



# Compression de données



## **Importance considérable dans des domaines variés :**

télévision, musique, télédétection, imagerie médicale,...

**Le graphiste utilise depuis longtemps (parfois sans le savoir) diverses méthodes de compression implémentées dans les logiciels du commerce.**

**On se propose ici, en se limitant au cas particulier des textes ou images fixes, de faire l'inventaire des méthodes disponibles, et de comprendre le principe.**

# Méthodes de compression

- **Les méthodes réversibles (sans perte).**
  - ✓ Codage statistique (algorithmes de Huffman et de Shannon-Fano)
  - ✓ Méthodes arithmétiques
  - ✓ Méthodes dites « à dictionnaire » (algorithme LZW)
- **Les méthodes irréversibles (avec pertes).**
  - ✓ Compression JPEG
  - ✓ Compression JPEG 2000



**Définition :** Nous dirons que nous avons compressé un fichier si nous parvenons à réduire le nombre de digits binaires nécessaires pour l'enregistrer.

On mesure l'efficacité de la compression par le **taux de compression :**

$$\sigma = \frac{\text{Nombre de digits binaires utilisés par le document original}}{\text{Nombre de digits binaires utilisés pour le document compressée}}$$



## Méthodes de compression sans perte

Un exemple simple : Supposons que nous devions essayer de compresser la série d'octets suivante :

```
0001 0100 0011 1111 0101 0101 0101 0101 0101 0101
0101 0101 1110 1111 0000 1111 1111 1111 1111 1111
1111 1111 1111 1111 1111 1111 0110 0111 0111 0000
0111 0000 1010 1111 0000 0000 0001 1111 0001 1111
0001 1111
```

Cette série compte 21 octets. (Vous pouvez l'interpréter comme une suite de caractères ou comme les niveaux de gris d'une suite de pixels, cela n'a pas d'importance pour ce qui nous préoccupe ici).



Commençons par écrire un octet de **signalisation**, pour le repérer, nous l'écrivons en gras.

**0000 0010**

Le premier bit indique si l'octet de donnée qui suit se répète, si c'est le cas le bit sera mis à 1, si ce n'est pas le cas il sera mis à 0.

**Si premier bit = 0**

Les 7 bits qui suivent indiqueront le nombre d'octets sans répétition

**Si premier bit = 1**

Les 7 bits qui suivent indiqueront le nombre de répétitions



## Résultat :

**0000 0010** 0001 0100 0011 1111 **1000 0100** 0101 0101 **0000 0010**  
1110 1111 0000 1111 **1000 0101** 1111 1111 **0000 0001** 0110 0111  
**1000 0010** 0111 0000 **0000 0010** 1010 1111 0000 0000 **1000 0011**  
0001 1111

Cette chaîne fait 19 octets soit 2 de moins que la chaîne initiale,  
nous avons un taux de compression de  $21/19 = 1,1$

Le décodage ne pose aucun problème à condition que le décodeur soit informé de la méthode utilisée, il interprétera dans ce cas le premier octet comme un octet de signalisation et tout ira bien...



## Méthode statistique

Coder toutes les valeurs que nous avons à stocker avec le même nombre de bits, Est-ce la meilleure solution ?

La théorie nous apprend que non.

**Idée** : Utiliser des codes plus courts pour des valeurs fréquentes et de réserver des codes plus longs pour les valeurs moins fréquentes.

Exemple de l'histogramme pour l'image

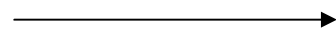
=> On va donc s'intéresser aux **VLC (Variable Length Code)**, on dit aussi au codage **entropique**.



## Exemple

Supposons que nous ayons à traiter un message qui ne contient que 4 caractères que nous nommerons A, B, C et D. et que les fréquences de ces caractères dans notre message soient les suivantes :

A : 60 %  
B : 30 %  
C : 5 %  
D : 5 %



On peut imaginer le code suivant :

A : 0  
B : 11  
C : 01  
D : 10

Il nous permettra assurément de gagner de la place mais si nous recevons le message suivant : **000110**, comment l'interpréter ?  
3A, B, A ? 2A, C, D ? Ça ne marche pas ! Nous venons de découvrir le problème de la synchronisation.

### Comment le résoudre ?

Utiliser des caractères séparateurs ? c'est contradictoire avec l'objectif de compression.



## La solution c'est le VLC préfixé.

**Définition :** Un code est préfixé s'il n'est le début d'aucun autre

Revenons à notre exemple et essayons de corriger.

A : 0 B : 10 C : 110 D : 111 Ça marche !

Est-ce la solution optimale ?

Comme nous l'avons obtenue par tâtonnement, on n'en sait rien mais rassurez-vous la théorie permet d'affirmer que oui.

Nous allons maintenant prendre un exemple un peu plus complexe et présenter une méthode permettant de déterminer un **VLC efficace** (même s'il n'est pas toujours optimal).



