Introduction to Computer Music

Week 13 Audio Data Compression

Version 1, 13 Nov 2018

Roger B. Dannenberg

Topics Discussed: **Compression Techniques, Coding Redundancy, Intersample Redundancy, Psycho-Perceptual Redundancy, MP3, LPC, Physical Models/Speech Synthesis, Music Notation**

1 Introduction to General Compression Techniques

High-quality audio takes a lot of space—about 10M bytes/minute for CD-quality stereo. Now that disk capacity is measured in terabytes, storing uncompressed audio is feasible even for personal libraries, but compression is still useful for smaller devices, downloading audio over the Internet, or sometimes just for convenience. What can we do to reduce the data explosion?

Data compression is simply storing the same information in a shorter string of symbols (bits). The goal is to store the most information in the smallest amount of space, without compromising the quality of the signal (or at least, compromising it as little as possible). Compression techniques and research are not limited to digital sound—–data compression plays an essential part in the storage and transmission of all types of digital information, from word-processing documents to digital photographs to full-screen, full-motion videos. As the amount of information in a medium increases, so does the importance of data compression.

What is compression exactly, and how does it work? There is no one thing that is "data compression." Instead, there are many different approaches, each addressing a different aspect of the problem. We'll take a look at a few ways to compress digital audio information. What's important about these different ways of compressing data is that they tend to illustrate some basic ideas in the representation of information, particularly sound, in the digital world.

There are three main principles or approaches to data compression you should know:

- *Coding Redundancy* eliminates extra bits used to represent symbols. For example, computer code generally uses 8-bit characters, but there are less than 100 different characters. We could use 7 bits per character for an immediate savings of 12.5%, and a variable-length encoding would be even more efficient.

- *Intersample Redudancy* looks at how sequences of symbols can be represented more compactly. For example, a fax machine could send a black-and-white image as a sequence of bits for black and white, e.g. 1111111000011111100110100000..., but since there are typically large black and white regions, we can encode this as the number of 1's followed by the number of 0's followed by the number of 1's, etc. The same bits could be represented as 7 4 6 2 2 1 1 5 ..., which is typically much more compact.

- *Psycho-Perceptual Redundancy* considers the waste of transmitting information that will never be perceived. A 16-bit audio recording contains a lot of information and detail we cannot hear. Yes, in the worst case, you really need 16 bits, and yes, in the worst case you really need a frequency response up to 20 kHz, but a lot of the audio information represented in the samples is inaudible, and that fact can be used to send less data.

2 Coding Redundancy

Let's look at how these principles are used in audio data compression. Starting with *coding redundancy*, we can think of each sample as a "symbol" that must be represented in bits. In "uncompressed" audio, each sample is encoded as a number on a linear scale. This is the so-called *PCM*, or *pulse-code modulation* representation[1].

Another representation of samples uses a more-or-less logarithmic encoding called μ-law. This representation is not truly logarithmic and instead uses a floating-point representation with 1 sign bit, 3 exponent bits, and 4 mantissa bits. The encoding spans a 13-bit dynamic range where the quantization levels are smaller at low amplitudes. A less common but similar encoding is A-law, which has a 12-bit dynamic range.

Figure 1 shows a schematic comparison of (linear) PCM to μ-law encoding. The tick-marks on the left (PCM) and right (μ-law) scales show actual values to which the signal will be quantized. You can see that at small signal amplitudes, μ-law has smaller quantization error because the tick marks are closer together. At higher amplitudes, μ-law has larger quantization error. At least in perceptual terms, what matters most is the signal-to-noise *ratio*. In the encoding process, noise is introduced by quantization, so PCM at low amplitudes has a *lower* signal-to-(quantization)noise ratio than μ-law. The opposite is true at higher amplitudes, but one could argue that PCM's signal-to-noise ratio at high amplitudes is much higher than necessary since very low noise is masked by the loud signal. All around, μ-law is generally better than PCM, at least when you do not have lots of bits to spare.

