Compression de données

Codage de source

www.lenna.org

I - Introduction

II- Codage sans perte

- les bases : la théorie de l'information
- codage d'Huffman
- codage à base de dictionnaire : Lempel-Ziv (Welch)
- codage arithmétique

III- Codage avec perte : quantification scalaire,

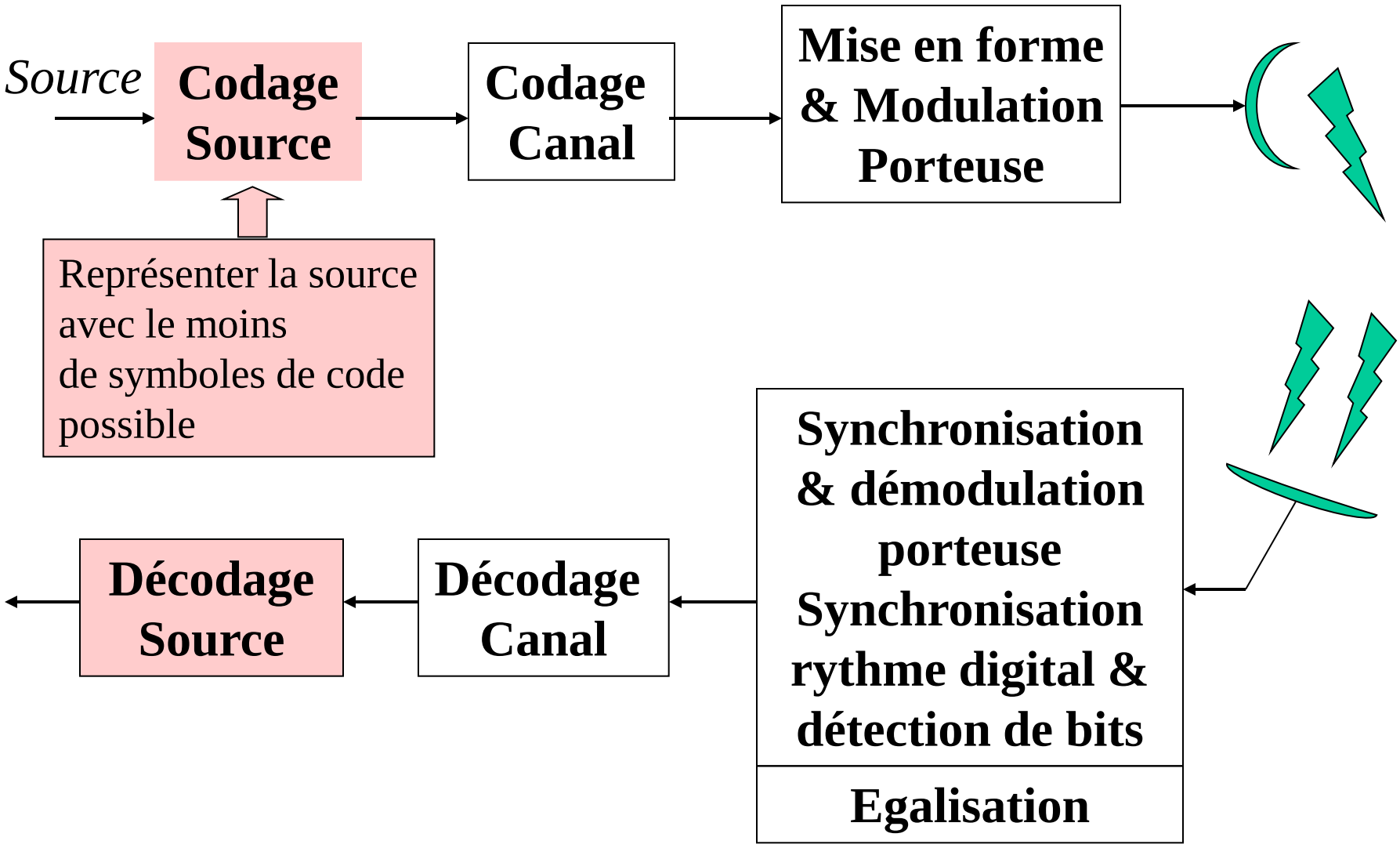
IV- Codage avec perte : méthodes prédictives,

V- Codage avec perte : méthodes par transformées,

VI- Codage avec perte : quantification vectorielle

corinne.mailhes@n7.fr

Chaîne de Transmission



Qu'est ce que la compression de données ?

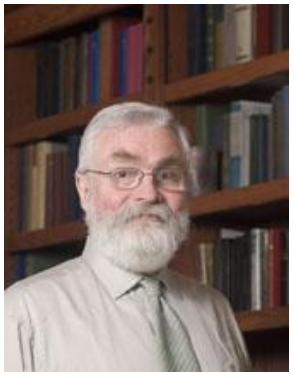
- Problème des communications
- Représenter la source avec le moins de symboles de code possible tout en préservant un certain degré de fidélité
- Bases : théorie de l ’information et Shannon
- Chiffres :

- ➢ étude du sommeil : 1 Go/nuit/patient
- ➢ parole : 64 kbps
- ➢ musique stéréo : 1.5 millions de bits / s
- ➢ CD : 44100 ech / s, 16 bits / éch, 2 canaux
- ➢ vidéo téléphone : 12 millions de bits / s
- ➢ vidéo N&B, basse résolution : 8 bits/pixel, 256x256 pixels/trame 24 trames /s
- ➢ HDTV : 1 billion de bits /s !!!

I - Introduction

Compression sans distorsion

Compression avec distorsion



http://ee.stanford.edu/~gray/

Bibliographie :

A.Gersho, R.M.Gray, « Vector Quantization and Signal Compression », Kluwer Academic, 1991.
D.Salomon, « Data compression, the complete reference », Springer, 1998.
K.Sayood, « Introduction to data compression », Morgan Kaufmann, 1996.
N.Moreau, « Techniques de compression des signaux », Ed Masson, collec. CNET / ENST.
C.Mailhes, « Théorie de l 'Information », Polycopié N7
T.M.Cover, J.A.Thomas, «Elements of Information Theory », Wiley Series Telecom, 1991.
Hrrp://ee.stanford.edu/~gray/
http://www.stanford.edu/group/compression
http://www.eas.asu.edu/~spanias/
http://www.data-compression.com/

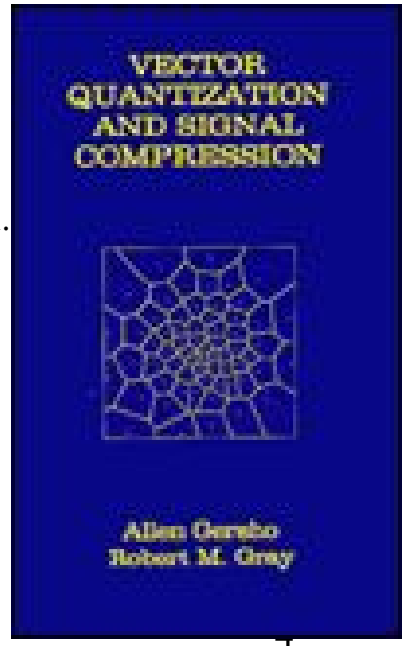
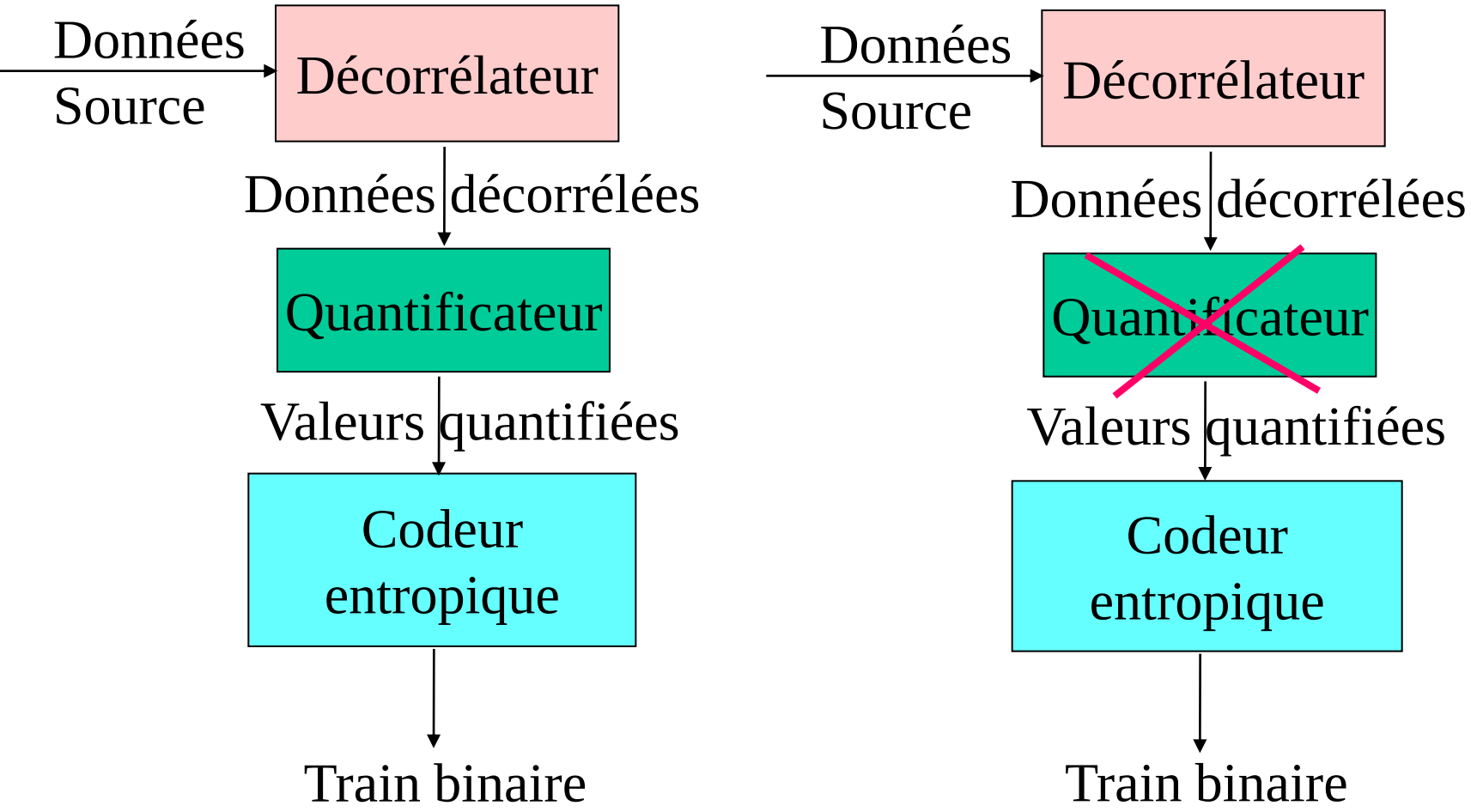


Schéma général d'un système de compression

La compression commence par éliminer la redondance



Comparer différents codeurs

Evaluation d'un algorithme de codage et comparaison :

- ***taux de bits i.e. taux de compression***
- ***qualité des données reconstruites : notion de distorsion***
- ***complexité de l'algorithme***
- ***retard introduit codage + décodage***
- ***robustesse de l'algorithme aux erreurs de canal et à l'interférence.***

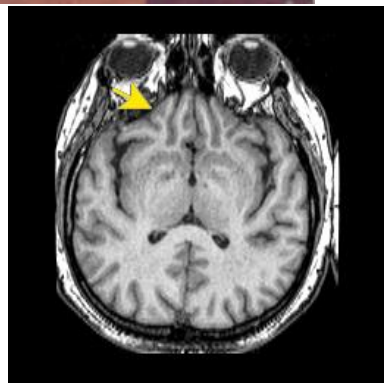
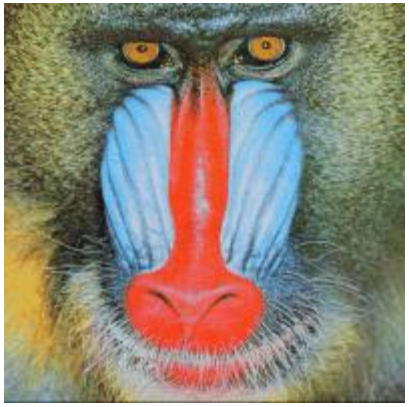
et comparer... sur quelles données ?

I - Introduction



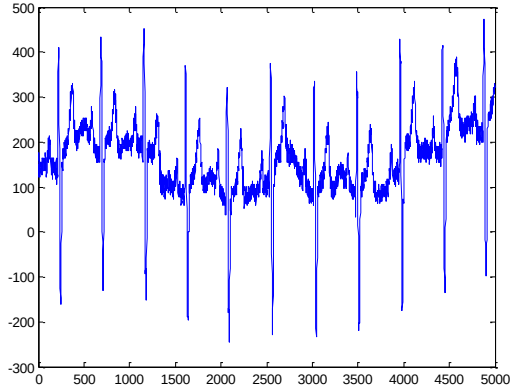


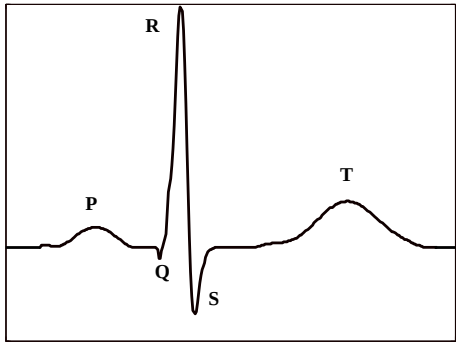


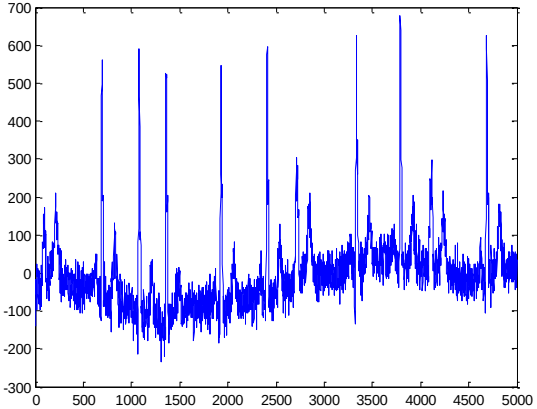



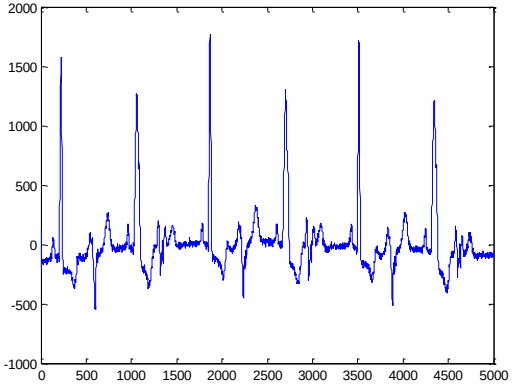

http://links.uwaterloo.ca/bragzone.base.html

I - Introduction

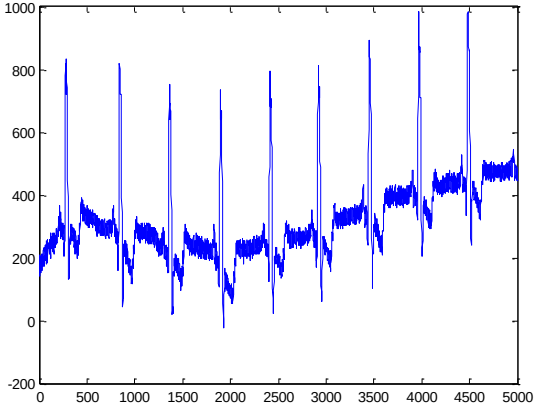


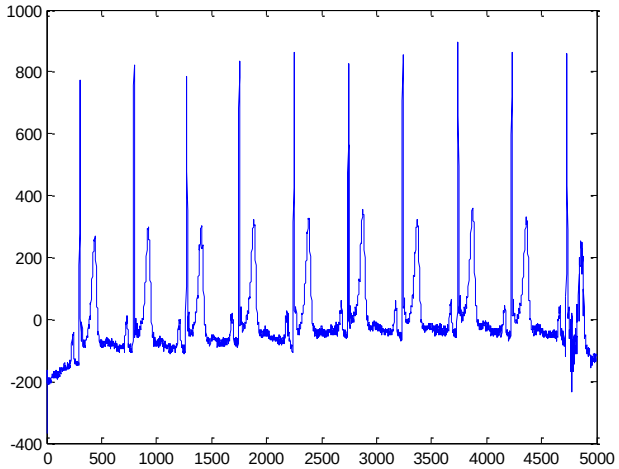


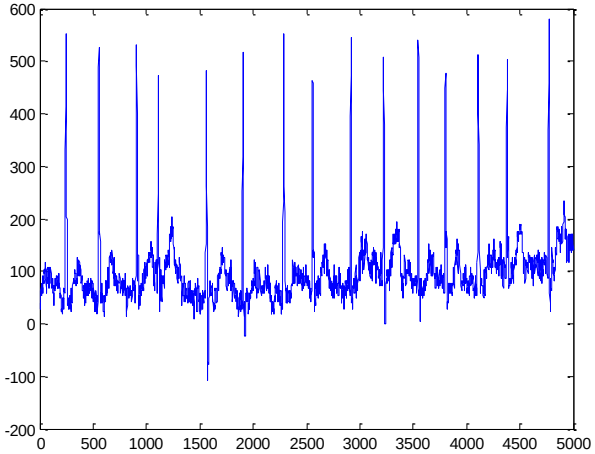




Banque d'ECG







Codage sans perte ou codage entropique

Claude Elwood Shannon (1916 - Febr 2001)

les bases : la théorie de l'information

codage d'Huffman

codage à base de dictionnaire : Lempel-Ziv (Welch)

codage arithmétique

C. E. Shannon, « A mathematical theory of communication »,
Bell System Technical Journal, vol. 27, pp. 379-423 and 623-656, July and October, 1948.
Disponible sur le web

http://cm.bell-labs.com/cm/ms/what/shannonday/shannon1948.pdf

Les bases : la théorie de l'information

Les principes énoncés par Shannon :
Comment définir la quantité d'information d'un message ?
Source émettant un message = source aléatoire
Sans aléatoire, pas d'information !

