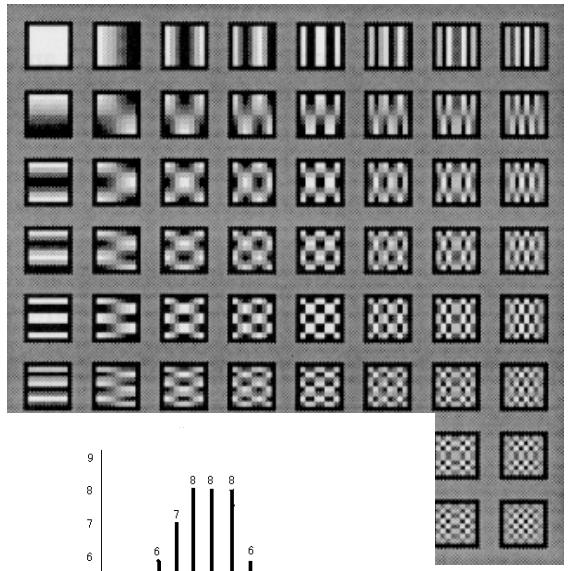
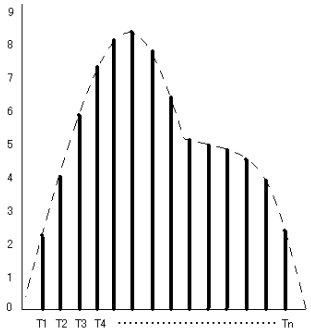
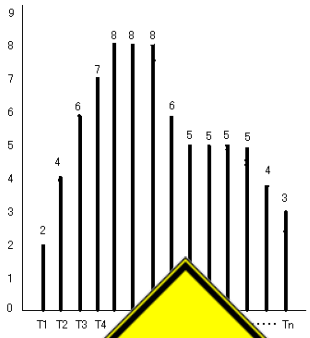
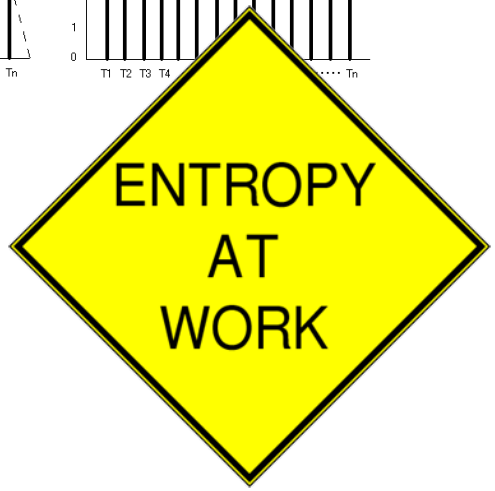
Basics of DCT, Quantization and Entropy Coding









Nimrod Peleg
Update: April 2009

Discrete Cosine Transform (DCT)

- First used in 1974 (Ahmed, Natarajan and Rao).
- Very close to the Karunen-Loeve* (KLT) transform (or: Hotelling** Transform), that produces un-correlated coefficients
- Coefficients can be quantized using visually-weighted quantization values.
- A fast algorithm is available and similar to FFT.

* Named after Kari Karhunen and Michel Loève
** Named after Harold Hotelling

DCT Based Compression

- HVS response is dependent on spatial frequency.
- We decompose the image into a set of waveforms, each with a particular **spatial frequency**

- DCT is a real transform.
- DCT de-correlating performance is very good.
- DCT is reversible (with Inverse DCT).
- DCT is a separable transform.
- DCT has a fast implementation (similar to FFT)

1D DCT* Transform:

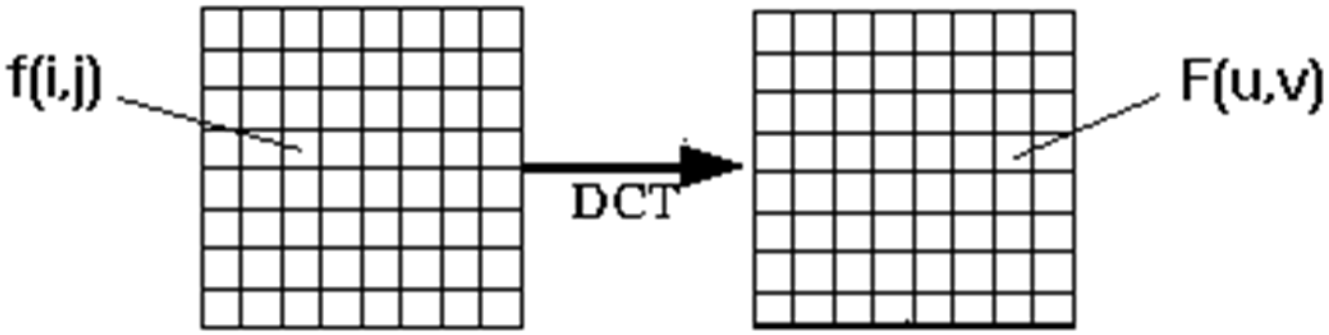
$$F_{(u)} = \left(\frac{2}{N}\right)^{1/2} \sum_{i=0}^{N-1} A_{(i)} \cos\left[\frac{\pi u}{2N}(2i+1)\right] f_{(i)}$$

$$A_{(i)} = \begin{cases} 1/\sqrt{2} & \text{for i=0} \\ 1 & \text{Otherwise} \end{cases}$$

*: one of four definitions !

DCT (Discrete Cosine Transform)

- The DCT helps separate the image into parts (or spectral sub-bands) of differing importance, with respect to the image's visual quality.



2D - DCT

- Image is processed one 8x8 block at a time
- A 2-D DCT transform is applied to the block:

$$F(u,v) = \frac{C(u)C(v)}{4} \sum_{x=0}^{7} \sum_{y=0}^{7} f(x,y) \cos\left(\frac{(2x+1)u\pi}{16}\right) \cos\left(\frac{(2y+1)v\pi}{16}\right)$$

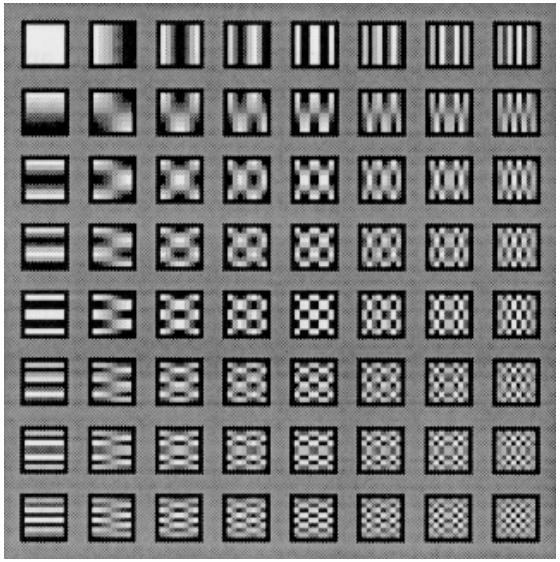
DCT coefficients

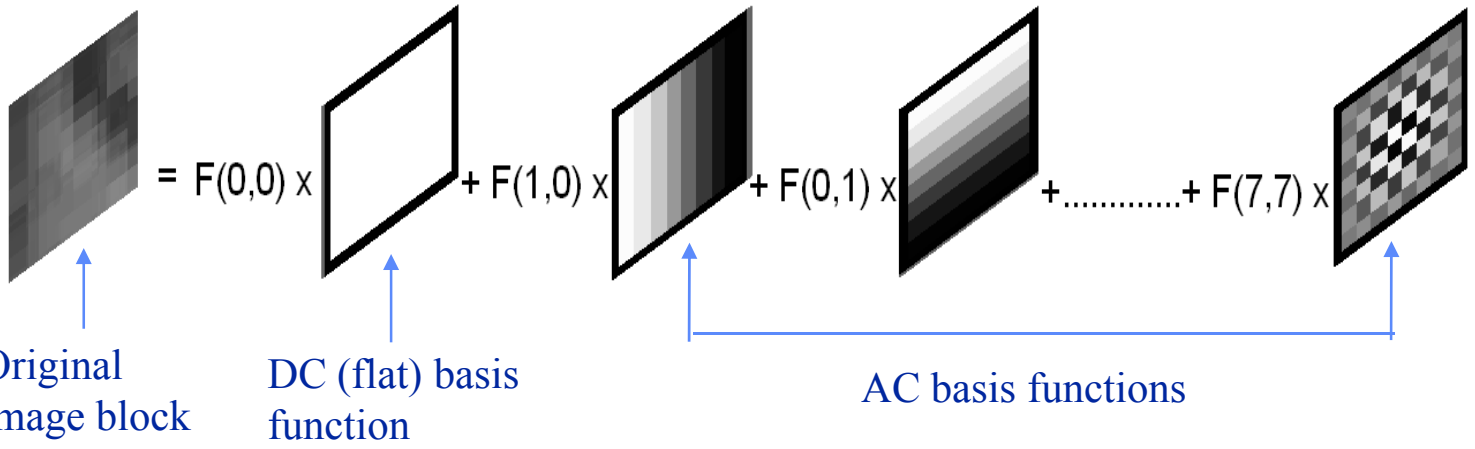
Samples

$$C(n) = \begin{cases} \frac{1}{\sqrt{2}} & n = 0 \\ 1 & n \neq 0 \end{cases}$$

  - Perform nearly as well as the ideal K-L transform
  - Closely related to the DFT transform
  - Computationally efficient

2D-DCT Basic Functions

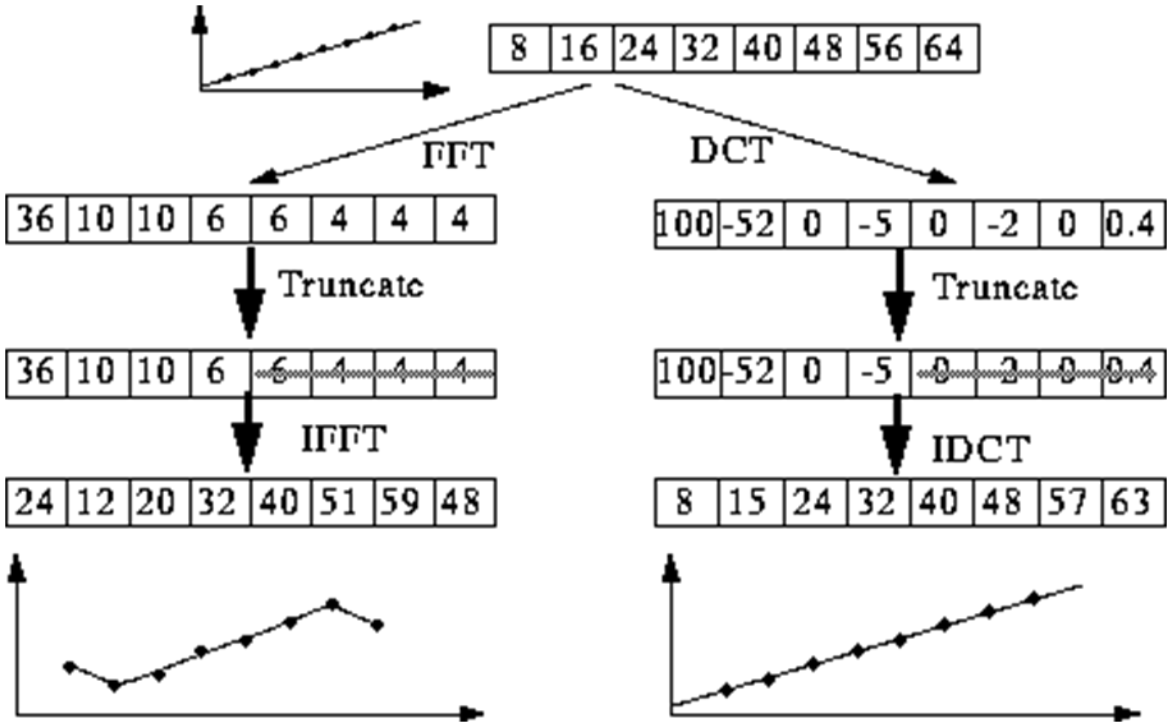




Why DCT and not FFT?

