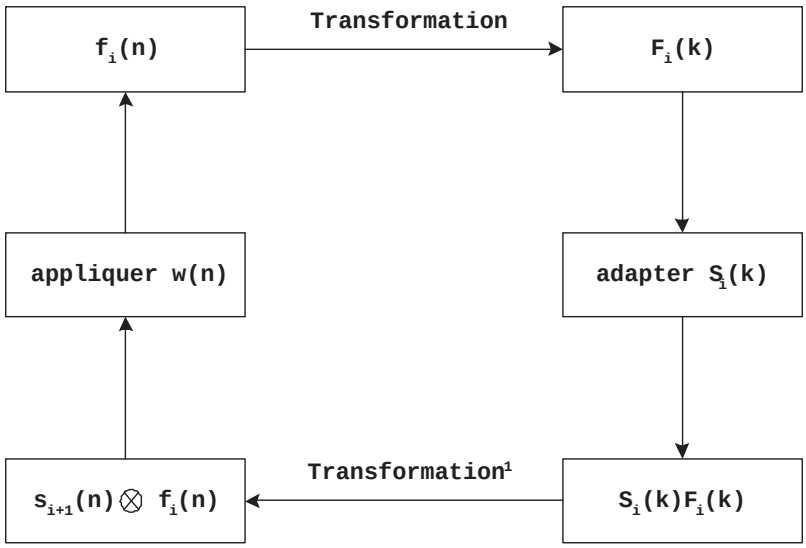



FIG. 5.1: Schéma de l'algorithme de déconvolution sélective. (Extrait de [5])

Principales propriétés :

- l'extrapolation est d'autant meilleure que le contenu de la fenêtre ressemble à une texture régulière.
- la ressemblance du signal extrapolé avec le contenu de la fenêtre est la plus forte en bordure de la fenêtre.
- plus la fenêtre de base est grande, meilleure est l'extrapolation.
- la pratique montre la convergence et l'unicité de la déconvolution sélective.

Les premières étapes de l'algorithme sont illustrées page 38 sur un motif simple.

### 5.1.3 Résultats attendus lors de l'application au codage

Dans le cas d'une application de codage, l'idée est simple. On utilise l'information d'un bloc pour prédire le contenu du ou des blocs voisins. Si la prédiction s'avère bonne, on décide de ne coder que la différence entre le bloc réel et sa prédiction, un peu comme en PCM différentiel (DPCM).

Soient deux blocs voisins $B_1$ et $B_2$. On connaît $B_1$ et on cherche à coder $B_2$. Sur base du contenu de $B_1$, on procède à l'extrapolation. Soit $E_2$, les valeurs obtenues par extrapolation sur base du premier bloc. Si l'écart entre $B_2$ et $E_2$ est faible, on code $B_2 - E_2$, sinon directement $B_2$. C'est cette idée qui est suggérée pour un codeur JPEG modifié. Il s'agit de tester les extrapolations de proche en proche par balayage en permettant une prédiction horizontale ou verticale. Une telle prédiction n'ayant pas été prévue dans le mode de codage JPEG, il en résultera un schéma de codeur qui ne peut être décrit avec la norme.

Il faudra donc également fournir au décodeur une sorte de carte indiquant si un bloc a été codé directement (codage intra-bloc) ou par prédiction (codage inter-bloc). Aussi, si l'on rassemble les blocs ayant été prédits entre eux, on obtient une sorte d'image de label qui conduit tout naturellement à une segmentation d'images. Notons que la carte de label peut se révéler utile en cours de codage car on peut assembler les blocs prédits entre eux pour construire une meilleure extrapolation d'un bloc courant. Cette carte de label n'introduit qu'un surcoût minime puisque dans la cadre d'un codeur JPEG modifié, seul 1 bit suffit pour préciser si un bloc $8 \times 8$ appartient à la même texture.

## 5.2 Exemples

Ci-dessous l'utilisation de l'algorithme de déconvolution sélective original sur l'image Lena. L'algorithme effectue la prédiction d'un bloc à partir du bloc supérieur et du bloc situé à sa gauche, compare les énergies du bloc initial aux énergies des différences entre l'original et chacune des extrapolations et conserve le bloc dont l'énergie est moindre. Nous avons choisi d'illustrer le résultat en ne gardant qu'un faible pourcentage des coefficients spectraux (3, 5 et 10) afin de montrer la qualité du résultat :

FIG. 5.2: Lena extrapolée avec 3% des coefficients

FIG. 5.3: Lena extrapolée avec 5% des coefficients

FIG. 5.4: Lena extrapolée avec 10% des coefficients

Ci-dessous se trouvent les images de label correspondant aux trois extrapolations.

Dans ces images (dont la dynamique est de 8 bits), une surface continue dont les valeurs de gris sont identiques, représente un ensemble de blocs prédits les uns par rapport aux autres :

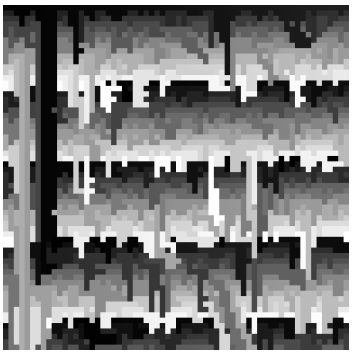


FIG. 5.5: image de label correspondant à la figure 5.2

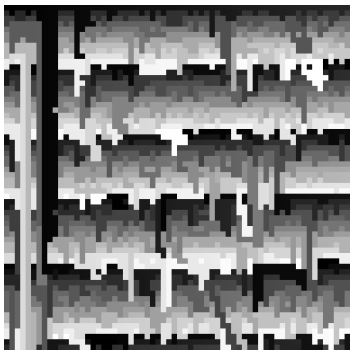


FIG. 5.6: image de label correspondant à la figure 5.3

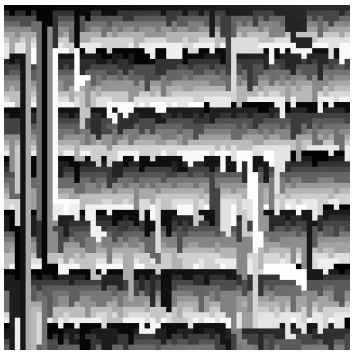
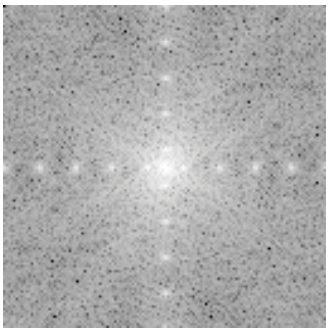


FIG. 5.7: image de label correspondant à la figure 5.4

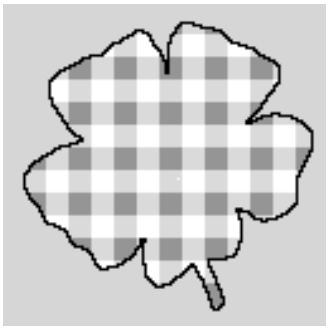
$f_0(n) = y(n)$



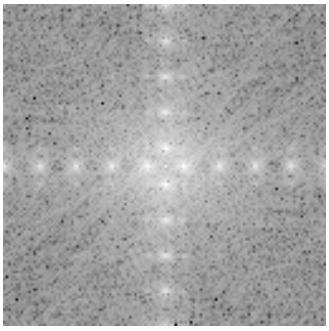
$\|F_0(k)\|$



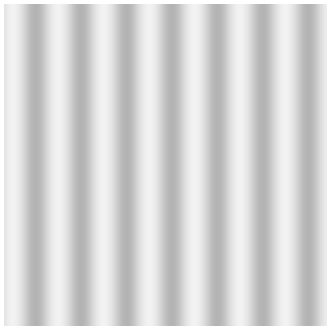
$s_1(n) \otimes f_0(n)$



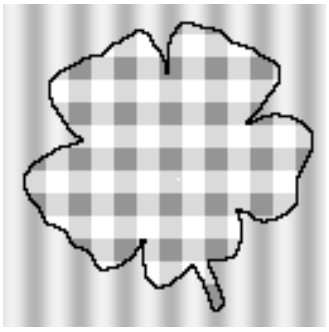
$f_1(n) = y(n)$



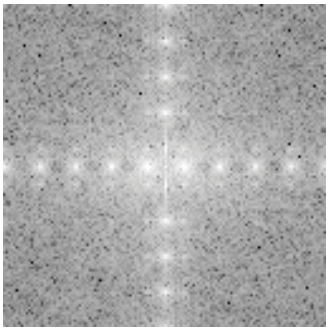
$\|F_1(k)\|$



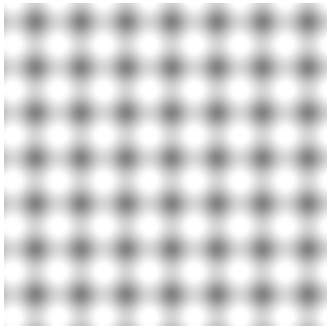
$s_2(n) \otimes f_1(n)$



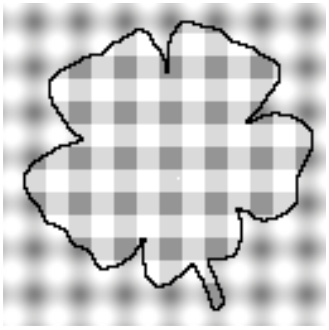
$f_2(n) = y(n)$



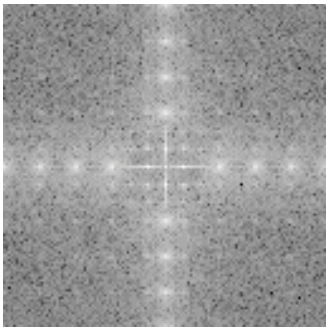
$\|F_2(k)\|$



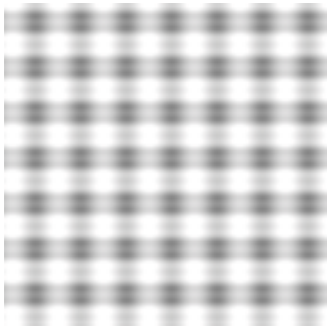
$s_3(n) \otimes f_2(n)$



$f_3(n) = y(n)$



$\|F_3(k)\|$



$s_4(n) \otimes f_3(n)$

FIG. 5.8: Premières étapes de l'algorithme de déconvolution sélective

# Chapitre 6

# Prédiction inter-bloc et segmentation

Dans le cadre général de la segmentation d'image, nous allons nous affranchir des contraintes de JPEG. Nous pouvons donc modifier la forme des fenêtres, leur taille, l'ordre des extrapolations, etc. L'idée est que si deux blocs sont très proches après extrapolation, on les fusionne. On dispose donc d'un grand nombre de degrés de liberté.

Un choix possible consiste à sectionner l'image en blocs de taille $n \times n$. On construit alors un graphe orienté dont les valeurs des arcs représentent la ressemblance entre blocs voisins après extrapolation. Par analyse du graphe, un algorithme détecte l'arc ayant la plus forte valeur de ressemblance. Il fusionne ensuite les noeuds situés aux extrémités de l'arc détecté et il recalcule les valeurs des arcs voisins. Cette étape est répétée jusqu'à la réalisation d'un critère d'arrêt qui peut être l'absence de ressemblance positive.

Bien que cette technique semble être limitée aux images contenant des textures, nous espérons qu'elle puisse être appliquée à tout type d'image.

## 6.1 Détermination d'un schéma de segmentation

L'algorithme prend en entrée une image déjà segmentée. Nous pouvons donc prendre comme point de départ :

- soit une image de label obtenue depuis l'algorithme précédent (de type bloc $8 \times 8$).
- soit une image segmentée par une autre technique : par exemple, l'image obtenue par application de l'algorithme de ligne de partage des eaux du gradient seuillé de l'image originale.

