

JPEG compression

Delve into the mysteries of how the web's favourite image format really works

In this issue...

▶ WHAT'S COVERED

Digital images, especially photographs, contain a lot of redundant information. This means that they're very compressible using specially tuned algorithms. One of the most famous compression algorithms for photos is also one of the oldest: the JPEG format. But how does it work?

RAW mode

Consumer point-and-shoot digital cameras tend to produce JPEG images only. This is very convenient for the home user – essentially what you see through the viewfinder is what's downloaded onto your PC and there's no post-processing required. For professionals or those who need exact control, however, many high-end cameras also have a RAW mode. In this mode, the exact data from the camera's charge-coupled device (CCD) is saved in a lossless, compressed image file. Using these files, the post-processing or editing of the image maintains the highest fidelity, and a JPEG is only produced at the end of the editing session. ■

Compression is the art of removing redundant or duplicate information from a file or a set of bytes in such a way that the original content can be reconstituted (decompressed) at a later stage.

The compression algorithm we are perhaps most familiar with is zip compression – although it's more formally known as 'Deflate'. This is a good example of a 'lossless' compression algorithm: when you compress something with Deflate and then decompress it, you get exactly the same file back, using the same set of bytes. For a data file (say a word-processing document or a spreadsheet), this is exactly what you want. You certainly don't want to lose information through the compression/decompression process.

There is, however, a class of files where it doesn't matter too much if you don't get exactly the same file after going through a compression/decompression cycle. The data in these files is represented in such a way that our human senses can't really tell the difference between the original file and the processed one.

I'm talking about data that we perceive through our eyes and ears: text, video, photos, music and the spoken word. Each of these kinds of files has its own set of compression algorithms that are fine-tuned to how we see or how we hear. For example, a WAV file is an uncompressed audio file (as is the digital data on a CD), but we use MP3s as the file of choice when we store songs in our music player. Files in MP3 format take up much less space but sound more or less as good as WAV files do.

For video, things are much the same, although in practice uncompressed video files aren't used much because of their large size. There are several varieties of video compression formats – among them the familiar MPEG-4 – and each involves a different algorithm. One of the main reasons for this variety of formats is legal: it's often easier to implement your own compression format than battle through the patent issues involved in using someone else's.

There is a plethora of different formats for still pictures, too: BMP, GIF, JPEG, TIFF, PNG and so on. Some of these formats are similarly bedevilled with thorny patent issues (although thankfully, most of these patents have now expired).

The format maze

BMP – or 'bitmap' – is a simple uncompressed format where, in essence, each pixel in a 24-bit image is represented by three bytes, usually encoded as RGB (red, green and blue values).

GIF (Graphics Interchange Format) is a fine example of a lossless compression algorithm for images. It compresses well, especially for non-photographic images. However, images can only contain 256 colours because the GIF format prepends a palette of colours for the byte values 0 to 255 and then uses these values when rendering. A GIF-encoding program must analyse the image being compressed and allocate a palette for the colours in the image. This means that for images using more than 256 colour values, ▶

“JPEGs exploit the imperfect workings of our vision so that the lost information in the image is all but imperceptible”



▲ **Figure 1:** The YCbCr colour space is made up of three components that show colour and intensity.

- ▶ such as photos, the encoder must throw away some information. As a result, GIF is generally only used for relatively simple images that contain contiguous blocks of colour.

JPEG is the interesting one here: it's a lossy compression format conceived explicitly for making photo files smaller and it exploits the imperfect characteristics of our perception.

JPEG files are more correctly described as being in JFIF (JPEG File Interchange Format), which is a limited expression of the full JPEG standard. JPEG stands for the Joint Photographic Experts Group, a committee set up in 1986 by the CCITT standards body with the remit to "establish a standard for the sequential progressive encoding of continuous tone greyscale and colour images". This it did by 1992, and the standard was ratified in 1994.

The basic workflow to create a JPEG file from a bitmap is as follows (Figure 1). First, the bitmap image is converted from RGB to another colour space known as YCbCr.

This colour space, which is also used by TV signals, encodes colour in a different way from RGB (although it covers the same colours).

