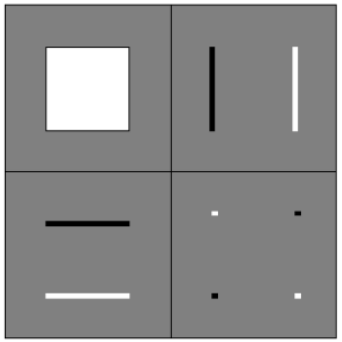
Multimedia Computing

Image Compression:

Part 2



Topics

- Lossless image compression
- Lossy image compression
  - Distortion and Quantization
  - Transform based coding
  - Wavelet based coding
- JPEG Standard

Wavelet-Based Coding

- The objective of the wavelet transform (WT) is to decompose the input signal into components that are easier to deal with, or can be thresholded away, for compression purposes.
- We want to be able to at least approximately reconstruct the original signal given these components.
- The basis functions of DCT are cosine wave, which is a long wave and localized in frequency domain. The basis functions of the wavelet transform are localized in both time and frequency.

An example: Haar wavelet transform

- Suppose we are given the following input sequence

$$\{x_{n,i}\} = \{10, 13, 25, 26, 29, 21, 7, 15\}$$

  where i=0,1,2,…
- Consider the transform that replaces the original sequence with its pairwise *average* $x_{n-1,i}$ and *difference* $d_{n-1,i}$ defined as follows:

$$\begin{aligned} x_{n-1,i} &= \frac{x_{n,2i} + x_{n,2i+1}}{2} \\ d_{n-1,i} &= \frac{x_{n,2i} - x_{n,2i+1}}{2} \end{aligned}$$

An example: Haar wavelet transform

- Suppose the length of original signal is even, the number of elements in each set $\{x_{n-1,i}\}$ and $\{d_{n-1,i}\}$ is exactly half of the number of elements in the original sequence.
- By concatenating the two sequences, the resulting sequence has the same length with the original sequence:

$$\{x_{n-1,i}, d_{n-1,i}\} = \{11.5, 25.5, 25, 11, -1.5, -0.5, 4, -4\}$$

- Since the first half of the above sequence contain averages from the original sequence, we can view it as a coarser approximation to the original signal. The second half of this sequence can be viewed as the details or approximation errors of the first half.

An example: Haar wavelet transform

- We can further apply the same transform to $\{x_{n-1,i}\}$, and can obtain the second level approximation $x_{n-2,i}$ and $d_{n-2,i}$

  $\{x_{n-2,i}, d_{n-2,i}, d_{n-1,i}\}=\{18.5,18,-7,7,-1.5,-0.5,4,-4\}$

  - This is the essential idea of multiresolution (multi-scale) analysis.
- For this eight element sequence, we can make a 3 (because $8=2^3$) scale decomposition:

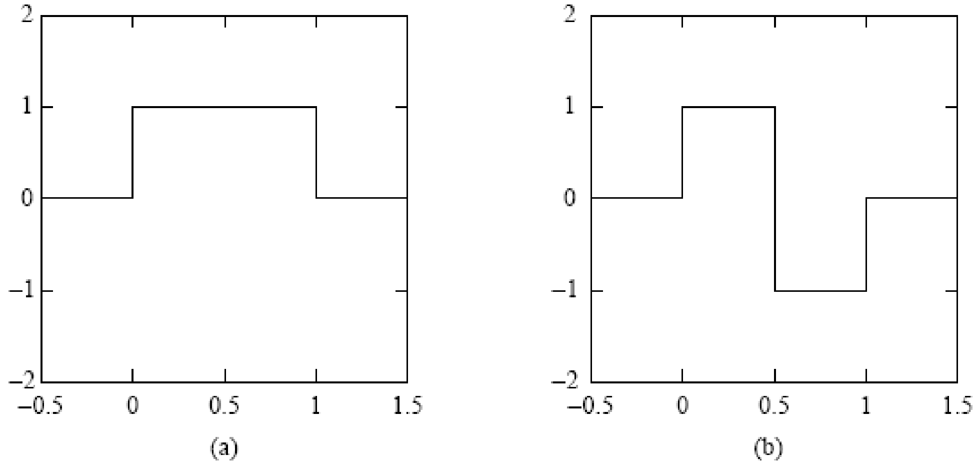
  $\{x_{n-3,i}, d_{n-3,i}, d_{n-2,i}, d_{n-1,i}\}=\{18.25,0.25,-7,7,-1.5,-0.5,4,-4\}$

An example: Haar wavelet transform

- It is easily verified that the original sequence can be reconstructed from the transformed sequence using the relations

$$x_{n,2i} = x_{n-1,i} + d_{n-1,i}$$
$$x_{n,2i+1} = x_{n-1,i} - d_{n-1,i}$$

- This transform is actually the discrete Haar wavelet transform



Haar Transform: (a) scaling function, (b) wavelet function.

Haar wavelet transform: reconstruction

- We are given:

  $\{x_{n-3,i}, d_{n-3,i}, d_{n-2,i}, d_{n-1,i}\}=\{18.25,0.25,-7,7,-1.5,-0.5,4,-4\}$

