Introduction to image Compression

Christophe Stolz

2014

Plan du cours

I. Introduction
   several source of acquisition
   necessity to store high quantity of data

II. Lossless methods

III. Lossy methods

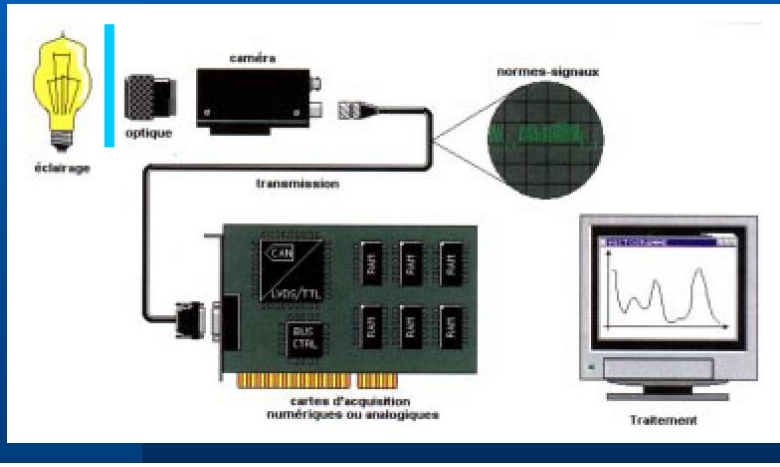
IV JPEG norm

V MPEG Norm

I. Introduction

- Recall: classic acquisition

Filters



3

I. Introduction

- Gray level or color image
- Acquisition mode :
  CCD (visible) or particular sensor (UV, IR, X rays)
- Image usage: analysis, storage, transmission

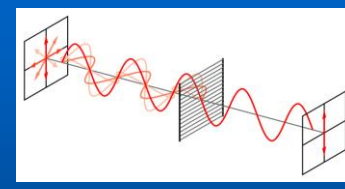
Example: UV camera Ofil DayCor II

Analysis of electric lines

4

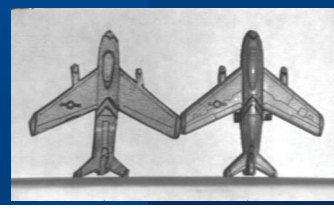
I. Introduction: rappels

- Polarizer Filter
  absorb a component
  => produce linearly
  polarized light
- Measure of 2 projections :
  degree of polarization



$$\frac{I_{\perp} - I_{//}}{I_{\perp} + I_{//}}$$

2 images of 1024x2014 12bits pixels
Approx 3Mb



5

I Introduction : size of data

For instance, a still image of the resolution of 480x640 pixels with 24-bit intensity levels has the data size of 480x640x24 = 7.4 megabits!

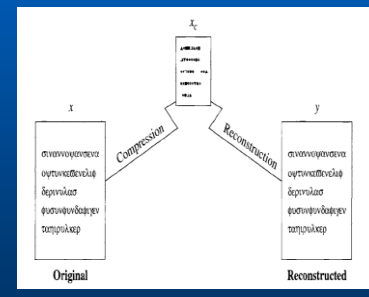
Video images, 30 frames/second, of the resolution 768x1024 pixels with 24-bit levels has the data size of 566 megabits per second

Résolution	Format	Débit	sur cd
352x288	YUV	5 Mo/s	2 min
	MJPEG 6.6	768 Ko/s	14 min
	MPEG1	175 Ko/s	60 min
	DIVX 910 kbs	114 Ko/s	95 min
640x480	YUV	15 Mo/s	40 secondes
	MJPEG 6.6	2,3 Mo/s	4,5 min
	MPEG2	490 Ko/s	22 min
	DIVX 910 kbs	114 Ko/s	40 min

6

I Introduction : Fundamentals

- Compression:
  - Logic
  - Physic
  - Symmetric et asymmetric
  - Adaptive or not
  - Lossy or lossless
- Compression ratio:
  size of compressed image TC,
  size of original image TO
  T = TC / T0
  => T small, small file
- Classic scheme :
  Extract the symbols, then model and code



I Introduction : Fundamentals

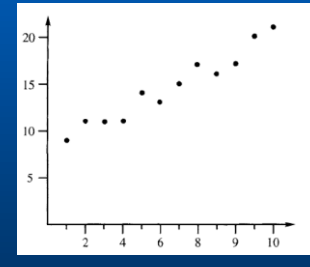
- Example: sequence :

9	11	11	11	14	13	15	17	16	17	20	21

- Linear model : $\hat{x}_n = x_n + 8$
- Error

$e_n = x_n - \hat{x}_n$: 0 1 0 -1 1 -1 0 1 -1 -1 1 1

- Compress by coding the error here with 3 values : -1, 0, 1
- Extension for any sequence



27	28	29	28	26	27	29	28	30	32	34	36	38

=> gives

27	1	1	−1	−2	1	2	−1	2	2	2	2	2

I Introduction : Fundamentals

If n_1 and n_2 denote the number of information-carrying units in two data sets that represent the same information, the *relative data redundancy* R_D of the first data set, n_1 is defined as

$$R_D = 1 - \frac{1}{C_R}$$

where C_R, commonly called the *compression ratio*, is

$$C_R = \frac{n_1}{n_2}.$$

9

I Introduction : Fundamentals

- If *n1=n2*, then *CR*=1 and *RD*=0.
  This indicates that the first data has no redundant data.
- If *n1>>n2*, then $CR \to \infty$ and *RD*=1.
  This implies significant compression and highly redundant data.
- If *n2>>n1*, then $CR \to 0$ and $RD \to \infty$.
  This implies data expansion, which is undesirable.

10

I Introduction : Definitions

- Lossless compression : the original image can be reconstructed perfectly as there is no information loss.
  Compression rates typically range from 2:1 to 50:1.
  Example: Run-length coding (JPEG), Huffman coding, LZW coding (TIFF, GIF), Bit-plane coding etc.

- Lossy compression : the original image and its reconstruction are not identical owing to some information loss.
  Psychological properties of the eye may be taken into account.
  Higher compression rates are possible than with lossless compression, typically up to 100:1.
  Example: DPCM, JPEG, MPEG etc.

11

II Lossless compression

- Entropy: average self information (length) of a message

$$H = -\sum_{i \in S} p_i \log_2 \left(p_i \right)$$

- Example 1 2 3 2 3 4 5 4 5 6 7 8 9 8 9 10

$$P(1) = P(6) = P(7) = P(10) = \tfrac{1}{16}$$
$$P(2) = P(3) = P(4) = P(5) = P(8) = P(9) = \tfrac{2}{16}.$$

H = 3.25 bits

- Not possible to compress without data loss with a data flow inferior to the entropy of the source.

12

II Lossless compression

- Estimated entropy is reduced after coding with the sequence of residues

  1 1 1 −1 1 1 1 −1 1 1 1 1 1 1 −1 1 1

  P(1)=13/16 and P(-1)=3/16 so H=0.70 bits/symbol
- Reconstruction : $x_n = x_{n-1} + r_n$
- Other sequence

  1 2 1 2 3 3 3 3 1 2 3 3 3 3 1 2 3 3 1 2

  P(1)=P(2)=1/4 and P(3)=1/2 => H=1.5 bits / symbol
  => 20 symbols => 30 bits to code

  better here to code groups of symbols => H= 1 bit/symbol

II Lossless compression

- Coding based on statistics and propabilities

TABLE 3.26 **Probabilities of occurrence of the letters in the English alphabet in the U.S. Constitution.**

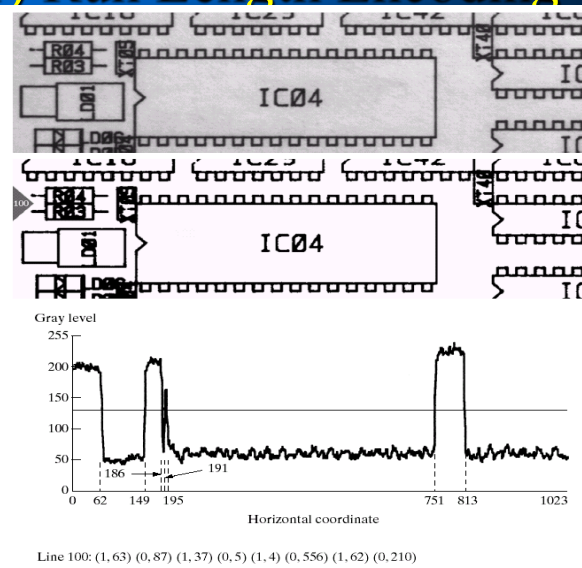
Letter	Probability	Letter	Probability
A	0.057305	N	0.056035
B	0.014876	O	0.058215
C	0.025775	P	0.021034
D	0.026811	Q	0.000973
E	0.112578	R	0.048819
F	0.022875	S	0.060289
G	0.009523	T	0.078085
H	0.042915	U	0.018474
I	0.053475	V	0.009882
J	0.002031	W	0.007576
K	0.001016	X	0.002264
L	0.031403	Y	0.011702
M	0.015892	Z	0.001502

1) Run Length Encoding

- Work on byte
- Formats TIFF, BMP, PCX, fax …
- Principle : reduce the size by encoding the redundancy:
  - AAAAAA coded 6A (redundancy/value)
  - but : ABCDEF => 1A1B1C1D1E1F
- Improvement: 3 bytes flag/counter/code
  - Rule: if flag = 255 then 3 bytes; if not flag then 1 byte
  - Ex: 28 times code ASCII 53
    13 times code 212
    1 times code 37
    1 times code 53
    1 times code 12
    4 times code 119

255/27/53/ 255/12/212/ 37/ 53/ 12/ 255/3/119

1) Run Length Encoding



a
b
c
d

FIGURE 8.3 Illustration of run-length coding: (a) original image. (b) Binary image with line 100 marked. (c) Line profile and binarization threshold. (d) Run-length code.

Line 100: (1, 63) (0, 87) (1, 37) (0, 5) (1, 4) (0, 556) (1, 62) (0, 210)

1) Run Length Encoding

- In the previous slide, the entire 1024×343 image was reduced to 12,166 runs. Thus, the compression ratio and the relative redundancy can be obtained by

$$C_R = \frac{n_1}{n_2} = \frac{1024 \cdot 343 \cdot 1}{12166 \cdot 11} = 2.63$$

$$R_D = 1 - \frac{1}{C_R} = 1 - \frac{1}{2.63} = 0.62 .$$

- (Consider why 11 bits are used for each run.)

17

1) Run Length Encoding

- Scan strategy :

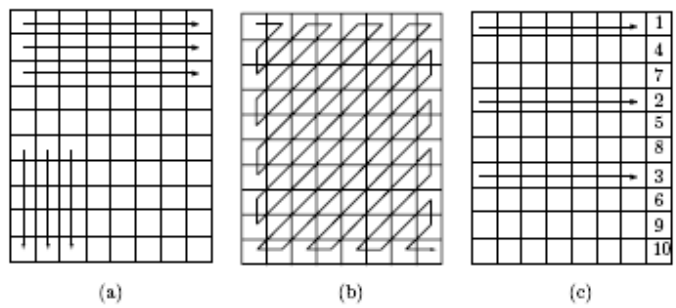


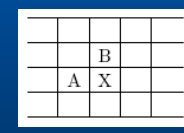
Figure 1.8: RLE Scanning.