Cette étape n'est pas à négliger puisque c'est à partir de cette image segmentée que l'algorithme va fusionner les régions.

Cette partition initiale est utilisée pour construire un graphe d'adjacence dans lequel les noeuds représentent chacune des régions et les arcs représentent un lien

de connexité entre deux régions. Ce graphe est bidirectionnel et valué, c'est-à-dire qu'entre deux régions connexes $A$ et $B$, on a une valeur pour $A \rightarrow B$ et une valeur pour $B \rightarrow A$. Ces valeurs représentent un facteur de fusion obtenu après extrapolation des régions.

## 6.2 Critère de fusion des régions

Le principe est que si la différence d'énergie entre le bloc prédit et le bloc original est inférieure à l'énergie de ce bloc original, la région considérée et celle à partir de laquelle l'extrapolation a eu lieu doivent être fusionnées.

Formalisons ceci : soient $D_{w1}$ et $D_{w2}$ deux régions voisines et $f_1(n)$ la valeur (connue) du signal sur $D_{w1}$. À partir de $f_1(n)$, nous extrapolons la texture en $D_{w2}$ : $g(n)$ (en fait l'extrapolation est effectuée à partir de $f_1(n) - \mu_{w1}$, où $\mu_{w1}$ est la moyenne statistique de $f_1(n)$ sur $D_{w1}$, et ce afin de diminuer l'influence du coefficients DC). Après l'extrapolation, le signal $g(n)$ est comparé au signal original $f_2(n)$. Nous comparons la variance du bloc original à la variance résiduelle après extrapolation, ce qui revient à calculer un *gain d'extrapolation* (EG) :

$$EG = \sigma^2_{w2}[f_1(n) - \mu_{w1}] - \sigma^2_{w2}[f_1(n) - \mu_{w1} - g(n)] \tag{6.1}$$

Si $EG$ est positif, l'extrapolation a conduit à une "bonne" prédiction. De plus, par le théorème de Parseval, nous savons qu'un gain équivalent a été obtenu dans le domaine spectral. La pratique [7] a malheureusement montré que l'utilisation de ce seul critère ne produit pas de bons résultats de segmentation. C'est pourquoi la différence des moyennes statistiques des deux régions $\mu_{diff} = \mu_{w2} - \mu_{w1}$ a été combinée avec $EG$ pour obtenir un *facteur de fusion* $MF$.

La fusion de deux régions ne dépend plus seulement de $EG$, mais obéit à plusieurs autres règles :

- si $EG > 0$ et que les moyennes sont proches, alors il faut fusionner. C'est le cas le plus favorable : $MF \gg$.
- de petites régions ayant une variance faible mais des valeurs moyennes proches doivent être fusionnées, même si $EG$ est négatif.
- deux régions pour lesquelles $EG$ est positif ne doivent pas être fusionnées si la différence de leurs moyennes est trop importante.

## 6.3 Résumé du fonctionnement de l'algorithme : ordre de fusion des régions

1. L'image doit être préalablement segmentée par un algorithme tel que la ligne de partage des eaux (Watershed).

2. Il faut également construire un graphe $G(V, N)$ dont les noeuds $N$ correspondent aux régions et les arcs $V$ relient deux noeuds si deux régions sont adjacentes. Ces arcs sont étiquetés par une valeur de fusion $MF$ obtenue depuis l'extrapolation d'une région $R_i$ sur une région $R_j$.
3. L'algorithme évalue ensuite tous les arcs du graphe $G$, il faut évaluer les deux arcs $R_i \rightarrow R_j$ et $R_j \rightarrow R_i$, puisque les extrapolations réciproques n'ont pas forcément la même valeur.
4. L'algorithme sélectionne l'arc ayant le plus grand $MF$ et fusionne les régions correspondantes.
5. L'algorithme met à jour les étiquettes des arcs et les régions connectées aux deux régions avant fusion.
6. L'algorithme recommence les étapes 4 et 5 tant que le plus grand $MF$ est positif.

# Deuxième partie

# Codage

# Chapitre 7

# Codage de la carte de segmentation

Afin d'illustrer l'application au codage de la déconvolution sélective, nous allons construire un codeur JPEG modifié. Celui-ci ne sera bien évidemment pas compatible avec la norme, mais nous pourrons comparer les débits de notre encodeur et ceux de l'original.
Quelles données devons-nous coder ? Bien sûr, l'image prédite elle-même, mais aussi l'image de label.

## 7.1 Données de la carte de segmentation

Les images de label telles que présentées à la section 5.2 permettent d'observer quand l'algorithme a décidé que le critère de ressemblance entre un bloc et sa prédiction est vérifié ou non, mais elle ne représente pas cette information de la manière la plus compacte qui soit ! Théoriquement, un seul bit suffit pour préciser si un bloc est codé en intra ou non. Cependant l'existence de 3 possibilités de codage (intra, inter gauche et inter supérieur), ainsi que le coût marginal du codage de la carte nous ont fait abandonner cette solution. Nous avons donc choisi de coder l'information à l'aide d'un octet par bloc $8 \times 8$. Ceci représente déjà un gain de 64 par rapport à l'image de label !

Nous avons travaillé de la manière suivante : nous ouvrons un fichier dans lequel nous écrivons 1 caractère par bloc :

| | |
|---|---|
| 0 | si le bloc est codé en intra |
| 1 | si le bloc est prédit horizontalement |
| 2 | si le bloc est prédit verticalement |

Pour une image de dimensions $n \times m$, nous obtenons donc un fichier dont la taille est égale à $\frac{n \times m}{64}$ octets.

## 7.2 Codage Arithmétique de la carte

Notre but est de minimiser le débit de cette carte. Pour ce faire, nous allons compresser ce fichier à l'aide d'un codeur arithmétique.

### 7.2.1 Rappels sur le codage arithmétique

Le codage arithmétique utilise un modèle statistique, tout comme le codeur de Huffman. Contrairement à ce dernier, il produit un code pour une séquence de symboles toute entière, et non pas un code par symbole. Chaque nouveau symbole lu modifie de façon incrémentale le code de sortie. Ce code de sortie est un nombre à virgule flottante compris entre 0 et 1, dont le nombre de chiffres après la virgule correspond au nombre de symboles. Contrairement à Huffman, il n'est pas obligatoire que chaque code ait un nombre entier de bits. Par exemple, un symbole de probabilité 0.9 a pour entropie 0.15, mais Huffman affectera probablement un code de un bit (ou plus), et la séquence codée aura un nombre de bits plus long qu'en théorie. Le codeur arithmétique est plus performant que le codeur de Huffman, mais il est plus complexe à implémenter.

L'algorithme considère chaque message à coder comme étant une feuille d'un arbre qui reste à construire. L'idée est d'attribuer aux deux messages de plus faibles probabilités, les mots codés les plus longs. Ces deux mots codés ne se différencient que par leur dernier bit, contrairement au codage de Shannon-Fano [4] qui part des feuilles de l'arbre et, par fusions successives, remonte vers la racine.

Le principe est le suivant :

1. Calculer la probabilité associée à chaque symbole dans la chaîne à coder.
2. Associer à chaque symbole un sous-intervalle dont la taille est proportionnelle à sa probabilité, dans l'intervalle [0,1] (l'ordre de rangement des intervalles sera mémorisé car il est nécessaire au décodeur).
3. Initialiser la limite inférieure de l'intervalle de travail à la valeur 0 et la limite supérieure à la valeur 1.
4. Tant qu'il reste un symbole dans la chaîne à coder :
   - largeur = limite supérieure - limite inférieure
   - limite inférieure = limite inférieure + largeur×(limite basse du sous-intervalle du symbole)
   - limite supérieure = limite inférieure + largeur ×(limite haute du sous-intervalle du symbole)
5. La limite inférieure code la chaîne de manière unique.

Les méthodes de compression de données statistiques peuvent être représentées comme étant composées de deux modules : l'algorithme de codage et le modèle statistique. Le modèle statistique passe à l'algorithme de codage de l'information conçernant la

nature statistique de la source à coder. L'algorithme de codage utilise cette information pour coder la source efficacement.

Le codeur arithmétique que nous avons utilisé [2] est un codeur d'ordre $N$ ; ceci signifie que le modèle statistique tient compte des dépendances entre symboles successifs.

### 7.2.2 Résultats obtenus

Voici quelques exemples de débits obtenus pour des images de taille 256, ainsi que le rapport du débit original sur le débit de la carte codée :

| Image de départ | Taille | Carte Originale | Carte Codée | Gain |
|---|---|---|---|---|
| Lena | $256 \times 256$ | 1024 Bytes | 151 Bytes | 6.7 |
| chat Pigeon | $256 \times 256$ | 1024 Bytes | 200 Bytes | 5.1 |
| dick | $256 \times 256$ | 1024 Bytes | 156 Bytes | 6.6 |
| Kairouan | $256 \times 256$ | 1024 Bytes | 139 Bytes | 7.3 |

Le gain moyen est de 6. Ces résultats restent identiques quel que soit le nombre de coefficients conservés lors de l'extrapolation.

# Chapitre 8

# Codage de l'image

Maintenant que la carte de label a été codée efficacement, nous pouvons nous attacher au codage de l'image elle-même. Notre but final sera de comparer les débits d'une image codée par un encodeur JPEG classique et le débit final de notre codeur (image + carte).

## 8.1 Fonctionnement général du codeur

Nous allons décrire ici le fonctionnement de l'algorithme réalisant le codage à l'aide de la déconvolution sélective. En voici les grandes étapes :

1. Le premier bloc de l'image est codé tel quel (intra) puisqu'il n'est pas prédit. On lui applique donc le processus JPEG.
2. Pour chaque bloc $B_k$ de l'image, nous en effectuons la prédiction à partir du ou des blocs précédents décodés (le nombre de blocs à partir desquels se fait la prédiction est dépendant des précédentes extrapolations). Nous décodons les blocs précédents (nous simulons le comportement du décodeur) et nous effectuons l'extrapolation du bloc courant, soit $P_k$ cette extrapolation.
3. Nous codons ce $k^{ème}$ bloc :
   - Si le critère déterminant la conservation de la prédiction est vérifié, ce qui signifie que la prédiction est proche du bloc considéré, nous appliquons JPEG à la différence entre le bloc original et sa prédiction : $B_K - P_k$.
   - Sinon, on ignore la prédiction, et on code le bloc $B_k$ en intra.

## 8.2 Codage d'image par JPEG

La norme JPEG prévoit la définition d'un facteur, appelé *facteur de qualité* et noté $Q$, pour permettre à l'utilisateur de choisir entre la qualité de l'image codée et la taille[1] du fichier. Pratiquement, on sait que le facteur de qualité affecte la matrice

[1]Par abus de langage, nous parlerons de *débit* pour citer la taille du fichier.

de quantification ; plus le facteur de qualité est faible, plus la quantification dégrade la valeur des coefficients DCT et plus la qualité diminue. Cet effet est illustré à la figure 8.1.

