

Les données numériques: compression des images

Diane Lingrand



2022 - 2023

- Idées pour la compression
- Codages utilisés dans les formats GIF, PNG et JPEG :
 - Codage LZW
 - Codage d'Huffman
- Algorithmes utilisés pour la compression GIF, PNG et JPEG
- Principe de compression video et nouveaux formats de compression d'images

Ce qui est attendu des algorithmes de compression

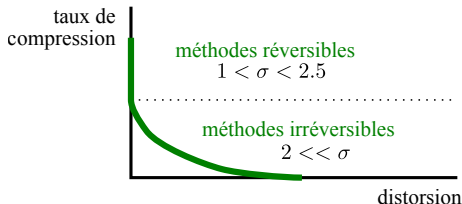
- Rapidité de la compression / décompression
- Robustesse lors de la décompression (pertes éventuelles de données)
- Taux de compression (taille du fichier compressé)
- Qualité :
 - meilleur possible (humain / matériel)
 - suffisante pour distinguer certains éléments
 - suffisante pour appliquer des 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

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 a fait l'objet d'un copyright. Domaine libre depuis au moins 2006 !
- 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

```

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


```
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'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)

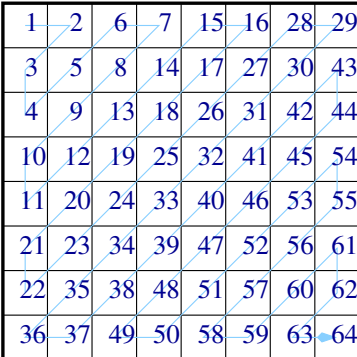
Transformée en cosinus discrète (DCT)

- 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
 - codage de tous les coefficients (1) par différence entre blocs voisins
 - codage des autres coefficients (2) à (64) par bloc
 - arbres prédéfinis pour les 2 types de coefficients



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

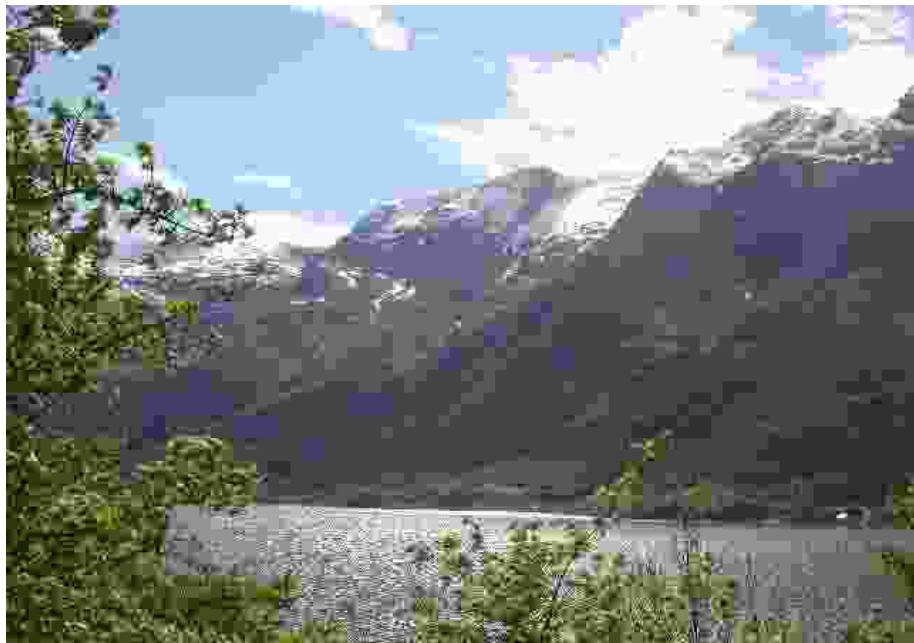
image originale (640x480) : JPEG 88kO (900kO non compressée) $\sigma = 10.2$













- Base orthogonale :

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

- Coefficients d'ondelettes :

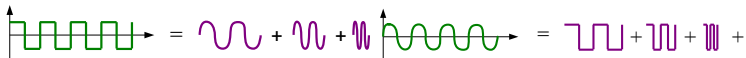
$$c_{m,n}(f) = \langle f, \psi_{m,n} \rangle = \int f(x) \bar{\psi}_{m,n}(x) dx$$