- We first reconstruct $x_{n-2,i}$ from $x_{n-3,I}$ and $d_{n-3,i}$

  $x_{n-2,2i} = x_{n-3,i} + d_{n-3,i} = 18.5$

  $x_{n-2,2i+1} = x_{n-3,i} - d_{n-3,i} = 18$

- Now the sequence is

  $\{x_{n-2,i}, d_{n-2,i}, d_{n-1,i}\}=\{18.5,18,-7,7,-1.5,-0.5,4,-4\}$

Haar wavelet transform: reconstruction

$\{x_{n-2,i}, d_{n-2,i}, d_{n-1,i}\}=\{18.5,18,-7,7,-1.5,-0.5,4,-4\}$

- We then reconstruct $x_{n-1,i}$ from $x_{n-2,i}$ and $d_{n-2,i}$

  $x_{n-1,2i}(1) = x_{n-2,i}(1) + d_{n-2,i}(1) = 18.5+(-7)=11.5$

  $x_{n-1,2i+1}(1) = x_{n-2,i}(1) - d_{n-2,i}(1) = 18.5-(-7)=25.5$

  $x_{n-1,2i}(2) = x_{n-2,i}(2) + d_{n-2,i}(2) = 18+7=25$

  $x_{n-1,2i+1}(2) = x_{n-2,i}(2) - d_{n-2,i}(2) = 18-7=11$

- Now the sequence is

  $\{x_{n-1,i}, d_{n-1,i}\}=\{11.5,25.5, 25,11,-1.5,-0.5,4,-4\}$

- Finally, we can reconstruct the whole original sequence as

$$\{x_{n,i}\} = \{10, 13, 25, 26, 29, 21, 7, 15\}$$

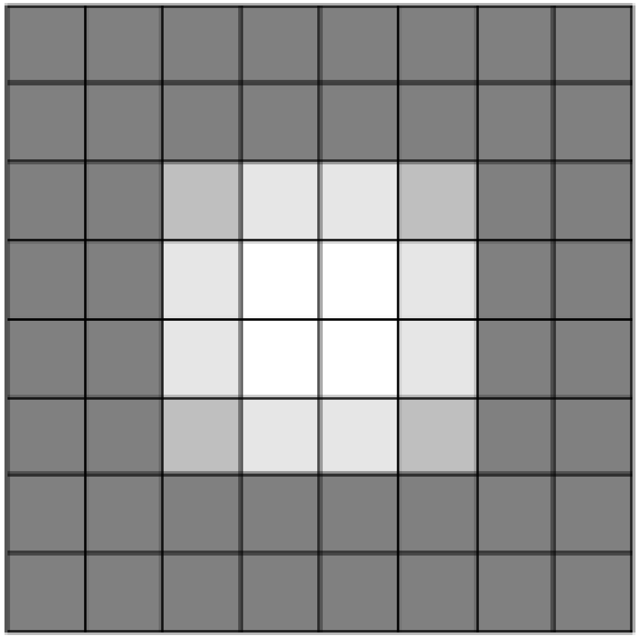
2D Haar WT

- Extending the 1D Haar discrete WT (DWT) to 2D case is very simple:
  1. We first apply the 1D transform to each row of the original image;
  2. We then apply 1D transform to each column of the row-transformed image;
  3. Such a procedure can be iteratively implemented for multi-scale 2D Haar DWT.

2D Haar WT: example

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	0	63	127	127	63	0	0
0	0	127	255	255	127	0	0
0	0	127	255	255	127	0	0
0	0	63	127	127	63	0	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

The pixel values



Shown as an 8×8 image

2D Haar WT: example

0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	95	95	0	0	−32	32	0
0	191	191	0	0	−64	64	0
0	191	191	0	0	−64	64	0
0	95	95	0	0	−32	32	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0

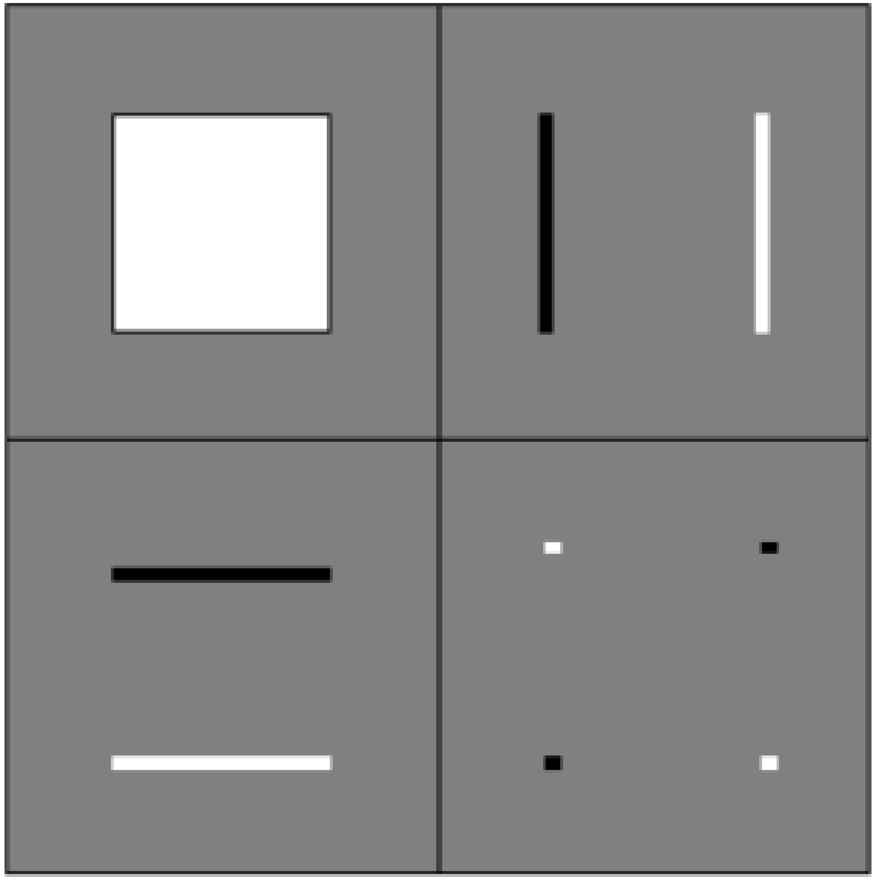
Intermediate output: 1st level, step 1 -- row transform

2D Haar WT: example

0	0	0	0	0	0	0	0
0	143	143	0	0	−48	48	0
0	143	143	0	0	−48	48	0
0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0
0	−48	−48	0	0	16	−16	0
0	48	48	0	0	−16	16	0
0	0	0	0	0	0	0	0