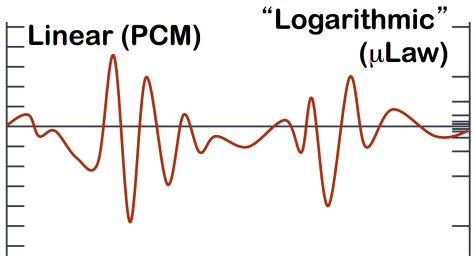

Figure 1: μ-law vs. PCM. Quantization levels are shown at the left and right vertical scales. μ-law has a better signal-to-quantization-error ratio at lower amplitudes but a higher ratio than PCM at higher amplitudes.

2.1 μ-law Implementation

To convert from 8-bit μ-law to PCM, we can treat the μ-law code as an unsigned integer, from 0 to 255, and store the corresponding PCM value in a table. Then, translation from μ-law to PCM is a simple table lookup: `ulaw_to_pcm[ulaw_code]`.

To convert from PCM to μ-law, we could also use a table to map from each of 2^{16} 16-bit PCM values to the corresponding μ-law value. It turns out the low-order 2 bits of the 16-bit code do not matter, so we can reduce the table size to 2^{14}, and we can use symmetry around zero to reduce the table size to 2^{13} or 8 kB[2].

2.2 Discussion on μ-law

μ-law is used mainly for telephony where the sound quality is enough for speech intelligibility, but not good for music. For music, we might consider using more bits to get lower noise. Studies show that with 16-bit samples, linear is better than logarithmic encodings, perhaps because quantization noise at 16-bits is so quiet that you simply cannot hear it. Finally, if you cannot afford 16-bits, you are much better off looking to other techniques.

[1] I've always thought PCM was odd terminology. It literally refers to the *transmission* of binary codes through pulses—see Patent US2801281A—which misses the point that this is about *representation*, not *transmission*, but since this work came out of a phone company, maybe it is not so surprising that PCM was viewed as a form or transmission.

[2] Of course, you could also do binary search to find the nearest μ-law code using the much smaller `ulaw_to_pcm` table, but it is much faster to use direct lookup.

μ-law was chosen as an example of coding redundancy. You might say that μ-law does not actually represent PCM samples, so this is not a true example of coding redundancy. It all depends on what it is you think you are encoding, and you could certainly argue that μ-law is based on perceptual principles so it is really an example of psycho-perceptual redundancy, which we will take up later.

A good example of "pure" coding redundancy is Huffman Coding, where each symbol is assigned a unique string of bits. Symbols that are very common receive shorter bit strings, and rare symbols use more bits. In most music audio, low amplitude samples are more common than high amplitude samples, which are used only for peaks in the signal. Analyzing one of my live recordings, I estimated that the 16-bit samples could be encoded with an average of 12 bits using a variable-length Huffman Code, which would eliminate much of the coding redundancy in the linear encoding. We will discuss Huffman Codes later as well.

3 Intersample Redundancy Examples

Audio signals also exhibit *intersample redudancy*, which means that samples can be predicted from previous samples, so we do not really need all those bits in every sample. The simplest predictor is to guess that the next sample is the same as the current one. While this is rarely exactly true, the error is likely to be a small number.

3.1 DPCM – delta PCM

We can encode the error or *delta* from one sample to the next rather than the full value of each sample. This encoding, caled DPCM, or *delta PCM* saves about 1 bit per sample in speech, giving a 6dB improvement in signal-to-noise ratio for a given bit rate.

Figure 2 shows a DPCM-encoded signal using single bit samples. With a single bit, you can only indicate "go up" or "go down," so if the signal is constant (e.g. zero), the samples alternate 1's and 0's. Note that when the signal to be encoded (the solid line) changes rapidly, the 1-bit DPCM encoding is not able to keep up with it.

Figure 2: DPCM. The solid line is encoded into 1's and 0's shown below. The reconstructed signal is shown as small boxes. Each box is at the value of the previous one plus or minus the step size.

A DPCM encoder is shown in 3. This is a common configuration where the output (on the right) is fed into a *decoder* (labeled "Integrate" in the figure). The decoded signal is subtracted from the incoming signal, providing a difference that is encoded as bits by the box labeled "Quantize." In this way, the overall encoding process drives the error toward zero.

