Part1 - Part2 - MultiPage

JPEG image compression FAQ, part 1/2

Newsgroups: comp.graphics.misc, comp.infosystems.www.authoring.images
From: tgl@netcom.com (Tom Lane)
Subject: JPEG image compression FAQ, part 1/2
Message-ID: <jpeg-faq-p1_922674260@netcom.com>
Summary: General questions and answers about JPEG
Keywords: JPEG, image compression, FAQ, JPG, JFIF
Reply-To: jpeg-info@uunet.uu.net
Date: Mon, 29 Mar 1999 02:24:27 GMT
Sender: tgl@netcom17.netcom.com

Archive-name: jpeg-faq/part1
Posting-Frequency: every 14 days
Last-modified: 28 March 1999

This article answers Frequently Asked Questions about JPEG image compression.
This is part 1, covering general questions and answers about JPEG. Part 2
gives system-specific hints and program recommendations. As always,
suggestions for improvement of this FAQ are welcome.

New since version of 14 March 1999:
  * Expanded item 10 to discuss lossless rotation and cropping of JPEGs.

This article includes the following sections:

Basic questions:

More advanced questions:

Miscellaneous:

This article and its companion are posted every 2 weeks. If you can't find
part 2, you can get it from the news.answers archive at rtfm.mit.edu
(see "[24] Where are FAQ lists archived?"). Part 2 changes very frequently;
get a new copy if the one you are reading is more than a couple months old.

Subject: [1] What is JPEG?

JPEG (pronounced "jay-peg") is a standardized image compression mechanism. JPEG stands for Joint Photographic Experts Group, the original name of the committee that wrote the standard.

JPEG is designed for compressing either full-color or gray-scale images of natural, real-world scenes. It works well on photographs, naturalistic artwork, and similar material; not so well on lettering, simple cartoons, or line drawings. JPEG handles only still images, but there is a related standard called MPEG for motion pictures.

JPEG is "lossy," meaning that the decompressed image isn't quite the same as the one you started with. (There are lossless image compression algorithms, but JPEG achieves much greater compression than is possible with lossless methods.) JPEG is designed to exploit known limitations of the human eye, notably the fact that small color changes are perceived less accurately than small changes in brightness. Thus, JPEG is intended for compressing images that will be looked at by humans. If you plan to machine-analyze your images, the small errors introduced by JPEG may be a problem for you, even if they are invisible to the eye.

A useful property of JPEG is that the degree of lossiness can be varied by adjusting compression parameters. This means that the image maker can trade off file size against output image quality. You can make *extremely* small files if you don't mind poor quality; this is useful for applications such as indexing image archives. Conversely, if you aren't happy with the output quality at the default compression setting, you can jack up the quality until you are satisfied, and accept lesser compression.

Another important aspect of JPEG is that decoders can trade off decoding speed against image quality, by using fast but inaccurate approximations to the required calculations. Some viewers obtain remarkable speedups in this way. (Encoders can also trade accuracy for speed, but there's usually less reason to make such a sacrifice when writing a file.)

Subject: [2] Why use JPEG?

There are two good reasons: to make your image files smaller, and to store 24-bit-per-pixel color data instead of 8-bit-per-pixel data.

Making image files smaller is a win for transmitting files across networks and for archiving libraries of images. Being able to compress a 2 Mbyte full-color file down to, say, 100 Kbytes makes a big difference in disk space and transmission time! And JPEG can easily provide 20:1 compression of full-color data. If you are comparing GIF and JPEG, the size ratio is usually more like 4:1 (see "[4] How well does JPEG compress images?").

Now, it takes longer to decode and view a JPEG image than to view an image of a simpler format such as GIF. Thus using JPEG is essentially a time/space tradeoff: you give up some time in order to store or transmit an image more cheaply. But it's worth noting that when network transmission is involved, the time savings from transferring a shorter file can be greater than the time needed to decompress the file.

The second fundamental advantage of JPEG is that it stores full color information: 24 bits/pixel (16 million colors). GIF, the other image format widely used on the net, can only store 8 bits/pixel (256 or fewer colors). GIF is reasonably well matched to inexpensive computer displays --- most run-of-the-mill PCs can't display more than 256 distinct colors at once. But full-color hardware is getting cheaper all the time, and JPEG photos look *much* better than GIFs on such hardware. Within a couple of years, GIF will probably seem as obsolete as black-and-white MacPaint format does today. Furthermore, JPEG is far more useful than GIF for exchanging images among people with widely varying display hardware, because it avoids prejudging how many colors to use (see "[8] What is color quantization?"). Hence JPEG is considerably more appropriate than GIF for use as a Usenet and World Wide Web standard photo format.

A lot of people are scared off by the term "lossy compression". But when it comes to representing real-world scenes, *no* digital image format can retain all the information that impinges on your eyeball. By comparison with the real-world scene, JPEG loses far less information than GIF. The real disadvantage of lossy compression is that if you repeatedly compress and decompress an image, you lose a little more quality each time (see "[10] Does loss accumulate with repeated compression/decompression?"). This is a serious objection for some applications but matters not at all for many others.

Subject: [3] When should I use JPEG, and when should I stick with GIF?

JPEG is *not* going to displace GIF entirely; for some types of images, GIF is superior in image quality, file size, or both. One of the first things to learn about JPEG is which kinds of images to apply it to.

Generally speaking, JPEG is superior to GIF for storing full-color or gray-scale images of "realistic" scenes; that means scanned photographs, continuous-tone artwork, and similar material. Any smooth variation in color, such as occurs in highlighted or shaded areas, will be represented more faithfully and in less space by JPEG than by GIF.

GIF does significantly better on images with only a few distinct colors, such as line drawings and simple cartoons. Not only is GIF lossless for such images, but it often compresses them more than JPEG can. For example, large areas of pixels that are all *exactly* the same color are compressed very efficiently indeed by GIF. JPEG can't squeeze such data as much as GIF does without introducing visible defects. (One implication of this is that large single-color borders are quite cheap in GIF files, while they are best avoided in JPEG files.)