Quantité d'information d'un message :
$i(x_k) = - \log2 (p_k)$ bits
p_k étant la probabilité d'émission du message x_k

Entropie d'une source X
= quantité d'information moyenne émise par la source
= quantité de doute moyen de la source
= mesure de son imprédictibilité

$H(X) = - \sum_k p_k \log_2 (p_k)$ (bits)

Codage sans distorsion

Borne inférieure de la longueur moyenne d'un code avec D symboles

Comment construire un code irréductible* tel que :

$$H(X) / log2\,(D) \leq \bar{n} \leq H(X) / log2\,(D) + 1$$

$H(X) = - \sum_k p_k \log_2 (p_k)$ (bits) entropie de la source

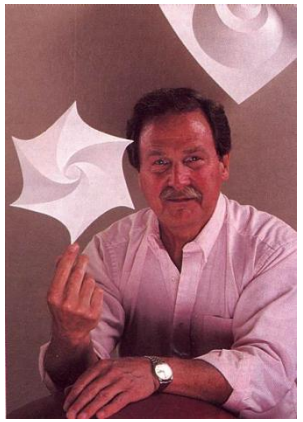
$\bar{n} = \sum_k p_k n_k$ longueur moyenne du code

Principes issus de la théorie de l'information :
1. Faire du codage à longueur variable
2. Coder des blocs de messages

- 2 approches générales :
- Par la statistique : Huffman, arithmétique
- À l'aide d'un dictionnaire

(*) Irréductible : code à longueur variable pouvant être déchiffré instantanément, sans ambiguïté (aucun mot n'est le début d'un autre)

Code d'Huffman



Code d'Huffman (1952) : codage optimal

David A. Huffman (9 août 1925 - 7 octobre 1999)

D.A. Huffman, "A method for the construction of minimum-redundancy codes", Proceedings of the I.R.E., sept 1952, pp 1098-1102

http://compression.ru/download/articles/huff/huffman_1952_minimum-redundancy-codes.pdf

Cas binaire :

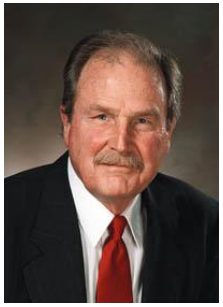
1. on classe les messages par ordre de probabilité décroissante,
2. on regroupe les 2 messages les moins probables, en leur affectant une probabilité égale à la somme des probabilités. Les 2 messages auront le même code sauf la fin : le 1er se verra affecter un symbole « 0 » et le 2ème un symbole « 1 »,
3. on refait 1 jusqu 'à épuisement.
4. La lecture des mots-code se fait en lisant le tableau ainsi constitué de gauche à droite : on lit les mots-code à l'envers (de la fin vers le début)

Cas D différent de 2 :

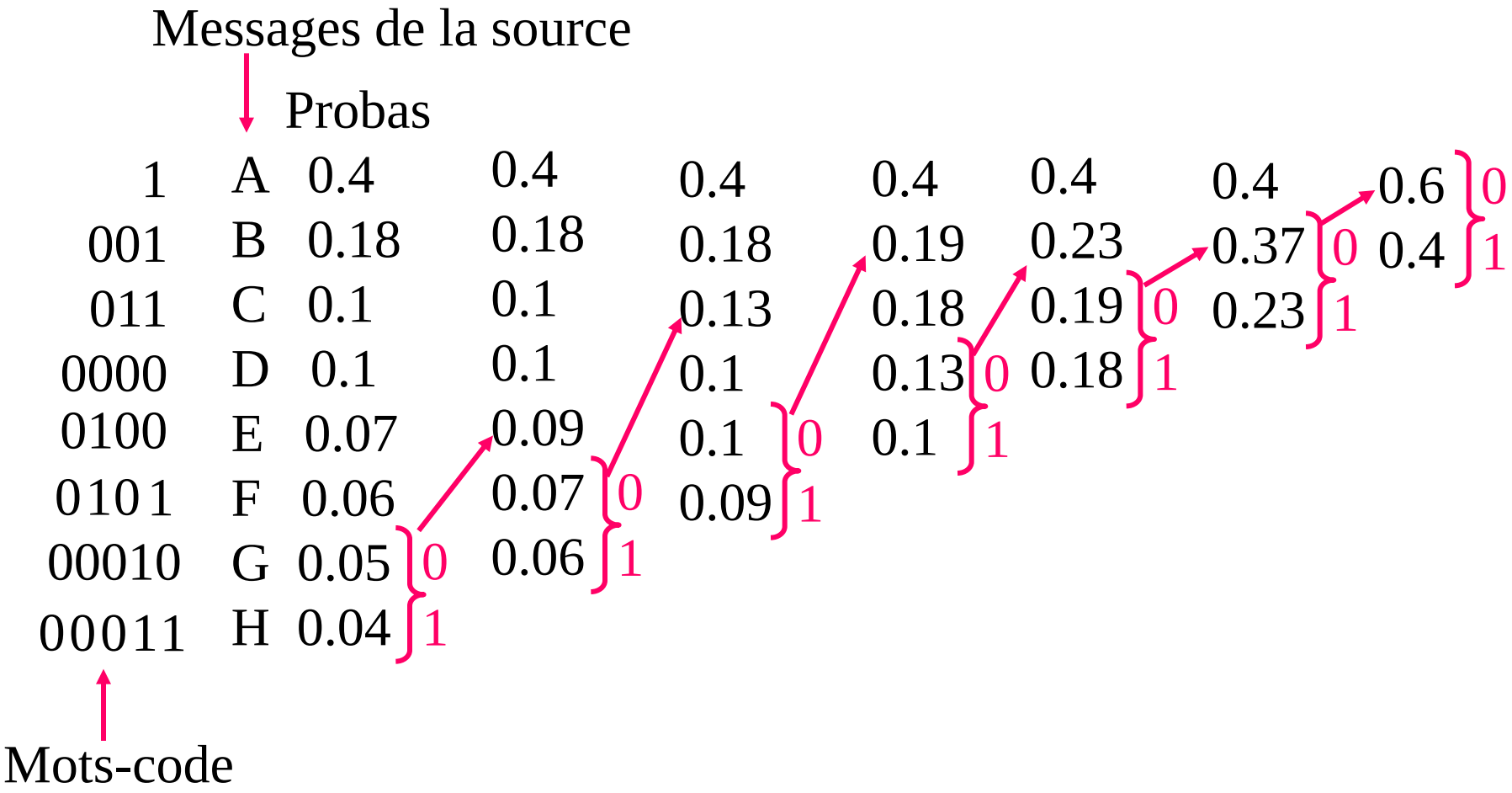
On fait de même en remplaçant « 2 » par « D » et les symboles 0 et 1 par $u_1,\ldots,u_D$

☠💣☠ Initialisation : on regroupe non pas D symboles à la 1ère itération mais :

2 + reste de la division de N-2 par D-1



Code d'Huffman : un exemple



Longueur moyenne atteinte par Huffman : 2.61
H(X)=2.55 bits soit une efficacité de E=97.8%

Code d'Huffman

Il existe trois variantes de l'algorithme de Huffman :

- statique : probabilités fixées au départ. La table de probas connue du décodeur par défaut.
- semi-adaptatif : algorithme à 2 passes.

  1- Le fichier est d'abord lu, de manière à estimer les probas, le code est construit,

  2- Codage

Il sera nécessaire pour la décompression de transmettre l'arbre.

- adaptatif : méthode offrant a priori les meilleurs taux de compression :

les probas sont mises à jour de manière dynamique au fur et à mesure de la compression.
Gros désavantage : devoir reconstruire le code à chaque fois,
ce qui implique un temps d'exécution énorme.

Faller (1973), Gallagher (1978) prennent en compte statistique
des messages déjà rencontrés
Knuth (1985), Vitter (1987) améliorent

Machine de Rice : plusieurs codes, prendre le meilleur

4 codes d'Huffman sur Voyager 1978-1990
32 codes d'Huffman dans MPEG Audio layer 3

II- Codage sans perte

Si les probabilités sont inconnues, Huffman n'est plus optimal...
Comment faire mieux ?

En s'appuyant sur principe énoncé par Shannon

$$H(X) / log2\,(D) \leq \overline{n}$$

Comment se rapprocher de la limite inférieure ?

Théorème de Shannon *: il faut coder les messages*
Non pas individuellement mais par blocs de plusieurs messages
Plus on groupe des messages, plus on se rapproche de la limite…

Mais avec Huffman, cela augmente la complexité du code (dictionnaire !!)

Mieux qu'Huffman sur données ASCII : Codes à base de dictionnaire

Lempel-Ziv (Welch) (1977-1978, 1984)

les poules du couvent couvent…

Codeur et décodeur au départ connaissent le code ASCII (8bits)

Code ASCII :

0 : _

⋮

255 : ÿ

LZ77 :
PKZIP
Lharc
ARJ

LZ78 (LZW)

LZ78 :
COMPRESS
ARC

On prévoit d'étendre le dictionnaire sur 9 bits par exemple,

Etapes de l'émission :

« l » : on envoie son code « 108 »

« e » : on envoie son code « 101 » et on rajoute dans la table de code : **256 : le**

« s » : on envoie son code « 115 » et on rajoute dans la table de code : **257 : es**

« _ » : on envoie son code « 0 » et on rajoute dans la table de code : **258 : s_**

« p » : on envoie son code « 112 » et on rajoute dans la table de code : **259 : _p**

« o » : on envoie son code « 111 » et on rajoute dans la table de code : **260 : po**

« u » : on envoie son code « 117 » et on rajoute dans la table de code : **261 : ou**

« le » : on envoie son code « 256 » et on rajoute dans la table de code : **262 : ule**

« s_ » : on envoie son code « 258 » et on rajoute dans la table de code : **263 : les_**

« d » : on envoie son code « 100 » et on rajoute dans la table de code : **264 : s_d**

etc…

le décodeur reçoit et enrichit sa table au fur-et-à-mesure...

Mieux qu'Huffman (mais plus cher !)
Codage arithmétique

JBIG
standart
Joint
Bi-level
Image
Compression
par ex.
fax

Associer au message (ou à un flot de messages)
un nombre compris entre 0 et 1
en tenant compte de la statistique de la source.

Huffman :1 symbole = 1 mot-code

Arithmétique : 1 flot de symboles = nbre en virgule flottante
mais coûteux en calculs !

Algorithmes supplémentaires

Run-length coding (RLE) : si un message apparaît *n* fois consécutives,
Au lieu de coder x x x x…x *n* fois, on code : n x
Ou plutôt pour prévenir qu'il s'agit de RLE : @ n x

Flag de RLE

Chaîne de N caractères avec M répétitions de longueur L moyenne chacune
Chaque répétition est remplacée par 3 caractères
Taux de compression : N / (N – ML + 3M)

II- Codage sans perte

Codage arithmétique

Associer au message (ou à un flot de messages) un nombre compris entre 0 et 1 en tenant compte de la statistique de la source.

Codage : $x_{inf}(n) < x < x_{sup}(n)$ *avec* $x_{inf}(n) = x_{inf}(n-1) + p_{inf}\,\% \,(x_{sup}(n-1) - x_{inf}(n-1))$

et $x_{sup}(n) = x_{inf}(n-1) + p_{sup}\,\% \,(x_{sup}(n-1) - x_{inf}(n-1))$

$[p_{inf}\,\% \,, p_{sup}\,\%\,]$: intervalle associé à la lettre en fonction de sa statistique

JBIG
standart
Joint
Bi-level
Image
Compression
par ex.
fax

Dans l'exemple : ***les_poules***

	_	e	l	o	p	s	u
$p_{inf}\,\%$:	0	0.1	0.3	0.5	0.6	0.7	0.9
$p_{sup}\,\%$:	0.1	0.3	0.5	0.6	0.7	0.9	1

Codage :

« l » : 0.3 < x < 0.5
« e » : 0.32 < x < 0.36 (10% et 30% de l'intervalle [0.3,0.5])
« s » : 0.348 < x < 0.356 (70% et 90% de l'intervalle [0.32,0.36])
« _ » : 0.3480 < x < 0.3488
« p » : 0.34848 < x < 0.34856
« o » : 0.34852 < x < 0.348528
« u » : 0.3485272 < x < 0.348528
« l » : 0.34852744 < x < 0.3485276
« e » : 0.348527456 < x < 0.348527488
« s » : 0.3485274784 < x < 0.3485274848

3485274784 code le message (compris entre 2^31 et 2^32) : 32 bits sont nécessaires (code ASCII : 10*8=80 bits, entropie = 2.72 bits)

Décodage : $p_{inf}\,\% < x(n) < p_{sup}\,\%$: on décode la lettre correspondante

et $x(n+1) = (x(n) - p_{inf}\,\%) / (p_{sup}\,\% - p_{inf}\,\%)$

algorithmes supplémentaires

Run-length coding (RLE) : si un message apparaît *n* fois consécutives,
Au lieu de coder x x x x…x *n* fois, on code : n x
Ou plutôt pour prévenir qu'il s'agit de RLE : @ n x

Flag de RLE

Chaîne de N caractères avec M répétitions de longueur L moyenne chacune
Chaque répétition est remplacée par 3 caractères
Taux de compression : N / (N – ML + 3M)

Move to front coding : pour données non numériques
Exemple avec 4 caractères a,b,c,d à coder 0,1,2,3

a b a b c c b b c c c c b d b c c

« a » on code « 0 » avec table de codage : a->0, b->1, c->2, d->3
« b » : on code 1, table de codage se modifie : b->0, a->1, c->2, d->3
« a » : on code 1, table de codage se modifie : a->0, b->1, c->2, d->3
« b » : on code 1, table de codage se modifie : b->0, a->1, c->2, d->3
« c » : on code 2, table de codage se modifie : c->0, b->1, a->2, d->3
« c » : on code 0 etc…

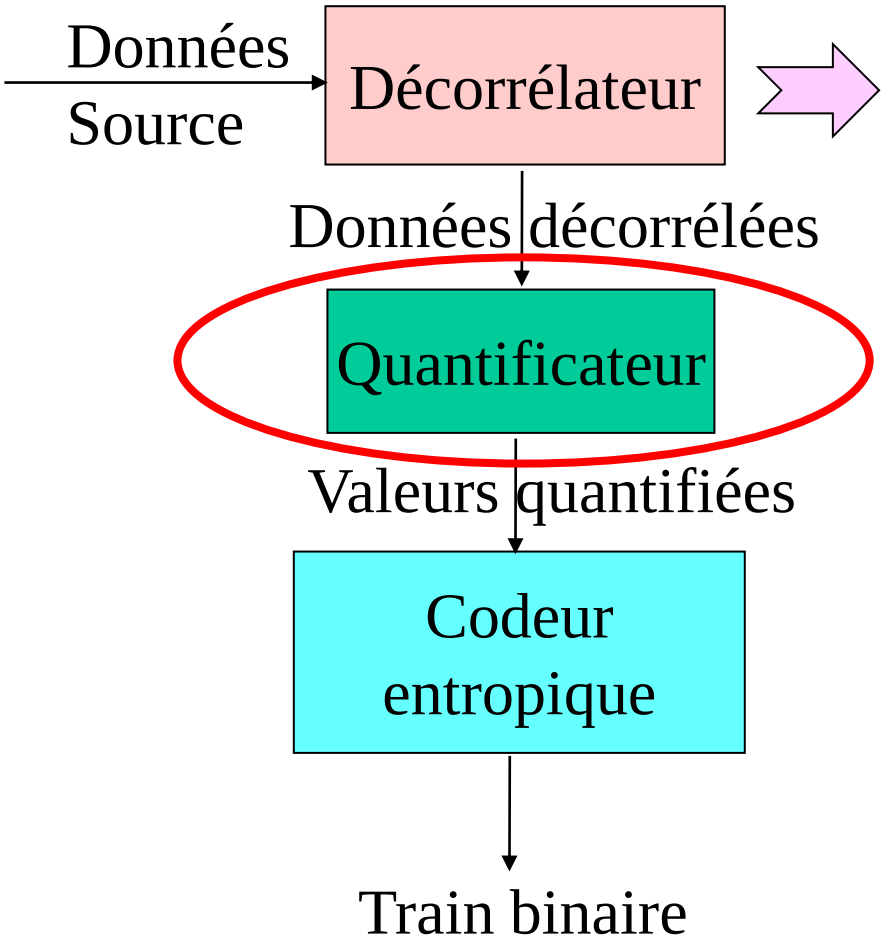
BZIP2 (Burrows-Wheeler)

→ Modifie les fréquences des symboles, faire ensuite code entropique

Nécessité du Codage avec perte

Chiffres :

- parole : 64 kbps
- musique stéréo : 1.5 millions de bits / s
- HDTV : 1 billion de bits /s !!!



méthodes prédictives,
méthodes par transformées,
quantification vectorielle
ET méthodes hybrides

III- Codage avec perte : quantification scalaire

Ce qu'il faut savoir…

Quantification uniforme : pas de quantification Δ constant,

si *N nombre de niveaux de quantification* $=2^n$ suffisamment grand
(*n* nombre de bits)

SNR $\approx 6n - 20\log(a) + 10.8$ dB avec $a = B/\sigma_x$

rapport signal à bruit = fonction linéaire du nombre de bits

+ 1 bit = + 6 dB

Ou vu autrement :

$$\sigma^2_\varepsilon = \sigma^2_x \, a^2 \, 2^{-2n}$$

La puissance de l'erreur de quantification est proportionnelle à la puissance du signal à l'entrée du quantificateur

Quantification non uniforme : en téléphonie, lois A et μ, Q. logarithmique

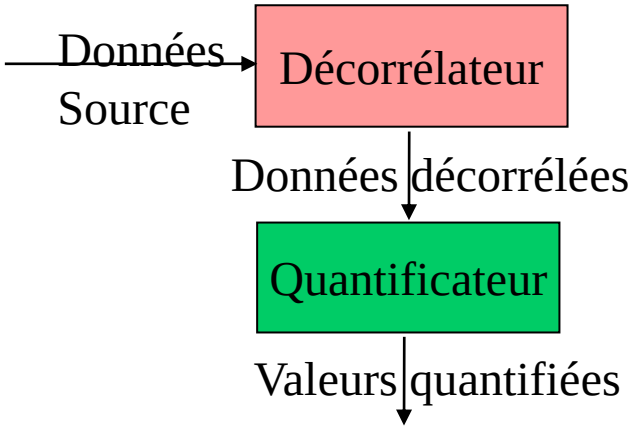
IV- Codage avec perte : méthodes prédictives

Idée : Coder les différences entre échantillons plutôt que les échantillons

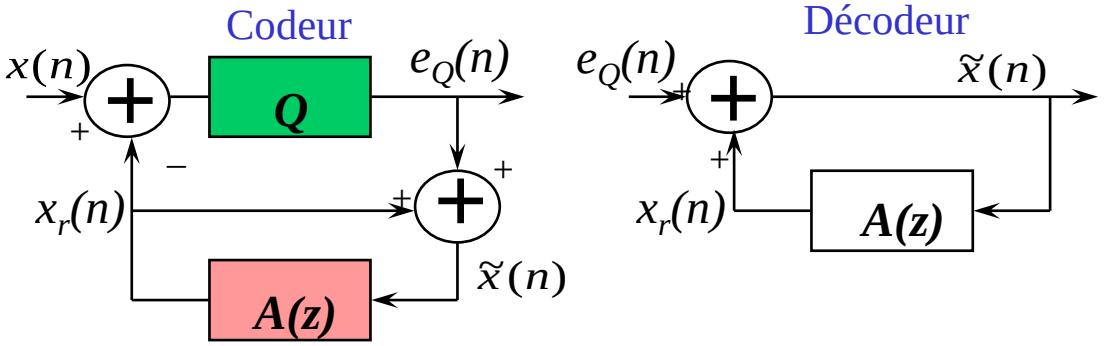
mieux : coder une différence pondérée dont la puissance est plus faible que celle du signal !

