COMPUTER SCIENCE MODULE

Chapter 4: Digital Audio[1]

4.1. Sampling and Quantization

Primer Chapter 4 explains fundamentals of sampling and quantization for digital audio. We now will look more closely at the consequences of choosing a sampling rate or bit depth that is too low for the type of sounds being recorded. The Nyquist Theorem tells us that the sampling rate for digital audio recording must be greater than twice the frequency of the highest frequency component in the sound being recorded. (The highest frequency component in a sound is called its *Nyquist frequency*, and twice the frequency of the highest frequency component is called the *Nyquist rate*.) If the sampling rate is below the Nyquist rate, aliasing can occur. That is, when the digitized sound is played, a frequency from the original sound may be translated to a different frequency, so the digitized sound doesn't sound exactly like the original. If the sound wave to be sampled has a frequency of f and f_s is the frequency at which the wave is sampled, no aliasing occurs for $f < ½ * f_s$. However, if $½ f_s < f < f_s$ then the frequency of the aliased wave will be $f_s - f$. For example, a 6000 Hz wave sampled at 8000 Hz (i.e., 8000 samples per second) will be aliased to a frequency of 2000 Hz. A 6000 Hz wave sampled at 20,000 Hz will not be aliased.

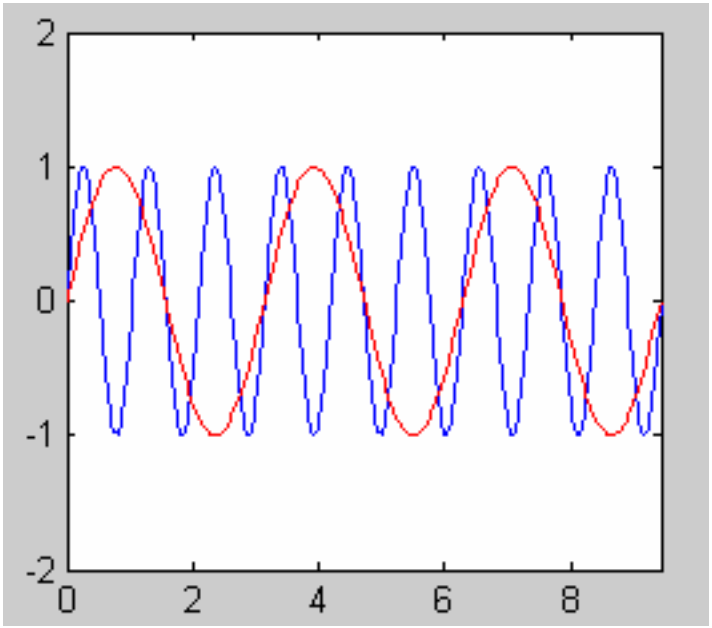

Figure 4.1. A 6000 Hz wave sampled at 8000 Hz, and thus aliased to 2000 Hz

We can do calculations such as these on simple waveforms, but you should keep in mind that the aliasing phenomenon occurs for more complex waveforms as well. Any complex waveform can be written as a sum of sine and cosine waves. (See Section 4.3.)

---

[1] This material is based on work supported by the National Science Foundation under Grant No. DUE-0127280. This chapter was written by Dr. Jennifer Burg (burg@wfu.edu).

Each of these simple waveforms is a component of the complex waveform. If the sampling rate for the complex waveform is less than the Nyquist rate, then components that are greater than ½ the sampling rate may be aliased after digitization.

An insufficient sampling rate can result in digitized sound that is not faithful to the original in that some frequencies may be aliased. An insufficient bit depth, on the other hand, can create distortion or quantization noise. Let's examine the mathematics of quantization error a little more closely to understand how dithering can work to remedy it. Figure 4.2 shows a continuous waveform (in blue) that has been quantized to 16 quantization levels (in green). The quantization error wave is shown in red. Note that the original wave plus the error wave equals the quantized wave. The error wave constitutes another sound component that can be heard as distortion.