18

1) Run Length Encoding

- Conditional RLE : lossy
- Assuming an image with n grayscales, the method starts by assigning an n-bit code to each pixel depending on its near neighbors. It then concatenates the n-bit codes into a long string and calculates run lengths. The run lengths are assigned prefix codes (Huffman or other) that are written on the compressed stream
- Conditional probabilities P(X| A,B)
- Example with 4 bits codes

A	B		W1	W2	W3	W4	W5	W6	W7 ...
2	15	value:	4	3	10	0	6	8	1 ...
		count:	21	6	5	4	2	2	1 ...
3	0	value:	0	1	3	2	11	4	15 ...
		count:	443	114	75	64	56	19	12 ...
3	1	value:	1	2	3	4	0	5	6 ...
		count:	1139	817	522	75	55	20	8 ...
3	2	value:	2	3	1	4	5	6	0 ...
		count:	7902	4636	426	264	64	18	6 ...
3	3	value:	3	2	4	5	1	6	7 ...
		count:	33927	2869	2511	138	93	51	18 ...
3	4	value:	4	3	5	2	6	7	1 ...
		count:	2859	2442	240	231	53	31	13 ...



1b) Move To front

- Maintain the alphabet A of symbols as a list where frequently occuring symbols are located near the front
- Symbols encoded as the number of symbols that precede in the list
- The input stream abcddcbamnopponm is encoded as *C* = (0, 1, 2, 3, 0, 1, 2, 3, 4, 5, 6, 7, 0, 1, 2, 3)
- Without the move-to-front step it is encoded as *C′* = (0, 1, 2, 3, 3, 2, 1, 0, 4, 5, 6, 7, 7, 6, 5, 4).
- Both *C* and *C′* contain codes in the same range [0, 7], but the elements of *C* are smaller on the average, since the input starts with a concentration of abcd and continues with a concentration of mnop. (The average value of *C* is 2.5, while that of *C′* is 3.5.)
- Can be mixed with a variable length code

1b) Move To front

- Example

a	abcdmnop	0	a	abcdmnop	0	a	abcdmnop	0	a	abcdmnop	0
b	abcdmnop	1	b	abcdmnop	1	b	abcdmnop	1	b	abcdmnop	1
c	bacdmnop	2	c	abcdmnop	2	c	bacdmnop	2	c	abcdmnop	2
d	cbadmnop	3	d	abcdmnop	3	d	cbadmnop	3	d	abcdmnop	3
d	dcbamnop	0	d	abcdmnop	3	m	dcbamnop	4	m	abcdmnop	4
c	dcbamnop	1	c	abcdmnop	2	n	mdcbanop	5	n	abcdmnop	5
b	cdbamnop	2	b	abcdmnop	1	o	nmdcbaop	6	o	abcdmnop	6
a	bcdamnop	3	a	abcdmnop	0	p	onmdcbap	7	p	abcdmnop	7
m	abcdmnop	4	m	abcdmnop	4	a	ponmdcba	7	a	abcdmnop	0
n	mabcdnop	5	n	abcdmnop	5	b	aponmdcb	7	b	abcdmnop	1
o	nmabcdop	6	o	abcdmnop	6	c	baponmdc	7	c	abcdmnop	2
p	onmabcdp	7	p	abcdmnop	7	d	cbaponmd	7	d	abcdmnop	3
p	ponmabcd	0	p	abcdmnop	7	m	dcbaponm	7	m	abcdmnop	4
o	ponmabcd	1	o	abcdmnop	6	n	mdcbapon	7	n	abcdmnop	5
n	opnmabcd	2	n	abcdmnop	5	o	nmdcbapo	7	o	abcdmnop	6
m	nopmabcd	3	m	abcdmnop	4	p	onmdcbap	7	p	abcdmnop	7
	mnopabcd						ponmdcba				
	(a)			(b)			(c)			(d)	

Table 1.14: Encoding With and Without Move-to-Front.

1c) Recursive Range Reduction

- Eliminating some most significant zero bits of integers
- Integer list is sorted
- Work well with data decreasing fast , but not too fast

Data	RR code	Rice code
	0101 = 6	
1011101	011101	000001 1101
1001011	1001011	00001 1011
0110001	**0**110001	0001 0001
0101100	101100	001 1100
0001110	**00**1110	1 1110
0001101	1101	1 1101
0001100	1100	1 1100
0001001	1001	1 1001
0000010	**00**10	1 0010
0000001	01	1 0001
70	53	64 bits

2) Bit plane coding

- Another effective technique for reducing an image's inter-pixel redundancies is to process the image's bit planes individually. The technique, called *bit-plane coding*, is based on the concept of decomposing a multilevel image into a series of binary images and compressing each binary image by run-length coding
- The figure below illustrates bit-planes of an 8-bit image. Each bit generates a binary image of the same size as the original image.

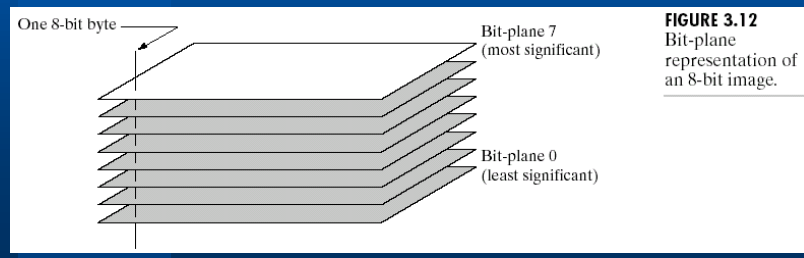


FIGURE 3.12 Bit-plane representation of an 8-bit image.

2) Bit plane coding

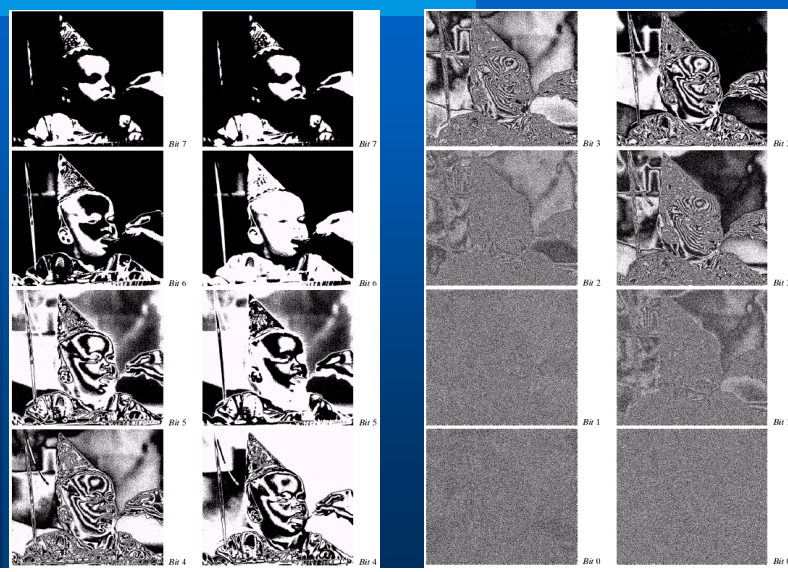
- The inherent disadvantage of bit-plane coding is that small changes in gray level can have a great effect on the resulting bit planes. For example, when a pixel value 127 (0111 11112) is adjacent to a pixel of value 128 (1000 00002), every bit plane will have a 0 to 1 (or 1 to 0) transition.
- This effect of small gray-level variations has a negative impact on the compression ratio.
- To cope with this problem, pixel values are converted to a Gray code that is obtained by

$$g_i = a_i \oplus a_{i+1} \qquad 0 \le i \le m-2$$
$$g_{m-1} = a_{m-1},$$

where the gray levels of an *m*-bit gray-scale image can be represented in the form of the base 2 polynomial.

$$a_{m-1}2^{m-1} + a_{m-2}2^{m-2} + \cdots + a_1 2^1 + a_0 2^0 .$$

2) Bit-Planes and Gray-coded Bit-Planes



3) Variable length coding

- Variable-length coding achieves data compression by assigning fewer bits to the most probable gray levels than to the less probable ones.

r_k	$p_r(r_k)$	Code 1	$l_1(r_k)$	Code 2	$l_2(r_k)$
$r_0 = 0$	0.19	000	3	11	2
$r_1 = 1/7$	0.25	001	3	01	2
$r_2 = 2/7$	0.21	010	3	10	2
$r_3 = 3/7$	0.16	011	3	001	3
$r_4 = 4/7$	0.08	100	3	0001	4
$r_5 = 5/7$	0.06	101	3	00001	5
$r_6 = 6/7$	0.03	110	3	000001	6
$r_7 = 1$	0.02	111	3	000000	6

TABLE 8.1 Example of variable-length coding.

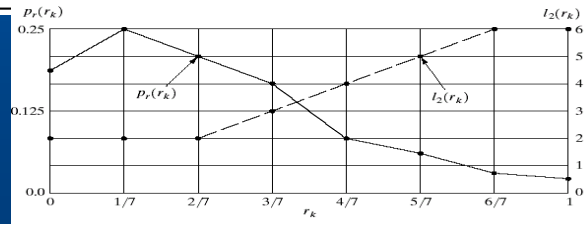


FIGURE 8.1 Graphic representation of the fundamental basis of data compression through variable-length coding.

3) Variable length coding

- Prefix property :
  once a certain bit pattern has been assigned as the code of a symbol, no other codes should start with that pattern
- Example :
- Code string :
  a1a3a2a1a3a3a4a2a1a1a2a2a1a1a3a1a1a2a3a1
- code 1 : ambiguous, code 2 no

Symbol	Prob.	Code1	Code2
a_1	.49	1	1
a_2	.25	01	01
a_3	.25	010	000
a_4	.01	001	001

Table 2.2: Variable-Size Codes.

27

3) Variable length coding

- *Unary code :* the positive integer *n* is defined as *n* − 1 ones followed by a single 0 or, alternatively, as *n* − 1 zeros followed by a single one
- General unary codes : start/step/stop

n	Code	Alt. Code
1	0	1
2	10	01
3	110	001
4	1110	0001
5	11110	00001

n	$a = 3 + n \cdot 2$	nth codeword	Number of codewords	Range of integers
0	3	$0xxx$	$2^3 = 8$	0–7
1	5	$10xxxxx$	$2^5 = 32$	8–39
2	7	$110xxxxxxx$	$2^7 = 128$	40–167
3	9	$111xxxxxxxxx$	$2^9 = 512$	168–679
		Total	680	

Table 2.4: The General Unary Code (3,2,9).

28

3) Shannon-Fano coding

- Symbols sorted in decreasing order of their probabilities
- Division in two subsets representing aproximately half of the probabilities
- Rq : no code is the starting of an other one

	Prob.	Steps				Final
1.	0.25	1	1			:11
2.	0.20	1	0			:10
3.	0.15	0		1	1	:011
4.	0.15	0		1	0	:010
5.	0.10	0		0	1	:001
6.	0.10	0		0	0	1 :0001
7.	0.05	0		0	0	0 :0000

29

3) Huffman coding

- Bit level coding
- Uses the statistics of the information
- Each symbol of the source is represented by a binary word of length inversely proportional to it's frequency
- Mechanism:
  - 1) build the occurrence table (search of probabilities p_i)
  - 2) reduction by grouping the 2 smallest frequencies; affect "0" to the smallest sort by decreasing order
  - Iterate to the root

TABLE 3.28 Huffman coding of 16-bit CD-quality audio.