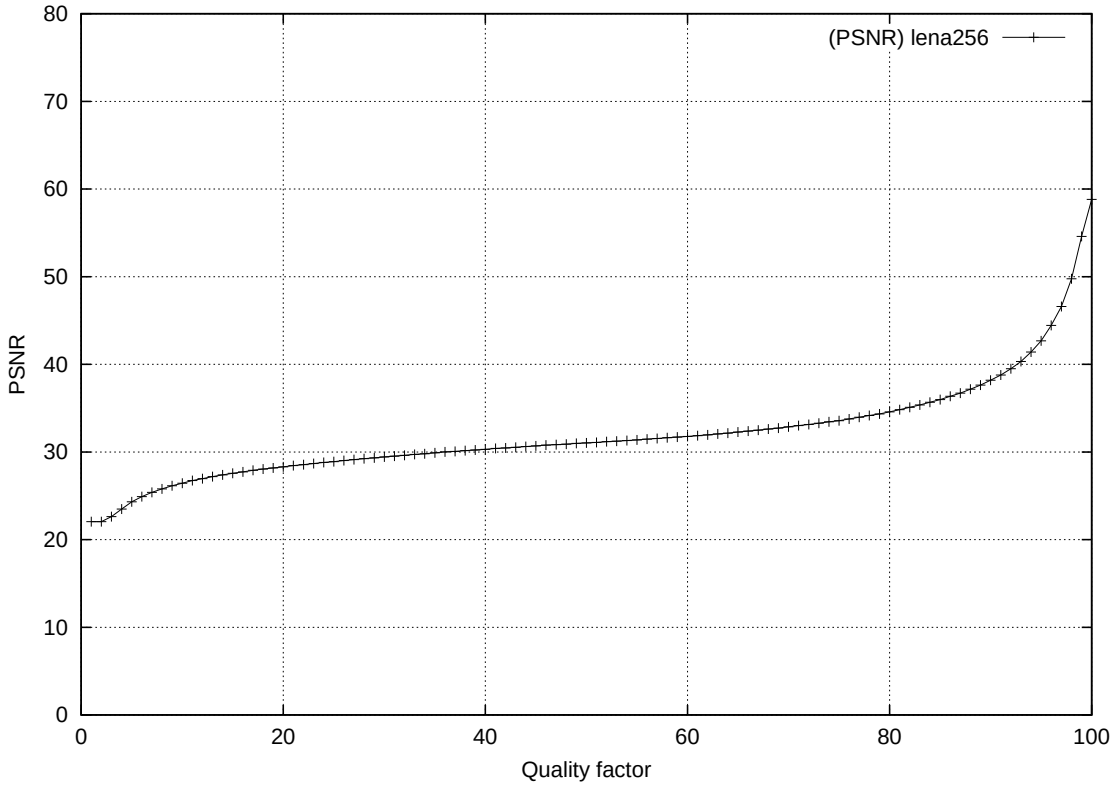



FIG. 8.1: Courbe de correspondance entre le facteur de qualité d'un encodeur JPEG et la qualité de l'image (exprimée par le PSNR).

Ce n'est pas tant le facteur de qualité qui nous intéresse mais le débit d'une image. La correspondance entre la qualité de l'image et le débit associé est représentée à la figure 8.2. Par comparaison avec la courbe précédente, on peut affirmer que le facteur de qualité est en relation directe avec la taille du fichier JPEG.

Notons au passage que le facteur de compression est typiquement de l'ordre de 6 à 10, ce qui, au départ d'une image originale de 65.536 bytes, comme lena256, correspond à l'intervalle approximatif de [6000, 10.000] bytes. C'est donc dans cette zone qu'il convient de comparer principalement l'efficience de notre encodeur par rapport à JPEG.

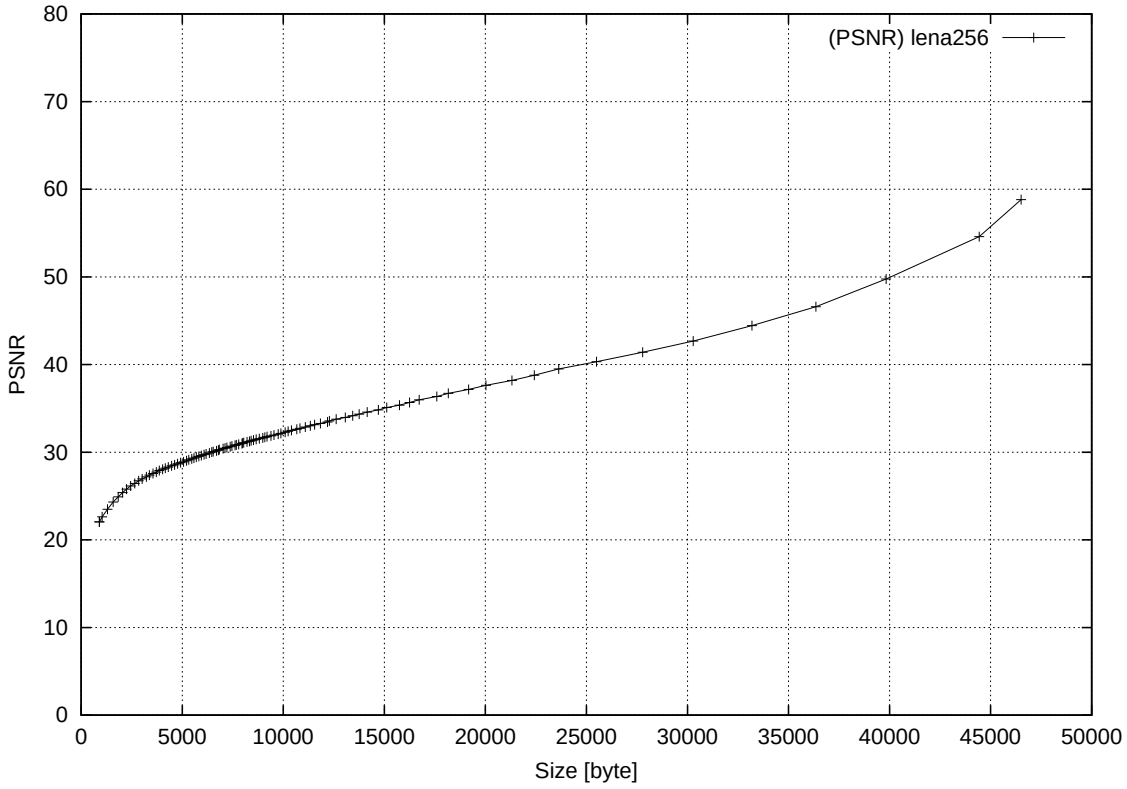



FIG. 8.2: Courbe de correspondance entre la taille d'un fichier JPEG et la qualité de l'image (exprimée par le PSNR).

## 8.3 Présentation et interprétation de résultats de codage par prédiction

Tel que présenté précédemment (équation 6.1), le critère de fusion ne fait pas l'affaire pour du codage. En effet, un surcoût non négligeable est associé au codage des blocs par prédiction. Dès lors, on ne peut se satisfaire de comparer l'énergie d'un bloc à l'énergie résiduelle d'un bloc prédit ; le critère de fusion doit être amendé.

Pour le codage, nous avons apporté les modifications suivantes :

1. possibilité de codage des blocs intra et inter avec des facteurs de qualité différents. Nous appellerons *Qintra* et *Qpred* les facteurs de qualité respectifs.
2. modification du facteur de fusion.

Nous avons remplacé le critère antérieur qui comparait l'écart-type du bloc (measure) avec l'écart-type de la différence entre la prédiction et le bloc (error) de manière à mieux considérer l'écart relatif et l'écart absolu. Auparavant, nous utilisions la différence measure-error. Dans le cas de blocs à très faible variance, le critère proposait parfois la prédiction pour un gain dérisoire. Nous avons donc ajouté un facteur multiplicatif $\alpha < 1$, pour l'énergie du signal. D'autre part, il est souhaitable d'avoir une mesure absolue du gain minimum pour favoriser la fusion, ce qui nous a conduit à considérer le critère de fusion (prédiction pour codage) suivant :

$$(\alpha * measure - error) - \beta \tag{8.1}$$

Il en résulte que la prédiction est moins souvent préférée qu'avec le critère précédent. C'est favorable pour le codage car il y a un surcoût au codage prédictif (à la fois en terme de débit et de perte de qualité PSNR car on divise par 2 la dynamique du signal d'erreur, ce qui introduit des erreurs irréversibles). Idéalement, il faudrait un modèle théorique pour fournir les valeurs des paramètres $\alpha$ et $\beta$. À défaut d'un tel modèle, nous avons choisi $\alpha = 0,95$ et $\beta = 2,0$ par une approche "essais-erreurs".

Pour juger de l'efficacité de codage par notre méthode, nous avons testé notre codeur avec toutes les combinaisons de facteur de qualité entre 10 et 100, par pas de 10, (ce qui fait 100 combinaisons possibles par image !) pour les couples (*Qintra*, *Qpred*) et avec un facteur de sélection de raies spectrales pour la déconvolution sélective de 10%. L'encodage des différentes informations a été effectué de la manière suivante :

1. encodage des blocs en intra au moyen d'un encodeur JPEG avec le facteur de qualité *Qintra*.
2. encodage de la différence des blocs prédits au moyen d'un encodeur JPEG avec le facteur de qualité *Qinter*.

De ces 3 parties, la plus délicate est l'encodage des signaux différences. En effet, la dynamique de ces signaux n'est pas limitée à 8 bits ; elle s'étend sur une plage légèrement supérieure à 9 bits si on tient compte des erreurs de quantification. De plus

ces signaux ont une moyenne proche de 0. Dans le cadre de ce travail, il aurait été impossible de réimplémenter un nouveau type d'encodeur de type JPEG convenant à ce type de signaux. Comme moindre mal, nous avons décidé de ramener la dynamique dans l'intervalle [0, 255] en divisant la dynamique par 2 et en décalant les valeurs de 127. Il est entendu que le décodeur a connaissance de cette opération ; il est donc en mesure de retirer 127 et de multiplier les valeurs reçues par 2 mais il faudra compter avec une erreur résiduelle supplémentaire de quantification.

Les résultats de ces tests systématiques, qui ont été effectués sur 4 images de taille $256 \times 256$, sont présentés à la figure 8.3 sous la forme de graphiques montrant la qualité de l'image en fonction de la taille du fichier JPEG.

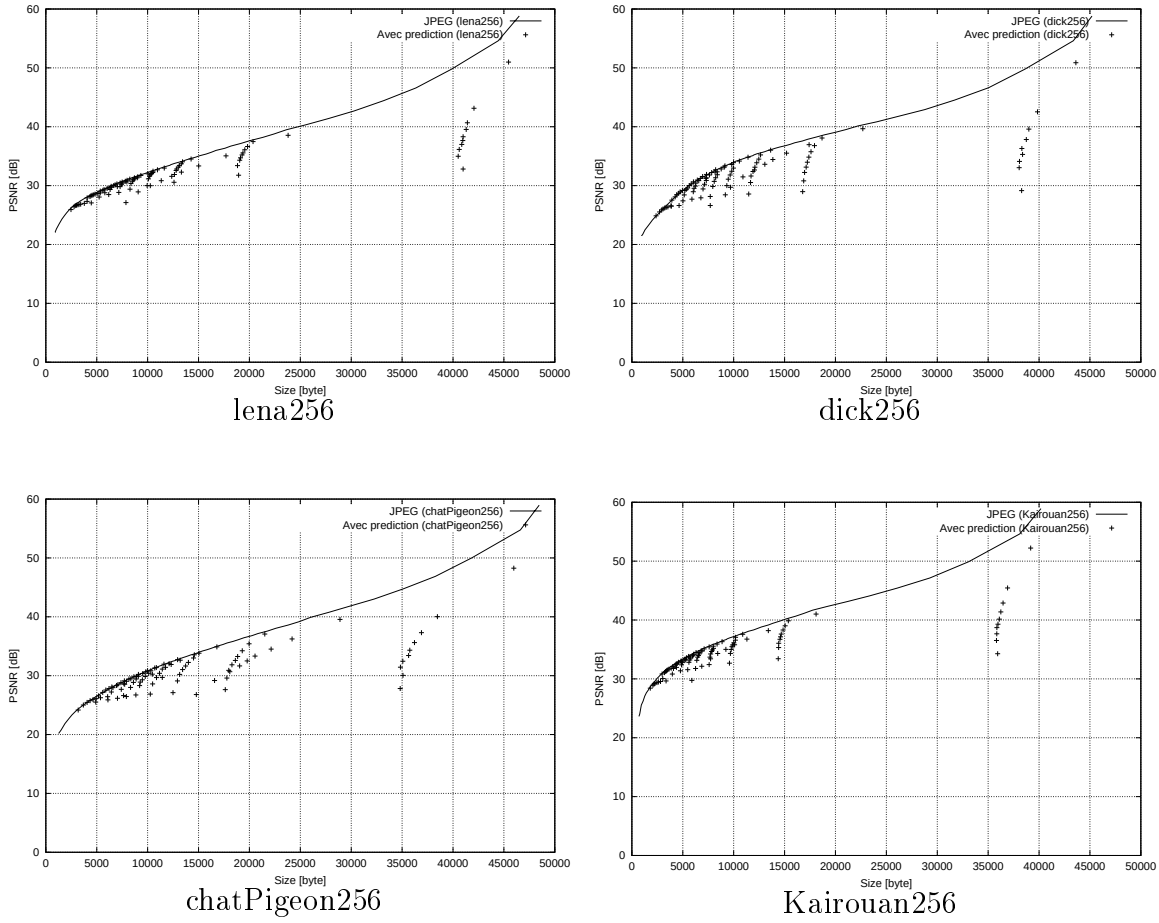



FIG. 8.3: Correspondances entre la taille d'un fichier JPEG et la qualité de l'image pour l'encodeur JPEG (courbe continue) et notre encodeur (symboles +).

