EE 6427 Video Signal Processing

Instructor: Dr. Chau Lap Pui

Room: S2-B2a-09

Email: elpchau@ntu.edu.sg

Tel: 67904239

***No lesson on 28 September 2018**

Topics

1. Compression fundamental
2. Motion compensation video coder
3. Motion estimation
4. Error control
5. 3D video
6. Wavelet
7. Bit rate control

Text Books

- Textbooks:

1. Y. Wang, J. Ostermann, and Y.-Q. Zhang, Video Processing and Communications, Prentice Hall, 2002.
2. Yun Q. Shi and Huifang Sun, Image and Video Compression for Multimedia Engineering: Fundamentals, Algorithms, and Standards, CRC Press, 2nd Edition (2008).

- Error Control part:

1. Y. Wang, J. Ostermann, and Y.-Q. Zhang, Video Processing and Communications (Chapter 14), Prentice Hall, 2002.
2. Yao Wang, Lecture Materials on “Error control,” http://eeweb.poly.edu/~yao/EL6123

References

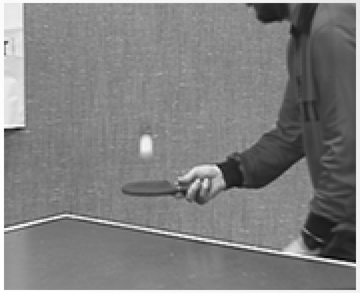
1. Iain E.G. Richardson, H.264 and MPEG-4 Video. Compression. Video Coding for Next-generation Multimedia, John Wiley & Sons, 2003
2. K.S. Thyagarajan, Still Image and Video Compression with MATLAB, Wiley, 2011
3. Oge Marques, Practical Image and Video Processing using MATLAB, Wiley, 2011
4. John W. Woods, Multidimensional Signal, Image, and Video Processing and Coding, Academic Press, 2012
5. [MPEG-1] ISO/IEC. IS 11172: Information technology - Coding of moving pictures and associated audio for digital storage media at up to about 1.5 mbit/s, 1993.
6. [MPEG-2 Video] ISO/IEC. IS 13818-2: Information technology - Generic coding of moving pictures and associated audio information, 1995.
7. [MPEG-4 Video] ISO/IEC. IS 14496-2: Information technology - coding of audio- visual objects, 1999.
8. ITU-T Recommendation H.264 & ISO/IEC 14496-10 (MPEG-4) AVC, Advanced Video Coding for Generic Audiovisual Services, version 3: 2005.
9. Digital Video Broadcasting (DVB), Frame Compatible Plano-Stereoscopic 3DTV (DVB-3DTV) DVB Document A154, February 2011

Video Compression Fundamental

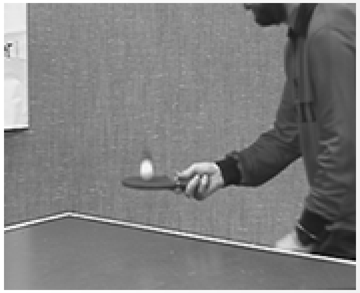
- For an uncompressed digitized DVB HD video signal with 1280 pixels times 720 lines (16:9), 8 bits per pixel for each colour component, and 50 frames per second, it requires a bandwidth of 1280x720x8x3x50=1105.92Mbits per second (bps) for transmission.
- For transmitting the video through a 100Mbps broadband network with an assured throughput of 20Mbps, it requires a compression ratio of at least 55:1 for real-time transmission.
- For storing one hour of the uncompressed video signal in harddisk, it requires 500G bytes disk space.
- Owing to these high compression requirements, the lossy compression techniques are much suitable compared with the lossless ones.

Video Compression Fundamental

- The most basic approach to compress a digital video signal is on a frame by frame basis. This approach achieves compression by exploiting the spatial, spectral and psychovisual redundancies of a single frame.
- The compression ratio is limited since it does not exploit the temporal redundancies between neighbor frames.
- Thus the interframe techniques are widely adopted by various video compression standards to reduce temporal redundancies of successive frames, in addition to the intraframe techniques.
- Among various inter/intra-frame compression techniques, the motion compensated transform coding technique is the most popular one, which is adopted by many video coding standards such as MPEG-1/2/4 and H.261/262/263/264/265, owing to its high compression efficiency.

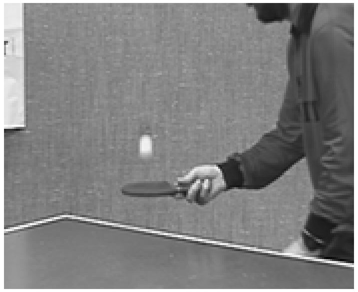


Frame 39



Frame 40

Frame 41



Animated Frame 39-41

Basics of Information Theory

- According to Shannon, the entropy of an information source S is defined as:

$$H(S) = \sum_i p_i \log_2 \frac{1}{p_i}$$

where p_i is the probability that symbol S_i in S will occur $\sum_i p_i = 1$.
$\log_2(1/p_i) = -\log_2(p_i)$ indicates the amount of information contained in S_i, i.e., the number of bits needed to code S_i.

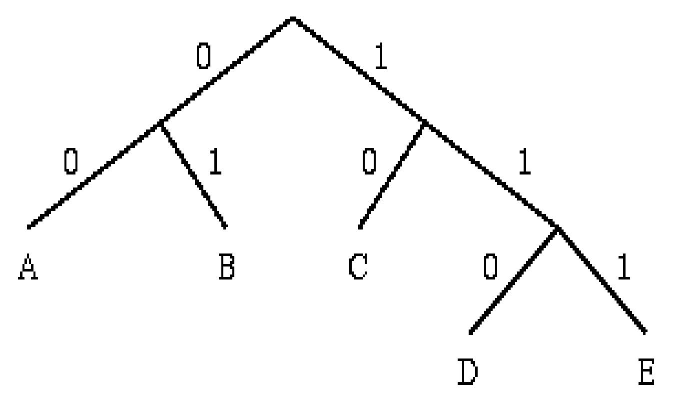
Higher probability symbols use fewer bits to represent it, that achieve compression. Shannon entropy tells us the theoretical lower bound.

Shannon-Fano Algorithm

- A simple example will be used to illustrate the algorithm:

Symbol	A	B	C	D	E
Count	15	7	6	6	5

- Encoding for the Shannon-Fano Algorithm:
- A top-down approach
  1. Sort symbols according to their frequencies/probabilities, e.g., ABCDE.
  2. Recursively divide into two parts, each with approx. same number of counts.



Shannon-Fano Algorithm

Symbol	Count	$\log_2(1/p_i)$	Code	Subtotal (# of bits)
A	15	1.38	00	30
B	7	2.48	01	14
C	6	2.70	10	12
D	6	2.70	110	18
E	5	2.96	111	15

Total count 39, p_A=15/39

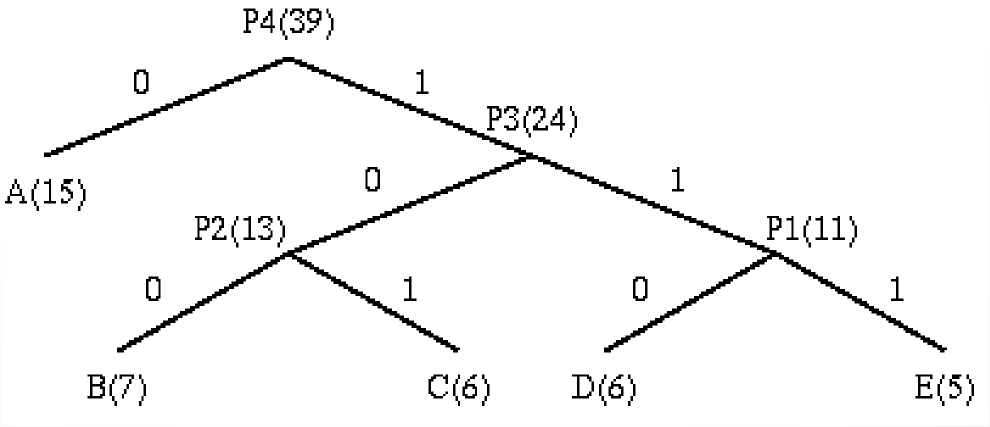
TOTAL (# of bits): 89

Huffman Coding

- **Encoding for Huffman Algorithm:**
- A bottom-up approach
- 1. Initialization: Put all nodes in an OPEN list, keep it sorted at all times (e.g., ABCDE).

Huffman Coding

- 2. Repeat until the OPEN list has only one node left:
  - From OPEN pick two nodes having the lowest frequencies/ probabilities, create a parent node of them.
  - Assign the sum of the children's frequencies/probabilities to the parent node and insert it into OPEN.
  - Assign code 0, 1 to the two branches of the tree, and delete the children from OPEN.



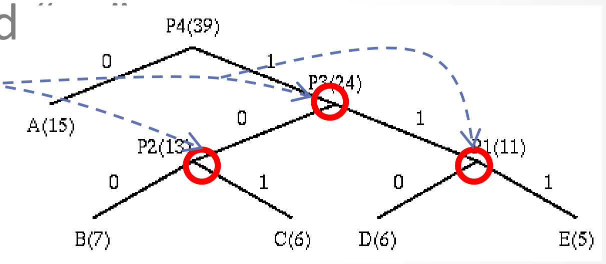
Huffman Coding

Symbol	Count	$\log_2(1/p_i)$	Code	Subtotal (# of bits)
A	15	1.38	0	15
B	7	2.48	100	21
C	6	2.70	101	18
D	6	2.70	110	18
E	5	2.96	111	15

TOTAL (# of bits): 87

Discussions

- Decoding for the above two algorithms is trivial as long as the coding table (the statistics) is sent before the data. (There is a bit overhead for sending this, negligible if the data file is big.)
- Unique Prefix Property: no code is a prefix to any other code (all symbols are at the leaf nodes). E.g. {9, 59, 55} has the prefix property, but {9, 5, 59, 55} does not, because “5” is a prefix of both “59” and
  --> great for decoder, unambiguous.
- No symbols is at parent node

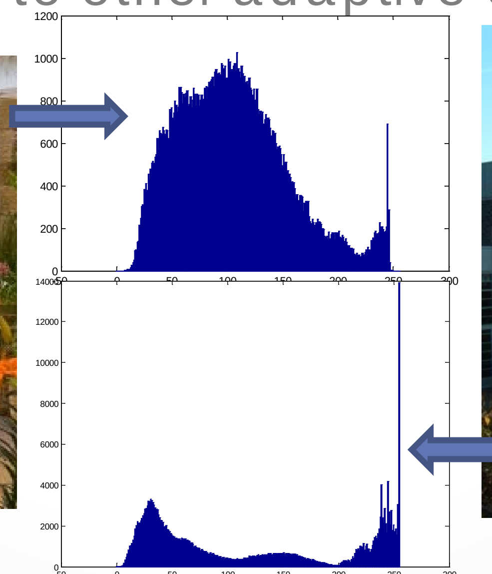


- If prior statistics are available and accurate, then Huffman coding is very good.
- In the above example: entropy = (15 x 1.38 + 7 x 2.48 + 6 x 2.7 + 6 x 2.7 + 5 x 2.96) / 39 = 85.26 / 39 = 2.19
- Number of bits needed for Huffman coding is: 87 / 39 = 2.23
- Number of bits needed for Shannon-Fano coding is: 89 / 39 = 2.28

Adaptive Huffman Coding

- **Motivations:**
  - o The previous algorithms require the statistical knowledge which is often not available (e.g., live audio, video).
  - o Even when it is available, it could be a heavy overhead especially when many tables had to be sent.
- The solution is to use adaptive algorithms. As an example, the Adaptive Huffman Coding is examined below. The idea is however applicable to other adaptive compression algorithms.

