Real Sound Synthesis *for* Interactive Applications

Perry R. Cook

Real Sound Synthesis for Interactive Applications

This page intentionally left blank

Real Sound Synthesis for Interactive Applications

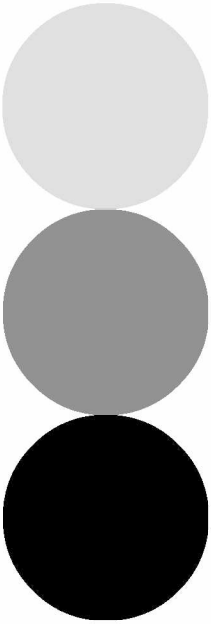
Perry R. Cook

A K Peters
Natick, Massachusetts

CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

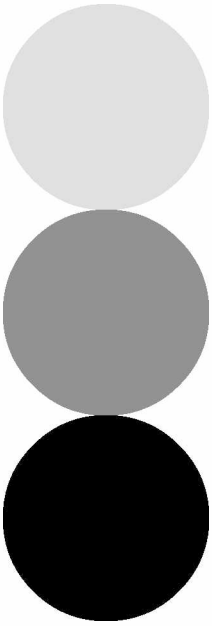
Version Date: 20151015

International Standard Book Number-13: 978-1-4987-6546-6 (eBook - PDF)



Contents

This page intentionally left blank



Introduction

If you're like most people, you haven't thought much about the digital synthesis of sound. If you have thought about sound, or even worked with sound on computers, you still might think that sound is something that you record, or get from a CD, or off the Web, and then manipulate to get the desired result. But there are means and methods to truly "synthesize" sound using physical models, by modeling the actual processes that make sound in nature. You might ask, "Isn't synthesizing sound from physics computationally expensive?" Actually, both old and recent theories in acoustics—combined with algorithmic advances in digital signal processing, and of course, faster processors—now allow many types of sound-producing systems to be physically modeled in real time.

It seems reasonable to ask why all of our requirements could not be satisfied by the status quo in "synthesis" based on prerecorded sounds. Also called *Pulse Code Modulation* (PCM), sampling, or wavetable synthesis, much effort has been spent on making sample-based synthesis more expressive. Using multiple recordings, filters, and various interpolation methods to traverse the space of possible sounds, the state of the art has indeed advanced to the point of absolute realism, at least for single sounds triggered only once. To have truly realistic continuously interactive sound synthesis, however,

essentially infinite memory would be required to store all possible samples. The equivalent of a truly exhaustive physical model would have to be computed in order to determine which subset of the samples and parameters would be required to generate the correct output sound. Finally, the requirement that the correct samples be loaded and available to the synthesis engine would tax any foreseeable real-time sound hardware/ software systems.

As an example, imagine a "Virtual Reality Wood Shop" or a "Virtual Kitchen" controlled by *Virtual Reality* (VR) controllers such as sensor gloves and haptic (combined senses of touch, position, and movement) force feedback devices. As sound designers, our required task might be to ensure that the various tools each make realistic sounds when manipulated, such that a user cannot detect (by audition) a difference between the real and virtual worlds. A frightening number of recorded samples would be required to realistically and responsively synthesize a wood saw, hammer, wrenches, etc., or whisking, mixing, frying, cutting, and all of the other everyday devices and interactions in these environments. The act of striking an object, then restriking it while it still resonates (hammering a nail into a large board, for example) causes changes in the sound that could only be predicted by an actual physical model of the system in question.

Even without the stringent requirement of sonic perfection, it is still scientifically and artistically interesting to inspect, measure, and simulate the physics of sound-producing objects. The computer graphics and animated movie communities march bravely ahead with more elaborate and realistic models for human, dinosaur, and other motion, and though the results are still not absolutely real, large industries have been built on using graphics models for movie and commercial production.