Computer-drawn images, such as ray-traced scenes, usually fall between photographs and cartoons in terms of complexity. The more complex and subtly rendered the image, the more likely that JPEG will do well on it. The same goes for semi-realistic artwork (fantasy drawings and such). But icons that use only a few colors are handled better by GIF.

JPEG has a hard time with very sharp edges: a row of pure-black pixels adjacent to a row of pure-white pixels, for example. Sharp edges tend to come out blurred unless you use a very high quality setting. Edges this sharp are rare in scanned photographs, but are fairly common in GIF files: consider borders, overlaid text, etc. The blurriness is particularly objectionable with text that's only a few pixels high. If you have a GIF with a lot of small-size overlaid text, don't JPEG it. (If you want to attach descriptive text to a JPEG image, put it in as a comment rather than trying to overlay it on the image. Most recent JPEG software can deal with textual comments in a JPEG file, although older viewers may just ignore the comments.)

Plain black-and-white (two level) images should never be converted to JPEG; they violate all of the conditions given above. You need at least about 16 gray levels before JPEG is useful for gray-scale images. It should also be noted that GIF is lossless for gray-scale images of up to 256 levels, while JPEG is not.

If you have a large library of GIF images, you may want to save space by converting the GIFs to JPEG. This is trickier than it may seem --- even when the GIFs contain photographic images, they are actually very poor source material for JPEG, because the images have been color-reduced. Non-photographic images should generally be left in GIF form. Good-quality photographic GIFs can often be converted with no visible quality loss, but only if you know what you are doing and you take the time to work on each image individually. Otherwise you're likely to lose a lot of image quality or waste a lot of disk space ... quite possibly both. Read sections 8 and 9 if you want to convert GIFs to JPEG.

Subject: [4] How well does JPEG compress images?

Very well indeed, when working with its intended type of image (photographs and suchlike). For full-color images, the uncompressed data is normally 24 bits/pixel. The best known lossless compression methods can compress such data about 2:1 on average. JPEG can typically achieve 10:1 to 20:1 compression without visible loss, bringing the effective storage requirement down to 1 to 2 bits/pixel. 30:1 to 50:1 compression is possible with small to moderate defects, while for very-low-quality purposes such as previews or archive indexes, 100:1 compression is quite feasible. An image compressed 100:1 with JPEG takes up the same space as a full-color one-tenth-scale thumbnail image, yet it retains much more detail than such a thumbnail.

For comparison, a GIF version of the same image would start out by sacrificing most of the color information to reduce the image to 256 colors (8 bits/pixel). This provides 3:1 compression. GIF has additional "LZW" compression built in, but LZW doesn't work very well on typical photographic data; at most you may get 5:1 compression overall, and it's not at all uncommon for LZW to be a net loss (i.e., less than 3:1 overall compression). LZW *does* work well on simpler images such as line drawings, which is why

GIF handles that sort of image so well. When a JPEG file is made from full-color photographic data, using a quality setting just high enough to prevent visible loss, the JPEG will typically be a factor of four or five smaller than a GIF file made from the same data.

Gray-scale images do not compress by such large factors. Because the human eye is much more sensitive to brightness variations than to hue variations, JPEG can compress hue data more heavily than brightness (gray-scale) data. A gray-scale JPEG file is generally only about 10%-25% smaller than a full-color JPEG file of similar visual quality. But the uncompressed gray-scale data is only 8 bits/pixel, or one-third the size of the color data, so the calculated compression ratio is much lower. The threshold of visible loss is often around 5:1 compression for gray-scale images.

The exact threshold at which errors become visible depends on your viewing conditions. The smaller an individual pixel, the harder it is to see an error; so errors are more visible on a computer screen (at 70 or so dots/inch) than on a high-quality color printout (300 or more dots/inch). Thus a higher-resolution image can tolerate more compression ... which is fortunate considering it's much bigger to start with. The compression ratios quoted above are typical for screen viewing. Also note that the threshold of visible error varies considerably across images.

Subject: [5] What are good "quality" settings for JPEG?

Most JPEG compressors let you pick a file size vs. image quality tradeoff by selecting a quality setting. There seems to be widespread confusion about the meaning of these settings. "Quality 95" does NOT mean "keep 95% of the information", as some have claimed. The quality scale is purely arbitrary; it's not a percentage of anything.

In fact, quality scales aren't even standardized across JPEG programs. The quality settings discussed in this article apply to the free IJG JPEG software (see part 2, item 15), and to many programs based on it. Some other JPEG implementations use completely different quality scales. For example:

* Apple used to use a scale running from 0 to 4, not 0 to 100.
* Recent Apple software uses an 0-100 scale that has nothing to do with the IJG scale (their Q 50 is about the same as Q 80 on the IJG scale).
* Paint Shop Pro's scale is the exact opposite of the IJG scale, PSP setting N = IJG 100-N; thus lower numbers are higher quality in PSP.
* Adobe Photoshop doesn't use a numeric scale at all, it just gives you "high"/"medium"/"low" choices. (But I hear this is changing in 4.0.)

Fortunately, this confusion doesn't prevent different implementations from exchanging JPEG files. But you do need to keep in mind that quality scales vary considerably from one JPEG-creating program to another, and that just saying "I saved this at Q 75" doesn't mean a thing if you don't say which program you used.

In most cases the user's goal is to pick the lowest quality setting, or smallest file size, that decompresses into an image indistinguishable from the original. This setting will vary from one image to another and from one observer to another, but here are some rules of thumb.

For good-quality, full-color source images, the default IJG quality setting (Q 75) is very often the best choice. This setting is about the lowest you can go without expecting to see defects in a typical image. Try Q 75 first; if you see defects, then go up.

