Image Compression

Dr. Zein Al Abidin IBRAHIM

ibrahim.zein@gmail.com

IN433
Multimedia Processing

Multimedia image data	Grayscale image	Color image	HDTV video frame	Medical image	Super High Definition (SHD) image
Size/duration	512 × 512	512 × 512	1280 × 720	2048 × 1680	2048 × 2048
Bits/pixel or bits/sample	8 bpp	24 bpp	12 bpp	12 bpp	24 bpp
Uncompressed size (B for bytes)	262 KB	786 KB	1.3 MB	5.16 MB	12.58 MB
Transmission bandwidth (b for bits)	2.1 Mb/image	6.29 Mb/ image	8.85 Mb/ frame	41.3 Mb/ image	100 Mb/ image
Transmission time (56 K modem)	42 seconds	110 seconds	158 seconds	12 min.	29 min.
Transmission time (780 Kb DSL)	3 seconds	7.9 seconds	11.3 seconds	51.4 seconds	2 min.

Why compress?

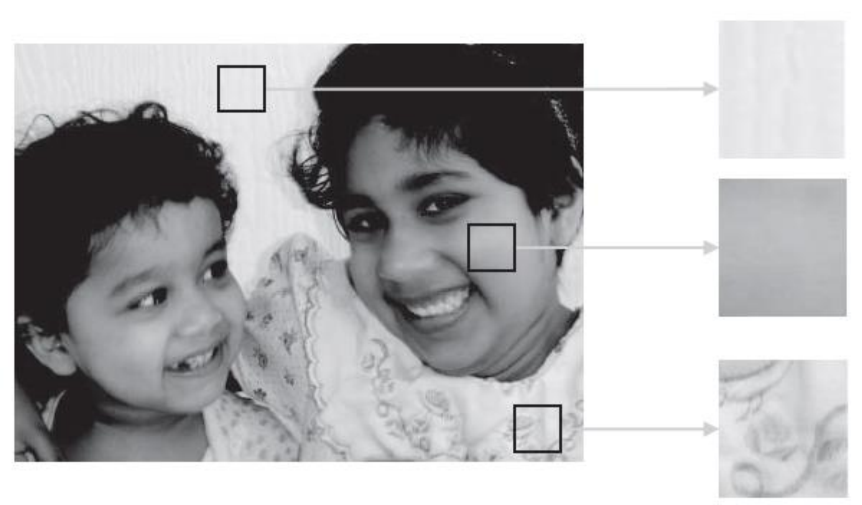
I. Irrelevancy reduction:

- Information associated with some pixels might be irrelevant and can be removed. Two category of irrelevancy:
  - **Visual irrelevancy** or
  - **Application-specific irrelevancy**
- **Visual**: when image density exceeds the limits of display or viewing
- **Application**: an entire region of the image is unneeded, like in medical images or military
  - Can be either heavily compressed or excluded.

Redundancy and Relevancy of images

II. Redundancy reduction:

- Images has statistical redundancy
- Why ? → Because pixels are not random but highly correlated either locally or globally
- We can benefit from entropy-coding in the compression process



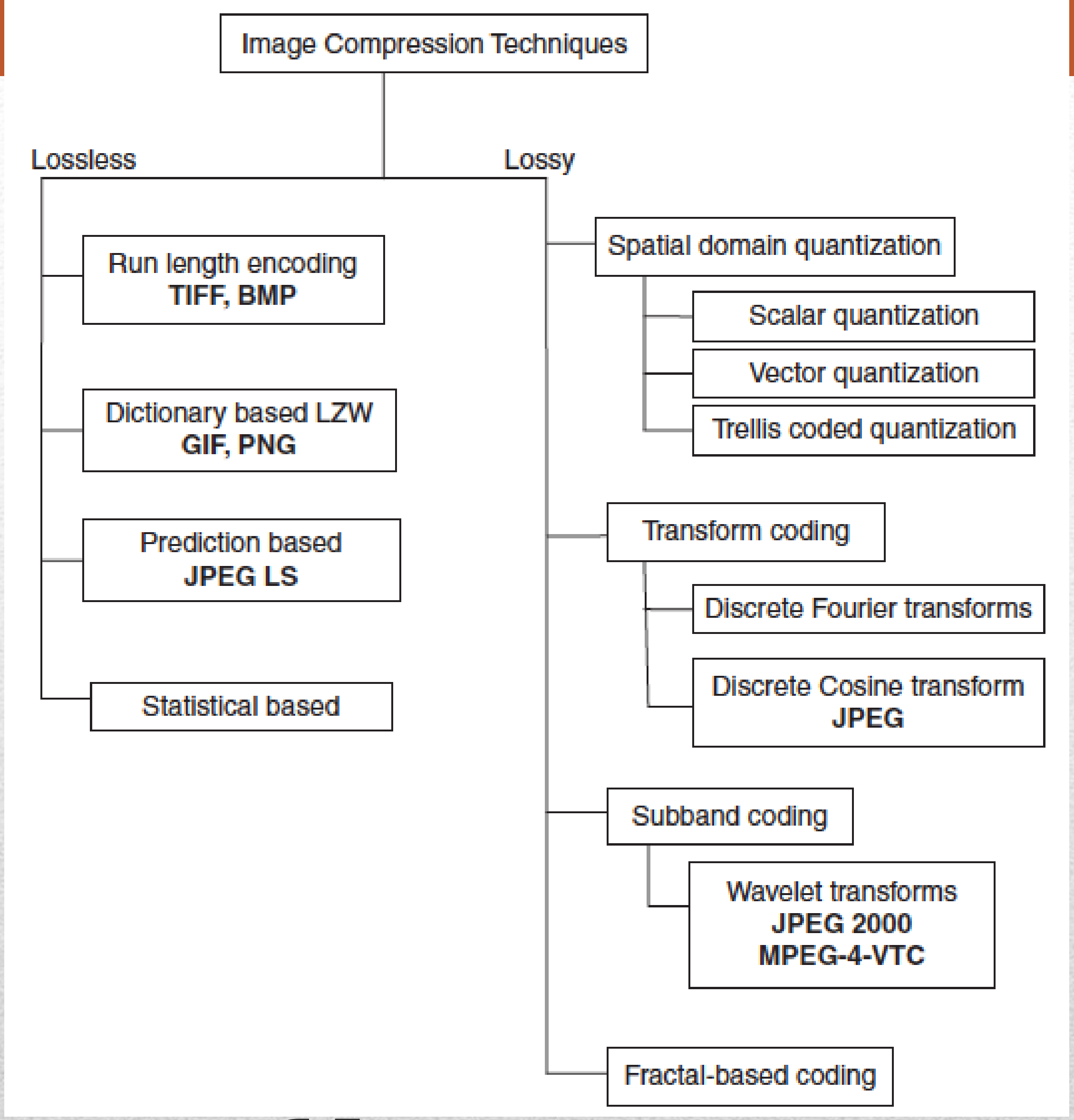
Redundancy and Relevancy of images

- **Spatial**: pixel intensities in a region

- **Spectral**: in frequency domain, some frequencies dominate over others

- **Temporal**: correlation from frame to frame, will be covered in Video compression.

Types of redundancy



Classes of image compression techniques

- Mainly used for storage or cinema
  - Run length
  - Dictionary-based
  - Prediction-based

Lossless image coding

- Run length encoding RLE: one of the earliest lossless schemes used
- BMP, TIFF
- Replaces run of same pixel by the value of the frequency (number of occurrence)

Run length image coding

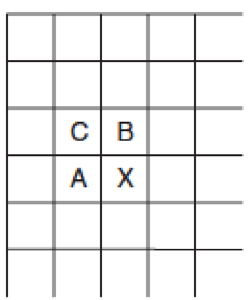
- GIF, PNG use LZW compression
- It’s not applied on initial pixels right away but on a set of indexed colors (for example 8 bits, hence 256 color palette)
- The LZW code words are based on the indexes not the pixels.

Dictionary based image coding

Dr. Zein Ibrahim

Prediction based coding

- Predict a pixel value based on 1-D or 2-D around the pixel
- Calculate the difference D between the prediction and actual
- Results in “ERROR image” which has lower entropy



Prediction index	Prediction
0	No prediction
1	A
2	B
3	C
4	$A + B - C$
5	$A + ((B - C)/2)$
6	$B + ((A - C)/2)$
7	$(A + B)/2$

Past and present observable random variables are prior scanned pixels within that image

When scanning from upper left corner to lower right corner:

