1 *The Science of Digital Media*, Chapter 1: Digital Data Representation and Communication[1]

I am an instrument in the shape
of a woman trying to translate pulsations
into images....
Adrienne Rich, "Planetarium"

Objectives for Chapter 1

- Understand the difference between analog and discrete phenomena.
- Understand how images and sound can be represented as sinusoidal waveforms.
- Understand how sinusoidal waveforms model sound waves of a given frequency.
- Understand how sinusoidal functions can be summed to create more complex waveforms.
- Understand how undersampling leads to aliasing.
- Understand how quantization leads to quantization error.
- Understand and be able to apply the equation for signal-to-noise ratio.
- Be able to calculate the storage space needed for digital images, audio, and video files given basic parameters.
- Know the storage capacity of common storage media.
- Understand basic concepts of digital data communication.
- Understand the various usages of the term *bandwidth*.
- Know how to compute maximum data rate given bandwidth and number of different signal values that can be communicated.
- Understand the relationship between bit rate and baud rate.
- Be able to perform run-length encoding, entropy encoding, and arithmetic encoding.
- Be familiar with common compression algorithms and codecs.

Mathematics Needed for Chapter 1

- functions and their graphs
- sinusoidal waveforms
- discrete and continuous number systems
- logarithms

[1] This material is based on work support by the National Science Foundation. The first draft of this chapter was written under Grant No. DUE-0127280. The final version was written under Grant No. 0340969. This chapter was written by Dr. Jennifer Burg (burg@wfu.edu).

- bases, particularly binary and decimal
- exponents
- summations
- Cartesian coordinates
- vectors and matrices

Programming and Data Structures Background Needed for Chapter 1

- basic programming skills
- ability to read pseudocode
- 1D and 2D arrays
- trees

1.1 Introduction

1.1.1 The Approach of this Book

Something caused you to open this book. Maybe this book is the required text for a course you're taking, a course such as Digital Media or Multimedia Systems. Maybe you've been dabbling in photographic, sound, or video processing on your own and have gotten curious about how your tools actually work. Maybe you're a mathematician, engineer, or physicist who finds that digital media is a fascinating arena in which to apply your discipline. Maybe you're a musician, a graphic artist, or a professional in some field that requires the support of digital media, and you want to know more. If science and mathematics have always interested you, you may be curious to look behind the scenes, expecting that a deeper understanding of your digital media tools will enhance your creative powers.

Whatever your reason for opening this book, we assume that the greatest motivation for most people interested in digital media is the excitement of creation. Digital media is multimedia driven by computers. You can see it, hear it, maybe even touch it, and certainly interact with it. It becomes even more exciting when the images, sound, and motion are the work of your own hands and brain. What draws most people to a study of digital media is the satisfaction of making something that communicates, entertains, or teaches.

With this goal in mind, we've chosen the topics in this book by considering the specific activities undertaken in digital media work – choosing color modes, compressing files, identifying aliased frequencies, filtering, transforming, and creatively editing. However, rather than present step-by-step methods related to specific application programs, we present the mathematical and algorithmic procedures upon which the tools are built. We attempt to make the explanations simple, straightforward, and relevant to hands-on activity. Procedures and algorithms are illustrated with interactive demos and mathematical modeling exercises that reinforce the text. We believe that the science behind digital media can be just as fascinating as the end product, and that understanding the science not only will increase your creative powers but also will make the creative process more intellectually satisfying. Dig in and see what you discover.

1.1.2 Where to Begin

So where do we begin? This book is organized around three main media – digital imaging, audio, and video – woven together at the end with multimedia programming. Topics in each chapter are motivated by activities done in digital media application programs such as Photoshop, Premiere, Final Cut, Audition, Sound Forge, Logic, Reason, Illustrator, Flash, and Director, but they are presented from a mathematical or conceptual perspective.

Chapter 1 covers concepts that are relevant to more than one medium. In particular, the digitization process of sampling and quantization is introduced, to be revisited in the context of image and sound in later chapters. Analog-to-digital conversion is where it all begins. We then move on to fundamentals of data communication and data storage. A survey of compression methods is given, covering concepts relevant to all media. We conclude the chapter with an overview of standards and standardization organizations. Accompanying the chapter are interactive tutorials introducing you to mathematical modeling tools that will help you to visualize and experiment with concepts.

There's a lot of material in this chapter. You'll probably find some of it easy and you'll be able to skim through it quickly. Some you can note as reference material to look back at later when you need the information. Some of the material is more challenging and may not sink in entirely until you read on and work more with your tools. A lot of the material is reinforced in later chapters, as it applies to a particular medium – images, sound, and video. Focus on what interests you most, and don't forget to look at the interactive tutorials and learning supplements on-line.

1.2 Analog to Digital Conversion

1.2.1 Analog versus Discrete Phenomena

Most of you already know the difference between analog and discrete phenomena, but for completeness, let's set the stage.

Analog phenomena are continuous – like a steady stream of water, a line on a graph, or a continuously rotating dial on a radio. With analog phenomena, there is no clear separation between one point and the next; in fact, between any two points, an infinite number of other points exist. ***Discrete phenomena***, on the other hand, *are* clearly separated. There's a point (in space or time), and then there's a neighboring point, and there's nothing between the two. A dripping faucet is a discrete phenomenon, whereas the water in a running tap is continuous and thus analog. Many of our sensory perceptions come to us in analog form (although some may consider this debatable, perhaps even a philosophical question). A live orchestra presents music to us in analog form, as continuous sound waves. A standard microphone is an analog device, detecting and recording sounds over time and transmitting them as a continuous wave of varying voltages.

Converting the continuous phenomena of images, sound, and motion into a discrete representation that can be handled by a computer is called ***analog-to-digital conversion***. You may wonder what the advantages are of digital as opposed to analog data. Photographs, music recordings, and television have been around for a long time without the benefit of being digitized, so why should we change now? Why did the consumer market switch so willingly from vinyl record albums to compact disks? Why should you buy a digital camera if you have a perfectly good analog one? What could motivate you to switch to digital television?

There are, in fact, quite a few advantages to digital media. It may seem that analog data – with infinite values that can run smoothly and continuously from one to the next – would be more precise and give better quality in photographs, music, and videos. But as storage media have increased in size and communication channels have increased in data rate, it has become possible to increase the resolution of digital images, audio, and video. This means that a digitized picture and sound can now be captured in fine detail. Digital data also has an advantage in the way that images and sound are communicated. Analog data communication is more vulnerable to noise than is digital, so it loses some of its quality in transmission. With digital data, which is communicated entirely as a sequence of 0s and 1s, error correcting strategies can be employed that ensure that the data is received and interpreted correctly. Moreover, digital data can be communicated more compactly than analog. This may seem counter-intuitive. You might think that a digitally-encoded television program, for example – in the primitive language of 0s and 1s – would require much more data than the same program broadcast in analog form. However, excellent compression algorithms exist for digital data that significantly reduce the amount of data without sacrificing quality. The result is that digital cable television transmissions are actually able to divide their bandwidth among numerous broadcasts, offering the consumer both a variety of programming and high quality picture and sound. How this all happens should become clearer to you as you learn more about analog-to-digital conversion, digital data communication, and compression algorithms.

1.2.2 Image and Sound Data Represented as Functions and Waveforms

The two primary media in digital media are images and sound. (They are primary in the sense that the two are combined to produce video. Haptic output – based on the sense of touch – is sometimes present in digital media but is not yet common.) Both images and sound can be represented as functions, and we can visualize these functions by means of their corresponding graphs. (If you don't need this refresher on the basic physics and mathematics of sinusoidal functions and waves, you can skip this section.)

Sound, for example, is a one-dimensional function – i.e., a function with one variable as input. If we think of sound as a continuous phenomenon, then we can model it as a continuous function $y = f(x)$ where x is time and y is the air pressure amplitude.

The essential form of the function representing sound is sinusoidal, which means that it has the shape of a sine wave In this book, we assume that you have some background in trigonometry, but let's review the sine function briefly. Assume you have a triangle with vertices A, B, and C. We give the name $\angle V$ to the angle at vertex V, for any vertex V. An angle can be measured in degrees, abbreviated as °, or radians, abbreviated as rad. The relationship between radians and degrees is as follows:

key equation

Let r be the size of an angle measured in radians, and let d be this size measured in degrees. Then

$$\frac{r}{d} = \frac{\pi}{180}$$

Equation 1.1

In this book, we will assume all angles are in radians.

Assume $\angle B$ is a *right angle*, which means that it is of size $\pi/2$, as pictured in Figure 1.1. A triangle that contains an angle of size $\pi/2$ is called a *right triangle*. The length of the side opposite the right angle is called the *hypotenuse*, which we abbreviate h. Consider either of the two non-right-angles in the triangle – say, for our example, $\angle C$, which has size α . The side opposite $\angle C$ is called the *opposite side* with regard to $\angle C$. Say that this side has length c. The other side is the *adjacent side* with regard to $\angle C$. Assume that the side adjacent has size a. Then the sine and cosine of $\angle C$ is defined by the angle's size as follows:

$$\sin(\alpha) = c/h$$
$$\cos(\alpha) = a/h$$

We call sines and cosines ***sinusoidal functions***.

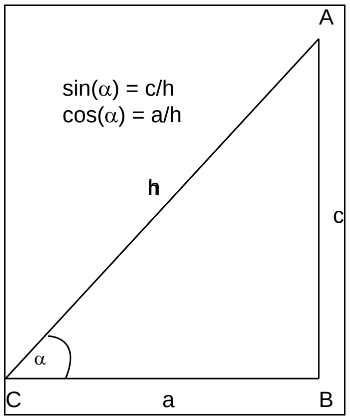


Figure 1.1 Sine and cosine functions defined within a right triangle

The previous definition defines the sine and cosine functions for angles that are less than $\pi/2$. We can generalize this definition for angles of any size by looking at angles at the center of a Cartesian coordinate system. In a Cartesian coordinate system, values increase horizontally from left to right across the x-axis and vertically from bottom to top up the y-axis. (x,y) denotes a point in this Cartesian space. x and y are called the *coordinates* of the point. The place where the two axes cross is called the *origin*, at point (0,0). Consider a circle of radius 1 around the origin, called a *unit circle*. Note that by the Pythagorean theorem, the equation for a unit circle is $x^2 + y^2 = 1$. Let point ***P*** be at position (1,0), which is located on the x-axis. For every point ***Q*** on the unit circle, you have an angle connecting ***P*** to the origin and then to ***Q***. Let the size of the angle be α. As you move ***Q*** around the unit circle counter-clockwise, α goes from 0 to 2π . Notice that when ***Q*** returns to its initial position on the x-axis, the angle can be considered to be of size 0 or 2π. In fact, it is angle $2\pi k$, k being the number of times you have gone around the unit circle. (k is positive if you move counterclockwise and negative otherwise.) Because ***Q*** lies on the unit circle, ***Q***'s *x* and *y* coordinates give the sine and cosine, respectively, of the angle created by ***P***, the origin, and ***Q***. That is $\sin(\alpha) = y$ and $\cos(\alpha) = x$. For angles of size θ where θ is greater than 2π or less than -2π and $\theta = \alpha + 2\pi k$, the definitions of the sine and cosine functions generalize as follows:

$$\text{If } \theta = \alpha + 2\pi k \text{ and } k \text{ is an integer then}$$
$$\sin(\theta) = \sin(\alpha + 2\pi k) = \sin(\alpha)$$
$$\cos(\theta) = \cos(\alpha + 2\pi k) = \cos(\alpha)$$

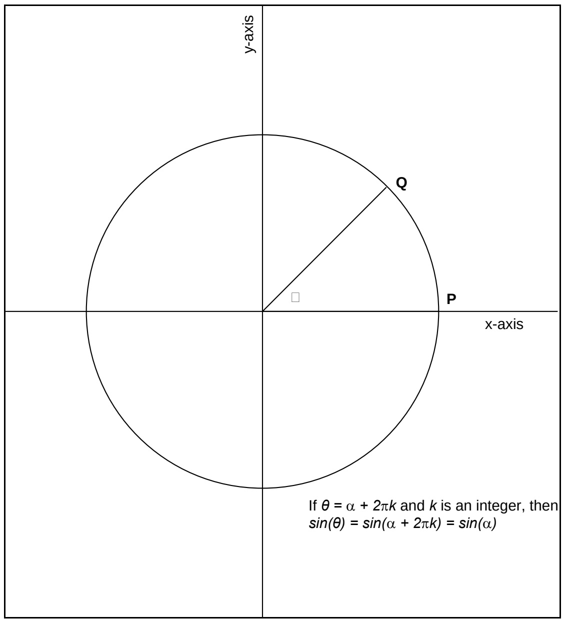

y-axis
Q
P
x-axis
If θ = α + 2πk and k is an integer, then
sin(θ) = sin(α + 2πk) = sin(α)

Figure 1.2 Sine and cosine functions defined within a unit circle

Thus, the sine and cosine functions are considered periodic functions in the sense that their values "cycle" in a regular pattern, as shown in Table 1.1.

angle in radians	angle in degrees	sine of angle
0	0	0
π/6	30	1/2
π/4	45	$\sqrt{2/2}$
π/3	60	$\sqrt{3/2}$
π/2	90	1
2π/3	120	$\sqrt{3/2}$
3π/4	135	$\sqrt{2/2}$
5π/6	150	1/2
π	180	0
7π/6	210	−1/2
5π/4	225	$-\sqrt{2/2}$
4π/3	240	$-\sqrt{3/2}$

3π/2	270	-1
5π/3	300	$-\sqrt{3}/2$
7π/4	315	$-\sqrt{2}/2$
11π/6	330	−1/2
2π	360	0

Table 1.1 Sine function

We can visualize this pattern through a graph of the values. In this graph, the x-axis represents the size of the angle and the y-axis represents the sine of the angle. This shape is referred to as a "sine wave," as shown in Figure 1.3. (The graph of the cosine is also shown in the figure.) We refer to this as a ***simple sine wave*** when we wish to emphasize that it is a single sine function rather than the sum of two or more.

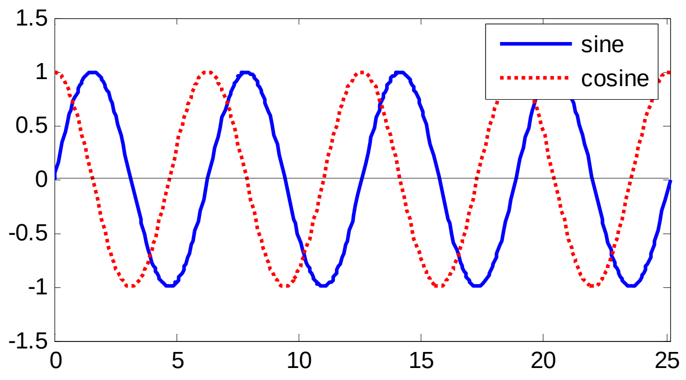


Figure 1.3 Graphs of sine and cosine waves

Now let's see how sinusoidal functions relate to waves and thus to sound and images. Sound is a good medium with which to begin this discussion because it is easy to understand as an analog phenomenon and to think of as a wave.

You're probably familiar with the concept of sound waves as a physical phenomenon, but this review may be useful since sinusoidal waves are an important abstraction in digital media, relevant to images as well. First, sound is a *mechanical wave*, which means that it results from the motion of particles through a transmission medium – for example, the motion of molecules in air. Because sound is a mechanical wave, it has to have something to move through; sound cannot be transmitted through a vacuum.

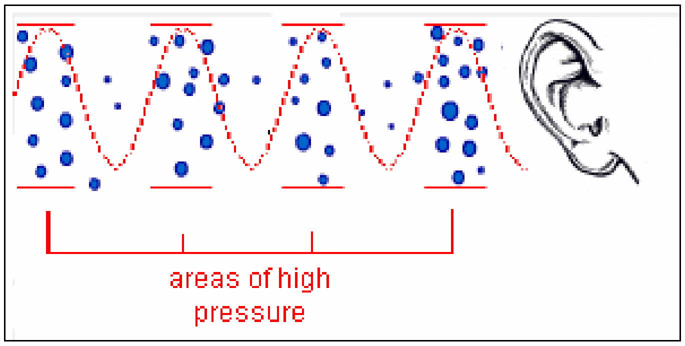


Figure 1.4 Changing air pressure caused by vibration of air molecules

The movement associated with a sound wave is initiated by a vibration. Imagine one of the strings inside a piano vibrating after one of the piano's soft hammers hits it. The air molecules next to the string are set in motion, radiating energy out from the vibrating string. For simplicity, let's just picture a single "wave" moving from left to right. As the string vibrates to the right, the molecules closest to the string are pushed to the right, moving closer to the molecules next to them, which in a chain reaction move closer to the molecules next to them, and so forth. When a group of molecules are pressed closer to their neighbors, air pressure rises. When the piano string moves back to the left, the molecules next to the string have space to spread out and move to the left as well, so the pressure between these molecules and the molecules to their right is reduced. This periodic changing of air pressure – high to low, high to low, etc. – radiates out from the string from left to right. (See Figure 1.4.)

