SH
SM58

M05_BURG5802_01_SE_C05.QXD 7/2/08 12:29 PM Page 253

CHAPTER 5

Digital Audio Processing

M05_BURG5802_01_SE_C05.QXD 7/2/08 12:29 PM Page 254

CHAPTER 5

No sound is dissonant which tells of life.
—Samuel Coleridge, This Lime-Tree Bower My Prison

OBJECTIVES FOR CHAPTER 5

- Know the basic hardware and software components of a digital audio processing environment.
- Know the common file types for digital audio and be able to choose an appropriate file type and compression for audio files.
- Understand the difference between destructive and nondestructive audio editing.
- Understand how normalization, compression, expansion, equalization, and reverb are applied and what they do to digital audio.
- Understand methods for audio restoration.
- Understand how filters are applied and how they work mathematically.
- Understand the concept and examples of time-based encoding for digital audio.
- Understand the concept and examples of perceptual encoding for digital audio.
- Understand the concept, implementation, and application of MPEG audio compression.

5.1 TOOLS FOR DIGITAL AUDIO PROCESSING

5.1.1 Digital Audio Work Environments

In Chapter 4, we introduced basic concepts of digital audio representation. Now it's time to consider how you actually record and edit digital audio. Throughout this book, the goal is to uncover the mathematics and algorithms that underlie work in digital media and to give you knowledge that lasts longer than the short shelf-life of most editions of digital media application programs. But because being able to *do something* with your knowledge is one of the biggest motivators you can have, we chose the book's topics based on real hands-on work. So this chapter begins with an overview of the hardware and software tools for working with digital audio.

There are two main environments in which you might work with audio: during live performance or in a sound recording studio. Let's look at the equipment for live performances first—the kind of equipment you would use as a sound engineer for a theater play, musical, or concert—comparing an analog to a digital hardware setup.

Figure 5.1 is a diagram of the type of analog equipment you could use to manage sound during a live performance or in studio recording. Central in this configuration is the ***mixer***, also called a ***mixing console*** or ***soundboard***. A mixer is used to gather inputs from microphones and instruments and dynamically adjust their amplitudes, equalize the frequency components, compress the dynamic range, apply special effects, and route the processed sound to outputs such as speakers. Each input to the mixer is designated as a ***channel***, with 24 or more channels in larger mixers. Multiple channels can be gathered into a ***bus*** (a subsignal that is the sum of the inputs to it), and the effects can be applied to the bus. The signal

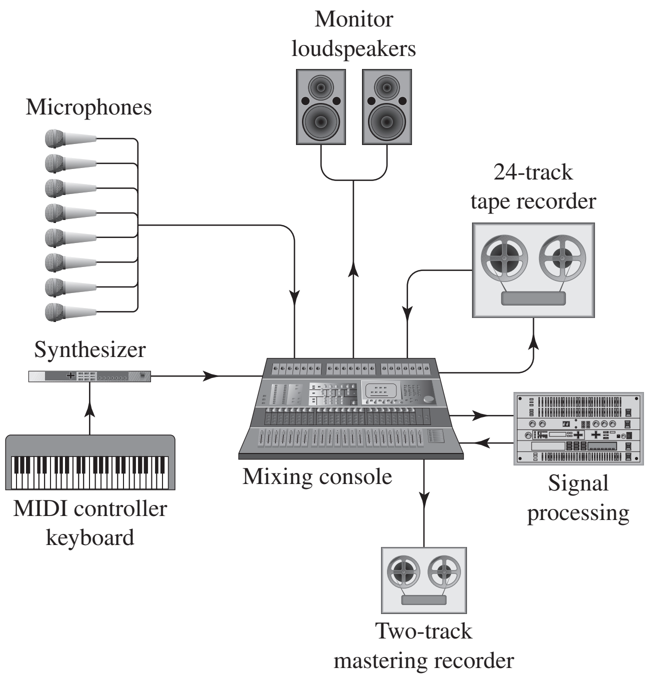


Figure 5.1 Analog sound equipment

can be split to multiple outputs so that speakers can be set up appropriately around an auditorium to create the desired sound environment.

Figure 5.2 shows what your live performance audio set-up would look like if you used a digital mixer. Notice how much hardware is absorbed into the software of the digital mixer. Inside the digital mixer is an analog-to-digital converter (ADC). As the signal enters the mixer, it is immediately converted to digital form so that further processing can be done on it in software. Before it is sent to the outputs, it is converted back to analog form with a digital-to-analog converter (DAC).