B	C	D
A	X	

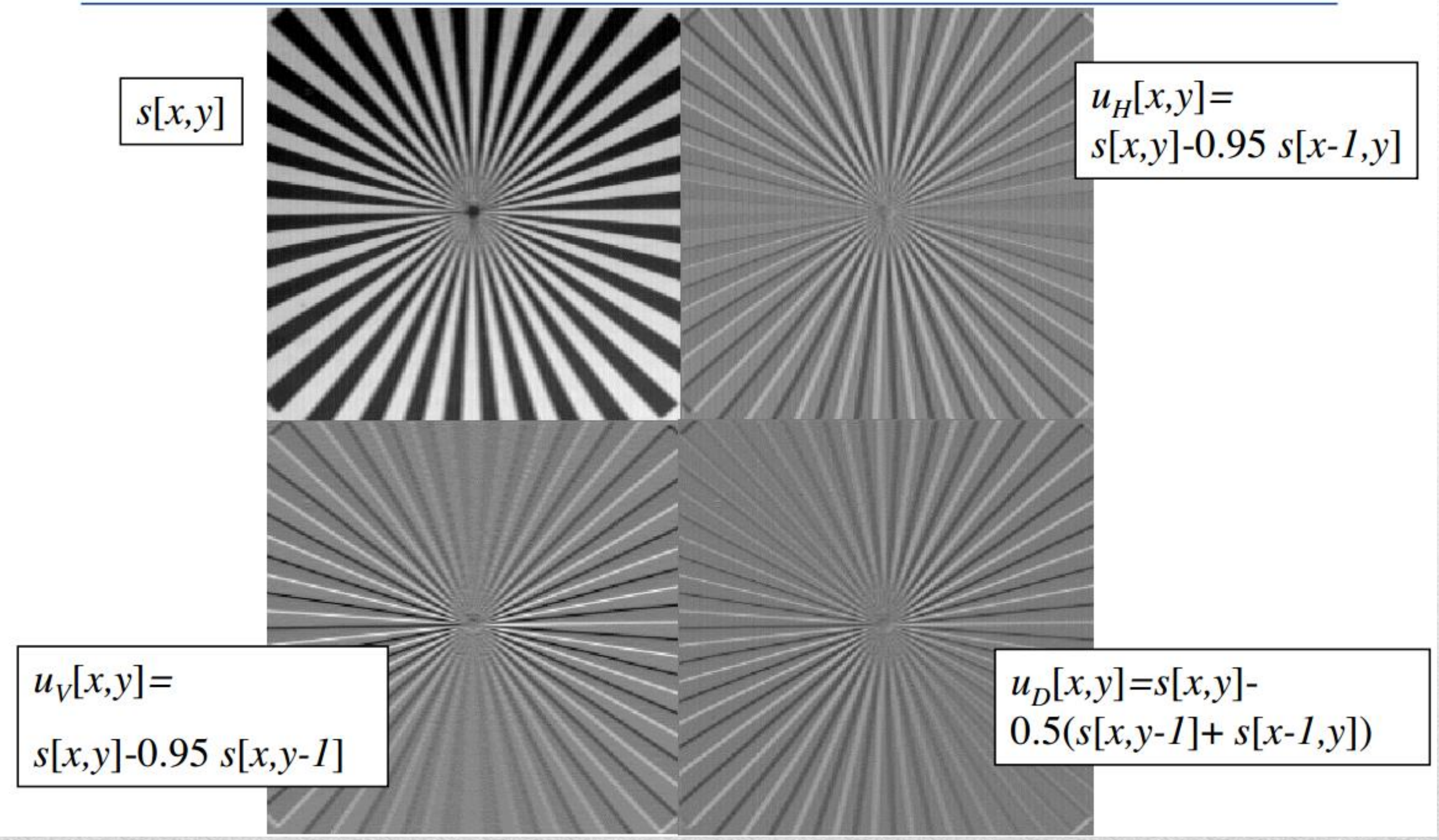
1-D Horizontal prediction: A only

1-D Vertical prediction: C only

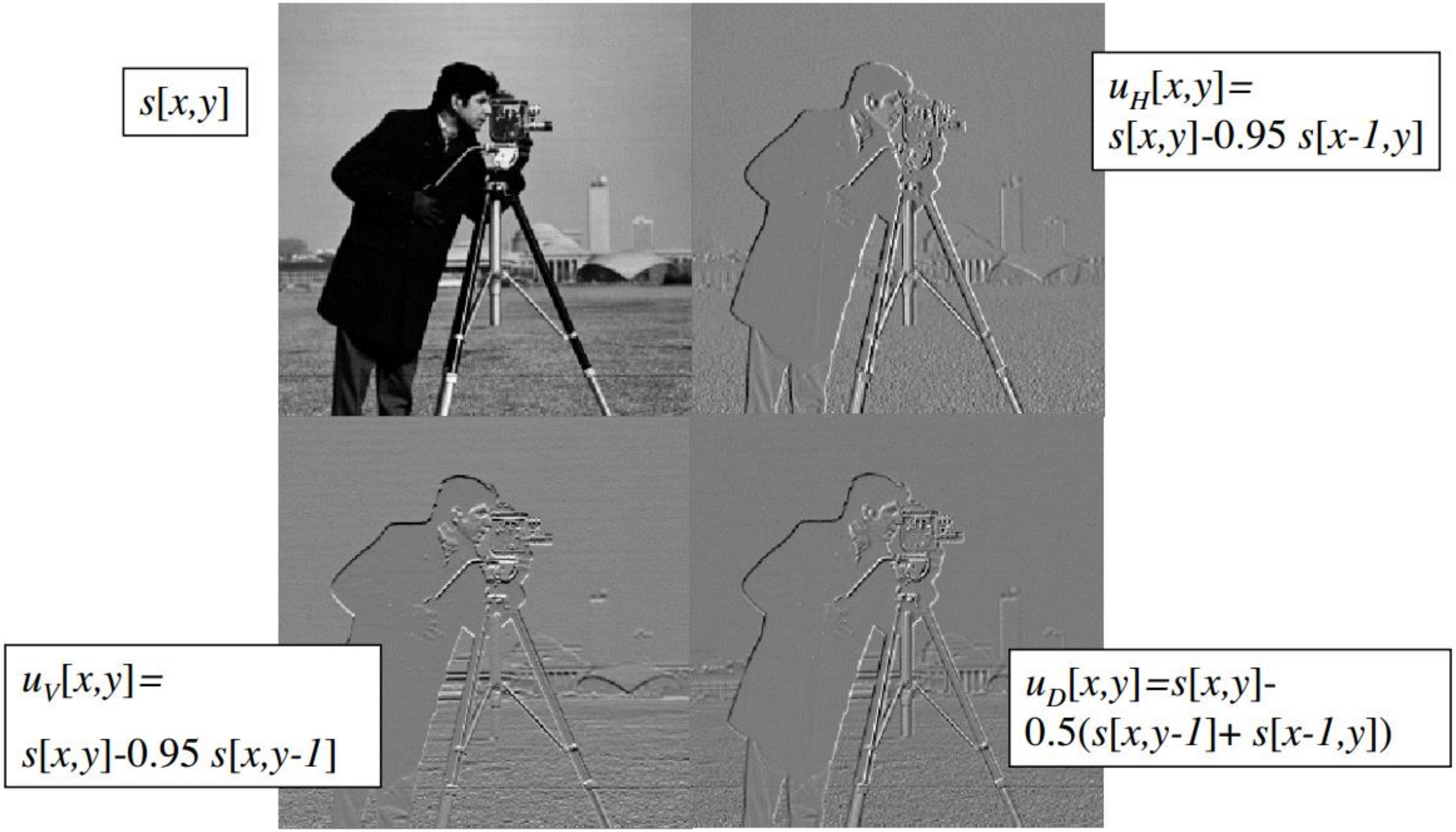
Improvements for 2-D approaches (requires line store)

$$\hat{s}(x,y) = \underbrace{\sum_{p=-P_1}^{P_2}\sum_{q=0}^{Q}}_{(p,q)\neq(0,0)} a(p,q)\cdot s(x-p,y-q)$$

Prediction based coding

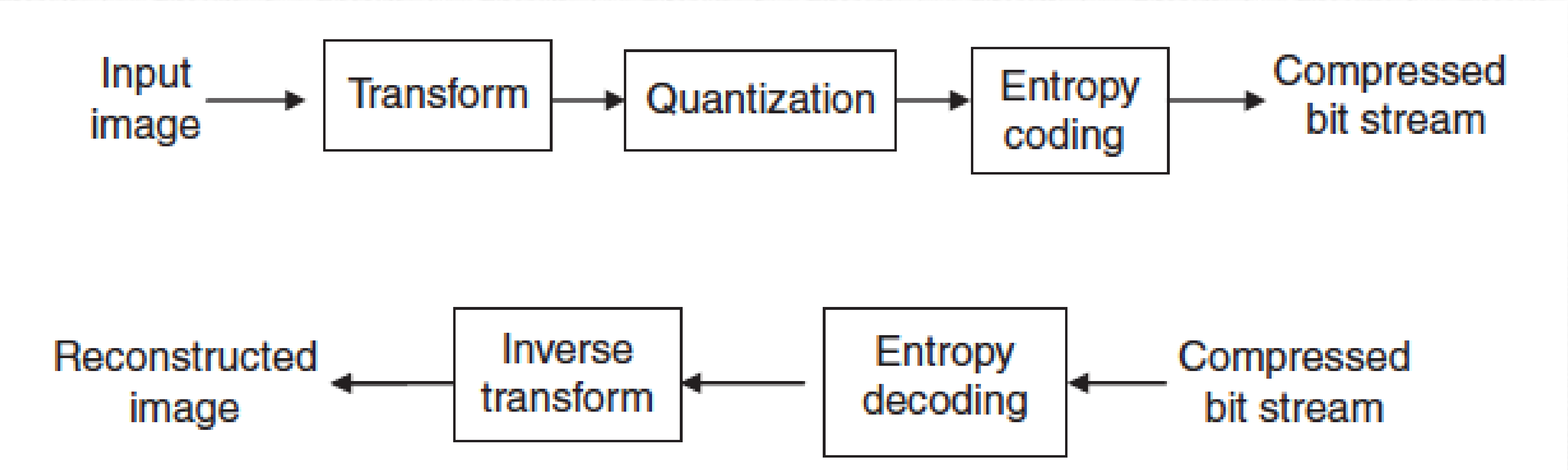


Prediction based coding



Prediction based coding

- Transform can be DFT, DCT (jpeg uses DCT)



- DCT has good frequency domain distribution
- Efficient for hardware implementation

Transform image coding

- Discrete Cosine Transform → has great advantages in energy compaction

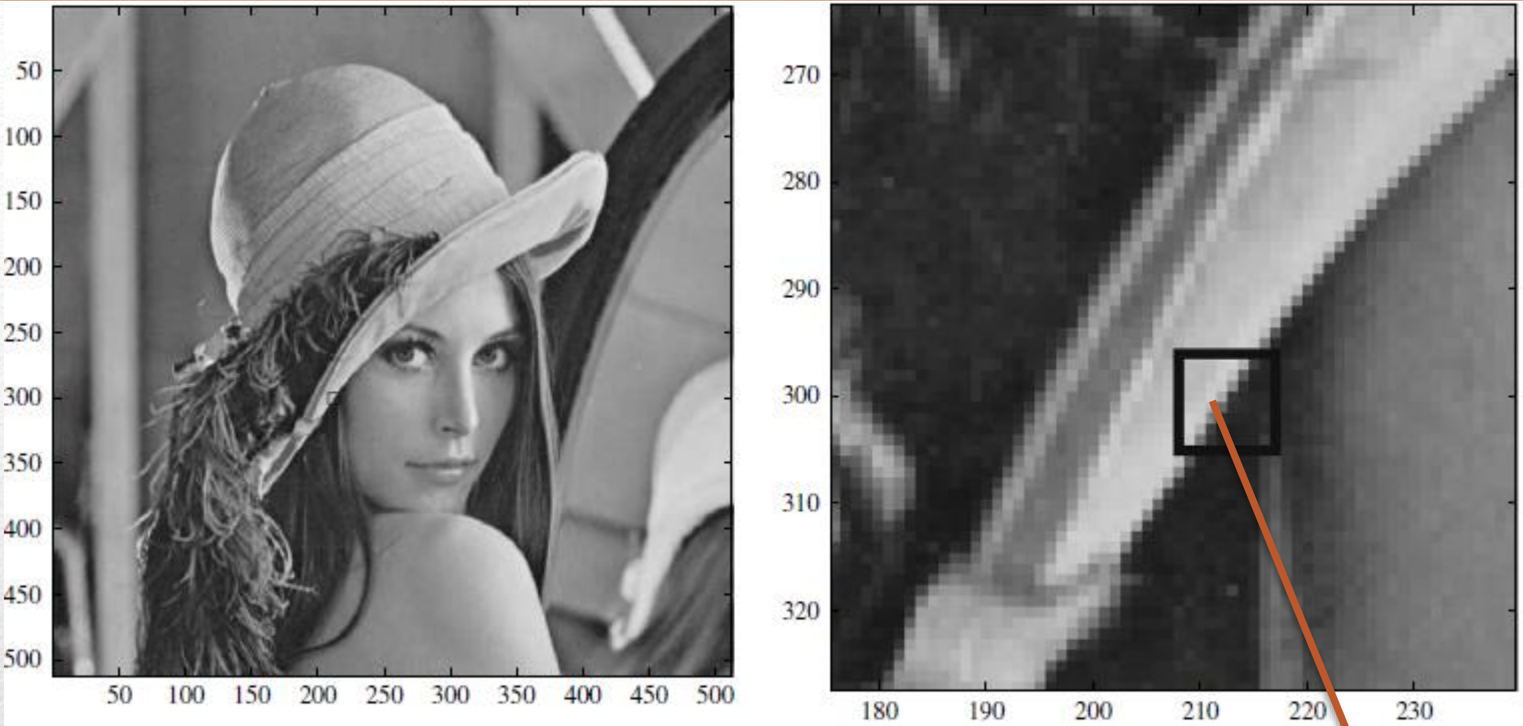
- DCT:

$$\mathbf{DP}_{u,v} = \begin{cases} \dfrac{1}{N}\displaystyle\sum_{x=0}^{N-1}\sum_{y=0}^{N-1}\mathbf{P}_{x,y} & \text{if} \quad u = 0 \quad \text{and} \quad v = 0 \\ \dfrac{2}{N}\displaystyle\sum_{x=0}^{N-1}\sum_{y=0}^{N-1}\mathbf{P}_{x,y} \times \cos\left(\dfrac{(2x+1)u\pi}{2N}\right) \times \cos\left(\dfrac{(2y+1)v\pi}{2N}\right) & \text{otherwise} \end{cases}$$

- IDCT → Inverse DCT

$$\mathbf{P}_{x,y} = \frac{2}{N}\sum_{u=0}^{N-1}\sum_{v=0}^{N-1}\mathbf{DP}_{u,v} \times \cos\left(\frac{(2x+1)u\pi}{2N}\right) \times \cos\left(\frac{(2y+1)v\pi}{2N}\right)$$

DCT Transform



187	188	189	202	209	175	66	41
191	186	193	209	193	98	40	39
188	187	202	202	144	53	35	37
189	195	206	172	58	47	43	45
197	204	194	106	50	48	42	45
208	204	151	50	41	41	41	53
209	179	68	42	35	36	40	47
200	117	53	41	34	38	39	63

DCT

915.6	451.3	25.6	−12.6	16.1	−12.3	7.9	−7.3
216.8	19.8	−228.2	−25.7	23.0	−0.1	6.4	2.0
−2.0	−77.4	−23.8	102.9	45.2	−23.7	−4.4	−5.1
30.1	2.4	19.5	28.6	−51.1	−32.5	12.3	4.5
5.1	−22.1	−2.2	−1.9	−17.4	20.8	23.2	−14.5
−0.4	−0.8	7.5	6.2	−9.6	5.7	−9.5	−19.9
5.3	−5.3	−2.4	−2.4	−3.5	−2.1	10.0	11.0
0.9	0.7	−7.7	9.3	2.7	−5.4	−6.7	2.5

$$g = \begin{array}{c} \xrightarrow{\;x\;} \\ \begin{bmatrix} -76 & -73 & -67 & -62 & -58 & -67 & -64 & -55 \\ -65 & -69 & -73 & -38 & -19 & -43 & -59 & -56 \\ -66 & -69 & -60 & -15 & 16 & -24 & -62 & -55 \\ -65 & -70 & -57 & -6 & 26 & -22 & -58 & -59 \\ -61 & -67 & -60 & -24 & -2 & -40 & -60 & -58 \\ -49 & -63 & -68 & -58 & -51 & -60 & -70 & -53 \\ -43 & -57 & -64 & -69 & -73 & -67 & -63 & -45 \\ -41 & -49 & -59 & -60 & -63 & -52 & -50 & -34 \end{bmatrix} \end{array} \Big\downarrow y.$$

$$\Big\downarrow \text{DCT}$$

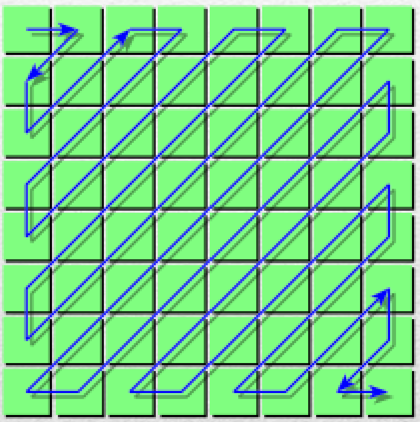
$$G = \begin{array}{c} \xrightarrow{\;u\;} \\ \begin{bmatrix} -415.38 & -30.19 & -61.20 & 27.24 & 56.12 & -20.10 & -2.39 & 0.46 \\ 4.47 & -21.86 & -60.76 & 10.25 & 13.15 & -7.09 & -8.54 & 4.88 \\ -46.83 & 7.37 & 77.13 & -24.56 & -28.91 & 9.93 & 5.42 & -5.65 \\ -48.53 & 12.07 & 34.10 & -14.76 & -10.24 & 6.30 & 1.83 & 1.95 \\ 12.12 & -6.55 & -13.20 & -3.95 & -1.87 & 1.75 & -2.79 & 3.14 \\ -7.73 & 2.91 & 2.38 & -5.94 & -2.38 & 0.94 & 4.30 & 1.85 \\ -1.03 & 0.18 & 0.42 & -2.42 & -0.88 & -3.02 & 4.12 & -0.66 \\ -0.17 & 0.14 & -1.07 & -4.19 & -1.17 & -0.10 & 0.50 & 1.68 \end{bmatrix} \end{array} \Big\downarrow v.$$

$$\xrightarrow{\text{Quantized by Q}}$$

$$B = \begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -3 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