## Algorithme de Shanon-Fano.

Supposons que nous voulions transmettre le message suivant :

### LE PRESIDENT EST ENTRE DANS LA SALLE

Il compte 36 caractères et occupe dans un logiciel comme WordPerfect 36 octets. Nous allons compter les occurrences des différents caractères de l'alphabet utilisés et les classer par fréquences décroissantes, nous obtenons la liste suivante :

E : 7

Espace : 6

L : 4 Nous allons maintenant regrouper les caractères en deux

S : 4 groupes dont les fréquences d'apparition sont aussi

N : 3 proches que possibles puis diviser chacun de ces groupes

T : 3 de la même façon jusqu'à parvenir à chacune des

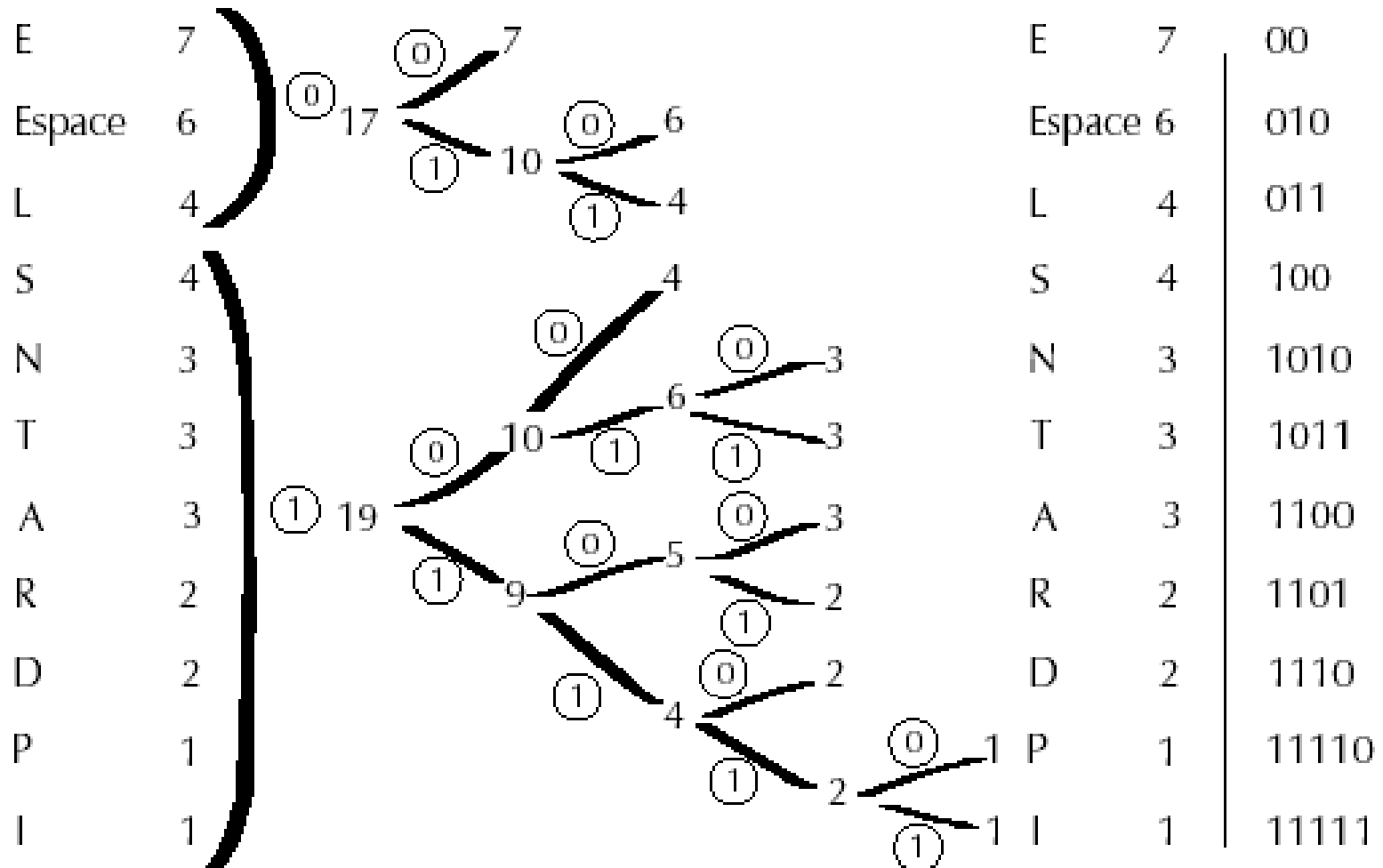
A : 3 fréquences de départ.

R : 2

D : 2

P : 1

I : 1





- On peut vérifier que l'on obtient bien un VLC préfixé.
- Mesurons son efficacité.

Nous sommes parti d'un message de 36 caractères soit  $36 \times 8 = 288$  bits

si l'on avait codé chaque caractère sur 4 bits (c'était suffisant), on utilisait 144 bits.

