Digital Image Processing

18 January 2007

Dr. ir. Aleksandra Pizurica
Prof. Dr. Ir. Wilfried Philips

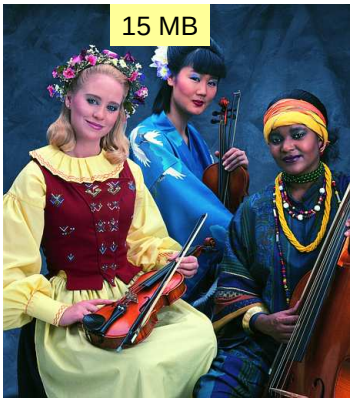
Aleksandra.Pizurica @telin.UGent.be
Tel: 09/264.3415

UNIVERSITEIT
GENT
Telecommunicatie en Informatieverwerking

INSTITUUT VOOR HERKENNING VORMING

Introduction to image compression

Need for compression: “Pre-press” images



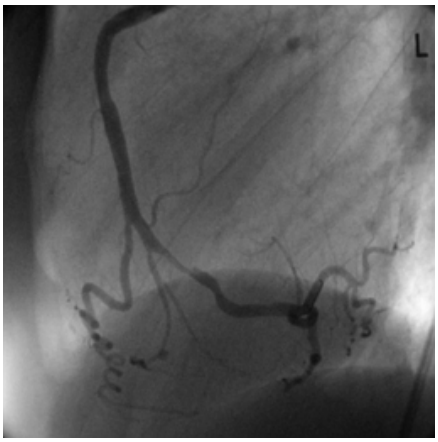
CMYK (4 bytes per pixel) instead of RGB

Combination of photographs, text and lines

Very high quality required

Need for compression: medical images

Angiography: 20 s ⇒ 120 MB



radiography 2048x1680x10 (4 Mbyte)

Big amounts of data, because of

- High spatial resolution (e.g., radiography)
- 3D-acquisition: volumetric data (e.g., MRI) or video (e.g., angiography)

Other applications

Other applications are less critical: much more errors can be tolerated

Introduction to image compression…

Image compression → compression of image data

In general, data compression means reducing the amount of data that represents a given quantity of information

Two main categories:

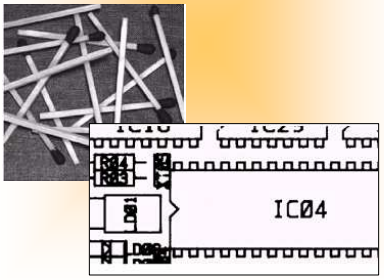
- **Lossless** compression
  - → After decompression the image is exactly the same as the original image
  - → Compression factors are small (2 to 5)
- **Lossy** compression
  - → The decompressed image is an approximation of the the original image
  - → Compression factor typically big and increasing with the decrease of the required quality

… Introduction to image compression…

Image compression makes use of spatial and psychovisual redundancies

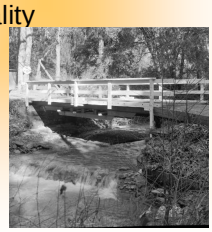
Spatial (inter-pixel) redundancy

In most images, the value of any pixel can be reasonably well predicted from its neighbors. (At which positions in the image this does not hold or holds less well?)



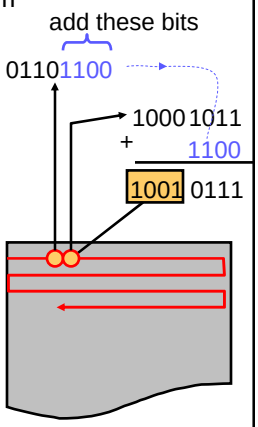
Psychovisual redundancy

Human eye is less sensitive to certain information (e.g., very high spatial frequencies in highly textured or in dark areas). Such information is psychovisually redundant and can be removed without significantly damaging the perceptual quality



… Introduction to image compression

Example: improved grey scale (IGS) quantization - takes into account the characteristics of the human visual system



Principle: add the four least significant bits of a neighbouring (previous) pixel and then quantize (to four most significant bits)

Remarks

In the reminder we treat grey scale image coding/compression

Lossy coding of color images

- The image is usually first transformed to YUV space
- The U and V images are then spatially subsampled: 3/4 of the pixels are removed (e.g., pixels from all even rows and columns are omitted)
- The Y, U and V images are then separately encoded

Lossless coding of color images

- Usually by separately encoding the R, G and B channels
- Joint coding performs better