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

DCT



−26
−3 0
−3 −2 −6
2 −4 1 −3
1 1 5 1 2
−1 1 −1 2 0 0
0 0 0 -1 -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

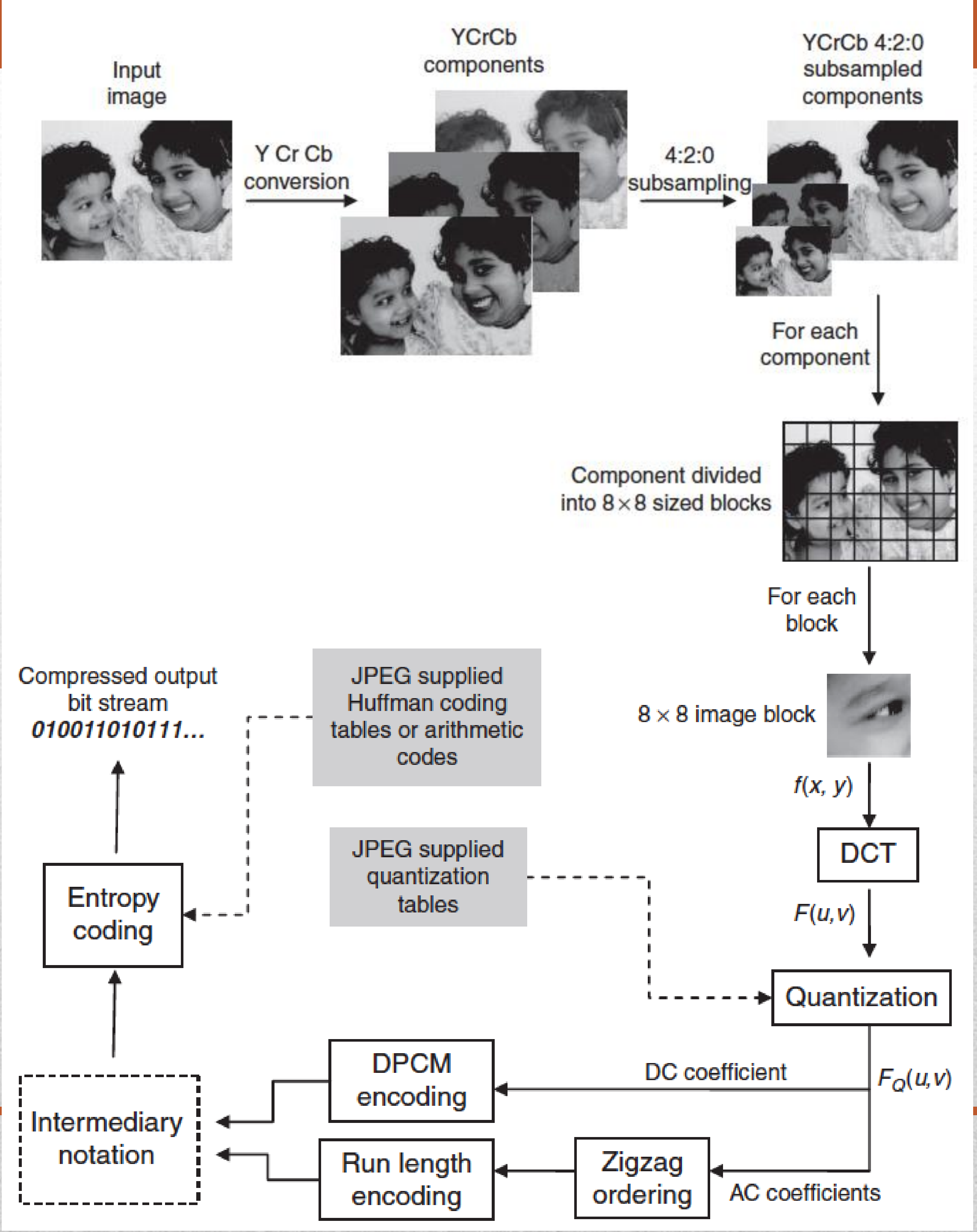
Zig-Zag Order

2025 / 2026

JPEG

Joint Photographic Experts Group

JPEG



Dr. Zein Ibrahim

- Convert into YCrCrb to decouple chrominance from luminance
- Subsample 4:2:0 to reduce the size of the image while keeping it visually pleasing
- Each channel is processed independently
- Each channel is divided into 8x8 blocks
- Less than that would result in so many blocks and more would decrease the correlation between pixels within the block
- If an image row/col size is not a multiple of 8, it’s padded by zeros

STEPS for JPEG compression

- Each block undergoes a DCT $f(x,y) \rightarrow F(u,v)$
- $F(0,0)$ is called DC component and usually the highest values because energy in natural photographs is concentrated among the lowest frequencies
- Remaining $F(u,v)$ are called AC components
- Then they are quantized using a table supplied by JPEG (each entry is quantization interval, also similar to natural DCT block in energy) using formula shown next.

STEPS for JPEG compression

DCT

178	187	183	175	178	177	150	183
191	174	171	182	176	171	170	188
199	153	128	177	171	167	173	183
195	178	158	167	167	165	166	177
190	186	158	155	159	164	158	178
194	184	137	148	157	158	150	173
200	194	148	151	161	155	148	167
200	195	172	159	159	152	156	154

Pixel values $f(x, y)$

1359	46	61	26	38	–21	–5	–18
31	–35	–25	–11	13	10	12	–3
13	20	–17	–14	–11	–7	6	5
–5	5	2	–8	–11	–26	8	–4
10	15	–10	–16	–21	–7	8	7
–6	1	0	7	5	–7	–1	–3
–13	–8	1	10	8	4	–3	–4
–5	–5	–2	5	5	0	0	–3

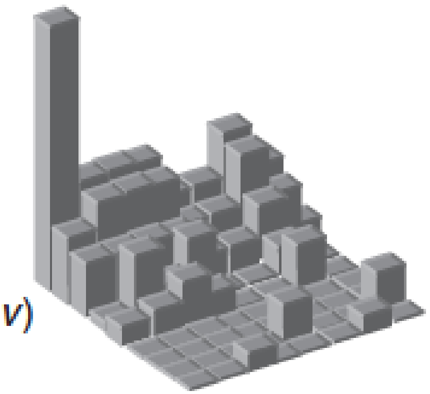
DCT values $F(u,v)$

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 table

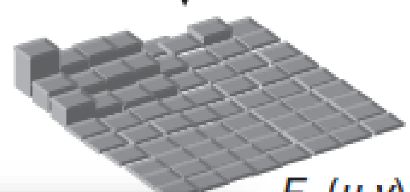
$f(x, y)$

DCT



$F(u,v)$

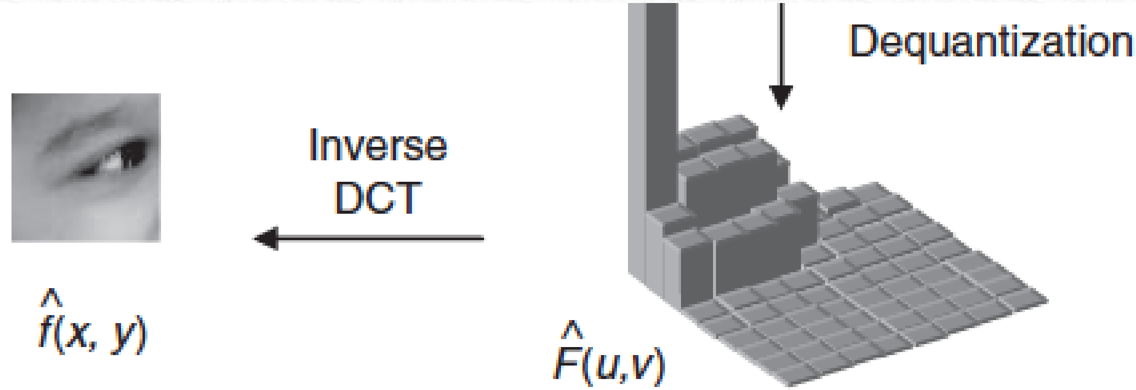
Quantization



$F_Q(u,v)$

85	4	6	2	2	–1	0	0
3	–3	–2	–1	1	0	0	0
1	2	–1	–1	0	0	0	0
0	0	0	0	0	0	0	0
1	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

$$F_Q(u,v) = \left[\frac{F(u,v)}{Q(u,v)}\right], \text{ where } Q(u,v) \text{ is the value in the quantization table}$$



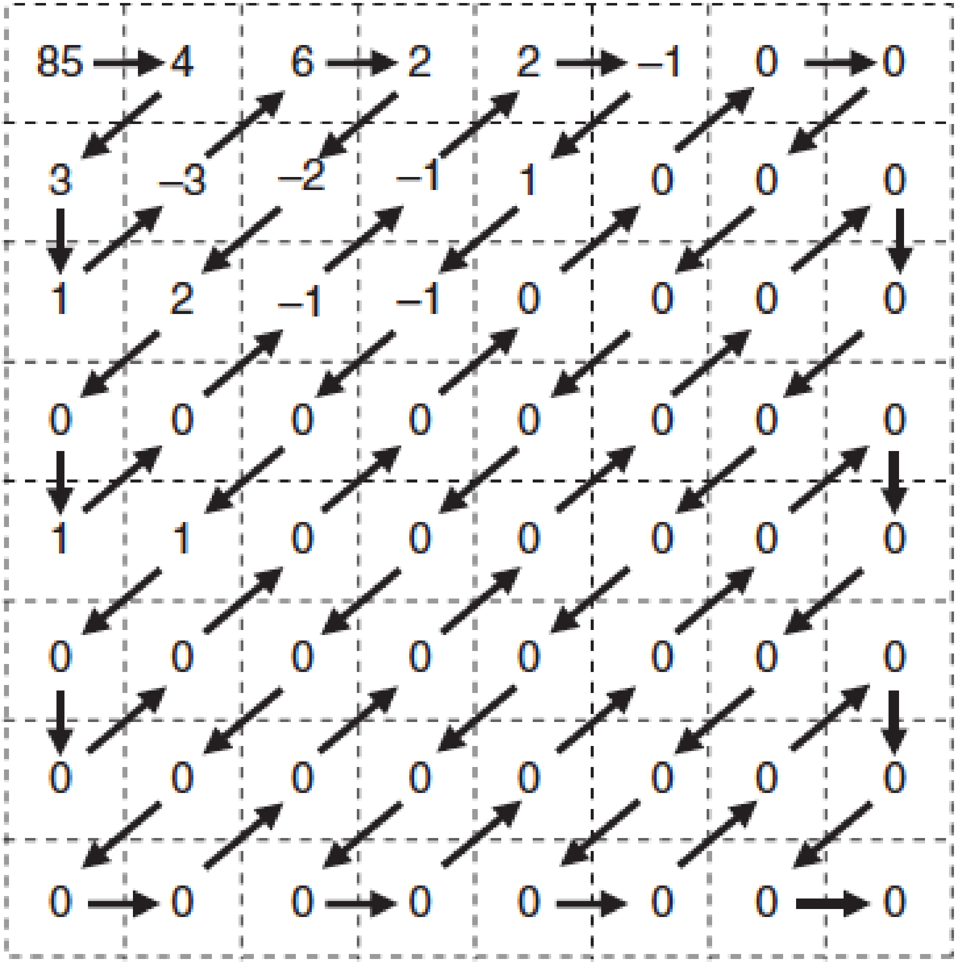
192	185	178	152	193	162	155	190
181	172	162	154	187	164	159	192
183	168	150	158	181	167	162	190
198	177	150	155	177	169	161	182
202	180	148	148	171	167	159	175
193	176	145	141	162	162	156	170
195	184	155	145	159	156	150	164
209	200	170	154	160	153	144	156

1360	44	60	32	48	–40	0	0
36	–36	–28	–19	26	0	0	0
14	26	–16	–24	0	0	0	0
0	0	0	0	0	0	0	0
18	22	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

DCT

- After obtaining the quantized values, it's time to encode them
- Intermediary step:
  - DC is encoded using DPCM (error calculation)
  - AC are calculated using run length (remember there are a lot of ZEROS)
  - Using ZIZGAG to scan images for the AC encoding produced longer runs of zeros compared with raster scanning. Look at it!