En utilisant le codage que nous venons de réaliser nous utiliserons :

$$(2 \times 7) + (3 \times 6) + (3 \times 4) + (3 \times 4) + (4 \times 3) + (4 \times 3) + (4 \times 3) + (4 \times 2) + (4 \times 2) + (5 \times 1) + (5 \times 1) = 14 + 18 + 12 + 12 + 12 + 12 + 12 + 8 + 8 + 5 + 5 = 118 \text{ bits}$$

Le taux de compression par rapport à la représentation sur 4 bits est

$$s = 144/118 = 1,22.$$



## Algorithme de Huffman.

**Principe :** on représente un élément  $i$  par une séquence de bits de longueur inversement proportionnelle à la probabilité d'apparition  $p(i)$ .

Optimal pour les codes à nombre de bits entiers.

Le problème est de définir un code à longueur variable dont aucun élément ne soit le début d'un autre (code préfixé).

La méthode repose sur la construction d'un arbre basé sur les probabilités d'apparition des éléments.

### Exemple :

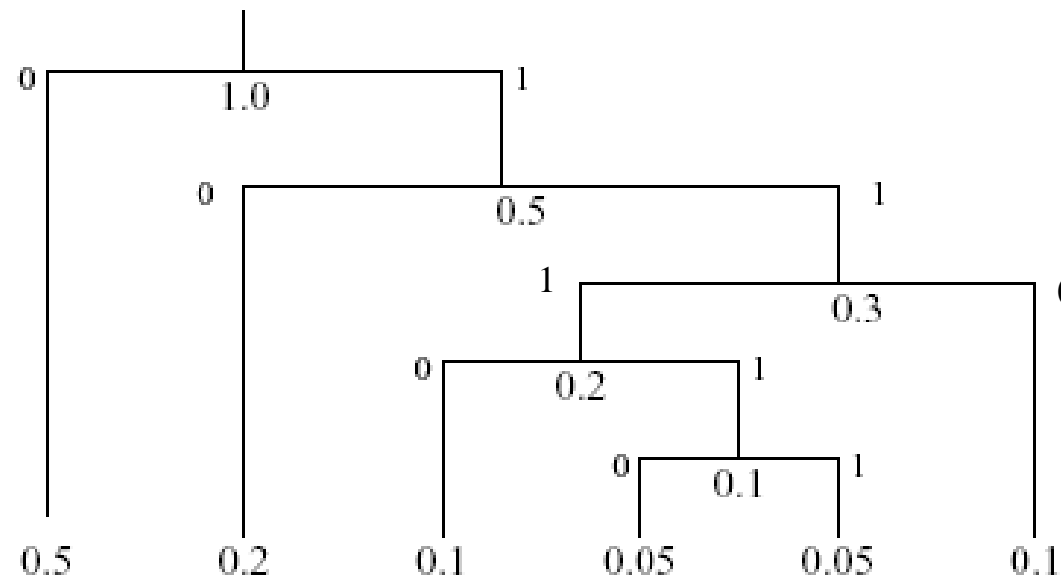
Soient à coder les éléments suivants, avec les probabilités suivantes :

$i$	1	2	3	4	5	6
$x_i$	a	b	c	d	e	f
$p(i)$	0.5	0.2	0.1	0.05	0.05	0.1



## Principe de la construction de l'arbre

1. Les feuilles sont des noeuds libres
2. choisir les deux noeuds libres de probabilités les plus faibles
3. créer avec un noeud père de poids somme des poids des noeuds fils.
4. **Associer 0 à la branche de plus petite probabilité et 1 pour l'autre**
5. Le père est un noeud libre et les fils ne le sont plus.
6. Tant qu'il y a plus d'un noeud libre, répéter 2–5



$x_i$	1	a	b	c	d	e	f
$x_i$	a	b	c	d	e	f	
code	0	10	1110	11110	11111	110	

On peut augmenter la compression en travaillant sur des suites de caractères.

**NB :** nécessite la connaissance ou l'envoi de la table au décodeur.

# Codage arithmétique

- Contrairement aux méthodes statistiques (à un message de haute fréquence on convertissait sa représentation avec une notation plus courte, et inversement pour l'information rare), on représente ici un flux d'informations à l'aide d'un intervalle numérique.
- Un message est représenté dans un intervalle de nombres réels compris entre  $[0, 1[$
- Plus le message devient important, plus l'intervalle nécessaire à sa représentation sera petit



# Codage arithmétique

Valeur = AncienneLimiteHaute - AncienneLimiteBasse

NouvelleLimiteHaute =

AncienneLimiteBasse + Valeur \* Valeur\_Haute(c)

NouvelleLimiteBasse =

AncienneLimiteBasse + Valeur \* Valeur\_basse(c)

# Traitement d'un exemple

- Message : **BRACADABRA**
- Chaque caractère aura un intervalle de représentation. L'affectation de l'intervalle n'a pas d'influence sur la compression / décompression du message
  - Mais, **ATTENTION** ! : à la décompression utiliser la même table ayant servi e à la compression

Caractère	Probabilité	Intervalle
A	4/10	[ 0.0- 0.4[
B	2/10	[ 0.4-0.6 [
C	1/10	[ 0.6-0.7 [
D	1/10	[ 0.7- 0.8[
R	2/10	[ 0.8-1.0 [