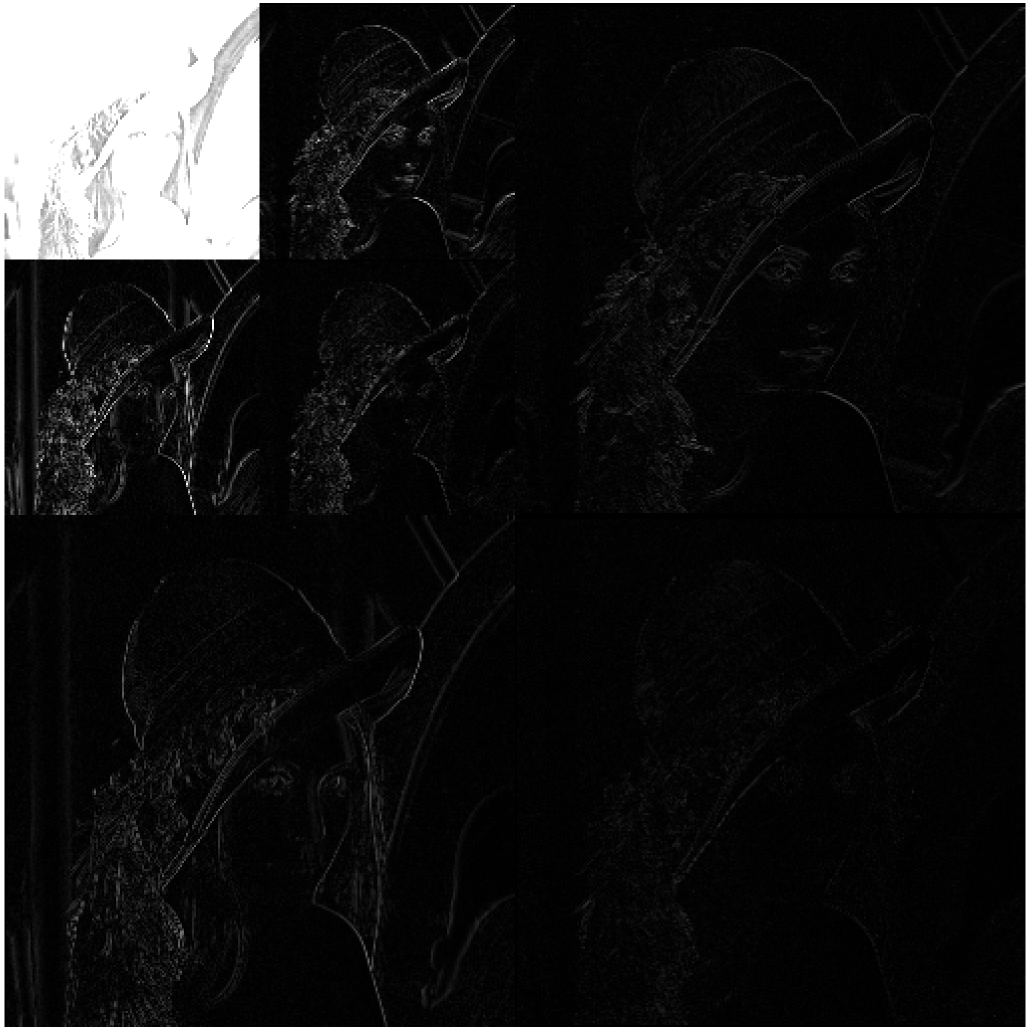
Intermediate output: 1st level, step 2 -- column transform

2D Haar WT: a simple graphical illustration



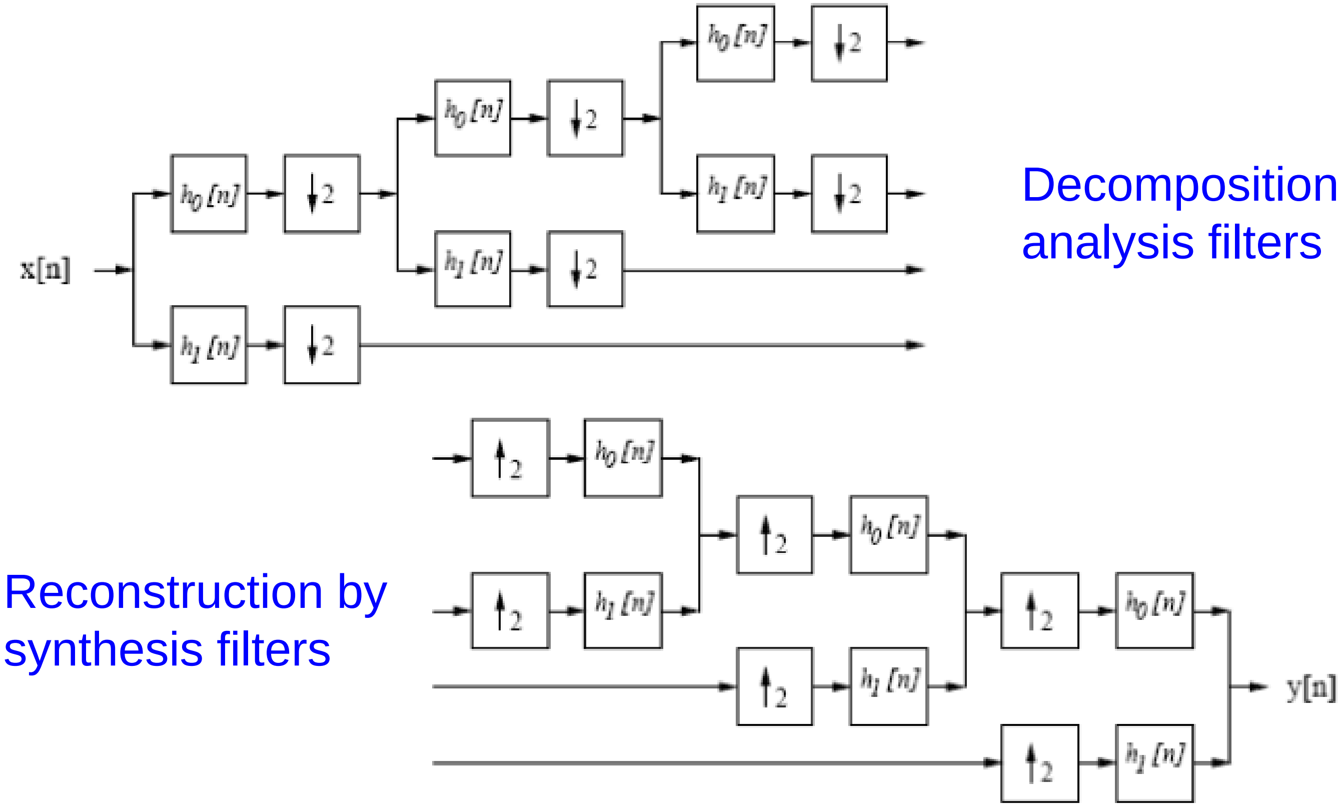
One level decomposition of a white square in grey background

2D Haar WT: Lena



Two-level Haar WT decomposition of the image Lena

1D discrete wavelet transform (DWT) *(optional)*



Decomposition by analysis filters

Reconstruction by synthesis filters

- In orthogonal DWT, analysis filters h_0 and h_1 are identical to synthesis filters.