$e(n)=x(n)- \sum_k a_k\, x(n-k)$ $\quad a_k$ minimisent la puissance de $e(n)$

On code $e(n)$, l'Erreur de Prédiction Linéaire (EPL), l'erreur de modèle, la partie non prédictible



Differential Pulse Code Modulation (DPCM)



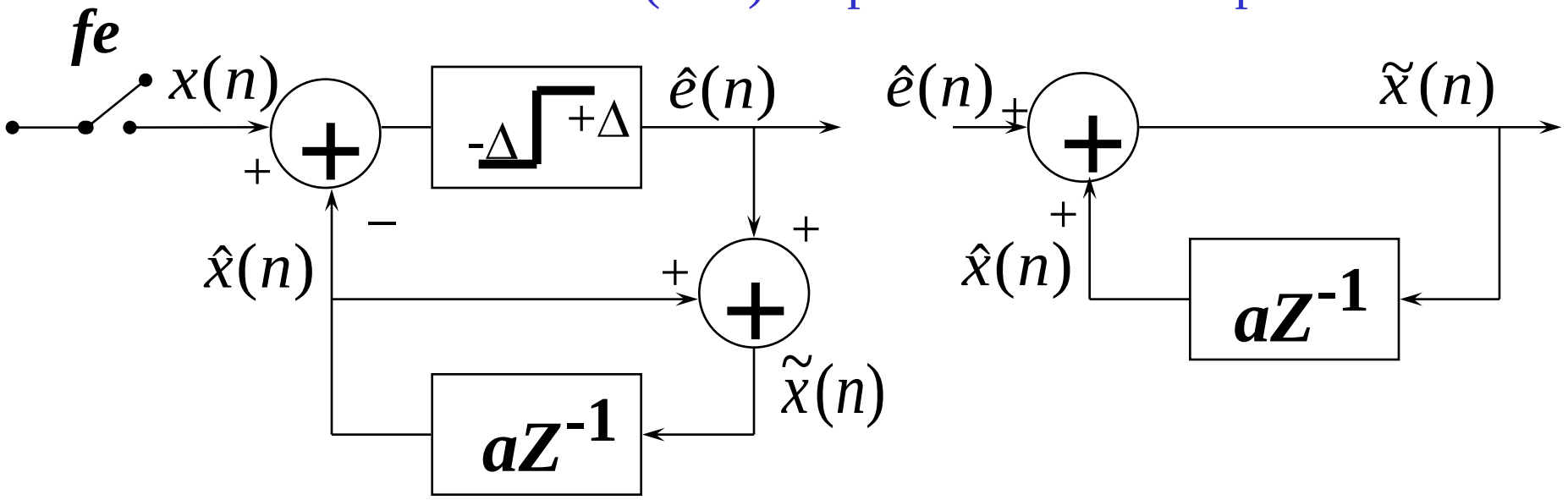
Gain constant par rapport au PCM : $SNR_{DPCM} = SNR_{QS} + 10 \log G_p$ avec $G_p = \sigma^2_x / \sigma^2_e$

Versions adaptatives : ADM, ADPCM

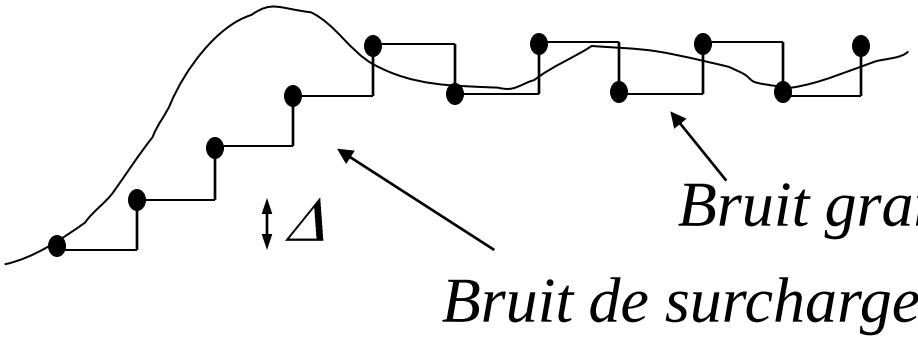
IV- Codage avec perte : méthodes prédictives

Version économique :

Modulation Delta (DM) ou plutôt version adaptative ADM

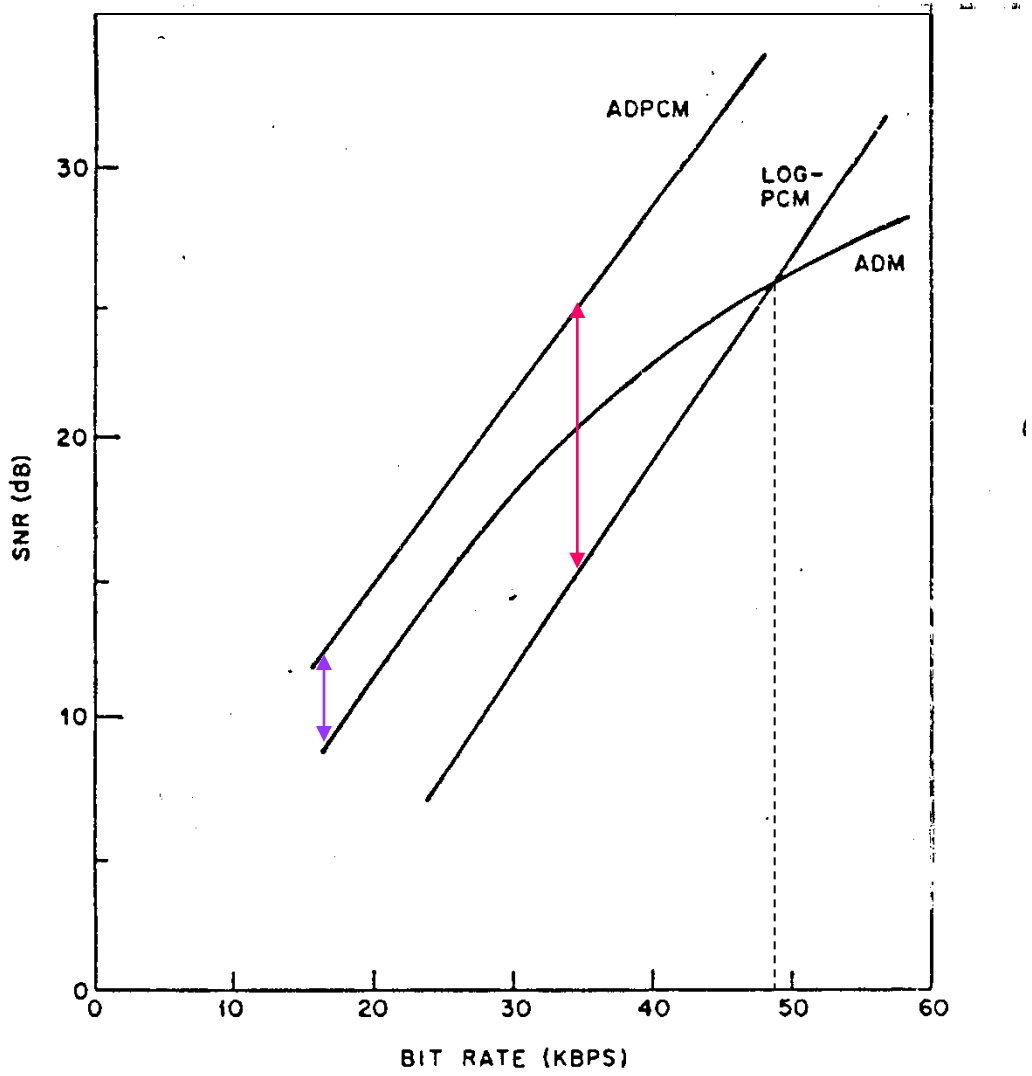


$\tilde{x}(n) = a\tilde{x}(n-1) + \Delta\, sign(x(n)-a\tilde{x}(n-1))$ *avec* $a=r_1$ *proche de 1*



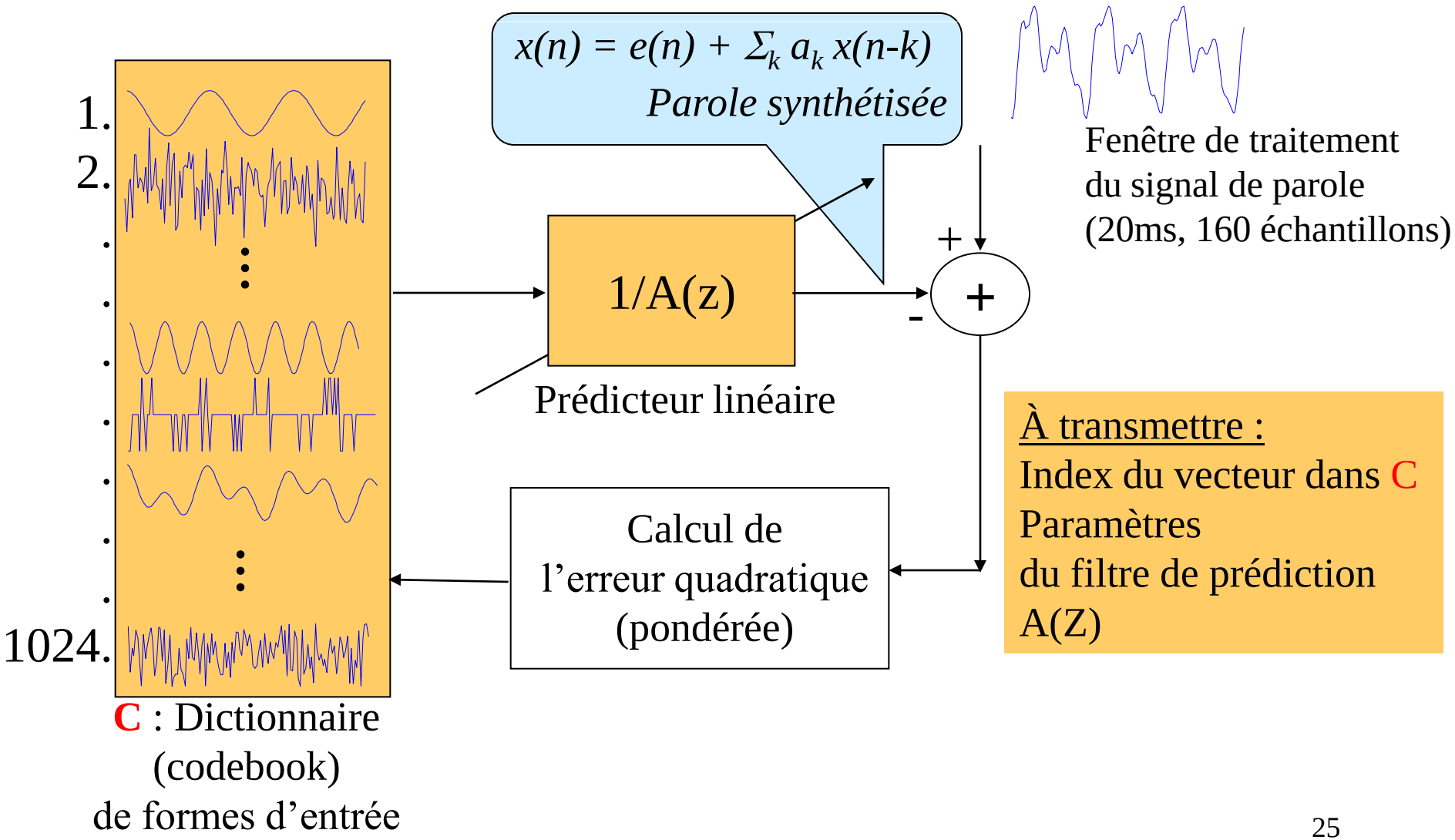
Pas de Quantif. à rendre **adaptatif**

IV- Codage avec perte : méthodes prédictives



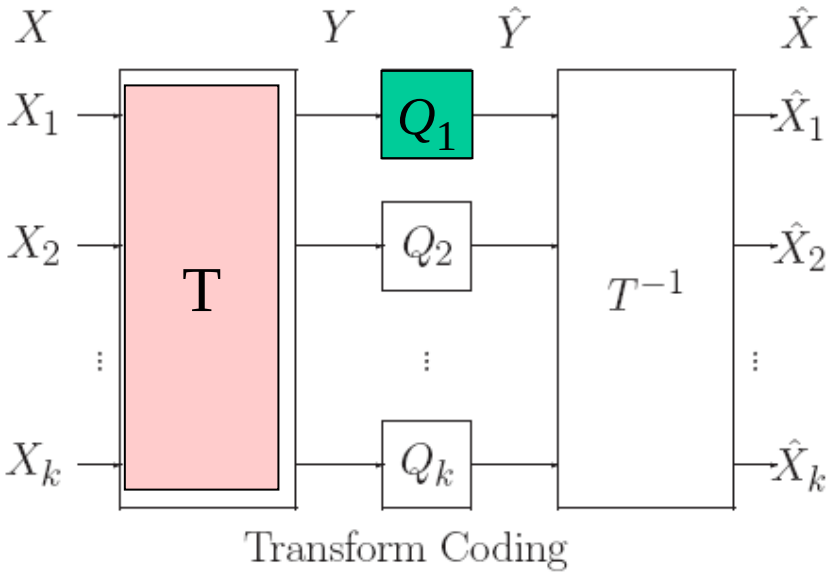
Comparaison des performances Log-PCM, ADM et ADPCM en termes de taux de bits et de SNR

Application au codage de la parole : les codeurs CELP
Code Excited Linear Predictor



V- Codage avec perte : méthodes par transformées

Codage par Transformée : le principe de base



Transform Coding

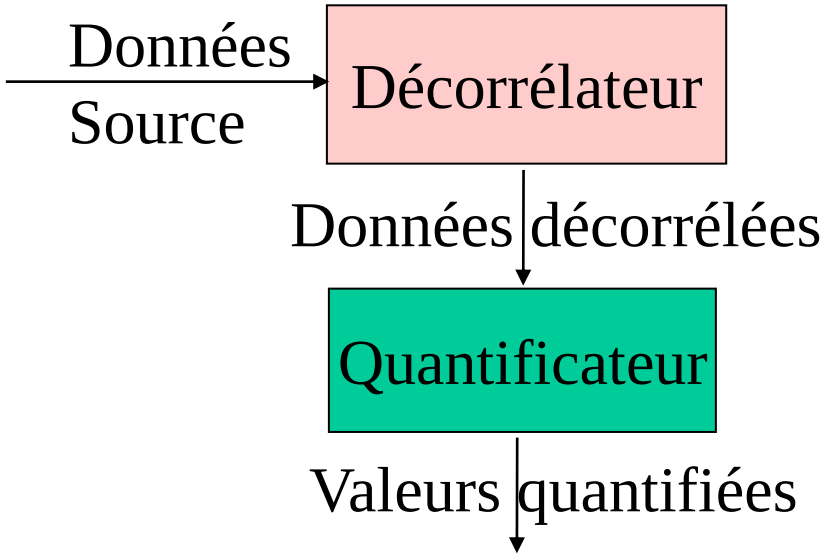
$$X = (X_1, X_2, \ldots, X_k)^t$$

$$Y = T(X), \hat{Y} = Q(Y), \hat{X} = T^{-1}(\hat{Y})$$

Transform Coding

Mathews and Kramer (1956)
Huang, Habibi, Chen (1960s)

Dominant image coding (lossy compression) method: ITU and other standards p*64, H.261, H.263, JPEG, MPEG
C-Cubed, PictureTel, CLI



V- Codage avec perte : méthodes par transformées

Quelle transformée ?
Unitaire : erreur entre Y et $\hat{Y}$ de même puissance qu'entre X et $\hat{X}$

Quelle transformée unitaire ?

- qui décorrèle
- qui concentre le maximum d'énergie sur le minimum de coefficients transformés

Se mesure par le gain de transformée

$$G_{TC} = \frac{\frac{1}{k}\sum_{n=1}^{k}\sigma_{Y_n}^2}{\left(\prod_{n=1}^{k}\sigma_{Y_n}^2\right)^{1/k}}$$

Quelle transformée unitaire ?

Solution optimale = Transformée de Karhunen-Loeve (KL)
Mais matrice T dépend de la corrélation du signal…pas commode !

Deux transformées dominent pour leurs performances :
1. La DCT (Discrete Cosine Transform)
2. La DWT (Discrete Wavelet Transform)

Intérêt de la DCT : simple à implanter,
Schéma sans quantification possible (tout ou rien)
Base du standard JPEG 92-93

Karhunen-Loeve Transform

Often referred to as "optimal transform"

Idea: Decorrelate components of vector X. If also Gaussian, makes independent.

Consider 1D case, assume real:

$R_X = E[XX^t]$
$X \rightarrow Y = TX$ with $R_Y = E[YY^t]$ diagonal.

u_i denote the eigenvectors of R_X (normalized to unit norm)
λ_i the corresponding eigenvalues (ordered)

The Karhunen-Loeve transform matrix is then defined as $T = U^t$ where

$$U = [u_i u_2 \dots u_k],$$

(the columns of U are the eigenvectors of R_X)

Then

$$R_Y = E[YY^t] = E[U^t XX^t U] = U^t R_X U = diag(\lambda_i).$$

Note that the variances of the transform coefficients are the eigenvalues of the autocorrelation matrix R_X.

V- Codage avec perte : méthodes par transformées

Quelle transformée unitaire choisir ?

Celle qui décorrèle et qui concentre le mieux l'énergie

TABLE 5.3 Summary of Image Transforms

DFT/unitary DFT	Fast transform, most useful in digital signal processing, convolution, digital filtering, analysis of circulant and Toeplitz systems. Requires complex arithmetic. Has very good energy compaction for images.
Cosine	Fast transform, requires real operations, near optimal substitute for the KL transform of highly correlated images. Useful in designing transform coders and Wiener filters for images. Has excellent energy compaction for images.
Sine	About twice as fast as the fast cosine transform, symmetric, requires real operations; yields fast KL transform algorithm which yields recursive block processing algorithms, for coding, filtering, and so on; useful in estimating performance bounds of many image processing problems. Energy compaction for images is very good.
Hadamard	Faster than sinusoidal transforms, since no multiplications are required; useful in digital hardware implementations of image processing algorithms. Easy to simulate but difficult to analyze. Applications in image data compression, filtering, and design of codes. Has good energy compaction for images.
Haar	Very fast transform. Useful in feature extracton, image coding, and image analysis problems. Energy compaction is fair.
Slant	Fast transform. Has "image-like basis"; useful in image coding. Has very good energy compaction for images.
Karhunen-Loeve	Is optimal in many ways; has no fast algorithm; useful in performance evaluation and for finding performance bounds. Useful for small size vectors e.g., color multispectral or other feature vectors. Has the best energy compaction in the mean square sense over an ensemble.
Fast KL	Useful for designing fast, recursive-block processing techniques, including adaptive techniques. Its performance is better than independent block-by-block processing techniques.
Sinusoidal transforms	Many members have fast implementation, useful in finding practical substitutes for the KL transform, analysis of Toeplitz systems, mathematical modeling of signals. Energy compaction for the optimum-fast transform is excellent.
SVD transform	Best energy-packing efficiency for any given image. Varies drastically from image to image; has no fast algorithm or a reasonable fast transform substitute; useful in design of separable FIR filters, finding least squares and minimum norm solutions of linear equations, finding rank of large matrices, and so on. Potential image processing applications are in image restoration, power spectrum estimation and data compression.

Quelle transformée ?

Many other transforms, but dominant ones are discrete cosine transform (DCT, the 800 lb gorilla) and discrete wavelet transform (DWT).

Theorems to show that if image $\approx$ Markovian, then DCT approximates KL

Wavelet fans argue that wavelet transforms $\approx$ decorrelate wider class of images.

Often claimed KL "optimal" for Xform codes

Only true (approximately) if high rate, optimal bit allocation among coefficients, and Gaussian.

