Techniques Multimédias

Compression des données multimédia

November 29, 2015

Houcemeddine HERMASSI

houcemeddine.hermassi@enit.rnu.tn

École Nationale d'Ingénieurs de Carthage ENI-CAR
Université Carthage
Tunisie

Plan de cour

Compression
Les motivations

Objectifs

Réduire la quantité de bits nécessaire pour représenter des données (images, vidéos...) avec **le minimum de pertes d'informations**.

Applications

- Base de données
- Visio-conférences
- Télé-médecine
- Télévision è haute définition
- Cinéma numérique
- ...

Compression
Comprimer pour transmettre

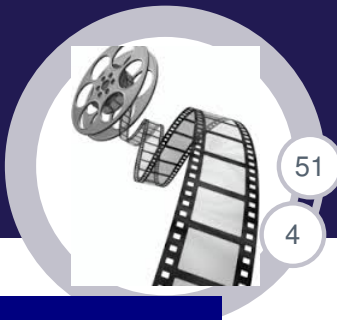
Sur les réseaux

- Informatiques (Internet) : fichiers texte, images, son, vidéo...
- Téléphoniques :voix numérisée...
- Radio-mobiles :GSM, UMTS, 3G, 4ème génération de téléphonie mobile...
- Satellites :Sondes spatiales, observation de la Terre, télévision à haute définition...

Comprimer pour stocker

- Disques durs, clés USB :fichiers...
- CD :sons, images...
- DVD: 2 heures de vidéo (Standard Definition)
- Disque "Blu-Ray" (25 Go) :2 heures de HDTV

Compression
Application et contraintes



"Temps réel"

- Téléphone, vidéo :**COMPRESSION / DECOMPRESSION RAPIDES**

"Temps différé"

- Stockage sur disque (CD, CD ROM, DVD...):**COMPRESSION LENTE / DECOMPRESSION RAPIDE**
- Imagerie satellitaire ou embarquée: **COMPRESSION RAPIDE / DECOMPRESSION LENTE**

Applications

- Médical: Pas d'artefact (erreur de diagnostic)
- Militaire:
  - Conservation des détails (détection de cibles)
  - Aspect mouvement (suivi de mobiles)
- Vidéo "grand public": Effet de masquage de l'œil (espace et temps)
- Vision par ordinateur: Détection des contours (ex. guidage d'un robot...)

Compression
Position du problème

Les performances d'un système de compression s'évaluent par :

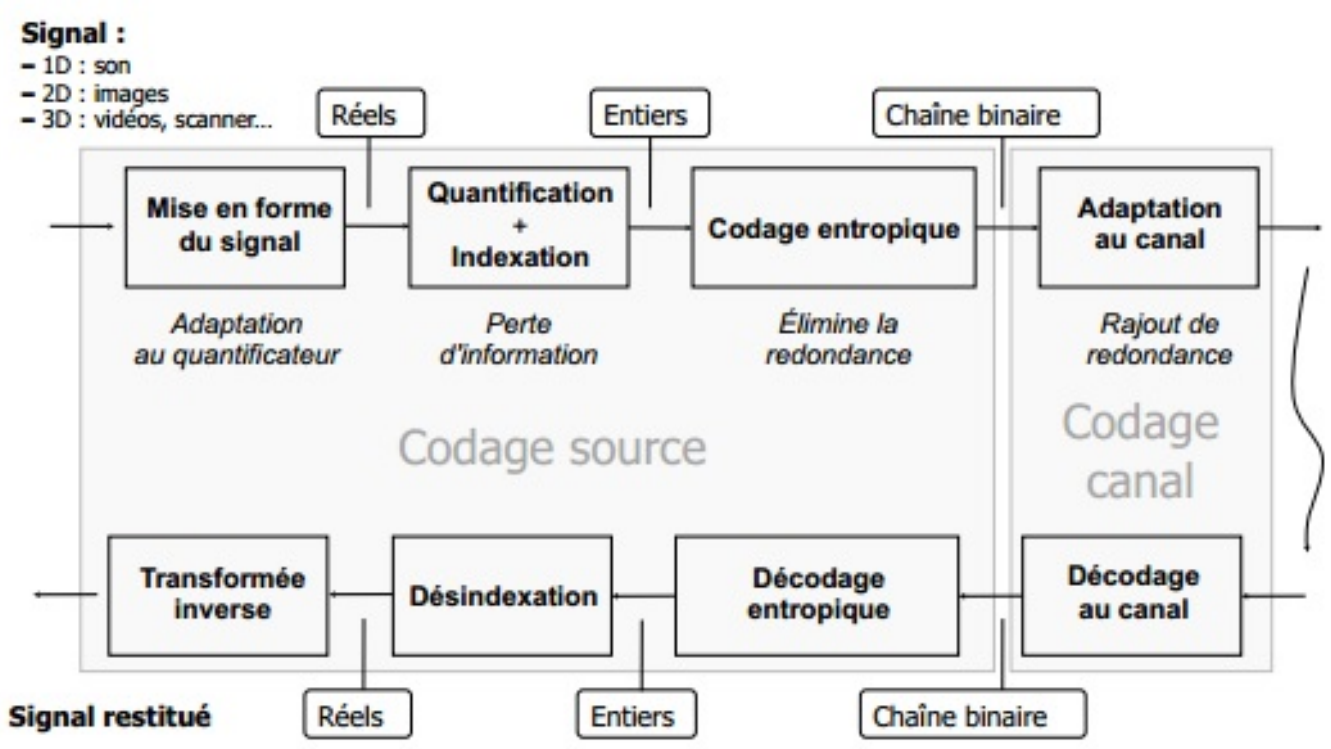
- **Le taux de compression** :Débit initial / débit après compression
- **La qualité du signal comprimé** :
  - Critère subjectif (visuel)
  - Critère objectif (EQM, SNR...)
- **La complexité du système**: Coût calcul, mémoire requise

PROBLEME

- Optimiser ces 3 facteurs en même temps

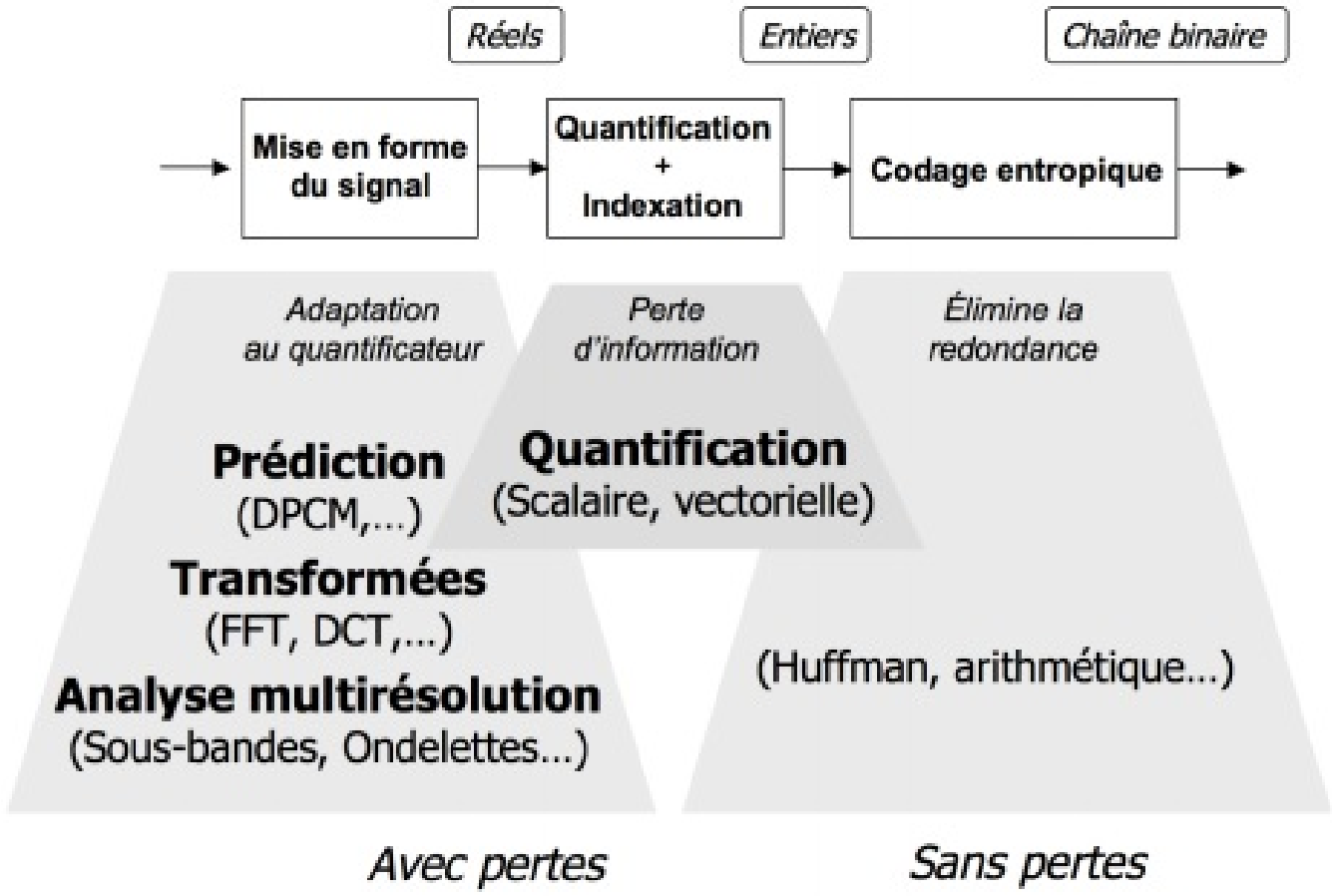
Introduction à la compression de données