If you can visualize a sound wave as we just described it above, you can see that the motion of the air molecules is back and forth from left to right, the same direction in which the wave is radiating out from the string. A wave of this type is called a ***longitudinal wave***, which is defined as a wave in which the motion of individual particles is in a direction parallel to the direction in which energy is being transported. Sound is a longitudinal mechanical wave.

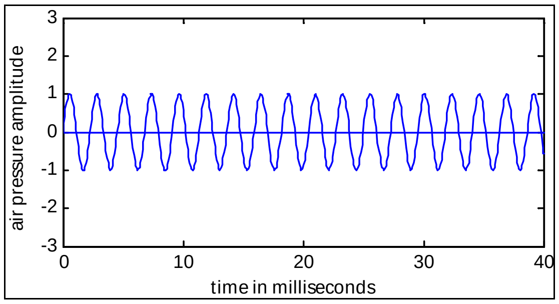


Figure 1.5 A single-frequency (440 Hz) tone with no overtones, represented as a waveform

The sound wave in Figure 1.5 is a graphical and mathematical abstraction of the physical phenomenon of sound. Time units are milliseconds. Prefix definitions are given in the key equations below. The graph represents represents the periodic change of air pressure. First the pressure increases as molecules are pushed closer together, shown by the upward arc in the graph. Then the pressure decreases as the molecules move apart, shown by the downward arc. These changes happen over time, so the x-axis in the graph represents time, while the y-axis represents air pressure.

key equations

Definitions of prefixes:
milli-   = 10^{-3}
micro- = 10^{-6}
nano-   = 10^{-9}

A wave is said to be *periodic* if it repeats a pattern over time. The pattern that is repeated constitutes one ***cycle*** of the wave. A ***wavelength*** is the length (in distance) of one complete cycle. The ***frequency*** of a wave is the number of times a cycle repeats per unit time, which in the case of sound corresponds to the rate at which the air molecules are vibrating. The frequency of a sound wave is measured in cycles per second, or Hertz. Frequency measurements are summarized in the key equations below.

key equations

Assume the following abbreviations:

Hertz	Hz
kilohertz	*kHz*
megahertz	*MHz*
second	*s*

Then

$$1\,Hz = 1\,cycle/s$$

$$1\,kHz = 1000\,Hz$$

$$1\,MHz = 1{,}000{,}000\,Hz$$

The ***period*** of a wave is the amount of time it takes for one cycle to complete. Period and frequency are reciprocals of each other.

key equation

Let T be the period and f be the frequency of a sinusoidal wave. Then

$$T = 1/f \text{ and}$$
$$f = 1/T$$

The height of a wave is called its ***amplitude***.

To create a sine function representing a sound wave of frequency f Hz, you must convert f to angular frequency first. The relationship between the two is as follows:

key equation

Let f be the frequency of a sine wave measured in Hz. Let ω the equivalent ***angular frequency*** measured in radians/s. Then

$$\omega = 2\pi f$$

Equation 1.2

Thus, the sine function representing a sound wave with a frequency of 440 Hz is $\sin(880\pi)$. This sine wave is pictured in Figure 1.5.

A graphical representation of sound in the form of a wave tells us something about the sound without our having to hear it. If the wave is completely regular like the one in Figure 1.5, then the sound is a pure tone, like a single musical note with no overtones. The amplitude of a wave corresponds to how loud the sound is; the larger the amplitude, the louder the sound. The frequency of a wave corresponds to the pitch of the sound; the higher the frequency, the higher-pitched the sound.

Aside: Amplitude and loudness are not exactly the same thing. Amplitude is a measure of air pressure while loudness is a subjective perception of sound. The two are related, however, in that higher amplitudes are generally perceived as louder sounds.

Figure 1.5 shows a simple

waveform corresponding to an idealized musical tone at a single frequency. A single-frequency tone takes the form of a sine function; i.e., it is a ***sinusoidal waveform***. Few sounds in nature are this pure in form. Then how are more complex sounds created and represented? It is possible to add a number of simple waveforms to get a more complex one. Figure 1.6 shows the result of adding the musical notes C, E, and G together. The sum of the three sound waves creates the harmony of the three notes played simultaneously.

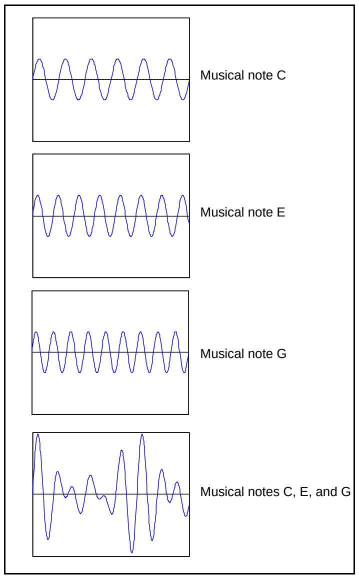


Figure 1.6 Adding sound waves

Aside: Jean Baptiste Joseph Fourier was an 18th century French mathematician whose analyses of periodic functions (i.e., sines and cosines) have been fundamental to advances in engineering and digital signal processing.

We can do this process in reverse. That is, we can take a complex waveform and break it down mathematically into its frequency components. The method for doing this is called the ***Fourier transform***. Fourier analysis shows that any periodic signal can be decomposed into an infinite sum of sinusoidal waveforms. The simple sinusoidal waves are called the ***frequency components*** of the more complex wave. The Fourier transform makes it possible to store a complex sound wave in digital form, determine the wave's frequency components, and filter out components that are not wanted. Filtering can be done in order to improve audio quality or to compress a digital audio file. These concepts and procedures will be discussed in more detail in Chapter 4.

In Chapter 2, you will see that sinusoidal waveforms can also be used to represent changing color amplitudes in a digital image. Look at the image in Figure 1.7. Imagine that each point in the picture can be represented by a number – its ***grayscale*** value. Grayscale values range from 0 to 255 and correspond to shades of gray, black, and white. (We could also think of 256 different colors. The point is that they are simply represented as numbers.) For the moment, assume that the image is continuous, with an infinite number of points in each row and column of the image. We'll worry about digitization later. Assume that the grayscale values vary smoothly from left to right, increasing and decreasing in a regular pattern. Now take just one horizontal line of grayscale values from that image. If the grayscale values are graphed over space, we get the sine wave pictured in Figure 1.8. The values in the vertical direction could be graphed similarly, as a one-dimensional waveform. By extension, we could graph the values taken from both the *x* and *y* directions on the image, representing the grayscale values as amplitudes – that is, representing them by their height above the image plane. In this sense, the values in the image, when graphed over the spatial domain, define a two-dimensional waveform, as pictured in Figure 1.9. (The shading of the waveform in this figure is only to help you see its contour better. It is the shape that is significant.) We'll return to the concept of images as waveforms in more detail in Chapter 2.

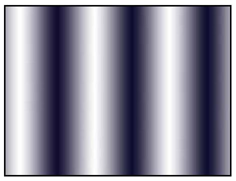


Figure 1.7 Grayscale image, smooth gradient

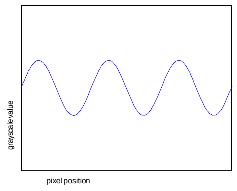


Figure 1.8 Sine wave corresponding to one line of grayscale values across the image to the left

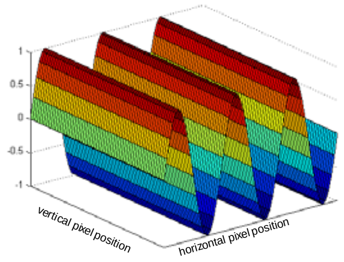


Figure 1.9 Two-dimensional waveform corresponding to the grayscale image above

Viewing a sound or an image as a waveform underlines the continuous nature of the data. The wave varies smoothly across time or space. For each two points we might choose, there are an infinite number of points between. How, then, do we capture this information in a computer? We have an infinite number of points of information to deal with, and that's too much data for a computer to handle. Thus, the first problem to be tackled in digital media is transforming analog information to a discrete form by means of analog-to-digital conversion**.**

Regardless of the medium, analog-to-digital conversion requires the same two steps: ***sampling*** and ***quantization***. The first step, sampling, chooses discrete points at which to measure a continuous phenomenon (which we will also call a *signal*). In the case of images, the sample points are evenly separated in space. In the case of sound, the sample points are evenly separated in time. The number of samples taken per unit time or unit space is called the ***sampling rate*** or, alternatively, the ***resolution***. The second step, quantization, requires that each sample be represented in a fixed number of bits, called the ***sample size***, or, equivalently, the ***bit depth***. The bit depth limits the precision with which each sample can be represented. We will give simple examples of sampling and quantization in digital imaging and sound in this chapter, and more detailed explanations in Chapters 2 and 4.

1.2.3 Sampling and Aliasing

What if you had a simple picture like the one in Figure 1.10 and you wanted to record it digitally by taking a sample of the color at evenly-spaced places in the horizontal and vertical directions? In Figure 1.11, the sample areas are designated by rectangles which we call *sample blocks*. At each sample block, a digital camera detects the color of the object being photographed, recording the information in a ***pixel***, short of picture element. To recreate the scene from the samples taken in each sample block, by extrapolation you could assume that the whole block was the color of the sample. If you took your samples in the center of the blocks shown in Figure 1.11, you could recreate your picture correctly. However, what if you sampled only every other block in a row, beginning on the left. Reconstructing from those samples, you'd mistakenly think your entire picture was black. This is a simple example of ***undersampling***. Your sampling rate did not "keep up" with the rate of change of the pattern in the image. ***Aliasing*** in a digital image arises from undersampling and results in an image that does not match the original source – it may be blurred or have a false pattern.

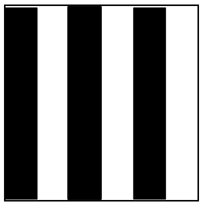


Figure 1.10 An image with a regular pattern

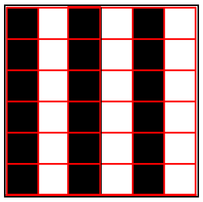


Figure 1.11 Squares denote areas where samples are taken, called *sample blocks.* (They are not part of the original image.)

A similar situation can occur when you sample an audio wave. A sound wave can be represented by a sine wave such as the one shown in Figure 1.12. This is a pure tone with a frequency of 637 Hz. Say that you sample this wave at a rate of 770 samples per second. That would be a rate of 770 Hz. (Note that with regard to sampling rate, Hz means *samples per second*, whereas with regard to a sound wave Hz means *cycles per second*.) Extrapolation could be used to reconstruct a sine wave from the samples, resulting in the lower-frequency wave shown in Figure 1.13. The circles represent the sample points. The lower-frequency wave connected through the circles is the aliased frequency that results from reconstructing the wave from too few samples. This is obviously not the original wave, and because the wave reconstructed from the samples has a different frequency from the original, it won't sound the same.

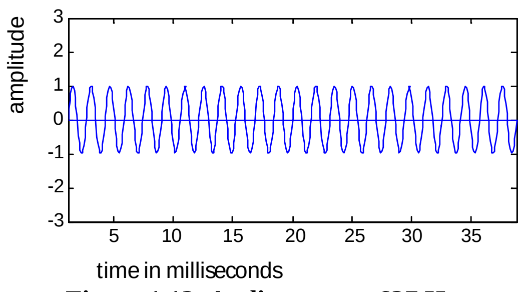


Figure 1.12 Audio wave at 637 Hz

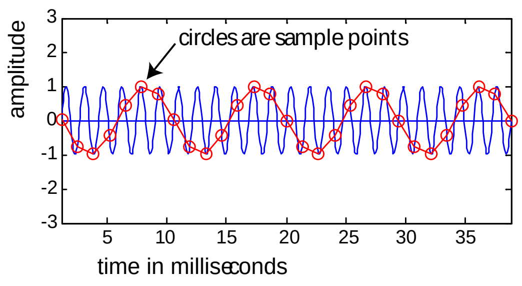


Figure 1.13 637 Hz audio wave sampled at 770 Hz

Both these examples illustrate how aliasing occurs as a result of undersampling. In general, aliasing is a situation where one thing takes the form or identity of another. Aliasing in digital images manifests itself as a lack of clarity, or a pattern in the digital image that did not exist in the original; in text, it shows up as jagged edges on letters that ought to be smooth; and in digital audio, it results in sound frequencies that did not exist in the original.

The precise cause and nature of aliasing is captured in the ***Nyquist theorem***, which specifies the sampling rate needed for a given spatial or temporal frequency. In simple terms, the Nyquist theorem states that to guarantee that no aliasing will occur, you must use a sampling rate that is greater than twice the frequency of the signal being sampled. The theorem is most easily understood as it relates to audio sampling, since we are already accustomed to thinking of sound in terms of frequency. In the example above, the sound wave being sampled has a frequency of 637 Hz. Thus, according to the Nyquist theorem, we need to sample at more than twice this rate; that is, the sampling rate must be greater than 1274 Hz. This makes sense if we consider that we want to be able to extrapolate the maximum and minimum amplitudes of the wave from our samples. If we sampled exactly twice per cycle and the samples were taken at the maximum and minimum points, this would suffice for the extrapolation. (But notice that if we sampled twice per cycle but always at the 0 points, we'd not be able to recreate the wave from the samples.) Thus, intuitively it makes sense that sampling more than twice per cycle should be enough. Anything less than twice per cycle results in audio aliasing – i.e., the digitized audio sounds different from the original.

Aside: Harry Nyquist was a physicist and electrical and communications engineer. He was born in Sweden and educated in the United States, where he worked at Bell Laboratories. He formulated his theorem regarding sampling rate in 1928. The theorem was later proven and extended by Claude Shannon in 1949. For this reason, the theorem is sometimes referred to as the Nyquist-Shannon theorem.

The Nyquist theorem applied to single-frequency, one-dimensional wave is summarized in the following equation:

key equation

Let *f* be the frequency of a sine wave. Let *r* be the minimum sampling rate that can be used in the digitization process such that the resulting digitized wave is not aliased. Then

$$r = 2f$$

r is called the ***Nyquist frequency***.

Equation 1.3

The Nyquist theorem applies equally to digital images, and is illustrated by the image shown in Figure 1.10. This image has a simple pattern with a regular frequency of repetition. If the sampling rate does not keep pace with this rate of repetition, the pattern cannot be reproduced faithfully, as explained above. Most images are much more complex than this simple pattern, but they have a spatial frequency in any case. The concept of spatial frequency and its relationship to the Nyquist theorem and aliasing will be explained in more detail in Chapter 2. The point to understand for now is that in both digital imaging and digital sound, undersampling creates aliasing and results in a digital reproduction of the image or sound that does not look or sound exactly like the original analog version.

1.2.4 Quantization, Quantization Error, and Signal-to-Noise Ratio

The second step in analog-to-digital conversion is quantization. We have seen that samples must be taken at discrete points in time or space, but we haven't yet considered how each sample is represented in a computer.

For digital images, each sample represents a color at a discrete point in a two-dimensional image. How many colors can we possibly represent in a digital image? The number of colors possible is determined by the number of bits used to represent each sample – i.e., the sample size or bit depth (which, in the case of an image file, can also be called the ***color depth***). If only one bit is used per sample, then only two colors are possible – because one bit can take on only two values – 0 or 1. If eight bits are used, then 2^8= 256 colors are possible. (Eight bits are also used for grayscale.) If 24 bits are used, then 2^{24} = 16,777,216 colors are possible. In general:

key equation

Let n be the number of bits used to quantize a digital sample. Then the maximum number of different values that can be represented, m, is

$$m = 2^n$$

Equation 1.4

The "different values" in the definition above correspond to color levels or sound amplitudes. In an image processing program, you are offered a choice of color modes with implicit bit depths. Clearly, the larger the bit depth, the more subtle the color changes there can be in a digitized image, but the file size also increases as the bit depth increases.

For digital audio, each sample represents the amplitude of a sound wave at a discrete point in time. Common sample sizes are eight or 16 bits. Stereo CD-quality digital audio uses 16 bits per sample in each of the two stereo channels, for a total of 32 bits per sample.