Orthogonal wavelet filters *(optional)*

Wavelet	Num. Taps	Start Index	Coefficients
Haar	2	0	[0.707, 0.707]
Daubechies 4	4	0	[0.483, 0.837, 0.224, -0.129]
Daubechies 6	6	0	[0.332, 0.807, 0.460, -0.135, -0.085, 0.0352]
Daubechies 8	8	0	[0.230, 0.715, 0.631, -0.028, -0.187, 0.031, 0.033, -0.011]

Bi-orthogonal wavelets *(optional)*

- For orthogonal wavelets, the forward transform and its inverse are transposes of each other and the analysis filters are identical to the synthesis filters.
- Without orthogonality, the wavelets for analysis and synthesis are called "bi-orthogonal". The synthesis filters are not identical to the analysis filters. To guarantee the perfect reconstruction, we require

$$h_1[n] = (-1)^n \tilde{h}_0[1-n]$$

$$\tilde{h}_1[n] = (-1)^n h_0[1-n]$$

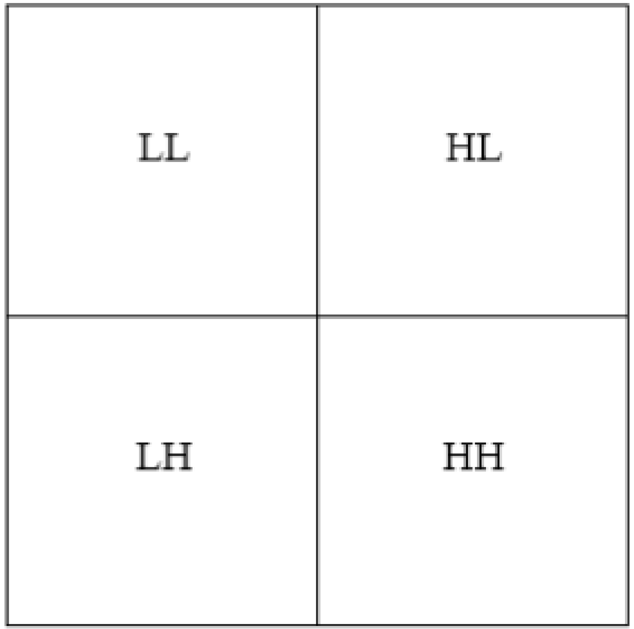
Bi-orthogonal wavelet filters *(optional)*

Wavelet	Filter	Num. Taps	Start Index	Coefficients
Antonini 9/7	$h_0[n]$	9	-4	[0.038, -0.024, -0.111, 0.377, 0.853, 0.377, -0.111, -0.024, 0.038]
	$\tilde{h}_0[n]$	7	-3	[-0.065, -0.041, 0.418, 0.788, 0.418, -0.041, -0.065]
Villa 10/18	$h_0[n]$	10	-4	[0.029, 0.0000824, -0.158, 0.077, 0.759, 0.759, 0.077, -0.158, 0.0000824, 0.029]
	$\tilde{h}_0[n]$	18	-8	[0.000954, -0.00000273, -0.009, -0.003, 0.031, -0.014, -0.086, 0.163, 0.623, 0.623, 0.163, -0.086, -0.014, 0.031, -0.003, -0.009, -0.00000273, 0.000954]
Brislawn	$h_0[n]$	10	-4	[0.027, -0.032, -0.241, 0.054, 0.900, 0.900, 0.054, -0.241, -0.032, 0.027]
	$\tilde{h}_0[n]$	10	-4	[0.020, 0.024, -0.023, 0.146, 0.541, 0.541, 0.146, -0.023, 0.024, 0.020]

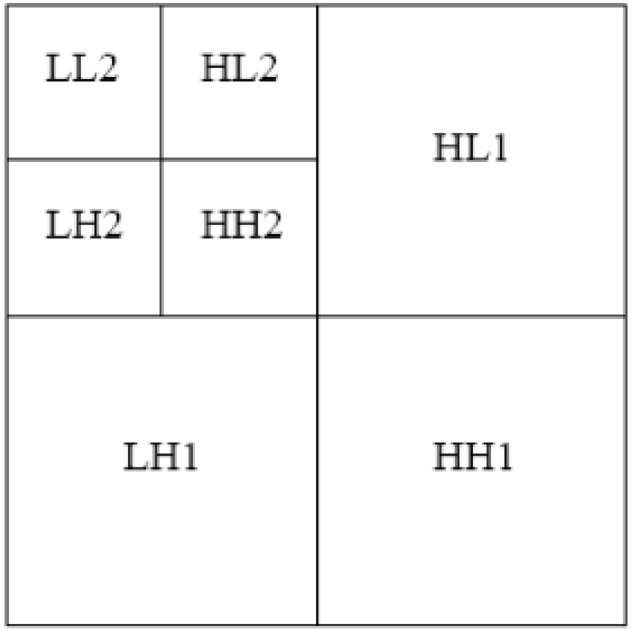
2D WT implementation *(optional)*

- For an N by N input image, the two-dimensional DWT proceeds as follows:
  - Convolve each row of the image with $h_0[n]$ and $h_1[n]$, discard the odd numbered columns of the resulting arrays, and concatenate them to form a transformed row.
  - After all rows have been transformed, convolve each column of the result with $h_0[n]$ and $h_1[n]$. Again discard the odd numbered rows and concatenate the result.
- After the above two steps, one stage of the DWT is complete. The transformed image now contains four subbands LL, HL, LH, and HH, standing for low-low, high-low, etc.
- The LL subband can be further decomposed to yield yet another level of decomposition. This process can be continued until the desired number of decomposition levels is reached.