# Comment affecter le code?

Caractère	Probabilité	Intervalle
A	4/10	[ 0.0- 0.4[
B	2/10	[ 0.4-0.6 [
C	1/10	[ 0.6-0.7 [
D	1/10	[ 0.7- 0.8[
R	2/10	[ 0.8-1.0 [

Valeur =

$AncienneLimiteHaute - AncienneLimiteBasse$

NouvelleLimiteHaute=

$AncienneLimiteBasse + Valeur * Valeur\_Haute(c)$

NouvelleLimiteBasse=

$AncienneLimiteBasse + Valeur * Valeur\_basse(c)$

Nouveau caractère	Limite basse	Limite haute
	0.0	1.0
B	0.4	0.6
R	0.56	0.60
A	0.560	0.576
C	0.5696	0.5712
A	0.56960	0.57024
D		
A		
B		
R		
A	0.570023360	

La dernière valeur de la limite basse codera de façon unique le message « BRACADABRA »





# Comment décoder ?

Caractère	Probabilité	Intervalle
A	4/10	[ 0.0- 0.4[
B	2/10	[ 0.4-0.6 [
C	1/10	[ 0.6-0.7 [
D	1/10	[ 0.7- 0.8[
R	2/10	[ 0.8-1.0 [

valeur	Intervalle	caractère
0.5700623360	[ 0.4-0.6 [	B
0.85031168	[ 0.8-1.0 [	R
0.2515584	[ 0.0- 0.4[	A
0.628896	[ 0.6-0.7 [	C
0.28896	[ 0.0- 0.4[	A
0.7224	[ 0.7- 0.8[	D
0.224	[ 0.0- 0.4[	A
0.56	[ 0.4-0.6 [	B
0.8	[ 0.8-1.0 [	R
0.0	[ 0.0- 0.4[	A

Valeur =

$$\text{Valeur} - \text{Valeur\_basse}(c)$$

Valeur =

$$\text{Valeur} / (\text{Valeur\_Haute}(c) - \text{Valeur\_Basse}(c))$$



# Méthode LZW

## DE « Lempel-Ziv\_Welch »

# Compression avec perte

**Le principe :** nous acceptons une dégradation de l'image décompressée - indiscernable à l'œil ou suffisamment faible pour être acceptable - en contrepartie d'un taux de compression beaucoup plus intéressant.

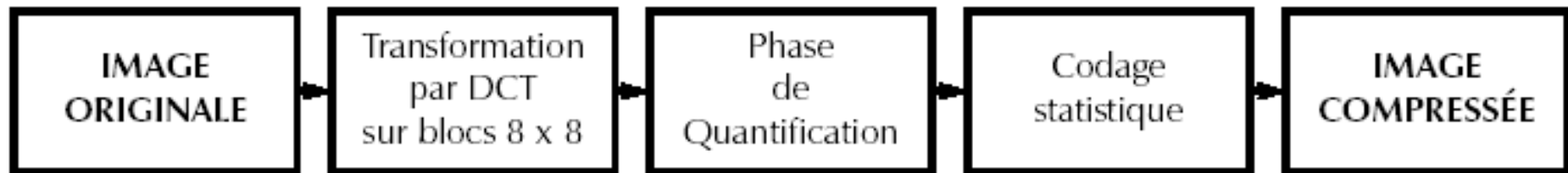
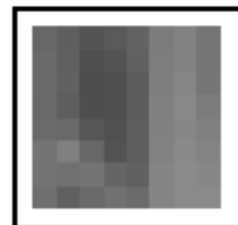


Image originale



Bloc 8x8



DCT (DC norm. 255)



DCT (coef. normal. 255)

# Principe de compression JPEG (Joint Photographic Expert Group)



Image originale

255	255	255	0	0	0	0	0
36	255	100	100	36	36	36	36
73	255	100	73	100	73	73	73
109	255	100	100	100	100	100	109
146	146	100	146	100	146	146	146
182	182	100	182	100	100	100	182
218	218	218	218	100	218	218	218
255	255	255	255	100	100	100	255

La matrice suivante représente la composante rouge de l'image codée en 16 millions de couleurs (soit 256 nuances de rouge).



La première étape consiste à soustraire 128 (ce qui correspond au nombre de couleurs rouge divisé par 2) à chaque valeur puis à appliquer la DCT. Nous obtenons le résultat suivant.

