JPEG

JPEG

Filename extension	`.jpg`, `.jpeg`, `.jpe` `.jif`, `.jfif`, `.jfi`
Internet media type	`image/jpeg`
Type code	`JPEG`
Uniform Type Identifier	public.jpeg
Magic number	`ff d8`
Developed by	Joint Photographic Experts Group
Type of format	lossy image format
Standard(s)	ISO/IEC 10918, ITU-T T.81, ITU-T T.83, ITU-T T.84, ITU-T T.86

In computing, **JPEG** (pron.: /ˈdʒeɪpɛɡ/ ***JAY**-peg*) is a commonly used method of lossy compression for digital photography (image). The degree of compression can be adjusted, allowing a selectable tradeoff between storage size and image quality. JPEG typically achieves 10:1 compression with little perceptible loss in image quality.

Continuously varied JPEG compression (between Q=100 and Q=0) for an abdominal CT scan.

JPEG compression is used in a number of image file formats. JPEG/Exif is the most common image format used by digital cameras and other photographic image capture devices; along with JPEG/JFIF, it is the most common format for storing and transmitting photographic images on the World Wide Web.[*citation needed*] These format variations are often not distinguished, and are simply called JPEG.

The term "JPEG" is an acronym for the Joint Photographic Experts Group, which created the standard. The MIME media type for JPEG is *image/jpeg* (defined in RFC 1341), except in Internet Explorer, which provides a MIME type of *image/pjpeg* when uploading JPEG images.[1]

It supports a maximum image size of 65535×65535.[2]

The JPEG standard

The name "JPEG" stands for Joint Photographic Experts Group, the name of the committee that created the JPEG standard and also other still picture coding standards. The "Joint" stood for ISO TC97 WG8 and CCITT SGVIII. In 1987 ISO TC 97 became ISO/IEC JTC1 and in 1992 CCITT became ITU-T. Currently on the JTC1 side JPEG is one of two sub-groups of ISO/IEC Joint Technical Committee 1, Subcommittee 29, Working Group 1 (ISO/IEC JTC 1/SC 29/WG 1) – titled as *Coding of still pictures*.[][][3] On the ITU-T side ITU-T SG16 is the respective body. The original JPEG group was organized in 1986,[] issuing the first JPEG standard in 1992, which was approved in September 1992 as **ITU-T Recommendation T.81**[4] and in 1994 as **ISO/IEC 10918-1**.

The JPEG standard specifies the codec, which defines how an image is compressed into a stream of bytes and decompressed back into an image, but not the file format used to contain that stream.[5] The Exif and JFIF standards define the commonly used file formats for interchange of JPEG-compressed images.

JPEG standards are formally named as *Information technology – Digital compression and coding of continuous-tone still images*. ISO/IEC 10918 consists of the following parts:

JPEG standard – Parts[][][6]

Part	ISO/IEC standard	ITU-T Rec.	First public release date	Latest amendment	Title	Description
Part 1	ISO/IEC 10918-1:1994 [7]	T.81 (09/92) [8]	1992		Requirements and guidelines	
Part 2	ISO/IEC 10918-2:1995 [9]	T.83 (11/94) [10]	1994		Compliance testing	rules and checks for software conformance (to Part 1)
Part 3	ISO/IEC 10918-3:1997 [11]	T.84 (07/96) [12]	1996	1999	Extensions	set of extensions to improve the Part 1, including the SPIFF file format
Part 4	ISO/IEC 10918-4:1999 [13]	T.86 (06/98) [14]	1998		Registration of JPEG profiles, SPIFF profiles, SPIFF tags, SPIFF colour spaces, APPn markers, SPIFF compression types and Registration Authorities (REGAUT)	methods for registering some of the parameters used to extend JPEG
Part 5ioou4	ISO/IEC FDIS 10918-5 [15]	T.871 (05/11) [16]	(under development since 2009)[]		JPEG File Interchange Format (JFIF)	A popular format which has been the de facto file format for images encoded by the JPEG standard. In 2009, the JPEG Committee formally established an Ad Hoc Group to standardize JFIF as JPEG Part 5.[]

Ecma International TR/98 specifies the JPEG File Interchange Format (JFIF); the first edition was published in June 2009.[17]

Typical usage

The JPEG compression algorithm is at its best on photographs and paintings of realistic scenes with smooth variations of tone and color. For web usage, where the amount of data used for an image is important, JPEG is very popular. JPEG/Exif is also the most common format saved by digital cameras.

On the other hand, JPEG may not be as well suited for line drawings and other textual or iconic graphics, where the sharp contrasts between adjacent pixels can cause noticeable artifacts. Such images may be better saved in a lossless graphics format such as TIFF, GIF, PNG, or a raw image format. The JPEG standard actually includes a lossless coding mode, but that mode is not supported in most products.

As the typical use of JPEG is a lossy compression method, which somewhat reduces the image fidelity, it should not be used in scenarios where the exact reproduction of the data is required (such as some scientific and medical imaging applications and certain technical image processing work).

JPEG is also not well suited to files that will undergo multiple edits, as some image quality will usually be lost each time the image is decompressed and recompressed, particularly if the image is cropped or shifted, or if encoding parameters are changed – see digital generation loss for details. To avoid this, an image that is being modified or may be modified in the future can be saved in a lossless format, with a copy exported as JPEG for distribution.

JPEG compression

JPEG uses a lossy form of compression based on the discrete cosine transform (DCT). This mathematical operation converts each frame/field of the video source from the spatial (2D) domain into the frequency domain (aka transform domain.) A perceptual model based loosely on the human psychovisual system discards high-frequency information, i.e. sharp transitions in intensity, and color hue. In the transform domain, the process of reducing information is called quantization. In laymen's terms, quantization is a method for optimally reducing a large number scale (with different occurrences of each number) into a smaller one, and the transform-domain is a convenient representation of the image because the high-frequency coefficients, which contribute less to the over picture than other coefficients, are characteristically small-values with high compressibility. The quantized coefficients are then sequenced and losslessly packed into the output bitstream. Nearly all software implementations of JPEG permit user control over the compression-ratio (as well as other optional parameters), allowing the user to trade off picture-quality for smaller file size. In embedded applications (such as miniDV, which uses a similar DCT-compression scheme), the parameters are pre-selected and fixed for the application.

The compression method is usually lossy, meaning that some original image information is lost and cannot be restored, possibly affecting image quality. There is an optional lossless mode defined in the JPEG standard; however, this mode is not widely supported in products.

There is also an interlaced "Progressive JPEG" format, in which data is compressed in multiple passes of progressively higher detail. This is ideal for large images that will be displayed while downloading over a slow connection, allowing a reasonable preview after receiving only a portion of the data. However, progressive JPEGs are not as widely supported,[*citation needed*] and even some software which does support them (such as versions of Internet Explorer before Windows 7)[18] only displays the image after it has been completely downloaded.

There are also many medical imaging and traffic systems that create and process 12-bit JPEG images, normally grayscale images. The 12-bit JPEG format has been part of the JPEG specification for some time, but again, this format is not as widely supported.

Lossless editing

A number of alterations to a JPEG image can be performed losslessly (that is, without recompression and the associated quality loss) as long as the image sizeUnit) (usually 16 pixels in both directions, for 4:2:0 chroma subsampling). Utilities that implement this include `jpegtran,` with user interface Jpegcrop, and the `JPG_TRANSFORM` plugin to IrfanView. the graphics are consisted of pixels

Blocks can be rotated in 90 degree increments, flipped in the horizontal, vertical and diagonal axes and moved about in the image. Not all blocks from the original image need to be used in the modified one.

The top and left edge of a JPEG image must lie on a 8 × 8 pixel block boundary, but the bottom and right edge need not do so. This limits the possible lossless crop operations, and also prevents flips and rotations of an image whose bottom or right edge does not lie on a block boundary for all channels (because the edge would end up on top or left, where – as aforementioned – a block boundary is obligatory).

When using lossless cropping, if the bottom or right side of the crop region is not on a block boundary then the rest of the data from the partially used blocks will still be present in the cropped file and can be recovered.

It is also possible to transform between baseline and progressive formats without any loss of quality, since the only difference is the order in which the coefficients are placed in the file.

Furthermore, several JPEG images can be losslessly joined together, as long as the edges coincide with block boundaries.

JPEG files

The file format known as "JPEG Interchange Format" (JIF) is specified in Annex B of the standard. However, this "pure" file format is rarely used, primarily because of the difficulty of programming encoders and decoders that fully implement all aspects of the standard and because of certain shortcomings of the standard:

- Color space definition
- Component sub-sampling registration
- Pixel aspect ratio definition.

Several additional standards have evolved to address these issues. The first of these, released in 1992, was JPEG File Interchange Format (or JFIF), followed in recent years by Exchangeable image file format (Exif) and ICC color profiles. Both of these formats use the actual JIF byte layout, consisting of different *markers*, but in addition employ one of the JIF standard's extension points, namely the *application markers*: JFIF use APP0, while Exif use APP1. Within these segments of the file, that were left for future use in the JIF standard and aren't read by it, these standards add specific metadata.

