

# Chapter 20

## Image Compression

### 20.1 References for Image Compression

Many WWW Sites on Compression – millions on *Google*.

*Key Papers in the Development of Information Theory*, IEEE Press, 1974

Raisbeck, **Information Theory for Scientists and Engineers**

Ash, **Information Theory**, Dover

Frieden, **Probability, Statistical Optics, and Data Testing**, Springer-Verlag, 1981.

Khinchin, **Mathematical Foundations of Information Theory**, Dover, 1957.

Pierce, J.R., **An Introduction to Information Theory, Signals, Systems, and Noise**, Dover

Jain, A.K., “Image data compression: a review”, **Proc. IEEE**, **69**, 349-389, 1981

Rabbani, M. and P.W. Jones, **Digital Image Compression Techniques**, SPIE, 1991

Abramson, N, **Information Theory and Coding**, McGraw-Hill, 1963.

Dainty, C. and R. Shaw, **Image Science**, Academic Press, 1974, sec 10.

Grant, R. E., A.B. Mahmoodi, and W.L. Nelson, Image “Compression and Transmission”, sec 11 in **Imaging Processes and Materials**, Neblette’s 8th edition, Van Nostrand Reinhold, 1989

### 20.2 Image Storage, Transmission, and Compression

All of you probably have an intuitive feeling for the fundamental difficulties of storing and transmitting information; the well-known phrase “*one a picture is worth 1024 words*” is an example. The subject has been quite important in imaging as far back as the early days of television development. It has become increasingly important as the digital juggernaut has taken hold. For comparison, consider the recent history of “imaging” of an audio music signal. The goal of the high fidelity industry for many years has been to record and play back ever more of the music signal by increasing the

bandwidth of the recorded signal to reproduce the sound as realistically as possible. In other words, the industry has attempted to increase the amount of recorded “information” to increase the “fidelity” of the reproduced signal. However, the recording industry has been revolutionized in the last few years (since before the turn of the millenium) by the influx of digital recording technology. The goal now is to discard as much information as possible (and thus “compress” the signal) without degrading the “message” conveyed by the sound. The goal is to increase the quantity of music recorded on a device with fixed data capacity, thus reducing the cost of storing and conveying the message. This is only possible because digitally recorded signals can be manipulated in ways that are not available for analog recordings.

The property of messages that allows them to be compressed is that some portion of the data is not essential to understanding of the message. For a simple example, consider the text message:

*Th qck brn fx jmpd ovr th lz dg*

which is perfectly readable (at least to most of us for whom English is their first language) but which requires only 75% of the letters in the complete text. The moral of this example is that vowels are (usually) unnecessary to the meaning of text messages. In general, vowels supply *redundant* information in the message and often may be discarded without loss of meaning. In an imaging sense, we might say that vowels supply low-frequency information (bulk properties of the words), while consonants carry the high-frequency information (information about the “edges”). Redundancies in text messages or images may be identified and eliminated using different but analogous schemes. The data compression process often is called coding, because the representation is altered and may require a reference source (codebook) to interpret the message in terms of the original symbols. The name coding also implies a relationship between the processes of message compression and message cryptography, which are (in a sense) opposites. Compression is the process of removing nonessential information from the message to reduce the quantity of data; cryptography is the process of adding nonessential information to different messages to make them look indistinguishable.

The principles of image compression require an understanding of the fundamental concepts of information theory, which were laid down by Claude Shannon in the late 1940s, with later contributions by many authors, notably Norbert Wiener and A.I. Khinchin. The technologies (hardware and software) were very hot topics in the 1980s because of limitations in storage space (as difficult as it may be to believe now, a common size for a PC hard drive was 30 MBytes in the mid-late 1980s). The topic heated up again in the late 1990s for internet applications because of limitations in transmission bandwidth.

There are (at least) two classes of data redundancy: *objective* and *subjective*. Objectively redundant data can be removed without any loss of information. In other words, deleted objectively redundant data may be recovered without loss. Subjectively redundant data may be removed from a message or image without any “visible” loss of information, though the original image cannot be recovered without error. By

consequence, we can divide information compression into three classes: objectively lossless, subjectively lossless, and subjectively lossy. The message may be recovered perfectly after encoding with an objectively lossless algorithm (e.g., run-length compression or Huffman coding). The message may be recovered with no apparent loss if encoded with a subjectively lossless algorithm (e.g., JPEG encoding with a good quality factor). Images encoded with subjectively lossy algorithms have visible artifacts, but require significantly less transmission or storage capacity.

In this discussion of image compression for storage and transmission, we will first relate the concepts of image representation by electronic signals, both analog and digital. This will introduce the concepts of analog signal bandwidth and the limitations of real systems. We will then review Shannon's basic description of information to introduce the concepts of image entropy and channel capacity, which will be related to analog bandwidth. After that, we will describe the definitive method developed by Huffman to encode an image to reduce the quantity of information in a message with a specific number and probability distribution of characters (gray levels). These concepts will be extended encode images after an invertible image transformation. Proposed and adopted standard algorithms for image compression will be described, and a review of various image storage technologies will follow. The section will conclude with a description of the image coding algorithms in the Kodak PhotoCD™ consumer image recording and playback system.

Oppenheim & Shafer, *p.416+*

## 20.3 Image Compression

### 20.3.1 Histograms, Information Theory, and Imaging

A digital image is a matrix of pixels with each gray level described as a binary integer of  $m$  bits, for a total of  $M = 2^m$  levels. An image of  $512 \times 512$  pixels with 256 levels requires  $512^2 \cdot 8$  bits (256 KBytes) to store the image. The number of bits needed to store a full-color image is three times as large, as images in each of the three primary colors are required. Though the cost of digital memory capacity continues to decrease, and disk drives with capacities exceeding 300 GByte ( $300 \cdot 1024^3 \cong 3 \cdot 10^{11}$  bytes) are becoming common, the sizes of digital images continue to increase as well. Six-megapixel cameras ( $3000 \times 2000$ ) are now quite affordable, even for many amateurs. An output image from such a camera requires 2000 lines by 3000 pixels by 12 bits by 3 colors, or a total of 216 Mbits ( $\cong 25$  Mbytes per image). To help scholars read old manuscripts, we regularly produce color images for display that are about  $5000 \times 7500$  color pixels, or 107 MBytes (approximately 6 such images may be stored on a standard CD-ROM). The requirements of this project have consistently confirmed Parkinson's Law of computing: that data to be stored always exceeds available storage capacity. However, it is usually possible to satisfactorily store images of  $512^2$  8-bit (256 KBytes) with good visual quality while using much less data. In other words, real images "always" contain less information than the maximum possible; the difference is due to the redundancy of image data, i.e., the gray value of a pixel in a realistic image

usually is not selected from a uniformly random distribution, but rather exhibits some type of correlation with the gray values of other (usually neighboring) pixels. An image or other message with redundant data may be compressed without loss of information by removing some (or even all) of the redundancy. Given the ratio of nonredundant data in a message or image to the total data, the redundancy  $R$  in the message may be defined as:

$$R = 1 - \frac{\text{Nonredundant Data [bits]}}{\text{Total Data [bits]}}$$

$R = 0$  if all bits are required to transmit the message without loss.

Any system that is constrained by transmission and/or storage capacity will benefit from reducing (or even removing) data redundancy. For example, the total resolution (spatial, temporal, and color) of the human visual system ultimately is limited by the transmission capacity of the channel connecting the eye and brain, and removal of redundancies is a principal task of the vision system. This also leads to the concept of *subjective redundancy*, which means that some information may be discarded from an image without visible impact on the quality, e.g., oscillations in gray value with very short periods.

Obviously, to understand the principles of image compression by removing redundancy, the concept of *information* must be considered first. “Information” is the quantity of data in a message, i.e., how much data is required to describe the message, measured in bits (for storage) or in bits per second or in channel bandwidth (Hz) for transmission.

An image can be considered to be a message consisting of different gray values obtained from some set of well-defined possible values. The quantity of information in the message is a function of the probability of occurrence of the event described by the message, e.g., a message indicating that the maximum air temperature  $T_{max} = 70^\circ\text{F}$  in Rochester on New Years’ Day contains more information than a message that  $T_{max}[\text{January 1}] = 25^\circ\text{F}$ . The reason is probably obvious; that  $T_{max} = 70^\circ\text{F}$  occurs rarely (if ever) in Rochester in January (at least until global warming really kicks in!). In words, we can define the importance of the data in the message by noting that “*the norm isn’t “news” but a rare occurrence is.*”

In 1947, Shannon strictly defined the quantity called “information” to satisfy two basic requirements that make intuitive sense. In so doing, he began the study of *information theory*, which is a part of applied probability theory. Though Shannon’s original definition of information is very intuitive, it must be generalized to provide a more complete description of the concept.

The concept of information may be interpreted as the minimum number of questions with a binary answer set (the only possible outcomes are yes/no or 1/0) that must be answered before the correct message may be determined. As thus defined, *information* is synonymous with the removal of *uncertainty*. The uncertainty about the result of experiment  $X$  increases with the number of possible outcomes  $n$ ; if more distinct outcomes are possible, there is more information in a message stating which outcome occurs. The information  $I$  about  $X$  should be a monotonically increasing

function of  $n$ , and thus a monotonically decreasing function of the probability of a specific occurrence.

Shannon defined information by establishing intuitive criteria that the concept should obey and finding a mathematical expression that satisfies these requirements. As a first example, consider the simplest case of an experiment  $X$  which has  $n$  *equally likely* possible outcomes (i.e., the probability of each outcome is  $n^{-1}$ ). The information about the result of experiment  $X$  must be related to  $n$ ; if  $n$  is large, then the information about the specific outcome should be large as well. If  $X$  may be decomposed into two independent experiments  $Y$  (with  $n_1$  equally likely possible outcomes) and  $Z$  (with  $n_2$  equally likely outcomes), then:

$$n = n_1 \cdot n_2 \quad (1)$$

and:

$$p[X] = \frac{1}{n} = p[Y] \cdot p[Z] = \frac{1}{n_1} \cdot \frac{1}{n_2} = \frac{1}{(n_1 \cdot n_2)} \quad (2)$$

Thus the information  $I$  about the outcome of experiment  $X$  should be a function  $f$  that satisfies the requirement:

$$I[X] = f[n] = f[n_1 \cdot n_2] \quad (3)$$

In addition, the information about a composite of independent experiments should be equivalent to the sum of the information of the component experiments. This establishes the second criterion for  $I[X]$ :

$$I[X] = I[Y] + I[Z] = f[n_1] + f[n_2] \quad (4)$$

Thus the information in a message describing the outcome of experiment  $X$  with  $n$  possible outcomes must satisfy:

$$I[X] \implies f[n] = f[n_1 \cdot n_2] = f[n_1] + f[n_2] \quad (5)$$

One appropriate function that satisfies requirement 5 is the logarithm:

$$I[X] \implies f[n] = c \log_b(n) = -c \log_b\left(\frac{1}{n}\right) = -c \log_b(p[X]) \quad (6)$$

where  $c$  is some constant (which can be assumed to be unity for now) and  $b$  is the base of the logarithm such that  $b > 1$ . In fact, Khinchin proved that the logarithm is the *only* continuous function that satisfies the required properties for any finite number  $n$  of possible outcomes (symbols). This definition of information satisfies the additivity requirement for information, i.e.,

$$I[X] = \log_b[n] = \log_b[n_1 \cdot n_2] = \log_b[n_1] + \log_b[n_2] = I[Y] + I[Z] \quad (7)$$

The units of information are *base- $b$  digits*, e.g., decimal digits 0 – 9 for  $b = 10$  and (*bi*)nary *digi(ts)*, or *bits*. for  $b = 2$ .

For a simple example, consider the quantity of information in a statement about the result of a fair coin (so that  $p_H = p_T = 0.5$ ). The number of equally likely outcomes is  $n = 2$  and thus the information in the message about one “toss” is:

$$I[X] = \log_2 [2] = 1 \text{ bit} \quad (8)$$

If the coin has two (indistinguishable!) heads, then there is only one (“equally likely”) outcome and  $p_H = 1, p_T = 0$ . The information content in a statement about the outcome of a toss of a two-headed coin is:

$$I[X] = \log_2 [1] = 0 \text{ bits} \quad (9)$$

Shannon’s original definition of information in eq. 6 applies only to equiprobabilistic outcomes and must be generalized if the probabilities of the different outcomes are not equal. Consider the case of an unfair coin with  $p_H \neq p_T$  (both  $p_H > 0$  and  $p_T > 0$ ). Recall that  $p_H \equiv \frac{N_H}{N}$  for large  $N$ . This is an intermediate case between the certain outcome case ( $p_T = 0 \implies 0$  bits of information per toss), and the fair coin ( $p_H = p_T = 0.5 \implies 1$  bit of information per toss). Intuitively, a message that specifies the outcome when flipping such a coin will convey  $\alpha$  bits of information where  $0 < \alpha < 1$ . The outcome of the unfair coin may be considered to be the result of a cascade of several experiments with equally likely outcomes, but where several outcomes are equivalent. For example, consider the case of an unfair coin where  $p_H = 0.75$  and  $p_T = 0.25$ . The outcome of a single flip can be considered to have four equally likely outcomes  $A - D$ , *i.e.*:

$$p_A = p_B = p_C = p_D = 0.25,$$

where outcomes  $A, B, C$  are heads and outcome  $D$  is a tail. The quantity of information in a single experiment with four equally likely outcomes is:

$$I[X] = \log_2 [4] = -\log_2 \left[ \frac{1}{4} \right] = 2 \text{ bits per toss}$$

However, since the probabilities of the two distinguishable outcomes are not equal, there must be *excess* information in statements about the identical outcomes that must be subtracted from the 2 bits. The excess information in the message about a head is  $\log_2 [n_H]$  multiplied by the probability of a head  $p_H$ . Similarly for a tail, the excess information is  $p_T \log n_T$ , where  $p_T = \frac{n_T}{n}$  and  $n = n_H + n_T$ .

$$\begin{aligned} \text{Excess Information for head} &= \frac{3}{4} \log_2 [3] = \frac{3}{4} \cdot 1.585 = 1.189 \text{ bits} \\ \text{Excess Information for tail} &= \frac{1}{4} \log_2 [1] = \frac{1}{4} \cdot 0 = 0 \text{ bits} \end{aligned}$$

After substituting these results, the total information is:

$$I[X] = \left( \log_2[4] - \frac{3}{4} \log_2[3] - \frac{1}{4} \log_2[1] \right) \text{ bits} \cong 2 - 1.189 = 0.811 \text{ bits}$$

The general expression for information content in the case of the unfair coin with probabilities  $p_H$  and  $p_T$  is:

$$\begin{aligned} I[X] &= \log_2[n] - (p_H \log_2[n_H] + p_T \log_2[n_T]) = (p_H + p_T) \log_2[n] - p_H \log_2[n_H] - p_T \log_2[n_T] \\ &= -p_H (\log_2[n_H] - \log_2[n]) - p_T (\log_2[n_T] - \log_2[n]) \\ &= -p_H \log_2 \left[ \frac{n_H}{n} \right] - p_T \log_2 \left[ \frac{n_T}{n} \right] \\ &= -p_H \log_2[p_H] - p_T \log_2[p_T]. \end{aligned} \quad (10)$$

Since  $p_i \leq 1$ ,  $\log p_i \leq 0$  and  $I[X] \geq 0$ ; the quantity of information is positive. For  $M$  possible outcomes with probabilities  $p_i$ , this definition of information can be extended:

$$I[X] = - \sum_{i=0}^{M-1} (p_i \log_2[p_i]), \text{ subject to the constraint } \sum_{i=0}^{M-1} p_i = 1. \quad (11)$$

Any “impossible” outcome (with  $p_i = 0$ ) is ignored to eliminate the problem of calculating the logarithm of “0”.

If the  $M$  outcomes are equally likely, so that  $p_i = M^{-1}$ , we have:

$$\begin{aligned} I[X] &= - \sum_{i=0}^{M-1} \left( \frac{1}{M} \log_2 \left[ \frac{1}{M} \right] \right) = -M \cdot \left( \frac{1}{M} \log_2 \left[ \frac{1}{M} \right] \right) \\ &= - \log_2 \left[ \frac{1}{M} \right] = + \log_2[M] = - \log_2[p_i], \end{aligned} \quad (12)$$

as before. This demonstrates that the generalized definition of information is consistent with the earlier one.

For the case of the unfair coin with probabilities  $p_H = 0.75$  and  $p_T = 0.25$ , the quantity of information in a statement about the coin toss is:

$$I[X] = -0.75 \log_2[0.75] - 0.25 \log_2[0.25] \cong 0.811 \text{ bits} \quad (13)$$

Now, you may be wondering what a fractional number of bits of information actually “means?” It can be considered as the *average* uncertainty that is removed by the message of the outcome of the experiment  $X$ . If we toss the *fair* coin 100 times, the resulting string of outputs (*e.g.*, *HTTHHHHTHT...*) may be transmitted with 100 bits. The outputs of 100 tosses of the two-headed coin requires 0 bits to transmit while the outcome of 100 tosses of the unfair coin may be specified by  $100 \cdot 0.811 \cong 82$  bits, if the proper coding scheme is used. A method for reducing the number of bits of such a message will be described shortly.

It is very important to note that Shannon's definition of information assumes that the outcomes of a particular experiment are *random* selections from some particular probability distribution. If we flip a "256-sided" fair coin a large number of times (say  $N = 512 \times 512 = 262,144$  "flips"), we can generate an "image" with  $N$  pixels, each with 256 "outcomes" (gray levels) that will be approximately equally populated. If the gray level of some specific pixel is 4, then the gray levels of its neighbors are (by assumption) as likely to be 199 or 255 as 3 or 5. Such a system for generating data may be called a *discrete memoryless source* or *DMS* because an individual output from the system (pixel gray value) is independent of previous outputs. However, we know that adjacent pixels in pictorial images of real scenes usually belong to the same object and tend to have similar gray levels. Thus the gray values of adjacent pictures are usually *correlated*. The effect of correlated pixels on the information content of imagery will be considered shortly.