Sample size affects how precisely the value of a sample can be represented. In digital sound, sample size affects how much you have to round off the amplitude of the wave when it is sampled at points in time. In digital images, this corresponds to how close the digital image's colors are to the original colors that it is supposed to represent.

In digital sound, sample size affects how much you have to round off the amplitude of the wave when it is sampled at points in time. This is shown in Figure 1.14. The samples, which are taken at evenly-spaced points in time, can take on the values only at the discrete quantization levels on the y-axis. A sample must be rounded to the closest discrete level. The difference between its actual value and its rounded value (represented by pairs of points parallel to the y-axis in the figure) is the quantization error.

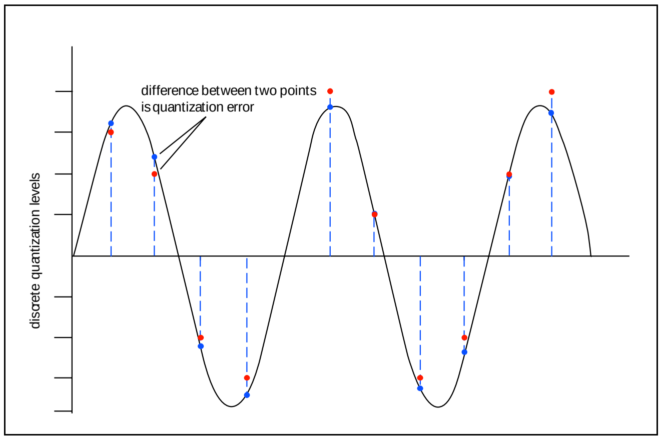

difference between two points is quantization error
discrete quantization levels

Figure 1.14 Quantization error

The amount of error implicit in a chosen bit depth can be measured in terms of the ***signal-to-noise ratio*** (***SNR***). Before looking at SNR specifically in the context of digital imaging and sound, let's consider the general definition.

Signal-to-noise ratio can generally be defined as the ratio of the meaningful content of a signal versus the associated noise. A signal is any type of communication – something a person says to you, a digital signal sending an image file across a network, a message posted on an electronic bulletin board, a piece of music being played from a cassette, etc. The noise is the part of the message that is not meaningful; in fact, it gets in the way of the "pure" message intended in the communication. You could use the term *signal-to-noise ratio* to describe communications in your everyday life. If you know someone who talks a lot but doesn't really convey a lot of meaning, you could say that he or she has a low signal-to-noise ratio. Web-based bulletin board and chat groups are sometimes described as having a low SNR – there may be quite a few postings, but without very much meaningful content. In these first two examples, the noise consists of all the empty "filler" words. In the case of a digital signal sent across a network, the noise is the electronic degradation of the signal. On a piece of music played from cassette, the noise could be caused by damage to the tape or mechanical imperfections in the cassette player.

More precise definitions of *signal-to-noise ratio* vary according to context. This might be confusing if you see the term used differently in different places, so let's differentiate the two main usages clearly.

- In analog data communication, the *signal-to-noise ratio* is defined as the ratio of the average power in the signal versus the power in the noise level. In this context, think of a signal being sent over a network connection compared to the extent to which the signal is corrupted. This is related to the general usage of the term described above. This usage of the term SNR applies to particular signals. The SNR depends on real-time conditions.

- For a digitized image or sound, the *signal-to-noise ratio* is defined as the ratio of the maximum sample value versus the maximum quantization error. In this usage of the term, the ratio depends on the bit depth chosen for the signal, not upon real-time conditions. Any signal encoded with a given bit depth will have the same ratio. This can also be called ***signal-to-quantization-noise ratio*** (***SQNR***), but you should be aware that in many sources the term *signal-to-noise ratio* is used here as well. Let's look at this more closely. (Henceforth, we'll use the term SQNR to distinguish this measurement from SNR.)

SQNR is measured in terms of decibels, so we need to make a short sidetrip into a definition of a ***decibel***. A decibel is a dimensionless unit that is used to describe the relative power or intensity of two phenomena. The definition of a *decibel*, abbreviated ***dB***, is as follows:

$$1\ dB = 10\log_{10}\left(\frac{I}{I_0}\right)$$

where I and I_0 are the intensities (also called *power* across a surface area) of two signals, which can be measured in Watts. So, you might ask, intensity of *what*? Once again, that depends on what you're interested in. Decibels are often used to measure sound intensity, and this is the use of the word that you're probably the most familiar with. But decibels can also measure the intensity of a data signal on a communication network, the optical power output of lasers, or anything that can be measured in watts. Notice that decibels are a dimensionless unit. Since both I and I_0 are measured in watts, the units "fall away" in the division, and you are left with just the number representing the ratio.

Aside: A decibel is 1/10th of a *bel*, named after Alexander Graham Bell. For two signals with power I and I_0, a ***bel*** is defined as follows: If $\log_{10}\left(\frac{I}{I_0}\right) = 1$, then the ratio is 1 bel. This is to say that I has 10 times the power of I_0, since $\log_{10}(10) = 1$. A bel turned out to be too coarse a measurement for its purposes, so the decibel came into common usage.

You may often see another definition for decibels, that is:

$$1dB = 20\log\left(\frac{E}{E_0}\right)$$

where E and E_0 are amplitude, potential, or pressure, measured in volts. You can see that the two definitions for a decibel are equivalent if you consider the relationship between power I, potential E, and resistance R.

$$I = \frac{E^2}{R}$$

Assuming that R is constant for the two signals, then

$$10\log_{10}\left(\frac{I}{I_0}\right) = 10\log_{10}\left(\frac{\frac{E^2}{R}}{\frac{E_0{}^2}{R}}\right) = 10\log_{10}\left(\frac{E^2}{E_0{}^2}\right) =$$

$$10\log_{10}\left(\left(\frac{E}{E_0}\right)^2\right) = 20\log_{10}\left(\frac{E}{E_0}\right)$$

(Recall that $\log_{10}(x^2) = 2\log_{10}(x)$.)

Because SQNR is just a ratio, it is measured in decibels. Using the second definition of *decibels* from above (because we are measuring amplitudes), we can now explain SQNR as it applies to linearly-quantized samples. First, we define max(*quantization value*) as the magnitude of the maximum sample value. With n bits for quantization, the samples values range from -2^{n-1} to $2^{n-1}-1$. (Audio signals are represented as sine waves that go from positive to negative values.) Thus, the maximum sample value in magnitude is 2^{n-1}. On the scale of sample values, the maximum quantization error is half a quantization level, which is where we get the denominator of ½. Then

$$SQNR = 20\log_{10}\left(\frac{\max(quantization\ value)}{\max(quantization\ error)}\right) = 20\log_{10}\left(\frac{2^{n-1}}{1/2}\right)$$

The above explanation should give you an intuitive sense of SQNR, defined more simply as follows:

Let *n* be the bit depth of a digitized media file – e.g., digital audio. Then the signal-to-quantization noise ratio, ***SQNR*** is

$$SQNR = 20\log_{10}(2^n)$$

Equation 1.5

key equation

Signal-to-quantization-noise ratio is directly related to ***dynamic range***. In fact, we will see that mathematically, they are the same value. Let's look at the concept intuitively. Dynamic range, informally defined, is the ratio of the largest-amplitude sound (or colors, for digital images) and the smallest that can be represented with a given bit depth. (Doesn't that sound like the definition of SQNR above?) The idea is that the fewer bits you have to represent a range of colors or a range of sound samples, and the wider the range of colors or sound you want to represent, the less ability you have to represent subtle differences between values. Say that you use three bits to represent colors, so you can represent only $2^3 = 8$ colors. Because of the small bit depth, you have a limited dynamic range in the palette you can use in digitizing an image. Think about the implications of being restricted to a palette with such a small dynamic range. What if the original image has many colors and these colors are widely ranged? You may need some blues, greens, reds, yellows, oranges, purples, and white and black. Since you can use only eight colors, you can use only one shade of each of those mentioned. Thus, the difference between one color and another is relatively large. If you take a photograph that has in it thousands of colors ranging widely across the spectrum but have to reproduce these with only eight colors, you're going to have a large rounding error. Alternatively, if you want subtle differences between one color and the next, you can have only a narrow range of colors. These two alternatives are shown in Figure 1.15. (Notice that the rounding error, which is relative to the narrow range of colors in this second example, will be the same. That is, the dynamic range, as dictated by the bit depth, is the same.)

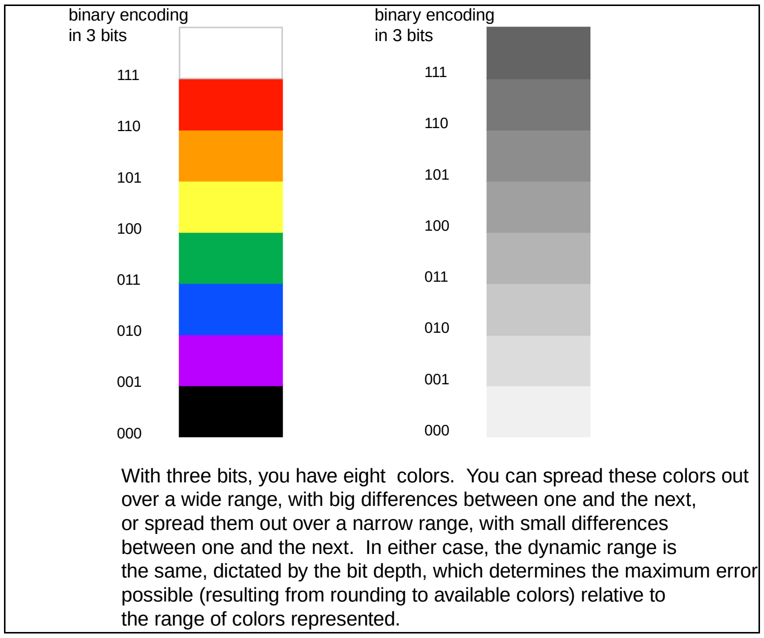

binary encoding
in 3 bits
111
110
101
100
011
010
001
000
binary encoding
in 3 bits
111
110
101
100
011
010
001
000
With three bits, you have eight colors. You can spread these colors out over a wide range, with big differences between one and the next, or spread them out over a narrow range, with small differences between one and the next. In either case, the dynamic range is the same, dictated by the bit depth, which determines the maximum error possible (resulting from rounding to available colors) relative to the range of colors represented.

Figure 1.15 Dynamic range of colors

The situation is similar in measuring dynamic range in digital audio. A small bit depth for an audio file limits the range of amplitudes in the digitized sound. A small bit depth is not good for a digital recording of a symphony orchestra playing a piece of music that has very loud and very soft passages.   In order to be able to represent sound amplitudes that are so far apart in value, but using a small bit depth, you are forced to have quantization intervals of a large size, and this increases the amount of error relative to the true signal. With a small bit depth, more values will round to 0 than with a higher bit depth. We will revisit the concept of SQNR and dynamic range related to digital audio in more detail in Chapter 4.

1.3 Data Storage

Working with digital media requires that you handle large amounts of data. Let's first consider the size of typical image, audio, and video files (leaving aside, for the moment, the possibility of compressing them). We'll return to the details of how each medium is represented in later chapters, but for now Table 1.2 gives you an overview of digital media file sizes.   The examples assume that RGB color is used, with three bytes per pixel. (RGB will be explained in more detail in Chapter 2.) The video example assumes a frame rate of about 30 frames/s. Each frame is like a still image.

Frame shown in quick succession create the effect of motion. (This will be explained in more detail in Chapter 6.)

The prefixes *kilo-*, *mega-*, *giga-* can lead to confusion because they are used inconsistently. You have seen that with regard to Hertz, *kilo-* means 10^3=1000, *mega-* means 10^6=1,000,000, and *giga-* means 10^9=1,000,000,000. However, with regard to data storage, *kilo-* is sometimes defined as 10^3 and sometimes as 2^{10}, *mega-* is sometimes 10^6 and sometimes 2^{20}, and *giga-* is sometimes 10^9 and sometimes 2^{30}. It depends on the source. Manufacturers want to make their storage media look larger, so they generally use powers of 10. On the other hand, many computers will give file sizes defining terms with powers of 2. For example, on your computer you may be able to click on a file to see its properties, which could be listed as 3,686,456 bytes in one place, 3,600 kB in another place, and 3.52 MB in another. Clearly, powers of two are being used for the definitions, since $3{,}686{,}456/1024 \approx 3600$ and $3{,}686{,}456/1{,}048{,}576 \approx 3.52$. We define *kilo-, mega-* and *giga-* with powers of two in the equations below and in the computation of file size in Table 1.2.

Table 1.3 lists the capacity of the storage media in use at the time of the writing of this chapter. For this table, you can assume that a kilobyte is 1000 bytes, a megabyte is 1,000,000 bytes, and a gigabyte is 1,000,000,000 bytes because this is the conversion used by the manufacturers. The bottom line, unfortunately, is that you can't always be sure of the definitions. We will generally use powers of two for storage size and powers of ten for data rate. Where values are approximate, the difference in definitions is often not important.

key equations

Assume the following abbreviations:

kilobyte	*kB*
megabyte	*MB*
gigabyte	*GB*
kilobit	*kb*
megabit	*Mb*
gigabit	*Gb*

Then

$$1\,byte = 8\,bits$$

$$1\,kB = 2^{10}\ bytes = 1024\,bytes$$

$$1\,MB = 2^{20}\ bytes = 1{,}048{,}576\,bytes$$

$$1\,GB = 2^{30}\ bytes = 1{,}073{,}741{,}824\,bytes$$

kb, Mb, and Gb are defined analogously.

(Note, however, that manufacturers are inconsistent in usage, sometimes using kilobyte to mean 1000 bytes, megabyte to be 1,000,000 bytes, and gigabyte to mean 1,000,000,000 bytes.)

Example digital image file size (without compression):	**Example digital audio file size** (without compression):	**Example digital video file size** (without compression):
Resolution: 1024 pixels × 768 pixels **Total number of**	**Sampling rate**: 44.1 kHz (44,100 samples per second) **Bit depth**: 32 bits per sample	**Frame size**: 720 pixels × 480 pixels **Bits per pixel**: 24 **Frame rate**: ~30 frames/s

pixels: 786,432 **Color mode**: RGB **Bits per pixel**: 24 (i.e., 3 bytes) **Total number of bits**: 18,874,368 (= 2,359,296 bytes) **File size**: 2.25 MB	(16 for each of two stereo channels) (i.e., 4 bytes) **Number of minutes**: one minute **Total number of bits**: 84,672,000 (= 10,584,000 bytes) **File size**: 10.09 MB for one minute **Data rate of the file**: 1.35 Mb/s	**Number of minutes**: one minute **Total image requirement**: 14,929,920,000 bits **Audio requirement**: 84,672,000 (See column 2.) **Total number of bits**: 15,014,592,000 (= 1,876,824,000 bytes) **File size**: > 1.7 GB **Data rate of the file**: 238.65 Mb/s (This calculation doesn't take chrominance subsampling into account. See Chapter 6 for a discussion of subsampling.)
Note: 1 kilobyte = 2^{10} bytes; 1 megabyte = 2^{20} bytes; 2 gigabyte = 2^{30} bytes.		

Table 1.2 Example file sizes for digital image, audio, and video

Storage Medium	**Maximum Capacity**
Portable Media	
CD (Compact Disk)	700 MB
DVD (Digital Versatile Disc or Digital Video Disk), standard one sided	4.7 GB standard; 8.5 GB dual-layered
DVD video or high capacity	17 - 27 GB
Memory stick or card	8 GB
HD-DVD (High Definition DVD), standard one sided	15 GB standard; 30 GB dual-layered
Blu-ray Disk	25 GB standard; 50 GB dual-layered
Flash drive	64 GB
Permanent Media	
Hard disk drive	1 terabyte (1000 GB)
Note: These values are approximate. Assume 1 GB = 1,000,000,000 bytes, as this is the assumption in the storage capacities reported by manufacturers.	

Table 1.3 Storage media and their capacity

A little arithmetic will show how quickly computer storage can be taken up with uncompressed digital media files, especially audio and video. The audio requires more than about 10 MB per minute, so about 70 minutes would fit on a CD. The situation is worse for digital video. Let's just take a rough estimate. One minute of digital video takes about 1.5 GB. So on a DVD that holds 17 GB, you'd be able to store just a little over 11 minutes of uncompressed video. Clearly, digital media files can be very large, and consequently, compression is crucial to reducing the file size. But before moving on to compression, let's consider an issue related to data storage, and that is data communication.