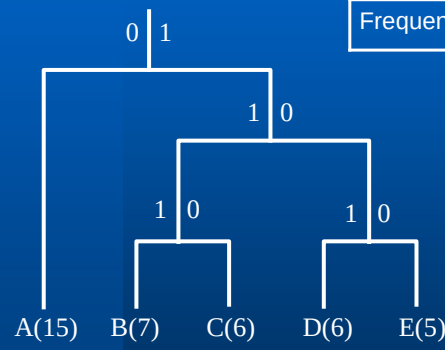
File Name	Original File Size (bytes)	Entropy (bits)	Estimated Compressed File Size (bytes)	Compression Ratio
Mozart	939,862	12.8	725,420	1.30
Cohn	402,442	13.8	349,300	1.15
Mir	884,020	13.7	759,540	1.16

30

b) Huffman code

- Example

Symbols	A	B	C	D	E
Frequencies	15	7	6	6	5



Symbol	Code
A	0
B	111
C	110
D	101
E	100

Message: ABCBEAC => 01111101111000110 => 7D | E3 …

31

b) Huffman code

Exercise 1:
Obtain the Huffman code for the characters from A to G with the following frequencies.

A: 29
B: 64
C: 32
D: 12
E: 9
F: 66
G: 23

32

b) Huffman code

Exercise 2:
Obtain the Huffman code for the image of 8 gray levels with the following occurrence probabilities, and also calculate the average bits/pixel, compression ratio C_R, and the relative data redundancy R_D.

Gray levels	Probabilities
0	0.21
1	0.27
2	0.24
3	0.11
4	0.08
5	0.05
6	0.03
7	0.01

33

c) Adaptative code

- Adapt to the incoming data stream
- Build the same dictionary onto the coder and the decoder => provide transmission. coder

```
Initialise_model();
Do {
C=getc(input);
Encode(c,output);
Update_model( c );
}
While (c!=EOF)
```

```
Initialise_model();
While ( (c=decode(input)) !=EOF) {
        put(c, output);
        update_model( c);
}
```

- Methods : Frequency coding, Adaptive Huffman and Lempel Ziv Welch (LZW)

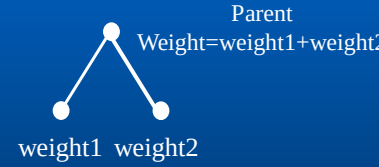
34

Frequency coding

- Work on byte
- 2 parts coding: header/body (variable size)
- Header: define size of the code (ex with 3 bits 8 possibilities)
- The last symbol fix the end of the message
- Compression rate <= 2
- Not used yet

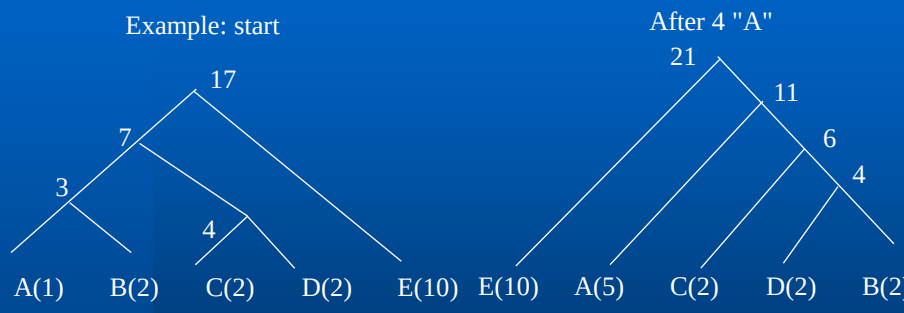
Adaptative Huffman

- Work on byte
- Property of "fraternity" : for a binary tree with leaf having an increasing weight.



- When a symbol come:
  if non respect of the property
  then exchange 2 terms
  without changing the tree
  (actual symbol and symbol the most far symbol with same weight)

Adaptive Huffman



- Problem: initializing the tree
  Empty or all symbols with same weight

37

Adaptive Huffman

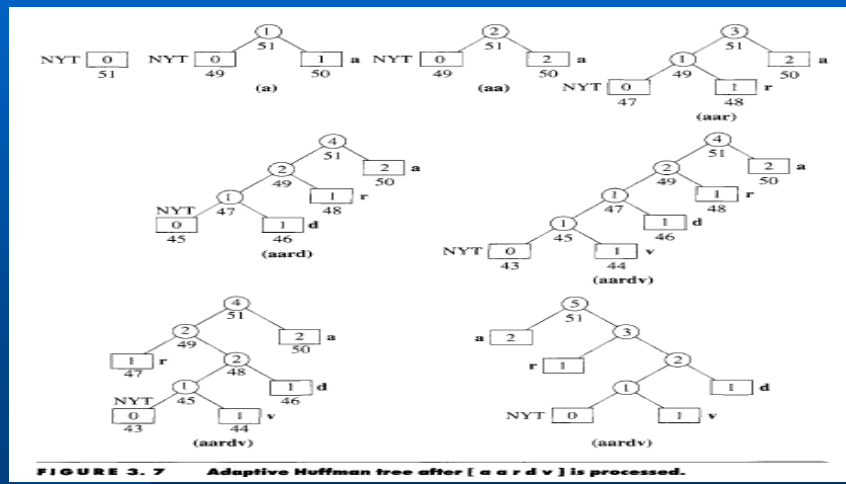
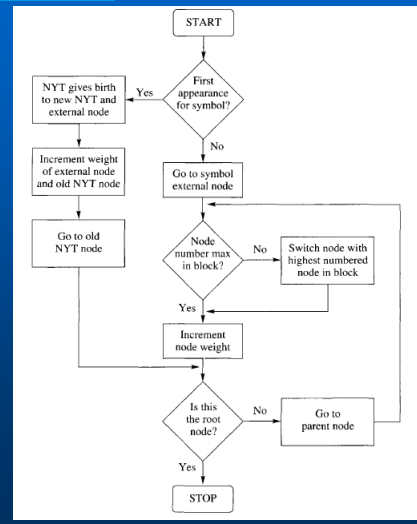


FIGURE 3.7 Adaptive Huffman tree after [a a r d v] is processed.

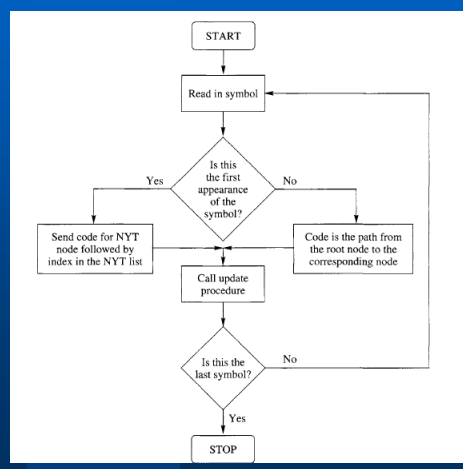
Adaptive Huffman

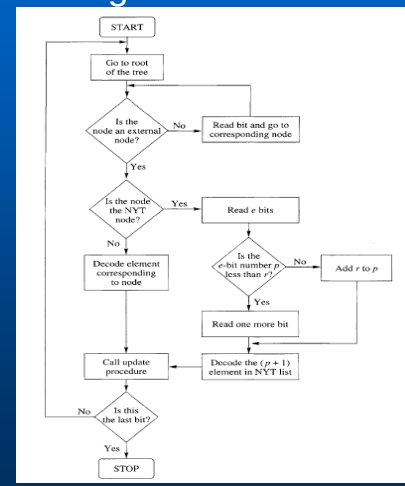
- Flowchart update procedure



Adaptive Huffman

- Flowchart : encoding / decoding





Arithmetic coding

- 1. Start by defining the “current interval” as [0, 1).
- 2. Repeat the following two steps for each symbol s in the input stream:
  2.1. Divide the current interval into subintervals whose sizes are proportional to the symbols’ probabilities.
  2.2. Select the subinterval for s and define it as the new current interval.
- 3. When the entire input stream has been processed in this way, the output should be any number that uniquely identifies the current interval (i.e., any number inside the current interval).

41

Arithmetic coding

- Example

Consider a three-letter alphabet $\mathcal{A} = \{a_1, a_2, a_3\}$ with $P(a_1) = 0.7$, $P(a_2) = 0.1$, and $P(a_3) = 0.2$. Using the mapping of Equation (4.1), $F_X(1) = 0.7$, $F_X(2) = 0.8$, and $F_X(3) = 1$. This partitions the unit interval as shown in Figure 4.1.

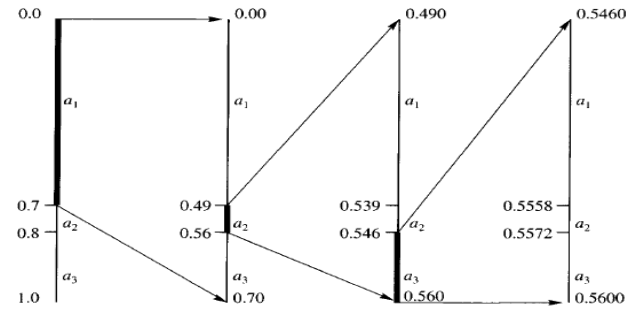


FIGURE 4. 1 **Restricting the interval containing the tag for the input sequence $\{a_1, a_2, a_3, \ldots\}$.**

42

Arithmetic coding

- Example : SWISS MISS

Char	Freq	Prob.	Range	CumFreq
		Total CumFreq=		10
S	5	$5/10 = 0.5$	[0.5, 1.0)	5
W	1	$1/10 = 0.1$	[0.4, 0.5)	4
I	2	$2/10 = 0.2$	[0.2, 0.4)	2
M	1	$1/10 = 0.1$	[0.1, 0.2)	1
␣	1	$1/10 = 0.1$	[0.0, 0.1)	0

- Update of Low and High bounds :
  NewHigh:=OldLow+Range*HighRange(X);
  NewLow:=OldLow+Range*LowRange(X);
  Range=OldHigh−*OldLow and LowRange(X), HighRange(X) indicate the low and* high limits of the range of symbol X

Arithmetic coding

- Example : SWISS MISS

Char.		The calculation of low and high
S	L	$0.0 + (1.0 - 0.0) \times 0.5 = 0.5$
	H	$0.0 + (1.0 - 0.0) \times 1.0 = 1.0$
W	L	$0.5 + (1.0 - 0.5) \times 0.4 = 0.70$
	H	$0.5 + (1.0 - 0.5) \times 0.5 = 0.75$
I	L	$0.7 + (0.75 - 0.70) \times 0.2 = 0.71$
	H	$0.7 + (0.75 - 0.70) \times 0.4 = 0.72$
S	L	$0.71 + (0.72 - 0.71) \times 0.5 = 0.715$
	H	$0.71 + (0.72 - 0.71) \times 1.0 = 0.72$
S	L	$0.715 + (0.72 - 0.715) \times 0.5 = 0.7175$
	H	$0.715 + (0.72 - 0.715) \times 1.0 = 0.72$
␣	L	$0.7175 + (0.72 - 0.7175) \times 0.0 = 0.7175$
	H	$0.7175 + (0.72 - 0.7175) \times 0.1 = 0.71775$
M	L	$0.7175 + (0.71775 - 0.7175) \times 0.1 = 0.717525$
	H	$0.7175 + (0.71775 - 0.7175) \times 0.2 = 0.717550$
I	L	$0.717525 + (0.71755 - 0.717525) \times 0.2 = 0.717530$
	H	$0.717525 + (0.71755 - 0.717525) \times 0.4 = 0.717535$
S	L	$0.717530 + (0.717535 - 0.717530) \times 0.5 = 0.7175325$
	H	$0.717530 + (0.717535 - 0.717530) \times 1.0 = 0.717535$
S	L	$0.7175325 + (0.717535 - 0.7175325) \times 0.5 = 0.71753375$
	H	$0.7175325 + (0.717535 - 0.7175325) \times 1.0 = 0.717535$

Table 2.48: The Process of Arithmetic Encoding.