The measure of information (eq.11) often is called the *entropy* because its form is identical to the quantity that appears frequently in statistical thermodynamics. The entropy of a body of a gas depends on its volume, mass, and (most of all) its temperature. The energy of the gas also depends on these three properties. One of the steps in the Carnot cycle of the ideal heat engine allows a gas to expand within a thermally insulated confined space by pushing against a (slowly moving) piston. The insulation ensures that no heat flows into or out of the gas. The expansion requires that the gas lose some of its thermal energy through cooling; this is the principle of the refrigerator or air conditioner. The lost thermal energy performs *work* on the piston. Such a process in which no heat flows into or out of the system is called *adiabatic* and is *reversible*; the piston may do work on the gas, thus raising the temperature. *The entropy of the system is unchanged by a reversible process.*

In real life, physical interactions are not reversible. For example, no gas expansion is truly adiabatic because the process cannot be thermally insulated perfectly from its surroundings. Real gas expansions increase the entropy of the system. Consider a gas that is confined in a box that is divided in equal parts by a removable partition. A gas is confined in one half and a vacuum exists in the other. Until the partition is removed, the gas is capable of performing useful work through expansion by pushing on a piston. However, if the partition is removed suddenly, the gas expands over time to fill the entire container without performing any work. No thermal energy is lost. However, restoration of the system to its original state would require work to be done on the gas to push it back into the original volume; the system is not reversible. We say that the expansion of the gas increased the entropy of the system because the process was not reversible. In this way, the entropy is a measure of the capability of the system to perform useful work by changing thermal energy into mechanical energy. Equivalently, it is a measure of the *disorder* of the system. The original system was more ordered before removing the partition because we had more knowledge about the location of the molecules of the gas, thus the entropy was lower before the expansion. The entropy is a statistical description, we do not have a complete description of the location and velocity of each molecule either before or after removal of the partition.

The concept of entropy as a measure of disorder is the more applicable to the current problem. The energy of a set of coin flips (or the arrangement of gray levels

in an image) is determined by the statistics of the outcome; by the histogram of results. The entropy of the set of coin flips or image gray levels is determined by the uncertainty in their arrangement given knowledge of the statistics (the histogram). The less likely the result, the more information in a message that the result has occurred or will occur.

### 20.3.2 Information Content of Images

An image can be considered as the output of an ensemble of pixels (experiments) whose results are derived from a discrete set of possibilities (the histogram). It is the histogram of the image with  $M$  gray levels which determines the quantity of information as defined by Shannon via the entropy equation:

$$I [X] = - \sum_{i=0}^{M-1} p_i \log_2 [p_i] \quad (14)$$

For most realistic monochrome images digitized to 8 bits per pixel, the entropy is in the range of 4-6 bits per pixel. To further investigate the meaning of Shannon's definition of information, consider that an image with  $N$  pixels and  $M$  gray levels can be produced from  $N$  experiments with each having  $M$  possible outcomes for a total of  $M^N$  possible distinct images. In the binary image case, the gray values may be produced by  $N$  coin flips. For a  $1 \times 1$  image,  $N = 1$  and the number of possible images is  $2^1 = 2$  (0 and 1), and there are also two possible histograms (1,0) and (0,1). There are  $2^2 = 4$  possible two-pixel binary images (11, 10, 01, and 00) and three possible histograms (2,0), (1,1), and (0,2). Note that there are two images with histogram (1,1). For  $N = 3$  pixels, there are  $2^3 = 8$  distinct images (111, 110, 101, 011, 100, 010, 001, 000) and four possible histograms (3,0), (2,1), (1,2), and (0,3). There is one image each with histogram (3,0) or (0,3), and three images each with histogram (2,1) and (1,2). The progression of the number of possible binary images divided into a number of distinct histograms specified by the set of binomial coefficients:

$$N [N_0, N_1] \equiv \frac{N!}{N_0! \cdot N_1!}$$

where  $N$  is the number of pixels in the image, and  $N_0, N_1$  are the number of pixels with level 0 and 1, respectively. Note that the constraint  $N_0 + N_1 = N$  must be satisfied. The array of the binomial coefficients defines Pascal's triangle as shown below, where the row number represents the number of pixels in the image, the sum of the numbers in the row is the number of possible images, the number of groups in each row is the number of possible histograms, and the number in each group is the number of images that has a particular histogram:

$N = 1$ pixel			1	1				$2^1 = 2$ images 1 binary pixel
$N = 2$			1	2	1			$2^2 = 4$ images 2 binary pixels
$N = 3$			1	3	3	1		$2^3 = 8$ images 3 binary pixels
$N = 4$			1	4	6	4	1	$2^4 = 16$ images 4 binary pixels
$N = 5$			1	5	10	10	5	$2^5 = 32$ with 5 binary pixels
$N = 6$			1	6	15	20	15	$2^6 = 64$ with 6 binary pixels
					$\vdots$			

Number of binary images with different histograms for  $N=1,2,3,4,5,6$

For example, there are  $1 + 4 + 6 + 4 + 1 = 16$  images composed of four binary pixels with five different histograms. One image has histogram (4,0), one with (0,4), four each with histograms (3,1) and (1,3), and six images with histogram (2,2). Note that the number of images for a specific histogram increases as the population of gray values becomes more equally distributed, i.e., toward the center of the row in Pascal’s triangle. As we will soon demonstrate, the information content of an image with a specified histogram is largest when the histogram populations are equal, i.e., for the histogram with the largest number of possible images and thus the greatest uncertainty of image content.

For images with  $M$  gray levels ( $M > 2$ ), calculation of the number of possible images and histograms is somewhat more complicated. The multilevel analog of the binomial coefficient used in Pascal’s triangle is the *multinomial coefficient*:

$$N [N_0, N_1, N_2, N_3, \dots, N_M] \equiv \frac{N!}{N_0! N_1! N_2! \dots N_{M-1}!}$$

where  $N$  is the number of pixels and  $N_0, N_1, N_2, \dots, N_M$  are the populations of each gray level subject to the constraint  $N_0 + N_1 + N_2 + \dots + N_M = N$ . For  $N = 8$  pixels and  $M = 4$  possible gray levels (0-3; 2 bits of quantization), the number of possible images is  $4^8 = 65,536 = 64K$ . The number of distinct 8-pixel images with histogram  $[N_0, N_1, N_2, N_3]$  is:

$$N [N_0, N_1, N_2, N_3] = \frac{8!}{N_0! N_1! N_2! N_3!},$$

where  $N_0 + N_1 + N_2 + N_3 = 8$ . For example, if the histogram is known to be (4,4,0,0), the number of possible 8-pixel 2-bit images is:

$$N [4, 4, 0, 0] = \frac{8!}{4! 4! 0! 0!} = 70.$$

Other specific histograms yield  $N [3, 3, 1, 1] = 1120$  possible images,  $N [3, 2, 2, 1] = 1680$  images, and  $N [2, 2, 2, 2] = 2520$  images. Again, note that the number of distinct

images increases as the histogram becomes “flatter”. Also recall that Shannon’s definition of information is determined by the image histogram. Because there are more distinct images with a flat histogram than with a clustered histogram, the “amount” of information should be larger for an image with a flat histogram. An image with a flat histogram indicates that there is maximum uncertainty and maximum information content, and thus requires the most bits of data for complete specification.

Now consider the number of possible images with specific histograms for more “reasonable” image formats. A common size for a monochrome digital image is  $N \times N = 512^2$  pixels and  $M = 256$  levels, so that the number of distinguishable images is:

$$M^{N \times N} = 256^{(512^2)} = 2^{(8 \cdot 2^9 \cdot 2^9)} = 2^{2,097,152} = (10^{\log_{10}[2]})^{2,097,152} \\ \cong (10^{0.30103})^{2,097,152} = 10^{0.30103 \cdot 2,097,152} \cong 10^{631,306.66}$$

The size of this number may be gauged against the estimates that there are  $10^{78}$  atoms and  $10^{88}$  cubic millimeters in the universe. Of these MANY images, only a single one has the histogram with all pixels clustered at gray level 0. There are  $512^2$  possible images with one pixel at gray level 1 and the rest (262,143) at gray level zero. The number of images with two pixels at gray level 1 and 262,142 at 0 is:

$$N [262142, 2, 0, \dots, 0] = \frac{(262144)!}{(262142!) (2!) (0!) \dots (0!)} \cong 3.44 \cdot 10^{10}$$

If we continue this progression and add more populated gray levels to “flatten” the histogram, the number of distinguishable images increases very rapidly. Such an image with a perfectly flat histogram has  $512^2/256 = 1024$  pixels in each level; the number of such images is:

$$N [1024_0, 1024_1, \dots, 1024_{255}] = \frac{(512^2)!}{(1024!)^{256}} \cong 10^{630,821}.$$

The approximation was derived via Stirling’s formula for  $N!$  where  $N$  is large:

$$\lim_{N \rightarrow \infty} [N!] \cong \sqrt{2\pi N} \cdot N^N \cdot e^{-N}$$

We should check this formula; substitute  $N = 10$  to find that

$$10! \cong 3.5987 \times 10^6$$

whereas the actual result is  $10! = 3.6288 \times 10^6$ , so the error is only about a factor of  $10^{-3}$

If 254 levels are populated with 1028 pixels each and one level has 1032 pixels (a “slightly clustered” histogram), the number of images is:

$$N [1028_0, 1028_1, 1028_2, \dots, 1028_{253}, 1032_{254}, 0_{255}] = 512^2 \frac{!}{(1028!)^{254} \cdot 1032! \cdot 0!} \cong 10^{630,377},$$

which is smaller than the number of images with a flat histogram (by a factor of  $10^{444}$ , which is still pretty large!). Note that the number of possible images again is maximized when the histogram is flat; the uncertainty, the information content, and thus the number of bits to specify the image are maximized when the histogram is flat.

### 20.3.3 Maximizing Image Information

We just showed by example that an image with a flat histogram has the maximum possible information content. The rigorous proof of this assertion is an optimization problem.

Given that the probability distribution of gray levels for  $f[n, m]$  is proportional to the image histogram  $H[f]$ .

$$p_i = \frac{H[f]}{N}$$

where  $N$  is the number of pixels in the image. Since the maximum possible population of a gray level is the total number of pixels (i.e.,  $N$ ),  $0 \leq p_i \leq 1$  as required. The problem is to find the set of gray-level probabilities  $\{p_i\}$  for  $M$  levels that maximizes information content:

$$I[f] = - \sum_{i=0}^{M-1} p_i \log_b [p_i],$$

subject to the constraint:

$$\sum_{i=0}^{M-1} p_i = 1.$$

To maximize  $I$ , we set its total derivative equal to zero:

$$dI = \frac{\partial I}{\partial p_0} dp_0 + \frac{\partial I}{\partial p_1} dp_1 + \frac{\partial I}{\partial p_2} dp_2 + \cdots + \frac{\partial I}{\partial p_{M-1}} dp_{M-1} = 0.$$

subject to the constraint that the probabilities sum to a constant:

$$d \left( \sum_{i=0}^{M-1} p_i \right) = 0$$

This optimization problem is easily solved by *Lagrangian multipliers*. If we maximize a linear combination of  $I[f]$  and  $\sum p_i$ , we will automatically maximize  $I[f]$ . Construct a function  $L[f]$  to be maximized:

$$L[f] = I[f] - \lambda \sum_{i=0}^{M-1} p_i = - \sum_{i=0}^{M-1} p_i \log_b [p_i] - \sum_{i=0}^{M-1} \lambda p_i = - \sum_{i=0}^{M-1} p_i [\log_b [p_i] + \lambda].$$

where  $\lambda$ , the Lagrangian multiplier, is a constant to be determined. To maximize  $L$ ,

we set its total derivative equal to zero:

$$dL = \sum_{i=0}^{M-1} \frac{\partial L}{\partial p_i} dp_i = 0$$

Because the differential probabilities  $p_i$  are arbitrary, a necessary and sufficient condition for  $dL$  to vanish is to have the component derivatives  $\frac{\partial L}{\partial p_i}$  vanish individually:

$$\begin{aligned} \frac{\partial L}{\partial p_i} &= \frac{\partial p}{\partial p_i} \cdot (\log_b [p_i] + \lambda) + p_i \cdot \frac{\partial}{\partial p_i} (\log_b [p_i] + \lambda) \\ &= (\log_b [p_i] + \lambda) + p_i \cdot \frac{1}{p_i} = 1 + \log_b [p_i] + \lambda = 0, \text{ for all } i \\ \log_b [p_i] &= -(1 + \lambda) \\ \implies p_i &= e^{-(1+\lambda)}, \text{ where } \lambda \text{ is constant.} \end{aligned}$$

Note that the probability  $p_i$  for the occurrence of the  $i^{\text{th}}$  level is constant, thus the probabilities of all outcomes must be equal when  $L[f]$  (and hence  $I[f]$ ) is maximized. The constraint allows us to calculate the numerical value of the Lagrangian multiplier:

$$\sum_{i=0}^{M-1} p_i = 1 = \sum_{i=0}^{M-1} b^{-(1+\lambda)} = M b^{-(1+\lambda)} = M p_i = M p \implies \left( p = \frac{1}{M} \right)$$

In words, the information content  $I[f]$  of an image  $f[n, m]$  is maximized when all gray levels are equally populated, *i.e.*, when the histogram is flat. Slow variations in gray level from pixel to pixel are most visible in an image with a flat histogram. This is (of course) the motivation for “histogram equalization” in digital image processing. We should note that histograms of quantized images cannot be “equalized” in a strict sense unless the gray values of some pixels are changed to fill underpopulated bins. This can be performed only as an approximation by considering the gray values of neighboring pixels.

### 20.3.4 Information Content (Entropy) of Natural Images

The redundancy  $R$  in a message was already defined as a dimensionless parameter based on the ratio of nonredundant data to the total data:

$$R = 1 - \frac{\text{Nonredundant Data}}{\text{Total Data}}$$

Because the human visual system is constrained by transmission or storage capacity, it is useful to reduce/remove data redundancy beforehand. For example, the total resolution (spatial, temporal, and color) ultimately is limited by the transmission capacity of the channel connecting the eye and brain, and removal of such redundancies is a principal task of the vision system. Examples of HVS mechanisms that remove redundancy include lateral inhibition of the neural net, opponent-color processing,

and the nonlinear response of the vision receptors.

In *Predictability and redundancy of natural images* (**JOSA A**, Vol. 4, 2395, 1987) D. Kersten estimated the redundancy of some natural images by presenting to observers eight  $128^2$  4-bit images (16,384 pixels, each with 16 gray levels stretched over the available dynamic range). The 4-bit quantization ensured that the images had significant redundancy and precluded the presence of significant additive noise in the quantized values (*i.e.*, additive noise would be typically quantized to zero). The images ranged in content from “simple” to “busy” and included natural scenes. The pixels subtended a rectangle of size  $10 \times 7$  arcminutes to the observer. A predetermined fraction of the pixels were deleted and replaced with one of the 16 gray levels chosen at random. For example, the original image were in fact a random image, then it would be impossible to determine which pixels were altered; such an image would have no redundancy. If the original image were pictorial (e.g., a human face), then most of the replaced pixels will be obvious; such an image has significant redundancy. Observers guessed at the gray value of the replaced pixel until the correct value was assigned. The larger the number of guesses, the more uncertain the gray level of the pixel. The histogram of the number of guesses was used to determine the entropy  $I[f]$ . The number of guesses at each replaced pixel for the random image should be large (mean value of 8), and small for the pictorial image. Therefore, the redundancy is:

$$R[f] = 1 - \frac{I[f]}{I_{\max}} = 1 - \frac{I[f]}{4 \text{ bits}}$$

where  $I[f]$  is the entropy of image  $f$  and  $I_{\max}$  is the number of quantization bits. Kersten’s results indicated that the eight images have redundancies in the interval:

$$0.46 \text{ (“busy” picture)} \leq R \leq 0.74 \text{ (picture of face)}$$

The corresponding content of nonredundant information is:

$$2.16 \text{ bits per pixel} \geq I[f] \geq 1.04 \text{ bits per pixel}$$

Based on the previous discussion, the number of  $128^2$  4-bit images is:

$$\text{Number} = 2^{4 \cdot 128^2} = 2^{65,536} \cong 10^{19,728}$$

But the number of natural images would lie in the range:

$$2^{2.16 \cdot 128^2} \cong 2^{35,389} \cong 10^{10,653} \geq \text{Number} \geq 2^{1.04 \cdot 128^2} \cong 2^{17,0397} \cong 10^{5,129}$$

which is STILL many times larger than the estimated number of atoms in the universe!

## 20.4 Lossless Compression

### 20.4.1 Run-Length Encoding

Run-length encoding is a very simple method for compression of sequential data (and particularly for binary data) that takes advantage of the fact that consecutive single “tokens” (gray values) are often identical in many data sets. Run length encoding inserts a special token each time a chain of more than two equal input tokens are found. This special input advises the decoder to insert the following token  $n$  times into his output stream. For example, consider the sequence of 3-bit data:

7	7	7	2	6	6	6	6	6	2	2	5	5	5	5	5	5	5	5	5
---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---	---

The encoded RLE image would be:

7 3 2 1 6 5 2 2 5 9

where the first digit in the encoded image is the gray value of the first pixel, the second digit is the number of occurrences, etc. The sequence is reduced from 20 3-bit numbers (60 bits) to 10 digits (though the long run of “5” at the end requires a 4-bit number to encode). This means that this sequence might be encoded into 5 3-bit numbers and 5 4-bit numbers, for a total of 35 bits. If the system were limited to three bits, the sequence of 9 examples of level “5” would be split into one sequence of 7 and one of 2:

7 3 2 1 6 5 2 2 5 7 5 2

for a total of 10 3-bit numbers, or 30 bits.