Sound processing can also take place in settings other than live performances—for example, in sound recording studios. Much of the equipment is the same, with the difference that the sound is recorded, refined, processed with special effects, and finally compressed and packaged for distribution. Much of this work can be done by means of a personal computer equipped with the right sound card, microphones, and audio processing software.

In this book, we're going to assume that you're working in a mostly digital environment, using your personal computer as the central component of your ***digital audio workstation*** (***DAW***). The most basic element of your DAW is a sound card or external sound interface. A ***sound card*** is a piece of hardware that performs the following basic functions:

- provides input jacks for microphones and external audio sources
- converts sound from analog to digital form as it is being recorded, using an ADC
- provides output jacks for headphones and external speakers
- converts sound from digital to analog form as it is being played, using a DAC
- synthesizes MIDI sound samples using either FM or wavetable synthesis, more often the latter. (Most sound cards have this capability.)

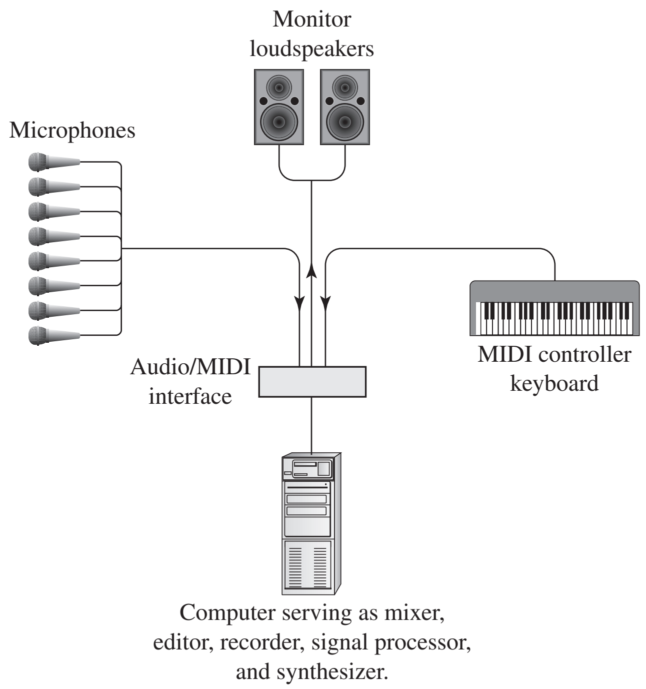

Monitor loudspeakers
Microphones
Audio/MIDI interface
MIDI controller keyboard
Computer serving as mixer, editor, recorder, signal processor, and synthesizer.

Figure 5.2 Digital sound equipment

If the sound card that comes with your computer doesn't do a very good job with ADC, DAC, or MIDI-handling, or if it doesn't offer you the input/output interface you want, you may want to buy a better internal sound card or even an external sound interface. An external sound interface can offer extra input and output channels and connections, including MIDI, and a choice of XLR, ¼ inch TRS, or other audio jacks to connect the microphones (Figure 5.5). To complete your DAW, you can add digital audio processing software, headphones, and good quality monitors. If you want to work with MIDI, you can also add hardware or software MIDI samplers, synthesizers, and/or a controller.

Many computers have built-in microphones, but their quality isn't always very good. The two types of external microphones that you're most likely to use are dynamic and capacitor mics. A ***dynamic mic*** (Figure 5.3) operates by means of an induction coil attached to a diaphragm and placed in a magnetic field such that the coil moves in response to sound pressure oscillations. Dynamic mics are versatile, sturdy, and fairly inexpensive, and they're good enough for most home or classroom recording studio situations.

A ***capacitor mic*** (also called a ***condenser mic***) has a diaphragm and a backplate that together form the two plates of a capacitor (Figure 5.4). As the distance between the plates is changed by vibration, the capacitance changes, and in turn, so does the voltage representing the sound wave. The material from which the plates are made is lighter and thus more sensitive than the coil of a dynamic mic.

The main differences between dynamic and capacitor mics are these:

- A dynamic mic is less expensive than a capacitor mic.
- A capacitor mic requires an external power supply to charge the diaphragm and to drive the preamp. A dynamic mic is sometimes more convenient because it doesn't require an external power supply.

