JPEG Compression of Still Images

Douglas A. Kerr, P.E.

Issue 1
August 16, 2003

ABSTRACT

A technique known as JPEG is widely used for the compression of digital data representing photographic still images. In this article, we explain how this technique works. Appendixes give tutorial insight into several technical concepts that are involved.

1. INTRODUCTION

Digital representations of photographic still images, especially when a fairly-high resolution is involved, constitute large sets of data ("image files"). Practical considerations of storage of these files and their transmission over data networks make it highly desirable, if possible, to replace the original data in the files with another file, smaller in size, from which the original image can nevertheless be reconstructed with an acceptable degree of fidelity. Such a process is referred to as *image data compression*.

A compression technique known as JPEG is widely used for this purpose. In the mode of most interest to us it provides non-reversible ("lossy") compression: the set of data recovered at the "destination" is not identical to the original set of data.[1] Thus the image represented by the recovered data is not identical to the original image. The hope is that the discrepancy in the destination image, as perceived by the human viewer, will not be noticeable, or in any event will not interfere to an unacceptable degree with the purpose to which the image is to be put.

JPEG is an acronym for Joint Photographic Experts Group, a working party operating under the joint auspices (hence the name) of the international standards bodies CCITT (now, ITU-T) and ISO/IEC. This group developed and codified the JPEG standard.

---

[1] For a discussion of the concept of reversible ("lossless") and non-reversible ("lossy") compression, see Appendix B.

2. TWO LAYERS

Actually, two standards are involved in the application of JPEG in which we are interested here. The JPEG standard itself defines the actual data compression algorithm. It is quite flexible, and can readily be used for original digital images in many different forms (differing, for example, in the color model used). However, if we are to be able to freely take a file containing an image whose data is compressed in JPEG form and actually display or print the image, we must know those other particulars of the original image.

This is the job of the second standard, JFIF—the JPEG File Interchange Format specification. It defines those particulars of the original image. Thus the "JPEG" files —typically with filetype extension "JPG" —are actually JFIF files.[2]

3. OPERATION OF JPEG COMPRESSION

3.1 General

In this section, we will describe, in a fairly-technical way, the operation of the JPEG compression algorithm. The discussion will presume the most-common modes and options.

3.2 Source data

We will assume that the source image is in the RGB color model, that is, there is an R, G, and B value for each pixel.

3.3 Gamma precompensation

The first step is to replace R, G, and B with their gamma-precompensated equivalents, R′, G′, and B′.[3] Conceptually, that is done by applying the following relationship:

$$X' = X^{0.45} \quad \text{or} \quad X' = \sqrt[2.2]{X}$$

where X represents R, G, or B, and the range of both X and X' is from 0 to 1.

---

[2] Often the JPG files we encounter aren't true JFIF files, not employing certain optional ingredients prescribed for JFIF files, but nevertheless honor the original image definitions of the JFIF standard so that we can view the images without difficulty.

[3] For a discussion of the concept of *gamma precompensation*, see the companion article by the same author, *Gamma in Film, TV, and Digital Still Camera Systems*.

In reality, the expression is a bit more complicated, as it provides for a proportional, not power, relationship for the lower values of R, G, or B. This is done to optimize overall perceptual performance in the face of noise components in the original camera outputs.

3.4 Conversion to the Y′ CbCr color model

For each pixel, the value in Y′ CbCr (luminance-chrominance) form is determined. The luminance value, Y' is calculated by multiplying the values of R', G', and B' by predetermined constants and summing the products. Then color-difference values (defining chrominance) are determined this way:

$$Cb = B'-Y'$$

$$Cr = R'-Y'$$

The values Y', Cb, and Cr are represented in 8 bits, with a range of 0 to 255.