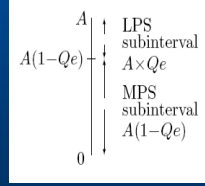
Char.	Code−low		Range
S	$0.71753375 - 0.5$	$= 0.21753375$	$/0.5 = 0.4350675$
W	$0.4350675 - 0.4$	$= 0.0350675$	$/0.1 = 0.350675$
I	$0.350675 - 0.2$	$= 0.150675$	$/0.2 = 0.753375$
S	$0.753375 - 0.5$	$= 0.253375$	$/0.5 = 0.50675$
S	$0.50675 - 0.5$	$= 0.00675$	$/0.5 = 0.0135$
␣	$0.0135 - 0$	$= 0.0135$	$/0.1 = 0.135$
M	$0.135 - 0.1$	$= 0.035$	$/0.1 = 0.35$
I	$0.35 - 0.2$	$= 0.15$	$/0.2 = 0.75$
S	$0.75 - 0.5$	$= 0.25$	$/0.5 = 0.5$
S	$0.5 - 0.5$	$= 0$	$/0.5 = 0$

Table 2.50: The Process of Arithmetic Decoding.

QM coder

- JPEG
- main idea : classify each input symbol (which is a single bit) as either the more probable symbol (MPS) or the less probable symbol (LPS).
- Rules :

After MPS: C is unchanged, $A \leftarrow A(1-Qe)$,
After LPS: $C \leftarrow C + A(1-Qe)$, $A \leftarrow A \times Qe$.



45

String compression

- compression methods based on strings of symbols can be more efficient than methods that compress individual symbols.
- Ex : huffman code of 2 symbols

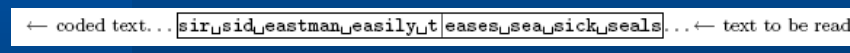
String	Probability	Code
a_1a_1	$0.8 \times 0.8 = 0.64$	0
a_1a_2	$0.8 \times 0.2 = 0.16$	11
a_2a_1	$0.2 \times 0.8 = 0.16$	100
a_2a_2	$0.2 \times 0.2 = 0.04$	101

String	Probability	Code
$a_1a_1a_1$	$0.8 \times 0.8 \times 0.8 = 0.512$	0
$a_1a_1a_2$	$0.8 \times 0.8 \times 0.2 = 0.128$	100
$a_1a_2a_1$	$0.8 \times 0.2 \times 0.8 = 0.128$	101
$a_1a_2a_2$	$0.8 \times 0.2 \times 0.2 = 0.032$	11100
$a_2a_1a_1$	$0.2 \times 0.8 \times 0.8 = 0.128$	110
$a_2a_1a_2$	$0.2 \times 0.8 \times 0.2 = 0.032$	11101
$a_2a_2a_1$	$0.2 \times 0.2 \times 0.8 = 0.032$	11110
$a_2a_2a_2$	$0.2 \times 0.2 \times 0.2 = 0.008$	11111

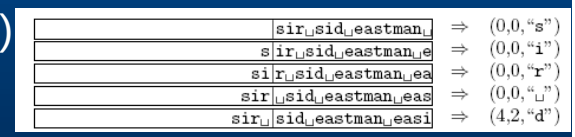
46

Dictionary methods

- LZ77; sliding window
- basis of pkzip, gzip;
- Window cutted in two parts : left is the search buffer and right the look ahead buffer



- e coded (16,3,e)



47

Lempel-Ziv Welch

- LZ78 with dictionnary, basis of compress, gif; improved in 1984 by Welch
- Work on byte
- Dictionnaire évolutif initialised with ASCII codes
- Learning step
- Example:

Symbole lu	Code numérique / binaire		Dictionnaire associé	
			Adresse	Séquence
A	rien			
I	65	01000001	256	AI
D	73	01001001	257	ID
E	68	01000100	258	DE
blanc	69	01000101	259	E blanc
T	32	00100000	260	blanc T
O	84	01010100	261	TO
I	79	01001111	262	OI
blanc	73	01001001	263	I blanc
L	32	01000000	264	blanc L
E	76	01001100	265	LE
blanc	SP	11111111	266	E blanc C
C	259	100000011	267	CI
I	67	001000011	268	IE
E	73	001001001	269	EL
L	69	001000101	270	L blanc
blanc	76	001001100	271	blanc T blanc
T	rien		272	blanc A
blanc	260	100000100	273	AID
A	32	000100000	274	DER
I	rien		275	RA
D	256	100000000		
E	rien			
R	258	100000010		
A	82	001010010		
	65	001000001		

48

Lempel-Ziv Welch

A unique feature of the LZW coding is that the coding dictionary or code book is created while the data are being encoded.
In addition, it is remarkable that an LZW decoder builds an identical decompression dictionary simultaneously the encoded data stream.
The LZW compression algorithm is summarized as follows:

```
w=NIL;
while (read a character k)
{
        if wk exists in the dictionary
                w=wk;
        else
                add wk to the dictionary
                output the code for w;
                w=k;
}
```

49

Lempel-Ziv Welch

The LZW decompression algorithm is as follows:

```
read a character k;
output k;
w=k;
while (read a character k)
/* k could be a character or a code */
{
        entry = dictionary entry for k;
        output entry;
        add w + entry[0] to dictionary;
        w = entry;
}
```

50

II Lossless compression

Lempel-Ziv Welch

Exercise : complete the following dictionary of the LZW coding, and find the encoded message for “ABACABA”.

Index	Entry
1	A
2	B
3	C
4	
5	
6	
7	
8	

51

III Lossless compression

FELICS

- Fast, Efficient, Lossless Image Compression System
- code each pixel with a variable-size code based on the values of two of its previously seen neighbor pixels
- Code depends on P

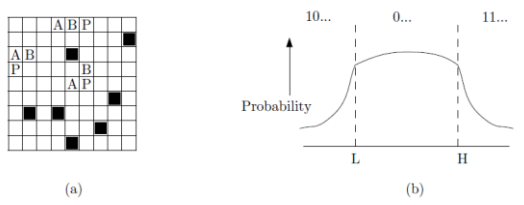


Figure 4.118: (a) The Two Neighbors. (b) The Three Regions.

Pixel P	Region code	Pixel code
L=15	0	0000
16	0	0010
17	0	010
18	0	011
19	0	100
20	0	101
21	0	110
22	0	111
23	0	0001
H=24	0	0011

Table 4.119: The Codes for the Central Region.

52

PPM

- Prediction with partial matching
- Use previous knowledge of the preceding data; order N context

Order 4	Order 3	Order 2	Order 1	Order 0
xyzz→x 2	xyz→z 2	xy→z 2	x→y 3	x 4
yzzx→y 1	yzz→x 2	→x 1	y→z 2	y 3
zzxy→x 1	zzx→y 1	yz→z 2	→x 1	z 4
zxyx→y 1	zxy→x 1	zz→x 2	z→z 2	
xyxy→z 1	xyx→y 1	zx→y 1	→x 2	
yxyz→z 1	yxy→z 1	yx→y 1		

(a)

Order 4	Order 3	Order 2	Order 1	Order 0
xyzz→x 2	xyz→z 2	xy→z 2	x→y 3	x 4
yzzx→y 1	yzz→x 2	xy→x 1	→z 1	y 3
→z 1	zzx→y 1	yz→z 2	y→z 2	z 5
zzxy→x 1	→z 1	zz→x 2	→x 1	
zxyx→y 1	zxy→x 1	zx→y 1	z→z 2	
xyxy→z 1	xyx→y 1	→z 1	→x 2	
yxyz→z 1	yxy→z 1	yx→y 1		

(b)

Table 2.72: (a) Contexts and Counts for "xyzzxyxyzzx". (b) Updated After Another z Is Input.

53

III Lossy compression

- In contrast to lossless coding methods, lossy coding is aimed at a greater data compression ratio by discarding fine details in an image.
- Many lossy encoding techniques can compress images by more than 100:1. If the compression ratio is at 10:1 to 50:1, a restored image can be even indistinguishable from its original.
- Psycho-visuals coding: the eye must not see the difference with the original image
- For example quantify the luminance
- Criteria:

$$EQM = \frac{\sum_{i/j} \left(x_{ij} - \hat{x}_{ij} \right)^2}{\mathrm{MN}}$$

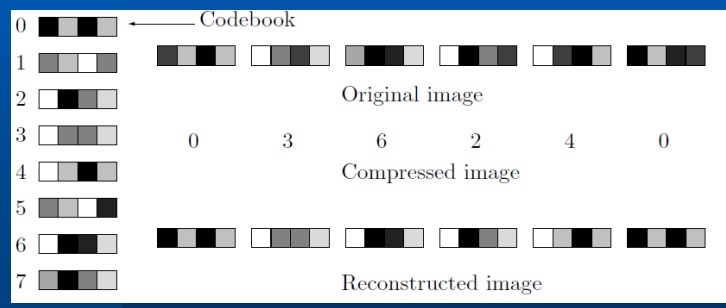
$$PSNR = 10 \log_N \frac{\frac{1}{MN} \sum_{i/j} \left(\hat{x}_{ij} \right)^2}{EQM} \text{ dB}$$

- correct PSNR ≥ 30 dB

54

1) Vector Quantization

- Intuitive methods :
  Subsampling
  Vector Quantization



55

1) Vector Quantization

- Choice of the codebook ; LBG (1980) method
- Iterations :
- Step 0: Select a threshold value e and set k = 0 and $D^{(-1)} = \infty$. Start with an initial codebook with entries $C^{(k)}_i$ (where k is currently zero, but will be incremented in each iteration). Denote the image blocks by Bi (or training vectors : uses them to find the best codebook entries).
- Step 1: Pick up a codebook entry $C^{(k)}_i$. Find all the image blocks Bm that are closer to Ci than to any other Cj . This set (or partition) is denoted by $P^{(k)}_i$. Repeat for all values of i. It may happen that some partitions will be empty.
- Step 2: Select an i and calculate the distortion $D^{(k)}i$ between codebook entry C(k)i and the set of training vectors (partition) $P^{(k)}i$ found for it in Step 1. Repeat for all i, then calculate the average $D^{(k)}$ of all the $D^{(k)}i$. A distortion $D^{(k)}i$ for a certain i is calculated by computing the distances $d(C^{(k)}i, Bm)$ for all the blocks Bm in partition $P^{(k)}i$, then computing the average distance. Alternatively, $D^{(k)}i$ can be set to the minimum of the distances $d(C^{(k)}i, Bm)$.
- Step 3: If $(D^{(k-1)} - D^{(k)})/D^{(k)} \leq e$, halt. The output of the algorithm is the last set of codebook entries $C^{(k)}i$. This set can now be used to (lossy) compress the image with vector quantization. In the first iteration k is zero, so $D^{(k-1)} = D^{(-1)} = \infty > e$. This guarantees that the algorithm will not stop at the first iteration.
- Step 4: Increment k by 1 and calculate new codebook entries $C^{(k)}i$; each equals the average of the image blocks (training vectors) in partition $P^{(k-1)}i$ that was computed in Step 1. (This is how the codebook entries are adapted to the particular image.) Go to Step 1.