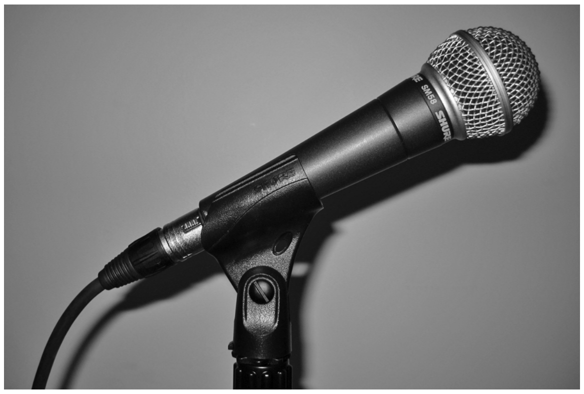


Figure 5.3 Dynamic mic

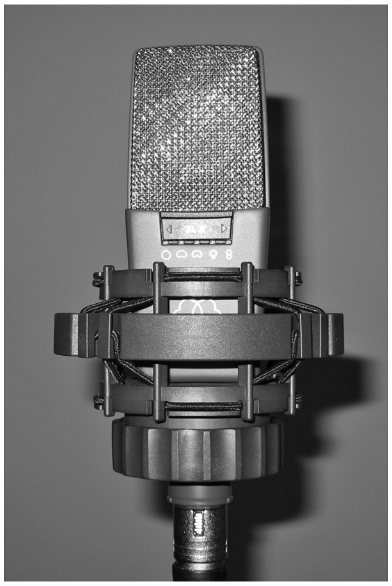


Figure 5.4 Condenser mic

- A dynamic mic has an upper frequency limit of about 16 kHz, while the capacitor's frequency limit goes to 20 kHz and beyond. This difference is probably most noticeable with regard to the upper harmonics of musical instruments.
- A dynamic mic is fine for loud instruments or reasonably close vocals. A capacitor mic is more sensitive to soft or distant sounds.
- A capacitor mic is susceptible to damage from moisture condensation.

You can't really say that one type of microphone is better than another. It depends on what you're recording, and it depends on the make and model of the microphone. Greater sensitivity isn't always a good thing, since you may be recording vocals at close range and don't want to pick up background noise; in this case a dynamic mic is preferable.

Microphones can also be classified according to the direction from which they receive sound. The ***cardioid mic*** is most commonly used for voice recording, since its area of sensitivity resembles a heart-shape around the front of the mic. An ***omnidirectional mic*** senses sound fairly equally in a circle around the mic. It can be used for general environmental sounds. A ***bidirectional mic*** senses sound generally in a figure-eight area. The direction from which sounds are sensed depends in part on the frequency of the sound. In the case of high frequencies, the directional response of a microphone depends in part on the angle from which the sound arrives. Some high quality microphones have switches that allow you to change their patterns of optimum reception. You can see the icon's for different reception patterns on the microphone in Figure 5.4.

Another property of microphones is their frequency response—the range of frequencies that it picks up. The Shure SM50 microphone has a frequency response between 50 and 15000 Hz. A good condenser mic like the one pictured in Figure 5.4 could have a frequency response of 20 to 20000 Hz.

You may want to buy monitors for the audio work you do on your computer. A ***monitor*** is just another name for a speaker—but one that treats all frequency components equally. Speakers can be engineered to favor some frequencies over others, but for your audio work you want to hear the sound exactly as you create it, with no special coloring. For this reason, you should pay particular attention to the frequency characteristics of the speakers or monitors you use. It's nice to have a good set of earphones as well, so that you can listen to your audio without environmental background noise.

You'll need a variety of cables and connectors to complete your digital audio workstation. Figure 5.5 shows commonly used audio connectors. To connect your computer to an external audio or hard disk drive, you'll probably use ***Firewire*** or ***USB***. Firewire provides a high speed data connection, and thus it is very popular for audio, video, and hard disk drive connections. USB and USB2 are used for the same purpose. USB connections can be convenient in that an external device may be able to draw its power from the USB connection and therefore doesn't require a power cord. Firewire is advertised as having a data rate of 400 Mb/s and USB2 a data rate of 480 Mb/s. However, USB2 goes through the CPU to determine the direction of data flow, and Firewire does not. Thus, Firewire actually turns out to be faster, not stealing any CPU time that could be used for audio processing. Firewire connections come in two types—six pin or four pin. You need to be sure to have the right type for your equipment. You might need a Firewire cable that is six pin on both ends, six pin on one end and four on the other end, and so forth.