1.4 Data Communication

1.4.1 The Importance of Data Communication in the Study of Digital Media

Data communication is a broad and deep field of study that straddles the disciplines of computer science and electrical engineering. We can only scratch the surface in this introduction and give superficial treatment to issues that are technically quite complex. Before we begin tackling this topic, you may find it helpful to know the ways in which data communication is relevant to your study of digital media. Here are our motivations for looking at issues of data communication:

- Digital media files are typically very large, and rarely do you keep them to yourself. You store them on CDs and DVDs, send them in email, and post them on web pages. Thus, you need to consider transmission media and communication methods for digital data.
- Sound and video are time-based media that require large amounts of data. Both capturing and transmitting sound and video in real-time require that the data transmission keep up with the rate at which the data is played. Thus, issues of bandwidth and data rate are crucial in the capturing and transmitting of digital audio and video.
- Communication media in our homes and offices are "going digital." We are surrounded by cellular phones, digital cable, digital television, HDTV, and more. To be knowledgeable in the field of digital media, you need to understand the difference between analog and digital modes of communication as they appear in your home and work environments.

1.4.2 Analog Compared to Digital Data Communication

Whether data is in analog or digital form, information needs a channel of communication to get from sender to receiver. You use these communication channels everyday – land-based or cellular telephones; shortwave or regular radios; cable, terrestrial, or satellite television; and wired or wireless computer networks. You hear that some of these are digital, though you don't have to think about this to use them. So how do you know which communications are being sent digitally?

An important thing to keep in mind is that the transmission medium does not determine whether the data is communicated in analog or digital form. Both analog and digital data can be communicated across copper wire (e.g., telephone or computer networks), coaxial cable (e.g., television), optical fiber (e.g., high-speed computer networks), and free space (e.g., radio or television). Copper wire, coaxial cable, and optical fiber all require a physical line between sender and receiver. Across copper wire or coaxial cable, data can be transmitted by changing voltages. Through optical fiber, data can be communicated by a fluctuating beam of light. Data can also be communicated through free space via electromagnetic waves sent by satellite or radio transmission. All of these transmissions methods use some type of analog waveform, but it is important to note that they all can be adapted to carry data in both analog and digital form. It is the representation of the data, not the transmission medium, which determines if the communication is analog or digital.

Then what is the difference between how analog and digital data are transmitted across a network? Analog telephone transmissions through wire are a good starting point for understanding how analog data is communicated. When sound is captured electronically, the changes in air pressure are translated to changes in voltage. In an analog telephone, a microphone picks up the changes in air pressure and translates them into voltage changes on an electrical wire. For the spoken word "hi," the voltages rise and fall in the pattern like the one in Figure 1.16. The voltage changes are continuous in the same way that the waveform representing the word "hi" is continuous.

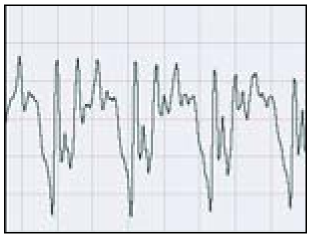


Figure 1.16 Waveform for part of the spoken word "hi"

If the word "hi" is digitized, on the other hand, it is sampled and quantized such that the data is transformed into a sequence of 0s and 1s. Over copper wire, these 0s and 1s are sent by means of an electrical current and can be represented as two discrete voltage levels. For example, a voltage of –V can represent a 0 bit, and a voltage of +V can represent a 1 bit. Communication begins with some initial synchronization between sender and receiver. Then to send each bit, the sending device maintains a steady voltage for a fixed amount of time. The receiving device, in turn, samples the transmission at evenly-spaced points in time to interpret whether a 0 or a 1 has been sent. Figure 1.17 illustrates this type of discrete transmission. Varying the voltage in two levels in the manner just described is called ***baseband transmission***, and the line of communication between sender and receiver is called a ***baseband channel***.

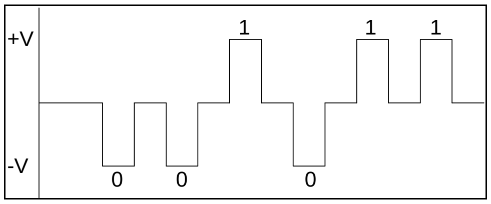


Figure 1.17 Baseband digital transmission. Bits are transmitted by discretely-changing voltages.

Baseband transmission is used across wire and coaxial cable, but it works well only across relatively short distances. Noise and attenuation cause the signal to degrade as it travels over the communication channel. Attentuation is the weakening of a signal over time and/or space.) Baseband transmission is not the only nor even the most common method of data communication. An alternative, called ***modulated data transmission*** (or sometimes ***bandpass transmission***), is based on the observation that a continuously oscillating signal degrades more slowly and thus is better for long distance communication. Modulated transmission makes use of a ***carrier signal*** on which data is "written."

Here again in digital media, sinusoidal waveforms become an important abstraction. A carrier wave is a signal – e.g., a beam of light passing through optical fiber or radio waves across free space – that is made to oscillate in a regular sinusoidal pattern at a certain frequency, as shown in Figure 1.18. This carrier wave is transmitted and oscillates at its characteristic frequency even when no data is being sent. Data is "written" on the carrier signal by means of ***modulation*** techniques.

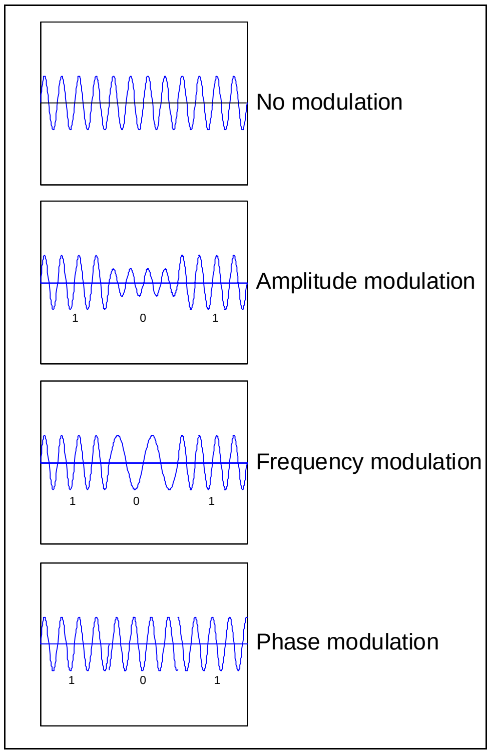


Figure 1.18 Modulation methods

The three basic methods for modulating a carrier wave are ***amplitude modulation***, ***frequency modulation***, or ***phase modulation***. In amplitude modulation, the amplitude of the carrier signal is increased by a fixed amount each time a digital 1 is communicated; in frequency modulation, the frequency is changed; and in phase modulation, the phase is shifted. These modulation methods are shown in Figure 1.18, where the digital signal 101 is being sent. You should note, however, that modulated signals are not necessarily digital. Carrier frequencies and modulation have long been used to transmit analog radio and television signals as well.

The name *bandpass transmission* arises from the fact that in this method of data communication, the carrier signal lies in the center of a ***frequency band***, called a ***channel***, that is allocated for communication. The sender and receiver both know the channel assigned to them. The sender sends information using only those frequencies that lie within its channel, and the receiver listens for communications only within that channel.

Communication across optical fiber or via electromagnetic waves in free space lends itself to bandpass transmission using carrier signals. Different colors of light actually have different frequencies that can be divided into bands or channels when communicated along optical fiber. The spectrum of light is shown in Figure 1.19. Electromagnetic waves also can be divided into frequency bands, as shown in Figure 1.20. Both analog and digital messages can be encoded using carrier signals in the form of light or other electromagnetic waves modulated to contain the information being sent.

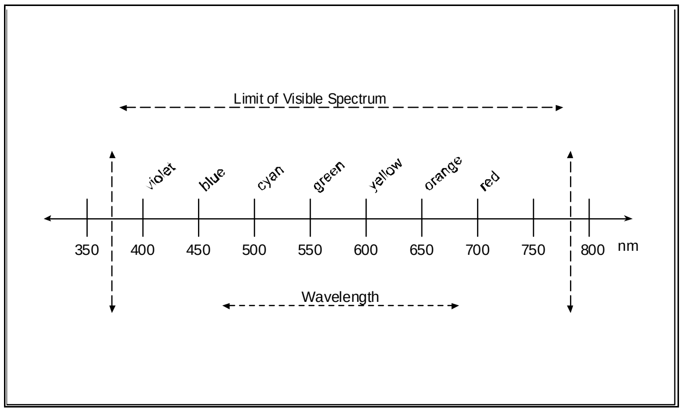


Figure 1.19 The spectrum of visible light

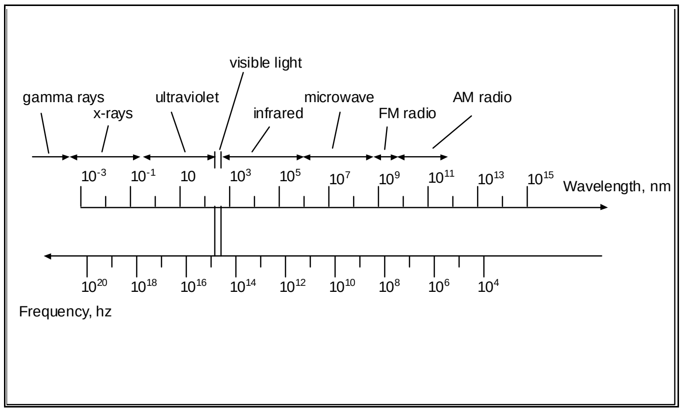


Figure 1.20 The electromagnetic spectrum

A continuously oscillating electrical voltage can also be used as a carrier signal. This fact has made it possible for the telephone system to handle digital data. Telephone systems were originally designed to communicate data in analog form. With the proliferation of computers in homes and offices, the need arose for network communication among them. The telephone system was already in place to accommodate this need, as long as a way could be found to send the digital signal along the analog telephone line. This is the role of a modem. The word ***modem*** stands for *modulator and demodulator*. A modem takes the data given to it by a computer and writes the 0s and 1s onto a continuously oscillating voltage using one of the modulation methods described above. Then at the other end of the call, another modem *demodulates* the signal for delivery to another computer.

Technical details for baseband and bandpass (modulated) communications have been studied and refined by engineers for many years. Standards have been developed so that data can be communicated around the world. The main organizations for developing standards are the International Telecommunications Union (ITU), the Institute for Electrical and Electronic Engineers (IEEE), and the Electronic Industries

Association (EIA). Many variations of communication methods exist, but the details are beyond the scope of this book. For the study of digital media, the important concepts to grasp are the essential differences between analog and digital data and the ways they are transmitted. We will discuss more details of data communication in the chapter on digital video, since the data-intensive, time-based nature of video makes data communication an important issue.

The discussion in this section has moved back and forth between analog and digital modes of communication, so it may help to summarize the key points:

- Both analog and digital data can be transmitted by means of copper wire, coaxial cable, optical fiber, and free space. The medium of transmission does not determine whether the communication is analog or digital. The form of the data does.
- Digital data can be transmitted by means of a baseband signal or a bandpass (i.e., modulated) signal. The baseband signal uses discrete pulses representing 0s and 1s. The modulated signal uses a carrier frequency and an encoding of data based on periodically altering the frequency, amplitude, or phase of the signal.
- Both analog and digital data can be communicated by means of a modulated signal. A carrier signal can be modulated to contain either analog or digital data.

We began this discussion by comparing how analog and digital data is communicated as a way of approaching the issue of bandwidth. The term ***bandwidth*** is one that you will hear frequently – in different contexts and with slightly different meanings. Bandwidth is important to the study of digital media because image, audio, and video files are large, and in the case of audio and video they are time-based. Thus, the rate at which these files can be created and transmitted is crucial.

In the next sections we'll discuss three definitions of *bandwidth* used in three different contexts. These usages are closely related, but not exactly the same, and the overloading of the term can lead to confusion. Separating the discussion into three different definitions should help to clarify the usage.

1.4.3 Bandwidth

1.4.3.1 Bandwidth as Maximum Rate of Change in Digital Data Communication

The first and perhaps most important usage of the term *bandwidth* (for our purposes) relates to digital data communication. In this context, keep in mind that we are talking about the transmission of discrete 0s and 1s. This transmission can be done by discrete pulses – that is, discrete changes of voltages in baseband data transmission. Alternatively, in the case of modulated communication, the data can be communicated by discrete changes in the frequency, amplitude, or phase of a carrier signal. The question with regard to bandwidth in this context is this: How fast can the signal be changed? That is, how fast can the sender change the signal from a voltage of V to a voltage of –V and back again? Or, in the case of modulated communication, how fast can the sender change the amplitude of the carrier signal (or the frequency or the phase)? This is not a matter of the sender only. Keep in mind that the receiver must be able to understand the changing signal as well.

An example from everyday life might help to clarify the concept of bandwidth used in this context. What if you wanted to tell a friend something *really fast*? How fast can you talk and still speak clearly? Don't forget that your friend has to be able to keep up with your talking; he or she has to be able to understand. The maximum rate at which you can talk and your friend can understand is the bandwidth of the communication. Notice that this has nothing to do with the speed of sound. Try another example. What if you had to send Morse code by means of a blinking flashlight? How fast could you send the code? The speed at which you can send the code is limited by how fast the hardware (your flashlight) can be operated and how fast your hand can click it. Even if you could flash the flashlight as fast as you wanted, at some point the person looking at it wouldn't be able to distinguish one signal from the next. Again, the bandwidth has nothing to do with the speed of light. It's a matter of the sending and receiving devices.

In digital data communication, there's a limit on the rate at which digital signals can be changed – whether it is voltages across copper wire or coaxial cable, light across optical fiber, or an electromagnetic wave changing frequency or amplitude from a satellite transmission. Just as we can move our vocal cords only so fast, the sending device can change its signal only so fast. The physical properties of the transmission medium also have an impact on the way the signal degrades and errors are introduced. At the receiving end of the transmission, there are limits on how fast the receiver can read and distinguish the signal correctly. Whatever the transmission medium and the engineering of the sending and receiving devices, there is a maximum rate of change for that communication system. This is the system's bandwidth.

Bandwidth is measured in cycles per second or Hz. Think about what this implies. A baseband transmission system with a bandwidth of 5000 Hz can *cycle* through its signal – that is, it can change its signal from one voltage level to a different one and then back again – at a maximum rate of 5000 times per second. We can say this another way: Every 1/5000th of a second, this system can communicate two things. If one voltage represents a 0 and another represents a 1, then the system can transmit a 0 and a 1 every 1/5000th of a second. This means that it can transmit 10,000 bits every second. Thus, we have the following definition:

key equation

Assume that a signal is sent with two possible signal levels and a bandwidth of b Hz. Then the data rate, d, in bits/s is

$$d = 2b$$

Equation 1.6

It makes sense that to get the data rate, you multiply the bandwidth by two. Bandwidth is defined by how fast the signal can change. Change implies that first there must be one thing, and then another – two different pieces of information. Those pieces of information, in this context, are bits.

What if more than one signal level is permitted? What if four instead of two voltages are possible? Instead of having one voltage represent 0 and the other represent 1, we could have one voltage represent 00, the second represent 01, the third represent 10, and the fourth represent 11. In this way, each change of voltage would transmit two bits instead of one. Allowing more than two signal levels such that more than one bit can be communicated at a time is called ***multilevel coding***. To generalize,

Assume that a signal is sent with k possible signal levels and a bandwidth of b Hz. Then the data rate, d, in bits/s is

$$d = 2b\log_2(k)$$

Equation 1.7

key equation

An example of this is shown in Figure 1.21. Four possible voltage levels are transmitted. If the lowest level is transmitted, it is meant to communicate 00. The second lowest level communicates 01, the third 10, and the highest level 11. Thus, two bits are communicated at a time.

It seems that if you can increase the data rate simply by increasing the number of voltage levels allowed, then why not have hundreds of voltage levels? The reason has to do with the engineering of the hardware. The more voltage levels, the more complex the hardware must be to be able to read and interpret the different levels, and the more change there is for error. Thus, the number of voltage levels used is generally not very high.

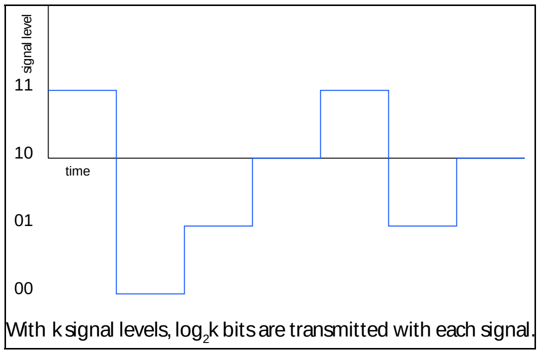


Figure 1.21 Data rate as determined by number of signal levels

1.4.3.2 Bandwidth of a Signal in Terms of Frequency

The ***bandwidth of a signal*** is easier to define. In this context, we are talking about a signal that is sent in the form of a wave. As discussed in Section 1.2.2, any complex ***periodic waveform*** can be decomposed – both mathematically and physically – into frequency components that are simple sine waves. These frequency components range from some minimum to some maximum. Then the bandwidth of a signal is the difference between the maximum and minimum frequency components. We will use the term ***width of a signal*** to avoid confusion with the other use of the term *bandwidth*.