2D DWT *(optional)*



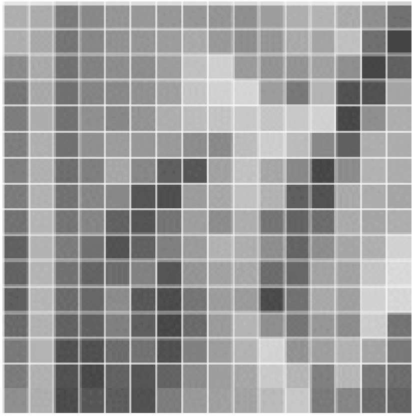
One level 2D DWT



Two level 2D DWT

Example *(optional)*

Input 16×16 image



$$I_{00}(x,y) = \begin{bmatrix}
158 & 170 & 97 & 104 & 123 & 130 & 133 & 125 & 132 & 127 & 112 & 158 & 159 & 144 & 116 & 91 \\
164 & 153 & 91 & 99 & 124 & 152 & 131 & 160 & 189 & 116 & 106 & 145 & 140 & 143 & 227 & 53 \\
116 & 149 & 90 & 101 & 118 & 118 & 131 & 152 & 202 & 211 & 84 & 154 & 127 & 146 & 58 & 58 \\
95 & 145 & 88 & 105 & 188 & 123 & 117 & 182 & 185 & 204 & 203 & 154 & 153 & 229 & 46 & 147 \\
101 & 156 & 89 & 100 & 165 & 113 & 148 & 170 & 163 & 186 & 144 & 194 & 208 & 39 & 113 & 159 \\
103 & 153 & 94 & 103 & 203 & 136 & 146 & 92 & 66 & 192 & 188 & 103 & 178 & 47 & 167 & 159 \\
102 & 146 & 106 & 99 & 99 & 121 & 39 & 60 & 164 & 175 & 198 & 46 & 56 & 56 & 156 & 156 \\
99 & 146 & 95 & 97 & 144 & 61 & 103 & 107 & 108 & 111 & 192 & 62 & 65 & 128 & 153 & 154 \\
99 & 140 & 103 & 109 & 103 & 124 & 54 & 81 & 172 & 137 & 178 & 54 & 43 & 159 & 149 & 174 \\
84 & 133 & 107 & 84 & 149 & 43 & 158 & 95 & 151 & 120 & 183 & 46 & 30 & 147 & 142 & 201 \\
58 & 153 & 110 & 41 & 94 & 213 & 71 & 73 & 140 & 103 & 138 & 83 & 152 & 143 & 128 & 207 \\
56 & 141 & 108 & 58 & 92 & 51 & 55 & 61 & 88 & 166 & 58 & 103 & 146 & 150 & 116 & 211 \\
89 & 115 & 188 & 47 & 113 & 104 & 56 & 67 & 128 & 155 & 187 & 71 & 153 & 134 & 203 & 95 \\
35 & 99 & 151 & 67 & 35 & 88 & 88 & 128 & 140 & 142 & 176 & 213 & 144 & 128 & 214 & 100 \\
89 & 98 & 97 & 51 & 49 & 101 & 47 & 90 & 136 & 136 & 157 & 205 & 106 & 43 & 54 & 76 \\
44 & 105 & 69 & 69 & 68 & 53 & 110 & 127 & 134 & 146 & 159 & 184 & 109 & 121 & 72 & 113
\end{bmatrix}$$

The Antonini 9/7 filter set is used. We have

Antonini 9/7	$h_0[n]$	[0.038, -0.024, -0.111, 0.377, 0.853, 0.377, -0.111, -0.024, 0.038]
	$\tilde{h}_0[n]$	[-0.065, -0.041, 0.418, 0.788, 0.418, -0.041, -0.065]

$$\begin{aligned}
h_1[n] &= [-0.065, 0.041, 0.418, -0.788, 0.418, 0.041, -0.065] \\
\tilde{h}_1[n] &= [-0.038, -0.024, 0.111, 0.377, -0.853, 0.377, \\
&\quad 0.111, -0.024, -0.038]
\end{aligned}$$

Example *(optional)*

- Convolve the first row with both $h_0[n]$ and $h_1[n]$ and discarding the values with odd-numbered index. The results of these two operations are:

$$
\begin{aligned}
(I_{00}(:,0) * h_0[n]) \downarrow 2 &= [245, 156, 171, 183, 184, 173, 228; 160], \\
(I_{00}(:,0) * h_1[n]) \downarrow 2 &= [-30, 3, 0, 7, -5, -16, -3, 16].
\end{aligned}
$$

- Form the transformed output row by concatenating the resulting coefficients. The first row of the transformed image is then:

$$[245, 156, 171, 183, 184, 173, 228, 160, -30, 3, 0, 7, -5, -16, -3, 16]$$

- Continue the same process for the remaining rows.

Example *(optional)*

The result after all rows have been processed

$$I_{10}(x, y) =$$

$$\begin{bmatrix}
245 & 156 & 171 & 183 & 184 & 173 & 228 & 160 & -30 & 3 & 0 & 7 & -5 & -16 & -3 & 16 \\
239 & 141 & 181 & 197 & 242 & 158 & 202 & 229 & -17 & 5 & -20 & 3 & 26 & -27 & 27 & 141 \\
195 & 147 & 163 & 177 & 288 & 173 & 209 & 106 & -34 & 2 & 2 & 19 & -50 & -35 & -38 & -1 \\
180 & 139 & 226 & 177 & 274 & 267 & 247 & 163 & -45 & 29 & 24 & -29 & -2 & 30 & -101 & -78 \\
191 & 145 & 197 & 198 & 247 & 230 & 239 & 143 & -49 & 22 & 36 & -11 & -26 & -14 & 101 & -54 \\
192 & 145 & 237 & 184 & 135 & 253 & 169 & 192 & -47 & 38 & 36 & 4 & -58 & 66 & 94 & -4 \\
176 & 159 & 156 & 77 & 204 & 232 & 51 & 196 & -31 & 9 & -48 & 30 & 11 & 58 & 29 & 4 \\
179 & 148 & 162 & 129 & 146 & 213 & 92 & 217 & -39 & 18 & 50 & -10 & 33 & 51 & -23 & 8 \\
169 & 159 & 163 & 97 & 204 & 202 & 85 & 234 & -29 & 1 & -42 & 23 & 37 & 41 & -56 & -5 \\
155 & 153 & 149 & 159 & 176 & 204 & 65 & 236 & -32 & 32 & 85 & 39 & 38 & 44 & -54 & -31 \\
145 & 148 & 158 & 148 & 164 & 157 & 188 & 215 & -55 & 59 & -110 & 28 & 26 & 48 & -1 & -64 \\
134 & 152 & 102 & 70 & 153 & 126 & 199 & 207 & -47 & 38 & 13 & 10 & -76 & 3 & -7 & -76 \\
127 & 203 & 130 & 94 & 171 & 218 & 171 & 228 & 12 & 88 & -27 & 15 & 1 & 76 & 24 & 85 \\
70 & 188 & 63 & 144 & 191 & 257 & 215 & 232 & -5 & 24 & -28 & -9 & 19 & -46 & 36 & 91 \\
129 & 124 & 87 & 96 & 177 & 236 & 162 & 77 & -2 & 20 & -48 & 1 & 17 & -56 & 30 & -24 \\
103 & 115 & 85 & 142 & 188 & 234 & 184 & 132 & -37 & 0 & 27 & -4 & 5 & -35 & -22 & -33
\end{bmatrix}$$