STEPS continued



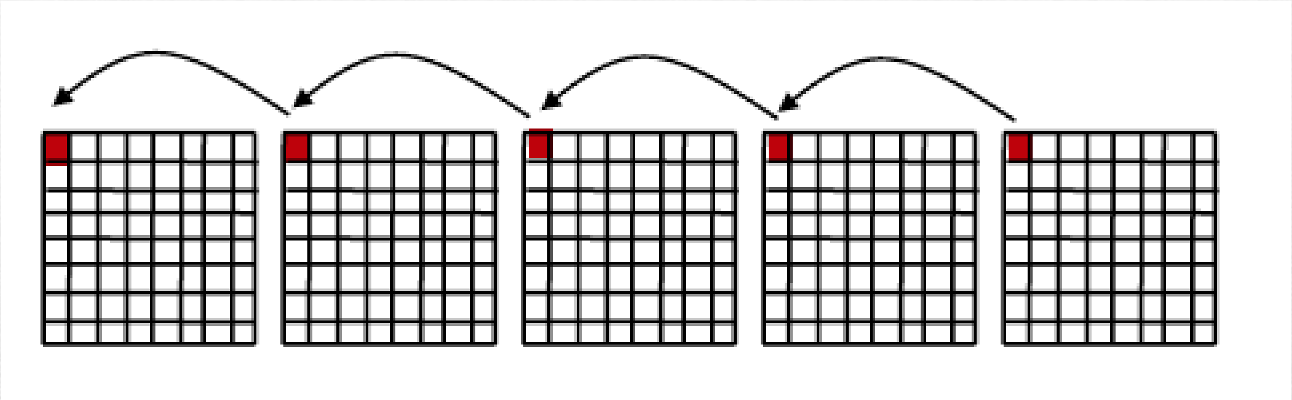
DC coefficient = 85

AC coefficient stream

4 3 1 -3 6 2 -2 2 0 1 0 -1 -1 2
-1 1 -1 0 1 0 0 0 0 0 0 0 0 0 ...

Zigzag ordering

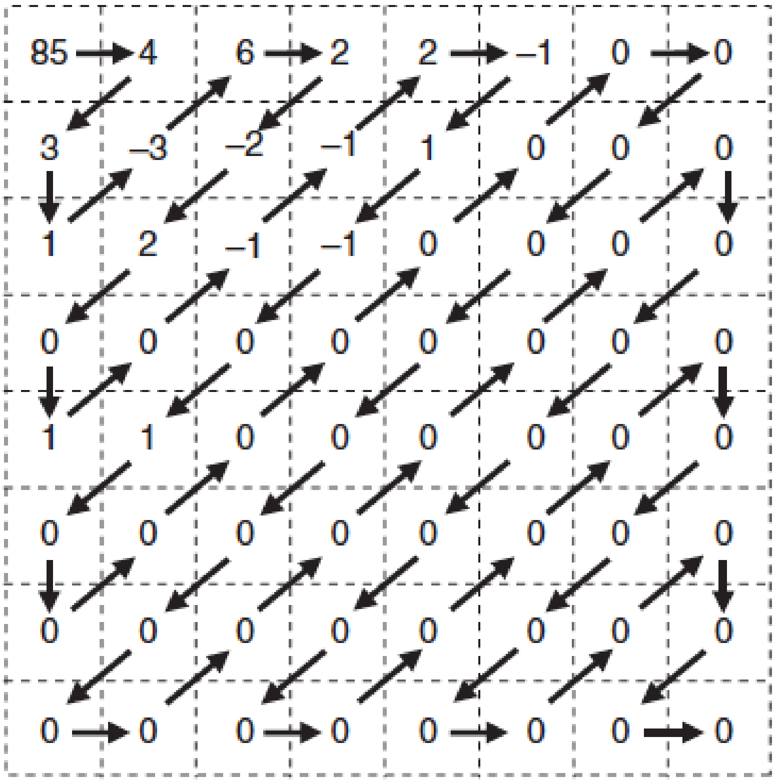
- For DC, the difference between 2 DC blocks values is encoded in 2 notations:
- <size><amplitude>
- Size is the size in bits needed to encode the amplitude
- Example: DC=85 and DC=82 → diff = 3
- <2><3>: 2 bits to represent value 3.



Steps continued

Steps continued

- For AC, only non-zero coefficients are used.
- <runlength , size> <Amplitude of non-zero>
- Runlength: number of zero AC that precedes it in the ZigZag
- Size: nb. of bits to represent the amplitude
- 1’s complement for negative numbers



DC coefficient = 85

AC coefficient stream

4 3 1 -3 6 2 -2 2 0 1 0 -1 -1 2
-1 1 -1 0 1 0 0 0 0 0 0 0 0 0 ...

Intermediary stream

<2><3> <0,3><4> <0,2><3> <0,1><1> <0,2><-3> <0,3><6>
<0,2><2> <0,2><-2> <0,2><2> <1,1><1> <1,1><-1> <0,1><-1>
<0,2><2> <0,1><-1> <0,1><1> <0,1><-1> <1,1><1> EOB

AC & DC coefficients

- For both DC and AC
- First part < > → use huffman
- Second part < > → use non-prefixed representation

Last steps

2025 / 2026

Dr. Zein Ibrahim

coding

Intermediary symbol	Binary representation of first symbol (prefixed Huffman Codes)	Binary representation of second symbol (non-prefixed variable integer codes)
<2> <3>	011	11
<0,3> <4>	100	100
<0,2> <3>	01	11
<0,1> <1>	00	1
<0,2> <−3>	01	00
<0,3> <6>	100	110
<0,2> <2>	01	10
<0,2> <−2>	01	01
<0,2> <2>	01	10
<1,1> <1>	11	1
<1,1> <−1>	11	0
<0,1> <−1>	00	0
<0,2> <2>	01	10
<0,1> <−1>	00	0
<0,1> <1>	00	1
<0,1> <−1>	00	0
<1,1> <1>	11	1
EOB	1010	

Binary Stream:

011111001000111001010010011001100101011011111000001100000010001111010

Dr. Zein Ibrahim

2025 / 2026

- Poor low bit compression → at low bit rates, the perceived distortion becomes unacceptable.
- Does not allow random access to bit stream
- Large image handling → JPEG does not allow compression of images larger than 64K x 64K.
- Transmission in noisy environments (especially in wireless) which was not taken into account in the standard.
- Not suited for computer generated images and documents (it was developed for natural tone images)
- This all led to the development of Jpeg2000

JPEG drawbacks

Wavelet Based Coding

2025 / 2026
Dr. Zein Ibrahim

- The Wavelet Transform achieves compression through a process called **Multi-Resolution Subband Coding**, which efficiently separates the image information based on frequency and scale.
- **Decomposition Process (Filtering):** The 2D Discrete Wavelet Transform (DWT) decomposes an image into subbands using a pair of filters:
  - **Low-Pass Filter (LPF):** Extracts the **Approximation** components (the scaled-down, low-frequency version).
  - **High-Pass Filter (HPF):** Extracts the **Detail** components (edges, textures, high-frequency noise).

DWT

DWT

- JPEG 2000 used Discrete Wavelet Transform
- It's better than DCT since it distributes energy among all coefficients
- DCT works on 8x8 blocks while DWT on the whole image

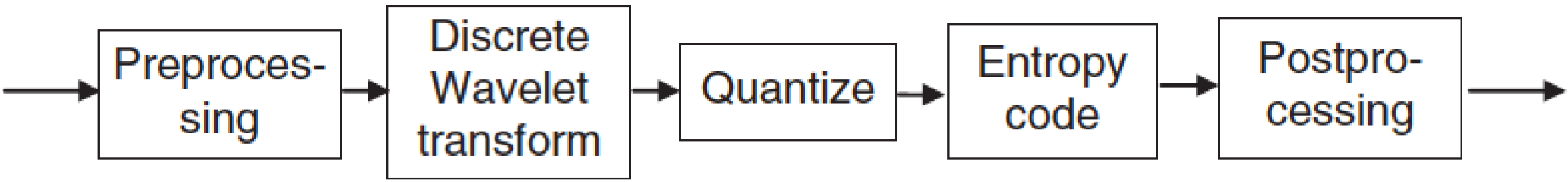
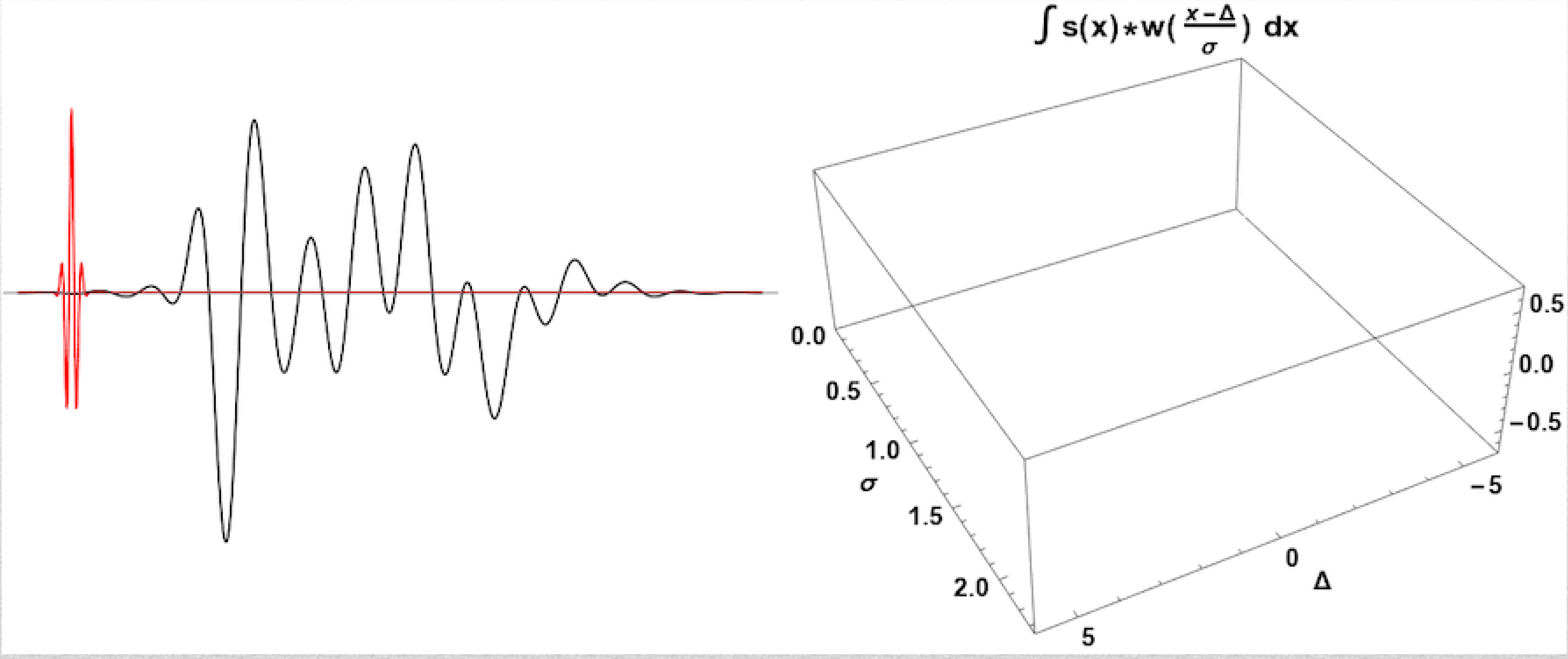


Figure 7-12 The JPEG 2000 pipeline

➢ Wavelet → Continuous WT (CWT) and Discrete WT (DWT).
➢ Several classes of wavelet based on the properties of the underlying wavelets (filter banks)
- Orthogonal Wavelets (Perfect Reconstruction)
  - **Haar Wavelet:** The simplest and earliest wavelet. It involves basic averaging (LPF) and differencing (HPF).
  - **Daubechies Wavelets (DbN)**
- Biorthogonal Wavelets (Compression Standard)
  - **Lifting Scheme**
  - **Biorthogonal 9/7 Wavelet**: This is the most famous biorthogonal wavelet. It is used as the lossy compression kernel in the JPEG 2000 standard because it offers an excellent balance between smoothness and energy compaction.
  - **Biorthogonal 5/3 Wavelet**: Used as the lossless compression kernel in the JPEG 2000 standard.
- Multi-Wavelet Transforms
  - Use multiple scaling and wavelet functions (not just one pair) to perform the decomposition.