1) Vector Quantization

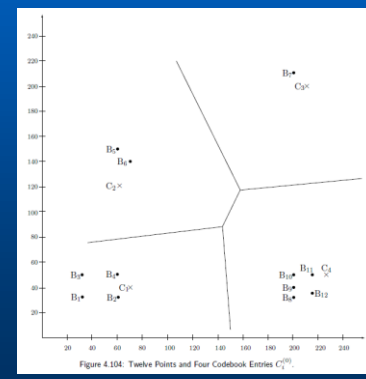
- Example : with 24 pixels

$B_1 = (32,32)$, $B_2 = (60,32)$, $B_3 = (32,50)$, $B_4 = (60,50)$, $B_5 = (60,150)$, $B_6 = (70,140)$, $B_7 = (200,210)$, $B_8 = (200,32)$, $B_9 = (200,40)$, $B_{10} = (200,50)$, $B_{11} = (215,50)$, and $B_{12} = (215,35)$ (Figure 4.104). It is clear that the 12 points are concentrated in four regions. We select an initial codebook with the four entries $C_1^{(0)} = (70,40)$, $C_2^{(0)} = (60,120)$, $C_3^{(0)} = (210,200)$, and $C_4^{(0)} = (225,50)$ (shown as x in the diagram). These entries were selected more or less at random but we show later how the LBG algorithm selects them methodically, one by one. Because of the graphical nature of the data, it is easy to determine the four initial partitions. They are $P_1^{(0)} = (B_1, B_2, B_3, B_4)$, $P_2^{(0)} = (B_5, B_6)$, $P_3^{(0)} = (B_7)$, and $P_4^{(0)} = (B_8, B_9, B_{10}, B_{11}, B_{12})$. Table 4.105 shows how the average distortion $D^{(0)}$ is calculated for the first iteration (we use the Euclidean distance function). The result is

$$\begin{aligned} D^{(0)} =& (1508 + 164 + 1544 + 200 + 900 + 500 \\ &+ 200 + 449 + 725 + 625 + 100 + 325)/12 \\ =& 603.33. \end{aligned}$$

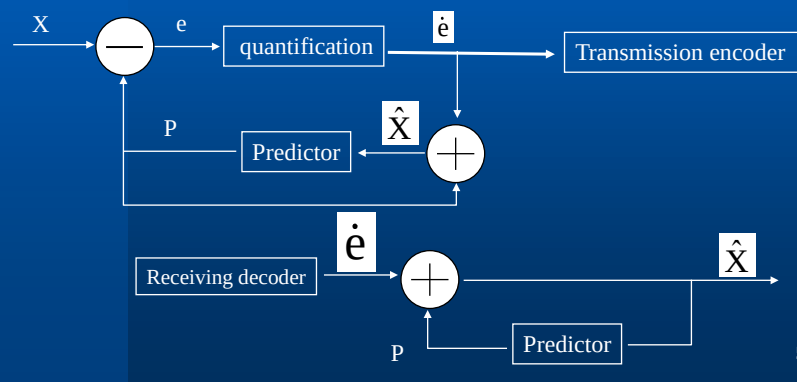
Step 3 indicates no convergence, since $D^{(-1)} = \infty$, so we increment k to 1 and calculate four new codebook entries $C_i^{(1)}$ (rounded to the nearest integer for simplicity)

$$\begin{aligned} C_1^{(1)} &= (B_1 + B_2 + B_3 + B_4)/4 = (46,41), \\ C_2^{(1)} &= (B_5 + B_6)/2 = (65,145), \\ C_3^{(1)} &= B_7 = (200,210), \\ C_4^{(1)} &= (B_8 + B_9 + B_{10} + B_{11} + B_{12})/5 = (206,41). \end{aligned} \quad (4.29)$$



2) Spatial method

- Differential coding : MICD (DPCD)
- Code the difference between the signal and the predicted signal

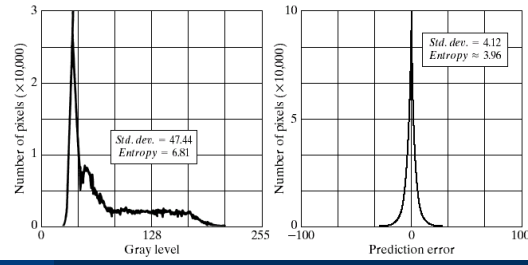


2) Spatial method

a
b c

FIGURE 8.20
(a) The prediction error image resulting from Eq. (8.4-9).
(b) Gray-level histogram of the original image.
(c) Histogram of the prediction error.

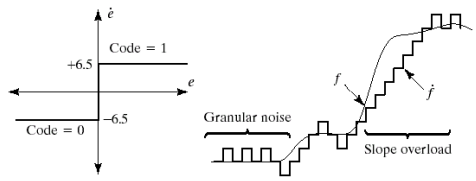
$$\hat{f}(x,y) = round(\alpha f(x,y-1))$$



Note that the dynamic range of the prediction error is smaller than that of the original data

2) Spatial method

Delta modulation



a b
c

FIGURE 8.22 An example of delta modulation.

Input		Encoder				Decoder		Error
n	f	$\hat{f}$	e	$\dot{e}$	$\dot{f}$	$\hat{f}$	$\dot{f}$	$[f-\dot{f}]$
0	14	—	—	—	14.0	—	14.0	0.0
1	15	14.0	1.0	6.5	20.5	14.0	20.5	–5.5
2	14	20.5	–6.5	–6.5	14.0	20.5	14.0	0.0
3	15	14.0	1.0	6.5	20.5	14.0	20.5	–5.5
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.
14	29	20.5	8.5	6.5	27.0	20.5	27.0	2.0
15	37	27.0	10.0	6.5	33.5	27.0	33.5	3.5
16	47	33.5	13.5	6.5	40.0	33.5	40.0	7.0
17	62	40.0	22.0	6.5	46.5	40.0	46.5	15.5
18	75	46.5	28.5	6.5	53.0	46.5	53.0	22.0
19	77	53.0	24.0	6.5	59.6	53.0	59.6	17.5
.	.	.	.	.	.	.	.	.
.	.	.	.	.	.	.	.	.

$$\hat{f} = \alpha \hat{X}$$

$$and \quad \dot{e} = \begin{cases} \xi & \text{for e>0} \\ -\xi \end{cases}$$

III Lossy compression

2) Spatial method

- Differential Pulse Coded Modulation DPCM: a staircase-like quantization function is employed (see next slide).
- In addition, the prediction is performed by a linear combination of m previous pixels (Eqs. (8.5-16) to (8.5-19))

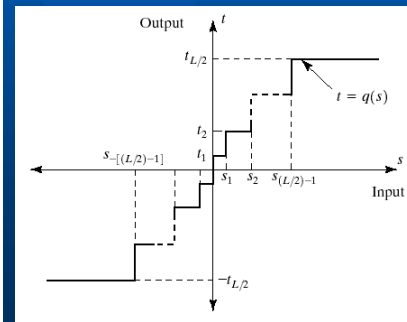


FIGURE 8.25 A typical quantization function.

Slowly changing regions are finely quantized, while the rapidly changing areas are quantized more coarsely.

III Lossy compression

2) Spatial method

- Prediction Errors in Four Cases

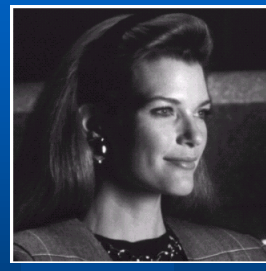
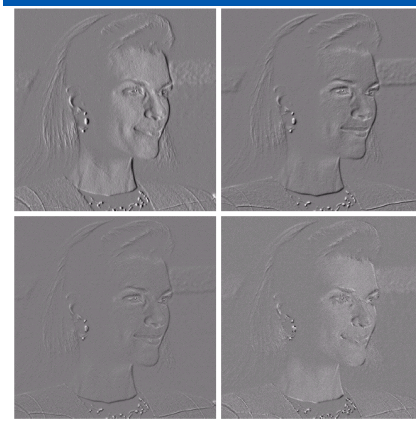


FIGURE 8.23 A 512 × 512 8-bit monochrome image.

a b
c d

FIGURE 8.24 A comparison of four linear prediction techniques.

Prediction error



$$\hat{f}(x,y) = 0.97\, f(x,y-1)$$

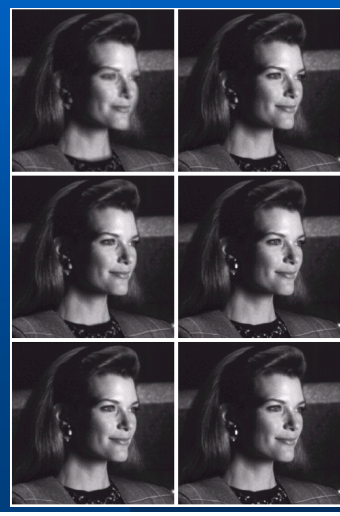
$$\hat{f}(x,y) = 0.5\, f(x,y-1) + 0.5\, f(x-1,y)$$

$$\hat{f}(x,y) = 0.75\, f(x,y-1) + 0.75\, f(x-1,y) - 0.5\, f(x-1,y-1)$$

$$\hat{f}(x,y) = \begin{cases} 0.97\, f(x,y-1) & \text{if } \Delta h \le \Delta v \\ 0.97\, f(x-1,y) & \text{otherwise} \end{cases}$$

2) Spatial method

- Reconstruction errors of DPCM



2) Spatial method

- The table below lists the RMS errors of the difference images, as well as for a number of other combinations of predictor and quantizers.

$$\int_{s_{i-1}}^{s_i} (s - t_i)\, p(s)\, ds = 0 \quad i = 1, 2, \ldots, L/2$$

$$s_i = \begin{cases} 0 & i = 0 \\ (t_i - t_{i+1})/2 & i = 1, 2, \ldots, L/2 - 1 \\ \infty & i = L/2 \end{cases}$$

Predictor	Lloyd-Max Quantizer			Adaptive Quantizer		
	2-level	4-level	8-level	2-level	4-level	8-level
Eq. (8.5-16)	30.88	6.86	4.08	7.49	3.22	1.55
Eq. (8.5-17)	14.59	6.94	4.09	7.53	2.49	1.12
Eq. (8.5-18)	9.90	4.30	2.31	4.61	1.70	0.76
Eq. (8.5-19)	38.18	9.25	3.36	11.46	2.56	1.14
Compression	8.00:1	4.00:1	2.70:1	7.11:1	3.77:1	2.56:1

TABLE 8.11 Lossy DPCM root-mean-square error summary.

