

Signaux, Sons et Images pour l'Informaticien: compression des images

Diane Lingrand

Polytech SI3

2016 - 2017

- Idées pour la compression
- Notions de théorie de l'information
- Codages utilisés dans les formats GIF, PNG et JPEG :
 - Codage LZW
 - Codage d'Huffman
- Algorithmes utilisés pour la compression GIF, PNG et JPEG

Ce qui est attendu des algorithmes de compression

- Rapidité de la compression / décompression
- Robustesse lors de la décompression (pertes éventuelles de données)
- Taux de compression (taille du fichier compressé)
- Qualité :
 - meilleur possible (humain / matériel)
 - suffisante pour distinguer certains éléments
 - suffisante pour appliquer des traitements dessus
- Quantité d'informations
 - avec ou sans pertes

- Réduire le nombre de couleurs
 - quantification
- Réduire le nombre de pixels
 - scaling
 - réduction des canaux chromatiques par rapport à l'intensité
- Mémoriser les motifs répétitifs
- Codage entropique
 - les éléments fréquents sont stockés sur moins de bits que les éléments rares
- Espace de représentation plus compact
 - transformée en cosinus discrète
 - ondelettes

- taille des supports
 - CD : 650 Mo
 - DVD : 4.7Go / 8.5Go
 - clef USB : 8Go / 512Go
 - carte mémoire (compact, SD) : 256 Go
 - disque dur : 16 To
- taille des images
 - appareil photo numérique : 20 millions de pixels
 - sans compression, RGB, 1 octet par composante
 - 1 image = 60 Mo
 - 1 CD : 11 images
 - carte mémoire : 4250 images
 - un disque de 16To : 270 000 images



- Comment évaluer la quantité d'informations contenue dans une image? *Théorie de l'Information*
 - chaque point d'une image est considéré comme une variable aléatoire
 - une image est considérée comme un ensemble de $w * h$ variables aléatoires
- Soit P un point d'une image :
 - valeurs possibles de la variable aléatoire : $[0 ; 255]$
 - variable indépendante des autres variables (points de l'image)
 - soit $p(n_i)$ la probabilité pour que le niveau de gris en P soit n_i
 - quantité d'information :

$$QI(n_i) = \log_a\left(\frac{1}{p(n_i)}\right) = -\log_a(p(n_i))$$

Entropie d'une image



- par analogie avec la thermodynamique
- entropie d'un point (exprimée en bits) :

$$H(P) = \sum_{i=0}^{255} p(n_i) QI(n_i) = - \sum_{i=0}^{255} p(n_i) \log_2(p(n_i))$$

- entropie d'un bloc de pixels :
 - sous hypothèse de stationnarité et d'ergodicité : $H(P^L) = LH(P)$

Théorème de codage sans pertes

- Permet de déterminer le nombre de bits minimal pour coder une image
- On code les pixels par blocs (1 mot binaire par bloc)
- La taille des mots mot_i est variable : ℓ_i
- Longueur moyenne :

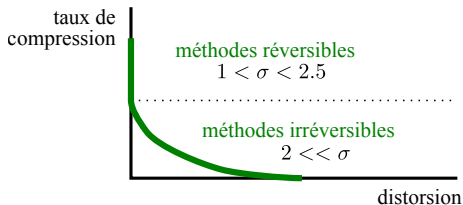
$$\ell = \frac{1}{L} E(\{\ell_i\}) \text{ avec : } E(\{\ell_i\}) = \sum_{i=1}^m p(\ell_i) \ell_i$$

- Théorème du codage :

$$\forall \delta > 0 \exists L \forall \text{mot}_i \quad \boxed{H(P) \leq \ell \leq H(P) + \delta}$$

Taux de compression et distorsion

- Taux de compression σ : rapport taille image originale / taille image compressée