Thus, in some ways JFIF is a cutdown version of the JIF standard in that it specifies certain constraints (such as not allowing all the different encoding modes), while in other ways it is an extension of JIF due to the added metadata. The documentation for the original JFIF standard states:[19]

> *JPEG File Interchange Format is a minimal file format which enables JPEG bitstreams to be exchanged between a wide variety of platforms and applications. This minimal format does not include any of the advanced features found in the TIFF JPEG specification or any application specific file format. Nor should it, for the only purpose of this simplified format is to allow the exchange of JPEG compressed images.*

Image files that employ JPEG compression are commonly called "JPEG files", and are stored in variants of the JIF image format. Most image capture devices (such as digital cameras) that output JPEG are actually creating files in the Exif format, the format that the camera industry has standardized on for metadata interchange. On the other hand, since the Exif standard does not allow color profiles, most image editing software stores JPEG in JFIF format, and also include the APP1 segment from the Exif file to include the metadata in an almost-compliant way; the JFIF standard is interpreted somewhat flexibly.[20]

Strictly speaking, the JFIF and Exif standards are incompatible because they each specify that their marker segment (APP0 or APP1, respectively) appears first. In practice, most JPEG files contain a JFIF marker segment that precedes the Exif header. This allows older readers to correctly handle the older format JFIF segment, while newer readers also decode the following Exif segment, being less strict about requiring it to appear first.

JPEG filename extensions

The most common filename extensions for files employing JPEG compression are **.jpg** and **.jpeg**, though .jpe, .jfif are also occasionally used.

Color profile

Many JPEG files embed an ICC color profile (color space). Commonly used color profiles include sRGB and Adobe RGB. Because these color spaces use a non-linear transformation, the dynamic range of an 8-bit JPEG file is about 11 stops; see gamma curve. some other files are png, tiff and gif. some other files are made up of pixels and colour (ʹΟε̋ῲ῝Εῄ is greek for JPEG0

Syntax and structure

A JPEG image consists of a sequence of *segments,* each beginning with a *marker*, each of which begins with a 0xFF byte followed by a byte indicating what kind of marker it is. Some markers consist of just those two bytes; others are followed by two bytes indicating the length of marker-specific payload data that follows. (The length includes the two bytes for the length, but not the two bytes for the marker.) Some markers are followed by entropy-coded data; the length of such a marker does not include the entropy-coded data. Note that consecutive 0xFF bytes are used as fill bytes for padding purposes, although this fill byte padding should only ever take place for markers immediately following entropy-coded scan data (see JPEG specification section B.1.1.2 and E.1.2 for details; specifically "In all cases where markers are appended after the compressed data, optional 0xFF fill bytes may precede the marker").

Within the entropy-coded data, after any 0xFF byte, a 0x00 byte is inserted by the encoder before the next byte, so that there does not appear to be a marker where none is intended, preventing framing errors. Decoders must skip this 0x00 byte. This technique, called *byte stuffing* (see JPEG specification section F.1.2.3), is only applied to the entropy-coded data, not to marker payload data. Note however that entropy-coded data has a few markers of its own; specifically the Reset markers (0xD0 through 0xD7), which are used to isolate independent chunks of entropy-coded data to allow parallel decoding, and encoders are free to insert these Reset markers at regular intervals (although not all encoders do this).

Common JPEG markers[21]

Short name	Bytes	Payload	Name	Comments
SOI	0xFF, 0xD8	*none*	Start Of Image	
SOF0	0xFF, 0xC0	*variable size*	Start Of Frame (Baseline DCT)	Indicates that this is a baseline DCT-based JPEG, and specifies the width, height, number of components, and component subsampling (e.g., 4:2:0).
SOF2	0xFF, 0xC2	*variable size*	Start Of Frame (Progressive DCT)	Indicates that this is a progressive DCT-based JPEG, and specifies the width, height, number of components, and component subsampling (e.g., 4:2:0).
DHT	0xFF, 0xC4	*variable size*	Define Huffman Table(s)	Specifies one or more Huffman tables.
DQT	0xFF, 0xDB	*variable size*	Define Quantization Table(s)	Specifies one or more quantization tables.

DRI	0xFF, 0xDD	2 bytes	Define Restart Interval	Specifies the interval between RST*n* markers, in macroblocks. This marker is followed by two bytes indicating the fixed size so it can be treated like any other variable size segment.
SOS	0xFF, 0xDA	*variable size*	Start Of Scan	Begins a top-to-bottom scan of the image. In baseline DCT JPEG images, there is generally a single scan. Progressive DCT JPEG images usually contain multiple scans. This marker specifies which slice of data it will contain, and is immediately followed by entropy-coded data.
RST*n*	0xFF, 0xD*n* (*n*=0..7)	*none*	Restart	Inserted every *r* macroblocks, where *r* is the restart interval set by a DRI marker. Not used if there was no DRI marker. The low 3 bits of the marker code cycle in value from 0 to 7.
APP*n*	0xFF, 0xE*n*	*variable size*	Application-specific	For example, an Exif JPEG file uses an APP1 marker to store metadata, laid out in a structure based closely on TIFF.
COM	0xFF, 0xFE	*variable size*	Comment	Contains a text comment.
EOI	0xFF, 0xD9	*none*	End Of Image	

There are other *Start Of Frame* markers that introduce other kinds of JPEG encodings.

Since several vendors might use the same APP*n* marker type, application-specific markers often begin with a standard or vendor name (e.g., "Exif" or "Adobe") or some other identifying string.

At a restart marker, block-to-block predictor variables are reset, and the bitstream is synchronized to a byte boundary. Restart markers provide means for recovery after bitstream error, such as transmission over an unreliable network or file corruption. Since the runs of macroblocks between restart markers may be independently decoded, these runs may be decoded in parallel.

JPEG codec example

Although a JPEG file can be encoded in various ways, most commonly it is done with JFIF encoding. The encoding process consists of several steps:

1. The representation of the colors in the image is converted from RGB to $Y'C_BC_R$, consisting of one luma component (Y'), representing brightness, and two chroma components, (C_B and C_R), representing color. This step is sometimes skipped.
2. The resolution of the chroma data is reduced, usually by a factor of 2. This reflects the fact that the eye is less sensitive to fine color details than to fine brightness details.
3. The image is split into blocks of 8×8 pixels, and for each block, each of the Y, C_B, and C_R data undergoes the Discrete Cosine Transform (DCT), which was developed in 1974 by N. Ahmed, T. Natarajan and K. R. Rao; see Reference 1 in discrete cosine transform. A DCT is similar to a Fourier transform in the sense that it produces a kind of spatial frequency spectrum.
4. The amplitudes of the frequency components are quantized. Human vision is much more sensitive to small variations in color or brightness over large areas than to the strength of high-frequency brightness variations. Therefore, the magnitudes of the high-frequency components are stored with a lower accuracy than the low-frequency components. The quality setting of the encoder (for example 50 or 95 on a scale of 0–100 in the Independent JPEG Group's library[22]) affects to what extent the resolution of each frequency component is reduced. If an excessively low quality setting is used, the high-frequency components are discarded altogether.
5. The resulting data for all 8×8 blocks is further compressed with a lossless algorithm, a variant of Huffman encoding.

The decoding process reverses these steps, except the *quantization* because it is irreversible. In the remainder of this section, the encoding and decoding processes are described in more detail.

Encoding

Many of the options in the JPEG standard are not commonly used, and as mentioned above, most image software uses the simpler JFIF format when creating a JPEG file, which among other things specifies the encoding method. Here is a brief description of one of the more common methods of encoding when applied to an input that has 24 bits per pixel (eight each of red, green, and blue). This particular option is a lossy data compression method.

Color space transformation

First, the image should be converted from RGB into a different color space called $Y'C_BC_R$ (or, informally, YCbCr). It has three components Y', C_B and C_R: the Y' component represents the brightness of a pixel, and the C_B and C_R components represent the chrominance (split into blue and red components).

Downsampling

Due to the densities of color- and brightness-sensitive receptors in the human eye, humans can see considerably more fine detail in the brightness of an image (the Y' component) than in the hue and color saturation of an image (the Cb and Cr components). Using this knowledge, encoders can be designed to compress images more efficiently.

The transformation into the $Y'C_BC_R$ color model enables the next usual step, which is to reduce the spatial resolution of the Cb and Cr components (called "downsampling" or "chroma subsampling"). The ratios at which the downsampling is ordinarily done for JPEG images are 4:4:4 (no downsampling), 4:2:2 (reduction by a factor of 2 in the horizontal direction), or (most commonly) 4:2:0 (reduction by a factor of 2 in both the horizontal and vertical directions). For the rest of the compression process, Y', Cb and Cr are processed separately and in a very similar manner.