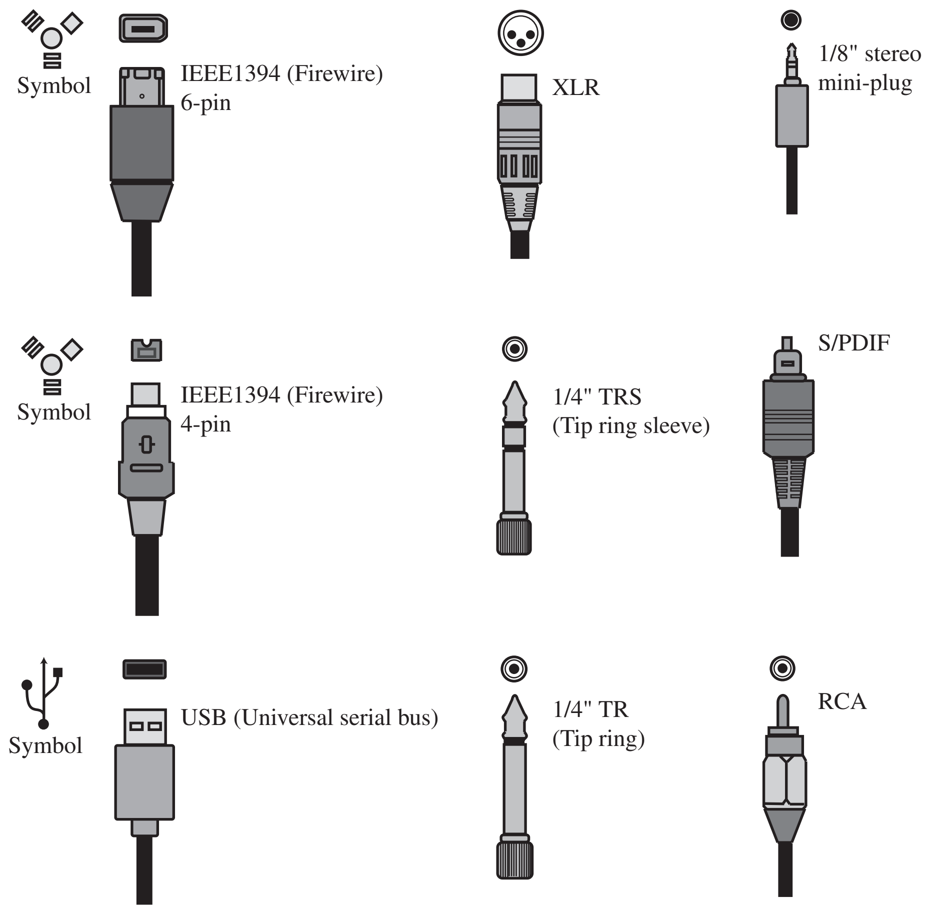
If you're connecting a digital audio recorder to your computer or external sound interface, you use an S/PDIF connection, which stands for Sony Phillips Digital Interface. S/PDIF is made to transmit audio data that is already digitized, so it doesn't have to go through an analog-to-digital converter in the sound interface.

The remaining connectors discussed in this section are analog, which link a microphone—which is an analog device—to an analog-to-digital converter that resides in your computer's sound card or in an external sound interface. The connector type that this entails depends on the type of microphone.

Inexpensive consumer microphones generally have an ⅛″ stereo miniplug on the end that go directly to your computer's internal sound card. This type of microphone doesn't provide professional quality recording.

TRS and TR ¼ inch connectors are used with a wide variety of consumer and professional equipment such as microphones amplifiers, guitars, headphones, and speakers. The ¼ connectors can be balanced or unbalanced (explained below).

To use a good quality microphone, you'll need an XLR cable. External sound interfaces generally have XLR inputs. The sound interface provides a female three pin connection port and the microphone has a male three pin connection. The XLR cable that connects the



Symbol
IEEE1394 (Firewire) 6-pin
XLR
1/8" stereo mini-plug
Symbol
IEEE1394 (Firewire) 4-pin
1/4" TRS (Tip ring sleeve)
S/PDIF
Symbol
USB (Universal serial bus)
1/4" TR (Tip ring)
RCA

Figure 5.5 Types of audio connectors

two is female on the microphone end and male on the sound interface end. XLR connectors provide a high quality, balanced, low impedance signal.