Indoor



Outdoor

Adaptive Huffman Coding

```
ENCODER
-------

Initialize_model();
while ((c = getc (input)) != eof)
  {
    encode (c, output);
    update_model (c);
  }
```

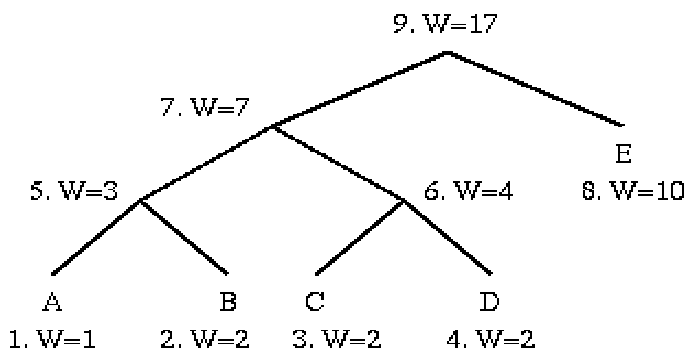
```
DECODER
-------

Initialize_model();
while ((c = decode (input)) != eof)
  {
    putc (c, output);
    update_model (c);
  }
```

- The key is to have both encoder and decoder to use exactly the same *initialization* and *update_model* routines.

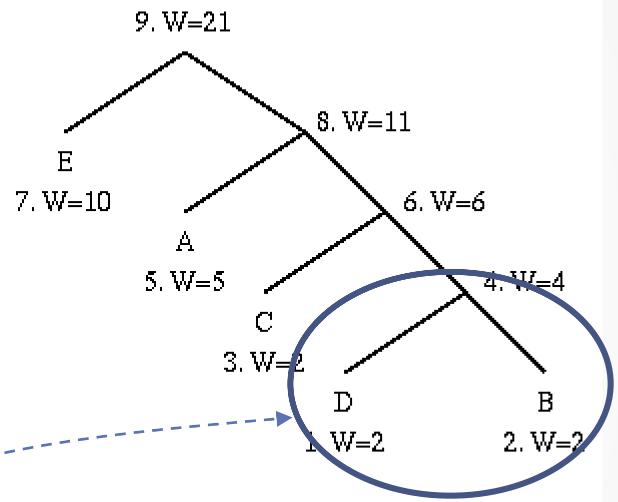
Adaptive Huffman Coding

- *update_model* does two things: (a) increment the count, (b) update the Huffman tree.
  - During the updates, the Huffman tree will be maintained its *sibling property*, i.e. the nodes (internal and leaf) are arranged in order of increasing weights (see figure).
  - When *swapping* is necessary, the farthest node with weight W is swapped with the node whose weight has just been increased to W+1.
    Note: If the node with weight W has a subtree beneath it, then the subtree will go with it.
  - The Huffman tree could look very different after node swapping, e.g., in the third tree, node A is again swapped and becomes the #5 node. It is now encoded using only 2 bits.

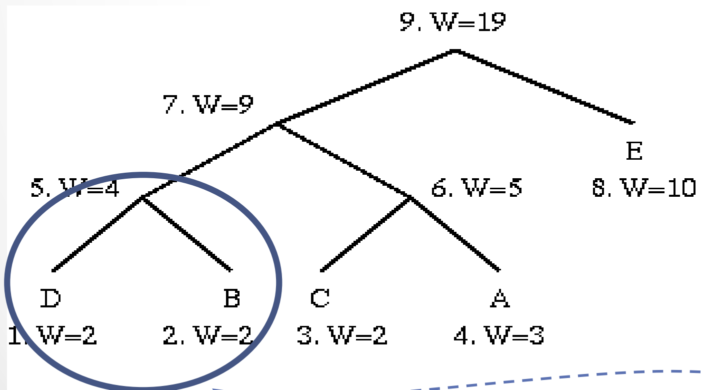
Adaptive Huffman Coding



A Huffman Tree



After A was incremented two more times



After a node switch (A was incremented twice)

- **Note:** Code for a particular symbol changes during the adaptive coding process.
- Maintain Huffman coding property after swapping.

Lempel-Ziv-Welch Algorithm

- **Motivation:** Suppose we want to encode the Webster's English dictionary which contains about 159,000 entries. Why not just transmit each word as an 18 bit number? (2^{18}=262144)
- **Problems:** (a) Too many bits, (b) everyone needs a dictionary, (c) only works for English text.
- Solution: Find a way to build the dictionary adaptively.
- Original methods due to Ziv and Lempel in 1977 and 1978. Terry Welch improved the scheme in 1984 (called LZW compression).

Lempel-Ziv-Welch Algorithm

LZW Compression Algorithm:

```
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;
    }
```

- Original LZW used dictionary with 4K entries, first 256 (0-255) are ASCII codes.

Lempel-Ziv-Welch Algorithm

- **Example:** Input string is "^WED^WE^WEE^WEB^WET"

w	*k*	*Output*	*Index*	*Symbol*
NIL	^			
^	W	^	256	^W
W	E	W	257	WE
E	D	E	258	ED
D	^	D	259	D^
^	W			
^W	E	256	260	^WE
E	^	E	261	E^
^	W			
^W	E			
^WE	E	260	262	^WEE
E	^			
E^	W	261	263	E^W
W	E			
WE	B	257	264	WEB
B	^	B	265	B^
^	W			
^W	E			
^WE	T	260	266	^WET
T	EOF	T		

LZW Compression Algorithm:

```
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;
    }
```

- Dictionary contains Index and Symbol.
- A 19-symbol input has been reduced to 7-symbol plus 5-code output. Each code/symbol will need more than 8 bits, say 9 bits.

Lempel-Ziv-Welch Algorithm

LZW Decompression Algorithm:

```
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;
    }
```

Lempel-Ziv-Welch Algorithm

- **Example:** Input string is "^WED<256>E<260><261><257>B<260>T".

w	k	Output	Index	Symbol
	^	^		
^	W	W	256	^W
W	E	E	257	WE
E	D	D	258	ED
D	<256>	^W	259	D^
<256>	E	E	260	^WE
E	<260>	^WE	261	E^
<260>	<261>	E^	262	^WEE
<261>	<257>	WE	263	E^W
<257>	B	B	264	WEB
B	<260>	^WE	265	B^
<260>	T	T	266	^WET

LZW Decompression Algorithm:

```
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;
     }
```

w	k	Output	Index	Symbol
NIL	^			
^	W	^	256	^W
W	E	W	257	WE
E	D	E	258	ED
D	^	D	259	D^
^	W			
^W	E	256	260	^WE

- Lempel-Ziv-Welch is a dictionary-based compression method. It maps a variable number of symbols to a fixed length code.

Arithmetic Coding

- Arithmetic coding don't use the idea of replacing an input symbol with a specific codeword. Instead, it takes input symbols and replaces it with one floating point output number.
- The longer and the more complex the message, the more bits are needed in the output number.
- The output from arithmetic coding process is a single number less than 1 and greater than or equal to 0. This single number can be uniquely decoded to create the exact stream of symbols that went into its construction. In order to construct the output number, the symbols being encoded have to have a set probabilities assigned to them.

Arithmetic Coding

- For example, if I was going to encode the random message "BILL GATES", I would have a probability distribution that looks like this:

Character	Probability
SPACE	1/10
A	1/10
B	1/10
E	1/10
G	1/10
I	1/10
L	2/10
S	1/10
T	1/10

Arithmetic Coding

- As the character probabilities are known, the individual symbols need to be assigned a range. It doesn't matter which characters are assigned which segment of the range, as long as it is done in the same manner by encoder and decoder.

Character	Probability	Interval (Range)
SPACE	1/10	[0.00 - 0.10)
A	1/10	[0.10 - 0.20)
B	1/10	[0.20 - 0.30)
E	1/10	[0.30 - 0.40)
G	1/10	[0.40 - 0.50)
I	1/10	[0.50 - 0.60)
L	2/10	[0.60 - 0.80)
S	1/10	[0.80 - 0.90)
T	1/10	[0.90 - 1.00)

Arithmetic Coding

Character	Probability	Interval (Range)
SPACE	1/10	[0.00 - 0.10)
A	1/10	[0.10 - 0.20)
B	1/10	[0.20 - 0.30)
E	1/10	[0.30 - 0.40)
G	1/10	[0.40 - 0.50)
I	1/10	[0.50 - 0.60)
L	2/10	[0.60 - 0.80)
S	1/10	[0.80 - 0.90)
T	1/10	[0.90 - 1.00)

New Character	Low value	High Value
	0.0	1.0
B	0.2	0.3
I	0.25	0.26
L	0.256	0.258
L	0.2572	0.2576
SPACE	0.25720	0.25724
G	0.257216	0.257220
A	0.2572164	0.2572168
T	0.25721676	0.2572168
E	0.257216772	0.257216776
S	0.2572167752	0.2572167756

- The final low value, 0.2572167752 representing the message "BILL GATES" using this encoding scheme.

