CM3106 Multimedia

# JPEG

Dr Kirill Sidorov
SidorovK@cardiff.ac.uk
www.facebook.com/kirill.sidorov

Prof David Marshall
MarshallAD@cardiff.ac.uk

School of Computer Science and Informatics
Cardiff University, UK

# JPEG compression of images

## What is JPEG?

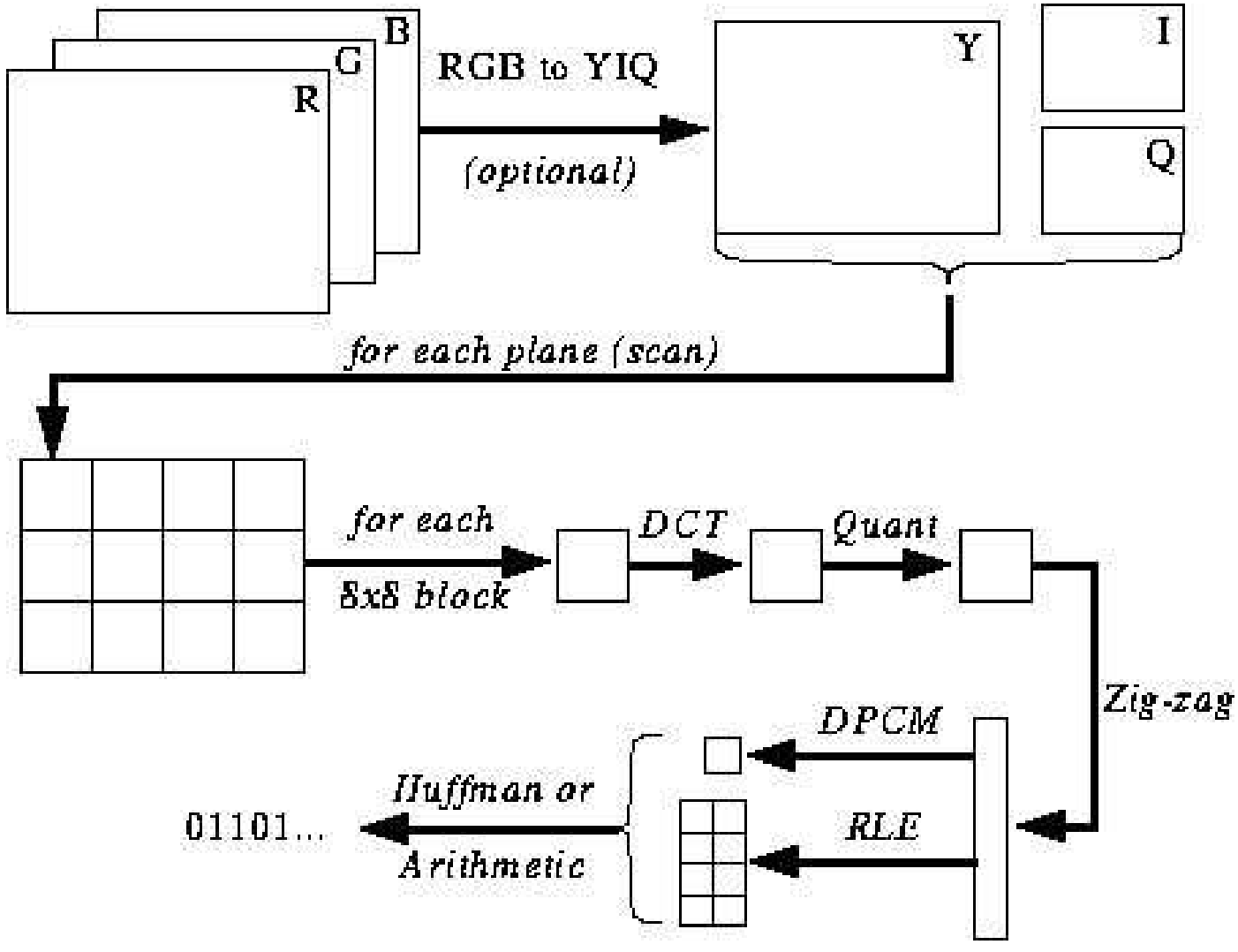
- JPEG: Joint Photographic Expert Group — an international standard since 1992.
- Works with colour and greyscale images.
- Up to 24 bit colour images (unlike GIF).
- Target photographic quality images (unlike GIF).
- Suitable for many applications *e.g.* satellite, medical, general photography...

## Basic idea:

- The human eye is less sensitive to fidelity in higher-frequency components.
- (Also less sensitive to colour than to intensity.)

# JPEG compression pipeline

JPEG compression involves the following:




Decoding — reverse the order for encoding.

# JPEG compression pipeline

The major steps in JPEG compression involve:

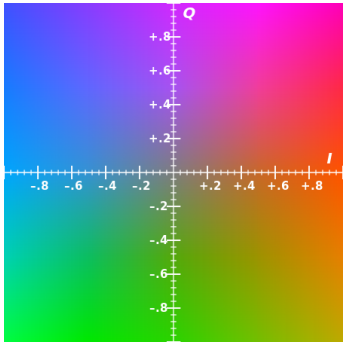
- Colour space transform and subsampling.
- DCT (Discrete Cosine Transform).
- Quantisation.
- Zigzag scan.
- DPCM on DC component.
- RLE on AC Components.
- Entropy coding — Huffman or arithmetic.

