Systèmes Multimédia

Compression de données

Ce livre s'adresse à toute personne désirant avoir des connaissances de base en compression de données. Il constitue un support de cours pour les étudiants du cycle Master en Systèmes Multimédia

Prof. SLIMANE LARABI, USTHB

Janvier 2022

Sommaire

Introduction

Ce polycopié dédié aux méthodes de compression de données multimédia est le résultat de mes efforts consentis durant les 12 années d'enseignement de ce cours aux étudiants de première année des Masters Réseaux et systèmes distribués (05 années) et Informatique Visuelle (07 années) à l'université des Sciences et Technologie Houari Boumediene.

Dans ce cours nous aborderons les aspects fondamentaux et pratiques de la compression de données sans perte et avec perte appliquée au texte, image, vidéo et l'audio. Nous étudions dans les quatre chapitres suivants :

- Les méthodes de compression sans perte qui sont largement utilisées pour compresser des fichiers tels que des documents, des images, des vidéos et des fichiers audio. Nous nous intéressons essentiellement aux méthodes RLE, LZW et HUFFMAN.
- La compression avec perte (JPEG) qui est largement utilisée pour les images en couleur. Le processus de compression JPEG utilise une transformation mathématique DCT et une quantification pour éliminer les données de haute fréquence de l'image. Les coefficients obtenus sont ensuite codés par l'algorithme de HUFFMAN.
- La compression avec perte (MPEG) qui est appliqué pour réduire la taille des fichiers vidéo tout en préservant une qualité d'image acceptable. Le processus de compression repose sur la subdivision des images vidéo en images clés (I-frames) et images prédictives (P-frames et B-frames). Les images clés sont compressées en JPEG, alors que pour les images prédictives, seuls les différences par rapport à l'image précédente sont codées.

- La compression avec perte de l'audio (MP3) permet de réduire la taille des fichiers audio tout en préservant une qualité d'écoute raisonnable. Le processus de compression MP3 repose sur deux principes de base : le masquage fréquentiel (Les composantes de haute fréquence qui ne sont pas perçues par l'oreille humaine sont éliminées) et le masquage temporel.

Nous terminons ce polycopié, par donner les énoncés des exercices réalisés en travaux dirigés et pratiques.

Chapitre 2.
Compression de données sans perte

2.1 Introduction

Dans cette partie, nous étudions les méthodes de compression de données multimédia. Ces données sont constituées du texte, de l'image, de l'audio et de la vidéo.

Nous présentons un A titre d'exemple, une vidéo de 1 minute à 25 images par seconde nécessitera dans le cas de dimensions d'images : 576x720 pixels : 593,26 Mega octets.

Les scientifiques ont travaillé depuis plusieurs décennies pour réduire cette espace sans autant perdre les données, ou tout au moins perdre ce qui n'est pas utile.

2.2. Intérêts de la Compression

Les intérêts de la compression de données sont multiples et se résument comme suit :

- Gain d'espace mémoire
- Gain en temps de traitement si la méthode appliquée aux données comprimées obtient les mêmes résultats que ceux obtenus sur les données brutes.
- Utilisation dans le cryptage de données. La donnée est d'abord transformée (comprimée) avant de la crypter. Ceci permet de rendre le décodage plus difficile.

2.3. Critères des algorithmes de la Compression

Les techniques de compression peuvent être comparées selon les critères suivants :

- **Taux de compression (Efficacité)**

  Ce taux est défini à l'aide du rapport entre l'espace mémoire du fichier comprimé (E_c) et l'espace mémoire du fichier initial (E) comme suit :

$$Taux = 1 - \frac{E_c}{\mathrm{E}} \times 100$$

  Tout algorithme de compression présente des taux de compression face à différents types de données.

- **Qualité de compression**

La compression peut produire des données de même qualité que les données initiales (brutes), nous disons dans ce cas qu'il s'agit d'une compression sans perte.

L'algorithme de compression peut produire aussi des données avec une qualité moindre, traduite par exemple par un manque de détails dans une image. Nous disons dans ce cas qu'il s'agit d'une compression avec perte.

- **Vitesse de compression/décompression**

La vitesse de compression de données ou décompression est importante notamment pour les applications sur réseau.

Il faudra aussi noter que l'étape de décompression est plus importante que celle de la compression, et nécessite un temps réduit pour fournir les données. A titre d'exemple, citons le cas d'affichage des images téléchargées en format compressé sur une page web, ou la lecture d'une vidéo téléchargée.

- **Accessibilité**

Les algorithmes développés peuvent être une propriété intellectuelle d'une entreprise et dans ce cas son utilisation se fait sous licence. Dans le cas où l'algorithme est développé par une communauté et cédé sans droits, il est appelé libre et peut être donc utilisé gratuitement et le code est accessible pour toute amélioration.

2.4. Algorithmes de Compression sans Perte

Les méthodes de compression sans perte sont basées sur la recherche et le codage des données redondantes.

En effet, la redondance d'une donnée élémentaire (caractère, chiffre, valeur d'intensité d'un pixel, sous chaine de caractères, …) est présente dans les données traitées et produite. L'application de ce type d'algorithme peut se faire aussi dans une des étapes d'autres algorithmes de compression avec perte appliqués à l'image, à la vidéo et au son.

Nous expliquons ci-après ces algorithmes en donnant des exemples et des exercices.

2.4.1 Algorithm RLE (Run Length Encoding):

Principe de base :

Le principe de base de cet algorithme consiste à rechercher des données redondantes (caractères, pixels, …) et de les coder moyennant le nombre d'occurrences et la donnée.

Ainsi à chaque répétition d'une donnée élémentaire plus d'un nombre **n** connu a priori, cette suite de pixels est remplacée par un caractère spécial indiquant la compression suivie par le nombre de répétitions et en fin sa valeur.

A titre d'exemple, soit le texte suivant :

"aaaaaaaabbccccccccccccccddddeeeeeeeeeeeeffffff"
Pour une valeur 4 pour la variable **n**, et supposant que le caractère "!" est utilisé comme symbole annonçant la compression, le codage en RLE de cette chaine est le suivant :
"!8a!14c!4d!11e!5f"

Soit un gain de 27 octets, ce qui est équivalent à 35,71 %

Utilisation de l'algorithme :

Cet algorithme est appliqué pour plusieurs données où les répétitions sont fréquentes, c'est le cas dans :

- Format BMP, PCX, TIFF
- Le fichier ***core*** *dans le système* Unix
- Etape de compression de la matrice DCT quantifiée dans la méthode JPEG
- Dans le codage d'un fax pour sa transmission. Suivant un standard (CCITT), les suites horizontales de pixels blancs et de pixels noirs sont considérées. Le sens bidirectionnel (différence entre lignes) est aussi utilisé pour déduire une ligne de la précédente. En fonction de la couleur du document fax, un algorithme spécifique est utilisé. Cet algorithme est très utile pour coder les lignes blanches ou noires.

Caractéristiques de l'algorithme :

Nous pouvons résumer les caractéristiques de cet algorithme comme suit :

- Simplicité d'implémentation.
- Taux de compression relativement faible par rapport à d'autres algorithmes, mais obtient des taux meilleurs pour les données avec répétitions. C'est le cas d'images contenant des régions homogènes (de même couleur).

Exercices :

Exercice 1 : Codage des images en couleur (RGB) par RLE

Il s'agit de coder un pixel (RGB) sous la forme d'une chaîne de 6 caractères, codé en hexadécimal

(00 à FF). Une image de **n** pixels sera donc représentée par une chaîne de **6n** caractères. Il s'agit de compresser et de décompresser une telle image. Pour cela, le codage suivant est utilisé :

– Si le même pixel se répète 3 fois ou plus, on utilisera un mot de deux octets : le bit de poids fort est à 1, et les 15 autres bits servent à coder le nombre de répétitions du pixel. Ces deux octets sont ensuite suivis de la couleur répétée.

– Si on a une suite de pixels différents (ou qui ne se répètent pas plus de 2 fois), on fera précéder cette suite d'un mot de deux octets où le bit de poids fort est à 0, et les 15 autres bits servent à coder la taille de la suite.

Calculez le taux de compression d'une image 6x6 pour différentes configurations (voir figure ci-dessous).

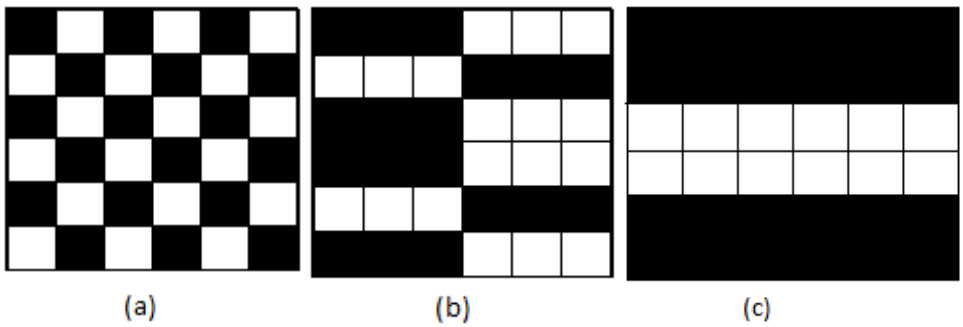


Figure 2.1 Exemple d'images

Solution:

1) (0028H) N B N B….N B, (2+1)x36=108 octets
2) (8003H) N (8006H)B (8006H) N (8003H) B (8003H) N (8006H) B (8006H) N (8003H) B, soit 3x8=24 octets
3) (800AH) N (800AH) B (800AH) N, soit 9 octets

2.4.2 Algorithm LZW (Lempel Ziv Welch)

La compression LZ77

Abraham Lempel et Jakob Ziv sont les créateurs du compresseur LZ77, inventé en 1977 (d'où son nom) [Ziv&Lempel77].

Figure 2.2. De gauche à droite les auteurs de LZW Abraham Lempel (Pologne), Jacob Ziv(Israel) et Terry Welch (MIT, USA)

Principe de base du Codage LZ77

LZ77 est proposé pour l'analyse des données et la réduction de son espace en remplaçant les informations redondantes avec des métas donnés.

Deux fenêtres : W_f (forward window) pour se déplacer dans le texte qui reste à compresser, W_b (backward window) qui contient ainsi la portion précédemment codée.

Si une sous chaine de W_f existe dans W_b, alors elle est récrite par une métadonnée : (**pos**, **long**, **car_suiv**), où :

pos : indique où commence la chaine équivalente dans W_b,

long : sa longueur

car_suiv : est le caractère non compressé suivant.

Ce caractère est inséré parce que les auteurs ont jugé que s'il n'a pas été codé dans l'expression lue, c'est qu'il y a de grandes chances qu'il ne soit pas compressé ensuite.

Cet algorithme utilise un dictionnaire constitué de chaines W_b.

La distance pos est limitée à 32KO, et la longueur long à 258 Octets.

Exemple de compression

Soit le texte à coder " how-much-wood-would-a-woodchuck " en entrée.

Taille de W_b=16, taille de W_f=5.

'd-' sera codé : (6, 2, a).

On déplace ensuite la fenêtre de 3 caractères. Ce qui donne alors :

'-wood' sera codé : (13, 5, c).

(0,0,h),(0,0,u), (0,0,c), (0,0,k).

Si on ne trouve pas d'équivalence, un caractère prend plus de place qu'un octet!

Les problèmes et inconvénients de la compression LZ77

Le problème est de trouver la plus grande équivalence au plus vite. La méthode brutale, qui consiste à tester une par une les équivalences, n'est pas acceptable. Il est alors recommandé d'utiliser un arbre binaire de recherche.

L'autre inconvénient de LZ77 est qu'il renvoie toujours un triplé (position, longueur, caractère suivant) même si l'équivalence n'est que d'un octet (1 caractère) ou même aucun. Dans ce cas, nous utilisons plus que 8 bits pour coder 1 caractère.

