

The Squeeze Is On. Using the right kind of compression can save you time *and* space

By Glenn Fleishman

With giant hard drives so widely available and cheap these days, you might think that image compression would be a thing of the past. Running out of space? Why not just throw another five-gigabyte drive on the end of the peripheral chain?

Well, maybe. But just because you can solve some problems with humongous hard drives doesn't mean you shouldn't pay attention to file compression. Compressed files are easier to deal with in many ways—they take up less space and they can be faster to open, place, transform, and print. They're also easier to transfer via e-mail and the Web. In fact, using compression wisely can make the difference between a printer job that outputs beautifully and one that

Using compression wisely can make the difference between . . . a file that downloads quickly and one that'll convince your customers that WWW stands for World Wide Wait.

croaks on an imagesetter, a group of files that can and a group that can't fit on a single floppy, or a file that downloads quickly and one that'll convince your customers that WWW stands for World Wide Wait.

Don't repeat yourself

Compression's *raison d'être* is to make files smaller. It can do this in two ways: either by finding redundant patterns of data and then replacing them with tokens or other symbols that take up less space (this is called lossless compression), or by removing ostensibly less important data from a file (lossy compression). This article covers lossless and lossy still-image compression, but the principles and methods of video and audio compression are the same (virtually all multimedia compression is lossy, however).

Both lossless and lossy compression use algorithms (mathematical operations that are applied to raw data,

such as an RGB image file scanned into Photoshop) to produce a smaller file. Some compression algorithms can be used on a variety of file types; other algorithms (like JPEG and CCITT) work only on images.

Lossless compression. If you compress an image using lossless compression, after decompressing it you have an identical image. No data is lost or changed in any way. It's like a sponge: you can squeeze it down, and when you let go it reverts to its original form.

Lossless schemes are used in programs like WinZIP and Aladdin's StuffIt Deluxe, which can compress any sequence of bytes, not just those in image files. Lossless compression is great for archiving programs and data, because you can't afford to lose a single bit in such cases. But it's also appropriate to use on images you want to preserve exactly—for instance, a high-end drum-scanned image that's been meticulously retouched, pixel by pixel.

Lossy compression. The second type of compression, lossy, actually removes information in the process of squeezing the data. Individual lossy compression methods work (or work well) only on specific kinds of images, and typically yield much smaller file sizes than lossless compression methods.

Good lossy schemes drop out data in a very intelligent manner to minimize the noticeable effect of lost pixels. To do so, they start with assumptions about what kinds of data are most important. For instance, JPEG "thinks" that coarse tonal details are most important and fine color details have the least value.

The skinny on specific algorithms

There are dozens of compression algorithms, but you'll encounter only a few of the most popular again and again. This is partly due to standards—for instance, the de facto TIFF standard uses the LZW algorithm for lossless compression.

Run-length encoding. One of the simplest forms of lossless compression is run-length encoding (RLE), which is used by the Macintosh PICT file format. RLE

works by looking for the same value multiple times in a row. For example, let's say you have a black-and-white raster image of a cow. A program that compresses image data with RLE sees that the first 245 sample points in the image are white, followed by 80 black points, followed by 16 white points, and so on. RLE can compress that data by summarizing: "245 white, 80 black, 16 white," and so on. This compresses a simple image down to almost nothing, and a complex image by just a little; in fact, if an image is complex enough (or has noise in flat-color areas), RLE compression can make the file bigger!

Huffman encoding. Somewhat more complex than RLE, Huffman encoding was invented by David Huffman back in 1952, and is used as part of a number of other compression schemes, like CCITT and JPEG. Huffman encoding is a technique that takes a set of symbols, like the letters in a text file, and analyzes them to determine the frequency of each symbol. It then uses the fewest possible bits to represent the most frequently occurring symbols.

For instance, *E* is the most common letter in standard English text. Huffman encoding might represent *E* in as few as 2 bits (1 followed by 0) instead of the 8 bits needed to signal *E* in ASCII, which is used to store and transmit virtually all text on and between computers. On the other hand, a little-used letter like *Q* or *Z* might require 11 or 12 bits to represent.

LZW. LZW (Lempel-Ziv-Welch), a "pattern-matching" algorithm, compresses data by identifying patterns, called phrases, and storing them in an encoding table that defines shorter "tokens" that can be used in their stead. So if you had the word *ishkabibble* appear 500 times in a text file and you apply LZW, *ishkabibble* would be replaced by a single number that might take only one or two bytes to represent. If you looked that number up in the table, you'd find *ishkabibble*.