V- Codage avec perte : méthodes par transformées

Codage par transformée :

Implantation simple
en ne transmettant que
M coefs sur les N coefs
transformés

TC = N/M

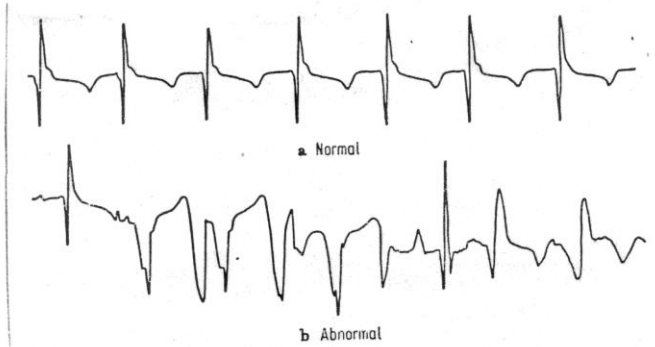


Fig. 9.7 Original ECG's using all the 128 data points

DCT :
Discrete Cosine Transform

The 1-D discrete cosine transform is defined by

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

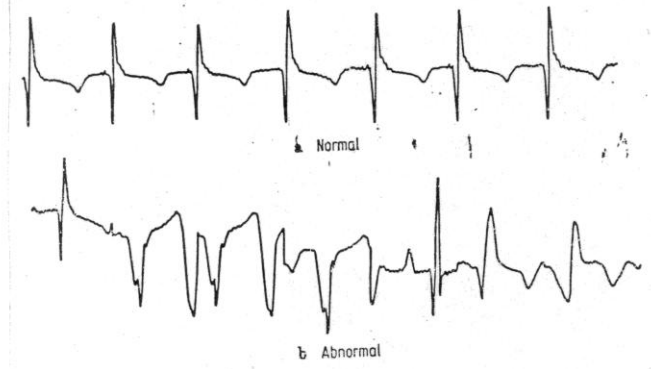


Fig. 9.8 Reconstructed ECG's using the dominant 43 DCT components

Hadamard transform :
Matrice de +1 et -1

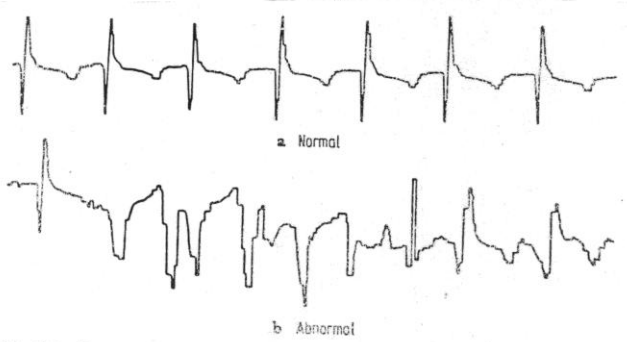


Fig. 9.9 Reconstructed ECG's using the dominant 43 HT components

V- Codage avec perte : méthodes par transformées

Variance des composantes du vecteur transformé (dimension 16)

(signal théorique processus de Markov) pour différentes transformées

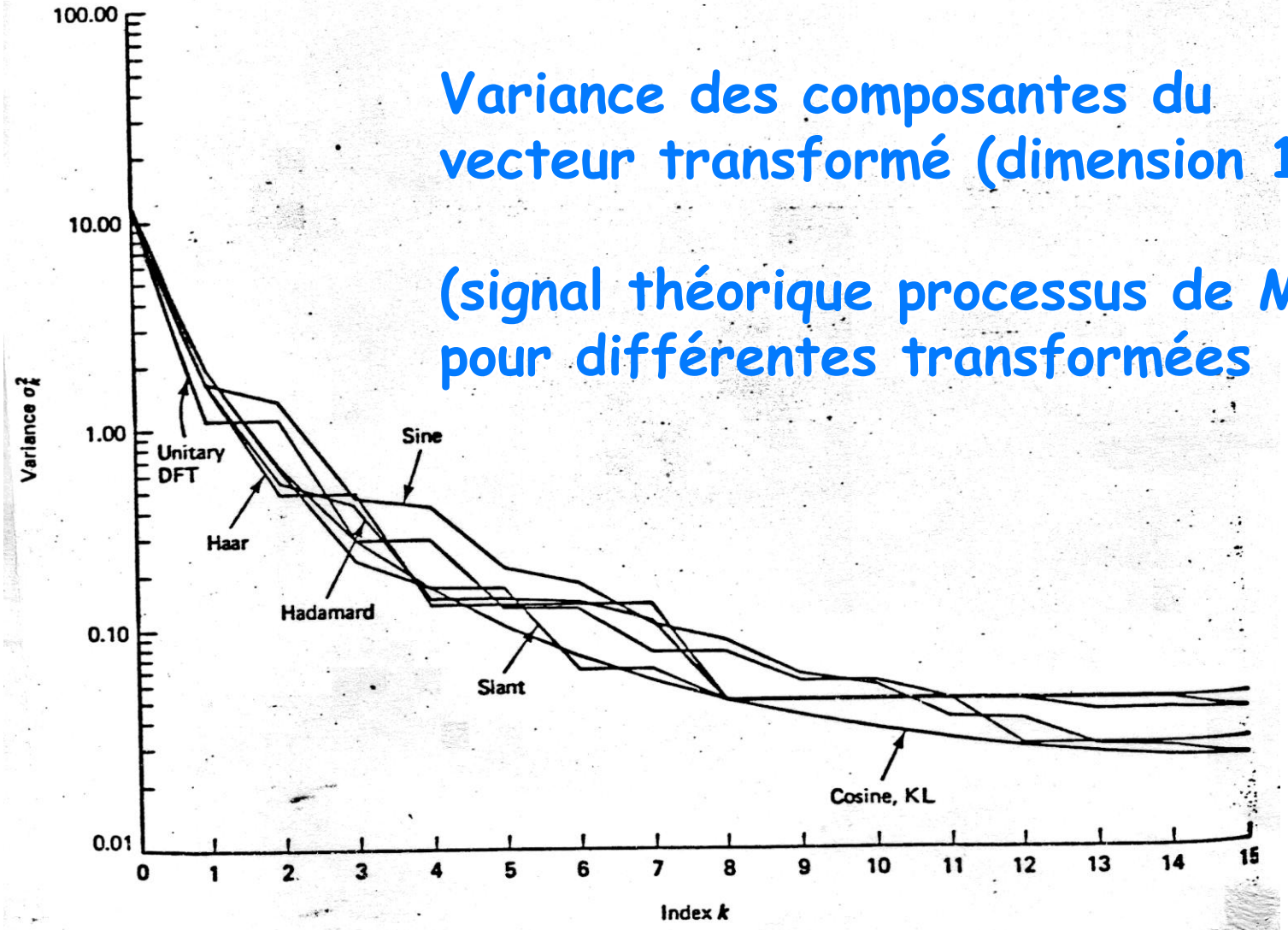


Figure 5.18 Distribution of variances of the transform coefficients (in decreasing order) of a stationary Markov sequence with $N = 16$, $\rho = 0.95$ (see Example 5.9).

V- Codage avec perte : méthodes par transformées

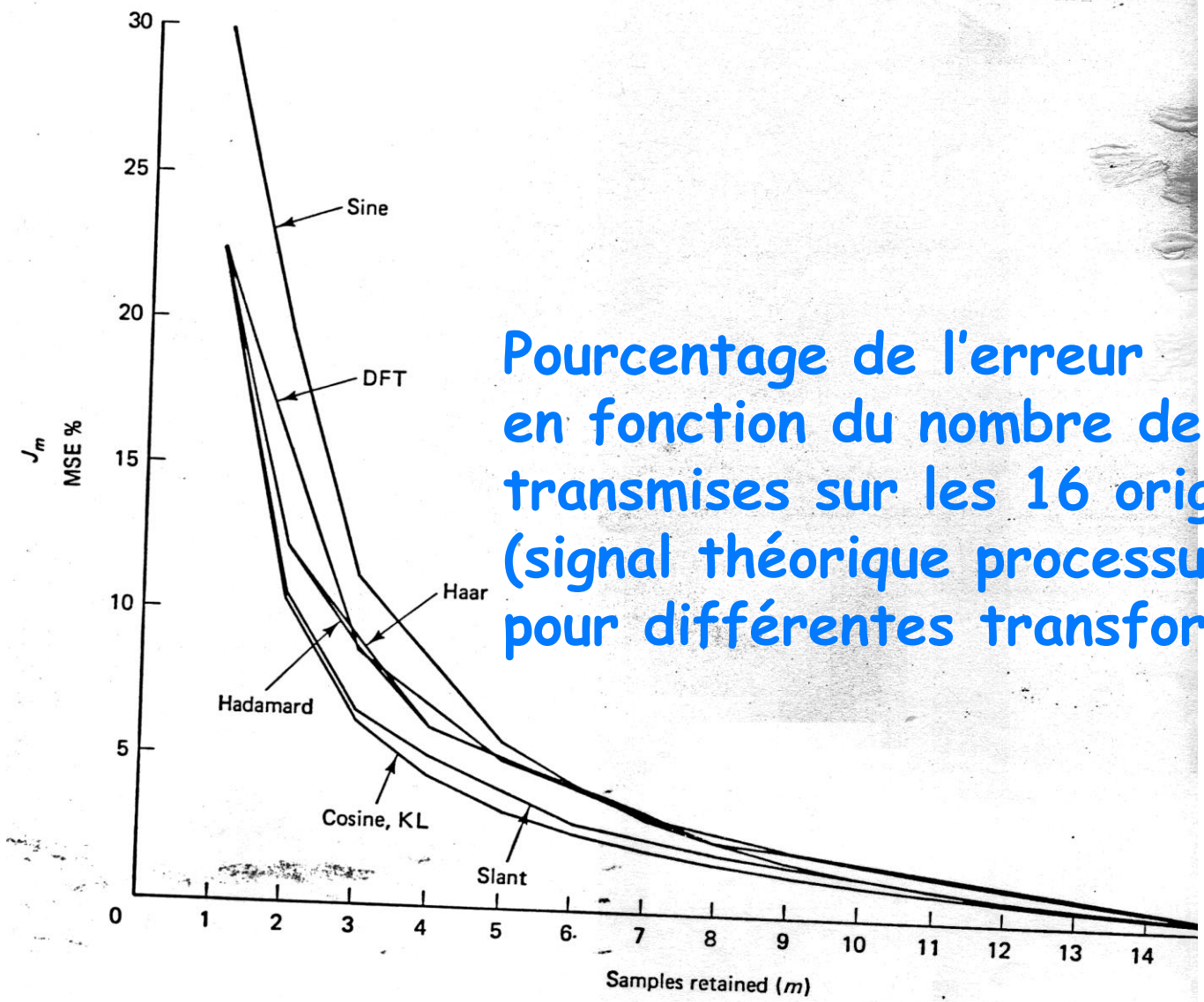
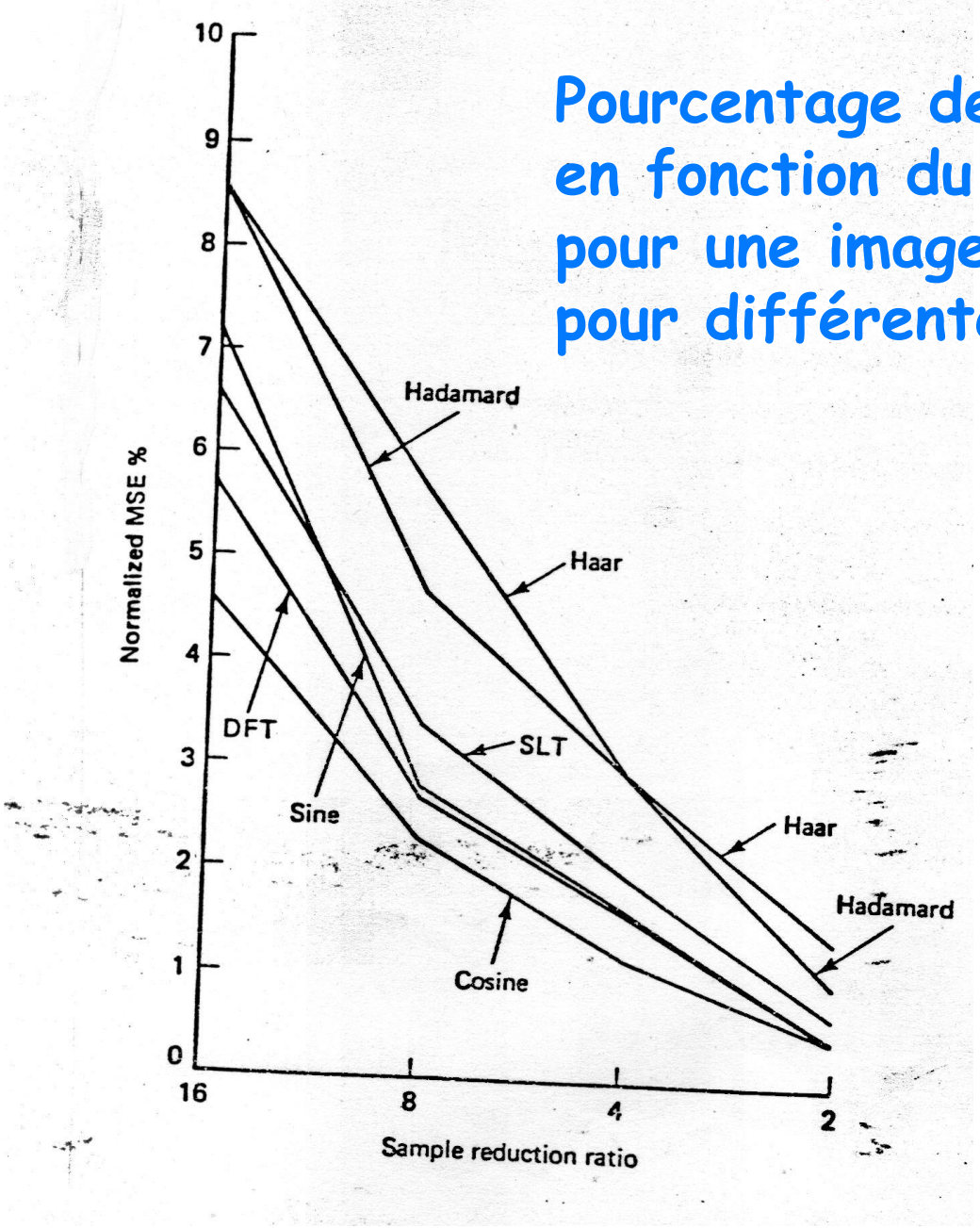


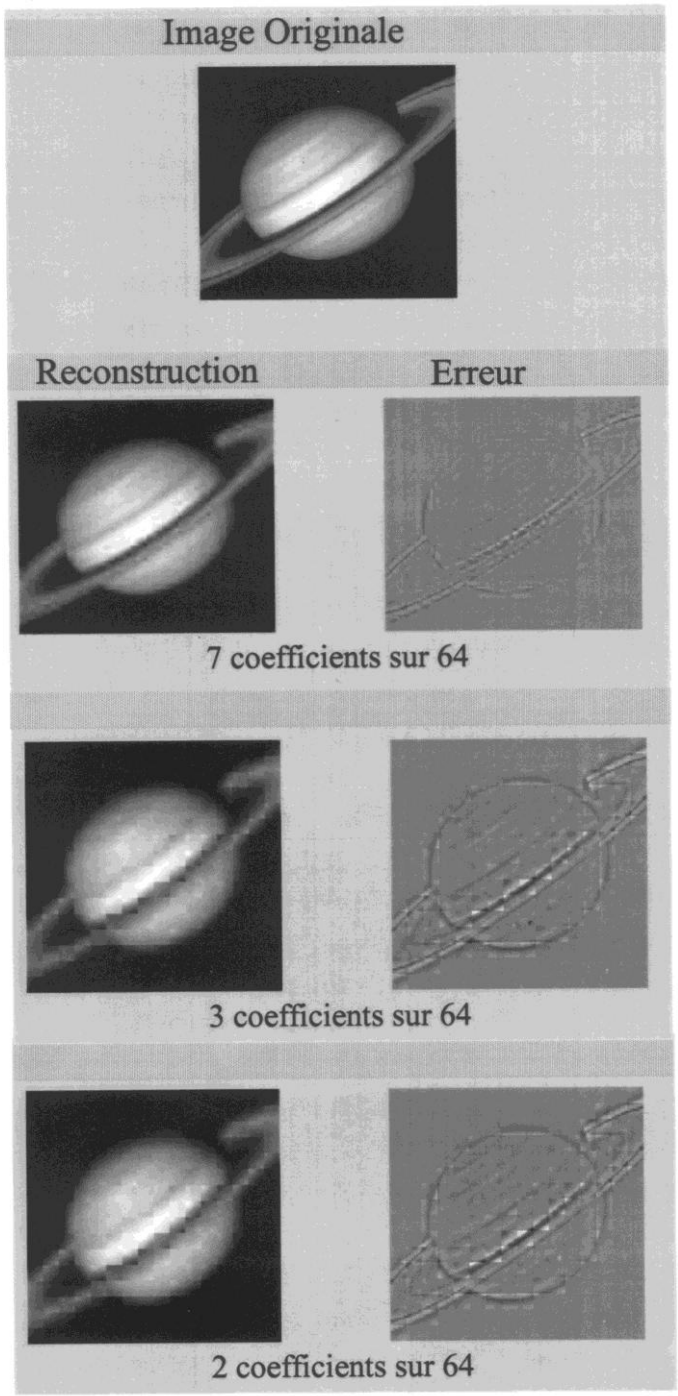
Figure 5.19 Performance of different unitary transforms with respect to basis restriction errors (J_m) versus the number of basis (m) for a stationary Markov sequence with $N = 16$, $\rho = 0.95$.

Pourcentage de l'erreur en fonction du nombre de composantes transmises sur les 16 originales (signal théorique processus de Markov) pour différentes transformées

V- Codage avec perte : méthodes par transformées



Pourcentage de l'erreur
en fonction du taux de compression
pour une image 256*256
pour différentes transformées



La transmission progressive d'images
est possible via la DCT

Why DCT so important?

- real
- can be computed by an FFT
- excellent energy compaction for highly correlated data
- good approximation to Karhunen-Loeve transform for Markov behavior and large images

Won in the open competition for an international standard.

- 1985: Joint Photographic Experts Group (JPEG) formed as an ISO/CCITT joint working group
- 6/87 12 proposals reduced to 3 in blind subjective assessment of picture quality
- 1/88 DCT-based algorithm wins
- 92-93 JPEG becomes intl standard

V- Codage avec perte : méthodes par transformées

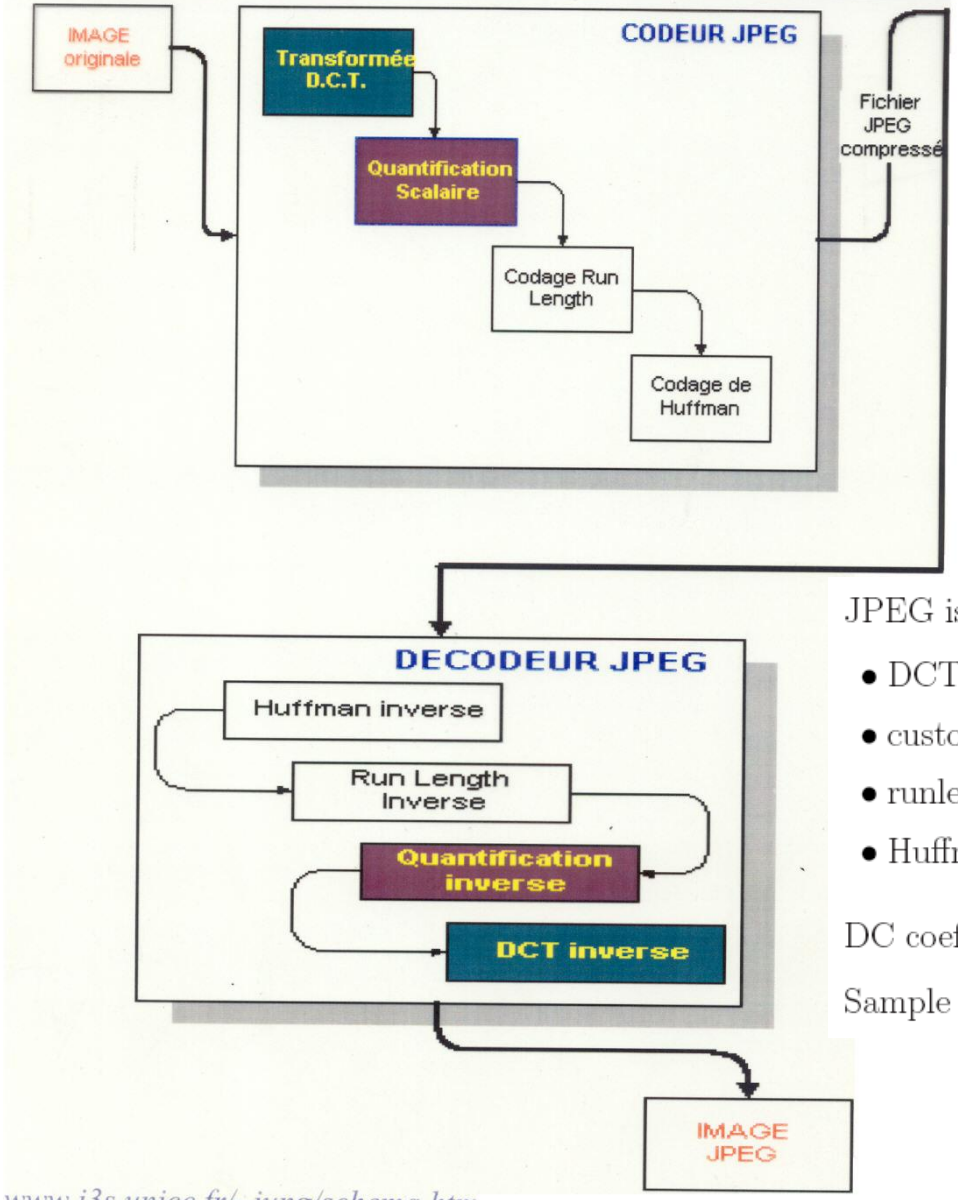
Once have transform, how actually quantize & code?