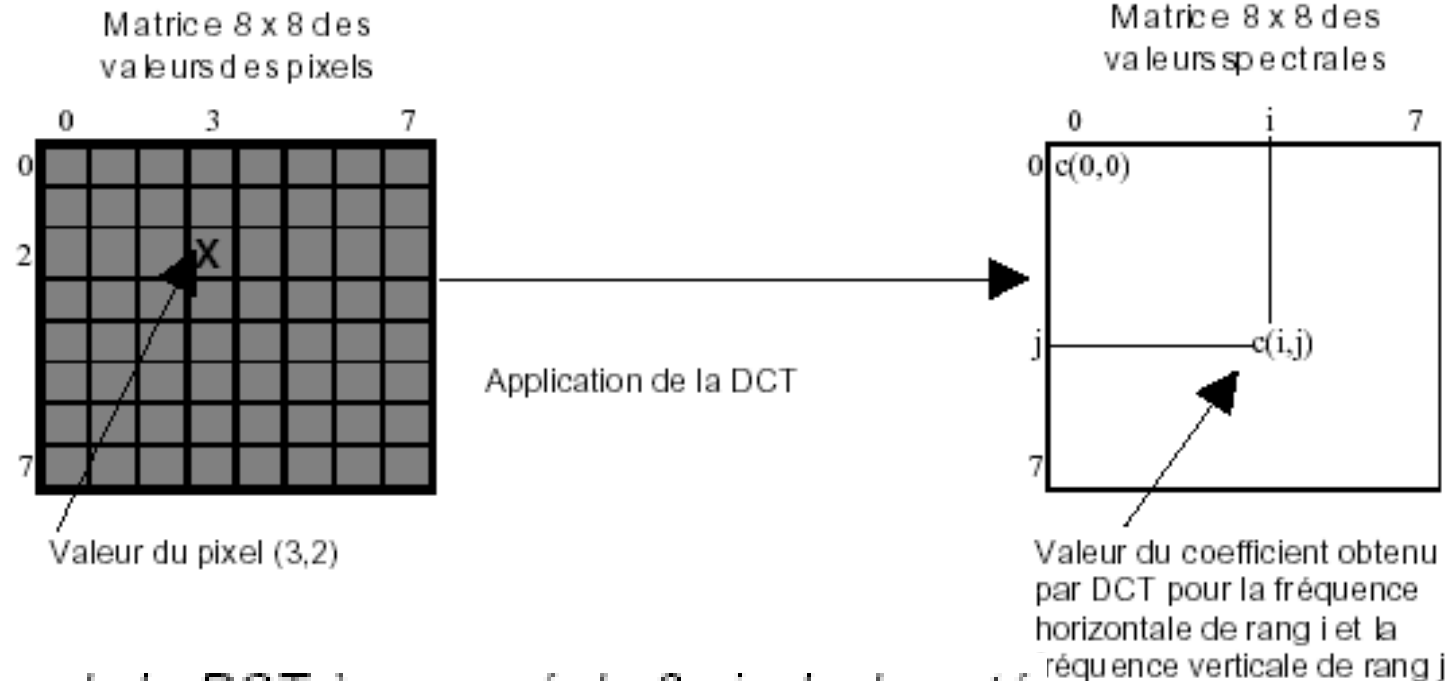
50	258	135	-120	-73	-76	-57	-103
-319	148	-9	28	-124	-101	-55	63
68	170	52	-89	-15	31	80	35
-21	25	40	44	16	68	60	30
22	81	46	-9	14	14	55	47
66	33	39	33	-54	4	29	38
-31	68	30	-10	54	-29	13	33
41	-30	0	34	-31	37	36	29

**Remarque :**

Les coefficients possédant les valeurs absolues les plus fortes se trouvent en haut à gauche de la matrice.

**DCT = Discret Cosin Transform** (méthode proposée en 1974 par le professeur Rao de l'université du Texas)

Notons qu'à cette étape, aucune information n'a été perdue. Le bloc original peut être reconstitué sans aucune perte en utilisant la DCT inverse et en ajoutant 128 à chaque terme.



Application de la DCT à un pavé de 8 pixels de coté.

$$DCT(i, j) = \frac{1}{\sqrt{2N}} C(i) C(j) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} Pixel(x, y) \cos\left[\frac{(2x+1) i \Pi}{2N}\right] \cos\left[\frac{(2y+1) j \Pi}{2N}\right]$$

$$Pixel(x, y) = \frac{1}{\sqrt{2N}} \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} C(i) C(j) DCT(i, j) \cos\left[\frac{(2x+1) i \Pi}{2N}\right] \cos\left[\frac{(2y+1) j \Pi}{2N}\right]$$

$$\left( \text{où } C(u) = \frac{1}{\sqrt{2}} \text{ si } u = 0; C(u) = 1 \text{ si } u \neq 0 \right)$$

(Pixel (x,y) désigne la valeur du pixel de coordonnées (x,y) et DCT(i,j) le coefficient repéré par la ligne i et la colonne j dans la matrice DCT.

## Étape de quantification

$$Q(i,j) = 1 + (1 + i + j) \times Fq$$

(pour les matheux, on indiquera une formule un peu plus complexe, permettant d'obtenir un grand nombre de matrice différentes :  $Q(i,j) = 1 + (1 + \mu (i^n + j^n)) \times Fq$ . Par souci de simplification, on a pris ici  $\mu = n = 1$  et  $Fq = 5$ . On peut bien entendu compliquer...)