Décompression LZ77

La décompression sera très rapide puisque la recherche d'équivalence se fait dans la fenêtre W_b.

Ce compresseur était alors utilisé pour l'archivage (les formats ZIP). Il a été ensuite amélioré par Terry Welch de la société Unisys en 1984. Il est basé sur un dictionnaire (bibliothèque) construit au fur et à mesure de la lecture du fichier à coder.

Les chaînes de caractères sont placées une par une dans la bibliothèque. Lorsqu'une chaîne est déjà présente dans la bibliothèque, son code de fréquence d'utilisation est incrémenté.

Les chaînes de caractères ayant des codes de fréquences élevés sont remplacées par un " mot " ayant un nombre de caractères le plus petit possible et le code de correspondance est inscrit dans la bibliothèque.

On obtient ainsi l'information encodée et sa bibliothèque.

La compression LZW

```
Algorithm
    Begin
    w = ø
    Repeat
        Lire un caractère k
        If wk ∈ Dictionary
                Then w = wk
                Else    sortie : (code (w))
                        Insérer wk dans le dictionnaire
                        w = k
        EndIf

    Until EOF
    End
```

Caractéristiques :

• Cet algorithme est breveté par la société Unisys, il a été utilisé dans les formats TIFF et GIF, par contre l'algorithme LZ77 est libre de droit et a été utilisé dans le format PNG.

• Il s'applique bien au texte et sur les images de faibles profondeurs (nombre réduit de couleurs différentes) puisque les motifs différents doivent être relativement faibles pour être répétés.

• Il est l'un des plus répandus algorithmes, et est très rapide aussi bien en compression qu'en décompression.

Utilisation de l'algorithme :

Le codage LZW est utilisé dans :
- les formats GIF, TIFF et MOD.
- *Man* pages (Unix), GIF (*Graphical Interface Format*)

Exemple de compression de texte :

Input: **^wed^we^wee^wed^**

Sub string	Code
^w	90
we	91
ed	92
d^	93
^we	94
e^	95
^wee	96
e^w	97
wed	98

w	k	wk	∈ Dict?	Insert in Dict	Output
ø	^	^	yes		
^	w	^w	no	^w	^
w	e	we	no	we	w
e	d	ed	no	ed	e
d	^	d^	no	d^	d
^	w	^w	yes		
^w	e	^we	no	^we	^w
e	^	e^	no	e^	e
^	w	^w	yes		
^w	e	^we	yes		
^we	e	^wee	no	^wee	^we
e	^	e^	yes		
e^	w	e^w	no	e^w	e^
w	e	we	yes		
we	d	wed	no	wed	we
d	^	d^	yes		
d^	ø	d^			d^

La décompression LZW (*Lempel Ziv Welch*) :

Algorithm
Begin

Lecture du code k;
Sortie: Dict[k];
w = Dict[k];
while (Lire k)
#k est le code d'un caractère ou d'une sous chaine #déjà insérée ou à insérer dans l'étape courante
{
Sortie: Dict[k];
Insérer **w Dict[k][0]** (Premier caractère de la sous chaine) dans le dictionnaire
w = Dict[k];
}
End

Exemple 1:

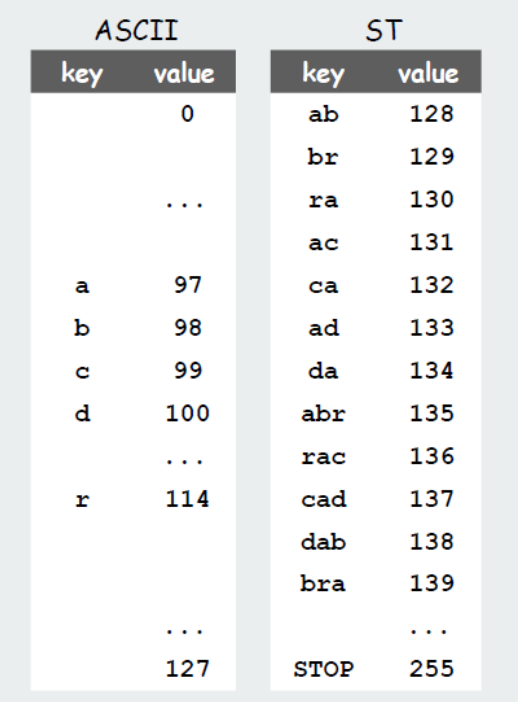
Dict={0->a, 1->b)
Chaine de codes: 0 1 2 4 3 6
Algorithm
W = 0;
output 'a';

K=1; output 'b'; ch='b'; Dict(2)='ab'; W='b';
K=2; output 'ab'; ch='a'; Dict(3)='ba'; W='ab';
K=4; output 'ab..a'; ch='a'; Dict(4)='aba'; W='aba';
K=3; output 'ba'; ch='b'; Dict(5)='abab'; W='ba';
K=6; output 'ba..b'; ch='b'; Dict(6)='bab'; W='bab';

Exemple 2:

Dict={0->a, 1->b, 2->c)
Chaine de codes: 0 3 1 2
Algorithm
W = 0;
output 'a';
K=3; output 'a..a'; ch='a'; Dict(3)='aa'; W='aa';
K=1; output 'b'; ch='b'; Dict(4)='aab'; W='b';

K=2; output ‘c’; ch=‘b’; Dict(5)=‘bc’; W=‘c’;
}



ASCII

key	value
	0
	...
a	97
b	98
c	99
d	100
	...
r	114
	...
	127

ST

key	value
ab	128
br	129
ra	130
ac	131
ca	132
ad	133
da	134
abr	135
rac	136
cad	137
dab	138
bra	139
	...
STOP	255

Décodage

Exemple 1 :

Dict={0->a, 1->b, 2 -> c)

Chaine de codes: 0 2 3 5 4 7

« **acacacacacac** »

Output: 0 2 3 5 4 7

Algorithm

prevcode = 0;

output ‘a’;

while (there is still data to read)

{

curcode=2; output ‘c’; ch=‘c’; Dict(3)=‘ac’; prevcode=‘c’;

curcode=3; output ‘ac’; ch=‘a’; Dict(4)=‘ca’; prevcode=‘ac’;

curcode=5; output ‘ac..a’; ch=‘a’; Dict(5)=‘aca’; prevcode=‘aca’;

curcode=4; output ‘ca’; ch=‘c’; Dict(7)=‘acac’; prevcode=‘ca’;

curcode=7; output ‘ca..c’; ch=‘c’; Dict(7)=‘cac’; prevcode=‘cac’;

}

Dict[i]	a	b	c	ac	ca	aca	acac	cac
i	0	1	2	3	4	5	6	7

2.4.3 Algorithme de HUFFMAN


David Huffman

•Cet algorithme est développé en 1952 par David Huffman [Référence].

Il est l'un des algorithmes les plus anciens, son codage est basé sur la fréquence d'apparition d'un caractère : plus le caractère apparaît souvent plus son code sera court et vice-versa.

• Pour permettre un décodage unique les codes attribués aux différents caractères doivent être préfixés, c'est-à-dire qu'aucun caractère n'est un préfixe d'un autre.

•On appelle aussi ce codage un VLC préfixé (Variable Length Code, code à taille variable).

•

Algorithme Huffman:

Début

– Chercher la fréquence d'apparition de chaque caractère.

Répéter

•- Trier les caractères par ordre décroissant de fréquence (poids).

•- Construire l'arbre binaire comme suit :

•- Relier les deux caractères de fréquences les plus basses et affecter à ce nœud la somme des fréquences des caractères.

•

Jusqu'à ce que tous les nœuds soient reliés

L'arbre étant construit, on met un 1 sur la branche à droite du nœud et un 0 sur celle de gauche.

Fin

Le codage de Huffman :

Parcourir l'arbre de la racine vers chacune des feuilles pour tirer le code de chaque caractère.

•Le codage de Huffman :

•Exemple: "^wed^we^wee^wed^":

- e: 5
- ^: 5
- w: 4
- d: 2

Caractéristiques :

Cet algorithme permet d'avoir un taux de compression très élevé (50% en moyenne) et un temps de compression assez rapide.

La bibliothèque doit être transmise avec le fichier.

Il est très sensible : la perte d'un bit entraîne une altération de toutes les données qui suivent lors de la décompression

Chapitre 3.
Compression de l'image

3.1 Introduction

La norme de compression JPEG (Joint Photographic Experts Group) représente actuellement le standard de compression avec perte le plus utilisé pour les images naturelles.

Cette norme de compression mondiale d'images fixes est apparue à la fin des années 80.

3.2 Principe de base de la compression JPEG

- L'image est décomposée en blocs de tailles 8x8 pixels.
- Chacun de ces blocs est transformé en une matrice contenant une représentation fréquentielle du signal intensité.

Pour cela la DCT (transformation en cosinus discrète) est utilisée. La matrice obtenue est caractérisée par le fait que les valeurs de haut gauche représentent l'information de basse fréquence, et plus on se déplace en bas à droite, elle code l'information de haute fréquence.

La matrice DCT (transformation en cosinus discrète) obtenue est ensuite quantifiée. Chaque élément de la DCT est divisé par une entier. Les entiers diviseurs utilisés sont calculés et rangés dans une matrice appelée matrice de quantification.

La matrice résultat est ensuite quantifiée en utilisant Huffman ou RLE en Zigzag

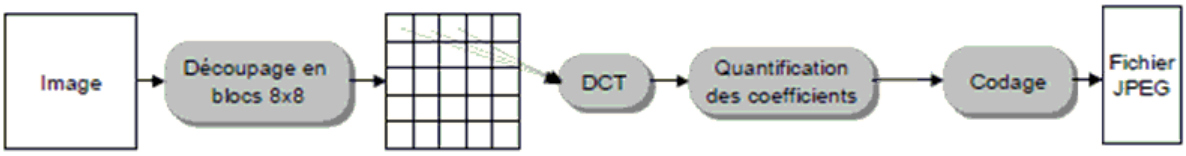


Figure 3.1. Schéma générale de la compression JPEG

3.2.1 Calcul de la matrice DCT

Le but des transformations opérés sur le bloc image est de compacter au mieux l'information contenue dans l'image et d'voir un nombre de coefficients représentatifs aussi faible de possible.

Soit F(x, y) les coefficients de la matrice où est stockée la partie de l'image à traiter, avec (x,y)=(0,0) ...(7,7)

Soit C(u, v) les coefficients de la matrice obtenue après la transformation D.C.T., avec (u,v)=(0,0) ...(7,7).

La formule de la D.C.T. est donnée par l'équation suivante :

$$C(u,v)=\frac{2}{N}\alpha(u)\alpha(v)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)\cos\left[\frac{(2x+1)u\pi}{2N}\right]\cos\left[\frac{(2y+1)v\pi}{2N}\right]$$

$$\alpha(u)=\frac{1}{\sqrt{2}} \text{ si u = 0 , sinon égal à 1}$$

Etudions d'abord le calcul de la DCT à une dimension pour comprendre que représentent les coefficients obtenus.