The compression in RLE occurs when the image exhibits strings of the same value. If the image is “noisy”, then the image after RLE coding will likely require more bits than the uncompressed image.

The common “bitmap” image format (.BMP) uses run-length encoding.

### 20.4.2 Huffman Code

We have already seen that the information content of an image is a function of the image histogram; an image with a clustered histogram contains less information than an image with a flat histogram because the former contains statistically redundant information. This implies the existence of methods to encode the information with fewer bits while still allowing perfect recovery of the image. The concept of such a code is quite simple: the most common gray levels are assigned to a code word that requires few bits to store/transmit, while levels which occur infrequently are encoded with many bits; the *average number* of bits per pixel is reduced by attempting to equalize the number of bits per gray level. A procedure for obtaining a code from the histogram was specified by David A. Huffman in 1951 while he was a 25-year-old

graduate student at MIT. Huffman developed the coding scheme as part of a final assignment in a course on information theory given by Prof. Robert A. Fano. After working unsuccessfully on the problem for months, the solution came to Huffman as he was tossing his notebooks into the trash (related in *Scientific American*, September 1991, pp. 54-58). He presented his method to Fano, who “*Is that all there is to it?*” Huffman’s coding scheme is now ubiquitous. (This story must be a metaphor for something.) (Huffman, DA., **Prod. IRE 40**, 1098-1101, 1952).

The Huffman code assumes that the gray values are selected at random from some known probability distribution. In other words, it assumes that the gray values of adjacent pixels are unrelated (which is, of course, not true for meaningful pictorial images, though perhaps true for images transformed to a different coordinate system). A source of uncorrelated random numbers is called “memoryless”.

The entropy of the Huffman-coded image always is within 1 bit per pixel of the information content defined by Shannon. The Huffman code removes bits from the message by discarding *objective redundancy*. It is *lossless* and *unambiguous*. This first quality means that the original image may be reconstructed without error from knowledge of the coded image and the code book. The quality of no ambiguity ensures that only one set of gray levels may be decoded from an ungarbled string of binary digits encoded by the Huffman procedure.

The Huffman code is perhaps best described by example. The pixels in an image with 8 gray levels are usually defined specified by a binary code that requires 3 bits per pixel. For attempted clarity, the gray levels will be indicated by alphabetic characters:

Decimal. = Binary  $\rightarrow$  Alphabetic

$$0. = 000_2 \rightarrow A$$

$$1. = 001_2 \rightarrow B$$

$$2. = 010_2 \rightarrow C$$

$$3. = 011_2 \rightarrow D$$

$$4. = 100_2 \rightarrow E$$

$$5. = 101_2 \rightarrow F$$

$$6. = 110_2 \rightarrow G$$

$$7. = 111_2 \rightarrow H$$

Consider a 3-bit 100-pixel image with levels distributed as in the following histogram:

$$H[A, B, C, D, E, F, G, H] = [0, 9, 12, 40, 30, 5, 4, 0]$$

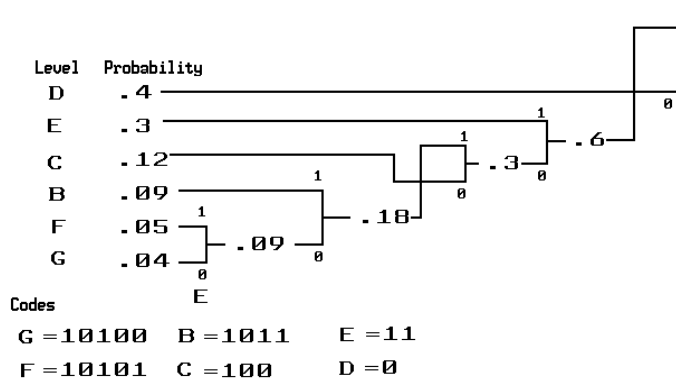
The probability of each gray level therefore is:

$$p[A] = 0, p[B] = .09, p[C] = .12, p[D] = .40, p[E] = .30, p[F] = .05, p[G] = .04, p[H] = 0$$

The information content in this image obtained from Shannon’s entropy formula:

$$\begin{aligned}
 I &= -0.4 \log_2 [0.4] - 0.3 \log_2 [0.3] - \dots \\
 &\cong 0.529 + 0.521 + \dots + 0 \\
 &\cong 2.131 \text{ bits/pixel}
 \end{aligned}$$

which is considerably less than the 3-bit quantization. To derive the Huffman code, arrange the occupied levels in descending order of probability. One bit is used to distinguish the least likely pair of levels using either the convention that the binary digit 1 is assigned to the more probable level and 0 to the less probable, or vice versa. This is the *least significant bit* in the Huffman code, as it distinguishes the most rarely occurring gray levels. The probabilities of the two rarest levels are then summed (giving a total probability of 0.09 in this case) and the list of gray values is rearranged in descending order. In this example, the sum of the probabilities of the rarest levels *F* and *G* is equal to the probability of the next level *B*, and so reordering is not required. The new pair of least-likely levels in the rearranged list are distinguished by again assigning a binary digit using the same convention as before. The last two probabilities are summed, the list is reordered, and the process continues until bits are assigned to the last pair of probabilities. The last binary digit derived using this procedure is the *most significant bit*. The schematic of the entire sequence is shown in the figure:



*Calculation of Huffman code for 3-bit image with probabilities as listed in the text. The gray levels are arranged in descending order of probability. One bit is used to distinguish the rarest levels, the probabilities are summed, the list is rearranged in descending order, and the process continues.*

The Huffman code for a specific gray level is the sequence of binary digits assigned from right to left for that level, i.e., from most significant to least significant bit. The code for level *B* is obtained by following the sequence from the right: the path for level *B* is in the upper branch on the far right (code = 1), and the lower branch at two more junctions (codes = 0 and 0) so that the code for level *B* is 1011<sub>2</sub>. The codes for the other levels are found similarly and the resulting code book is listed above. The number of bits to encode all pixels having each gray level may be calculated, and their sum is the average number of bits per pixel required to encode the entire image:

Gray Level $f$	$p[f]$	Code	Bits in Code	$\langle Bits \rangle$ for Level
D	0.40	$0_2$	1	$0.40 \times 1 = 0.4$
E	0.30	$11_2$	2	$0.30 \times 2 = 0.6$
C	0.12	$100_2$	3	$0.12 \times 3 = 0.36$
B	0.09	$1011_2$	4	$0.09 \times 4 = 0.36$
F	0.05	$10101_2$	5	$0.05 \times 5 = 0.25$
G	0.04	$10100_2$	5	$0.05 \times 4 = 0.20$
				$\langle Bits \rangle$ for Image = 2.17 bits/pixel

To demonstrate the unambiguous nature of the code, consider the sequence of 14 pixels “CDDEEBEDFGCDEC”, which encodes to a sequence of 35 bits (2.5 bits per pixel)

$$CDDEEBEDFGCDEC = 10000111110111101010110100100011100$$

The message is decoded by examining the order of bits. Since the first bit is not a “0”, it cannot be the most common gray value “D”. Since the second bit is not a “1”, the first pixel cannot be an “E”. The third bit is “0”, and therefore the first pixel must be “C”

$$\boxed{100}0011110111101010110100100011100$$

Now repeat: the fourth bit is “0”, and only “D” has this first bit:

$$\boxed{100} \boxed{0} 011110111101010110100100011100$$

Ditto, “D” is the third character:

$$\boxed{100} \boxed{0} \boxed{0} 111110111101010110100100011100$$

The fourth pixel begins with “11”, and therefore is “E”:

$$\boxed{100} \boxed{0} \boxed{0} \boxed{11} 1110111101010110100100011100$$

Ditto for the fifth”

$$\boxed{100} \boxed{0} \boxed{0} \boxed{11} \boxed{11} 10111101010110100100011100$$

The sequence continues until all bits have been decoded:

100	0	0	11	11	1011	11	0	10101	10100	100	0	11	100
C	D	D	E	E	B	E	D	F	G	C	D	E	C

This example also may be used to illustrate the pitfall of Huffman coding. If a bit is garbled (“flipped”), then it is likely that many (if not all) of the following characters in the message will not be decodable, because the redundancy in the message has been removed. Consider the example where the sixth bit is flipped from a “1” to a “0”:

10000011110111101010110100100011100

In this case, the decoded message would be:

100	0	0	0	11	11	0	11	11	0	10101	10100	100	0	11	100
C	D	D	E	E	D	E	E	D	F	G	C	D	E	C	

instead of:

C	D	D	E	E	B	E	D	F	G	C	D	E	C
---	---	---	---	---	---	---	---	---	---	---	---	---	---

The “flipped” bit resulted in the incorrect decoding of the sample “B” as two samples “DE”.

This code exceeds the theoretical limit of Shannon information by only 0.04 bits/pixel, and reduces the number of bits required to store the image to  $(2.17/3)$  bits, or 72% of that required for the uncompressed original. The efficiency of the Huffman code is defined as the ratio of the average number of bits per pixel for the code to the theoretical limit determined by Shannon’s definition. In this case, the compression efficiency is:

$$\eta = \frac{2.131}{2.17} = 0.982$$

For real images, lossless compression via a Huffman code on a pixel-by-pixel basis can achieve compression efficiencies in the range of  $0.67 \leq \eta \leq 0.25$ , a modest improvement. Note that the code book must be available to the receiver to allow reconstruction of the image. This extra data is known as the *overhead* of the compression scheme and has not been included in the calculations of compression efficiency. The length of the Huffman codeword of a gray level whose histogram probability is  $p$  is  $-\log_2 [p]$ , e.g., if the probability is 0.125, the ideal codeword length is 3 bits. If  $\log_2 [p]$  is significantly different from an integer (e.g.,  $p = 0.09 \implies -\log_2 [p] = 3.47$  bits), the coding efficiency will suffer.

A significant shortcoming of a Huffman code is that it cannot adapt to locally varying image statistics, or equivalently, a particular Huffman code will not be optimum for a variety of images. In fact, inappropriate application of a Huffman code can actually lead to an increase in the storage requirements over the original, e.g., if the image contains many levels which were rare in the original source image.

### 20.4.3 Information in Correlated Images – Markov Model

We've just seen how the Huffman code takes advantage of clustering of the histogram to reduce the number of bits required to store/transmit an image whose gray levels are not equally populated, and assuming that the gray values of pixels are obtained from a discrete memoryless source, i.e., the gray value at a pixel is a number selected at random from the probability distribution (the image histogram). Obviously, this last assumption is false for most (if not all) realistic images consisting of objects whose component pixels have similar properties (e.g., gray level, color, or texture). These correlations provide a context for a pixel and add additional redundancy, which may be exploited to achieve additional compression. Redundancy may be considered as creating clusters in particular histograms generated from the original images. Examples of redundancy include similarities of pixel gray level in local neighborhoods (interpixel redundancy) in a single image, that may be exploited by constructing codes for groups of pixels of a single image (*vector coding*) or by coding linear combinations of blocks of pixels, as in the JPEG standard. Similarities in color (*spectral redundancy*) generate clusters in the multispectral histogram, thus reducing data-transmission/storage requirements in color images. This clustering is used by the NTSC video transmission standard and the Kodak *PhotoCD*<sup>TM</sup>. Correlations of corresponding pixels across image frames in a motion picture (*temporal redundancy*) allows significant additional compression and is exploited in the MPEG standards. In addition, images meant for human viewing may be compressed by removing image content that is not visible to the eye; these subjective redundancies or superfluous information are present the spectral, spatial, and temporal dimensions, and are utilized for additional compression in all of the consumer compression standards, such as JPEG, MPEG, and *PhotoCD*<sup>TM</sup>.

The statistical properties of a correlated image are more complicated than those of a discrete memoryless (random) source, and one of the difficulties in developing efficient standard algorithms for image compression is the creation of a mathematical model of these various redundancies. The simplest model of interpixel redundancies is the *Markov* source, where the probability that the pixel located at coordinate  $n$  has gray level  $f$  is a function of the gray level at some number of neighboring pixels. The number of these neighboring pixels is the *order* of the Markov source; the higher the order, the more correlated are the gray values in a neighborhood and the more predictable the image content from previous levels. Thus we should be able to define a measure of information for a Markov source that is less than the normal entropy. For a correlated source model, such as a Markov source of order 1, we can define the first-order, or *conditional*, entropy of the image with  $M$  gray levels as:

$$I[f_k|f_{k-1}] = \sum_{i=0}^{M-1} \sum_{j=0}^{M-1} p_{i,j} \log_2 [p_{i,j}|p_j]$$

where  $f_k$  and  $f_{k-1}$  are the gray levels of the  $k^{\text{th}}$  and  $(k-1)^{\text{st}}$  pixels and  $I[f_k|f_{k-1}]$  is the information in the  $k^{\text{th}}$  pixel  $f_k$  given that the  $(k-1)^{\text{st}}$  pixel has gray value  $f_{k-1}$ . The conditional probability of  $p_{i,j}$  given  $p_j$  is denoted  $[p_{i,j}|p_j]$ . It may be clearer

to think of the conditional entropy as just the information content of the set of 2-D gray-level “vectors”  $\underline{\mathbf{f}} = [f_1, f_2]$

$$I[\underline{\mathbf{f}}] = - \sum_{\underline{\mathbf{f}}} p[\underline{\mathbf{f}}] \log_2 [p[\underline{\mathbf{f}}]]$$

where the sum is over all the 2-D pixel vectors  $\underline{\mathbf{f}}$ . Note that the total number of gray-level states of the 2-D histogram is the square of the number of gray levels; the vector histogram of an image with  $M$  levels has  $M^2$  bins.

The first-order entropy is the average information content of the experiment at the  $k^{th}$  pixel given that the  $k - 1^{st}$  pixel gray value is known. If the gray values are correlated, the first-order entropy per pixel may be much less than the Shannon entropy (pixels considered independently); in other words, the vector histogram of the image is clustered. Note that the Shannon entropy may be considered as the zeroth-order, or scalar entropy of the image. If the pixel gray levels are independent (random DMS source), then the first-order entropy will be  $\log_2 [M^2] = 2 \cdot \log_2 [M]$  bits per pixel because every pair of gray levels will be equally likely; the 2-D vector histogram is thus flat, i.e., all 2-D vectors (whose components are gray levels of adjacent pixels) will be equally likely. The image from a first-order Markov source may have a flat histogram, but the 2-D histogram of neighboring pixels may be clustered; 2-D vector coding of such an image will exhibit significant data compression.

#### 20.4.4 “Vector” Coding (Compression)

Images generated by a Markov source often may be compressed by examining the histogram of groups of pixels to look for correlations. As a simple example of the difference between a DMS and a Markov source, consider first a bilevel DMS with a priori probabilities  $p[0] = 0.2$  and  $p[1] = 0.8$ . In words, the output of the source is more likely a 1 (white) than a zero. A FAX output is a bilevel image that might have this probability function. The Shannon entropy of this source is:

$$\begin{aligned} I &= -0.2 \log_2 [0.2] - 0.8 \log_2 [0.8] \\ &= 0.7219 \text{ bits per pixel} \end{aligned}$$

Now consider the probabilities of the four cases for two pixels generated by the DMS. Since the pixels are independent, the probability that the current pixel is a “1” given that the previous pixel is a “1” is just the product of the probabilities – there is no influence of the previous pixel on the choice of the current pixel:

$$f[n] = \boxed{1} \boxed{1} : p \boxed{1} \boxed{1} = p[1] \cdot p[1] = 0.8 \cdot 0.8 = 0.64$$

Similarly, the probability that the current pixel is a “0” given that the previous pixel is a “0” is:

$$\begin{aligned} f[n] = \begin{bmatrix} 0 & 0 \end{bmatrix} & : p \begin{bmatrix} 0 & 0 \end{bmatrix} = p[0] \cdot p[0] = 0.2 \cdot 0.2 = 0.04 \\ f[n] = \begin{bmatrix} 1 & 0 \end{bmatrix} & : p \begin{bmatrix} 1 & 0 \end{bmatrix} = p[1] \cdot p[0] = 0.8 \cdot 0.2 = 0.16 \\ f[n] = \begin{bmatrix} 0 & 1 \end{bmatrix} & : p \begin{bmatrix} 0 & 1 \end{bmatrix} = p[0] \cdot p[1] = 0.8 \cdot 0.2 = 0.16 \end{aligned}$$

Note that the sum of these conditional probabilities is still unity, as required. The entropy of the conditional probabilities is:

$$\begin{aligned} I &= -0.64 \log_2 [0.64] - 0.04 \log_2 [0.44] - 0.16 \log_2 [0.16] - 0.16 \log_2 [0.16] \\ &= 1.4438 \text{ bits per element} \end{aligned}$$

Since there are two pixels per element, the entropy of the pixels taken two at a time is just:

$$\frac{I = 1.4438 \text{ bits per pair}}{2 \text{ pixels per pair}} = 0.7219 \text{ bits per pixel}$$

which is identical to the scalar entropy of the *DMS*. In words, the information in the pixels from the *DMS* taken two at a time is identical to that from the *DMS* taken one at a time; there is no additional compression because there is no interpixel correlation.