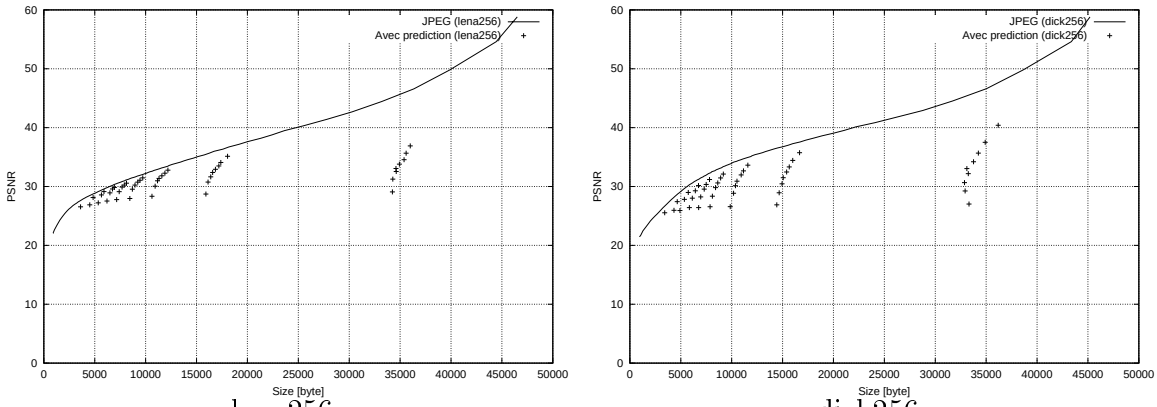
*A priori*, il est difficile de savoir comment se comporte l'encodeur pour les différentes combinaisons de facteur de qualité. Si $Qintra > Qpred$, on met l'accent sur un codage correct des blocs intra, en considérant que l'énergie résiduelle est faible et qu'elle peut donc être codée avec un pas de quantification plus grand. Ce raisonnement est réaliste pour un encodeur classique mais il est en général moins souhaitable

pour un encodeur différentiel car un bloc prédit peut lui-même servir à prédire un bloc voisin. Il est alors pénalisant d'avoir des blocs prédits de piètre qualité sous peine de diminuer le nombre de blocs prédits, ce qui entraînerait un accroissement du débit total.

Il faut avoir à l'esprit qu'un gain est réalisé par rapport au codeur JPEG dans la mesure où le nombre de blocs prédits correctement est important. De fait, dans ce cas, l'énergie résiduelle est faible et on peut espérer une réduction du coût du codage. Concernant le choix des facteurs de qualité respectifs, deux scénarios sont possibles :

1. on choisit $Qintra > Qpred$, l'idée étant qu'avec notre critère de sélection de la prédiction, l'énergie résiduelle est faible et que l'on peut se permettre de coder cette dernière avec moins de précision.
2. on choisit $Qintra < Qpred$. Dans ce cas, on fait en sorte de garantir qu'un bloc prédit est codé avec plus de précision qu'un bloc intra, de manière à permettre une prédiction, de proche en proche, de meilleure qualité.

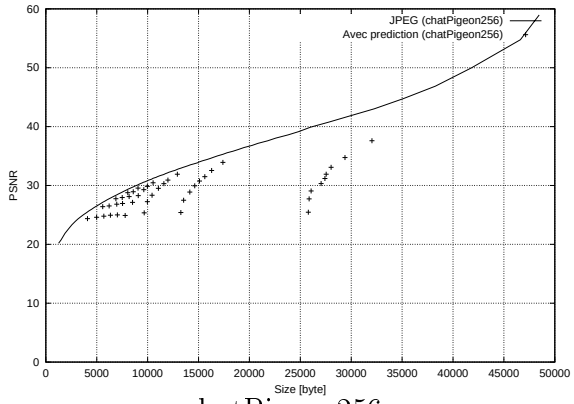
Pour vérifier si l'un de ces scénarios est plus favorable pour notre encodeur, nous avons sélectionné, parmi toutes les combinaisons possibles de ($Qintra$, $Qpred$), celles pour lesquelles $Qintra > Qpred$ au départ des courbes de la figure 8.3. Le résultat de cette sélection est affiché à la figure 8.4.



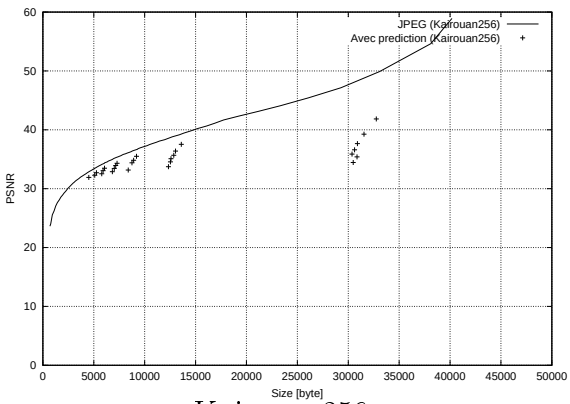

lena256 dick256

Comme le montre cette figure, la qualité pour ce type de combinaisons de facteur de qualité est souvent inférieure à des combinaisons pour lesquelles $Qintra \leq Qinter$. Néanmoins, si notre encodeur parvient à égaler les performances d'un encodeur JPEG, il ne parvient pas à les surpasser souvent alors qu'en raison de la prédiction, la somme des énergies des blocs a diminué, ce qui par application du principe de Parseval signifie que l'énergie des coefficients a diminué ! Les quelques cas ayant mené à un gain sont représentés à la figure 8.5 (ces courbes constituent une partie agrandie des graphes de la figure 8.3). Les gains ont été observés sur les images dick256 et chatPigeon256 qui contiennent des textures. Aucun gain n'a été relevé sur les images ne contenant aucune texture régulière.

En fait, même si nous prenons un facteur de qualité légèrement supérieur pour coder les blocs prédits, le PSNR est inférieur à celui que l'on aurait avec un facteur de qualité identique pour tous les blocs ! L'explication la plus plausible est celle d'un mauvais codage de l'erreur résiduelle qui affecte les performances de l'encodeur. Autrement dit, malgré la réduction de l'énergie à coder, l'encodeur ne parvient pas à

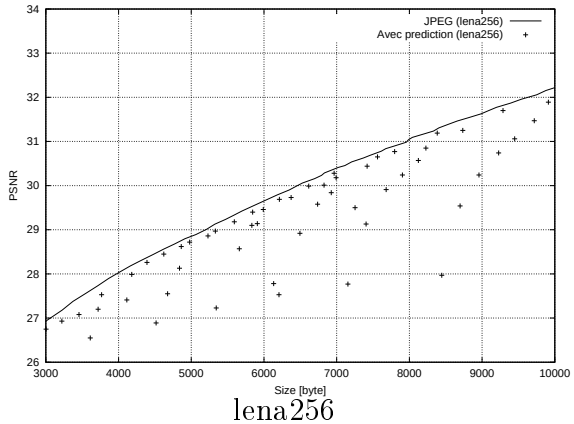



chatPigeon256

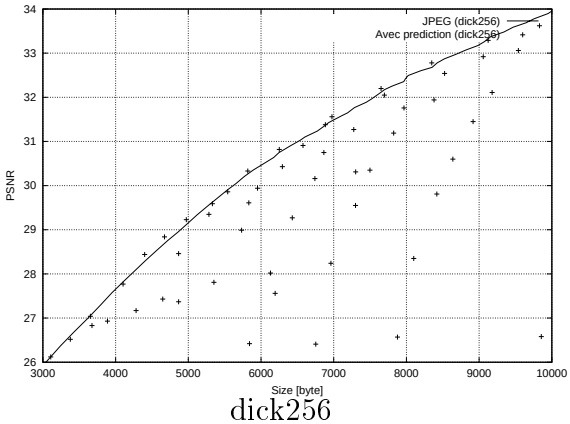



Kairouan256

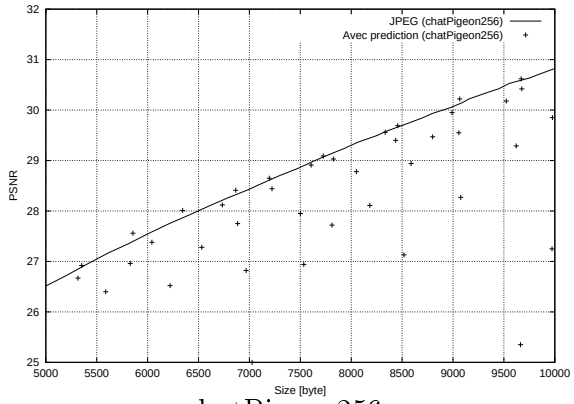
FIG. 8.4: Sélection des résultats présentés à la figure 8.3, montrant les correspondances entre la taille d'un fichier JPEG et la qualité de l'image pour l'encodeur JPEG (courbe continue) et notre encodeur (symboles +).



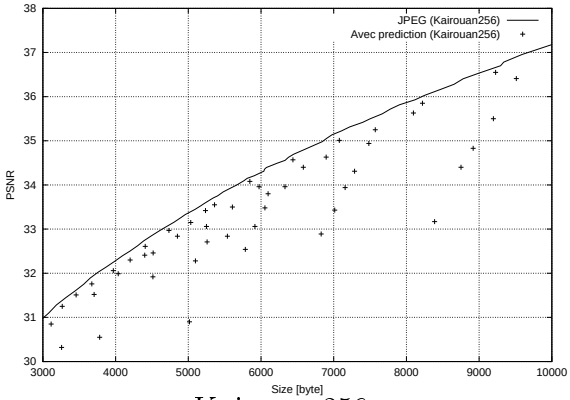

lena256




dick256




chatPigeon256




Kairouan256

compenser la perte de qualité par une réduction significative du débit. Il faut signaler que notre façon d'encoder les informations (c'est-à-dire par le recours au codage JPEG de 2 images) conduit à encoder 2 tables de quantification et dédoubler toute

FIG. 8.5: Zoom de la figure 8.3 montrant la correspondance entre la taille d'un fichier JPEG et la qualité de l'image pour l'encodeur JPEG (courbe continue) et notre encodeur (croix).

une série d'informations. Ce surcoût est négligeable à haut débit mais il ne l'est plus à bas débit. Avant de tirer une conclusion définitive sur la qualité de notre encodeur, il s'agirait donc de concevoir un encodeur adapté au codage de nos signaux de différence pour maintenir la qualité de l'image à un niveau plus élevé, tout en réduisant le débit.

# Troisième partie

# Segmentation

# Chapitre 9

# Segmentation

Dans ce chapitre, nous allons faire un bref rappel sur la segmentation préalable à notre technique avant de présenter les résultats : l'image originale sur-segmentée, ainsi que l'image segmentée par croissance de régions.