3.2 ADPCM – Adaptive Delta PCM

You can see that the DPCM coder suffers when the input changes rapidly (the limited step size means that the encoded signal cannot change fast enough) and when the input does not change at all (the minimum step size introduces error and noise). In *Adaptive Delta PCM* (ADPCM), the step size is variable. In the one-bit case, repeating bits mean the encoded signal is moving too fast in one direction, so the step size is increased to keep up. Alternating bits mean the encoded signal is overshooting the target, so the step size is decreased. ADPCM achieves an improvement of 10-11 dB, or about 2 bits per sample, over PCM for speech.

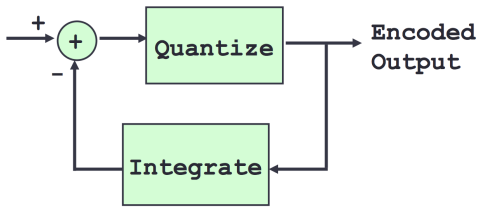

Figure 3: DCPM Coder. A reconstructed signal is subtracted from the signal to be encoded (at left) to decide whether the next output should be up (1) or down (0).

Figure 4 illustrates the action of ADPCM. Notice how the steps in the encoded signal (shown in small squares) changes. In this example, the step size changes by a factor of two, and there is a minimum step size so that reasonably large step sizes can be reached without too many doublings.

Figure 4: ADPCM. The step size is increased when bits repeat, and the step size is decreased (down to a minimum allowed step size) when bits alternate.

Figure 5 shows a schematic of an ADPCM coder. It is similar to the DPCM coder except repetitions or alternations in the output bits are used to modify the step size. The step size is also incorporated into the decoder labeled "Integrate," which decodes the signal.

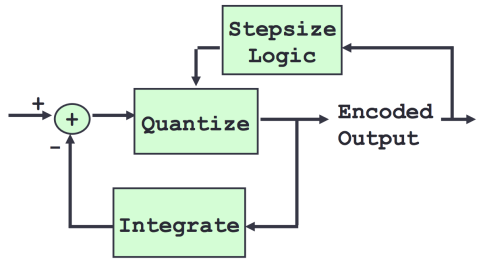

Figure 5: ADPCM Coder. This is similar do DPCM, but the step size is variable and controlled by additional logic that looks for same bits or alternating bits in the encoded output.

3.3 Adaptive Prediction

If the previous sample is a good predictor and helps us to encode signals, why not use more samples to get even better prediction? A simple N_{th}-order predictor forms a weighted sum of previous samples to predict the next sample. If the weights are adapted, this is called *adaptive prediction*. The gain is about 3 or 4 dB, or a fraction of a bit per sample, and there is little to be gained beyond 4^{th}- or 5^{th}-order prediction.

3.4 DPCM for High-Quality Audio

It might seem that we could replace 16-bit PCM with DPCM. Maybe 12 bits would be enough to store deltas, or maybe we could get better-than 16-bit quality with only 16 bits. Unfortunately, to encode *any* PCM signal with N-bit samples, you need $(N+1)$-bit deltas because the delta must span the entire range of an N-bit number, plus it needs a sign bit for the direction of change. Thus, DPCM at the same bit rate cannot be a lossless encoding. In careful listening experiments, PCM is slightly better than DPCM.

3.5 Review

The term PCM refers to uncompressed audio where amplituded is converted to binary numbers using a linear scale. By coding differently with μ-law and A-law, we can achieve better quality, at least for low-fidelity speech audio. Using *intersample redundancy* schemes such as DPCM and ADPCM, we can achieve further gains, but when we consider very high quality audio (16-bit PCM), these coding schemes lose their advantage.

As we will see in the next section, *psycho-perceptual redundancy* offers a better path to high-quality audio compression.