Example *(optional)*

- Apply the filters to the columns of the resulting image. Apply both $h_0[n]$ and $h_1[n]$ to each column and discard the odd indexed results:

$$
\begin{aligned}
(I_{11}(0,:) * h_0[n]) \downarrow 2 &= [353, 280, 269, 256, 240, 206, 160, 153]^T \\
(I_{11}(0,:) * h_1[n]) \downarrow 2 &= [-12, 10, -7, -4, 2, -1, 43, 16]^T
\end{aligned}
$$

- Concatenate the above results into a single column and apply the same procedure to each of the remaining columns.

$$
I_{11}(x,y) =
$$

$$
\begin{bmatrix}
353 & 212 & 251 & 272 & 281 & 234 & 308 & 289 & -33 & 6 & -15 & 5 & 24 & -29 & 38 & 120 \\
280 & 203 & 254 & 250 & 402 & 269 & 297 & 207 & -45 & 11 & -2 & 9 & -31 & -26 & -74 & 23 \\
269 & 202 & 312 & 280 & 316 & 353 & 337 & 227 & -70 & 43 & 56 & -23 & -41 & 21 & 82 & -81 \\
256 & 217 & 247 & 155 & 236 & 328 & 114 & 283 & -52 & 27 & -14 & 23 & -2 & 90 & 49 & 12 \\
240 & 221 & 226 & 172 & 264 & 294 & 113 & 330 & -41 & 14 & 31 & 23 & 57 & 60 & -78 & -3 \\
206 & 204 & 201 & 192 & 230 & 219 & 232 & 300 & -76 & 67 & -53 & 40 & 4 & 46 & -18 & -107 \\
160 & 275 & 150 & 135 & 244 & 294 & 267 & 331 & -2 & 90 & -17 & 10 & -24 & 49 & 29 & 89 \\
153 & 189 & 113 & 173 & 260 & 342 & 256 & 176 & -20 & 18 & -38 & -4 & 24 & -75 & 25 & -5 \\
-12 & 7 & -9 & -13 & -6 & 11 & 12 & -69 & -10 & -1 & 14 & 6 & -38 & 3 & -45 & -99 \\
10 & 3 & -31 & 16 & -1 & -51 & -10 & -30 & 2 & -12 & 0 & 24 & -32 & -45 & 109 & 42 \\
-7 & 5 & -44 & -35 & 67 & -10 & -17 & -15 & 3 & -15 & -28 & 0 & 41 & -30 & -18 & -19 \\
-4 & 9 & -1 & -37 & 41 & 6 & -33 & 2 & 9 & -12 & -67 & 31 & -7 & 3 & 2 & 0 \\
2 & -3 & 9 & -25 & 2 & -25 & 60 & -8 & -11 & -4 & -123 & -12 & -6 & -4 & 14 & -12 \\
-1 & 22 & 32 & 46 & 10 & 48 & -11 & 20 & 19 & 32 & -59 & 9 & 70 & 50 & 16 & 73 \\
43 & -18 & 32 & -40 & -13 & -23 & -37 & -61 & 8 & 22 & 2 & 13 & -12 & 43 & -8 & -45 \\
16 & 2 & -6 & -32 & -7 & 5 & -13 & -50 & 24 & 7 & -61 & 2 & 11 & -33 & 43 & 1
\end{bmatrix}
$$

Example *(optional)*

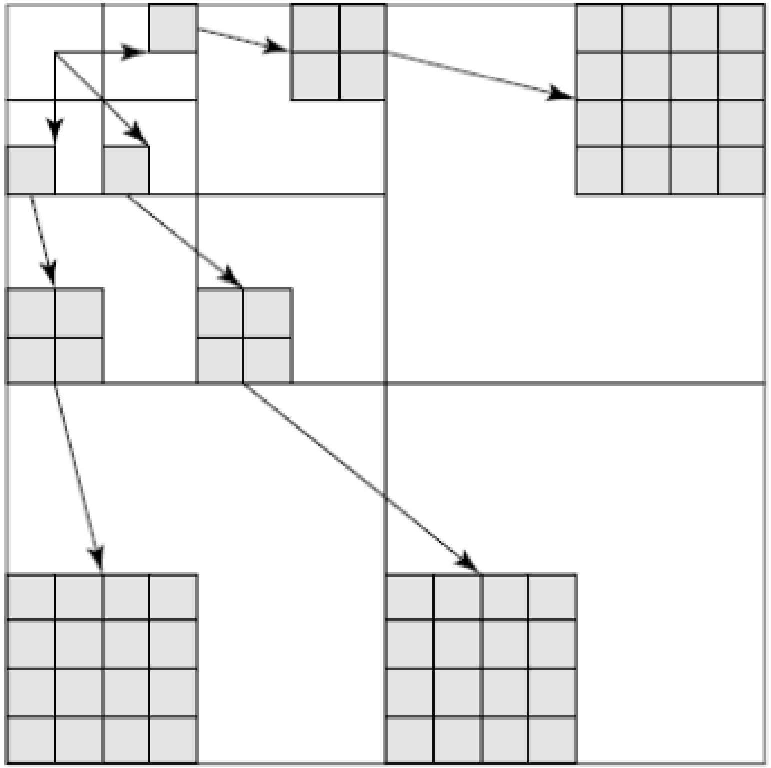
We can perform another stage of the DWT by applying the same transform procedure illustrated above to the upper left 8×8 DC image of $I_{12}(x, y)$. The resulting two-stage transformed image is