Principles of data compression

Variable length coding

	fixed length	variable length
A	000	11110
B	001	1101
C	010	101
D	011	0
E	100	100
F	101	1100
G	110	1110
H	111	11111

bits/symbol:

fixed length: $l = \lceil \log_2(N) \rceil$

variable length: $\bar{l} = \sum_{i=1}^{N} p_i l_i$

A D F D G D D D D C D E

000011101011110011011011011010011100

111100110001110000001010100

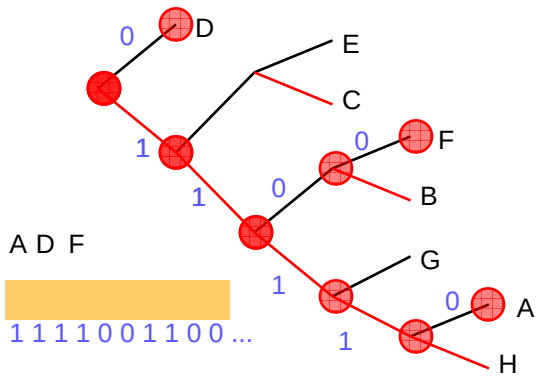
Denote p_i probability of occurrence of a simbol i

Principle: encode frequent symbols with a short bit string (small l_i) and rarely appearing symbols with a long bit string

The code has to be decodable
- Every codeword has to be unique
- The end of a codeword has to be recognizable in the coded bit sequence

Prefix-code

	output bits
A	11110
B	1101
C	101
D	0
E	100
F	1100
G	1110
H	11111



Prefix code: no code word is at the beginning of an another code word

$\Rightarrow$ The decoder can always recognize the end of a code word

Such a code can be represented by leaves of a decoder tree

Measuring information

A random event A that occurs with probability P(A) has

$$I(A) = \log \frac{1}{P(A)} = -\log P(A) \qquad (*)$$

units of information

The base of the logarithm in this equation determines the unit used to measure the information.

→ If the base 2 is selected the resulting unit is bit

Equation (*) says: less probable events carry more information than the more probable ones. Examples?

Entropy

Stationary source → Symbols with possible values α_i, i=1…N, and probability $P(\alpha_i)=p_i$

Entropy of the source $H = -\sum_{k=1}^{N} p_k \log_2 p_k$ bits remark: $0\log_2 0 = 0$

H the smallest possible mean value of the code word length

- For any technique that encodes symbol per symbol
- And even for any coding technique if the successive symbols are statistically independent (and have the same distribution)

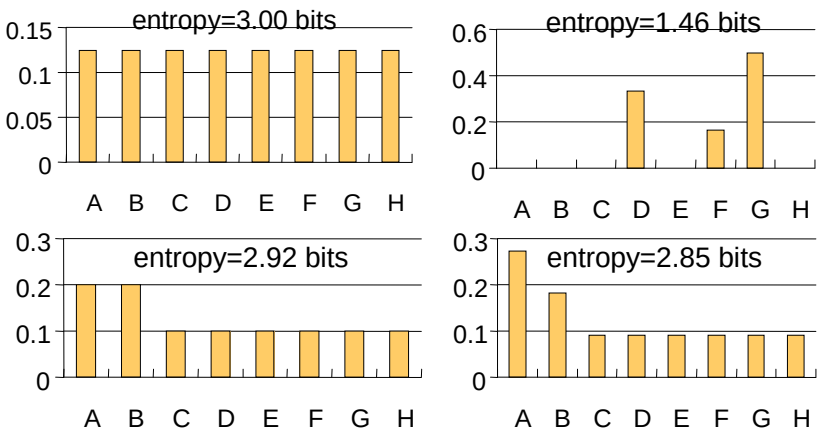
0 bit $\leq H \leq \log_2 N$ bit, N - number of symbols of the alphabet

H is maximal when all the symbols are equally probable: $H_{\max} = \log_2 N$

$H \leq \log_2 M$ where M is the number of symbols with $p_k \neq 0$:

- logical: symbols that never appear (p_k=0) can be deleted from the alphabet

Entropy: Examples



Huffman coding

Stationary source → Symbols with possible values α_i, i=1…N, and probability $P(\alpha_i)=p_i$

The Huffman code is a specific pre-fix code

- the code word length $l_k \approx -\log_2 p_k$ $\quad \Rightarrow \bar{l} = \sum_{k=1}^{N} p_k l_k \approx H$

$\Rightarrow$ approximately optimal among the codes that encode each symbol individually

$\Rightarrow$ and also approximately optimal among all the codes if the source generates statistically independent symbols