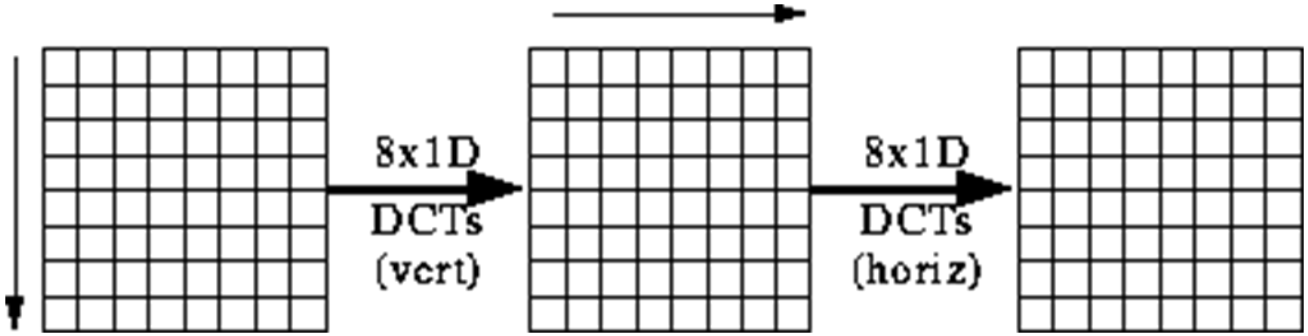
- DCT is similar to the Fast Fourier Transform (FFT), but can approximate lines well with fewer coefficients

A better
"De-Correlator":
Concentrates energy
In the
"lower Frequencies"



Computing the 2D DCT

- Factoring reduces problem to a series of 1D DCTs:
  - apply 1D DCT (Vertically) to Columns
  - apply 1D DCT (Horizontally) to resultant Vertical DCT
    - Or alternatively Horizontal to Vertical.

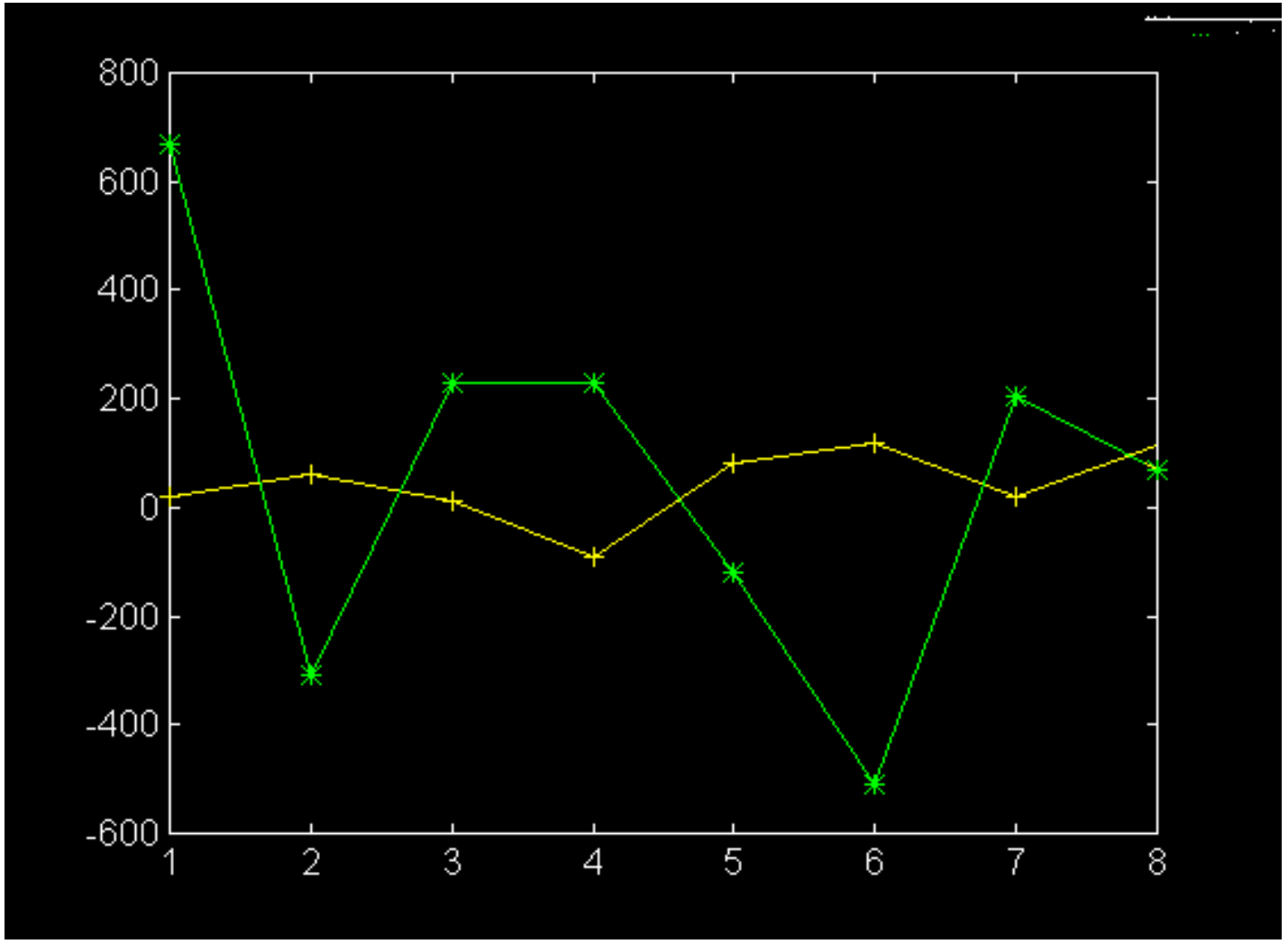


Most software implementations use fixed point arithmetic. Some fast implementations approximate coefficients so all multiplies are shifts and adds.

World record is 11 multiplies and 29 adds. (C. Loeffler, A. Ligtenberg and G. Moschytz, "Practical Fast 1-D DCT Algorithms with 11 Multiplications", Proc. Int'l. Conf. on Acoustics, Speech, and Signal Processing 1989 (ICASSP `89), pp. 988-991)

1-D DCT Correlation Removal

(original signal : yellow, DCT: green)

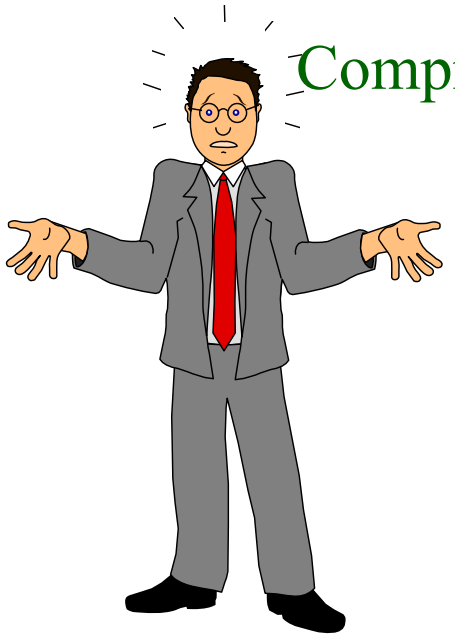


And in numbers...

Original Values

20 60 10 -90 80 120 20 115

DCT



Compression ???

DCT Coeff. Values (Not normalized)

670.0000 -308.3878 229.6567 231.0802
-120.2082 -509.6407 203.3661 69.0309

And the Answer ...

Quantization of DCT Coeff.

- Quantization reduces accuracy of the coefficients representation when converted to integer.
- This zero's many high frequency coefficients.
- We measure the threshold for visibility of a given basis function (coefficient amplitude that is just detectable by human eye).
- We divide (*Quantize*) the coefficient by that value (+ appropriate rounding to integer).

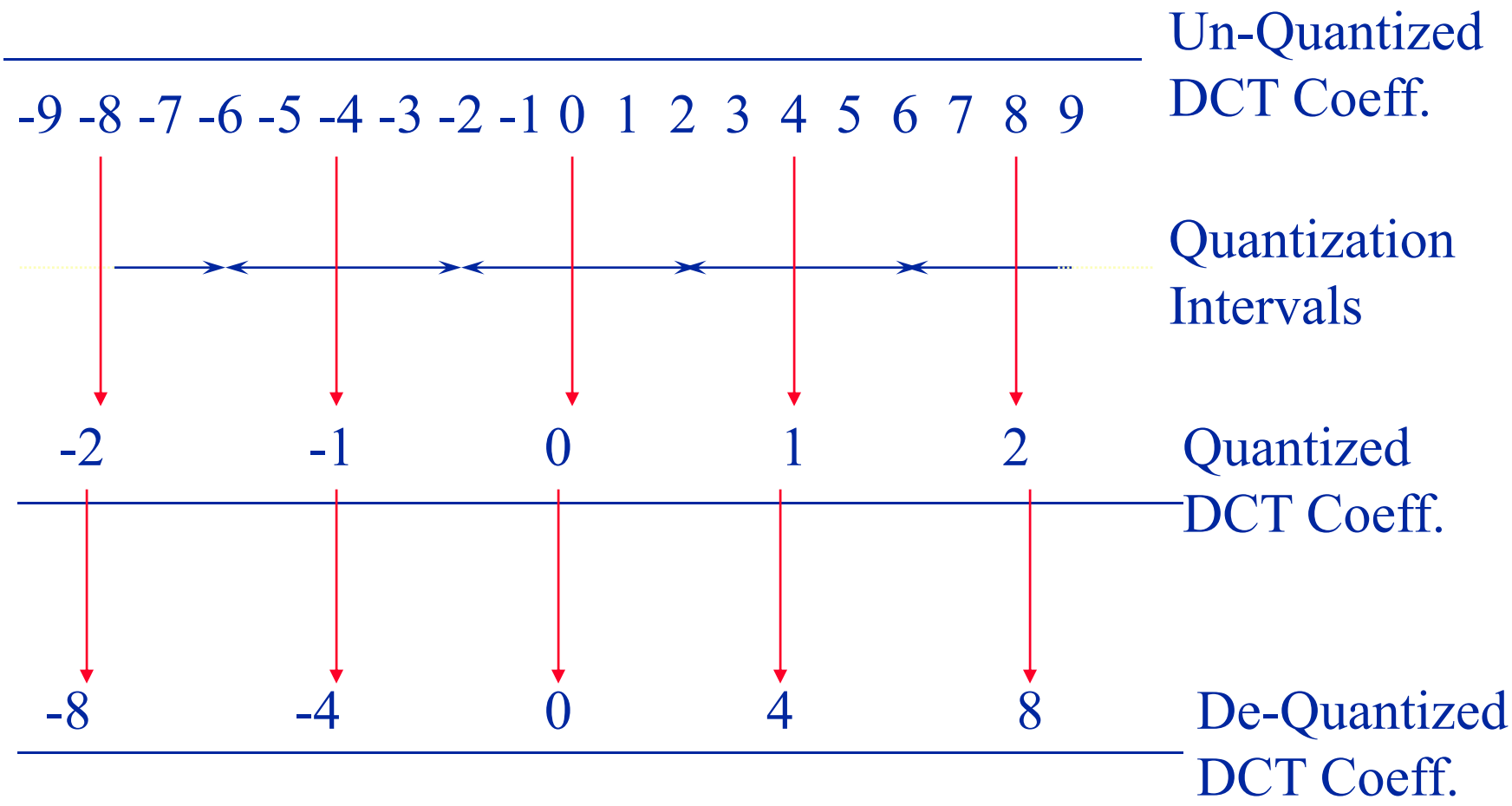
What is Quantization ?