For a signal that can be represented as a periodic waveform, let $f_{\max}$ be the frequency of the highest-frequency component and let $f_{\min}$ be the frequency of the lowest-frequency component. Then the width of the signal, w, is

$$w = f_{\max} - f_{\min}$$

Equation 1.8

key equation

For example, the complex waveform at the bottom of Figure 1.6 is composed of three frequencies: the notes C (264 Hz), E (330 Hz) and A (440 Hz). The difference between the maximum and minimum frequency components – 176 Hz, in this case – is the width of the signal. (Note that in this definition, we make no assumption about the nature of the data communicated by the signal. It could be either analog or digital.)

This use of the term *bandwidth* differs from the one in the previous section in that we are no longer dealing with the maximum rate that a signal can change on a transmission medium. Instead, we are talking about the frequency range for a particular signal, assuming that the signal is sent as a waveform. In this case, the signal in question may be transmitted on a carrier signal that lies in the center of a frequency band, called a *channel*. The significance of this usage lies in the fact that the width of the signal must fit within the width of the channel on which it is transmitted. Otherwise, some information will be lost. For example, high definition television (HDTV) requires a large bandwidth so that the television "frames" can be transmitted at a rate of 30 per second. For this to be possible, the bandwidth of the channel on which HDTV is broadcast must be large enough to accommodate the bandwidth of the HDTV signal. This leads us to another use of the term *bandwidth*, as it relates to channels.

1.4.3.3 Bandwidth of a Communication Channel in Terms of Frequency

When data is communicated across the airwaves, it is sent along some particular channel, which is a band of frequencies. The sender communicates within its designated frequency band, and the receiver tunes to that band to receive the communication. The range of frequencies allocated to a band constitutes the ***bandwidth of a channel***. (We could also call this simply the ***width of a channe***l, since it correlates with the width of a signal.) .In the case of the airwaves for television and radio, the Federal Communications Commission allocates channels of an appropriate bandwidth large enough to accommodate the type of communication. For example, each AM radio station is allocated a bandwidth of 10 kHz. FM radio stations have bandwidths of 200 kHz. Analog television has a bandwidth of about 6 MHz. Digital high definition television (HDTV) requires a bandwidth of approximately 20 MHz. Again in this usage of the term *bandwidth*, we are talking about data that is transmitted by means of a carrier signal of a given frequency that lies at the center of channel, but we make no assumptions about the nature of the data being communicated. Both analog and digital data can be sent by means of a carrier signal. However, there are repercussions to modulating the carrier signal so that it contains data, regardless of whether it is analog or digital. Modulation adds frequency components called ***sidebands*** to the original carrier signal, and these sidebands must lie within the designated channel. Thus, the bandwidth of a channel affects the amount of information that can be communicated.

RADIO
AM, 535 kHz to 1.7 MHz
short wave radio, 5.9 MHz to 26.1 MHz
CB radio, 26.96 MHz to 27.41 MHz
FM radio, 88 MHz to 108 MHz, allocated in 200 kHz channels
TELEVISION

54 to 88 MHz for channels 2 to 6
174 to 216 MHz for channels 7 to 13
470 to 890 MHz for UHF channels 14 to 83

Table 1.4 Frequency bands for radio and television

This definition of bandwidth raises a number of relevant questions. How is an appropriate bandwidth determined for AM radio, FM radio, television, and digital HDTV? What makes 10 kHz, 200 kHz, and 6 MHz, and 20 MHz, respectively, the right sizes? How does modulation of a carrier signal give rise to sidebands? What are the frequencies of these sidebands, and how do they affect the bandwidth requirements for channels? These questions will be examined in more detail in Chapter 6.

1.4.4 Data Rate

1.4.4.1 Bit Rate

The main goal in this section is to give you a basic understanding of the terminology used to compare data communication systems and the rates at which they transmit data. Let's return to the first definition of bandwidth: the maximum rate of change of a signal, as a property of the communication system on which the signal is being sent. This definition is closely related to ***data rate*** or ***bit rate***. In fact, bandwidth is often loosely used as a synonym for *data rate* or *bit rate*, and this has become widely accepted. However, in our discussion, we distinguish between the terms. Bandwidth is measured in cycles per second – Hertz. Data rate is measured in bits per second – more precisely, in kilobits per second (kb/s), kilobytes per second (kB/s), megabits per second (Mb/s), megabytes per second (MB/s), gigabits per second (Gb/s), or gigabytes per second (GB/s). If measured in bits per second, *data rate* is synonymous with *bit rate*. (The inconsistency in the definitions of *kilo-*, *mega-*, and *giga-* that we noted in the section on data storage carries over here. Generally, powers of ten define these terms when they are associated with data rate, and that's the convention adopted in this book.)

Recall that bandwidth and data rate are related by the equation $d = 2b\log_2(k)$. where *k* is the number of different signal values that can be transmitted. What we did not mention earlier is that in this equation, *d* is a theoretical data rate – a maximum that is not achievable in reality. The actual amount of data that can be sent per unit time is limited by the noise that is present in any communication system. No signal can be sent with perfect clarity over an indefinite span of space and time. Some amount of noise is introduced by electromagnetic interference. If too much noise is introduced, the receiver cannot always interpret the signal correctly. Some transmission systems are more susceptible to noise than others, and this lowers their actual achievable data rate.

A refinement of the relationship between data rate and bandwidth is given by ***Shannon's theorem***, which quantifies the achievable data rate for a transmission system that introduces noise. According to Shannon's theorem, $c = b\log_2(1 + s/p)$. s is a measure of the signal power, and *p* is a measure of the noise power. (Note that *s/p* is another application of the signal-to-noise ratio discussed above.) For our purposes, we don't need to do precise

Aside: Claude Shannon (1916-2001) was an American mathematician and electrical engineer educated at the Massachusetts Institute of Technology and later employed at Bell Labs. His work has been seminal in information theory.

measurements of signal and noise and calculations of data rates, but this theorem helps us to understand the origin of data rates reported for certain digital communication systems. These data rates arise from the rate at which a signal can be changed on the given transmission medium coupled with the amount of noise typically introduced in proportion to the actual signal.

Data rate is important in three aspects of digital media – not only in communicating the data, but also in capturing it, and (in the case of audio and video) playing it. Data rate is important in digital data communication because no one wants to wait an unreasonable length of time to transfer pictures, sound, and video from one place to another.

Refer back to Table 1.2, which gives typical file sizes for digital images, audio, and video. With these file sizes in mind, consider the situations in which you might want to send a digital picture, sound, or movie from one computer to another. You may have posted an image on your web page, and someone at another location has gone to your page and is downloading the image. How long will this take? It depends on the image size and the type of network connection. Table 1.5 lists some common network types and their typical data transfer rates as of the writing of this chapter. With this information, you can estimate the time it would take to download, for example, a 2.25 MB digital image (from the example in Table 1.2). Over a network connected by means of a 28.8 kb/s telephone modem, the file would take over ten minutes – far too long for the patience of most web users. In contrast, a cable modem achieving a data transfer rate of 20 Mb/s can download the image file in under a second.

It's even more important to have a high-speed connection for the transfer of digital audio and video files. First of all, video files are larger than still images, containing thousands of images in the form of frames, along with sound. If you want to download a DVD-formatted video so that you can play it later, you have a very large file to deal with. Secondly, digital audio and video are both time-based media. What if you want to play an audio or video transmission in real-time, as it downloads? If this is the case, the communication link must be fast enough to deliver the data so that it can be played at the appropriate speed. For the example audio file in Table 1.2, it would be necessary to transmit data at a rate of 1,411,200 kb/s. For the example video file, the rate would be 222,595,200 kb/s. None of the data connection types in Table 1.5 can accommodate such a data rate. This is why compression becomes so important in digital media applications.

Wide Area Network	
Type of Data Connection	**Data Rate**
telephone modem	28.8-56 kb/s
ISDN (Integrated Services Digital Network)	64-128 kb/s
ADSL (Asymmetric Digital Subscriber Line)	1.544-8.448 Mb/s (downstream) 16-640 kb/s (upstream)
ADSL2	0.8-3.5 Mb/s up, 5-12 Mb/s down
ADSL2+	1-3.5 Mb/s up, 24 Mb/s down
VDSL (Very High Bit DSL)	12.96-55.2 Mb/s (~12 Mb/s down and 52 Mb/s up, or ~26 Mb/s symmetrical at 1000 feet, 10 Mb/s at 4000 feet)

Cable modem	20-40 Mb/s
VDSL2	50-250 Mb/s
Local Area Network	
Type of Data Connection	**Data Rate**
Token ring	16 Mb/s
Ethernet (10base-X)	10 Mb/s
Fast ethernet (100base-X)	100 Mb/s
FDDI	100 Mb/s
Gigabit ethernet	1 Gb/s
Wireless 802.11b	11 Mb/s
Wireless 802.11g	54 Mb/s
Computer Interfaces	
Type of Data Connection	**Data Rate**
Serial	10-230 kb/s
Parallel	8 Mb/s
SCSI 1	12 Mb/s
SCSI 2	80 Mb/s
Fast wide SCSI	160 Mb/s
SCSI (various ultra versions)	320-2560 Mb/s
USB, USB2	12-480 Mb/s
SDI (serial digital interface)	143-360 Mb/s
Firewire (IEEE 1394)	400-800 Mb/s
DMA ATA	264-1064 Mb/s

Table 1.5 Data transfer rates for common communication links

Data rate is important in capturing digital video, since video is time-based and the most data-intensive of the media. When digital video is captured, motion and sound are happening in real time. Motion is captured as a sequence of frames, with a common frame rate of 30 frames per second. Each frame is a digital image comprised of pixels defined by their color values, often three bytes per pixel. Sound is captured in a sequence of audio samples, often four bytes per sample at a sampling rate of 44.1 kHz. Sound and image together generate a large amount of data, on the order of 30 MB per second or more if uncompressed. This data must move from the digital camera to the computer and be stored at a rate that keeps up with the live motion that is being captured. Thus, the data rate of the connection from camera to computer and the storage rate on the hard disk are crucial. Table 1.5 gives the data transfer rates of commonly-used camera-to-computer connections as of the writing of this chapter. Firewire (IEEE 1394) is a commonly-used physical connection between a digital video camera and a computer for live video capture because of its high data transfer rate.

CD Drives (Compact Disc)	
1X	150 kB/s (1200 Kb/s or 1.2 Mb/s)
2X	300 kB/s
8X	1200 kB/s
52X	7.8 MB/s
DVD Drives (Digital Versatile Disc or Digital Video Disc)	

1X	1.32 MB/s
16X	21.09 MB/s
Note: One MB is assumed to be 2^{20} bytes in this table.	

Table 1.6 Data transfer rates for common storage devices. (Not all generations are listed.)

Data rate is also an issue when you play a digital ***CD*** or ***DVD*** (***compact disc*** and ***digital versatile disk***, respectively). In this case, you don't have to transfer the data over a network. You already have it on a CD or DVD in your possession. But the CD or DVD player needs to have a data transfer rate that is adequate for your medium. Consider, for example, a music CD. Each strain of music is composed of audio samples. If the CD player cannot play the data fast enough, it will not be possible to play the music at the correct speed. CD-player speeds have evolved steadily since the 1980s. The first CDs were playable at a rate of 1200 kb/s. The next generation could play at twice the rate – 2400 kb/s – and were called 2X speed. Each subsequent generation was based on the speed of the original CD players. The current 8X to 52X CD players now range from 1200 kB/s to 7800 kB/s. Similarly, the first generation of DVD-ROMs had a data transfer rate of 1.32 MB/s, and all subsequent generations have been measured as multiples of this original benchmark.

1.4.4.2 Baud Rate

A term close in meaning to *bandwidth* and *bit rate* is ***baud rate***. As is the case with bandwidth, there is some confusion – or at least lack of agreement – about the definition of *baud*. The most precise definition of *baud rate* is "the number of changes in the signal per second, as a property of sending and receiving devices." This would be a property measured in cycles per second, i.e., Hertz. Under this definition, *baud rate* is synonymous with bandwidth, not *bit rate*. As you saw earlier in this chapter, if the sending device uses more than one signal level – say, *k* signal levels – then the bit rate, *d*, is given by the equation $d = 2b\log_2(k)$. Thus, to convert from baud rate to bit rate, you have to consider how many different signal levels are possible. However, just like bandwidth, the term *baud rate* is used loosely. Baud rates for telephone modems are commonly reported in bits per second rather than cycles per second. For example, current telephone modems are often described as having a baud rate of 56 kb/s (whereas this is actually their bit rate). In light of this confusion, here are the main points to understand:

Aside: *Baud rate* is named for J.M.E. Baudot (1845-1903), a French engineer who invented the Baudot telegraph code and the first successful teleprinter.

- *Baud rate* is close in meaning to *bandwidth* as bandwidth relates to digital data communication. The main difference is that *baud rate* is usually used to refer to sending and receiving devices, whereas bandwidth has other meanings related to frequencies over the airwaves.
- A device like a modem can have a maximum baud rate as well as an actual baud rate. The actual baud rate is the rate agreed upon between sender and receiver for a particular communication.

- What is often reported as a baud rate is really a bit rate. (But bit rate is generally what you want to know anyway, so no harm done.) To be precise, baud rate and bit rate are related by the equation $d = 2b\log_2(k)$.

1.5 Compression Methods

1.5.1 Types of Compression

For good fidelity to the original source, it is necessary that image, sound, and motion be digitized at a fine degree of resolution and with quantization levels covering a wide dynamic range. This typically translates to thousands or millions of samples, each of which is, say, two to four bytes in size – sometimes larger. In short, digital media files are usually very large, and they need to be made smaller – compressed, that is. Without compression, you probably won't have the storage capacity to save as many files as you'd like to, and you won't be able to communicate them across networks without overly taxing the patience of the recipients. On the other hand, you don't want to sacrifice the quality of your digital images, audio files, and videos in the compression process. Is it possible to reduce the size of digital media files with no significant loss of quality?

The answer is *yes*. Compression algorithms can be divided into two basic types: ***lossless compression*** and ***lossy compression***. In lossless compression, as the name implies, no information is lost between the compression and decompression steps. Compression reduces the file size to fewer bits. Then decompression restores the data values to exactly what they were before the compression. Lossy compression methods, on the other hand, sacrifice some information. However, these algorithms are designed so that the information lost is generally not important to human perception. In image files, it could be subtle changes in color that the eye cannot detect. In sound files, it could be frequencies that are imperceptible to the human ear.

Aside from the broad categories of lossy and lossless compression, you will see other labels given to types of compression algorithms, including *dictionary-based, entropy, arithmetic, adaptive, perceptual* and *differential compression methods*. Dictionary-based methods (e.g. LZW compression) use a look-up table of fixed-length codes, where one code-word may correspond to a string of symbols rather than a single symbol in the file being compressed. Entropy compression uses a statistical analysis of the frequency of symbols and achieves compression by encoding more frequently-occurring symbols with shorter code-words, with one code-word assigned to each symbol. Shannon-Fano and Huffman encoding are examples of entropy compression. Arithmetic encoding benefits from a similar statistical analysis, but encodes an entire file in a single code-word rather than creating a separate code for each symbol. Adaptive methods gain information about the nature of the file in the process of compressing it, and adapt the encoding to reflect what has been learned at each step. LZW compression is by nature adaptive because the code table is created "on the fly" during compression and decompression. Huffman encoding can be made adaptive if frequency counts are updated as compression proceeds rather than being collected beforehand; the method adapts to the nature of the data as the data is read. Differential encoding is a form of lossless compression that reduces file size by recording the difference between neighboring values rather than the values themselves. Differential encoding can be applied to digital images, audio, or video. In the sections that follow, we will look more closely at three algorithmic approaches to lossless compression. Other methods will be examined in detail in the chapters on digital image, audio, and video.

The ***compression rate*** of a compression algorithm is the ratio of the original file size *a* to the size of the compressed file *b*, expressed as *a:b*. Alternatively, you can speak of the ratio of *b* to *a* as a percentage. For example, for a file that is reduced by coompression to ½ of its original size, you could say that 50% compression is achieve, or alternatively, that the compression rate is 2:1.