Types of DWT



DWT

Original (16 values)

```
[4, 6, 10, 12, 14, 16, 18, 20, 5, 7, 9, 11, 13, 15, 17, 19]
```

Encoding (Haar Transform)

Level 1 (pairs → averages | details)

Averages:

```
[5, 11, 15, 19, 6, 10, 14, 18]
```

Details:

```
[-1, -1, -1, -1, -1, -1, -1, -1]
```

Full sequence after L1:

```
[5, 11, 15, 19, 6, 10, 14, 18 | -1, -1, -1, -1, -1, -1, -1, -1]
```

Haar Transform Encoding

```
[5, 11, 15, 19, 6, 10, 14, 18 | -1, -1, -1, -1, -1, -1, -1, -1]
```

Level 2 (on the first 8 averages)

Averages:

```
[8, 17, 8, 16]
```

Details:

```
[-3, -3, -2, -2]
```

Full sequence after L2:

```
[8, 17, 8, 16 | -3, -3, -2, -2 | -1, -1, -1, -1, -1, -1, -1, -1]
```

Level 3 (on the first 4 averages)

Averages:

```
[12.5, 12]
```

Details:

```
[-4.5, -4]
```

Full sequence after L3:

```
[12.5, 12 | -4.5, -4 | -3, -3, -2, -2 | -1, -1, -1, -1, -1, -1, -1, -1]
```

Haar Transform Encoding

```
[12.5, 12 | -4.5, -4 | -3, -3, -2, -2 | -1, -1, -1, -1, -1, -1, -1, -1]
```

Level 4 (on the first 2 averages)

Average:

```
[12.25]
```

Detail:

```
[0.5]
```

Final Haar encoding (all levels combined):

```
[12.25, 0.5, -4.5, -4, -3, -3, -2, -2, -1, -1, -1, -1, -1, -1, -1, -1]
```

```
[12.25, 0.5, -4.5, -4, -3, -3, -2, -2, -1, -1, -1, -1, -1, -1, -1, -1
```

Decoding (Inverse Haar)

Level 4 → Level 3

```
[12.5, 12]
```

Sequence:

```
[12.5, 12, -4.5, -4, -3, -3, -2, -2, -1, -1, -1, -1, -1, -1, -1, -1]
```

Level 3 → Level 2

```
[8, 17, 8, 16]
```

Sequence:

```
[8, 17, 8, 16, -3, -3, -2, -2, -1, -1, -1, -1, -1, -1, -1, -1]
```

Level 2 → Level 1

```
[5, 11, 15, 19, 6, 10, 14, 18]
```

Sequence:

```
[5, 11, 15, 19, 6, 10, 14, 18, -1, -1, -1, -1, -1, -1, -1, -1]
```

Haar Transform Decoding

$$\mathbf{X} = [10, 20, 30, 40, 50, 60, 70, 80, 75, 65, 55, 45, 35, 25, 15, 5]$$

$$[\underbrace{15.0, 35.0, 55.0, 75.0, 70.0, 50.0, 30.0, 10.0}_{A\text{ (Approximation)}}, \underbrace{-5.0, -5.0, -5.0, -5.0, 5.0, 5.0, 5.0, 5.0}_{D\text{ (Detail)}}]$$

$$[\underbrace{25.0, 65.0, 60.0, 20.0}_{\mathbf{A}_2}, \underbrace{-10.0, -10.0, 10.0, 10.0}_{\mathbf{D}_2}, \underbrace{-5.0, -5.0, -5.0, -5.0, 5.0, 5.0, 5.0, 5.0}_{\mathbf{D}_1}]$$

$$\underset{\mathbf{A}_3}{[45.0, 40.0]} \quad \underset{\mathbf{D}_3}{[-20.0, 20.0]}$$

$$\underbrace{42.5]}_{\mathbf{A}_4}\underbrace{[2.5]}_{\mathbf{D}_4}\underbrace{[-20.0, 20.0]}_{\mathbf{D}_3}\underbrace{[-10.0, -10.0, 10.0, 10.0]}_{\mathbf{D}_2}\underbrace{[-5.0, -5.0, -5.0, -5.0, 5.0, 5.0, 5.0, 5.0}_{\mathbf{D}_1}$$

Example

- We apply row by row then column by column
- To decode, we decode column by column then row by row.

Let the input image patch $\mathbf{X}$ be a 4×4 matrix of grayscale pixel values:

$$\mathbf{X} = \begin{pmatrix} 10 & 20 & 30 & 40 \\ 10 & 20 & 30 & 40 \\ 50 & 60 & 70 & 80 \\ 50 & 60 & 70 & 80 \end{pmatrix}$$

1. Step 1: Filter Across Rows (Horizontal Pass)

We apply the 1D Haar LPF (Average) and HPF (Difference) to **each row** of $\mathbf{X}$ independently. The resulting 4×4 matrix $\mathbf{R}$ now contains coefficients mixed by row.

Row Pair	LPF (Avg)	HPF (Diff)
$(10, 20)$	15	-5
$(30, 40)$	35	-5

$$\mathbf{R} = \begin{pmatrix} \underbrace{15}_{LPF} & \underbrace{35}_{LPF} & \underbrace{-5}_{HPF} & \underbrace{-5}_{HPF} \\ 15 & 35 & -5 & -5 \\ 55 & 75 & -5 & -5 \\ 55 & 75 & -5 & -5 \end{pmatrix}$$

2d Haar Transform Encoding

2. Step 2: Filter Across Columns (Vertical Pass)

Now, we apply the 1D Haar LPF and HPF to the **columns** of the intermediate matrix $\mathbf{R}$. This separates the coefficients into the final four subbands.

Column Pair	LPF (Avg)	HPF (Diff)
$(15, 15)$	15	0
$(55, 55)$	55	0

$$\mathbf{W} = \begin{pmatrix} \underbrace{15}_{LL} & \underbrace{35}_{LL} & \underbrace{-5}_{LH} & \underbrace{-5}_{LH} \\ \underbrace{55}_{LL} & \underbrace{75}_{LL} & \underbrace{-5}_{LH} & \underbrace{-5}_{LH} \\ \underbrace{0}_{HL} & \underbrace{0}_{HL} & \underbrace{0}_{HH} & \underbrace{0}_{HH} \\ \underbrace{0}_{HL} & \underbrace{0}_{HL} & \underbrace{0}_{HH} & \underbrace{0}_{HH} \end{pmatrix}$$

2d Haar Transform Encoding

The $\mathbf{W}$ matrix is partitioned into four 2×2 quadrants:

Subband	Location	Coefficients	Description
LL	Top-Left	$\begin{pmatrix} 15 & 35 \\ 55 & 75 \end{pmatrix}$	**Approximation** (Low-Low): The scaled-down version of the image. Contains almost all the energy.
HL	Bottom-Left	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$	**Horizontal Details** (High-Low): The image has no vertical structure changes.
LH	Top-Right	$\begin{pmatrix} -5 & -5 \\ -5 & -5 \end{pmatrix}$	**Vertical Details** (Low-High): Captures the horizontal edge/detail present in the image.
HH	Bottom-Right	$\begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix}$	**Diagonal Details** (High-High): No complex diagonal texture is present.

2d Haar Transform Encoding

$$\mathbf{W} = \begin{pmatrix} 15 & 35 & -5 & -5 \\ 55 & 75 & -5 & -5 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

After applying this to all columns, the resulting intermediate matrix $\mathbf{R'}$ is:

$$\mathbf{R'} = \begin{pmatrix} \underbrace{15}_{LL/LH} & \underbrace{35}_{LL/LH} & \underbrace{-5}_{LL/LH} & \underbrace{-5}_{LL/LH} \\ 15 & 35 & -5 & -5 \\ 55 & 75 & -5 & -5 \\ 55 & 75 & -5 & -5 \end{pmatrix}$$

After applying this to all rows, the final reconstructed image $\mathbf{X'}$ is:

$$\mathbf{X'} = \begin{pmatrix} 10 & 20 & 30 & 40 \\ 10 & 20 & 30 & 40 \\ 50 & 60 & 70 & 80 \\ 50 & 60 & 70 & 80 \end{pmatrix}$$

2d Haar Transform Decoding

Example 2

```
[100, 200,  50, 150, 200,  50, 100, 150]
[200,  50, 150, 100,  50, 200, 150, 100]
[ 50, 150, 100, 200, 150,  50, 200,  50]
[150, 100,  50, 150, 100, 200,  50, 200]
[200,  50, 200,  50, 100, 150, 100,  50]
[ 50, 200,  50, 200, 150,  50, 200, 150]
[150,  50, 150, 100,  50, 200,  50, 100]
[ 50, 150, 100,  50, 200,  50, 150, 200]
```

rows

```
[150, 100, 125, 125,   -50,   -50,    75,   -25]
[125, 125, 125, 125,    75,   -50,    25,    25]
[100, 150, 100, 125,   -50,   -50,    75,    75]
[125, 100, 150, 125,    25,   -50,   -75,   -75]
[125, 125, 125,  75,    75,    75,    25,    25]
[125, 125, 100, 175,   -75,   -75,    25,    25]
[100, 125, 125,  75,    50,    25,   -75,    25]
[100,  75, 125, 175,   -50,    25,   -25,   -25]
```

columns

```
|------ LL1 ------|------ LH1 ------|
| 138   113   125   125 |  13   -50    50     0 |
| 113   125   125   125 | -13   -50     0     0 |
| 125   125   113   125 |   0     0    25    25 |
| 100   100   125   125 |   0    25   -50     0 |
|------ HL1 ------|------ HH1 ------|
|  13   -13     0     0 | -63     0    25   -25 |
| -13    25   -25     0 | -38     0    75    75 |
|   0     0    13   -50 |  75    75     0     0 |
|   0    25     0   -50 |  50     0   -25    25 |
```

Stage 2

```
|-- LL2 --|-- LH2 --|
| 122     122 |  3.25     3    |
| 112     122 | -3.25    -4.75|
|-- HL2 --|-- HH2 --|
|  3.25   -3   |  6.25     1.5 |
|  0      -7.75|  0        6.25|
```

$$\mathbf{X} = [10, 20, 30, 40, 50, 60, 70, 80]$$

First, split $\mathbf{X}$ into **Even** (X_e) and **Odd** (X_o) samples:

$$X_e = [10, 30, 50, 70]$$

$$X_o = [20, 40, 60, 80]$$

- Step 1: Details calculation (D)

$$D_i = X_{o,i} - \left\lfloor \frac{X_{e,i} + X_{e,i+1}}{2} \right\rfloor$$

- $D_0 = 20 - \lfloor (10 + 30)/2 \rfloor = 20 - 20 = \mathbf{0}$
- $D_1 = 40 - \lfloor (30 + 50)/2 \rfloor = 40 - 40 = \mathbf{0}$
- $D_2 = 60 - \lfloor (50 + 70)/2 \rfloor = 60 - 60 = \mathbf{0}$
- $D_3 = 80 - \lfloor (70 + 10)/2 \rfloor = 80 - 40 = \mathbf{40}$ (Using periodic boundary conditions, $X_{e,4} = X_{e,0} = 10$)

$$\mathbf{D} = [0, 0, 0, 40]$$

1D orthogonal 5/3 wavelet (lossless)

- Step 2: Update calculation (Approximation calculation)

$$X_e = [10, 30, 50, 70]$$

$$\mathbf{D} = [0, 0, 0, 40]$$

$$X_o = [20, 40, 60, 80]$$

$$A_i = X_{e,i} + \left\lfloor \frac{D_{i-1} + D_i + 2}{4} \right\rfloor$$

- $A_0 = 10 + \lfloor (40 + 0 + 2)/4 \rfloor = 10 + 10 = \mathbf{20}$ (Using periodic boundary conditions, $D_{-1} = D_3 = 40$)
- $A_1 = 30 + \lfloor (0 + 0 + 2)/4 \rfloor = 30 + 0 = \mathbf{30}$
- $A_2 = 50 + \lfloor (0 + 0 + 2)/4 \rfloor = 50 + 0 = \mathbf{50}$
- $A_3 = 70 + \lfloor (0 + 40 + 2)/4 \rfloor = 70 + 10 = \mathbf{80}$

$$\mathbf{A} = [20, 30, 50, 80]$$

5/3 Encoded Signal: $\mathbf{W} = [\mathbf{A}, \mathbf{D}] = [20, 30, 50, 80, 0, 0, 0, 40]$

1D orthogonal 5/3 wavelet (losseless)

Decoding process

$$\mathbf{W} = [20, 30, 50, 80, 0, 0, 0, 40]$$

5. Decoding (Inverse Transform)

Decoding reverses the process, applying the inverse of the update, then the inverse of the prediction.

Step 3: Inverse Update (Restore $\mathbf{X}_e$)

$$\mathbf{X}_{e,i} = \mathbf{A}_i - \left\lfloor \frac{\mathbf{D}_{i-1} + \mathbf{D}_i + 2}{4} \right\rfloor$$

This is simply A_i minus the update term calculated in Step 2.

- $\mathbf{X}_{e,0} = 20 - 10 = \mathbf{10}$
- $\mathbf{X}_{e,1} = 30 - 0 = \mathbf{30}$
- $\mathbf{X}_{e,2} = 50 - 0 = \mathbf{50}$
- $\mathbf{X}_{e,3} = 80 - 10 = \mathbf{70}$

$$\mathbf{X}_e = [10, 30, 50, 70] \text{ (Restored)}$$

1D orthogonal 5/3 wavelet (losseless)

Decoding process

$$\mathbf{W} = [20, 30, 50, 80, 0, 0, 0, 40]$$

Step 4: Inverse Prediction (Restore $\mathbf{X}_o$)

$$\mathbf{X}_{o,i} = \mathbf{D}_i + \left\lfloor \frac{\mathbf{X}_{e,i} + \mathbf{X}_{e,i+1}}{2} \right\rfloor$$

This is $\mathbf{D}_i$ plus the prediction term calculated in Step 1.