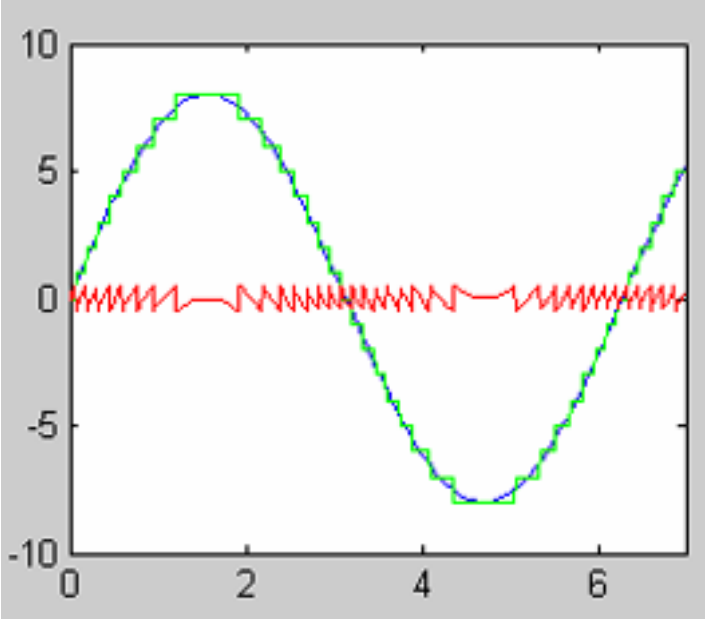

Figure 4.2. Original wave, quantization values, and quantization error wave.

Notice that the error waveform – the distortion – is periodic, i.e. it repeats in a regular pattern, and its period is related to the period of the original wave. This is what distinguishes *distortion* from *noise*. Noise is random, while distortion sounds "meaningful" even though it is not. For this reason, distortion can be more distracting in an audio signal than noise. Distortion artifacts modulate in tandem with the original wave and thus, to human hearing, the distortion seems to be a meaningful part of the sound. It is easier for the human brain to tune out noise because it seems to have no logical relationship to the dominant patterns of the sound. (You should note that not all sources make a distinction between distortion and noise. The term *noise* is often used in place of distortion. You will see an example of this below in the discussion of signal-to-quantization-noise ratio.)

Another way to understand quantization error is to observe the relatively greater effect that it has on low vs. high amplitude signals. Consider that for an n-bit digital audio file, the amplitude of each sample is normalized to a value between 0 and 2^n-1. Then the value is rounded to the nearest integer. This yields a maximum error of 0.5. It

is easy to see that the percentage error is larger for small amplitude values than for large ones. For example, if a sample value of 0.5 is rounded down to 0, then there is a 100% error. If a sample value of 64.5 is rounded down to 64, the error is only $\frac{0.5}{64} * 100$, that is, a 0.8% error. The smaller the sample value, the larger the relative error can be, depending on how much it must be rounded.

So what is the remedy for this problem of quantization error? It may be counterintuitive to understand that artificially introducing noise can actually have a helpful effect on sound that is quantized in an insufficient number of bits. It is possible to add noise to a quantized audio wave in a way that reduces the effects of distortion. Imagine that we add between -1 and 1 unit to each sample. (A unit is the size of a quantum.) The exact amount to be added to the sample is determined at random by a triangular probability function. The function, shown in Figure 4.3, indicates that there is the greatest probability that 0 will be added to a sample. As x goes from 0 to 1 (and symmetrically as x goes from 0 to -1), the probability that x is the value to be added to a sample decreases.

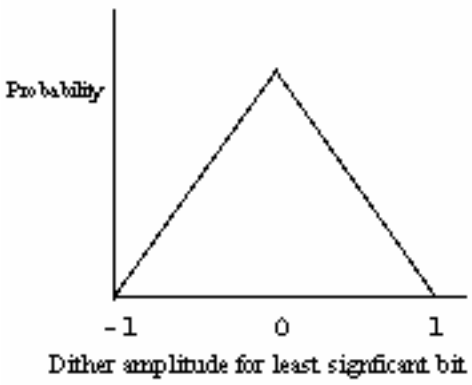

Figure 4.3. Triangular Probability Function

Adding this random noise to the original wave eliminates the sharp stairstep effect in the quantized symbol. Instead, the quantized wave jumps back and forth between two neighboring quantum values. Figure 4.4 shows the triangular dithering function, which produces random values between -1 and 1. The dithered wave in blue represents the original quantized wave to which the dither function has been added. The resulting dithered wave generally sounds more like the original waveform than the undithered wave does.

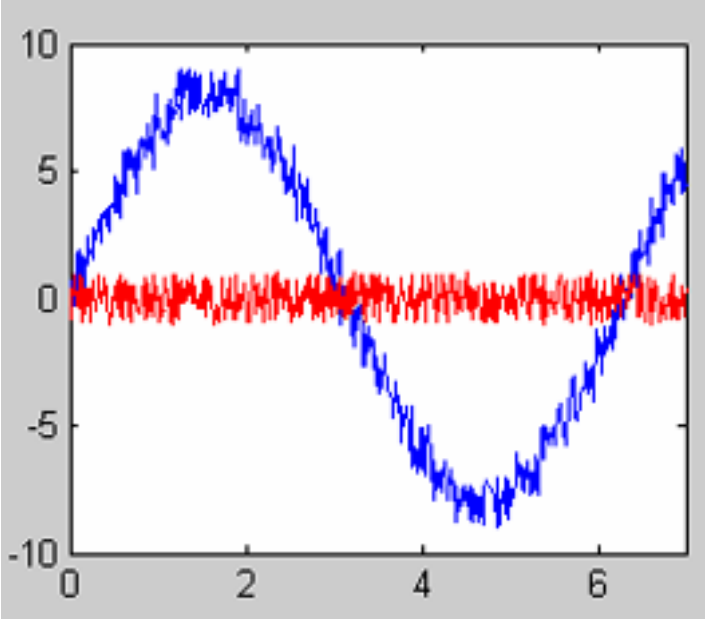

Figure 4.4. Dithered Quantized Wave (blue) with Triangular Dither Function (red)