4 Psycho-Perceptual Redundancy and MP3

We have seen how *coding redundancy* and *intersample redundancy* are used in compression, especially for speech signals. For high-quality music audio, the FLAC lossless encoder can compress audio by about 50%, which gives an idea of how redundant is PCM. To get well beyond a 2:1 compression ratio, we must turn to lossy schemes, where the original audio is not preserved, but where the differences are not perceptible and therefore acceptable in most applications.

4.1 Masking and Perceptual Coding

Our auditory system is complex and has some surprising behaviors. First, our ears act as filter banks, separating frequencies into different channels. There is some interaction between frequencies so that a loud sound in a channel *masks* or covers up soft sound in adjacent channels. Masking also applies within a channel so that if you mix soft noise with a loud sound, the noise will be inaudible. This means that it is not always necessary to encode audio with very low noise. We can get away with a lot of quantization as long as there is sound to mask the quantization noise.

Another form of masking occurs over time. A loud sound masks soft sounds that follow for a brief period of time. Most of the masking dissipates within 100ms. Amazingly, masking also works *backwards* in time: a loud sound can mask sounds that happened a few milliseconds earlier!

Another important idea for compression is that when we are digitizing a narrow frequency channel, we do not have to sample at the Nyquist rate. This matters because if we model human hearing and split a sound into, say, 32 channels, we might expect that we need 32 times as many samples. But suppose we have a channel with frequencies from 1000 to 1100 Hz. Rather than sample at 2200 Hz or higher, we can frequency shift the signal down to the range of 0 to 100 Hz. Now, we can sample at only 200 Hz and capture all of the signal. To recover the original sound, we have to frequency shift back to the 1000 to 1100 Hz range. In theory, you can encode a signal as N bands, each at $1/N$ of the sample rate, so the total information is the same.

4.2 Some Insight on Frequency Domain and Coding

All perceptual coding is done in the frequency domain even though PCM works in the time domain. Redundancy is hard to find in the time domain because sound is vibration, implying constant change. Spectral data tends to be more static. The spectrum at time t is a good predictor for spectrum at time $t+1$. In tonal music, the spectrum also tends to be relatively sparse. It is non-zero only where there are sinusoidal partials. Sparse data is easier to encode efficiently.

4.3 MP3 - MPEG Audio Layer 3

MPEG is a video compression standard. Within MPEG is the "MPEG Audio Layer 3" standard for audio compression, commonly known as "MP3." This standard specifies how to decode audio so that MP3 files are treated consistently. Interestingly, the standard does *not* specify how to *encode* audio, so there are slight variations among encoders, resulting in different levels of quality and efficiency. In tests, a 6-to-1 compression of 48 kHz audio gives no perceptual difference from the original. The Fraunhofer Institute, who created the MP3 standard, claims 12-to-1 compression with no perceptual differences.

Let's take a high-level view of how MP3 works. The first step is to apply a filter bank that separates the signal into 32 bands of equal width. As described earlier, each band is sub-sampled by factor of 32 to avoid increasing the overall sample rate. In practice, bands have some overlap and this sub-sampling causes some aliasing. Also, the filters and their inverses are slightly lossy in terms of recovering the original signal.

Next, a psychoacoustic model is applied to estimate the amount of masking that is present in each channel. This determines how much each band can be quantized. The model uses a 1024-point FFT and identifies sinusoids because the masking effect of sinusoids differs from that of noise. The model produces a signal-to-mask ratio for each of the 32 subbands.

Each subband is then transformed with modified discrete cosine transform (MDCT) of length 18 or 6 subband samples. Essentially, this is a short-term frequency representation of the subbands, which allows for more efficient coding. For example, if there is only one sinusoid in the subband, only one MDCT coefficient should be high and others should be low. Next, the MDCT coefficients are quantized according to signal-to-masking ratio. This results in 576 coefficients per frame (18 MDCT coefficients $\times$ 32 subbands).

The coefficients are ordered by increasing frequency because the highest frequencies tend to be zeros—these can be encoded without expending any bits. Next, there will be a run of coefficients that are -1, 0, or 1. These are encoded 4 at a time into alphabet of 81 symbols. The remaining values coded in pairs.