3.5 Decimation (subsampling) of the chrominance information

The human eye is more sensitive to fine detail presented as changes in luminance than to changes presented as changes in chromaticity. We can take advantage of this by conveying the chrominance information at a lower resolution than the basic resolution of the image. We do this by not actually conveying the original chrominance information (Cb, Cr) for each pixel but rather only one Cb, Cr pair for each four pixels.[4] This could be done by just discarding three out of each four Cb, Cr pairs. A better result is achieved by replacing the Cb, Cr pairs for four pixels with a new Cb, Cr pair derived from the four Cb, Cr pairs (or even from a larger span of Cb, Cr pairs). In any case, the process is referred to as *decimating* (or *subsampling*) the chrominance information.

3.6 Organizing the pixel information into 8 x 8 blocks

The pixel information is now organized into 8 x 8 blocks of contiguous pixels to prepare for the next step. Each block in effect consists of three 8 x 8 matrixes (three layers), one each for the Y′, Cb, and Cr values.

---

[4] An alternative is to convey one Cb, Cr pair for each two pixels.

3.7 Determining the discrete cosine transform (DCT) for each of the 8 x 8 matrixes

Each of the 8 x 8 matrixes carries 64 values (of Y', Cb, or Cr) in spatial form. The plot of values along a horizontal path, or a vertical path, is like a waveform.

The *discrete cosine transform* represents each of these waveforms as a sum of cosine waveforms of 8 different spatial frequencies, the first having a frequency of 0 (a constant, or "DC", component), the second having a frequency half the *fundamental frequency*, which is a frequency of one cycle per the width of the block (8 pixels), and the remaining six having "harmonic" frequencies of 2 through 7 times half the fundamental frequency.[5]

The transform result contains 64 *coefficients*, each of which represents the amplitude of a cosine component at one of those frequencies in the horizontal direction and a cosine component at one of those frequencies in the vertical direction. For example, one coefficient gives the amplitude of a "3rd harmonic" cosine waveform in the horizontal direction and a "5th harmonic" cosine waveform in the vertical direction.

These coefficients are considered to have a range of -1 to +1, but are represented as 1023 times the actual value (-1023 to +1023), expressed in sign-magnitude form as 11 bit numbers.

3.8 Quantizing the sets of DCT coefficients

In order to reduce the amount of information to be conveyed, we take each of the 64 coefficients in each of the three sets, divide it by a constant, and round it to the nearest integer, reducing its value and allowing it to be represented in a smaller number of bits (with correspondingly less precision). This spoken of as "quantizing"; it is actually "requantizing" the coefficients, as they were already in discrete (quantized) form. It is equivalent to rounding the number and scaling down the result. [6]

---

[5] For a detailed discussion of the principles of the discrete cosine transform, see Appendix ***.

[6] We're not used to think of rounding by division. As an illustration of why this works, suppose that in a decimal environment, using only integers, we start with the value 14659 and divide it by 100. The result (to integer precision) is 147. If we know at the receiving end that we divided it by 100, we know that this really means

The quantizing constants differ among the 64 coefficients, and are prescribed by two separate tables, one for the quantizing of the coefficients derived from the Y′ matrix and one for the quantizing of the coefficients derived from the Cb and Cr matrixes.

The pattern in the table is one in which the constants are higher (leading to a more severe "rounding") for the coefficients representing the higher-spatial-frequency components. The higher frequency components are less important to the eye's ability to interpret the image that will eventually be reconstructed, and thus we can afford to round them more severely in the interest of reducing the amount of data in the file.

It is this quantizing process that is responsible for most of the reduction in the size of the compressed image file.

In most cases, owing to the typically-smaller range of amplitudes of the higher-frequency components before quantizing, many of the higher-frequency coefficients will be driven to zero by the quantizing process. In a later step, we will see that, through clever coding, only a tiny amount of information is required to convey a bunch of zero values.

When we choose a "quality" level for JPEG compression, the encoder scales the quantization constants up or down by the same amount for all entries in both tables. The standard set of values is associated with "50% quality". If we choose a "lower" quality value, these constants are all bumped up from those standard values, and thus all coefficients are given a more stringent rounding (requiring fewer bits to convey) and more of them will be driven to zero (requiring **really** few bits to convey).