Le coefficient C(u) uo...7 est évalué pour x=0...7, (N=8) moyennant f(x) et le cosinus dans l'expression suivante :

$$C(u)=\frac{\sqrt{2}}{\sqrt{N}}\alpha(u)\sum_{x=0}^{N-1} f(x)\cos\left[\frac{(2x+1)u\pi}{2N}\right]$$

$$\underset{u=1}{\cos}\left[\frac{(2x+1)u\pi}{16}\right]$$

x=	0	1	2	3	4	5	6	7
	0,98	0,83	0,55	0,19	-0,19	-0,55	-0,83	-0,98

$$u=1,\ DCT(u)=DCT(1)=\text{Coeff}^*\sum_{x=0}^{7} F(x)\ \text{Cos}\left[\frac{(2x+1)\pi}{16}\right]$$

$$u=2,\ C(2)=\text{Coeff}^*\sum_{x=0}^{7} F(x)\ \text{Cos}\left[\frac{(2x+1)2\pi}{16}\right.$$

$$\text{Cos}\left[\frac{(2x+1)2\pi}{16}\right]$$

x=	0	1	2	3	4	5	6	7
	0.92	0.38	-0.38	-0.92	-0.92	-0.38	0.38	0.92

Valeurs de cosinus pour

u=0, 1, 2, 3, 4, 5, 6, 7 et pour x= 0, 1, 2, 3, 4, 5, 6, 7

x\u	0	1	2	3	4	5	6
0	1	0.980	0.923	0.831	0.707	0.555	0.382
1	1	0.831	0.382	-0.195	-0.707	-0.980	-0.923
2	1	0.555	-0.382	-0.980	-0.707	0.195	0.923
3	1	0.195	-0.923	-0.555	0.707	0.831	-0.382
4	1	-0.195	-0.923	0.555	0.707	-0.831	-0.382
5	1	-0.555	-0.382	0.980	-0.707	-0.195	0.923
6	1	-0.831	0.382	0.195	-0.707	0.980	-0.923

Image= [200 200 200 200 100 100 100 100]
DCT[ ]=[424,26 127.5 0 -44.5 0 28 0 -26.5]
DCT[0]=1200*sqrt(2/8)/sqrt(2)=424.26
DCT[1]=[(0,98+0,83+0,55+0,19)*100]*sqrt(0.25)=127.5
DCT[2]=0
DCT[3]=[(0,83-0,19-0,98-0,55)*200+(0,55+0,98+0,19-0,83)*(100)]**sqrt(0.25)= -44.5
DCT[4]=0
DCT[5]=[(0,55+0,19+0,82-0,98)*200+(0,98-0,19-0,83-0,56)*100] *sqrt(0.25)=28
DCT[6]=0
DCT[7]=[(0,19+0,82-0,55-0,98)*200+(0,98+0,56-0,83-0,20)*100]*sqrt(0.25)=-26.5

Pour retrouver les valeurs des pixels, nous appliquons la DCT inverse. La formule est donnée dans le sous-section suivante.

x(0)= 424,26*1*0,35+ (127,5*0,98 -44,5*0,83+27,5*0,55-26,5*0,19)*0,5=197,54
x(1)= 424,26*1*0,35+(127,5*0,83+44,5*0,19 -27,5*0,98+26,5*0,55)*0,5 =199,44
x(4)= 424,26*1*0,35+(-127,5*0,19-44,5*0,55 -27,5*0,83-26,5*0,98)*0,5 = 99,74
x(5)= 424,26*1*0,35+(-127,5*0,55-44,5*0,98 -27,5*0,19+26,5*0,83)*0,5=100

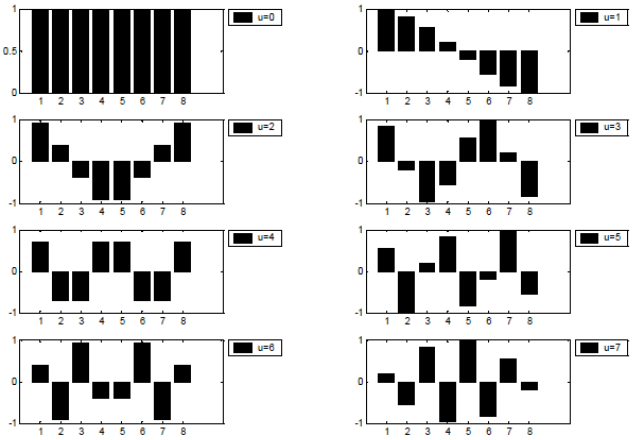


Figure 3.2. DCT en 1D

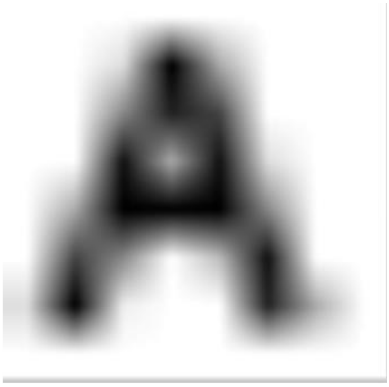


Figure 3.3. Exemple d'image initiale qui s'écrira comme étant la somme pondérée d'images élémentaires DCT 2D.

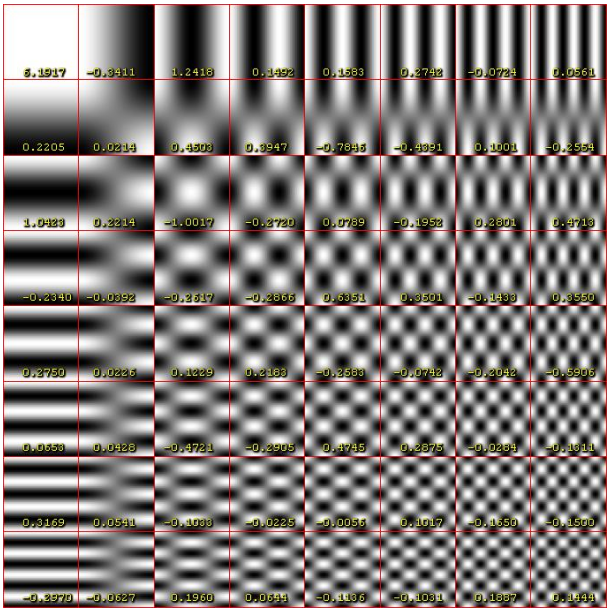


Figure 3.4 La DCT en 2D

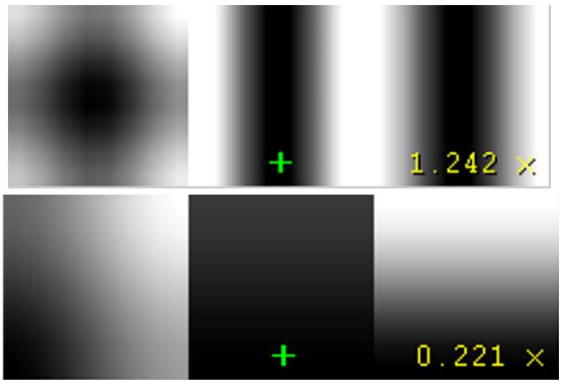


Figure.3.5. Image cumul (la plus à gauche) obtenue suite à l'addition de l'image courante (milieu) avec le produits d'un facteur et de la DCT en 2D correspondante.

3.2.2- Quantification des coefficients transformées

→ Coder au mieux des coefficients souvent réels ou complexes en introduisant une erreur de quantification.

- Phase qui provoque une dégradation dans l'image reconstruite

Opération qui permet d'obtenir des taux de compression beaucoup plus importants que dans le cas d'une compression sans perte.

Soit F(u,v) les coefficients de la matrice obtenue après la D.C.T.
et Q(i,j) les coefficients de la matrice de quantification, avec (u,v,i,j)=0,7.
Soit Cq(k,l) les coefficients de la matrice obtenue après quantification, avec (k,l) =0,7.
Alors Cq(k, l)=E(F(k, l)/a(k, l)), avec E(x) la partie entière de x.

Après le calcul de la D.C.T., nous trouvons les fréquences élevées sont dans le coin droit du bas de la matrice. Or l'œil humain discerne mal ces fréquences.
Ainsi la manière la plus astucieuse de créer la matrice de quantification est de mettre des valeurs faibles dans le coin gauche du haut et des valeurs fortes dans le coin droit du bas.

Cette opération consiste à diviser chaque coefficient *ai;j de la matrice DCT par le coefficient de la matrice de quantification associé qi,j* .
Les coefficients de la matrice de quantification ont été choisis de sorte que le coefficient par lequel on va diviser le coefficient associé dans la matrice DCT, soit d'autant plus grand que la fréquence est élevée.
Une fois qu'on a appliqué la DCT on peut négliger les hautes fréquences.
Afin de contrôler la perte de qualité de l'image, un facteur de qualité *Fq est défini :*
Avec ce coefficient on va créer une matrice de quantification *Q = q(i,j) définie par la relation suivante :*
*Q(i,j) = 1 + (1+i + j) * Fq.*

Exemple de quantification

Matrice de quantification pour *Fq = 5, q(i,j) = 1 + (1+i + j) * Fq.*

6	11	16	21	26	31	36	41
11	16	21	26	31	36	41	46
16	21	26	31	36	41	46	51
21	26	31	36	41	46	51	56
26	31	36	41	46	51	56	61
31	36	41	46	51	56	61	66
36	41	46	51	56	61	66	71
41	46	51	56	61	66	71	76

Exemple de Matrice DCT

1758	54	6	-1	-13	-5	23	-11
9	-15	-9	-16	27	31	-32	1
-8	7	8	-15	-16	3	-67	-40
-30	4	44	-36	14	-73	20	-4
20	2	-23	-10	-16	-16	-8	7
2	8	-12	17	-16	-21	-40	36
-14	45	-49	-20	-31	29	41	51
-1	-66	1	20	-4	-31	-2	-31

Division de Matrice DCT par la matrice de quantification

293	4	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
-1	0	1	-1	0	-1	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	1	-1	0	0	0	0	0
0	-1	0	0	0	0	0	0

Cette opération a éliminé les hautes fréquences non primordiales dans l'image d'où l'apparition de nombreux zéros en bas à droite.

3.2.3- Codage

D'autre part, afin de faciliter le codage par place des coefficients DCT, il est important de choisir un sens de parcours des coefficients qui permette d'aller des coefficients les plus importants vers les coefficients faibles et souvent nulles.

Méthodes de codage des coefficients
Codage de Huffman
Le codage de Huffman est ensuite utilisé pour coder résultat du codage par plage ainsi que le codage différentiel des coefficients. Le fichier ainsi crée est le fichier compressé JPG.

3	5	7	9	11	13	15	17
5	7	9	11	13	15	17	19
7	9	11	13	15	17	19	21
9	11	13	15	17	19	21	23
11	13	15	17	19	21	23	25
13	15	17	19	21	23	25	27
15	17	19	21	23	25	27	29
17	19	21	23	25	27	29	31

exemple de table de quantification
et sens de parcours

3.2.4- Exemple

100 100 100 100 100 100 100 100
100 100 100 100 100 100 100 100
100 100 100 100 100 100 100 100
100 100 100 100 100 100 100 100
100 100 100 100 100 100 100 100
100 100 100 100 100 100 100 100
100 100 100 100 100 100 100 100
100 100 100 100 100 100 100 100

Image (8x8)

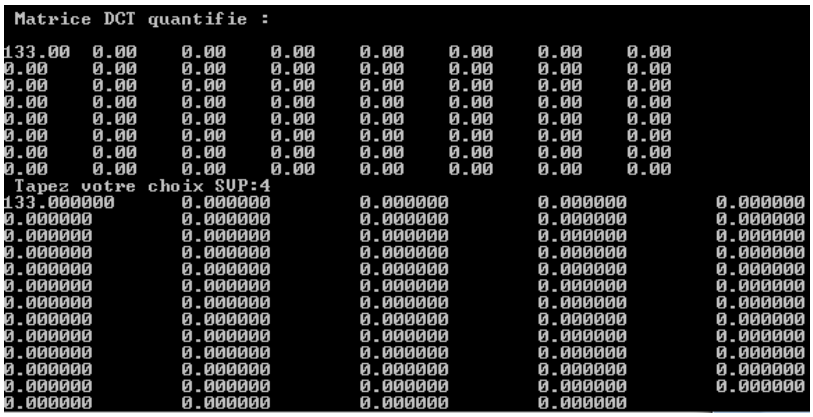
```
Matrice DCT calculee:

799.00  0.00    0.00    0.00    0.00    0.00    0.00    0.00
0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
0.00    0.00    0.00    0.00    0.00    0.00    0.00    0.00
```

```
Facteur de qualite : 5

Matrice de qualite :

6   11   16   21   26   31   36   41
11   16   21   26   31   36   41   46
16   21   26   31   36   41   46   51
21   26   31   36   41   46   51   56
26   31   36   41   46   51   56   61
31   36   41   46   51   56   61   66
36   41   46   51   56   61   66   71
41   46   51   56   61   66   71   76
```



3.3 Décompression JPEG

Etapes :

- Décodage de Huffman

(Matrice Quantifiée)

- Déquantification

Multiplication par le facteur de qualité de la matrice quantifiée. On obtient la matrice DCT

- Calcul de la DCT INVERSE qui donnera l'image initiale

DCT Inverse s'écrit :

$$f(i,j) = F^{-1}(u,v) = \left(\frac{2}{N}\right)^{\frac{1}{2}} \cdot \left(\frac{2}{M}\right)^{\frac{1}{2}} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} \Lambda(u) \cdot \Lambda(v) \times \cos\frac{\pi u}{2N}(2i+1) \cos\frac{\pi v}{2M}(2j+1) \cdot F(u,v)$$

$$f(x,y) = \frac{2}{N} \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} \alpha(u)\alpha(v)C(u,v) \cos\left[\frac{(2x+1)u\pi}{2N}\right] \cos\left[\frac{(2y+1)v\pi}{2N}\right]$$

$$\alpha(u) = \frac{1}{\sqrt{2}} \text{ si u} = 0 \text{ , sinon égal à 1}$$

Chapitre 4.
Compression de la vidéo

Chapitre 4.
Compression de la vidéo

Sommaire

4.1 Introduction

Pour réduire la quantité d'informations relatives à une vidéo, il faudra compresser : bit rate réduction.