- Mean code word length is between H and $H+1$

$\Rightarrow$ far from optimal if H is small (e.g., for $H \leq 1$), in particular, not well suited for binary images

Huffman coding: example…

Original source		Source reduction			
Symbol	Probability	1	2	3	4
a_2	0.4	0.4	0.4	0.4	0.6
a_6	0.3	0.3	0.3	0.3	0.4
a_1	0.1	0.1	0.2	0.3	
a_4	0.1	0.1	0.1		
a_3	0.06	0.1			
a_5	0.04				

…Huffman coding: example

Original source			Source reduction							
Sym.	Prob.	Code	1		2		3		4	
a_2	0.4	1	0.4	1	0.4	1	0.4	1	0.6	0
a_6	0.3	00	0.3	00	0.3	00	0.3	00	0.4	1
a_1	0.1	011	0.1	011	0.2	010	0.3	01		
a_4	0.1	0100	0.1	0100	0.1	011				
a_3	0.06	01010	0.1	0101						
a_5	0.04	01011								

Huffman coding for text

The Huffman coding makes use of

- the fact that some letters are more frequent than others (first-order statistics)
- but not that some combinations are more frequent than others (second or higher order statistics)

Example: a Huffman code for English language

- exploits the fact that the letter E is more frequent than the letter Q
- But not that the combination TH is more frequent than XP

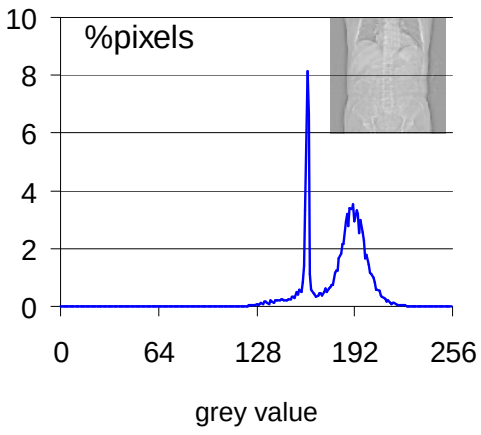
Higher order redundancy can be exploited by treating two or more successive letters as ONE "meta" letter (block code)

Example: English text

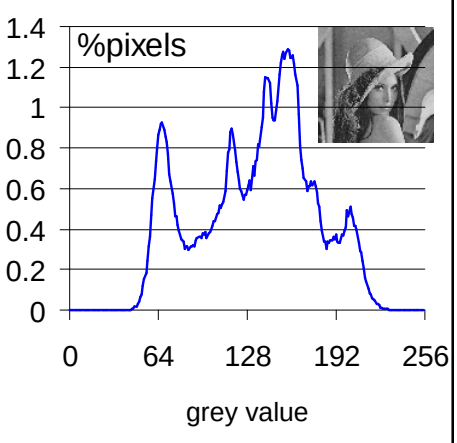
- Huffman code per letter $\Rightarrow$ 4.03 bit/letter
- Block-Huffman code per two letters $\Rightarrow$ 3.32 bit/letter
- Using all redundancy $\Rightarrow$ 0.6-1.3 bit/letter

Entropy coding of image pixels

"CT", entropy=5.7 bit/pixel



"Lenna", entropy=7.2 bit/pixel



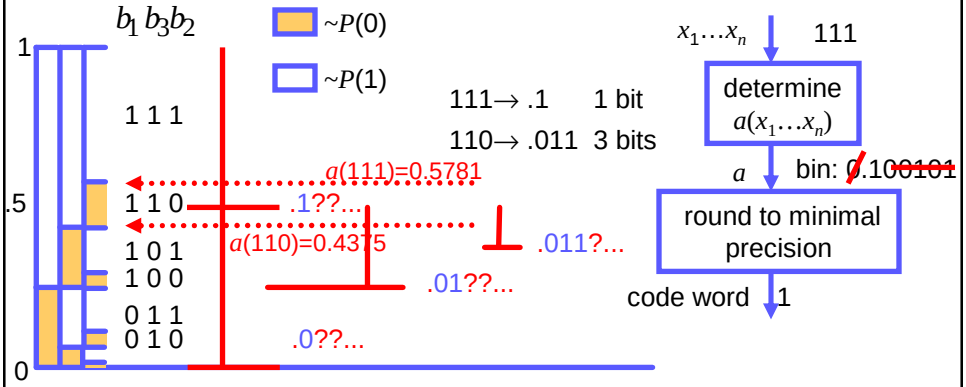
Remarks

Huffman coding
- •is not efficient for image pixels because the first-order entropy of image pixels is too big
- •Is useful in combination with transform coding