## 9.1 Introduction

Bien que le principe de l'algorithme ait été présenté au chapitre 6, nous allons rappeler ici pourquoi l'image résultante d'une segmentation basée sur l'algorithme de ligne de partage des eaux est sur-segmentée.

Tout d'abord, un gradient est appliqué à l'image. Le gradient morphologique de Beucher ($GMB$) est un opérateur non-linéaire défini à partir du gradient d'érosion ($GE$) et du gradient de dilatation ($GD$), eux-mêmes définis à partir de l'érosion et de la dilatation morphologique (respectivement notées $\ominus$ et $\oplus$) :

$$\begin{aligned} GE(f) &= f - (f \ominus B) \\ GD(f) &= (f \oplus B) - f \\ GMB(f) &= GE(f) + GD(f) = (f \oplus B) - (f \ominus B) \end{aligned}$$

Le résultat produit (figure 9.4) est une image où subsistent uniquement les principaux contours. C'est à cette image que l'on fait subir l'algorithme de partage des eaux.

Rappelons les grandes lignes de cet algorithme :

- Le gradient appliqué à l'image originale produit une nouvelle image $f$ pouvant être considérée comme une surface topologique.
- Les minima locaux de cette surface sont des *bassins versants* (figure 9.1).
- Nous pouvons donner à l'algorithme l'interprétation suivante : imaginons que chaque bassin soit "percé", et que l'on inonde la surface par le bas. Chaque bassin va se remplir progressivement, et lorsque deux bassins adjacents se rejoignent, on érige un *barrage* afin d'éviter qu'ils ne fusionnent (figure 9.2).
- L'algorithme s'arrête lorsqu'il ne reste plus de barrages à ériger. Seuls les barrages subsistent et ceux-ci constituent la segmentation de l'image.

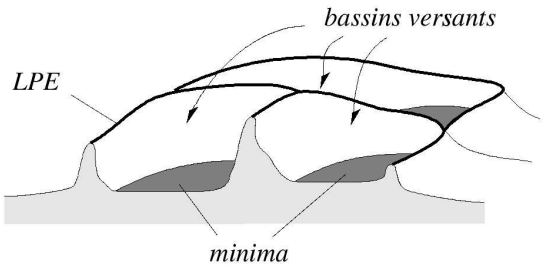



FIG. 9.1: Minima et bassins de la ligne de partage des eaux

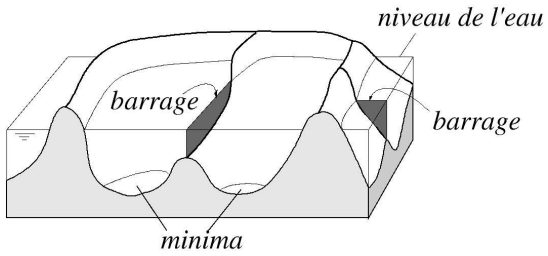



FIG. 9.2: Construction du barrage

Le problème de ce schéma est que le gradient est très sensible au bruit. Cela a pour conséquence de produire des minima et donc des bassins supplémentaires. L'image résultante sera donc trop segmentée. Il existe bien différentes techniques afin de supprimer cet effet ; malheureusement elles nécessitent toutes une intervention humaine (marquage) ou requièrent certaines caractéristiques de l'image (filtrage préalable). L'algorithme de fusion des régions sur base d'un critère spectral constitue donc une alternative à ces différentes techniques.

FIG. 9.3: L'image originale

FIG. 9.4: Le gradient de Beucher (vidéo inverse)

## 9.2 Résultats

Les résultats présentés ci-après proviennent d'images de taille $256 \times 256$. Par commodité, nous avons surimpressionné la carte de label. L'extrapolation des différentes régions a été effectuée en ne gardant que 3% des coefficients spectraux.

Le tableau situé page 61 reprend, pour chaque image, le nombre de régions de départ ainsi que le nombre de régions après fusion par extrapolation.

FIG. 9.5: Lena segmentée par la ligne de partage des eaux

FIG. 9.6: Lena après fusion des régions par extrapolation

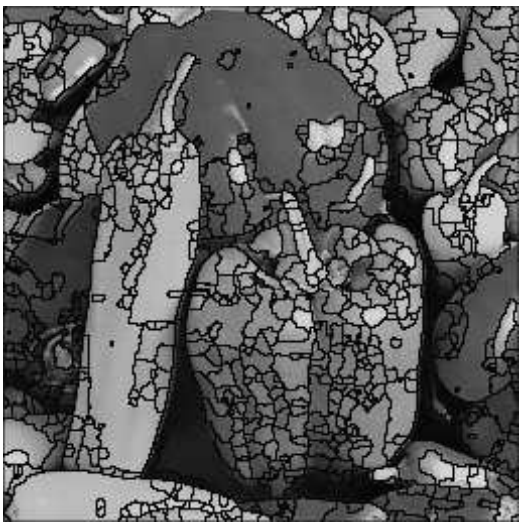


FIG. 9.7: Peppers segmentée par la ligne de partage des eaux

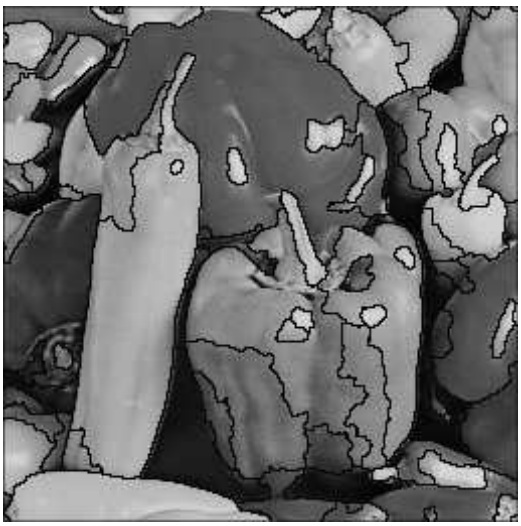


FIG. 9.8: Peppers après fusion des régions par extrapolation

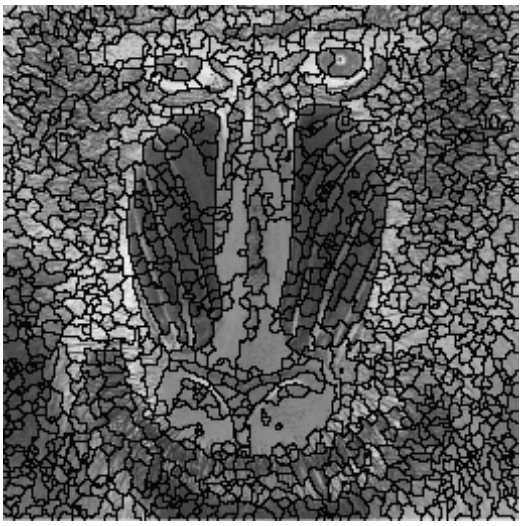


FIG. 9.9: Mandril segmentée par la ligne de partage des eaux

FIG. 9.10: Mandril après fusion des régions par extrapolation

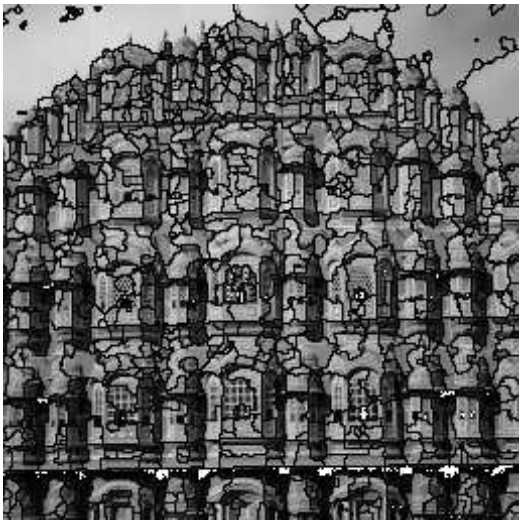


FIG. 9.11: Palais segmentée par la ligne de partage des eaux

FIG. 9.12: Palais après fusion des régions par extrapolation

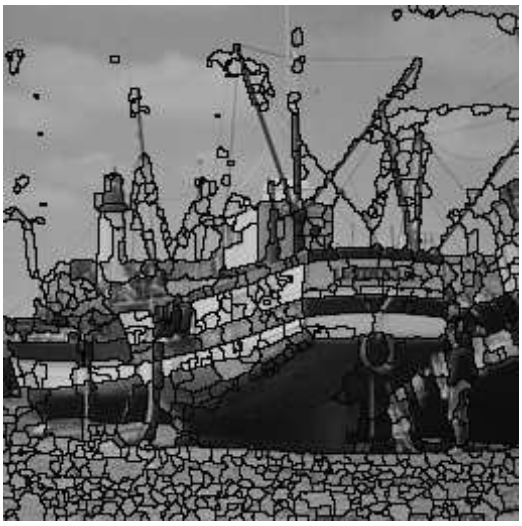


FIG. 9.13: Bateau segmentée par la ligne de partage des eaux

FIG. 9.14: Bateau après fusion des régions par extrapolation

FIG. 9.15: San-Francisco segmentée par la ligne de partage des eaux

FIG. 9.16: San-Francisco après fusion des régions par extrapolation

FIG. 9.17: Chat Pigeon segmentée par la ligne de partage des eaux

FIG. 9.18: Chat Pigeon après fusion des régions par extrapolation

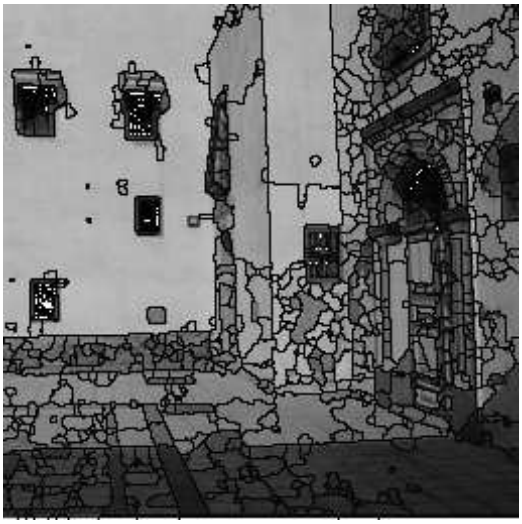


FIG. 9.19: Kairouan segmentée par la ligne de partage des eaux

FIG. 9.20: Kairouan après fusion des régions par extrapolation

| Image | Nombre de régions initial | Nombre de régions final |
|---|---|---|
| Lena | 3377 | 174 |
| Peppers | 2382 | 183 |
| Mandril | 3692 | 177 |
| Palais | 2356 | 811 |
| Bateau | 2026 | 166 |
| San-Francisco | 2187 | 144 |
| Chat Pigeon | 3649 | 470 |
| Kairouan | 2125 | 114 |

Nous allons maintenant observer l'impact du nombre de coefficients spectraux conservés lors de l'extrapolation sur la qualité du résultat ainsi que sur le nombre final de régions :

FIG. 9.21: 3% de coefficients

FIG. 9.22: 5% de coefficients

| Coefficients conservés | Nombre de régions initial | Nombre de régions final |
|---|---|---|
| 3 % | 3649 | 470 |
| 5 % | 3649 | 468 |
| 10 % | 3649 | 559 |
| 20 % | 3649 | 551 |

On peut donc conclure à l'absence d'influence du nombre de coefficients gardés sur le résultat. Cela est dû à la limitation à 30 coefficients lors de l'extrapolation : le nombre de coefficients conservés est fonction de la surface à partir de laquelle l'extrapolation est effetuée. Si cette surface est importante, le contenu spectral de la texture sera moins modulé par le spectre de la fenêtre ; dès lors, il serait inutile d'effectuer

FIG. 9.23: 10% de coefficients

FIG. 9.24: 20% de coefficients

l'extrapolation avec un grand nombre de coefficients.