Arithmetic Coding

Decoded Number	Output Symbol	Low	High	Interval
0.2572167752	B	0.2	0.3	0.1
0.572167752	I	0.5	0.6	0.1
0.72167752	L	0.6	0.8	0.2
0.6083876	L	0.6	0.8	0.2
0.041938	SPACE	0.0	0.1	0.1
0.41938	G	0.4	0.5	0.1
0.1938	A	0.1	0.2	0.1
0.938	T	0.9	1.0	0.1
0.38	E	0.3	0.4	0.1
0.8	S	0.8	0.9	0.1
0.0				

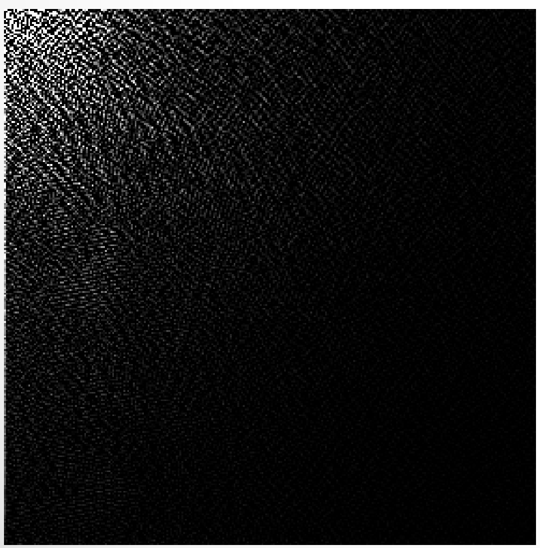
New Decoded Number=(Decoded Number-Low)/Interval

Character	Probability	Interval (Range)
SPACE	1/10	[0.00 - 0.10)
A	1/10	[0.10 - 0.20)
B	1/10	[0.20 - 0.30)
E	1/10	[0.30 - 0.40)
G	1/10	[0.40 - 0.50)
I	1/10	[0.50 - 0.60)
L	2/10	[0.60 - 0.80)
S	1/10	[0.80 - 0.90)
T	1/10	[0.90 - 1.00)

Image Compression -- JPEG

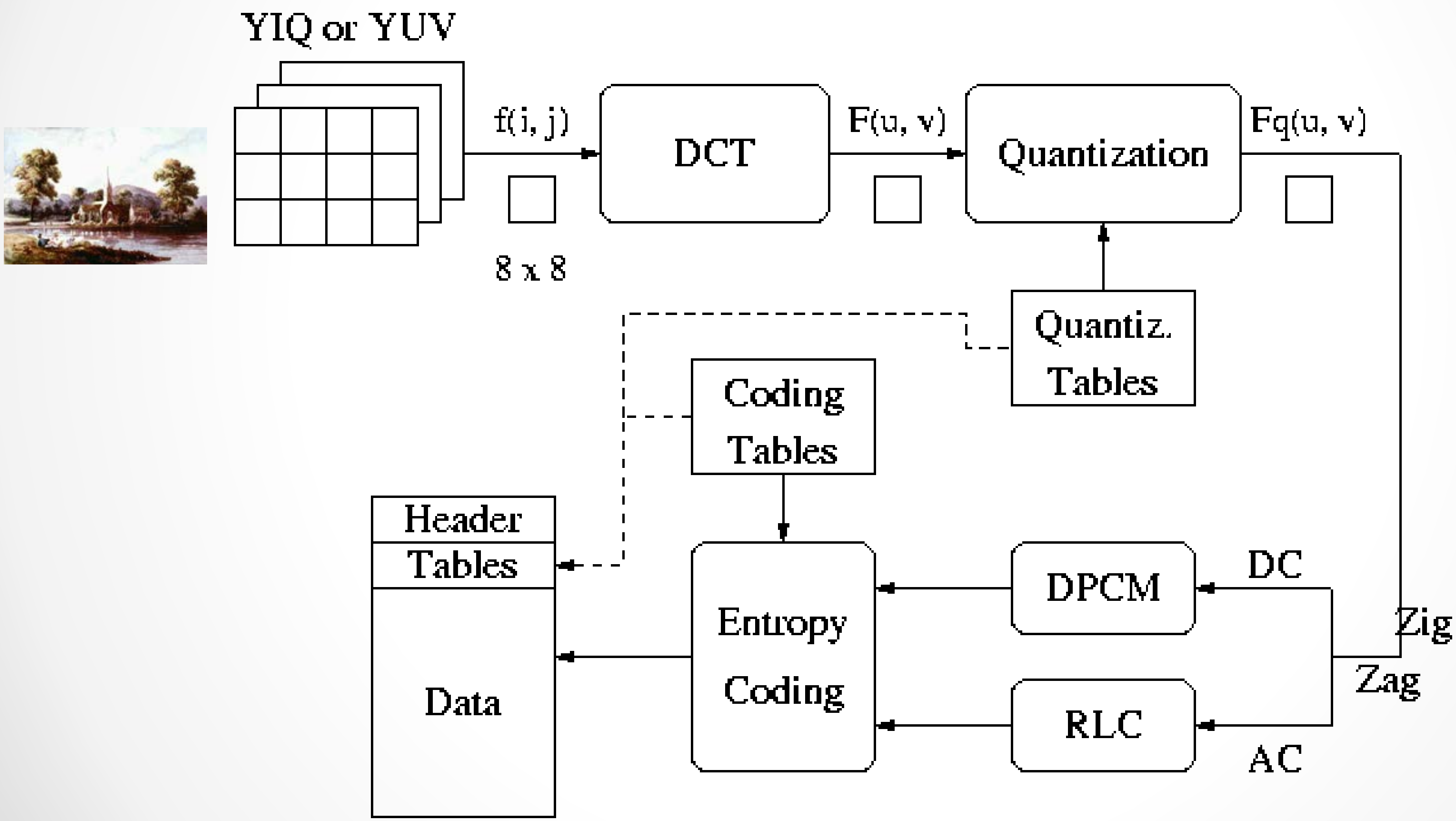
- **What is JPEG?**
- "Joint Photographic Expert Group". Voted as international standard in 1992.
- Works with color and grayscale images, e.g., satellite, medical, ...
- **Motivation**
- The *compression ratio* of lossless methods (e.g., Huffman, Arithmetic, LZW) is not high enough for image and video compression, especially when the distribution of pixel values is relatively flat.
- JPEG uses *transform coding*, it is largely based on the following observations:

Image Compression -- JPEG



- Observation 1: A large majority of useful image contents change relatively slowly across images, i.e., it is unusual for intensity values to alter up and down several times in a small area, for example, within an 8 x 8 image block. Translate this into the spatial frequency domain, it says that, generally, lower spatial frequency components contain more information than the high frequency components which often correspond to less useful details and noises.
- Observation 2: Psychophysical experiments suggest that humans are less likely to notice the loss of higher spatial frequency components than loss of lower frequency components.

Encoder & Decoder

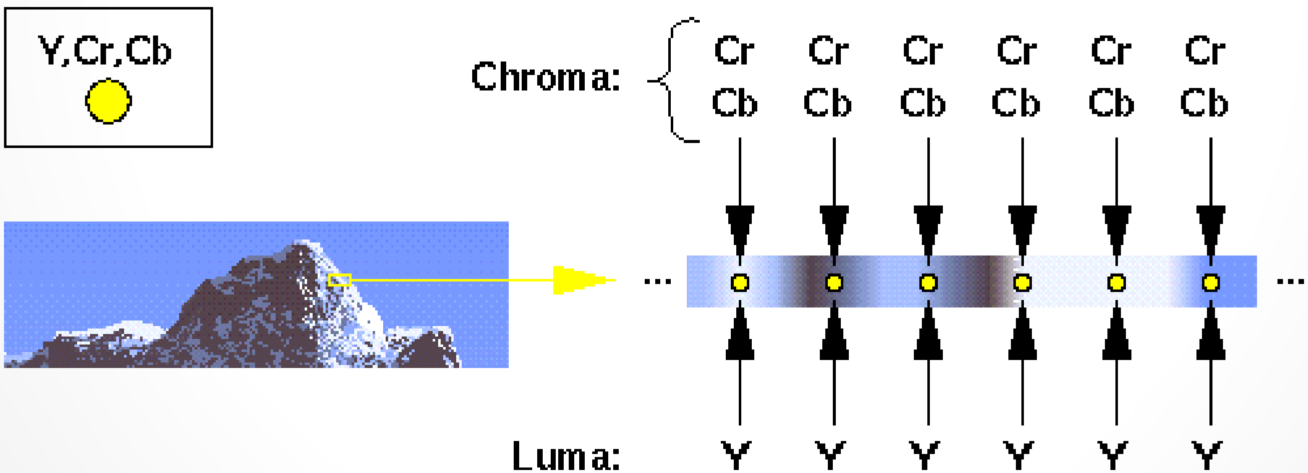


Major Steps

- Sampling
- Discrete Cosine Transformation (DCT)
- Quantization
- Zigzag Scanning
- DPCM on DC component
- RLE on AC Components
- Entropy Coding

Sampling Pattern Definitions

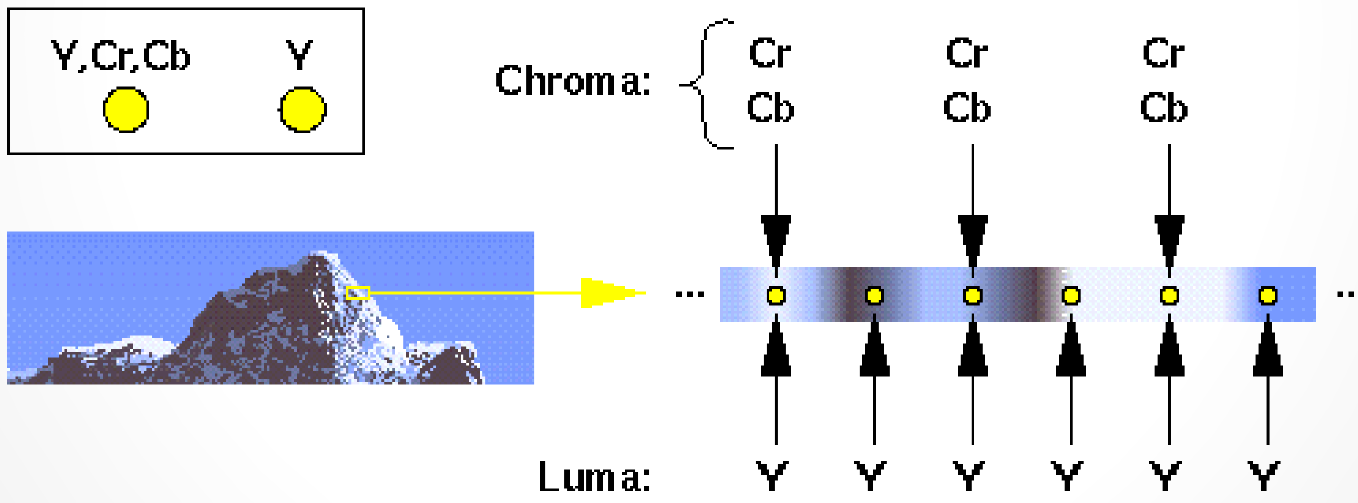
- **4:4:4 Sampling --** Some of the diagrams indicate 4:4:4 sampling. This video industry terminology simply means that each of your 3 components is sampled at every pixel. Here's an example with Y, Cr, and Cb.