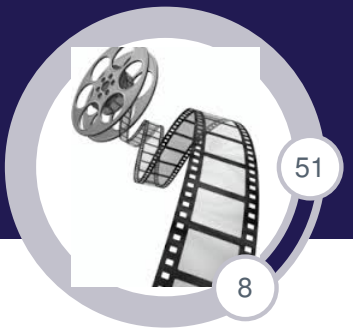
La chaîne de compression



Introduction à la compression de données

Les outils





Entropie

Soit **S** une source, chaque élément pouvant prendre des valeurs d'un ensemble discret **E(S)**. La probabilité qu'un élément ait une valeur $i \in E$ est $p(i)$. Deux principes:

- **La quantité d'information** d'un signal est différente de **la quantité de données** qu'il produit
- Pour un événement donné, elle est étroitement liée à sa probabilite d'apparition.

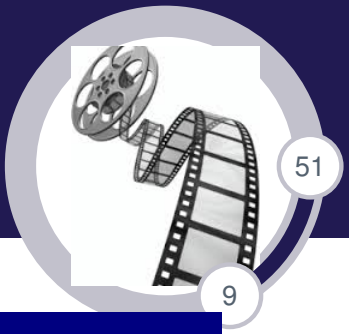
D'après la théorie de l'information (**Shannon 48**), on définit **la quantité d'information** d'une valeur i de la source comme:

$$q(i) = -log(p(i))$$

L'entropie de la source est la valeur moyenne de q.

$$H(S) = -\sum_{i \in E(S)} p(i) log p(i)$$

Elle représente le nombre de bits moyen nécessaire pour coder la source.



Entropie: Exemples

- Symboles équiprobables

Si $p(i) = 1/card(E(S))\forall i$, $H(S) = log(card(E(S)))$. ainsi, pour coder 256 symboles équiprobables, il faut $log_2 256$ bits, soit 8 bits!!!

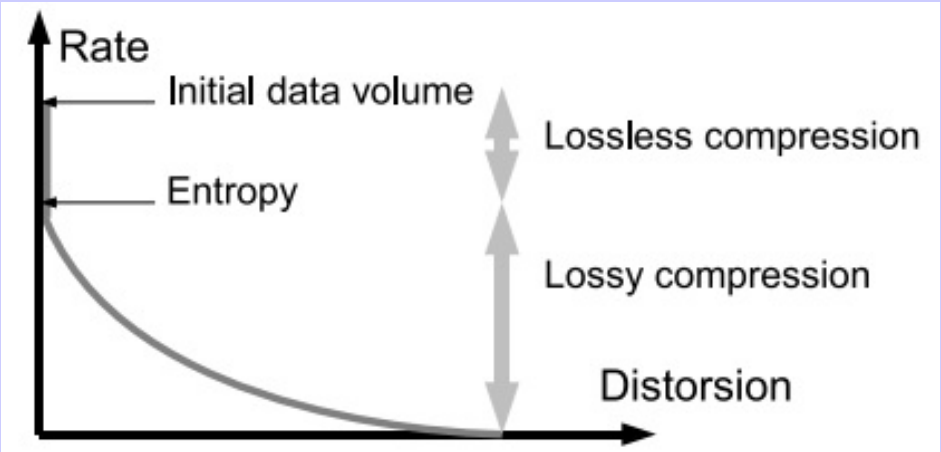
- Pour une source binaire telle que $p(0) = p$ et $p(1) = 1 - p$, nous avons:
  - si $p = 1/2$, $H = (1/2)log_2 1/2 + (1/2)log_2 1/2 = 1 bit/symbole$
  - si $p = 1/4$, $H = (1/4)log_2 1/4 + (3/4)log_2 3/4 = 0.81 bit/symbole$
  - si $p = 1/16$, $H = (1/16)log_2 1/16 + (15/16)log_2 15/16 = 0.44 bit/symbole$
  - si $p = 0$, $H = 0log01log1 = 0$

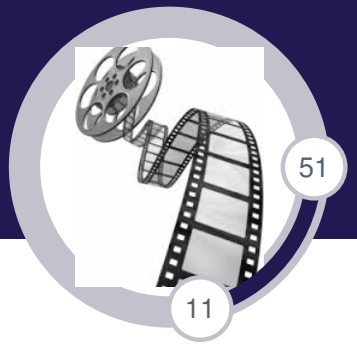
Redondance statistique

Soit $\lambda(S)$, le nombre de bits utilisé pour coder la source. On a toujours $\lambda(S) \geq H(S)$. Si $\lambda(S) > H(S)$, il y a rédondance statistique.
Exemple : soient l'ensemble des programmes d'une chaîne de télévision pendant 10 ans. Il y a au plus $10 \times 365 \times 24 \times 3600 \times 25 = 1.884 \times 10^9 = 2^{43}$ images différentes. Si elles sont équiprobables $H(S) = 43$.
Si les images sont codees sans compression
$\lambda(S) = (576 \times 720 + 576 \times 360) \times 8 = 6635520 \gg H(S) \Rightarrow$ Rédondance très forte dans les images et la vidéo.

Fonction débit-distorsion

- Sans perte d'information, on peut augmenter le taux de compression (reduire le débit) jusqu'a l'entropie.
- Au delà, on va devoir dégrader l'image (augmenter la distorsion qui mesure la différence entre la source originale et comprimée).



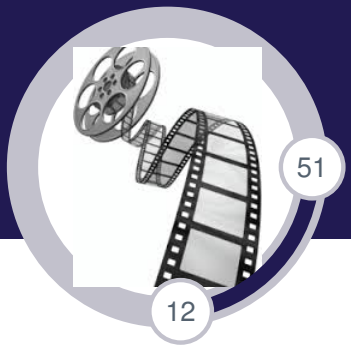


Principe

- Les codes usuels sont a **longueur fixe**. Par exemple un caractère d'un alphabet latin sera toujours code par un octet.
- Pour comprimer, on doit utiliser des codes a **longueur variable** (variable length codes ou VLC).
- Un élément sera codé par un code dont la longueur est **en proportion inverse de sa probabilité**.
- Les symboles fréquents sont codés par un code court, les symboles rares par un plus grand nombre de bits.

Problèmes:

- Comment associer un code a un symbole pour une source ?
- Comment decoder sans confusion entre symboles ?
- Peut-on decoder au vol ou doit on attendre que toutes les données soient arrivées ?

