Chapter 9
Image Compression Standards

9.1 The JPEG Standard

- JPEG is an image compression standard that was developed by the "Joint Photographic Experts Group". JPEG was formally accepted as an international standard in 1992.

- JPEG is a **lossy** image compression method. It employs a **transform coding** method using the DCT (*Discrete Cosine Transform*).

- An image is a function of i and j (or conventionally x and y) in the *spatial domain*.

  The 2D DCT is used as one step in JPEG in order to yield a frequency response which is a function $F(u, v)$ in the *spatial frequency domain*, indexed by two integers u and v.

Observations for JPEG Image Compression

- The effectiveness of the DCT transform coding method in JPEG relies on 3 major observations:

Observation 1: Useful image contents change relatively slowly across the image, i.e., it is unusual for intensity values to vary widely several times in a small area, for example, within an 8×8 image block.

- much of the information in an image is repeated, hence "spatial redundancy".

Observations for JPEG Image Compression (cont'd)

Observation 2: Psychophysical experiments suggest that humans are much less likely to notice the loss of very high spatial frequency components than the loss of lower frequency components.

- the spatial redundancy can be reduced by largely reducing the high spatial frequency contents.

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

- chroma subsampling (4:2:0) is used in JPEG.

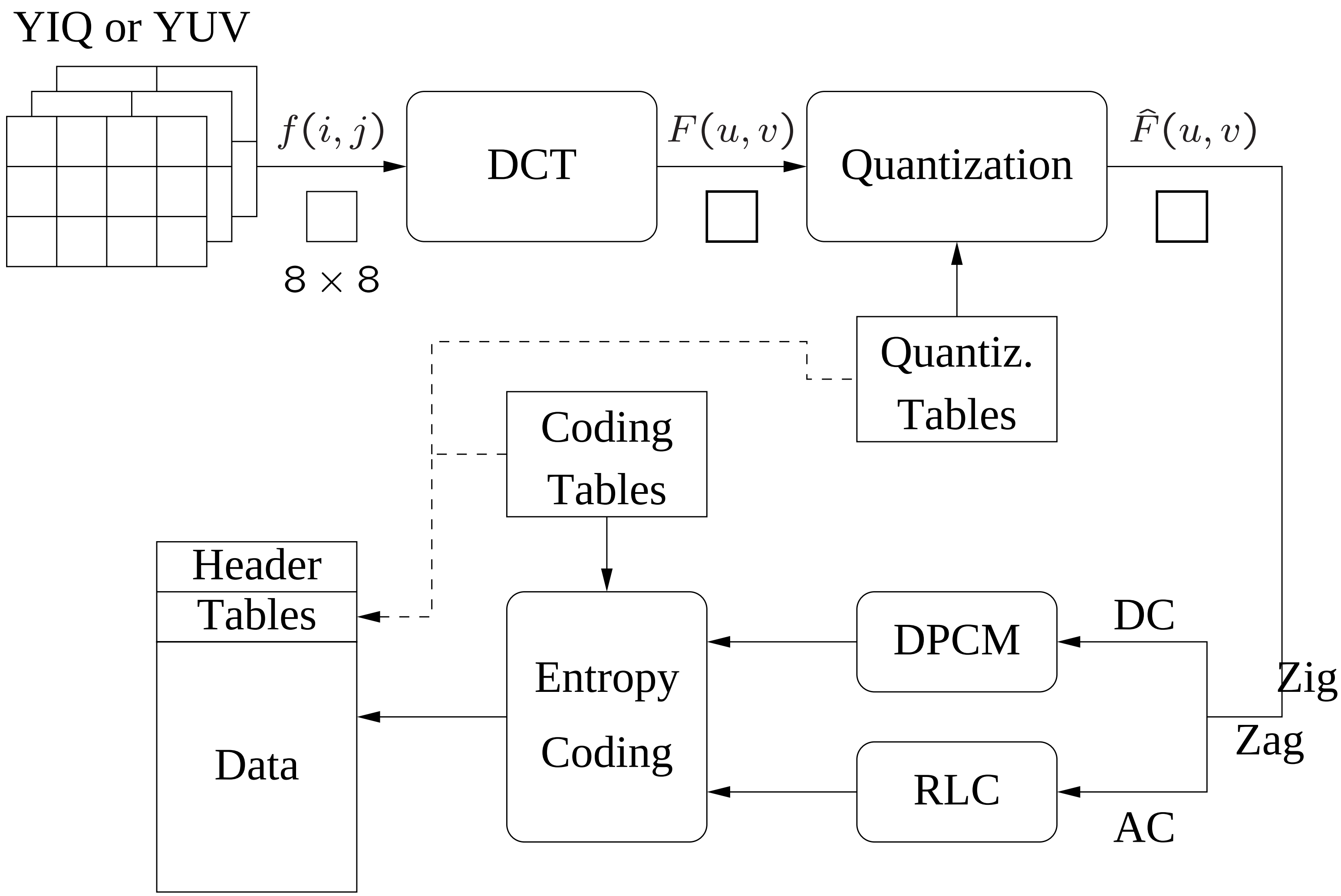

Fig. 9.1: Block diagram for JPEG encoder.

9.1.1 Main Steps in JPEG Image Compression

- Transform RGB to YIQ or YUV and subsample color.
- DCT on image blocks.
- Quantization.
- Zig-zag ordering and run-length encoding.
- Entropy coding.

DCT on image blocks

- Each image is divided into 8×8 blocks. The 2D DCT is applied to each block image $f(i, j)$, with output being the DCT coefficients $F(u, v)$ for each block.