2 a) Roberts Method

- Diminish the size of the code (classically 8 bits per pixel)
- 1st step: X1=X + random noise b
- 2nd step: reduction of the number of bits
- After construction: subtract a random noise with same dynamic as the signal
- Low rate 8/3 before Huffmann coding

3) Block Coding

- Low modification of luminance between adjacent pixels
- Blocks of 4x4 or 8x8
- Parameters :

mean

$$\bar{X} = \frac{1}{N}\sum_{i=1}^{N} X_i$$

mean shift

$$\varepsilon = \frac{1}{N}\sum_{i=1}^{N} \left| X_i - \bar{X} \right|$$

threshold

$$X_i \geq \bar{X} \Rightarrow p = 1$$
$$X_i < \bar{X} \Rightarrow p = 0$$

- Example: compression

119	121	115	90		1	1	1	0
110	100	88	89	$\bar{X} = 94$	1	1	0	0
121	77	80	83	$\varepsilon = 16$	1	0	0	0
70	75	81	79		0	0	0	0

- Reconstruction from 2 states:

$$Y_1 = \bar{X} + \frac{n\varepsilon}{2n_1} \qquad Y_0 = \bar{X} - \frac{n\varepsilon}{2n_0}$$

115	115	115	81
115	115	115	81
115	81	81	81
81	81	81	81

4) Method with transform

- Image transform ; linear, orthogonal
- To decorrelate input datas (ex RGB colors)

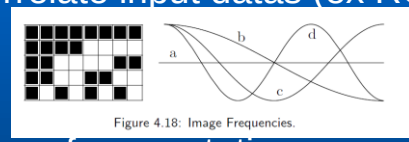
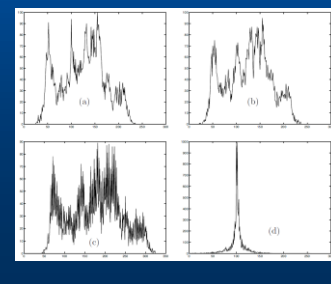


Figure 4.18: Image Frequencies.

- Linear transform : rotations , consider each pair of pixels as a point



67

III Lossy compression

4) Method with transform

- Classical transforms :
- 1. The Walsh-Hadamard transform (WHT, Section 4.5.2) is fast and easy to compute (it requires only additions and subtractions), but its performance, in terms of energy compaction, is lower than that of the DCT.
- 2. The Haar transform [Stollnitz et al. 96] is a simple, fast transform. It is the simplest wavelet transform.
- 3. The Karhunen-Loève transform (KLT) is the best one theoretically, in the sense of energy compaction (or, equivalently, pixel decorrelation). However, its coefficients are not fixed; they depend on the data to be compressed. Calculating these coefficients (the basis of the transform) is slow, as is the calculation of the transformed values themselves. Since the coefficients are data dependent, they have to be included in the compressed stream. For these reasons and because the DCT performs almost as well, the KLT is not generally used in practice.
- 4. The discrete cosine transform (DCT). This important transform is almost as efficient as the KLT in terms of energy compaction, but it uses a fixed basis, independent of the data. There are also fast methods for calculating the DCT. This method is used by JPEG and MPEG audio.

4) Method with transform

- Walsh-Hadamard transform
- Given an *N*×*N* block of pixels *Pxy* (where *N* must be a power of 2, *N* = 2*n*), its two-dimensional WHT and inverse WHT are defined by Equations

$$H(u,v) = \sum_{x=0}^{N-1}\sum_{y=0}^{N-1} p_{xy} g(x,y,u,v)$$
$$= \frac{1}{N}\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} p_{xy}(-1)^{\sum_{i=0}^{n-1}[b_i(x)p_i(u)+b_i(y)p_i(v)]},$$
$$P_{xy} = \sum_{u=0}^{N-1}\sum_{v=0}^{N-1} H(u,v)h(x,y,u,v)$$
$$= \frac{1}{N}\sum_{u=0}^{N-1}\sum_{v=0}^{N-1} H(u,v)(-1)^{\sum_{i=0}^{n-1}[b_i(x)p_i(u)+b_i(y)p_i(v)]},$$

$$p_0(u) = b_{n-1}(u),$$
$$p_1(u) = b_{n-1}(u) + b_{n-2}(u),$$
$$p_2(u) = b_{n-2}(u) + b_{n-3}(u),$$
$$\vdots$$
$$p_{n-1}(u) = b_1(u) + b_0(u).$$

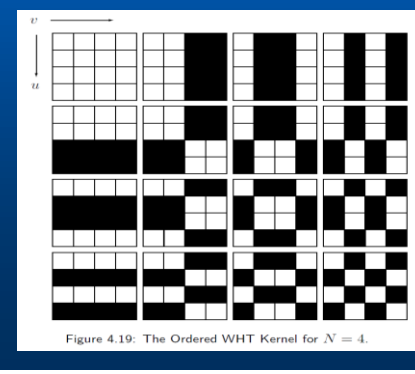


Figure 4.19: The Ordered WHT Kernel for $N = 4$.

III Lossy compression

4) Method with transform

- Haar transform ; first wavelet

$$h_0(x) \stackrel{\text{def}}{=} h_{00}(x) = \frac{1}{\sqrt{N}}, \quad \text{for } 0 \le x \le 1,$$

$$h_k(x) \stackrel{\text{def}}{=} h_{pq}(x) = \frac{1}{\sqrt{N}} \begin{cases} 2^{p/2}, & \frac{q-1}{2^p} \le x < \frac{q-1/2}{2^p}, \\ -2^{p/2}, & \frac{q-1/2}{2^p} \le x < \frac{q}{2^p}, \\ 0, & \text{otherwise for } x \in [0,1]. \end{cases}$$

Matrix example for N=2

$$\mathbf{A}_2 = \begin{pmatrix} h_0(0/2) & h_0(1/2) \\ h_1(0/2) & h_1(1/2) \end{pmatrix} = \frac{1}{\sqrt{2}} \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}$$

Given an image block **P** of order *N*×*N* where *N* = 2*n*, its Haar transform is the matrix product **A***N***PA***N*


Figure 4.20: The Basis Images of the Haar Transform for $n = 8$.

III Lossy compression

- Karhunen-Loève (KLT); PCA based approach
- Given an image, we break it up into *k* blocks of *n* pixels each, where *n* is typically 64 but can have other values, and *k* depends on the image size.

vectors $v^{(i)} = \mathrm{b}^{(i)} - \overline{\mathrm{b}}$ $\qquad w^{(i)} = A\mathrm{v}^{(i)}$

nxk matrix $V = (v^{(1)} \quad \ldots \quad v^{(k)})$ and $W = (w^{(1)} \quad \ldots \quad w^{(k)})$

The *n* coefficient vectors **c**(*j*) of the Karhunen-Loève transform are given by

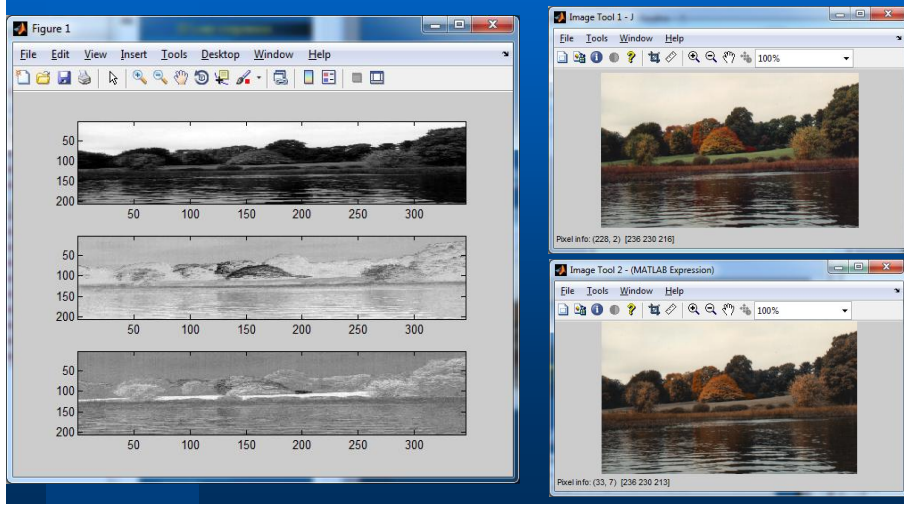
$$c^{(j)} = (w^{(1)}{}_j \quad \ldots \quad w^{(k)}{}_j) \ , \ \mathrm{j}=1,2,\ldots,\mathrm{n}$$

Matrix **A:** its rows are the basis vectors of the KLT, and the energies (variances) of the transformed vectors are the eigenvalues $\lambda 1, \lambda 2, \ldots, \lambda n$ of $\mathbf{V}\cdot\mathbf{V}^T$.

$$\mathbf{W}\cdot\mathbf{W}^T = \mathbf{A}(\mathbf{V}\cdot\mathbf{V}^T)\mathbf{A}^T = \begin{pmatrix} \lambda_1 & 0 & 0 & \cdots & 0 \\ 0 & \lambda_2 & 0 & \cdots & 0 \\ 0 & 0 & \lambda_3 & \cdots & 0 \\ \vdots & \vdots & & \vdots & \vdots \\ 0 & 0 & \cdots & 0 & \lambda_n \end{pmatrix}$$

III Lossy compression

- Karhunen-Loève (KLT), example



4) Method with transform

- DCT

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

$$C(i) \text{ ou } C(j) = \begin{cases} \frac{1}{\sqrt{2}} & \text{if } i = 0 \\ 1 \end{cases}$$

- Example of 1D data : p = (12, 10, 8, 10, 12, 10, 8, 11)
- DCT : 28.6375, 0.571202, 0.46194, 1.757, 3.18198, −1.72956, 0.191342, −0.308709
- Quantization 1: 28.6, 0.6, 0.5, 1.8, 3.2,−1.8, 0.2,−0.3 ; IDCT : 12.0254, 10.0233, 7.96054, 9.93097, 12.0164, 9.99321, 7.94354, 10.9989
- Quantization 2: 28, 0, 0, 2, 3,−2, 0, 0 ; IDCT : 11.236, 9.62443, 7.66286, 9.57302, 12.3471, 10.0146, 8.05304, 10.6842
  Max error 0.764 (or 6.4% of 12).

Transform → Quantization → Re-ordering of coefficients → Statistic Coding

4) Method with transform

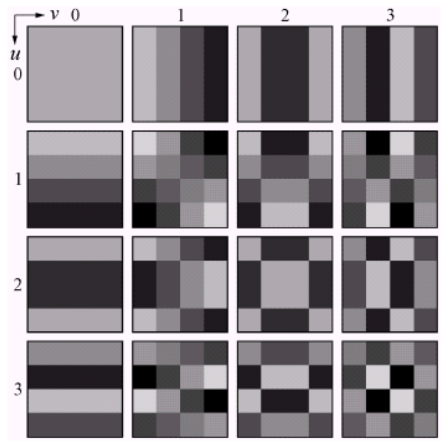
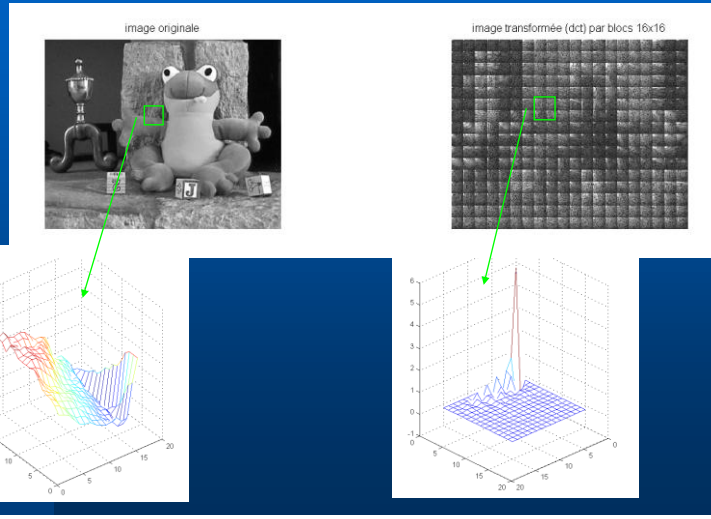


FIGURE 8.30 Discrete-cosine basis functions for $N = 4$. The origin of each block is at its top left.