In a realistic imaging situation, we might expect that black and white pixels will be grouped together in the message. Thus the probabilities  $p \begin{bmatrix} 1 & 1 \end{bmatrix}$  and  $p \begin{bmatrix} 0 & 0 \end{bmatrix}$  would be larger in a real image than for a discrete memoryless source. To ensure that the sum of the probabilities is unity,  $p \begin{bmatrix} 0 & 1 \end{bmatrix}$  and  $p \begin{bmatrix} 1 & 0 \end{bmatrix}$  would be expected to decrease, and also to be the same, since we would expect the same number of transitions from black to white as from white to black. A possible table of probabilities from a first-order Markov source with  $p[0] = 0.2$  and  $p[1] = 0.8$  would be:

$$\begin{aligned} p \begin{bmatrix} 1 & 1 \end{bmatrix} &= 0.8 \\ p \begin{bmatrix} 0 & 0 \end{bmatrix} &= 0.16 \\ p \begin{bmatrix} 0 & 1 \end{bmatrix} &= p \begin{bmatrix} 1 & 0 \end{bmatrix} = 0.02 \end{aligned}$$

The resulting entropy of the pixels taken two at a time is:

$$\begin{aligned} I &= -0.80 \log_2 [0.80] - 0.16 \log_2 [0.16] - 0.02 \log_2 [0.02] - 0.02 \log_2 [0.02] \\ &= 0.9063 \text{ bits per element} = 0.4632 \text{ bits per pixel} \end{aligned}$$

There is a significant reduction in the entropy of 0.7291 bits per pixel for this example of a first-order Markov source. The additional compression is due to interpixel

correlation.

The concept may be extended to higher-order entropies. If the source is second-order Markov, the gray value of a pixel is influenced by those of two adjacent pixels. In this case, the set of 3-D vectors defined by triplets of gray values would be clustered, and 3-D vector coding will further reduce the number of bits.

As an example of the effect of spatial correlations on image compression, consider first the the  $4 \times 4$  2-bit image shown:

$$f[n, m] = \begin{array}{|c|c|c|c|} \hline 0 & 1 & 0 & 0 \\ \hline 0 & 1 & 2 & 2 \\ \hline 0 & 1 & 2 & 3 \\ \hline 1 & 2 & 2 & 3 \\ \hline \end{array}$$

$$H[f] = [5, 4, 5, 2], \quad p[f] = \left[ \frac{5}{16}, \frac{4}{16}, \frac{5}{16}, \frac{2}{16} \right]$$

$$I[f] = - \left( \frac{5}{16} \log_2 \left[ \frac{5}{16} \right] + \frac{1}{4} \log_2 \left[ \frac{1}{4} \right] + \frac{5}{16} \log_2 \left[ \frac{5}{16} \right] + \frac{1}{8} \log_2 \left[ \frac{1}{8} \right] \right) \\ = 1.924 \text{ bits/pixel}$$

Because the histogram of this image is approximately "flat", little (if any) benefit would be obtained by using a Huffman code. But note that adjacent pairs of pixels exhibit some correlation; if a pixel is dark (0 or 1), it is more likely that its neighbor to the right is dark than bright. This is demonstrated by constructing the two-dimensional histogram of the pairs of left-right pixels of  $f[n,m]$ :

$$H[L, R] = \begin{array}{|c|c|c|c|c|} \hline \textit{Gray Values} & & \textit{LEFT} & \textit{SIDE} & \\ \hline & & 0 & 1 & 2 & 3 \\ \hline 0 & & 0 & 0 & 2 & 0 \\ \hline \textit{RIGHT} 1 & & 0 & 1 & 1 & 0 \\ \hline \textit{SIDE} 2 & & 3 & 0 & 0 & 0 \\ \hline 3 & & 1 & 0 & 0 & 0 \\ \hline \end{array}$$

Note that each pixel appears in only one pair; there is no double-counting. The sum of the elements of the 2-D histogram is 8, and thus there are 16 pixels (8 pairs) in  $f[n, m]$ . Also note that there is a total of 16 possible elements in the 2-D histogram, four times as many as in the 1-D histogram; the 2-D histogram of a 2-bit image is a 4-bit image. Even so, if the 2-D histogram of pixel pairs is clustered, it may be possible to encode the representation in fewer bits. The entropy of this specific 2-D

histogram is:

$$I[H[L, R]] = -\frac{3}{8} \log_2 \left[ \frac{3}{8} \right] + 3 \cdot \left( -\frac{1}{8} \log_2 \left[ \frac{1}{8} \right] \right) - \frac{2}{8} \log_2 \left[ \frac{2}{8} \right] = 2.156 \frac{\text{bits}}{\text{element}}$$

Because there are two pixels per element, the information content per pixel in the 2-D histogram is:

$$\frac{2.156 \text{ bits}}{2 \text{ pixel}} = 1.078 \frac{\text{bits}}{\text{pixel}} < 1.924 \frac{\text{bits}}{\text{pixel}}$$

The reduction in entropy of the image is due to the correlations between neighboring gray values. As before, the elements of the 2-D histogram may be coded by a Huffman procedure. This approach is called *vector coding* or *block coding*, because each pixel pair describes a two-dimensional vector of gray levels. In analogous fashion, the original Huffman approach to encode individual pixels is sometimes called *scalar coding*.

Of course, it is feasible to encode larger blocks, which is equivalent to constructing gray-level vectors with more dimensions. For example, consider the  $4 \times 4$  3-bit image:

$$f[n, m] = \begin{array}{|c|c|c|c|} \hline 0 & 1 & 0 & 1 \\ \hline 2 & 3 & 2 & 3 \\ \hline 0 & 1 & 0 & 1 \\ \hline 2 & 3 & 2 & 3 \\ \hline \end{array}$$

If we encode  $2 \times 2$  blocks of four pixels, the four resulting four-dimensional vectors are identical. Because the 4-D histogram has only one populated bin, the information content of the vector image is 0 bits per block, or 0 bits per pixel. Of course, we must also transmit the gray-level formation of the block (i.e., the codebook that specifies the gray values assigned to each code). This represents the necessary overhead of the code. In this case, the number of required bits is determined by the codebook alone.

Note that the effectiveness of a vector code depends strongly on gray-level correlations of the image. A vector code that is effective for one image (e.g., a human face) may be ridiculously ineffective for a different type of image (e.g., an aerial image of Kuwait). Conversely, a vector code that is appropriate for a particular type of image will likely be so for images of objects in the same class. Also note that if the vector histogram is flat (i.e., approximately equal populations in each bin), then there will be no reduction in storage requirements obtained by using the Huffman code.

### Example – Entropy of the English Alphabet

Sources: N. Abramson, **Information Theory and Coding**, McGraw-Hill, 1963.

J.R. Pierce, **An Introduction to Information Theory**

Shannon, “*Prediction and Entropy of Printed English*,” **Bell Syst Tech Jour.** **30**, pp.50-64, 1951.

The simplest messages in the English language may be written with 26 letters (one

case) and the space. If these 27 characters were equally probable, the information content in a message would be:

$$I = \sum_{i=1}^{27} - \left( \frac{1}{27} \right) \log_2 \left[ \frac{1}{27} \right] = -\log_2 \left[ \frac{1}{27} \right] = +\log_2 [27]$$

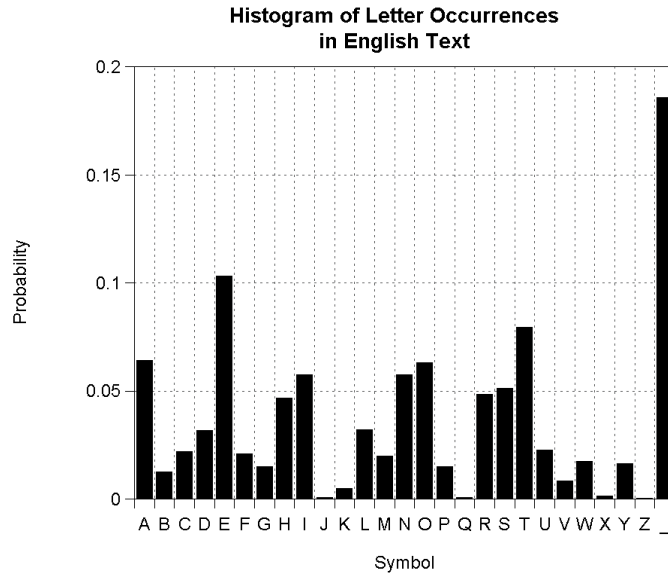
$\cong 4.755$  bits per character

A sample of typical text with equal probabilities is:

XFOML RXKHRJFFJUJ ZLPWCFWKCYJFFJ EYVKCQSGHYD QPAAMKB

Of course, the probabilities of English characters are not equal. The histogram of the characters may be determined from a statistical study of words. Abramson gives the following table of character occurrences:

Symbol	Probability	Symbol	Probability
A	0.0642	N	0.0574
B	0.0127	O	0.0632
C	0.0218	P	0.0152
D	0.0317	Q	0.0008
E	0.1031	R	0.0484
F	0.0208	S	0.0514
G	0.0152	T	0.0796
H	0.0467	U	0.0228
I	0.0575	V	0.0083
J	0.0008	W	0.0175
K	0.0049	X	0.0013
L	0.0321	Y	0.0164
M	0.0198	Z	0.0005
		(space)	0.1859



*Graph of histogram of letter occurrences in the English language if the case of the character is not considered.*

Using this “1-D” histogram of probabilities (i.e., assuming that the characters occur “independently” but are taken from this histogram of probabilities), the entropy of an English-language message would be somewhat less than for equally likely occurrences:

$$I = \sum_{i=1}^{27} (-p_i \log_2 [p_i]) \cong 4.080 \text{ bits per character}$$

A sample of text that might be selected from this probability distribution is:

OCRO HLI RGWR NMIELWIS EU LL NBNESEBYA TH EEI ALHENHTTPA L

Just as obviously, we know that English characters do not occur “independently”; the probability of occurrence of a character depends on the particular characters in the preceding sequence. The next most realistic description is based on the frequency of occurrence of pairs (“digrams”) of characters, and thus on the 2-D histogram of  $27^2 = 729$  pairs. We know that some combinations (e.g., “QX”, “ZJ”) occur very rarely if at all, and so these “bins” of the 2-D histogram will be unoccupied. Therefore, the 2-D histogram of digrams is “clustered” and thus we expect some characters to be “predictable”. Therefore the information content of those characters will be decreased. The histogram of character frequencies may be computed from statistical digram frequency tables that were constructed by cryptographers. Shannon computed the resulting entropy to be:

$$I(27 \text{ characters as pairs}) = \sum_{i=1}^{27} \sum_{j=1}^{27} (-p[i|j] \log_2 [p[i|j]]) \cong 3.56 \text{ bits per character}$$

Typical text selected from the 2-D histogram is:

ON IE ANTSOUTINYS ARE T INCTORE ST BE S DEAMY  
ACHIN D ILONASIVE TUOOOWE AT

Note that this text appears to be (perhaps) slightly more “intelligible” than that selected from the independent histogram – some characters almost form words! (e.g., inctore, deamy)

To continue the idea, consider the computation of entropy based on “trigrams”, or triplets of characters. Shannon computed the entropy:

$$I(27 \text{ characters as triplets}) = \sum_{i=1}^{27} \sum_{j=1}^{27} \sum_{k=1}^{27} (-p[i|j|k] \log_2 [p[i|j|k]]) \cong 3.3 \text{ bits per character}$$

A typical sample of such text is:

IN NO IST LAT WHEY CRATICT FROURE BIRS GROCID PONDENOME  
OF DEMONSTURES OF THE REPTAGIN IS REGOACTIONA OF CRE

Note that as more characters are included in a group for statistical computation of the entropy, the more the typical text resembles English. Shannon continued the process and estimated the upper and lower bounds of the entropy per character for groups of English letters up to 15 characters and also for 100 characters. The entropy per character approaches a limit for more than 9 or so letters in the interval  $1 \leq I \leq 2$  bits per character, but drops to the range  $0.6 \leq I \leq 1.3$  bits for groups of 100. This means that characters in long strings of English characters are correlated; the 100-dimensional histogram of groups of 100 characters exhibits clustering.

Of course, messages could be constructed from tables of *word* frequencies instead of character frequencies. A message based on first-order word frequencies (one word at a time) is:

REPRESENTING AND SPEEDILY IS AN GOOD APTOR COME CAN DIFFERENT  
NATURAL HERE HE THE A IN CAME THE TO OF TO EXPERT GRAY COME  
TO FURNISHES THE LINE MESSAGE HAD BE THESE

and a message using second-order frequencies is:

“THE HEAD AND IN FRONTAL ATTACK ON AN ENGLISH WRITER THAT  
THE CHARACTER OF THIS POINT IS THEREFORE ANOTHER METHOD FOR  
THE LETTERS THAT THE TIME OF WHO EVER TOLD THE PROBLEM FOR  
AN UNEXPECTED”

## 20.4.5 Other Flavors of Huffman Coding

### Modified Huffman Codes

In many cases of both message and image compression, the codebook is quite large but includes many symbols with very small probabilities. It often is useful to combine the very unlikely codes into a single symbol ELSE, which is encoded by the Huffman process and is transmitted along with the actual binary code for the character or gray level. A variation of these scheme is used for digital facsimile.

### Adaptive Huffman Codes

Another modification of Huffman coding allows the process to adapt to varying statistics, but the algorithms are complicated to implement.

## 20.4.6 Arithmetic Coding

**IBM Jour. Res. & Dev.**, **32**(6), November 1988

Gonzalez and Woods, pp. 348-9

Rabbani and Jones, §3.5

Like the Huffman code, the result of an arithmetic code is a sequence of variable-length code symbols, but the symbols are not assigned to pixels (or blocks thereof) which were quantized to fixed numbers of bits. In other words, the arithmetic code for a group of gray levels is not restricted to an integer number of bits. For example, consider encoding a bitonal image (such as a FAX), where one level (probably white) is much more likely than the other. Because there are only two levels, it is not possible to improve on one bit per pixel, even using a Huffman code. Instead, the arithmetic code is a *tree code*, where one codeword is assigned to each string of input pixels of some fixed length. The generated code word for this string is a representation of an interval on the real line within the range  $[0,1)$  whose length is proportional to the likelihood of the string. If the string to be coded is lengthened, then the corresponding subinterval becomes shorter and requires more bits to distinguish it from its neighbors. A slight change in the sequence can result in a significantly different codeword. The algorithm for generating an arithmetic code is most easily described by example. Consider a sequence of 1-bit characters where the frequencies of 0 and 1 are  $3/4$  and  $1/4$ , respectively. One example of such a sequence is 0010. A unit interval is divided into subintervals based on the order of occurrences of characters (quantized pixels) in the sequence. If the next character in the message is 0, the bottom  $3/4$  of the interval will be selected; if 1, the upper  $1/4$  will be selected. At the start the full interval is:

$$0 \leq x_0 < 1$$

The first character in the message is “0” with known frequency of  $3/4$ ; the unit interval is shrunk to the lower  $3/4$ :

$$0 \leq x_1 < \frac{3}{4}, |x_1| = \frac{3}{4}$$

The second character also is “0”, and the interval is subdivided to the lower 3/4 again:

$$0 \leq x_2 < \frac{9}{16}, |x_2| = \frac{9}{16}$$

The third character is “1” with frequency  $\frac{1}{4}$ , so the next subinterval is the upper  $\frac{1}{4}$  of  $x_2$ :

$$\frac{9}{16} - \frac{1}{4} \cdot \frac{9}{16} = \frac{27}{64} \leq x_3 < \frac{9}{16}, |x_3| = \frac{9}{64}$$

$$(0_{\Delta}011011)_2 \leq (0_{\Delta}1)_2 < (0_{\Delta}10000111)_2$$

where the symbol “ $\Delta$ ” is the “binary point” (analogous to the “decimal point”; it separates the bits for positive and negative powers of 2). The last character in the message is “0”, so the interval is subdivided to the lower 3/4:

$$\frac{27}{64} = 0.421875 \leq x_4 < \frac{27}{64} + \left(\frac{3}{4} \cdot \frac{9}{64}\right) = \frac{135}{256} = 0.5273475, |x_4| = \frac{27}{256}$$

Any point in the subinterval  $x_4$  can be used as the code for the string, because that  $x_4$  can only be obtained by that specific sequence of  $m$  input characters.

Because the probabilities are multiples of powers of two in this specific example, the length of the subinterval is easily represented as a binary fraction. Though this is not always true, it is useful to continue the analysis to show how the code may be represented. Recall that fractional numbers may be represented in binary notation by breaking up into inverse powers of 2, so the the bit closest to the binary point represents  $2^{-1}$ , the second bit represents  $2^{-2}$ ,  $\dots$ . The endpoints of the interval  $x_4$  in this example are:

$$\begin{aligned} \frac{27}{64} &= \frac{0}{2} + \frac{1}{4} + \frac{1}{8} + \frac{0}{16} + \frac{1}{32} + \frac{1}{64} + \frac{0}{128} + \frac{0}{256} = (0_{\Delta}01101100)_2 \rightarrow (0_{\Delta}011011)_2 \\ \frac{135}{256} &= \frac{1}{2} + \frac{0}{4} + \frac{0}{8} + \frac{0}{16} + \frac{0}{32} + \frac{1}{64} + \frac{1}{128} + \frac{1}{256} = (0_{\Delta}1000011)_2 \\ &(0_{\Delta}011011)_2 \leq x_4 < (0_{\Delta}1000011)_2 \end{aligned}$$

The arithmetic code for the sequence “0010” is produced by selecting the representation of ANY binary fractional number in the interval  $x_4$ , though the shortest binary fractional number will give the shortest code. In this example, the binary fractional number  $(0_{\Delta}1)_2$  could be selected because it lies between the endpoints:

$$(0_{\Delta}011011)_2 \leq (0_{\Delta}1)_2 < (0_{\Delta}1000011)_2$$

The binary sequence to the right of the binary point is the encoded sequence, which in this case is the one-bit number 1; four letters have been encoded with a single bit.

