CMPN206: Multimedia

Lecture 8: Image Compression

Mohamed Alaa El-Dien Aly
Computer Engineering Department
Cairo University
Spring 2014

Agenda

- Lossless Image Compression
  - GIF
  - PNG
  - Lossless JPEG
- JPEG Compression

Acknowledgments: Most slides are adapted from Richard Ladner, from Li and Drew, and from Khaled Sayood.

Differential Coding

- Usually pixel values are close to their neighbors
- If we take the difference between a pixel value and its neighbor, the range of difference values is much *smaller*
- For example, we can *predict* a pixel by assuming it's equal to the one on its left, and then encode the difference between the pixel and its prediction

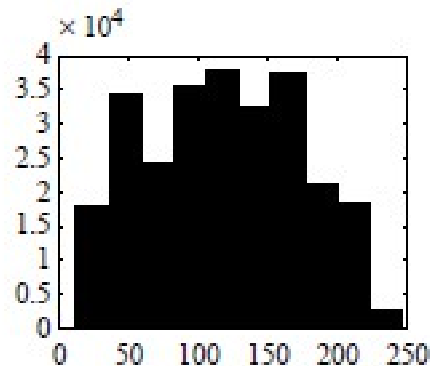
$$d(x, y) = I(x, y) - I(x-1, y)$$

Differential Coding

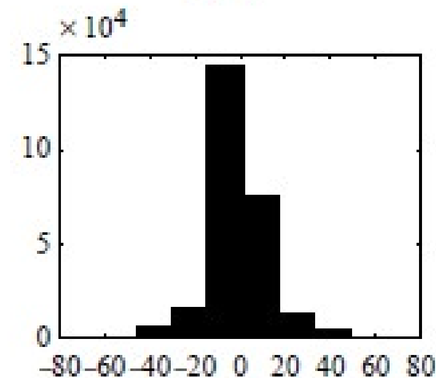
- The histogram of the pixel values is much *wider* than the histogram of the difference values

(a)

(b)



(c)



(d)

Fig. 7.9: Distributions for Original versus Derivative Images. (a,b): Original gray-level image and its partial derivative image; (c,d): Histograms for original and derivative images.

GIF

- Graphics Interchange Format
- Uses LZW algorithm
- The first byte stores the minimum number of bits per pixel b
- The initial dictionary size is 2^{b+1} and keeps doubling till a maximum of 4096 i.e. 12 bits per entry
- Once this is reached, the dictionary becomes *static* i.e. fixed
- It compresses pixel values themselves (not differences)

TABLE 5.16 Comparison of GIF with arithmetic coding.

Image	GIF	Arithmetic Coding of Pixel Values	Arithmetic Coding of Pixel Differences
Sena	51,085	53,431	31,847
Sensin	60,649	58,306	37,126
Earth	34,276	38,248	32,137
Omaha	61,580	56,061	51,393

PNG

- Portable Network Graphics
- Developed due to *patent* issues with the GIF format, which is based on the patented LZW algorithm
- A free and open standard based on the LZ77 algorithm
- Based on the *deflate* algorithm implementation of LZ77
- Match lengths between 3 and 258
- Offset between 1 and 32,768
- The match length k is at least 3. If no match in three bytes, outputs the byte *literal* (a value $0 \rightarrow 255$) and tries again
- Can go up to 258 bytes (symbols). When a match is found, a pair $\langle$match length, offset$\rangle$ is output

PNG

- The literals (0 → 255) and the match lengths (3 → 258) are coded into *one* alphabet using a Huffman Tree with 286 symbols
  - symbols 0 → 255 are literals (8 or 9 bits)
  - symbol 256 is for End of Block (7 bits)

TABLE 5.18 **Huffman codes for the *match length* alphabet [59].**

Index Ranges	# of bits	Binary Codes
0–143	8	00110000 through 10111111
144–255	9	110010000 through 111111111
256–279	7	0000000 through 0010111
280–287	8	11000000 through 11000111

PNG

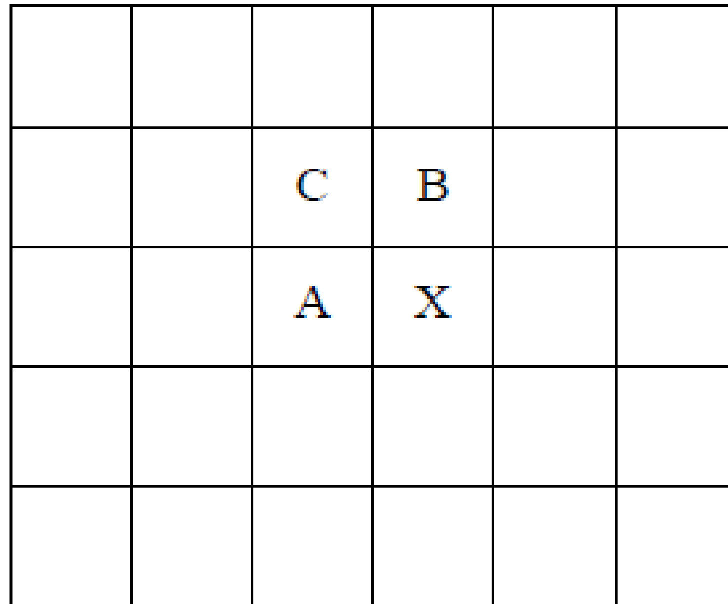
- The literals (0 → 255) and the match lengths (3 → 258) are coded into one alphabet using a Huffman Tree with 286 symbols
  - symbols 257 → 285 indicate different *ranges* of match lengths (7 or 8 bits followed by *selector* bits)
- E.g. 277 represents match lengths between 67 and 82, to know which one, 4 bits are required (the 4 bits following the 7 bits of 277)

TABLE 5.17 Codes for representations of *match length* [59].