1.5.2 Run-Length Encoding

Run-length encoding (***RLE***) is a simple example of lossless compression. It is used in image compression. For example, files that you see with the *.bmp* suffix – a Microsoft version of bitmap image files – optionally use run-length encoding. Here's how it works. An image file is stored as a sequence of color values for consecutive pixel locations across rows and down columns. If the file is in RGB color mode, there are three bytes per pixel, one for each of the red, green, and blue color channels. If the file is grayscale, there is one byte per pixel. For simplicity, let's use a grayscale file in this example. (Extending the algorithm to three bytes per pixel is straightforward.) Since each pixel position is encoded in one byte, it represents one of 256 grayscale values. (You recall that 2^8 = 256, so eight bits can encode 256 different things.) Thus, a grayscale image file consists of a string of numbers, each of them between 0 and 255. Assume that the image has dimensions 100 × 100, for a total of 10,000 pixels. Assume also that the pixels are stored in row-major order, which means that the values from a whole row are stored from left to right, then the next row from left to right, and so forth.

You can easily imagine that in many images, there could be rows with strings of repeated grayscale values. The simple idea in run-length encoding is that instead of storing each of the 10,000 pixels as individual values, it can be more concise to store number pairs (c, n), where c indicates the grayscale value and n indicates how many consecutive pixels have that value. For example, say that in a 10,000 pixel grayscale image, the first 20 pixels are:

255 255 255 255 255 255 242 242 242 242 238 238 238 238 238 238 255 255 255 255

The run-length encoding of this sequence would be

(255, 6), (242, 4), (238, 6), (255, 4)

Let's compare the number of bytes needed to store the run-length encoded version of this line of pixels versus the number of pixels needed to store it originally. For this example, we'll assume that everything has to be rounded to units of bytes. (We could compute how many bits are needed, but working with bytes suffices to make the point in this example.) Without RLE, 20 pixels require

$$20\, pixels * 1\, byte / pixel = 20\, bytes$$

To determine the number of bytes required with RLE, we first need to figure out how many bytes are needed to store n in each (c, n) pair. Clearly, *c* can be stored in one byte since its values range from 0 to 255. Now we have to consider how large *n* could possibly be. In our example, the image has 10,000 pixels in it. It is possible that all 10,000 pixels have the same color, in which case $n = 10{,}000$. To store 10,000, we'd need 14 bits. ($10{,}000_{10} = 10011100010000_2$.) Since we're assuming we allocate memory only in byte increments, this means we need two bytes.

Say that the run-length encoding algorithm scans the image file in a preprocessing step to determine the size of the largest "run" of colors. Let the size of this largest run be *r*. Try some examples values for *r*. What if the largest run is 300 – that is, *r* = 300? Then how many bits would you need to store the second value in each (*c,n*) pair? You can represent the numbers 0 through 255 with eight bits, right? So eight bits is not enough, because you can't represent a number as big as 300. You can represent the values 0 through 511 with nine bits, so nine bits IS enough. If you round this up to the nearest byte, that's two bytes. If you think this through intuitively, you should be able to see that the formula for figuring out how many bytes you need to represent a number that can be anywhere between 1 and *r* is $b = \left\lceil \frac{\log_2(r+1)}{8} \right\rceil$. If you need two bytes to store *n* in the (*c,n*) pair, then in the example above, the RLE encoded string of values would require 12 bytes rather than 20.

Now, what if you have a sequence like the following?

255 255 255 255 243 240 242 242 242 241 238 238 237 237 237 237 255 255 255 255

The run-length encoding of this sequence would be

(255, 4), (243, 1), (240, 1), (242, 3), (241, 1), (238, 2), (237, 4), (255, 4)

In this case, run-length encoding actually requires more rather than fewer bytes than the original uncompressed image – 24 as opposed to 20.

The actual implementation of this might be different. Rather than determine the largest *n* in advance and then set the bit depth of *n* accordingly, it makes sense to choose a bit depth of *n* in advance. Then if more than *n* consecutive pixels of the same color are encountered, the "runs" are divided into blocks. For example, if one byte is used to represent each *n*, then the largest value for *n* is 255. If 1000 consecutive "whites" exist in the file (with color value 255), they would be represented as

(255, 255), (255, 255), (255, 255), (255, 235)

The image in Figure 1.22 is a good example where run-length encoding is beneficial. Disregarding heading information on the image file, the image contains $100 \times 100 = 10{,}000$ pixels. With run-length encoding, again disregarding the size of the header on the file, the file requires 1084 bytes. The compression rate is $\frac{10000}{1084}$, or about 9:1. This is an excellent compression rate, but obviously most images will not have such clearly-defined color areas, and so few of these areas. (The encoded file size was obtained by saving the file as a *.bmp* image in an application program and then looking at the file properties.)

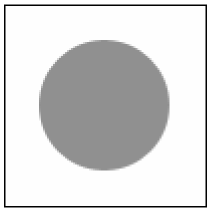


Figure 1.22 Simple image that is a good candidate for run-length encoding

Run-length encoding is a simple algorithm that gives acceptable results on some types of images, with no risk of loss of quality. It should be clear that no information is lost in the encoding process. We still have precisely the same values for the pixels after the file is encoded and then decoded. The encoded values are just represented in a way that is potentially more concise. In practice, RLE algorithms are fine-tuned and do not operate in precisely the way shown here, but the main principle is always the same.

The second example of run-length encoding given above illustrates a case where the encoding did not result in a smaller file. You should note that for *any lossless compression algorithm and for any length file, there will always exist at least one case where the algorithm does not reduce the size of an input file of that length*. Why is this necessarily so? You should be able to prove this to yourself with what is called a *counting argument*. Let's walk through the argument together.

- Consider an arbitrary file that you want to compress. Whether it is an image, audio, or video file, it is just a sequence of bits.
- Consider all the files that have n bits in them. How many different files of length n are there? Each bit can have a value of zero or one. So there's 1 zero-bit file, 2 one-bit files, 4 two-bit files, etc. That's 2^n files with length n.
- Assume that we have a lossless compression algorithm that can reduce the size of any file given to it as input. If the algorithm is given a file of length n, it will produce a compressed file of length $\leq n-1$. How many different files are there of length $\leq n-1$? There are $1+2+4+...+2^{n-1}=2^n-1$ such files.
- A key to this argument is that we are talking about *lossless* compression algorithms. Since the algorithm we are considering is lossless, it cannot be the case that two files of length n compress to yield the same file (some file of length $\leq n-1$). If two files did compress to yield the same file, then in decompression, how would it be determined which of the two original files was the source? In such a case, some information would be lost, and the algorithm could not be lossless. Thus, we want to map each of the input files of length n to exactly one compressed file of length $\leq n-1$.
- We've discovered a contradiction. There are fewer files of length $\leq n-1$ than there are files of length n. This argument shows that for any lossless algorithm and for any length of input file, there will exist cases where the algorithm does not reduce the size of the input file.

Lossless compression algorithms are applied in situations where loss of data cannot be tolerated. This is the case in the compression of both text and of binary-encoded computer programs, for example, and thus lossless algorithms are used in tools such as *gzip* and *compress* (on the Unix platform) or *pkzip* and *winzip* (on the Windows platform). Sound files do not lend themselves well to lossless compression. Consecutive audio samples with the same value are unusual, so RLE is not very effective on sound. Image files are better candidates for lossless compression. While an image file is still being edited, it is preferable to keep as much of the original data as possible. Thus, a number of image file formats – PNG and TIFF, for example – offer forms of lossless compression such as LZW. (See Chapter 3 for more information on image file types.) Lossless compression can also be used as one step in a more complex algorithm that does include lossy steps. This is the case with Huffman encoding, one step in the JPEG compression algorithm. We will look at Huffman encoding and LZW compression in more detail in Chapter 3.

1.5.3 Entropy Encoding

Claude Shannon's work in information theory sheds light on the limits of lossless compression and methods for achieving better compression rates with entropy encoding. Recall that entropy encoding works by means of variable-length codes, using fewer bits to encode symbols that occur more frequently, while using more bits for symbols that occur infrequently. Shannon's equation, below, gives us a way a judging whether our choice of number of bits for different symbols is close to optimal.

Borrowing the term *entropy* from physics, Shannon defines the ***entropy of an information source*** S as follows:

key equation

Let S be a string of symbols and p_i be the frequency of the i^{th} symbol in the string. (p_i can equivalently be defined as the probability that the i^{th} symbol will appear at any given position in the string.) Then

$$H(S) = \eta = \sum_i p_i \log_2\left(\frac{1}{p_i}\right)$$

Equation 1.9

Applying ***Shannon's entropy equation***, you can determine an optimum value for the average number of bits needed to represent each symbol-instance in a string of symbols, based on how frequently each symbol appears. Shannon proves that you can't do better than this optimum. You'll see more clearly what this means in the example below. Let's step through the equation and an example to get a better understanding.

Think about an image file that has exactly 256 pixels in it, and each pixel is a different color. Then the frequency of each color is 1/256. Thus, Shannon's equation reduces to

$$\sum_0^{255}\frac{1}{256}\left(\log_2\left(\frac{1}{\frac{1}{256}}\right)\right) = \sum_0^{255}\frac{1}{256}\left(\log_2(256)\right) = \sum_0^{255}\frac{1}{256}(8) = 8$$

This means that the average number of bits needed to encode each color is eight, which makes sense in light of the fact that $\log_2 256 = 8$.

But what if you had an image file that had many instances of some colors, but only a few instances of others? For example, what if you had an image file of 256 pixels and only eight colors in the image with the following frequencies:

color	**frequency**	**optimum number of bits to encode this color**	**relative frequency of the color in the file**	**product of columns 3 and 4**

black	100	1.356	0.391	0.530
white	100	1.356	0.391	0.530
yellow	20	3.678	0.078	0.287
orange	5	5.678	0.020	0.111
red	5	5.678	0.020	0.111
purple	3	6.415	0.012	0.075
blue	20	3.678	0.078	0.287
green	3	6.415	0.912	0.075

Table 1.7 Color frequencies

Then Shannon's equation becomes

$$\frac{100}{256}\log_2\left(\frac{256}{100}\right)+\frac{100}{256}\log_2\left(\frac{256}{100}\right)+\frac{20}{256}\log_2\left(\frac{256}{20}\right)+\frac{5}{256}\log_2\left(\frac{256}{5}\right)+\frac{5}{256}\log_2\left(\frac{256}{5}\right)+$$
$$\frac{3}{256}\log_2\left(\frac{256}{3}\right)+\frac{20}{256}\log_2\left(\frac{256}{20}\right)+\frac{3}{256}\log_2\left(\frac{256}{3}\right)\approx$$
$$0.530+0.530+0.287+0.111+0.111+0.075+0.287+0.075\approx 2.006$$

So what's the significance of this? Let's see how Shannon's equation can be applied to compression. Consider each term individually. The first term corresponds to the color black. Relative to the size of the entire file, the symbol for black carries $\log_2\left(\frac{256}{100}\right)=1.356$ bits of information every time it appears in the file, a measure of how many times black appears relative to the size of the file. The third term corresponds to the color yellow. Yellow conveys $\log_2\left(\frac{256}{20}\right)=3.678$ bits of information each time it appears. The implication is that if we were to encode each color with a number of bits equal to its information content, this would be an optimum encoding. That is, we couldn't encode the file in any fewer bits. Overall, the minimum value for the average number of bits required to represent each symbol-instance in this file is 2.006. This implies that in an optimally-compressed file, the average number of bits used to encode each symbol-instance cannot be less than 2.006.

Keep in mind that the implication in Shannon's equation is that we don't necessarily need to use the same number of bits to represent each symbol. A better compression ratio is achieved if we use fewer bits to represent symbols that appear more frequently in the file. Algorithm 1.1, the ***Shannon-Fano algorithm***, describes one way that Shannon's equation can be applied for compression. It attempts to approach an optimum compression ratio by assigning relatively shorter code words to symbols that are used infrequently, and vice versa.

```
algorithm Shannon-Fano_compress
/*Input: A file containing symbols. The symbols could represent characters of text,
colors in an image file, etc.
Output: A tree representing codes for the symbols. Interior nodes contain no data.
Each leaf node contains a unique symbol as data.*/
```

```
{
  lst = a list of the symbols in the input file, sorted by their frequency of appearance
  code_tree = split_evenly(lst)
  }

algorithm split_evenly(lst)
/*Input:  A sorted list of symbols, lst.
Output:  A tree representing the encoding of the symbols in lst.*/
{
  if the size of lst = =1, then {
    create a node for the one symbol in lst
    put the symbol as the data of the node
    return the node
  }
  else {
    Divide lst into two lists lst1 and lst2 such that the sum of the frequencies in lst1 is
close as possible to the sum of the frequencies in s2.
    t = a tree with one node
    attach lst2 as the left child of t
    attach lst2 as the right child of t
    return t
  }
}
```

Algorithm 1.1

The algorithm takes a recursive top-down approach, each recursive step dividing the symbols in half such that the symbols in the two halves have approximately the same number of instances in the file being compressed. The algorithm returns a code-tree where the branches are implicitly labeled with 0s and 1s. Gathering the 0s and 1s down each path from root to leaf node gives you the code for the symbol related to the leaf node. In our example, the tree could look the one shown in Figure 1.23. Colors are abbreviated, and frequencies are shown beside the colors.

The Shannon-Fano algorithm for compression works toward achieving the theoretical minimum number of bits per symbol by the method described in Algorithm 1.1.

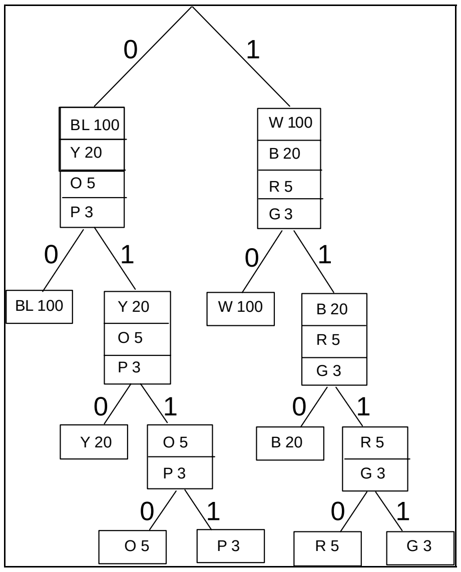


Figure 1.23 Example of the Shannon-Fano algorithm applied to compression

In this example, the colors would be encoded as shown in Table 1.8. The codes vary in length from two to four bits. How close is the number of bits per symbol used in this encoding to the minimum of 2.006 determined by Shannon's equation? All the symbols for black and white are encoded with two bits, all the symbols for yellow and blue are encoded with three bits, and so forth. This give us

$$(100*2)+(100*2)+(20*3)+(5*4)+(5*4)+(3*4)+(20*3)+(3*4)=584$$

The colors require 584 bits to encode. For 256 symbols in the file, this is an average of $\frac{584}{256}=2.28$ bits per symbol-instance – close to the minimum average number of bits per symbol given by Shannon's equation. If we assume that before compression, eight bits were used for each symbol, then the compression rate is $\frac{8}{2.28}$, which is about 3.5:1.

color	frequency	code
black	100	00
white	100	10
yellow	20	010
orange	5	0110
red	5	1110
purple	3	0111
blue	20	110
green	3	1111

Table 1.8 Codes resulting from application of Shannon-Fano algorithm

As an aside, you might also want to consider why the word *entropy* – borrowed from physics – has been adopted in the context of information theory and encoding. The most common definition you will see for the word *entropy* is *disorder*. However, a better definition in most situations is *multiplicity in the way something can be ordered*. Here's a simple example. On a pair of dice, there are six different ways you can roll a value of seven, while there is only one way to roll a value of two. Thus, the value seven has greater entropy than the value two. In the latter definition, entropy is a measure of the number of ways that you can produce a seven. How does this relate to Shannon's equation, and why did Shannon use the term *entropy* to describe the average minimum number of bits needed to represent each symbol in a string of symbols? You could think of it this way. A file that has a large number of symbols in it relative to the file size will yield a large number from Shannon's equation, as compared to a file that has a small number of symbols relative to its size. It has greater entropy because the more symbols you have, the more unique ways those symbols could be arranged. Thus, each encoded symbol carries with it more information, and you need a greater number of distinct codes.