As a second example, consider encoding of the sequence “1000” with the same  $a$

*a priori* probabilities. The first subinterval is the upper fourth of the unit interval:

$$\frac{3}{4} \leq x_1 < 1, |x_1| = \frac{1}{4}$$

The second subinterval is the lower 3/4 of  $x_1$ :

$$\frac{3}{4} \leq x_2 < \frac{3}{4} + \frac{3}{4} \cdot \frac{1}{4} = \frac{15}{16}, |x_2| = \frac{3}{16}$$

The third subinterval is the lower 3/4 of  $x_2$ :

$$\frac{3}{4} \leq x_3 < \frac{3}{4} + \left(\frac{3}{4}\right)^2 \cdot \frac{1}{4} = \frac{57}{64}, |x_3| = \frac{9}{64}$$

The final subinterval is the lower 3/4 of  $x_4$ :

$$\frac{3}{4} \leq x_4 < \frac{3}{4} + \left(\frac{3}{4}\right)^3 \cdot \frac{1}{4} = \frac{219}{256}, |x_4| = \frac{9}{256}$$

The binary codes for the endpoints are:

$$\frac{1}{2} + \frac{1}{4} = (0_{\Delta}11000000)_2 \leq x_4 < \frac{1}{2} + \frac{1}{4} + \frac{0}{8} + \frac{1}{16} + \frac{1}{32} + \frac{1}{64} + \frac{1}{128} + \frac{1}{256} = (0_{\Delta}11011111)_2$$

The code for the subinterval is the lower limit  $(0_{\Delta}11)_2$ , thus encoding the four-bit sequence with two bits.

Endpoints for and lengths of other possible sequences are:

$$\text{“0000”} \rightarrow [0, (0.75)^4] = [0, 0.31640625] \rightarrow |x_4| = (0.75)^4$$

$$\text{“1111”} \rightarrow [1 - (0.25)^4, 1] = [0.99609375, 1] \rightarrow |x_4| = (0.25)^4 = 0.0039 \dots$$

$$\text{“1010”} \rightarrow [0.890625, 0.92578125] \rightarrow |x_4| = 0.03515625$$

$$\text{“1011”} \rightarrow [0.92578125, 0.9375] \rightarrow |x_4| = 0.01171875$$

Notice that strings of more frequent characters yield longer subintervals, while strings of rare characters result in short subintervals. If the histogram of input characters is flat, then the intervals derived by the arithmetic code to representing a string of characters will be of equal length and will require equal numbers of bits to represent them. If the histogram is clustered so that some characters are more likely to occur, then strings of frequently occurring characters will be mapped to longer subintervals within  $[0,1)$ , and may be represented by short codes which indicate a location in the interval.

To decode the sequences, the *a priori* probabilities must be known. The first two

examples will be demonstrated, where the codes were binary strings. First, the binary points are added on the left of the code to yield:

$$x = (0_{\Delta}1)_2 = \frac{1}{2}$$

for the first example. The decision point which determines the first character is located at  $t_1 = 3/4$ , so that  $x < t_1$  and specifying that the first character must have probability  $3/4$ , i.e., it is 0. The decision point for the second character is the lower  $3/4$  of the interval determined by  $t_1$  i.e.,  $t_2 = 9/16$ . Again  $x \leq t_2$ , so the second character also must be 0. The decision point for the third character is located at:

$$t_3 = \frac{3}{4} \cdot t_2 = \frac{27}{64} < x = \frac{1}{2}$$

Because the coded point is larger than  $t_3$ , the third character is a 1 with probability equal to 0.25. The final decision point is divides the upper quarter of the interval in the proportion 3:1:

$$t_4 = \frac{135}{256} > x$$

So the fourth character also is 0.

In the second example, the code is

$$x = (0_{\Delta}11)_2 = \frac{3}{4}$$

The decision point which determines the first character is located at  $t_1 = \frac{3}{4}$ , so that  $x = t_1$ . By the convention specified for the unit interval, a point at the threshold is in the upper subinterval. This specifies that the first character has probability  $\frac{1}{4}$  and is 1. The decision point for the second character divides the first subinterval in proportion 3:1, so that:

$$t_2 = \frac{3}{4} + \left( \frac{3}{4} \cdot \frac{1}{4} \right) = \frac{15}{16}$$

Because  $x < t_2$ , the second character is 0. The decision point for the third character is located at:

$$t_3 = \frac{3}{4} + \left( \left( \frac{3}{4} \right)^2 \cdot \frac{1}{4} \right) = \frac{57}{64}$$

Because  $x < t_3$ , the third character is 0. The fourth threshold divides the fourth interval in the same ratio, and is located at:

$$t_4 = \frac{3}{4} + \left( \left( \frac{3}{4} \right)^3 \cdot \frac{1}{4} \right) = \frac{219}{256}$$

The coded point  $x < t_4$ , so the fourth character is 0.

An arithmetic code may be generated by redrawing the chart of the Huffman coding process to make a Huffman decision tree, where each bit represents the result



entered as the statistics change. A common application is the compression of data before sending it to magnetic tape/disk. One nice side effect is an increase in the effective transfer rate of the tape. The QIC (Quarter-Inch Cartridge) tape standard, a common backup medium for computer disks, uses a dictionary-based compression scheme which intermixes plain text and dictionary symbols by adding a one-bit symbol to distinguish the two.

Though these dictionary-based compression schemes were introduced for storing text files, they may be used for storing image or graphics files as well; they just assume that the binary files are, in fact, ASCII data. The GIF image compression scheme developed by Compuserve is a dictionary-based algorithm that was designed for graphics files and images.

### Lempel-Ziv-Welch (LZW) Coding

During the discussion of Huffman coding, it became obvious that it is desirable to encode an image while it is being read, without knowledge of either the histogram of the final image or the correlations among neighboring pixels. A simple method for this type of real-time encoding was developed by Lempel and Ziv (*A universal algorithm for sequential data compression*, **IEEE Trans. Info. Thry.** **23**, 337-343, 1977 and *Compression of individual sequences via variable-rate coding*, **IEEE Trans. Info. Thry.** **24**, 530-536, 1978), with a later extension by Welch (*A Technique for high-performance data compression*, **IEEE Computer** **17**, 8, 1984). Compression based on these works are referred to as LZ77, LZ78, and LZW, respectively. LZ77 allowed a 4 KByte dictionary of symbols for the codebook; matches within the text with data strings already seen are encoded as fixed-length pointers. LZ77 uses a large sliding text window (several thousand characters) that scans over the text, viewing a large block of recently coded text and a small block of text to be coded. Text in the look-ahead buffer is compared to the dictionary to find matches. This may take much time during the compression step, but decompression is not so constrained. The length of the longest possible match is limited by the size of the look-ahead buffer.

LZ78 abandoned the concept of the text buffer and built up the dictionary of character strings in increments of one character. The encoded strings may be very long, thus allowing significant compression if strings repeat themselves frequently. The LZW process is used in several of the common PC shareware utilities for compressing data files (*e.g.* PKARC, PKZIP, PAK). The ARC scheme was introduced in 1985 and dominated compression of MS-DOS files for several years, in no small part due to the fact that it was available as shareware. LZW generates a fairly efficient code as pixel gray levels are read, without prior knowledge of the image statistics. For an  $m$ -bit image ( $2^m$  gray levels), the LZW implementation is:

1. Select the number of bits  $k$  in the codeword, for a total of  $2^k$  codewords. The number  $k$  must be greater than  $m$  (preferably,  $k \gg m$ ).
2. Initialize the codes by setting the first  $2^m$  codes to the available gray levels – this ensures that legitimate codes exist for all pixels, even after the code table is filled.

3. Read the gray level of the first pixel; it is the first element in the string “ $S$ ” and is a member of the code table (from step 2).
4. If all pixels have been read, then output the code for “ $S$ ” (k bits) and quit.
5. If pixels are left, then read the gray level  $P$  of the next pixel and append to “ $S$ ” to create string “ $SP$ ”.
6. If “ $SP$ ” is in the table of codewords, then set “ $S$ ” = “ $SP$ ” and go to Step 4, otherwise continue
7. Append the codeword for “ $S$ ” (k bits) to the coded image.
8. If there are unused codewords left, then add “ $SP$ ” to the table.
9. Reset the string “ $S$ ” = “ $P$ ” and go to step 5.

Consider this simple example of a 36-pixel 2-bit image ( $m = 2, 4$  levels  $\alpha, \beta, \gamma, \delta$ ), which requires 72 bits to transmit as is:

$$f[n] = \alpha\alpha\alpha\beta\alpha\alpha\gamma\alpha\alpha\delta\alpha\alpha\alpha\beta\alpha\alpha\alpha\gamma\alpha\alpha\beta\alpha\alpha\beta\alpha\alpha\alpha\alpha\alpha\alpha\alpha\alpha\alpha\beta\alpha\alpha\gamma$$

Assume that  $k = 4$ , so that the codebook contains  $2^4 = 16$  symbols. From step 2, the first four codes are  $A \equiv \alpha$ ,  $B \equiv \beta$ ,  $C \equiv \gamma$ , and  $D \equiv \delta$ . The coding process proceeds in this manner:

1.  $S_1 = \alpha$ , in codebook as “A”  
 $P_1 = \alpha \rightarrow S_1P_1 = \alpha\alpha$ , not in codebook,  
 $S_1P_1 = \alpha\alpha$  is assigned to the first available symbol “E” in codebook,  
 First character in output is code for  $\alpha = \text{“A”}$   
 $S_2 \rightarrow P_1 = \alpha$
2.  $P_2 =$  third character = “ $\alpha$ ”  
 $S_2P_2 = \alpha\alpha\alpha$ , already exists in codebook as symbol “E”  
 $S_3 \rightarrow S_2P_2 = \alpha\alpha\alpha$
3.  $P_3 = \beta$ ,  $S_3P_3 = \alpha\alpha\beta$ , not in codebook  
 $S_3P_3 = \alpha\alpha\beta$  assigned to “F” in codebook  
 Second character in coded image = “E” = “ $\alpha\alpha$ ”  
 $S_4 \rightarrow S_3P_3 = \beta$
4.  $P_4 = \alpha$ ,  $S_4P_4 = \beta\alpha$ , not in codebook  
 $S_4P_4 = \beta\alpha$  assigned to “G” in codebook  
 Third character in coded image = “B” = “ $\beta$ ”  
 $S_5 \rightarrow P_4 = \alpha$
5.  $P_5 = \alpha$ ,  $S_5P_5 = \alpha\alpha$  exists in codebook as “E”  
 $S_6 \rightarrow \alpha\alpha$
6.  $P_6 = \gamma$ ,  $S_6P_6 = \alpha\alpha\gamma$ , not in codebook  
 $S_6P_6 = \alpha\alpha\gamma$  assigned to “H” in codebook  
 Fourth character in coded image = “E”, code = “AEBE...”  
 $S_7 = \gamma$

7...

After all pixels have been interrogated (and if I made no mistakes), the codebook is:

Symbol	String
A	$\alpha$
B	$\beta$
C	$\gamma$
D	$\delta$
E	$\alpha\alpha$
F	$\alpha\alpha\beta$
G	$\beta\alpha$
H	$\alpha\alpha\gamma$
I	$\gamma\alpha$
J	$\alpha\alpha\delta$
K	$\delta\alpha$
L	$\alpha\alpha\alpha$
M	$\alpha\beta$
N	$\beta\alpha\alpha$
O	$\alpha\alpha\gamma\alpha$
P	$\alpha\alpha\beta\alpha$

The entire coded message is “AEBECEDEAGHFPLEFH”. Note that new elements are added to the codebook quite rapidly, and the later elements typically represent longer and longer sequences. When the codebook is full, the last element (or the least used) can be deleted, and the process can proceed with the available elements. The coded image requires 17 characters at 4 bits per character, for a total of 68 bits, which is not much of a reduction when compared to the original quantity of 72 bits.

If a 3-bit code had been used, the last character in the codebook would be *H*, and the coded message would be AEBECEDEAGECFEEEFH, for a total of 18 3-bit characters, or 54 bits. Note that the total number of bits for the 3-bit code is less than for the 4-bit code (which is not the usual case). This illustrates the sensitivity of the process to the local image statistics.

As mentioned above, the LZW algorithm was used in most PC file compressors (archiving programs such as “PKARC” and “PKZIP”) back in the days of small disk drives. In those applications, the files consisted of 1-bit ASCII characters. LZW with a 13-bit codeword was known as “squashing”, while 12-bit LZW was “crunching”.

## 20.5 Transform Coding

The various lossless compression schemes (Huffman, arithmetic, and LZW coding) are useful for compressing images with clustered histograms (scalar or vector). However, the compression ratios are fixed by the statistics of the images. In many cases, “lossy” coding is useful where *nonredundant* information is discarded. If the loss of this information is not objectionable to the viewer, then the reduction in storage requirements may well be beneficial. Probably the most familiar example of transform compression is JPEG encoding.

One convenient method for implementing lossy coding is to first construct a new representation of the original image via an invertible image transformation. Such a transformation may be a gray-scale remapping (lookup tables) for monochrome images, a color-space transformation for color images, or a shift-invariant or shift-variant spatial transformation. These transformations “reorganize” the gray values of the image and thus change the correlation properties of the image. It is possible to compress the reorganized gray values using one (or more) of the schemes already considered.

In the discussion of image processing operations, we have seen that images may be recast into a different (and possibly equivalent) form by an image transformation. The general form for the transformation of a 1-D vector is:

$$F[\ell] = \sum_n f[n] m[\ell, n]$$

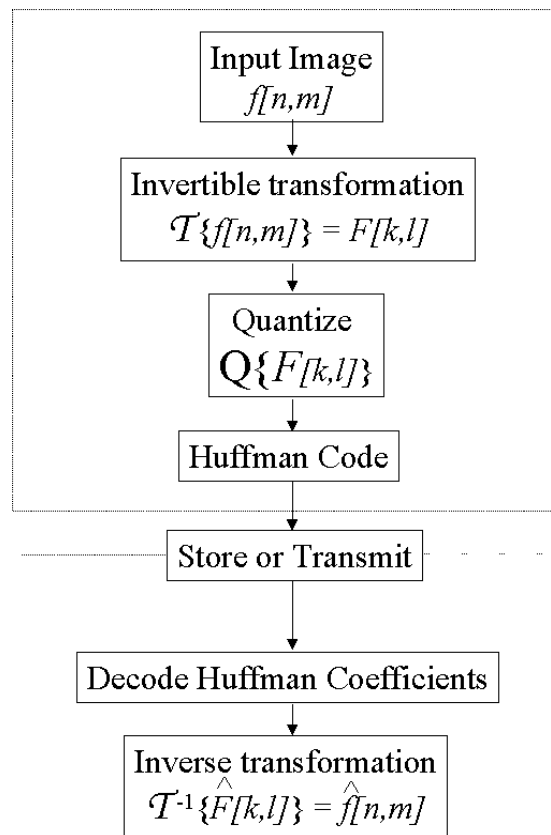
where  $m[\ell, n]$  is the 2-D “reference” or “mask” function of the transform. A familiar such function is the Fourier transform, where  $m[\ell, n] = \exp[-2\pi i \frac{n\ell}{N}]$ . If the transform is invertible, then there exists a mask function  $M[n, \ell]$  such that:

$$f[n] = \sum_{\ell} F[\ell] M[n, \ell]$$

The transformation is a space-invariant convolution if the mask function  $m[\ell, n]$  has the form of a convolution kernel:  $m[\ell - n] = h[\ell - n]$ . In the study of linear systems, it is demonstrated that convolution with an impulse response  $h[n]$  is invertible if the transfer function  $H[k]$  is nonzero everywhere. A space-variant transformation is invertible if the set of mask functions  $m[\ell, n]$  is complete. In either case, the gray level  $F$  at pixel  $\ell_1$  of the transformed image is a measure of the similarity of the input image  $f[n]$  and the specific mask  $m[\ell_1, n]$ . As such, the transformed image pixels  $F[\ell_1]$  are measures of the correlation of  $f[n]$  and  $m[\ell_1, n]$ ; the greater the similarity between image and mask, the larger the amplitude of  $F$  at  $\ell_1$ .

The goal of transform coding is to generate an image via an invertible transform whose histogram is less flat (more clustered) than that of the original image, thus having less entropy and requiring fewer bits for storage/transmission. Many authors describe the operation as *compacting* the image information into a small number of coefficients (corresponding to pixels of the transformed image), though I prefer the

picture of the transform as generating an image with a more clustered histogram. Though transform coding often clusters the histogram, the lunch is not free; we pay by increasing the number of possible gray levels to be encoded and thus the number of bits per recorded pixel if the image is to be recovered without loss. In the example of encoding the image derivative considered below, the transformed image  $F[\ell]$  may occupy up to twice as many levels as the input. If the transform  $F[\ell]$  is requantized to the same number of levels before storage/transmission, some information is lost and  $f[n]$  cannot be recovered perfectly. This is one example of *lossy* coding where the compression efficiency can be quite high. If the statistics of the source can be quantified (e.g., discrete memoryless source, Markov source, etc.), it is possible to quantify the effect of reducing the number of encoded levels on the fidelity of the final image. These studies form a branch of information theory which is known as *rate distortion theory*.

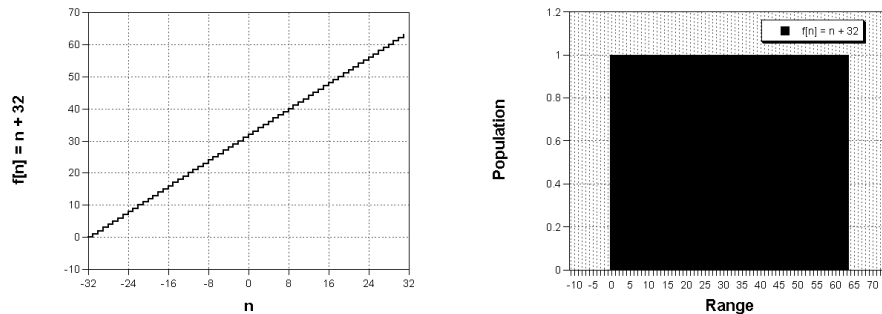


Block diagram of transform compression. The original image  $f[n,m]$  is converted to a different coordinate system via the invertible transformation  $\mathcal{T}\{f[n,m]\} = F[k,\ell]$ , which is then quantized and Huffman coded. The quantization step ensures that the recovered image is generally an estimate  $\hat{f}[n,m]$

To introduce the concept of compression via image transformations, consider a 64-pixel, 6-bit, 1-D image  $f[n]$  of a linear ramp in the interval  $-32 \leq n \leq 31$ , where  $n$  is the pixel address:

$$f[n] = n + 32; \quad -32 \leq n \leq 31, \quad 0 \leq f[n] \leq 63$$

This histogram of this image is flat with a population of one pixel per bin, and therefore the information content of the image is 6 bits per pixel.