4:4:4 Sampling

Sampling Pattern Definitions

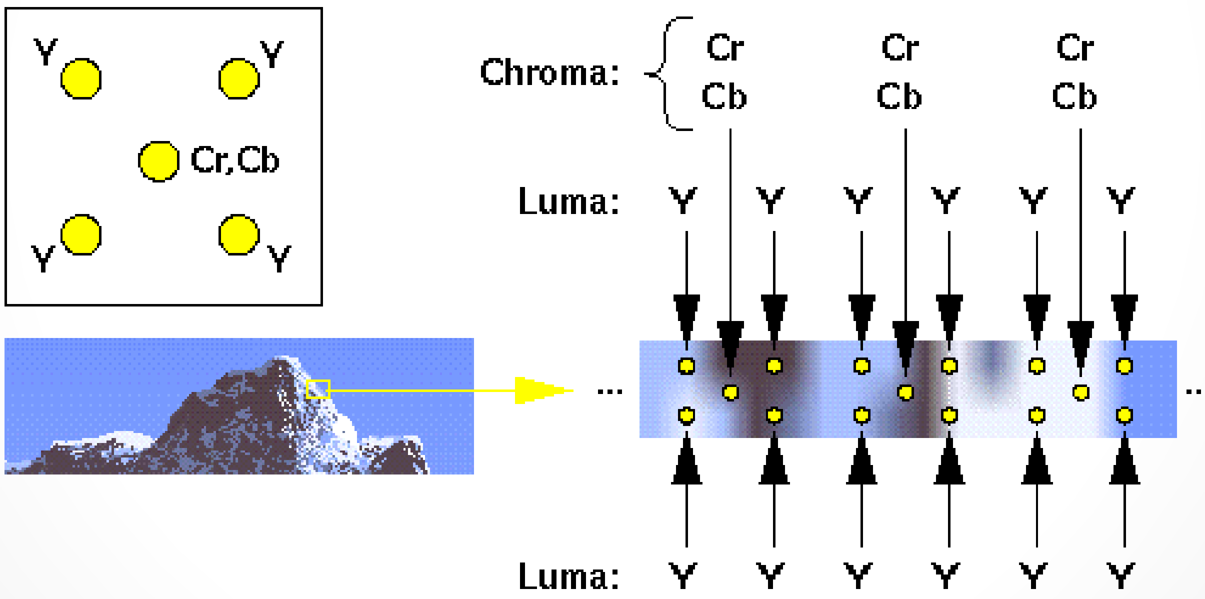
- **4:2:2 Sampling --** 4:2:2 sampling is described by ITU-R BT.601-4 (Rec. 601). It means that for every four pixels, we get four luma samples (four Y's) but only two chroma samples (two samples of Cr and Cb respectively, which together determine the chroma), like this:



Rec. 601 4:2:2 Sampling

Sampling Pattern Definitions

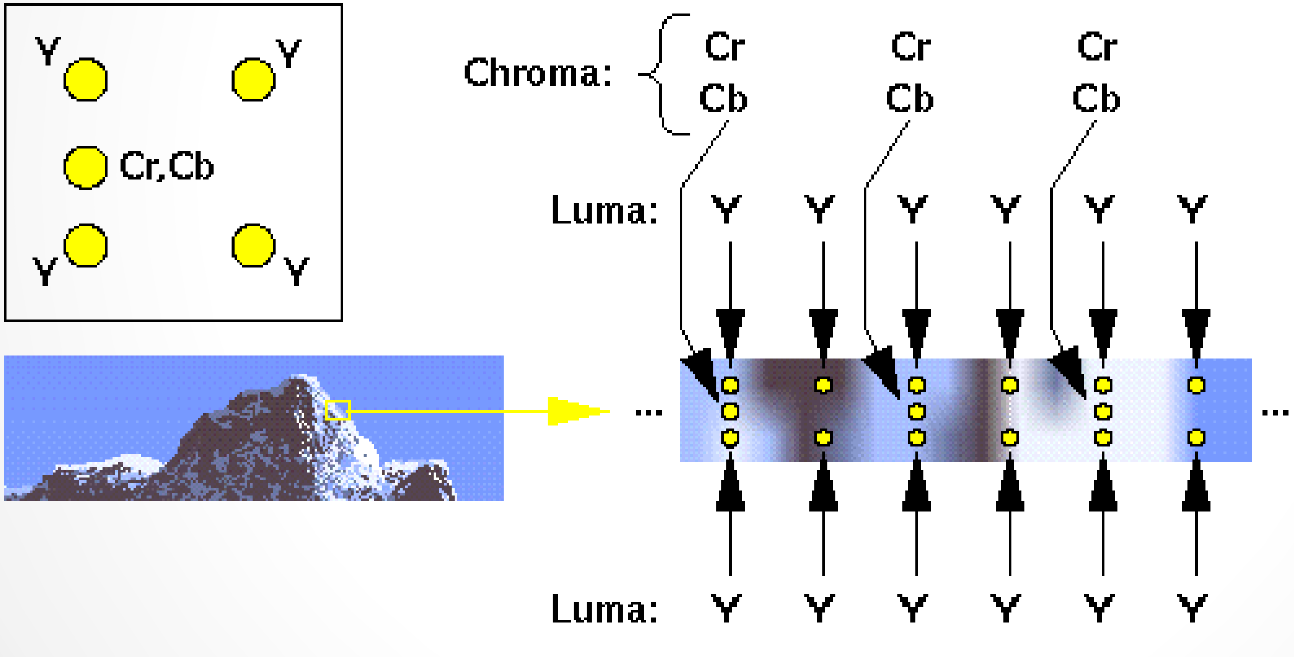
- **4:2:0 Sampling --** In this case, there is one chroma sample (one pair of Cr and Cb) for every four luminance samples (Y). The MPEG-1 and H.261 video compression standards use this 4:2:0 sampling pattern:



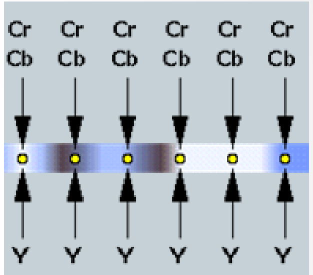
MPEG I, H.261 4:2:0 Sampling

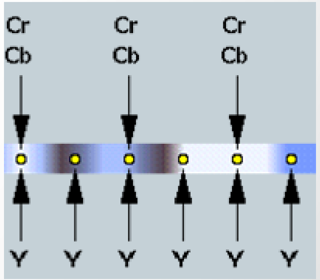
Sampling Pattern Definitions

- In addition to supporting 4:4:4 and Rec.601 4:2:2, the MPEG–2 video compression standard supports this 4:2:0 sampling pattern:



MPEG II 42:0 Sampling



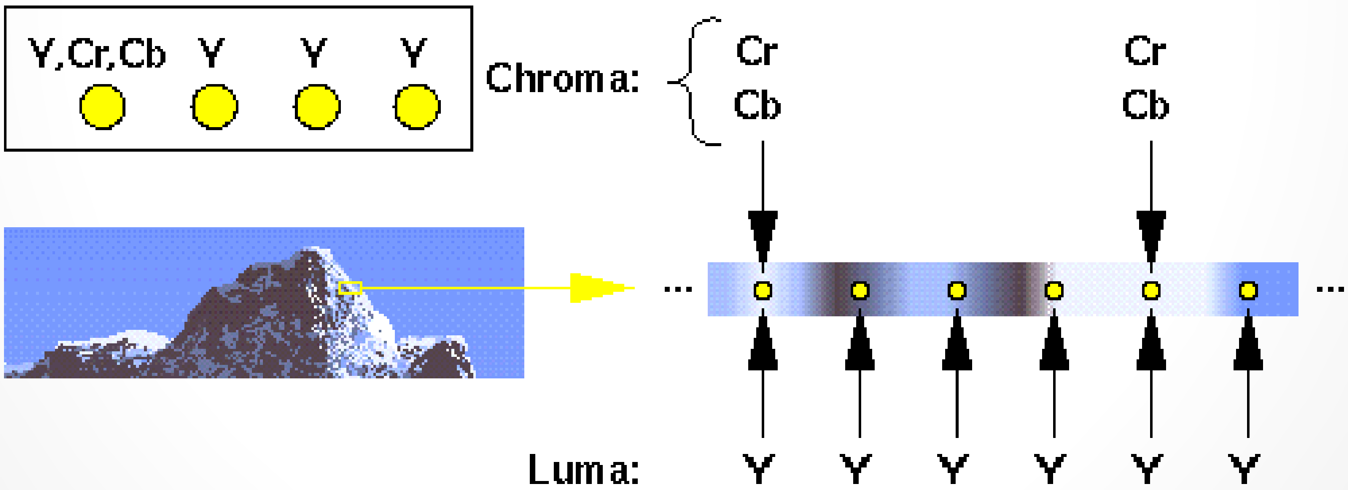


Sampling Pattern Definitions

- Why the MPEG committee changed it? Perhaps they wanted to make conversion from Rec.601 4:2:2 to MPEG-2 4:2:0 less computationally expensive.

Sampling Pattern Definitions

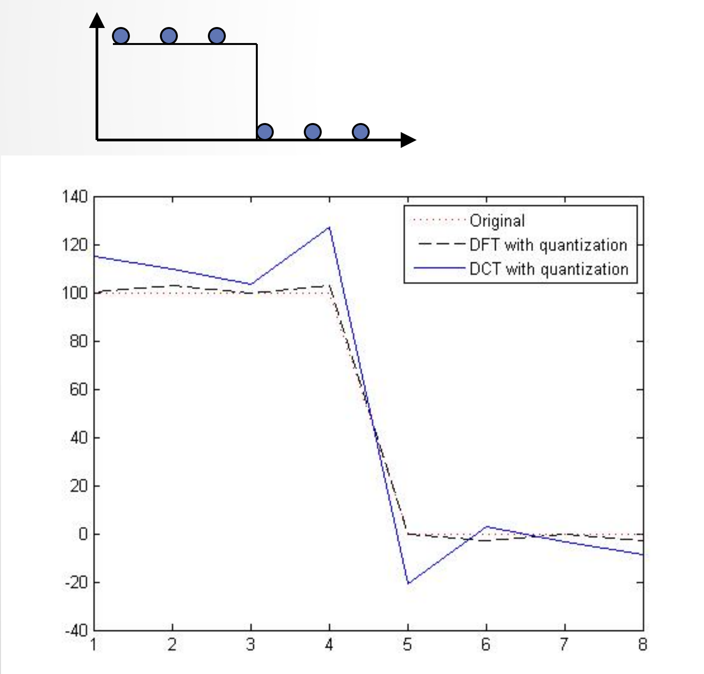
- **4:1:1 Sampling --** Another sub-sampling method with similar "compression" as 4:2:0 is 4:1:1. The 525-line DVC compression standard, and both the 525- and 625-line variants of the DVCPRO compression standard, use this pattern:

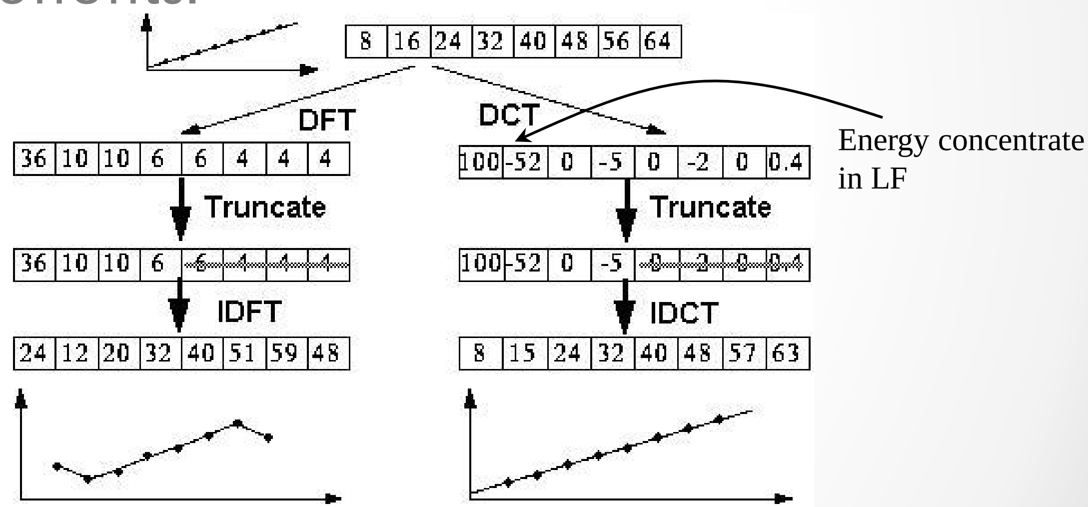


525-Line DVC, 525/625-Line DVCPRO 4:1:1 Sampling

DCT

- Why DCT not DFT? -- DCT is like DFT, but can approximate linear signals well with few coefficients. DFT use to model the discontinuity of the samples, energy will be located at high frequency components.