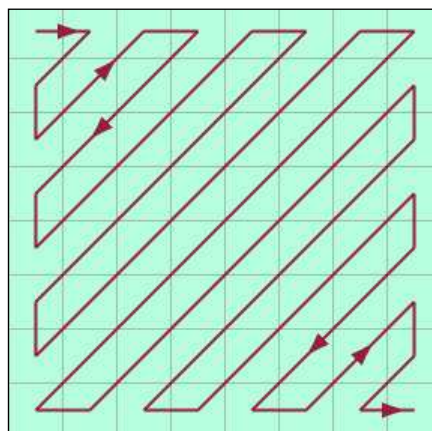
Two of the replacement components, Cb and Cr, are highly compressible. The Y component

$$\begin{aligned}
 Y &= 0.299 R + 0.587 G + 0.114 B \\
 Cb &= -0.1687 R - 0.3313 G + 0.5 B + 128 \\
 Cr &= 0.5 R - 0.4187 G - 0.0813 B + 128
 \end{aligned}$$

▲ **Formula 1:** This formula converts files from RGB to YCbCr colour space.

Editing JPEGs

Since JPEG is a lossy compression algorithm, when you edit a JPEG (say by removing red-eye) and save it, the resulting image will have two sets of compression errors: those artifacts from the original image and those artifacts generated from the save operation. These errors are cumulative. If you need to make several edits to an image, you should perform them all at the same time, and if possible edit a copy of the original JPEG file. Making edits over several sessions, where the image is saved and reloaded as a JPEG, will result in a blurrier, more artifact-ridden image. ■



▲ **Figure 2:** Encoding in a zig-zag pattern rather than along the rows saves even more space.

– the luma – is a value that indicates how bright the pixel is. The Cb and Cr components – the chroma – are the blue and red differences. (Figure 1 represents the YCbCr components as shades of grey. In the Cr image, the lighter the grey, the redder the pixel.)

Positive discrimination

To make this conversion process easy, there's a set of equations to get the luma and chroma values for a single RGB pixel (shown in Formula 1). Rather than compute these various multiplications at run-time, the results can be pre-computed and stored in tables for performance enhancements.

The human eye is able to discriminate the brightness of an image much more finely than its colour information (indeed, at low levels of light we actually see in black and white, since the illumination is too dim to stimulate the cone cells in the retina). This means that the luma value needs much higher fidelity than the two chroma components do. The JPEG format exploits the eye's imbalance by downsampling the chroma values.

Downsampling is simply the process of reducing the chroma values by some factor (and therefore is the first step in losing information). In the JPEG format, there are three accepted possibilities: no downsampling at all, dividing the chroma values horizontally by two, or dividing the chroma values both horizontally or vertically by two.

$$\begin{aligned}
 S_{uv} &= \frac{1}{4} C_u C_v \sum_{x=0}^7 \sum_{y=0}^7 x_{xy} \cos \frac{(2x+1)u\pi}{16} \cos \frac{(2y+1)v\pi}{16} \\
 C_u &= \begin{cases} 1/\sqrt{2} & \text{for } u=0 \\ 1 & \text{otherwise} \end{cases}
 \end{aligned}$$

▲ **Formula 2:** The DCT process zeroes out the higher frequencies that we don't see as clearly.

The next step is to split the downsampled pixels in the image into 8 x 8 blocks. Each colour component is split up separately, and each component sample goes through the same process in what follows. Note that on many occasions, the size of the image will not be a simple multiple of eight pixels in either direction. This can result in some pixel artifacts being created along the right and bottom sides of a JPEG picture.

The next step is fun, but puzzling. Each 8 x 8 block is converted into another matrix using a Discrete Cosine Transform (DCT). This transform, which is similar to a Fourier transform, analyses the frequencies of the original values along each row and column using a set of cosine waves oscillating at different frequencies and amplitudes.

The reason for doing this is that the higher frequencies can be minimized or zeroed out since we do not perceive their loss as acutely as the more energetic lower frequencies. The interesting thing about this transform is that the value with the biggest amplitude of the matrix is found at the top-left cell (known as



▲ **Figure 3:** Typical JPEG block artifacts caused by the compression algorithm (magnification x7).

“ So how do you decompress a JPEG image? Pretty obviously, you need to perform all these steps in reverse order ”

JPEG quality

Image editing programs will allow you to save an image and specify its quality, generally represented as a number from 1 to 10, or maybe as a percentage. The quality value is merely a different quantisation matrix, with a lower quality value represented by a quantisation matrix that will force more AC coefficients to be zero. A lower-quality JPEG file (see Figure 4a) will therefore be smaller but will have more visible artifacts than a higher-quality JPEG file (Figure 4b). Images for the web, for example, would be saved with a middle-to-lower quality whereas images that will be printed would need the maximum quality. ■

the DC coefficient) and the values get smaller the further away from that point they get (all 63 other values are known as the AC coefficients). Generally, we'll need more bits to represent the values in this transformed matrix than can be held in a byte (which is what we've been using up to now).