Another example: If, in an image, you have a pattern of pink, orange, and green pixels that repeats 50 times, LZW notices this, gives that pattern a number (let's say, 6), and then stores the data as 6 repeated 50 times. Like RLE, LZW works best if you have areas of consistent, noise-free colors, but it's much better than RLE in compressing patterns of data that are more irregular and complex, as in photographic images.

For these reasons, the standard implementations of TIFF and GIF use LZW. But these standards were established before it was clear that developers would have to pay fees for using LZW, which is patented.

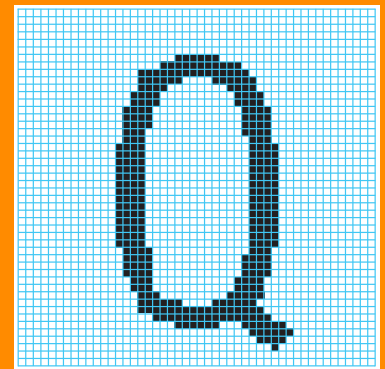
ZIP. To avoid the LZW licensing fees, software developers have begun using a lossless, pattern-based, and patent-free algorithm. The compressor part of the algorithm is called *flate* and the decompressor is called *deflate*. A patent-free implementation of flate/deflate is the zlib library, written by Jean-loup Gailly and Mark Adler (a library is a set of programming code that a developer can include in a program to avoid rewriting an algorithm or other commonly used routine). Adobe adapted the zlib library and introduced the compress-

ion scheme they call ZIP in Acrobat 3.0 and PostScript 3. (Note: You may run into various other compression schemes called "ZIP" or something similar, as in the DOS- and Windows-based PKZIP and WinZIP utilities. Although they use the same compression algorithm, they don't produce identical archives or files.)

ZIP and LZW are both based on pattern-matching, but ZIP typically compresses about 20 percent more than LZW by combining pattern matching with Huffman encoding. Although ZIP itself is a lossless compression scheme, Acrobat offers ZIP compression in both lossless and lossy "flavors." In Acrobat, when you see ZIP or ZIP (8-bit) listed as an option, it's the lossless variety. But Acrobat also offers an option listed as ZIP (4-bit), which reduces the color depth of your images to 4-bit (16 levels of gray or color). Note: Acrobat 3.0 offers ZIP when you select the Acrobat 3.0 com-

Run-length encoding

Run-length encoding works well on simple black-and-white images. In this sample (pixels are shown as grid squares), a single line might be compressed as "21 white, 6 black, 21 white."



patibility option; when you select Acrobat 2.1 compatibility, you get LZW as an option instead of ZIP.

One more note: The PNG format, developed for Web images, also uses ZIP-like compression. For more information, see "Look Out, GIF and JPEG," *Adobe Magazine*, Spring 1998, available on the Web at www.adobe.com/publications/adobemag/pastissues.html.

CCITT encoding. CCITT (for the International Telegraph and Telephone Consultative Committee) encoding was originally developed for fax transmission. It combines run-length and Huffman encoding to compress one-bit-per-pixel, black-and-white raster images. There are several CCITT standards, but you'll see the terms Group 3 and Group 4 most often.

Group 3 was designed for fax transmission, and its compressed data is about twice as big as Group 4, which omits error-correction information and encodes the data more efficiently. Although CCITT itself is lossless, it works only on black-and-white (1-bit) image

data, and it compresses text and line art best.

One place you may run into CCITT is in Adobe Acrobat. For compression of monochrome images, Acrobat lets you choose among CCITT Group 3, Group 4 (the default), Run Length (same as RLE), or ZIP. ZIP is best for monochrome images if most of those images have lots of irregular, complex, and frequent transitions between black and white areas. Otherwise, the Run Length or CCITT Group 4 options work better.

JPEG. The most common lossy compression scheme is currently JPEG. It's almost always implemented with a very simple specification called JFIF (JPEG file interchange format), which supports 8-bit grayscale and 24-bit RGB. However, Adobe (and other vendors that read Adobe JPEG files) also implement a variation that handles 32-bit CMYK raster data as well.

JPEG uses several tricks to maximize compression.