La figure 9.25 montre l'évolution du gain pendant le processus de fusion des régions :

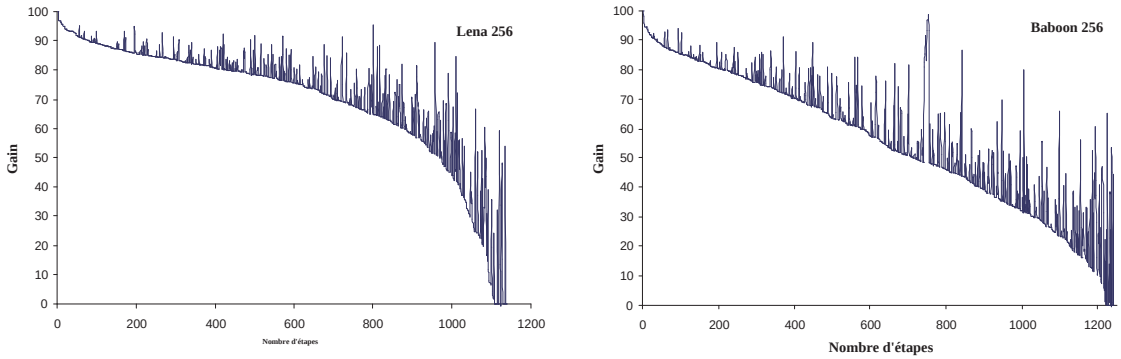



FIG. 9.25: Evolution du gain de fusion en fonction du nombre d'étapes.

On peut constater que le gain maximum évolue considérablement lors de la fusion. Malheureusement cette évolution n'est pas une fonction monotone décroissante. En fait, pour accélérer le processus de segmentation, il faudrait avoir la courbe la plus lisse possible. De cette manière, nous pourrions nous passer de recalculer systématiquement les gains de fusion pour les régions voisines à la région fusionnée. Or, l'évolution du gain de fusion montrant que le gain fluctue considérablement, nous avons la confirmation qu'il est impératif de recalculer régulièrement les gains de fusion. A moins de changer de critère (pour arriver à lisser la courbe), les corrections s'avèrent obligatoires.

# Quatrième partie

# Conclusions

L'objectif de ce mémoire est de tester l'emploi de l'algorithme de déconvolution sélective dans le cadre de deux applications : le codage d'images afin d'en réduire le débit et la segmentation.

Dans la cadre de l'application de l'algorithme au codage d'images, l'idée originale est de prédire un bloc de l'image à l'aide des blocs précédents. Si le contenu énergétique de la différence entre cette prédiction et le bloc original est inférieure à l'énergie de ce bloc, la représentation de ce dernier nécessite moins de bits.

L'inclusion de ce système de prédiction dans la norme JPEG nous a conduit à un codeur modifié. Notre objectif a alors essentiellement été de comparer les débits du codeur original et du codeur modifié à qualité d'image équivalente. Certaines contraintes imposées par la norme JPEG ne nous ont malheureusement pas permis de satisfaire à la fois les exigences de débit minimal et de conservation de la qualité de l'image. Il nous a fallu continuellement trouver un compromis, car la diminution du débit entraînait souvent une dégradation de l'image. D'un point de vue théorique, pour deux blocs successifs appartenant à la même texture, l'algorithme doit pouvoir satisfaire les deux exigences. Paradoxalement, le codage des blocs dont la prédiction était correcte, conduisait parfois à un débit plus élevé. Pour résoudre ce problème, il faudrait alors récrire une partie de l'encodeur JPEG lui-même.

En ce qui concerne la technique de segmentation par croissance de régions, un critère énergétique similaire est utilisé pour décider de la fusion de deux régions voisines. C'est dans ce critère énergétique que réside la nouveauté de la technique.

À partir des premiers résultats obtenus, nous pouvons conclure que cette technique est prometteuse, même si les temps de calculs nécessaires sont importants. Une étude plus poussée de l'impact des différents paramètres (taille des régions, nombre de coefficients spectraux) sur la qualité visuelle du résultat permettrait d'ajuster au mieux ces paramètres aux caractéristiques de l'image. Ces essais n'ont pu être réalisés, étant donné le temps imparti à la réalisation de ce travail. L'évaluation subjective des résultats reste, dans ce domaine, la seule mesure de la qualité et de l'efficacité de la technique.

# Cinquième partie

# Annexes

# Annexe A

# Images de tests

FIG. A.1: lena

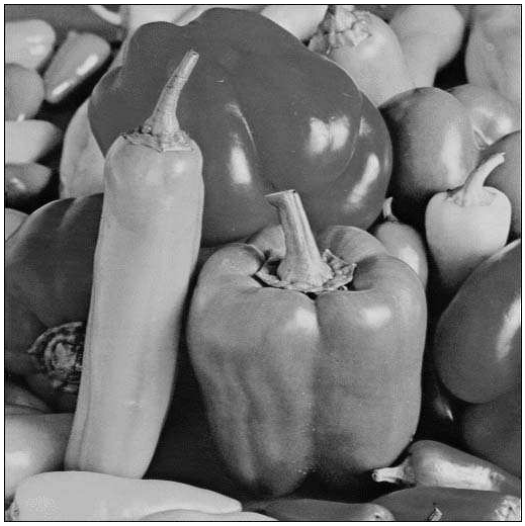


FIG. A.2: peppers

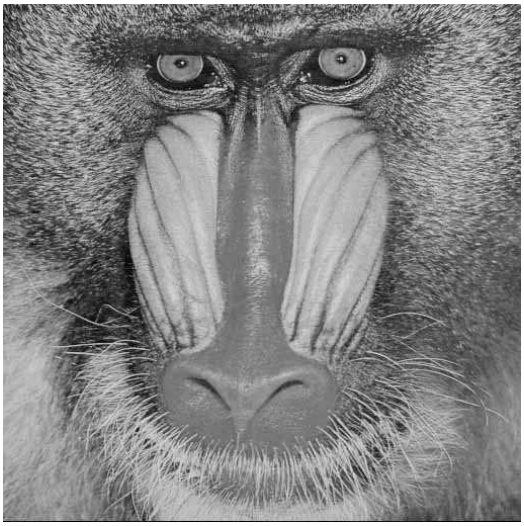


FIG. A.3: mandril

FIG. A.4: palais

FIG. A.5: bateau

FIG. A.6: San Francisco

FIG. A.7: Chat Pigeon

FIG. A.8: Kairouan

# Annexe B

# Outils de mesures

**Entropie** L'entropie est une grandeur qui caractérise la quantité d'information que contient une source d'information. Par exemple, une image dont tous les pixels ont la même valeur contient très peu d'information car elle est extrêmement redondante : son entropie est faible. En revanche, une image dont les pixels ont des valeurs complètement aléatoires contient beaucoup d'information : son entropie est forte.

L'entropie est donc relative à une source $X$. Mais pour passer de cette source à une image, il faut des hypothèses supplémentaires. On peut par exemple, supposer que l'image est la réalisation d'un processus stochastique stationnaire[1] et ergodique[2]. Dès lors, la moyenne statistique est égale à la moyenne spatiale sur les pixels (cette hypothèse est très courante en pratique).

L'entropie d'une image numérique à une composante est donc inversement liée à la probabilité d'apparition des différents niveaux de gris dans l'image. Plus une valeur de gris $k$ est rare, plus sa probabilité d'apparition $p(k)$ est faible, et cela contribue à une entropie globale plus grande. Par définition, l'entropie d'ordre zéro $H_0$ est donnée par :

$$H_0 = \sum_{k=0}^{2^R-1} p(k) \log_2 p(k) \quad \text{bpp}$$

L'utilisation du logarithme de base 2 fait de $H_0$ le nombre de bits moyens par pixel nécessaire pour coder toute l'information contenue dans l'image. Une image codée avec $R$ bits par pixel a en fait presque toujours une entropie d'ordre zéro inférieure à $R$. Dans sa théorie de l'information applicable à tout signal, y compris les images, Shannon a démontré un théorème qui indique qu'il est possible de coder (comprimer) un signal d'entropie $H$ avec $H$ bpp, tout en obtenant une distorsion arbitrairement faible. Par conséquent *l'entropie $H_0$ d'une image originale fournit le débit minimal qu'il est possible d'atteindre par compression*

[1] Dont les lois de probabilités sont indépendantes de l'origine des temps.

[2] Ce qui signifie que l'on peut étudier ce processus en observant une seule réalisation de longueur suffisante de ce processus.

*pixel par pixel sans dégrader l'image, et par là même, un taux de compression sans perte maximal.*

**MSE** (Mean Square Error) L'erreur quadratique moyenne est la mesure de distorsion habituelle utilisée en compression d'images. Cette grandeur est définie par la moyenne des écarts au carré entre le pixel $(m, n)$ de l'image originale $I(m, n)$ et le même pixel de l'image reconstruite $\hat{I}(m, n)$.

$$MSE = \frac{1}{MN} \sum_{m=0}^{M-1} \sum_{n=0}^{N-1} [I(m, n) - \hat{I}(m, n)]^2$$

L'inconvénient de l'erreur quadratique moyenne est qu'elle ne rend pas compte de la perte de qualité visuelle engendrée par la compression. Si tous les pixels d'une image étaient translatés, la MSE serait très élevée alors que la qualité visuelle serait parfaitement bonne. De plus, la MSE est une mesure globale sur toute l'image, qui gomme les variations locales.

**PSNR** On référence souvent la MSE par rapport à la dynamique de l'image en décibels. On obtient la crête du rapport signal sur bruit pour une image dont le maximum est $2^R - 1$, dénoté PSNR (Peak Signal to Noise Ratio). Si l'image reconstruite est parfaite, la MSE est nulle et le PSNR tend vers l'infini.

$$PSNR = 10 \log_{10} \frac{(2^R - 1)^2}{\text{MSE}} \text{ dB}$$

Ces mesures de distorsion sont objectives et simples à calculer. Certaines méthodes de compression recherchent le meilleur compromis entre la performance et la distorsion, et optimisent des courbes taux-distorsion.