Interactive Demo
Link to interactive demo on Audio Dithering

4.2 Amplitude and Dynamic Range

Amplitude can be measured with a variety of units, including voltages, air pressure, or decibels. The most common unit of sound amplitude measurement is decibels, but to understand decibels it helps to consider first how amplitude can be measured in terms of air pressure.

Section 4.1 of Primer Chapter 4 describes how a vibrating object pushes molecules closer together, creating changes in air pressure, so it makes sense to measure the loudness of a sound in terms of these air pressure changes. Atmospheric pressure is customarily measured in units of newtons/meters^2 (abbreviated N/m^2). The average atmospheric pressure is approximately 10^5 N/m^2. For sound waves, *air pressure amplitude* is defined as the average deviation from normal background atmospheric air pressure. For example, the *threshold of human hearing* (for a 1000 Hz sound wave) varies from the normal background atmospheric air pressure by $2*10^{-5}$ N/m^2, so this is its pressure amplitude.

Measuring sound in terms of pressure amplitude is intuitively easy to understand, but in practice decibels are a more common, and in many ways a more convenient, way to measure sound amplitude. While you may be familiar with the term *decibel* used in relation to sound, it's important to realize that decibels can be used to measure many things in physics, optics, electronics, and signal processing. Furthermore, a decibel is not an absolute unit of measurement. A decibel is always based upon some agreed upon reference point, and the reference point varies according to the phenomenon being measured. In networks, for example, decibels can be used to measure the attenuation of a signal across the transmission medium, so in this case the reference point is the strength of the original signal, and decibels describe how much of the signal is lost. In measuring

the strength of a sound, the reference point for decibels is the pressure amplitude for the threshold of hearing. A decibel in the context of sound pressure level is defined as

$$dB\_SPL = 20 * \log_{10}\left(\frac{V}{V_0}\right) \qquad \textbf{Equation 1}$$

where V is the pressure amplitude of the sound being measured and V_0 is the sound pressure level of the threshold of hearing. Often, this is abbreviated simply as *dB*, but since decibels are always relative, it is helpful to indicate what the reference point is if the context is not clear. *dB_SPL* is meant to indicate that that V_0, the threshold of hearing, is the point of comparison for the sound being measured.

Given a pressure amplitude value, you can compute the amplitude of the sound in decibels with the equation above. For example, what would be the amplitude of the audio threshold of pain, given as 30 N/m^2?

$$dB\_SPL = 20 * \log_{10}\left(\frac{30N/m^2}{.00002N/m^2}\right) = 20 * \log_{10}(1500000) = 20 * 6.17 \approx 123$$

So 30 N/m^2 is approximately equal to 123 decibels. You can also compute the pressure amplitude given the decibels. For example, what would be the pressure amplitude of normal conversation, given as 60 dB?

$$60 = 20 * \log_{10}\left(\frac{x\,N/m^2}{.0002\,N/m^2}\right)$$

$$60 = 20 * \log_{10}(5000 * x)$$

$$3 = \log_{10}(5000 * x)$$

$$10^3 = 5000 * x$$

$$\frac{10^3}{5000} = x$$

$$x = 0.2$$

So 60 dB is approximately equal to 0.2 N/m^2.

Decibels can also be used to describe sound intensity (as opposed to sound pressure amplitude). Decibels-sound-intensity-level (dB_SIL) are defined as follows:

$$dB\_SIL = 10 * \left(\log_{10}\frac{I}{I_0}\right) \qquad \textbf{Equation 2}$$

I_0 is the intensity of sound at the threshold of hearing, given as 10^{-12} W/m^2. It is sometimes more convenient to work with intensity decibels rather than pressure amplitude decibels, but essentially the two give the same information. The relationship between the two lies in the relationship between pressure (which can be thought of as *potential* or volts) and intensity (which can be thought of as *power* or watts). In this respect, I is proportional to the square of V.