- Mapping of a continuous-valued signal value *x(n)* onto a limited set of discrete-valued signal *y(n)* : ***y(n) = Q [x(n)]***

such as ***y(n)*** is a "good" approximation of ***x(n)***

- *y(n)* is represented by a limited number of bits
- We define Decision levels and Representation levels

Inverse Quantization

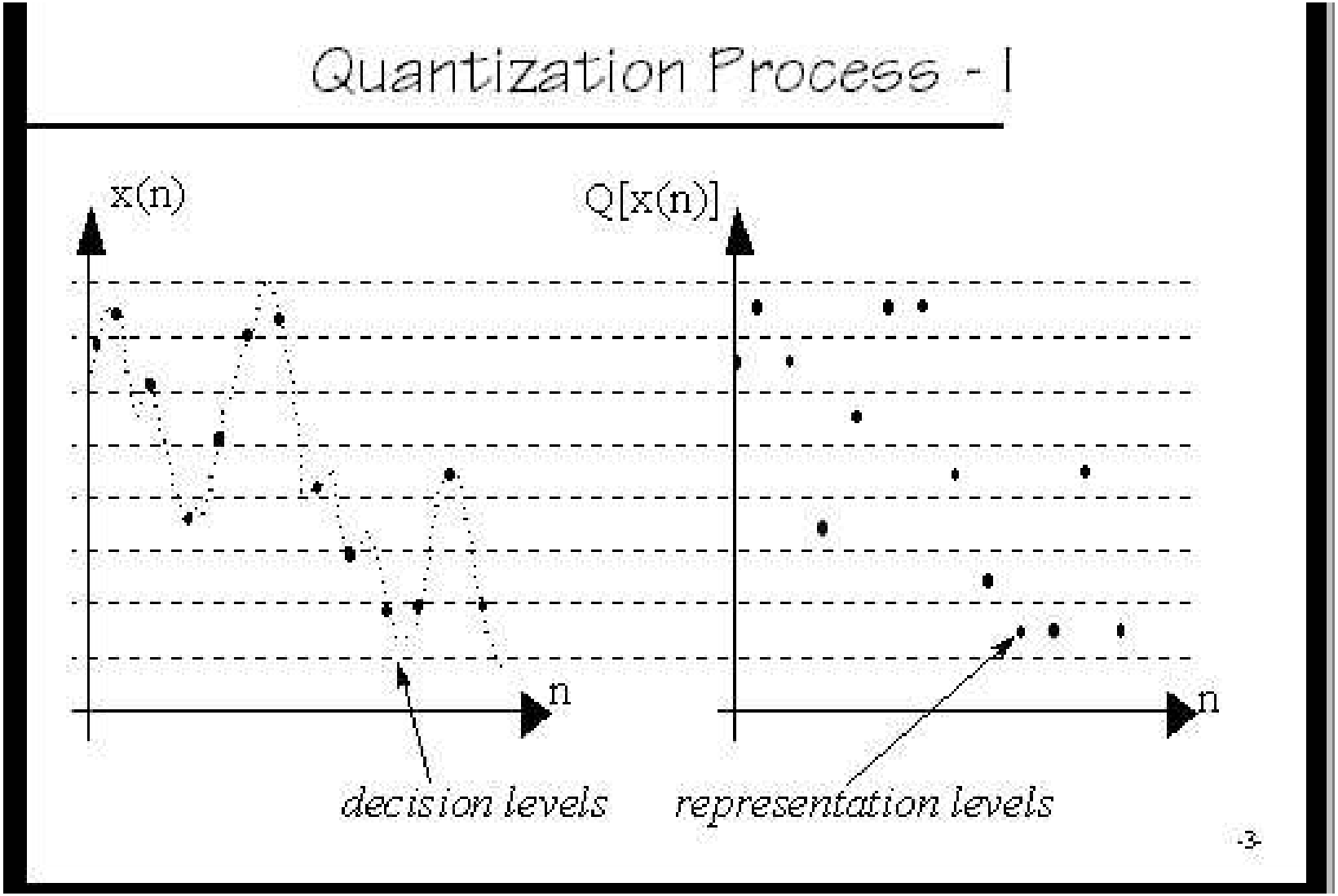


Inverse Quantization (Cont'd)

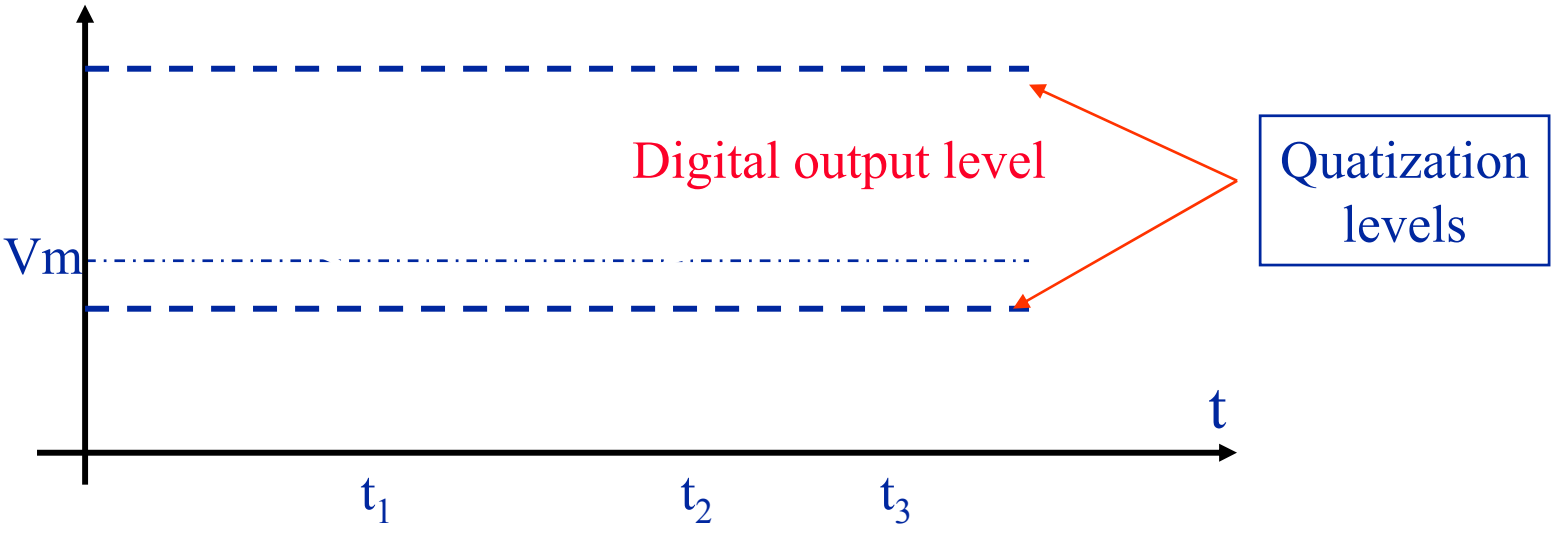
Note that:

- Quantization factor is 4 (for the given example)
  - A half of the Q-factor (i.e 2 for the above) is added to coefficient magnitude before it is divided and truncated
  - done in JPEG
- Linear quantization: for a given quantization value, steps are uniform in size
- 8-bit sample transforms to 11-bit quantized DCT coefficients, for Q value of 1
  - for an 8-bit sample, quantization value of 16 produces 7-bit DCT coefficients)

Decision levels Vs. Representation levels



Quantization Process



Vm is the Average value ('dc') of Vin, and far away from the digital output level

- **e.g. , this can be heard in audio !**

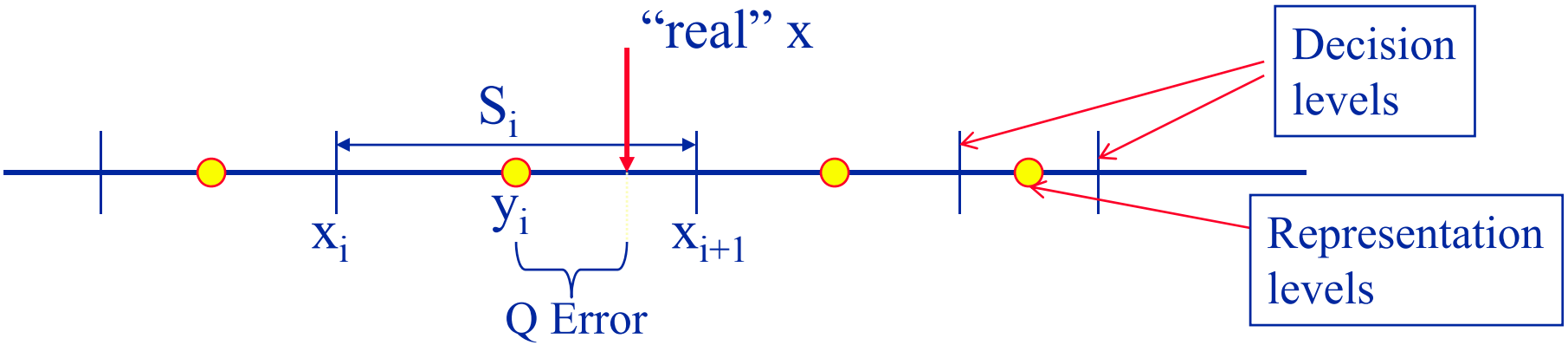
Pseudo random noise can save the need for more quantization levels (Dithering)

Quantization Definition & Error

- Given a signal *x(t)*, divide *x* amplitude into L non-overlapping intervals S_k:

$$S_k = \{x \mid x_k < x \leq S_{k+1}\} \qquad k=1, 2, \ldots, L$$

- If x is in the area of S_i than it is represented by y_i and a *Quantization Error* is introduced