4.4 Details of Huffman Coding

To encode the MDCT coefficients, Huffman Coding is used. Huffman Coding is the most popular technique for removing coding redundancy. A Huffman Code is a variable length encoding of symbols into bit strings, and assuming that each symbol is encoded separately into a string of bits, Huffman Coding is optimal in producing the smallest *expected* bit length of any string of symbols[3].

Figure 6 illustrates the Huffman coding process. To begin with, each symbol is given a probability. E.g. in English, we know that the letter E occurs about 12% of the time, while Z occurs only 0.07%. Thus, we should use fewer bits for E than Z.

The algorithm produces a binary tree (6) using a simple process: Starting with the symbols as leaves of the tree, create a node whose branches have the 2 smallest probabilities. (This would combine D and E in Figure 6.) Then, give this node the probability of the sum of the two branches and repeat the process. For example, we now have C with probability 0.1 and the DE node with probability 0.1. Combining them, we get a new internal node of probability 0.2. Continue until there is just one top-level node. Now, each symbol is represented by the path from the root to the leaf, encoding a left branch as 0 and a right branch as 1. The final codes are shown at the right of Figure 6.

4.5 MP3 Coefficient Coding

Returning to MP3, the terminology of "codes" and "symbols" may be confusing. In Figure 6, the symbols were A, B, C, etc., but in general, these symbols can stand for numbers or even vectors. Any finite set of values can be considered a set of symbols that we can encode. In MP3, one of the things we want to encode are runs of coefficients -1, 0, or 1. These are grouped into 4, so the "symbols" are 4-tuples such as [-1, 0, 1, -1]. To make a Huffman Code, we can let:

[3] Another technique, *arithmetic coding*, and some variations, offer even shorter encodings, essentially by representing symbols by fractional rather than whole bits.

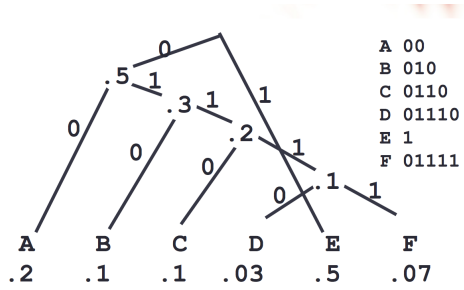

Figure 6: Huffman Coding Tree example.

A = [-1, -1, -1, -1]
B = [-1, -1, -1, 0]
C = [-1, -1, -1, 1]
D = [-1, -1, 0, -1]
E = [-1, -1, 0, 0]
F = [-1, -1, 0, 1]
etc.

Encoding in groups of 4 allows Huffman Coding to find a more compact overall representation.

4.6 Bit Reservoir and Bit Allocation

Even if audio encoding uses a fixed bit rate, some frames are going to be easier to encode than others. A trick used in MP3 is to take advantage of easy-to-compress frames. When bandwidth is left over, bits can be "donated" to a "bit reservoir" and used later to temporarily exceed the maximum data rate and improve the quality of more difficult frames.

Deciding how to use bits for encoding is a difficult problem. Encoders allocate bits to bands where the masking threshold is exceeded by quantization noise. The band is re-encoded and quantization noise is recalculated. This makes encoding slow.

4.7 Summary

Masking reduces our ability to hear "everything" in a signal, and in particular quantization noise. MP3 uses a highly quantized frequency domain representation. Because of different quantization levels and coefficient sizes, Huffman coding is used to encode the coefficients.

5 LPC: Linear Predictive Coding

Another way to achieve compression is to use better, more predictive models of the source sound. This is especially important for voice, where a lot is known about the voice and where there is a big demand for voice data compression. For voice, source/filter models are used because the voice pitch and filter coefficients change slowly. Data rates for speech can be as low as 1 to 2 kbps (kilobits per second) and in cell phone communication are generally in the 5-10 kbps range.