Block splitting

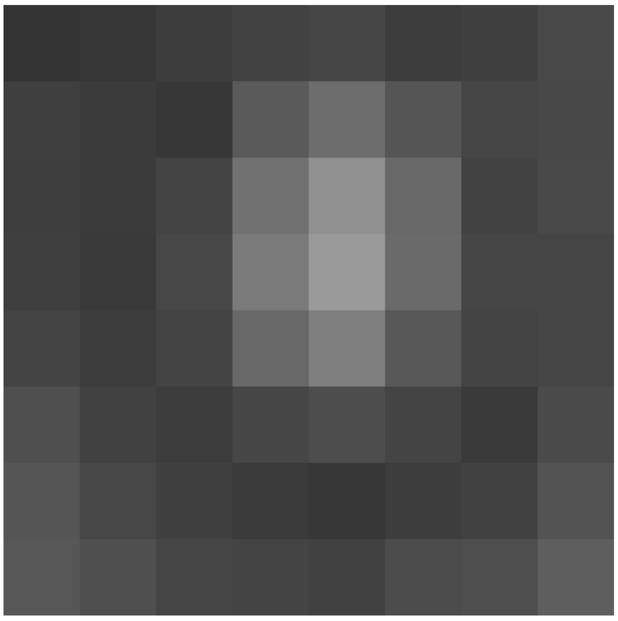
After subsampling, each channel must be split into 8×8 blocks. Depending on chroma subsampling, this yields (Minimum Coded Unit) MCU blocks of size 8×8 (4:4:4 – no subsampling), 16×8 (4:2:2), or most commonly 16×16 (4:2:0). In video compression MCUs are called macroblocks.

If the data for a channel does not represent an integer number of blocks then the encoder must fill the remaining area of the incomplete blocks with some form of dummy data. Filling the edges with a fixed color (for example, black) can create ringing artifacts along the visible part of the border; repeating the edge pixels is a common technique that reduces (but does not necessarily completely eliminate) such artifacts, and more sophisticated border filling techniques can also be applied.

Discrete cosine transform

Next, each 8×8 block of each component (Y, Cb, Cr) is converted to a frequency-domain representation, using a normalized, two-dimensional type-II discrete cosine transform (DCT), which was introduced by N. Ahmed, T. Natarajan and K. R. Rao in 1974; see Reference 1 in discrete cosine transform. The DCT is sometimes referred to as "type-II DCT" in the context of a family of transforms as in discrete cosine transform, and the corresponding inverse (IDCT) is denoted as "type-III DCT".

As an example, one such 8×8 8-bit subimage might be:

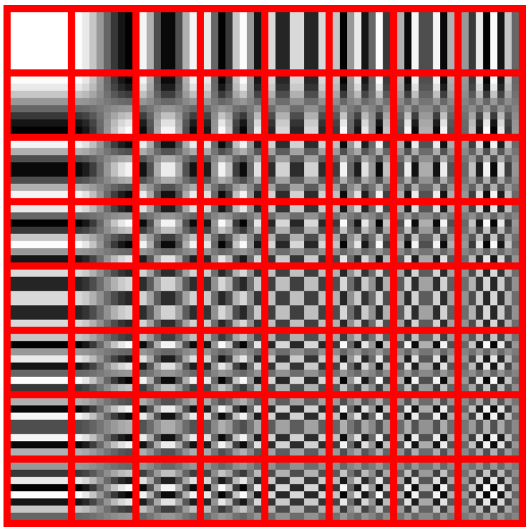

The 8×8 sub-image shown in 8-bit grayscale

$$\begin{bmatrix} 52 & 55 & 61 & 66 & 70 & 61 & 64 & 73 \\ 63 & 59 & 55 & 90 & 109 & 85 & 69 & 72 \\ 62 & 59 & 68 & 113 & 144 & 104 & 66 & 73 \\ 63 & 58 & 71 & 122 & 154 & 106 & 70 & 69 \\ 67 & 61 & 68 & 104 & 126 & 88 & 68 & 70 \\ 79 & 65 & 60 & 70 & 77 & 68 & 58 & 75 \\ 85 & 71 & 64 & 59 & 55 & 61 & 65 & 83 \\ 87 & 79 & 69 & 68 & 65 & 76 & 78 & 94 \end{bmatrix}.$$

Before computing the DCT of the 8×8 block, its values are shifted from a positive range to one centered around zero. For an 8-bit image, each entry in the original block falls in the range $[0, 255]$. The midpoint of the range (in this case, the value 128) is subtracted from each entry to produce a data range that is centered around zero, so that the modified range is $[-128, 127]$. This step reduces the dynamic range requirements in the DCT processing stage that follows. (Aside from the difference in dynamic range within the DCT stage, this step is mathematically equivalent to subtracting 1024 from the DC coefficient after performing the transform – which may be a better way to perform the operation on some architectures since it involves performing only one subtraction rather than 64 of them.)
This step results in the following values:

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

The next step is to take the two-dimensional DCT, which is given by:


The DCT transforms an 8×8 block of input values to a linear combination of these 64 patterns. The patterns are referred to as the two-dimensional DCT *basis functions*, and the output values are referred to as *transform coefficients*. The horizontal index is u and the vertical index is v .

$$G_{u,v} = \sum_{x=0}^{7} \sum_{y=0}^{7} \alpha(u)\alpha(v) g_{x,y} \cos\left[\frac{\pi}{8}\left(x+\frac{1}{2}\right)u\right] \cos\left[\frac{\pi}{8}\left(y+\frac{1}{2}\right)v\right]$$

where

- u is the horizontal spatial frequency, for the integers $0 \le u < 8$.
- v is the vertical spatial frequency, for the integers $0 \le v < 8$.
- $\alpha(u) = \begin{cases} \sqrt{\frac{1}{8}}, & \text{if } u = 0 \\ \sqrt{\frac{2}{8}}, & \text{otherwise} \end{cases}$ is a normalizing scale factor to make the transformation orthonormal
- $g_{x,y}$ is the pixel value at coordinates (x, y)
- $G_{u,v}$ is the DCT coefficient at coordinates (u, v).

If we perform this transformation on our matrix above, we get the following (rounded to the nearest two digits beyond the decimal point):

$$G = \begin{array}{cc} \overset{u}{\longrightarrow} & \\ \begin{bmatrix} -415.38 & -30.19 & -61.20 & 27.24 & 56.13 & -20.10 & -2.39 & 0.46 \\ 4.47 & -21.86 & -60.76 & 10.25 & 13.15 & -7.09 & -8.54 & 4.88 \\ -46.83 & 7.37 & 77.13 & -24.56 & -28.91 & 9.93 & 5.42 & -5.65 \\ -48.53 & 12.07 & 34.10 & -14.76 & -10.24 & 6.30 & 1.83 & 1.95 \\ 12.12 & -6.55 & -13.20 & -3.95 & -1.88 & 1.75 & -2.79 & 3.14 \\ -7.73 & 2.91 & 2.38 & -5.94 & -2.38 & 0.94 & 4.30 & 1.85 \\ -1.03 & 0.18 & 0.42 & -2.42 & -0.88 & -3.02 & 4.12 & -0.66 \\ -0.17 & 0.14 & -1.07 & -4.19 & -1.17 & -0.10 & 0.50 & 1.68 \end{bmatrix} & \Bigg\downarrow v. \end{array}$$

Note the top-left corner entry with the rather large magnitude. This is the DC coefficient. The remaining 63 coefficients are called the AC coefficients. The advantage of the DCT is its tendency to aggregate most of the signal in one corner of the result, as may be seen above. The quantization step to follow accentuates this effect while simultaneously reducing the overall size of the DCT coefficients, resulting in a signal that is easy to compress efficiently in the entropy stage.

The DCT temporarily increases the bit-depth of the data, since the DCT coefficients of an 8-bit/component image take up to 11 or more bits (depending on fidelity of the DCT calculation) to store. This may force the codec to temporarily use 16-bit bins to hold these coefficients, doubling the size of the image representation at this point; they are typically reduced back to 8-bit values by the quantization step. The temporary increase in size at this stage is not a performance concern for most JPEG implementations, because typically only a very small part of the image is stored in full DCT form at any given time during the image encoding or decoding process.

Quantization

The human eye is good at seeing small differences in brightness over a relatively large area, but not so good at distinguishing the exact strength of a high frequency brightness variation. This allows one to greatly reduce the amount of information in the high frequency components. This is done by simply dividing each component in the frequency domain by a constant for that component, and then rounding to the nearest integer. This rounding operation is the only lossy operation in the whole process (other than chroma subsampling) if the DCT computation is performed with sufficiently high precision. As a result of this, it is typically the case that many of the higher frequency components are rounded to zero, and many of the rest become small positive or negative numbers, which take many fewer bits to represent.

The elements in the quantization matrix control the compression ratio, with larger values producing greater compression. A typical quantization matrix, as specified in the original JPEG Standard, is as follows:

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

The quantized DCT coefficients are computed with

$$B_{j,k} = \mathrm{round}\left(\frac{G_{j,k}}{Q_{j,k}}\right) \text{ for } j = 0,1,2,\ldots,7; k = 0,1,2,\ldots,7$$

where G is the unquantized DCT coefficients; Q is the quantization matrix above; and B is the quantized DCT coefficients.