If your sound card has ¼ inch inputs and you want to connect a microphone that uses XLR connections, you can get an XLR to ¼ inch adaptor. However, the transformation of the signal that is done via the adaptor sacrifices some audio quality in your recording.

RCA connectors have been around for a long time as part of consumer audio and video equipment. They often are used with devices like external tape decks or video cameras. You have probably seen them in color coded three-jack connections for where the yellow plug is for composite video, white is for the left or mono audio connection, and red is for the right audio connection. (See Figure 6.11.) RCA connectors transmit an unbalanced signal.

As you consider the types of connections you need, you'll encounter the terms impedance and balance. ***Impedance*** (abbreviated Z) is the measure of the total opposition to current flow in an alternating current circuit. Measured in ohms, impedance is the sum of two components, resistance (R) and reactance (X). Ohms is abbreviated Ω, 1,000 ohms is abbreviated kΩ, and 1,000,000 ohms is abbreviated MΩ.

The source of an audio signal—like a microphone or electric guitar—is called the ***source*** or ***output***. The place where you plug this source into a sound card, external sound interface, or mixer is called the ***load*** or ***input***. The significance of impedance is that it affects what types of outputs should be plugged into what kinds of inputs.

A microphone up to about 600 Ω is low impedance, between 600 and 10,000 Ω is medium impedance, and 10,000 Ω or more is high impedance. A high impedance mic or instrument generally outputs a higher amplitude signal, measured in voltage, than a lower impedance one. This does not imply that high impedance is better than low impedance. Most good mics are low impedance, particularly those with XLR connectors.

It's not good to connect a higher impedance output to a lower impedance input because the frequencies that are recorded may be distorted. An electric guitar generally is a high impedance source. It often has a ¼ inch TRS output connection like the one shown in Figure 5.5. You shouldn't connect a high impedance electric guitar to a low impedance input jack. Some external audio interfaces provide ¼ inch inputs that are designed to receive an electric guitar for direct recording of its output. You can check the specifications of your sound interface to see if an input is provided with correct impedance for an electric guitar. (USB outputs also exist for electric guitars.)

Another thing you should know about high impedance output that it isn't good when you have to connect the output to the input by means of a long cable. With a long cable, there's more chance that high frequencies will be lost in the audio signal. This is because high frequencies are affected more by reactance. Also, a longer cable provides more chance for the high impedance output to attract electrical interference on its way to the input. This can add noise to the recording in the form of a hum. A low-impedance mic can be used with hundreds of feet of cable without picking up hum or losing high frequencies. A medium-impedance mic cable is limited to about 40 feet, and a high-impedance mic is limited to about ten feet. The extent to which high frequencies are lost depends also on the capacitance of the cable.

The other consideration is whether a connection transmits a balanced or unbalanced signal. ***Balanced signals*** have two signal conductors and one shield. The two signal conductors are identical except that they are 180 degrees out of phase with each other. (Note that adding two sine waves that are 180 degrees out of phase yields 0 amplitude at all points.) If noise is added to the line, it is identical in both of the signals. Thus, adding the two signals cancels the original signal and yields two times the noise. This identifies the frequency components of the noise in the signal, which can then be eliminated. The significance of this is that a balanced connection generally has less noise than an unbalanced one. ***Unbalanced signals*** use only two signal conductors—a center conductor and a grounded shield. The shield is intended to protect the center signal from electrical interference, but it doesn't block all the interference. When interference does get through, it results in a hum or buzz added to an audio recording. This is more likely to happen when the cable between the output and input is long.

The bottom line is that balanced low impedance microphones (e.g., those with XLR connections) provide a superior audio signal in recording.