More importantly for our purposes, when you see the term *entropy encoding* related to digital media, you should understand it to mean encoding that is based on a statistical analysis of the file or type of data being encoded. This analysis could yield the frequency of occurrence of each symbol in an actual file that is to be compressed. Alternatively, the analysis could yield the probability that certain symbols will appear in a *type* of file. For example, in text files, we can expect that some characters will appear more frequently than others. The letter *e* appears frequently in English-language text, while *q* and *x* are more rare. Either type of analysis can be used to create the frequency table upon which entropy encoding is based. Entropy encoding also implies variable-length codes. The benefit in compression arises from using shorter codes for more frequently occurring symbols.

Huffman encoding is another type of entropy encoding that is useful in image compression. We will see in Chapter 3 that it uses the same strategy of statistically-based variable-length encoding, while improving upon the Shannon-Fano algorithm in the compression rate.

1.5.4 Arithmetic Encoding

One drawback to the Shannon-Fano algorithm is that each symbol must be treated individually; each symbol has its own code, and that code must be represented in an integral number of bits. Huffman encoding has the same disadvantage. By Shannon's equation, we may be able to determine that an optimum encoding is achievable by using a non-integer number of bits for each code. In our example above, if it were possible to use exactly 1.3561 bits to encode black and white; 3.6781 bits to encode yellow and blue; 5.6781 bits to encode orange and red; and 6.415 bits to encode purple and green, then we could achieve an optimum compression rate. (Each of these values is given by $\log_2\left(\frac{1}{p_i}\right)$.) The problem is that using the Shannon-Fano algorithm, the optimum encoding isn't possible because we have to use an integer number of bits for each code.

Arithmetic encoding overcomes some of the disadvantage of the Shannon-Fano algorithm. Like the Shannon-Fano algorithm, arithmetic encoding is based on a statistical analysis of the frequency of symbols in a file. It differs, however, in that it derives additional advantage from encoding an entire file (or string of symbols) as one entity rather than creating a code symbol-by-symbol.

The idea is that a string of symbols will be encoded in a single floating point number. In theory, this floating point number can be represented in whatever number of bits is needed for the compression at hand. To implement this with no limit on the number of bits used, we would need infinite precision in the representation of floating point numbers, which is not possible. Let's put this issue aside for now. Later, we will give a sketch of how arithmetic encoding is implemented, but for the moment let's focus on the basic idea.

Arithmetic encoding uses the same strategy as entropy encoding, beginning with a list of the symbols in the input file and their frequency of occurrence. Say that you have a file that contains 100 pixels in five colors: black (K), white (W), yellow (Y), red (R), and blue (B). The frequency of appearance of each color in the file is given in Table 1.9. The frequencies are expressed as numbers between 0 and 1, shown in column two. Each color symbol is assigned a *probability interval* whose size corresponds to its frequency of occurrence, as shown in column three. The entire range between 0 and 1 is called the *probability range*, and the section assigned to one symbol a *probability interval*. For example, black's probability interval is 0–0.4. (Assume that these are half open intervals, e.g. [0,0.4)). The order of color symbols in the probability range is not important, as long as the symbols are given the same order by the encoder and decoder.

color	**frequency out of total number of pixels in file**	**probability interval assigned to symbol**
black (K)	40/100 = 0.4	0 – 0.4
white (W)	25/100 = 0.25	0.4 – 0.65
yellow (Y)	15/100 = 0.15	0.65 – 0.8
red (R)	10/100 = 0.1	0.8 – 0.9
blue (B)	10/100 = 0.1	0.9 – 1.0

Table 1.9 Frequencies of colors relative to number of pixels in file

Range	**Low value for probability interval**	**High value for probability interval**	**Symbol**
1−0 = 1	0+1∗0.4 = 0.4	0+1∗0.65 = 0.65	White
0.65−0.4 = 0.25	0.4+0.25∗0 = 0.4	0.4+0.25∗0.4 = 0.5	Black
0.5−0.4 = 0.1	0.4+0.1∗0 = 0.4	0.4+0.1∗0.4 = 0.44	Black
0.44−0.4 = 0.04	0.4+0.04∗0.65 = 0.426	0.4+0.04∗0.8 = 0.432	Yellow
0.432−0.426 = 0.006	0.426+0.006∗0.8 = 0.4308	0.426+0.006∗0.9 = 0.4314	Red
0.4314−0.4308= 0.0006	0.4308+0.0006∗0.9 = 0.43134	0.4308+0.0006∗1= 0.4314	Blue

Table 1.10 Values for encoding the example problem

Now let's consider only the first six pixels in the file to make the example manageable. These pixels have the colors white, black, black, yellow, red and blue – represented by the symbols W, K, K, Y, R, B. The arithmetic encoding algorithm assigns a floating point number to a sequence of symbols by successively narrowing the range between 0 and 1. In our example, the first symbol is W. This means that the floating point number encoding the total string will fall somewhere in the probability interval assigned to W, between 0.4 and 0.65, as shown at the step 2 of Figure 1.24.

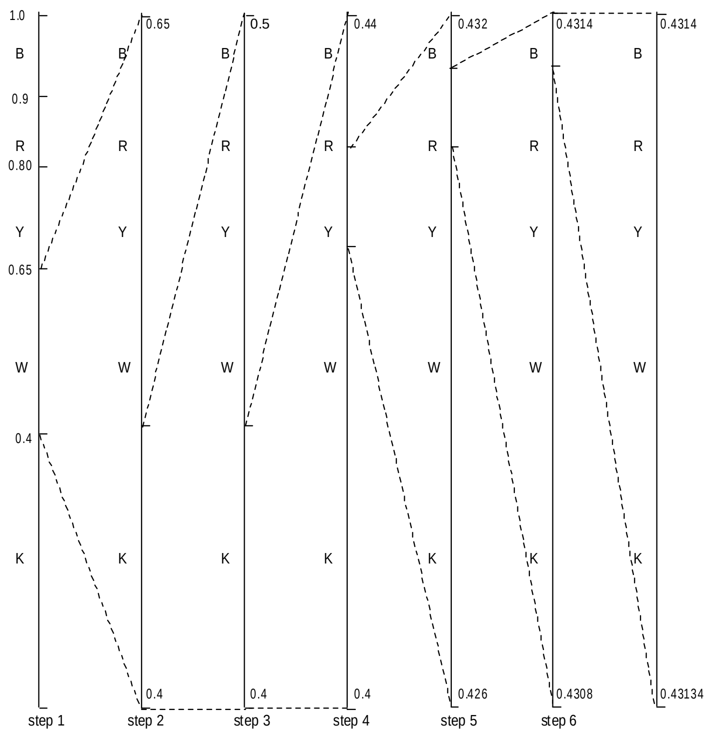


Figure 1.24 Example of arithmetic encoding

The second symbol is K. At step 2, the range has a size of 0.25, going from 0.4 to 0.65. K was originally assigned the probability interval 0 to 0.4. Moving to step 3, we want to narrow the probability range so that it begins at 0.4 and ends at 0.4 + 0.25∗0.4. Thus, the range at step three goes from 0.4 to 0.5. At successive steps, we narrow the range in accordance with the size and position of the symbol just read, as described in Algorithm 1.2. In the final step, the low value of the range can be taken as the encoding of the entire string. We've cut our example short by considering only the first six pixels in the file, but the idea is the same.

```
algorithm arithmetic_encoding
/*Input:  A string of symbols and their assigned probability intervals.
Output:  A floating-point number that encodes the string*/
  low = 0.0
  high = 1.0
  while input symbols remain {
    s = the next input symbol
    range = high – low
    /*s_high (s) represents the high value of symbol s's assigned probability interval,
and s_low (s) represents the low value of symbol s's assigned probability interval. */
    high = low + range * s_high(s)
    low = low + range * s_low(s)
  }
  return low
}
```

Algorithm 1.2

You can probably see for yourself how decoding would proceed. Given a floating point number, you can determine the first symbol of the encoded string by finding, from the initial probability range, the probability interval into which this number fits. The number 0.43135 fits in the interval assigned to W, so W is the first symbol from the encoded string. You now remove the scaling of W's interval by subtracting the low value from W's probability interval and dividing by the size of W's interval, yielding

$$(0.43135 - 0.4)/0.25 = 0.1254$$

The value 0.1254 lies in the probability interval of K. Thus, you subtract K's low range, 0, and divide by the size of K's interval, 0.4, giving

$$0.1254/0.4 = 0.3135$$

This value again lies in K's probability interval.

The computation proceeds as described in the decoding procedure given in Algorithm 1.3, yielding the values in Table 1.11. Although our example worked out nicely, with a maximum of five digits after the decimal point, you should be able to see that a larger example would require very high precision arithmetic. In fact, this would appear to be an issue that makes arithmetic encoding impractical in the implementation. Is there a way to implement arithmetic encoding without requiring infinite precision floating point arithmetic?

```
algorithm arithmetic_decoding
/*Input:        A floating point number, f, encoding a string of symbols, a list of
symbols encoded by the number, and the probability intervals assigned to these symbol.
Output:         The string of symbols, s, decoded
Assumptions:  A terminator symbol has been encoded at the end of the string*/

  symbolDecoded = NULL
   while symbolDecoded != TERMINATOR_SYMBOL {
      s = a symbol whose probability interval contains f
      output s
      /*Let s_high(s) represent the high value of symbol s's probability interval, and
s_low(s) represent the low value of s's probability interval*/
      range = s_high(s) – s_low(s)
      f = (f – s_low(s)) / range
   }
}
```

Algorithm 1.3

Floating point number f, representing code	**Symbol whose probability interval surrounds f**	**Low value for symbol's probability interval**	**High value for symbol's probability interval**	**Size of symbol's probability interval**
0.43135	W	0.4	0.65	0.25
(0.43135−0.4)/(0.65−0.4) =	K	0	0.4	0.4

0.1254				
(0.1254−0)/(0.4−0) = 0.3135	K	0	0.4	0.4
(0.3135−0)/(0.4−0) = 0.78375	Y	0.65	0.8	0.15
(0.78375-0.65)/(0.8−0.65) = 0.891666...	R	0.8	0.9	0.1
(0.891666... −0.8)/(0.9−0.8) = 0.916666...	B	0.9	1.0	0.1

Table 1.11 Values for decoding in the example problem

We've shown you the essential idea behind arithmetic encoding. However, actual implementations use integer arithmetic and bit shifting operations that effectively accomplish the procedure described here, but without requiring infinite precision floating point operations – in fact, without using any floating point operations at all. There are other issues that we have not dealt with here that have been fine-tuned in various implementations, including how to terminate the input string and how to speed up what can be a time-consuming compression method. The important facts to note about arithmetic encoding are these:

- Arithmetic encoding is a form of entropy encoding, where the number of bits that a symbol contributes to the compressed string is proportional to the probability of that symbol appearing in the original input string.
- In arithmetic encoding, an entire input string is encoded as one value. In comparison, Huffman encoding – another example of entropy encoding – uses variable-length codes assigned on a symbol-by-symbol basis.
- Because arithmetic encoding theoretically can encode a symbol in a fractional number of bits, it is closer to optimal than Huffman encoding.
- Arithmetic encoding can be applied as one step in JPEG compression of photographic images.
- IBM and other companies hold patents on algorithms for arithmetic encoding.

1.5.5 Transform Encoding

The compression algorithms we have looked at so far are all lossless. With these algorithms, a digital image, audio, or video file can be compressed, and the original data can be perfectly reconstructed when it is decompressed (not considering rounding errors that may result from floating point arithmetic).

The disadvantage of lossless methods is that they do not always give enough compression, especially for large audio and video files. Lossy methods are needed. Fortunately, these methods are designed so that the information lost is relatively unimportant to the perceived quality of the sound or images.

Lossy methods are often based upon ***transform encoding***. The idea is that changing the representation of data can sometimes make it possible to extract sounds or visual details that won't be missed because they are beyond the acuity of human perception. Thus, this type of compression begins with a *transform* of the data from one way of representing it to another. Two of the most commonly used transforms in digital media are the ***discrete cosine transform* (*DCT*)** and the ***discrete Fourier transform* (*DFT*)**. You should note that it is not the transform that is lossy. No information is lost in the DCT or DFT. However, when a transform is used as one step in a compression algorithm, it becomes possible to discard redundant or irrelevant information in later steps, thus reducing the digital file size. This is the lossy part of the process.

The discrete cosine transform is applied to digital images to change their representation from the spatial to the frequency domain. We introduced the notion of the *frequency of an image* earlier in this chapter, and we will go into more detail in Chapter 2. What we want to point out here is that transforming image data to the frequency domain with the DCT can be the first step in compression. Once you have separated out the high frequency components of an image, you can "remove" them. High frequency components correspond to quick fluctuations of color in a short space – changes that aren't easy for the human eye to see. When this information is removed, later steps in the compression (e.g., Huffman encoding) are even more effective. What we have just described is the basis of JPEG compression.

The discrete Fourier transform is applied to sound, transforming audio data from the temporal to the frequency domain. With the frequency components separated out, it is possible to determine which frequencies "mask" or block out other ones and then discard the masked frequencies. By this method, transform encoding is followed by perceptual encoding, and the result is a smaller audio file.

1.5.6 Compression Standards and Codecs

Thus far in this chapter, we have looked at types of compression methods and the concepts behind them. In practice, compression algorithms are implemented in a wide variety of ways that are fine-tuned and optimized in their details. It is not uncommon for methods to be used in combination with each other, as is the case with JPEG and MPEG compression, which combine the DCT, run-length encoding, and Huffman encoding for compression of photographic images. Some algorithms are standardized by official committees so that the various implementations all produce files in the same format. If a standardized algorithm is patented, then commercial companies must pay a license fee to implement the algorithm and sell it in a commercial product. Two prominent examples of standardized compression algorithms are DV for camcorder-generated digital video and the family of MPEG algorithms. Arithmetic encoding is an example of an image compression algorithm that is covered by patents.

Specific implementations of compression algorithms are called ***codecs*** – short for compression/decompression. The word *codec* is usually reserved for audio/video compression (as opposed to still images), since real-time decompression is just as important as initial compression with these time-based media. Some codecs are offered as shareware or freeware. Most codecs are commercial products. Codecs can be embedded in image, audio, or video processing programs, or they can be sold and used separately. Sorenson is an example of a codec that is embedded in other environments (e.g. QuickTime) and also available in a professional-grade version that runs apart from other application programs. The professional-grade Sorenson compressor is actually a suite of codecs that includes implementations of MPEG and DV compression and the standard Sorenson codec.

With most codecs, the compression rate can be adjusted by the user in accordance with the desired quality, up to the maximum compression ability of the codec. The choice may be offered in terms of quality, perhaps as a percentage or on a scale from one to ten. Alternatively, the user may have the option of compressing at different bit rates. The bit rate represents the number of bits that are transferred per second. Bit rate and compression rate are inversely related. Increasing the compression rate reduces the bit rate; if there are fewer bits after the data has been compressed, then fewer bits need to be transferred per second to play the sound or video in real time. A lower bit rate makes it possible to play the media on a device that has a relatively low transfer rate – like a CD-ROM as compared to a DVD player. However, the higher the compression rate, the more chance there is that the quality of the sound will degrade as compared to the original uncompressed audio. Fortunately, compression algorithms have evolved to the point where they can compress audio and video at a high rate without unduely sacrificing quality.

Table 1.12 lists examples of standardized or patented algorithms and popular codecs (as of the writing of this chapter). These algorithms and codecs will be examined more closely in the chapters on digital audio and digital video.