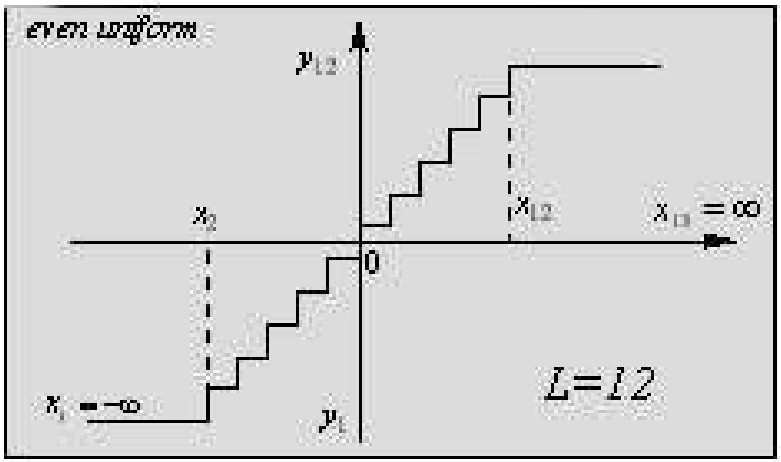


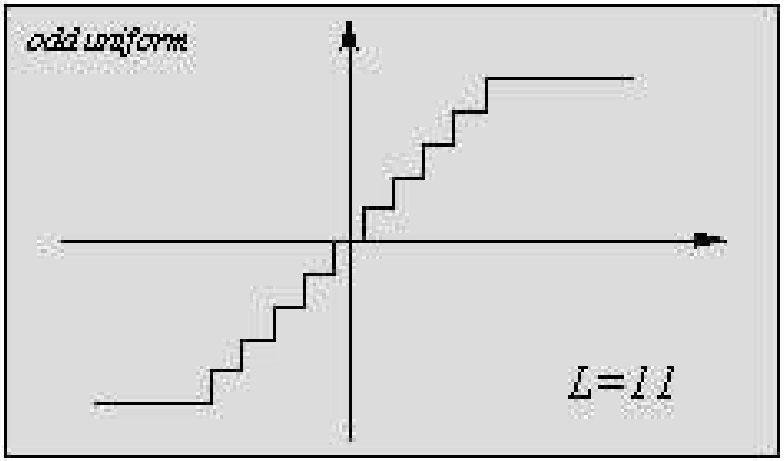
Typical Quantizers - I

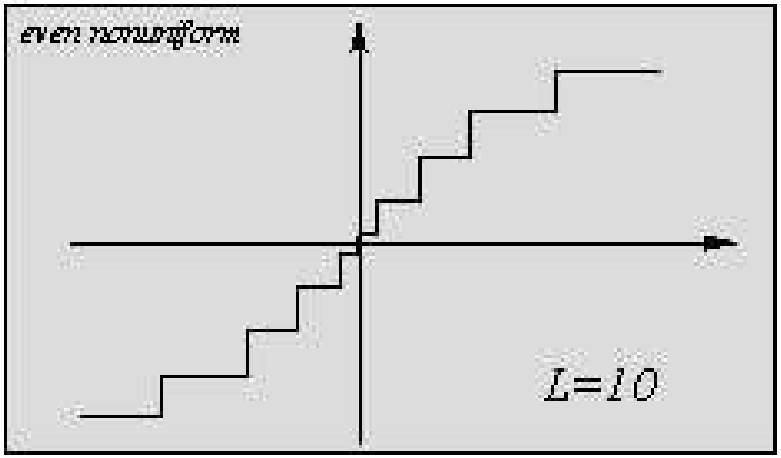
- The signal is assumed to be symmetrical around zero, so, quantizers are usually designed to be symmetrical about the origin

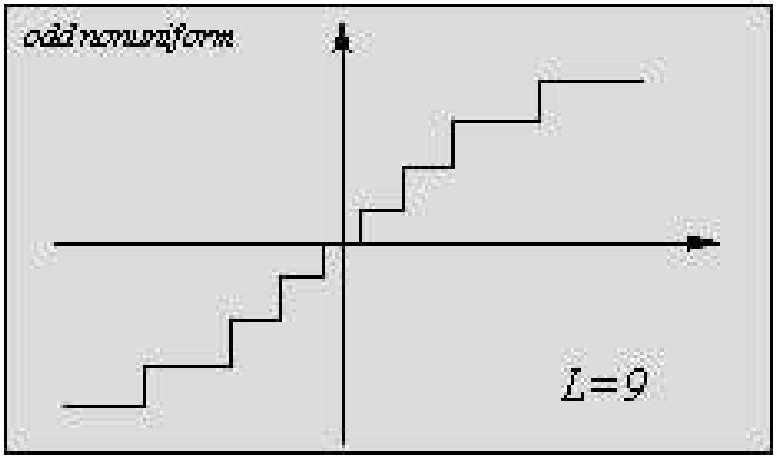
- Options:
  - Odd/Even number of levels
  - Uniform or Non-uniform
  - Optimized (or not) for Probability Density Function (PDF)

Typical Quantizers - I









Luminance Q-Table

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

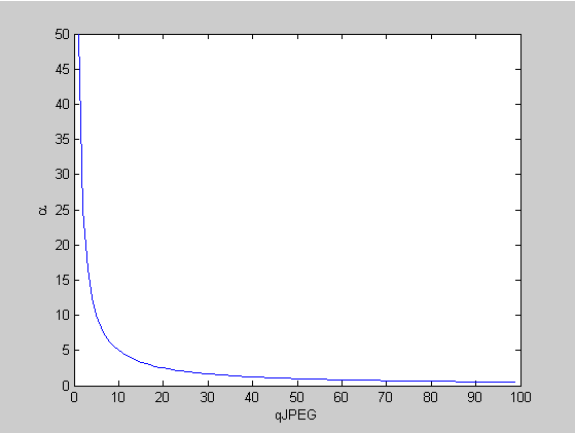
Chrominance Q-Table

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

Quality Factor

- For other quality factors, the elements of these quantization matrices are multiplied by the compression factor α, defined as:

$$\alpha = \begin{cases} \dfrac{50}{q\_JPEG} & 1 \le q\_JPEG \le 50 \\ 2 - \dfrac{2q\_JPEG}{100} & 51 \le q\_JPEG \le 99 \end{cases}$$



- The minimum legal α·Q(u,v) is 1
  - **This cannot be a lossless compression !!!**

Quantization table and quality factor are not specified in the Standard – just recommended (Annex K)

Notes on Q-Tables

- Visibility measured for:
  - 8x8 DCT basis functions
  - Luminance resolution: 720x576
  - Chrominance resolution: 360x576
  - Viewing distance: 6 times screen width
- Noticeable artifacts on high-quality displays, since little is known about visibility thresholds when more than one coefficient is zero (simultaneously)

More About JPEG DCT Process

- Zero-shift on input samples to convert range from 0:255 to -128:127, reduces internal precision requirements in DCT calculations.
- DCT is always done on 8x8 blocks:
  - Spatial frequencies in the image and of the cosine basis are not precisely equivalent.
  - "Blocking artifacts" appear if not enough AC coeff. are taken.

DCT Computational Complexity

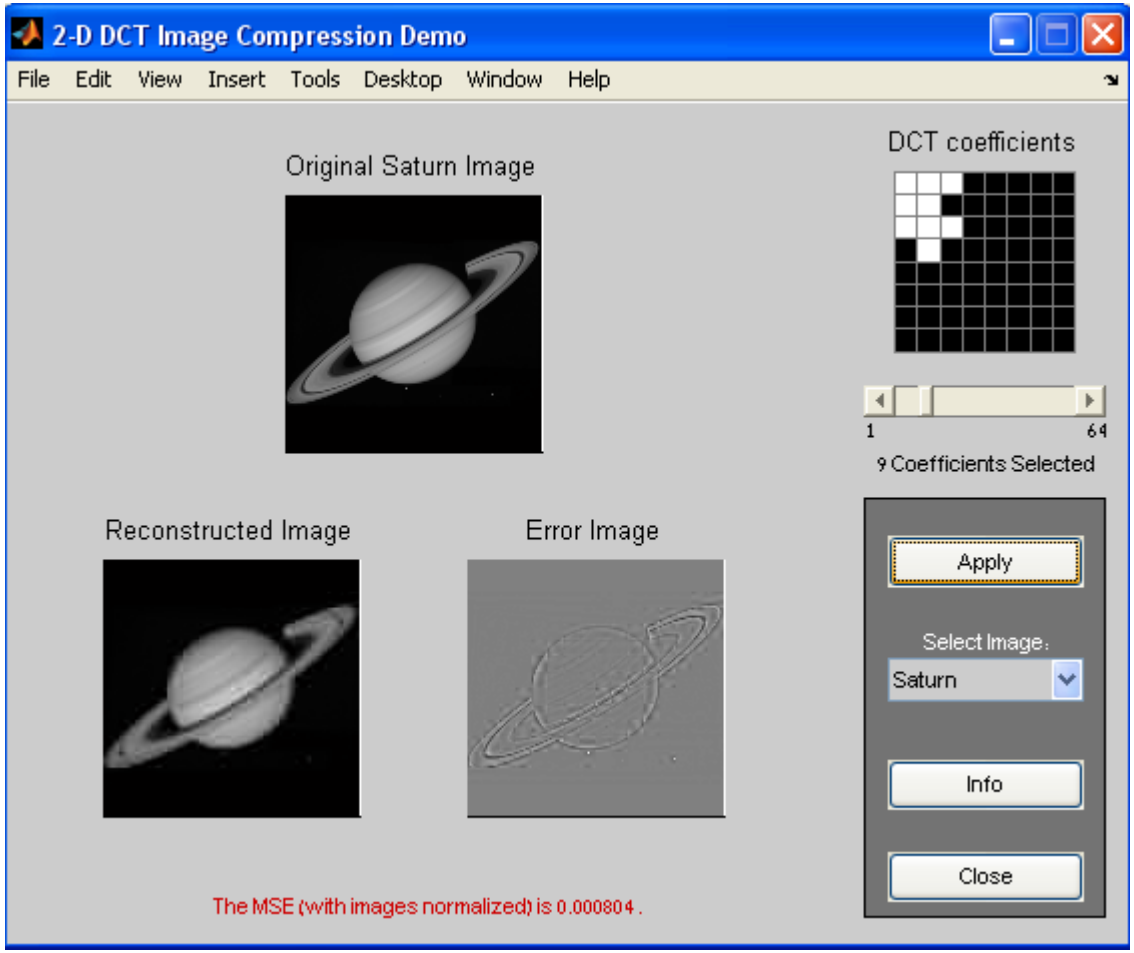
- Straightforward calculation for 8x8 block:

each coeff. needs 64 multiplies and 63 additions. Quantization requires one more addition and a division.

- Fast algorithm requires 1 multiply and 9 additions.
- Modern pipelined processors architectures

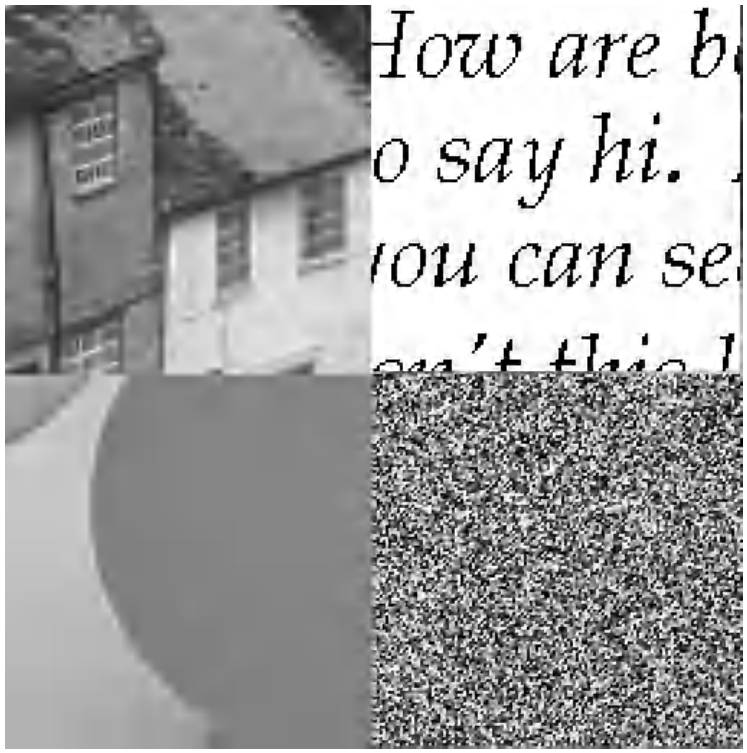
makes the calculation very economic and fast.