Méthodes simples de réduction : (pas suffisantes)
- Réduction de la taille de l'image
- Réduction du nombre d'images par seconde

Méthodes avancées :
- Une image contient beaucoup d'informations redondantes.
- L'œil humain est moins sensible à la couleur qu'à la luminance de l'image. Il est possible donc de réduire l'information de chrominance sans que l'œil se rende compte.

Résultat : Dissocier l'information chrominance, luminance

4.2 Principe de base de la compression MPEG

Deux techniques peuvent être utilisées pour la compression vidéo :
- **Compression intra-trame** : Prise en compte de la similarité dans les zones de la même image.
- **Compression inter-frames** : Prise en compte de la similitude d'une image et celles qui l'entoure dans la séquence (précédente, suivante), on ne tiendra en compte que la différence en elles.

4.2.1 Format YUV 4 : 1 : 1

L'œil humain étant plus sensible aux variations de luminance que de chrominance, on utilise l'espace de représentation de couleurs YUV, en sous-échantillonnant les composantes U et V.

Y représente la luminance, **U** et **V** les composantes Rouge et Bleu (chrominances).

Ces 3 informations permettent de restituer les composantes RGB.

Dans le format YUV 4 :1 :1, les composantes de chrominance sont réduites à la moitié de la résolution verticale et horizontale, soit 4 composantes de luminance pour une composante U, et une composante V

Ce sous-échantillonnage correspond à une réduction de la redondance psycho-visuelle.

Le signal YUV est créé depuis une source RGB (rouge, vert et bleu).
Les valeurs de R, G et B sont additionnées selon leur poids relatif pour obtenir le signal Y.

Ce dernier représente la luminance de la source.

Le signal U est obtenu en soustrayant le Y du signal bleu d'origine.

Le signal V est obtenu en soustrayant Y du signal rouge.

De RVB à YUV :

De RVB à YUV :
$$Y = 0{,}299 \cdot R + 0{,}587 \cdot V + 0{,}114 \cdot B$$
$$U = 0{,}492 \cdot (B - Y) = -0{,}147 \cdot R - 0{,}289 \cdot V + 0{,}436 \cdot B$$
$$V = 0{,}877 \cdot (R - Y) = 0{,}615 \cdot R - 0{,}515 \cdot V - 0{,}100 \cdot B$$
De YUV à RVB :
$$R = Y + 1{,}140 \cdot V$$
$$G = Y - 0{,}395 \cdot U - 0{,}581 \cdot V$$
$$B = Y + 2{,}032 \cdot U$$

Figure 4.1. Image RGB, et composantes R, G et B

Figure 4.2. Les composantes Y, U et V

4.2.2 La compression spatiale

Elle utilise sur les images la technique propre à la norme JPEG :

Divise la vidéo en frames et applique la compression JPEG sur chaque image (JPEG motion).

Ne prend pas en considération la corrélation qui existe entre les différents frames.

La redondance temporelle n'est pas prise en compte
Simple et facile à implémenter

4.2.3 La compression temporelle

La compression temporelle a pour objectif l'e*xploitation de la redondance temporelle entre frames adjacentes dans une vidéo mais non identiques.*
La compression par compensation de mouvement
Elle consiste à :
La vidéo est en réalité un fond statique (background) et un ensemble de d'objets en mouvement (foreground)
- Coder le premier frame en JPEG et l'utiliser comme référence
- Diviser le prochain frame en blocs et comparer chacun d'eux avec le bloc correspondant dans le frame de référence.
- Coder les différences entre les blocs
- Continuer avec les autres frames

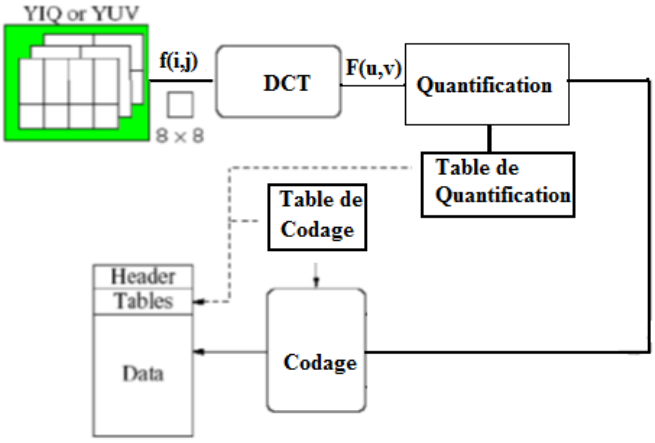


Figure 4.3. Schéma synoptique de la compression JPEG

Figure 4.4. Exemple de séquence d'images

Comment comparer les blocs (Blocs Matching)

- Trouver les blocs entre les frames n+1 et n ayant les intensités similaires
- Utiliser l'erreur de moyenne quadratique pour sélectionner le plus similaire.

$$MSE = \left(\frac{1}{NxM}.\right) \sum_{x,y} (I_{n+1}(x+dx, y+dy) - I_n(x,y))^2$$

Pour chaque bloc b_i de l'image I_n chercher le bloc B_j correspondant dans l'image I_{n+1} permettant d'avoir MSE minimum.

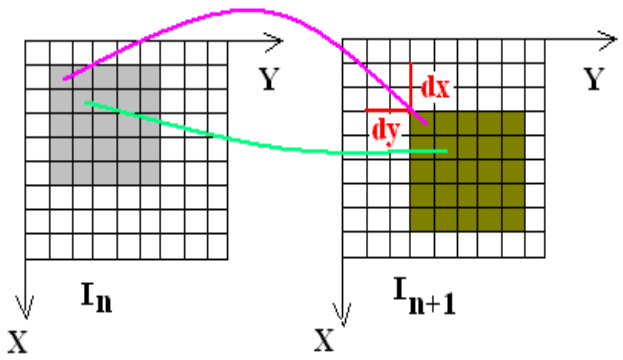


Figure 4.5. Principe du bloc matching

L'efficacité du codage dépend de la taille des blocs (pour MPEG, les blocs sont à 16x16)

- Plus la taille est grande, les erreurs (résidus) des blocs deviennent importants.
- Plus la taille est petite, le nombre de blocs en mouvement est important

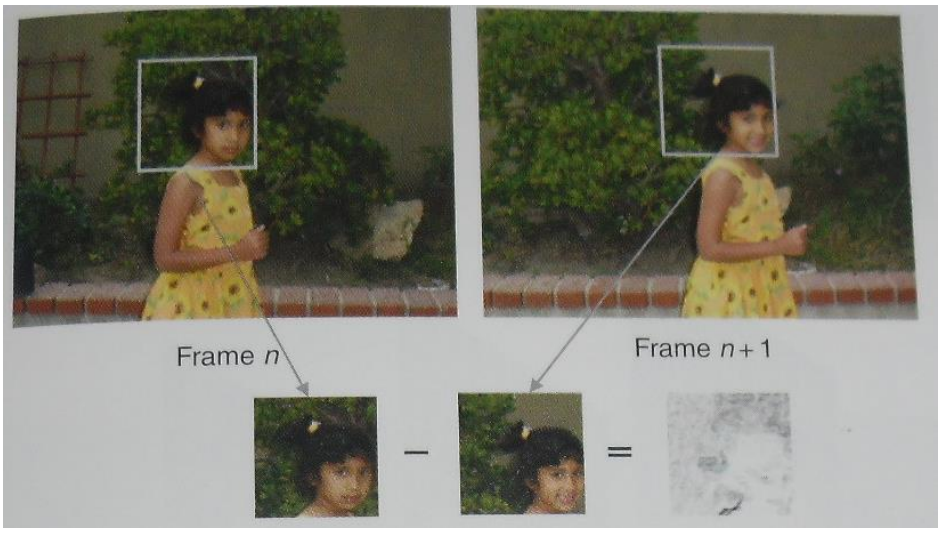


Figure 4.6. Exemple du calcul du bloc similaire et du résidu [2]

Vecteurs de mouvement (Motion vectors)

Calculer le vecteur déplacement entre les blocs appariés

Ce vecteur est utilisé par le décodeur pour retrouver l'emplacement du bloc sur la frame n à partir du bloc sur la frame précédente (n-1)

Le vecteur mouvement est nul pour un background statique, mais ceci n'est pas toujours vrai.

Espace de recherche

Si la recherche d'un bloc correspondant se fait sur tout le frame, il y aura beaucoup de calculs.

Restreindre la recherche d'un bloc sur un voisinage très limité.

- Petit espace de recherche pour un mouvement lent
- Grand espace de recherche pour un mouvement rapide

Codage du résidu

La différence entre le bloc codé et le meilleur bloc correspondant dans le frame précédent est connu par le terme résidu.

Ce résidu doit être codé et transmis avec le vecteur mouvement pour pouvoir reconstruire le bloc.