Using this quantization matrix with the DCT coefficient matrix from above results in:

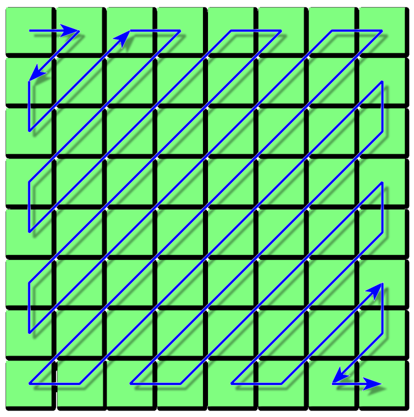
$$B = \begin{bmatrix} -26 & -3 & -6 & 2 & 2 & -1 & 0 & 0 \\ 0 & -2 & -4 & 1 & 1 & 0 & 0 & 0 \\ -3 & 1 & 5 & -1 & -1 & 0 & 0 & 0 \\ -3 & 1 & 2 & -1 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

For example, using −415 (the DC coefficient) and rounding to the nearest integer

$$\mathrm{round}\left(\frac{-415.38}{16}\right) = \mathrm{round}\left(-25.96\right) = -26.$$

Entropy coding

Entropy coding is a special form of lossless data compression. It involves arranging the image components in a "zigzag" order employing run-length encoding (RLE) algorithm that groups similar frequencies together, inserting length coding zeros, and then using Huffman coding on what is left.


Zigzag ordering of JPEG image components

The JPEG standard also allows, but does not require, decoders to support the use of arithmetic coding, which is mathematically superior to Huffman coding. However, this feature has rarely been used as it was historically covered by patents requiring royalty-bearing licenses, and because it is slower to encode and decode compared to Huffman coding. Arithmetic coding typically makes files about 5–7% smaller.

The previous quantized DC coefficient is used to predict the current quantized DC coefficient. The difference between the two is encoded rather than the actual value. The encoding of the 63 quantized AC coefficients does not use such prediction differencing.

The zigzag sequence for the above quantized coefficients are shown below. (The format shown is just for ease of understanding/viewing.)

−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							

If the i-th block is represented by Bi and positions within each block are represented by (p,q) where p = 0, 1, ..., 7 and q = 0, 1, ..., 7, then any coefficient in the DCT image can be represented as Bi(p,q). Thus, in the above scheme, the order of encoding pixels (for the i-th block) is Bi(0,0), Bi(0,1), Bi(1,0), Bi(2,0), Bi(1,1), Bi(0,2), Bi(0,3), Bi(1,2) and so on.

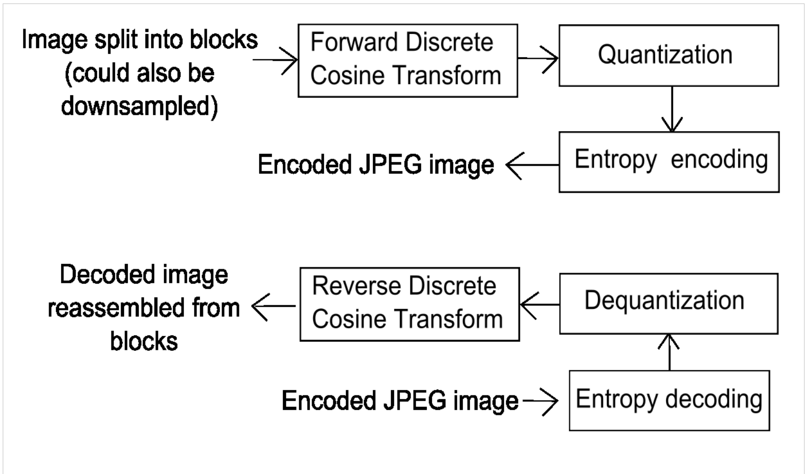


Image split into blocks (could also be downsampled)
Forward Discrete Cosine Transform
Quantization
Entropy encoding
Encoded JPEG image
Decoded image reassembled from blocks
Reverse Discrete Cosine Transform
Dequantization
Encoded JPEG image
Entropy decoding

Baseline sequential JPEG encoding and decoding processes

This encoding mode is called baseline *sequential* encoding. Baseline JPEG also supports *progressive* encoding. While sequential encoding encodes coefficients of a single block at a time (in a zigzag manner), progressive encoding encodes similar-positioned coefficients of all blocks in one go, followed by the next positioned coefficients of all blocks, and so on. So, if the image is divided into N 8×8 blocks {B0,B1,B2, ..., Bn-1}, then progressive encoding encodes Bi(0,0) for all blocks, i.e., for all i = 0, 1, 2, ..., N-1. This is followed by encoding Bi(0,1) coefficient of all blocks, followed by Bi(1,0)-th coefficient of all blocks, then Bi(2,0)-th coefficient of all blocks, and so on. It should be noted here that once all similar-positioned coefficients have been encoded, the next position to be encoded is the one occurring next in the zigzag traversal as indicated in the figure above. It has been found that Baseline Progressive JPEG encoding usually gives better compression as compared to Baseline Sequential JPEG due to the ability to use different Huffman tables (see below) tailored for different frequencies on each "scan" or "pass" (which includes similar-positioned coefficients), though the difference is not too large.

In the rest of the article, it is assumed that the coefficient pattern generated is due to sequential mode.

In order to encode the above generated coefficient pattern, JPEG uses Huffman encoding. JPEG has a special Huffman code word for ending the sequence prematurely when the remaining coefficients are zero.

Using this special code word: "EOB", the sequence becomes:

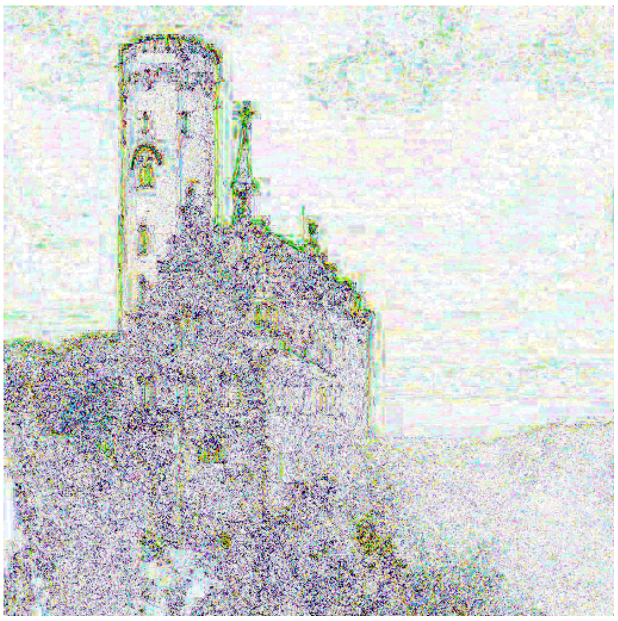
$$\begin{array}{rrrrrr}
-26 & & & & & \\
-3 & 0 & & & & \\
-3 & -2 & -6 & & & \\
2 & -4 & 1 & -3 & & \\
1 & 1 & 5 & 1 & 2 & \\
-1 & 1 & -1 & 2 & 0 & 0 \\
0 & 0 & 0 & -1 & -1 & \text{EOB}
\end{array}$$

JPEG's other code words represent combinations of (a) the number of significant bits of a coefficient, including sign, and (b) the number of consecutive zero coefficients that precede it. (Once you know how many bits to expect, it takes 1 bit to represent the choices {-1, +1}, 2 bits to represent the choices {-3, −2, +2, +3}, and so forth.) In our example block, most of the quantized coefficients are small numbers that are not preceded immediately by a zero coefficient. These more-frequent cases will be represented by shorter code words.

The JPEG standard provides general-purpose Huffman tables; encoders may also choose to generate Huffman tables optimized for the actual frequency distributions in images being encoded.

Compression ratio and artifacts

The resulting compression ratio can be varied according to need by being more or less aggressive in the divisors used in the quantization phase. Ten to one compression usually results in an image that cannot be distinguished by eye from the original. 100 to one compression is usually possible, but will look distinctly artifacted compared to the original. The appropriate level of compression depends on the use to which the image will be put.


This image shows the pixels that are different between a non-compressed image and the same image JPEG compressed with a quality setting of 50. Darker means a larger difference. Note especially the changes occurring near sharp edges and having a block-like shape.

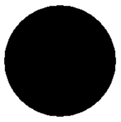
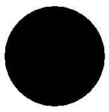
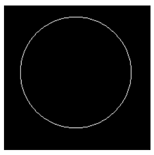
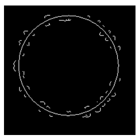
The compressed 8×8-squares are visible in the scaled up picture, together with other visual artifacts of the lossy compression.