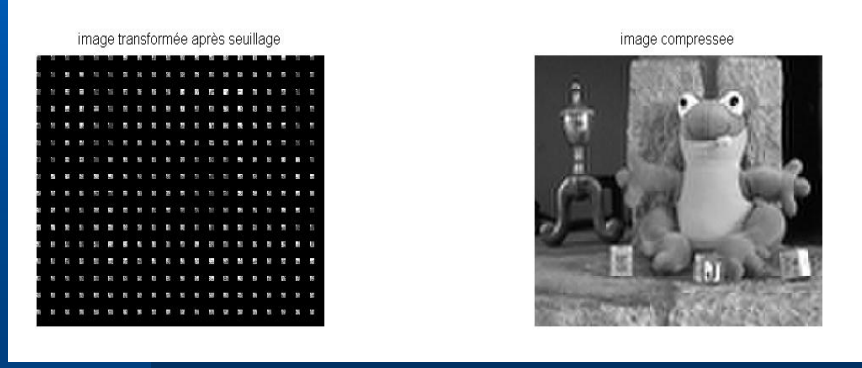
4) Method with transform

- Example of DCT over 16x16 blocks



4) Method with transform

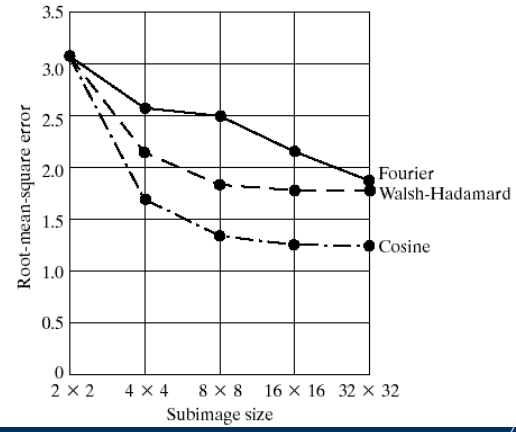
- Compression by low pass filter on each block



4) Method with transform

- Effect of block size

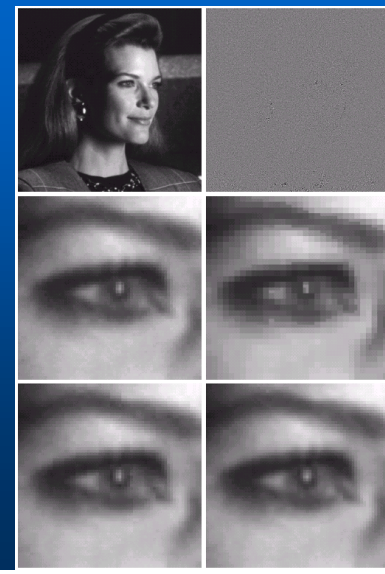
FIGURE 8.33 Reconstruction error versus subimage size.



77

4) Method with transform

- Effect of block size



a b
c d
e f

FIGURE 8.34 Approximations of Fig. 8.23 using 25% of the DCT coefficients: (a) and (b) 8 × 8 subimage results; (c) zoomed original; (d) 2 × 2 result; (e) 4 × 4 result; and (f) 8 × 8 result.

IV JPEG norm

- Joint Photographic Expert group; 1991
- Describe the scheme; freedom on implementation
- 2 classes of process: lossless or lossy
- 4 modes : sequential DCT ,
  progressive DCT (successive approx.),
  lossless or hierarchic
  (approx. on several resolution)
- Samples of 8 ou 12 bits
- Color Images treated in Y,Cr,Cb
  (color downsampled)

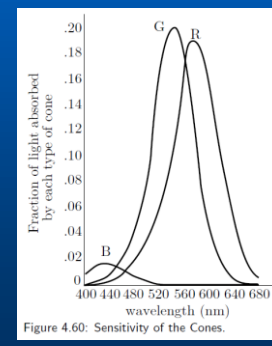
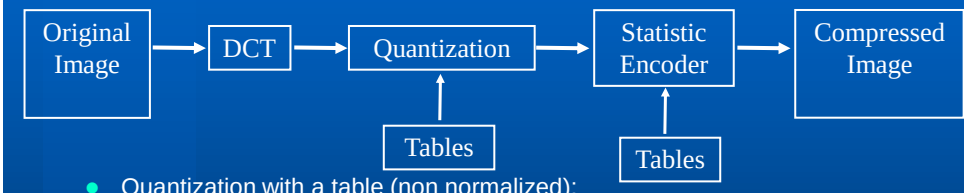


Figure 4.60: Sensitivity of the Cones.

79

IV JPEG Norm

- Scheme



- Quantization with a table (non normalized):
  example of generation of Q

$$Z = E\left(\frac{F(u,v)}{Q(u,v)}\right) \qquad Q(u,v) = 1 + \left(1 + \alpha\left(i^n + j^n\right)\right)F_q$$

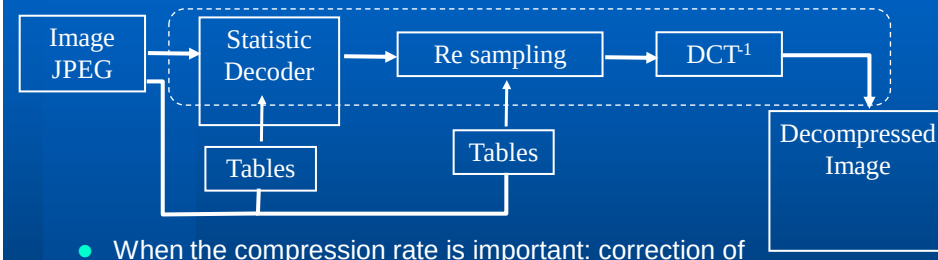
  with "quality factor " Fq; if Fq augment => image degraded

- Zig-zag way

80

IV JPEG Norm

- Restoration with lossy process



- When the compression rate is important: correction of discontinuities of the image by prediction of the first low frequency coefficient of the DCT matrix with the 9 neigbouring DCT blocks (models with quadratic polynomials)

$$blocs 8\times 8 \begin{bmatrix} DC_5 & AC_1 & AC_5 \\ AC_2 & AC_4 & . \\ AC_3 & . & . \end{bmatrix} \quad matrice \begin{bmatrix} DC_1 & DC_2 & DC_3 \\ DC_4 & DC_5 & DC_6 \\ DC_7 & DC_8 & DC_9 \end{bmatrix}$$

81

IV JPEG Norm

- Progressive mode
- higher-frequency DCT coefficients are written on the compressed stream in blocks called “scans.”
- Each scan that is read and processed by the decoder results in a sharper image. The idea is to use the first few scans to quickly create a low-quality, blurred preview of the image, and then either input the remaining scans or stop the process and reject the image.
- Trade-off : the encoder has to save all the coefficients of all the data units in a memory buffer before they are sent in scans, and also go through all the steps for each scan, slowing down the progressive mode

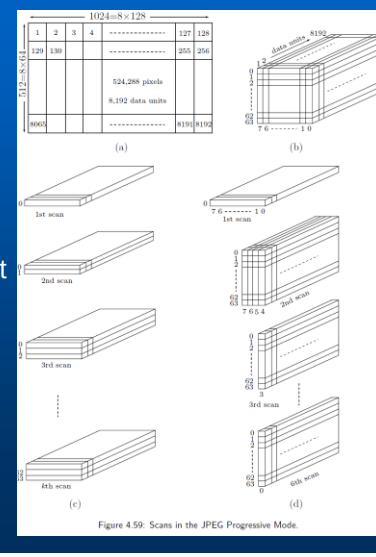


Figure 4.59: Scans in the JPEG Progressive Mode.

IV JPEG Norm

- Example
- Frequency encoding
- Colorshades
- Layers

IV JPEG Norm

- Progressive mode
- higher-frequency DCT coefficients are written on the compressed stream in blocks called “scans.”
- Each scan that is read and processed by the decoder results in a sharper image. The idea is to use the first few scans to quickly create a low-quality, blurred preview of the image, and then either input the remaining scans or stop the process and reject the image.
- Trade-off : the encoder has to save all the coefficients of all the data units in a memory buffer before they are sent in scans, and also go through all the steps for each scan, slowing down the progressive mode

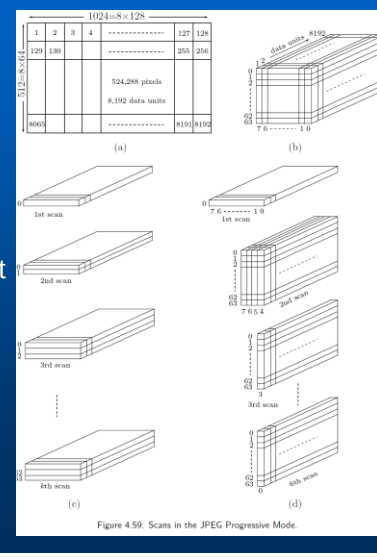
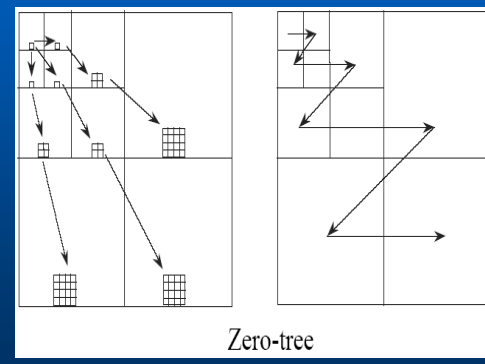


Figure 4.59: Scans in the JPEG Progressive Mode.