Erasing DCT Coefficients: a Matlab demo

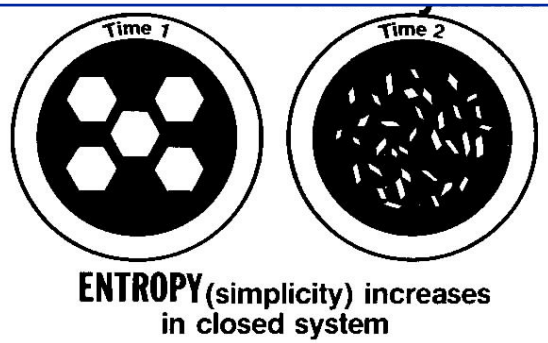


- Run the script: "dctdemo.m"
- Zero coefficients and see the effect

Entropy Coding: VLC



Second low of Thermodynamics



אנטרופיה:
רמת אי-הסדר במערכת

בתרמודינמיקה:
מדד לכמות האנרגיה שאינה זמינה לעבודה

Entropy Encoding

- Amount of information in a symbol: $F = \log_2 \frac{1}{Pi}$
  - Entropy of an Image:

"Average amount of information"

  - A source with M possible symbols, With a uniform distribution:

$$H_1 = -\sum_{i=1}^{M} Pi \log_2 Pi \quad [\text{bit}]$$

- Shannon Theorem:

H= minimum number of bits to code the source

The Shannon Theorem

- Shannon (1948) showed that for the coding of a discrete (independent) source, a uniquely decodable binary code exists, with average codeword length : $H(Y(n)) \leq Save$

- Shannon also showed that for this source, a binary prefix code (such as Huffman) exists, and holds: $Save \leq H(Y(n)) + 1$

The Shannon Theorem (Cont'd)

- If N independent symbols are taken together we get: $\text{Save} \leq \text{H}(\text{Y}(\text{n})) + 1/\text{N}$

- For very large N , S_{ave} is very close to the entropy, but the practical problems become severe:
  - *coding delay*
  - *complexity*
  - *size of VLC tables*

k-th order entropy 1/2

- For an ‘alphabet’ A (set of symbols):

$$H_k = -\frac{1}{k} \sum_{x_1 \ldots x_k \in A^k}^{M} P(x_1,\ldots x_k) \log_2 P(x_1,\ldots x_k) \quad \text{[bit]}$$

- The important property of the entropy is:

$$\log 2|A| \geq H_k(X) \geq H_{k+1}(X) \geq 0$$

- The expected value (per-letter in ‘A’ !) is always grater than or equal to entropy:

$$\frac{E\{L(X_1,\ldots X_n)\}}{n} \geq H_n(X)$$

Shannon, Theorem 1

k-th order entropy 2/2

- Shannon, Theorem 2

There exists a coding algorithm such that:

$$\frac{E\{L(X_1,\ldots X_n)\}}{n} < H_n(X) + \frac{1}{n}$$

- Important to remember:

for a "memoryless" source (symbols are independent): entropy of any order is equal to the first order entropy !

Influence of block length

- Does theorem 2 mean that for memory-less source, we can reach entropy by 1-symbol blocks?

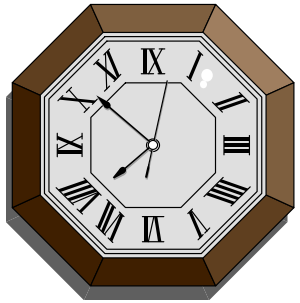
Consider a binary source with entropy : H_1=0.4
According to the above, the lower bound is H_1 +1/n , with n=1 we get : 0.5 + 1 =1.5 bpp

But …

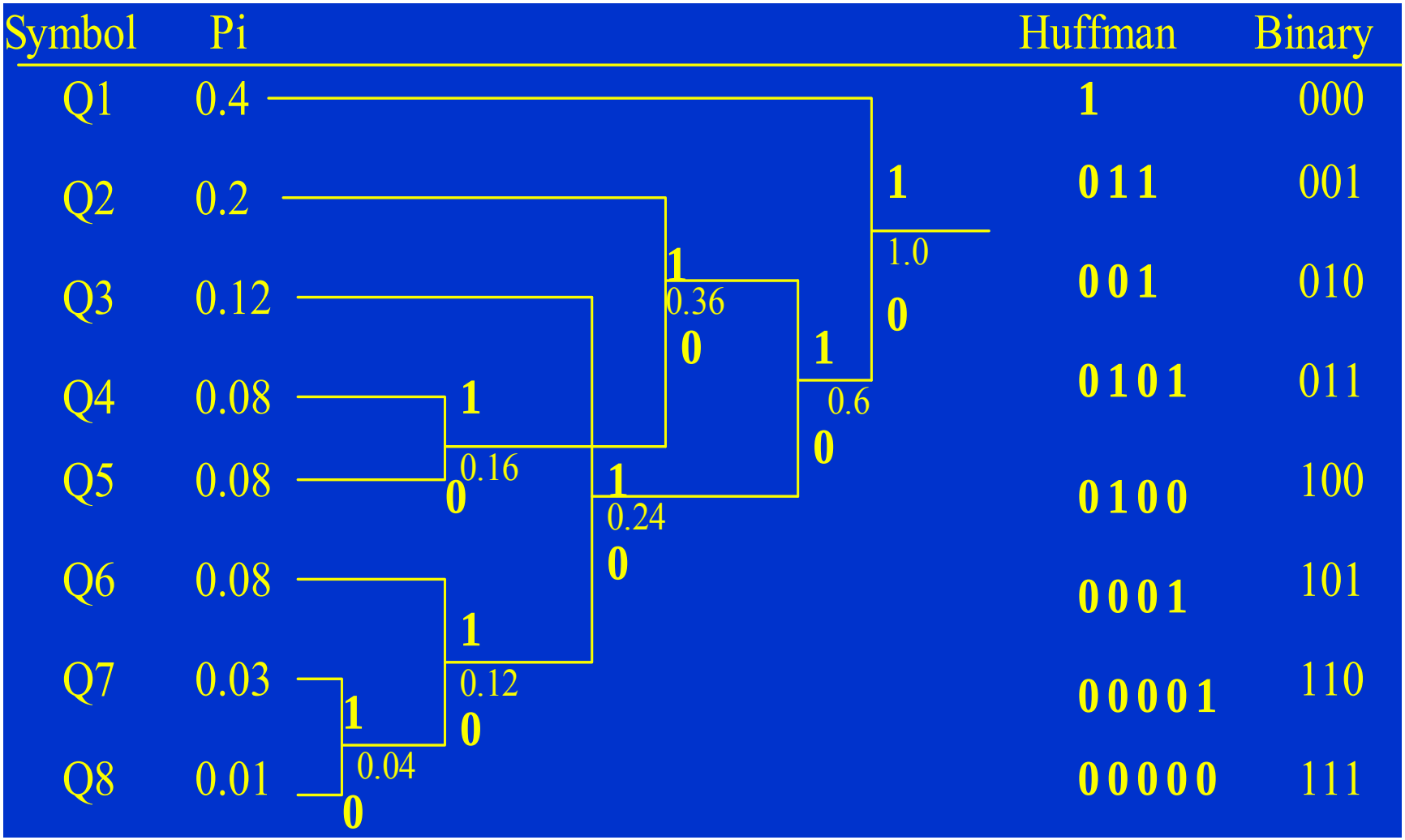
we can easily do it with 1 bpp ….

Huffman Coding

- The symbol with the highest probability is assigned the shortest code and vice versa.
- Codewords length is not fixed (VLC)
- A codeword cannot be a prefix for another codeword (Self Synchronized Code)



Huffman Coding Example



Huffman Coding Example (Cont'd)

- Source Entropy (H_0): $H = -\sum_{i=1}^{M} Pi \log_2 Pi = 2.45$

- Huffman Rate: $R = \sum PiLi = 2.52$
- No Coding: 3 bps
- Compression Ratio: 2.52/3 = 0.84

- Problems:
  - Sensitive to BER
  - Statistics Dependent
  - Not Unique

Other Coding Techniques

- Run-Length

Coding of strings of the same symbol

(Used in JPEG)

- Arithmetic (IBM)

Probability coding (A JPEG Option)

- Ziv-Lempel (LZW)

Used in many public/commercial application such as ZIP etc.. (LZ Demo - Java Applet)

(http://www.cs.sfu.ca/cs/CC/365/li/squeeze/LZW.html)

So…how JPEG do it ?

- The theoretical entropies of each quuantized DCT sub-band are known.
- In practice this would be a poor way to code the data because:
  - 64 separate entropy codes would be required, each requiring many extra states to represent run-length coding of zeros.
  - The statistics for each code are likely to vary significantly from image to image.
  - To transmit the code table for each sub-band as header information would involve a large coding overhead.
  - Coding the sub-bands separately does not take account of the correlations which exist between the positions of the non-zero coefficients in one sub-band with those of nearby sub-bands.

Coding the DC Coefficients

- The size of the differences can in theory be up to

$\pm(255\times8) = \pm2040$ if the input pixels occupy the range -128 to +127 (The DCT has a gain of 8 at very low frequencies)

- Hence the Huffman code table would have to be quite large.
- JPEG adopts a much smaller code by using a form of floating-point representation, where *Size* is the base-2 exponent and *Additional Bits* are used to code the polarity and precise amplitude
- There are only 12 Sizes to be Huffman coded, so specifying the code table can be very simple and require relatively few bits in the header: 16 header bytes + 12 codes = 28 bytes for the DC table.

The DC Codes

DC Coef Difference	Size	Typical Huffman codes for Size	Additional Bits (in binary)
0	0	00	-
-1,1	1	010	0,1
-3,-2,2,3	2	011	00,01,10,11
-7,...,-4,4,...,7	3	100	000,...,011,100,...111
-15,...-8,8,...,15	4	101	0000,...,0111,1000,...,1111
-1023,...-512,512,...,1023	10	1111 1110	00 0000 0000,...,11 1111 1111
-2047,...-1024,1024,...2047	11	1 1111 1110	000 0000 0000,...,111 1111 1111

- Only Size needs to be Huffman coded in the above scheme: within a given Size, all the input values have sufficiently similar probabilities

- Each coded Size is followed by the appropriate number of Additional Bits to define the sign and magnitude of the coefficient difference exactly

The Run-Size Technique

- The remaining 63 quantized AC-coefficients usually contain many zeros and so are coded with a combined run-amplitude Huffman code.
- The codeword represents the run-length of zeros before a non-zero coefficient *and* the Size of that coefficient.

- This is followed by the Additional Bits which define the coefficient amplitude and sign precisely.

- This 2-dimensional Huffman code (Run, Size) is efficient because there is a strong correlation between the Size of a coefficient and the expected Run of zeros which precedes it: small coefficients usually follow long runs; larger coefficients tend to follow shorter runs.

The AC Run-Size Table

• To keep the code table size *n* below 256, only the following Run and Size values are coded: Run=1→15 ; Size=1→10
These require 150 codes.

• Two extra codes, corresponding to (Run,Size) = (0,0) and (15,0) are used for EOB (End-of-block) and ZRL (Zero run length).

• EOB is transmitted after the last non-zero coef in a 64-vector.
It is omitted in the rare case that the final element is non-zero.