A staple of production for movies, stage performance, television, and radio dramas is the (often last minute) addition of artificial and natural sound effects. For movies, "Foley" artists put shoes on their hands and "walk" in boxes of gravel, leaves, cornstarch (for the sound of snow), and other materials in real time, effectively acting the sounds as they watch the scenes over and over. Radio and stage sound effects performers/engineers use tape cartridges or CDs of prerecorded sounds, and physical noisemakers to add sounds in real time. For offline production and real-time dramas, these methods might indeed be the best means to add sounds. However, virtual reality, training simulations, and games are real-time computer-mediated interactive applications. The computer, not a human, is responsible for providing the sounds algorithmically, in response to external (sensors) and internal (simulations) variables. Thus, achieving the best sonic quality and responsiveness for such applications is basically impossible by simply playing back prerecorded sounds.

If we think about our interactions with sound-producing objects in the world, we note that we excite these objects with a variety of other objects, and in a wide variety of ways. Walking, preparing food, working with metal and wood tools, and riding a bicycle all create sounds. Our evolution and experience in the world has trained us to expect certain sonic responses from certain input gestures and parameters. For example, hitting something harder causes the sound not only to increase in power, but to change in sound "quality" as well. Rubbing a flat object on its edge causes different sonic results than scraping it in the center. In beginning to develop a taxonomy of interactive sounds, a list of the sound-producing interactions under our control would be helpful. These would include:

- blowing (voice, whistles, wind instruments, etc.)
- striking, plucking, etc.
- rubbing, scraping, stroking, bowing, etc.

That this is such a short list seems surprising, but it is instructive. To arrive at a more subtle classification of sounds requires looking at sound-producing mechanisms as well as the human gestures that initiate and control sound. Of course, that is exactly what this book is about: Looking at the physics of sound production and how we cause objects to make sound. Some objects exhibit strong ringing resonances (metal, glass, tubes, plates, and musical instruments). Others are characterized as noisy, but have too much structure to be modeled by just simple noise. We will develop techniques and tools for analyzing and synthesizing sounds, looking at a wide variety of methods. In the process, we will learn about *Digital Signal Processing* (DSP), *Fourier analysis*, and the general nature of sounds in the world. The goal is to come up with simple, physical (or physically motivated), and parametric models for sound synthesis. We will define a *parametric model* as one that has a few variable parameters that can be manipulated to change the interaction, object, and sound.

In the process of designing and calibrating a parametric model, we often learn a lot about the system, and we also can achieve significant compression (data reduction) over the raw sound itself. Thus, one clear benefit of parametrized sound analysis/synthesis is the possibility of significantly reducing the memory storage required for a large library of digital sound effects. Even greater benefits, however, include the ability to drive the synthesis in real time from parameters generated by sensors, game simulations, animations, and other interactive applications. The flexibility and expressiveness offered by parametric modeling makes it attractive for music composition and performance, simulation in virtual reality, computer

games, interactive arts projects, human-computer interfaces, and for other real-time systems where believable sonic responses to user inputs are necessary or desired.

In this book, environmental background sounds (those produced without our own actions and interactions) could be discussed as well. Such sounds would include rain, wind, animal sounds (at least those animals that are not reacting to our presence), trucks and cars, other industrial sounds, and the sounds of humans (again, those not reacting to us). But these sounds do not respond instantly to our actions, and thus have less stringent requirements for simulation. This is not to say that we do not notice if things aren't right in the background sound, and many of the techniques we'll talk about could be directly applicable to background sounds. However, we're not going to deal directly with background sounds here, instead concentrating our focus on interactive sounds.

Thus motivated by scientific, commercial, and artistic interest, and informed by thinking a little about both physical gestures and sound production mechanisms, we will dive into the structure of the book, with some tips on reading it and using materials at http://www.akpeters.com/rss/ to supplement the reading.

Reading Strategy

Since sounds in the real world are made by physical processes, this book will take a physics-based perspective wherever possible. The book is intended for technical, semi-technical and aspiring technical people who know some physics, some programming, and some math. None of these technical skills are specifically required to understand much of what we'll talk about here.