LPC, discussed earlier, is an example of using intersample redundancy. Analysis estimates a vocal tract filter model that allows for good predictions of the output. By inverse filtering the desired signal, we get a source signal that is relatively easy to encode. Figure 7 illustrates the LPC model. This is also an example of an "object model" for data compression. The idea is to extract simple control parameters for an "object" that generates sound. The control parameters and object parameters can be quite small compared to the audio signal that is generated.

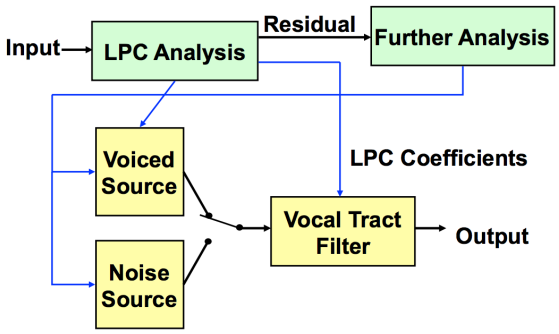

Figure 7: Linear Predictive Coding in Practice

6 Physical Models – Speech Analysis/Synthesis

It seems feasible that object models and physical models can offer interesting compression techniques in the future. Our speech is controlled by muscles, and we know that muscles have low bandwidth compared to audio signals. Also, speech sounds are highly constrained. If we could transmit a model for speech production and muscle control parameters, perhaps the data rate for speech could be much lower and the quality could be higher.

7 Music Notation

The physical and object models approach to compression leads us to consider music notation. In some sense, music notation is high-level, low-bandwith "control" information for performers who actually produce audio. Of course, in normal situations, no two music performances are alike, so we cannot really say that music notation is a compressed form of audio, but it is still interesting to consider notation, what it encodes, and what it does not.

Music notation is compact and symbolic as opposed to digital audio. Music notation is missing a lot of detail of how music is performed, for example details of instruments or muscle movement of performers. Furthermore, getting *any* musical rendition of music notation, complete with expressive interpretation and natural sounding voices and instruments is a big challenge that has not been automated.

Another representation that is similar to music notation is the performance information found in MIDI, or Musical Instrument Digital Interface. The idea of MIDI is to encode performance "gestures" including keyboard performances (when do keys go down?, how fast?, when are they released?) as well as continuous controls such as volume pedals and other knobs and sensors.

The bandwidth of MIDI is small, with a maximum over MIDI hardware of 3 KB/s. Typically, the bandwidth is closer to 3 KB/minute. For example, the complete works of ragtime composer Scott Joplin, in MIDI form, takes about 1 MB. The complete output of 50 composers (400 days of continuous music) fits into 500 MB of MIDI—less data than a single compact disc.

MIDI has many uses, but one big limitation is the inability in most cases to synthesize MIDI to create, say, the sound of an orchestra or even a rock band. An exception is that we can send MIDI to a player piano to at least produce a real acoustic piano performance that sounds just like the original. Another limitation, in terms of data compression, is that there is no "encoder" that converts music audio into a MIDI representation.

8 Summary

We have discussed three kinds of redundancy: *coding redundancy*, *intersample redundancy*, and *psycho-perceptual redundance*. For audio, μ-law, ADPCM, etc. offer simple, fast, but not high-quality or high-compression representations. MP3 and related schemes are more general, of higher quality, and offer higher compression ratios, but the computation is fairly high. We also considered model-based analysis and synthesis, which offer even greater

compression when the source can be accurately modeled and control parameters can be estimated. Music notation and MIDI are examples of very abstract and compact digital encodings of music, but currently, we do not have good automatic methods for encoding and decoding.

9 Acknowledgments

Thanks to Shuqi Dai and Sai Samarth for editing assistance.

Portions of this work are taken almost verbatim from *Music and Computers, A Theoretical and Historical Approach* (http://sites.music.columbia.edu/cmc/MusicAndComputers/) by Phil Burk, Larry Polansky, Douglas Repetto, Mary Robert, and Dan Rockmore. Other portions are taken almost verbatim from *Introduction to Computer Music: Volume One* (http://www.indiana.edu/~emusic/etext/toc.shtml) by Jeffrey Hass. I would like to thank these authors for generously sharing their work and knowledge.