$$I_{22}(x, y) =$$

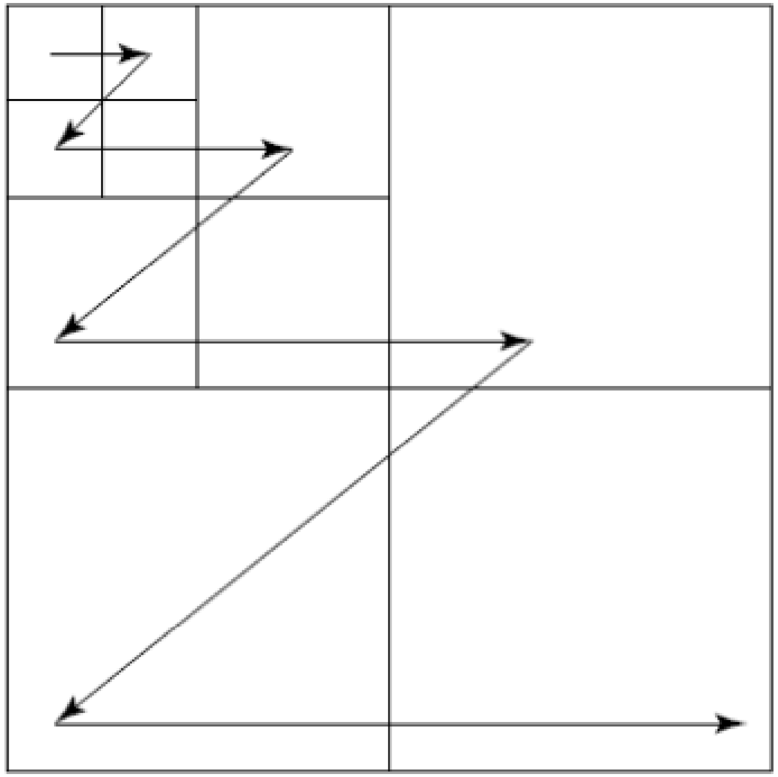
$$\begin{bmatrix}
558 & 451 & 608 & 532 & 75 & 26 & 94 & 25 & -33 & 6 & -15 & 5 & 24 & -29 & 38 & 120 \\
463 & 511 & 627 & 566 & 66 & 68 & -43 & 68 & -45 & 11 & -2 & 9 & -31 & -26 & -74 & 23 \\
464 & 401 & 478 & 416 & 14 & 84 & -97 & -229 & -70 & 43 & 56 & -23 & -41 & 21 & 82 & -81 \\
422 & 335 & 477 & 553 & -88 & 46 & -31 & -6 & -52 & 27 & -14 & 23 & -2 & 90 & 49 & 12 \\
14 & 33 & -56 & 42 & 22 & -43 & -36 & 1 & -41 & 14 & 31 & 23 & 57 & 60 & -78 & -3 \\
-13 & 36 & 54 & 52 & 12 & -21 & 51 & 70 & -76 & 67 & -53 & 40 & 4 & 46 & -18 & -107 \\
25 & -20 & 25 & -7 & -35 & 35 & -56 & -55 & -2 & 90 & -17 & 10 & -24 & 49 & 29 & 89 \\
46 & 37 & -51 & 51 & -44 & 26 & 39 & -74 & -20 & 18 & -38 & -4 & 24 & -75 & 25 & -5 \\
-12 & 7 & -9 & -13 & -6 & 11 & 12 & -69 & -10 & -1 & 14 & 6 & -38 & 3 & -45 & -99 \\
10 & 3 & -31 & 16 & -1 & -51 & -10 & -30 & 2 & -12 & 0 & 24 & -32 & -45 & 109 & 42 \\
-7 & 5 & -44 & -35 & 67 & -10 & -17 & -15 & 3 & -15 & -28 & 0 & 41 & -30 & -18 & -19 \\
-4 & 9 & -1 & -37 & 41 & 6 & -33 & 2 & 9 & -12 & -67 & 31 & -7 & 3 & 2 & 0 \\
2 & -3 & 9 & -25 & 2 & -25 & 60 & -8 & -11 & -4 & -123 & -12 & -6 & -4 & 14 & -12 \\
-1 & 22 & 32 & 46 & 10 & 48 & -11 & 20 & 19 & 32 & -59 & 9 & 70 & 50 & 16 & 73 \\
43 & -18 & 32 & -40 & -13 & -23 & -37 & -61 & 8 & 22 & 2 & 13 & -12 & 43 & -8 & -45 \\
16 & 2 & -6 & -32 & -7 & 5 & -13 & -50 & 24 & 7 & -61 & 2 & 11 & -33 & 43 & 1
\end{bmatrix}$$

Embedded Zerotree of Wavelet Coefficients

- The embedded zerotree wavelet (EZW) algorithm is effective and computationally efficient for image coding.
- The EZW algorithm addresses two problems:
  - 1. Obtaining the best image quality for a given bit-rate, and
  - 2. Accomplishing this task in an embedded fashion.
- Using an embedded code allows the encoder to terminate the encoding at any point. Hence, the encoder is able to meet any target bit-rate exactly.
- Similarly, a decoder can cease to decode at any point and can produce reconstructions corresponding to all lower-rate encodings.



Parent child relationship
in a zerotree



EZW scanning order

The Zerotree Data Structure

- The EZW algorithm efficiently codes the "significance map" which indicates the locations of nonzero quantized wavelet coefficients.
- This is achieved using a new data structure called the *zerotree*.
- Using the hierarchical wavelet decomposition presented earlier, we can relate every coefficient at a given scale to a set of coefficients at the next finer scale of similar orientation.
- The coefficient at the coarse scale is called the "parent" while all corresponding coefficients at the next finer scale of the same spatial location and similar orientation are called "children".

The Zerotree Data Structure

- Given a threshold T, a coefficient x is an element of the zerotree if it is insignificant and all of its descendants are insignificant as well.
- The significance map is coded using the zerotree with a four symbol alphabet:
  - **The zerotree root**: The root of the zerotree is encoded with a special symbol indicating that the insignificance of the coefficients at finer scales is completely predictable.
  - **Isolated zero**: The coefficient is insignificant but has some significant descendants.
  - **Positive significance**: The coefficient is significant with a positive value.
  - **Negative significance**: The coefficient is significant with a negative value.

Topics

- Lossless image compression
- Lossy image compression
  - Distortion and Quantization
  - Transform based coding
  - Wavelet based coding
- JPEG Standard

The JPEG Standard

- JPEG is an image compression standard that was developed by the “Joint Photographic Experts Group". It 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 in JPEG in order to yield a frequency response which is a function $F(u, v)$ in the *frequency domain*, indexed by two integers *u* and *v*.