- $\mathbf{X}_{o,0} = 0 + 20 = \mathbf{20}$
- $\mathbf{X}_{o,1} = 0 + 40 = \mathbf{40}$
- $\mathbf{X}_{o,2} = 0 + 60 = \mathbf{60}$
- $\mathbf{X}_{o,3} = 40 + 40 = \mathbf{80}$

$$\mathbf{X}_o = [20, 40, 60, 80] \text{ (Restored)}$$

1D orthogonal 5/3 wavelet (losseless)

1. **Split:** X_e, X_o.
2. **Predict ($\mathbf{D}$):** $D_i = X_{o,i} + \alpha(X_{e,i} + X_{e,i+1})$
3. **Update ($\mathbf{A}$):** $A_i = X_{e,i} + \beta(D_{i-1} + D_i)$
4. **Predict ($\mathbf{D}$):** $D_i = D_i + \gamma(A_i + A_{i+1})$ (Refining $\mathbf{D}$ using the updated $\mathbf{A}$)
5. **Update ($\mathbf{A}$):** $A_i = A_i + \delta(D_{i-1} + D_i)$ (Refining $\mathbf{A}$ using the refined $\mathbf{D}$)
6. **Scale:** $A_i = K \cdot A_i$ and $D_i = (1/K) \cdot D_i$.

1D orthogonal 9/7 wavelet (lossy) - Encoding

$$\mathbf{X} = [10, 20, 30, 40, 50, 60, 70, 80]$$

First, split $\mathbf{X}$ into **Even** (X_e) and **Odd** (X_o) samples:

$$X_e = [10, 30, 50, 70]$$

$$X_o = [20, 40, 60, 80]$$

- $\alpha \approx -1.586134342$
- $\beta \approx -0.05298011854$
- $\gamma \approx 0.8829110762$
- $\delta \approx 0.4435068522$
- $K \approx 1.230174105$ (Normalization)

$1/K \approx 0.81057813$

1. Prediction Step 1 (α - Calculate $\mathbf{D}'$)

The odd samples are predicted using the even samples and the multiplier α. This results in the intermediate Detail coefficients ($\mathbf{D}'$).

$$\mathbf{D}'_i = X_{o,i} + \alpha(X_{e,i} + X_{e,i+1})$$

(Using periodic boundary conditions: $X_{e,4} = X_{e,0} = 10$)

i	$X_{o,i}$	$X_{e,i} + X_{e,i+1}$	$\alpha(X_{e,i} + X_{e,i+1})$	$\mathbf{D}'_i$
0	20	$10 + 30 = 40$	-63.445	-43.445
1	40	$30 + 50 = 80$	-126.891	-86.891
2	60	$50 + 70 = 120$	-190.336	-130.336
3	80	$70 + 10 = 80$	-126.891	-46.891

$$\mathbf{D}' = [-43.445, -86.891, -130.336, -46.891]$$

1D orthogonal 9/7 wavelet (lossy) - Encoding

$$\mathbf{X} = [10, 20, 30, 40, 50, 60, 70, 80]$$

First, split $\mathbf{X}$ into **Even** (X_e) and **Odd** (X_o) samples:

$$X_e = [10, 30, 50, 70]$$

$$X_o = [20, 40, 60, 80]$$

- $\alpha \approx -1.586134342$
- $\beta \approx -0.05298011854$
- $\gamma \approx 0.8829110762$
- $\delta \approx 0.4435068522$
- $K \approx 1.230174105$ (Normalization)
- $1/K \approx 0.81057813$

$$\mathbf{D}' = [-43.445, -86.891, -130.336, -46.891]$$

2. Update Step 1 (β - Calculate $\mathbf{A}'$)

The even samples are updated using the intermediate detail coefficients ($\mathbf{D}'$) and the multiplier β. This results in the intermediate Approximation coefficients ($\mathbf{A}'$).

$$\mathbf{A}'_i = X_{e,i} + \beta(\mathbf{D}'_{i-1} + \mathbf{D}'_i)$$

(Using periodic boundary conditions: $\mathbf{D}'_{-1} = \mathbf{D}'_3 = -46.891$)

i	$X_{e,i}$	$\mathbf{D}'_{i-1} + \mathbf{D}'_i$	$\beta(\mathbf{D}'_{i-1} + \mathbf{D}'_i)$	$\mathbf{A}'_i$
0	10	$-46.891 + (-43.445) = -90.336$	4.787	14.787
1	30	$-43.445 + (-86.891) = -130.336$	6.894	36.894
2	50	$-86.891 + (-130.336) = -217.227$	11.499	61.499
3	70	$-130.336 + (-46.891) = -177.227$	9.380	79.380

$$\mathbf{A}' = [14.787, 36.894, 61.499, 79.380]$$

$$\mathbf{D}' = [-43.445, -86.891, -130.336, -46.891]$$

3. Prediction Step 2 (γ - Refine $\mathbf{D}'$ to $\mathbf{D}''$)

The detail coefficients are refined using the intermediate approximation coefficients ($\mathbf{A}'$) and the multiplier γ.

$$\mathbf{D}''_i = \mathbf{D}'_i + \gamma(\mathbf{A}'_i + \mathbf{A}'_{i+1})$$

(Using periodic boundary conditions: $\mathbf{A}'_4 = \mathbf{A}'_0 = 14.787$)

i	$\mathbf{D}'_i$	$\mathbf{A}'_i + \mathbf{A}'_{i+1}$	$\gamma(\mathbf{A}'_i + \mathbf{A}'_{i+1})$	$\mathbf{D}''_i$
0	-43.445	$14.787 + 36.894 = 51.681$	45.641	2.196
1	-86.891	$36.894 + 61.499 = 98.393$	86.891	-0.000
2	-130.336	$61.499 + 79.380 = 140.879$	124.398	-5.938
3	-46.891	$79.380 + 14.787 = 94.167$	83.152	36.261

$$\mathbf{D}'' = [2.196, -0.000, -5.938, 36.261]$$

- $\alpha \approx -1.586134342$
- $\beta \approx -0.05298011854$
- $\gamma \approx 0.8829110762$
- $\delta \approx 0.4435068522$
- $K \approx 1.230174105$ (Normalization)
- $1/K \approx 0.81057813$

1D orthogonal 9/7 wavelet (lossy) - Encoding

$$\mathbf{A}' = [14.787, 36.894, 61.499, 79.380]$$

4. Update Step 2 (δ - Refine $\mathbf{A}'$ to $\mathbf{A}$)

The approximation coefficients are refined using the new detail coefficients ($\mathbf{D}''$) and the multiplier δ. This yields the final Approximation coefficients ($\mathbf{A}$).

$$\mathbf{A}_i = \mathbf{A}'_i + \delta(\mathbf{D}''_{i-1} + \mathbf{D}''_i)$$

(Using periodic boundary conditions: $\mathbf{D}''_{-1} = \mathbf{D}''_3 = 36.261$)

i	$\mathbf{A}'_i$	$\mathbf{D}''_{i-1} + \mathbf{D}''_i$	$\delta(\mathbf{D}''_{i-1} + \mathbf{D}''_i)$	$\mathbf{A}_i$
0	14.787	$36.261 + 2.196 = 38.457$	17.060	31.847
1	36.894	$2.196 + (-0.000) = 2.196$	0.975	37.869
2	61.499	$-0.000 + (-5.938) = -5.938$	-2.634	58.865
3	79.380	$-5.938 + 36.261 = 30.323$	13.456	92.836

$$\mathbf{A}_{\text{unscaled}} = [31.847, 37.869, 58.865, 92.836]$$

- $\alpha \approx -1.586134342$
- $\beta \approx -0.05298011854$
- $\gamma \approx 0.8829110762$
- $\delta \approx 0.4435068522$
- $K \approx 1.230174105$ (Normalization)
- $1/K \approx 0.81057813$

1D orthogonal 9/7 wavelet (lossy) - Encoding

$$\mathbf{A}_{\text{unscaled}} = [31.847, 37.869, 58.865, 92.836]$$

$$\mathbf{D}'' = [2.196, -0.000, -5.938, 36.261]$$

- $\alpha \approx -1.586134342$
- $\beta \approx -0.05298011854$
- $\gamma \approx 0.8829110762$
- $\delta \approx 0.4435068522$
- $K \approx 1.230174105$ (Normalization)
- $1/K \approx 0.81057813$

5. Scaling/Normalization (Final $\mathbf{A}$ and $\mathbf{D}$)

The final step scales the coefficients to ensure energy conservation.

$$\mathbf{A}_i = \mathbf{A}_i \cdot K \quad ; \quad \mathbf{D}_i = \mathbf{D}_i'' \cdot (1/K)$$

Component	Final $\mathbf{A}$ ($\times 1.23017$)	Final $\mathbf{D}$ ($\times 0.81058$)
$\mathbf{A}_0/\mathbf{D}_0$	$31.847 \times 1.23017 = \mathbf{39.178}$	$2.196 \times 0.81058 = \mathbf{1.778}$
$\mathbf{A}_1/\mathbf{D}_1$	$37.869 \times 1.23017 = \mathbf{46.583}$	$-0.000 \times 0.81058 = \mathbf{-0.000}$
$\mathbf{A}_2/\mathbf{D}_2$	$58.865 \times 1.23017 = \mathbf{72.417}$	$-5.938 \times 0.81058 = \mathbf{-4.813}$
$\mathbf{A}_3/\mathbf{D}_3$	$92.836 \times 1.23017 = \mathbf{114.209}$	$36.261 \times 0.81058 = \mathbf{29.393}$

1D orthogonal 9/7 wavelet (lossy) - Encoding

Final Encoded Signal (Wavelet Coefficients $\mathbf{W}$)

$$\mathbf{W} = [\mathbf{A}, \mathbf{D}]$$

$$\mathbf{W} = [\underbrace{39.178, 46.583, 72.417, 114.209}_{\mathbf{A}\ \text{(Approximation)}}, \underbrace{1.778, -0.000, -4.813, 29.393}_{\mathbf{D}\ \text{(Detail)}}]$$

1D orthogonal 9/7 wavelet (lossy) - Encoding

$$\mathbf{A} = [39.178, 46.583, 72.417, 114.209]$$

$$\mathbf{D} = [1.778, -0.000, -4.813, 29.393]$$

The inverse multipliers are: $1/K \approx 0.810578$, $K \approx 1.230174$, and the original multipliers $\alpha, \beta, \gamma, \delta$.

1. Inverse Scaling (Restore Unscaled $\mathbf{A}'$ and $\mathbf{D}''$)

We multiply $\mathbf{A}$ by $1/K$ and $\mathbf{D}$ by K.

$$\mathbf{A}'_{\text{unscaled}} = \mathbf{A} \cdot (1/K) \quad ; \quad \mathbf{D}''_{\text{unscaled}} = \mathbf{D} \cdot K$$

Component	Calculation	$\mathbf{A}'_{\text{unscaled}}$	$\mathbf{D}''_{\text{unscaled}}$
0	39.178×0.81058	31.756	1.778×1.23017
1	46.583×0.81058	37.750	-0.000×1.23017
2	72.417×0.81058	58.706	-4.813×1.23017
3	114.209×0.81058	92.570	29.393×1.23017

1D orthogonal 9/7 wavelet (lossy) - Decoding

2. Inverse Update Step 2 (δ)

We remove the δ update from $\mathbf{A}'_{\text{unscaled}}$ using $\mathbf{D}''$. This restores the $\mathbf{A}'$ coefficients from the β step.

$$\mathbf{A}'_i = \mathbf{A}'_{\text{unscaled},i} - \delta(\mathbf{D}''_{i-1} + \mathbf{D}''_i)$$

(Using $\mathbf{D}''_{-1} = 36.159$)

i	$\mathbf{A}'_{\text{unscaled}}$	$\mathbf{D}''_{i-1} + \mathbf{D}''_i$	$\delta(\mathbf{D}''_{i-1} + \mathbf{D}''_i)$	$\mathbf{A}'_i$
0	31.756	$36.159 + 2.187 = 38.346$	17.009	14.747
1	37.750	$2.187 + 0 = 2.187$	0.970	36.780
2	58.706	$0 + (-5.921) = -5.921$	-2.628	61.334
3	92.570	$-5.921 + 36.159 = 30.238$	13.418	79.152

1D orthogonal 9/7 wavelet (lossy) - Decoding

3. Inverse Prediction Step 2 (γ)

We remove the γ update from $\mathbf{D}''$ using the newly restored $\mathbf{A}'$. This restores the $\mathbf{D}'$ coefficients from the α step.

$$\mathbf{D}'_i = \mathbf{D}''_i - \gamma(\mathbf{A}'_i + \mathbf{A}'_{i+1})$$

(Using $\mathbf{A}'_4 = 14.747$)

i	$\mathbf{D}''_i$	$\mathbf{A}'_i + \mathbf{A}'_{i+1}$	$\gamma(\mathbf{A}'_i + \mathbf{A}'_{i+1})$	$\mathbf{D}'_i$
0	2.187	$14.747 + 36.780 = 51.527$	45.495	-43.308
1	-0.000	$36.780 + 61.334 = 98.114$	86.643	-86.643
2	-5.921	$61.334 + 79.152 = 140.486$	124.053	-130.074
3	36.159	$79.152 + 14.747 = 93.899$	82.919	-46.760

1D orthogonal 9/7 wavelet (lossy) - Decoding

4. Inverse Update Step 1 (β)

We remove the β update from $\mathbf{A}'$ using the newly restored $\mathbf{D}'$. This restores the original Even samples ($\mathbf{X}_e$).