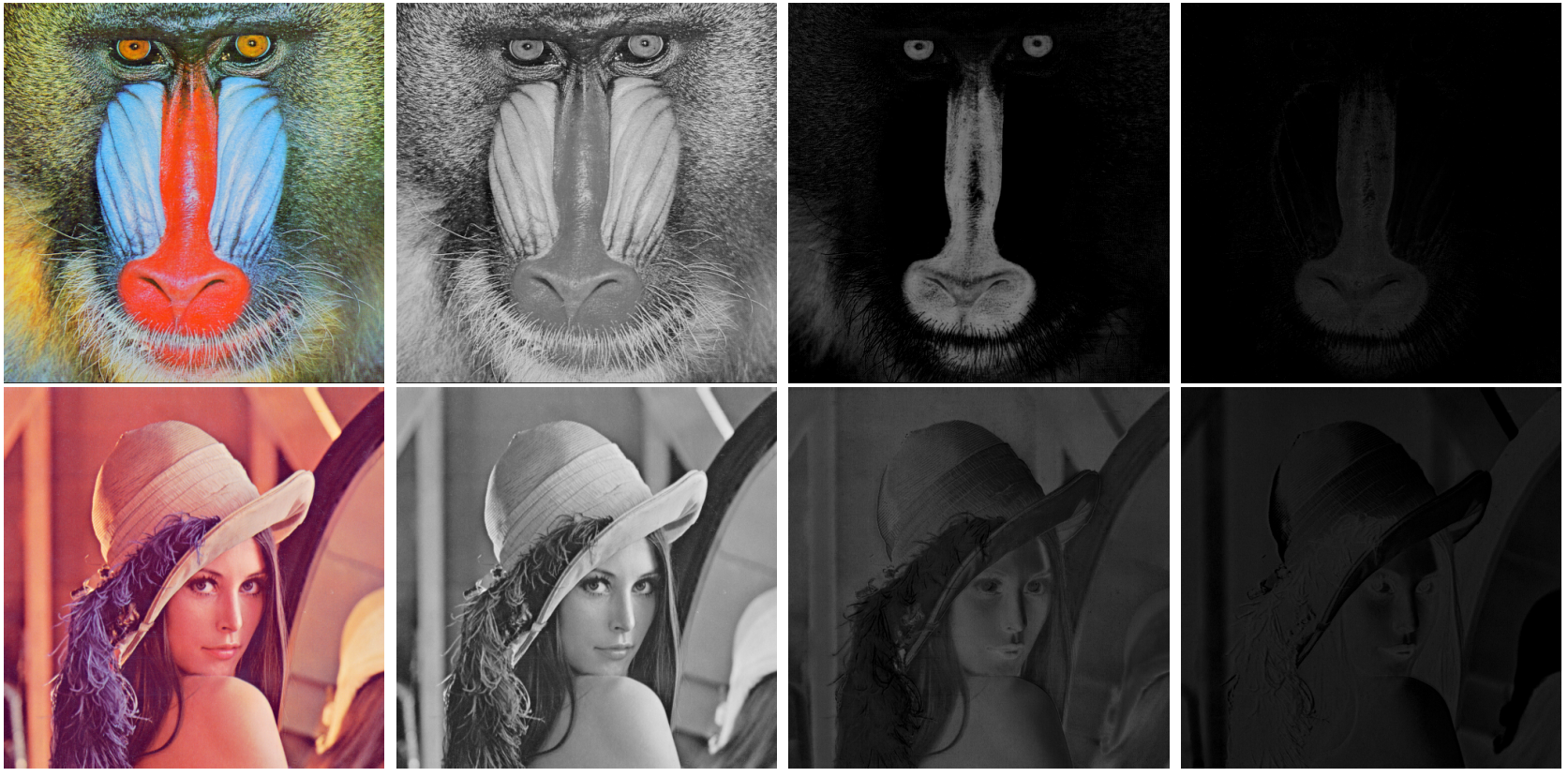
We have met most of the algorithms already:

- JPEG exploits them in the compression pipeline to achieve maximal overall compression.

# Colour space transform

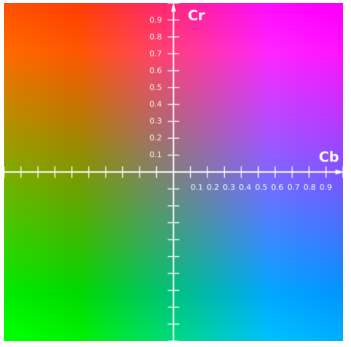
$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.211 & -0.523 & 0.312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