We will present a variety of techniques, ending in Chapter 16 (Examples, Systems, and Applications) with a discussion of practical applications and scenarios. I recommend that you start by scanning Chapter 16, and then come back and start with Chapter 1. I know the world is divided into two types of readers. One type of reader reads the last chapter of any book first, to decide whether to read the book at all, and to break the tension of waiting to see how things come out. The other type of reader views it as almost immoral to learn the surprise before it is time. But for this book, I really do recommend checking out the last chapter, then reading the whole book. The final chapter will first set the stage for reading the book, and will be even more meaningful on the second reading after having learned about sound synthesis.

At the end of every chapter is a short list of recommended reference readings. These lists by no means represent complete sets of references in the various topic areas. Rather, I have tried to glean out the two or so main references on each topic, in case you want to go search and get a better

understanding. Perhaps the most exhaustive and math-intensive reference on physical sound modeling is currently being written by Julius O. Smith. This very large book is not available yet, but in a wonderful move toward open publication, his book is on the web as it is being developed. The topics and techniques of Smith's web book are searchable as well, so I would recommend you check out his web book for extra in-depth reading related to many of the topics in this book.

http://www-ccrma.stanford.edu/~jos/waveguide/

Every chapter in my book also has a list of code examples that illustrate and implement the concepts and techniques that are covered. Many of the examples for the book were generated directly from this code. Some of these are simple stand-alone ANSI C programs that run in a command line and generate a sound file. Others are fairly elaborate C++ projects with real-time synthesis controlled by MIDI and/or Graphical User Interfaces. All of this code, plus much more, can be found at http://www.akpeters.com/rss/. I encourage you to check out the code while you're reading. Appendix D lists and describes the code. Appendix E is a short introductory article on the Synthesis Toolkit in C++, a large collection of classes and applications for real-time sound synthesis.

Every chapter also ends with a list of sound examples from http://www.akpeters.com/rss/. Whenever you're reading along and see an icon like this

that means that there is a sound example on the Web site that pertains to the current topic. The first segment is an audio segment. I encourage you to listen to the sound examples as you read along. Appendix D lists and describes these sound files.

Appendices A, B, and C present mathematical material not covered in the chapters. You are encouraged to refer to those appendices when they are mentioned in the text.

Chapters 1 and 2 look at PCM sampling and define some terms and fundamentals of dealing with recorded digital sound. In those chapters we will also look at problems and issues with working with prerecorded sounds, to further motivate why we want to dig deeper into the physics of sound production.

Chapter 3 dives into the heart *of Digital Signal Processing* (DSP), covering digital filters. If this chapter bores or scares you, you could skip it, but digital filter theory forms the basis of our efficient physical modeling algorithms, so

I do recommend reading as much of Chapter 3 as you can. You should at least understand the concept of linearity before moving ahead.

With Chapter 4, our development of physical modeling really gets underway. By analyzing and simulating a very simple physical mass/spring/damper system, we get our first look at oscillation and sine waves. Synthesizing sounds by adding together sinusoidal modes is introduced, as our first real parametric synthesis method.

Once we know that sine waves arise out of physical vibrations, Chapter 5 looks more at the math of sine waves, and introduces the powerful tool of Fourier analysis. We will see that the *spectrum* (energy in a sound broken up by different frequencies) is a powerful perception-related tool for analyzing and understanding sounds. Appendix A has proofs, theorems, and some other thoughts on Fourier analysis.

Chapter 6 looks further at using the Fourier analysis spectrum as a sound analysis tool. The techniques of spectral modeling are introduced, breaking up sounds into important perceptual and physical components.

Chapter 7 looks at breaking up the spectrum into fairly wide bands, extracting information (parameters) from those bands, and using those parameters to modify and "sculpt" sounds.

Chapter 8 deals with a topic that combines pure mathematics, digital filters, and physical models. Linear prediction is developed and applied to various physical systems such as the human voice, hammering a nail, and a guitar body.

Chapter 9 looks at one of the most fundamental physical vibration systems, the plucked/struck string. We develop an increasingly more realistic, yet still computationally efficient, series of models of plucked and bowed string instruments. Appendix B has derivations and proofs related to the plucked string.

Chapter 10 looks at a family of phenomena that make sonic life very lively. Nonlinearity (you'll have to read Chapter 3 and 10 to get what this really means) is investigated and modeled.