Code book has to be made available to the decoder as well
- •fixed code book, chosen once for ever
- •saving the code book in a compressed file
- •Starting from a "standard" code book that is continuously adapted based on the decoded data ⇒ "adaptive" operation

Huffman coding cannot compress 1-bit symbols (A=0, B=1)
- •The block Huffman coding offers a solution for this
- •A better solution is arithmetic coder:
  - →Encodes arbitrarily long sequence of symbols (usually bits) as one code word that is computed from the input sequence

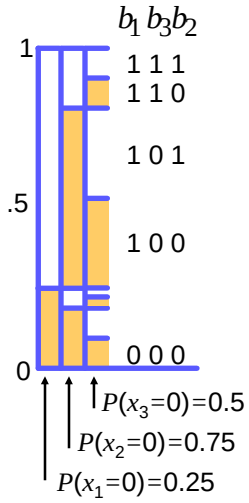
Arithmetic coding

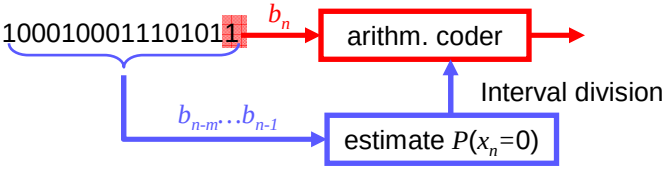


The arithmetic coder associates with every possible bit string $x_1 \ldots x_n$ a subinterval $[a(x_1 \ldots x_n), b(x_1 \ldots x_n)]$ of [0,1] with the length $P(x_1 \ldots x_n)$

The code word for $x_1 \ldots x_n$ is a binary floating point representation of $a(x_1 \ldots x_n)$, but rounded up to a minimal number of bits that are needed to distinguish $a(x_1 \ldots x_n)$ from other possible $a(x'_1 \ldots x'_n)$

Adaptive arithmetic coders





The arithmetic (de)coder continues to work when the interval distribution is adapted from bit to bit

$\Rightarrow$ "adaptive" operation possible: if $P(x_n=0)$ changes in function of n (not-stationary source) the coder can adapt itself $\Rightarrow$ beter compression

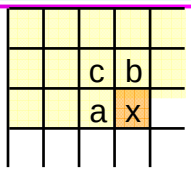
The decoder has to be able to compute itself the division of the intervals; that can be done based on the "previous" bits:

$$P(x_n=0)=1-P(x_n=1) \text{ with } P(x_n=1)\approx\frac{1}{m}\sum_{k=1}^{m}b_{n-k}$$

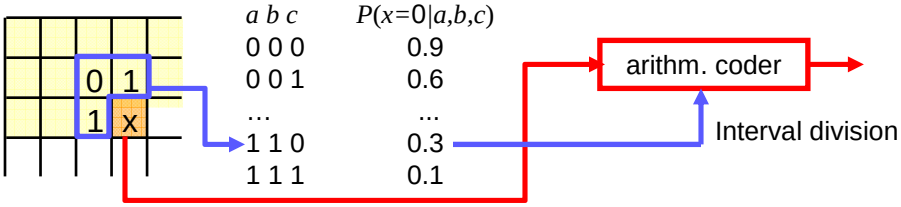
Context modeling: Example binary image…

already coded

current pixel



Binary image: all image pixels take values 0 or 1



We code her a binary image with an arithmetic coder

The interval division of the arithmetic coder now depends on the "context", i.e., on the values of the neighboring pixels:

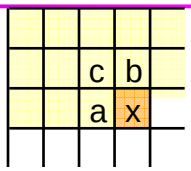
- We estimate, e.g., $P(x=0|a,b,c)$ and base on this the interval division of the coder instead of $P(x=0)\Rightarrow$ we make use of spatial correlation

In practice we estimate $P(x=0|a,b,c)$ adaptively for each context

Context modeling: where is the gain?

already coded

current pixel



Typical images consist of large uniform areas divided by edges

A. If $a=b=c=1$ it is likely that x is in a uniform "white" area:
- $P(x=1|a=1,b=1,c=1)$ is very high and $P(x=0|a=1,b=1,c=1)$ is very small