If the image was less than perfect quality to begin with, you might be able to drop down to Q 50 without objectionable degradation. On the other hand, you might need to go to a *higher* quality setting to avoid further loss. This is often necessary if the image contains dithering or moire patterns (see "[9] What are some rules of thumb for converting GIF images to JPEG?").

Except for experimental purposes, never go above about Q 95; using Q 100 will produce a file two or three times as large as Q 95, but of hardly any better quality. Q 100 is a mathematical limit rather than a useful setting. If you see a file made with Q 100, it's a pretty sure sign that the maker didn't know what he/she was doing.

If you want a very small file (say for preview or indexing purposes) and are prepared to tolerate large defects, a Q setting in the range of 5 to 10 is about right. Q 2 or so may be amusing as "op art". (It's worth mentioning that the current IJG software is not optimized for such low quality factors. Future versions may achieve better image quality for the same file size at low quality settings.)

If your image contains sharp colored edges, you may notice slight fuzziness or jagginess around such edges no matter how high you make the quality setting. This can be suppressed, at a price in file size, by turning off chroma downsampling in the compressor. The IJG encoder regards downsampling as a separate option which you can turn on or off independently of the Q setting. With the "cjpeg" program, the command line switch "-sample 1x1" turns off downsampling; other programs based on the IJG library may have checkboxes or other controls for downsampling. Other JPEG implementations may or may not provide user control of downsampling. Adobe Photoshop, for example, automatically switches off downsampling at its higher quality settings. On most photographic images, we recommend leaving downsampling on, because it saves a significant amount of space at little or no visual penalty.

For images being used on the World Wide Web, it's often a good idea to give up a small amount of image quality in order to reduce download time. Quality settings around 50 are often perfectly acceptable on the Web. In fact, a user viewing such an image on a browser with a 256-color display is unlikely to be able to see any difference from a higher quality setting, because the browser's color quantization artifacts will swamp any imperfections in the JPEG image itself. It's also worth knowing that current progressive-JPEG-making programs use default progression sequences that are tuned for quality settings around 50-75: much below 50, the early scans will look really bad, while much above 75, the later scans won't contribute anything noticeable to the picture.

Subject: [6] Where can I get JPEG software?

See part 2 of this FAQ for recommendations about programs for particular systems. Part 2 also tells where to find free source code for implementing JPEG, in case you want to write your own programs using JPEG.

The comp.graphics.* FAQs and the alt.binaries.pictures FAQ are more general sources of information about graphics programs available on the Internet (see "[24] Where are FAQ lists archived?").

Subject: [7] How do I view JPEG images posted on Usenet?

Image files posted on the alt.binaries.pictures.* newsgroups are usually "uuencoded". Uuencoding converts binary image data into text that can safely be posted. Most posters also divide large posts into multiple parts, since some news software can't cope with big articles. Before your viewer will recognize the image, you must combine the parts into one file and run the text through a uudecode program. (This is all true for GIF as well as JPEG, by the way.) There are programs available to automate this process.

For more info see the alt.binaries.pictures FAQ, which is available from
http://www.faqs.org/faqs/pictures-faq/
(see also "[24] Where are FAQ lists archived?").

Subject: [8] What is color quantization?

Many people don't have full-color (24 bit per pixel) display hardware. Inexpensive display hardware stores 8 bits per pixel, so it can display at most 256 distinct colors at a time. To display a full-color image, the computer must choose an appropriate set of representative colors and map the image into these colors. This process is called "color quantization". (This is something of a misnomer; "color selection" or "color reduction" would be a better term. But we're stuck with the standard usage.)

Clearly, color quantization is a lossy process. It turns out that for most images, the details of the color quantization algorithm have *much* more impact on the final image quality than do any errors introduced by JPEG itself (except at the very lowest JPEG quality settings). Making a good color quantization method is a black art, and no single algorithm is best for all images.

Since JPEG is a full-color format, displaying a color JPEG image on 8-bit-or-less hardware requires color quantization. The speed and image quality of a JPEG viewer running on such hardware are largely determined by its quantization algorithm. Depending on whether a quick-and-dirty or good-but-slow method is used, you'll see great variation in image quality among viewers on 8-bit displays, much more than occurs on 24-bit displays.

On the other hand, a GIF image has already been quantized to 256 or fewer

colors. (A GIF always has a specific number of colors in its palette, and the format doesn't allow more than 256 palette entries.) GIF has the advantage that the image maker precomputes the color quantization, so viewers don't have to; this is one of the things that make GIF viewers faster than JPEG viewers. But this is also the *disadvantage* of GIF: you're stuck with the image maker's quantization. If the maker quantized to a different number of colors than what you can display, you'll either waste display capability or else have to requantize to reduce the number of colors (which usually results in much poorer image quality than quantizing once from a full-color image). Furthermore, if the maker didn't use a high-quality color quantization algorithm, you're out of luck --- the image is ruined.

For this reason, JPEG promises significantly better image quality than GIF for all users whose machines don't match the image maker's display hardware. JPEG's full color image can be quantized to precisely match the viewer's display hardware. Furthermore, you will be able to take advantage of future improvements in quantization algorithms, or purchase better display hardware, to get a better view of JPEG images you already have. With a GIF, you're stuck forevermore with what was sent.

A closely related problem is seen in many current World Wide Web browsers: when running on an 8-bit display, they force all images into a pre-chosen palette. (They do this to avoid having to worry about how to allocate the limited number of available color slots among the various items on a Web page.) A GIF version of a photo usually degrades very badly in this situation, because it's effectively being forced through a second quantization step. A JPEG photo won't look wonderful either, but it will look less bad than the GIF equivalent because it's been quantized only once.

A growing number of people have better-than-8-bit display hardware already: 15- or 16-bit/pixel "high color" displays are now quite common, and true 24-bit/pixel displays are no longer rare. For these people, GIF is already obsolete, as it cannot represent an image to the full capabilities of their display. JPEG images can drive these displays much more effectively.