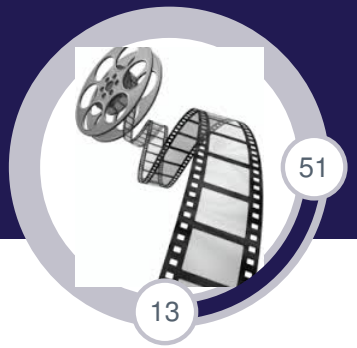


Méthodes de compression sans perte

Suivant les caractéristiques de la source et les performances attendues, différentes méthodes peuvent être utilisées. Elles utilisent :

- Soit la redondance statistique
  - **Méthode de Huffman**
  - **Codage arithmètique**
- Soit la corrélation entre valeurs successives
  - **Algorithme de Lempel-Ziv**
  - **Codage par plage**

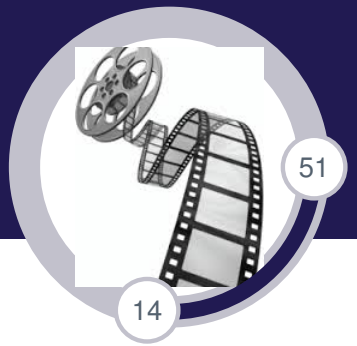
En règle générale, le gain de compression directement sur des images est limite.
Elles sont toutefois toujours présentes comme phase finale de codage des éléments d'une image comprimée. (codage entropique).



Méthodes de Huffman

1. Ordonner les symboles par probabilité décroissante.
2. En partant du bas, affecter ’0’ au symbole le moins probable and ’1’ au suivant.
3. Supprimer ces symboles de la liste et les combiner en un symbole composite. La probabilité de ce symbole est la somme des probabilités initiales des symboles.
4. Reordonner la liste
5. repéter 2-4, tant qu’il reste au moins deux symboles dans la liste.

Cela définit un arbre dont les feuilles sont les symboles originaux et les noeuds intermédiaires sont les noeuds composites.



Principe de construction de l'arbre

1. On représente un élément i par **une séquence de bits de longueur inversement proportionnelle à la probablilité d'apparition** $p(i)$.
2. Optimal pour les codes à nombre de bits entiers.
3. Le problème est de définir un code à longueur variable dont aucun élément ne soit le début d'un autre (code préfixé).
4. La méthode de Huffman (1952) repose sur la construction d'**un arbre basé** sur les probabilités d'apparition des éléménts.

Algorithme de décodage

- En partant de la racine, suivre les branches de l'arbre en fonction des bits reçus.
- Quand on atteint une feuille, un symbole a été trouvé.

Compression sans perte
Méthodes de Huffman

Exemple

- Soient a coder les éléments suivants, avec les probabilités suivantes :

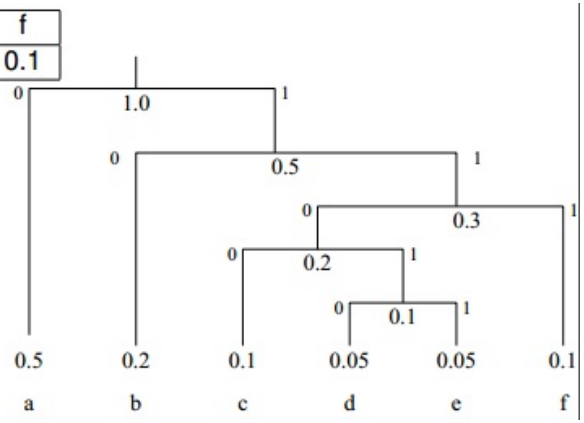
x_i	a	b	c	d	e	f
$p(i)$	0.5	0.2	0.1	**0.05**	**0.05**	0.1

x_i	a	b	c	de	f
$p(i)$	0.5	0.2	**0.1**	**0.1**	0.1

x_i	a	b	cde	f
$p(i)$	0.5	0.2	**0.2**	**0.1**

x_i	a	b	cdef
$p(i)$	0.5	**0.2**	**0.3**

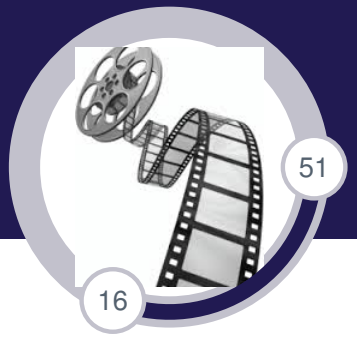
x_i	a	bcdef
$p(i)$	**0.5**	**0.5**



i	1	2	3	4	5	6
x_i	a	b	c	d	e	f
code	0	10	1100	11010	11011	111

L'entropie de la source est:
0.5*1+0.2*2+0.1*4+0.05*5+0.05*5+0.1*3=2.1
Le mot 'face' est code by 1110110011011.

Compression sans perte
Méthodes de Huffman



Codage de Huffman

1. L'utilisation d'un arbre (c.a.d un graphe acyclique) assure la condition préfixe.
2. Le decodage peut se faire à la volée.
3. Il faut connaître les caractéristiques de la source. Soit en la balayant (par exemple un fichier), soit par un modèle pertinent de la source.
4. Nécessite la connaissance préalable de la table par le décodeur ou son envoi.

Problèmes:

- Optimal uniquement si les probabilités sont des puissance de 2.
- Le meilleur débit est de 1 bit/symbole. Problémes avec une source binaire.

Ces methodes évitent la contrainte du codage par un nombre de bits entier.

Principe

[proposé par Elias] On code un ensemble de valeurs par un nombre flottant $\in [0, 1[$.

Rappel: représentation binaire des nombres fractionnaires

Utilisation des puissances négatives de deux.
$0.625 = 0.5 + 0.125 = \frac{1}{2^1} + \frac{1}{2^3} = 0.101$

- Plus efficace qu'Huffman.
- Peut être utilisé pour coder une source binaire.
- Peut être simplement adapaté à des changements des statistiques de la source.

L'intervalle [0,1[est découpé en sous-intervalles associés à chaque éléments et de taille égale à la probabilité d'apparition de l'élément.

a	b	c	d
1/2	1/8	1/8	1/4

Algorithme

Soient $l_0(e)$ et $D(e)$ la borne inférieure et la largeur de l'intervalle associes à l' élément e.

1. Lire un nouvel élément e
2. $l_0 = l_0 + D \times l_0(e)$; $D = D \times D(e)$
3. repéter 2 - 3 pour tous les éléments à encoder.
4. Le code est tout nombre $l \in [l_0, l_0 + D[$.