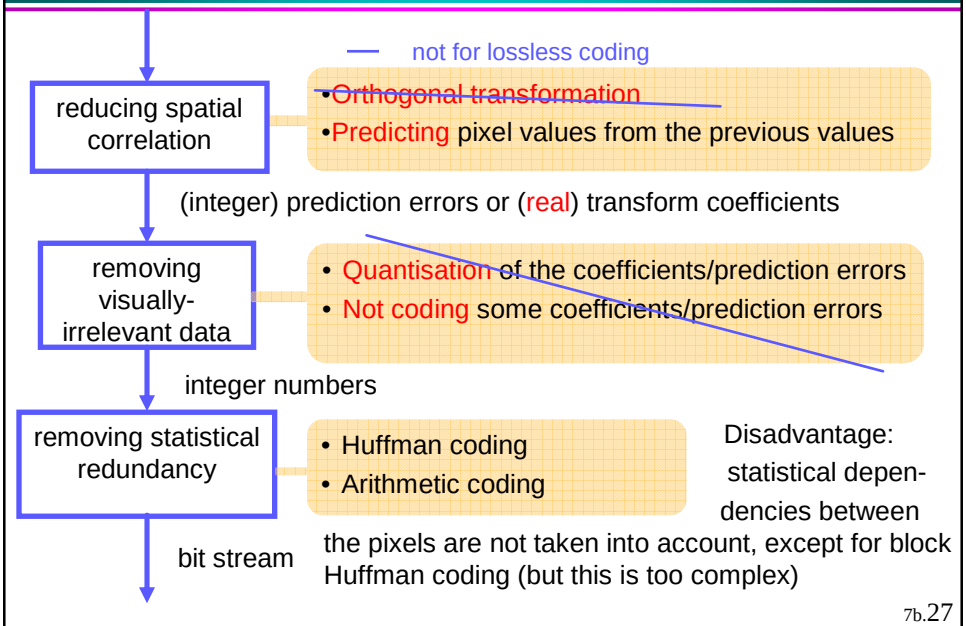
B. In the surrounding of an image edge is e.g. a=c=1 and b=0; here it is not a priori clear whether x is on the white or on the black side of the edge
- $P(x=1|a=1,b=0,c=1)$ and $P(x=0|a=1,b=0,c=1)$ will be closer to 0.5: e.g. $P(x=1|a=1,b=0,c=1)=0.3$ en $P(x=0|a=1,b=0,c=1)=0.7$

It is evident that the optimal interval division will not be the same for the cases A. and B.

$\Rightarrow$we change the interval division based on the context so that for every context an optimal division is found

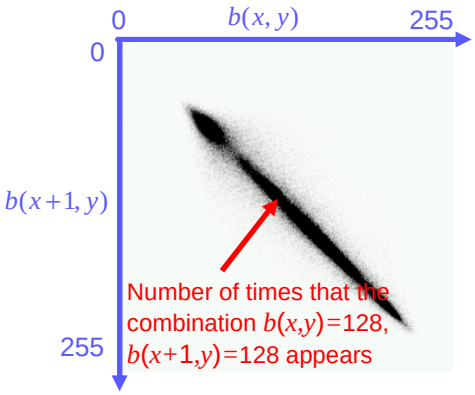
Lossless image compression techniques

A general compression scheme

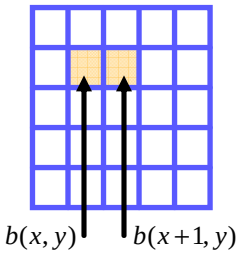


Disadvantage: statistical dependencies between the pixels are not taken into account, except for block Huffman coding (but this is too complex)

Statistical properties of images



$$\rho = \frac{E\left(b(x,y)b(x+1,y)\right)}{\sqrt{E\left(b^2(x,y)\right)E\left(b^2(x+1,y)\right)}}$$



Neighboring pixels in images tend to have similar values

⇒ Almost diagonal co-occurrence matrix (2-nd order histogram)

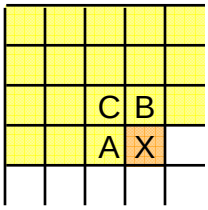
⇒ Block codes perform better than pixel codes; $H_2 << 2H_1$

⇒ Very high correlation coefficient: $\rho > 0.9$

Simple predictive techniques

already coded

current pixel



Prediction: $X_p = \lfloor aA+bB+cC \rfloor$

Coded error: $X-X_p$

Principle:

- The "current" pixel value is first predicted from other values
  - The pixel is predicted based on the neighboring ones
  - We can use only already coded pixels (because the decoder has to be able to make the same prediction)
- The prediction error is coded using the Huffman or the arithmetic coder