3.9 Reordering the DCT coefficients

For each of the three sets of 64 DCT coefficients (for Y′, Cb,. and Cr), the coefficients are now put in a sequence described by a zig-zag path across the DCT output matrix. The purpose is to put the coefficients describing the higher-frequency components (the ones most likely to be driven to zero during quantization) into adjacent positions, in preparation for the next step.

14700. Thus we have in effect rounded the number to the nearest 100, but did it in such a way that we can convey it in only 3 bits instead of 5.

3.10 Run-length encoding of the zero-value coefficients

After the reordering in the previous step, we will typically have long strings of consecutive 0 values. We can convey these economically by merely stating, in effect, for example, "13 zeroes next", a concept described as *run-length encoding* (RLE). Four bits are used to tell the number of zeroes in a run, allowing up to 16 consecutive zeros to be thus represented. If there are more than 16 in a row, we say, in effect, for example, "16 zeros next, 16 zeros next, 7 zeros next."

3.11 Recognition of the variable significant length of the non-zero coefficient values.

The values of the coefficients, (both before and after quantizing) are stated in an 11-bit sign-magnitude form, the rightmost bit indicating the sign. The value can range from -1023 to +1023. If a coefficient, after quantizing, has a value of +3, it can be represented in 3 significant bits: 111 ("11+"). However, if it has a value of −214, it requires 9 significant bits: 110101100 ("11010110 −").

If we take advantage of this, and only include significant bits (strip all leading zeroes) in the bitstream, we can reduce the average number of bits needed to represent a non-zero coefficient. However, to do this, we must know how many bits are involved, so the decoder will know how many of the consecutive bits form one value. To allow this, we prefix the actual value with a four-bit "length" indicator.

3.12 Forming the bit stream

Every non-zero value is considered to be preceded by a string of zero values, whose length may, however, be zero (if in fact a non-zero value immediately follows another one).

The bit stream at this point is built up of "phrases", one for each non-zero value, made up this way:

- 4 bits telling the number of zero values preceding this non-zero value (could be zero)
- 4 bits (the "length indicator") giving the number of bits representing the non-zero value
- That number of bits, representing the non-zero value.

3.13 Huffman coding of the prefix

If we look at the first 8 bits of the above phrase (the "prefix"), we find that not all 256 possible values occur with equal probability. To

exploit this, we code that 8-bit group using Huffman coding.[7] Under this scheme, the 8 bit group is represented with a variable length code, sometimes involving more than 8 bits, but most often involving substantially less than 8. Thus the bitstream is further reduced in size. No further prefix is needed to identify the length of the Huffman code word; its length can be determined from its structure (certain bits of the codeword are in effect devoted to this function).

Since so many of the non-zero code words are small, their average length is also small, and so the 8-bit prefix byte for each represents a substantial fraction of the entire bit stream at this point. Thus, the reduction in the average size of the prefix through Huffman encoding yields a significant further reduction in the overall size of the data set.

3.14 Put it all together

Each of the bitstreams formed as described above (one for Y' , one for CB, and Cr) is packed into bytes, and the larger-scale structure of the compressed data set is assembled.

3.15 Encapsulate it

The entire JPEG-compressed data set is now provided with an appropriate overall header and other housekeeping fields according to the JFIF specification, yielding our JPG file.

---

[7] For a discussion of Huffman coding, see Appendix ***.

APPENDIX A

Understanding the Discrete Cosine Transform

INTRODUCTION

The discrete cosine transform (DCT) is a way of representing a "waveform" which exists in the time or space domain with an alternative representation in the frequency domain. A waveform represented in this way can be manipulated in ways which are advantageous in such matters as data compression.

In this appendix, we will describe the DCT and explain how it works.

BACKGROUND

First, for the benefits of readers who may not be familiar with some of the underlying technical concepts, we will review a number of those concepts.

Frequency

If we have an electrical "signal", a plot of its instantaneous voltage against time is called the *waveform* of the signal. If that waveform repeatedly follows a certain pattern, indefinitely, it is said to be a *periodic* waveform. The rate at which the pattern repeats is said to be the *frequency* of the waveform. It is usually measured in units of cycles per second, a unit for which there is a special name, the *hertz* (abbreviated Hz)[8].