Index	# of selector bits	Length	Index	# of selector bits	Length	Index	# of selector bits	Length
257	0	3	267	1	15,16	277	4	67–82
258	0	4	268	1	17,18	278	4	83–98
259	0	5	269	2	19–22	279	4	99–114
260	0	6	270	2	23–26	280	4	115–130
261	0	7	271	2	27–30	281	5	131–162
262	0	8	272	2	31–34	282	5	163–194
263	0	9	273	3	35–42	283	5	195–226
264	0	10	274	3	43–50	284	5	227–257
265	1	11, 12	275	3	51–58	285	0	258
266	1	13, 14	276	3	59–66			

PNG

- Encodes difference from *predicted* pixel value: $P - X$
- Different predictions:
  - $P = B$ (pixel above)
  - $P = A$ (pixel to the left)
  - $P = (A + B) / 2$
  - $P = A$ *or* B *or* C that is closest to $(A + B - C)$



PNG

- Much better compression than GIF due to the use of pixel predictions and entropy coding
- Similar to Arithmetic coding of pixel differences

TABLE 5.19 **Comparison of PNG with GIF and arithmetic coding.**

Image	PNG	GIF	Arithmetic Coding of Pixel Values	Arithmetic Coding of Pixel Differences
Sena	31,577	51,085	53,431	31,847
Sensin	34,488	60,649	58,306	37,126
Earth	26,995	34,276	38,248	32,137
Omaha	50,185	61,580	56,061	51,393

Lossless JPEG

- Joint Photographic Experts Group
- Old lossless standard
- Uses *differential coding*, with 8 different prediction modes (P0 means no prediction) and uses entropy coding e.g. Huffman coding

Predictor	Prediction
P1	A
P2	B
P3	C
P4	A + B - C
P5	A + (B - C) / 2
P6	B + (A - C) / 2
P7	(A + B) / 2

		C	B		
		A	X		

Lossless JPEG

- Compression ratios comparable to PNG, better than GIF

TABLE 7.2 **Comparison of the file sizes obtained using JPEG lossless compression, GIF, and PNG.**

Image	Best JPEG	GIF	PNG
Sena	31,055	51,085	31,577
Sensin	32,429	60,649	34,488
Earth	32,137	34,276	26,995
Omaha	48,818	61,341	50,185

JPEG

- Adopted as a standard in 1992
- The most widely used *lossy* compression image format
- Uses Transform Coding with *Discrete Cosine Transform* (DCT)

JPEG Philosophy

The choices inside the JPEG standard based upon:

- Observation 1: useful image contents change relatively slowly across the image e.g. in an 8×8 block of the image intensity values don't vary wildly

  → the DCT high frequency AC components are usually *small*

JPEG Philosophy

The choices inside the JPEG standard based upon:

- Observation 2: Psychophsysical experiments (experiments that measure the response of the human visual system) suggest that humans are much less likely to notice the loss of very high spatial frequency components than the loss of lower frequency components

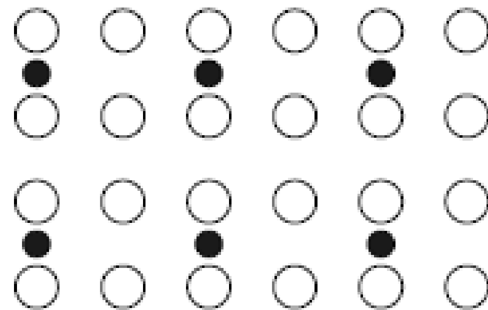
  → setting the DCT *high frequency* AC components to zero (getting rid of high spatial frequency) is not very noticeable to humans while the DC component is very important

JPEG Philosophy

The choices inside the JPEG standard based upon:

- Observation 3: *visual acuity* (accuracy in distinguishing closely spaced lines) is much greater for gray ("black and white") than for color.

  → using less color information is not very noticeable and thus JPEG uses *chroma subsampling* (4:2:0) i.e. for every four pixels, send only one color value e.g. if using YUV, send Y for every pixel, and U and V for every four pixels