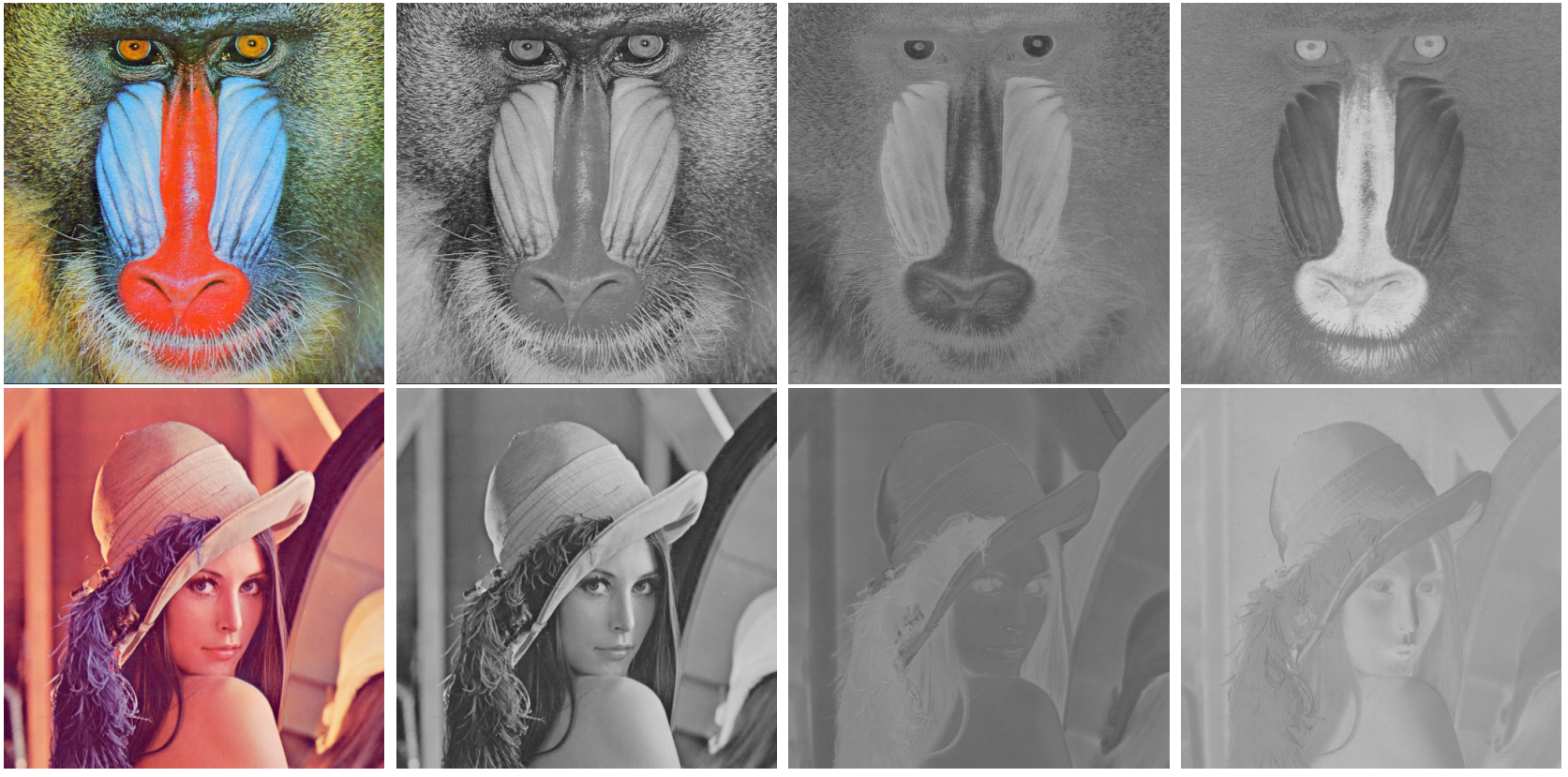


# Colour space transform

$$\begin{bmatrix} Y \\ Cb \\ Cr \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$



# Quantisation

- Encode the DCT coefficients more efficiently.
- Example (uniform quantization): $(101101)_2 = 45$ (in 6 bits of resolution)
  Quantise to 5 bits: $(10111)_2 = 23$
  Quantise to 4 bits: $(1011)_2 = 11$
  Quantise to 3 bits: $(110)_2 = 6$
  Quantise to 2 bits: $(11)_2 = 3$
- Quantisation error is the main source of lossy compression. DCT itself is not lossy.
- Divide by constant $Q$ and round result
  ($Q$ was $2^N$ in the above examples).
- Non powers-of-two gives fine control
  (*e.g.* $Q = 6$ loses $\log_2 6 \approx 2.59$ bits).

# Quantisation tables

- In JPEG, each $F[u, v]$ is divided by a constant $q(u, v)$.
- Table of $q(u, v)$ is called quantisation table.
- Eye is most sensitive to low frequencies (upper left corner), less sensitive to high frequencies (lower right corner).
- JPEG standard defines two default quantisation tables, one for luminance (below), one for chrominance.

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 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 |

# Quantisation tables

- Q: How would changing the numbers affect the picture?

  *E.g.* if we doubled them all?

  Quality factor in most implementations is the scaling factor for default quantization tables.
- Custom quantization tables can be used (part of JPEG header).