In short, JPEG is an all-around better choice than GIF for representing photographic images in a machine-independent fashion.

It's sometimes thought that a JPEG converted from a GIF shouldn't require color quantization. That is false; even when you feed a 256-or-less-color GIF into JPEG, what comes out of the decompressor is not 256 colors, but thousands of colors. This happens because JPEG's lossiness affects each pixel a little differently, so two pixels that started with identical colors will usually come out with slightly different colors. Considering the whole image, each original color gets "smeared" into a cluster of nearby colors. Therefore quantization is always required to display a color JPEG on a colormapped display, regardless of the image source.

The same effect makes it nearly meaningless to talk about the number of colors used by a JPEG image. Even if you tried to count the number of distinct pixel values, different JPEG decoders would give you different results because of roundoff error differences. I occasionally see posted images described as "256-color JPEG". This tells me that the poster (a) hasn't read this FAQ and (b) probably converted the JPEG from a GIF. JPEGs can be classified as color or gray-scale, but number of colors just isn't a useful concept for JPEG, any more than it is for a real photograph.

Subject: [9] What are some rules of thumb for converting GIF images to JPEG?

Converting GIF files to JPEG is a tricky business --- you are piling one set of limitations atop a quite different set, and the results can be awful. Certainly a JPEG made from a GIF will never be as good as a JPEG made from true 24-bit color data. But if what you've got is GIFs, and you need to save space, here are some hints for getting the best results.

With care and a clean source image, it's often possible to make a JPEG of quality equivalent to the GIF. This does not mean that the JPEG looks pixel-for-pixel identical to the GIF --- it won't. Especially not on an 8-bit display, because the color quantization process used to display the JPEG probably won't quite match the quantization process used to make the GIF from the original data (see "[8] What is color quantization?"). But remember that the GIF itself is not all that faithful to the full-color original, if you look at individual pixels. Looking at the overall image, a converted JPEG can look as good as its GIF source. Some people claim that on 24-bit displays, a carefully converted JPEG can actually look better than the GIF source, because dither patterns have been eliminated. (More about

dithering in a moment.)

On the other hand, JPEG conversion absolutely *will* degrade an unsuitable image or one that is converted carelessly. If you are not willing to take the amount of trouble suggested below, you're much better off leaving your GIF images alone. Simply cranking the JPEG quality setting up to a very high value wastes space (which defeats the whole point of the exercise, no?) and some images will be degraded anyway.

The first rule is never to convert an image that's not appropriate for JPEG (see "[3] When should I use JPEG, and when should I stick with GIF?"). Large, high-visual-quality photographic images are usually the best source material. And they take up lots of space in GIF form, so they offer significant potential space savings. (A good rule of thumb is not to bother converting any GIF that's much under 100 Kbytes; the potential savings isn't worth the hassle.)

The second rule is to know where the image came from. Repeated GIF<=>JPEG conversions are guaranteed to turn an image into mush, because you pay a steep quality price on each round trip. Don't reconvert images that have been converted before.

The third rule is to get rid of the border. Many people have developed an odd habit of putting a large single-color border around a GIF image. While useless, this is nearly free in terms of storage cost in GIF files. It is *not* free in JPEG files, either in storage space or in decoding time. Worse, the sharp border boundary can create visible artifacts (ghost edges). Furthermore, when viewing a bordered JPEG on an 8-bit display, the quantizer will think the border color is important because there's so much of it, and hence will waste color palette entries on the border, thus actually reducing the displayed quality of the main part of the image! So do yourself a favor and crop off any border before JPEGing.

The final rule is to look at each JPEG, to make sure you are happy with it, before throwing away the corresponding GIF. This will give you a chance to re-do the conversion with a higher quality setting if necessary. Also compare the file sizes --- if the image isn't suitable JPEG material, a JPEG file of reasonable quality may come out *larger* than the GIF.

Gray-scale photos usually convert without much problem. When using cjpeg, be sure to use the -gray switch. (Otherwise, cjpeg treats a GIF as color data; this works, but it wastes space and time if the image is really only gray-scale.) Quality settings around the default (75) are usually fine.

Color images are much trickier. Color GIFs of photographic images are usually "dithered" to fool your eye into seeing more than the 256 colors that GIF can actually store. If you enlarge the image, you will find that adjacent pixels are often of significantly different colors; at normal size the eye averages these pixels together to produce the illusion of an intermediate color value. The trouble with dithering is that, to JPEG, it looks like high-spatial-frequency color noise; and JPEG can't compress noise very well. The resulting JPEG file is both larger and of lower image quality than what you would have gotten from JPEGing the original full color image (if you had it). To get around this, you need to "smooth" the GIF image before compression. Smoothing averages together nearby pixels, thus approximating the color that you thought you saw anyway, and in the process getting rid of the rapid color changes that give JPEG trouble. Proper use of smoothing will both reduce the size of the compressed file and give you a better-looking output image than you'd get without smoothing.

With the IJG JPEG software (cjpeg or derived programs), a simple smoothing capability is built in. Try "-smooth 10" or so when converting GIFs. Values of 10 to 25 seem to work well for high-quality GIFs. GIFs with heavy-handed dithering may require larger smoothing factors. (If you can see regular fine-scale patterns on the GIF image even without enlargement, then strong smoothing is definitely called for.) Too large a smoothing factor will blur the output image, which you don't want. If you are an image processing wizard, you can also do smoothing with a separate filtering program, but appropriate use of such tools is beyond the scope of this FAQ.

Quality settings around 85 (a bit higher than default) usually work well when converting color GIFs, assuming that you've picked a good smoothing factor. You may need still higher quality settings if you can't hide the dithering pattern with a reasonable smoothing factor. Really badly dithered GIFs are best left as GIFs.

Don't expect JPEG files converted from GIFs to be as small as those created directly from full-color originals. The dithering noise wastes space, but you won't be able to smooth away all the noise without blurring the image.