External images

Illustration of edge busyness [23].Wikipedia:Link rot[24]

Those who use the World Wide Web may be familiar with the irregularities known as compression artifacts that appear in JPEG images, which may take the form of noise around contrasting edges (especially curves and corners), or 'blocky' images. These are due to the quantization step of the JPEG algorithm. They are especially noticeable around sharp corners between contrasting colors (text is a good example as it contains many such corners). The analogous artifacts in MPEG video are referred to as *mosquito noise,* as the resulting "edge busyness" and spurious dots, which change over time, resemble mosquitoes swarming around the object.[24][25]

These artifacts can be reduced by choosing a lower level of compression; they may be eliminated by saving an image using a lossless file format, though for photographic images this will usually result in a larger file size. The images created with ray-tracing programs have noticeable blocky shapes on the terrain. Certain low-intensity compression artifacts might be acceptable when simply viewing the images, but can be emphasized if the image is subsequently processed, usually resulting in unacceptable quality. Consider the example below, demonstrating the effect of lossy compression on an edge detection processing step.

Image	Lossless compression	Lossy compression
Original		
Processed by Canny edge detector		

Some programs allow the user to vary the amount by which individual blocks are compressed. Stronger compression is applied to areas of the image that show fewer artifacts. This way it is possible to manually reduce JPEG file size with less loss of quality.

JPEG artifacts, like pixelation, are occasionally intentionally exploited for artistic purposes, as in *Jpegs,* by German photographer Thomas Ruff.[26][27]

Since the quantization stage *always* results in a loss of information, JPEG standard is always a lossy compression codec. (Information is lost both in quantizing and rounding of the floating-point numbers.) Even if the quantization matrix is a matrix of ones, information will still be lost in the rounding step.

Decoding

Decoding to display the image consists of doing all the above in reverse.

Taking the DCT coefficient matrix (after adding the difference of the DC coefficient back in)

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

and taking the entry-for-entry product with the quantization matrix from above results in

$$\begin{bmatrix} -416 & -33 & -60 & 32 & 48 & -40 & 0 & 0 \\ 0 & -24 & -56 & 19 & 26 & 0 & 0 & 0 \\ -42 & 13 & 80 & -24 & -40 & 0 & 0 & 0 \\ -42 & 17 & 44 & -29 & 0 & 0 & 0 & 0 \\ 18 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

which closely resembles the original DCT coefficient matrix for the top-left portion.

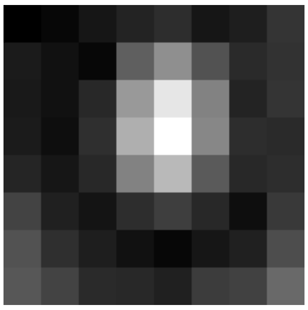
The next step is to take the two-dimensional inverse DCT (a 2D type-III DCT), which is given by:

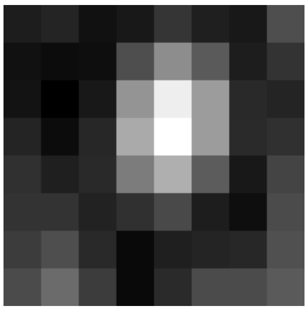
$$f_{x,y} = \sum_{u=0}^{7}\sum_{v=0}^{7} \alpha(u)\alpha(v)F_{u,v} \cos\left[\frac{\pi}{8}\left(x+\frac{1}{2}\right)u\right] \cos\left[\frac{\pi}{8}\left(y+\frac{1}{2}\right)v\right]$$

where

- x is the pixel row, for the integers $0 \leq x < 8$.
- y is the pixel column, for the integers $0 \leq y < 8$.
- $\alpha(u)$ is defined as above, for the integers $0 \leq u < 8$.
- $F_{u,v}$ is the reconstructed approximate coefficient at coordinates (u, v).
- $f_{x,y}$ is the reconstructed pixel value at coordinates (x, y)

Rounding the output to integer values (since the original had integer values) results in an image with values (still shifted down by 128)





Notice the slight differences between the original (top) and decompressed image (bottom), which is most readily seen in the bottom-left corner.

$$\begin{bmatrix} -66 & -63 & -71 & -68 & -56 & -65 & -68 & -46 \\ -71 & -73 & -72 & -46 & -20 & -41 & -66 & -57 \\ -70 & -78 & -68 & -17 & 20 & -14 & -61 & -63 \\ -63 & -73 & -62 & -8 & 27 & -14 & -60 & -58 \\ -58 & -65 & -61 & -27 & -6 & -40 & -68 & -50 \\ -57 & -57 & -64 & -58 & -48 & -66 & -72 & -47 \\ -53 & -46 & -61 & -74 & -65 & -63 & -61 & -45 \\ -47 & -34 & -53 & -74 & -60 & -47 & -47 & -41 \end{bmatrix}$$

and adding 128 to each entry

$$\begin{bmatrix} 62 & 65 & 57 & 60 & 72 & 63 & 60 & 82 \\ 57 & 55 & 56 & 82 & 108 & 87 & 62 & 71 \\ 58 & 50 & 60 & 111 & 148 & 114 & 67 & 64 \\ 65 & 55 & 66 & 120 & 155 & 114 & 68 & 70 \\ 70 & 63 & 67 & 101 & 122 & 88 & 60 & 78 \\ 71 & 71 & 64 & 70 & 80 & 62 & 56 & 81 \\ 75 & 82 & 67 & 54 & 63 & 65 & 66 & 83 \\ 81 & 95 & 75 & 54 & 68 & 81 & 81 & 87 \end{bmatrix}.$$

This is the decompressed subimage. In general, the decompression process may produce values outside of the original input range of $[0, 255]$. If this occurs, the decoder needs to clip the output values keep them within that range to prevent overflow when storing the decompressed image with the original bit depth.

The decompressed subimage can be compared to the original subimage (also see images to the right) by taking the difference (original − uncompressed) results in the following error values:

$$\begin{bmatrix} -10 & -10 & 4 & 6 & 2 & 2 & 4 & -9 \\ 6 & 4 & -1 & 8 & 1 & -2 & 7 & 1 \\ 4 & 9 & 8 & 2 & -4 & -10 & -1 & 8 \\ -2 & 3 & 5 & 2 & -1 & -8 & 2 & -1 \\ -3 & -2 & 1 & 3 & 4 & 0 & 8 & -8 \\ 8 & -6 & -4 & 0 & -3 & 6 & 2 & -6 \\ 10 & -11 & -3 & 5 & -8 & -4 & -1 & 0 \\ 6 & -15 & -6 & 14 & -3 & -5 & -3 & 7 \end{bmatrix}$$

with an average absolute error of about 5 values per pixels (i.e., $\frac{1}{64}\sum_{x=0}^{7}\sum_{y=0}^{7}|e(x,y)| = 4.875$).

The error is most noticeable in the bottom-left corner where the bottom-left pixel becomes darker than the pixel to its immediate right.

Required precision

The encoding description in the JPEG standard does not fix the precision needed for the output compressed image. However, the JPEG standard (and the similar MPEG standards) includes some precision requirements for the *de*coding, including all parts of the decoding process (variable length decoding, inverse DCT, dequantization, renormalization of outputs); the output from the reference algorithm must not exceed:

- a maximum 1 bit of difference for each pixel component
- low mean square error over each 8×8-pixel block
- very low mean error over each 8×8-pixel block
- very low mean square error over the whole image
- extremely low mean error over the whole image

These assertions are tested on a large set of randomized input images, to handle the worst cases. The former IEEE 1180–1990 standard contained some similar precision requirements. The precision has a consequence on the implementation of decoders, and it is critical because some encoding processes (notably used for encoding sequences of images like MPEG) need to be able to construct, on the encoder side, a reference decoded image. In order to support 8-bit precision per pixel component output, dequantization and inverse DCT transforms are typically implemented with at least 14-bit precision in optimized decoders.

Effects of JPEG compression

JPEG compression artifacts blend well into photographs with detailed non-uniform textures, allowing higher compression ratios. Notice how a higher compression ratio first affects the high-frequency textures in the upper-left corner of the image, and how the contrasting lines become more fuzzy. The very high compression ratio severely affects the quality of the image, although the overall colors and image form are still recognizable. However, the precision of colors suffer less (for a human eye) than the precision of contours (based on luminance). This justifies the fact that images should be first transformed in a color model separating the luminance from the chromatic information, before subsampling the chromatic planes (which may also use lower quality quantization) in order to preserve the precision of the luminance plane with more information bits.

Sample photographs

For information, the uncompressed 24-bit RGB bitmap image below (73,242 pixels) would require 219,726 bytes (excluding all other information headers). The filesizes indicated below include the internal JPEG information headers and some meta-data. For highest quality images (Q=100), about 8.25 bits per color pixel is required. On grayscale images, a minimum of 6.5 bits per pixel is enough (a comparable Q=100 quality color information requires about 25% more encoded bits). The highest quality image below (Q=100) is encoded at 9 bits per color pixel, the medium quality image (Q=25) uses 1 bit per color pixel. For most applications, the quality factor should not go below 0.75 bit per pixel (Q=12.5), as demonstrated by the low quality image. The image at lowest quality uses only 0.13 bit per pixel, and displays very poor color. This is useful when the image will be displayed in a significantly scaled down size.

Note: The above images are not IEEE / CCIR / EBU test images, and the encoder settings are not specified or available.