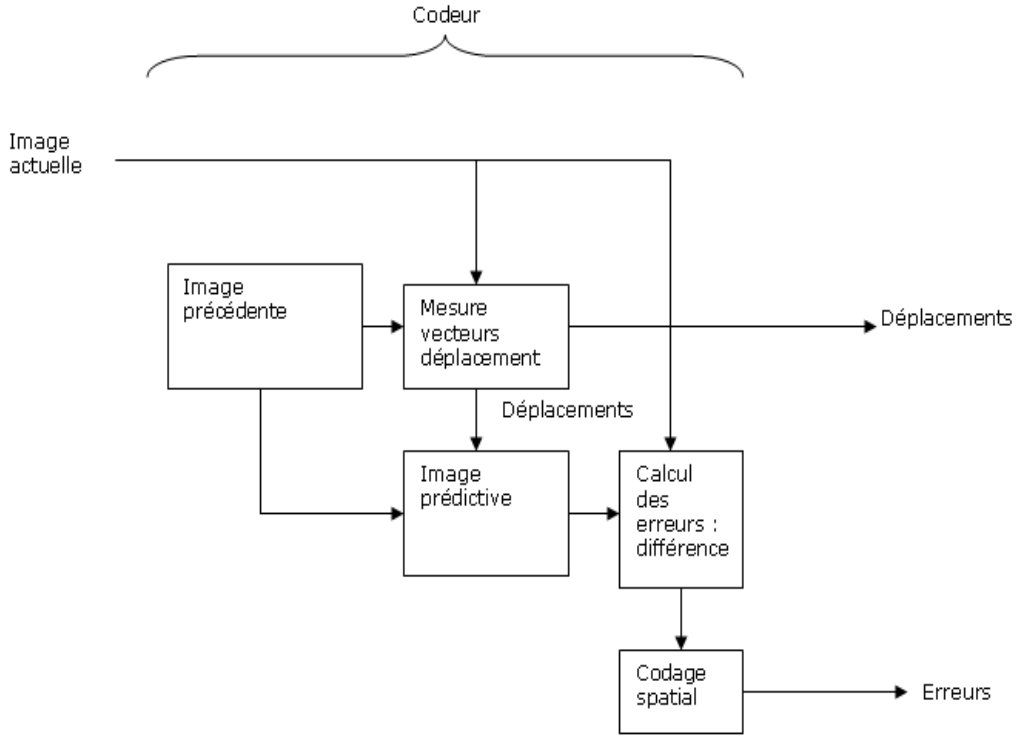


Figure 4.6. Codage temporel MPEG

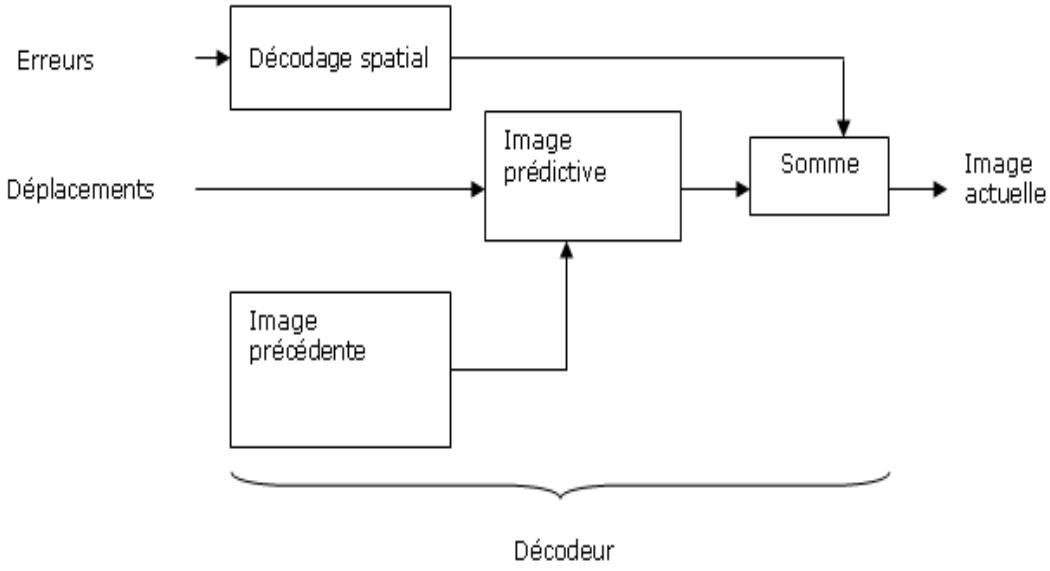


Figure 4.7. Décodage temporel MPEG

Types de prédiction

La compression des frames vidéo peut se faire soit :

- En intra-mode qui suit le standard jpeg pour la compression de chaque image (type I).
- Inter mode le frame est prédite en se basant sur le frame de référence qui est choisie comme étant la précédente frame (type P).
- Dans certaines circonstances, l'utilisation d'une frame suivante est utile (type B).

Les frames de type I

Pour ce type de frame, seule la redondance spatiale est utilisée pour la compression (jpeg).

Les frames de type **I** sont insérées périodiquement dans la séquence des frames prédites.

Pour ce type de frame, seule la redondance spatiale est utilisée pour la compression (jpeg).

Les frames de type **I** sont insérées périodiquement dans la séquence des frames prédites.

Les frames de type P

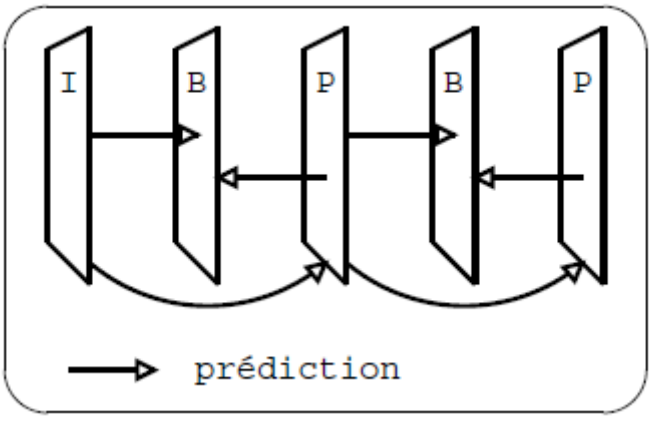
Elles sont codées en utilisant la redondance temporelle en les comparant avec le frame précédent.

Les frames de type B (Bidirectionnelle)

Elles sont codées en utilisant la redondance temporelle en les comparant avec la frame précédente et suivante (I, P) ou (P, P).

Un bloc est codé :

- En intra
- Interpolé entre P et I
- Par une frame précédente ou suivante I ou P ou dans les deux directions



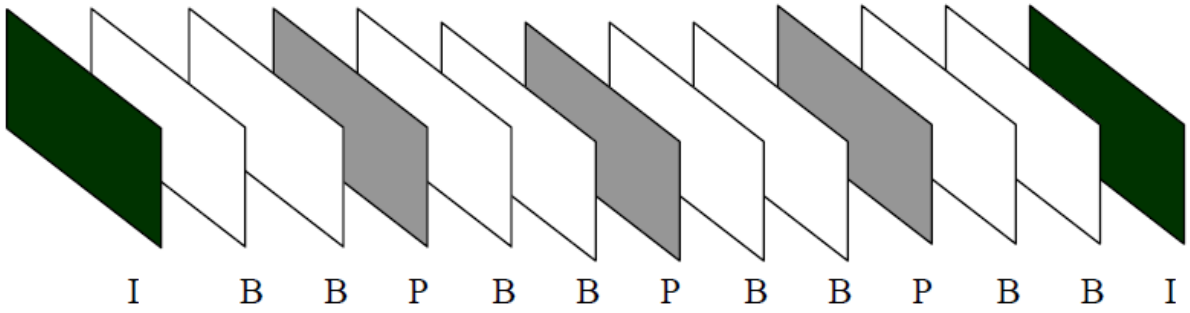


Figure 4.8. Frames de Type I, P et B

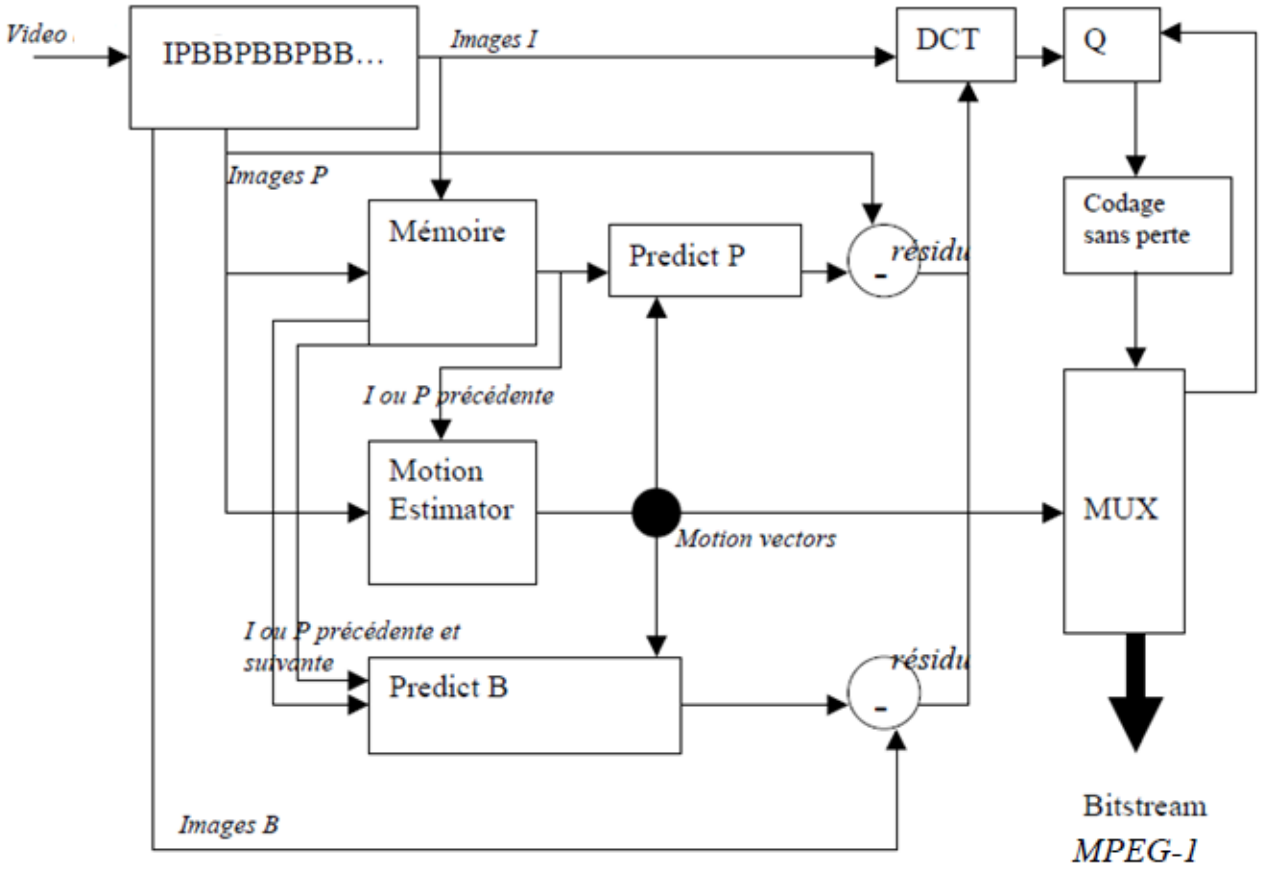


Figure 4.9. Codeur MPEG

Chapitre 5.
Compression de l'Audio

Chapitre 5.
Compression de l'Audio

Sommaire

5.1 Introduction

Description du format WAV

Un fichier audio non compressé est enregistré par défaut au format WAV (Microsoft).
Un son d'une minute peut occuper entre 644Ko (kilo-octets) et 27 Mo (mégaoctets).
La taille de ce fichier dépend de la fréquence d'échantillonnage, du type de son (mono ou stéréo) et du nombre de bits utilisés pour l'échantillonnage (8 ou 16 bits).

Structure d'un fichier wave :

Le taux d'échantillonnage peut varier entre 11kHz, 22kHZ et 44kHz, avec un échantillonnage sur 8 ou 16 bit.
Le volume d'un fichier wave stéréo pour 1 minute échantillonné à 44kHz en 16 bit est de : 60 (secondes) * 44100 (taux d'échantillonnage) * 2 (stéréo) * 2 (16 bits = 2 octets) = 10.56Mo.
Un morceau de musique comprimé en MP3 à 128kbps et à 44kHz a une taille de 3Mo environ (pour 3 à 4 minutes), soit environ 1Mo par minute.

Un fichier WAV 44Khz / 16bits / Stéréo est à 1375Kbps. Donc, un MP3 compressé à 128Kbps a un taux de compression de 11 pour 1.

Le principe de base de la compression audio repose sur les éléments suivants :

- On entend correctement les fréquences situées dans la gamme 2 kHz à 5 kHz. En effet, il faut moins de 5dB pour entendre les fréquences de cette bande
- alors qu'il faut plus de 20dB pour entendre les fréquences situées en dessous de 100Hz ou au-dessus de 10kHZ.
- Ces constatations peuvent être exploitées pour réduire la taille des fichiers. On peut par exemple décider que toutes les fréquences au-dessus de 15kHz seront supprimées.