Examples of Standardized or Patented Compression Algorithms				
Algorithm	**Source Material**	**Developer**	**Compression Rate and/or Bit Rate***	**Comments**
LZ and variants	still images	• Lempel, Zev, and Welch	• varies with input • works best on images with large contiguous sections of the same color	• the basic compression algorithm for *.gif* files • optional compression for *.tiff* files • Unisys held the patent on LZW until 2003
arithmetic encoding	still images	• evolved through Shannon, Elias, Jelinek, Pasco, Rissanen, and Langdon	• varies with input • usually better than Huffman encoding	• a number of variants have been created, mostly patented by IBM
MP3 (MPEG 1, Audio Layer III)	audio	• invented and patented by an institute of the Fraunhofer Society	• 10:1 or greater compression rate • generally used for bit rates from about 96 to 192 kb/s	• Lame is an open-source MP3 encoder that gives high-quality compression at bit rates of 128 kb/s or higher.
AAC (Advanced Audio Coding; MPEG 2 version updated to MPEG-4)	audio	• patented by Dolby Labs in cooperation with other developers	• generally achieves same quality as MP3 with a 25 – 40% increase in compression rate • reports bit rates of 64 kb/s with good quality	• got recognition when used in Apple's iPod • also used in RealAudio for bit rates > 128 kb/s
MPEG-1, 2, 4	audio/video	• developed by a working group of the ISO/IEC • see ISO/IEC 11172 (MPEG-1), ISO/IEC 13818 (MPEG-2), and ITU-T H.200 (MPEG-4)	• MPEG-1 ~ 1.5 Mb/s • MPEG-2 ~ 4 Mb/s • MPEG-4 ~ 5 kb/s to 10 Mb/s • MPEG7 and 21 are next in evolution of standard	• standards focused primarily on defining a legal format for the bitstream and specifying methods of synchronization and multiplexing • implemented in a variety of ways in codecs such as DivX and Sorenson
DV (Digital Video)	video	• a consortium of 10 companies: Matsushita (Panasonic), Sony, JVC, Philips, Sanyo, Hitachi, Sharp, Thomson Multimedia, Mitsubishi, and Toshiba.	• usually 5:1 • like JPEG and MPEG, uses discrete cosine transform	• this is the consumer video format for camcorder, recorded onto a cassette tape • compression is done as video is recorded • transferred by Firewire to computer

Examples of Popular Codecs				
Codec	**Source Material**	**• Developer**	**Compression Rate and/or Bit Rate***	**Comments**
IMA ADPCM (Interactive Multimedia Association Adaptive Differential Pulse Code Modulation)	audio	• Apple, migrated to Windows	• 4:1 compression of 16 bit samples • built into QuickTime	• some incompatibility between Mac and Windows versions • comparable to the ITU G.726 and G.727 international standard for ADPCM • somewhat outdated
Vorbis	audio	• non-proprietary, unpatented format created by Xiph.org foundation	• bit rate comparable to MP3 and AAC with comparable quality	• a family of compression algorithms, the most successful one being Ogg Vorbis
FLAC (Free Lossless Audio Codec)	audio	• non-proprietary, unpatented format created by Xiph.org foundation	• compression rates range from about 1.4:1 to 3.3:1	• a lossless compression algorithm
Sorenson	video	• Sorenson Media	• wide variety of compression rates available, including very high compression with good quality	• a suite of codecs available in a "pro" version • standard Sorenson codec is part of QuickTime • other codecs in the suite offer MPEG-4 and AVC compression
Indeo	video	• Intel, later acquired by Ligos	• CD-quality compression (352 X 240 resolution NTSC, 1.44 Mb/s bit rate), comparable to MPEG-1 • you can set compression quality	• faster compression than Cinepak • good for images where background is fairly static
Cinepak	video	• Originally developed by SupermacRadius	• CD-quality compression	• originally good for 2x CD-ROM • takes longer to compress than decompress • can be applied to QuickTime and Video for Windows movies
DivX	video	• DivX	• reports 10:1 • compresses video so that it can be downloaded over DSL or cable modem	• uses MPEG-4 standard

Table 1.12 Examples of standardized and patented compression algorithms and codecs

1.6 Standards and Standardization Organizations for Digital Media

In the previous section, we spoke of standards being set for multimedia algorithms and data so that application programs and hardware are able to communicate with each other. In your work in multimedia, you'll encounter references to the organizations and workgroups that set the standards, so you'll want to be familiar with their names and acronyms. The topic of standards is very complicated; whole books are written on the subject. We just want you to be familiar with the main types of standards and the prominent standardization organizations. If you need to know more in your work, you can take it from there.

Standards can be divided into three main types: proprietary, official, and *de facto*. ***Proprietary standards*** are set and patented by commercial companies. Companies lay claim to their way of doing things and naturally hope to make money from their designs and inventions. The competition for better ways of doing things leads to constantly newer and better products, though some would argue that too much insistence on proprietary standards slows development because of lack of coherence and interoperability in the industry. The patents on LZW compression and arithmetic encoding are examples of proprietary standards.

The term ***de facto standard*** is used to describe a method or format that has become the accepted way of doing things in the industry without any official endorsement. For example, TIFF files are considered by many to be the de facto standard for image files. Nearly all image processing programs and operating systems are equipped to handle TIFF files.

Official standards are developed by large industry consortia and/or government agencies. These organizations can exist on either a national or an international level. The main international standardization bodies with which you should be familiar are the ***International Telecommunications Union*** (***ITU***), ***International Organization for Standardization*** (***ISO***, pronounced "eye-so"), and the ***International Electrotechnical Commission*** (***IEC***).

The ITU (formerly called CCITT), is an international organization where governments and the private sector coordinate global telecom networks and services. It has some regulatory power – allocating radio frequency bands, for example. It is divided into three sectors: ITU-T for telecommunications standards, ITU-R for radiocommunications standards, and ITU-D for telecommunications development. The standards most relevant to multimedia are the G and H series. The G series covers speech coding and telephone communication. For example, the G.700 series relates to the coding of analog to digital telephone using methods such as ADPCM. The H series is the realm of real-time digital audio and video communication. For example, H.323 is a standard for voice and video over packet-switched networks (i.e., the internet). H.262 gives the standard for MPEG compression.

ISO, an international body that develops industrial and commercial standards, is comprised of representatives from various national standards organizations. The American National Standards Institute (ANSI) is the agency representing the United States. Working closely with ISO is the IEC, which formulates standards for electrical, electronic, and related technologies. ISO and the IEC have formed a joint committee called ISO/IEC JTC1, and through this committee they develop standards in information technology. For example, their ISO/IEC 11172 standard defines MPEG-1, and ISO/IEC 13818 defines MPEG-2.

1.7 Mathematical Modeling Tools for the Study of Digital Media

Using MATLAB to m,odel digital sound and images

interactive tutorial

Modeling images as waveforms

mathematical modeling worksheet

Modeling sound as a waveform

mathematical modeling worksheet

This book focuses on the mathematics, science, and algorithms behind digital media for the purpose of giving you greater understanding of and control over your work environment. As you make your way through the remaining chapters, you may often find it helpful to graph functions, visualize waveforms, and solve problems with real values so that you can see the results for yourself rather than just talking about abstractions. Mathematical modeling tools are helpful in this regard. A widely-used tool for this purpose is MATLAB. The on-line tutorial associated with Chapter 1 shows you how to use MATLAB to model waveforms. Even if you don't have access to MATLAB, you should go through this tutorials because it contains an introduction to waveforms used to model digital sound and images. (You may find that you're able to do some of the exercises with a graphing calculator.)

With the names, terminology, and basic concepts covered in Chapter 1, you should be ready now to look more deeply into each medium. Pick your focus, or cover it all, and don't forget to do look at the on-line learning supplements, including worksheets, creative exercises, and interactive demos and tutorials.

1.8 Vocabulary

aliasing
amplitude
analog phenomena
analog-to-digital conversion
angular frequency
arithmetic encoding
attentuation
bandpass transmission
bandwidth
    bandwidth as maximum rate of change
    bandwidth (width) of a signal
    bandwidth (width) of a channel
baseband transmission
baseband channel
baud rate
bit, kilobit (kb), megabit (Mb), gigabit (Gb)
bit depth
byte, kilobyte (kB), megabyte (MB), gigabyte (GB)
carrier signal
CD (compact disc)
channel
codecs
color depth
compression
    lossless compression
    lossy compression
compression rate

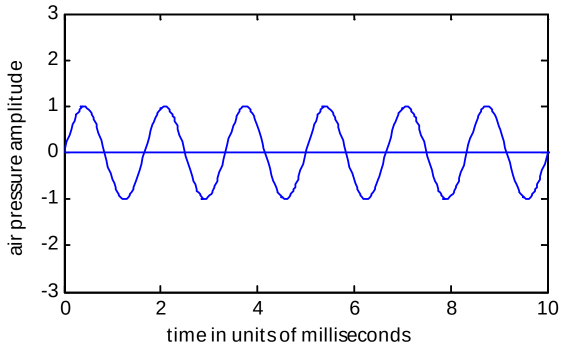
cycle
data rate (bit rate)
decibel (dB)
discrete cosine transform (DCT)
discrete Fourier transform (DFT)
discrete phenomena
DVD (digital versatile disc)
dynamic range
entropy encoding
entropy of an information source
Fourier transform
frequency
frequency band
frequency components
grayscale
Hertz (Hz), kilohertz (kHz), megahertz (MHz)
IEC (International Electrotechnical Commission)
ISO (International Organization for Standardization)
ITU (International Telecommunications Union)
longitudinal wave
modem
modulated data transmission
modulation
- amplitude modulation
- frequency modulation
- phase modulation

multilevel encoding
Nyquist theorem
period
periodic waveform
pixel
quantization
resolution
run-length encoding (RLE)
sample size
sampling
sampling rate
Shannon's entropy equation
Shannon-Fano encoding algorithm
Shannon's theorem
signal-to-noise ratio (SNR)
signal-to-quantization noise ratio (SQNR)
simple sine wave
sinusoidal waveform
standards
- proprietary standards

official standards
de facto standards
transform encoding
undersampling
wavelength

1.9 Exercises

1. Using MATLAB to model digital sound an images, interactive tutorial, online
2. Modeling sound as a waveform, mathematical modeling exercise, online
3. Modeling images as waveforms, mathematical modeling exercise, onlin
4. What is the frequency of the sine wave below, in cycles per second?



What is its angular frequency?
5. What is the equation for a sine wave that has a frequency of 660 Hz?
6. You have a digital image in eight-bit color that has blocky areas of color, lacking the subtle gradations from one color to the next that you would like to see. Is this a matter of aliasing or quantization error? Explain.
7. If you are recording an audio file and you expect that the highest frequency in the file will be 10,000 Hz, what is the minimum sampling rate you can use to ensure that you won't get audio aliasing?
8. How many different colors can be represented with 12 bits?
9. True or false? You can't represent as wide a range of colors with eight bits as you can with 16 bits. Explain your answer.
10. According to the definitions of kilobytes, megabytes, and gigabytes used by computer scientists, 4,276,012,000 bytes is equal to approximately how many kilobytes, how many megabytes, and how many gigabytes?
11. What is the signal-to-noise ratio of a sound file represented with eight bits per sample?
12 Compute the number of bytes needed for one minute of uncompressed video that has 720×576 pixels per frame, 25 frames per second, three bytes per pixel, and CD-quality stereo audio.
13. What is the data rate of a communication medium with a bandwidth of 3 MHz using 4 different signal levels?
14. Do a run-length encoding of the following sequence of grayscale values. Explain your encoding strategy, and compute the compression rate.
240 240 240 240 240 240 240 238 238 238 238 238 230 230 230 230 229 228 228 227
227 227 227 227 227 227 227 227 227 227 227 227 227 227 227 227 227 227 227 227

15. Do a Shannon-Fano encoding of an image file on the basis of the frequency table below. Fill in the table below using Shannon's entropy equation. Then compare the average number of bits per color arising from your encoding with the minimum possible average number of bits derived from Shannon's entropy equation.

color	**frequency**	**optimum number of bits to encode this color**	**relative frequency of the color in the file**	**product of columns 3 and 4**
black	200			
white	175			
yellow	90			
orange	75			
red	70			
purple	35			
blue	20			
green	10			

16. Say that you have an image that has 100 pixels. The first six colors of the image are W K Y B R Y. Do the first six steps of arithmetic encoding based on the frequency table below.

color	**frequency out of total number of pixels in file**		**probability interval assigned to symbol**
black (K)	40/100 =	0.4	0 – 0.4
white (W)	30/100 =	0.3	0.4 – 0.7
yellow (Y)	15/100 =	0.15	0.7 – 0.85
red (R)	10/100 =	0.1	0.85 – 0.95
blue (B)	5/100 =	0.05	0.95 – 1.0

17. Computer technology changes very quickly. At the time of the writing of this book, Table 1.5 and Table 1.6 list data rates for common communication links and storage devices. Do some research to determine if new technology is now available that gives faster data rates, and update the tables accordingly.
18. The arithmetic encoding algorithm described in this chapter operates as if your computer is capable of infinite precision arithmetic, which of course it is not. Do some research to find out how this algorithm is made practical.

1.10 Applications

1. Take inventory of the software available to you for work in digital media, in the following categories:
- digital image processing
- vector graphics
- digital sound processing
- digital video
- multimedia programming

What software was included on your computer when you bought it? What software is available in computer labs to which you have access? What type of multimedia work would you like to do? What software do you need for this work? Compare the features of the products in each category. Do some web research to find out what is available and what each piece of software costs. What can you afford? If there are types of software that are out of your price range, see if you can find scaled-down versions that are less expensive, or freeware or shareware alternatives. If you simply want to experiment with certain application programs, find out if trial versions are available. Often, you can use a trial version for 30 days, although sometimes these versions don't offer all the features of the full program.

2. Look at the specifications of your computer and other hardware to determine its suitability for work in digital media, in the following categories:

- amount of RAM
- amount of hard disk space
- processor speed

Does your computer meet the minimum requirements of the digital media software you'd like to use?

3. As you work through this book, you'll be learning about how to create and edit digital images, sound, and video. The last chapter will introduce you to multimedia authoring languages like Director, Flash, and Java, where you can put together your images and sound. One way to tie all of your learning together is to develop a long-term project as you work through the book. Here are some suggestions:

- a game program (e.g. Tic-Tac-Toe, Othello, Mancala, Connect Four, Master Mind, or a Solitaire)
- an interactive tutorial (e.g., "How to Do Long Division," or "How to Change a Tire," or whatever you want to teach)
- your own interactive résumé and/or portfolio of your work
- an interactive commercial website

Plan a project that will tie together the media that you study in this book. Decide what media you need for your project. Just for the practice, you might want try to incorporate as many different types of media as possible – digital photographs, vector graphic animations, digital audio, MIDI, and digital video. As you go through the chapters on each medium, develop the material you'll need for your project. For example, if you're creating a game program for, say, a checkers game, you could create digital images for the splash screen as you work through Chapters 2 and 3, digital audio and MIDI-generated music as you work through Chapters 4 and 5, and digital video as you work through Chapters 6 and 7. Then in Chapter 8, you'll learn how to weave these together with multimedia authoring environments and languages.

Additional exercises or applications may be found at the book or author's websites.

1.11 References

1.11.1 Print Publications

Chapman, Nigel, and Jenny Chapman. *Digital Multimedia*. 2nd ed. West Sussex, England: John Wiley & Sons, Ltd, 2005.

Comer, Douglas E., and Ralph E. Droms. *Computer Networks and Internets*. 4th ed. Upper Saddle River, NJ: Prentice Hall, 2003.

Couch, II, Leon W. *Digital and Analog Communication Systems*. 7th ed. Upper Saddle River, NJ: Prentice Hall, 2006.

Fano, R. *Transmission of Information*. Cambridge, MA: MIT Press, 1961.

Gibson, Jerry D., et al. *Digital Compression for Multimedia: Principles & Standards*. San Francisco: Morgan Kaufmann Publishers, 2006.

Halsall, Fred. *Multimedia Communications: Applications, Networks, Protocols, and Standards*. Harlow, England: Pearson Education/Addison Wesley, 2000.

Hanselman, Duane, and Bruce Littlefield. *Mastering MATLAB 7*. Upper Saddle River, NJ: Pearson/Prentice-Hall, 2005.

Howard, P. G., and J. S. Vitter. *Practical Implementation of Arithmetic Coding. Image and Text Compression*. Storer, Boston: Kluwer Academic Publishers, 1992.

Li, Ze-Nian, and Mark S. Drew. *Fundamentals of Multimedia*. Upper Saddle River, NJ: Pearson/Prentice Hall, 2004.

Pasco, R. Source Coding Algorithms for Fast Data Compression. Ph.D. dissertation, Stanford University, 1976.

Rissanen, J. Generalized Kraft Inequality and Arithmetic Coding. *IBM Journal of Research and Development* 20 (1976): 198-203.

Rissanen, J. Universal Coding, Information, Prediction, and Estimation. *IEEE Trans. on Information Theory* IT-30 (July 1984): 629-636.

Rissanen, J., and G. G. Langdon. "Arithmetic Coding." *IBM Journal of Research and Development*, 23(2): 149-162, 1979.

Shannon, C. E. A Mathematical Theory of Communication. *Bell System Technical. Journal* 27 (1948): 379-423, 623-656.

Shannon, C. E. *The Mathematical Theory of Communication*. Champaign, IL: University of Illinois Press, 1949.

Witten, I. H., R. M. Neal, and J. G. Cleary. *Arithmetic Coding for Data Compression. Communications of the ACM* 30 (6) (June 1987): 520-540.

1.11.2 Websites

ISO (International Organization for Standardization). http://www.iso.org
ITU (International Telecommunication Union). http://www.itu.int/home/
IEC (International Electrotechnical Commission). http://www.iec.ch/