Example: LJPEG (Lossless JPEG; Joint Photographic Experts Group)

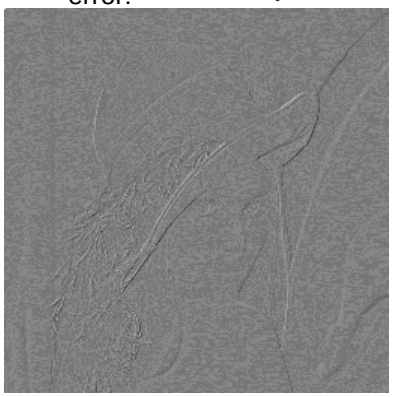
- LJPEG offers choice from 7 predefined predictors
- Predictor "7" works usually the best:

  $X_p=(A + B) / 2$ ("integer" division)

Prediction works: Example...

Lenna, original

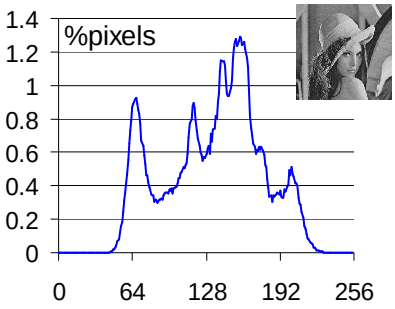
Prediction error: $X - X_p + 128$



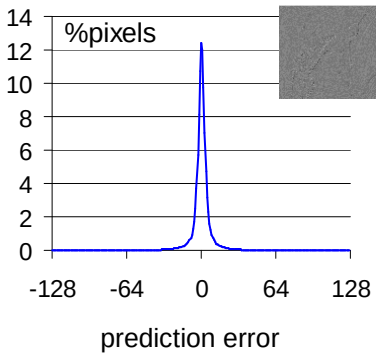
Prediction works well except at edges

…Example...

"Lenna" entropy=7.2 bit/pixel



After LJPG-prediction
entropy=4.4 bit/pixel



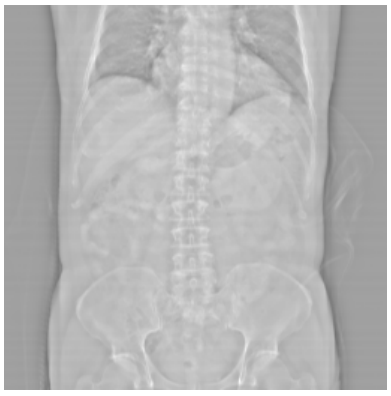
After LJPEG-prediction:

- The errors are less unifor distributed than the original values
- The entropy of the errors is smaller than that of the original values

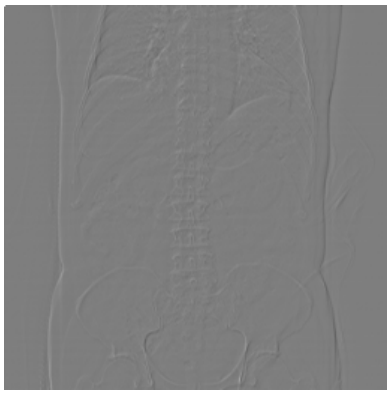
⇒ An arithmetic coder (usually) efficiently codes these than the original values

…Example...