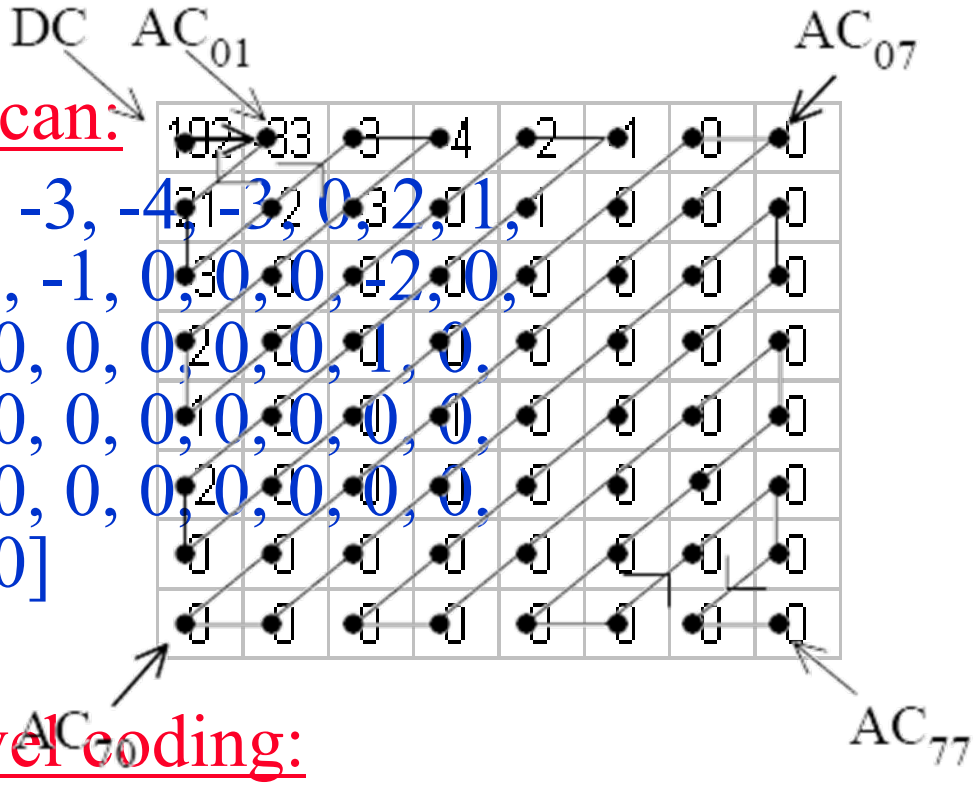
• ZRL is transmitted whenever Run>15: a run of 16 zeros which can be part of a longer run of any length.

- Hence a run of 20 zeros followed by -5 would be coded as:
  (ZRL) (4,3) 010

The AC Huffman Table

(Run,Size)	Code Byte (hex)	Code Word (binary)	(Run,Size)	Code Byte (hex)	Code Word (binary)
(0,1)	01	00	(0,6)	06	1111000
(0,2)	02	01	(1,3)	13	1111001
(0,3)	03	100	(5,1)	51	1111010
(EOB)	00	1010	(6,1)	61	1111011
(0,4)	04	1011	(0,7)	07	11111000
(1,1)	11	1100	(2,2)	22	11111001
(0,5)	05	11010	(7,1)	71	11111010
(1,2)	12	11011	(1,4)	14	111110110
(2,1)	21	11100			
(3,1)	31	111010	(ZRL)	F0	11111111001
(4,1)	41	111011			

Run-Level Coding



- After zizag scan:

[-33, 21, -3, -2, -3, -4, -3, 2, 3, 2, 1,
0, 1, 0, -2, -1, -1, 0, 0, 0, -2, 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, 0, 0, 0]

- After run-level coding:

(0,-33) (0,21) (0,-3) (0,-2)
(0,3)(0,-4) (0,-3) (1,2) (0,1)
(1,1) (1,-2) (0,-1) (0,-1) (3,-2)
(11,1)

A Coding Example

Assume the values of a quantized DCT matrix are given (Zig-Zag order):

42	16	-21	10	-15	0	0	0
3	-2	0	2	-3	0	0	0
0	0	2	-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

A Coding Example (Cont'd)

- If the DC value of the previous block was 44, then the difference is -2. If Huffman codeword for *size=2* is 011 then the codeword for the DC is: 01101
- Using Annex K in the JPEG for the luminance AC coeff., we receive the following codewords:

A Coding Example (Cont'd)

Value	Run/Size	Huffman	Ampl.	Total bits
16	0/5	11010	10000	10
-21	0/5	11010	01010	10
10	0/4	1011	1010	8
-15	0/4	1011	0000	8
3	3/2	111110111	11	11
-2	0/2	01	01	4
2	1/2	11011	10	7
-3	0/2	01	00	4
2	5/2	11111110111	10	13
-1	0/1	00	0	3
EOB	0/0	1010		4

A Coding Example (Cont'd)

- Compression results:
  - DC coeff. : 5 bits
  - AC coeff. : 82 bits
  - Average bitrate: 87/64 = 1.36 bit/pixel
  - Starting from 8 bpp resolution, the compression ratio is : 8/1.36 = 5.88

A Coding Example (Cont'd)

- Run-length coding contribution:

Direct Huffman coding of DC differentials [-2047, 2047] and AC differentials [-1023,1023] requires code tables with 4,095 and 2,047 entries respectively.

By Huffman coding only the *(run,size)* pairs, tables are reduced to 12 and 161 entries respectively

(Run,Size) and Quantization

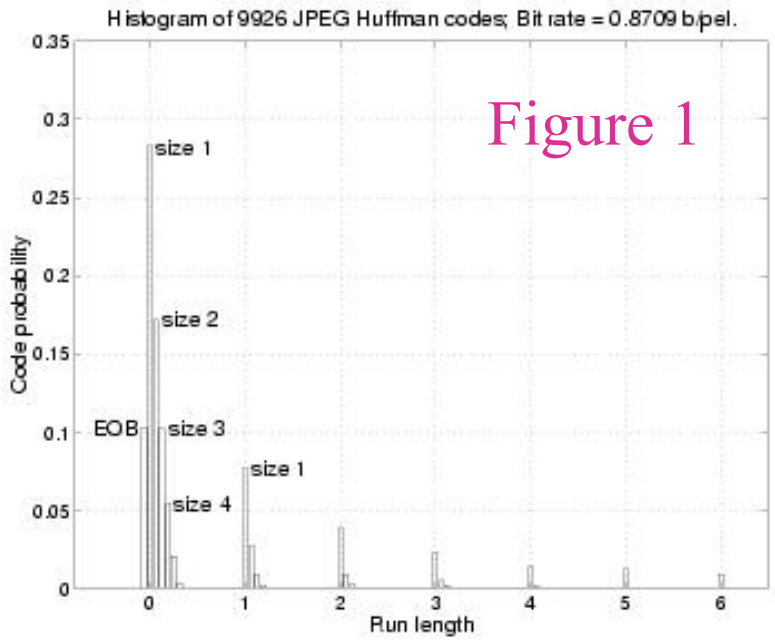


Figure 1

Histogram of the (Run,Size) codewords for the DCT of Lenna, quantised using *Q*lum.

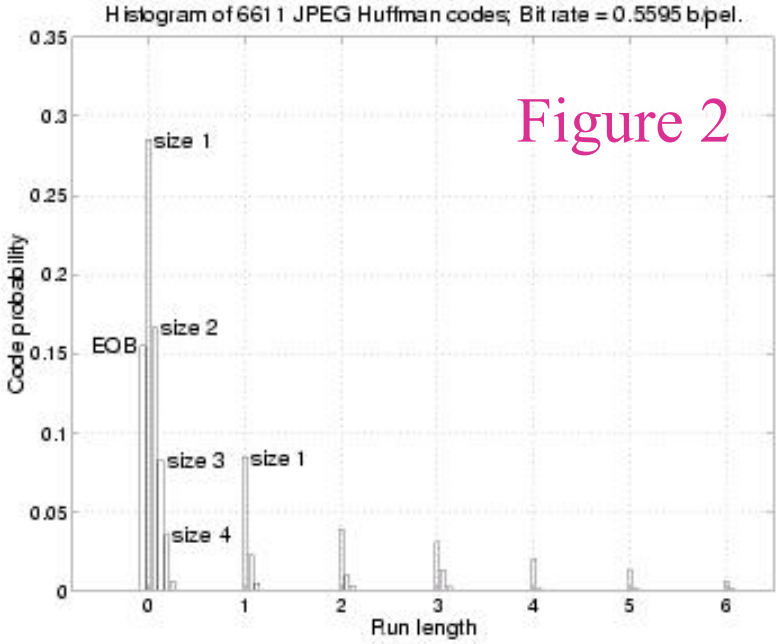


Figure 2

Histogram of the (Run,Size) codewords for the DCT of Lenna, quantised using 2*Q*lum.

The bin number represents the decoded byte value

(Run,Size) and Quantization Cont'd

- Note the strong similarity between the histograms, despite the fact that figure 2 represents only 2/3 as many events.

- Only the EOB probability changes significantly
  - because its probability goes up as the number of events (non-zero coefficientss) per block goes down.
- It turns out that the (Run,Size) histogram remains relatively constant over a wide range of images and across different regions of each image.
  - This is because of the strong correlation between the run lengths and expected coefficients sizes.
- The number of events per block varies considerably depending on the local activity in the image, but the probability distribution of those events (except for EOB) changes much less.

(Run,Size) and Quantization Cont'd

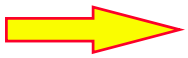
Comparing the mean bit rates to code Lenna for the two quantization matrices with the theoretical entropies:

Q matrix	Mean Entropy (b/pel)	JPEG Bit Rate (b/pel)	JPEG efficiency
Q_{lum}	0.8595	0.8709	98.7%
$2\ Q_{lum}$	0.5551	0.5595	99.21%

• We see the high efficiency of the (Run,Size) code at two quite different compression factors. This tends to apply over a wide range of images and compression factors and is an impressive achievement.

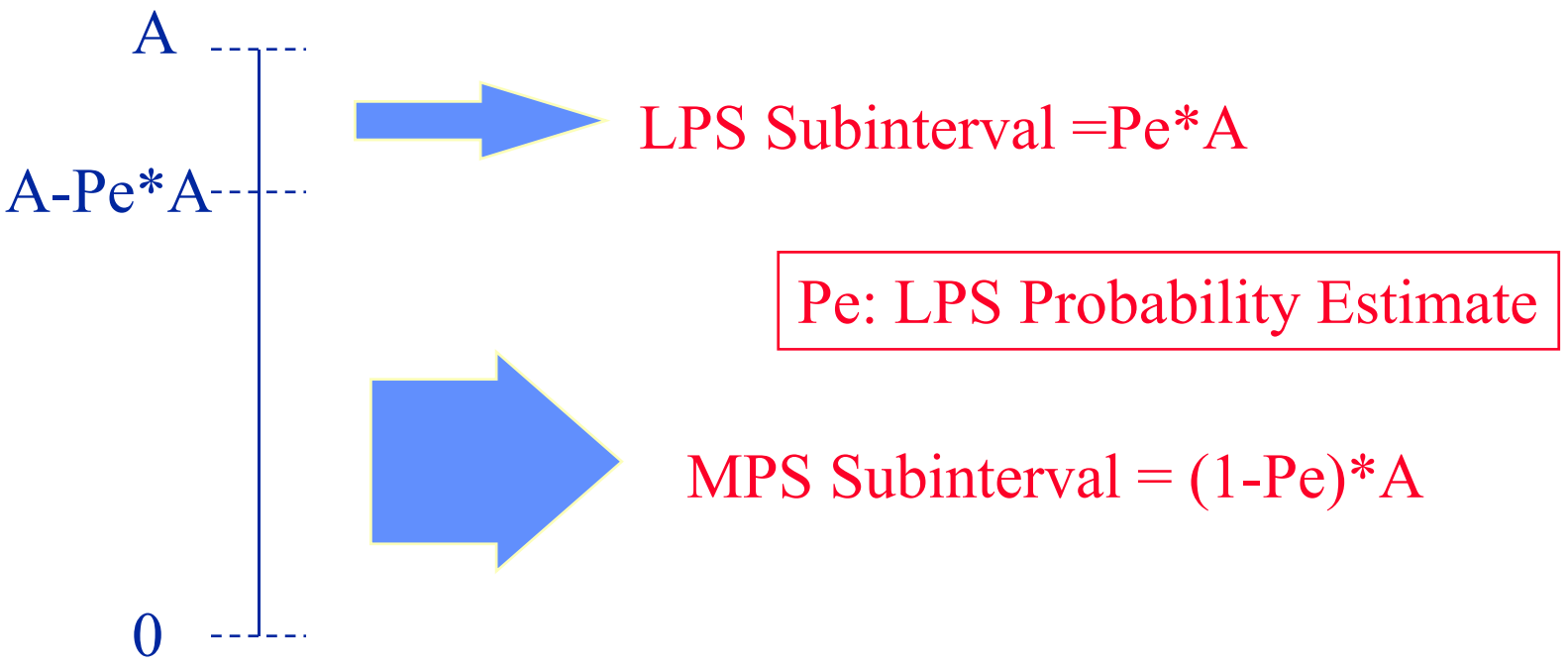
• There is very little efficiency loss if a single code table is used for many images, which can avoid the need to transmit the 168 bytes of code definition in the header of each image. Using the recommended JPEG default luminance tables (Annex K.3.3) the above efficiencies drop to 97.35% and 95.74% respectively.