Exemple

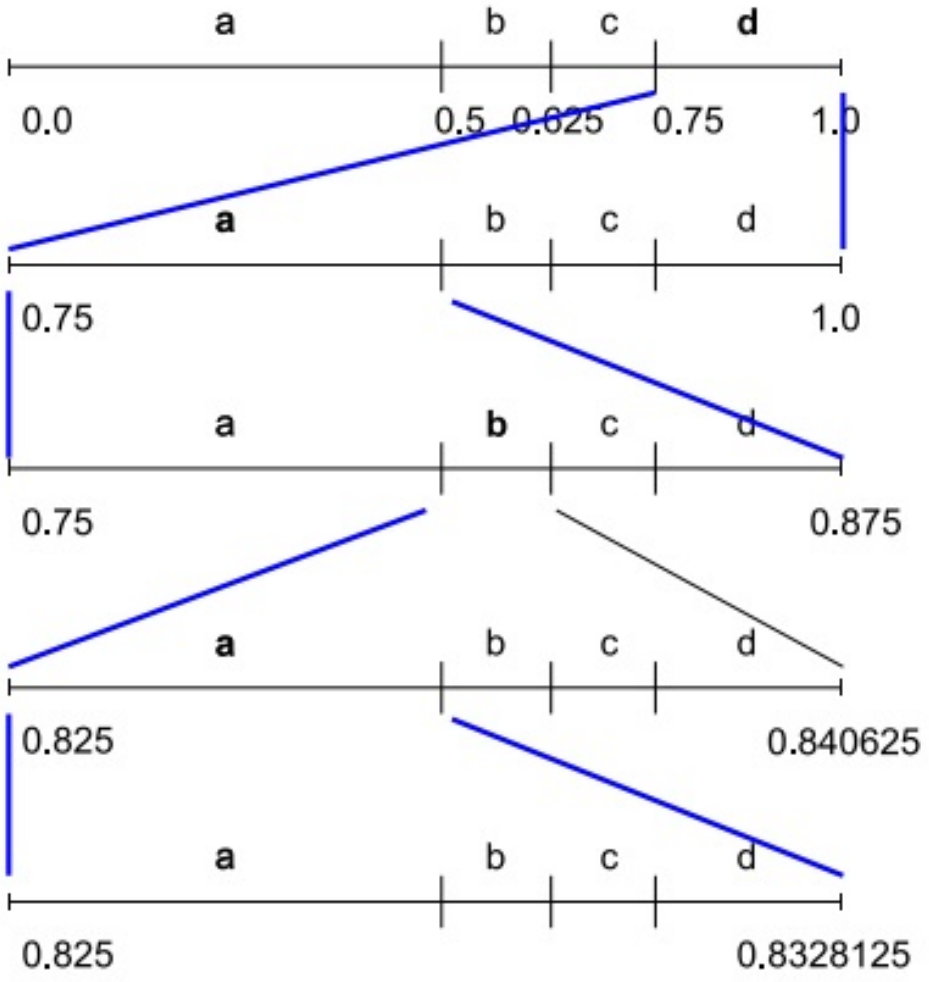
a	b	c	d
1/2	1/8	1/8	1/4

codage “daba”:

1. $l_0 = 0; D = 1.0$
2. lire ’d’. $l_0 = l_0 + 3/4 * D = 3/4; D = D * 1/4 = 1/4$ lire ’a’. $l_0 = l_0 + 0 * D = 3/4; D = D * 1/2 = 1/8$
3. lire ’b’. $l_0 = l_0 + 1/2 * D = 13/16; D = D * 1/8 = 1/64$
4. lire ’a’. $l_0 = l_0 + 0 * D = 13/16; D = D * 1/2 = 1/128$

$l_0 = 13/16 = 0.1101; l_1 = l_0 + D = 105/128 = 0.1101001$
Tout nombre $\in [l = 0, l_1[$ peut coder la séquence. Par exemple 0.1101 (4 bits).

Compression sans perte
Codage arithmètique



Compression sans perte
Codage arithmètique

Le codage arithmètique approche l'entropie de la source.

Pour coder un intervalle de taille s $s = \prod_{i=1}^{n} p_i$, b bits sont nécessaires

$$b = log\ s = \sum_{i=1}^{n} logp_i = \sum_{i=1}^{N} p_i logp_i$$

Problèmes :

- Comment différentier des séquences qui produisent le même code (par exemple 'd', 'da' and 'daa' peuvent toutes être codées avec 0,11)?Introduire un caractère spécial de fin (ou coder un nombre fixe de symboles).
- Codage incrémental ou faut il attendre la fin de la séquence ? On peut envoyer la partie commune aux codes des deux bornes des qu'elle est connue.
- Faut-il un précision arithmétique infinie ? On peut faire les calculs avec une précision arithmètique finie. Méthode Pasco-Jones.
  - Les calculs sont fait en précision finie sur m bits.
  - Les codes sont des entiers $\in [0, 2^m 1]$.
  - Après chaque calcul, l'intervalle est dilaté et ses bornes sont arrondies.

Compression sans perte
Algorithme de Lempel-Ziv

Principe

Famille d'algorithmes proposés par Lempel et Ziv (1977). Basés sur la répétition de séquences de symboles dans une source. Marche pour les sources Markoviennes

Rappels : sources Markoviennes

Une source est dite sans mémoire si la probabilité d'un symbole à l'instant *t* ne dépend pas des symboles produits aux instants t - 1, t - 2, ...
Une source est dite markovienne d'ordre *m* si la probabilité d'occurrence d'un symbole a l'instant *t* dépend des valeurs prises par les symboles produits par la source aux instants t - 1, t - 2, ..., t - m.
Beaucoup de sources sont Markoviennes. Textes, images, séquences d'ADN, etc...
Les méthodes de compression peuvent utiliser cette propriété pour améliorer les résultats.

On suppose que l'on dans a une fifo *T* les *n* derniers symboles lus de la source. Pour coder un ensemble de nouveaux symboles produits par la source, on essaie de trouver dans *T* une occurrence de cette séquence de symbole et on envoie sa longueur et sa position.
Utilisé par le format de compression de fichiers **gzip** et par **png** (format de fichier d'images open source).

Compression sans perte
Algorithme de Lempel-Ziv

Algorithme

1. La table T (initiallement remplie de 0), contient les n derniers symboles lus.
2. On lit des symboles à coder, et on cherche dans T une chaine correspondant aux k premiers.
3. soit s cette chaine, ps sa position dans T et ls sa longueur. soit c le premier symbole à coder qui n'appartient pas à cette chaine. On envoie le code (ps, ls, c) pour coder les $k+1$ premiers symboles de la source.
4. Ces $k+1$ symboles sont inserés dans T, en supprimant les $k+1$ premiers
5. Retourner en 2 tant qu'il y a des symboles à coder

Exemple: papa papou papa pou

0	1	2	3	4	5	6	7	8		
									*p*apa␣papou␣papa␣pou	(0,0,'p')
								p	*a*pa␣papou␣papa␣pou	(0,0,'a')
							p	a	**pa**␣papou␣papa␣pou	(7,2,' ')
				p	a	p	a		**pap***o*u␣papa␣pou	(4,3,'o')
p	a	p	a		p	a	p	o	*u*␣papa␣pou	(0,0'u')
a	p	a		p	a	p	o	u	␣**pap***a*␣pou	(3,4,'a')
a	p	o	u		p	a	p	a	␣pou	(0,0,' ')
p	o	u		p	a	p	a		**pou**	(0,3,'')

Les triplets (position, longueur, nouveau symbole) peuvent être encodés par Huffman pour une meilleure efficacité

LZW (Lempel Ziv Welsh) 1984

Egalement base sur des chaînes de symboles. On mémorise les chaîes que l'on rencontre au fur et à mesure de leur production par la source. Donne une compression tres efficace sur les fichiers texte (utilise dans les programmes de compression zip, compress) et pour les images GIF.

1. soit *D* une table avec les chaînes de symboles deja trouvées. Initialement *D* contient l'ensemble des symboles que peut produire la source.
2. lire le premier symbole et soit la chaîne s=ce symbole.
3. lire le symbole suivant *c*
4. Tant que s + c est une chaine presente dans D
5. s = s + c
6. lire le symbole suivant c
7. Insérer la nouvelle chaine s + c dans la première case disponible dans D
8. Envoyer le code (index dans D) de la chaîne *s*
9. s = c
10. Tant qu'il y a des symboles a lire, aller en 3

Le decodeur réalise la méme opération pour déterminer à la volée le contenu du dictionnaire.

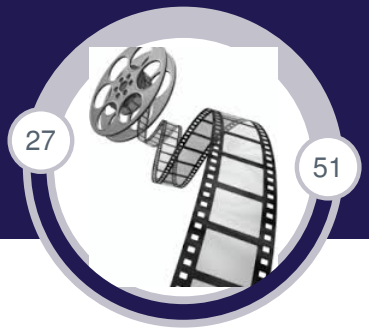
LZW (Lempel Ziv Welsh) 1984(Exemple)

exemple: papa papou papa pou

Initialement le dictionnaire contient les 256 caracters ISO.

		papa␣papou␣ papa␣pou	('p')
256	pa	**a**pa␣papou␣papa␣pou	('a')
257	ap␣	**pa**␣papou␣papa␣pou	(256)
258	pa␣	␣papou␣papa␣pou	(␣)
259	␣p	**pa***p*ou␣papa␣pou	(256)
260	pap	**p***o*u␣papa␣pou	('p')
261	po	**o***u*␣papa␣pou	('o')
262	ou	**u**␣papa␣pou	('u')
263	u␣	␣**p**apa␣pou	(259)
264	␣pa	**apa**␣pou	(257)
265	apa	**a**␣pou	(␣)
266	a␣	␣**p**ou	(259)
267	␣po	**ou**	(262)

Compression sans perte
Codage par plage



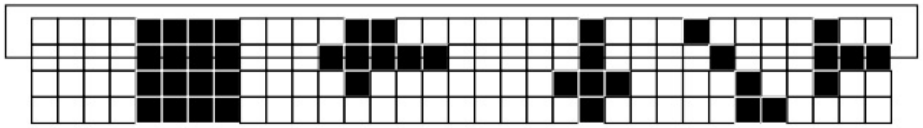
Principe

Base sur la répétition du même symbole. On envoie le code du symbole avec le nombre de repétitions (run-length encoding ou RLE). Marche bien si le nombre of symboles potentiels est faible.
exemple: format fax G3 (CCITT RFC 804)

- Permet d'envoyer sur un reseau téléphonique commuté des images binaires.
- Une ligne est composees d'au plus 1728 pixels. Elle commence toujours par une plage de pixels blanc. On code ensuite uniquement le nombre de pixels de chaque couleur.
- Une plage est de 0 a 63 pixels.
- Un code de Huffman optimise sur des documents types à été défini pour les pixels blancs et noirs.
- Des codes particuliers permettent de traiter les cas de plages de longueur > 63.
- Des codes permettent également de finir une ligne, une page, la transmission, etc.