4:2:0

JPEG Encoder

1. Color Transformation
2. DCT on image blocks
3. Quantization
4. Zig-zag and RLC
5. Entropy coding

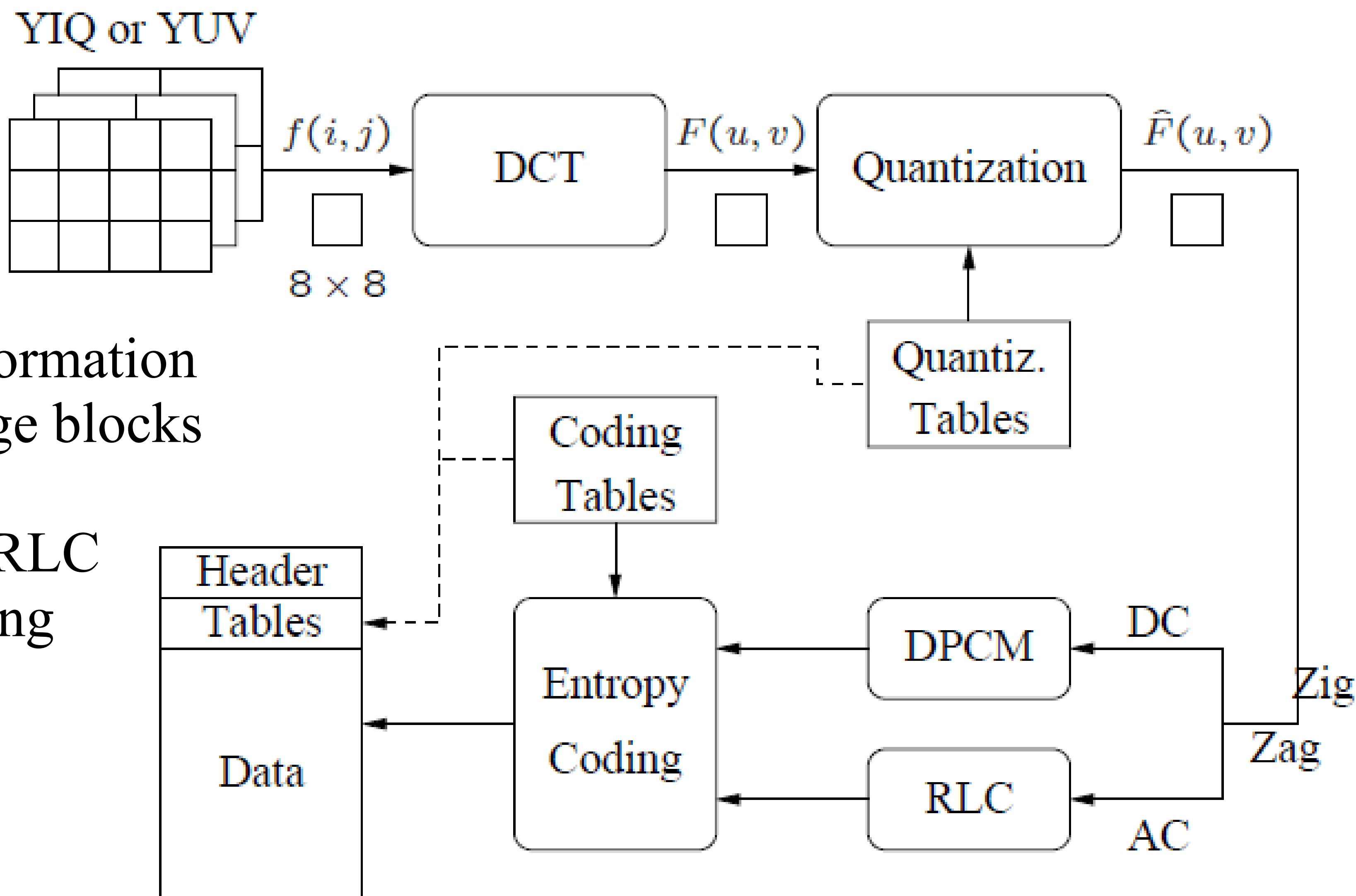


Fig. 9.1: Block diagram for JPEG encoder.

1. Color Transformation

- JPEG works on grayscale or color images
- Usually the color image is in RGB, in which case it is converted to YUV or YIQ color spaces
  - The *Y* channel represents the *luminance* or the brightness information
  - The *UV* or *IQ* represent the *chrominance* or the color information
- The *two* color channels are then subsampled

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.595716 & -0.274453 & -0.321263 \\ 0.211456 & -0.522591 & 0.311135 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$\begin{bmatrix} Y' \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.2126 & 0.7152 & 0.0722 \\ -0.09991 & -0.33609 & 0.436 \\ 0.615 & -0.55861 & -0.05639 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

[https://en.wikipedia.org/wiki/YIQ] [https://en.wikipedia.org/wiki/YUV]

2. DCT Transform

- Blocks of 8×8 pixels X are transformed using 2D DCT transform matrix A to produce the transformed coefficients Y

$$Y = A\,X\,A^T$$

where A is defined by

$$a_{i,j} = \begin{cases} \sqrt{\dfrac{1}{N}} & \text{for } i=0 \text{ and } j=0,\cdots,7 \\ \sqrt{\dfrac{2}{N}}\cos\dfrac{(2j+1)i\pi}{2N} & \text{for } i>0 \text{ and } j=0,\cdots,7 \end{cases}$$

2. DCT Transform

- This produces the *blocking artifacts* in the JPEG-coded images, which make them appear *blocky* at higher compression ratios
- Before applying the DCT, the pixels are *centered* around 0 by subtracting 128 from each pixel
- This makes the compression more efficient by not coding the *average* value of the block (in the DC component)

3. Quantization

- The DCT coefficients are *uniformly* quantized using a *quantization table*

$$\hat{y}_{i,j} = \text{round}\left(\frac{y_{i,j}}{q_{i,j}}\right) = \left\lfloor \frac{y_{i,j}}{q_{i,j}} + 0.5 \right\rfloor$$

where $q_{i,j}$ is the *quantization step* for coefficient $y_{i,j}$

- The DC and lower frequency AC components are more important, and hence are assigned *more bits* i.e. quantized with *smaller intervals* (smaller steps)
- The values for $q_{i,j}$ tend to get larger towards the lower right end of the 8×8 block, following the observations earlier
- These values are derived from psychophysical experiments aiming at optimizing the compression ratios while having acceptable *subjective quality* for the images

3. Quantization

- The *luminance* and *chrominance* are quantized separately
- The *quality* setting for JPEG compression is directly related to the quantization tables

Table 9.1 The Luminance Quantization 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

Table 9.2 The Chrominance Quantization 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

Example: Smooth Block

An 8×8 block from the Y image of 'Lena'

Most high frequency DCT Coefficients are small

200	202	189	188	189	175	175	175
200	203	198	188	189	182	178	175
203	200	200	195	200	187	185	175
200	200	200	200	197	187	187	187
200	205	200	200	195	188	187	175
200	200	200	200	200	190	187	175
205	200	199	200	191	187	187	175
210	200	200	200	188	185	187	186

$f(i,j)$

515	65	-12	4	1	2	-8	5
-16	3	2	0	0	-11	-2	3
-12	6	11	-1	3	0	1	-2
-8	3	-4	2	-2	-3	-5	-2
0	-2	7	-5	4	0	-1	-4
0	-3	-1	0	4	1	-1	0
3	-2	-3	3	3	-1	-1	3
-2	5	-2	4	-2	2	-3	0

$F(u,v)$

Fig. 9.2: JPEG compression for a smooth image block.