In the strict sense, the concept of **a** frequency for a waveform only applies to waveforms of a certain shape, one which is identical to the plot of the trigonometric sine of an angle plotted against the angle. Such waveforms are said to be *sine waves*, or to be *sinusoidal*.

Interestingly enough, when sinusoidal waveforms are represented mathematically, the representation is most often not in terms of the *sine* function but rather its cousin the *cosine* function. The cosine

---

[8] The unit is named in honor of Heinrich Hertz, the noted German physicist. According to the protocols of the International System of Units (the codification of the modern "metric system"), a unit based on the name of a person is not capitalized (each is, after all, a unit, not a person's name) but the abbreviation is capitalized!

function has the same shape as the sine function, but starts 90° earlier in terms of angle. (The waveform represented, though, is nevertheless spoken of in most cases as a "sine wave".)

Spatial frequency

We have so far been talking about waveforms that occur in time. Waveforms can also occur in space.

Suppose we follow a path across the width of a piece of corrugated roofing and plot its height (above some reference surface) against the distance along the path. This plot is also a waveform, but we say it is in the *space domain* rather than the *time domain*.

If the pattern of height repeats, it is a periodic waveform, and thus can be said to have a frequency. In this case, the frequency is expressed in units of cycles per foot (or some other unit of length).

Such a frequency is said to be a *spatial frequency* ("spatial" meaning "pertaining to space"). By contrast, the type of frequency we discussed earlier can be called a *temporal frequency* ("temporal" meaning "pertaining to time"). When frequency is mentioned without a modifier such as temporal or spatial, by convention it is temporal frequency that is meant.

The Fourier series

If we have a period waveform that is not sinusoidal, it can be shown that it may be considered as made up of the sum of two or more "component" sine waves, having frequencies that are integer multiples of the rate of recurrence of the waveform, each one having a certain amplitude (size) and phase angle (a property that describes how the waveform's time reference compares to the time reference of the other component waveforms).

The component whose frequency is the rate of recurrence of the waveform is said to be the *fundamental* component, and its frequency is said to be the *fundamental frequency* of the waveform. The components whose frequencies are higher integer multiples of the fundamental frequency are said to be *harmonics*, and their frequencies are said to be *harmonic frequencies*.

There may also be a component at a frequency of zero, called the *DC component*. If that component were plotted, it would be a straight horizontal line, representing a constant voltage, just like a direct-current (thus its name). The value of this component is actually the average of the value of the waveform over its cycle.

The representation of an arbitrary periodic waveform by a set of sinusoidal components of harmonically-related frequency is called the *Fourier series* representation of the waveform.

We say that the waveform itself exists in the *time domain*, but its description as a Fourier series is in the *frequency domain*.

In the case of certain waveforms, especially those with "sharp corners" (such as the so-called "square wave"), the Fourier series representation comprises an infinite number of components, with frequencies running to infinity.

Note that these concepts are equally applicable to waveforms existing in the time domain or the space domain. The units of the frequency scales of course differ between the two situations.

The Fourier transform

If we have a waveform which does not repeat the same pattern forever (actually, the mathematical requirement is that it also has been repeating since the infinite past!), or for which we only have knowledge over a limited time period, we cannot represent it with a Fourier series. (Since the components of a Fourier series go on forever, the waveform that they collectively represent would go on forever, and that isn't what we have here!)

We can, however, represent such a waveform in the frequency domain by means of its *Fourier transform*. That is a plot, against frequency, of the content of the waveform in terms of cosine waves of different frequency. Here, these components aren't evenly spaced in frequency, as in the case of a Fourier series representation of a periodic waveform. In fact, in the typical case, they are "continuous" over a certain range—there are an infinite number of components, spaced infinitely closely together in frequency, and perhaps extending to infinite frequency as well.