Compression sans perte
Codage par plage

Principe



Code : 4,4,4,2,7,1,3,1,4,1,2

Ce principe est également utilisé pour certains dessins bitmap (format MacPaint).
Il marche tres mal pour des images réelles en niveaux de gris.

Compression destructive (avec pertes)
Principe

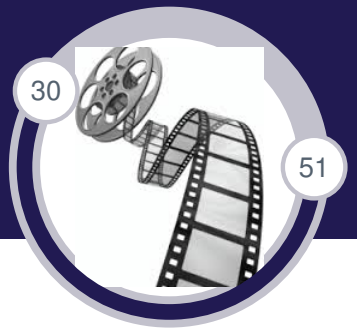
Principe fondamental

Etant donné un pixel, il y a de forte chance que ses voisins possèdent la même couleur, ou du moins une couleur similaire

Procédés

- Les techniques de compression que nous allons voir sont donc basés sur le fait que les niveaux de gris de pixels voisins sont **fortement corrélés**. On parle alors de **redondance spatiale**.
- C'est cette corrélation qui permet la compression

Compression destructive (avec pertes)
Principe



Redondance spatiale : un exemple

Voici une séquence de niveaux de gris :
12, 17, 14, 19, 21, 26, 23, 29, 41, 38, 31, 44, 46, 57, 53, 50, 60, 58, 55, 54, 52, 51, 56, 60
Ici, seulement 2 pixels sont identiques. La valeur moyenne des NdG est de 40.3.
Travaillons maintenant avec les différences de 2 pixels adjacents :
12, 5, -3, 5, 2, 4, -3, 6, 11, -3, -7, 13, 4, 11, -4, -3, 10, -2, -3, 1, -2, -1, 5, 4
Cette séquence illustre **le potentiel** de la compression :

- les différences possèdent des valeurs plus faibles que les pixels originaux, leur valeur moyenne étant seulement de 2.58,
- les différences possèdent des valeurs répétitives
- les différences sont décorrélées : les valeurs adjacentes sont différentes

Compression destructive (avec pertes)
Principe

Mesure de la perte

Erreur quadratique moyenne :

$$SNR = \frac{1}{N}\sum_{i=0}^{N-1}(n_{compr}(i) - n(i))^2 \quad (1)$$

Une autres grandeur est traditionnellement utilisée dans la littérature : la rapport sur bruit maximum, ou peak signal to noise ratio, noté PSNR, avec

$$PSNR = 20\log_{10}\frac{max_i(n(i)}{RMSE} \quad [dB] \quad (2)$$

- mesure normative : pour le MPEG, un seuil sur le PSNR de 0.5dB permet de décider ou non une optimisation du codage
- valeurs typiques entre 20 et 40 dB
- mais attention ... dire qu'un PSNR de 25dB est bon n'a pas de signification !

Compression destructive (avec pertes)
Premières méthodes intuitives

Sous échantillonnage

on ignore simplement certains pixels. Les effets sur l'image sont très visibles (grande perte de détails) : cette méthode simple n'est que très peu utilisée.

Sous échantillonnage des couleurs

notre oeil est moins sensible aux variations de chrominance que de luminance. Les deux chrominances sont sous-échantillonnées d'un facteur 2, conduisant à une réduction de 50taille du fichier

Dans l'ordre : Original, Chrominance / 2, Luminance / 2, Résolution / 2

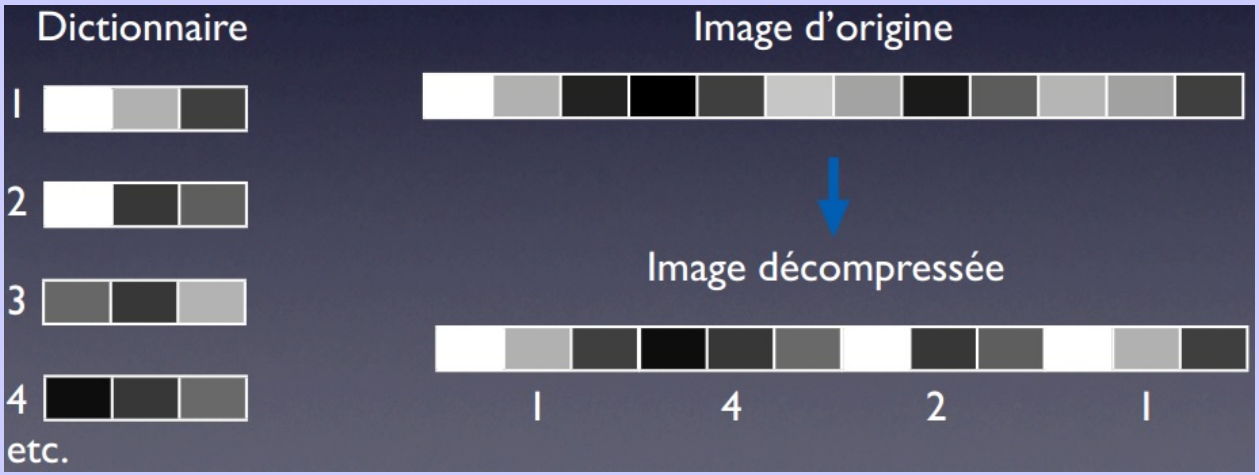
Compression destructive (avec pertes)
Premières méthodes intuitives

Quantification scalaire

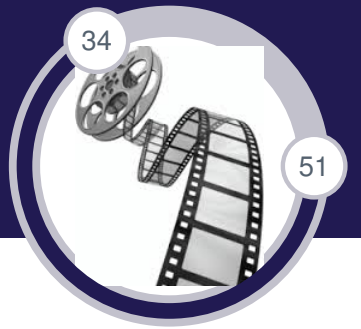
on supprime simplement les bits les moins significatifs du codage.
Exemple :
215 214 197 211 210 209, en binaire s'écrit :
11010111 11010110 11000101 11010011 11010010 11010001, qui devient :
1101 1101 1100 1101 1101 1101, qui est décodé en :
11010000 11010000 11000000 11010000 11010000 11010000, ou encore :
208 208 192 208 208 208

Quantification vectorielle

l'image est partitionnée en blocs de tailles fixe (appelés vecteurs). Le codeur dispose d'une liste de vecteurs prédéfinis (dictionnaire de vecteur), et c'est l'index du vecteur qui est inscrit dans le fichier compressé



Compression destructive (avec pertes)
Transformations orthogonales



Principe

Ces transformations sont conçues pour posséder 2 propriétés :

- Réduire la redondance de l'image
- Identifier les parties les moins importantes de l'image à compresser

Ces parties sont le plus souvent identifiées en travaillant sur les différentes fréquences (spatiales) constituant l'image.
Pourquoi ?
Les basses fréquences correspondent aux éléments importants d'une image, tandis que les hautes fréquences décrivent les détails d'une image.
De plus, notre oeil n'est pas sensible aux variations rapides de contraste dans une image.