Decibels are also a useful way to quantify the proportion of meaningful sound in a signal relative to the noise, called the *signal-to-noise-ratio* (*SNR*). Quantization error manifests itself as noise that is a function of the bit depth of a digital audio file. The signal-to-quantization-noise-ratio (*SQNR*) is thus defined in terms of the bit depth n of the digital audio file.

$$SQNR = 20 * \log_{10}\left(\frac{\max(sample\,value)}{\max(quantization\,error)}\right) =$$

$$20 * \log_{10}\left(\frac{2^{n-1}-1}{\frac{1}{2}}\right) \approx 20 * \log_{10}\left(\frac{2^{n-1}}{\frac{1}{2}}\right) =$$

$$20 * \log_{10}\left(2^{n}\right) =$$

$$20 * n * \log_{10}(2) \approx 6.02 * n \ \ decibels$$

(Note that the maximum sample value for an n-bit digital audio file is $2^{n-1} - 1$ rather than 2^{n-1} because the 2^n different sample values are centered on 0 and can be either positive or negative.)

WORKSHEET
Link to Worksheet on Converting Air Pressure Amplitude to Decibels

Dynamic range can also be measured in decibels. The term *dynamic range* has two definitions: 1) the level difference in dB between the loudest and the softest parts of an audio piece and 2) for an audio file quantized in n bits, the level difference in dB between the maximum value and the minimum quantization value that can be represented in n bits. This second definition makes *SQNR* and *dynamic range* synonymous terms when used in the context of an audio file quantized in n bits. As a rule of thumb you can estimate that an n-bit digital audio file has a dynamic range (or, equivalently, a signal-to-noise-ratio) of 6*n dB. For example, 16-bit digital audio has a dynamic range of about 96 dB, while 8-bit digital audio file has a range of about 48 dB. Be careful not to interpret this to mean that a 16 bit file allows louder amplitudes than an 8 bit file. Remember that dB is a relative term. We are not talking about dB_SPL here. Instead, we are talking about decibels relative to the smallest non-zero value that can be represented with any number of bits; that is, 1.

Rather than display amplitudes in units of dB_SPL, audio editing programs generally use decibels-full-scale, abbreviated dBFS, where 0 dBFS is taken to be the maximum amplitude possible for the given system. All actual amplitudes in an audio file are therefore negative values relative to this maximum. When you look at a waveform with amplitude given in dBFS, the horizontal center of the waveform is $-\infty$ dBFS, and above and below this axis the values progress to the maximum of 0 dBFS. Given that an 8-bit digital audio file has a dynamic range of 48 dB, any sample that is more than 48 dB from the maximum possible amplitude will be below the noise level, which means that it cannot be represented.

4.3. Audio Transforms and Filters

The Fourier transform is a mathematical procedure for moving digital audio samples from the time domain to the frequency domain. In Primer Chapter 4, we illustrated that simple waveforms can be summed into more complex forms. Now we'll look at the reverse – the decomposition of a complex wave into pure sinusoidals. Let's look at some of the mathematical details.

One detail not mentioned earlier is the fact that sound waves, being time-dependent, have a phase component that must be considered. Two pure waves of the

same frequency may not necessarily be in the same phase. In the extreme case, if they are completely out-of-phase, then adding them together yields 0. To account for phase information, we need to have both cosine and sine components in the decomposition of a complex waveform. More precisely, let x(t) be a sound wave expressed as a function over the time domain t. Then we can write this function as a sum of cosine and sine waves, which are themselves functions over time:

$$x(t) = \sum_{k=0}^{\infty} a_k \cos(2\pi k f_0(t)) + b_{k,} \sin(2\pi k f_0(t)) \qquad \textbf{Equation 3}$$

where a_k is the amplitude for the k^{th} cosine frequency component, b_k is the amplitude for the k^{th} sine frequency component, and f_0 is the base frequency. This equation – the basis for the Fourier transform that moves sound data from the time domain to the frequency domain – can be taken as a statement of fact, but it does not give us an algorithm for decomposing a complex sound wave into its component parts in the frequency domain. It says that the component waves exist, but it does not tell us how to get the coefficients a_k and b_k for each component.

Giving a full derivation of the Fourier transform is beyond the scope of this book, but we will attempt to give enough of the mathematics here to help you understand what the transform means and how it is applied. The equation for computing the Fourier transform of a sound wave is as follows:

$$F_p = \sum_{k=0}^{N-1} a_k (\cos(2\pi k \tfrac{p}{N}) + i \sin(2\pi k \tfrac{p}{N})) \text{ for } 0 \le p \le N \qquad \textbf{Equation 4}$$

Let's examine the equation more closely. First note that the equation is defined for a block of N samples and is executed N times to yield amplitudes corresponding to N frequency components. a_k is given as the k^{th} sample value in the input. Also note that $i = \sqrt{-1}$, giving the equation both a real term (the cosine) and an imaginary term (the sine). The real component corresponds to the cosine waves summed to yield the composite audio waveform. The imaginary component tells what phase the wave is in. The output of the equation is a complex number, with real and imaginary components. (Recall that a complex number is a number $a + b * i$. a is the real component, b is the coefficient of imaginary component, and $i = \sqrt{-1}$.)