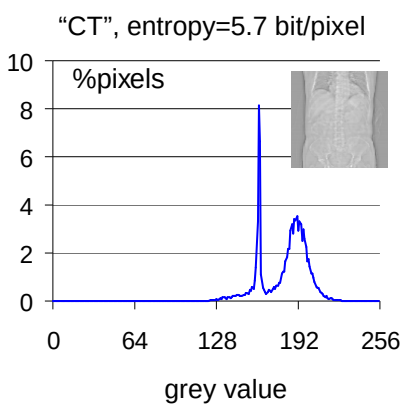
CT, original

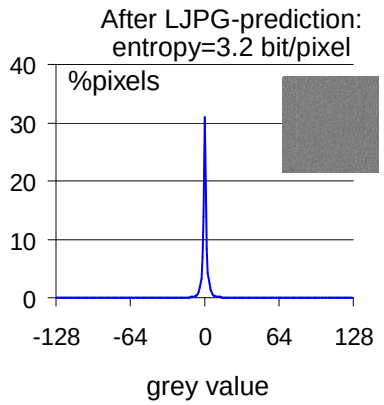


Prediction error: $X - X_p + 128$



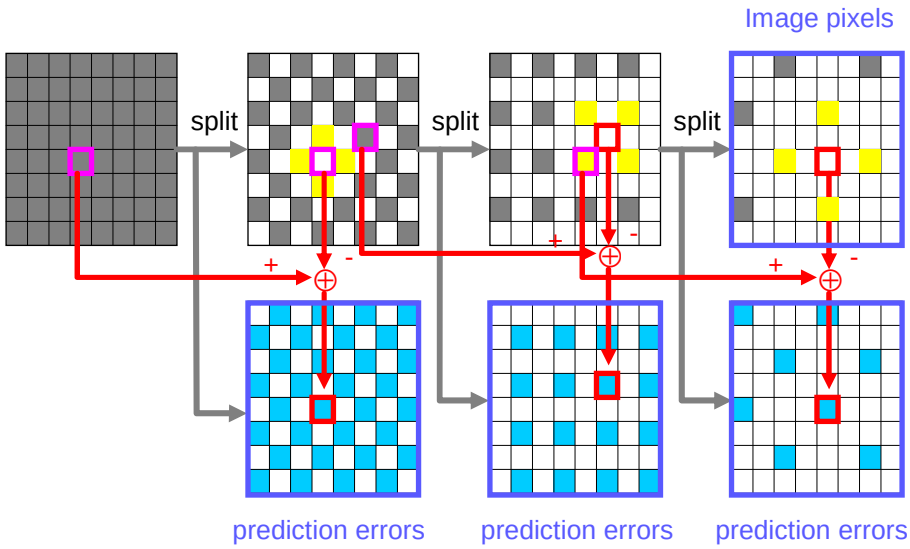
...Example





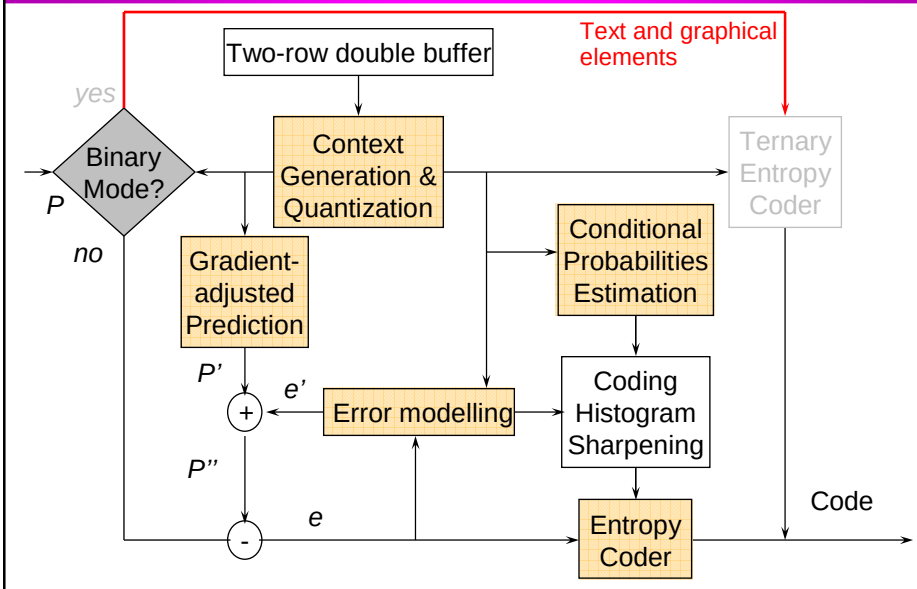
Gain: 1.78 bigger compression factor

The "Binary Tree Predictive Coder"...