Compression destructive (avec pertes)
Transformations orthogonales

Principe

En pratique, ces transformations se doivent d'être **rapides** et **faciles** à implémenter. Ainsi, ce sont des **transformations linéaires** qui sont les plus utilisées, pour lesquelles on peut écrire (cas 1D où n=4) :

$$\begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \\ w_{41} & w_{42} & w_{43} & w_{44} \end{pmatrix} \begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \end{pmatrix}$$

Ou encore, **C = WD**, où **D** est la matrice contenant les pixels de l'image d'origine, et **C** celle contenant les coefficients de la transformée.
La nature de la transformation est donc entièrement définie à partir de la matrice **W** ou des coefficients qui la compose.

Compression destructive (avec pertes)
Transformations orthogonales

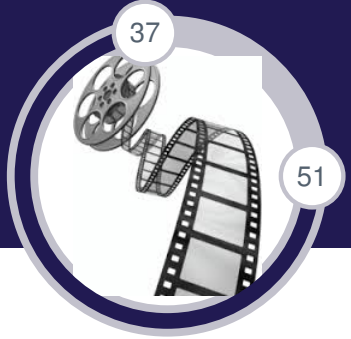
Exemple

$$W = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix} \qquad D = \begin{pmatrix} 5 & 6 & 7 & 4 \\ 6 & 5 & 7 & 5 \\ 7 & 7 & 6 & 6 \\ 8 & 8 & 8 & 8 \end{pmatrix}$$

$$C' = WD = \begin{pmatrix} 26 & 26 & 28 & 23 \\ -4 & -4 & 0 & -5 \\ 0 & 2 & 2 & 1 \\ -2 & 0 & -2 & -3 \end{pmatrix}$$: seule 1 dimension est traitée

$$C = WDW^T = \begin{pmatrix} 103 & 1 & -5 & 5 \\ -13 & -3 & -5 & 5 \\ 5 & -1 & -3 & -1 \\ -7 & 3 & -3 & -1 \end{pmatrix}$$: les 2 dimensions de l'image sont traitées

Compression destructive (avec pertes)
Transformations orthogonales



Exemple

Valeur dominante, contenant presque toute l'énergie de **D**: Réduction de la redondance

$$C = WDW^T = \begin{pmatrix} 103 & 1 & -5 & 5 \\ -13 & -3 & -5 & 5 \\ 5 & -1 & -3 & -1 \\ -7 & 3 & -3 & -1 \end{pmatrix}$$

Parties moins importantes de l'image: Valeurs plus faibles que celles des pixels traités

Principe fondamental & Idée de la compression

- Concentration de l'énergie dans les basses fréquences
- Quantification des éléments de C, et plus particulièrement des termes décrivant les hautes fréquences.

Compression destructive (avec pertes)
Transformations orthogonales

Transformée de Walsh-Hadamard

$$H(u,v) = \frac{1}{N} \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} I(x,y)(-1)^{\sum_{i=0}^{n-1} b_i(x)p_i(u)+b_i(y)p_i(v)}, \text{ avec } N = 2^n$$

ième digit de la représentation binaire de l'entier x.
Exemple :
b2(6) = b2(110) = 1

$$\begin{aligned} p_0(u) &= b_{n-1}(u) \\ p_1(u) &= b_{n-1}(u) + b_{n-2}(u) \\ p_2(u) &= b_{n-2}(u) + b_{n-3}(u) \end{aligned}$$

- Calcul très rapide (additions et soustractions seulement)
- Performances faibles en compression : faible concentration de l'énergie dans les basses fréquences
- Finalement très peu utilisé ...

Compression destructive (avec pertes)
Transformations orthogonales

Transformée de Karhunen-Loève

- Aussi appelée transformée de Hotelling, ou transformée en vecteurs propres, elle est utilisée pour l'analyse en composantes principales.
- Elle est théoriquement la transformée qui possède **la meilleur efficacité** en terme de concentration de l'énergie, mais reste peu utilisée en pratique puisqu'elle est très lourde en temps de calcul.

$\Rightarrow$ Elle donne la plus grande distribution de variance sur les axes
Soient A la matrice de transformation de Karhunen-Loève et W la matrice contenant le résultat de la transformation

L'image d'origine est partitionnée en k blocs de longueur n (vecteur colonne). La matrice **V** de l'image d'origine est constituée des k matrices précédentes, préalablement centrées.

On a donc : $W = AV$

Compression destructive (avec pertes)
Transformations orthogonales

Transformée de Karhunen-Loève

- La matrice VV^T est symétrique, et ses éléments sont les covariances des colonnes de **V**.
- La transformation de KL consiste à choisir comme matrice **A** la matrice constituée des vecteurs propres (orthogonaux) normalisés de VV^T, de sorte que :

$$WW^T = A(VV^T)A^T = \begin{pmatrix} \lambda_1 & 0 & \dots & 0 \\ 0 & \lambda_2 & \dots & 0 \\ \dots & & & \\ 0 & 0 & \dots & \lambda_n \end{pmatrix}$$

où les lambda désignent les valeurs propres de VV^T

Conséquence

Ainsi, cette transformation dépend de l'image ! De plus, la matrice A doit être incluse dans le fichier de l'image compressé, et aucune méthode rapide de calcul de A n'a encore été découverte. En conséquence, elle est très peu utilisée.

Compression destructive (avec pertes)
Transformations orthogonales

Transformée en cosinus

C'est la transformation de très loin la plus utilisée en compression.

Transformation directe (1D) :

$$C_k = \sqrt{\frac{2}{n}} K_k \sum_{i=0}^{n-1} p_i \cos\left[\frac{(2i+1)k\pi}{2n}\right] \qquad K_0 = \frac{1}{\sqrt{2}}$$

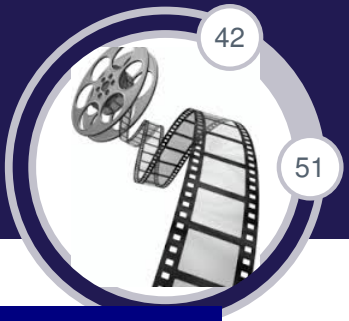
$$K_n = 1,\ n \neq 0$$

Transformation inverse (1D) :

$$p_i = \sqrt{\frac{2}{n}} \sum_{j=0}^{n-1} K_j C_j \cos\left[\frac{(2i+1)j\pi}{2n}\right]$$

- Cette transformation tends à concentrer de manière importante dans les premiers coefficients Ck toute l'information. Ces premiers coefficients représente les **informations importantes** de l'image, et sont liées aux basses fréquences des cosinus de la transformation.
- Les autres coefficients sont nuls ou quasi-nuls, et correspondent à des plus **hautes fréquences** des cosinus.

Compression destructive (avec pertes)
Transformations orthogonales



Transformée en cosinus: Exemple

p = (12, 10, 8, 10, 12, 10, 8, 11) $\Rightarrow$ C = (28.6, 0.6, 0.5, 1.8, 3.2, -1.7, 0.2, -0.3)
C' = (28, 0, 0, 2, 3, -2, 0, 0) $\Rightarrow$ p' = (11.2, 9.6, 7.7, 9.6, 12.3, 10, 8.1, 10.7)
soit au maximum une erreur de 6% seulement !
Dans le cas précédent, les données étaient fortement corrélées (p = (12, 10, 8, 10, 12, 10, 8, 11)).
Et si ce n'est pas le cas ?
p = (-12, 24, -181, 209, 57.8, 3, -184, -250) $\Rightarrow$ C = (-117.8, -240.8, 126.9, 121.2, 9, -109.5, -185.2)
On quantifie en C' = (-120, 170, -240, 125, 120, 9, -110, -185)
$\Rightarrow$ p' = (-12.1, 25.5, -179.9, 208.2, 55.6, 0.36, -185.42, -251.7)
soit une erreur au maximum de 88% !

Conséquence

De la même façon, la taille n de la transformée a une importance capitale :

- si n est trop faible, le nombre de coefficients est également faibles, et ils ont donc tous des valeurs importantes,
- si n est trop grand, alors les valeurs à traiter peuvent devenir trop décorrélées.

En pratique

une valeur de **n=8** est un bon compromis, et la plupart des algorithmes de compression utilisent une telle valeur.