If you have encountered Fourier transforms in other contexts, you may be familiar with another form of this equation. Equation 4 can be rewritten based on Euler's Identity, which states

$$e^{i\theta} = \cos(\theta) + i \sin(\theta) \qquad \textbf{Equation 5}$$

where e is the e is the base of the natural logarithm. Rewriting the equation using Euler's Identity for the substitution, we get

$$F_p = \sum_{k=0}^{N-1} a_k e^{i 2\pi k \frac{p}{N}} \qquad \textbf{Equation 6}$$

This equation is the basis for the *Discrete Fourier Transform* (*DFT*). It should not be too difficult for you to picture how you would program this equation. Given as input is an array k of sample values, $[a_0 ... a_{N-1}]$. e is a constant. The equation is executed N times, to yield as output N values corresponding to the amplitudes of the component frequencies. [***To be continued.***]

The Fourier transform is widely used in image and audio processing, and it is implemented in a wide variety of mathematical programs and multimedia editing tools. Fast versions of the algorithm have been devised, most notably the *Fast Fourier Transform* (*FFT*), which reduces the complexity from $O(N^2)$ to O(N*logN). This is a big savings in computational time in light of the fact that CD quality stereo, for example, has 44,100 samples in every second. However, the method used in the FFT for eliminating redundant computation relies on the computation being done on chunks of samples that are powers of 2, each chunk being called a *window*. If, for a particular waveform, the window of the FFT is not aligned with an integer number of cycles, then the waveform for consecutive windows will be pieced back together incorrectly, with discontinuities where the segments are joined. These discontinuities are called *spectral leakage*, and they cause sound distortions in the form of clicks and pops. FFT implementations avoid this problem by applying windowing functions that help to splice the segments together in a way that smooths over the spectral leakage. Common windowing functions are triangular, Hanning, Hamming, Blackman, Gaussian, and Blackman-Harris. [***To be continued.***[2]]

4.4. Digital Audio Compression

Both lossless and lossy compression methods can be used on digital audio files. Run-length or Huffman encoding are applicable and have the advantage of being lossless, but in general neither offers sufficient compression to reduce audio files to a manageable size. Three of the most common forms of digital audio compression are differential encoding, non-linear encoding, and perceptual compression.

Adaptive differential pulse code modulation (*ADPCM*) is a statistically-based method that is well-suited for human speech. Because the amplitudes of sounds in human speech do not change dramatically from one moment to the next (i.e., human speech does not require a wide dynamic range), it is possible to encode them in fewer bits by recording only the difference between one sample and the succeeding one. In this way, a 16 bit sample is reduced to 4 bits, for a 4:1 compression rate. The algorithm can be fine-tuned so that the quantization level varies in accordance with the dynamically-changing characteristics of the speech pattern being encoded, yielding even better compression rates. A variety of ADCPM compression algorithms have been devised, including ones by the International Multimedia Association (IMA) and Intel/DVI.

Non-linear encoding, or *companding,* arose from the need for compression of telephone signals across low bandwidth lines. (The word *companding* is derived from the fact that this encoding scheme requires compression and then expansion.) The non-linear encoding method begins with the observation that the human auditory system is perceptually non-uniform. That is, humans can perceive small differences between quiet sounds, but as sounds get louder our ability to perceive a difference in their amplitude diminishes. Furthermore, in normal human speech there are far more low amplitude than high amplitude signals. In light of these facts, it makes sense to use *more* quantization levels for low amplitude signals and *fewer* quantization levels for high amplitudes. The graph in Figure 4.5 shows this relationship as a logarithmic function. Assume that the original sample values are normalized to values between −1 and 1. Figure 4.6 pictures

[2] CSC361/661 students. Just ignore parts marked "To be continued." You will not be responsible for these parts.

how you would compress the sample values to fewer bits using the logarithmic function. Notice that input amplitudes at lower levels are mapped to a wider range of amplitudes than are input values at higher levels. Compare, for example, the range of values to which 0.1 to 0.2 are mapped, versus the range of values to which 0.8 to 0.9 are mapped. This means that you can divide low amplitude signals in a finer-grained manner, for greater accuracy at the amplitudes where quantization noise would have the greatest negative impact.