Typically, a good-quality converted JPEG will be one-half to one-third the size of the GIF file, not one-fourth as suggested in section 4. If the JPEG comes out much more than half the size of the GIF, this is a good sign that the image shouldn't be converted at all.

The upshot of all this is that "cjpeg -quality 85 -smooth 10" is probably a good starting point for converting color GIFs. But if you care about the image, you'll want to check the results and maybe try a few other settings. Blindly converting a large GIF library at this or any other setting is a recipe for disaster.

Subject: [10] Does loss accumulate with repeated compression/decompression?

It would be nice if, having compressed an image with JPEG, you could decompress it, manipulate it (crop off a border, say), and recompress it without any further image degradation beyond what you lost initially. Unfortunately THIS IS NOT THE CASE. In general, recompressing an altered image loses more information. Hence it's important to minimize the number of generations of JPEG compression between initial and final versions of an image.

There are a few specialized operations that can be done on a JPEG file without decompressing it, and thus without incurring the generational loss that you'd normally get from loading and re-saving the image in a regular image editor. In particular it is possible to do 90-degree rotations and flips losslessly, if the image dimensions are a multiple of the file's block size (typically 16x16, 16x8, or 8x8 pixels for color JPEGs). This fact used to be just an academic curiosity, but it has assumed practical importance recently because many users of digital cameras would like to be able to rotate their images from landscape to portrait format without incurring loss --- and practically all digicams that produce JPEG files produce images of the right dimensions for these operations to work. So software that can do lossless JPEG transforms has started to pop up. But you do need special software; rotating the image in a regular image editor won't be lossless.

It turns out that if you decompress and recompress an image at the same quality setting first used, relatively little further degradation occurs. This means that you can make local modifications to a JPEG image without material degradation of other areas of the image. (The areas you change will still degrade, however.) Counterintuitively, this works better the lower the quality setting. But you must use *exactly* the same setting, or all bets are off. Also, the decompressed image must be saved in a full-color format; if you do something like JPEG=>GIF=>JPEG, the color quantization step loses lots of information.

Unfortunately, cropping doesn't count as a local change! JPEG processes the image in small blocks, and cropping usually moves the block boundaries, so that the image looks completely different to JPEG. You can take advantage of the low-degradation behavior if you are careful to crop the top and left margins only by a multiple of the block size (typically 16 pixels), so that the remaining blocks start in the same places. (True lossless cropping is possible under the same restrictions about where to crop, but again this requires specialized software.)

The bottom line is that JPEG is a useful format for compact storage and transmission of images, but you don't want to use it as an intermediate format for sequences of image manipulation steps. Use a lossless 24-bit format (PNG, TIFF, PPM, etc) while working on the image, then JPEG it when you are ready to file it away or send it out on the net. If you expect to edit your image again in the future, keep a lossless master copy to work from. The JPEG you put up on your Web site should be a derived copy, not your editing master.

Subject: [11] What is progressive JPEG?

A simple or "baseline" JPEG file is stored as one top-to-bottom scan of the image. Progressive JPEG divides the file into a series of scans. The first scan shows the image at the equivalent of a very low quality setting, and therefore it takes very little space. Following scans gradually improve the quality. Each scan adds to the data already provided, so that the total storage requirement is roughly the same as for a baseline JPEG image of the same quality as the final scan. (Basically, progressive JPEG is just a rearrangement of the same data into a more complicated order.)

The advantage of progressive JPEG is that if an image is being viewed on-the-fly as it is transmitted, one can see an approximation to the whole

image very quickly, with gradual improvement of quality as one waits longer; this is much nicer than a slow top-to-bottom display of the image. The disadvantage is that each scan takes about the same amount of computation to display as a whole baseline JPEG file would. So progressive JPEG only makes sense if one has a decoder that's fast compared to the communication link. (If the data arrives quickly, a progressive-JPEG decoder can adapt by skipping some display passes. Hence, those of you fortunate enough to have T1 or faster net links may not see any difference between progressive and regular JPEG; but on a modem-speed link, progressive JPEG is great.)

Up until recently, there weren't many applications in which progressive JPEG looked attractive, so it hasn't been widely implemented. But with the popularity of World Wide Web browsers running over slow modem links, and with the ever-increasing horsepower of personal computers, progressive JPEG has become a win for WWW use. IJG's free JPEG software (see part 2, item 15) now supports progressive JPEG, and the capability is spreading fast in WWW browsers and other programs.

Except for the ability to provide progressive display, progressive JPEG and baseline JPEG are basically identical, and they work well on the same kinds of images. It is possible to convert between baseline and progressive representations of an image without any quality loss. (But specialized software is needed to do this; conversion by decompressing and recompressing is *not* lossless, due to roundoff errors.)

A progressive JPEG file is not readable at all by a baseline-only JPEG decoder, so existing software will have to be upgraded before progressive JPEG can be used widely. See item 16 in part 2 for the latest news about which programs support it.

Subject: [12] Can I make a transparent JPEG?

No. JPEG does not support transparency and is not likely to do so any time soon. It turns out that adding transparency to JPEG would not be a simple task; read on if you want the gory details.

The traditional approach to transparency, as found in GIF and some other file formats, is to choose one otherwise-unused color value to denote a transparent pixel. That can't work in JPEG because JPEG is lossy: a pixel won't necessarily come out *exactly* the same color that it started as. Normally, a small error in a pixel value is OK because it affects the image only slightly. But if it changes the pixel from transparent to normal or vice versa, the error would be highly visible and annoying, especially if the actual background were quite different from the transparent color.

A more reasonable approach is to store an alpha channel (transparency percentage) as a separate color component in a JPEG image. That could work since a small error in alpha makes only a small difference in the result. The problem is that a typical alpha channel is exactly the sort of image that JPEG does very badly on: lots of large flat areas and sudden jumps. You'd have to use a very high quality setting for the alpha channel. It could be done, but the penalty in file size is large. A transparent JPEG done this way could easily be double the size of a non-transparent JPEG. That's too high a price to pay for most uses of transparency.