Arithmetic Coding

- Used in both JPEG and JBIG
- Also known as QM-Coder (A "child" of well-known Q-Coder)
- A Binary coder (1 or 0 symbols only)
- descriptor is decomposed into a tree of binary decisions
- Rather than assigning intervals to the symbols it assigns intervals to the most and less probable symbols (MPS, LPS) ... 

Arithmetic Coding (Cont'd)

...Then orders the symbols such that the LPS sub-interval is above the MPS:



Arithmetic Coding (Example)

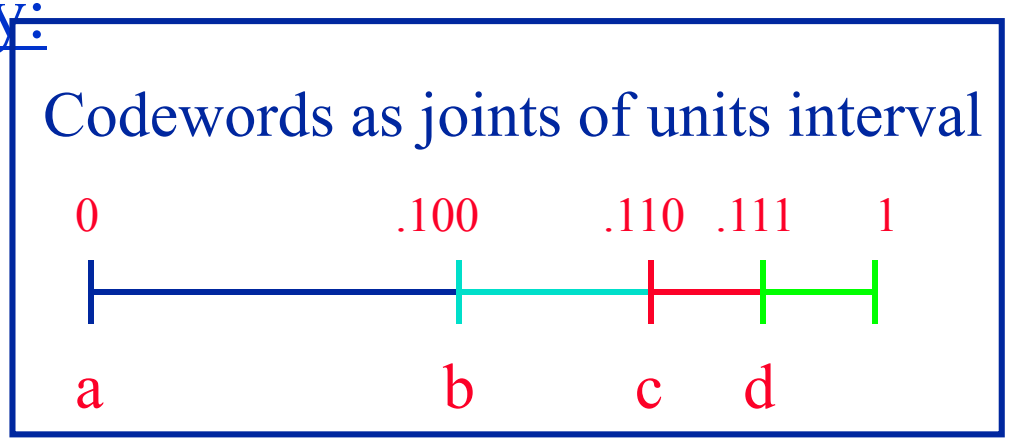
Assume quad-dictionary:

a (Pa=1/2 -> .100)

b (Pb=1/4 -> .010)

c (Pc=1/8 -> .001)

d (Pd=1/8 -> .001)

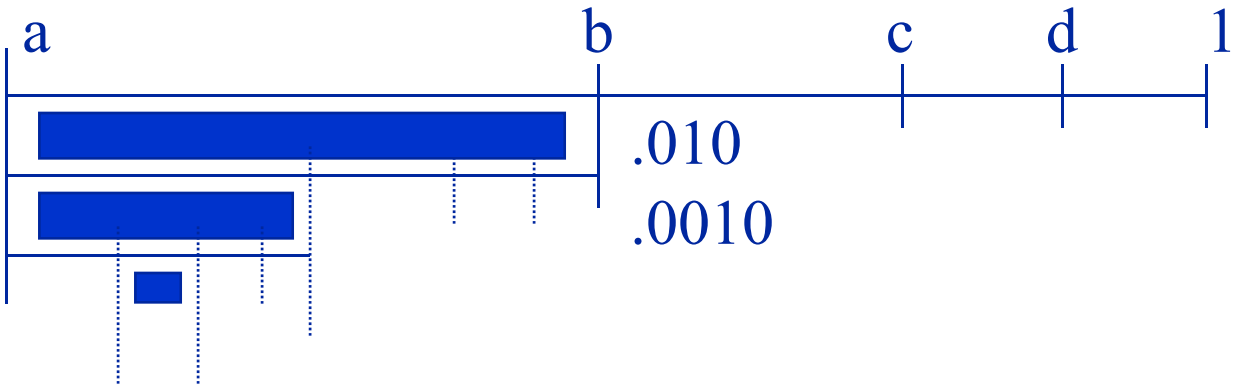


- This is an accumulative probabily function : $F(x) = \sum_{k=0}^{x} p(k) = p(0) + p(1) + \ldots + p(x)$
- Each "code point" is the probability sum of previous probabilities
- The length of the interval on the right side of the point is equal to probability of the symbol
- Intervals in our case: [0, 0.1) , [0.1, 0.11), ...

Arithmetic Coding (Example)

Successive Subdivision

of unit interval: *a a b ...*



New_Code = Current_Code + A*Pi

New_A = Current_A + Pi

A:interval length

Pi: probability

Arithmetic Coding Main Problems

- Needs ‘infinite precision’ for interval A
- Needs a multiplication for interval subdivision: A*Pe
- Results about 10% better than Huffman but more complicated

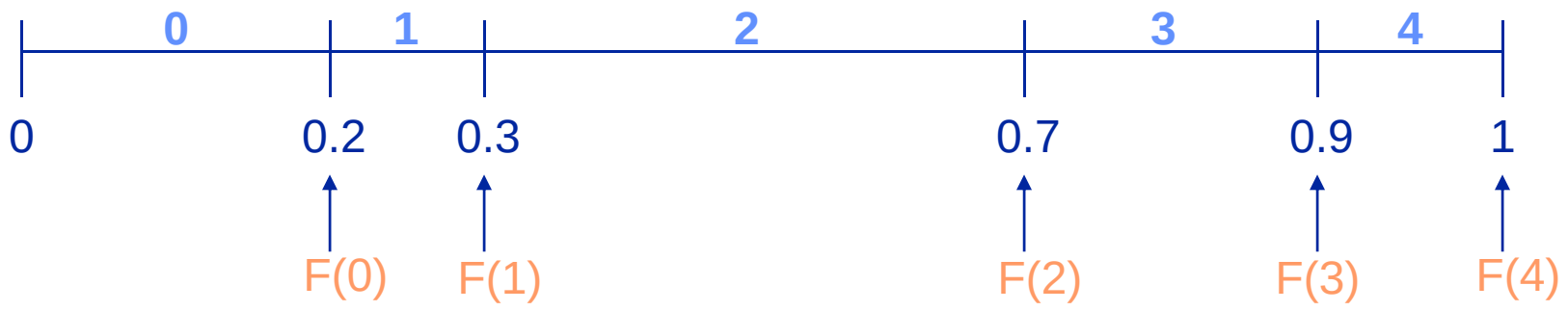
A Coding Example

- The accumulative distribution function of the random variable X with optional values {0,1,2,3,4} are:

X	P(x)	F(x)
0	0.2	0.2
1	0.1	0.2+0.1=0.3
2	0.4	0.2+0.1+0.4=0.7
3	0.2	0.2+0.1+0.4+0.2=0.9
4	0.1	0.2+0.1+0.4+0.2+0.1=1

A Basic coding and decoding algorithm

- We have n open segments [a,b), where n is the size of the alphabet we use:

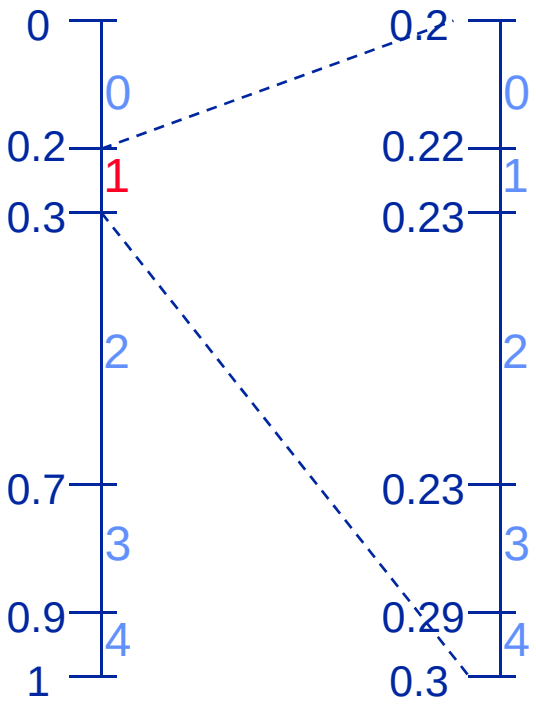


A Coding Example: 1/5

- We'll represent the "message" "1432" using the accumulative distribution function we have:

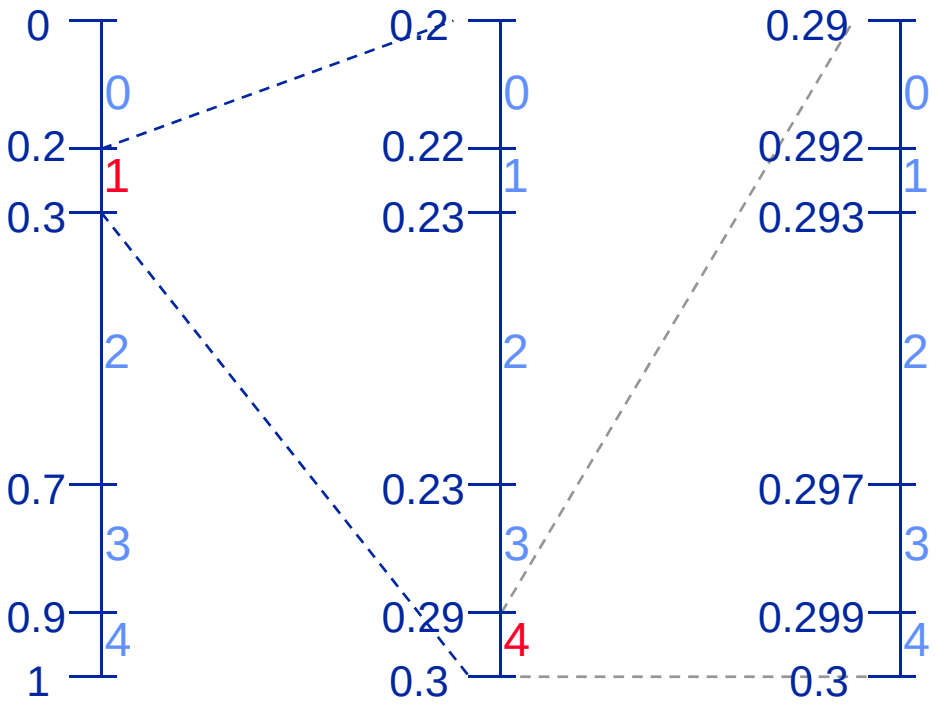
A Coding Example: 2/5

- We'll represent the "message" "1432" using the accumulative distribution function we have:
- First letter: 1