$$\mathbf{X}_{e,i} = \mathbf{A}'_i - \beta(\mathbf{D}'_{i-1} + \mathbf{D}'_i)$$

(Using $\mathbf{D}'_{-1} = -46.760$*)*

i	$\mathbf{A}'_i$	$\mathbf{D}'_{i-1} + \mathbf{D}'_i$	$\beta(\mathbf{D}'_{i-1} + \mathbf{D}'_i)$	$\mathbf{X}_{e,i}$
0	14.747	$-46.760 + (-43.308) = -90.068$	4.766	$\mathbf{9.981}$
1	36.780	$-43.308 + (-86.643) = -129.951$	6.877	$\mathbf{29.903}$
2	61.334	$-86.643 + (-130.074) = -216.717$	11.464	$\mathbf{49.870}$
3	79.152	$-130.074 + (-46.760) = -176.834$	9.359	$\mathbf{69.793}$

$$\mathbf{X}_e = [9.981, 29.903, 49.870, 69.793]$$

1D orthogonal 9/7 wavelet (lossy) - Decoding

5. Inverse Prediction Step 1 (α)

We remove the α prediction from $\mathbf{D}'$ using the restored $\mathbf{X}_e$. This restores the original Odd samples ($\mathbf{X}_o$).

$$\mathbf{X}_{o,i} = \mathbf{D}'_i - \alpha(\mathbf{X}_{e,i} + \mathbf{X}_{e,i+1})$$

(Using $\mathbf{X}_{e,4} = 9.981$)

i	$\mathbf{D}'_i$	$\mathbf{X}_{e,i} + \mathbf{X}_{e,i+1}$	$\alpha(\mathbf{X}_{e,i} + \mathbf{X}_{e,i+1})$	$\mathbf{X}_{o,i}$
0	-43.308	$9.981 + 29.903 = 39.884$	-63.261	$\mathbf{19.953}$
1	-86.643	$29.903 + 49.870 = 79.773$	-126.375	$\mathbf{39.732}$
2	-130.074	$49.870 + 69.793 = 119.663$	-189.489	$\mathbf{59.415}$
3	-46.760	$69.793 + 9.981 = 79.774$	-126.377	$\mathbf{79.617}$

$$\mathbf{X}_o = [19.953, 39.732, 59.415, 79.617]$$

1D orthogonal 9/7 wavelet (lossy) - Decoding

Final Reconstructed Signal

Combining the restored Even and Odd samples gives the final lossy reconstructed signal:

$$\mathbf{X}_{\text{reconstructed}} = [\mathbf{X}_{e,0}, \mathbf{X}_{o,0}, \mathbf{X}_{e,1}, \mathbf{X}_{o,1}, \ldots]$$

$$\mathbf{X}_{\text{reconstructed}} \approx [\mathbf{9.98}, \mathbf{19.95}, \mathbf{29.90}, \mathbf{39.73}, \mathbf{49.87}, \mathbf{59.42}, \mathbf{69.79}, \mathbf{79.62}]$$

Conclusion: The original signal was $[10, 20, 30, 40, 50, 60, 70, 80]$. The reconstructed signal is very close but not identical, confirming the **lossy nature** of the Biorthogonal 9/7 wavelet.

1D orthogonal 9/7 wavelet (lossy) - Decoding

1) Tiling: splitting the image into rectangular but equal sized blocks

- Each will be DWTed independently so encoding and decoding will be faster
- Better memory management due to smaller sizes
- optional

I. Preprocessing

2) **YCrCb conversion**

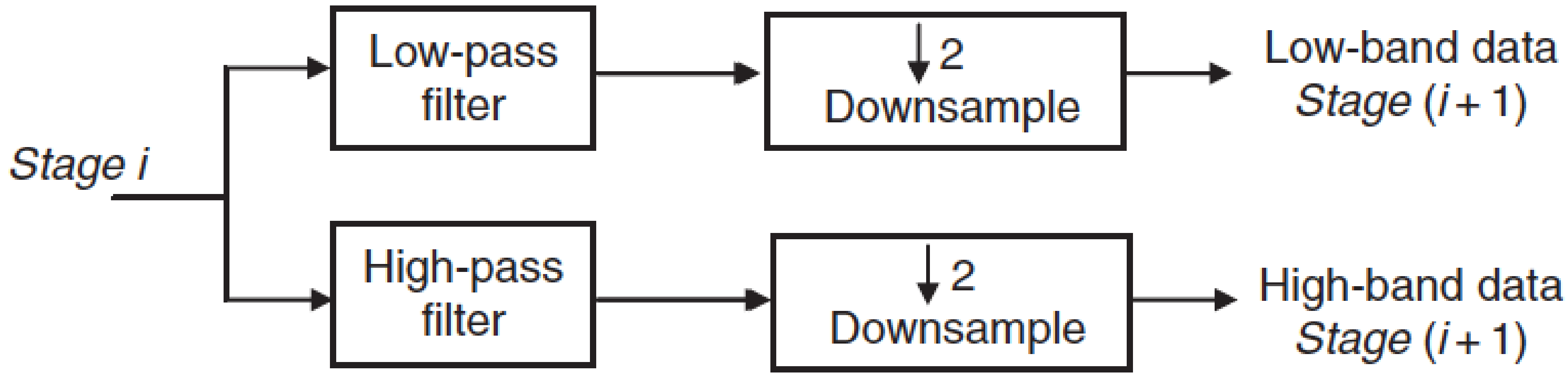
- Convert to YCrCb color space to take advantage of human vision characteristics
- Each channel is then processed independently with different tolerances

3) **Level offsetting**: shifting the DC levels

- Done by subtracting a constant value from all the pixels
- DWT requires the dynamic range of pixel values to be centered around 0
- For n-bit representation, values should be centered from $-2^{(n-1)}$ to $2^{(n-1)}-1$

I. Preprocessing

- A low pass and high pass are applied to the image
- Each result is the same size of the image resulting in double original size
- So we down sample by 2 each one, so we keep same size.



The DWT

The DWT

- Since the image is 2-D, we can use Mallat's algorithm to use 1-D filters
- Filter the rows first (low and high) using 1-D
- Then each result is filtered along the columns using also 1-D
- Result: 4 bands: LL, LH, HL, HH

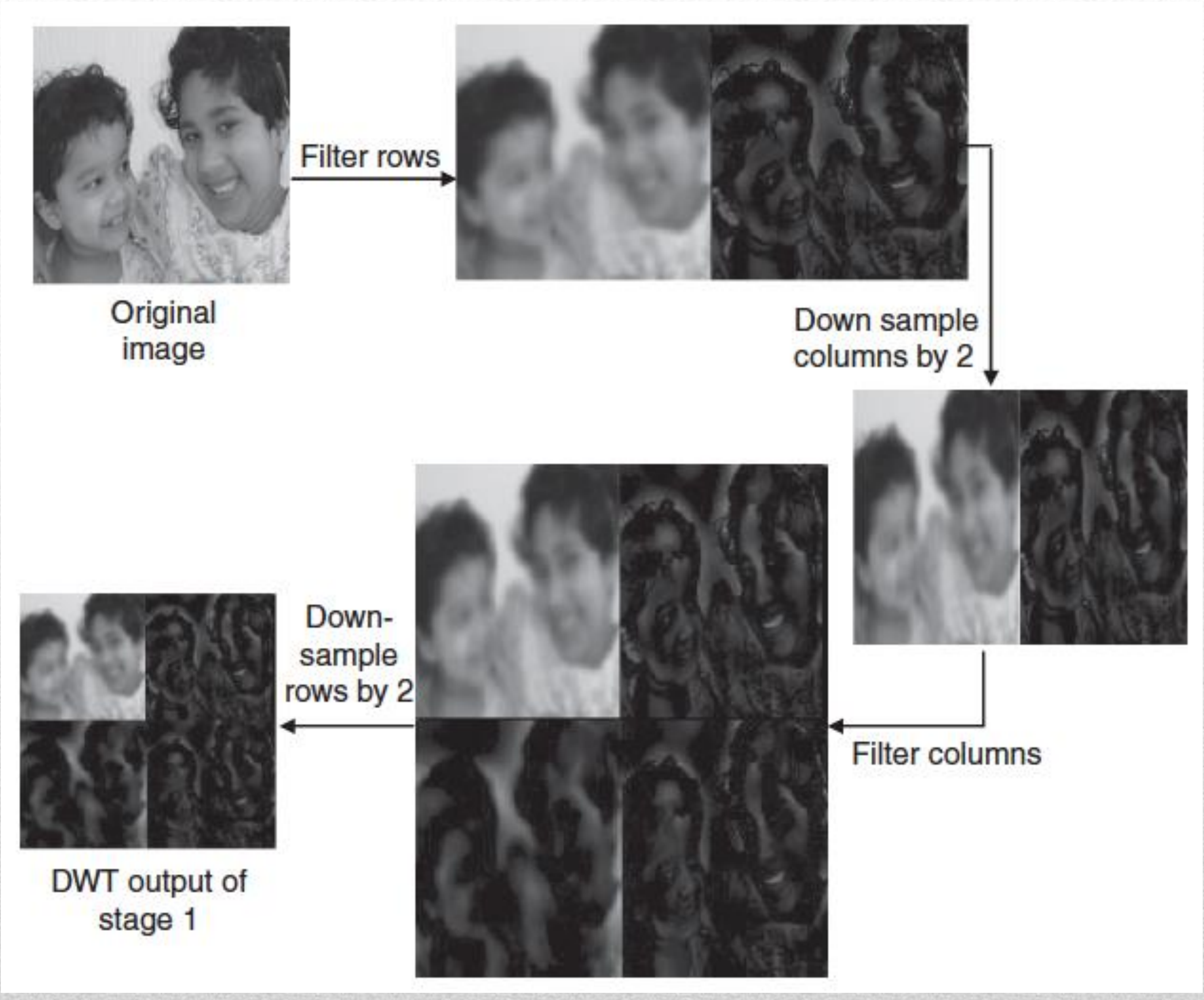
LL—Low subbands of the filtering in both dimensions, rows and columns

HL—High subbands after row filtering and low subbands after column filtering

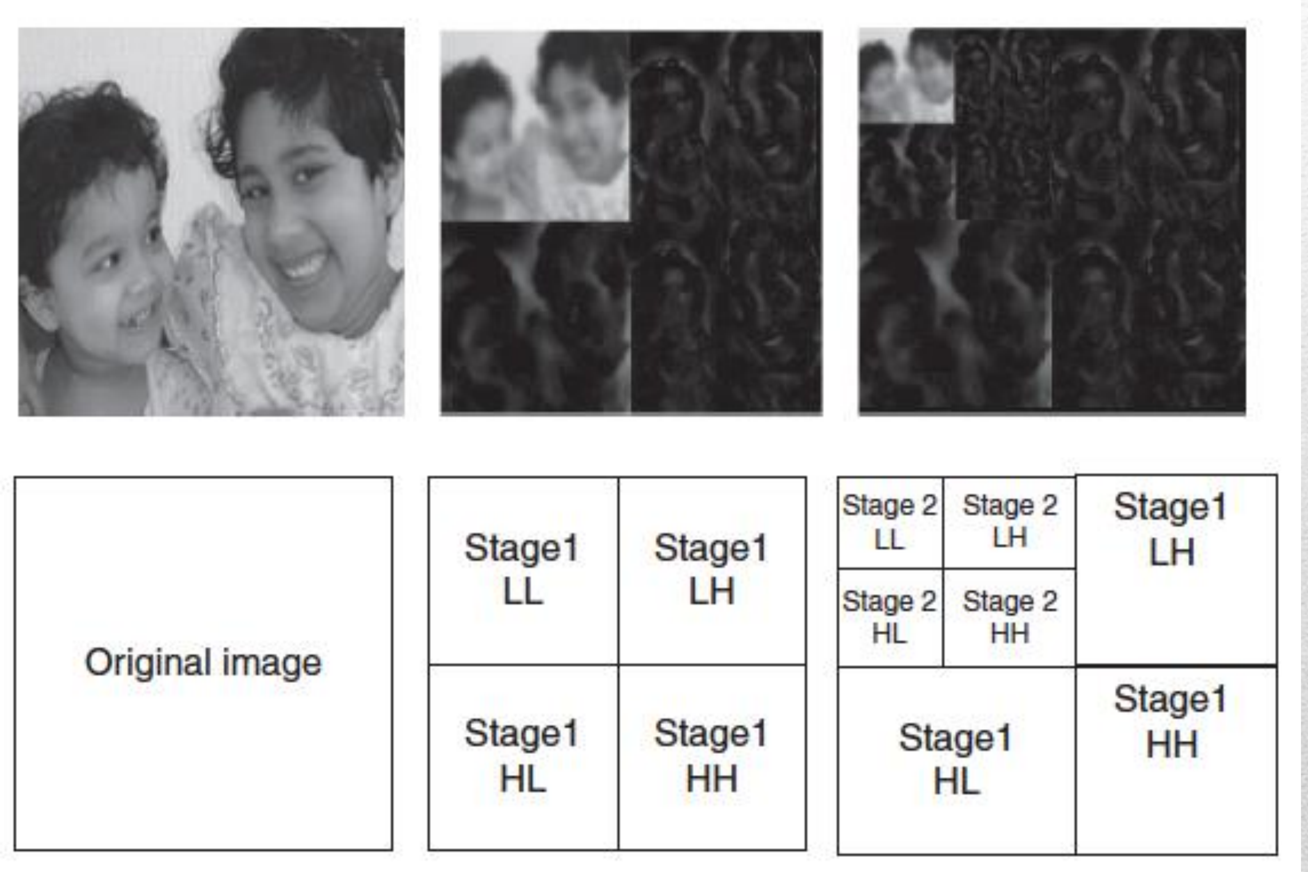
LH—Low subbands after row filtering and high subbands after column filtering