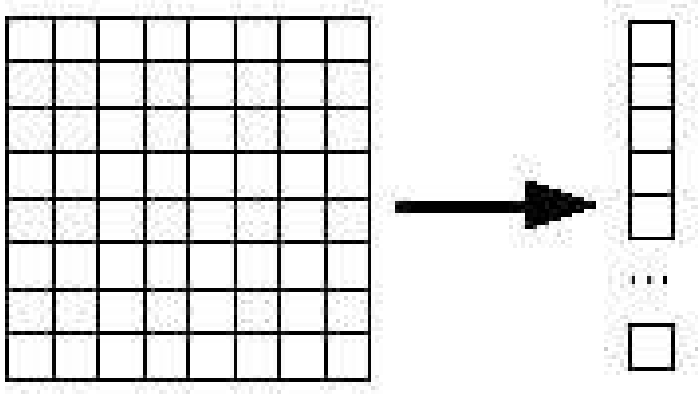
**JPEG Quantisation Example**

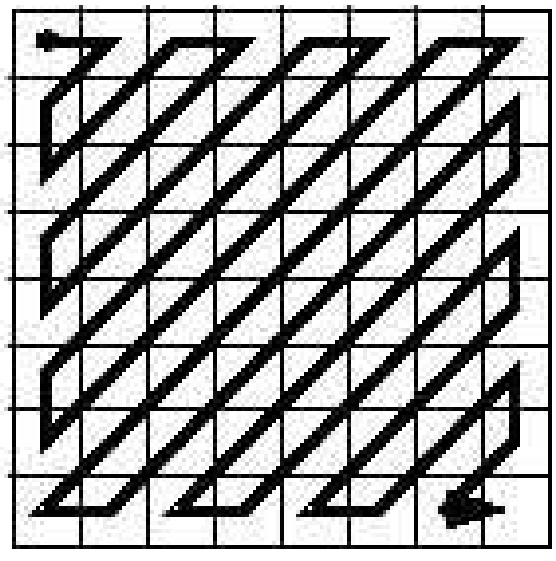
- JPEG Quantisation Example (Java Applet)

# Zig-zag scan

What is the purpose of the zig-zag scan:

- To group low frequency coefficients in top of vector.
- Maps $8 \times 8$ block to a $1 \times 64$ vector.





# DPCM on DC component

- DPCM is then employed on the DC component.
- Why is this strategy adopted:
  - DC component is large and varies slowly, often close to previous value (strongly correlated).
  - Encode the difference from previous $8 \times 8$ blocks — DPCM.

# RLE on AC components

Yet another simple compression technique is applied to the AC component:

- $1 \times 63$ vector (AC) has lots of zeros in it.
- Encode as (*skip*, *value*) pairs, where *skip* is the number of zeros and *value* is the next non-zero component.
- Send (0, 0) as end-of-block sentinel value.

# Entropy coding

DC and AC components finally are subject to entropy coding (arithmetic coding or Huffman coding):

- (Variant of) Huffman coding: each DPCM-coded DC coefficient is represented by a pair of symbols : (`Size`, `Amplitude`)
  where `Size` indicates the number of bits needed to represent the coefficient and `Amplitude` contains the bits representing the coefficient.
- Only the `Size` is Huffman-coded in JPEG:
  - `Size` does not change too much, generally smaller `Size`s have skewed distribution (= low entropy so is suitable for entropy coding).
  - `Amplitude` can change widely so coding has no real benefit.

- Example: `Size` category for possible `Amplitudes`:

| Size | Typical Huffman code for Size | Amplitude |
|---|---|---|
| 0 | 00 | 0 |
| 1 | 010 | -1,1 |
| 2 | 011 | -3,-2,2,3 |
| 3 | 100 | -7..-4,4..7 |
| 4 | 101 | -15..-8,8..15 |
| . | . | . |
| . | . | . |

- Use one's complement scheme for negative values: *e.g.* 10 is binary for 2 and 01 for -2 (bitwise inverse). Similarly, 00 for -3 and 11 for 3.

# Huffman coding of DC example

- Example: if DC values are 150, -6, 5, 3, -8
- Then 8, 3, 3, 2 and 4 bits are needed respectively.
  Send off `Size`s as Huffman symbol, followed by actual values in bits:

  $$(8_{\text{huff}}, 10010110), (3_{\text{huff}}, 001), (3_{\text{huff}}, 101), (2_{\text{huff}}, 11), (4_{\text{huff}}, 0111)$$

  where $8_{\text{huff}}$ ... are the Huffman codes for respective numbers.
- Huffman tables can be custom (sent in header) or default.