What about the phase of the component cosine waves? It is important here as it was for the Fourier series representation. In fact, a complete Fourier transform representation of a waveform consists of two plots against frequency, one showing the variation in amplitude of the components and one showing the variation of phase.

The discrete Fourier transform

The waveforms we have been presuming so far are "continuous"—their instantaneous voltage is defined for any instant of time we can

imagine (over the "time window" for which we have visibility of the waveform).

In digital technology, we have waveforms in discrete form, their amplitude described to us only at periodic instants. This is often described as a "sampled" representation.

In this setting, the job of the Fourier transform in giving us a "frequency domain" representation of a waveform we know in the time (or space) domain is taken over by its cousin, the discrete Fourier transform (DFT). Its result is a series of coefficients, each describing the amplitude and phase of a sinusoidal component at an integer multiples of the "fundamental frequency", the frequency such that one cycle of it would last as long as the length of the waveform sequence. (The first of those, as with the Fourier series, is value of the "zero-frequency", or DC, component.)

If we have a signal window embracing 16 samples of the original waveform, the DFT will consist of 8 components, the DC component and sinusoidal components at frequencies of 1 through 7 times the "fundamental frequency". In general, each component is represented by two coefficients (describing amplitude and phase). For the "O" (DC) component, phase is not meaningful. For the top component, the phase is always 0, and no coefficient for its phase is required. Thus there are a total of 16 coefficients in the transform.

For this to work well, it is necessary that the length of the sequence (the number of sample values describing it) be an integral power of two (2, 4, 8, 16, etc.). The number of coefficients will always be the same as the number of points in the sequence.

THE DISCRETE COSINE TRANSFER

A close relative to the discrete Fourier transform is the *discrete cosine transform* (DCT). It is used in the same circumstances as the DFT. It differs in that the coefficients do not describe both the amplitude and phase of the individual harmonically-related components, but rather the amplitude only of components of fixed phase, about twice as many of them as in the case of the DFT.

If we have a waveform sequence embracing 16 samples of the original waveform, the DCT will consist of 16 coefficients (just as in the case of the DFT). One describes the amplitude of the DC (zero-frequency) component and 15 describe the amplitude of components with frequencies ranging from 1 to 15 times the "fundamental frequency". In this case, the fundamental frequency is

the frequency for which one cycle is twice the length of the sequence. No coefficients are required to indicate the phase of the components: all have a fixed phase. The fixed phase is not zero; it is in fact positive and its size is half the spacing of the original sample points. As a result, the reference point of each of the cosine waves (its first "1" value) is "one-half tick" before the beginning of the sequence being transformed.

The 2-dimensional discrete cosine transform

We have so far talked in terms of a "waveform sequence" which might be part of a horizontal "pass" across a digital image[9]. But our image is actually two-dimensional. Patterns of pixel values often change in consistent ways vertically and horizontally. We can exploit this by using a "2-dimensional" discrete cosine transform.

Suppose now that we have a "window" on the image that is 8 pixels wide and 8 pixels high – 64 values altogether. We can apply the 2-dimensional DCT to this "block" of values. The output of the transform is 64 coefficients. Each of these coefficients tells us the amplitude of two cosine wave components: a component at one frequency pertaining to the overall change in pixel values as we move from left to right across the block, and a component at one frequency pertaining to the overall change in pixel values as we move from top to bottom across the block. The frequencies of the components include zero (the DC component) and frequencies of from 1 to 7 times the fundamental frequency.

A particular coefficient, for example, tells the amplitude applicable to both a cosine-wave component at, say, 3 times the fundamental frequency, applicable to the horizontal patterns of pixels in all 8 rows, and to a cosine-wave component at, say, 6 times the fundamental frequency, applicable to the vertical patterns of pixels in all 8 columns.

With this approach, the 2-dimensional DCT of an 8x8 pixel array from an image (64 points) will have 64 coefficients. If we had treated the 64 original pixels as 8 strings of 8 pixels, each with a separate (one-dimensional) DCT, each DCT would have 8 coefficients, a total of 64.