Compression destructive (avec pertes)
Transformations orthogonales

Transformée en cosinus 2D

Transformation directe :

$$C_{ij} = \frac{2}{\sqrt{mn}} K_i K_j \sum_{x=0}^{n-1} \sum_{y=0}^{m-1} p_{xy} \cos\left[\frac{(2x+1)i\pi}{2n}\right] \cos\left[\frac{(2y+1)j\pi}{2m}\right]$$

Transformation inverse :

$$p_{xy} = \frac{2}{\sqrt{mn}} \sum_{i=0}^{n-1} \sum_{j=0}^{m-1} K_i K_j C_{ij} \cos\left[\frac{(2x+1)i\pi}{2n}\right] \cos\left[\frac{(2y+1)j\pi}{2m}\right]$$

Compression destructive (avec pertes)
Transformations orthogonales

Transformée en cosinus 2D: méthode de compression

1. Partitionner l'image à compresser en blocs carrés de 8x8 pixels. Si les dimensions ne sont pas multiples de 8, périodiser les derniers blocs.
2. Calculer la transformée en cosinus de chacun des blocs. Remarque: aucune information n'est encore perdue
3. Arrondir (quantifier) les transformées de chacun des blocs. **La destruction de l'information a lieu à ce stade**. Plus l'arrondi sera important (concrètement, cela revient à arrondir à 0 plus de coefficients), plus la destruction sera perceptible.

Compression destructive (avec pertes)
Transformations orthogonales

Transformée en cosinus 2D: méthode de compression

12	10	8	10	12	10	8	11
11	12	10	8	10	12	10	8
8	11	12	10	8	10	12	10
10	8	11	12	10	-8	10	12
12	10	8	11	12	10	8	10
10	12	10	8	11	12	10	8
8	10	12	10	8	11	12	10
10	8	10	12	10	8	11	12

(a) Original data

81	0	0	0	0	0	0	0
0	1.57	0.61	1.90	0.38	1.81	0.20	0.32
0	0.61	0.71	0.35	0	0.07	0	0.02
0	1.90	0.35	4.76	0.77	3.39	0.25	0.54
0	0.38	0	0.77	8.00	0.51	0	0.07
0	1.81	0.07	3.39	0.51	1.57	0.56	0.25
0	0.20	0	0.25	0	0.56	0.71	0.29
0	0.32	0.02	0.54	0.07	0.25	0.29	0.90

(b) DCT coefficients

81	0	0	0	0	0	0	0
0	2	1	2	0	2	0	0
0	1	1	0	0	0	0	0
0	2	0	5	1	3	0	1
0	0	0	1	8	1	0	0
0	2	0	3	1	2	1	0
0	0	0	0	0	1	1	0
0	0	0	1	0	0	0	1

(c) Quantized

12.29	10.26	7.92	9.93	11.51	9.94	8.18	10.97
10.90	12.06	10.07	7.68	10.30	11.64	10.17	8.18
7.83	11.39	12.19	9.62	8.28	10.10	11.64	9.94
10.15	7.74	11.16	11.96	9.90	8.28	10.30	11.51
12.21	10.08	8.15	11.38	11.96	9.62	7.68	9.93
10.09	12.10	9.30	8.15	11.16	12.19	10.07	7.92
7.87	9.50	12.10	10.08	7.74	11.39	12.06	10.26
9.66	7.87	10.09	12.21	10.15	7.83	10.90	12.29

(d) Reconstructed data (good)

Compression destructive (avec pertes)
Transformations orthogonales

Transformée en cosinus 2D: méthode de compression

8	10	9	11	11	9	9	12
11	8	12	8	11	10	11	10
9	11	9	10	12	9	9	8
9	12	10	8	8	9	8	9
12	8	9	9	12	10	8	11
8	11	10	12	9	12	12	10
10	10	12	10	12	10	10	12
12	9	11	11	9	8	8	12

(a) Original data

79.12	0.98	0.64	1.51	0.62	0.86	1.22	0.32
0.15	1.64	0.09	1.23	0.10	3.29	1.08	2.97
1.26	0.29	3.27	1.69	0.51	1.13	1.52	1.33
1.27	0.25	0.67	0.15	1.63	1.94	0.47	1.30
2.12	0.67	0.07	0.79	0.13	1.40	0.16	0.15
2.68	1.08	1.99	1.93	1.77	0.35	0	0.80
1.20	2.10	0.98	0.87	1.55	0.59	0.98	2.76
2.24	0.55	0.29	0.75	2.40	0.05	0.06	1.14

(b) DCT coefficients

79	1	1	2	1	1	1	0
0	2	0	1	0	3	1	3
1	0	3	2	0	1	2	1
1	0	1	0	2	2	0	10
20	1	0	1	0	10	0	0
3	1	2	2	2	0	0	1
1	2	1	1	2	1	1	3
2	1	0	1	2	0	0	1

(c) Quantized

7.59	9.23	8.33	11.88	7.12	12.47	6.98	8.56
12.09	7.97	9.3	11.52	9.28	11.62	10.98	12.39
11.02	10.06	13.81	6.5	10.82	8.28	13.02	7.54
8.46	10.22	11.16	9.57	8.45	7.77	10.28	11.89
9.71	11.93	8.04	9.59	8.04	9.7	8.59	12.14
10.27	13.58	9.21	11.83	9.99	10.66	7.84	11.27
8.34	10.32	10.53	9.9	8.31	9.34	7.47	8.93
10.61	9.04	13.66	6.04	13.47	7.65	10.97	8.89

(d) Reconstructed data (bad)

Compression destructive (avec pertes)
Transformations orthogonales

Transformée en cosinus 2D: Interprétation de la transformation

Dans le cas 1D, il est facile de voir la transformée en cosinus comme la projection du vecteur d'origine sur une base (orthonormée) formée de cosinus possédant une fréquence croissante. Par exemple, pour n=3 (sans normalisation) :

$$\begin{pmatrix} C_0 \\ C_1 \\ C_2 \end{pmatrix} = \begin{pmatrix} \cos 0 & \cos 0 & \cos 0 \\ \cos(\pi/6) & \cos(3\pi/6) & \cos(5\pi/6) \\ \cos[2(\pi/6)] & \cos[2(3\pi/6)] & \cos[2(5\pi/6)] \end{pmatrix} \begin{pmatrix} p_0 \\ p_1 \\ p_2 \end{pmatrix}$$