Image	Quality	Size (bytes)	Compression ratio	Comment
	Highest quality (Q = 100)	83,261	2.6:1	Extremely minor artifacts
	High quality (Q = 50)	15,138	15:1	Initial signs of subimage artifacts
	Medium quality (Q = 25)	9,553	23:1	Stronger artifacts; loss of high frequency information
	Low quality (Q = 10)	4,787	46:1	Severe high frequency loss; artifacts on subimage boundaries ("macroblocking") are obvious
	Lowest quality (Q = 1)	1,523	144:1	Extreme loss of color and detail; the leaves are nearly unrecognizable

The medium quality photo uses only 4.3% of the storage space required for the uncompressed image, but has little noticeable loss of detail or visible artifacts. However, once a certain threshold of compression is passed, compressed images show increasingly visible defects. See the article on rate distortion theory for a mathematical explanation of this threshold effect. A particular limitation of JPEG in this regard is its non-overlapped 8×8 block transform structure. More modern designs such as JPEG 2000 and JPEG XR exhibit a more graceful degradation of quality as the bit usage decreases – by using transforms with a larger spatial extent for the lower frequency coefficients and by using overlapping transform basis functions.

Lossless further compression

From 2004 to 2008, new research has emerged on ways to further compress the data contained in JPEG images without modifying the represented image.[28][29][30][31] This has applications in scenarios where the original image is only available in JPEG format, and its size needs to be reduced for archival or transmission. Standard general-purpose compression tools cannot significantly compress JPEG files.

Typically, such schemes take advantage of improvements to the naive scheme for coding DCT coefficients, which fails to take into account:

- Correlations between magnitudes of adjacent coefficients in the same block;
- Correlations between magnitudes of the same coefficient in adjacent blocks;
- Correlations between magnitudes of the same coefficient/block in different channels;
- The DC coefficients when taken together resemble a downscale version of the original image multiplied by a scaling factor. Well-known schemes for lossless coding of continuous-tone images can be applied, achieving somewhat better compression than the Huffman coded DPCM used in JPEG.

Some standard but rarely used options already exist in JPEG to improve the efficiency of coding DCT coefficients: the arithmetic coding option, and the progressive coding option (which produces lower bitrates because values for each coefficient are coded independently, and each coefficient has a significantly different distribution). Modern methods have improved on these techniques by reordering coefficients to group coefficients of larger magnitude together;[28] using adjacent coefficients and blocks to predict new coefficient values;[30] dividing blocks or coefficients up among a small number of independently coded models based on their statistics and adjacent values;[29][30] and most recently, by decoding blocks, predicting subsequent blocks in the spatial domain, and then encoding these to generate predictions for DCT coefficients.[31]

Typically, such methods can compress existing JPEG files between 15 and 25 percent, and for JPEGs compressed at low-quality settings, can produce improvements of up to 65%.[30][31]

A freely available tool called packJPG[32] is based on the 2007 paper "Improved Redundancy Reduction for JPEG Files."

Derived formats for stereoscopic 3D

JPEG Stereoscopic

JPEG Stereoscopic (JPS, extension .jps) is a JPEG-based format for stereoscopic images.[33][34] It has a range of configurations stored in the JPEG APP3 marker field, but usually contains one image of double width, representing two images of identical size in cross-eyed (i.e. left frame on the right half of the image and vice versa) side-by-side arrangement. This file format can be viewed as a JPEG without any special software, or can be processed for rendering in other modes.

An example of a stereoscopic .JPS file

JPEG Multi-Picture Format

JPEG Multi-Picture Format (MPO, extension .mpo) is a JPEG-based format for multi-view images. It contains two or more JPEG files concatenated together.[35][36] There are also special EXIF fields describing its purpose. This is used by the Fujifilm FinePix Real 3D W1 camera, Panasonic Lumix DMC-TZ20, DMC-TZ30 & DMC-TS4 (FT4), Sony DSC-HX7V, HTC Evo 3D, the JVC GY-HMZ1U AVCHD/MVC extension camcorder and by the Nintendo 3DS for its 3D Camera.

Patent issues

In 2002, Forgent Networks asserted that it owned and would enforce patent rights on the JPEG technology, arising from a patent that had been filed on October 27, 1986, and granted on October 6, 1987 (U.S. Patent 4,698,672 [37]). The announcement created a furor reminiscent of Unisys' attempts to assert its rights over the GIF image compression standard.

The JPEG committee investigated the patent claims in 2002 and were of the opinion that they were invalidated by prior art.[38] Others also concluded that Forgent did not have a patent that covered JPEG.[39] Nevertheless, between 2002 and 2004 Forgent was able to obtain about US$105 million by licensing their patent to some 30 companies. In April 2004, Forgent sued 31 other companies to enforce further license payments. In July of the same year, a consortium of 21 large computer companies filed a countersuit, with the goal of invalidating the patent. In addition, Microsoft launched a separate lawsuit against Forgent in April 2005.[40] In February 2006, the United States Patent and Trademark Office agreed to re-examine Forgent's JPEG patent at the request of the Public Patent Foundation.[] On May 26, 2006 the USPTO found the patent invalid based on prior art. The USPTO also found that Forgent knew about the prior art, and did not tell the Patent Office, making any appeal to reinstate the patent highly unlikely to succeed.[41]

Forgent also possesses a similar patent granted by the European Patent Office in 1994, though it is unclear how enforceable it is.[]

As of October 27, 2006, the U.S. patent's 20-year term appears to have expired, and in November 2006, Forgent agreed to abandon enforcement of patent claims against use of the JPEG standard.[]

The JPEG committee has as one of its explicit goals that their standards (in particular their baseline methods) be implementable without payment of license fees, and they have secured appropriate license rights for their upcoming JPEG 2000 standard from over 20 large organizations.

Beginning in August 2007, another company, Global Patent Holdings, LLC claimed that its patent (U.S. Patent 5,253,341 [42]) issued in 1993, is infringed by the downloading of JPEG images on either a website or through e-mail. If not invalidated, this patent could apply to any website that displays JPEG images. The patent emergedWikipedia:Please clarify in July 2007 following a seven-year reexamination by the U.S. Patent and Trademark Office in which all of the original claims of the patent were revoked, but an additional claim (claim 17) was confirmed.[43]

In its first two lawsuits following the reexamination, both filed in Chicago, Illinois, Global Patent Holdings sued the Green Bay Packers, CDW, Motorola, Apple, Orbitz, Officemax, Caterpillar, Kraft and Peapod as defendants. A third lawsuit was filed on December 5, 2007 in South Florida against ADT Security Services, AutoNation, Florida Crystals Corp., HearUSA, MovieTickets.com, Ocwen Financial Corp. and Tire Kingdom, and a fourth lawsuit on January 8, 2008 in South Florida against the Boca Raton Resort & Club. A fifth lawsuit was filed against Global Patent Holdings in Nevada. That lawsuit was filed by Zappos.com, Inc., which was allegedly threatened by Global Patent Holdings, and seeks a judicial declaration that the '341 patent is invalid and not infringed.

Global Patent Holdings had also used the '341 patent to sue or threaten outspoken critics of broad software patents, including Gregory Aharonian[] and the anonymous operator of a website blog known as the "Patent Troll Tracker."[] On December 21, 2007, patent lawyer Vernon Francissen of Chicago asked the U.S. Patent and Trademark Office to reexamine the sole remaining claim of the '341 patent on the basis of new prior art.[]

On March 5, 2008, the U.S. Patent and Trademark Office agreed to reexamine the '341 patent, finding that the new prior art raised substantial new questions regarding the patent's validity.[44] In light of the reexamination, the accused infringers in four of the five pending lawsuits have filed motions to suspend (stay) their cases until completion of the U.S. Patent and Trademark Office's review of the '341 patent. On April 23, 2008, a judge presiding over the two lawsuits in Chicago, Illinois granted the motions in those cases.[] On July 22, 2008, the Patent Office issued the first "Office Action" of the second reexamination, finding the claim invalid based on nineteen separate grounds.[] On Nov. 24, 2009, a Reexamination Certificate was issued cancelling all claims.

Standards

Here are some examples of standards created by ISO/IEC JTC1 SC29 Working Group 1 (WG 1), which includes the Joint Photographic Experts Group and Joint Bi-level Image experts Group:

- JPEG (lossy and lossless): ITU-T T.81, ISO/IEC 10918-1
- JPEG extensions: ITU-T T.84
- JPEG-LS (lossless, improved): ITU-T T.87, ISO/IEC 14495-1
- JBIG (lossless, bi-level pictures, fax): ITU-T T.82, ISO/IEC 11544
- JBIG2 (bi-level pictures): ITU-T T.88, ISO/IEC 14492
- JPEG 2000: ITU-T T.800, ISO/IEC 15444-1
- JPEG 2000 extensions: ITU-T T.801
- JPEG XR (formerly called HD Photo prior to standardization) : ITU-T T.832, ISO/IEC 29199-2

References

[1] MIME Type Detection in Internet Explorer: Uploaded MIME Types (http://msdn.microsoft.com/en-us/library/ms775147(v=vs.85).aspx#_replace) (msdn.microsoft.com)

[2] JPEG File Layout and Format (http://www.jpeg.org/public/jfif.pdf)

[7] http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=18902

[8] http://www.itu.int/rec/T-REC-T.81

[9] http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=20689

[10] http://www.itu.int/rec/T-REC-T.83

[11] http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=25037

[12] http://www.itu.int/rec/T-REC-T.84

[13] http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=25431

[14] http://www.itu.int/rec/T-REC-T.86

[15] http://www.iso.org/iso/iso_catalogue/catalogue_tc/catalogue_detail.htm?csnumber=54989

[16] http://www.itu.int/rec/T-REC-T.871

[20] (q. 14: "Why all the argument about file formats?")