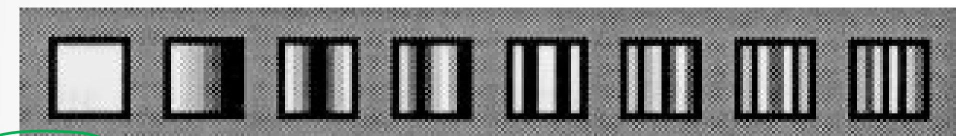




Computing the 1-D Discrete Cosine Transform

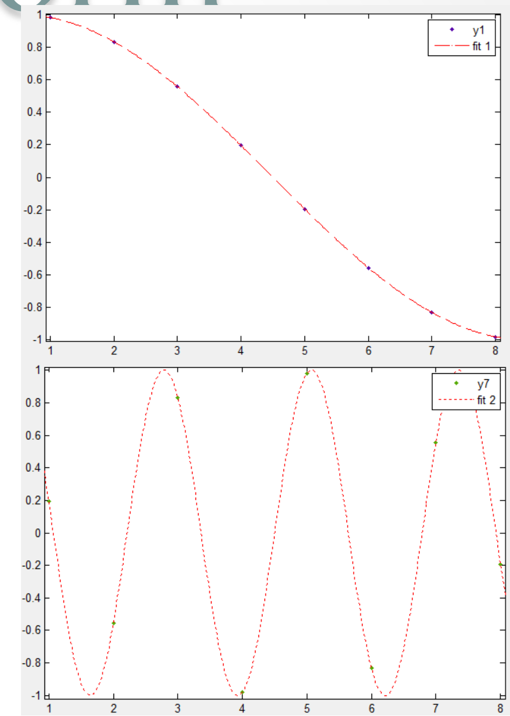
$$Y(k)=\sum_{i=0}^{7} x(i)\cos\left(\frac{(2i+1)k\pi}{16}\right) \qquad k=0,..,7$$

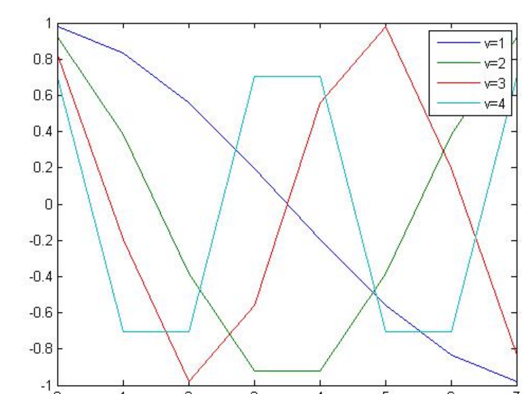
- Y(0) takes the average of the pixels.
- For Y(1), calculate low frequency. Y1 will be large if signal gradually change.



- Y(7) tends to emphasize the high frequency elements of the pixels (difference between pixels).

$$\begin{pmatrix} Y(0)\\ Y(1)\\ Y(2)\\ Y(3)\\ Y(4)\\ Y(5)\\ Y(6)\\ Y(7)\end{pmatrix}=\begin{pmatrix}
1 & 1 & 1 & 1 & 1 & 1 & 1 & 1\\
\cos(\frac{\pi}{16}) & \cos(\frac{3\pi}{16}) & \cos(\frac{5\pi}{16}) & \cos(\frac{7\pi}{16}) & -\cos(\frac{7\pi}{16}) & -\cos(\frac{5\pi}{16}) & -\cos(\frac{3\pi}{16}) & -\cos(\frac{\pi}{16})\\
\cos(\frac{\pi}{8}) & \cos(\frac{3\pi}{8}) & -\cos(\frac{3\pi}{8}) & -\cos(\frac{\pi}{8}) & -\cos(\frac{\pi}{8}) & -\cos(\frac{3\pi}{8}) & \cos(\frac{3\pi}{8}) & \cos(\frac{\pi}{8})\\
\cos(\frac{3\pi}{16}) & -\cos(\frac{7\pi}{16}) & -\cos(\frac{\pi}{16}) & -\cos(\frac{5\pi}{16}) & \cos(\frac{5\pi}{16}) & \cos(\frac{\pi}{16}) & \cos(\frac{7\pi}{16}) & -\cos(\frac{3\pi}{16})\\
\cos(\frac{\pi}{4}) & -\cos(\frac{\pi}{4}) & -\cos(\frac{\pi}{4}) & \cos(\frac{\pi}{4}) & \cos(\frac{\pi}{4}) & -\cos(\frac{\pi}{4}) & -\cos(\frac{\pi}{4}) & \cos(\frac{\pi}{4})\\
\cos(\frac{5\pi}{16}) & -\cos(\frac{\pi}{16}) & \cos(\frac{7\pi}{16}) & \cos(\frac{3\pi}{16}) & -\cos(\frac{3\pi}{16}) & -\cos(\frac{7\pi}{16}) & \cos(\frac{\pi}{16}) & -\cos(\frac{5\pi}{16})\\
\cos(\frac{3\pi}{8}) & -\cos(\frac{\pi}{8}) & \cos(\frac{\pi}{8}) & -\cos(\frac{3\pi}{8}) & -\cos(\frac{3\pi}{8}) & \cos(\frac{\pi}{8}) & -\cos(\frac{\pi}{8}) & \cos(\frac{3\pi}{8})\\
\cos(\frac{7\pi}{16}) & -\cos(\frac{5\pi}{16}) & \cos(\frac{3\pi}{16}) & -\cos(\frac{\pi}{16}) & \cos(\frac{\pi}{16}) & -\cos(\frac{3\pi}{16}) & \cos(\frac{5\pi}{16}) & -\cos(\frac{7\pi}{16})
\end{pmatrix}\times\begin{pmatrix} x(0)\\ x(1)\\ x(2)\\ x(3)\\ x(4)\\ x(5)\\ x(6)\\ x(7)\end{pmatrix}$$



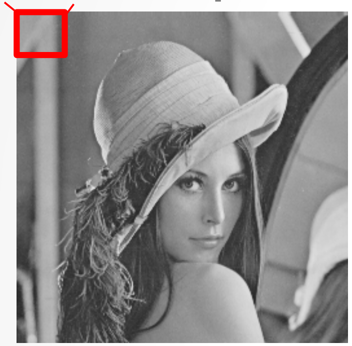


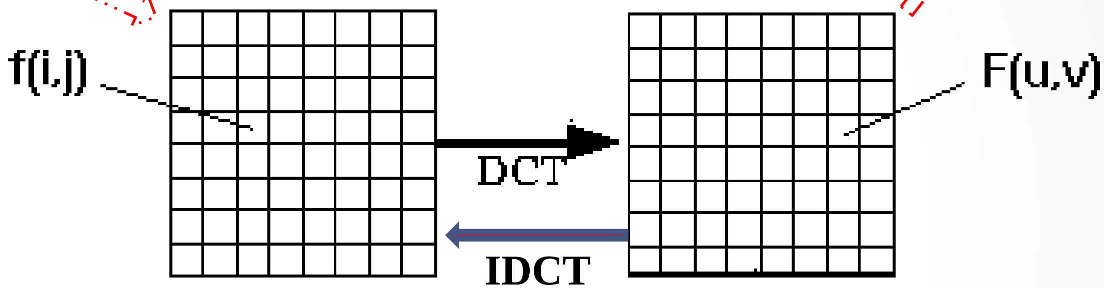
2-D Discrete Cosine Transformation

76	68	65	65	65	65	68	73
76	68	66	66	64	65	68	73
75	67	66	66	64	65	68	72
74	68	65	65	64	65	68	73
73	67	65	65	64	65	68	73
73	67	65	65	64	65	67	73
73	70	66	63	63	66	70	73
73	70	66	63	63	66	70	73

-483.1250	1.7102	25.5989	-0.2148	11.3750	3.1852	3.3324	-0.4426
3.5185	2.2448	1.1681	1.8343	0.1998	0.6538	-0.3247	0.2546
-0.2590	0.4080	0.3384	-0.1283	0.8562	0.1920	0.2866	0.3167
0.2695	-0.3552	0.2529	0.6294	0.3285	-1.0440	-0.1421	0.1350
-0.3750	-0.7855	0.4339	0.1022	-0.3750	-0.6576	-0.5856	-0.0507
-0.1464	0.0721	0.0876	0.4864	0.3698	-0.3669	-0.2240	0.3948
-0.2986	0.0187	-0.4634	0.2122	-0.2194	0.0268	0.1616	-0.0938
-0.2979	-0.2150	-0.5027	0.0962	0.1672	0.6272	0.1019	0.4927

- From spatial domain to frequency domain:





- A reversible, linear transform maps the image f(i,j) into transform coefficients F(u,v), then quantized & coded
- For most natural images, a significant number of coefficients have small magnitudes and can be coarsely quantized or discarded with little distortion ---> compression
- Hence: DCT --> good compromise between information packing and computational complexity