Several options:

- Scalar quantize coefficients
  - ⋆ Uniform simple and pretty good if high rate and use subsequent entropy coding. Only need to pick bin size for each quantizer.
  - Lloyd-optimal good for fixed rate applications and very low bit rate ECQ.
- Quantize groups (subbands) as vectors (complicated)
- Entropy code
  - Many zeros after quantizer, runlength code
  - Huffman or arithmetic
  - Combination

Algo d'allocation (sous-)optimale de bits (cf annexe 1)

DCT - Compression JPEG



JPEG is basically

- DCT transform coding (DCT) on 8×8 pixel blocks
- custom uniform quantizers (user-specified Quantization tables)
- runlength coding
- Huffman or arithmetic coding on (runs, levels)

DC coefficient coded separately, DPCM

Sample quantization table

www.i3s.unice.fr/~jung/schema.htm

V- Codage avec perte : méthodes par transformées

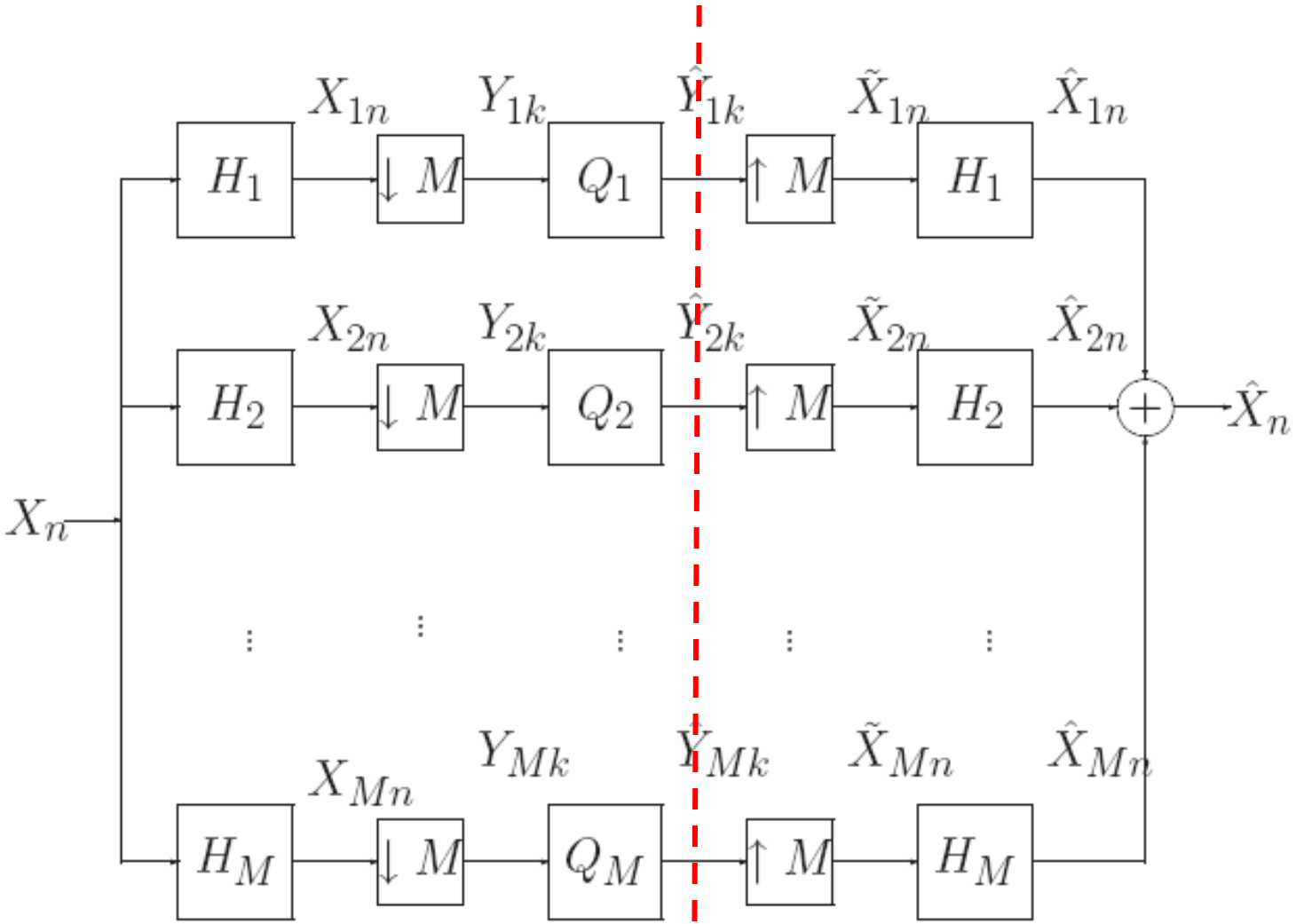
Subband/pyramid/wavelet coding (late 1980s)
(Lots of people, e.g., Barnwell, Smith, Adelson, Burt, Mallat, Vetterli)
Filter and downsample.
Traditional uniform subbands: MUSICAM digital audio (European standard)

Use filter bank of roughly orthogonal equal bandwidth filters.
(Classical subband decomposition.)
Generalize to nonorthogonal with *perfect reconstruction property*

Two dimensional subband decomposition: Separately decompose rows and columns.

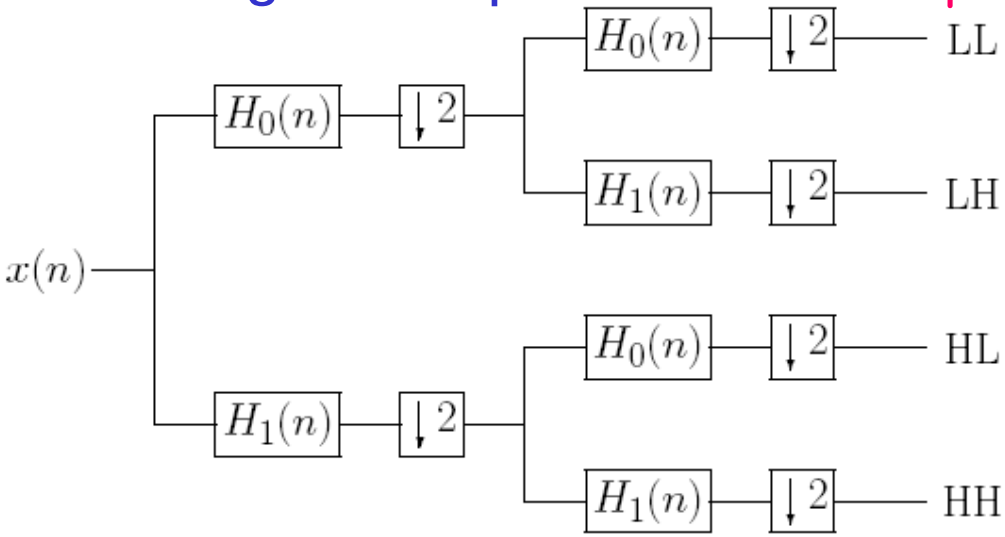
Wavelet, pyramid, or octave subbands: Only split low frequency band each time.

V- Codage avec perte : méthodes par transformées

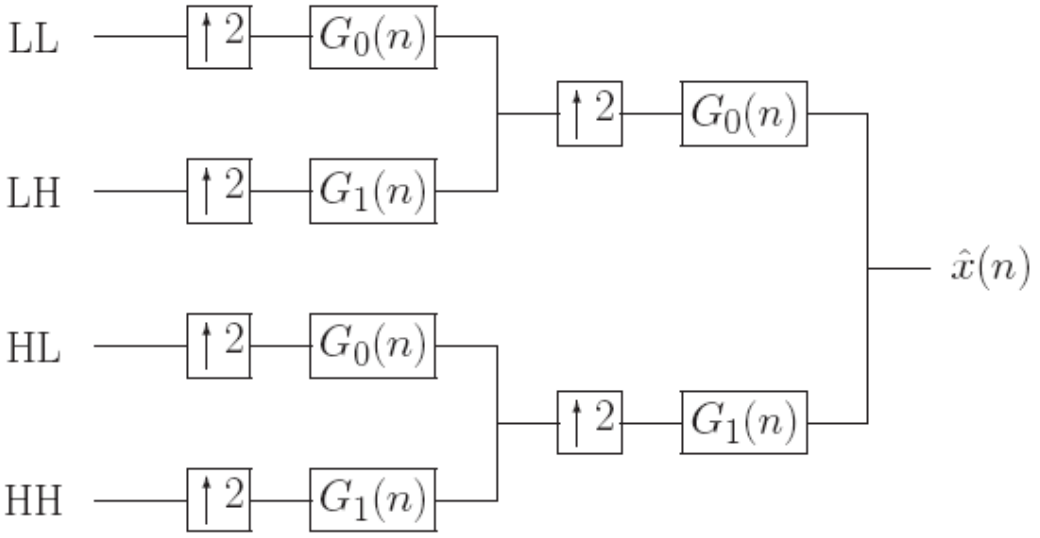


Décimateurs, interpolateurs, voir annexe 2

V- Codage avec perte : méthodes par transformées

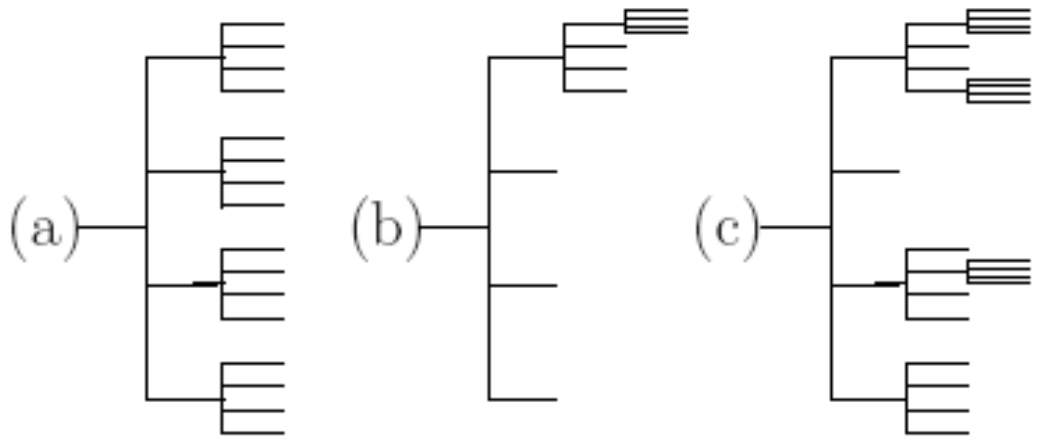


One stage of subband decomposition with 2-D separable filters



One stage of subband reconstruction with 2-D separable filters

V- Codage avec perte : méthodes par transformées



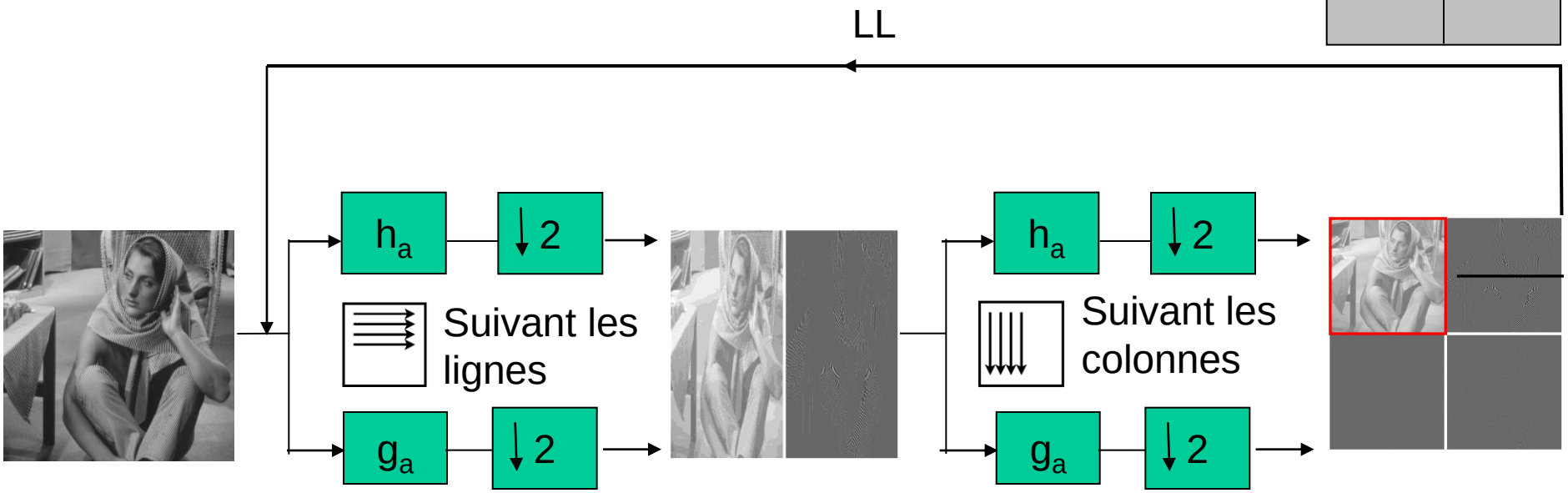
(a) Two stages of uniform decomposition leading to 16 uniform subbands, (b) Three stages of pyramidal decomposition, leading to 10 octave-band subbands, (c) a wavelet packet

For each subband in an image (except the very lowest one, and the three highest ones), each coefficient A is related to exactly 4 coefficients in the next higher band.

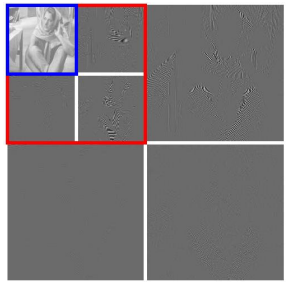
Codage par Ondelettes

LL	HL
LH	HH

Très utilisé en compression d'images.



Filtres passe-bas
et passe-haut :
Daubechies 9/7 par ex



wavedemo, command on line, wavelet2D
wavemenu, wavelet2D et compress

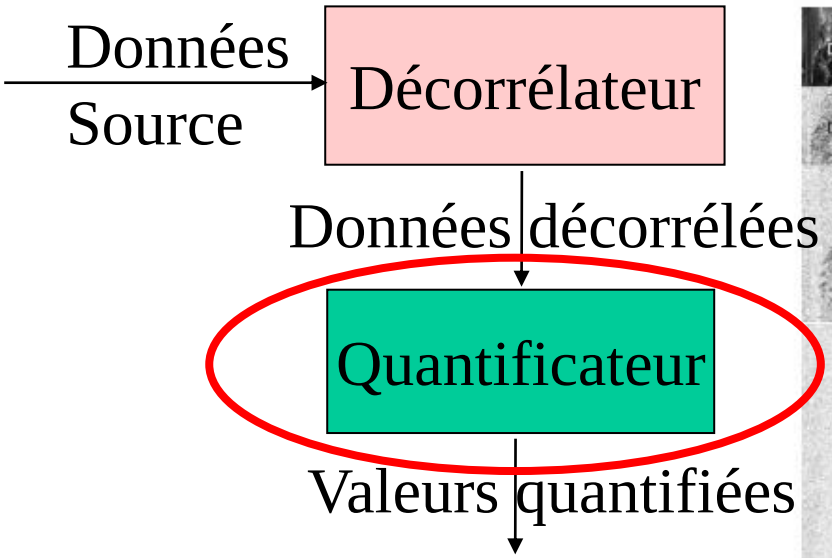
V- Codage avec perte : méthodes par transformées

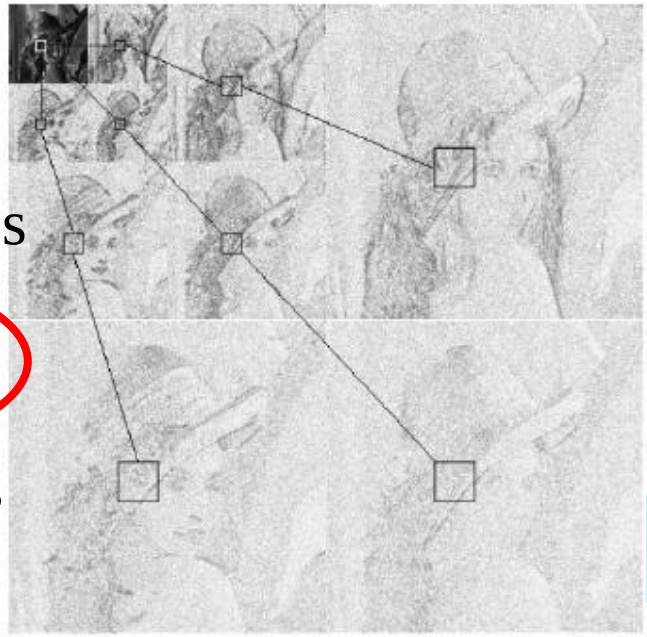
Question is then: How code?

I.e., how quantize wavelet coefficients and then lossy code them?

Early attempts used traditional (JPEG-like & VQ) techniques.
Not impressive performance.

Breakthrough came with Knowles and Shapiro (1993):
Embedded zerotree coding.





