

Compression: Images (JPEG)

What is JPEG?

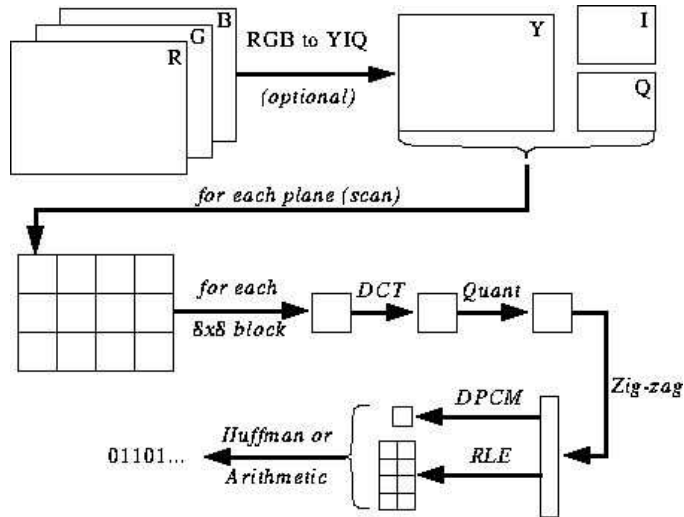
- **JPEG: Joint Photographic Expert Group** — an international standard since 1992.
- Works with colour and greyscale images
- Up to **24 bit colour** images (**Unlike GIF**)
- Target **photographic** quality images (**Unlike GIF**)
- Suitable for many applications e.g., satellite, medical, general photography...



Basic JPEG Compression Pipeline

JPEG compression involves the following:

- Encoding



- Decoding – Reverse the order for encoding

Major Coding Algorithms in JPEG

The Major Steps in JPEG Coding involve:

- Colour Space Transform and subsampling (YIQ)
- DCT (Discrete Cosine Transformation)
- 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.



Back

Close

Quantisation

Why do we need to quantise:

- To throw out bits from DCT.
- *Example:* $(101101)_2 = 45$ (6 bits).

Truncate to 4 bits: $(1011)_2 = 11$.

Truncate to 3 bits: $(101)_2 = 5$.

- Quantisation error is the main source of **Lossy Compression**.
- **DCT itself is not Lossy**
- How we **throw away bits** in **Quantisation Step** is **Lossy**



Back

Close

Quantisation Methods

Uniform quantisation

- Divide by constant N and round result
($N = 4$ or 8 in examples on previous page).
- Non powers-of-two gives fine control
(e.g., $N = 6$ loses 2.5 bits)



Back

Close

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 2 default quantisation tables, one for luminance (below), one for chrominance. *E.g Table below*

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



Back

Close

Quantization Tables (Cont)

- 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 put in image/scan header.

JPEG Quantisation Example

- [JPEG Quantisation Example \(Java Applet\)](#)



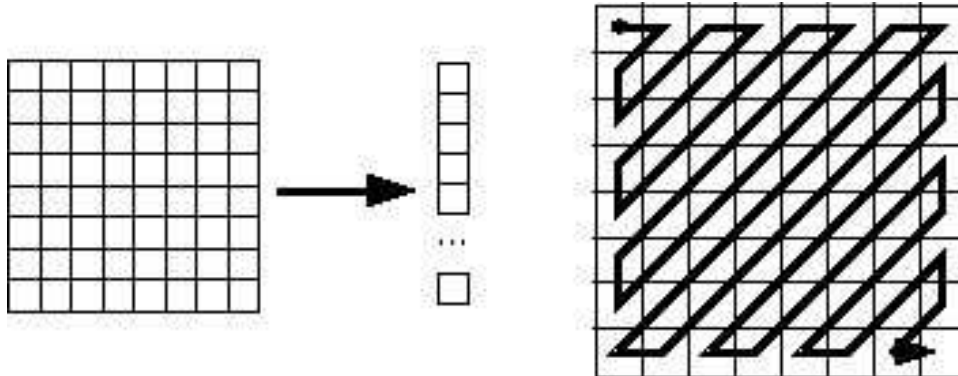
Back

Close

Zig-zag Scan

What is the purpose of the Zig-zag Scan:

- To group low frequency coefficients in top of vector.
- Maps 8 x 8 to a 1 x 64 vector



Back

Close

Differential Pulse Code Modulation (DPCM) on DC Component

- Another encoding method is employed
- DPCM on the DC component.
- Why is this strategy adopted:
 - DC component is large and varies, but often close to previous value (like lossless JPEG).
 - Encode the difference from previous 8x8 blocks – DPCM



Back

Close

Run Length Encode (RLE) on AC Components

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

- 1x63 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.

[Back](#)[Close](#)

Huffman (Entropy) Coding

DC and AC components finally need to be represented by a smaller number of bits

(Arithmetic coding also supported in place of Huffman coding):

- (Variant of) Huffman coding: Each DPCM-coded DC coefficient is represented by a pair of symbols :
(*Size*, *Amplitude*)
where *Size* indicates number of bits needed to represent coefficient and
Amplitude contains actual bits.
- *Size* **only** Huffman coded in JPEG:
 - *Size* does not change too much, generally smaller *Sizes* occur frequently (= **low entropy** so is suitable for coding,
 - *Amplitude* can change widely so coding **no real benefit**



Back

Close

Huffman (Entropy) Coding (Cont)

- 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 *ones complement* scheme for negative values: *i.e* 10 is binary for 2 and 01 for -2 (bitwise inverse). Similarly, 00 for -3 and 11 for 3.