2-D Discrete Cosine Transform

$$\begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \cos(\frac{\pi}{16}) & \cos(\frac{3\pi}{16}) & \cos(\frac{5\pi}{16}) & \cos(\frac{7\pi}{16}) & -\cos(\frac{7\pi}{16}) & -\cos(\frac{5\pi}{16}) & -\cos(\frac{3\pi}{16}) & -\cos(\frac{\pi}{16}) \\ \cos(\frac{\pi}{8}) & \cos(\frac{3\pi}{8}) & -\cos(\frac{3\pi}{8}) & -\cos(\frac{\pi}{8}) & -\cos(\frac{\pi}{8}) & -\cos(\frac{3\pi}{8}) & \cos(\frac{3\pi}{8}) & \cos(\frac{\pi}{8}) \\ \cos(\frac{3\pi}{16}) & -\cos(\frac{7\pi}{16}) & -\cos(\frac{\pi}{16}) & -\cos(\frac{5\pi}{16}) & \cos(\frac{5\pi}{16}) & \cos(\frac{\pi}{16}) & \cos(\frac{7\pi}{16}) & -\cos(\frac{3\pi}{16}) \\ \cos(\frac{\pi}{4}) & -\cos(\frac{\pi}{4}) & -\cos(\frac{\pi}{4}) & \cos(\frac{\pi}{4}) & \cos(\frac{\pi}{4}) & -\cos(\frac{\pi}{4}) & -\cos(\frac{\pi}{4}) & \cos(\frac{\pi}{4}) \\ \cos(\frac{5\pi}{16}) & -\cos(\frac{\pi}{16}) & \cos(\frac{7\pi}{16}) & \cos(\frac{3\pi}{16}) & -\cos(\frac{3\pi}{16}) & -\cos(\frac{7\pi}{16}) & \cos(\frac{\pi}{16}) & -\cos(\frac{5\pi}{16}) \\ \cos(\frac{3\pi}{8}) & -\cos(\frac{\pi}{8}) & \cos(\frac{\pi}{8}) & -\cos(\frac{3\pi}{8}) & -\cos(\frac{3\pi}{8}) & \cos(\frac{\pi}{8}) & -\cos(\frac{\pi}{8}) & \cos(\frac{3\pi}{8}) \\ \cos(\frac{7\pi}{16}) & -\cos(\frac{5\pi}{16}) & \cos(\frac{3\pi}{16}) & -\cos(\frac{\pi}{16}) & \cos(\frac{\pi}{16}) & -\cos(\frac{3\pi}{16}) & \cos(\frac{5\pi}{16}) & -\cos(\frac{7\pi}{16}) \end{pmatrix}$$

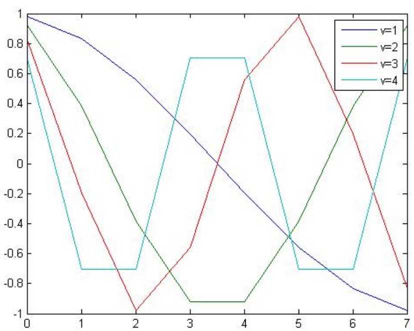
2-D Discrete Cosine Transform (DCT)

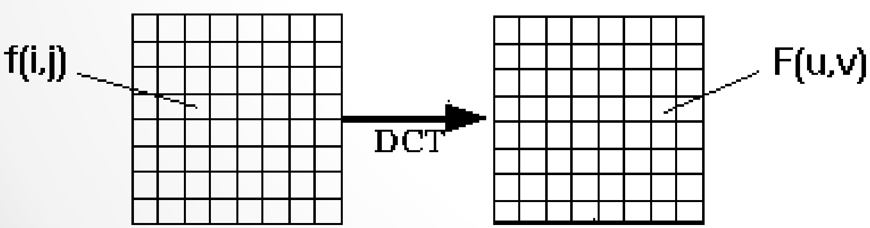
$$F(u,v) = c(u)c(v)\sum_{j=0}^{7}\sum_{i=0}^{7} f(i,j)\cos\left[\frac{(2i+1)u\pi}{16}\right]\cos\left[\frac{(2j+1)v\pi}{16}\right]$$

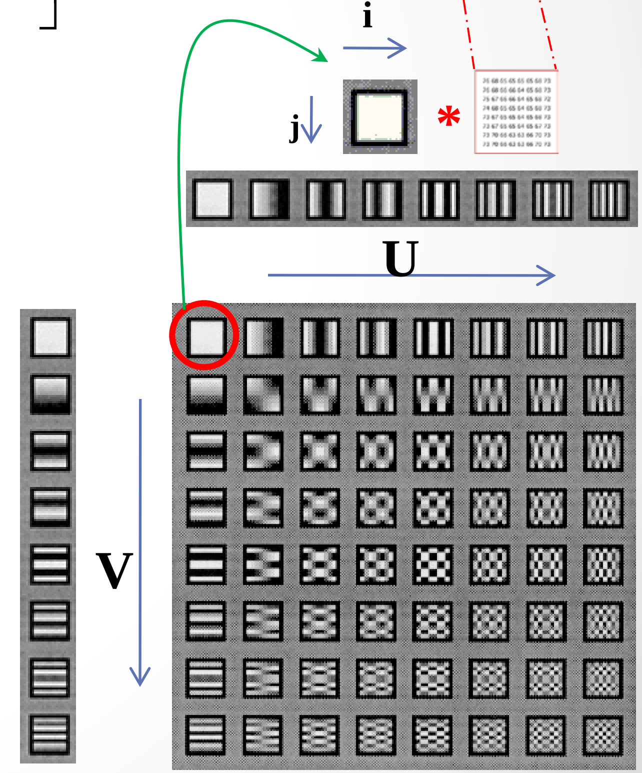
$$u, v = 0, 1, 2, \ldots, 7$$

$$c(u) = \sqrt{\frac{1}{8}} \text{.......for..} u = 0$$

$$c(u) = \sqrt{\frac{1}{4}} \text{........for..} u = 1, 2, \ldots, 7$$







(At receiver: 2-D IDCT performed by decoder)

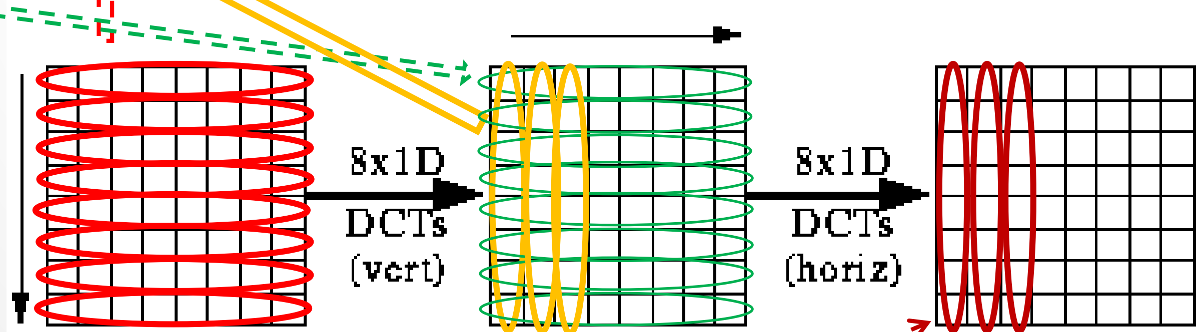
Computing the 2-D DCT by 1-D DCT

- Row-Column Decomposition. The 2-D DCT is calculated using row-column decomposition. First, the 1-D DCT of each row is computed. Then, the 1-D DCT of each of the resulting columns is computed, which yields the 2-D transform. The transformation of the columns cannot begin until all the rows have been transfc

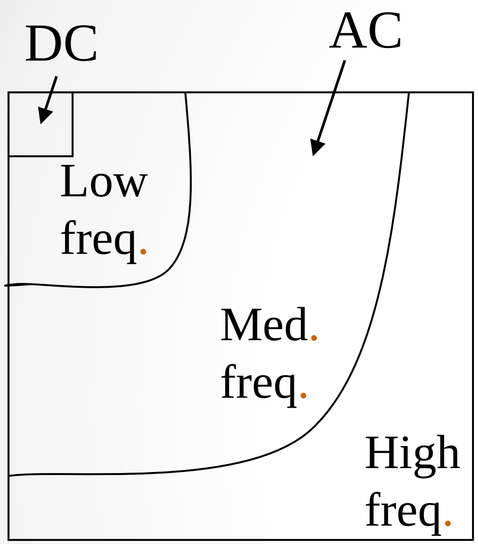
$$F[u,v] = \frac{1}{2}\sum_{i} A(u)\cos\frac{(2i+1)u\pi}{16} G[i,v]$$

$$G[i,v] = \frac{1}{2}\sum_{j} A(v)\cos\frac{(2j+1)v\pi}{16} f[i,j]$$

$$\begin{pmatrix} Y(0) \\ Y(1) \\ Y(2) \\ Y(3) \\ Y(4) \\ Y(5) \\ Y(6) \\ Y(7) \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 1 & 1 \\ \cos(\frac{\pi}{16}) & \cos(\frac{3\pi}{16}) & \cos(\frac{5\pi}{16}) & \cos(\frac{7\pi}{16}) & -\cos(\frac{7\pi}{16}) & -\cos(\frac{5\pi}{16}) & -\cos(\frac{3\pi}{16}) & -\cos(\frac{\pi}{16}) \\ \cos(\frac{\pi}{8}) & \cos(\frac{3\pi}{8}) & -\cos(\frac{3\pi}{8}) & -\cos(\frac{\pi}{8}) & -\cos(\frac{\pi}{8}) & -\cos(\frac{3\pi}{8}) & \cos(\frac{3\pi}{8}) & \cos(\frac{\pi}{8}) \\ \cos(\frac{3\pi}{16}) & -\cos(\frac{7\pi}{16}) & -\cos(\frac{\pi}{16}) & -\cos(\frac{5\pi}{16}) & \cos(\frac{5\pi}{16}) & \cos(\frac{\pi}{16}) & \cos(\frac{7\pi}{16}) & -\cos(\frac{3\pi}{16}) \\ \cos(\frac{\pi}{4}) & -\cos(\frac{\pi}{4}) & -\cos(\frac{\pi}{4}) & \cos(\frac{\pi}{4}) & \cos(\frac{\pi}{4}) & -\cos(\frac{\pi}{4}) & -\cos(\frac{\pi}{4}) & \cos(\frac{\pi}{4}) \\ \cos(\frac{5\pi}{16}) & -\cos(\frac{\pi}{16}) & \cos(\frac{7\pi}{16}) & \cos(\frac{3\pi}{16}) & -\cos(\frac{3\pi}{16}) & -\cos(\frac{7\pi}{16}) & \cos(\frac{\pi}{16}) & -\cos(\frac{5\pi}{16}) \\ \cos(\frac{3\pi}{8}) & -\cos(\frac{\pi}{8}) & \cos(\frac{\pi}{8}) & -\cos(\frac{3\pi}{8}) & -\cos(\frac{3\pi}{8}) & \cos(\frac{\pi}{8}) & -\cos(\frac{\pi}{8}) & \cos(\frac{3\pi}{8}) \\ \cos(\frac{7\pi}{16}) & -\cos(\frac{5\pi}{16}) & \cos(\frac{3\pi}{16}) & -\cos(\frac{\pi}{16}) & \cos(\frac{\pi}{16}) & -\cos(\frac{3\pi}{16}) & \cos(\frac{5\pi}{16}) & -\cos(\frac{7\pi}{16}) \end{pmatrix} \begin{pmatrix} x(0) \\ x(1) \\ x(2) \\ x(3) \\ x(4) \\ x(5) \\ x(6) \\ x(7) \end{pmatrix}$$



Transform Coefficients & Quantization



- Human vision -- low frequencies are more important than high frequencies. Hence higher freqs. can be more coarsely quantized or discarded. The bits saved for coding high frequencies are used for lower frequencies to obtain better subjective coded images.