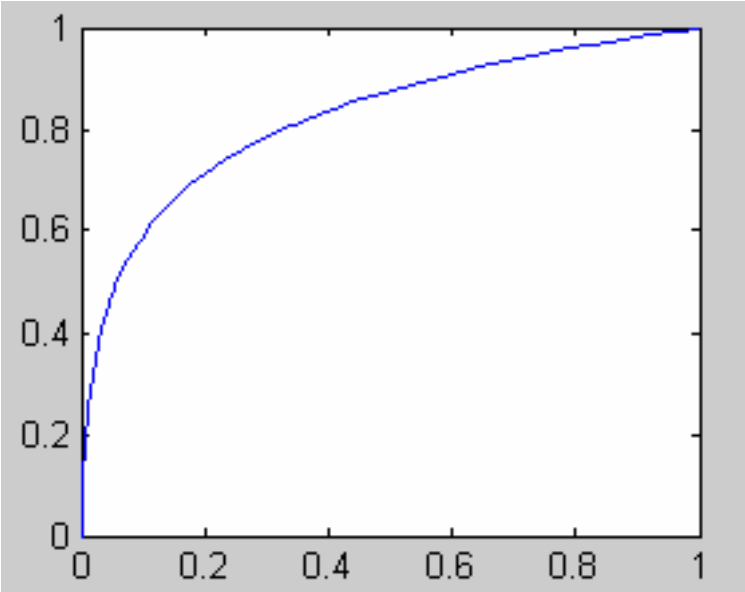

Figure 4.5. Logarithmic function for non-linear audio encoding

Non-linear companding schemes are widely used and have been standardized under the CCITT Recommendations for telecommunications (Comité Consulatif Internationale de Télégraphique et Téléphonique). In the United States and Japan, *μ-law* (also called *mu-law*) *encoding* is the standard for compressing telephone transmissions, using a sampling rate of 8 KHz and a bit depth of only 8 bits, but achieving about 12 bits of dynamic range through the design of the compression algorithm. The equivalent standard for the rest of the world is called A-law encoding. Let's look more closely at μ-law to understand non-linear companding in general.

The equation for encoding under μ-law compression is given as

$$y = \frac{\text{sgn}(x) * \log_2(1 + \mu|x|)}{\log_2(1 + \mu)}$$

where $-1 \le x \le 1$, sgn() is the sign function, and μ is generally taken to be 255.

This equation is defined for input values between -1 and 1, but the purpose is to apply it to 16-bit samples in order to compress them to 8-bits. Note that n-bit quantized sample values range between -2^{n-1} and $2^{n-1} - 1$. For 16 bits, this is –32768 to 32767, while for 8 bits this is –128 to 127. Thus, to scale input values to a maximum magnitude of 32768 and output to a maximum magnitude of 128, we have:

$$y = \mathrm{sgn}(x) * 128 * \frac{\log_2(1+\mu\left|\frac{x}{32768}\right|)}{\log_2(1+\mu)} =$$

$$\mathrm{sgn}(x) * 128 * \frac{\log_2(1+255\left|\frac{x}{32768}\right|)}{8} \quad for\ \mu = 255$$

After compression using the above equation, samples can be saved and transmitted in 8-bits. At the user-end, they are decompressed to 16 bits with the inverse function:

$$y = \mathrm{sgn}(x) * \frac{1}{\mu}\left[(\mu+1)^{x} - 1\right]$$

Low-amplitude samples are compressed and decompressed with less resulting error.

WORKSHEET
Link to Worksheet on Non-linear Companding

An algorithmic approximation for the μ-law encoding function is given below. (This algorithm does not yield exactly the same encoded/decoded values as the equation above, but the logarithmic effect of encoding on dynamic range is the equivalent.)

Step 1. Convert the 16-bit sample from sign-and-magnitude format to two's complement, saving the value of the sign bit.
Step 2. Clip the magnitude to 32,635. (The purpose of the clipping is to avoid overflow when the bias is added in the next step. Note that the values for 16 bit samples range from $-2^{n-1} - 1$ to 2^{n-1}. The largest magnitude is 32,767.)
Step 3. Add a bias of 132 to the magnitude. (The purpose of adding the bias is to ensure that a 1-bit will appear somewhere in the exponent region, as determined in Step 4 below.)
Step 4. Let the exponent region be defined as the 8 bits to the right of the sign bit. Let p be the position of the leftmost 1-bit in the exponent region, counting positions from 0 to 7 from right to left.
Step 5. Let the mantissa region be defined as the 4 bits to the right of position p.
Step 6. Encode 8 bits by making the leftmost bit the sign bit (saved in Step 1), the next 3 bits the binary encoding of p, and the last 4 bits the mantissa region identified in Step 5.
Step 7. Take the one's complement of the 8 bits encoded in Step 6.

Algorithm 4.1. μ-Law encoding

Let's try this on an example. The original sample is 0010111010011011.