Fq est le facteur de qualité



Nous obtenons la matrice de quantification suivante :

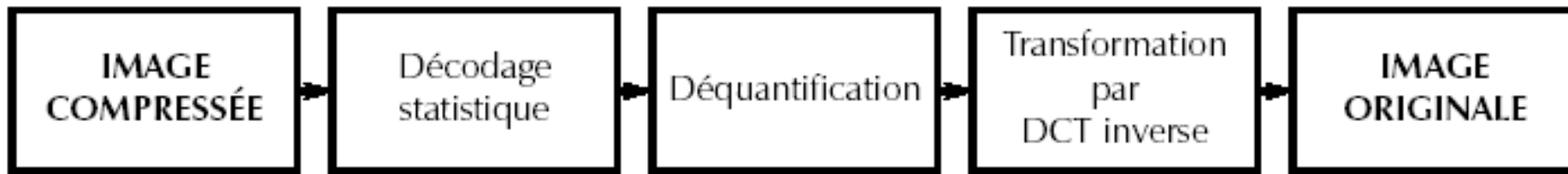
6	11	16	21	26	31	36	41
11	16	21	26	31	36	41	46
16	21	26	31	36	41	46	51
21	26	31	36	41	46	51	56
26	31	36	41	46	51	56	61
31	36	41	46	51	56	61	66
36	41	46	51	56	61	66	71
41	46	51	56	61	66	71	76

Divisons maintenant les valeurs de la matrice de données par notre matrice de quantification. Le résultat est le suivant :

8	23	8	-5	-2	-2	-1	-2
-29	9	0	1	-4	-2	-1	1
4	8	2	-2	0	0	1	0
-1	0	1	1	0	1	1	0
0	2	1	0	0	0	0	0
2	0	0	0	-1	0	0	0
0	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0

Ce qui frappe c'est d'une part le nombre de zéros et d'autre part la position des valeurs non nuls





Après décodage statistique, nous retrouvons notre matrice

8	25	8	-5	-2	-2	-1	-2
-	9	0	1	-4	-2	-1	1
4	8	2	-2	0	0	1	0
-1	0	1	1	0	1	1	0
0	2	1	0	0	0	0	0
2	0	0	0	-1	0	0	0
0	1	0	0	0	0	0	0
1	0	0	0	0	0	0	0

Il faut maintenant déquantifier en multipliant chaque valeur par son coefficient de quantification. Nous obtenons ceci :

48	253	128	-105	-52	162	-36	-82
-319	144	0	26	-124	-72	-41	46
64	168	52	-62	0	0	46	0
-21	0	31	36	0	46	51	0
0	62	36	0	0	0	0	0
62	0	0	0	-51	0	0	0
0	41	0	0	0	0	0	0
41	0	0	0	0	0	0	0

Appliquons ensuite la DCT inverse, et ajoutons 128 à chaque valeur.

Voici la matrice décompressée :

210	275	199	39	-9	27	5	3
110	205	124	55	42	24	20	40
94	237	133	98	98	72	80	71
100	209	111	100	94	102	119	102
153	156	100	136	103	121	126	150
179	150	137	170	102	145	133	174
212	218	210	224	131	181	175	241
265	263	224	243	121	107	107	232