- Using blocks, however, has the effect of isolating each block from its neighboring context. This is why JPEG images look choppy ("blocky") when a high *compression ratio* is specified by the user.

Quantization

$$\hat{F}(u,v) = round\left(\frac{F(u,v)}{Q(u,v)}\right) \tag{9.1}$$

- $F(u,v)$ represents a DCT coefficient, $Q(u,v)$ is a "quantization matrix" entry, and $\hat{F}(u,v)$ represents the *quantized DCT coefficients* which JPEG will use in the succeeding entropy coding.
  - **The quantization step is the main source for loss in JPEG compression**.
  - The entries of $Q(u,v)$ tend to have larger values towards the lower right corner. This aims to introduce more loss at the higher spatial frequencies — a practice supported by Observations 1 and 2.
  - Table 9.1 and 9.2 show the default $Q(u,v)$ values obtained from psychophysical studies with the goal of maximizing the compression ratio while minimizing perceptual losses in JPEG images.

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

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

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.

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

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

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

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

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.

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.

-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

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

-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.

Run-length Coding (RLC) on AC coefficients

- RLC aims to turn the $\widehat{F}(u,v)$ values into sets {*#-zeros-to-skip* , *next non-zero value*}.
- To make it most likely to hit a long run of zeros: a *zig-zag scan* is used to turn the 8×8 matrix $\widehat{F}(u,v)$ into a *64-vector*.

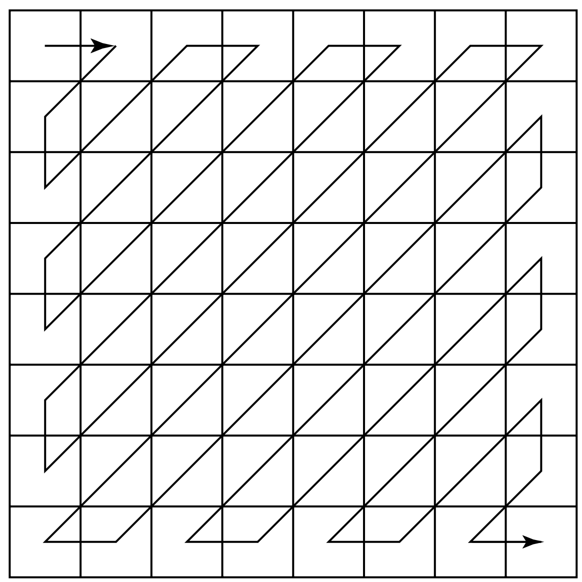

Fig. 9.4: Zig-Zag Scan in JPEG.