Parent-descendants

Détails
EZW
et
SPIHT
Voir annexe 3

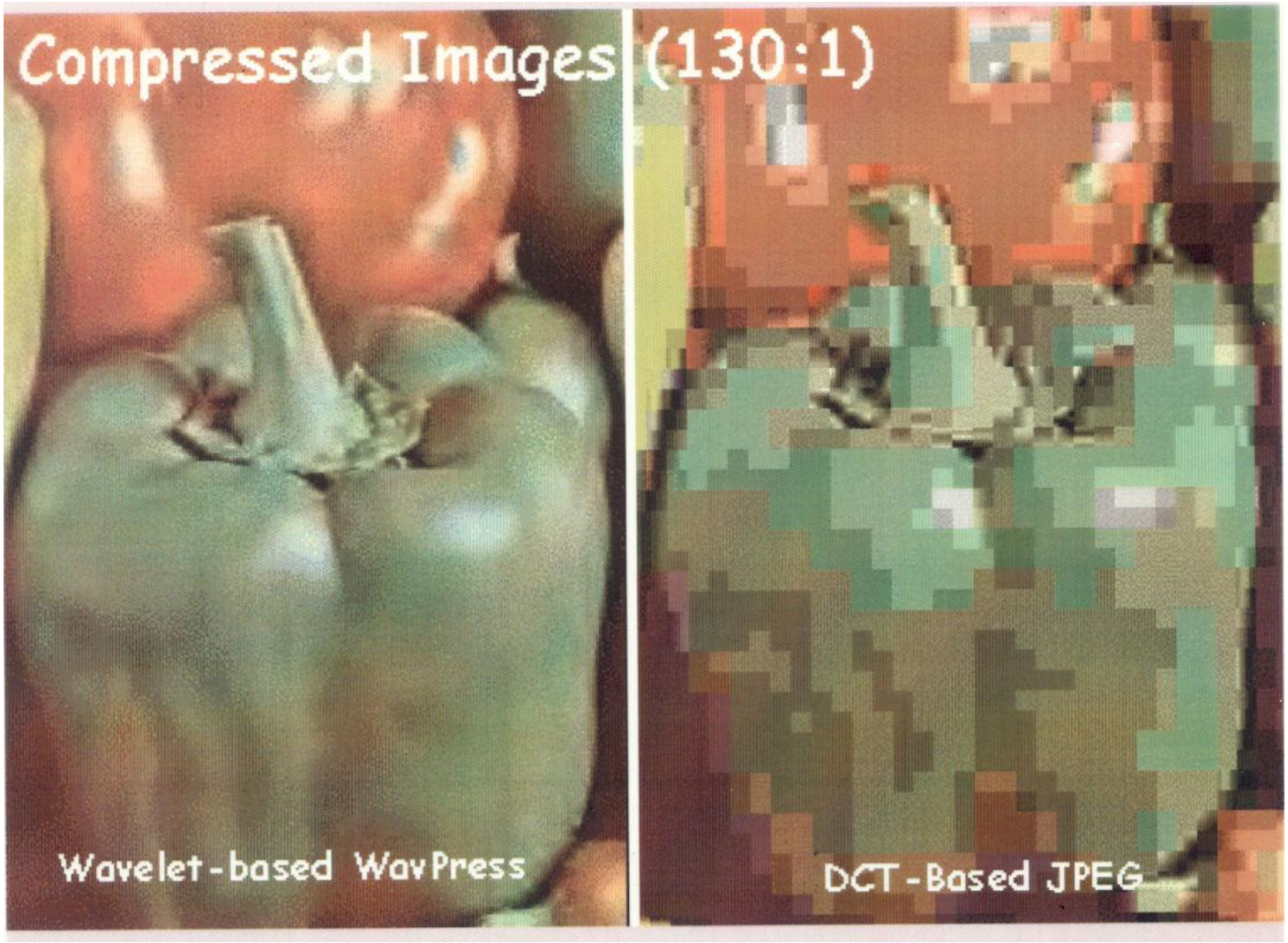
(a) Original LENA

(b) Rate = 0.5 bpp, PSNR = 37.2 dB

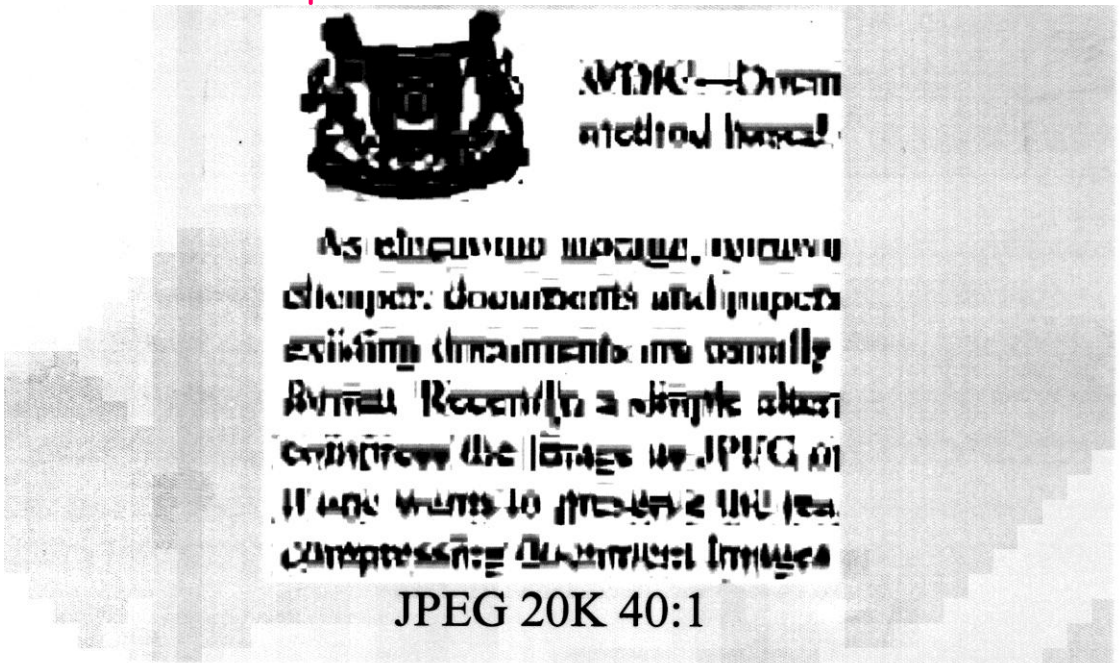
(c) Rate = 0.25 bpp, PSNR = 34.1 dB

(d) Rate = 0.15 bpp, PSNR = 31.9

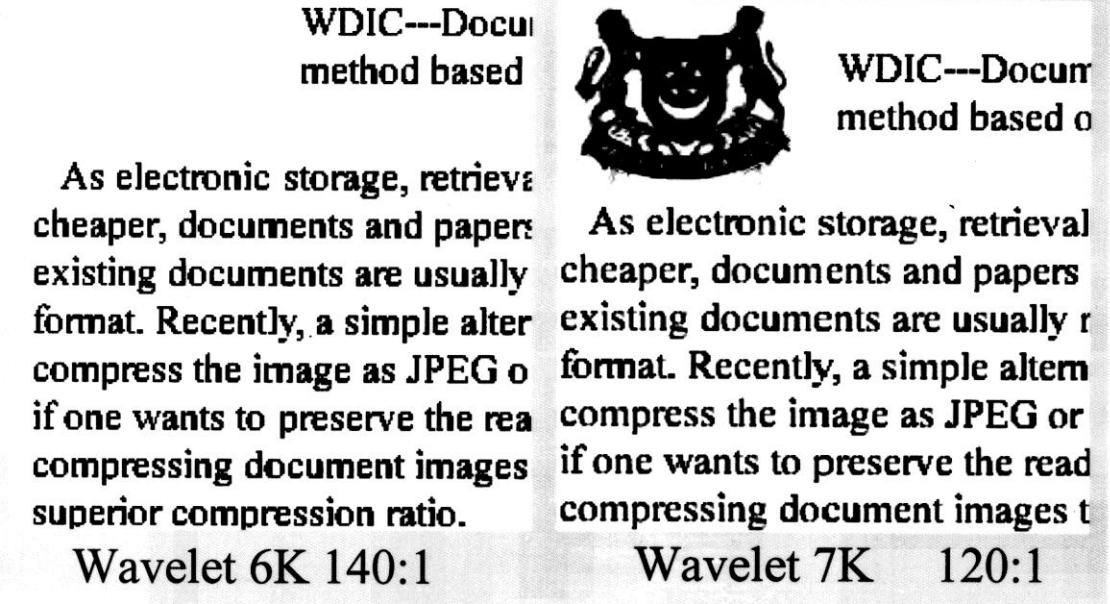
SPHIT Examples


Compressed Images (130:1)
Wavelet-based WavPress
DCT-Based JPEG

V- Codage avec perte : méthodes par transformées



JPEG 20K 40:1



Wavelet 6K 140:1

Wavelet 7K 120:1

VI- Codage avec perte : quantification vectorielle

QV de dimension k et de taille N :
application de IR^k vers le « codebook » $C=\{\underline{y}_1,\underline{y}_2,\ldots,\underline{y}_N\}$ (dictionnaire)

$\underline{x} \mapsto \hat{\underline{x}} = \underline{y}_i$ *on ne transmet que le numéro de l'indice*

$\log_2(N)$ bits par vecteur d 'entrée soit $\log_2(N)/k$ bits par échantillon

Mesure de la distorsion produite : $d(\underline{x}, \hat{\underline{x}})$ distances euclidienne, pondérée, Mahalanobis…

Classe importante : QV du ppv : évite la description des cellules

QV optimal : CN : conditions du ppv et du centroïde
Algorithmes nombreux
LBG (Linde Buzo Gray) : algorithme itératif sur les 2 CN
Pb de l'initialisation : aléatoire, par *Pairwise Nearest Neighbor*, par *Splitting*…

ANNEXES

ANNEXE 1

V- Codage avec perte : méthodes par transformées

Problème de l'allocation optimale de bits

codage des N composantes de y sur Nb bits au total.

Pb : trouver $b_0, b_1, \ldots, b_{N-1}$ tels que ***Min*** $\sigma^2_Q = 1/N \Sigma \sigma^2_Q(k)$ avec $\Sigma b_k \leq Nb$

or $\sigma^2_Q(k) = \Delta^2_k /12$ avec la dynamique du Quantificateur telle que $D_k = \Delta_k . 2^{bk} = \alpha_k \sigma_k$
d 'où

$$\sigma^2_Q(k) = c(k) . \sigma^2_k 2^{-2bk}$$

En choisissant les c(k) de manière identique = c, on cherche à

$$\textit{Min} \; \Sigma \sigma^2_k 2^{-2bk} \text{ sous la contrainte } \Sigma b_k \leq Nb$$

Pour **N** réels positifs, la **moyenne arithmétique** est supérieure (*ou égale si les N nombres sont égaux*) à la **moyenne géométrique**.

Donc,

$$1/N \Sigma \sigma^2_k 2^{-2bk} \leq (\Pi \sigma^2_k 2^{-2bk})^{1/N}$$

d'où

$$1/N \Sigma \sigma^2_k 2^{-2bk} \leq g^2 2^{-2b} \text{ avec } g^2 \text{ moyenne géométrique des variances } (\Pi \sigma^2_k)^{1/N}$$

L'égalité est obtenue lorsque tous les termes de la somme sont égaux :

Pour tout k, $\sigma^2_k 2^{-2bk} = g^2 2^{-2b}$: le QS optimal de chaque composante entraîne la même EQM.
Soit

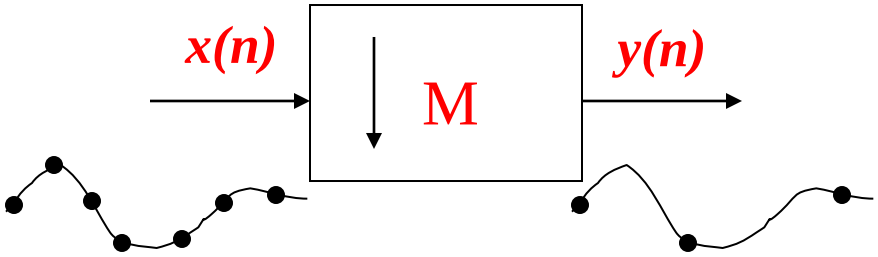
$$\mathbf{b_k = b + 1/2 \log_2 (\sigma^2_k / g^2)}$$

pas forcément un nombre entier...

ANNEXE 2

V- Codage avec perte : méthodes par transformées

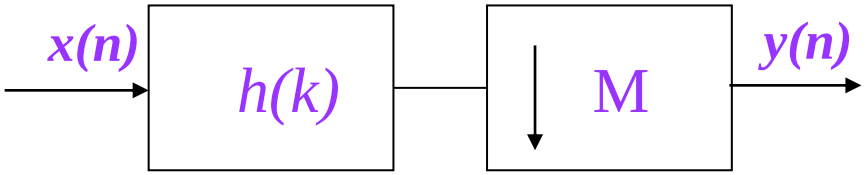
Décimateur - Interpolateur



$y(n)=x(nM)$

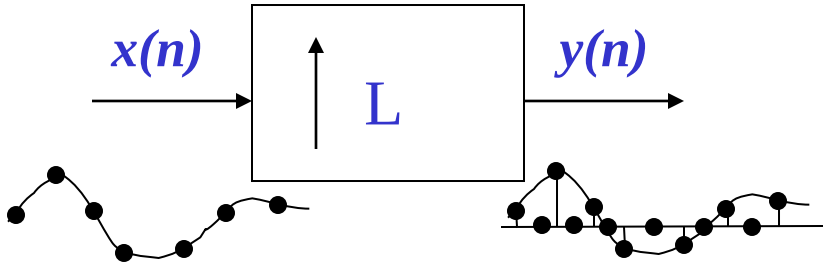
Exemple M=3

$$Y(z)=1/M \sum_{k=0}^{M-1} X(\exp(-i2\pi k/M)\, z^{1/M})$$



Filtre anti-repliement
H(f)=1 pour |f| < Fe1/(2M)
H(f)=0 ailleurs

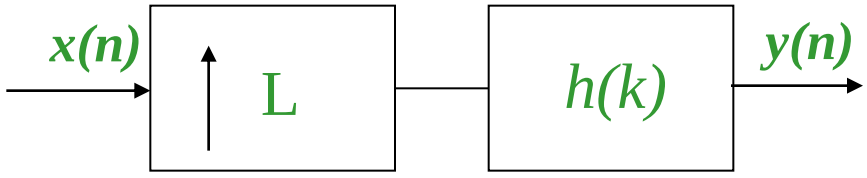
$$y(n) = \sum x(k)\, h(nM-k)$$



Exemple L=2

$y(n)= x(n / L)$ *si* $n=kL$
$= 0$ *sinon*

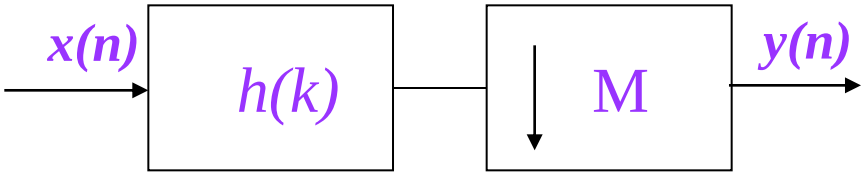
$$Y(z)=X(z^{L})$$

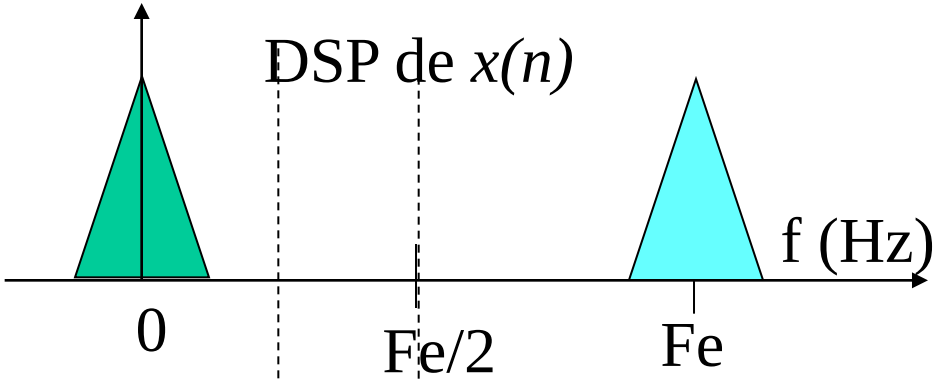


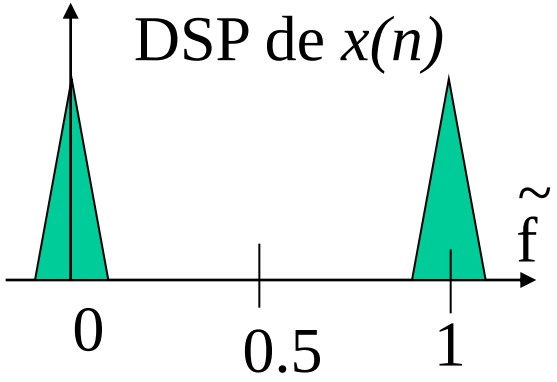
Filtre anti-image
H(f)=L pour |f| < Fe1/(2L)
H(f)=0 ailleurs

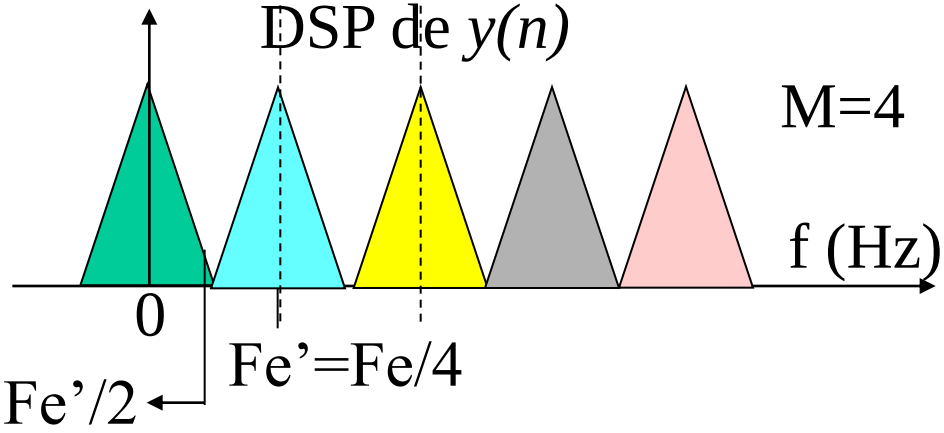
$$y(n) = \sum x(k)\, h(n-kL)$$

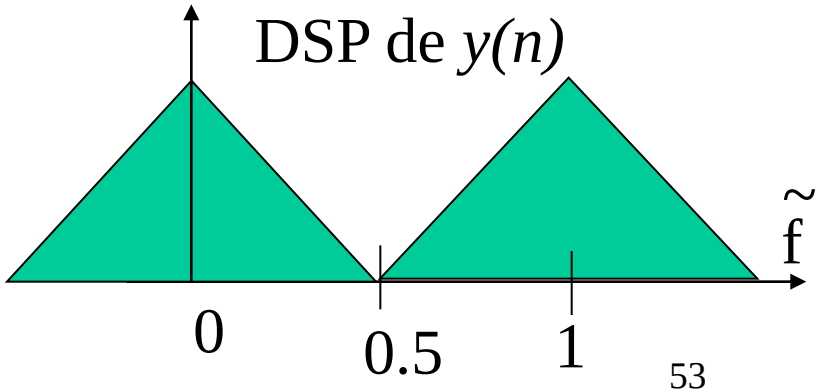
Décimateur


x(n)
h(k)
M
y(n)