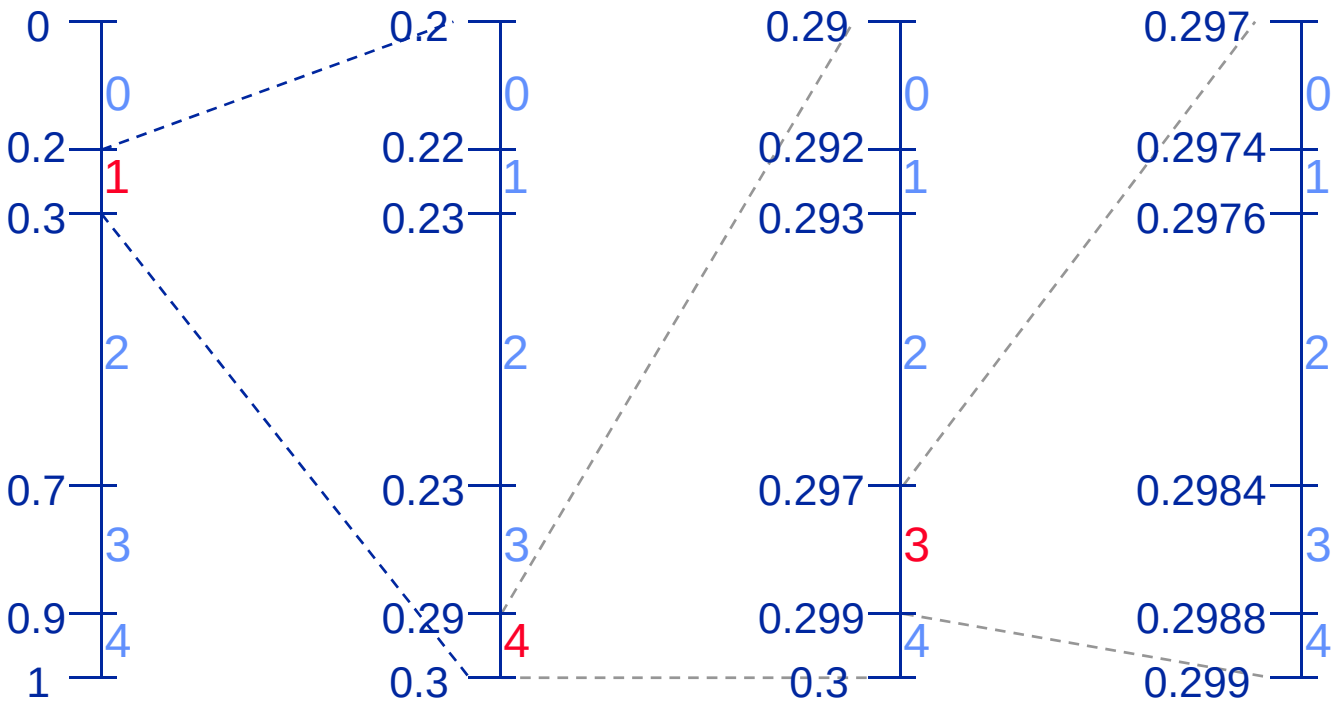
A Coding Example: 3/5

- We'll represent the "message" "1432" using the accumulative distribution function we have:
- Second letter: 4



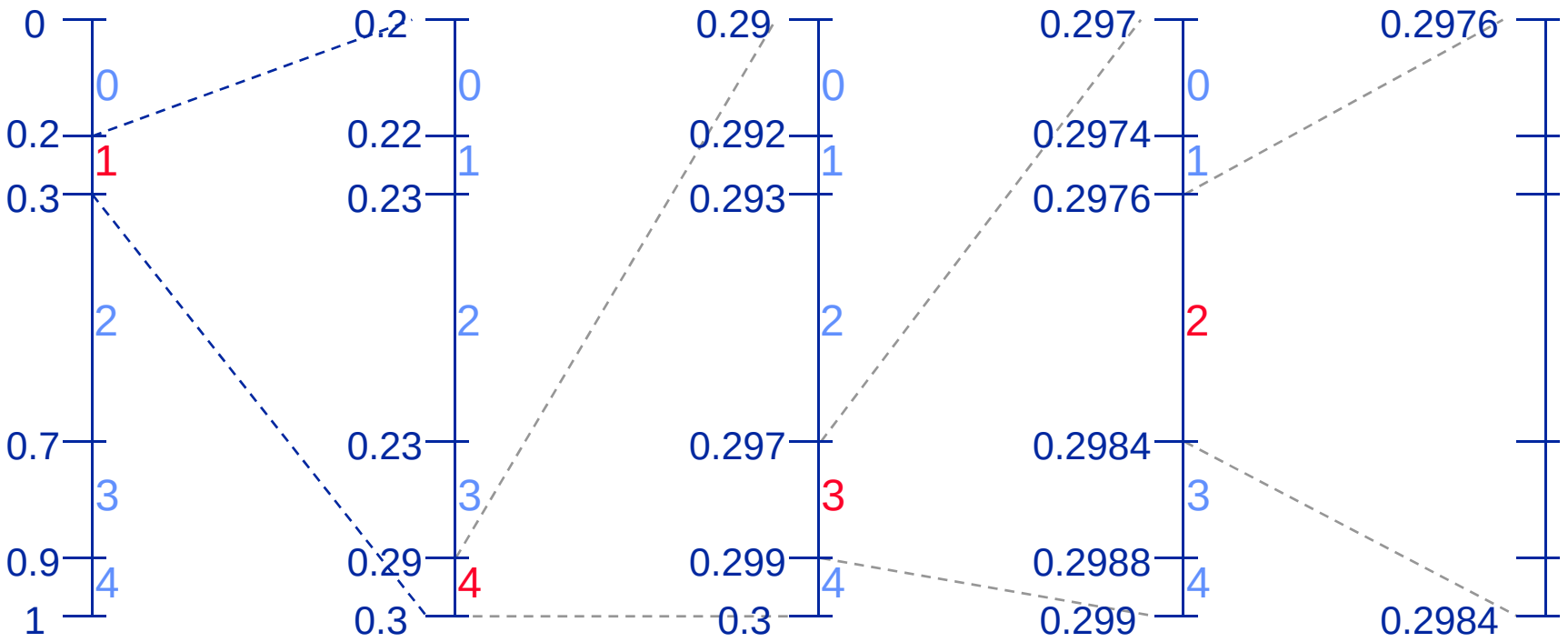
A Coding Example: 4/5

- We'll represent the "message" "1432" using the accumulative distribution function we have:
- Third letter: 3.



A Coding Example: 5/5

- We'll represent the "message" "1432" using the accumulative distribution function we have:
- Fouth letter: 2

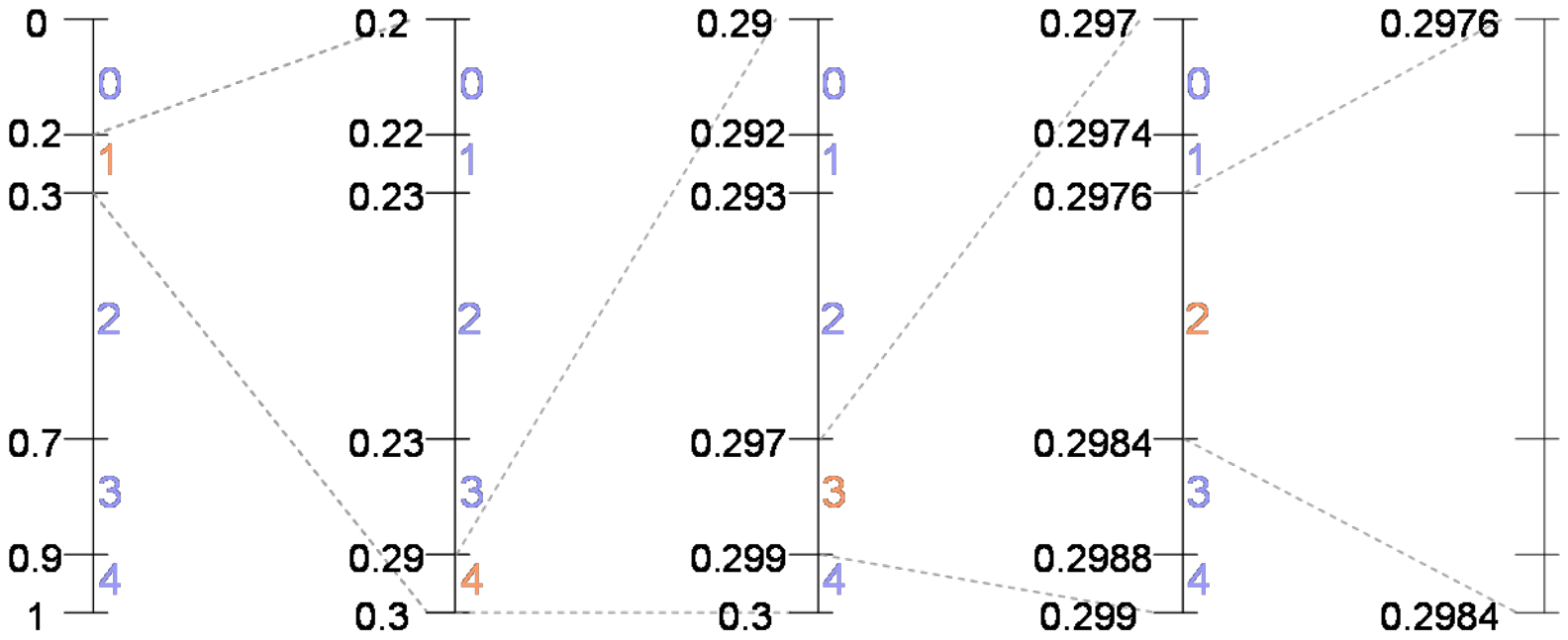


Choosing the coding word

- The string “1432” should be represented by a binary word in the boundary[0.2976,0.2984)
- For simplicity, lets choose the middle point: 0.298
- 0.298_{10}=0.$0100110001001001101110\ldots_2$
- It can be shown that we need $\lceil -\log p \rceil + 1$ bit to represent the exact message, where p is the boundary size, in our case: p= 0.0008, so we need 12 bits
- Since the two zeros at the end mean nothing we can use a shorter string: 0.0100110001

Decoding Procedure

We just need to “open” the boundary where 0.298 is in, the same way we encoded it, letter by letter.



Mathematical Formulation

- Through every step we are inside the boundary [*low*,*high*)
- We begin in the area [0,1) so we initialize with *low*=0, *high*=1
- Every step, we get x, and need to calculate the
- next boundary limits *low'*, *high'*
- *F(x) is the accumulative distribution function*

$$low' = low + (high - low) \cdot F(x-1)$$

$$high' = low + (high - low) \cdot F(x)$$

Pro's

- Adaptive : optional, even "on-the-fly", using symbols probability or other mathematical model
  - No need to send side information !
- The coding algorithm is not depended on the probability (or other) model
- Very efficient …

And Con's …

- Computational complexity: need floating point arithmetic
- Sensitive to BER
- Decoding can't be parallelized
- Not progressive: no output until the whole message is coded
- Message length or "end-of-message" sign are needed

Advanced Algorithms

- Progressive: we want to have the output start before the "end-of-message"
  - There are coding/decoding "predictive" algorithms
- Fixed point arithmetic
  - There are fixed point and table based methods

An excellent reference...

- By Ilan Sutskover,

can be found in the course web site