1-D 6-bit “ramp” image  $f[n] = n + 32$  for  $-32 \leq n \leq 31$ . The histogram is “flat” with one count per gray level. The information content is 6 bits per pixel.

Because the slope of  $f[n]$  is constant, the image of its derivative is constant also. Recall that discrete differentiation can be implemented as a convolution:

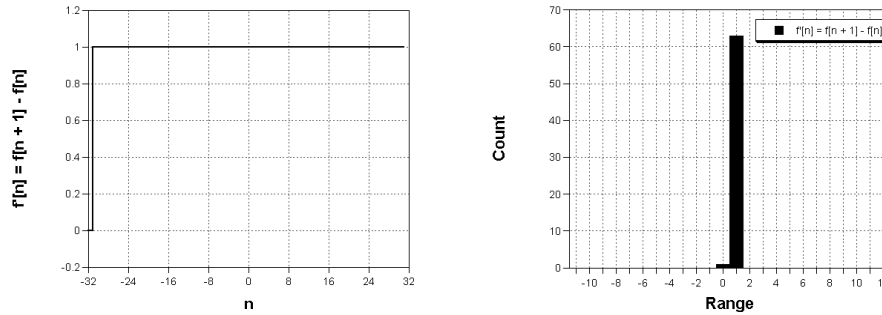
$$\frac{df}{dn} = f[n] * h[n]$$

$$h[n] = \begin{array}{|c|c|c|} \hline +1 & -1 & 0 \\ \hline \end{array}$$

For this derivative operation, the discrete transfer function (discrete Fourier transform of the impulse response  $h[n]$ ) is:

$$H[k] = |k|$$

and is nonzero at all samples  $k$  except at the origin (i.e., at zero frequency – the constant part or average value). The derivative of a 1-D image is invertible if the average (or initial) value is known as a boundary condition. In this example, the histogram of the derivative has only two occupied gray levels, and the entropy of the derivative image is 0.116 bits/pixel. This transformed image can be encoded by one-bit characters with a compression efficiency of  $\eta = \frac{6}{0.116} \cong 52$ , which means that this coding scheme produced a bit rate that is very much smaller than the Shannon limit. How is this possible? Because we encoded a quality of *groups* of characters rather than the individuals.



Derivative of the ramp image:  $f'[n] = f[n] - f[n - 1]$ ; the “gray value” of the first pixel is “0” and all of the rest are “1”. The histogram has 63 pixels at 1 and 1 pixel at 0, for an information content of 0.116 bits per pixel.

Obviously, derivative images can have negative gray values (though it did not in this case). In fact, if the dynamic range of  $f[n]$  is 64 levels in the interval  $[0,63]$  (6 bits per pixel), the theoretical dynamic range of the derivative image is 127 levels in the interval  $[-63,63]$  (not quite 7 bits per pixel). So although the histogram of the derivative image may have less entropy if the pixels of the original image are well correlated, an additional bit per pixel must be stored if the image is to be recovered without loss. If some error in the uncompressed image is tolerable, sparsely populated levels in the transform may be substituted with a similar gray level, or levels with small amplitudes may be quantized to 0. These can significantly reduce the information content. The latter process is called *coring* by some in the image compression community.

The process of encoding the derivative image rather than the original is the basis for both *run-length encoding* and *differential pulse code modulation (DPCM)*. These are examples of *predictive coding*, where a reduction in entropy is obtained by transmitting the difference between the actual gray value at a pixel and a prediction obtained by some rule. In run-length encoding of images with large uniform regions (*e.g.*, binary text images for FAX machines), the transmitted data are the number of consecutive pixels with the same gray level before the next switch. If the strings of 0s and 1s are long, run-length encoding can reduce the data stream significantly. In DPCM, the gray value at a pixel is predicted from some linear combination of previous gray levels; the error  $\epsilon$  between the actual gray level and the prediction is quantized and encoded. The predictors of the pixel gray value  $f[x, y]$  may include one or several previous pixels, including  $f[x - 1, y]$ ,  $f[x - 1, y - 1]$ ,  $f[x, y - 1]$ , and  $f[x + 1, y - 1]$ , as shown below:

$f[x - 1, y - 1]$	$f[x, y - 1]$	$f[x + 1, y - 1]$
$f[x - 1, y]$	$f[x, y]$	

The predictor may be expressed as a linear combination of these levels:

$$f[x, y] = \sum_{i,j} a_{ij} f[x-i, y-j]$$

where the  $a_{ij}$  are the weights of the linear combination. Among the predictors commonly used are:

- (1) 1-D first-order (based on one pixel):  $f[x_0, y_0] = f[x_0 - 1, y_0]$
- (2) 2-D second-order (based on two pixels):  $f[x_0, y_0] = \frac{1}{2} (f[x_0 - 1, y_0] + f[x_0, y_0 - 1])$
- (3) 2-D third-order (based on three pixels):

$$f[x_0, y_0] = \frac{1}{4} (3 f[x_0 - 1, y_0] - 2 f[x_0 - 1, y_0 - 1] + 3 f[x_0, y_0 - 1])$$

In the first case, the difference between the actual and predicted gray values is just the discrete first derivative:

$$\epsilon[x_0, y_0] = f[x_0, y_0] - f[x_0, y_0] = f[x_0, y_0] - f[x_0 - 1, y_0] = \left. \frac{\partial f}{\partial x} \right|_{x=x_0}$$

The 2-D predictor usually improves compression efficiency significantly over 1-D predictors for real images.

In *adaptive DPCM*, the mathematical form of the predictor may vary based on the image structure. The compression in DPCM results because the prediction error is quantized to fewer levels than the original image data. Note that the final image may exhibit grainy or contouring errors if the minimum quantization level is coarse and the gray levels vary slowly across the image.

### 20.5.1 Color-Space Transformations

$$\begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.274 & -0.322 \\ 0.211 & -0.523 & 0.312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} Y \\ I \\ Q \end{bmatrix}$$

The “luminance”  $Y$  is a weighted sum of the color-value triplet  $[R, G, B]$ . Note that the weights sum to one, which means that a gray pixel with the values  $R = G = B$  will have the same luminance value.

$$Y = 0.299R + 0.587G + 0.114B$$

$I$  and  $Q$  are the “chrominance” values and are weighted differences of the color-value triplet, where the weights sum to zero.  $I$  is a weighted sum of green and blue subtracted from a weighted red; in other words, it may be thought of as Red - Cyan. The  $Q$  channel is weighted green subtracted from a weighted combination of red and blue, or Magenta - Green.

### 20.5.2 Space-Variant Transformations

Most authors consider only space-variant operations as transforms for image coding; for 1-D images, the form of the space-variant operation is:

$$F[k] = \sum_n f[n] p[n; k]$$

while for 2-D images, the form of the space-variant operation is:

$$F[k, \ell] = \sum_{n,m} f[n, m] p[n, m; k, \ell]$$

In words, the 2-D transform is the product of a 4-D matrix and the 2-D input. The most common such transform in imaging is the discrete Fourier transform (DFT); the mask  $m[n, k]$  for the 1-D DFT is the set of 1-D complex-valued sinusoids with spatial frequency  $k/N$ :

$$\begin{aligned} p[n; k] &= \exp\left[-i\frac{2\pi nk}{N}\right] \\ &= \cos\left(\frac{2\pi nk}{N}\right) - i \sin\left(\frac{2\pi nk}{N}\right) \end{aligned}$$

The mask  $p[n, m; k, \ell]$  for 2-D images is the set of 2-D complex-valued sinusoids; they vary in a sinusoidal fashion along one direction and are constant in the orthogonal direction. For an  $N \times N$  input image, the mathematical expression for the mask function is:

$$\begin{aligned} p[n, m; k, \ell] &= \exp\left[-\frac{2\pi i (nk + m\ell)}{N}\right] \\ &= \cos\left[2\pi\left(nk + \frac{m\ell}{N}\right)\right] - i \sin\left[2\pi\left(nk + \frac{m\ell}{N}\right)\right] \end{aligned}$$

The spatial frequencies of the sinusoidal mask indexed by  $k$  and  $\ell$  are respectively  $\xi = \frac{k}{N}$  and  $\eta = \frac{\ell}{N}$ . The gray level of each pixel in the transformed image  $F[k, \ell]$  describes the degree of similarity between  $f[n, m]$  and that specific 2-D sinusoid. Recall that the amplitudes of all pixels in a 1-D sinusoid can be completely specified by three numbers: the magnitude, spatial frequency, and phase. In other words, the gray value of a particular sample of a sinusoid is determined completely by any other pixel if the parameters of the sinusoid are known; a perfect interpixel correlation exists among the amplitudes. The DFT operation compresses the image information into the number of bits required to represent those three parameters. Therefore, an image composed of a small number of sinusoidal components can be compressed to a very significant degree by converting to its Fourier representation. We begin by considering a simple 1-D case; the image to be compressed has 64 pixels and is

a 1-D sinusoid of period 32 pixels, as shown below. The image has been sampled but not quantized, *i.e.*, all real numbers between 0 and 31 are allowed gray levels. Because the Shannon entropy (information content) is defined as a sum over discrete probabilities (gray levels), it strictly can not apply to images with real-valued gray levels; the image must be quantized to calculate the entropy. The histogram of the sinusoid after quantizing to 64 bins is shown; note that it is approximately flat:

$$f[n] = 16 + 15 \cos\left(\frac{2\pi n}{32}\right), 0 \leq f[n] \leq 31$$

One definition of the 1-D DFT is:

$$F[\xi] = F[k \cdot \Delta\xi] \implies F[k] = \sum_{n=-\frac{N}{2}}^{\frac{N}{2}-1} f[n] \exp\left[-\frac{2\pi ink}{N}\right]$$

Note that the transform often includes a scale factor of  $N^{-1}$  that is not included here. In words, the transform is the sum of gray values of the product of the input image and the mask function, which is real-valued in the interval  $[-1, +1]$ . In general, the transform  $F[k]$  generates noninteger values that typically lie outside the dynamic range of  $f[n]$ . The amplitude of the transform at  $k = 0$  is the sum of the gray values of the image. In the example under consideration which has a mean gray value of 16, the DC value of the DFT is:

$$F[k = 0] = 64 \text{ pixels} \cdot 16 \text{ (mean gray)} = 4096$$

The entire discrete spectrum  $F[k]$  has 61 samples with value zero, two with value 480 (located at  $k = \pm 2$  so that  $\xi = \pm \frac{k}{N} = \pm \frac{1}{32}$ ) and one with value 4096 at  $k = 0$ . The histogram of  $F[k]$  generally needs an infinite number of bins to account for the continuously valued range of  $F[k]$ , but is certainly much less flat than the histogram of  $f[n]$ ; thus there is less entropy in  $F[\xi]$ .

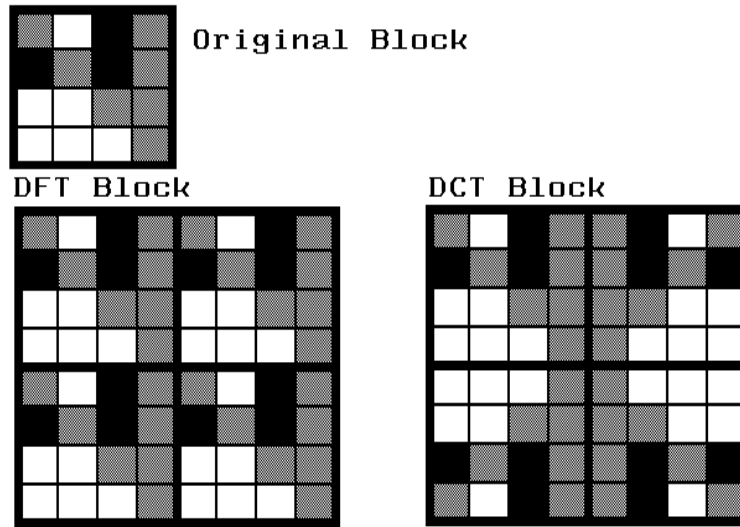
The sinusoid after quantization to 32 gray levels (5 bits per pixel) and the resulting histogram are shown below. The original image entropy is  $3.890 \frac{\text{bits}}{\text{pixel}}$ . Since the image is quantized, it no longer is a sampled pure sinusoid and the Fourier transform includes extra artifacts. The histogram of the quantized transform indicates that the information content of the transform is only  $0.316 \frac{\text{bits}}{\text{pixel}}$ .

The 1-D discrete Fourier transform of an  $N$ -pixel real-valued array is an  $N$ -pixel complex-valued array. Because each complex-valued pixel is represented as a pair of real-valued numbers, the DFT generates twice as much data as the input array. However, the DFT of a real-valued array is *redundant*, meaning that values in the resulting DFT array are repeated. The real part of the DFT of a real-valued function is even and the imaginary part is odd. Therefore, half of the data in the DFT of a real-valued array may be discarded without loss.

### 20.5.3 Block DFT Coding via DCT

Because the statistical properties of typical images vary across the image (i.e., the statistics are shift variant), it is common to apply the selected transform to local blocks of pixels (often of size  $8 \times 8$  or  $16 \times 16$ ) that are coded individually using a spatial transform (to reduce gray-level redundancy), followed by Huffman coding. The discrete Fourier transform (DFT) is a possible such transform, but its advantage of generating an equivalent image which exhibits a clustered histogram often is offset somewhat by its assumption that an input array of size  $N \times M$  pixels actually is infinite array that is periodic over  $N \times M$  blocks of pixels. In words, the DFT assumes that the gray value of a pixel off the edge on one side of the array is identical to the gray value on the edge of the other side of the array. Of course, it is common for pixels at opposite edges of the array to have different gray levels (for sky at the top and for ground at the bottom, for example). The gray-level transitions at the boundaries of the periods of the array generate artifacts in the DFT known as leakage, or false frequency components. Though these false frequency terms are necessary to generate the true sharp edge boundaries of the block, the additional samples of the DFT with non-zero amplitudes increase redundancy (and thus the entropy) of the transformed image. Therefore, the potential efficiency of compression using the DFT often suffers. An additional problem arises because the DFT of a real-valued discrete input image is a complex-valued Hermitian array. The symmetry of the transform array (even real part and odd imaginary part) ensures that only half of each part need be stored, but the histograms of both parts must be accounted when computing the entropy of the transform coefficients.

To reduce the impact of the sharp transitions at the edges of the blocks, as well as to obtain a transform that is real-valued for a real-valued input image, the *discrete cosine transform* (DCT) may be used instead of the DFT. The DCT has become very important in the image compression community, being the basis transformation for the JPEG and MPEG compression standards. The DCT of an  $M \times M$  block may be viewed as the DFT of a synthetic  $2M \times 2M$  block that is created by replicating the original  $M \times M$  block after folding about the vertical and horizontal edges:



*The original  $4 \times 4$  block of image data is replicated 4 times to generate an  $8 \times 8$  block of data via the DFT format and an  $8 \times 8$  DCT block by appropriate reversals. The transitions at the edges of the  $4 \times 4$  DCT blocks do not exhibit the “sharp” edges in the  $4 \times 4$  DFT blocks.*

The resulting  $2M \times 2M$  block exhibits smaller discontinuities at the edges. The symmetries of the Fourier transform for a real-valued image ensure that the original  $M \times M$  block may be reconstructed from the DCT of the  $2M \times 2M$  block.

Consider the computation of the DCT for a 1-D  $M$ -pixel block  $f[n]$  ( $0 \leq n \leq M - 1$ ). The  $2M$ -pixel synthetic array  $g[n]$  is indexed over  $n$  ( $0 \leq n \leq 2M - 1$ ) and has the form:

$$g[n] = f[n] \text{ for } 0 \leq n \leq M - 1$$

$$g[n] = f[2M - 1 - n] \text{ for } M \leq n \leq 2M - 1.$$

In the case  $M = 8$ , the array  $g[n]$  is defined:

$$g[n] = \begin{cases} f[n] & \text{for } 0 \leq n \leq 7 \\ f[15 - n] & \text{for } 8 \leq n \leq 15 \end{cases}$$

The values of  $g[n]$  for  $8 \leq n \leq 15$  is a “reflected replica” of  $f[n]$ :

$$\begin{aligned} g[8] &= f[7] \\ g[9] &= f[6] \\ g[10] &= f[5] \\ g[11] &= f[4] \\ g[12] &= f[3] \\ g[13] &= f[2] \\ g[14] &= f[1] \\ g[15] &= f[0] \end{aligned}$$

If the “new” array  $g[n]$  is assumed to be periodic over  $2M$  samples, its amplitude is defined for all  $n$ , e.g.,

$$g[n] = f[-1 - n] \text{ for } -M \leq n \leq -1 \implies -16 \leq n \leq -1$$

$$g[n] = f[n + 2M] \text{ for } -2M \leq n \leq -M - 1 \implies -32 \leq n \leq -17$$

Note that the 16-sample block  $g[n]$  is NOT symmetric about the origin of coordinates because  $g[-1] = g[0]$ ; to be symmetric,  $g[-\ell]$  would have to equal  $g[+\ell]$ . For example, consider a 1-D example where  $f[n]$  is an 8-pixel ramp as shown:

The  $2M$ -point representation of  $f[n]$  is the  $g[n]$  just defined:

$$g[n] = \begin{cases} f[n] & \text{for } 0 \leq n \leq 7 \\ f[2M - 1 - n] & \text{for } 8 \leq n \leq 15 \end{cases}$$

If this function were symmetric (even), then circular translation of the 16-point array by 8 pixels to generate  $g[n - 8 \bmod 16]$  also be an even function.