Conséquence

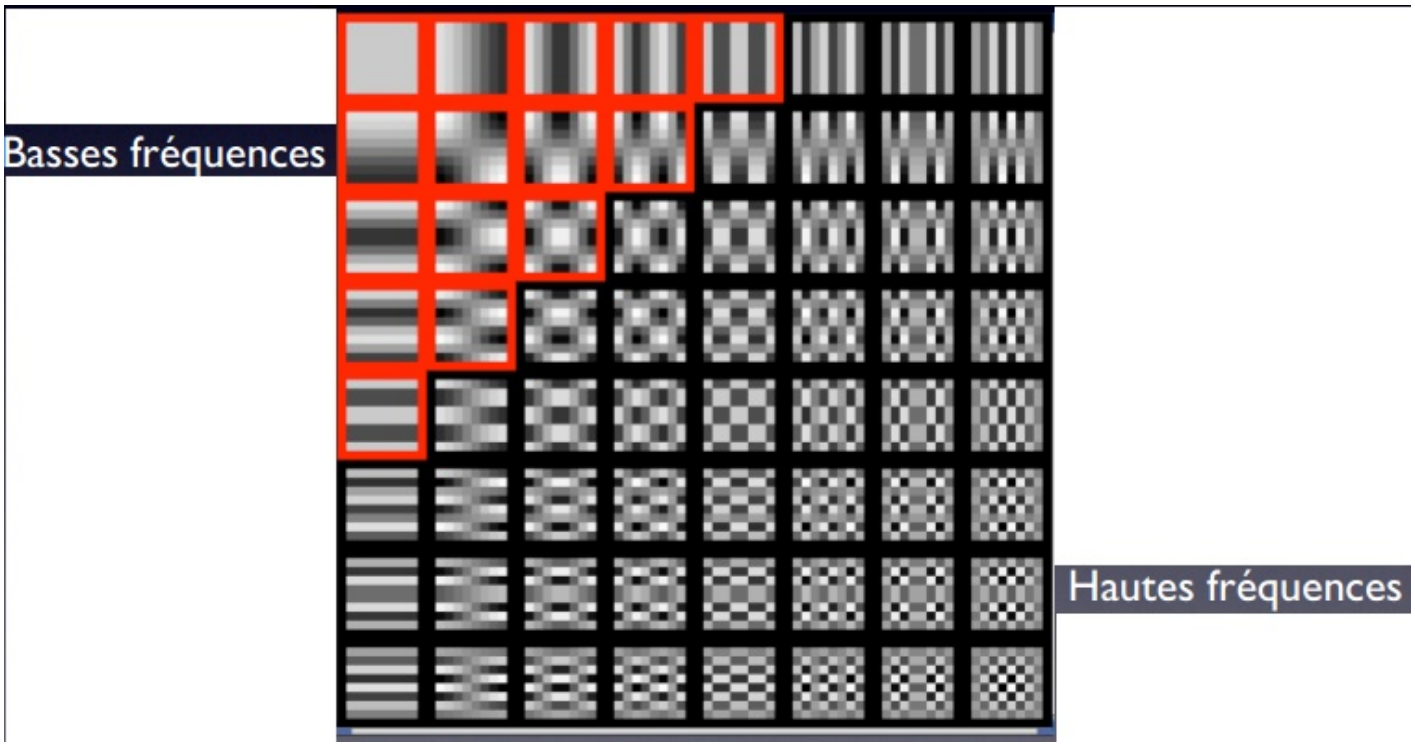
Ainsi, tout vecteur p de longueur 3 peut s'exprimer sous la forme d'une combinaison linéaire de ces 3 vecteurs, les pondérations étant les coefficients de la transformation en cosinus

Cette remarque reste évidemment valable également dans le cas 2D. Cette fois, les "vecteurs de base" sont 2D, et sont traditionnellement représentés dans le cas n=8.

Compression destructive (avec pertes)
Transformations orthogonales

Transformée en cosinus 2D: Interprétation de la transformation

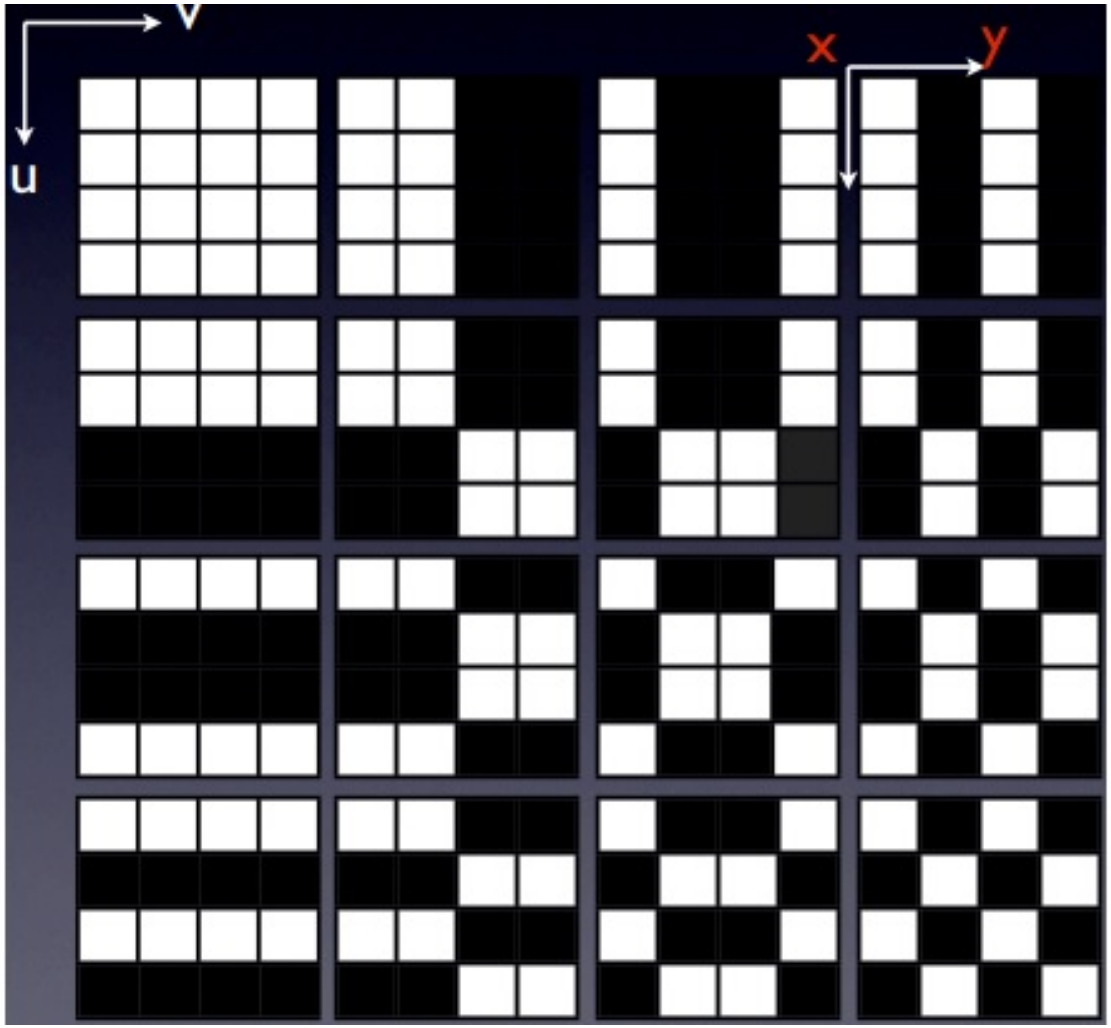
Représentation graphique des 64 bases d'image :



Compression destructive (avec pertes)
Transformations orthogonales

Retour sur la ransformée de Walsh-Hadamard

noir : -1
blanc : +1
N = 4



Compression destructive (avec pertes)
Transformations orthogonales

Transformée en cosinus 2D: Interprétation de la transformation

Implémentation pratique : il existe 4 formes de transformée en cosinus, appelées DCT-I, DCT-II, DCT-III et DCT-IV, et définies par :

$$C_k^I = \frac{1}{2}(p_0 + (-1)^k p_{n-1}) + \sum_{i=1}^{n-2} p_n \cos\left[\frac{\pi}{n-1} ik\right]$$

$$C_k^{II} = \sum_{i=0}^{n-1} p_i \cos\left[\frac{\pi}{n}(i+\frac{1}{2})k\right]$$

→ la DCT de type II est la plus utilisée

$$C_k^{III} = \frac{1}{2}p_0 + \sum_{i=1}^{n-1} p_i \cos\left[\frac{\pi}{n} i(k+\frac{1}{2})\right]$$

→ la DCT de type III est la transformée inverse de la DCT-II

$$C_k^{IV} = \sum_{i=0}^{n-1} p_n \cos\left[\frac{\pi}{n}(i+\frac{1}{2})(k+\frac{1}{2})\right]$$

Compression destructive (avec pertes)
Transformations orthogonales

Transformée en cosinus 2D: Interprétation de la transformation

$$C_{ij} = \frac{2}{\sqrt{mn}} K_i K_j \sum_{x=0}^{n-1} \sum_{y=0}^{m-1} p_{xy} \cos\left[\frac{(2x+1)i\pi}{2n}\right] \cos\left[\frac{(2y+1)j\pi}{2m}\right]$$

1. Quelle que soit la taille de l'image, il n'y a que 32 cosinus impliqués dans le calcul de la TCD 2D. Ceux-ci peuvent être précalculés de façon à améliorer le temps de calcul.
2. La double somme précédente peut s'exprimer sous la forme matricielle où P est la matrice 8x8 des pixels, et les éléments de C s'expriment sous la forme :

$$C_{ij} = \frac{1}{2} \cos\left[\frac{(2j+1)i\pi}{16}\right]$$

Merci pour votre attention!