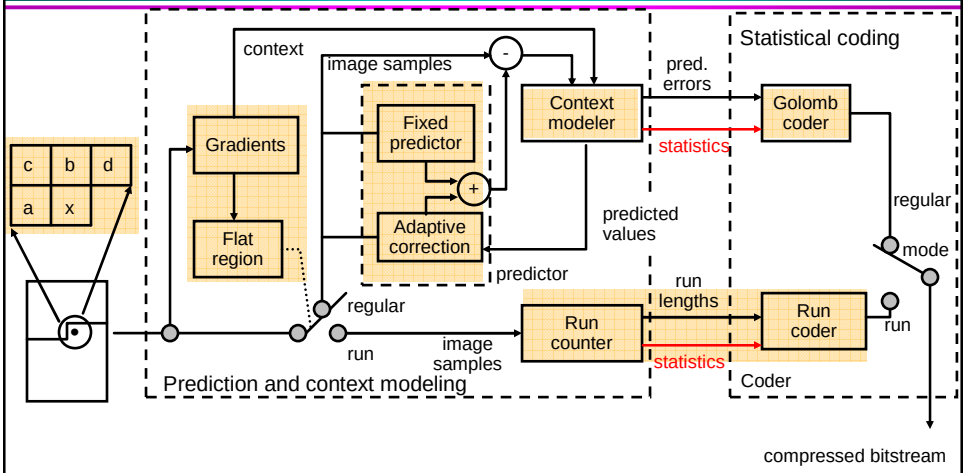


Lossless compression in practice

The CALIC coding scheme

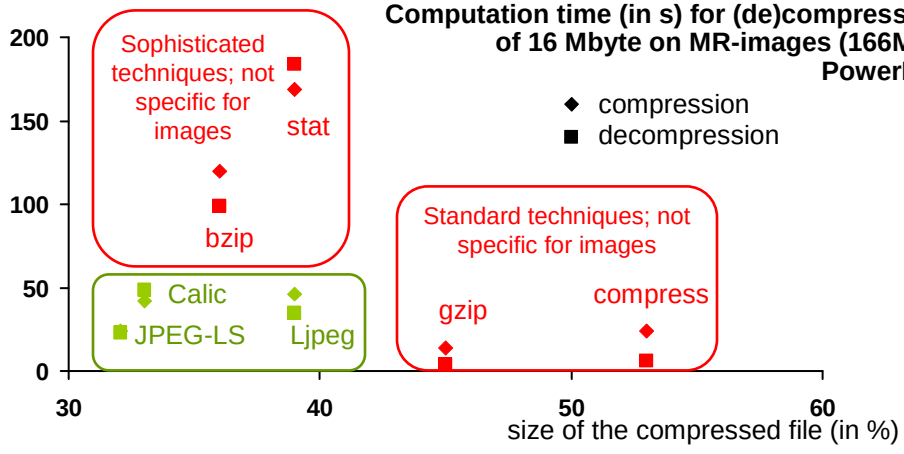


The new standard: JPEG-LS



A very complex predictor and statistical coder
However, quite short computation time
Inspired by the so-called Calic technique, which compresses equally well but much slower

Lossy compression: Example

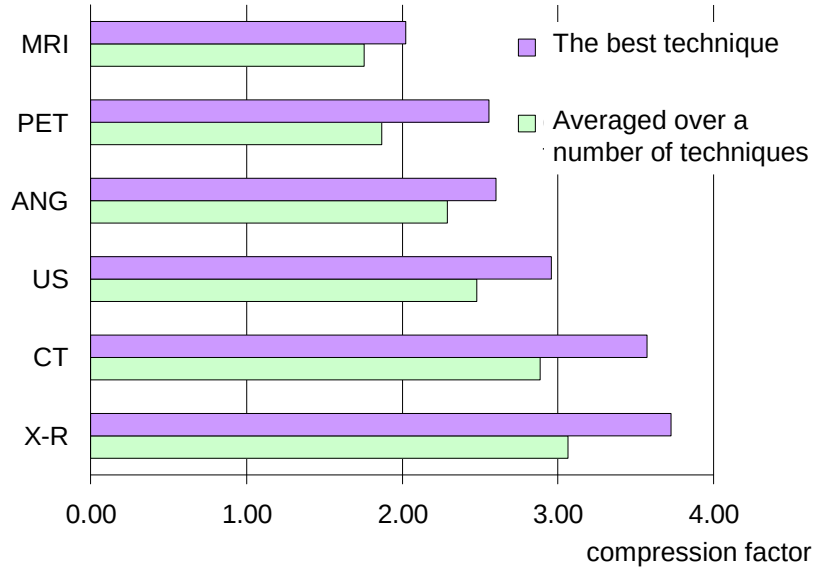


The achievable compression factor is small for lossless compression; this holds for most types of images;

Arithmetic coders perform better than Huffman coders

Advanced techniques for “general purpose” are also quite good but slower

Dependence on the type of the image



The compression factor depends strongly on the type of the image

Conclusions

Important principles
- •pre-processing: linear and non-linear prediction
- •context modeling
- •Statistical coding techniques: Huffman codes and arithmetic codes

Performance comparison:
- •Sophisticated techniques (e.g. JPEG-LS en Calic) are better than simple ones (b.v. LJPG)
- •Arithmetic coders perform better than Huffman coders

Influence of the type of the data
- •The compression factor strongly depends on the type of the image
- •Images with big spatial resolution can be compressed best

Results
- •Typical compression factor is 2 to 4 on medical and “pre-press” images