From the graph, it is apparent that the translated array is not symmetric about the origin; rather, it has been translated by  $-\frac{1}{2}$  pixel from symmetry in the  $2M$ -pixel array. Thus define a new 1-D array  $c[n]$  that is shifted to the left by  $\frac{1}{2}$  pixel:

$$c[n] = g\left[n - \frac{1}{2}\right]$$

This result may seem confusing at first; how can a sampled array be translated by  $\frac{1}{2}$  pixel? For the answer, consider the continuous Fourier transform of a sampled array translated by  $\frac{1}{2}$  unit:

$$\begin{aligned} \mathcal{F}_1 \{c[n]\} &\equiv C[\xi] = \mathcal{F}_1 \left\{ g\left[x - \frac{1}{2}\right] \right\} = \mathcal{F}_1 \left\{ g[x] * \delta\left[x - \frac{1}{2}\right] \right\} \\ &= G[\xi] \cdot \exp\left[-2\pi i \xi \cdot \frac{1}{2}\right] C[\xi] = G[\xi] \cdot \exp[-i\pi \xi] \end{aligned}$$

Thus the effect of translation by  $\frac{1}{2}$  pixel on the transform is multiplication by the specific linear-phase factor:

$$\exp[-i\pi\xi] = \cos[\pi\xi] - i \sin[\pi\xi].$$

The  $2M$ -point DFT of the symmetric discrete array (original array translated by  $\frac{1}{2}$  pixel) has the form:

$$\begin{aligned} F_{2M}\{c[n]\} &= \mathcal{F}_{2M}\left\{g\left[n - \frac{1}{2}\right]\right\} = \mathcal{F}_{2M}\left\{g[n] * \delta\left[n - \frac{1}{2}\right]\right\} \\ &= G[k] \cdot \exp\left[-i\pi\left(\frac{k}{2M}\right)\right] C[k] = G[k] \cdot \exp\left[-\frac{i\pi k}{2M}\right] \\ &= G[k] \cdot \left(\cos\left[\frac{\pi k}{2M}\right] - i \sin\left[\frac{\pi k}{2M}\right]\right) \end{aligned}$$

where the continuous spatial frequency  $\xi$  has been replaced by the sampled frequency  $\frac{k}{2M}$ . This function  $C[k]$  is the DCT of  $f[n]$ . Because the  $2M$ -point translated function  $c[n]$  is real and even, so must be the  $2M$ -point discrete spectrum  $C[k]$ ; therefore only  $M$  samples of the spectrum are independent. This array is the DCT of  $f[n]$ .

### Steps in Forward DCT

To summarize, the steps in the computation of the 1-D DCT of an  $M$ -point block  $f[n]$  are:

1. create a  $2M$ -point array  $g[n]$  from the  $M$ -point array  $f[n]$  :

$$\begin{aligned} g[n] &= f[n] : 0 \leq n \leq M - 1 \\ g[n] &= f[2M - 1 - n] : M \leq n \leq 2M - 1 \end{aligned}$$

2. compute the  $2M$ -point DFT of  $g[n] = G[k]$
3. the  $M$ -point DCT  $C[k] = \exp\left[-\frac{i\pi k}{2M}\right] \cdot G[k]$  for  $0 \leq k \leq M - 1$

The entire process may be cast into the form of a single equation, though the algebra required to get there is a bit tedious,

$$C[k] = \sum_{n=0}^{M-1} 2 f[n] \cos\left(\pi k \cdot \frac{2n+1}{2M}\right) \quad \text{for } 0 \leq k \leq M - 1$$

### Steps in Inverse DCT

The inverse DCT is generated by applying the procedures in the opposite order:

1. create a  $2M$ -point array  $G[k]$  from the  $M$ -point DCT  $C[k]$ :

$$G[k] = \exp\left[+\frac{i\pi k}{2M}\right] \cdot C[k] \text{ for } 0 \leq k \leq M-1$$

$$G[k] = -\exp\left[+\frac{i\pi k}{2M}\right] \cdot C[2M-k] \text{ for } M+1 \leq k \leq 2M-1$$

2. compute the inverse  $2M$ -point DFT of  $G[k] \rightarrow g[n]$
3.  $f[n] = g[n]$  for  $0 \leq n \leq M-1$

The single expression for the inverse DCT is:

$$f[n] = \frac{1}{M} \sum_{k=0}^{M-1} w[k] C[k] \cos\left(\pi k \cdot \frac{2n+1}{2M}\right) \text{ for } 0 \leq n \leq M-1$$

where  $w[k] = \frac{1}{2}$  for  $k=0$  and  $w[k] = 1$  for  $1 \leq k \leq M-1$ .

### Forward DCT OF 2-D Array

The corresponding process to compute the 2-D DCT  $C[k, \ell]$  of an  $M \times M$ -pixel block  $f[n, m]$  is:

1. create the  $2M \times 2M$ -pixel  $g[n, m]$ :

$$g[n, m] = f[n, m] : 0 \leq n, m \leq M-1$$

$$g[n, m] = f[2M-1-n, m] : M-1 \leq n \leq 2M-1, 0 \leq m \leq M-1$$

$$g[n, m] = f[n, 2M-1-m] : M-1 \leq n \leq 2M-1, 0 \leq m \leq M-1$$

$$g[n, m] = f[2M-1-n, 2M-1-m] : M-1 \leq n, m \leq 2M-1 \leq k, \ell \leq M-1$$

2. compute the  $2M$ -point DFT of  $g[n, m] \rightarrow G[k, \ell]$
3.  $C[k, \ell] = \exp\left[-\frac{i\pi(k+\ell)}{2M}\right] \cdot G[k, \ell]$  for  $0 \leq k, \ell \leq M-1$

**Inverse DCT OF 2-D Array**

1. create a  $2M \times 2M$  array  $G[k, \ell]$  from the DCT  $C[k, \ell]$ :

$$G[k, \ell] = \exp \left[ \frac{i\pi (k + \ell)}{2M} \right] \cdot C[k, \ell] : 0 \leq k, \ell \leq M - 1$$

$$G[k, \ell] = -\exp \left[ +\frac{i\pi (k + \ell)}{2M} \right] \cdot C[2M - n, m] : M - 1 \leq n \leq 2M - 1, 0 \leq m \leq M - 1$$

$$G[k, \ell] = -\exp \left[ +\frac{i\pi (k + \ell)}{2M} \right] \cdot C[n, 2M - m] : M - 1 \leq n \leq 2M - 1, 0 \leq m \leq M - 1$$

$$G[k, \ell] = \exp \left[ +\frac{i\pi (k + \ell)}{2M} \right] \cdot C[2M - n, 2M - m] : M - 1 \leq k, \ell \leq 2M - 1$$

2. compute the inverse  $2M$ -point DFT of  $G[k, \ell] \rightarrow g[n, m]$

3.  $f[n, m] = g[n, m]$  for  $0 \leq n, m \leq M - 1$

The form of the forward DCT may be written more simply as:

$$F[k, \ell] = 4 \frac{w[n] w[m]}{N^2} \sum_{n=0}^{N-1} \sum_{m=0}^{N-1} f[n, m] \cos \left[ \frac{(2n+1)k\pi}{2N} \right] \cos \left[ \frac{(2m+1)\ell\pi}{2N} \right]$$

$$\text{where: } w[j] \equiv \begin{cases} \frac{1}{\sqrt{2}} & \text{if } j = 0 \\ 1 & \text{if } j = 1, \dots, N-1 \end{cases}$$

and the corresponding form of the inverse 2D DCT is:

$$f[n, m] = \sum_{k=0}^{N-1} \sum_{\ell=0}^{N-1} w[k] w[\ell] F[k, \ell] \cos \left[ \frac{(2k+1)k\pi}{2N} \right] \cos \left[ \frac{(2\ell+1)\ell\pi}{2N} \right]$$

The action of the 2-D  $8 \times 8$  block DCT will be detailed for blocks in the  $64 \times 64$  5-bit image LIBERTY.



The corresponding  $8 \times 8$  DCT block is:

$$\begin{bmatrix} 248 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

The amplitude of the upper-left pixel in the DCT is the zero-frequency (DC) term; its amplitude is eight times the average gray value of the block. The other terms in the DCT are proportional to the amplitude of oscillating sinusoidal components and often are called the AC terms; in this constant block, the oscillating terms have null amplitude because all gray values in the block are equal.

The gray values of a second  $8 \times 8$  block located near the center of the image are:

$$\begin{bmatrix} 9 & 7 & 21 & 12 & 11 & 13 & 15 & 13 \\ 31 & 8 & 9 & 13 & 9 & 12 & 12 & 11 \\ 31 & 22 & 7 & 10 & 13 & 11 & 11 & 10 \\ 23 & 8 & 10 & 7 & 10 & 13 & 10 & 11 \\ 11 & 31 & 31 & 26 & 9 & 8 & 9 & 10 \\ 31 & 31 & 31 & 14 & 14 & 20 & 9 & 8 \\ 31 & 31 & 28 & 10 & 29 & 31 & 31 & 30 \\ 31 & 31 & 21 & 18 & 31 & 31 & 30 & 29 \end{bmatrix}$$

with an average gray value of 17.95.

The amplitudes of the DCT of the block near the center of the image are approximately (rounded to one decimal place):

$$\begin{bmatrix} 143.6 & 14.6 & 9.7 & 5.7 & -3.3 & -1.1 & 4.1 & 2.9 \\ -30.9 & -0.1 & -0.9 & -0.7 & 5.0 & 6.5 & 1.1 & -1.0 \\ 13.9 & -9.3 & 1.3 & 3.7 & 0.3 & -0.4 & 0.5 & 3.0 \\ -1.5 & 3.8 & -5.7 & -8.6 & -6.1 & 0.3 & 2.9 & 0.5 \\ -4.3 & -4.7 & -5.2 & -6.1 & -1.2 & 0.5 & -0.9 & -0.6 \\ 0.3 & -5.8 & 2.2 & 3.2 & -1.1 & -1.8 & -2.0 & 3.7 \\ -1.5 & 4.7 & -1.0 & 0.5 & -1.1 & -1.6 & -0.7 & -1.0 \\ 5.9 & -0.1 & -1.8 & -5.4 & -2.4 & -2.4 & -4.2 & -0.8 \end{bmatrix}$$

Again, the amplitude of the sample in the upper-left corner is eight times the average gray value of 17.95. The other 63 coefficients (the AC terms) are bipolar. A negative AC coefficient means that the particular cosine term oscillates out of phase by  $\pi$  radians. The rate of oscillation of an AC component increases with distance from the DC term, and the direction of oscillation is determined by the orientation relative to the DC term at the origin; cosines that oscillate horizontally are specified by the amplitudes along the first row and those that oscillate vertically by the amplitudes in the first column. Note that the amplitudes of higher-frequency AC components (away from the upper-left corner) tend to be smaller than the low-frequency terms (towards the upper left). This is the usual result for realistic imagery, and is utilized to obtain additional compression in the JPEG standard.

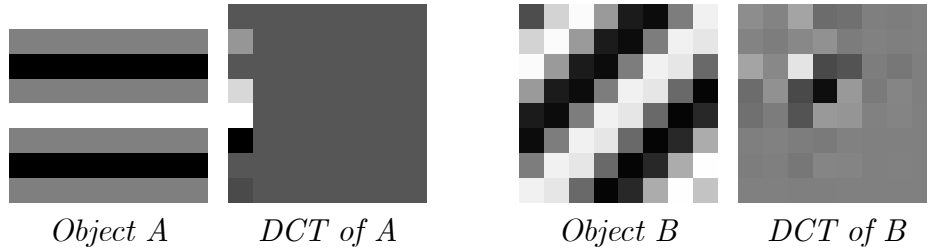
The DCT is useful as a transform for image compression because:

1. (a) it is real valued,
- (b) the amplitudes are proportional to the amplitudes of sinusoids with different oscillation rates, and
- (c) the amplitudes of higher-frequency terms are smaller for realistic imagery.

Object B:  $f[n, m] = \cos[2\pi(n\xi' + m\eta')]$ ,  $\xi' = \frac{1}{4} \sin\left[\frac{\pi}{4}\right] = \frac{\sqrt{2}}{8} \cong 0.1768$ , oscillates two times in diagonal direction, period  $X' \cong 0.1768^{-1} = 5.6561$ ,  $\eta' = \frac{1}{4} \cos\left[\frac{\pi}{4}\right]$

The “low-frequency” content of the signal concentrates the DCT amplitude near the origin, but the fact that the array is only pseudoperiodic over 8 samples produces larger amplitudes at more pixels in the DCT array.

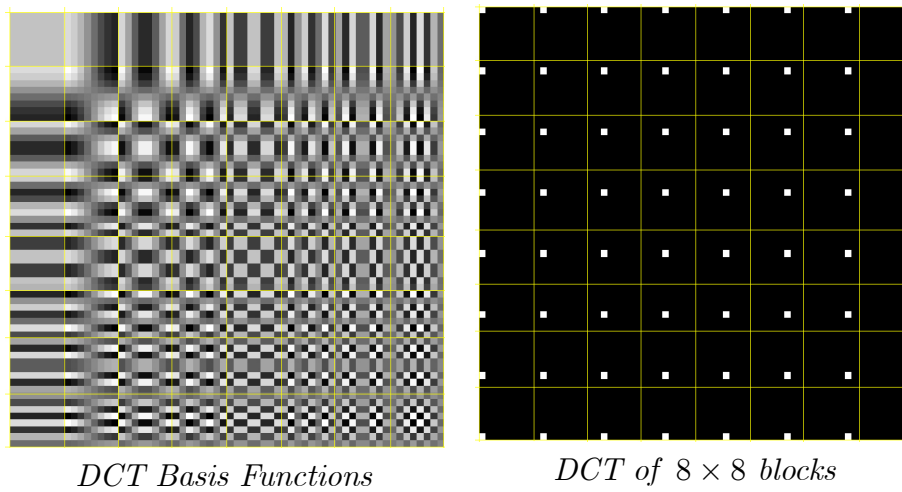
Examples of Individual  $8 \times 8$  blocks and their associated DCT arrays



Object A:  $f[n, m] = \cos \left[ 2\pi \left( 0n + \frac{m}{4} \right) \right]$ , oscillates “vertically” two times in 8 samples,  $\xi = 0, \eta = \frac{1}{4}$  average gray value = 0

Note that the DCT array is zero except in the first vertical column (horizontal frequency = 0 cycles per pixel)

In the next example, we compute the DCT of the individual  $8 \times 8$  “basis images”. Each of these images produces an  $8 \times 8$  DCT that has all pixels at zero except one. The object composed of these “blocks” is shown on the left and the DCT on the right. The low-frequency terms appear in the upper left and the highest-frequency terms to the right and bottom. This observation provides the basis for the JPEG encoding standard that is considered next.



## 20.6 JPEG Image Compression of Static Images

??

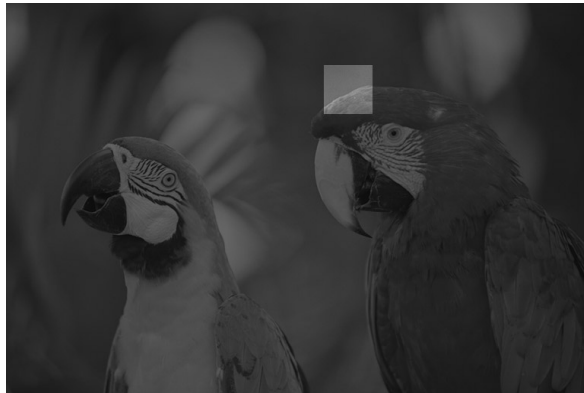
A standard has been developed by the “Joint Photographic Experts Group” and has become very common since the early 1990s. In its original form, it was based on the DCT, to the point where the method is now sometimes called “JPEG (DCT)”. This standard is based on the properties of human vision where the sensitivity of the eye generally decreases with increasing spatial frequency.