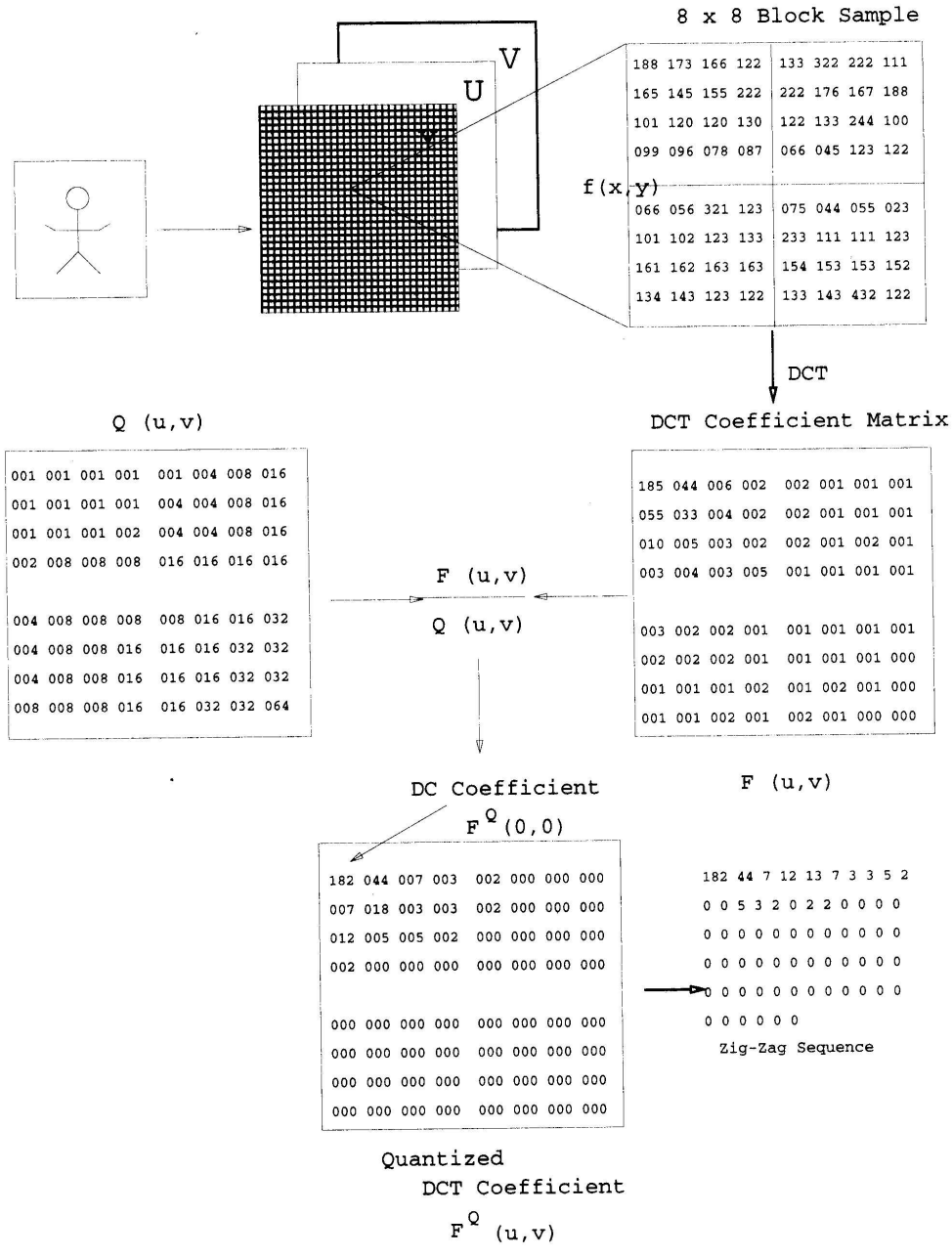
# Huffman Coding on AC Component

AC coefficients are run-length encoded (RLE)

- RLE pairs (`Runlength`, `Value`) are Huffman-coded as with DC only on `Value`.
- So we get a triple: (`Runlength`, `Size`, `Amplitude`)
- However, `Runlength`, `Size` allocated 4-bits each and put into a single byte with is then Huffman coded.
  Again, `Amplitude` is not coded.
- So only two symbols transmitted per RLE coefficient:

$$(\texttt{RLESIZEbyte}_{\text{huff}}, \texttt{Amplitude})$$

# Example of JPEG compression



# Another numeric example

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 139 | 144 | 149 | 153 | 155 | 155 | 155 | 155 |
| 144 | 151 | 153 | 156 | 159 | 156 | 156 | 156 |
| 150 | 155 | 160 | 163 | 158 | 156 | 156 | 156 |
| 159 | 161 | 162 | 160 | 160 | 159 | 159 | 159 |
| 159 | 160 | 161 | 162 | 162 | 155 | 155 | 155 |
| 161 | 161 | 161 | 161 | 160 | 157 | 157 | 157 |
| 162 | 162 | 161 | 163 | 162 | 157 | 157 | 157 |
| 162 | 162 | 161 | 161 | 163 | 158 | 158 | 158 |

(a) source image samples

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 235.6 | -1.0 | -12.1 | -5.2 | 2.1 | -1.7 | -2.7 | 1.3 |
| -22.6 | -17.5 | -6.2 | -3.2 | -2.9 | -0.1 | 0.4 | -1.2 |
| -10.9 | -9.3 | -1.6 | 1.5 | 0.2 | -0.9 | -0.6 | -0.1 |
| -7.1 | -1.9 | 0.2 | 1.5 | 0.9 | -0.1 | 0.0 | 0.3 |
| -0.6 | -0.8 | 1.5 | 1.6 | -0.1 | -0.7 | 0.6 | 1.3 |
| 1.8 | -0.2 | 1.6 | -0.3 | -0.8 | 1.5 | 1.0 | -1.0 |
| -1.3 | -0.4 | -0.3 | -1.5 | -0.5 | 1.7 | 1.1 | -0.8 |
| -2.6 | 1.6 | -3.8 | -1.8 | 1.9 | 1.2 | -0.6 | -0.4 |

(b) forward DCT coefficients

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 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 |

(c) quantization table

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 15 | 0 | -1 | 0 | 0 | 0 | 0 | 0 |
| -2 | -1 | 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 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

(d) normalized quantized coefficients

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 240 | 0 | -10 | 0 | 0 | 0 | 0 | 0 |
| -24 | -12 | 0 | 0 | 0 | 0 | 0 | 0 |
| -14 | -13 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |

(e) denormalized quantized coefficients

| | | | | | | | |
|---|---|---|---|---|---|---|---|
| 144 | 146 | 149 | 152 | 154 | 156 | 156 | 156 |
| 148 | 150 | 152 | 154 | 156 | 156 | 156 | 156 |
| 155 | 156 | 157 | 158 | 158 | 157 | 156 | 155 |
| 160 | 161 | 161 | 162 | 161 | 159 | 157 | 155 |
| 163 | 163 | 164 | 163 | 162 | 160 | 158 | 156 |
| 163 | 164 | 164 | 164 | 162 | 160 | 158 | 157 |
| 160 | 161 | 162 | 162 | 162 | 161 | 159 | 158 |
| 158 | 159 | 161 | 161 | 162 | 161 | 159 | 158 |