Example: Smooth Block

Quantized values: lots of zeros at the lower right

32	6	-1	0	0	0	0	0
-1	0	0	0	0	0	0	0
-1	0	1	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

$\hat{F}(u,v)$

Reconstructed DCT Coefficients

512	66	-10	0	0	0	0	0
-12	0	0	0	0	0	0	0
-14	0	16	0	0	0	0	0
-14	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

$\tilde{F}(u,v)$

199	196	191	186	182	178	177	176
201	199	196	192	188	183	180	178
203	203	202	200	195	189	183	180
202	203	204	203	198	191	183	179
200	201	202	201	196	189	182	177
200	200	199	197	192	186	181	177
204	202	199	195	190	186	183	181
207	204	200	194	190	187	185	184

$\tilde{f}(i,j)$

1	6	-2	2	7	-3	-2	-1
-1	4	2	-4	1	-1	-2	-3
0	-3	-2	-5	5	-2	2	-5
-2	-3	-4	-3	-1	-4	4	8
0	4	-2	-1	-1	-1	5	-2
0	0	1	3	8	4	6	-2
1	-2	0	5	1	1	4	-6
3	-4	0	6	-2	-2	2	2

$\epsilon(i,j) = f(i,j) - \tilde{f}(i,j)$

Fig. 9.2 (cont'd): JPEG compression for a smooth image block.

Example: Textured Block

Fewer high frequency DCT Coefficients are small

Another 8×8 block from the Y image of 'Lena'

70	70	100	70	87	87	150	187
85	100	96	79	87	154	87	113
100	85	116	79	70	87	86	196
136	69	87	200	79	71	117	96
161	70	87	200	103	71	96	113
161	123	147	133	113	113	85	161
146	147	175	100	103	103	163	187
156	146	189	70	113	161	163	197

$f(i,j)$

-80	-40	89	-73	44	32	53	-3
-135	-59	-26	6	14	-3	-13	-28
47	-76	66	-3	-108	-78	33	59
-2	10	-18	0	33	11	-21	1
-1	-9	-22	8	32	65	-36	-1
5	-20	28	-46	3	24	-30	24
6	-20	37	-28	12	-35	33	17
-5	-23	33	-30	17	-5	-4	20

$F(u,v)$

Fig. 9.3: JPEG compression for a textured image block.

Example: Textured Block

Quantized values: still lots of zeros at the lower right

-5	-4	9	-5	2	1	1	0
-11	-5	-2	0	1	0	0	-1
3	-6	4	0	-3	-1	0	1
0	1	-1	0	1	0	0	0
0	0	-1	0	0	1	0	0
0	-1	1	-1	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

$\hat{F}(u,v)$

Reconstructed DCT Coefficients

-80	-44	90	-80	48	40	51	0
-132	-60	-28	0	26	0	0	-55
42	-78	64	0	-120	-57	0	56
0	17	-22	0	51	0	0	0
0	0	-37	0	0	109	0	0
0	-35	55	-64	0	0	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

$\tilde{F}(u,v)$

70	60	106	94	62	103	146	176
85	101	85	75	102	127	93	144
98	99	92	102	74	98	89	167
132	53	111	180	55	70	106	145
173	57	114	207	111	89	84	90
164	123	131	135	133	92	85	162
141	159	169	73	106	101	149	224
150	141	195	79	107	147	210	153

$\tilde{f}(i,j)$

0	10	-6	-24	25	-16	4	11
0	-1	11	4	-15	27	-6	-31
2	-14	24	-23	-4	-11	-3	29
4	16	-24	20	24	1	11	-49
-12	13	-27	-7	-8	-18	12	23
-3	0	16	-2	-20	21	0	-1
5	-12	6	27	-3	2	14	-37
6	5	-6	-9	6	14	-47	44

$\epsilon(i,j) = f(i,j) - \tilde{f}(i,j)$

Fig. 9.3 (cont'd): JPEG compression for a textured image block.

4. Zig-zag and RLC

- A lot of the *quantized coefficients* are zeros at high frequencies
- To make it more likely to have long *runs of zeros*, the coefficients are arranged in a zig-zag order to form a 64-dimensional vector

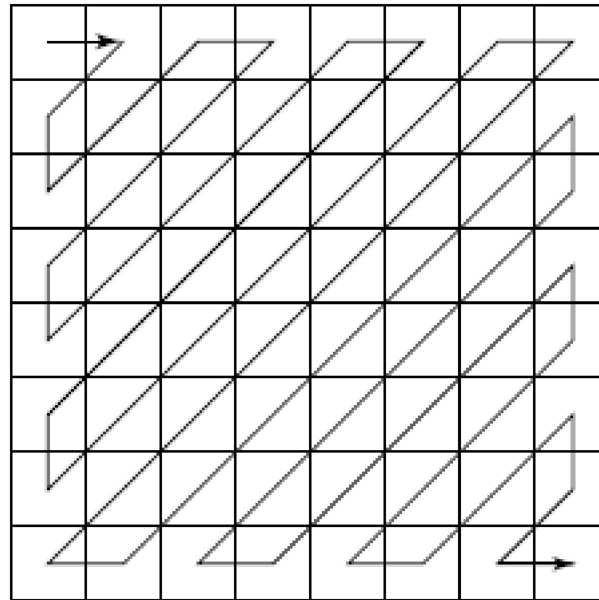


Fig. 9.4: Zig-Zag Scan in JPEG.