DPCM on DC coefficients

- The DC coefficients are coded separately from the AC ones. *Differential Pulse Code Modulation (DPCM)* is the coding method.

- If the DC coefficients for the first 5 image blocks are 150, 155, 149, 152, 144, then the DPCM would produce 150, 5, -6, 3, -8, assuming $d_i = DC_{i+1} - DC_i$, and $d_0 = DC_0$.

Entropy Coding

- The DC and AC coefficients finally undergo an entropy coding step to gain a possible further compression.
- Use DC as an example: each DPCM coded DC coefficient is represented by (SIZE, AMPLITUDE), where SIZE indicates how many bits are needed for representing the coefficient, and AMPLITUDE contains the actual bits.
- In the example we're using, codes 150, 5, -6, 3, -8 will be turned into

$$(8, 10010110), (3, 101), (3, 001), (2, 11), (4, 0111) \;.$$

- SIZE is Huffman coded since smaller SIZEs occur much more often. AMPLITUDE is not Huffman coded, its value can change widely so Huffman coding has no appreciable benefit.

Table 9.3 Baseline entropy coding details – size category.

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

9.1.2 Four Commonly Used JPEG Modes

- Sequential Mode — the default JPEG mode, implicitly assumed in the discussions so far. Each graylevel image or color image component is encoded in a single left-to-right, top-to-bottom scan.
- Progressive Mode.
- Hierarchical Mode.
- Lossless Mode — discussed in Chapter 7, to be replaced by JPEG-LS (Section 9.3).

Progressive Mode

Progressive JPEG delivers low quality versions of the image quickly, followed by higher quality passes.

1. **Spectral selection**: Takes advantage of the "spectral" (spatial frequency spectrum) characteristics of the DCT coefficients: higher AC components provide detail information.

   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 ACs, e.g., AC61, AC62, AC63.

Progressive Mode (Cont'd)

2. **Successive approximation**: Instead of gradually encoding spectral bands, all DCT coefficients are encoded simultaneously but with their most significant bits (MSBs) first.

   Scan 1: Encode the first few MSBs, e.g., Bits 7, 6, 5, 4.

   Scan 2: Encode a few more less significant bits, e.g., Bit 3.

   ⋮

   Scan m: Encode the least significant bit (LSB), Bit 0.

Hierarchical Mode

- The encoded image at the lowest resolution is basically a compressed low-pass filtered image, whereas the images at successively higher resolutions provide additional details (differences from the lower resolution images).

- Similar to Progressive JPEG, the Hierarchical JPEG images can be transmitted in multiple passes progressively improving quality.