[23] http://i.cmpnet.com/videsignline/2006/02/algolith-fig2.jpg

[24] Phuc-Tue Le Dinh and Jacques Patry. Video compression artifacts and MPEG noise reduction (http://www.videsignline.com/howto/180207350). Video Imaging DesignLine. February 24, 2006. Retrieved May 28, 2009.

[25] "**3.9 mosquito noise:** Form of edge busyness distortion sometimes associated with movement, characterized by moving artifacts and/or blotchy noise patterns superimposed over the objects (resembling a mosquito flying around a person's head and shoulders)." ITU-T Rec. P.930 (08/96) Principles of a reference impairment system for video (http://eu.sabotage.org/www/ITU/P/P0930e.pdf)

[26] *jpegs,* Thomas Ruff, *Aperture,* May 31, 2009, 132 pp., ISBN 978-1-59711-093-8

[27] Review: jpegs by Thomas Ruff (http://jmcolberg.com/weblog/2009/04/review_jpegs_by_thomas_ruff/), by Joerg Colberg, Apr 17, 2009

[28] I. Bauermann and E. Steinbacj. Further Lossless Compression of JPEG Images. Proc. of Picture Coding Symposium (PCS 2004), San Francisco, USA, December 15–17, 2004.

[29] N. Ponomarenko, K. Egiazarian, V. Lukin and J. Astola. Additional Lossless Compression of JPEG Images, Proc. of the 4th Intl. Symposium on Image and Signal Processing and Analysis (ISPA 2005), Zagreb, Croatia, pp.117–120, September 15–17, 2005.

[30] M. Stirner and G. Seelmann. Improved Redundancy Reduction for JPEG Files. Proc. of Picture Coding Symposium (PCS 2007), Lisbon, Portugal, November 7–9, 2007

[31] Ichiro Matsuda, Yukio Nomoto, Kei Wakabayashi and Susumu Itoh. Lossless Re-encoding of JPEG images using block-adaptive intra prediction. Proceedings of the 16th European Signal Processing Conference (EUSIPCO 2008).

[33] J. Siragusa, D. C. Swift, "General Purpose Stereoscopic Data Descriptor" (http://vrex.com/developer/sterdesc.pdf), VRex, Inc., Elmsford, New York, USA, 1997.

[34] Tim Kemp, JPS files (http://ephehm.com/jps/)

[37] http://www.google.com/patents?vid=4698672

[39] JPEG and JPEG2000 – Between Patent Quarrel and Change of Technology (http://web.archive.org/web/20040817154508/www.algovision-luratech.com/company/news/patentquarrel.jsp?OnlineShopId=164241031081525276) (Archive)

[42] http://www.google.com/patents?vid=5253341

[43] Ex Parte Reexamination Certificate for U.S. Patent No. 5,253,341 (http://www.uspto.gov/web/patents/patog/week30/OG/html/1320-4/US05253341-20070724.html)

[44] U.S. Patent Office – Granting Reexamination on 5,253,341 C1

External links

- JPEG Standard (JPEG ISO/IEC 10918-1 ITU-T Recommendation T.81) (http://www.w3.org/Graphics/JPEG/itu-t81.pdf) at W3.org
- Official Joint Photographic Experts Group site (http://www.jpeg.org/)
- JFIF File Format (http://www.w3.org/Graphics/JPEG/jfif3.pdf) at W3.org
- JPEG viewer in 250 lines of easy to understand python code (http://code.google.com/p/micro-jpeg-visualizer/)
- Wotsit.org's entry (http://www.wotsit.org/list.asp?page=3&fc=1&search=&al=) on the JPEG format
- Example images over the full range of quantization levels from 1 to 100 (http://www.visengi.com/products/jpeg_hardware_encoder) at visengi.com
- Public domain JPEG compressor in a single C++ source file, along with a matching decompressor (http://code.google.com/p/jpeg-compressor/) at code.google.com
- Example of .JPG file decoding (http://l1032265.myweb.hinet.net/huffman.htm)
- Jpeg Decoder Open Source Code , Copyright (C) 1995–1997, Thomas G. Lane. (http://opensource.apple.com/source/WebCore/WebCore-1C25/platform/image-decoders/jpeg/)
- JPEG compression and decompression on GPU. (http://developer.download.nvidia.com/GTC/PDF/GTC2012/PresentationPDF/S0273-GTC2012-JPEG-Coding-GPU.pdf)