The only real solution is to combine lossy JPEG storage of the image with lossless storage of a transparency mask using some other algorithm. Developing, standardizing, and popularizing a file format capable of doing that is not a small task. As far as I know, no serious work is being done on it; transparency doesn't seem worth that much effort.

Subject: [13] Isn't there a lossless JPEG?

There's a great deal of confusion on this subject, which is not surprising because there are several different compression methods all known as "JPEG". The commonly used method is "baseline JPEG" (or its variant "progressive JPEG"). The same ISO standard also defines a very different method called "lossless JPEG". And if that's not confusing enough, a new lossless standard called "JPEG-LS" is about to hit the streets.

When I say "lossless", I mean mathematically lossless: a lossless compression algorithm is one that guarantees its decompressed output is bit-for-bit identical to the original input. This is a much stronger claim than "visually indistinguishable from the original". Baseline JPEG can reach visual indistinguishability for most photo-like images, but it can never be truly lossless.

Lossless JPEG is a completely different method that really is lossless. However, it doesn't compress nearly as well as baseline JPEG; it typically can compress full-color data by around 2:1. And lossless JPEG works well only on continuous-tone images. It does not provide useful compression of palette-color images or low-bit-depth images.

Lossless JPEG has never been popular --- in fact, no common applications support it --- and it is now largely obsolete. (For example, the new PNG standard outcompresses lossless JPEG on most images.) Recognizing this, the ISO JPEG committee recently finished an all-new lossless compression standard called JPEG-LS (you may have also heard of it under the name LOCO). JPEG-LS gives better compression than original lossless JPEG, but still nowhere near what you can get with a lossy method. It's anybody's guess whether this new standard will achieve any popularity.

It's worth repeating that cranking a regular JPEG implementation up to its maximum quality setting *does not* get you lossless storage; even at the highest possible quality setting, baseline JPEG is lossy because it is subject to roundoff errors in various calculations. Roundoff errors alone are nearly always too small to be seen, but they will accumulate if you put the image through multiple cycles of compression (see section 10).

Many implementations won't even let you get to the maximum possible setting, because it's such an inefficient way to use regular JPEG. With the IJG JPEG software, for example, you have to not only select "quality 100" but also turn off chroma downsampling to minimize loss of information. The resulting files are far larger and of only fractionally better quality than files generated at more reasonable settings. And they're still slightly lossy! If you really need lossless storage, don't try to approximate it with regular JPEG.

Subject: [14] Why all the argument about file formats?

Strictly speaking, JPEG refers only to a family of compression algorithms; it does *not* refer to a specific image file format. The JPEG committee was prevented from defining a file format by turf wars within the international standards organizations.

Since we can't actually exchange images with anyone else unless we agree on a common file format, this leaves us with a problem. In the absence of official standards, a number of JPEG program writers have just gone off to "do their own thing", and as a result their programs aren't compatible with anyone else's.

The closest thing we have to a standard JPEG format is some work that's been coordinated by people at C-Cube Microsystems. They have defined two JPEG-based file formats:

* JFIF (JPEG File Interchange Format), a "low-end" format that transports pixels and not much else.
* TIFF/JPEG, aka TIFF 6.0, an extension of the Aldus TIFF format. TIFF is a "high-end" format that will let you record just about everything you ever wanted to know about an image, and a lot more besides :-).

JFIF has emerged as the de-facto standard on Internet, and is what is most commonly meant by "a JPEG file". Most JFIF readers are also capable of handling some not-quite-JFIF-legal variant formats.

The TIFF 6.0 spec for incorporating JPEG is not widely implemented, partly because it has some serious design flaws. A revised TIFF/JPEG design is now described by TIFF Technical Note #2; this design will be the one used in TIFF 7.0. New implementations of TIFF should use the Tech Note's design for embedding JPEG, not the TIFF 6.0 design. (As far as I know, NeXTStep systems are the only ones making any significant use of TIFF 6.0 style TIFF/JPEG.) Even when TIFF/JPEG is stable, it will never be as widely used as JFIF. TIFF is far more complex than JFIF, and is generally less transportable because different vendors often implement slightly different, nonoverlapping subsets of TIFF. Adding JPEG to the mix hasn't helped any.

Apple's Macintosh QuickTime software uses a JFIF-compatible datastream wrapped inside the Mac-specific PICT format. Conversion between JFIF and PICT/JPEG is pretty straightforward, and several Mac programs are available to do it (see part 2, item 8). If you have an editor that handles binary files, you can even strip a PICT/JPEG file down to JFIF by hand; see the next section for details.

News flash: the ISO JPEG committee seems to have won their turf wars. They have defined a complete file format spec called SPIFF in the new "Part 3" extensions to the JPEG standard. It's pretty late in the game though, so

whether this will have much impact on real-world files remains to be seen.
SPIFF is upward compatible with JFIF, so if it does get widely adopted, most
users probably won't even notice.

Subject: [15] How do I recognize which file format I have,
and what do I do about it?

If you have an alleged JPEG file that your software won't read, it's likely
to be some proprietary JPEG-based format. You can tell what you have by
inspecting the first few bytes of the file:

1. A JFIF-standard file will start with the four bytes (hex) FF D8 FF E0,
followed by two variable bytes (often hex 00 10), followed by 'JFIF'.

2. If you see FF D8 FF at the start, but not the 'JFIF' marker, you
probably have a not-quite-JFIF JPEG file. Most JFIF software should
read it without complaint. If you are using something that is picky
enough to complain about the lack of a JFIF marker, try another decoder.
(Both very old JPEG files and very new ones may lack JFIF markers ---
the new SPIFF standard mentioned above doesn't use a JFIF marker.
So gripe to your software vendor if you find this to be the problem.)