(f) reconstructed image samples

# JPEG: example MATLAB implementation

The JPEG algorithm may be summarised as follows:

im2jpeg.m (encoder) jpeg2im.m (decoder)

mat2huff.m (Huffman coder)

```
m = [16 11 10 16 24 40 51 61 % JPEG normalizing array
12 12 14 19 26 58 60 55 % and zig-zag reordering
14 13 16 24 40 57 69 56 % pattern.
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] * quality;

order = [1 9 2 3 10 17 25 18 11 4 5 12 19 26 33 ...  41 34 27 20 13 6 7 14
21 28 35 42 49 57 50 ...  43 36 29 22 15 8 16 23 30 37 44 51 58 59 52 ...
45 38 31 24 32 39 46 53 60 61 54 47 40 48 55 ...  62 63 56 64];

[xm, xn] = size(x); % Get input size.
x = double(x) - 128; % Level shift input
t = dctmtx(8); % Compute 8 x 8 DCT matrix

% Compute DCTs of 8x8 blocks and quantize the coefficients.
y = blkproc(x, [8 8], 'P1 * x * P2', t, t'); y = blkproc(y, [8 8],
'round(x ./ P1)', m);
```

# JPEG Example MATLAB Code

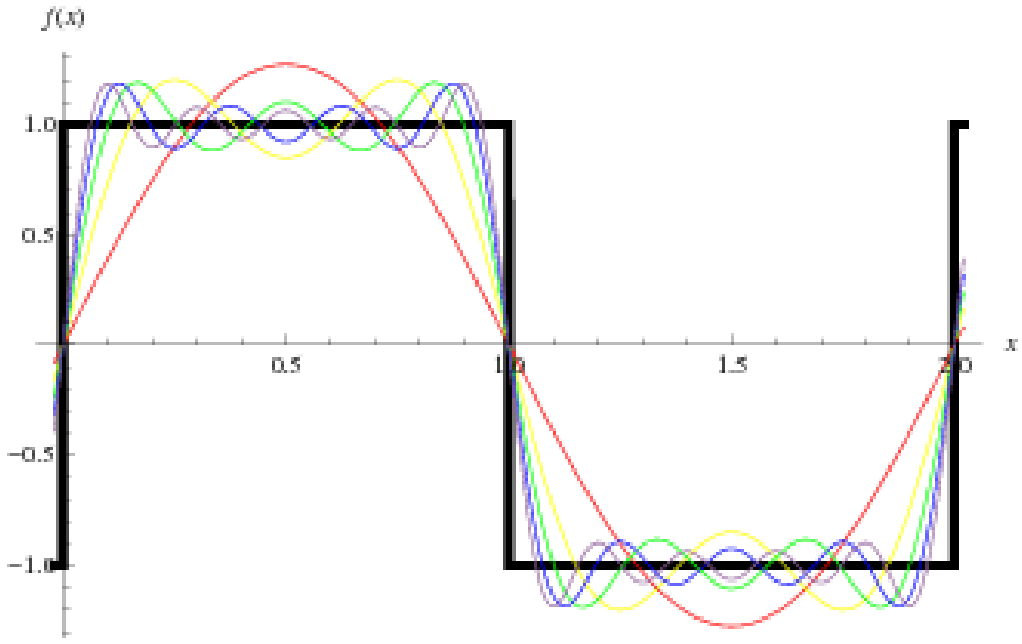
```
y = im2col(y, [8 8], 'distinct'); % Break 8x8 blocks into columns
xb = size(y, 2); % Get number of blocks
y = y(order, :); % Reorder column elements

eob = max(y(:)) + 1; % Create end-of-block symbol
r = zeros(numel(y) + size(y, 2), 1); count = 0;
for j = 1:xb % Process 1 block (col) at a time
  i = max(find(y(:, j))); % Find last non-zero element
  if isempty(i) % No nonzero block values
    i = 0; end;
  p = count + 1; q = p + i;
  r(p:q) = [y(1:i, j); eob]; % Truncate trailing 0's, add EOB,
  count = count + i + 1; % and add to output vector
end

r((count + 1):end) = []; % Delete unusued portion of r

y = struct; y.size = uint16([xm xn]); y.numblocks = uint16(xb);
y.quality = uint16(quality * 100); y.huffman = mat2huff(r);
```

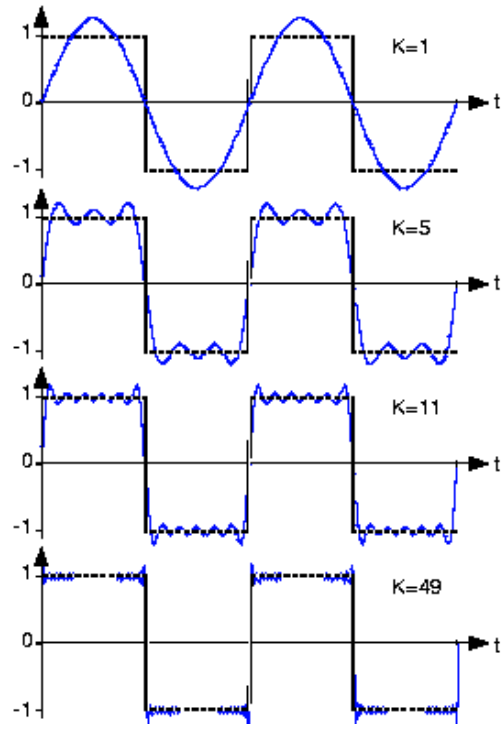
# Artefacts

- This image is compressed increasingly more from left to right.
- Note ringing artefacts and blocking artefacts.