Back

Close

Huffman Coding 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 `Sizes` as Huffman symbol, followed by actual values in bits.

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

where $\delta_{huff} \dots$ are the Huffman codes for respective numbers.

- Huffman Tables can be custom (sent in header) or default.



Back

Close

Huffman Coding on AC Component

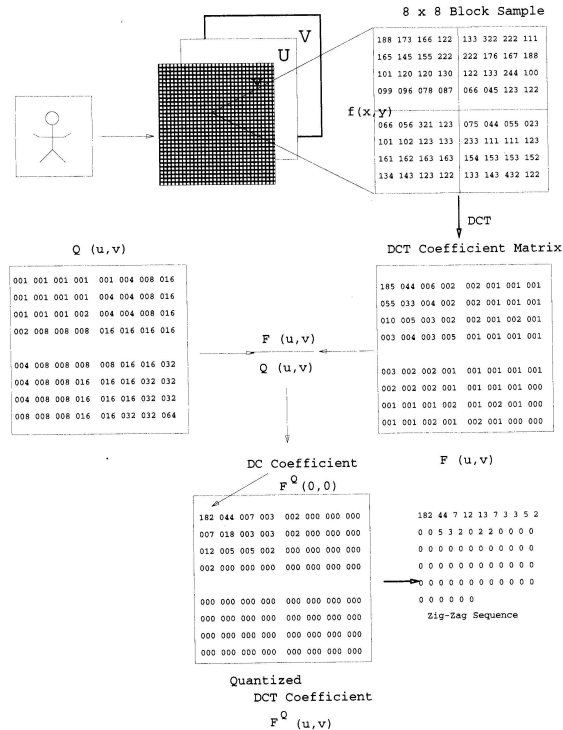
AC coefficient are **run-length encoded (RLE)**

- 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:

$(\text{RLE SIZE byte}_{huff}, \text{Amplitude})$



Example JPEG Compression



Another Enumerated Example

139	144	149	153	155	155	155	155	235.6	-1.0	-12.1	-5.2	2.1	-1.7	-2.7	1.3	16	11	10	16	24	40	51	61
144	151	153	156	159	156	156	156	-22.6	-17.5	-6.2	-3.2	-2.9	-0.1	0.4	-1.2	12	12	14	19	26	58	60	55
150	155	160	163	158	156	156	156	-10.9	-9.3	-1.6	1.5	0.2	-0.9	-0.6	-0.1	14	13	16	24	40	57	69	56
159	161	162	160	160	159	159	159	-7.1	-1.9	0.2	1.5	0.9	-0.1	0.0	0.3	14	17	22	29	51	87	80	62
159	160	161	162	162	155	155	155	-0.6	-0.8	1.5	1.6	-0.1	-0.7	0.6	1.3	18	22	37	56	68	109	103	77
161	161	161	161	160	157	157	157	1.8	-0.2	1.6	-0.3	-0.8	1.5	1.0	-1.0	24	35	55	64	81	104	113	92
162	162	161	163	162	157	157	157	-1.3	-0.4	-0.3	-1.5	-0.5	1.7	1.1	-0.8	49	64	78	87	103	121	120	101
162	162	161	161	163	158	158	158	-2.6	1.6	-3.8	-1.8	1.9	1.2	-0.6	-0.4	72	92	95	98	112	100	103	99

(a) source image samples

(b) forward DCT coefficients

(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
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
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 Code

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);
```



Back

Close

```

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 unused portion of r

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

```



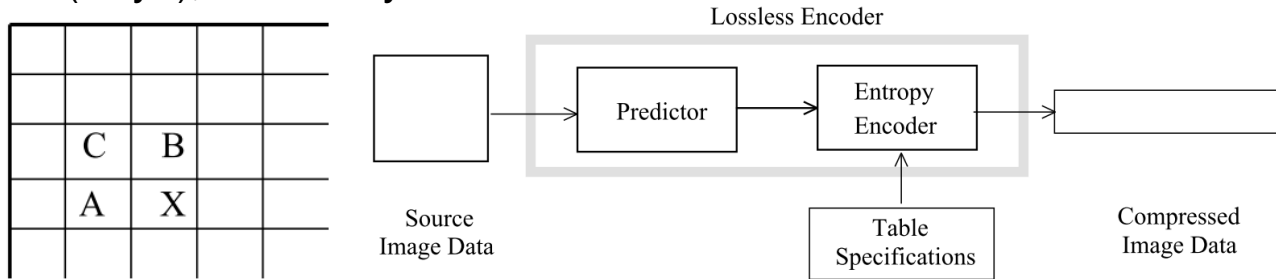
Back

Close

Further Information

Further standards:

- Lossless JPEG: Predictive approach for lossless compression (why?), not widely used



- JPEG 2000: ISO/IEC 15444
 - Based on **wavelet** transform, instead of DCT, no 8×8 blocks, less artefacts
 - Often better compression ratio, compared with JPEG



Further Information

References:

- <http://www.jpeg.org>
- [Online JPEG Tutorial](#)
- [The JPEG Still Picture Compression Standard](#)
- [The JPEG 2000 Still Image Compression Standard](#)