- Distorsion :
 - erreur quadratique moyenne (EQM ou MSE) :

$$EQM = \frac{1}{N} \sum_{i=1}^N (\hat{n}_i - n_i)^2$$

- rapport signal à bruit crête (PSNR), en dB :

$$PSNR = 10 \log_{10} \left(\frac{255^2}{EQM} \right)$$

- plusieurs versions : LZ77, LZ78, LZW
- utilisé dans : gif(LZW, 8 bits), tiff (LZ77, optionnel), png (LZ77, optionnel), .Z, .gzip
- LZW fait l'objet d'un copyright
- compression sans pertes
- fonctionne bien pour des images avec zones homogènes (horizontales)
- principe : découpage des pixels en mots de longueurs différentes et attribution d'un code, de même longueur, à chaque mot

Algorithme de codage LZW

```
mot <- ""
tant que (lecture c)
  si concat(mot,c) dans dic
    mot <- concat(mot,c)
  sinon
    ajouter concat(mot,c) dans dic
    retourner code(mot)
  mot <- c
```

Codage de : ABRACADABRACADA...

mot	c	mot+c	existe ?	retour	entrée	@
A	B	AB	non	@(A)	AB	@ ₀
B	R	BR	non	@(B)	BR	@ ₀ +1
R	A	RA	non	@(R)	RA	@ ₀ +2
A	C	AC	non	@(A)	AC	@ ₀ +3
C	A	CA	non	@(C)	CA	@ ₀ +4
A	D	AD	non	@(A)	AD	@ ₀ +5
D	A	DA	non	@(D)	DA	@ ₀ +6
A	B	AB	oui			
	R	ABR	non	@(AB)=@ ₀	ABR	@ ₀ +7
R	A	RA	oui			
	C	RAC	non	@(RA)=@ ₀ +2	RAC	@ ₀ +8
C	A	CA	oui			
	D	CAD	non	@(CA)=@ ₀ +4	CAD	@ ₀ +9

Algorithme de décodage LZW

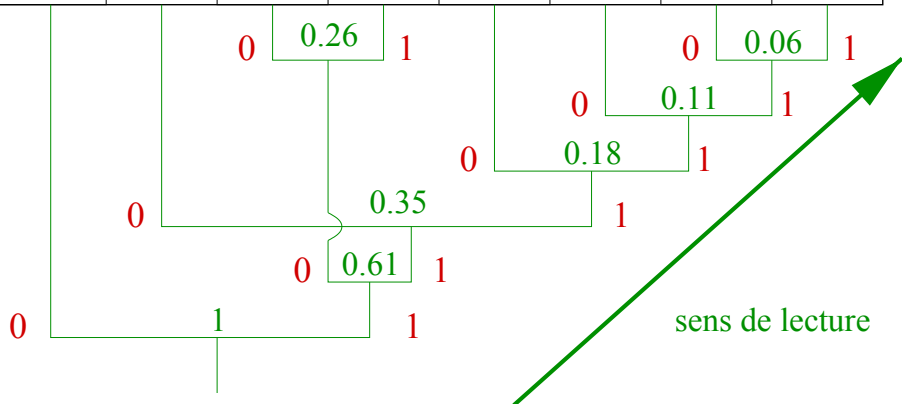
```
prec <- ""  
tant que (lecture d'un code)  
  mot <- contenu stocké à @ code  
  retourner mot  
  c <- 1er caractère de mot  
  à la 1ère adresse libre :  
    stocker concat(prec,c)  
  prec <- mot
```

code reçu	mot courant	sortie c	entrée	@	mot
A	A	A			A
B	B	B B	AB	@ ₀	B
R	R	R R	BR	@ ₀ +1	R
A	A	A A	RA	@ ₀ +2	A
C	C	C C	AC	@ ₀ +3	C
A	A	A A	CA	@ ₀ +4	A
D	D	D D	AD	@ ₀ +5	D
100	AB	AB A	DA	@ ₀ +6	AB
102	RA	RA R	ABR	@ ₀ +7	RA
104	CA	CA C	RAC	@ ₀ +8	CA

- codage entropique (longueur variable)
- code préfixe (aucun code n'est préfixé par un autre code)
- 2 phases :
 - phase descendante : construction de l'arbre
 - phase ascendante : codage de l'information