Le principe de base de la compression audio repose sur les éléments suivants :

- Conserver uniquement les données *pertinentes du signal audio*
- *Utiliser les résultats de la psycho-acoustique qui indique que les fréquences d'un signal audio ne pas toutes perçues. L'œil* entend les fréquences situées dans la gamme 20 Hz à 20 kHz. Si un morceau contient des fréquences hors de cette gamme, il est supprimé sans perte de qualité audio.
- Créer le fichier audionumérique,

- Utiliser le principe des fréquences masquées.

A propos de MP3

MP3 est une technique de compression des formats numériques audio.
Le brevet en a été déposé par l'institut allemand de recherche Fraunhofer.

Dans le cadre du projet Eureka EU147 en 1987, les chercheurs ont travaillé en accord avec les normes établies par le MPEG (Moving Pictures Experts Group), un groupe d'experts au sein de l'ISO définissant le standard international pour la compression vidéo.

La 3ème version de cette technique est la MPEG-1 Layer 3 dont la variante spécifiquement audio est la MPEG-1 Audio Layer 3. Ce format a été standardisé par l'ISO fin 1992. En novembre 1996, un brevet est déposé aux USA.

L'institut allemand Fraunhofer détient 10 de 18 brevets MP3, Thomson Multimedia détient les 8 autres et gère les licences.

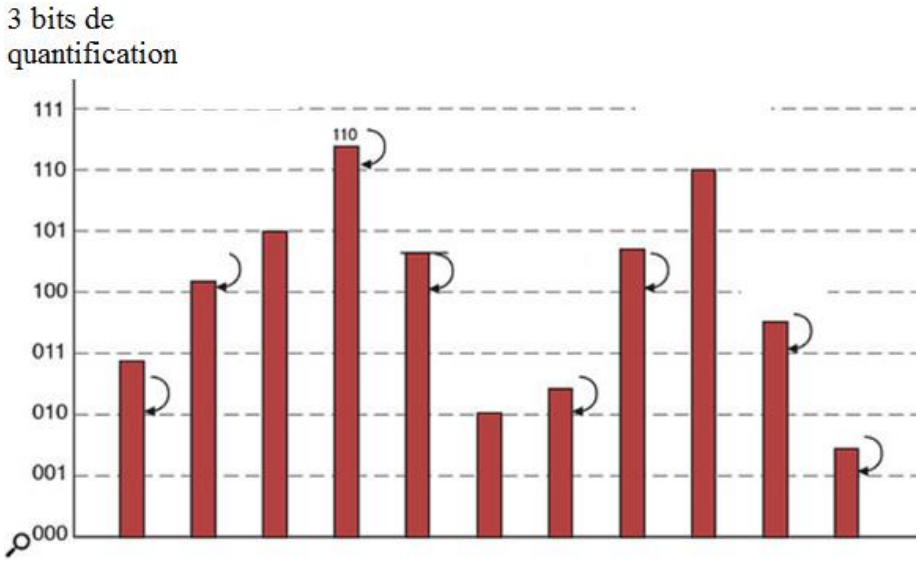


Figure 5.1. Erreurs de quantification

5.2 Les bases de l'audio

Si quelque chose vibre, ça fait vibrer l'air, l'air fait vibrer les os sensibles de l'oreille et cette sensation est interprétée par le cerveau comme du son.

Ce qui vibre rapidement est perçu comme un son à haute tonalité (aigu), et ce qui vibre lentement produit un son basse tonalité (grave).

Les grandes vibrations produisent un son fort, et ainsi de suite. Ces vibrations peuvent être représentées graphiquement par un signal

Dans le diagramme de la figure 5.2, l'amplitude des vibrations est mesurée sur l'axe vertical (y) et l'instant où sont prises les mesures est représenté sur l'axe horizontal (x).

Une note à tonalité **plus élevée** aurait plus de cycles sur une même durée plus de pics et de creux (Fréquence élevée).

Un son plus faible aurait des pics et des creux d'amplitude plus petite.

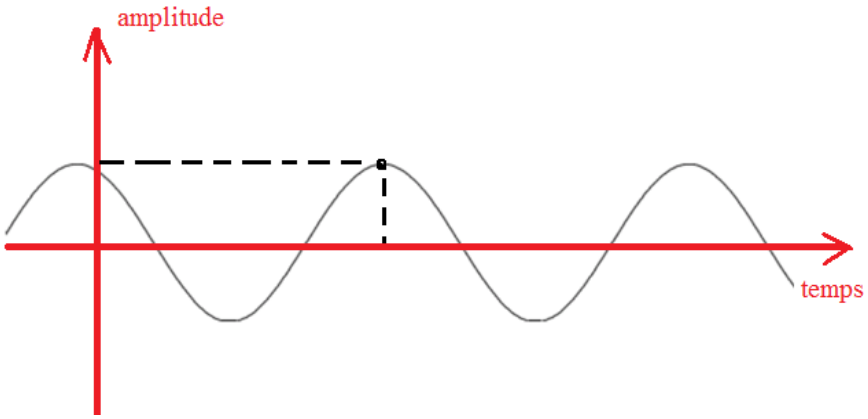


Figure 5.2. Représentation d'un exemple de signal audio

5.3 Numérisation du signal audio

5.3.1 La numérisation du son : échantillonnage et quantification

La numérisation est une opération qui consiste à transformer un signal analogique en un signal numérique.

Elle se déroule en deux étapes :

1- **Un découpage temporel et régulier du signal analogique (échantillonnage)**

La première étape consiste à découper temporellement le signal analogique en « tranches » : c'est l'opération dite d'échantillonnage.

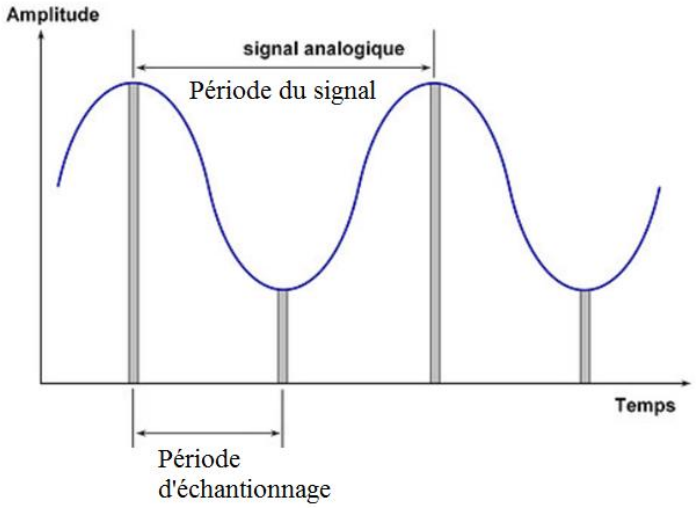


Figure 5.3. Echantillonnage

La fréquence d'échantillonnage c'est le nombre de fois où le son est échantillonné,

Le choix de la fréquence d'échantillonnage est important. Elle s'exprime en hertz et doit être en rapport avec la fréquence maximum à reproduire.

Pour pouvoir retransmettre intégralement l'étendue du spectre audible (donc jusqu'à 20 000 Hz), il faudra une fréquence d'échantillonnage au moins égale à 40000Hz.

On souhaite une bande passante moins étendue (parole, par exemple), on pourra opter pour des fréquences d'échantillonnage moins élevées, ce qui, corollaire immédiat, impliquera des fichiers moins volumineux (et donc des débits moins élevés).

Une évaluation de l'amplitude des échantillons par rapport à des références selon une échelle précisément établie (la quantification).

Interprétation : "Audio 16 bits 44,1 kHz" - signifie 65536 niveaux possibles mesurés 44100 fois par seconde.

Les erreurs de quantification

Affecter une valeur numérique à un signal analogique peut poser des problèmes. Etant donné qu'il n'y a qu'un certain nombre de valeurs (niveaux de quantification) que l'on peut assigner aux amplitudes, il y a un nombre considérable d'erreurs d'arrondi.

Evidemment, choisir un échantillonnage à 16 bits plutôt que 8 bits atténueront ces erreurs d'arrondi, puisqu'il y a plus de niveaux, comme il a été expliqué précédemment.

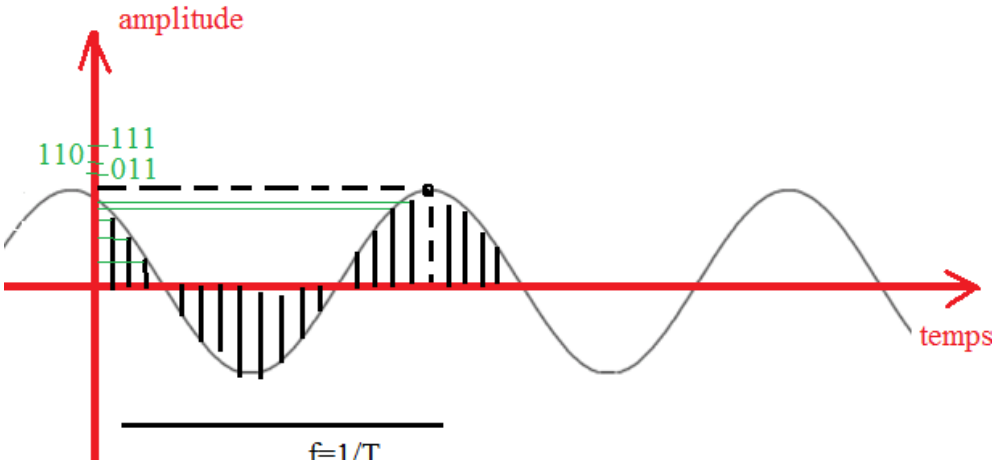


Figure 5.4. Processus de quantification

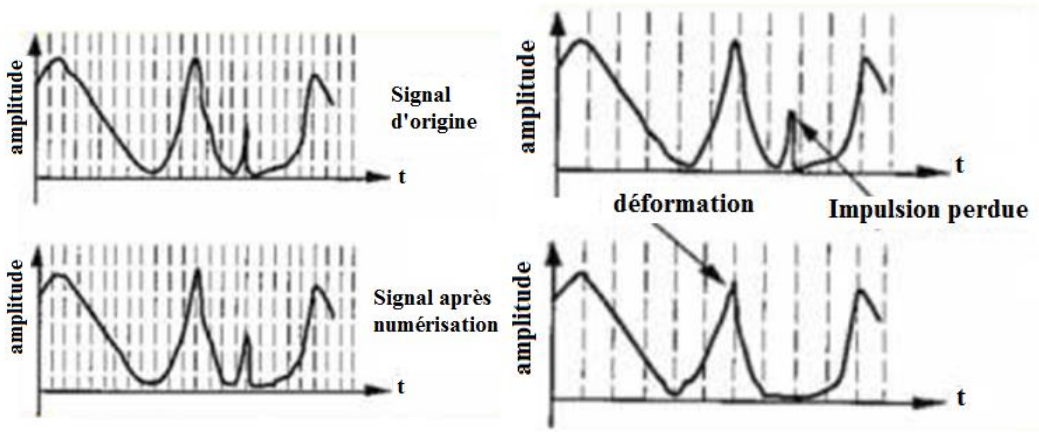


Figure 5.5 A gauche: Cas de fréquence d'échantillonnage suffisante : bonne restitution du signal. A droite : Cas de fréquence d'échantillonnage non suffisante : mauvaise restitution du signal

Normalisation

Dans le cadre de l'audio, il s'agit d'un processus à deux étapes qui vise à maximiser le volume de l'extrait sonore. Le processus de normalisation fait ce qui suit:

1) Trouve la plus haute amplitude dans le signal audio
2) Amplifie tout le signal de manière à ce que ladite amplitude ne dépasse pas le maximum qui puisse être stocké en numérique.

5.4 Compression du signal audio

5.4.1 Compression de données avec perte : Compression Audio MP3

Il y a deux étapes essentielles à la compression Audio MP3 (codec).