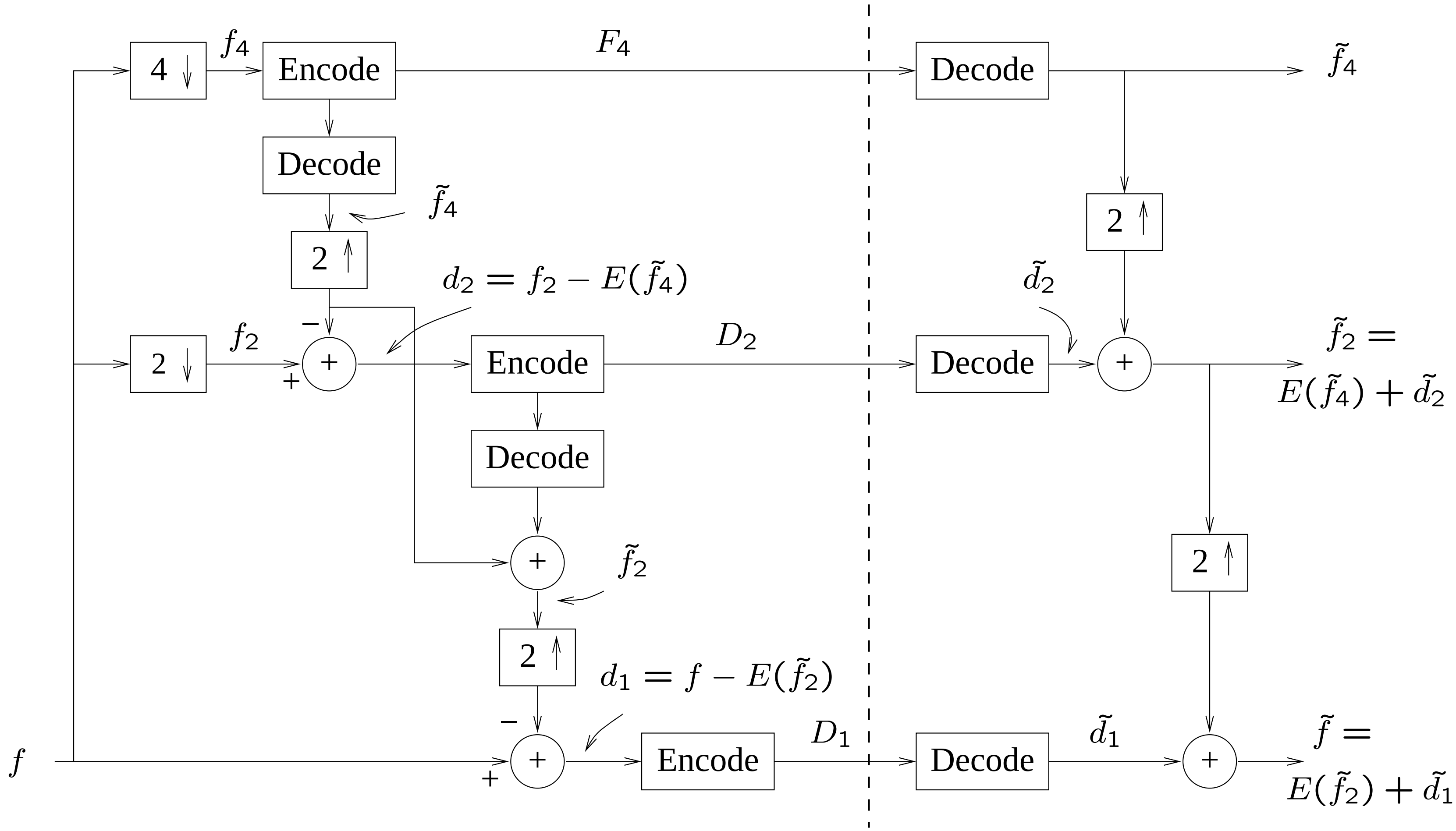

Fig. 9.5: Block diagram for Hierarchical JPEG.

Encoder for a Three-level Hierarchical JPEG

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.

Decoder for a Three-level Hierarchical JPEG

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 $\tilde{f}$.

9.5 Further Explorations

- **Text books:**
  - *The JPEG Still Image Compression Standard* by Pennebaker and Mitchell
  - *JPEG2000: Image Compression Fundamentals, Standards, and Practice* by Taubman and Marcellin
  - *Image and Video Compression Standards: Algorithms and Architectures, 2nd ed.* by Bhaskaren and Konstantinides
- Interactive JPEG demo, and comparison of JPEG and JPEG2000
- **Web sites:** ⟶ Link to Further Exploration for Chapter 9.. including:
  - JPEG and JPEG2000 links, source code, etc.
  - Original paper for the LOCO-I algorithm
  - Introduction and source code for JPEG-LS, JBIG, JBIG2