4. Zig-zag and RLC

- The zig-zag-ordered coefficients are then compressed using Run Length Coding to produce pairs (RUNLENGTH, VALUE) where RUNLENGTH is the number of zeros to skip and VALUE is the nonzero value following the zero run
- The AC coefficients are coded differently from the DC coefficients:
  - AC coefficients are run length coded within each block independently
  - DC coefficients from blocks of the whole image are coded together

4. Zig-zag and RLC: AC

- The AC coefficients: RLC in each block

32	6	-1	0	0	0	0	0
-1	0	0	0	0	0	0	0
-1	0	1	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

- Example: zig-zag ordering this DCT block produces:

(32, 6, –1, –1, 0, –1, 0, 0, 0, –1, 0, 0, 1, 0, … 0)

and RLC produces:

(0,6) (0, –1) (0, –1) (1, –1) (3, –1) (2, 1) (0, 0)

where (0, 0) is a special end-of-block symbol

4. Zig-zag and RLC: DC

- The DC coefficients:
  - DC values from all the blocks in the image are coded together using DPCM
  - The intuition is that the DC values don't change much from one block to the next

- Example: If the first five blocks have DC coefficients of

  150, 155, 149, 152, 144

  they are coded as

  150, 5, –6, 3, –8

5. Entropy Coding

- The output of the RLC and DPCM from the previous step undergo *entropy coding* (e.g. Huffman or Arithmetic Coding) to achieve further compression
- The AC and DC parts are coded separately

5. Entropy Coding: DC

- Each DPCM coded DC coefficient is represented by (SIZE, AMPLITUDE) pairs, where SIZE indicates how many bits are needed for representing the coefficient, and AMPLITUDE contains the actual bits.
- The AMPLITUDE can be +ve or –ve, where –ve numbers are the 1's complement of the +ve numbers, e.g. 3 is 11, so –3 is 00 …
- Example: (3, 001) represents –6

SIZE	AMPLITUDE
1	-1, 1
2	-3, -2, 2, 3
3	-7..-4, 4..7
4	-15..-8, 8..15
.	.
.	.
.	.
10	-1023..-512, 512..1023

5. Entropy Coding: DC

- The SIZE is coded using Huffman Coding, since smaller magnitudes are more likely than larger ones
- The Huffman table can be standard or depends on the image (stored in the header)
- The AMPLITUDE is *not* coded with Huffman because the values vary and not much is gained

SIZE	AMPLITUDE
1	-1, 1
2	-3, -2, 2, 3
3	-7..-4, 4..7
4	-15..-8, 8..15
.	.
.	.
.	.
10	-1023..-512, 512..1023

- Example: the DC coefficients from before:
  150, 5, –6, 3, –8
  are coded as:
  (8, 10010110), (3, 101), (3, 001), (2, 11), (4, 0111)

5. Entropy Coding: AC

- The RLC coded AC coefficients are represented by pairs (RUNLENGTH, VALUE)
- The VALUE is represented by a pair (SIZE, AMPLITUDE)
- In the JPEG implementation:
  - RUNLENGTH and SIZE are allocated 4 bits each (one byte for both, called Symbol1)
  - AMPLITUDE is allocated 8 bits (Symbol2)
  - Symbol1 = (RUNGLENGTH, SIZE)
  - Symbol2 = (AMPLITUDE)
- Symbol1 is Huffman coded, while Symbol2 is *not*
- In case a zero run is longer than 15, special Symbol1 (15, 0) = 0xF0 can be used e.g. if we have a run of 20, it can be represented by two symbols (15, 0) (4, ?)