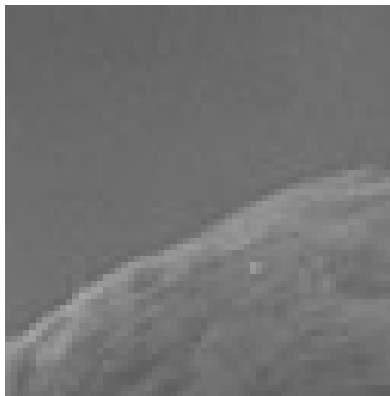
*8-bit grayscale image used to illustrate JPEG image compression.*

### 20.6.1 Example 1: “Smooth” Region

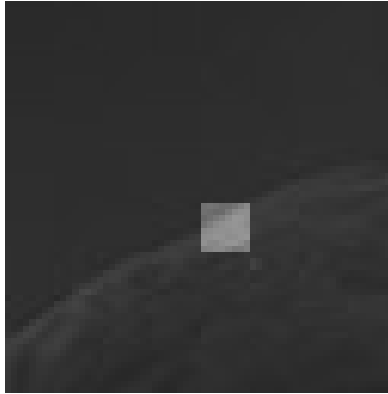
#### JPEG Encoding



*64 × 64 block of pixels in illustration*



Magnified view of 64 × 64 block of pixels



*Single  $8 \times 8$  block of pixels highlighted*

“Smooth” Region of image of “Parrots”



*Magnified view of  $8 \times 8$  block*

Gray values of  $8 \times 8$  block

$$f[n, m] = \begin{array}{cccccccc} 110 & 111 & 111 & 110 & 111 & 109 & 112 & 117 \\ 112 & 115 & 111 & 110 & 108 & 111 & 129 & 139 \\ 111 & 112 & 107 & 109 & 120 & 140 & 142 & 142 \\ 108 & 107 & 119 & 126 & 142 & 150 & 145 & 143 \\ 111 & 131 & 137 & 141 & 152 & 148 & 147 & 140 \\ 138 & 138 & 141 & 148 & 144 & 140 & 147 & 148 \\ 141 & 143 & 150 & 144 & 139 & 149 & 151 & 148 \\ 143 & 136 & 137 & 142 & 148 & 148 & 138 & 130 \end{array}$$

The midgray value of 128 is subtracted from the values in the  $8 \times 8$  block to produce

bipolar data

$$f[n, m] - 128 = \begin{matrix} -18 & -17 & -17 & -18 & -17 & -19 & -16 & -11 \\ -16 & -13 & -17 & -18 & -20 & -17 & 1 & 11 \\ -17 & -16 & -21 & -19 & -8 & 12 & 14 & 14 \\ -20 & -21 & -9 & -2 & 14 & 22 & 17 & 15 \\ -17 & 3 & 9 & 13 & 24 & 20 & 19 & 12 \\ 10 & 10 & 13 & 20 & 16 & 12 & 19 & 20 \\ 13 & 15 & 22 & 16 & 11 & 21 & 23 & 20 \\ 15 & 8 & 9 & 14 & 20 & 20 & 10 & 2 \end{matrix}$$

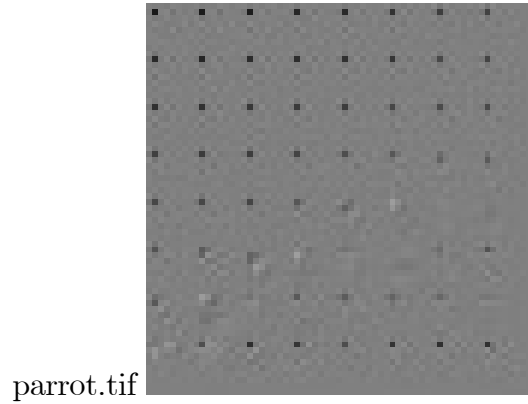
The DCTs of the  $8 \times 8$  blocks are evaluated (data rounded to one decimal place):

$$\text{DCT } F[k, \ell] \cong \begin{matrix} 24.4 & -49.6 & -2.4 & 2.1 & -4.6 & -0.4 & -0.2 & 0.8 \\ -92.0 & -18.0 & 18.8 & -5.6 & 1.2 & -3.9 & -1.9 & -0.2 \\ -23.7 & 34.8 & 13.7 & -2.5 & 4.2 & 3.3 & 1.1 & -1.0 \\ 8.5 & 18.8 & -5.9 & -16.1 & 3.0 & 0.3 & -4.9 & 1.8 \\ -8.4 & 0.4 & -24.8 & 9.2 & 1.6 & -4.7 & 1.2 & 3.1 \\ 1.8 & -5.6 & 0.5 & 4.5 & -2.6 & 2.8 & 7.3 & -0.6 \\ -4.7 & 3.0 & -0.2 & 9.2 & 3.8 & -0.1 & -0.2 & -2.0 \\ 2.6 & -0.5 & -2.3 & 0.8 & -9.8 & -3.8 & -1.0 & -1.2 \end{matrix}$$



If the block were pure white (gray value 255), then the DCT would produce a single nonzero amplitude of +1016 at the DC term and zero elsewhere; if the block were black, the DC coefficient of the DCT would be -1024

The DCT of the 64  $8 \times 8$  blocks can be displayed in pictorial form:



DCT values are divided by the values in a *normalization matrix* that accounts for the contrast sensitivity function of the eye. Smaller numbers imply more weighting applied. Largest numbers apply to the high-frequency terms positioned toward the lower right corner. Note that the DC component (less 128) is divided by a larger number than the neighboring low-frequency AC components.

$$Q[u, v] =$$

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	50	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

$$DCT \frac{F[k, \ell]}{N[k, \ell]} =$$

1.5	-4.5	-0.2	0.1	-0.2	0.0	0.0	0.0
-7.7	-1.5	1.3	-0.3	0.0	-0.1	0.0	0.0
-1.7	2.7	0.9	-0.1	0.1	0.1	0.0	0.0
0.6	1.1	-0.3	-0.6	0.1	0.0	-0.1	0.0
-0.5	0.0	-0.7	0.2	0.0	0.0	0.0	0.0
0.1	-0.2	0.0	0.1	0.0	0.0	0.1	0.0
-0.1	0.0	0.0	0.1	0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0	-0.1	0.0	0.0	0.0

Normalized  $8 \times 8$  block displayed as an image.



The DC coefficients of blocks are encoded separately using a differential coding scheme. The AC coefficients are encoded based the entropy (Huffman code) with three pieces of information:

1. the run length of consecutive zeros that preceded the current element in the zigzag sequence.
2. the number of bits to follow in the amplitude number
3. the amplitude of the coefficient

Bit Count	Amplitudes
1	-1,+1
2	-3,-2,+2,+3
3	-7 to -4, +4 to +7
4	-15 to -8, +8 to +15
5	-31 to -16, +16 to +31
6	-63 to -32, +32 to +63
7	-127 to -64, +64 to +127
8	-255 to -128, +129 to +255
9	-511 to -256, +256 to +511
10	-1023 to -512, +512 to +1023

2 (DC term coded separately) -5 -8 -2 1 3 1 0 0 0 1 1  
 0 0 0 0 0 0 0 0 0 0 -1 -1 0 × 39

This string is encoded using a predetermined Huffman code based on the number of zeros in the string to the next nonzero coefficient and the numerical value of the coefficient. Short strings of zeros are encoded with shorter sequences

(0, 3) (0, 4) (0, 2) (0, 1) (0, 2) (0, 1) (2, 1) (0, 1) (10, 1) (0, 1) EOB

The Huffman code has the partial form:

100+0010 1011+0+100 01+0+ ...

**Recover from JPEG coding:**

First we reconstruct the approximate DCT from the Huffman code and multiplication by the normalization matrix:

$$\text{DCT } \hat{F}[k, \ell] \cdot N[k, \ell] = \begin{array}{cccccccc} 32 & -55 & 0 & 0 & 0 & 0 & 0 & 0 \\ -96 & -24 & 14 & 0 & 0 & 0 & 0 & 0 \\ -28 & 39 & 16 & 0 & 0 & 0 & 0 & 0 \\ 14 & 17 & 0 & -29 & 0 & 0 & 0 & 0 \\ 0 & 0 & -37 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{array}$$

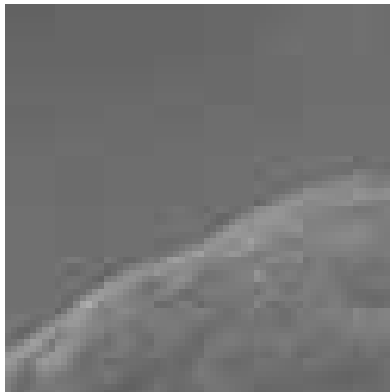
The inverse DCT produceds an  $8 \times 8$  block of bitonal “gray values:”

$$\begin{array}{cccccccc} -22.6 & -16.3 & -11.2 & -13.0 & -18.5 & -19.6 & -13.6 & -6.6 \\ -12.7 & -18.4 & -24.6 & -25.7 & -19.5 & -8.7 & 1.8 & 8.1 \\ -14.2 & -22.1 & -27.2 & -20.4 & -4.3 & 10.0 & 15.5 & 15.1 \\ -23.4 & -19.1 & -8.8 & -6.1 & 19.3 & 23.9 & 19.5 & 13.5 \\ -15.1 & -2.8 & 12.9 & 22.6 & 24.1 & 21.9 & 20.9 & 21.4 \\ 8.8 & 15.4 & 20.2 & 17.7 & 12.1 & 12.9 & 22.5 & 32.3 \\ 20.1 & 18.8 & 16.0 & 12.7 & 11.2 & 13.4 & 18.5 & 22.8 \\ 14.8 & 11.8 & 11.4 & 16.8 & 22.8 & 21.4 & 11.4 & 1.6 \end{array}$$

These values are rounded and the constant 128 is added back to obtain the recovered

block, which is compared to the original values:

$$\hat{f}[n, m] = \begin{array}{cccccccc} 105 & 112 & 117 & 115 & 110 & 108 & 114 & 121 \\ 115 & 110 & 103 & 102 & 108 & 119 & 130 & 136 \\ 114 & 106 & 101 & 108 & 124 & 138 & 143 & 143 \\ 105 & 109 & 119 & 122 & 147 & 152 & 148 & 142 \\ 113 & 125 & 141 & 151 & 152 & 150 & 149 & 149 \\ 137 & 143 & 148 & 146 & 140 & 141 & 150 & 160 \\ 148 & 147 & 144 & 141 & 139 & 141 & 147 & 151 \\ 142 & 140 & 139 & 145 & 151 & 149 & 139 & 130 \\ \\ 110 & 111 & 111 & 110 & 111 & 109 & 112 & 117 \\ 112 & 115 & 111 & 110 & 108 & 111 & 129 & 139 \\ 111 & 112 & 107 & 109 & 120 & 140 & 142 & 142 \\ 108 & 107 & 119 & 126 & 142 & 150 & 145 & 143 \\ 111 & 131 & 137 & 141 & 152 & 148 & 147 & 140 \\ 138 & 138 & 141 & 148 & 144 & 140 & 147 & 148 \\ 141 & 143 & 150 & 144 & 139 & 149 & 151 & 148 \\ 143 & 136 & 137 & 142 & 148 & 148 & 138 & 130 \end{array}$$



The error in the block is obtained by subtracting the recovered image from the original:

$$\varepsilon[n, m] = f[n, m] - \hat{f}[n, m] = \begin{bmatrix} +5 & -1 & -6 & -5 & +1 & +1 & -2 & -4 \\ -3 & +5 & +8 & +8 & 0 & -8 & -1 & +3 \\ -3 & +6 & +6 & +1 & -4 & +2 & -1 & -1 \\ +3 & -2 & 0 & +4 & -5 & -2 & -3 & +1 \\ -2 & +6 & -4 & -10 & 0 & -2 & -2 & -9 \\ +1 & -5 & -7 & +2 & +4 & -1 & -3 & -12 \\ -7 & -4 & +6 & +3 & 0 & +8 & +4 & -3 \\ +1 & -4 & -2 & -3 & -3 & -1 & -1 & 0 \end{bmatrix}$$

### 20.6.2 Example 2: “Busy” Image

Consider the central  $8 \times 8$  block of the “Liberty” image, with gray values:

$$f[n, m] = \begin{bmatrix} 160 & 184 & 96 & 160 & 192 & 136 & 168 & 152 \\ 176 & 152 & 160 & 144 & 160 & 88 & 80 & 128 \\ 56 & 64 & 120 & 168 & 120 & 72 & 64 & 200 \\ 112 & 56 & 40 & 56 & 136 & 120 & 48 & 192 \\ 168 & 128 & 40 & 56 & 136 & 120 & 48 & 192 \\ 176 & 40 & 112 & 24 & 88 & 120 & 64 & 160 \\ 152 & 48 & 168 & 48 & 16 & 120 & 8 & 8 \\ 48 & 176 & 160 & 128 & 16 & 32 & 96 & 104 \end{bmatrix}$$

Subtract the constant:

$$f[n, m] - 128 = \begin{bmatrix} 32 & 56 & -32 & 32 & 64 & 8 & 40 & 24 \\ 48 & 24 & 32 & 16 & 32 & -40 & -48 & 0 \\ -72 & -64 & -8 & 40 & -8 & -56 & -64 & 72 \\ -16 & -72 & -88 & -72 & 8 & -8 & -80 & 64 \\ 40 & 0 & -88 & -72 & 8 & -8 & -80 & 64 \\ 48 & -88 & -16 & -104 & -40 & -8 & -64 & 32 \\ 24 & -80 & 40 & -80 & -112 & -8 & -120 & -120 \\ -80 & 48 & 32 & 0 & -112 & -96 & -32 & -24 \end{bmatrix}$$

Evaluate the  $8 \times 8$  DCT

$$F[k, \ell] = \begin{bmatrix} -164.0 & 40.5 & 79.7 & -56.8 & 86.0 & -31.2 & 77.4 & -32.3 \\ 161.7 & -55.2 & -38.6 & 14.4 & 90.5 & -28.3 & -83.6 & -3.2 \\ 72.4 & 89.6 & -70.4 & -27.5 & -107.1 & 27.4 & -71.4 & 35.3 \\ 60.4 & 8.6 & 48.5 & 156.9 & 8.5 & -35.0 & 35.9 & 54.3 \\ 34 & -39 & 45.1 & -14.1 & 0 & -24.3 & -105.3 & -66.5 \\ -70 & -16.2 & -16.3 & 12.4 & -14.8 & 68.9 & 46.6 & -34.4 \\ 28.5 & -92.1 & 0.6 & -50.2 & 18.4 & -12.1 & -33.6 & 8.7 \\ 13.6 & 0.3 & -40.6 & 19.2 & 0.7 & -43.7 & -7.2 & 9.4 \end{bmatrix}$$

Normalize by the weighting matrix

$$\frac{F[k, \ell]}{N[k, \ell]} = \begin{bmatrix} -10.3 & 3.7 & 8.0 & -3.6 & 3.6 & -0.8 & 1.5 & -0.5 \\ 13.5 & -4.6 & -2.8 & 0.8 & 3.5 & -0.5 & -1.4 & -0.1 \\ 5.2 & 6.9 & -4.4 & -1.1 & -2.7 & 0.5 & -1.0 & 0.6 \\ 4.3 & 0.5 & 2.2 & 5.4 & 0.2 & -0.4 & 0.4 & 0.9 \\ 1.9 & -1.8 & 1.2 & -0.3 & 0.0 & -0.2 & -1 & -0.9 \\ -2.9 & -0.5 & -0.3 & 0.2 & -0.2 & 0.7 & 0.4 & -0.4 \\ 0.6 & -1.4 & 0.0 & -0.6 & 0.2 & -0.1 & -0.3 & 0.1 \\ 0.2 & 0.0 & -0.4 & 0.2 & 0.0 & -0.4 & -0.1 & 0.1 \end{bmatrix}$$

Round to nearest integer

$$\text{integer} \left\{ \frac{F[k, \ell]}{N[k, \ell]} \right\} = \begin{bmatrix} -10 & 4 & 8 & -4 & 4 & -1 & 2 & -1 \\ 13 & -5 & -3 & 1 & 3 & 0 & -1 & 0 \\ 5 & 7 & -4 & -1 & -3 & 0 & -1 & 0 \\ 4 & 1 & 2 & 5 & 0 & 0 & 0 & 1 \\ 2 & -2 & 1 & 0 & 0 & 0 & -1 & -1 \\ -3 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & -1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Renormalize by multiplying by  $N[k, \ell]$ :

$$\hat{F}[k, \ell] = N[k, \ell] \cdot \text{integer} \left\{ \frac{F[k, \ell]}{N[k, \ell]} \right\} = \begin{bmatrix} -160 & 44 & 80 & -64 & 96 & -40 & 102 & -61 \\ 156 & -60 & -42 & 19 & 78 & 0 & -60 & 0 \\ 70 & 91 & -64 & -24 & -120 & 0 & -69 & 56 \\ 56 & 17 & 44 & 145 & 0 & 0 & 0 & 62 \\ 36 & -44 & 37 & 0 & 0 & 0 & -103 & -77 \\ -72 & 0 & 0 & 0 & 0 & 104 & 0 & 0 \\ 48 & -64 & 0 & -87 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

Calculate the inverse  $8 \times 8$  DCT

$$\hat{f}[n, m] - 128 = \begin{bmatrix} 42 & 45 & -14 & 41 & 41 & -4 & 59 & 10 \\ 30 & 21 & 19 & -6 & 52 & 1 & -79 & 8 \\ -56 & -72 & 21 & 45 & -47 & -47 & -86 & 96 \\ -11 & -79 & -104 & 26 & -99 & -43 & -74 & 58 \\ 44 & -17 & -104 & -48 & -28 & 8 & -81 & 70 \\ 39 & -65 & 6 & -83 & -36 & -8 & -81 & 32 \\ 31 & -70 & 13 & -87 & -105 & -52 & -105 & -113 \\ -85 & 64 & 37 & -12 & -100 & -61 & -72 & -3 \end{bmatrix}$$

Add back the constant:

$$\hat{f}[n, m] = \begin{bmatrix} 170 & 173 & 114 & 169 & 169 & 124 & 187 & 138 \\ 158 & 149 & 147 & 122 & 180 & 129 & 49 & 136 \\ 72 & 56 & 149 & 173 & 81 & 81 & 42 & 224 \\ 117 & 49 & 24 & 154 & 29 & 85 & 54 & 186 \\ 172 & 111 & 24 & 80 & 100 & 136 & 47 & 198 \\ 167 & 63 & 134 & 45 & 92 & 120 & 47 & 160 \\ 159 & 58 & 141 & 41 & 23 & 76 & 23 & 15 \\ 43 & 192 & 165 & 116 & 28 & 67 & 56 & 125 \end{bmatrix}$$

The error is the difference.

$$\varepsilon[n, m] = f[n, m] - \hat{f}[n, m] = \begin{bmatrix} -10 & 11 & -18 & -9 & 23 & 12 & -19 & 14 \\ 18 & 3 & 13 & 22 & -20 & -41 & 31 & -8 \\ -16 & 8 & -29 & -5 & 39 & -9 & 22 & -24 \\ -5 & 7 & 16 & -98 & 107 & 35 & -6 & 6 \\ -4 & 17 & 16 & -24 & 36 & -16 & 1 & -6 \\ 9 & -23 & -22 & -21 & -4 & 0 & 17 & 0 \\ -7 & -10 & 27 & 7 & -7 & 44 & -15 & -7 \\ 5 & -16 & -5 & 12 & -12 & -35 & 40 & -21 \end{bmatrix}$$

Note that the error is much larger in several of the locations, because the high-frequency coefficients that are necessary to produce the “sharp edges” have been quantized to zero.