Chapter 11 looks at sounds produced by air in tubes and cavities. We develop some simple, but amazingly rich, models of a clarinet and a blown pop bottle. Appendix C has derivations and proofs related to acoustic tubes.

Chapter 12 deals with physical modeling of plates, membranes, bowls, cups, and other high-dimensional geometric structures.

Chapter 13 looks at sonic "particles" as the fundamental unit of sound production, examining wavetables, physical particle-controlled sinusoidal synthesis, and noise/filter-based statistical synthesis of noisy sounds.

Chapter 14 deals with plucking, striking, bowing, and rubbing excitations for physical models. Friction is discussed at length. MIDI and other protocols for controlling physical models are discussed. Finally, some custom-built physical modeling synthesis controllers are shown.

Chapter 15 describes a complete signal processing system for analyzing and synthesizing the sounds of walking.

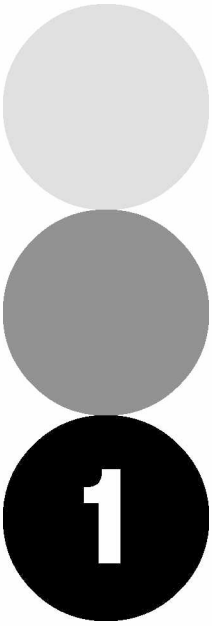
Chapter 16 looks at applications for parametric sound synthesis, including user-interfaces, data sonification, digital Foley, virtual reality, augmented reality, computer music, interactive art, animation, and gaming. The chapter concludes with thoughts on the future of parametric digital sound synthesis.

Acknowledgements

There are scores of people who had major influences on this book being created in the first place, and in it coming to completion. Certainly I must credit and thank the music DSP researchers at Stanford's Center for Computer Research in Music and Acoustics (CCRMA), specifically Julius O. Smith, John Chowning, Chris Chafe, Xavier Serra, Gary Scavone, and lots of others who were around during the exciting development of DSP-based physical modeling. I am grateful for being able to teach the yearly CCRMA summer DSP courses, where I first began to sketch out the chapters and topics for this book. I also thank the students of those courses for asking great questions. I thank Paul Lansky and Ken Steiglitz for recruiting me to Princeton, and getting me to teach the wonderful undergrad Music/DSP course ("Transforming Reality Through Computers") that they had created. I thank the students of Spring 2001 Princeton COS/MUS325 for their reading and use of the preliminary chapters of this book. I thank the researchers at Interval Research for helping support some of my work, and for giving me a wonderful environment in which to hang out for short periods. Specifically I thank Bob Adams, Bill Verplank, Malcolm Slaney, Geoff Smith, Dan Levitan, and many others. For careful reading and criticisms, I thank Georg Essl, Georgos Tzanetakis, Ge Wang, Tae Hong Park, and Liz Douthitt. Thanks to Georg, Georgos, and James O'Brien for recent collaborations on sound synthesis and analysis. Thanks to Gary Scavone for his energy and persistence in pushing the Synthesis ToolKit forward. Thanks to Steve Lakatos for great collaborations on sound perception. Thanks to Dan Trueman and Curtis Bahn for founding and advancing the sensor-enhanced spherical speaker instrument cult. And thanks to many others for long conversations on the topics of digeridoos, dulcimers, and djembes, on prayer bowls, guitars, and accordions, and so many other aspects of the wonderful world of sound.

Development of this book and some of the algorithms described therein was funded in part by gifts from Intel, Interval Research, and the Arial Foundation, and by grants from the National Science Foundation (Grant # 9984087) and The New Jersey Council on Science and Technology (Grant # 01-2042-007-22).

This page intentionally left blank



1 Digital Audio Signals

1.0 Introduction

This chapter will introduce some basics of digital audio. Sampling, quantization, interpolation for sample-rate conversion and pitch shifting, and some basics of audio compression will be covered. The main purpose of this book is to derive and use techniques for generating digital audio samples directly from physical models. But in doing this we will also be analyzing and manipulating digital audio signals from recordings. Some of the models we develop will be based on the analysis of sampled sounds, and all of the models are based on digital audio fundamentals, so it is important to start out with an understanding of the basics of sampling and digital signals.