Article Sources and Contributors

JPEG *Source*: http://en.wikipedia.org/w/index.php?oldid=548313797 *Contributors*: (, *drew, 06109599, 130.225.29.xxx, 15.253, 19.154, 1exec1, 2404:2000:2000:5:0:0:0:C2, 2help, 3-5 file, 84user, A bit iffy, ATLBeer, AVRS, AbJ32, Adoniscik, Aeons, Aeusoes1, Ahoerstemeier, Aitias, Akhristov, Alansohn, Ale jrb, Alejo2083, Alex, AlexMld, Alexwagner, Allen3, Almwi, AlphaPyro, Alphachimp, Alphax, Alsandro, AltiusBimm, Amarrajsingh, Amit man, Anakin101, Andewulfe, Andrew-916, AndrewWTaylor, Andrewrp, Anow2, Antandrus, Antialias, Arisa, Arthena, Astronautics, Atcold, Athaba, Atl123, Audiodude, Audriusa, AxelBoldt, AzaToth, B4hand, Badgernet, Balko Kabo, Bdesham, Bdijkstra, Beetstra, Ben-Zin, BenFrantzDale, Bender235, Benhoyt, Benstown, Berland, Biker JR, Bkell, Blue520, Boardersparadise, Bobblehead, Bomazi, Boomshadow, Brandmeister (old), Brianski, Brownsteve, Bruce89, Bruzie, Bryan986, C0nanPayne, Ca michelbach, Calbaer, CanadianLinuxUser, Cancan101, Carminox, Caspar Cedro, Causa sui, Cburnett, Cgrenier, Charles Gaudette, CharlieEchoTango, Chasingsol, Chocoforfriends, Chowbok, Chris G, ChrisRuvolo, Clivestaples, Cmglee, CoJaBo, Compact disk, Conversion script, Cortega, Crissov, Crotalus horridus, CyberSkull, Cybercobra, D, DARTH SIDIOUS 2, DGaw, DVdm, Dakusan, Damian Yerrick, Daniel Brockman, Daniel Mietchen, Daniel.Cardenas, Danski14, David Crawshaw, David1217, Davidolivan, Dbenbenn, Dcoetzee, DeadEyeArrow, Dee Earley, Deflective, Deineka, Deityguns, Demkop, Derek Ross, Dicklyon, Dispenser, DmitriyV, DocWatson42, Dogru144, Donarreiskoffer, Dream out loud, Dschwen, Dtgriscom, E23, Ed Cormany, Ed g2s, Edward, Edwinstearns, Elassint, Electron9, Eleos, Elliotbay, Epbr123, Epson291, Escape Orbit, Eugen Dedu, Euryalus, Eyreland, Fabrictramp, Falcon8765, FalseAlarm, Favonian, Fbrazill, FelixH, Feraudyh, Fernsalan, Filemon, Fir0002, Fleminra, Foxandpotatoes, Francs2000, Frap, Freakofnurture, Frecklefoot, Fru1tbat, Funkybearman, Furrykef, Fyodorser, GDonato, Gaelen S., Gaius Cornelius, Gareth Griffith-Jones, Garrett Albright, Generalnat2, GeorgeMoney, Geyserit, Ghclark, Ghettoblaster, Giftlite, GilbertoSilvaFan, Glen, Glenn, Gnp, Gobbleswoggler, Gogo Dodo, Goldom, GorillaWarfare, Gpvos, Grafen, Graham101, Graham87, Grammaticus Repairo, Graue, Green caterpillar, Greypyjamas2, Groogle, Gurch, Guy Harris, Guy M, Gwernol, H, H2g2bob, HAl, Haakon, Hankwang, Hannes Hirzel, HannesP, Harold f, Henry.guillotine, Heron, Hhielscher, Hinotori, Hussainsab100, Huw Powell, Ian Fieggen, Ilmari Karonen, Imroy, Inkling, Interiot, Ish rishabh, J.delanoy, JForget, JNW, Ja 62, Jakub Vrána, JamesGeddes, JamesHoadley, Jamesedwardlong, Jan1nad, Janke, Jason, Jclemens, Jcoy, Jedibob5, Jeffrey Mall, Jerome Charles Potts, Jesse Viviano, Jmcnamera, Jmgonzalez, Jncraton, JoanneB, Joe4440, Joesehpwilliam, Jogfalls1947, John of Reading, Jshadias, Justanemokid, Justme89, KaiSeun, Kaldari, Kangdang, Karam.Anthony.K, Kaszeta, Kauai68, Kbdank71, Kbh3rd, Kcacciatore, Ke4roh, Khazar2, Khukri, Kimse, Kitsurenda, Kjoonlee, Klingoncowboy4, KnowledgeOfSelf, Koavf, Konstable, Kostiq, Kostmo, Kozuch, Kriegaffe, Ksheka, Kurykh, Kwamikagami, Kzl.zawlin, L'Aquatique, LOL, Larrymcp, Lawrence Cohen, Lawrencegold, LeaveSleaves, Legocool, LightStarch, Lilllian76, Lisbk, LittleOldMe, Llort, Llull, Lmcgign, Lmitchell6, Loggie, Lonelyprogrammer264, Longhair, Lonwolve, Loopylo2000, Lotje, Lunagron, Lupo, M4gnum0n, MER-C, MK8, Mamuf, Mandarax, Mani1, Manning james, Marc Mongenet, MarkSweep, Markus Kuhn, Martarius, Mathieu, Mattbrundage, Max theprintspace, Mayank geek, McGeddon, Mchavez, Mean as custard, Meisam, Meters, Michael A. White, Michael Hardy, Michael Snow, Mifter, Mikay, Mike Rosoft, MikeLynch, Milkcrate, Minghong, Miracle Pen, Mjb, Mk dexter, Mnemo, Mormegil, MrOllie, MrZoolook, Mschel, Mugunth Kumar, Mulligatawny, Mwilso24, Myanw, NAHID, Nanshu, NawlinWiki, Nbarth, Ncmvocalist, Ndenison, Neilc, Neo139, NewEnglandYankee, Newone, Nfvr, NickW557, Nicoontheweb, Nikpapag, Nimur, Nivix, Nobar, Noname58, Notinasnaid, NuclearWarfare, Nuujinn, Nxavar, Nyq, Oli Filth, Omicronpersei8, Onewhohelps, Ornicks, OverlordQ, Oxymoron83, Ozhiker, PAR, Pabouk, Palashrijan, Paul Heckbert, Pawnbroker, Pedantic of Purley, Pegase, Perey, Peter M Dodge, Phillipsacp, PhnomPencil, Photonique, Piet Delport, Pinethicket, Pitzik4, Plasticup, Plugwash, Porges, ProhibitOnions, Prunesqualer, Psychonaut, Quibik, Qutezuce, RG2, Rajb245, RayKiddy, Reelrt, Reim, Requestion, RexNL, Rhsimard, Rich Farmbrough, Richgel999, Rick Sidwell, Rickyrazz, Rixs, Rjwilmsi, Rl, Rob Hooft, RobinK, Rocastelo, Roscoe x, Rsrikanth05, Rune.welsh, RupertMillard, Rursus, STrRedWolf, Samboy, Sanao, SarahPalmerson, Sburke, Seandylanw, Seaphoto, Shadowjams, Shanes, ShaunMacPherson, Shoessss, Shotgunlee, Skabraham, Skater, Slady, Slashme, Sleske, Sloman, Slowking Man, Smjg, Smrti18, Soumyasch, SpaceFlight89, Spe88, Speck-Made, SpeedyGonsales, Ssd, Stephenb, SteveSims, Stevenj, Steveprutz, Stolsvik, Stone, Strait, Stuartyeates, Superdood, Superm401, Syp, TCMike, TRosenbaum, Tamariki, Tamarkot, Tardis5923, Tarquin, Techdawg667, Ted Longstaffe, TeemuN, Tendim, The Anome, The Thing That Should Not Be, TigerShark, Timbo76, Timeshifter, Timotab, Timothy57, Timwi, Tjb777, Tohd8BohaithuGh1, Tom k&e, Toybuilder, Toytoy, Tpbradbury, Trademarx, Tree Kittens, Tristaess, Trusilver, Tuxick, Twas Now, UU, Viory, Vuo, WODUP, WP Randomno, WWC, WadeSimMiser, Waggers, Wanion, Warren, Wavelength, West.andrew.g, WhatamIdoing, Wikiuser100, Willhsmit, William Avery, Wimt, WookieInHeat, Wookipedian, Wysprgr2005, Xboxmaster17, Xcrivener, XenonofArcticus, Xompanthy, Xorx77, Xpclient, Yangez, Yevgeniwebmaster, Yoderj, Yswismer, Zardragon1, ZeroOne, Zfr, Zundark, Zzuuzz, Ὁ οἶστρος, 927 anonymous edits

Image Sources, Licenses and Contributors

File:Felis silvestris silvestris small gradual decrease of quality.png *Source*: http://en.wikipedia.org/w/index.php?title=File:Felis_silvestris_silvestris_small_gradual_decrease_of_quality.png *License*: Creative Commons Attribution 3.0 *Contributors*: User:AzaToth

File:Continuously varied JPEG compression for an abdominal CT scan - 1471-2342-12-24-S1.ogv *Source*: http://en.wikipedia.org/w/index.php?title=File:Continuously_varied_JPEG_compression_for_an_abdominal_CT_scan_-_1471-2342-12-24-S1.ogv *License*: Creative Commons Attribution 2.0 *Contributors*: Flint A

File:JPEG example subimage.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:JPEG_example_subimage.svg *License*: Creative Commons Attribution-ShareAlike 3.0 Unported *Contributors*: en:User:Cburnett

File:Dctjpeg.png *Source*: http://en.wikipedia.org/w/index.php?title=File:Dctjpeg.png *License*: Public Domain *Contributors*: AnonMoos, Devcore, FelixH, Foroa, Stephantom, WikipediaMaster, 1 anonymous edits

File:JPEG ZigZag.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:JPEG_ZigZag.svg *License*: Public Domain *Contributors*: Alex Khristov

File:JPEG process.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:JPEG_process.svg *License*: Public Domain *Contributors*: User:Konstable

File:Lichtenstein jpeg difference.png *Source*: http://en.wikipedia.org/w/index.php?title=File:Lichtenstein_jpeg_difference.png *License*: Creative Commons Attribution-ShareAlike 3.0 Unported *Contributors*: Alessio Damato

File:Jpegvergroessert.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:Jpegvergroessert.jpg *License*: GNU Free Documentation License *Contributors*: User:FelixH

File:Nuvola apps kview.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:Nuvola_apps_kview.svg *License*: unknown *Contributors*: Ch1902, Saibo

file:Searchtool.svg *Source*: http://en.wikipedia.org/w/index.php?title=File:Searchtool.svg *License*: GNU Lesser General Public License *Contributors*: Anomie

File:Lossless-circle.png *Source*: http://en.wikipedia.org/w/index.php?title=File:Lossless-circle.png *License*: Public Domain *Contributors*: Meisam

File:Lossy-circle.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:Lossy-circle.jpg *License*: Public Domain *Contributors*: Meisam

File:Lossless-circle-canny.png *Source*: http://en.wikipedia.org/w/index.php?title=File:Lossless-circle-canny.png *License*: Public Domain *Contributors*: Meisam

File:Lossy-circle-canny.png *Source*: http://en.wikipedia.org/w/index.php?title=File:Lossy-circle-canny.png *License*: Public Domain *Contributors*: Meisam

file:JPEG example image.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:JPEG_example_image.jpg *License*: GNU Free Documentation License *Contributors*: Maksim, Shyam, WikipediaMaster

file:JPEG example image decompressed.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:JPEG_example_image_decompressed.jpg *License*: GNU Free Documentation License *Contributors*: Original uploader was Cburnett at en.wikipedia

File:JPEG example JPG RIP 100.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:JPEG_example_JPG_RIP_100.jpg *License*: Creative Commons Attribution-ShareAlike 3.0 Unported *Contributors*: Original uploader was Toytoy at en.wikipedia

File:JPEG example JPG RIP 050.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:JPEG_example_JPG_RIP_050.jpg *License*: Creative Commons Attribution-ShareAlike 3.0 Unported *Contributors*: Original uploader was Toytoy at en.wikipedia

File:JPEG example JPG RIP 025.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:JPEG_example_JPG_RIP_025.jpg *License*: Creative Commons Attribution-ShareAlike 3.0 Unported *Contributors*: Original uploader was Toytoy at en.wikipedia

File:JPEG example JPG RIP 010.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:JPEG_example_JPG_RIP_010.jpg *License*: Creative Commons Attribution-ShareAlike 3.0 Unported *Contributors*: Original uploader was Toytoy at en.wikipedia

File:JPEG example JPG RIP 001.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:JPEG_example_JPG_RIP_001.jpg *License*: Creative Commons Attribution-ShareAlike 3.0 Unported *Contributors*: Original uploader was Toytoy at en.wikipedia

File:JPS-sample.jpg *Source*: http://en.wikipedia.org/w/index.php?title=File:JPS-sample.jpg *License*: Creative Commons Attribution-Sharealike 3.0 *Contributors*: User:Ms burosch

License