5.1.2 Digital Audio Processing Software

Digital audio processing software and MIDI controllers give you an interface to your sound card, allowing you to record, edit, compress, and save audio and MIDI files. As of the writing of this chapter, Digidesign Pro Tools (Mac or Windows), Adobe Audition (Windows), Apple Logic (Mac), Sony Sound Forge (Windows), and Nuendo (Mac or Windows) are among the most popular commercial audio application programs. Cakewalk Music Creator and Cakewalk Sonar are combined audio/MIDI processing packages. Propellerhead Reason (Mac or Windows) is a sophisticated audio/MIDI software package that provides a sequencer, drum machines, and a wide variety of software synthesizers that are visually and functionally modeled after analogous hardware. Reason can be used as a stand-alone or as a plug-in to other audio processing software. Acid Pro is a loop-based music creation tool. Max/MSP (Mac and Windows), developed by Cycling '74, is an interesting development environment for interactive music and audio processing. Many freeware, shareware, and commercial digital audio tools are available for Linux and Unix, also, including (at the time of this writing) DAP, MixViews, ReZound, Slab, JoKosher, KWave, Brahms, Jazz++, TiMidity++, Playmidi, BladeEnc, and aRts. Audacity is an open source program that runs under Mac, Windows, and Linux operating systems.

Features that you generally find in digital audio processing software include the following:

- the ability to import and save audio files in a variety of formats
- an interface (called ***transport controls***) for recording and playing sound
- a waveform view that allows you to edit the wave, often down to the sample level
- multitrack editors
- audio restoration tools to remove hisses, clicks, pops, and background noise
- the ability to take input from or direct output to multiple channels
- special effects such as reverb, panning, or flange
- controls for equalizing and adjusting volume and dynamic range
- frequency filters
- the ability to handle the MIDI format along with digital audio and to integrate the two types of data into one audio file
- the ability to record samples and add to the bank of MIDI patches
- compression codecs

Not all audio application programs include all the features listed above. In some cases, you have to install plug-ins to get all the functionality you want. Three common plug-in formats are VST (Virtual Studio Technology), TDM (time division multiplexing), DirectX, and Audio Units by Core Audio. Plug-ins provide fine-tuned effects such as dynamic compression, equalization, reverb, and delay (explained later in this chapter).

In digital audio processing programs, you often have a choice between working in a ***waveform view*** or a ***multitrack view***. The waveform view gives you a graphical picture of sound waves like we have described in Chapter 4. You can view and edit a sound wave down to the level of individual sample values, and for this reason the waveform view is sometimes called the ***sample editor***. The waveform view is where you apply effects and processes that cannot be done in real-time, primarily because these processes require that the whole audio file be examined before new values can be computed. When such processes are executed, they permanently alter the sample values. In contrast, effects that can be applied in real-time do not alter sample values.

The waveform view of an audio file displays time across the horizontal axis and amplitude up the vertical axis. The standard representation of time for digital audio and video is ***SMPTE***, which stands for ***Society of Motion Picture and Television Engineers***. SMPTE divides the timeline into units of hours, minutes, seconds, and frames. The unit of a frame is derived from digital audio's association with video. A video file is divided into a sequence of individual still images called ***frames***. One standard frame rate is 30 frames per second, but this varies according to the type of video. A position within an audio file can be denoted as $h:m:s:f$. For example, $0:10:42:14$ would be 10 minutes, 42 seconds, and 14 frames from the beginning of the audio file. Audio editing programs allow you to slide a "Current

Position Cursor" to any moment in the audio file, or alternatively you can type in the position as $h:m:s:f$. Other time formats are also possible. For example, some audio editors allow you to look at time in terms of samples, bars and beats, or decimal seconds.

The multitrack view allows you to record different sounds, musical instruments, or voices on separate tracks so that you can work with these units independently, modifying frequency or dynamic range, applying reverb or phase shifts, whatever you want for your artistic purposes. A ***track*** is a sequence of audio samples that can be played and edited as a separate unit. Often, different tracks are associated with different instruments or voices. You can record one instrument on one track and then you can record a second instrument or voice on another. If you play the first track while recording the second, you can easily synchronize the two. With the right digital processing software and a good computer with plenty of RAM and external storage, you can have your own sound studio. If you're also a musician, you can do all of the musical parts yourself, recording one track after another. In the end, you'll probably want to ***mix down*** the tracks, collapsing them all into one unit. This is very much like flattening the layers of a digital image. Once you have done this, you can't apply effects to the separate parts (but if you're smart, you'll have kept a copy of the multitrack version just in case). The mixed-down file might then go through the mastering process. ***Mastering*** puts the final touches on the audio and prepares it for distribution. For example, if the audio file is one musical piece to be put on a CD with others, mastering involves sequencing the pieces, normalizing their volumes with respect to one another so one doesn't sound much louder than another, altering their dynamic ranges for artistic reasons, and so forth.