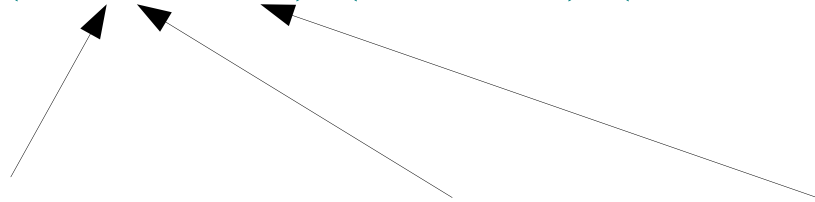
5. Entropy Coding: AC

- Example: the RLC coded AC

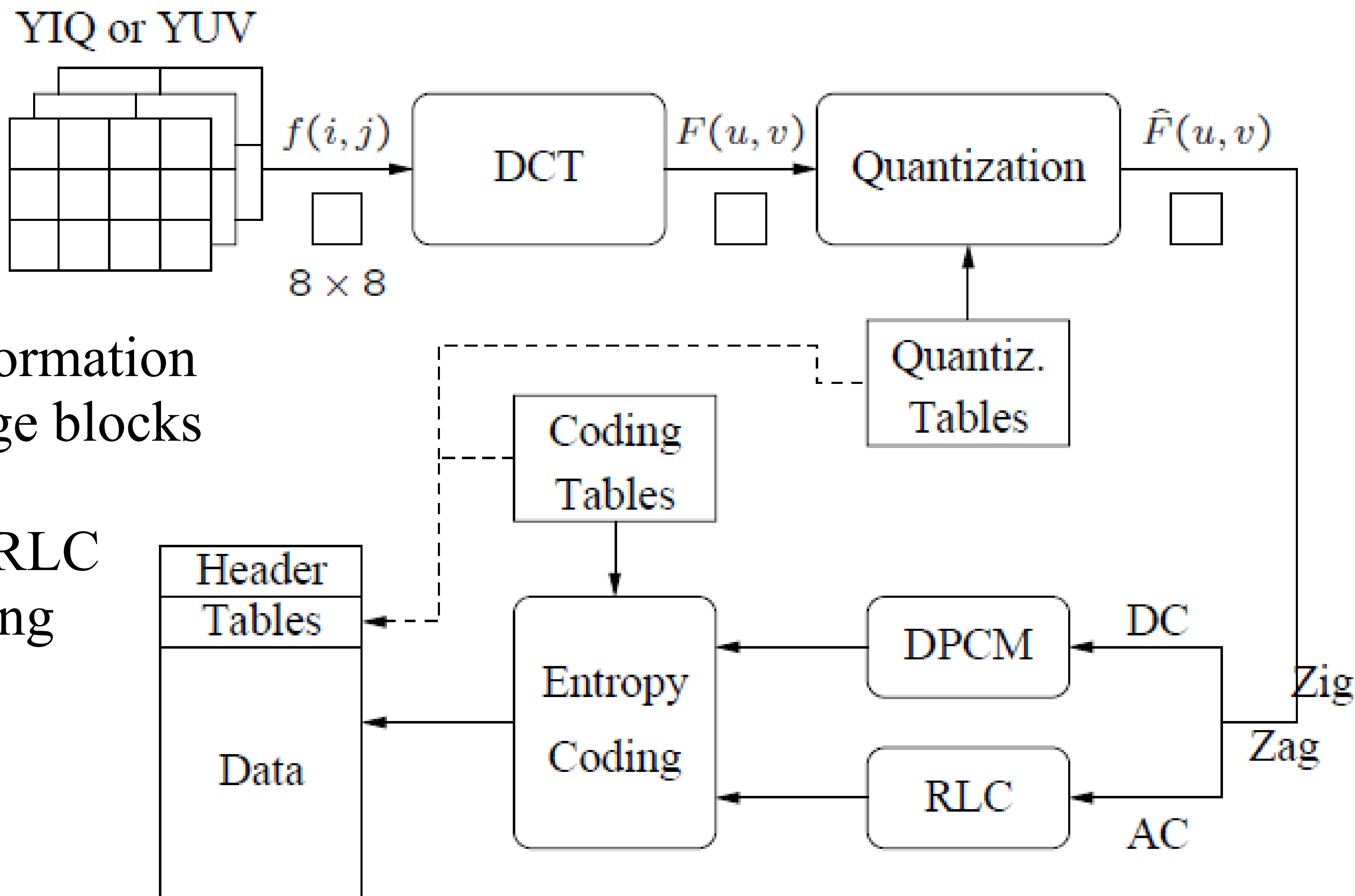
  (0,6) (0, –1) (0, –1) (1, –1) (3, –1) (2, 1) (0, 0)

  are represented as

  (0x03, 110) (0x01, 0) (0x01, 0) (0x11, 0) (0x31, 0) (0x21, 1) ...



JPEG Encoder



1. Color Transformation
2. DCT on image blocks
3. Quantization
4. Zig-zag and RLC
5. Entropy coding

Fig. 9.1: Block diagram for JPEG encoder.

Common JPEG Modes

- Sequential Mode:
  - The default mode JPEG mode explained so far
  - Encodes each color channel in a left-to-right top-to-bottom (row major) order
- Progressive Mode:
  - Sends the image in passes of increasing detail
- Hierarchical Mode:
  - Sends different versions of the image at different resolutions
- Lossless Mode:
  - explained earlier

Progressive Mode

- Delivers low quality versions of the image, followed by high quality passes
- Supported in web browsers to render a quick low resolution version before receiving the full higher quality image
- Can be done in two ways:
  - Spectral Selection
  - Successive Approximation

Progressive Mode

- Spectral Selection: Takes advantage of the *spectral* properties of the DCT coefficients where the higher AC components provide only more *detail* while most of the information is in the lower AC and DC components
- Arranges the data in *scans*:
  - Scan 1: encode DC and first few AC components e.g. AC1, AC2
  - Scan 2: encode a few more AC components e.g. AC3, AC4, AC5
  - …
  - Scan *k*: Encode the last few AC components e.g. AC61, AC62, AC63

Progressive Mode

- Successive Approximation: This is like *bit plane coding*, where all the DCT coefficients are encoded but with their *most significant bits* (MSBs) first
- Arranges the data in *scans*:
  - Scan 1: encode the first few MSBs of all the coefficients e.g. bits 7, 6, 5
  - Scan 2: encode a few more bits e.g. bits 4, 3
  - …
  - Scan k: Encode the least significant bit

Hierarchical Mode

- Sends different versions of the image with different *resolutions* (sizes)
- The lowest resolution version is a low-passed version
- The higher resolution versions provide additional *details* (differences from the lower resolution versions)
- Similar to the idea of an *image pyramid*
- The different versions can be sent in different passes, similar to the progressive mode

Hierarchical Mode

Why do we need that?

Because the decoder only has access to the decoded image.