1.1 Digital Audio Signals

Typically, digital audio signals are formed by *sampling* analog (continuous in time and amplitude) signals at regular intervals in time, and then *quantizing* the amplitudes to discrete values. The time between successive samples is usually denoted as T. Sampling an analog signal requires holding the value steady for a period while a stable measurement can be made, then associating

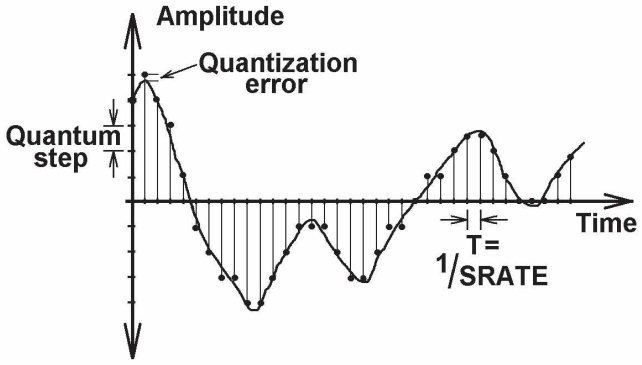


Figure 1.1. Linear sampling and quantization.

the analog value with a digital number (coding). Analog signals can have any of the infinity of real (float) amplitude values. Digital signals can only have a finite number of amplitude values. In converting from analog to digital, rounding takes place and a given analog value must be quantized to the nearest digital value. In the case of rounding down, or truncation, the analog value is quantized to the next lower digital value. The difference between quantization steps is called the quantum. Sampling and quantization is shown in Figure 1.1. Note the errors introduced in some sample values due to the quantization process.

The process of sampling a waveform, holding the value, and quantizing the value to the nearest number that can be digitally represented (as a specific integer on a finite range of integers) is called *Analog to Digital* (A to D, or A/D) conversion. A device which does A/D conversion is called an *Analog to Digital Converter* (ADC). Coding and representing waveforms in sampled digital form is called *Pulse Code Modulation* (PCM), and digital audio signals are often called PCM audio. The corresponding process of converting a sampled signal back into an analog signal is called *Digital to Analog Conversion* (D to A, or D/A), and the device is called a *Digital to Analog Converter* (DAC). Low pass filtering (smoothing the samples to remove unwanted high frequencies) is necessary to reconstruct the sampled signal back into a smooth, continuous time analog signal. This filtering is usually contained in the DAC hardware.

1.2 Sampling and Aliasing

A fundamental law of digital signal processing states that if an analog signal is bandlimited with bandwidth B Hz (Hz = Hertz = samples per second), the

signal can be periodically sampled at a rate of $2B$ Hz, and exactly reconstructed from the samples. Bandlimited with bandwidth B means that no frequencies above B exist in the signal. The rate $2B$ is called the Nyquist rate, and B is called the Nyquist frequency. Intuitively, a sine wave at the highest frequency B present in a bandlimited signal can be represented using two samples per period (one sample at each of the positive and negative peaks), corresponding to a sampling frequency of $2B$. All signal components at lower frequencies can be uniquely represented and distinguished from each other using this same sampling frequency of $2B$. If there are components present in a signal at frequencies greater than half the sampling rate, these components will not be represented properly, and will "alias" as frequencies different from their true original values. To avoid aliasing, ADC hardware often includes filters that limit the bandwidth of the incoming signal before sampling takes place, automatically changing as a function of the selected sampling rate. Figure 1.2 shows aliasing in complex and simple (sinusoidal) waveforms. Note the loss of details in the complex waveform. Also note that samples at less than two times the frequency of a sine wave could also have arisen from a sine wave of much lower frequency. That is the fundamental nature of aliasing, because higher frequencies "alias" as lower frequencies.