What then is the advantage of the 2-dimensional approach. If the transform is used on image data (likely as a step of an image data compression system, such as JPEG), then for typical data:

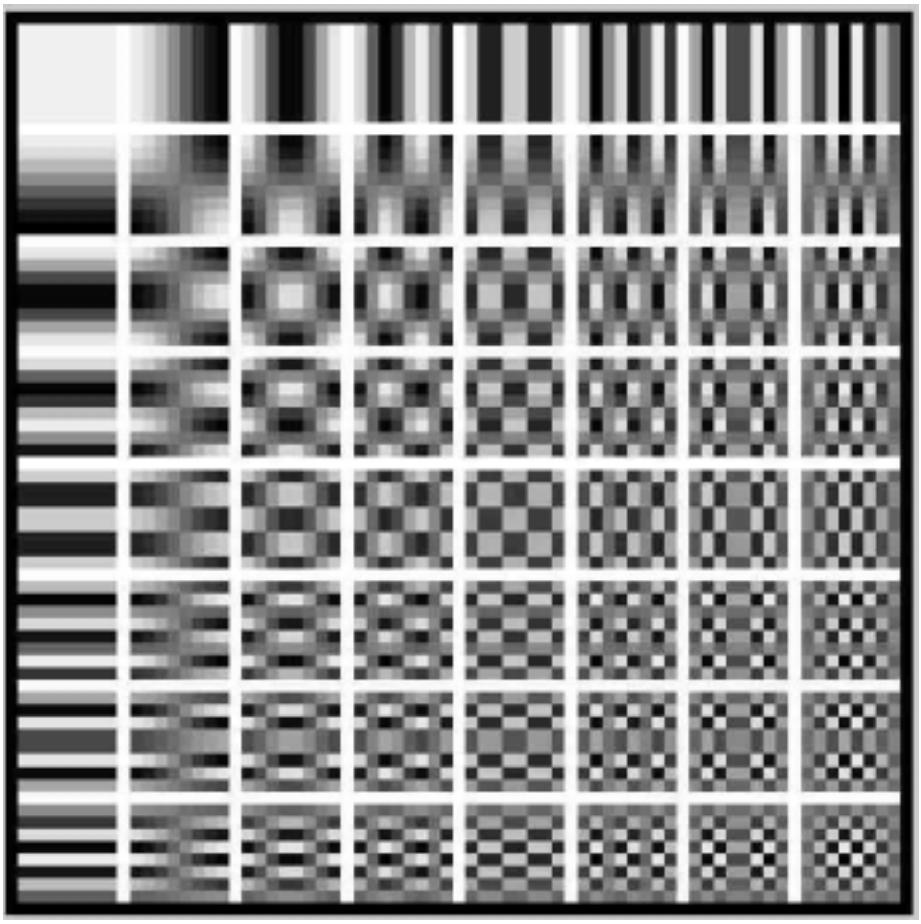
---

[9] For one particular property of the pixels, for example Y' (luminance)

- a greater fraction of the 2-dimensional DCT coefficients will be smaller than for the coefficients of eight 1-dimensional DCTs
- a greater fraction will be amenable to significant requantizing (so that they can be acceptably represented in fewer bits)
- the coefficients that become zero when requantized will "cluster" better within this larger body of coefficients, allowing them to be compactly represented through run-length encoding

All of these lead to improvements in the degree of data compression that is achieved by the JPEG compression system.

This figure shows the patterns which are implied by each of the 64 DCT coefficients:



Each little figure shows, for each of the 64 coefficients, the pattern, vertically and horizontally, that would be given to the 8x8 pixel grid in the reconstructed image if that particular coefficient had its maximum value (+1) and the other coefficients were zero.

Note that although the underlying vertical and horizontal patterns in these 2-dimensional patterns are described as *cosine waves*, they actually only exist at 8 discrete points across the span of the grid, one for each pixel position. One result of this is that for the patterns with frequencies of 1, 2, and 4 times the fundamental frequency[10] (for example, top row, patterns 2, 3, and 5 from the left), we can easily see the underlying cosine wave; for patterns with frequencies at other multiples of the fundamental frequency, the underlying cosine wave is hard to visualize. For example, the cosine nature of the rightmost pattern in the top row, at 7 times the fundamental frequency, is hard to visualize when we only see its value at intervals of 7/16 of its period!