DSP de x(n)
f (Hz)
0
Fe/2
Fe

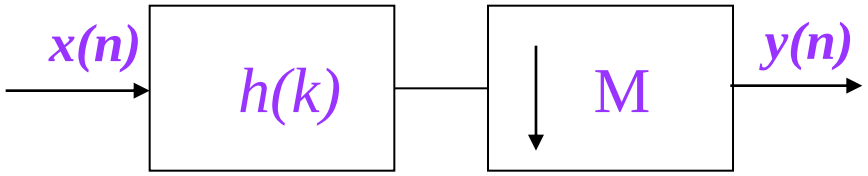

DSP de x(n)
f~
0
0.5
1

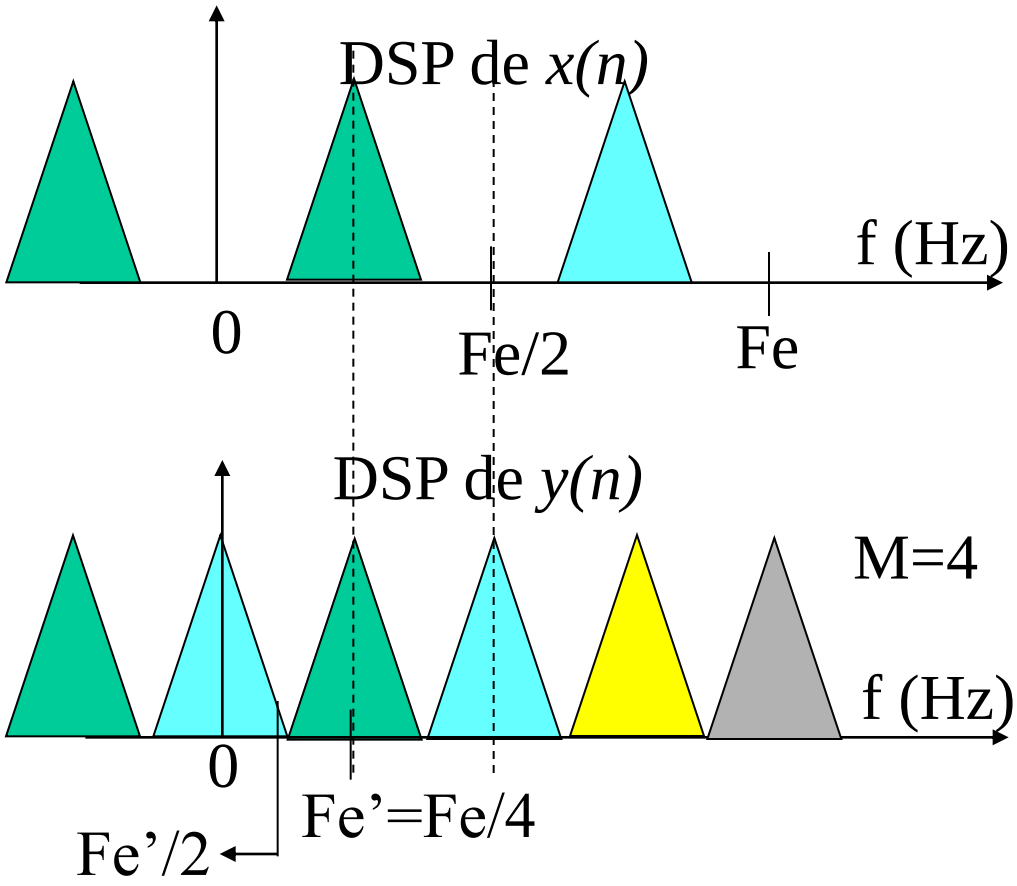

DSP de y(n)
M=4
f (Hz)
0
Fe'=Fe/4
Fe'/2


DSP de y(n)
f~
0
0.5
1

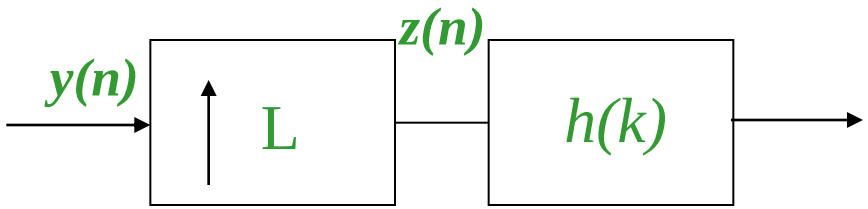
V- Codage avec perte : méthodes par transformées

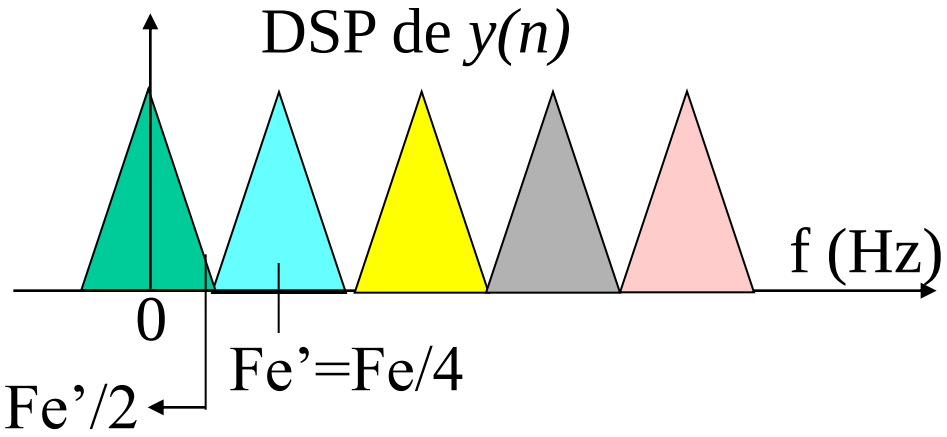
Décimateur

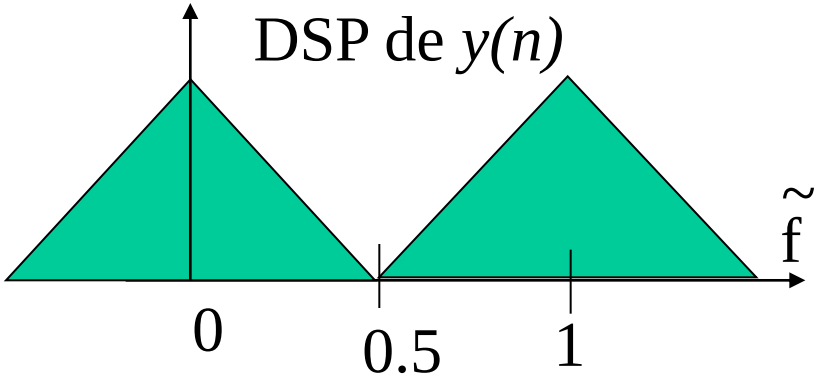


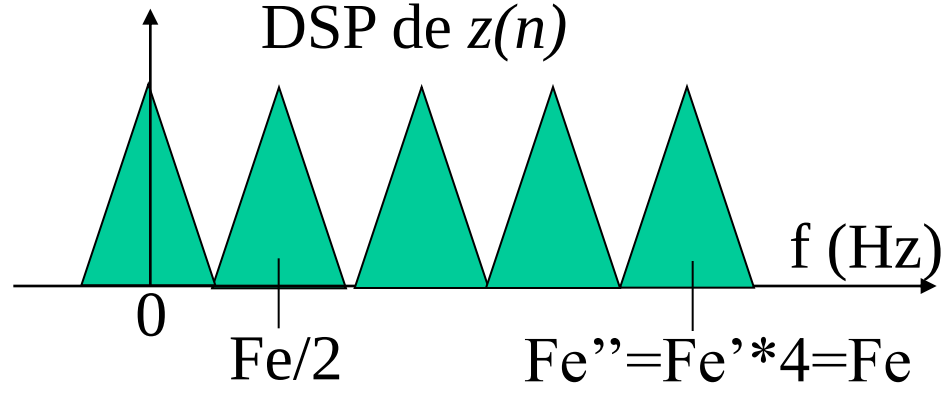


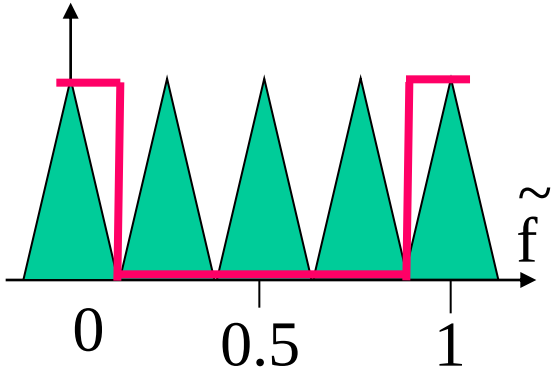
Interpolateur











Filtre anti-fantôme

ANNEXE 3

V- Codage avec perte : méthodes par transformées

Idea: Coefficients have a tree structure, high frequency wavelet pixels are descended from low frequency pixels.

If a low frequency pixel has low energy (and hence will be quantized into zero), then the same will likely be true for its descendents.
$\Rightarrow$ zerotree encoding

Successively send bits describing higher energy coefficients to greater accuracy.

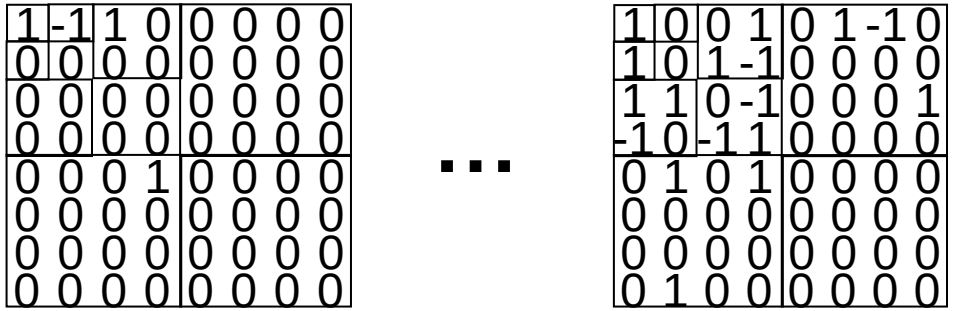
Additional bits describe most important coefficients with increasing accuracy and less important coefficients when their magnitude becomes significant.

Codage des coefficients d'ondelettes

Les clefs : train de bits emboités + codage par arbres

• « Embedded » : envoie les infos les + importantes d'abord,
codage / décodage peut s'arrêter n'importe où
et obtenir le meilleur résultat

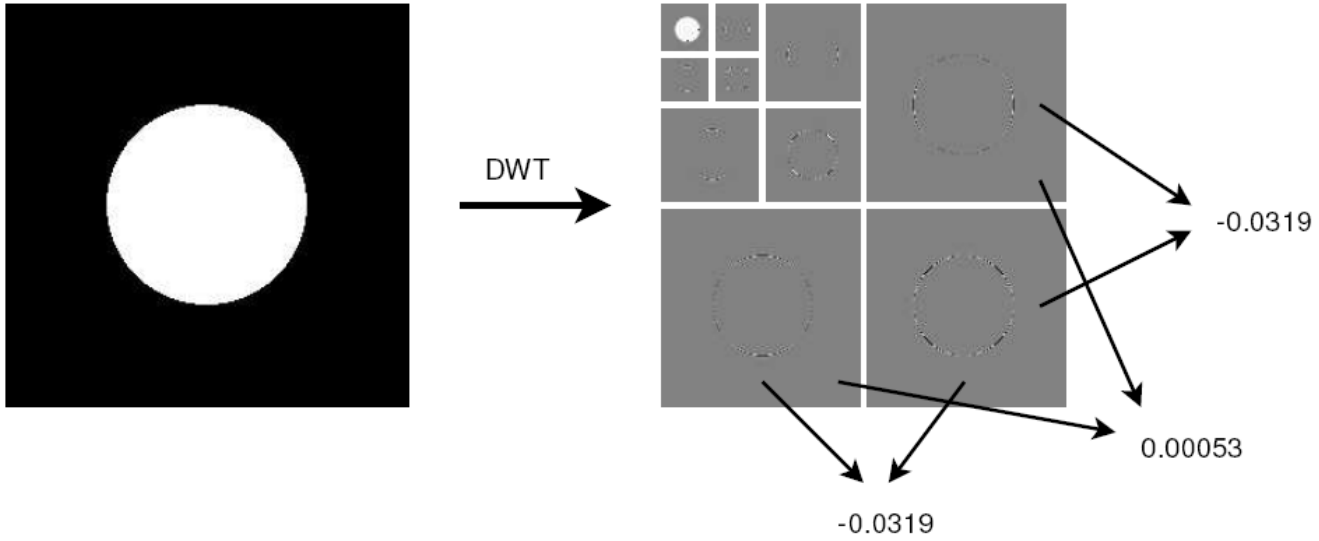
Codage par plan de bits



• « Zerotree » :

ordre de lecture
optimal
des coefficients

+ codage efficace
des zéros



Codage des coefficients d'ondelettes

Nombre de plans de bits : $n = \text{floor}(\log2\ (c_{max}))$
Pour chaque plan, un seuil : $T_k = 2^k$, $k=n,n-1,\ldots,0$

Un coefficient c est dit significatif au plan k si : $|c| >= T_k$

EZW : Embedded Zerotree coding of Wavelet coefficients

Shapiro, 1993

SPIHT : Set Partitioning in Hierarchical Trees

Said, Pearlman, 1996

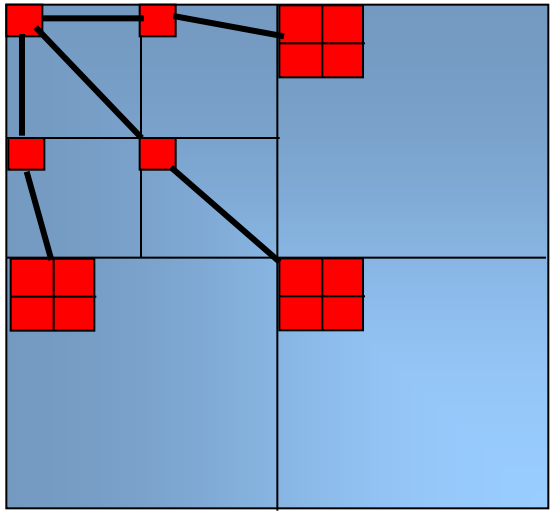
Codes (Matlab par ex) disponibles

La suite du document peut être chargée sur www.enseeiht.fr/~mailhes
Donnant les détails des algorithmes EZW et SPIHT
Non imprimés pour sauver la planète…

V- Codage avec perte : méthodes par transformées

EZW

Structure d'arbres :
Chaque nœud a 4 enfants
(sauf LL qui en a 3)



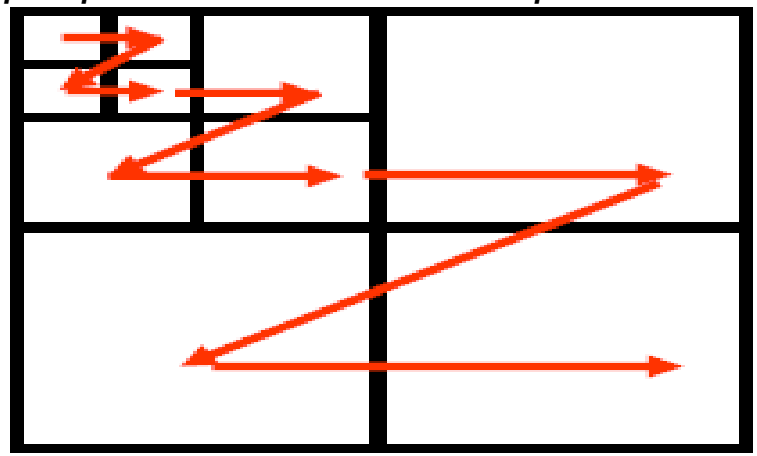
Codage : 4 symboles :

SP : coefficient significatif positif
SN : coefficient significatif négatif
ZTR : zerotree root :
coef et tous ses descendants non significatifs
IZ : isolated zero :
coef non significatif mais a des descendants significatifs

Algorithme à 2 passes pour chaque plan de bits :

Significance pass :
codage des coefficients
Refinement pass :
codage du bit suivant des coefs déclarés significatifs

Pour chaque plan de bits, sens de parcours :



Exemple (Shapiro, 1993)

Coefficients issus de la transformée en ondelettes :

63	-34	49	10	7	13	-12	7
-31	23	14	-13	3	4	6	-1
15	14	3	-12	5	-7	3	9
-9	-7	-14	8	4	-2	3	2
-5	9	-1	47	4	6	-2	2
3	0	-3	2	3	-2	0	4
2	-3	6	-4	3	6	3	6
5	11	5	6	0	3	-4	4

Seuils successifs correspondants aux plans de bits :
$T_5 = 32$, $T_4 = 16$, $T_3 = 8$, $T_2 = 4$, $T_1 = 2$, $T_0 = 1$

CODAGE

63	-34	49	10	7	13	-12	7
-31	23	14	-13	3	4	6	-1
15	14	3	-12	5	-7	3	9
-9	-7	-14	8	4	-2	3	2
-5	9	-1	47	4	6	-2	2
3	0	-3	2	3	-2	0	4
2	-3	6	-4	3	6	3	6
5	11	5	6	0	3	-4	4

plan de bits n°5 (MSB) : $T_5 = 32$

1	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	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	0	0	0	0	0
0	0	0	0	0	0	0	0

Significance pass :

SP (pour 63), SN (pour -34), IZ (pour -31, à cause du 47), ZTR (pour 23)
SP (pour 49), ZTR (pour 10), ZTR (pour 14), ZTR (pour -13)
ZTR (pour 15), IZ (pour 14, à cause du 47), ZTR (pour -9), ZTR (pour -7)
IZ (pour 7), IZ (pour 13), IZ (pour 3), IZ (pour 4)
IZ (pour -1), SP (pour 47), IZ (pour -3), IZ (pour 2)

33 symb. binaires

Remarque : lorsqu'on code des derniers enfants (qui n'ont pas de descendant), IZ ou ZTR sont équivalents et remplacés par Z (codé par un symbole)
SP -> 11, SN -> 10, IZ -> 01, ZTR -> 00 et Z -> 0

Refinement pass :

1 (pour bit suivant de 63), 0 (pour bit suivant de -34), 1 (pour bit suivant de 49), 0 (pour bit suivant de 47)

63	-34	49	10	7	13	-12	7
-31	23	14	-13	3	4	6	-1
15	14	3	-12	5	-7	3	9
-9	-7	-14	8	4	-2	3	2
-5	9	-1	47	4	6	-2	2
3	0	-3	2	3	-2	0	4
2	-3	6	-4	3	6	3	6
5	11	5	6	0	3	-4	4

plan de bits n°4 : $T_4 = 16$

1	0	1	0	0	0	0	0
1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Déjà codés à l'étape précédente

Significance pass :
SN (pour -31), SP (pour 23)
ZTR (pour 10), ZTR (pour 14), ZTR (pour -13),
ZTR (pour 15), ZTR (pour 14), ZTR (pour -9), ZTR (pour -7)
ZTR (pour 3), ZTR (pour -12), ZTR (pour -14), ZTR (pour 8)
Z, Z, Z, Z

34 symb. binaires

Refinement pass :
1 (pour bit suivant de 63), 0 (pour bit suivant de -34), 0 (pour bit suivant de 49),
1 (pour bit suivant de 47)
1 (pour bit suivant de -31), 0 (pour bit suivant de 23)

V- Codage avec perte : méthodes par transformées

63	-34	49	10	7	13	-12	7
-31	-23	14	-13	3	4	6	-1
15	14	3	-12	5	-7	3	9
-9	-7	-14	8	4	-2	3	2
-5	9	-1	47	4	6	-2	2
3	0	-3	2	3	-2	0	4
2	-3	6	-4	3	6	3	6
5	11	5	6	0	3	-4	4

plan de bits n°3 : $T_3 = 8$

1	0	0	1	0	1	1	0
1	0	1	1	0	0	0	0
1	1	0	1	0	0	0	1
1	0	1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0

Déjà codés à l'étape précédente

Significance pass :

SP (pour 10), SP (pour 14), SN (pour -13)
SP (pour 15), SP (pour 14), SN (pour -9), ZTR (pour -7)
ZTR (pour 3), SN (pour -12), SN (pour -14), SP (pour 8)
Z, SP, SN, Z, Z, Z, Z, Z, Z, Z, Z, SP, Z, Z, Z, Z,
Z, SP, Z Z, Z, Z, Z, Z, Z, Z, SP
Z, Z, Z, Z, Z, Z, Z, Z, Z, Z, Z, Z, Z, Z, Z, Z

Refinement pass : (voir le plan de bits suivant)

1, 0, 0, 1,
1, 1,
0, 1, 1, 1, 1, 0, 1, 1, 0,
1, 1, 0,
0, 0

etc...