((1))

Humans can perceive frequencies from roughly 20 Hz to 20 kHz, thus requiring a minimum sampling rate of at least 40 kHz. Speech signals are often sampled at 8 kHz ("telephone quality") or 11.025 kHz, while music is usually sampled at 22.05 kHz, 44.1 kHz (the sampling rate used on audio Compact Disks), or 48 kHz. New and proposed formats use sampling rates of 96 kHz, and 192 kHz. I consider these ultra-high sampling rates to be generally

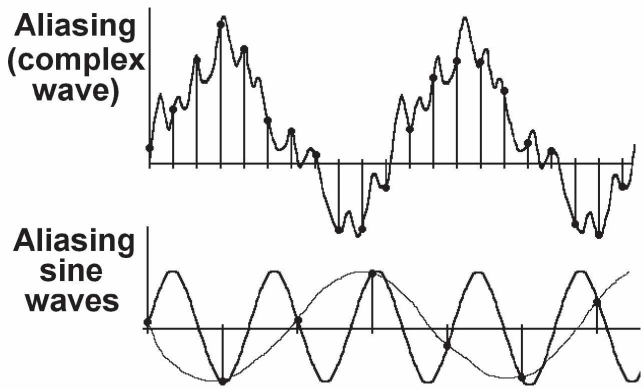


Figure 1.2. Because of inadequate sampling rate, aliasing causes important features to be lost.

quite wasteful (unless you're a dolphin). Extra bits could be much better spent to allow for more quantization levels (see next section), as in 24-bit sampling standards. Even though a roughly 96 dB *Signal-to-Noise Ratio* (see next section) enabled by 16 bits is probably sufficient perceptually, having another eight bits of "headroom" for recording, mixing, and processing is helpful in avoiding overload. For virtual reality, entertainment, and gaming applications, it seems reasonable to allocate more bits to extra channels and speakers, rather than spending them on ultra-high sample rates. In many of the examples in this book, a sampling rate of 22050 Hz is used, to provide adequate quality while conserving processor power and memory.

1.3 Quantization

In a digital system, a fixed number of *binary digits* (bits) are used to represent numbers that approximate the actual values of the samples of the analog waveform. This quantization is accomplished either by rounding to the quantum value nearest the actual analog value (see Figure 1.1), or by truncation to the nearest quantum value less than or equal to the actual analog value. With uniform sampling in time, a properly bandlimited signal can be exactly recovered provided that the sampling rate is twice the bandwidth or greater, but only if there is no quantization. When the signal values are rounded or truncated, the amplitude difference between the original signal and the quantized signal is lost forever. This can be viewed as a noise component upon reconstruction (see Figure 1.3). A common analytical technique assumes that the discarded signal component is uniformly distributed randomly at +/– half the quantum for quantization by rounding, or from zero to the quantum for quantization by truncation.

Using the additive noise assumption gives an approximate best-case *Quantization Signal-to-Noise Ratio* (SNR) of $6N$ dB, where N is the number

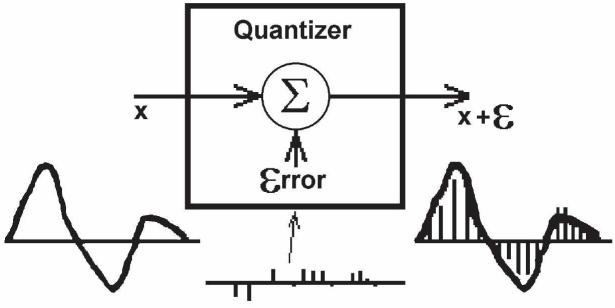


Figure 1.3. Quantization shown as an additive noise signal.

of bits. A dB is a decibel, defined as $20\log_{10}$ (highest amplitude/lowest amplitude). This SNR can be derived from assuming a square wave at maximum amplitude, which exhibits a power (power is defined as the sum of the squares of signal values) of

$$2^{N-1} \cdot 2^{N-1} = 2^{2N-2} \cong 4^N$$

and uniform random quantization noise of 1/2 bit average amplitude, which exhibits power of 1/12 (look this up in a statistics textbook as the variance of a uniform distribution). Yes, one would not find a constant amplitude square wave exhibiting random quantization noise, but play along here for the purposes of the derivation.

$$\text{SNR} = 10\log_{10}[4^N/(1/12)] = 10N\log_{10}(4) + C = N \cdot 6.0206 + C$$

Using the above approximations implies that a system using 16-bit linear quantization will exhibit an SNR of approximately 96 dB. 8-bit quantization exhibits a signal to quantization noise of approximately 48 dB. Each extra bit improves the SNR by about 6 dB.