1) La compression de données sans perte. C'est le type de compression "zip" (Huffman), qui recherche des séquences afin de réduire la quantité de données qui doivent être stockées.

2) Les modèles psycho acoustiques, c'est le domaine principal pour compression Audio MP3.

C'est la partie où la compression s'effectue avec des pertes, durant laquelle un encodeur va jeter de l'information pour réduire la taille.

C'est fondé sur un modèle mathématique qui tente de décrire ce que l'oreille humaine entend réellement - i.e. avec l'objectif de se débarrasser de l'information qu'on ne peut pas entendre.

L'information exacte à éliminer dépend du codec utilisé.

Certains codecs sont faits pour enlever certaines fréquences pour que la compression soit meilleure pour les voix.

Divers modèles ont étés formulés au fil des années afin de réduire la taille des fichiers audio.

Mais le plus marquant ces dernières années est sans aucun doute le modèle psycho acoustique utilisé dans la compression mpeg1 layer 3 (mp3).

5.4.2 Les étapes de la compression MP3

Le signal est découpé en petites sections appelées frames

Ces chiffres sont ensuite comparés à des tables de données propres au Codec, qui contiennent des informations sur les modèles psycho-acoustiques.

Dans le codec mp3, ces modèles sont très avancés, et une grosse partie de la modélisation se fonde sur un principe du nom de Masking.

Toute information qui correspond au modèle psycho-acoustique est conservée et le reste est rejeté. Ce sont les bases de la compression Audio MP3.

En fonction du bit-rate (nombre de bits par seconde), le codec utilise la taille allouée pour stocker ces données.

Ceci étant fait, le résultat passe par la compression sans pertes Huffman, qui réduit encore la taille de 10%.

La technique principale du codec mp3 pour enlever de l'information est en détectant quels sons ne sont pas détectables, ou 'masqués', et qui du coup ne peuvent pas être entendus.

Ces sons sont ensuite enlevés sans perte audible dans le signal.

Le modèle Psycho-acoustique et sons masqués :

L'oreille humaine peut percevoir en théorie toutes les fréquences comprises entre 20 et 20 000Hz

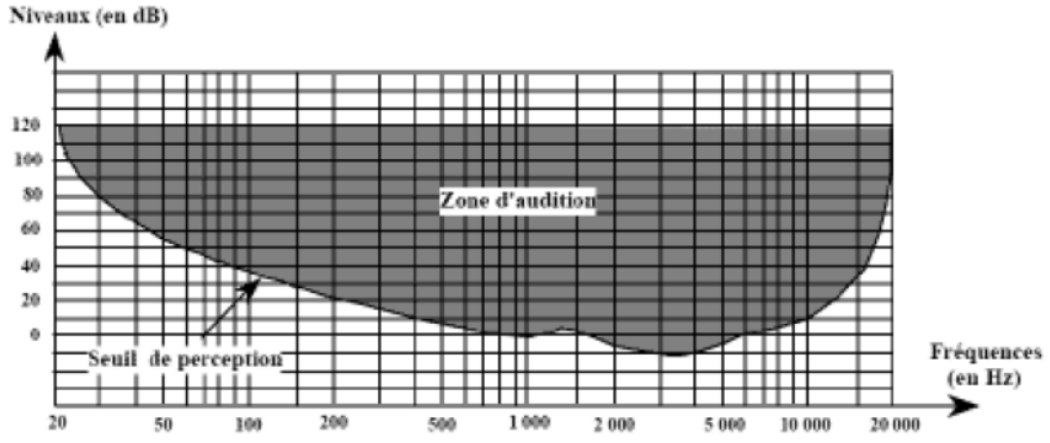


Figure 5.6. Modèle psychoacoustique

Il y a deux principaux types d'effets de masquage - le masquage simultané et le masquage temporel.

Masquage fréquentiel

On parle de masquage fréquentiel lorsqu'un son faible - qui serait parfaitement audible s'il était émis seul - est masqué parce qu'il se trouve accompagné simultanément par un son fort de fréquence voisine (son « masquant).

Il est inutile de coder les signaux qui sont situés en dessous. Cette courbe de « masquage » (les variations du seuil d'audition donc) variant à chaque instant en fonction du contenu spectral du signal, c'est donc une véritable analyse en temps réel qui doit être réalisée par les circuits de codage.

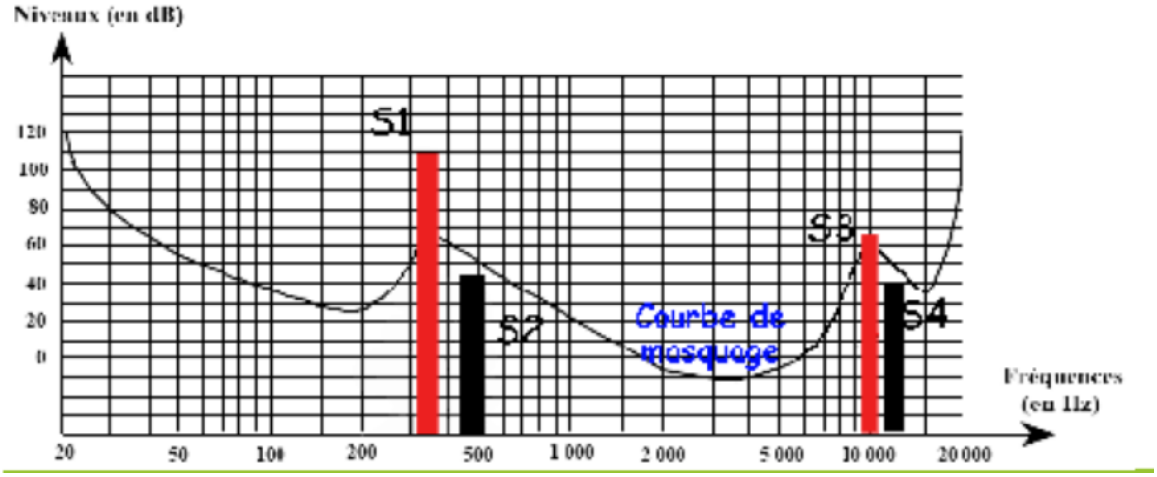


Figure 5.7. Masquage fréquentiel

Le masquage temporel

Le terme de masquage temporel fait référence au masquage réalisé après l'apparition d'un son masquant de forte intensité, mais également, au masquage avant la perception de ce son.

Le terme de temporel indique que le son masquant et le son masqué sont décalés dans le temps par opposition au masquage fréquentiel qui ne concerne que des sons simultanément présents.

Le pré-masquage est très court et ne dure que quelques millisecondes (les sons ayant une durée trop courte ne seront pas perçus.

Le post-masquage est beaucoup plus long 100-200 ms et dépend du son masquant.

Après un son fort, l'oreille ne pourra percevoir un son plus faible qu'au terme de ce laps de temps.

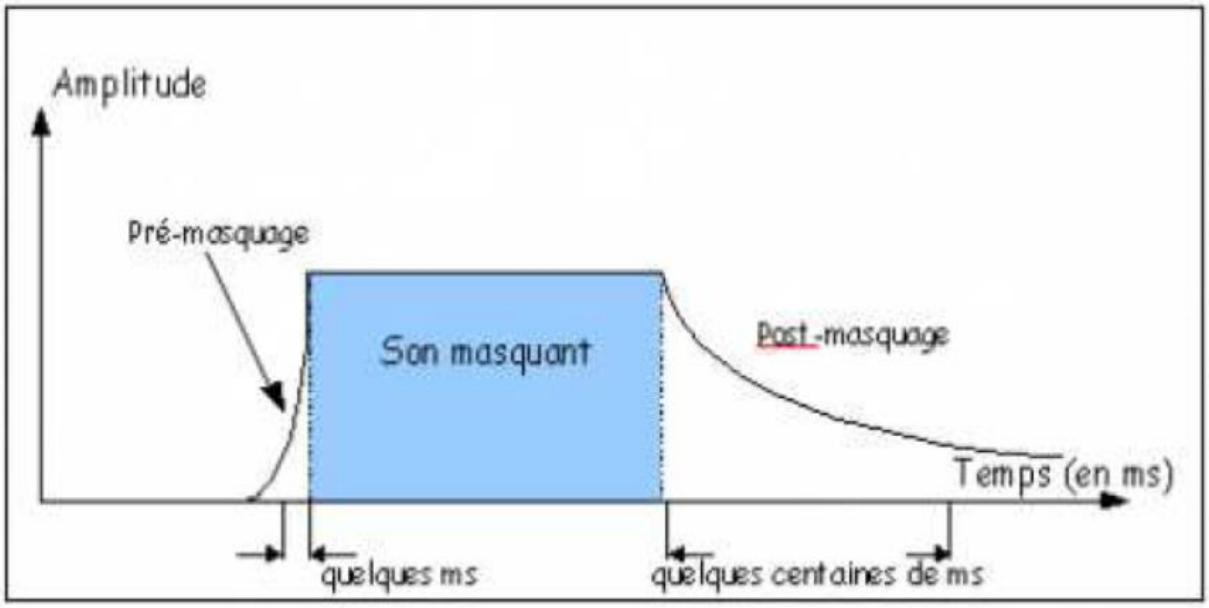


Figure 5.8. Le masquage temporel

EXERCICES DE TRAVAUX DIRIGES ET PRATIQUES

Exercice 1 :

Donnez le codage LZW pour la chaine de caractères suivante et calculez le taux de compression :

aaaaaaababababababaaaaaacacaeaeaeaea

Le dictionnaire est composé du jeu de caractères :

a codé 0, **b** codé 1, **c** codé 2, **e** codé 3.

Exercice 2 :

Le codage avec l'algorithme LZW a donné le codage suivant :

65 95 65 67 97 65. Sachant que le code de code a=65, b=66,...., code(z)=90, code de derniers caractères du dictionnaire :{ | } ~ sont 91, 92, 93, 94. Donnez la séquence de caractères qui a été codée.

Exercice 3 :

Donner le codage de Huffman pour la chaine de caractères suivante et calculez le taux de compression:

Aaaaaaaaaaaaaaababababababaaaaaacacaeaeaeaea

Exercice 4:

Soit la matrice 8x8 (ci-dessous) de valeurs transformées et quantifiées. Donner le codage de Huffman associé et calculer le taux de compression pour les parcours possibles (zigzag et lignes). Que peut-on conclure ?

50	0	0	-3	0	0	0	0
-9	0	2	0	0	0	0	0
0	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	-1	0	0	0	0	0
0	0	0	0	0	0	0	0
-1	0	0	0	0	0	0	0
-1	0	0	0	0	0	0	0

Exercice 5 :

Soit la liste suivante de frames codées par un codec MPEG :

$I_1 P_2 P_3 P_4 I_5 P_6 P_7 P_8 I_9 P_{10} P_{11} P_{12} I_{13} P_{14} P_{15} P_{16} I_{17} P_{18} P_{19} P_{20}$

Le décodeur fonctionne avec deux threads s'exécutant en parallèle : l'un pour le décodage, l'autre pour la présentation (visualisation).

Le tableau suivant illustre un état du diagramme du décodeur donnant les frames reçues, décodées, en décodage, présentées à la visualisation.

Un buffer (playout buffer) de 10 frames est considéré dans ce codec. Le décodage commence donc au 11ème cycle.

1- Complétez la table
2- A quel cycle le décodeur commence et à quel cycle finit-il ?
3- A quel cycle la visualisation commence et à quel cycle finit-elle ?