Image originale

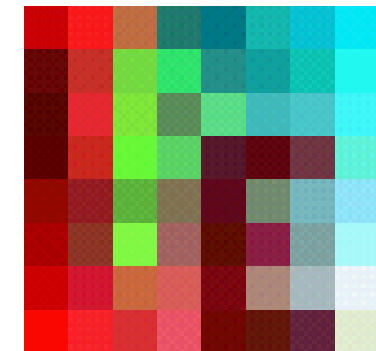
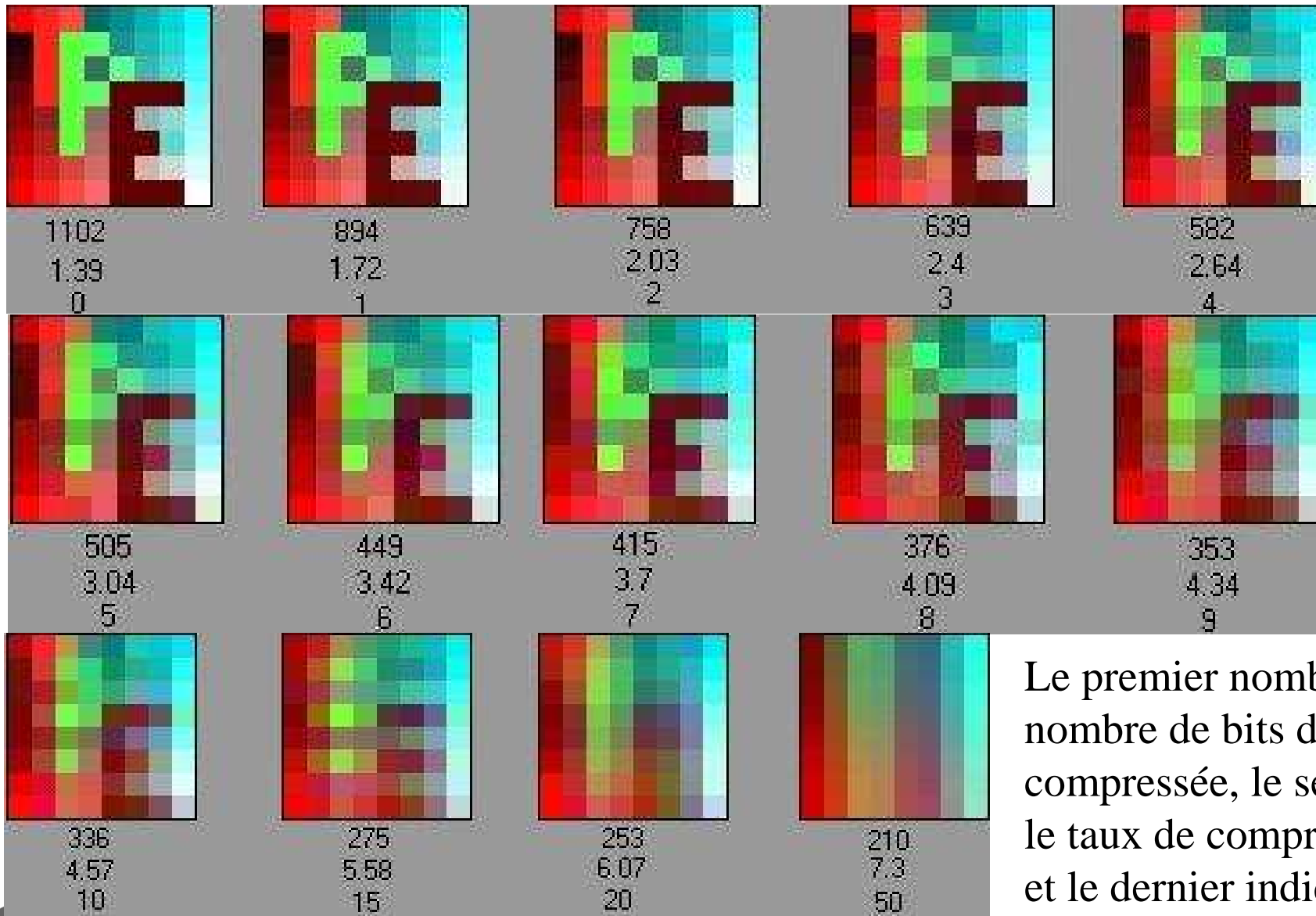
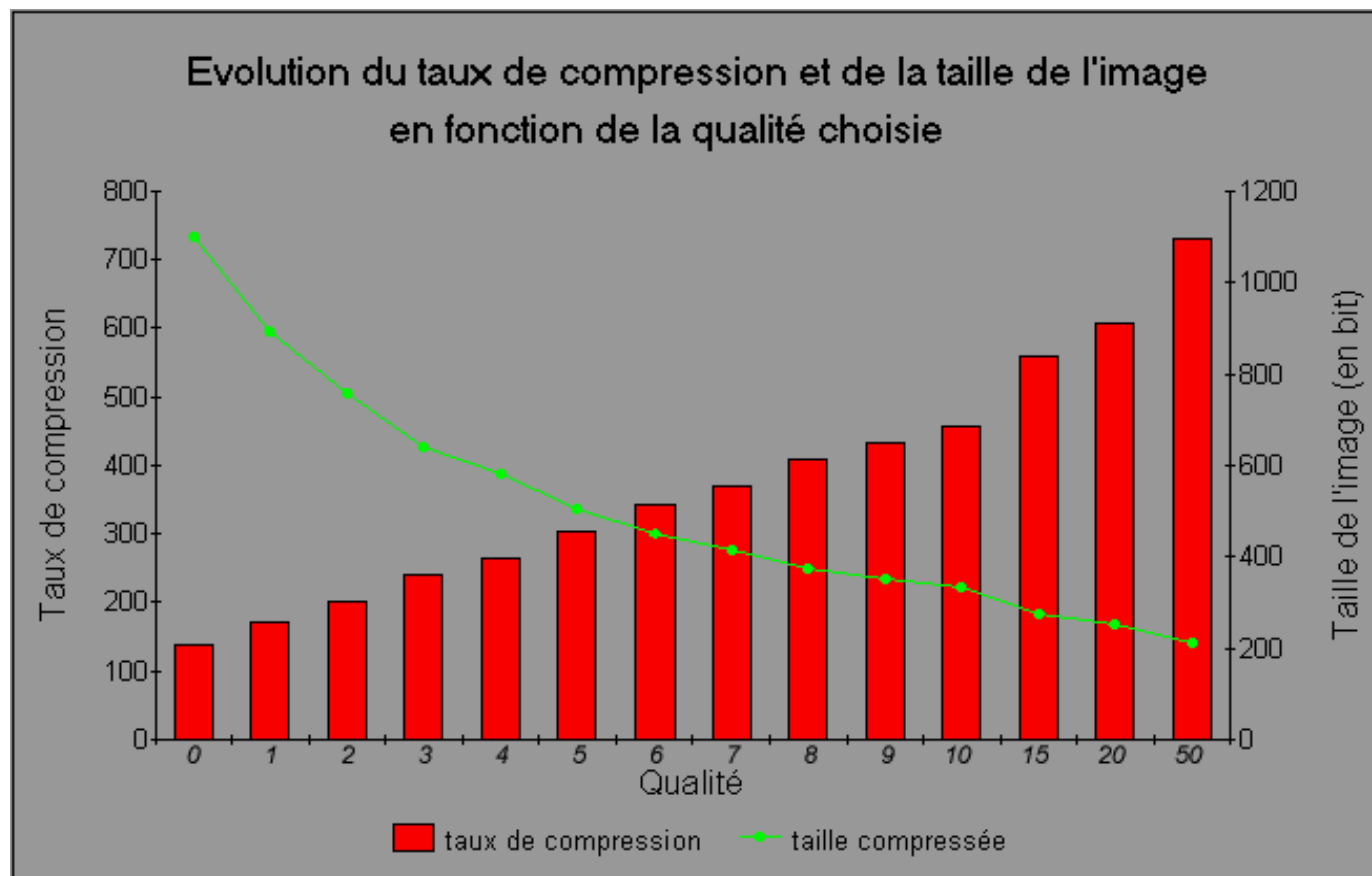