((2))

Most computer audio systems use two or three types of audio data words. 16-bit (per channel) data is quite common, and this is the data format used in Compact Disk systems. 8-bit data is common for speech data in PC and telephone systems. There are also methods of quantization that are non linear, meaning that the quantum is not constant over the range of input signal values. In these systems, the quantum is smaller for small amplitudes, and larger for large amplitudes. This nonlinear quantization will be discussed more in Section 1.5.1.

1.4 Resampling and Interpolation

In order to synthesize music, speech, or sounds in general, accurate and flexible control of pitch is necessary. The most accurate pitch control is necessary for music synthesis. In sampling synthesis, this is accomplished by dynamic sample rate conversion (interpolation). In order to convert a sampled data signal from one sampling rate to another, three steps are required: *bandlimiting, interpolation, and resampling.*

Bandlimiting: First, it must be assured that the original signal was properly bandlimited to less than half the new sampling rate. This is always true when "upsampling" (converting from a lower sampling rate to a higher one). When "downsampling," however, the signal must be first low pass filtered to exclude frequencies greater than half the new target rate.

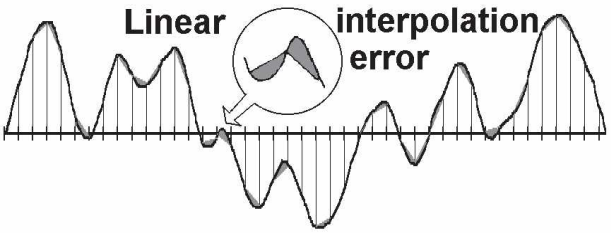


Figure 1.4. Gray regions show errors due to linear interpolation.

Interpolation: When resampling by any but trivial factors such as 2, ½, ¼, etc., the task requires computing the values of the original bandlimited analog signal at the correct points between the existing sampled points. This is called interpolation. The most common form of interpolation used is linear interpolation, where the fractional time samples needed are computed as a linear combination of the two surrounding samples. Many assume that linear interpolation is the correct means, or at least an adequate means for accomplishing audio sample rate conversion. Figure 1.4 shows resampling by linear interpolation. Note the errors shown by the shaded regions.

Some might notice that linear interpolation isn't quite adequate, so they might assume that the correct solution must lie in more elaborate curve-fitting techniques using quadratics, cubics, or higher order splines. Actually the task should be viewed and accomplished as a filtering problem, with the filter designed to meet some appropriate error criterion. As we will see in Chapter 3, linear time-invariant filtering can be accomplished by "convolution" with a filter function. If the resampling filter is defined appropriately, we can exactly reconstruct the original analog waveform from the samples.

The "correct" (ideal in a provable digital signal processing sense) way to perform interpolation is convolution with the sinc function, defined as:

$$\text{sinc}(t/T) = \sin(\pi t/T)/(\pi t/T), \text{ where } T = 1/\text{SRATE}.$$

The sinc function is the ideal low pass filter with a cutoff frequency of SRATE/2. Figure 1.5 shows reconstruction of a continuous waveform by convolution of a digital signal with the sinc function. All samples are multiplied by a corresponding continuous sinc function, and added up to arrive at the continuous reconstructed signal.

((3))

Resampling: This is usually accomplished at the same time as interpolation, because it is not necessary to reconstruct the entire continuous waveform in order to acquire new discrete samples. The resampling ratio can be time-varying, making the problem a little more difficult. However, viewing

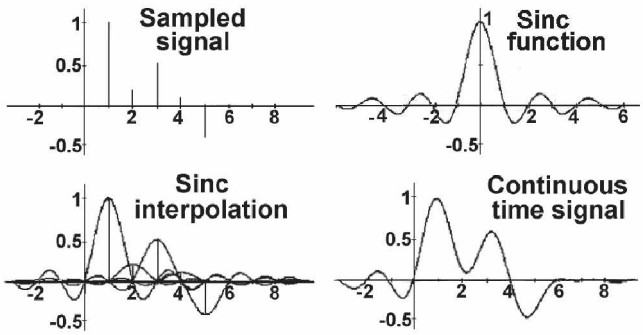


Figure 1.5. Sinc interpolation for reconstruction and resampling.

the problem as a filter-design and implementation issue allows for guaranteed tradeoffs of quality and computational complexity.