3. A Macintosh PICT file, if JPEG-compressed, will have several hundred
bytes of header (often 726 bytes, but not always) followed by JPEG data.
Look for the 3-byte sequence (hex) FF D8 FF. The text 'Photo - JPEG'
will usually appear shortly before this header, and 'AppleMark' or
'JFIF' will usually appear shortly after it. Strip off everything
before the FF D8 FF and you will usually be able to decode the file.
(This will fail if the PICT image is divided into multiple "bands";
fortunately banded PICTs aren't very common. A banded PICT contains
multiple JPEG datastreams whose heights add up to the total image
height. These need to be stitched back together into one image.
Bailey Brown has some simple tools for this purpose on a Web page at
http://www.isomedia.com/homes/bailey/photo-jpeg/photo-jpeg.html.)

4. If the file came from a Macintosh, it could also be a standard JFIF
file with a MacBinary header attached. In this case, the JFIF header
will appear 128 bytes into the file. Get rid of the first 128 bytes
and you're set.

5. Anything else: it's a proprietary format, or not JPEG at all. If you
are lucky, the file may consist of a header and a raw JPEG data stream.
If you can identify the start of the JPEG data stream (look for FF D8),
try stripping off everything before that.

At least one release of HiJaak Pro writes JFIF files that claim to be
revision 2.01. There is no such spec; the latest JFIF revision is 1.02.
It looks like HiJaak got the high and low bytes backwards. Unfortunately,
most JFIF readers will give up on encountering these files, because the JFIF
spec defines a major version number change to mean an incompatible format
change. If there ever *were* a version 2.01, it would be so numbered
because current software could not read it and should not try. (One wonders
if HiJaak has ever heard of cross-testing with other people's software.)
If you run into one of these misnumbered files, you can fix it with a
binary-file editor, by changing the twelfth byte of the file from 2 to 1.

Subject: [16] What other common compatibility problems are there?

Aside from the file format difficulties mentioned in the previous section,
there are a few other common causes of trouble with transferring JPEGs.

Old decoders that don't handle progressive JPEG will often give rather
cryptic error messages when fed a progressive JPEG. If you get a complaint
like "Unsupported marker type 0xC2", then you definitely have a progressive
JPEG file and a non-progressive-capable decoder. (See part 2 of this FAQ
for information about more up-to-date programs.) Or you may get a generic
error message that claims the file is corrupted or isn't JPEG at all.

Adobe Photoshop and some other prepress-oriented applications will produce
four-channel CMYK JPEG files when asked to save a JPEG from CMYK image mode.
Hardly anything that's not prepress-savvy will cope with CMYK JPEGs (or any
other CMYK format for that matter). When making JPEGs for Web use, be sure
to save from RGB or grayscale mode.

Photoshop also has a habit of stuffing a rather large thumbnail/preview
image into an application-private segment of JPEG files. Some other
applications (notably early releases of Sun's Java library) are known to

choke on this data. This is definitely a bug in those other applications, but the best available workaround is still to tell Photoshop not to save a thumbnail. If you're putting up an image on the Web, having a thumbnail embedded in it is just a waste of download time anyway.

When transferring images between machines running different operating systems, be very careful to get a straight "binary" transfer --- any sort of text format conversion will corrupt a JPEG file. Actually that's true for all image formats not just JPEG.

Subject: [17] How does JPEG work?

Technical details are outside the scope of this FAQ, but you can find an introduction and references for further reading in the comp.compression FAQ, which is available from http://www.faqs.org/faqs/compression-faq/ (see also "[24] Where are FAQ lists archived?").

The comp.compression FAQ is also a good starting point for information on other state-of-the-art image compression methods, such as wavelets and fractals. A quick comparison: wavelets are likely to be the basis of the next generation of lossy image-compression standards, but they are perhaps 10 years behind JPEG in the standardization pipeline. Fractals have been terribly over-hyped by their chief commercial proponent, and seem to be losing favor as people learn more about their true capabilities and limitations.

Subject: [18] What about arithmetic coding?

The JPEG spec defines two different "back end" modules for the final output of compressed data: either Huffman coding or arithmetic coding is allowed. The choice has no impact on image quality, but arithmetic coding usually produces a smaller compressed file. On typical images, arithmetic coding produces a file 5 to 10 percent smaller than Huffman coding. (All the file-size numbers previously cited are for Huffman coding.)

Unfortunately, the particular variant of arithmetic coding specified by the JPEG standard is subject to patents owned by IBM, AT&T, and Mitsubishi. Thus *you cannot legally use JPEG arithmetic coding* unless you obtain licenses from these companies. (Patent law's "experimental use" exception allows people to test a patented method in the context of scientific research, but any commercial or routine personal use is infringement.)

I recommend that people not use JPEG arithmetic coding; the space savings isn't great enough to justify the potential legal hassles. In particular, arithmetic coding *should not* be used for any images to be exchanged on the Internet. Even if you don't care about US patent law, other folks do.

Subject: [19] Could an FPU speed up JPEG? How about a DSP chip?

Since JPEG is so compute-intensive, many people suggest that using an FPU chip (a math coprocessor) should speed it up. This is not so. Most production-quality JPEG programs use only integer arithmetic and so they are unaffected by the presence or absence of floating-point hardware.

It is possible to save a few math operations by doing the DCT step in floating point. On most PC-class machines, FP operations are enough slower than integer operations that the overall speed is still much worse with FP. Some high-priced workstations and supercomputers have fast enough FP hardware to make an FP DCT method be a win.

DSP (digital signal processing) chips are ideally suited for fast repetitive integer arithmetic, so programming a DSP to do JPEG can yield significant speedups. DSPs are available as add-ons for some PCs and workstations; if you have such hardware, look for a JPEG program that can exploit it.

Subject: [20] Isn't there an M-JPEG standard for motion pictures?