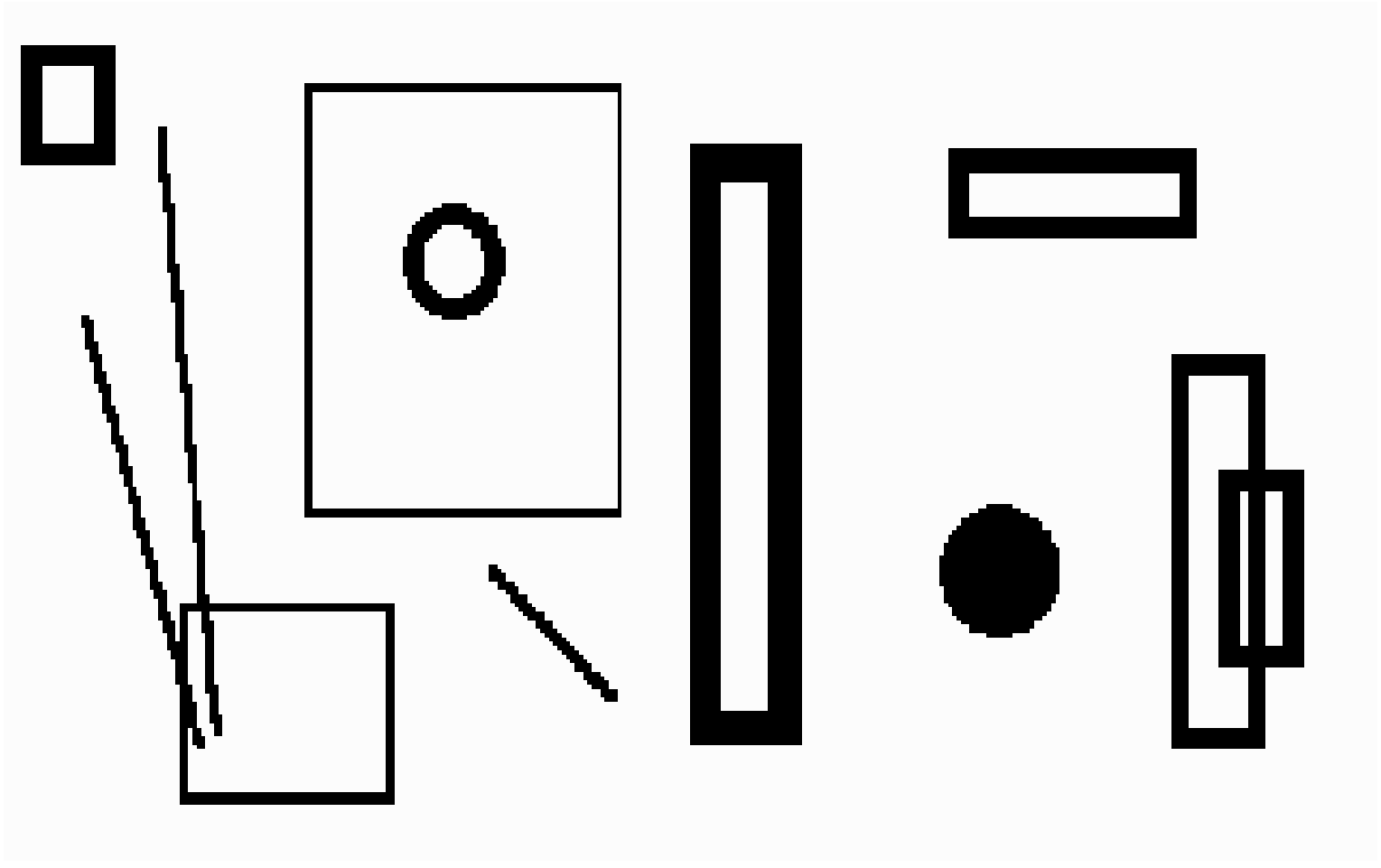
# Gibb's Phenomenon

- Artefacts around sharp boundaries are due to **Gibb's phenomenon**.
- Basically: inability of a finite combination of cosines to describe jump discontinuities.