1.5 Digital Audio Compression

The data required to store or transmit CD-quality audio at a 44.1 kHz sampling rate, in stereo (two channels, left and right), and at 16 bits, is 176 kbytes per second. For storage, this means that a three minute song requires 30 megabytes of storage. For transmission, this means that an ISDN line at 128 kbits per second is over ten times too slow to carry uncompressed CD-quality audio. When using PCM to synthesize a large variety of sounds for games or virtual reality, storage and access can quickly become a consideration. Audio compression, as with any type of data compression, strives to reduce the amount of data required to store and/or transmit the audio. Compression algorithms can be either lossless (meaning that the original data can be retrieved exactly from the compressed data), as they must be for computer text and critical data files, or lossy, as they usually are for images and audio.

1.5.1 μ-Law Compression

Rather than using a linear quantization scheme, non linear quantization can be used to try to keep a more constant quantization noise, with respect to the amplitude of the signal. Figure 1.6 depicts simple non linear quantization.

The most common system for nonlinear quantization is called μ-law (mu law), which uses 8 nonlinear bits to achieve the same approximate SNR as 12-bit linear quantization. Nonlinear quantization is used in telephone systems, where 8-bit μlaw at 8 kHz is used to transmit speech over a standard US digital phone line. 8-bit μlaw decoding is easily accomplished with a

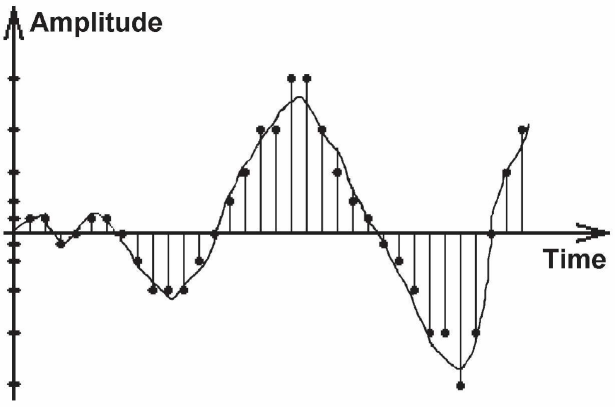


Figure 1.6. Nonlinear waveform quantization.

256 point lookup table, by storing 256 16-bit short integers for reconstruction to the standard 16-bit linear format. Since the quantization noise is related to the size of the quantum, having a small quantum ensures small quantization noise, and an improved signal-to-noise ratio for small amplitude signals. For large amplitude signals, the quantum is large, but the SNR is held relatively constant. A common metric applied to nonlinear μ-Law sampling is that N bits performs roughly as well as N+4 linearly quantized bits. 8-bit μ-Law is common for speech coding and transmission, providing 2:1 compression over 16-bit linear audio data with a perceptual quality roughly equal to 12-bit linear quantization.

((4))

1.5.2 Adaptive Delta Pulse Code Modulation

Adaptive Delta Pulse Code Modulation (ADPCM) endeavors to adaptively change the quantum size to best match the signal. The changes (deltas) in the signal are coded rather than the absolute signal values. For each sample, the change in the signal value versus the last value is computed, this delta is compared to the current adapted delta value, and the new adapted delta value is increased or decreased accordingly. To code ADPCM, the sign of the delta for the given step, and a 3-bit value reflecting the current quantized adapted value are stored. This allows 4 bits of information to store a 16-bit linearly quantized sample value, providing 4:1 compression over 16-bit linear audio data. ADPCM is supported by many multimedia audio systems, but the quality is generally considered too low for music and other quality-critical audio material.

((5))