- 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

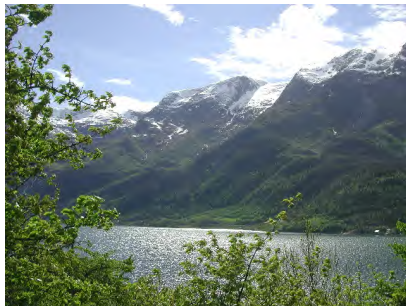


JPEG 2000 : 80 kO

Comparaison JPEG / JPEG 2000



JPEG à 30 % : 40 kO

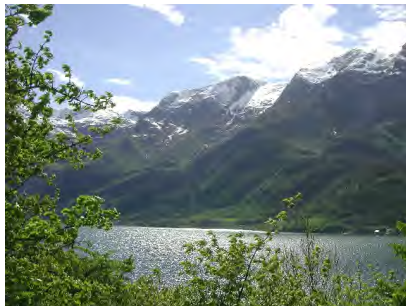


JPEG 2000 : 40 kO

Comparaison JPEG / JPEG 2000



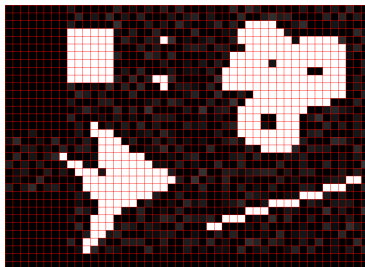
JPEG à 12 % : 20 kO



JPEG 2000 : 20 kO

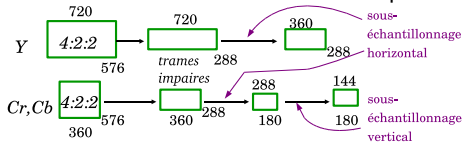
Conclusion (provisoire)

- GIF (1987) : pour les logos de petites tailles avec peu de couleurs
- PNG (1997) : pour n'avoir aucune perte
- JPEG (1992) : pour compresser plus (pas de transparence, sauf JPEG XR)
- animations : GIFs ou extensions de PNG (APNG, MNG)
- JPEG 2000 : peu adopté au final

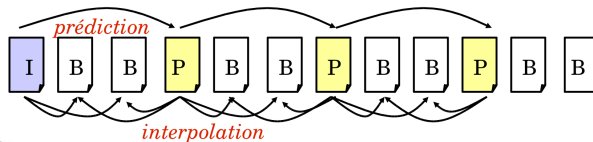


Principe de codage video - MPEG

- GOP (*Group Of Pictures*) : 3 types d'images
 - I (intra) : au format SIF (*Source Intermediate Format*)
 - réduction de moitié de la résolution spatiale et temporelle

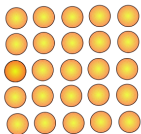


- P (prédite) : vecteurs de mouvement par macroblocs
 - 4 blocs de luminance
 - 2 ou 4 blocs de chrominance
- B (bidirectionnelle) : interpolation I ou P passée et future

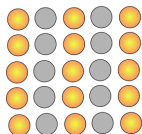


Diversion : codage luminance/chrominance

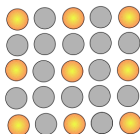
- 4 correspond à une référence (fréquence d'échantillonnage de 13.5MHz).
- DVD : 4:2:0



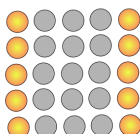
4:4:4



4:2:2



4:2:0



4:1:1

- volonté de regrouper dans un même format
 - compression sans perte
 - compression avec pertes
 - animations (GIFs, ...)
 - gestion de la transparence
- mais aussi augmenter le taux de compression à “qualité équivalente” .
 - notamment pour le web
- format globalement bien supporté
 - il faudra néanmoins ajouter des extensions ou *plugins* pour certaines applications
 - ou les convertir, par exemple :
 - `ffmpeg -i monImage.webp monImage.jpg`
- Certains l'ont utilisé pour le TP en 2020 (en nommant les fichiers .jpg !)

- basé sur le codage video VP8
 - 8-bit, YUV, format 4:2:0
 - utilisation de macro-blocs (4x4 et 16x16 en luminance, 8x8 en chrominance)
 - prédiction de blocs selon plusieurs types : horizontale (copie de la colonne de gauche), verticale (copie de la ligne du dessus), moyenne (de la colonne de gauche et ligne du dessus), mouvement.
 - quantification adaptative des blocs (en fonction de l'entropie des blocs)
 - codage arithmétique binaire

- Codage entropique qui, contrairement à Huffman, code un ensemble de symboles et non pas les symboles individuellement et peut ainsi être plus efficace.
- algorithme de codage des symboles
 - on découpe l'intervalle $[0 \ 1]$ en intervalles pour chaque symbole, de taille proportionnelle à son occurrence (probabilité)
 - initialisation : $\min = 0$ et $\max = 1$
 - lecture d'un symbole s (selon sa valeur, on recherche les bornes s_{\min} et s_{\max})
 - calcul de la taille de l'intervalle : $l = \max - \min$
 - $\min \leftarrow \min + l * s_{\min}$
 - $\max \leftarrow \min + l * s_{\max}$
 - quand tous les symboles sont lus, on renvoie un réel r dans l'intervalle $[\min \ \max]$
- algorithme de décodage du symbole r
 - chercher son intervalle : $[\min \ \max]$ et le symbole associé s
 - remplacer r par $(r - \min) / (\max - \min)$
 - jusqu'au symbole de fin

Codage arithmétique : un exemple

- codage d'une image à 4 couleurs : ROUGE, VERT, BLEU, JAUNE.
 - 50% des pixels sont ROUGES, 25% BLEUS, 12.5% VERTS et 12.5% JAUNES