HH—High subbands after both row and column filtering

DWT


Filter rows
Original image
Down sample columns by 2
Filter columns
Down-sample rows by 2
DWT output of stage 1

- JPEG 2000 allows up to 32 stages but usually is 4 to 8
- Each time, we take LL and apply another level of DWT



DWT

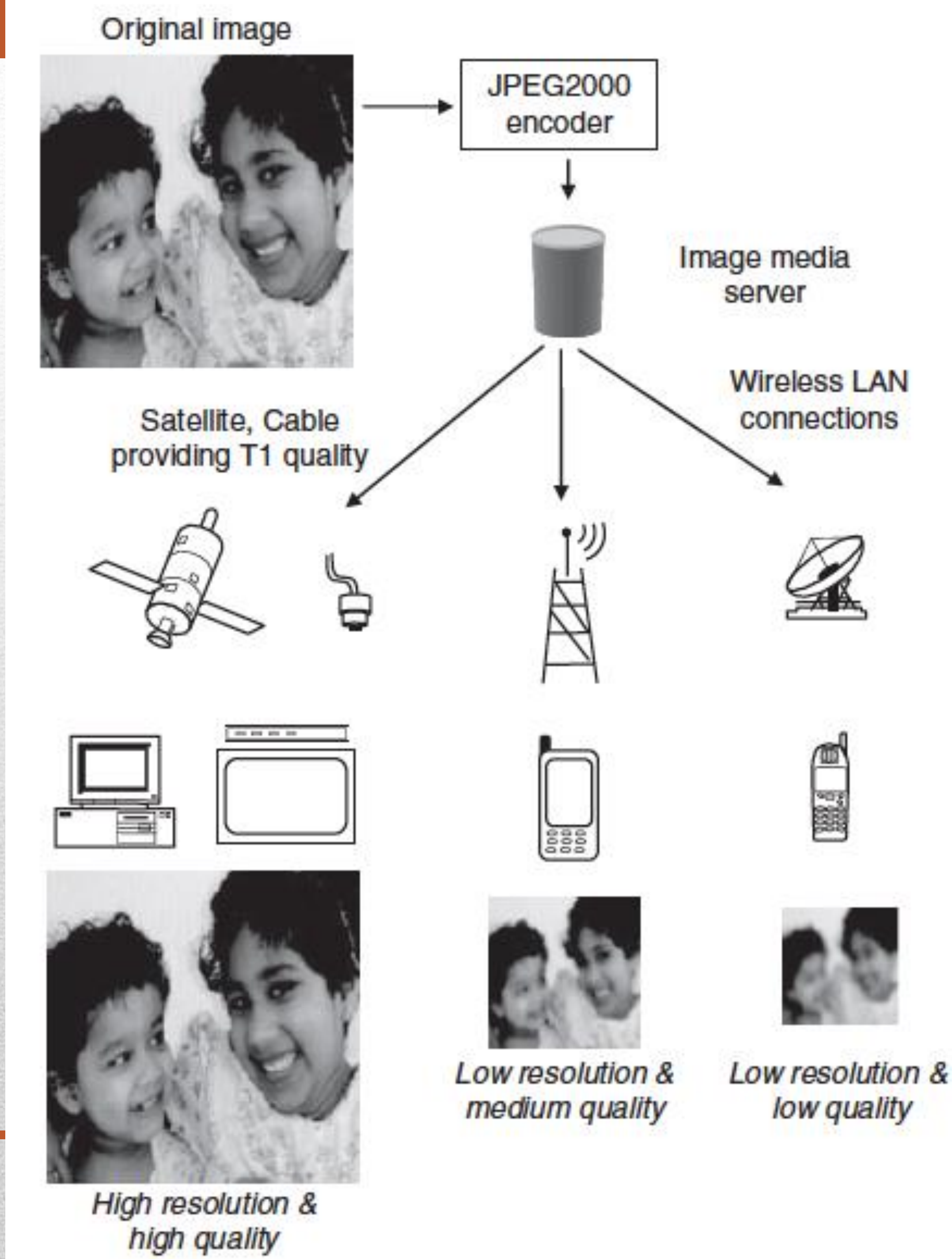
4) **Quantization of coefficients**

5) **Embedded Block Coding with Optimal Truncation (EBCOT)** algorithm, which is an advanced form of arithmetic coding.

Advantages of JPEG2000

- Encode once, Platform dependent decoding
  - We can decode at several frequency or spatial resolutions depending on the display and bandwidth available
- Working with compressed images:
  - Imaging operations can be performed on the compressed version (crop, flip, rotate,…)
- Region of interest encoding

Advantages of JPEG2000


Original image
JPEG2000 encoder
Image media server
Satellite, Cable providing T1 quality
Wireless LAN connections
High resolution & high quality
Low resolution & medium quality
Low resolution & low quality

Transmission issues

- **Spectral selection:**
- First scan: send all DC of all blocks
- Second scan: send 1^{st} AC coefficient
- Third scan: send 2^{nd} AC and so on
- **Successive bit approximation**
- All coefficients in one scan but bit by bit manner
- First scan: send MSB of all coef
- Second scan: send second MSB and so on.

Progressive Transmission of DCT-based images

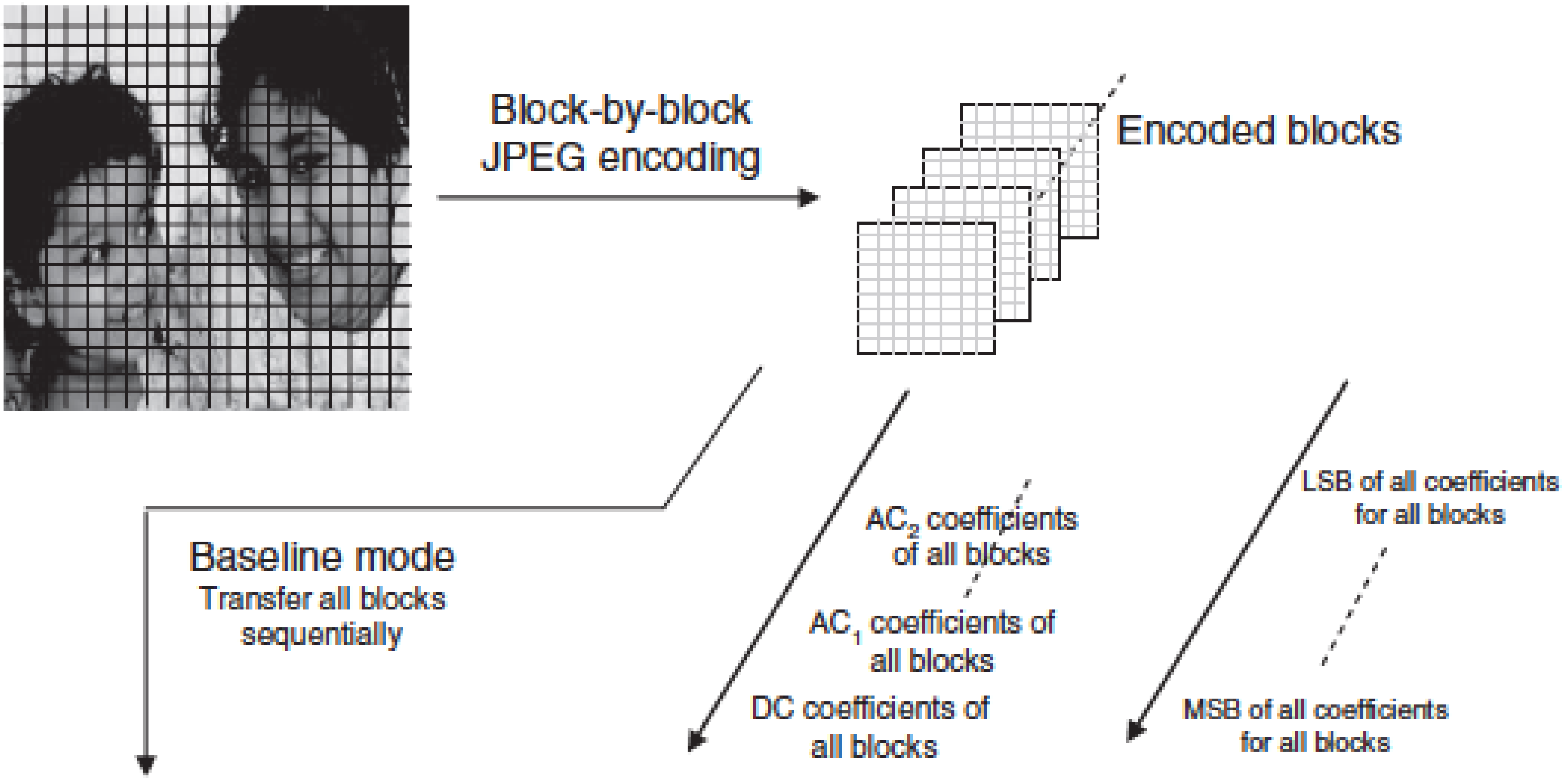
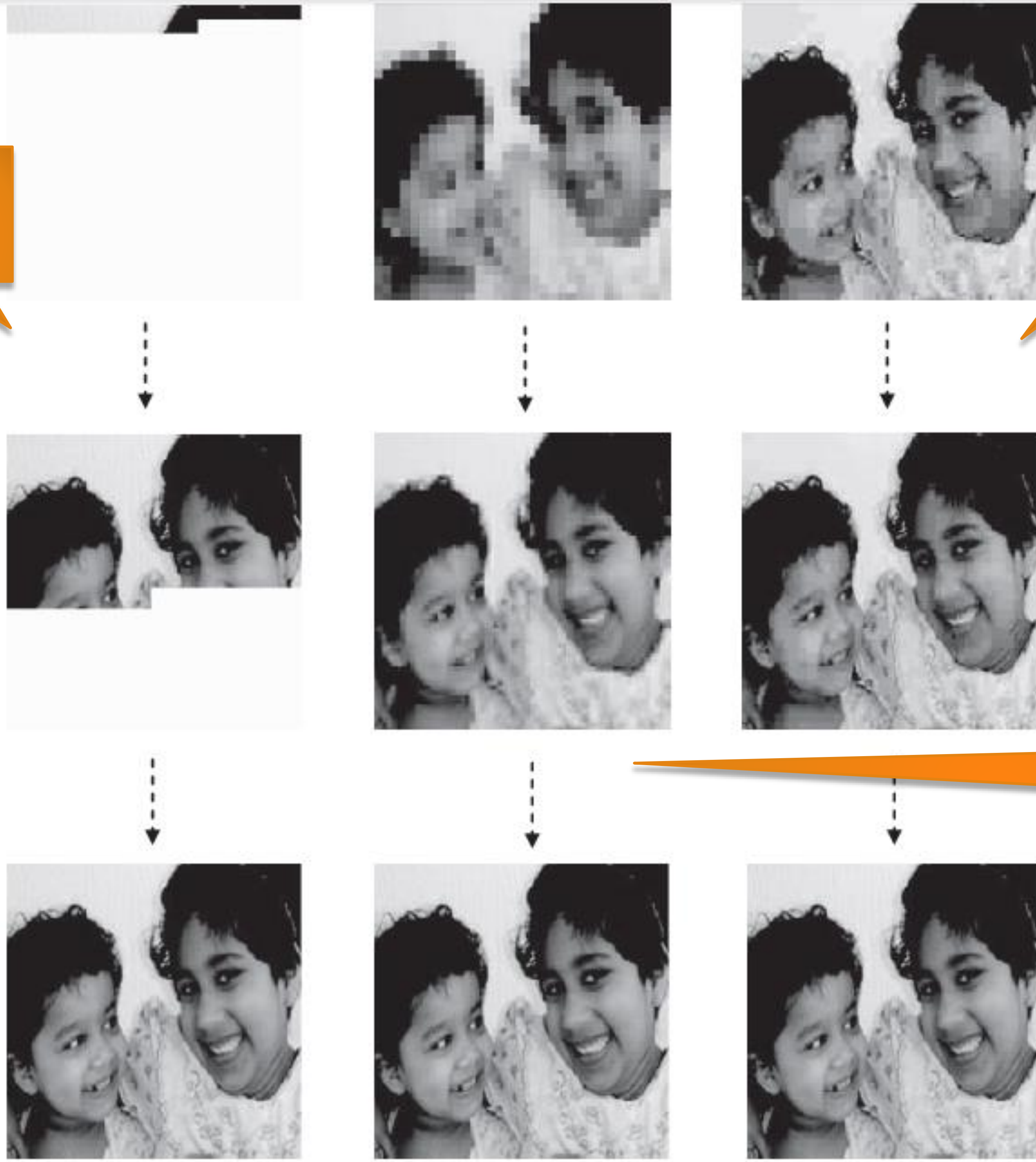


Figure 7-21 Progressive transmission in JPEG.

Progressive Transmission

baseline



Bit approximation

Spectral