The DCT numbers don't describe the color and position of specific pixels in each 8-by-8 block—they describe the configuration of the block more indirectly. So when you decompress, JPEG reconstructs an approximation of each block based on the DCT numbers.

To get the most from JPEG, it's important to use it on the kind of images it was designed for—"natural" images like photographs, which feature gradual, irregular color transitions. "Synthetic" images, such as screen captures and images created from vector drawings, which feature sharp, artificial color transitions, are most likely to develop artifacts (like out-of-place speckles) when you compress them using JPEG.

Also, using JPEG repeatedly on an image can make it look worse than just doing it once. It's always best to use JPEG at the last stage in a process, such as choosing it as a compression method in Acrobat Distiller.

LZW compression

Uncompressed text

After she asked him why, he asked her why she was asking why. And then, of course, she asked why he had to ask why she was asking why—why didn't he just tell her. "Why don't you trust me?" he asked. "Why not?" she stammered, incredulous. "Yes, why not?" he demanded.

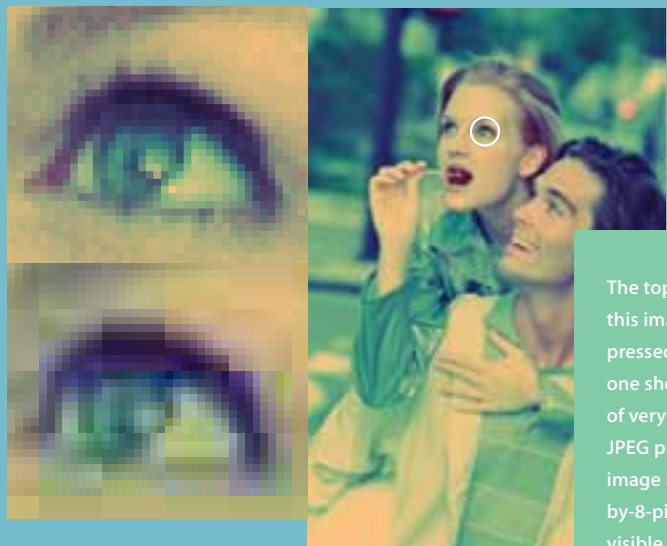
Compressed text

After **1 2 3 4, 5 2 6 4 1 7 8 4**. And then, of course, **1 2 4 5** had to ask **4 1 7 8 4—4** didn't **5** just tell **6**. "**9** don't you trust me?" **5 2**. "**9 10?**" **1** stammered, incredulous. "Yes, **4 10?**" **5** demanded.

Phrase table

1	she
2	asked
3	him
4	why
5	he
6	her
7	was
8	asking
9	Why
10	not

LZW replaces redundant patterns (shown here as words) with short "tokens" (shown as numbers), which it stores in a phrase table.



JPEG compression

The top close-up of this image is uncompressed; the bottom one shows the results of very high JPEG. JPEG processes an image in units of 8-by-8-pixels, which are visible here due to the high compression.

It first converts image data to a LAB-like color space with a luminance channel and two color channels. JPEG then throws away half or three-quarters of the color information (depending on the implementation).

It next applies an algorithm (called DCT for discrete cosine transform) that analyzes 8-by-8-pixel blocks in the file. For each block, it generates a series of numbers that represent the block's features on a spectrum from coarse to fine. The first few numbers represent the overall color of the block, while later numbers represent finer details. The hierarchy of details is based on human perception, so the coarse details (or more "important" features) are most noticeable; the finest, least. Depending on what level of JPEG compression you use, it stores varying amounts of these DCT numbers—more numbers and more image detail for "low" compression, fewer numbers and details for "high" compression. In the last step, JPEG uses Huffman encoding to most efficiently store the resulting data.

Less is more

Choosing the right compression method can produce markedly better results in image quality, file size, and workflow. And that can help you spend less time moving images around and more time working on them.

Thanks to Tom Lane, organizer of the Independent JPEG Group, for his assistance. James D. Murray and William vanRyper's *Encyclopedia of Graphics File Formats* (O'Reilly and Associates, 1994) was also an indispensable resource. See www.ora.com/centers/gff/faqs.htm for more information. ♦

Glenn Fleishman is a contributing editor for Adobe Magazine. This article is partly adapted from the second edition of Real World Scanning and Halftones (Peachpit Press, 1998, ISBN 0201696835), which he co-authored with David Blatner and Stephen F. Roth. More information on the book and related resources is available at www.rwsh.com.