Image décompressée :



Le premier nombre est le nombre de bits de l'image compressée, le second est le taux de compression obtenu et le dernier indique la qualité choisie.

# Résultat



On peut remarquer que la qualité se dégrade rapidement alors que l'augmentation du gain de place est de moins en moins important.



# L'approximation par polynôme

Les variations de couleurs entre deux pixels consécutifs d'une image est souvent faible. On peut donc approcher la matrice de couleurs par une fonction polynôme de degrés plus ou moins élevés.

Prenons pour exemple la série de couleurs suivantes : 255, 234, 204, 213, 203, 197, 169, 147.

x	0	1	2	3	4	5	6	7
série originale	255	234	204	213	203	197	169	147
$f(x)=-13x+249$	249	236	223	210	197	184	171	158
$f(x)=-x^2-10x+256$	256	245	232	217	200	181	160	137
$f(x)=-x^3+9x^2-35x+256$	256	229	214	205	196	181	154	109

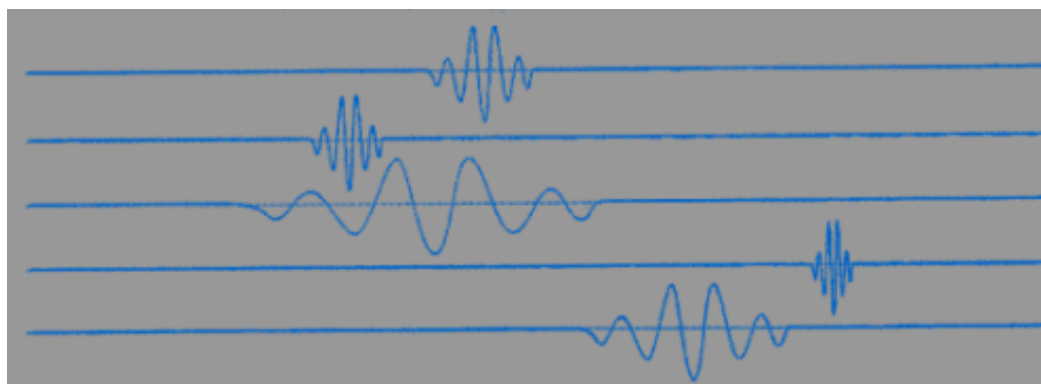
# Le JPEG 2000

## Principe :

- ☞ Contrairement au JPEG qui analyse l'image par blocs, le JPEG 2000 analyse l'image dans sa globalité.
- ☞ Il considère directement chaque ligne comme la variation d'un signal de luminosité et de couleur.
- ☞ Ce signal est ensuite décomposé par une suite d'ondelettes dont on élimine les moins significatives.

# Le JPEG 2000 (suite)

**Les ondelettes** sont des fonctions inventées par Alfred HAAR qui sont nulles la plupart du temps, mais qui varient soudainement entre deux valeurs précises.



Pour décompresser l'image, il suffit d'additionner ces ondelettes.

Remarque : Les images compressées avec l'algorithme JPEG 2000, pour une même qualité, occupent dix fois moins de place qu'avec le JPEG.



# Avantages des ondelettes (JPEG 2000) par rapport à la DCT (JPEG)

- La compression DCT du format JPEG analyse l'image par bloc 8 par 8 pixels ce qui produit un effet de mosaïque (les limites des blocs sont visibles à fort taux de compression)
- La compression par ondelettes ne présente pas cet effet de mosaïque indésirable. Il est donc possible de compresser des images avec un taux de compression élevé tout en concevrant une bonne qualité picturale.
- Les blocs JPEG 8x8 sont quantifiés indépendamment les uns des autres ce qui ne permet pas de réduire les redondances au-delà d'un bloc. Au contraire, la compression par ondelettes est une méthode globale sur toute l'image. Cet avantage se traduit par une efficacité encore plus importante sur les grosses images. Une image de 50 MO peut être réduite à 1 MO.