-483.1250	1.7102	25.5989	-0.2148	11.3750	3.1852	3.3324	-0.4426
3.5185	2.2448	1.1681	1.8343	0.1998	0.6538	-0.3247	0.2546
-0.2590	0.4080	0.3384	-0.1283	0.8562	0.1920	0.2866	0.3167
0.2695	-0.3552	0.2529	0.6294	0.3285	-1.0440	-0.1421	0.1350
-0.3750	-0.7855	0.4339	0.1022	-0.3750	-0.6576	-0.5856	-0.0507
-0.1464	0.0721	0.0876	0.4864	0.3698	-0.3669	-0.2240	0.3948
-0.2986	0.0187	-0.4634	0.2122	-0.2194	0.0268	0.1616	-0.0938
-0.2979	-0.2150	-0.5027	0.0962	0.1672	0.6272	0.1019	0.4927

- **Quantization**
- **$F_q(u,v) = \text{round}\,[F(u,v)/Z(u,v)]$**

-20	0	2	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

Z matrix for Luminance Quantization →

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

Quantization

- Eye is most sensitive to low frequencies (upper left corner), less sensitive to high frequencies (lower right corner).

The *Chrominance Quantization Table* q(u, v)

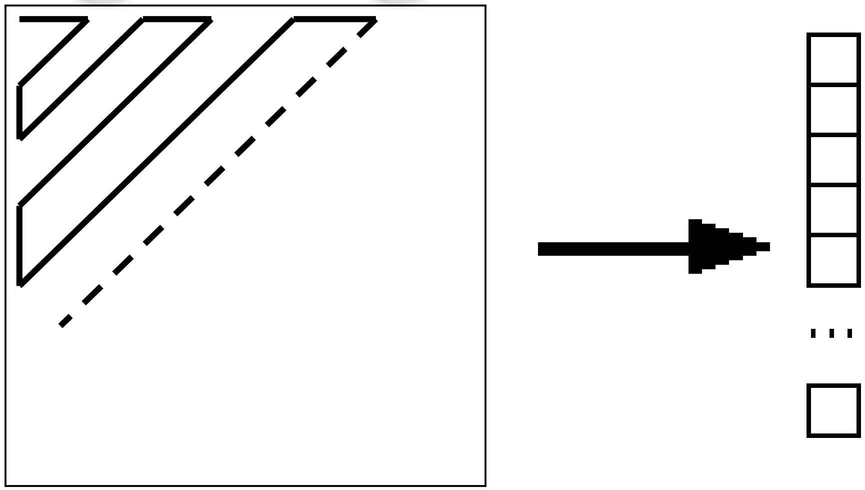
17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99

- The numbers in the above quantization tables can be scaled up (or down) to adjust the so called quality factor.
- Custom quantization tables can also be put in image/scan header.

Zig-Zag Scaning

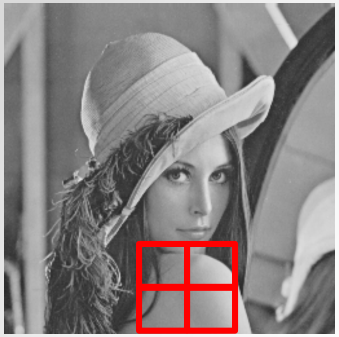
-20	0	2	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

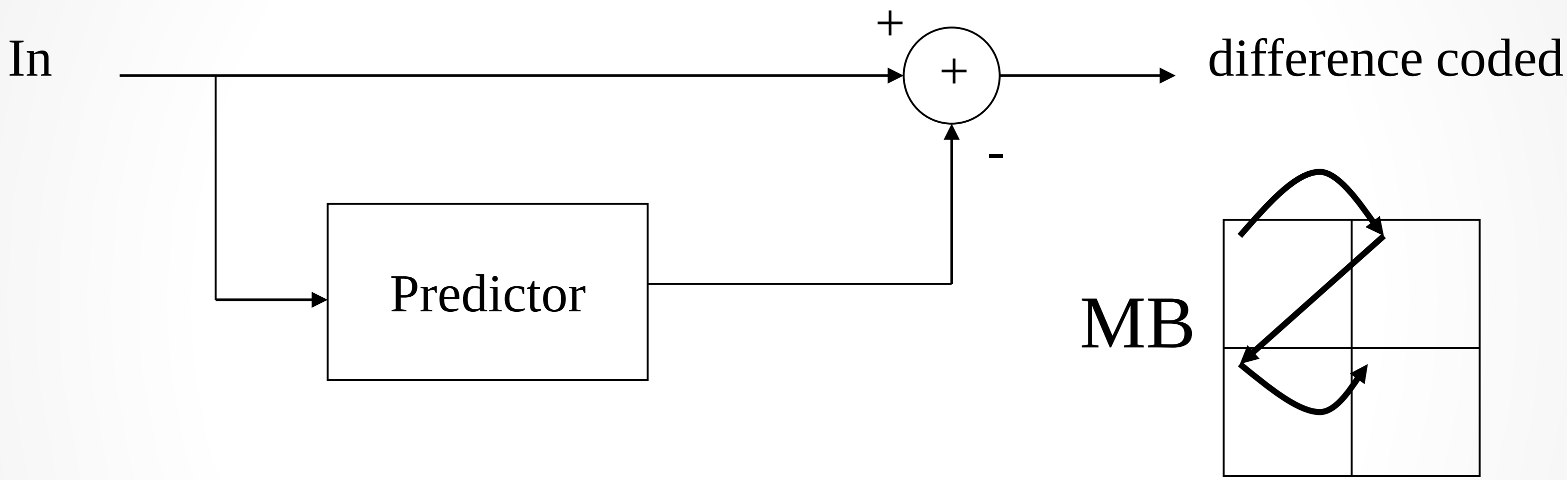
-20 0 0 0 0 2 0 …. 0



- Why? -- to group low frequency coefficients in top of vector. Increase the likelihood of grouping all nonzero coefficients together
- Maps 8 x 8 to a 1 x 64 vector.
- The reordered 1-D sequence contains long runs of 0’s, and can be run-length coded.
- The non-zero coefficients are represented by variable-length codes.

Predictive Coding (DPCM)





- DC coefficients of successive blocks often vary only slightly --> Use DPCM to code DC coefficients
- For each DC, the predictor is the DC of previous block. Hence produce small difference

Run Length Encode (RLE) on AC Components

- 1 x 64 vector has lots of zeros in it
- Keeps *skip* and *value*, where *skip* is the number of zeros and *value* is the next non-zero component.
- Send (0,0) as end-of-block sentinel value.

Entropy Coding

- Categorize DC values into SIZE (number of bits needed to represent) and actual bits.
- Example: if DC value is 4, 3 bits are needed.
- Send off SIZE as Huffman symbol, followed by actual 3 bits.

DC Coefficient	Size	Huffman codes for Size
0	0	00
-1,1	1	010
-3,-2,2,3	2	011
-7,…,-4,4,…,7	3	100
-15,…,-8,8,…,15	4	101
-31,…,-16,16,…,31	5	110
⋮	⋮	⋮
-1023,…-512,512,…,1023	10	1111 1110
-2047,…-1024,1024,…2047	11	1 1111 1110

Entropy Coding

- For AC components two symbols are used because there is a strong correlation between the Size of a coefficient and the expected Run of zeros : Symbol_1: (*Run*, *Size*), Symbol_2: actual bits. Symbol_1 (*Run*, *Size*) is encoded using the Huffman coding, Symbol_2 is not encoded.
- Small coefficients usually follow long runs; larger coefficients tend to follow shorter runs. Huffman Tables can be custom (sent in header) or default.
- ZRL represents a run of 16 zeros which can be part of a longer run of any length.
- EOB is transmitted after the last non-zero coefficient in a 64-vector. It is omitted in case the final element of the vector is non-zero.

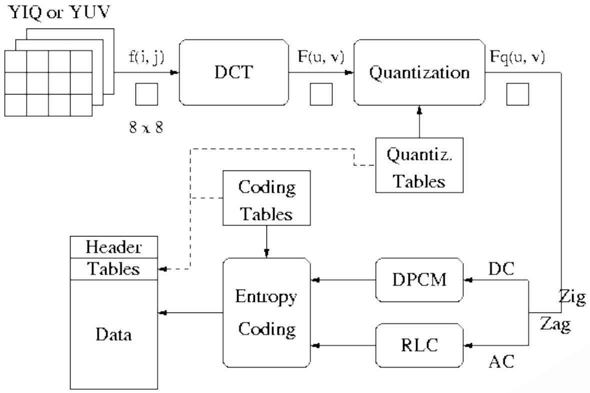
(Run,Size)	Code Word	(Run,Size)	Code Word
(0,1)	00	(0,6)	1111000
(0,2)	01	(1,3)	1111001
(0,3)	100	(5,1)	1111010
(EOB)	1010	(6,1)	1111011
(0,4)	1011	(0,7)	11111000
(1,1)	1100	(2,2)	11111001
(0,5)	11010	(7,1)	11111010
(1,2)	11011	(1,4)	111110110
(2,1)	11100		
(3,1)	111010	(ZRL)	11111111001
(4,1)	111011		