Step 1. Two's complement is 1101000101100101. Sign bit is 0.
Step 2. No clipping necessary.
Step 3. 1101000101100101 + 10000100 = 1101000111101001 (base 2)
Step 4. Exponent region underlined and in boldface.
1**10100011**1101001
Leftmost 1-bit in exponent region is in position 7, which is 111 in base 2.
Step 5. Mantissa region is underlined and in boldface.

11**0100**0111101001

Step 6. Encode 8 bits as
01110100

Step 7. One's complement is
10001011

Thus, 10001011 is the 8-bit μ-law encoding of 0010111010011011.

This procedure effectively encodes the samples in the logarithmic manner depicted in Figure 4.5. Neighboring quantization values at high amplitudes correspond to larger differences in amplitudes, as compared to neighboring values at low amplitudes. In effect, the encoding scheme uses more bits to represent low amplitude signals and fewer bits to represent high amplitude signals, thereby lowering the signal-to-noise ratio at lower amplitudes, where the noise is otherwise most pronounced.

Decoding reverses this process. The steps are given in Algorithm 4.2.

Step 1. Take the ones' complement of the 8-bit μ-law encoded sample.
Step 2. Save the sign bit (the most significant bit of the 8 bits given) as s.
Step 3. Save the exponent region (the 3 bits to the right of the sign bit) as a 3-bit field called e.
Step 4. Save the mantissa region (the remaining 4 bits) as a 4-bit field called m.
Step 5.
a = 12 – 8 + (e – 1)
b = (10000100 left shifted by e bits) – 10000100
sample = (m left shifted by a bits) + b
Step 6. Take the two's complement of the sample
Two's complement 0101001001111100
Step 7. Affix the sign bit s as the leftmost bit.

Algorithm 4.2. μ-Law decoding

For our example, the steps work as follows:

Step 1. One's complement of 10001011 is 01110100.
Step 2. Sign bit is 0.
Step 3. Exponent region is $111_2 = 7_{10}$.
Step 4. Mantissa region is 0100.
Step 5. a = 12 – 8 + 6 = 10
b = (10000100 left shifted by 7 bits) – 10000100 = 100001001111100
sample = (0100 left shifted by 10 bits) + b =
01000000000000 + 100001001111100 = 101001001111100
Step 6. Two's complement of 101001001111100 is 010110110000100
Step 7. With sign bit affixed, sample value returned is 0010110110000100

Note that the decoded sample ($0010110110000100_2 = 11{,}652_{10}$) is not exactly the same as the original sample value ($0010111010011011_2 = 11{,}931_{10}$). There is a 2% error in this case, but recall that is a lossy compression method. The benefit of non-linear encoding can be understood more clearly if we consider an alternative method of reducing from 16 bits to 8 bits. What if we simply truncate a 16 bit value by deleting the 8 least significant bits? Then, in our example, 0010111010011011 would be truncated to 001011100000000, for an 11% error. This seems even worse, but consider the difference

in error produced by μ-law encoding versus truncation for smaller amplitudes. Using μ-law encoding, a sample value of 193_{10} is decoded as 196_{10}, with a error of 1.5%. Using truncation to achieve 8 bits, 193_{10} (0000000011000001_2) is encoded as 0, for a 100% error. μ-law encoding distributes the error in a way that matches human hearing better, with less noise in low amplitude signals.

Other compression methods use the characteristics of the human auditory system to an even greater extent. Perceptual audio compression methods are based upon an analysis of the how the ear and brain perceive sound. Just as digital image compression takes advantages of visual elements to which the human eye is not very sensitive, perceptual audio compression exploits audio elements that the human ear cannot hear very well. In the case of digital images, human vision is not very good at perceiving high frequency changes in color, so some of this information can be filtered out. The same method would not work with digital audio, however, since filtering out high frequency sound waves would effectively filter out high pitched sounds that are a meaningful component of the audio signal being captured. Then what *can* be filtered out of digital audio? What kinds of sounds do humans not hear very well, and in what conditions?

Perceptual audio compression begins with a psychoacoustical model of hearing – that is, a mathematical relationship between sound waves and the acuity with which they are perceived by the human ear and brain. We have already discussed the *threshold of hearing* in the context of decibels, but a detail that we omitted was that the threshold varies non-linearly with frequency. This relationship is pictured as the blue line in Figure 4.6. Humans hear mid-frequency sounds more easily than high or low frequencies, and we are most sensitive to sounds in the range of human speech – approximately 1000 to 2000 Hz. Furthermore, the threshold of hearing (also referred to as the *threshold of quiet*) for a given frequency changes in the presence of another frequency. For example, a 500 Hz frequency component is easy to hear, but when a 400 Hz pure tone is played simultaneously and at a sufficient amplitude, it may mask the 500 Hz signal. That is, the threshold of hearing for 500 Hz is higher in the presence of the 400 Hz signal. A frequency that raises the threshold of hearing of another is called a *masking frequency* or *masking tone*. As shown in Figure 4.6, the thresholds of all the frequencies in the neighborhood of a masking frequency are raised, indicated by the *masking curve* in the figure. All frequencies that appear at amplitudes beneath the masking curve will be inaudible. In theory, we could create a graph like the one in Figure 4.6 for every masking frequency in the domain. The important point to note is that frequencies that are masked out in an audio signal cannot be heard, so they do not need to be stored.