As was stated in section 1, JPEG is only for still images. Nonetheless, you will frequently see references to "motion JPEG" or "M-JPEG" for video. *There is no such standard*. Various vendors have applied JPEG to individual frames of a video sequence, and have called the result "M-JPEG". Unfortunately, in the absence of any recognized standard, they've each done it differently. The resulting files are usually not compatible across different vendors.

MPEG is the recognized standard for motion picture compression. It uses many of the same techniques as JPEG, but adds inter-frame compression to exploit the similarities that usually exist between successive frames. Because of this, MPEG typically compresses a video sequence by about a factor of three more than "M-JPEG" methods can for similar quality. The disadvantages of MPEG are (1) it requires far more computation to generate the compressed sequence (since detecting visual similarities is hard for a computer), and (2) it's difficult to edit an MPEG sequence on a frame-by-frame basis (since each frame is intimately tied to the ones around it). This latter problem has made "M-JPEG" methods rather popular for video editing products.

It's a shame that there isn't a recognized M-JPEG standard. But there isn't, so if you buy a product identified as "M-JPEG", be aware that you are probably locking yourself into that one vendor.

Recently, both Microsoft and Apple have started pushing (different :-() "standard" M-JPEG formats. It remains to be seen whether either of these efforts will have much impact on the current chaos. Both companies were spectacularly unsuccessful in getting anyone else to adopt their ideas about still-image JPEG file formats, so I wouldn't assume that anything good will happen this time either...

See the MPEG FAQ for more information about MPEG.

Subject: [21] What if I need more than 8-bit precision?

Baseline JPEG stores images with 8 bits per color sample, in other words 24 bits per pixel for RGB images, 8 bits/pixel for grayscale, 32 bits/pixel for CMYK, etc. There is an extension that stores 12 bits/sample for applications that need higher accuracy. Medical images, for example, are often 12-bit grayscale. The 12-bit extension is not very widely supported, however. One package that does support it is the free IJG source code (see part 2, item 15).

For lossless JPEG, the standard permits any data precision between 2 and 16 bits per sample, but high-precision lossless JPEG is even less widely supported than high-precision lossy JPEG. The Stanford PVRG codec (see part 2, item 15) reportedly supports up to 16 bits/sample for lossless JPEG.

Subject: [22] How can my program extract image dimensions from a JPEG file?

The header of a JPEG file consists of a series of blocks, called "markers". The image height and width are stored in a marker of type SOFn (Start Of Frame, type N). To find the SOFn you must skip over the preceding markers; you don't have to know what's in the other types of markers, just use their length words to skip over them. The minimum logic needed is perhaps a page of C code. (Some people have recommended just searching for the byte pair representing SOFn, without paying attention to the marker block structure. This is unsafe because a prior marker might contain the SOFn pattern, either by chance or because it contains a JPEG-compressed thumbnail image. If you don't follow the marker structure you will retrieve the thumbnail's size instead of the main image size.) A profusely commented example in C can be found in rdjpgcom.c in the IJG distribution (see part 2, item 15). Perl code can be found in wwwis, from http://www.tardis.ed.ac.uk/~ark/wwwis/.

Subject: [23] Where can I learn about using images on the World Wide Web?

If you want to display still images on the World Wide Web, you have a choice of using JPEG or GIF; those two formats are by far the most widely supported by WWW browsers. (We can hope that PNG will soon become popular enough to replace GIF on the Web; see http://www.cdrom.com/pub/png/ for PNG info.) For most images it's pretty obvious which format to choose (see "[3] When should I use JPEG, and when should I stick with GIF?"). JPEG's ability to trade off file size against image quality is especially helpful for trimming download times of Web photos.

But there's a good many things to know that are specific to Web design, and even specific to the currently-most-popular browsers. This FAQ doesn't try to cover Web graphics design. Good basic information can be found at:
http://www.boutell.com/faq/
http://www.servtech.com/public/dougg/graphics/index.html
http://www.webreference.com/dev/graphics/
http://www.adobe.com/studio/tipstechniques/GIFJPGchart/main.html
http://ppewww.ph.gla.ac.uk/~flavell/www/palette.html

http://the-light.com/netcol.html
and here are some sites with more advanced info:
 http://www.inforamp.net/~poynton/Poynton-colour.html
http://www.photo.net/philg/how-to-scan-photos.html
http://www.scantips.com/

Subject: [24] Where are FAQ lists archived?

Many FAQs are crossposted to news.answers. Well-run netnews sites will have the latest versions available in that newsgroup. However, there are a *lot* of postings in news.answers, and they can be hard to sort through.

The latest versions of news.answers postings are archived at rtfm.mit.edu. You can retrieve this FAQ by FTP as rtfm.mit.edu:/pub/faqs/jpeg-faq/part1 and rtfm.mit.edu:/pub/faqs/jpeg-faq/part2. If you have no FTP access, send e-mail to mail-server@rtfm.mit.edu containing the lines
 send faqs/jpeg-faq/part1
 send faqs/jpeg-faq/part2
(If you don't get a reply, the server may be misreading your return address; add a line such as "path myname@mysite" to specify your correct e-mail address to reply to.) For more info about the FAQ archive, retrieve the file rtfm.mit.edu:/pub/faqs/news-answers/introduction.

The same FAQs are also available from several places on the World Wide Web, of which my favorite is http://www.faqs.org/faqs/.
This FAQ is http://www.faqs.org/faqs/jpeg-faq/.
Other popular WWW FAQ archives include http://www.cs.ruu.nl/cgi-bin/faqwais and http://www.lib.ox.ac.uk/internet/news/.

--
tom lane
organizer, Independent JPEG Group
tgl@netcom.com or tgl@sss.pgh.pa.us

Part1 - Part2 - MultiPage

Related questions and answers

[Usenet FAQs | Search | Web FAQs | Documents | RFC Index]

Send corrections/additions to the FAQ Maintainer:
jpeg-info@uunet.uu.net

Last Update June 15 2004 @ 00:27 AM