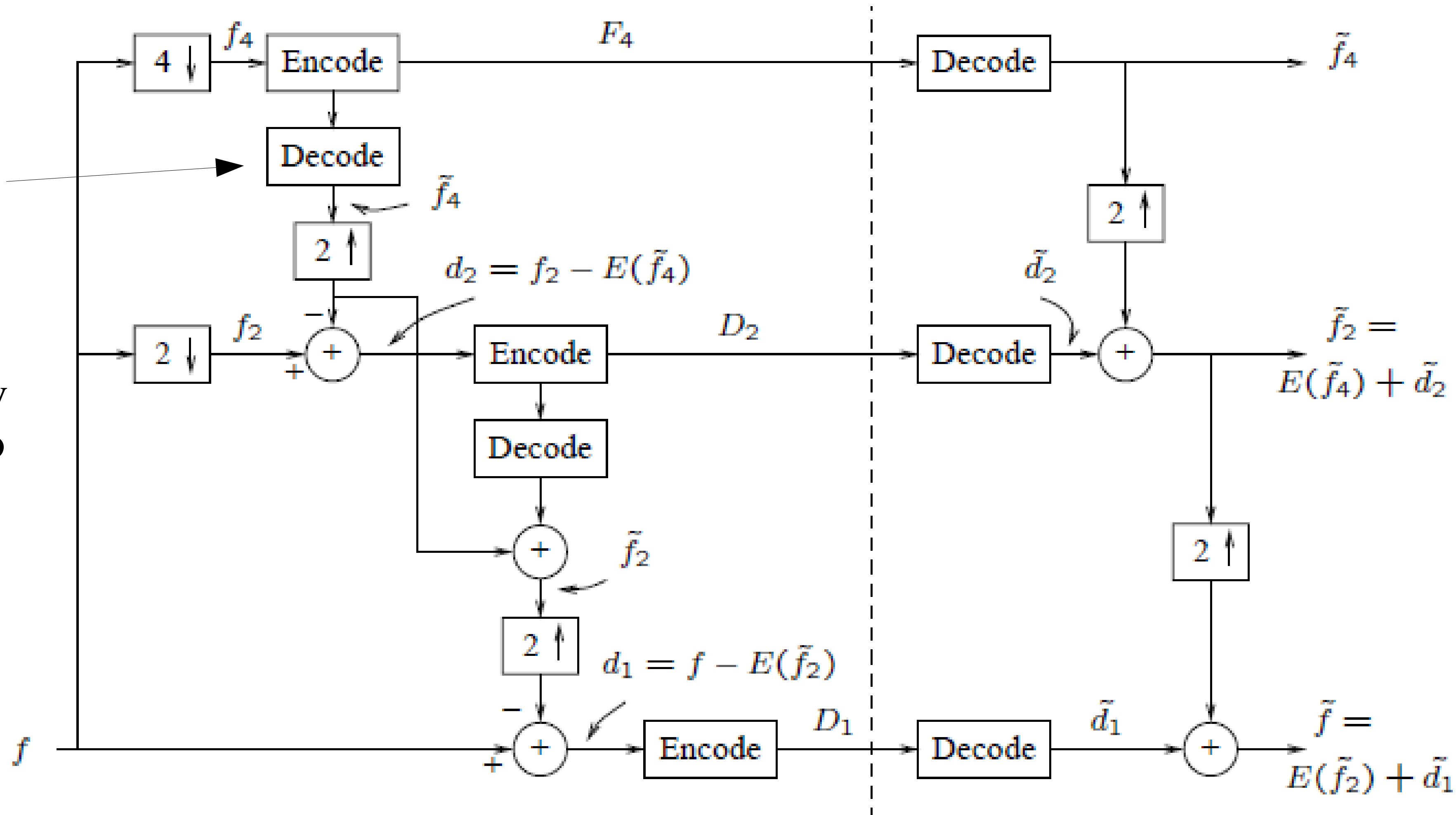


Fig. 9.5: Block diagram for Hierarchical JPEG.

Hierarchical Mode: Encoder

1. Reduction of image resolution:

   Reduce resolution of the input image f (e.g., 512×512) by a factor of 2 in each dimension to obtain f_2 (e.g., 256×256). Repeat this to obtain f_4 (e.g., 128×128).

2. Compress low-resolution image f_4:

   Encode f_4 using any other JPEG method (e.g., Sequential, Progressive) to obtain F_4.

3. Compress difference image d_2:

   (a) Decode F_4 to obtain $\tilde{f}_4$. Use any interpolation method to expand $\tilde{f}_4$ to be of the same resolution as f_2 and call it $E(\tilde{f}_4)$.

   (b) Encode difference $d_2 = f_2 - E(\tilde{f}_4)$ using any other JPEG method (e.g., Sequential, Progressive) to generate D_2.

4. Compress difference image d_1:

   (a) Decode D_2 to obtain $\tilde{d}_2$; add it to $E(\tilde{f}_4)$ to get $\tilde{f}_2 = E(\tilde{f}_4) + \tilde{d}_2$ which is a version of f_2 after compression and decompression.