# Annexe C

# Codes sources

Se trouvent dans cette annexe les deux principaux sources de ce mémoire : *intraPrediction.c* se rapporte à la partie codage, on peut y retrouver la structure générale de l'algorithme tel que présenté dans le chapitre 8. *IntraSegmentation.c*, quant à lui, contient la fonction principale réalisant la segmentation par croissance de régions présentée à la section 6.3

## C.1 intraPrediction.c

```
/*********************************************************/
/*                                                       */
/* Author : Marc Van Droogenbroeck                       */
/* Date : August 26, 2002                                */
/* Modifications : Renaud Dardenne - April 2003          */
/*********************************************************/

#include <math.h>
#include <stdlib.h>
#include "intraPrediction.h"
#include "renaud.h"
#include "mydct.h"

void intraPrediction(imin,imlabel,imout1,imout2,blockSize,threshold,fd, filenamePred,
                     qualityIntra, qualityInter)
     struct image *imin,*imlabel,*imout1,*imout2;
     int blockSize;
     float threshold;
     FILE *fd;
     char *filenamePred;
     int qualityInter, qualityIntra;
{
  datim in,label,out,lab;
  int size,ok;
```

```
datim lastLabel,regionLabel;
int i,j,coef,Border;
int nbrBlocksH,nbrBlocksV;
int *matrixLabel,*matrixLabelCounter;
struct image immask,imtmp;
struct image imin2,immask2,imtmp2;
struct image imPred;
double mes1,mes2;
int count=0;
int** bloc;
int mean;
int width=imin->x;

/* Allocate bloc */
if((bloc=(int**)malloc(8*sizeof(int*)))==NULL)
  fprintf(stderr,"\nERROR malloc bloc");

for(i=0;i<8;i++)
  if((bloc[i]=(int*)malloc(8*sizeof(int)))==NULL)
    fprintf(stderr,"\nERROR malloc bloc");

/* Size of the blocks */
size = blockSize;
printf("Intra prediction with blocks of size %dx%d\n",size,size);
printf("Intra prediction with %1.2f of coefficents\n\n",threshold);
Border = 2*blockSize+1;

/* Check for error */
if (((imin->x)%size)!=0) runerr("ERROR(interPrediction) : horizontal size should be"
                          " a multiple of block size",1);
if (((imin->y)%size)!=0) runerr("ERROR(interPrediction) : vertical size should be a"
                          " multiple of block size",1);

/* Initialisation */
imera(imlabel,(datim)0);
imera(imout1,(datim)0);
imera(imout2,(datim)0);

// Prepare to fill both images
imout1->x = 0;
imout1->y = 8;
imout2->x = 0;
imout2->y = 8;
lastLabel = (datim)0;

/* Memory allocation */
timmalloc(imin,&immask);
```

```
timmalloc(imin,&imtmp);
timmalloc(imin,&imPred);

imin->x=(imin->x)*2; imin->y=(imin->y)*2;
timmalloc(imin,&imin2);
timmalloc(imin,&immask2);
timmalloc(imin,&imtmp2);
imin->x=(imin->x)/2; imin->y=(imin->y)/2;

in  = (datim *)imin->f;
label = (datim *)imlabel->f;
nbrBlocksH = imin->x/size;
nbrBlocksV = imin->y/size;
matrixLabel = (int *)calloc(nbrBlocksH*nbrBlocksV,sizeof(int));
matrixLabelCounter = (int *)calloc(nbrBlocksH*nbrBlocksV+1,sizeof(int));

/* 1. Treat the first raw */
/* 1.1 First block */
lastLabel=(datim)1;
printf("lastLabel = %d\n",lastLabel);
matrixLabel[0] = lastLabel;
matrixLabelCounter[lastLabel] = size*size;
fillImageWithLabel(imlabel,lastLabel,0,0,size);
fetchBloc(imin, bloc, 0, 0, size);
comprimeBloc(bloc,qualityIntra);
imout1->x+=8;
/* for the output to compress AND for the next extrapolations */
fillOutputImageWithBloc(imout1, bloc, size, width);
fillImageWithBloc(&imPred, bloc, 0, 0, size);
// the Map
count++;
writeMap(0,fd,count);

/* 1.2 Other blocks of the first raw */
for (i=1; i<nbrBlocksH; i++)
  {
    printf("Block[%d,%d]\n",i,0);
    lab = (datim)matrixLabel[i-1];
    coef = (int)rint((double)threshold*(matrixLabelCounter[lab]+size*size));
    icomp_as(imlabel,lab,"==",&immask,lab,0);
    printf("Extrapolation of region %d with %d coefficients\n",lab,coef);

    /* Perform the extrapolation (imtmp) */
    cutPasteTo(&imPred,&immask,&imin2,&immask2,&imtmp2,Border);
    extendFast(&imin2,&immask2,coef,&imtmp2);
    cutPasteFrom(imin,&immask,&imin2,&immask2,&imtmp2,Border,&imtmp);
```

```
    /* Compare the extrapolated signal to the original signal */
    fillImageWithLabel(imlabel,lastLabel+1,i,0,size);
    mes1 = gainExtrapolation(imin,imlabel,&imtmp,(lastLabel+1));

    /* Write the result in imout and adjust the label */
    /* Decoding coded signal for next extrapolations */
    if (mes1>=0.0) {
      fetchBloc(&imtmp, bloc, i, 0, size);
      mean = computeDifference(imin, bloc, i, 0, size);
      // DCT, Q, Q-1, DCT-1
      comprimeBloc(bloc,qualityInter);
      imout2->x+=8;
      fillOutputImageWithBloc(imout2, bloc, size, width);
      computeSum(&imtmp, bloc, i, 0, size, mean);
      fillImageWithBloc(&imPred, bloc, i, 0, size);
      regionLabel=lab; printf("Keep extrapolated result\n");
      /* the map */
      count++;
      writeMap(1,fd,count);
    }
    else {
      fetchBloc(imin, bloc, i, 0, size);
      comprimeBloc(bloc,qualityIntra);
      imout1->x+=8;
      fillOutputImageWithBloc(imout1, bloc, size, width);
      fillImageWithBloc(&imPred, bloc, i, 0, size);
      lastLabel++; regionLabel=lastLabel;
      /* the map */
      count++;
      writeMap(0,fd,count);
    }

    /* Adjust the label of the current block */
    matrixLabel[i] = regionLabel;
    matrixLabelCounter[regionLabel] += size*size;
    fillImageWithLabel(imlabel,regionLabel,i,0,size);
    raster_write(imlabel,"imlabel.ras","n");
  }

/* 2. Treat second raws and following */
for (j=1; j<nbrBlocksV; j++)
  {
    fprintf(stderr,"Treat raw %d\n",j);
    for (i=0; i<nbrBlocksH; i++)
      {
        fprintf(stderr,"Block[%d,%d]\n",i,j);
        ok = 0;
```

```
/* Test the above block */
lab = (datim)matrixLabel[i+(j-1)*nbrBlocksH];
coef = (int)rint((double)threshold*(matrixLabelCounter[lab]+size*size));
icomp_as(imlabel,lab,"==",&immask,lab,0);
printf("Vertical extrapolation of region %d with %d coefficients\n",lab,coef);

/* Perform the extrapolation */
cutPasteTo(&imPred,&immask,&imin2,&immask2,&imtmp2,Border);
extendFast(&imin2,&immask2,coef,&imtmp2);
cutPasteFrom(imin,&immask,&imin2,&immask2,&imtmp2,Border,&imtmp);

/* Compare the extrapolated signal to the original signal */
fillImageWithLabel(imlabel,lastLabel+1,i,j,size);
mes1 = gainExtrapolation(imin,imlabel,&imtmp,(lastLabel+1));

/* Write the result in imout */
if (mes1>=0.0)     {
  fetchBloc(&imtmp, bloc, i, j, size);
  mean = computeDifference(imin, bloc, i, j, size);
  comprimeBloc(bloc,qualityInter);
  imout2->x+=8;
  fillOutputImageWithBloc(imout2, bloc, size, width);
  computeSum(&imtmp, bloc, i, j, size, mean);
  fillImageWithBloc(&imPred, bloc, i, j, size);
  regionLabel=lab; printf("Keep vertical extrapolation\n");
  ok=1;
  /* the map */
  count++;
  writeMap(2,fd,count);
}

/* Adjust the label of the current block */
if ((i!=0) &
    (matrixLabel[i+(j-1)*nbrBlocksH]!=matrixLabel[i-1+j*nbrBlocksH]))
  {
    /* Test the horizontal extrapolation */
    lab = (datim)matrixLabel[i-1+j*nbrBlocksH];
    coef = (int)rint((double)threshold*(matrixLabelCounter[lab]+size*size));
    icomp_as(imlabel,lab,"==",&immask,lab,0);
    printf("%c[91mHorizontal extrap"
           "olation of region %d with %d coefficients%c[0m\n",27,lab,coef,27);

    /* Perform the extrapolation */
    cutPasteTo(&imPred,&immask,&imin2,&immask2,&imtmp2,Border);
    extendFast(&imin2,&immask2,coef,&imtmp2);
    cutPasteFrom(imin,&immask,&imin2,&immask2,&imtmp2,Border,&imtmp);
```

```
    /* Compare the extrapolated signal to the original signal */
    fillImageWithLabel(imlabel,lastLabel+1,i,j,size);
    mes2 = gainExtrapolation(imin,imlabel,&imtmp,(lastLabel+1));

    /* Write the result in imout */
    if ((mes2>=0.0) && (mes2>mes1))
      {
        fetchBloc(&imtmp, bloc, i, j, size);
        mean = computeDifference(imin, bloc, i, j, size);
        comprimeBloc(bloc,qualityInter);
        if(ok==0)
          imout2->x+=8;
        fillOutputImageWithBloc(imout2, bloc, size, width);
        computeSum(&imtmp, bloc, i, j, size, mean);
        fillImageWithBloc(&imPred, bloc, i, j, size);
        regionLabel=lab; printf("Keep horizontal extrapolation\n");
        ok=1;
        /* the map */
        count++;
        if(mes1>=0.0) {
          count--;
          if(fseek(fd,-1.0,SEEK_CUR)==-1) {
            perror("\n Error fseek ");
            exit(-1);
          }
        }
        writeMap(1,fd,count);
      }
  }

/* None of the extrapolated signal improves the S/N */
if (ok==0)
  {
    fetchBloc(imin, bloc, i, j, size);
    comprimeBloc(bloc,qualityIntra);
    // Filling both images
    imout1->x+=8;
    fillOutputImageWithBloc(imout1, bloc, size, width);
    fillImageWithBloc(&imPred, bloc, i, j, size);
    lastLabel++; regionLabel=lastLabel;
    count++;
    writeMap(0,fd,count);
  }

/* Adjust the label of the current block */
matrixLabel[i+j*nbrBlocksH] = regionLabel;
```

```
            matrixLabelCounter[regionLabel] += size*size;
            fillImageWithLabel(imlabel,regionLabel,i,j,size);
          }
        raster_write(imlabel,"imlabel.ras","n");
        raster_write(&imPred, filenamePred,"n");
      }

  /* Memory desallocation */
  imfree(&imtmp);
  imfree(&immask);
  imfree(&imin2);
  imfree(&imtmp2);
  imfree(&immask2);

}
```