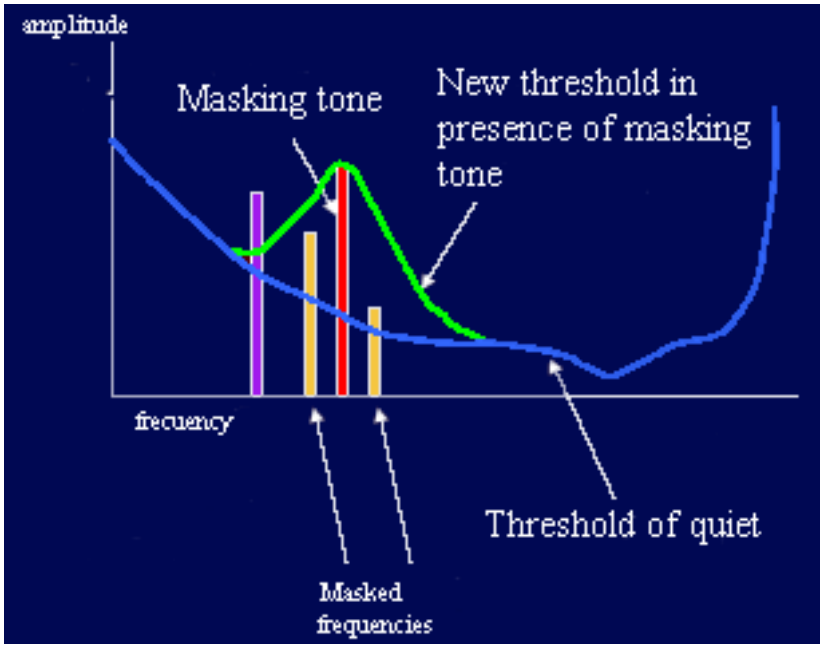

Figure 4.6. Threshold of quiet and a masking tone

There is another, even more profitable, way in which masking can be used for audio compression. A masking frequency may have the beneficial effect of hiding noise. It may even be possible to reduce the bit depth and raise the noise level if it is the case that the noise introduced by the increased quantization error still remains under the masking curve. A sketch of how this is done is as follows: Spectral analysis by means of a *filter bank* divides the audio file into bands of frequencies, usually about 32 of these bands. The average amplitude for each band of frequencies is calculated. Then a masking curve for each band is calculated based upon the relative amplitudes of the frequencies in that band. Analysis of the masking curve reveals the information that can be discarded or compressed. First, some amplitudes of some of the frequencies in the band may fall entirely below the masking curve, so these can be discarded. Secondly, the lowest possible bit-depth can be determined for the frequencies in the band such that the quantization noise for this bit depth is under the masking curve. Further refinements can be made to this scheme. For example, temporal masking can be applied, based on the phenomenon of one signal blocking out another that occurs just before or just after it. Following the perceptual filtering step, the remaining information is encoded using lossless methods such as Huffman and run-length encoding.

MPEG audio compression uses the method of perceptual encoding described above. MPEG, an acronym for the Motion Picture Experts Group, represents a family of standardized compression algorithms for both digital audio and video developed in three phases: MPEG1, MPEG2, and MPEG3. The original MPEG1 achieved CD quality audio and a data rate of 1.5 Mb/sec to transmit both audio and video information. The second phase, MPEG2, added compression algorithms for standard television transmission (as opposed to HDTV), DVD, variable bit rate, and multichannel theatre-style sound systems like surround-sound. MPEG3 was introduced to handle the challenges of digital HDTV.

To add to the complexity of the family of algorithms, MPEG1 is divided into Audio Layers I, II, III. The compression algorithms increase in complexity from Layer I to Layer III, requiring more computation for encoding and decoding, but with a payoff of higher playback quality in proportion to the bit depth. The usual compression rate is approximately 4:1, 8:1, and 10:1 with corresponding data rates of 384 Kb/sec, 256 Kbit/sec, and 128 Kb/sec for Layers I, II, and III, respectively (for two-channel stereo). The well-known MP-3 audio file format is actually MPEG1 Audio Layer III. Its popularity arises from the fact that it can achieve a compression rate of over 10:1 while retaining CD-quality sound.

4.4 Midi Sound Files

[***Coming soon***]

Lab Worksheet
Link to Lab Worksheet on Digital Audio (Adobe Audition 1.0)