Codage d'Huffman : construction de l'arbre et codage

0	110	100	101	1110	11110	111110	111111
A	B	C	D	E	F	G	H
0.39	0.17	0.14	0.12	0.07	0.05	0.04	0.02



Codage d'Huffman : exemple

- Codage d'un mot
 - codage ordinaire : sommes des $p(n_i) * 3$ bits
 - codage d'Huffman : somme des $p(n_i) * l_i$ bits
 - pour notre exemple : 3 bits contre 2.57 bits
 - pour une image (640*480) : économie de 132096 bits soit environ 13 kO
- Codage de l'arbre
 - statique : pas besoin de coder l'arbre
 - semi-statique : arbre calculé 1 fois pour toutes les données
 - adaptatif : modifications d'un arbre non transmis

Différentes possibilités :

- sans compression
- avec compression (sans pertes)
 - prédiction des données (DPCM = Differential pulse-code modulation)
 - algorithme Deflate = combinaison de LZW puis Huffman (arbre prédéfini ou non)

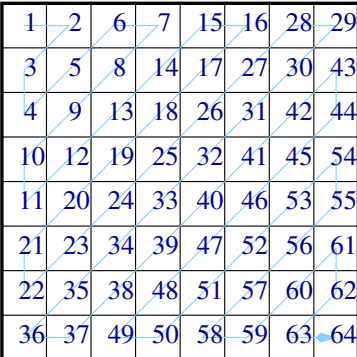
Transformée en cosinus discrète (DCT)

- coefficients réels et plus petits
- espace DCT plus approprié à la dynamique des images

$$n_{\text{dct}}(u, v) = \frac{2}{N} C(u) C(v) \sum_{i=0}^{N-1} \sum_{j=0}^{N-1} \cos \left[\frac{\pi}{N} u \left(i + \frac{1}{2} \right) \right] \cos \left[\frac{\pi}{N} v \left(j + \frac{1}{2} \right) \right] n(i, j)$$

avec $C(0) = \frac{1}{\sqrt{2}}$ et $\forall \alpha \quad C(\alpha) = 0$

- Découpage de l'image en blocs de 8x8 pixels
- Sur chaque bloc :
 - transformée DCT
 - quantification des valeurs
 - parcours en zigzag
 - RLC sur les zéros
 - codage d'Huffman



An 8x8 grid representing a block of pixels. The cells are numbered from 1 to 64 in a zigzag pattern, starting from the top-left corner (1) and ending at the bottom-right corner (64). Blue arrows indicate the path of the zigzag traversal, showing the sequence of pixels visited.

1	2	6	7	15	16	28	29
3	5	8	14	17	27	30	43
4	9	13	18	26	31	42	44
10	12	19	25	32	41	45	54
11	20	24	33	40	46	53	55
21	23	34	39	47	52	56	61
22	35	38	48	51	57	60	62
36	37	49	50	58	59	63	64

Exemples de compression JPEG

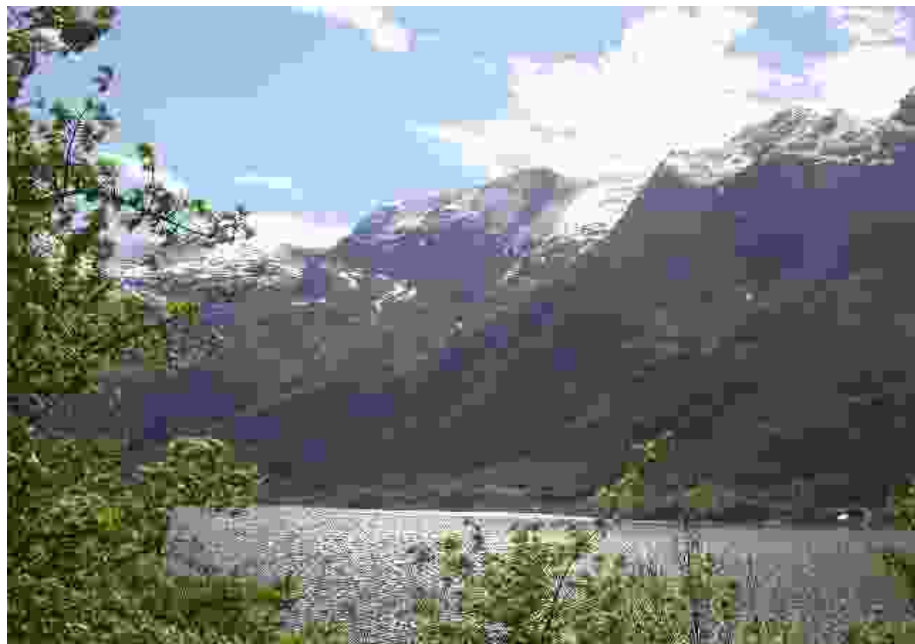
image originale (640x480) : JPEG 88kO (900kO non compressée) $\sigma = 10.2$