- on utilise les probabilités pour diviser l'intervalle [0 1]



- puis on va lire l'image de gauche à droite et de haut en bas :
 - initialisation : $\min = 0$ et $\max = 1$
 - 1er pixel rouge : $\min = 0$ et $\max = 0.5$ donc \min prend 0 et \max 0.5
 - 2ème pixel bleu : $\min = 0.5$ et $\max = 0.75$ et $l = 0.5$ donc \min prend 0.25 et \max prend 0.375 ...
- décodage : on considère l'image dont le code vaut 0.765
 - le premier pixel est donc jaune avec $\min = 0.75$ et $\max = 0.875$
 - r prend alors pour valeur $0.015/0.125=0.12$: le pixel suivant est rouge avec $\min = 0$ et $\max = 0.5$
 - r prend alors pour valeur $0.12/0.5=0.24$: encore rouge, puis 0.48 rouge puis 0.96 vert ...

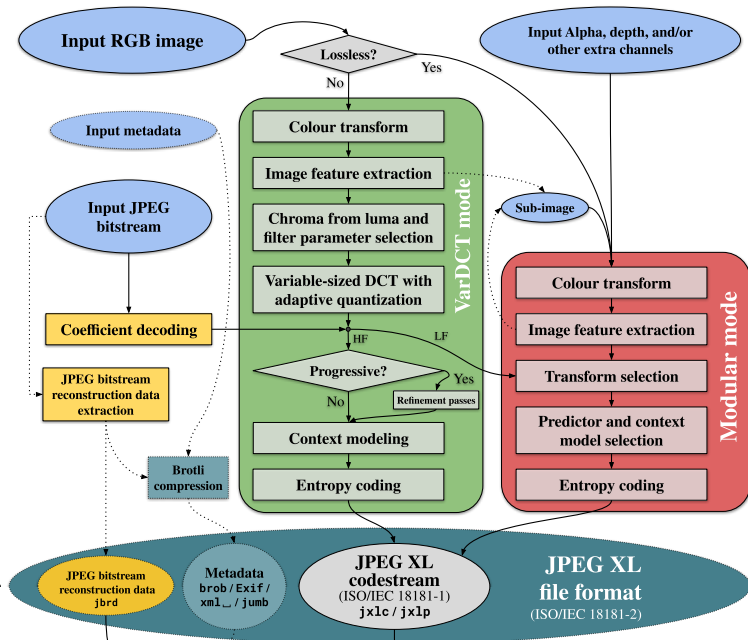
Better Portable Graphics (BPG), 2014

- codage avec ou sans pertes
- reprend le codage des images dans le codage video HECV
 - mais encodage et décodage sous license libre
- 4:4:4, 4:2:2, 4:2:0
- différents espaces de couleurs avec/sans transparence
 - 8 à 14 bits par canal
- animations
- meilleurs performances que JPEG (comparer avec <http://xooyoozoo.github.io/yolo-octo-bugfixes/#fruits&jpg=t&webp=t>)
- CABAC

- utilisé en video (H.264/MPEG4 AVC, HEVC)
- sans pertes
- codage entropique
- plusieurs étapes :
 - binarisation : 1 symbole est converti en une suite de 0 ou 1, de longueur variable
 - choix d'un modèle statistique (contexte local)
 - codage arithmétique du message binaire en fonction du contexte (0 ou 1)
 - mise à jour du contexte

- voudrait remplacer JPEG (et conserver cette place aussi longtemps;-)
 - 'L' : long term
- basé sur PIK (Google) et FUIF, successeur de FLIF (Clouldinary)
- Propriétés :
 - avec ou sans pertes
 - très hautes résolutions (1Tpixels)
 - large gamme de valeur (jusqu'à 32 bits par composantes)
 - grand nombre de canaux (RGBAD...)
 - décodage progressif
 - plusieurs frames : animation, édition (layers,...)
 - possibilité de coder des images JPEG sans perte supplémentaire mais avec gain en compression
- papier de référence (2023) :
<https://ds.jpeg.org/whitepapers/jpeg-xl-whitepaper.pdf>
- démo : https://eclipseo.github.io/image-comparison-web/#swallowtail*1:1&Original=s&MOZJPEG=t&subset1

JPEG XL



- modèle de couleurs adapté à la perception humaine

$$\begin{bmatrix} X \\ Y \\ B \end{bmatrix} = \begin{bmatrix} 1 & -1 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} L \\ M \\ S \end{bmatrix}$$

Où LMS (Long Medium Short) correspond aux réponses des trois types de cônes (S les moins présents).

- DCT sur des tailles variables (2 à 256)x(2 à 256)
- caractéristiques des images
 - détection des motifs répétitives
 - détection des courbes fines (ex :cheveux) pour représentation par splines
 - modélisation du bruit
 - filtre pour la préservation des contours (8x8)
- Performances
 - taux de compression entre 20 :1 et 50 :1
 - environ 20% de moins pour les images JPEG
 - au moins aussi rapide que JPEG en compression/décompression