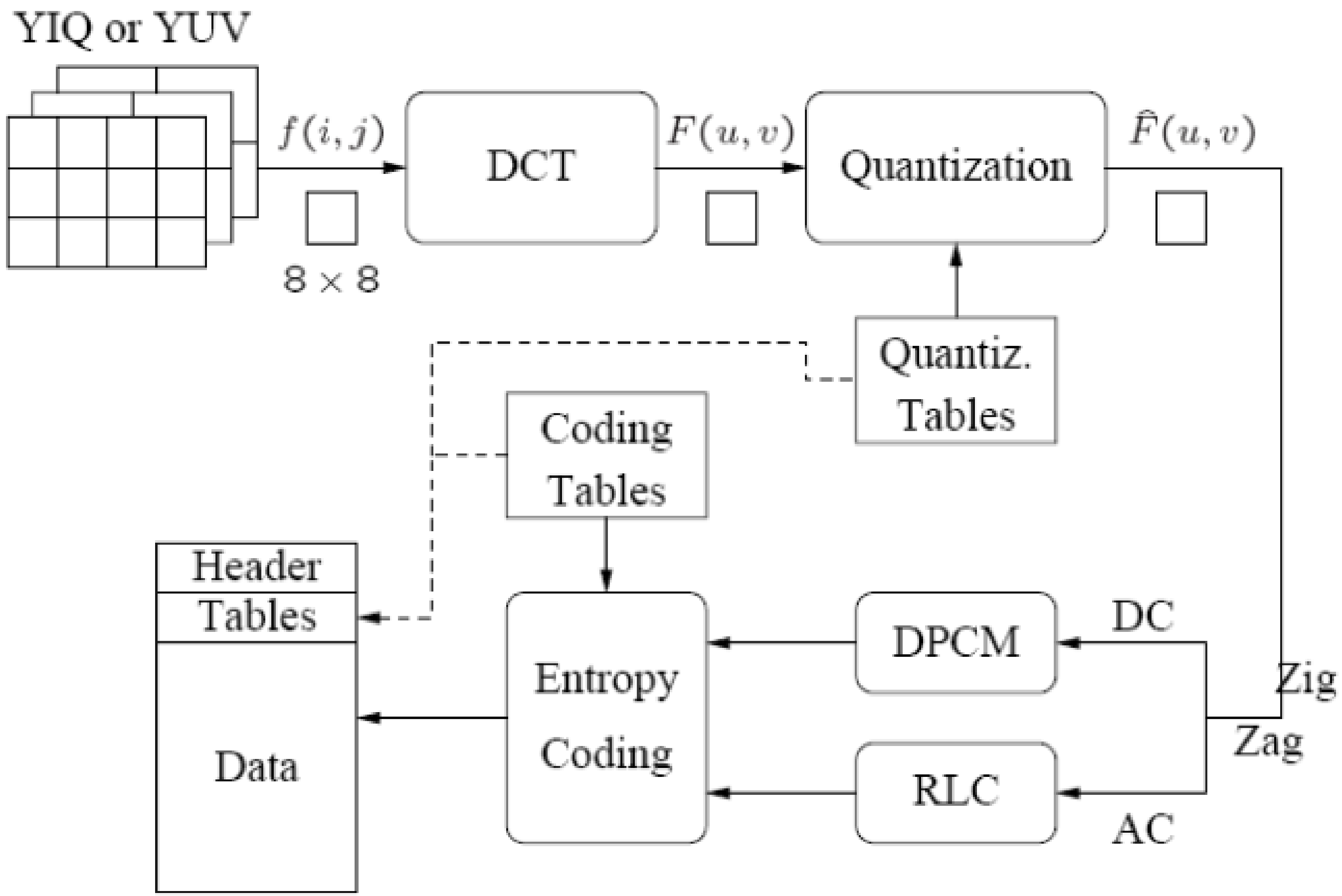
Observations for JPEG Compression

- 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 Compression

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

Block diagram for JPEG encoder



Note: We will discuss YIQ, YUV color models and chroma subsampling in another lecture

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 neighbouring context. This is why JPEG images look blocky when a high *compression ratio* is specified by the user.

Quantization

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

- $F(u,v)$ 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.

Quantization Tables Q(u,v)

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

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

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

$F(u,v)$

JPEG compression for a smooth image block

Example

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

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

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

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

$\tilde{F}(u,v)$ is the de-quantized DCT coefficients.

Another example

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

$F(u,v)$

JPEG compression for a textured image block

Another example

-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

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

-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

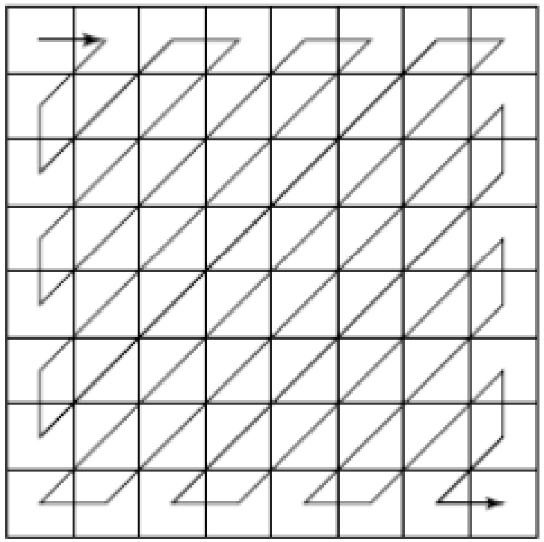
$\tilde{f}(i,j)$

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

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

Run-length Coding on AC coefficients

- RLC aims to turn the $\hat{F}(u, v)$ 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 $\hat{F}(u, v)$ into a *64-vector*.



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). (Note that for negative values the “0” and “1” will be exchanged.)
- SIZE is Huffman coded since smaller SIZEs occur much more often. AMPLITUDE is not Huffman coded because its value can change widely so Huffman coding has no appreciable benefit.

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

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 (JPEG-LS).

References

- Ze-Nian Li, M. S. Drew, *Fundamentals of Multimedia*, Prentice Hall Inc., 2004. Chapters 8 and 9.