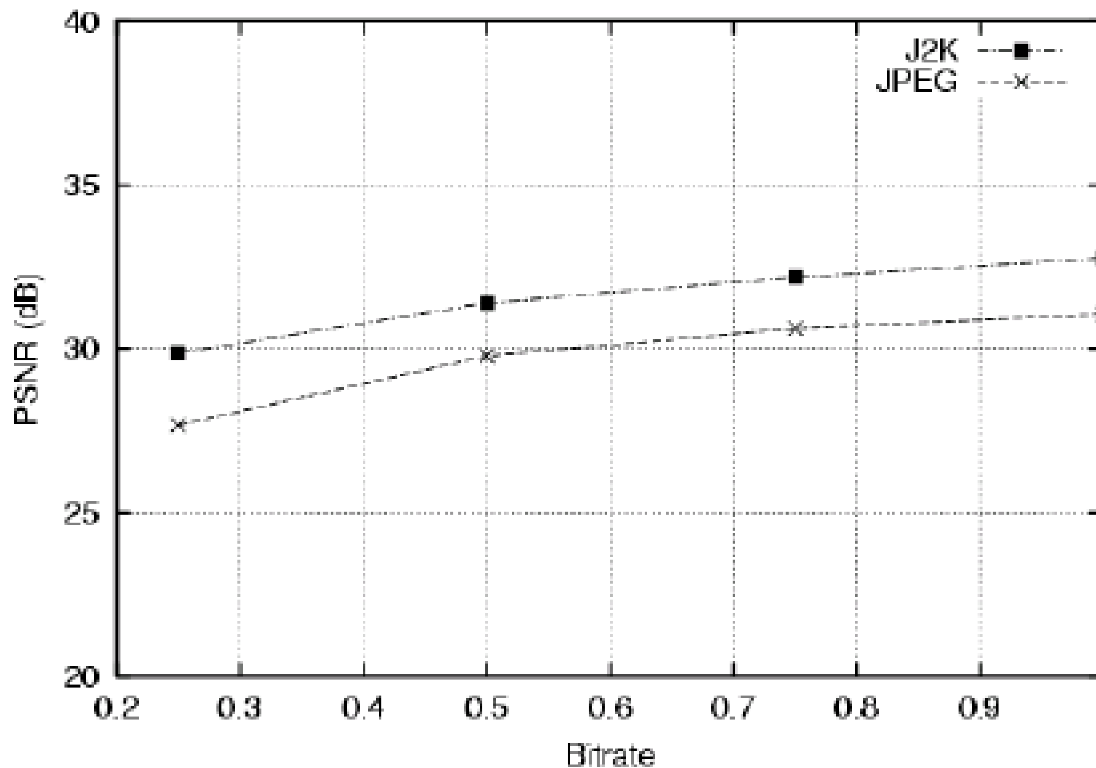
   (b) Encode difference $d_1 = f - E(\tilde{f}_2)$ using any other JPEG method (e.g., Sequential, Progressive) to generate D_1.

Hierarchical Mode: Decoder

1. Decompress the encoded low-resolution image F_4:
   - Decode F_4 using the same JPEG method as in the encoder to obtain $\tilde{f}_4$.
2. Restore image $\tilde{f}_2$ at the intermediate resolution:
   - Use $E(\tilde{f}_4) + \tilde{d}_2$ to obtain $\tilde{f}_2$.
3. Restore image $\tilde{f}$ at the original resolution:
   - Use $E(\tilde{f}_2) + \tilde{d}_1$ to obtain $\tilde{f}$.

JPEG2000

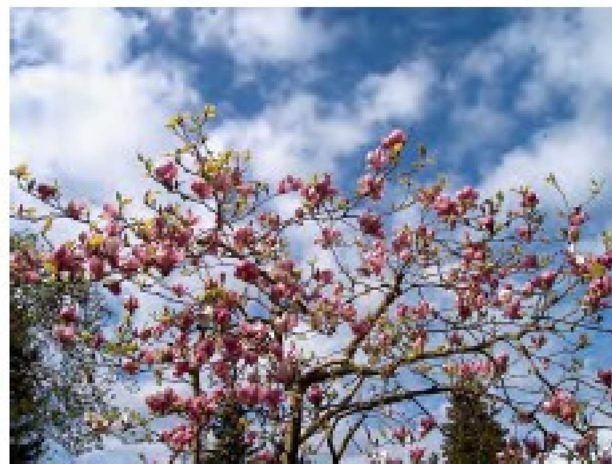
- The newest JPEG standard
- Based on *Wavelet Coding* instead of DCT Coding
- Provides much better quality than JPEG at the same bit rate



JPEG2000

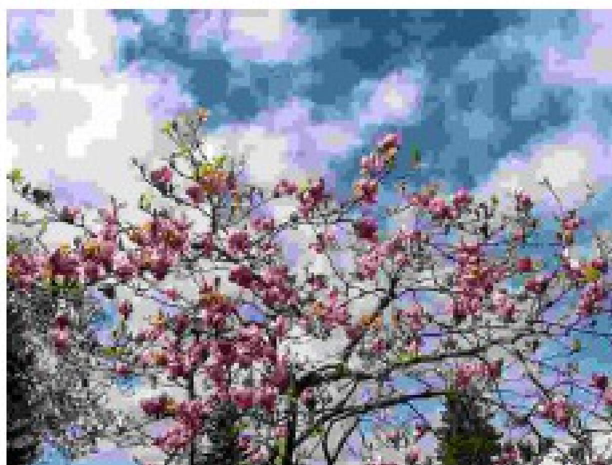
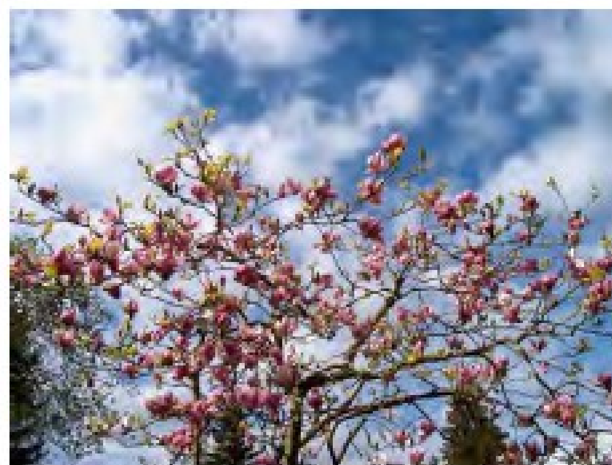
JPEG | JPEG2000

0.75 bpp



(b)

0.25 bpp

(c)

Fig. 9.13 (Cont'd): Comparison of JPEG and JPEG2000. (b) JPEG (left) and JPEG2000 (right) images compressed at 0.75 bpp. (c) JPEG (left) and JPEG2000 (right) images compressed at 0.25 bpp.

Recap

- Lossless Image Compression
  - GIF
  - PNG
  - Lossless JPEG
- JPEG Compression

- Next: Video Compression

- More information: **FM** Ch. 9.