#

[10] Recall that in this case the fundamental frequency is that for which one cycle lasts twice the width of the block; that is, the block spans one half cycle of the fundamental frequency.

APPENDIX A

Principles of Data Compression

INTRODUCTION

Compression, in the sense of interest here, refers to taking a body of data and replacing it with another smaller body of data for storage or transmission such that the original data can later be reconstructed, either precisely or not. An important application of this concept is in the field of digital still photography, where substantial reductions in the size of the file originally generated to capture an image are critical to practical operation. Here we review the concepts and principles of data compression, and give a brief overview of the approach used for digital still images.

REVERSIBLE COMPRESSION

With reversible compression, a body of data is replaced with a smaller body of data from which the original body can later be reconstructed precisely. Compression of this type is mandatory for data files that carry computer software. Even a single bit changed in the reconstructed file can make the file unusable. Such compression is also usually demanded for computer document files (word processing documents, spreadsheets, engineering drawing graphic files, and so forth).

Today, reversible compression is usually called "lossless compression", a term which I consider unfortunate. I'll talk about that in a separate section of this article.

How is such compression possible? It exploits *redundancy* in the original body of data (which is often called the *message*, a term that we will continue to use here) in a specific technical sense: redundancy is the degree to which the data in the message departs from having the same statistical properties as would a random stream of 1's and 0's. For example, in a truly random data stream there will be equal numbers of 1's and 0's; over all pairs of bits (adjacent or not) there will be equal numbers of "00", "01", "10", and "11"; over all groups of three bits (adjacent or not) there will be equal numbers of "000", "001, "010", and so forth.

Huffman coding

The compression system devised by D.A. Huffman exploits that situation. It uses the following principle.

Suppose we are talking about text files carrying general text in the English language. Each character is represented by a fixed 8-bit code word. We know that, averaged over a lot of text, the character "e" will occur substantially more frequently than the character "Z", and that substantially more frequently than the character "Ð" (*thorn*) (used in Icelandic words).

Suppose that, instead of giving each of the characters a fixed 8- bit representation, we give them code words of different lengths. We give the most-frequently-appearing characters short code words (shorter than 8 bits) and the less-frequently-appearing characters longer code words (often longer than 8 bits). As a result, the average number of bits per character will be less than 8.

Huffman' s technique gives us an orderly procedure for designing such a code. The code is defined in "tree" form. The decoder needs no special clues to determine how many bits of the data stream make up the code word being received—as each bit is encountered, the decoder progresses through the tree until it encounters an "end node", which corresponds to one of the characters, which is noted. The next bit is of necessity the first bit of the next code word.

Suppose, however, that in a particular application, the distribution of characters is not consistent with the distribution in English text (the data may in fact not represent text at all), and may in fact vary from message to message within the application.

If nevertheless, in any particular message, certain characters appear more frequently than others, we can still use our technique by adding a new feature. When a message is submitted for compression, we first survey it to determine how many times each character occurs (if at all). We then design a custom recoding scheme, again with the most-frequent characters (for this message) given the shortest codes.

So that the "receiver" can decode this message, we put a description of this particular code scheme in a table at the beginning of the recoded message. Although that adds to the overall length of the file, especially for longer messages (for which the table is no longer) the dilution of compression effectiveness is minimal.

Run-length encoding

In many types of data, we regularly encounter long strings of the same value. Here, a compression approach called *run-length encoding* can be useful.

For convenience, let us assume a data sequence in which the elements can be thought of as characters. Suppose we have this sequence:

aEEEEEEEEkkkMMMMMpppppppZZZ

We could describe it this way, verbally:

One “a”, eight “E”s, three “k”s, five “M”s, seven “p”s, three “Z”s.