1	0	0	0	1	1	1	1
1	1	1	1	0	1	1	0
1	1	0	1	1	1	0	0
0	1	1	0	1	0	0	0
1	0	0	1	1	1	0	0
0	0	0	0	0	0	0	1
0	0	1	1	0	1	63	0	1
1	0	1	1	0	0	1	1

Codage arithmétique

Compression en utilisant un codage arithmétique du train de bits généré
5 contextes :
-1 pour le « refinement pass »
- 4 pour le « significance pass »

DECODAGE

Reçoit :

SP, SN, IZ, ZTR …

1	-1	-	-	-	-	-	-
0	0	-	-	-	-	-	-
-	-	0	0	-	-	-	-
-	-	0	0	-	-	-	-
-	-	-	-	0	0	0	0
-	-	-	-	0	0	0	0
-	-	-	-	0	0	0	0
-	-	-	-	0	0	0	0

plan de bits MSB

…SP, ZTR, ZTR, ZTR …

1	-1	1	0	-	-	0	0
0	0	0	0	-	-	0	0
-	-	0	0	0	0	0	0
-	-	0	0	0	0	0	0
-	-	-	-	0	0	0	0
-	-	-	-	0	0	0	0
-	-	-	-	0	0	0	0
-	-	-	-	0	0	0	0

…ZTR, IZ, ZTR, ZTR …

1	-1	1	0	-	-	0	0
0	0	0	0	-	-	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	-	-	0	0	0	0
0	0	-	-	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

…Z, Z, Z, Z, Z, SP, Z, Z

1	-1	1	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	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	0	0	0	0	0
0	0	0	0	0	0	0	0

Plan de bits MSB fini

DECODAGE

Reçoit :

plan de bits MSB - 1

1, 0, 1, 0 …

…SN, SP, ZTR, ZTR, ZTR …

Plan de bits suivant

1	0	1	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	0	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

1	0	1	0	-	-	0	0
-1	1	0	0	-	-	0	0
-	-	-	-	0	0	0	0
-	-	-	-	0	0	0	0
-	-	-	0	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

…ZTR, ZTR, ZTR, ZTR,
ZTR, ZTR, ZTR, ZTR …

1	0	1	0	-	-	0	0
-1	1	0	0	-	-	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Z, Z, Z, Z

1	0	1	0	0	0	0	0
-1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Plan de bits fini

DECODAGE

Reçoit :

1, 0, 0, 1, 1, 0 … → …SP, SP, SN …

plan de bits MSB - 2

Plan de bits suivant

1	0	0	-	-	-	-	-
1	0	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	1	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-
-	-	-	-	-	-	-	-

…SP, SP, SN, ZTR…

1	0	0	1	-	-	-	-
1	0	1	-1	-	-	-	-
1	1	-	-	-	-	-	-
-1	0	-	-	-	-	-	-
-	-	-	1	-	-	-	-
-	-	-	-	-	-	-	-
-	-	0	0	-	-	-	-
-	-	0	0	-	-	-	-

…ZTR, SN, SN, SP, Z, SP, SN, Z…

1	0	0	1	0	1	-1	0
1	0	1	-1	0	0	0	0
1	1	0	-1	0	0	0	1
-1	0	-1	1	0	0	0	0
-	-	-	1	0	0	-	-
-	-	-	-	0	0	-	-
-	-	0	0	-	-	-	-
-	-	0	0	-	-	-	-

…Z, Z, Z, Z…

…Z, Z, Z, SP…

…Z, Z, Z, Z…

…Z, SP, Z …

…Z, Z, Z, Z…

…Z, Z, Z, SP…

1	0	0	1	0	1	-1	0
1	0	1	-1	0	0	0	0
1	1	0	-1	0	0	0	1
-1	0	-1	1	0	0	0	0
0	1	0	1	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	1	0	0	0	0	0	0

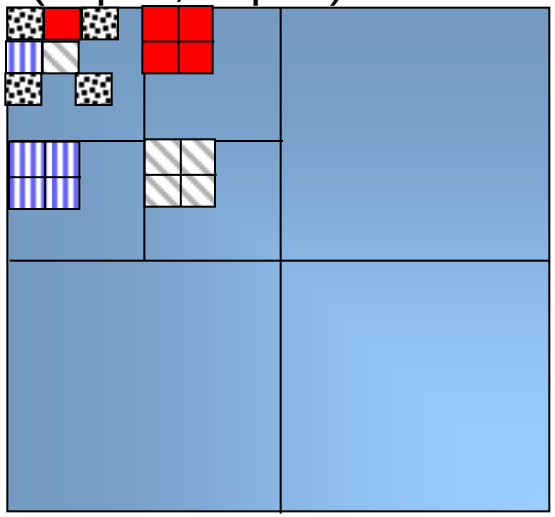
…Z, Z, Z, Z, Z, Z, Z, Z , Z, Z, Z, Z

V- Codage avec perte : méthodes par transformées

SPIHT

Structure d'arbres : Chaque nœud a 4 enfants
Sauf LL (impair, impair) = 0

Sans enfant

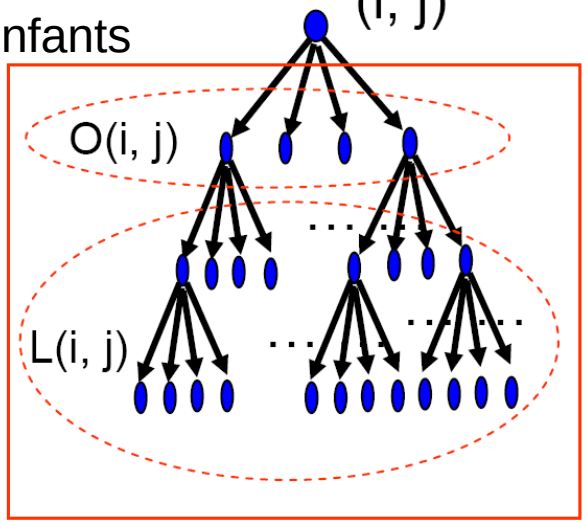


Codage : binaire

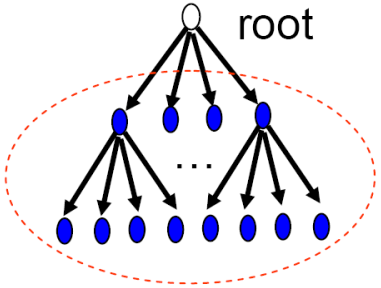
3 listes sont maintenues :
- LSP : List of Significant Pixels
- LIP : List of Insignificant Pixels
- LIS : List of Insignificant Sets
  (spécifiée par coordonnées de la racine)

ARBRES de Zéros :
- Dans EZW : la racine et tous les descendants sont non significatifs
- Dans SPIHT : la racine non incluse dans l'arbre.

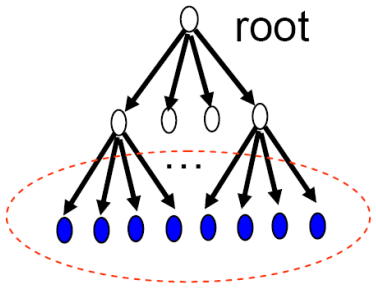
Arbres :
descendants, enfants, petits-enfants



Type A : D(i,j)
n'est pas significatif



Type B : L(i,j)
n'est pas significatif



V- Codage avec perte : méthodes par transformées

SPIHT

- « hierarchical trees » : un ensemble est significatif si un de ses coefs l'est
- Lorsqu'un arbre est significatif :
  - vérifier si les 4 enfants directs sont significatifs et les passer en LIP ou en LSP
  - vérifier si les petits-enfants (type B) sont significatifs,
    si oui, diviser en 4 sous-arbres
- LIP et LSP : passer un coef de la LIP à la LSP s'il devient significatif
  rafiner les coefs de la LSP à chaque passe.
- Initialisation
  - LIP : tous les coefs sans parent
  - LIS : coordonnées des racines de tous les arbres de type A par défaut,
  - LSP : vide

V- Codage avec perte : méthodes par transformées

SPIHT L'algorithme

Pour chaque plan de bits :

- Pour chaque entrée de la LIP :
  si le coef est significatif : envoyer 1 et signe, déplacer le coef dans LSP
  sinon : envoyer 0
- Pour chaque entrée de la LIS :
  - si l'ensemble est de type A :
    - si D(i,j) est non significatif, envoyer 0
    - sinon envoyer 1 et
      - vérifier les 4 enfants O(i,j)
        si significatif, envoyer 1 et signe, déplacer dans la LSP,
        sinon, envoyer 0 et déplacer dans la LIP
      - si L(i,j) n'est pas vide, déplacer (i,j) à la fin de la LIS, type B
        sinon, enlever (i,j) de la LIS
  - si l'ensemble est de type B :
    - si L(i,j) n'est pas significatif, envoyer 0
    - sinon, envoyer 1 et
      - ajouter les 4 enfants O(i,j) à la fin de la LIS, type A
      - enlever (i,j) de la LIS
- Raffinement : pour chaque entrée de la LSP (sauf les entrées de la passe actuelle) :
  envoyer le *k*ième bit de la valeur absolue
- Seuil suivant (plan de bits suivant) : on recommence.

SPIHT : exemple

Même exemple que pour EZW.

Initialisation :
LIP = { (1,1),(1,2),(2,1),(2,2) }
LIS = { (1,2)A, (2,1)A, (2,2)A }
LSP = Ø

Plan de bits MSB

1	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	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	0	0	0	0	0
0	0	0	0	0	0	0	0

- LIP={(1,1),(1,2),(2,1),(2,2) }
  - (1,1) significatif, envoyer 1+ et LSP ={(1,1)}, LIP={(1,2),(2,1),(2,2)}
  - (1,2) significatif, envoyer 1- et LSP= ={(1,1),(1,2)}, LIP={(2,1),(2,2)}
  - (2,1) non significatif, envoyer 0
  - (2,2) non significatif, envoyer 0, LSP= ={(1,1),(1,2)}, LIP={(2,1),(2,2)}
- LIS={(1,2)A ,(2,1)A ,(2,2)A }
  - pour (1,2)A : D est significatif, envoyer 1
    - vérifier ses enfants : envoyer 1+ et LSP ={(1,1),(1,2),(1,3)},
      envoyer 0 et LIP={(2,1),(2,2),(1,4)}
      envoyer 0 et LIP={(2,1),(2,2),(1,4),(2,3)}
      envoyer 0 et LIP ={(2,1),(2,2),(1,4),(2,3),(2,4)}
    - L : non vide donc LIS={(2,1)A ,(2,2)A, (1,2)B }
  - pour (2,1)A : D est significatif, envoyer 1
    - vérifier ses enfants : tous non significatifs : envoyer 0 0 0 0
      LIP={(2,1),(2,2)),(1,4),(2,3),(2,4),(3,1),(3,2),(4,1),(4,2)}
    - L : non vide : LIS={(2,2)A, (1,2)B, (2,1)B }
  - pour (2,2)A : D est non significatif, envoyer 0

SPIHT : suite

1	1	1	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	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	0	0	0	0	0
0	0	0	0	0	0	0	0

Plan de bits MSB

LIS={(2,2)A, (1,2)B, (2,1)B }
LSP ={(1,1),(1,2),(1,3)},
LIP={(2,1),(2,2)),(1,4),(2,3),(2,4),(3,1),(3,2),(4,1),(4,2)}

- on continue sur la LIS

pour (1,2)B : L non significatif, envoyer 0
pour (2,1)B : L est significatif, envoyer 1 et rajouter les enfants dans LIS :
LIS={(2,2)A, (1,2)B, (2,1)B, (3,1)A, (3,2)A, (4,1)A, (4,2)A }
mais enlever le noeud actuel de la LIS
LIS={(2,2)A, (1,2)B, (3,1)A, (3,2)A, (4,1)A, (4,2)A }
pour (3,1)A : D est non significatif, envoyer 0
pour (3,2)A : D est significatif, envoyer 1 et
enfants : envoyer 0 1+ 0 0 LSP ={(1,1),(1,2),(1,3),(5,4)},
LIP={(2,1),(2,2),(1,4),(2,3),(2,4),(3,1),(3,2),(4,1),(4,2),(5,3),(6,3),(6,4)}
L vide : LIS={(2,2)A, (1,2)B, (3,1)A, (4,1)A, (4,2)A }
pour (4,1)A : D est non significatif, envoyer 0
pour (4,2)A : D est non significatif, envoyer 0

FIN DE LA LIS

- pas de rafinement la 1ère fois
- on passe au plan de bits suivant.

29 symb. binaires

V- Codage avec perte : méthodes par transformées

1	0	1	0	0	0	0	0
1	1	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Plan de bits suivant

LSP ={(1,1),(1,2),(1,3),(5,4)}
LIS={(2,2)A, (1,2)B, (3,1)A, (4,1)A, (4,2)A }
LIP={(2,1),(2,2)),(1,4),(2,3),(2,4),(3,1),(3,2),(4,1),(4,2),(5,3),(6,3),(6,4)}

- Pour la LIP :
  - pour (2,1), envoyer 1-, déplacer dans LSP
  - pour (2,2), envoyer 1+, déplacer dans LSP
  - pour les 10 autres de la LIP, envoyer 0000000000
    - LSP ={(1,1),(1,2),(1,3),(5,4), (2,1),(2,2) }
    - LIP={(1,4),(2,3),(2,4),(3,1),(3,2),(4,1),(4,2),(5,3),(6,3),(6,4)}
- Pour la LIS :
  - pour (2,2)A, D non significatif, envoyer 0
  - pour (1,2)B, L non significatif, envoyer 0
  - pour (3,1)A, (4,1)A, (4,2)A : D non significatif, envoyer 000
- Rafinement : pour les entrées de la LSP de la passe précédente :
  - (1,1),(1,2),(1,3),(5,4), envoyer le bit du plan actuel : 1010
- Passage au plan de bits suivant etc…

23 symb. binaires

Décodage SPIHT

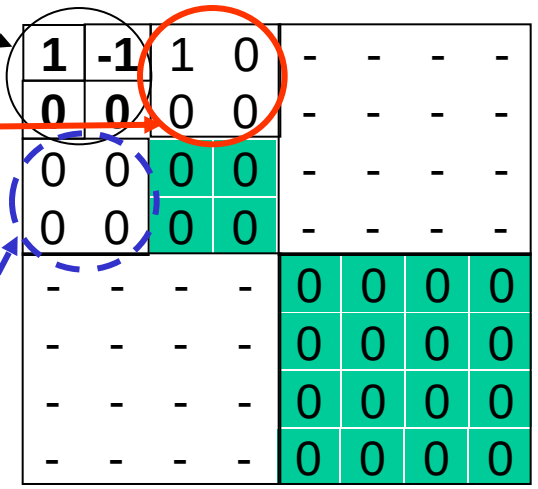
Initialisation :
LIP = { (1,1),(1,2),(2,1),(2,2) }
LIS = { (1,2)A, (2,1)A, (2,2)A }
LSP = Ø

Reçoit **1+1-00 1 1+000 1 0000 0 0 10101+0000**

- la LIP contient 4 éléments donc on reçoit 1+1-00 pour ces 4 éléments
  LSP = {(1,1),(1,2)} et LIP = {(2,1),(2,2) }
- Pour la LIS :
  - pour (1,2)A, reçoit 1 donc D significatif

  Les 4 suivants correspondent aux enfants
  1+000
  LSP = {(1,1),(1,2),(1,3)}
  LIP = {(2,1),(2,2),(1,4),(2,3),(2,4) }
  LIS = {(2,1)A, (2,2)A , (1,2)B}
  - pour (2,1)A reçoit 1 donc D significatif

  Les 4 suivants correspondent aux enfants
  0000
  LIP = {(2,1),(2,2),(1,4),(2,3),(2,4),(3,1),(3,2),(4,1),(4,2) }
  LIS ={(2,2)A , (1,2)B, {(2,1)B}
  - pour (2,2)A, reçoit 0 donc D non significatif



Décodage SPIHT

Reçoit **(1+1-00 1 1+000 1 0000 0) … 0 1 0 1 01+00 0 0**

LSP = {(1,1),(1,2),(1,3)}

LIP = {(2,1),(2,2),(1,4),(2,3),(2,4),(3,1),(3,2),(4,1),(4,2) }

LIS ={(2,2)A , (1,2)B, {(2,1)B}

- pour (1,2)B, reçoit 0 donc L non significatif
- pour (2,1)B reçoit 1 donc L significatif

LIS ={(2,2)A,(1,2)B,(3,1)A,(3,2)A,(4,1)A,(4,2)A}

- pour (3,1)A, reçoit 0 donc D non significatif
- pour (3,2)A, reçoit 1 donc D significatif

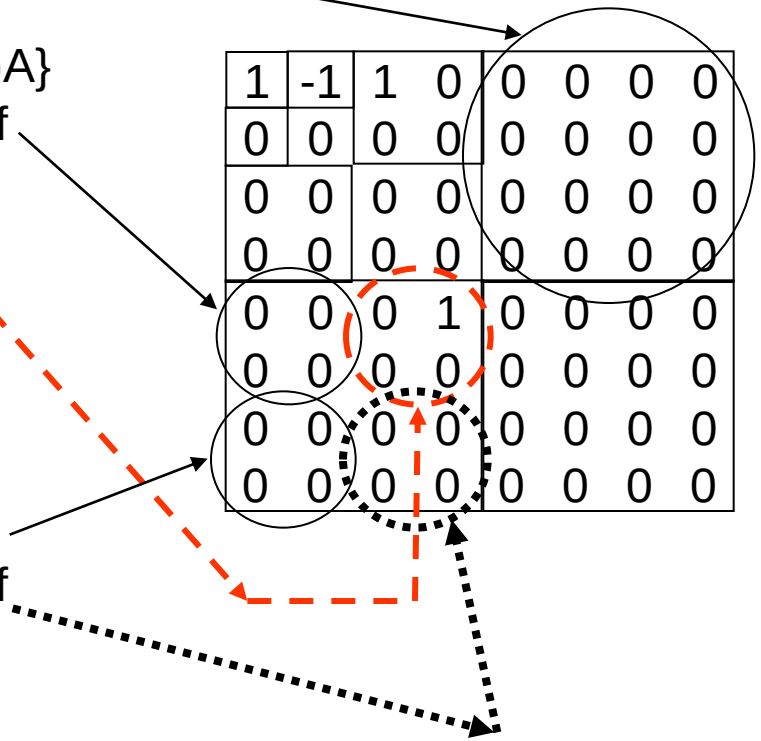
Les 4 suivants sont les 4 enfants : 01+00

LIS ={(2,2)A,(1,2)B,(3,1)A,(4,1)A,(4,2)A}

LSP = {(1,1),(1,2),(1,3),(5,4)}

LIP = {(2,1),(2,2),(1,4),(2,3),(2,4),
(3,1),(3,2),(4,1),(4,2),(5,3),(6,3),(6,4) }

- pour (4,1)A reçoit 0 donc D non significatif
- pour (4,2)A, reçoit 0 donc D non significatif



Plan de bits MSB fini

Décodage SPIHT

Plan de bits suivant

LIS ={(2,2)A, (1,2)B, (3,1)A, (4,1)A, (4,2)A} LSP = {(1,1),(1,2),(1,3),(5,4)}

LIP = {(2,1),(2,2),(1,4),(2,3),(2,4),(3,1),(3,2),(4,1),(4,2),(5,3),(6,3),(6,4) }

Reçoit **1-1+0000000000 0 0 0 0 0 1 0 1 0**

- la LIP contient 12 éléments : reçoit **1- 1+ 0000000000**

  LSP = {(1,1),(1,2),(1,3),(5,4),(2,1),(2,2)}

  LIP = {(1,4),(2,3),(2,4),(3,1),(3,2),(4,1),(4,2),
  (5,3),(6,3),(6,4) }

  - la LIS

  Pour (2,2)A, reçoit 0 : D non significatif

  Pour (1,2)B, reçoit 0 : L non significatif

  Pour (3,1)A, reçoit 0 : D non significatif

  Pour (4,1)A, reçoit 0 : D non significatif

  Pour (4,2)A, reçoit 0 : D non significatif

  - les bits suivants sont de rafinement

  Voir LSP : 1010 bits de ces noeuds