Compression JPEG à 1% ; 8kO ; $\sigma = 112.5$



- Base orthogonale :

$$\psi_{m,n}(x) = 2^{-\frac{m}{2}} \psi(2^m(x - n))$$

- Coefficients d'ondelettes :

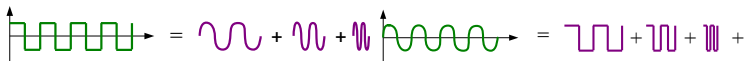
$$c_{m,n}(f) = \langle f, \psi_{m,n} \rangle = \int f(x) \bar{\psi}_{m,n}(x) dx$$

- Base d'ondelettes de Haar :

$$f(x) = \sum_{m,n} c_{m,n}(f) \psi_{m,n}(x)$$

avec

$$\psi(x) = \begin{cases} 1 & \text{si } x \in [0, \frac{1}{2}[\\ -1 & \text{si } x \in [\frac{1}{2}, 1[\\ 0 & \text{ailleurs} \end{cases}$$



- la décomposition DCT est remplacée par une décomposition sur base d'ondelettes de Haar
- autres propriétés :
 - organisation progressive du train binaire
 - taux de compression sans pertes meilleur que JPEG
 - possibilité de faire varier le taux de compression selon les régions de l'image

Comparaison JPEG / JPEG 2000



JPEG à 65 % : 80 kO

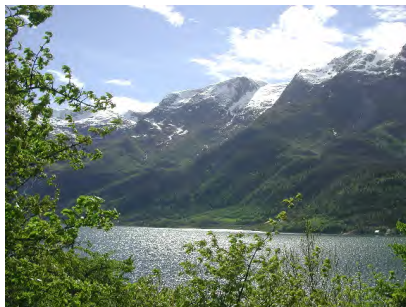


JPEG 2000 : 80 kO

Comparaison JPEG / JPEG 2000



JPEG à 30 % : 40 kO

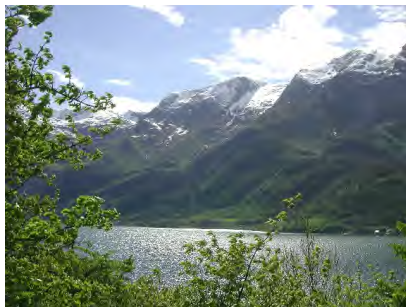


JPEG 2000 : 40 kO

Comparaison JPEG / JPEG 2000

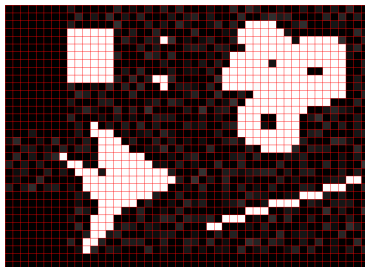


JPEG à 12 % : 20 kO



JPEG 2000 : 20 kO

- GIF : pour les logos de petites tailles avec peu de couleurs
- PNG : pour n'avoir aucune perte
- JPEG : pour compresser plus



- Comment coder l'image de Lena sur 1 bit ?



- Comment coder l'image de Lena sur 1 bit ?



- Le bit vaut :
 - True si c'est l'image de Lena
 - False sinon