## C.2 intraSegmentation.c

```
/*********************************************************/
/*                                                       */
/* Author : Marc Van Droogenbroeck                       */
/* Date : January 26, 2003                               */
/*********************************************************/

#include <math.h>
#include <stdlib.h>
#include "intraPrediction.h"

void intraSegmentation(imin,imlabel,imout,borderSize,threshold)
     struct image *imin,*imlabel,*imout;
     int borderSize;
     float threshold;
{
  datim        *in,*label,*out;
  int   size;
  int   largestLabel;
  int   r1,r2;
  struct image immask,imtmp;
  struct rectangularZone *listRect;
  D2double gain;
  Edge  edge,edge1,edge2;
  datim        lab;
  double       gainMax;
  int   originalContent=1;
  int   limiteInf=3;
  int   limiteSup=30;
  int i,j,coef,Border;
  struct image imin2,immask2,imtmp2;
  double mes1;
  Graph G;

  /* 1. Message to the world and test */
  /* 1.1 Starting message */
  size = borderSize;
  printf("\n-- Intra segmentation via region merging. Border size of %d--\n",size);
  printf("-- Threshold for extrapolation = %7.3g ---------------\n",threshold);

  /* 1.2 Check for error */
  if (((imin->x)%size)!=0) runerr("ERROR(interSegmentation) : horizontal size should "
                                  "be a multiple of block size",1);
  if (((imin->y)%size)!=0) runerr("ERROR(interSegmentation) : vertical size should be"
                                  " a multiple of block size",1);
```

```
/* 2. Construction of an image of labels */
/* By convention, 0==no label.         */
/* Moreover, all labels are comprised between 0 and largestLabel */
/* This will allow to change the way imlabel is built in the future */
largestLabel = maxval(imlabel);
raster_write(imlabel,"imlabelBeforeMerging.ras","n");
printf("-- There are %d labels at the start -------\n\n",largestLabel);

/* 3. Creation and initialization of many intermediate structures */
/* 3.1.a Creation of structures needed to handle the graph */
G = GRAPHinit(largestLabel+1);
listRect = (RectangularZone *)malloc(sizeof(RectangularZone)*(largestLabel+1));

/* 3.1.b Initialization ot the structures */
/* 3.1.b.1 Images and pointers to data structure */
imera(imout,(datim)0);
in  = (datim *)imin->f;
out = (datim *)imout->f;

/* 3.1.b.2 The Graph */
label = imlabel->f;
for (i=0; i<imlabel->x-1; i++)
  for (j=0; j<imlabel->y-1; j++)
    {
      if (label[i+j*imlabel->x]!=label[i+1+j*imlabel->x])
        GRAPHinsertE(G,(int)label[i+j*imlabel->x],(int)label[i+1+j*imlabel->x]);
      if (label[i+j*imlabel->x]!=label[i+(j+1)*imlabel->x])
        GRAPHinsertE(G,(int)label[i+j*imlabel->x],(int)label[i+(j+1)*imlabel->x]);

    }

/* 3.1.b.3 List of Rectangles */
for (i=0; i<=largestLabel; i++)
  {
    listRect[i].x1=imlabel->x; listRect[i].x2=0;
    listRect[i].y1=imlabel->y; listRect[i].y2=0;
    listRect[i].area=0;
    listRect[i].label=i;
  }
for (i=0; i<imlabel->x; i++)
  for (j=0; j<imlabel->y; j++)
    {
      lab = label[i+j*imlabel->x];
      if (i<listRect[lab].x1) listRect[lab].x1=i;
      if (i>listRect[lab].x2) listRect[lab].x2=i;
      if (j<listRect[lab].y1) listRect[lab].y1=j;
      if (j>listRect[lab].y2) listRect[lab].y2=j;
```

```
      listRect[lab].area++;
    }

/* 3.2.a Structures needed for the extrapolation */
gain = (D2double)D2doubleConstruct(largestLabel+1,largestLabel+1,0.0);
Border = size+1;

/* Memory allocation */
timmalloc(imin,&imtmp);
timmalloc(imin,&immask);

imin->x=(imin->x)*2; imin->y=(imin->y)*2;
timmalloc(imin,&imin2);
timmalloc(imin,&immask2);
timmalloc(imin,&imtmp2);
imin->x=(imin->x)/2; imin->y=(imin->y)/2;


/* 3.2.b Computation of the initial matrix containing all the gains */
printf("Start computing the matrix with all the initial gains \n");
for (i=0; i<G->V; i++)
  for (j=0; j<G->V; j++)
    {
      edge.v=i; edge.w=j;
      if (GRAPHget(G,edge)!=0)
        {
          printf("\tExtrapolation of region %d [size=%d] over region %d [size=%d]"
                 ,edge.v,listRect[edge.v].area,edge.w,listRect[edge.w].area);
          lab = edge.v;
          icomp_as(imlabel,lab,"==",&immask,lab,0);

          // Determine the number of spectral coefficients
          coef = (int)rint((double)threshold*listRect[lab].area);
          if (coef<limiteInf) { coef = limiteInf; }
          if (coef>limiteSup)
            { coef = limiteSup; printf("ATTENTION, limitation a %d coefficients!\n"
                                       ,limiteSup); }
          printf("\tExtrapolation of region %d with %d coefficients\n",lab,coef);

          /* Perform the extrapolation */
          if (originalContent==0)
            {
              extendFast(imin,&immask,coef,imout);
              cutPasteToRect(imout,&immask,&imin2,&immask2,&imtmp2,Border,
                             listRect+edge.v,listRect+edge.w);
            }
          else
```

```
            {
              cutPasteToRect(imin,&immask,&imin2,&immask2,&imtmp2,Border,
                             listRect+edge.v,listRect+edge.w);
            }
          extendFast(&imin2,&immask2,coef,&imtmp2);
          cutPasteFromRect(imin,&immask,&imin2,&immask2,&imtmp2,Border,&imtmp,
                           listRect+edge.v,listRect+edge.w);

          /* Compare the extrapolation with the original */
          mes1 = mergingFactor(imin,imlabel,&imtmp,edge.w);
          printf("Gain = %g\n",mes1);
          setValueD2double(gain,edge.v,edge.w,mes1);
          printf("\n");
        }
    }

/* 4. Computation of the maximal gain */
gainMax = getMax(G,gain,&r1,&r2);

/* 5. Region merging loop */
while (gainMax > 0.0)
  {
    printf("Gain Max = %7.3g, en [%d,%d]\n",gainMax,r1,r2);

    /* Merge two regions */
    printf("%c[92mMerging %d and %d -> to region %d%c[0m\n",27,r1,r2,r1,27);
    printf("Before merging : region %d [%d,%d]->[%d,%d] of area=%d\n",
           r1,listRect[r1].x1,listRect[r1].y1,listRect[r1].x2,listRect[r1].y2,
           listRect[r1].area);
    printf("Before merging : region %d [%d,%d]->[%d,%d] of area=%d\n",
           r2,listRect[r2].x1,listRect[r2].y1,listRect[r2].x2,listRect[r2].y2,
           listRect[r2].area);

    /* Adapt the label image */
    label = imlabel->f;
    for (i=0; i<imlabel->x*imlabel->y; i++)
      { if (label[i]==r2) label[i]=r1; }
    raster_write(imlabel,"imlabel.ras","n");

    /* Copy adjancy, add egdes, and remove old ones */
    listRect[r1].area += listRect[r2].area;
    if (listRect[r1].x1 > listRect[r2].x1)
      listRect[r1].x1 = listRect[r2].x1;
    if (listRect[r1].y1 > listRect[r2].y1)
      listRect[r1].y1 = listRect[r2].y1;
    if (listRect[r1].x2 < listRect[r2].x2)
      listRect[r1].x2 = listRect[r2].x2;
```

```
if (listRect[r1].y2 < listRect[r2].y2)
  listRect[r1].y2 = listRect[r2].y2;
printf("REGION %d [%d,%d]->[%d,%d] of area=%d\n",
       r1,listRect[r1].x1,listRect[r1].y1,listRect[r1].x2,listRect[r1].y2,
       listRect[r1].area);
printf("There remains %c[94m %d links %c[0min the graph\n",27,G->E,27);
for (i=0; i<=largestLabel; i++)
  {
    edge1.v=i; edge1.w=r2;
    edge2.v=i; edge2.w=r1;
    if (GRAPHget(G,edge1)!=0)
      {
        if ((GRAPHget(G,edge2)==0)&&(i!=r1))
          GRAPHinsertE(G,edge2.v,edge2.w);
        GRAPHremoveE(G,edge1);
      }
  }

/* Recompute the gains */
for (i=0; i<=largestLabel; i++)
  {
    edge.v=i; edge.w=r1;
    if (GRAPHget(G,edge)!=0)
      {
        printf("\tExtrapolation of region %d [size=%d] over region %d [size=%d]",
               edge.v,listRect[edge.v].area,edge.w,listRect[edge.w].area);
        lab = edge.v;
        icomp_as(imlabel,lab,"==",&immask,lab,0);

        // Determine the number of spectral coefficients
        coef = (int)rint((double)threshold*listRect[lab].area);
        if (coef<limiteInf) { coef = limiteInf; }
        if (coef>limiteSup)
          { coef = limiteSup; printf("ATTENTION, limitation a %d coefficients!"
                                     ,limiteSup); }
        printf("\tExtrapolation of region %d with %d coefficients\n",lab,coef);

        /* Perform the extrapolation */
        if (originalContent==0)
          {
            extendFast(imin,&immask,coef,imout);
            raster_write(imout,"imextra.ras","s");
            cutPasteToRect(imout,&immask,&imin2,&immask2,&imtmp2,Border,
                           listRect+edge.v,listRect+edge.w);
          }
        else cutPasteToRect(imin,&immask,&imin2,&immask2,&imtmp2,Border,
                            listRect+edge.v,listRect+edge.w);
```

```
        extendFast(&imin2,&immask2,coef,&imtmp2);
        cutPasteFromRect(imin,&immask,&imin2,&immask2,&imtmp2,Border,&imtmp,
                         listRect+edge.v,listRect+edge.w);

        /* Compare the extrapolation with the original */
        mes1 = mergingFactor(imin,imlabel,&imtmp,edge.w);
        printf("Gain = %g\n",mes1);
        setValueD2double(gain,edge.v,edge.w,mes1);

        /* Compute the opposite extrapolation */
        lab=edge.v;
        edge.v=edge.w; edge.w=lab;

        printf("\tExtrapolation of region %d over region %d\n",edge.v,edge.w);
        lab = edge.v;
        icomp_as(imlabel,lab,"==",&immask,lab,0);

        // Determine the number of spectral coefficients
        coef = (int)rint((double)threshold*listRect[lab].area);
        if (coef<limiteInf) { coef = limiteInf; }
        if (coef>limiteSup)
          { coef = limiteSup; printf("ATTENTION, limitation a %d coefficients !\n",
                                     limiteSup); }
        printf("\tExtrapolation of region %d with %d coefficients\n",lab,coef);

        /* Perform the extrapolation */
        if (originalContent==0)
          {
            extendFast(imin,&immask,coef,imout);
            raster_write(imout,"imextra.ras","s");
            cutPasteToRect(imout,&immask,&imin2,&immask2,&imtmp2,Border,
                           listRect+edge.v,listRect+edge.w);
          }
        else cutPasteToRect(imin,&immask,&imin2,&immask2,&imtmp2,Border,
                            listRect+edge.v,listRect+edge.w);
        extendFast(&imin2,&immask2,coef,&imtmp2);
        cutPasteFromRect(imin,&immask,&imin2,&immask2,&imtmp2,Border,&imtmp,
                         listRect+edge.v,listRect+edge.w);

        /* Compare the extrapolation with the original */
        mes1 = mergingFactor(imin,imlabel,&imtmp,edge.w);
        printf("Gain = %g\n",mes1);
        setValueD2double(gain,edge.v,edge.w,mes1);
      }
    }

  gainMax = getMax(G,gain,&r1,&r2);
```

```
    }


  /* 6. Free memory */
  imfree(&imtmp);
  imfree(&immask);
  imfree(&imin2);
  imfree(&imtmp2);
  imfree(&immask2);
}
```

# Bibliographie


[1] Y. Lam and J. W. Goodman. A mathematical analysis of the DCT coefficients distributions for images. *IEEE Transactions on Image Processing*, 2000.

[2] M. Nelson. *La compression de données*. Dunod, Paris, 1993.

[3] W. Pennebaker and J. Mitchell. JPEG : Still image data compression standard. *Van Nostrand ReinHold, New York*, 1993.

[4] D. Salomon. Data compression. *Springer-Verlag New York Inc*, 2000.

[5] M. Van Droogenbroeck. *Traitement d'images numériques au moyen d'algorithmes utilisant la morphologie mathématique et la notion d'objet : application au codage*. PhD thesis, Université catholique de Louvain, 1994.

[6] M. Van Droogenbroeck and F. Lapierre. New techniques for intra-prediction and segmentation of images by block-based extrapolation. To appear in Proc. ICASSP2003, April 2003, China.

[7] M. Van Droogenbroeck and H. Talbot. Image segmentation by adaptative extrapolation and region merging.

[8] G. K. Wallace. The JPEG still picture compression standard. *Communications af the ACM*, 1991.

Je remercie *Marc Van Droogenbroeck* et *Jean-Marc Wagner* pour leur aide et leurs explications sur l'ensemble du présent travail, *Moune* pour le café du matin et tout le reste, *Rodolphe* pour le LIDL (mais pas que pour ça), *Régis* pour m'avoir piqué ma page de garde, *Milan Kundera* pour *La Plaisanterie*, ainsi que, en vrac (ça peut toujours servir) : *Miossec*, *Lewis Trondheim*, la FNAC, *Ali San*, mon PC, la boîte du Maroc (merci Rod), ma motivation au travail, toutes les personnes qui ne viendront pas le jour de la défense, la RTBF, *Max* et *Frédé*, la mise à trois bandes de la E42, la Théorie des Groupes, google.be, mes 78 kg à jeun et le passage à l'heure d'été.

Merci.