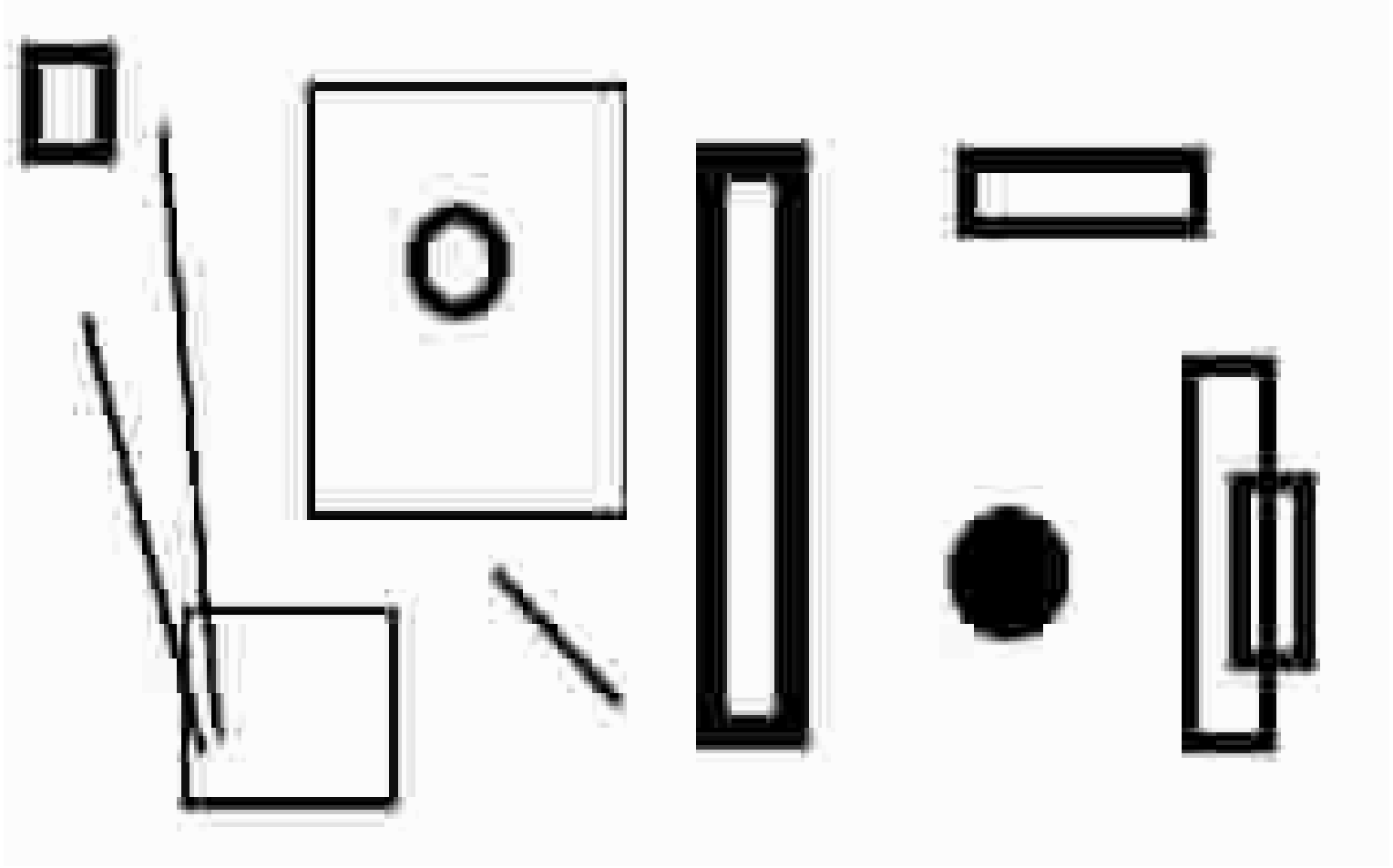


# Gibb's Phenomenon



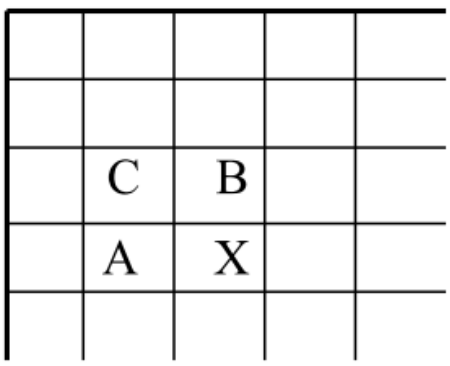
# Gibb's Phenomenon

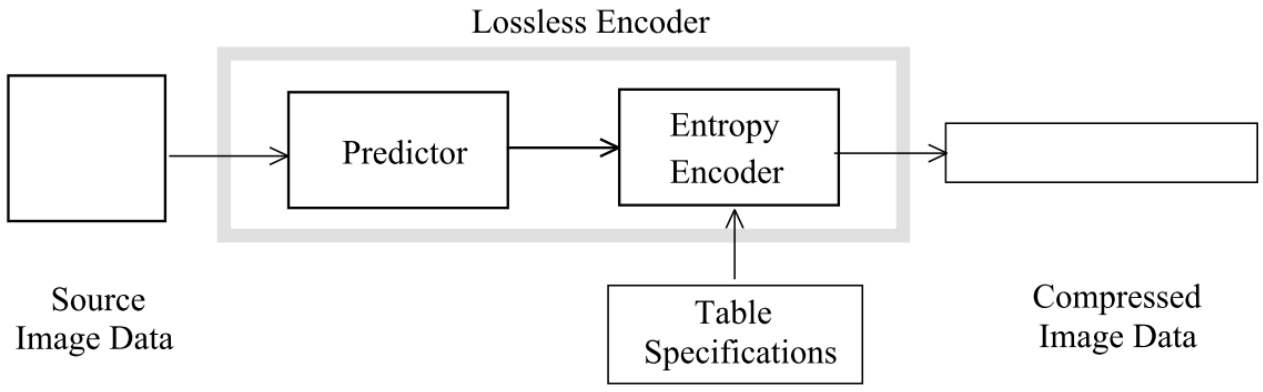


# Other variants of JPEG

Further standards:

- Lossless JPEG: Predictive approach for lossless compression, not widely used.




- JPEG 2000: ISO/IEC 15444
  - Based on wavelet transform, instead of DCT, no $8 \times 8$ blocks, fewer artefacts.
  - Often better compression ratio, compared with JPEG.

# Further reading

References:

- http://www.jpeg.org
- Online JPEG Tutorial
- The JPEG Still Picture Compression Standard
- The JPEG 2000 Still Image Compression Standard