In coded form, we might state:

1a8E3k5M7p3Z

a much more compact representation of the original sequence.

NON-REVERSIBLE COMPRESSION

Especially when the “message” does not have much redundancy in the simple sense we discussed above (such as digital image files), reversible (“lossless”) compression may not yield much reduction in message size. But in many such cases we can achieve substantial compression through the use of non-reversible (“lossy”) compression systems. In such systems, the data set recovered at the destination is not identical to the original data set. But (in the case of use for image encoding) perhaps the image represented by the delivered data differs from the image represented by the original data in a way that will hardly be noticed by the receiving viewer.

There are numerous approaches to non-reversible compression. In the case of data streams representing speech (as in wireless telephone systems), we replace the original verbatim description of the speech waveform with what amounts to a script that controls a speech synthesizer at the destination. While the waveform delivered by the synthesizer may not appear anything like the original waveform, it will nevertheless be perceived by the recipient as sounding much like the original speech.

For digital photographic images, we typically replace the original data with another form that represents the same image. In that form, we can reduce the precision with which certain factors are expressed,

reducing the number of bits in a way that impacts the delivered image in a benign way from the perspective of the viewer. The JPEG compression system, which is the subject of this paper, operates upon that principle. Its details are covered in the body of the paper.

"REVERSIBLE/NON-REVERSIBLE" —"LOSSLESS/LOSSY"

It is today common to speak of reversible compression processes as "lossless" and non-reversible ones as "lossy". The basis of this notation is the notion that, with non-reversible compression, a part of the original information is "lost" from the reconstructed file.

I feel that this notation is misleading as to what happens to information in non-reversible compression. Information is not lost—it is errored.

If in fact information were being lost, we should be able, at least conceptually, to quantify how much is being lost. Suppose we begin with a file representing a 3 million pixel full-color image. For each pixel, there are three data values, for Y', Cb, and Cr—a total of 9 million values.

After encoding the file under JPEG and then decoding it, we again have 9 million values. How many of those are the same as the corresponding original values? Unless the image has some large areas of "full black" or "full white", likely less than 0.5% of the values. If values that are errored are considered "lost", then we have lost over 99.5% of the information. Seems hardly worth storing the file!

"But", you may say, "in most cases, the discrepancy will be small. Can't we take that into account?" Absolutely. There are well-accepted statistical concepts for assessing errors in a body information. The result would be reported in terms of *RMS error*. But *RMS error* isn't a measure of loss of data; it is a measure of—error.

The adoption of the "loss" notion is understandable from the viewpoint of one whose perspective is the inner details of many compression systems. For example, in the JPEG compression system, we quantify the DCT coefficients, in effect discarding some of the lower-significance bits from the values.[11] Isn't that "loss of information"? I suppose so. But remember, when we describe compression systems, we are trying to characterize the overall result

---

[11] Actually, it isn't done exactly that way, but the description is close enough to be valid for our purpose here.

on the reconstructed image (all that matters to the user), and as I just pointed out, talking there about "loss of information" just doesn't fit.

Another area which might seem to justify the use of the "loss" notion is the *decimation* (or *subsampling*) of chrominance data in JPEG. We do not carry forward the chrominance information (Cb, Cr) for each pixel in the image. Rather, we commonly only carry forward one set of chrominance information for each four pixels.

If we did this in an "agricultural" way, we would take the chrominance pairs for four pixels and just discard three of them. Surely that constitutes a loss of information.

Indeed. But we rarely do it that way. Instead, for each four pixels, we encode a single chrominance pair derived from the values from four pixels, or even from a larger group, following some interpolation algorithm. None of the original chrominance pairs are included in the compressed file. When the image is decoded, there will be chrominance pairs for each pixel. None of them will (except by chance) be the same as in the original image. So how much of the chrominance information have we "lost"? Evidently all of it!

Thus we see that the notion of the "loss" of information under compression, while catchy, is not meaningful. Of course, the terminology is well established, and will certainly prevail. But it is important that we do not take its implications literally.

#