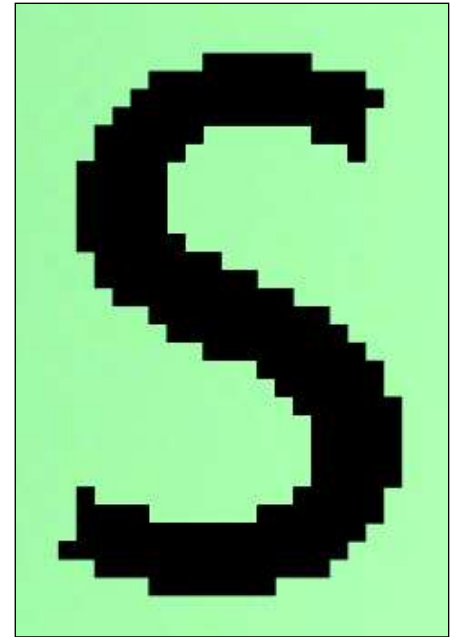
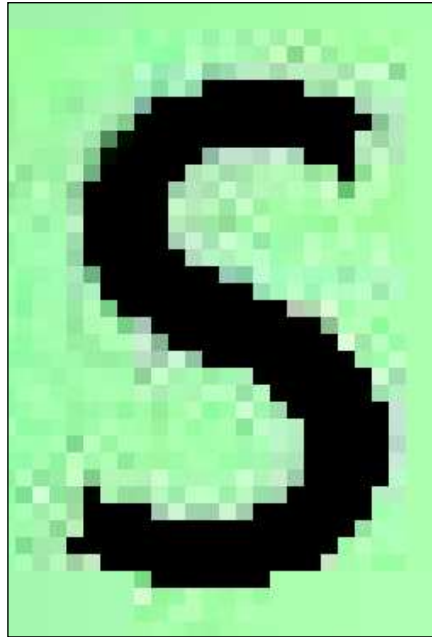
The matrix reloaded

This converted matrix is then quantised. This is the main lossy part of the algorithm and the stage where we minimise the higher frequencies over the lower frequencies. One major result of this quantisation is that many higher DCT coefficients are zeroed out, making them extremely compressible in the next step. The quantisation is accomplished by a set of 8 x 8 matrices, each one representing a different 'quality factor' for the JPEG image. Each cell is divided by the corresponding cell in the quantisation matrix and the result rounded (another lossy operation). Note that this does not involve matrix multiplication in the mathematical sense of the phrase.

Spotlight on... JPEG jaggies

Being a lossy compression algorithm, some information is thrown away when compressing an image using the JPEG format, and that information is approximated when uncompressing the image. The result can be that parts of the image don't look quite right to the human eye. These areas are known as compression artifacts or 'jaggies'.

With JPEG, the main loss of data occurs in the quantisation phase, when the result of a division with a fractional part is rounded into a byte value. The most noticeable jaggies are



▲ Figures 4a and 4b: It's easy to see which file (both magnified) is the low-quality JPEG.

Finally, the resulting quantised matrix is encoded using Huffman compression. To make the most use of the way the values in the matrix seem to radiate out from the top-left corner, the values are encoded not across each row for all rows but in a zig-zag pattern (Figure 2). This means that the zero cells tend to appear at the end of the zig-zag chain and therefore can be ruthlessly compressed (in fact, there's a special code that indicates that all remaining cell values are zero in the 8 x 8 block). Huffman encoding is a lossless compression algorithm. Remember, for each 8 x 8 block of pixels in the original image (192 bytes of information), you will

$$\begin{aligned} R &= Y + 1.402 (Cr-128) \\ G &= Y - 0.34414 (Cb-128) - 0.71414 (Cr-128) \\ B &= Y + 1.772 (Cb-128) \end{aligned}$$

▲ Formula 3: This is the maths needed to convert from JPEG's YCbCr colour space back to RGB.

end up with three compressed 8 x 8 quantised matrices, with the Cr and Cb matrices being the most compressed.

After all that, how do you decompress a JPEG image to a raster bitmap to display on a screen? Well, pretty obviously, you should perform all of these steps in reverse order. First of all, you have to decode the Huffman-compressed 8 x 8 block. This gives you the quantised matrix. Now you can multiply the quantised matrix by the relevant quantisation matrix to give the matrix of DCT coefficients. This is then transformed by the inverse DCT to give the original component matrix in the YCbCr colour space. For each set of three component matrices, we can convert them to the RGB colour space using the inverse colour transformation equations shown in Formula 3. The image will now be decompressed and ready to display. ■

*Julian M Bucknall has worked for companies ranging from TurboPower to Microsoft and is now CTO for Developer Express.
feedback@peplus.co.uk*