Cycle	**Frames reçues**	**Décodeur (thread)**	**Présentation (thread)**
1	I_1		
2	$I_1 P_2$		
3	$I_1 P_2 P_3$		
4	$I_1 P_2 P_3 P_4$		
5	$I_1 P_2 P_3 P_4 I_5$		
6	$I_1 P_2 P_3 P_4 I_5 P_6$		
7	$I_1 P_2 P_3 P_4 I_5 P_6 P_7$		
8	$I_1 P_2 P_3 P_4 I_5 P_6 P_7 P_8$		
9	$I_1 P_2 P_3 P_4 I_5 P_6 P_7 P_8 I_9$		
10	$I_1 P_2 P_3 P_4 I_5 P_6 P_7 P_8 I_9 P_{10}$		
11	$I_1 P_2 P_3 P_4 I_5 P_6 P_7 P_8 I_9 P_{10} P_{11}$	I_1	
12	$I_1 P_2 P_3 P_4 I_5 P_6 P_7 P_8 I_9 P_{10} P_{11} P_{12}$	$I_1 P_2$	I_1

Exercice 6 :

Soit la séquence de frames suivante :

$I_1\ B_2\ B_3\ P_4\ B_5\ B_6\ B_7\ B_8\ I_9\ B_{10}\ B_{11}\ P_{12}\ B_{13}\ B_{14}\ I_{15}\ B_{16}\ B_{17}\ P_{18}\ B_{19}\ P_{20}$

1- Quel est le retard (delay) que doit faire le codeur avant de commencer à transmettre les frames codés, et quelle est la séquence envoyée ?
2- Refaire les trois questions de l'exercice 1 sur la séquence reçue par le décodeur en prenant une taille de 10 frames pour le buffer playout.

Exercice 7 :

Une vidéo à résolution de 352x288 et à 30 frames par seconde est compressée en utilisant les frames de type I et P. Le calcul de chaque vecteur de mouvement nécessite en moyenne 3ms. Entre 2 frames de type I on insère 20 frames de type P.

1. Quel est le temps (en secondes) qui sera consommé dans le calcul des vecteurs de mouvement pour la compression de 2 secondes de vidéo ?
2. Peut-on utiliser ceci pour la codage et distribution en temps réel ?
3. Que peut-on faire pour que le codeur calcule plus vite les vecteurs de mouvements?

Exercice 8 :

Une vidéo à résolution de 176x144 et à 30 frames par seconde est compressée en utilisant les frames de type I, B et P. Le calcul de chaque vecteur de mouvement nécessite en moyenne 3ms. Entre 2 frames de type I on insère 8 frames de type P et 2 frames de type B.

Quel est le temps (en secondes) qui sera consommé dans le calcul des vecteurs de mouvement pour la compression de 5 secondes de vidéo ?

Exercice 9 :

Il s'agit d'implémenter la compression d'Huffman en réalisant les opérations suivantes :

- Lire une chaine de caractères
- Compter le nombre d'occurrences de chaque caractère
- Trier les occurrences obtenues
- Construire et visualisez la construction de l'arbre de Huffman
- Trouver pour chaque élément le code correspondant.

Un exemple de cette implémentation (sans production de la table de codage) est donné par :

https://people.ok.ubc.ca/ylucet/DS/Huffman.html

Exercice 10.

Lire un fichier texte contenant une matrice 8x8 et donnez le codage de Huffman associé en utilisant le résultat obtenu par l'exercice 1. La matrice est de la forme suivante indiquée par la figure :

50	0	0	-3	0	0	0	0
-9	0	2	0	0	0	0	0
0	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0
0	0	-1	0	0	0	0	0
0	0	0	0	0	0	0	0
-1	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Figure . Exemple de matrice à coder avec Huffman

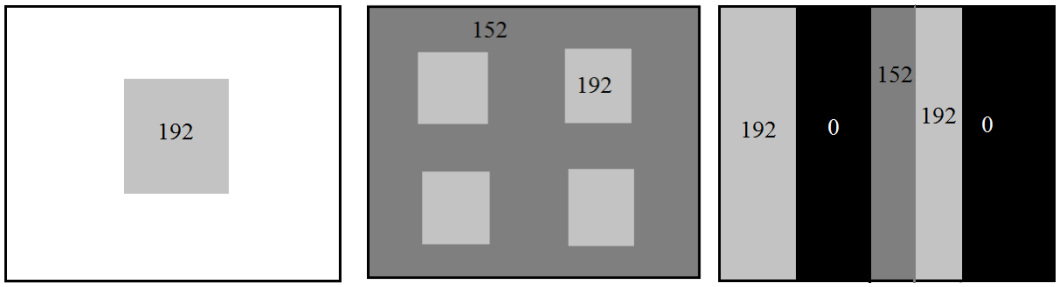
Exercice 11

Le programme qui vous été remis permet de lire une image sous forme de matrice 8x8 (fichier texte), calculer la DCT, quantifier la DCT, coder en ZIGZAG de la matrice quantifiée, appliquer la dé-quantification et DCT inverse.

Il est demandé de réaliser les tests suivants et de répondre aux questions posées :

1- Lire une image dont les pixels sont rangés dans le fichier "image.txt" joint.
2- Calculer la DCT, et la matrice DCT quantifiée avec facteur de qualité égal à 5.
3- Commentez le résultat obtenu.
4- Ayant obtenu le codage de Huffman et la codification dans le fichier codage.txt, donnez le taux de compression d'une image de même nature de taille 512x512
5- Calculer la DCT inverse, et calculer la matrice 8x8 d'erreurs d'intensités des pixels décodés.
6- Quel est l'effet d'augmenter le facteur de qualité à 10 puis 20 sur le taux de compression. Expliquez.

Il est demandé de considérer 3 types d'images :



Exercice 12

Un programme de compression/décompression JPEG est supposé disponible pour utilisation en TP.

Ce programme permet de lire une image sous forme de matrice 8x8 (fichier texte), calculer la DCT, quantifier la DCT, coder en ZIGZAG de la matrice quantifiée, appliquer la dé-quantification et DCT inverse.

Il est demandé de réaliser les tests suivants et de répondre aux questions posées :

7- Lire une image dont les pixels sont rangés dans le fichier "image.txt" joint.
8- Calculer la DCT, et la matrice DCT quantifiée avec facteur de qualité égal à 5.
9- Commentez le résultat obtenu.
10- Ayant obtenu le codage de Huffman et la codification dans le fichier codage.txt, donnez le taux de compression d'une image de même nature de taille 512x512
11- Calculer la DCT inverse, et calculer la matrice 8x8 d'erreurs d'intensités des pixels décodés et donc visualisées. Donnez cette matrice.
12- Quel est l'effet d'augmenter le facteur de qualité à 10 puis 20 sur le taux de compression. Expliquez.
13- A l'aide de Processing.js, visualiser une telle image (avec 128x128 pixels, chacun des blocs ayant le contenu image.txt) et visualiser l'image décompressée pour les différents facteurs de qualité.

Exercice 13 :

Nous disposons d'une séquence d'images vidéo Im_1, Im_2, Im_3, ..., Im_n.

Localisez par programmation à l'aide de la souris un rectangle englobant un piéton comme indiqué par la figure (a)(click sur les deux extrémités le définissant).

(a) (b)

Figure . Deux images prises d'une vidéo

Sur l'image suivante (frame suivante dans la vidéo), le piéton pourra se déplacer dans la même zone indiquée en rectangle rouge avec la possibilité d'aller dans les 4 sens. La zone de recherche du piéton est localisée par le rectangle de couleur verte avec (dx, dy)=(5 , 5pixels) comme différence du rectangle rouge.

Implémentez l'algorithme qui permet de retrouver dans la zone encadrée en vert la zone rectangulaire la plus similaire à la zone encadrée en rouge (On travaillera sur la composante luminance).

Calculez la différence en luminance des deux zones similaires.

Exercice 14 :

Installez VirtualDub qui est un utilitaire de capture et de traitement de vidéo et principalement orienté vers le traitement de fichiers AVI, pouvant contenir différents codecs audio (MP3,..) et vidéo (MPEG-1, MPEG-2, MPEG-4, DivX, Xvid, H264,..)

1. Charger et lire une vidéo au format avi
2. Exporter la vidéo sous format d'images bmp, et constater les différences entre frames successives
3. Sélectionner une partie de frames et sauvegarder sous forme de vidéo.
4. La sélection d'un segment de la vidéo peut se faire aussi après marquage à l'aide des boutons  .

Exercice 15 :

Refaire les questions 1, 2 de l'exercice 1 pour une image GIF animée.
Construire votre GIF animée.

Exercice 16 :

Lire une vidéo :

1. En appliquant quelques filtres :

- Niveau de gris
- Rotation
- Perspective

2. Changer le paramètre FrameRate
3. Insérant un fichier audio et sauvegarder la vidéo avec le son. Refaire le sens inverse pour supprimer le son de la vidéo

Exercice 17 :

1. Montrer que sans compression, une seconde d'un son occupe 1.4 Mb avec la qualité CD alors qu'en utilisant MP3, la taille décroît d'un facteur de 11 (128Kb).

2. Un fichier audio mp3 de durée 6mn 36s a été encodé avec la qualité CD stéréo (44,1KHz, 44100 échantillons par sec et par canal). Le fichier est de taille 6 332 562 octets. (6.03MO)

- Quel est la taille initiale du fichier avant compression
- Quel est le taux de compression
- Le débit de MPEG-1 Layer 3 (MP3) doit être entre 112 et 128 kbits/sec. Le fichier est-il conforme à cette indication ?

Solution :

Sans utiliser mp3 **:**
44100 échantillons / sec
44100*16*2 BITS = 1411200 bits = 1,34 Mb
Il faut environ 10 Mo pour stocker une minute de musique.

En utilisant mp3 : 128Kb

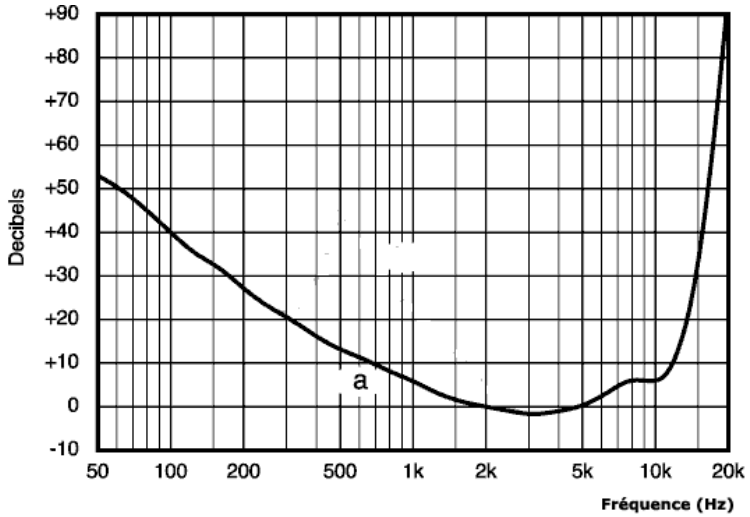
Taille du fichier est : 6 332 562 octets
Durée = 396 sec soit 396 * 44 100*2 échantillons, 34927200 échantillons, soit 558835200 bits, 532,94677734375 Mb = 66,61834716796875 MO
Taux de compression = 11
6.03 MO = 48.24 Mb = 49397,76 Kb
49397,76/396= 124kbs. Oui conforme à MP3

Exercice 18 :

Soit la courbe de masquage ci-dessous.

Quelle est la nouvelle allure de cette courbe suite à la présence des signaux : (500Hz, 20dB), (1KHz, 30dB), Expliquer ce changement d'allure.

Si un son de (520Hz, 18dB) est reçu au même temps, sera-t-il audible ? Justifiez



Références

[1] Parag Havaldar and Gérard Medioni.
Multimedia Systems: Algorithms, Standards, and Industry Practices, Edition Course Technology,
USA, 2010

[2] Ze-Nian Li, Mark S. Drew, Fundamentals of Multimedia, edition Pearson Prentice Hall, 2004

[3] Pierre COURTELLEMONT, Bases du Multimedia, Support de cours, Université La Rochelle, France

[4] [Ziv&Lempel77]. Jacob Ziv and Abraham Lempel. A universal algorithm for sequential data compression. IEEE Transactions on Information Theory, 23: 337–343, 1977.