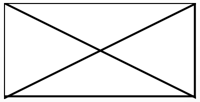
Four JPEG Modes

- Sequential Mode
- Lossless Mode
- Progressive Mode
- Hierarchical Mode

Sequential Mode

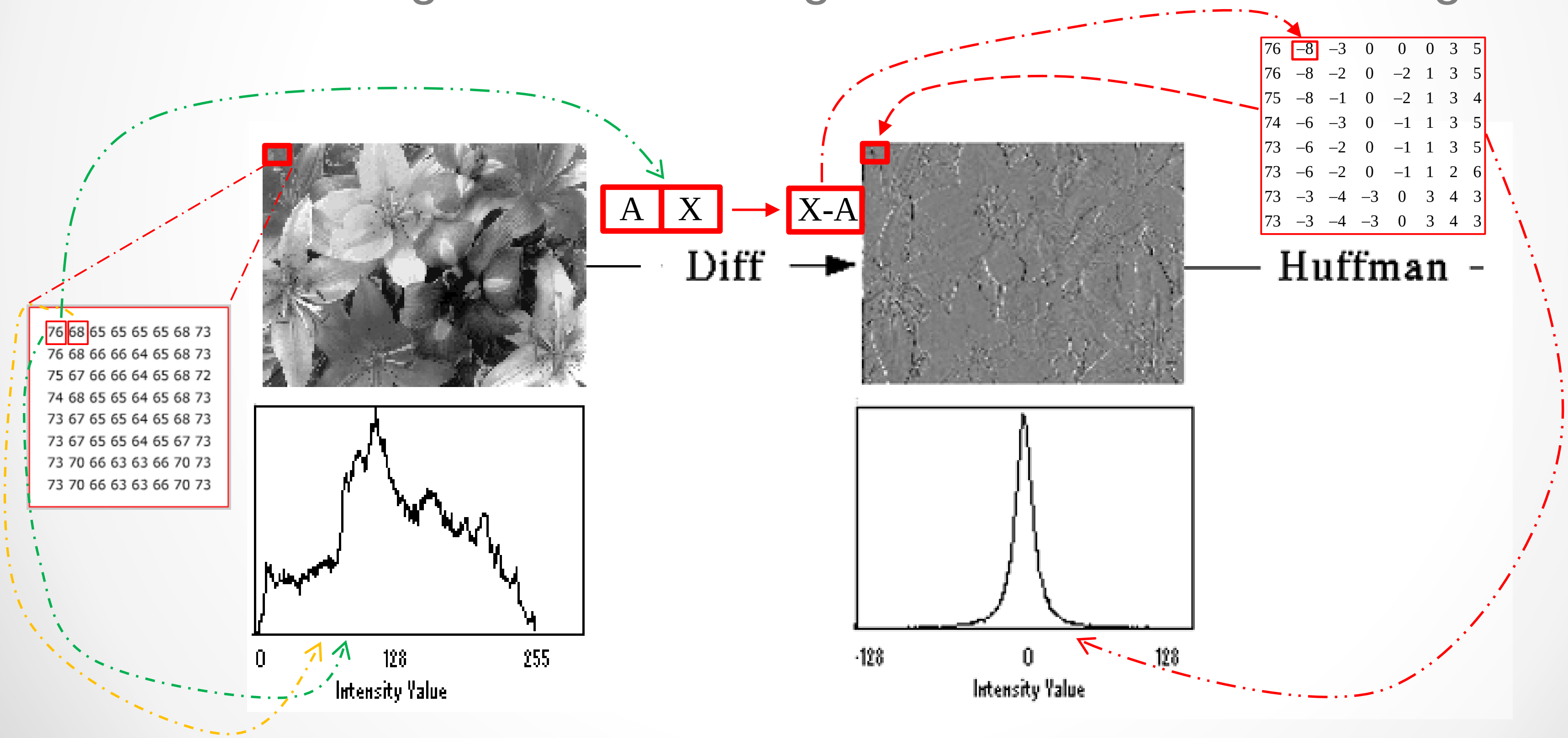
- Each image component is encoded in a single left-to-right, top-to-bottom scan. *Baseline Sequential Mode*, the one that we described above, is a simple case of the Sequential mode:
- It supports only 8-bit images (not 12-bit images)
- It uses only Huffman coding (not Arithmetic coding)





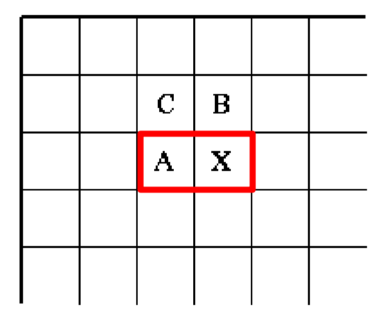
Lossless Mode

- A special case of the JPEG where indeed there is no loss. Its block diagram and histograms are in the followings.



Lossless Mode

- It does not use DCT-based method! Instead, it uses a predictive (differential coding) method.
- A predictor combines the values of up to three neighboring pixels (not blocks as in the Sequential mode) as the predicted value for the current pixel, indicated by "X" in the figure on the right.
- The encoder then compares this prediction with the actual pixel value at the position "X", and encodes the difference (prediction residual) losslessly.
- It can use any one of the seven predictors.
- Since it uses only previously encoded neighbors, the very first pixel I(0, 0) will have to use itself. Other pixels at the first row always use P1, at the first column always use P2.

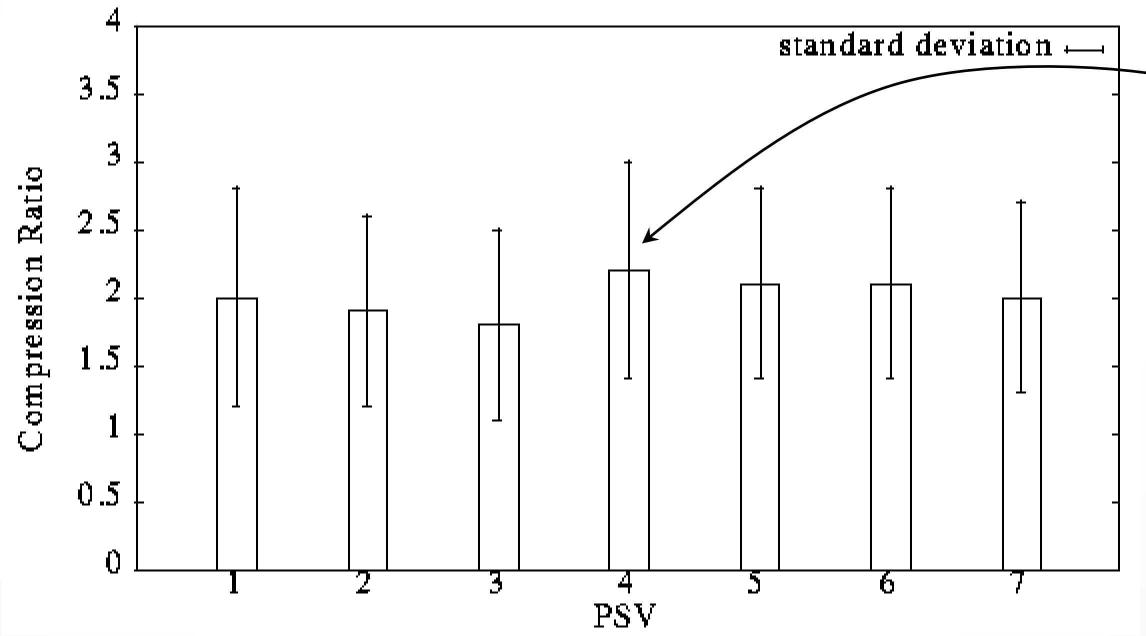


Predictor	Prediction
1	A
2	B
3	C
4	A + B - C
5	A + (B - C) / 2
6	B + (A - C) / 2
7	(A + B) / 2

Lossless Mode

- Effect of Predictor (test result with 20 images):

Predictor	Prediction
1	A
2	B
3	C
4	A + B - C
5	A + (B - C) / 2
6	B + (A - C) / 2
7	(A + B) / 2



Intensity gradually change in horizontal and vertical direction.

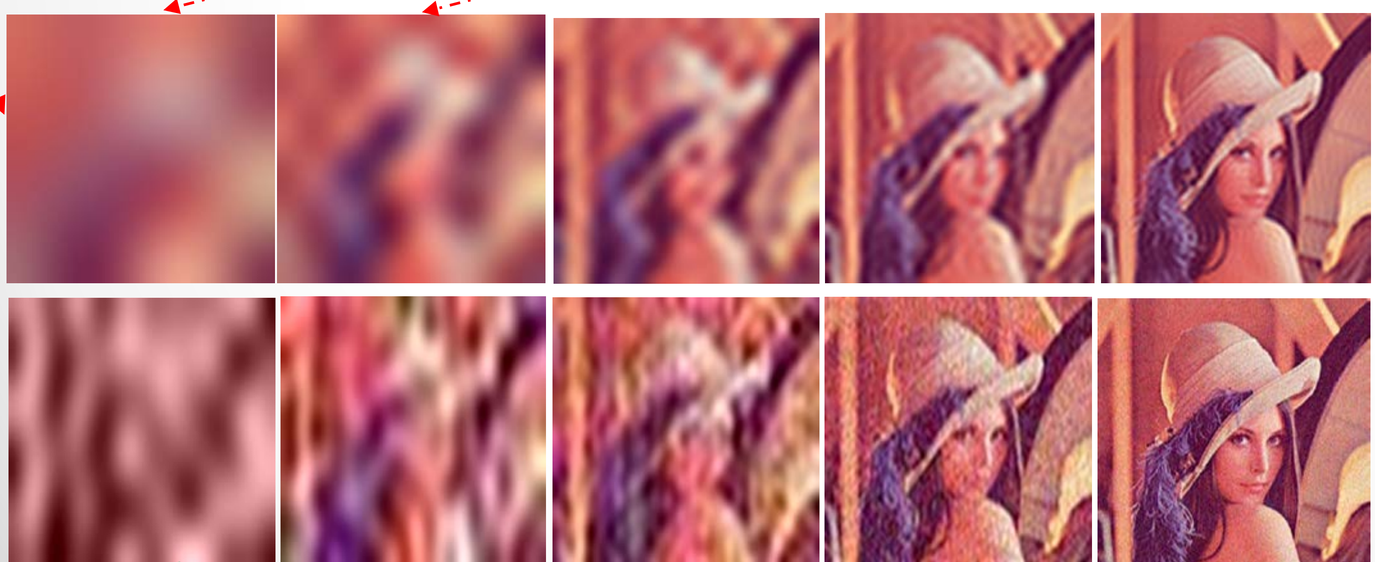
- Predictors (4-7) always do better than predictors (1-3).

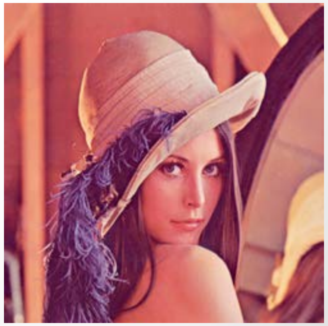
Progressive Mode

- Goal: display low quality image and successively improve.
- Two ways to successively improve image:
  1. Spectral selection: Send DC component and first few AC coefficients first, then gradually some more ACs.
  2. Successive approximation: send DCT coefficients MSB (most significant bit) to LSB (least significant bit).

17	2	0	0	0	0	0	0
4	1	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	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

17	2	4	0	1	0	0	0	1	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	1	0	0	0	0	0	0	0		0
0	1	0	0	0	0	0	0	0	0		0
1	0	0	0	1	0	0	0	1	0	......	0





Hierarchical Mode

- A Three-level Hierarchical JPEG Encoder
  - Down-sample by factors of 2 in each dimension, e.g., reduce 640 x 480 to 320 x 240
  - Code smaller image using another JPEG mode (Progressive, Sequential, or Lossless).
  - Decode and up-sample encoded image
  - Encode difference between the up-sampled and the original using Progressive, Sequential, or Lossless.
- It can be repeated multiple times.
- Good for viewing high resolution image on low resolution display.

Hierarchical Mode