JPEG Norm , JPEG2000

- Taubann, EBCOT
- wavelet transform



85

JPEG Norm , JPEG2000

- Wanted functionalities:
  - Gray and binary coding, lossless or lossy
  - Progressive mode by resolution/quality
  - ROI access
  - Error robustness, security
- Technical elements:
  - Tiles partitioning of the image and coding by bit planes (code blocks)

86

V MPEG Norm

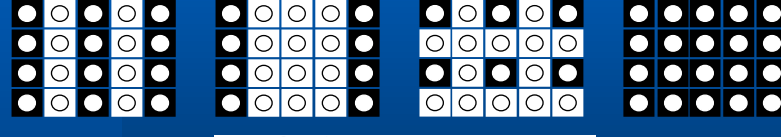
- MPEG1 since 1992; principles of JPEG but exploit the temporal redundancies .
  Low resolution images ; flow of 1,5 Mbits/s with sound => VHS quality on CDRom (VideoCD)
- MPEG2 since 1994: higher image quality (3 à 300 Mbits/s)
  Broadcasting and DVD
- MPEG4 objects coding
- AVI Files : container => use the right coder/decoder

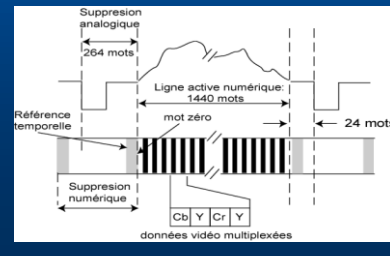
87

V MPEG Norm

- Color space : Y Cr Cb
- Sampling frequency : CCIR 601 norm ; 13.75 MHz, "4", for Y and 6.75 MHz for Cr and Cb , "2"

4:2:2 4:1:1 4:2:0 4:4:4

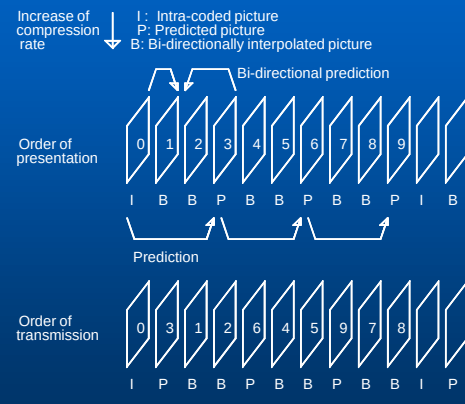




88

V MPEG Norm :MPEG1

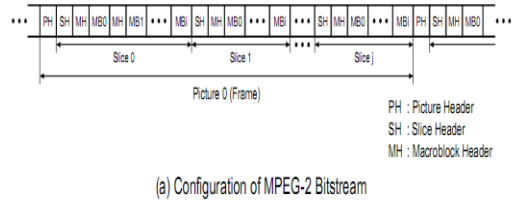
- Source Intermediate Format: suppress data before compression: one frame over 2 and one point over 2 in each line => 320 pixels x288 at 25 images Hz (code 2:1:0); 31,5 MHz before compression
- Groups of image:
  Intra JPEG coded, Predicted (from image I or P), Bidirectional (movement vectors between images I or P neighboring)



89

V MPEG Norm :MPEG1

- Bitstream



(a) Configuration of MPEG-2 Bitstream

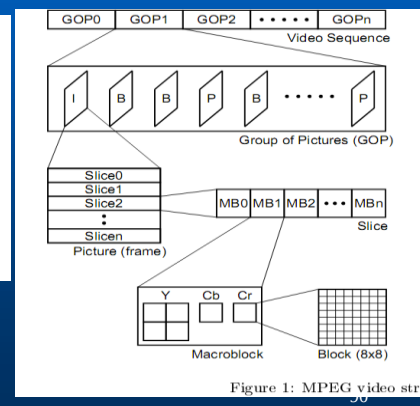
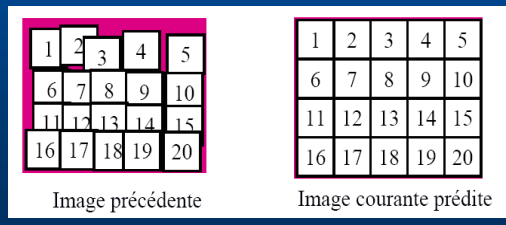


Figure 1: MPEG video stre

V MPEG Norm : MPEG1

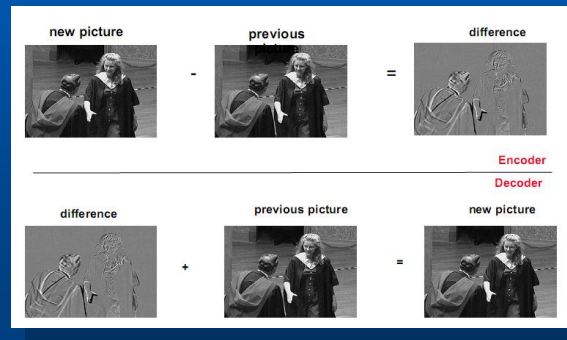
- Movement estimator: predict the displacements
- work on macroblocks: 6 blocks (4 of luminance 2 of chrominance)
- Work on a lookup zone
- Keeping of movement vectors (<> of spatial position of the mblocks between 2 images)



91

V MPEG Norm : MPEG1

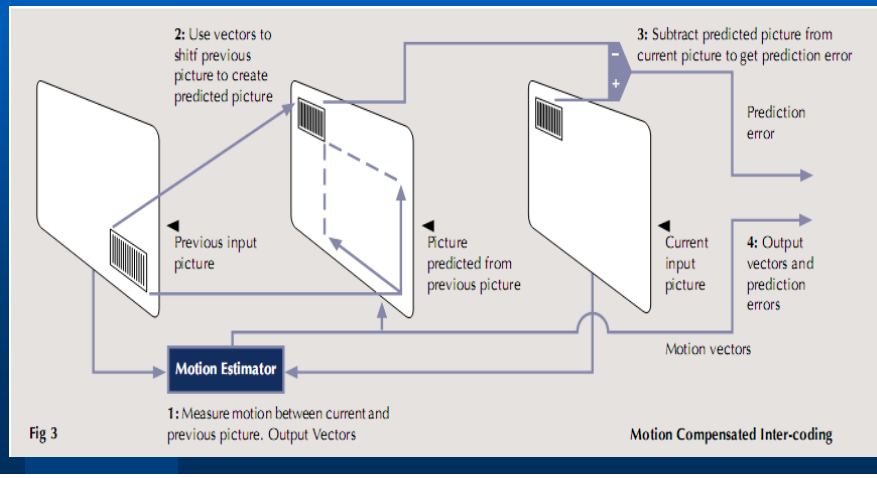
- Predicted Pictures
- The difference between the current picture and the previous one.



92

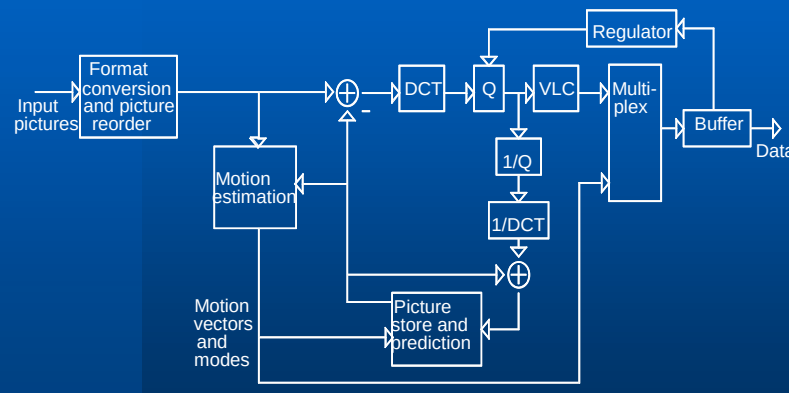
V MPEG Norm : MPEG1

- Prediction error block



V MPEG Norm : MPEG1

- Possible Scheme of encoder



94

V MPEG Norm : MPEG2

- Support higher resolution
- Processing of interlaced (50 frames /s) and progressive scan => complexity of movement detection
- hierarchic Coding for transmitting at different quality levels
- Compatibility with MPEG1
- 6 profiles for broadcasting
- 4 levels corresponding to the maximum image resolution

95

V MPEG Norm : MPEG2

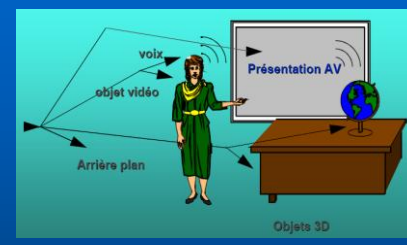
- Profiles :

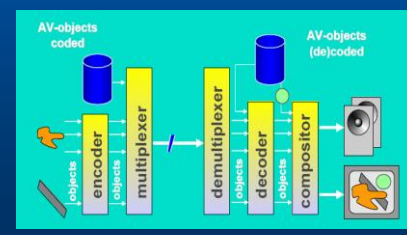
Profil Niveau	Simple	Main	4:2:2	SNR	Spatial	High
High		4:2:0 1920x1152 80 Mbits/s I,P,B	4:2:0, 4:2:2 1920x1152 300 Mbits/s I,P,B			4:2:0, 4:2:2 1920x1152 15 Mbits/s I,P,B
High 1440		4:2:0 1440x1152 60 Mbits/s I,P			4:2:0, 4:2:2 1440x1152 60 Mbits/s I,P,B	4:2:0, 4:2:2 1440x1152 80 Mbits/s I,P,B
Main	4:2:0 720x576 15 Mbits/s I,P	4:2:0 720x576 15 Mbits/s I,P	4:2:0, 4:2:2 720x608 50 Mbits/s I,P,B	4:2:0, 4:2:2 720x576 15 Mbits/s I,P,B		4:2:0, 4:2:2 720x576 20 Mbits/s I,P,B
Low		4:2:0 352x288 4 Mbits/s I,P,B		4:2:0, 4:2:2 352x288 15 Mbits/s I,P,B		

96

V MPEG Norm : MPEG4

- Objects decomposition of the scene
- Natural Video Processing tools, 2D and 3D compression
- Synthesized objects texture statistic and dynamic coding





97

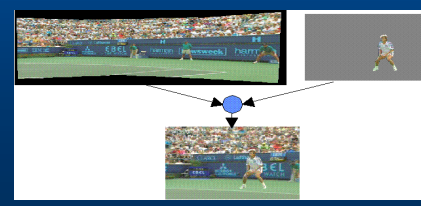
V MPEG Norm : MPEG4

- Large applications :
  Real time communication
  Video surveillance
  Multimedia mobile phone
  Storage and content based search
  Streaming without uploading all the source
  Simultaneous scene visualization (vision conference)
  Data transmission (all types of data : video, audio,…)
  Post production (movies, tv)
  DVD
  Face tracking
  Hierarchy and audio objects in a scene
- Interactivity
- DivX/Xvid : based on the vidéo part of MPEG4

98

V MPEG Norm : MPEG4

- Example:
  - Isolate sequence background and rebuild a panoramic view (movement estimation and compensation with de 8 or 16 pixels blocks).
  - Extract the moving people
  - Transfer one time the background then the player
  - The decoder rebuilds the scene thanks to:
    - The camera parameters for the background
    - The player data



99

H264

- May 2003 by ITU; New : the filter
- 3 profiles
- 16x16 to 4x4 block coding
- T : DCT and Hadamar on DC coefs.

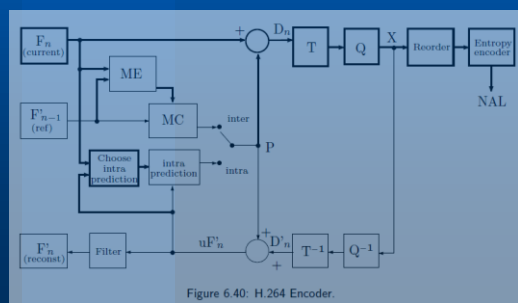


Figure 6.40: H.264 Encoder.

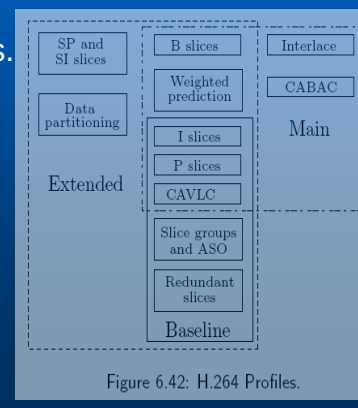


Figure 6.42: H.264 Profiles.

references

- Digital Image Processing : R.Gonzales, E.Woods
- Data compression, the complete reference , David Salomon, Springer
- D.S. Taubman et M.W. Marcellin, "JPEG2000 Image compression fundamentals, standards and practice", Kluwer Academic Publishers (2001).