A channel is different from a track. A ***channel*** corresponds to a stream of audio data, both input and output. Recording on only one channel is called monophonic or simply ***mono***. Two channels are ***stereo***. When you record in stereo, the recording picks up sound from slightly different directions. When the sound is played back, the intention is to send different channels out through different speakers so that the listener can hear a separation of the sound. This gives the sound more dimension, as if it comes from different places in the room. Stereo recording used to be the norm in music, but with the proliferation of DVD audio formats and the advent of home theaters that imitate the sound environments of movie theatres, the popularity of multichannel audio is growing. A 5.1 multichannel setup has five main channels: front center, front right, front left, rear right, rear left, and a ***low frequency extension channel*** (***LFE***). LFE, often called a ***subwoofer***, has frequencies from 10 to 120 Hz. A 6.1 multichannel setup has a back center channel as well.

5.1.3 Destructive Vs. Nondestructive Editing

Changes made to digital audio files by means of audio processing software are generally handled in one of two ways: by means of ***destructive editing*** or ***nondestructive editing***.

Think about what happens when you work on an audio file that you previously recorded and saved in permanent storage—say, on your hard disk. Audio files can be very large, so the whole file will not be held in RAM and operated on there as you edit it. The audio processing program will have to record your changes as you work by writing to a file on the disk drive. The question is, will it change your original audio file even before you do a "Save" operation? In immediately destructive editing, this is exactly what would happen. If you added reverb to your sound, a destructive editor would immediately apply the reverb operation to the original file on disk, so you couldn't get the original file back.

Fortunately, audio processing programs generally don't handle editing in this manner. Instead, they create a temporary (temp) file on the hard disk and write any changes you make to this temporary copy of the audio file. If you never save the edited version, the temp file goes away. If you do save the edited audio, the previous version is overwritten with the temp file.

If most edits are not *immediately* destructive, then what's the distinction between destructive and non-destructive edits? In most audio processing programs, destructive edits are handled in the manner described in the previous paragraph. If your editing environment allows "undos"—and most of them do—then your operations will be saved in a series of temp files, and you can revert back to previous states by undoing instructions. The editing operations don't become permanent until you save the audio file.

Nondestructive edits are handled by means of additional instructions that are added to the audio file. For example, you could indicate that you want a volume change in a portion of the audio. Instead of altering the amplitudes of the samples, the editing program would save an instruction indicating the level, beginning point, and ending point of the volume change. The assumption here is that you would be saving the audio file in a file format that accommodates these extra instructions—for example, the proprietary file format of the audio processing program. If you eventually decide to mix your audio down to another standard file format like WAV, at that point the changes become destructive in that the sample values are changed.

Sometimes a distinction is made between the waveform view and the multitrack view, the former being an environment for destructive edits and the latter for nondestructive. You should consult your application program's Help so you know how editing is handled in your working environment. You'll also want to set your maximum number of "undos." It's nice to be able to undo a lot of operations, but you'll need sufficient disk space for all the temp files.

Real-time processing is always nondestructive. It's amazing to discover just how much real-time processing can go on while an audio file is playing or being mixed in live performance. It's possible to equalize frequencies, compress dynamic range, add reverb, or even combine these processing algorithms. The processing can happen so quickly that the delay isn't noticed. However, all systems have their limit for real-time processing, and you should be aware of these limits. In mixing of live performances, there are also times when you have to pay attention to the delay inevitably introduced in any kind of digital audio processing. ADC and DAC themselves introduce some delay right off the top, to which you have to add the delay of special effects.

5.1.4 Digital Audio File Types

If you have worked at all with digital audio, you've seen that audio files are saved in a variety of formats, as indicated by the many file extensions: *.wav*, *.au*, *.mp3, .aac*, and so forth. How do you know which is right for your application? What distinguishes one file type from another?

File formats differ in how they answer the following questions:

- How are the samples encoded?
  - Using linear quantization?
  - Using logarithmic quantization?
  - Are sample values signed or unsigned?
- What is the format of the data?
  - Is there a header?
  - Are bytes stored in little-endian or big-endian order?
  - Are channels interleaved?

- What type of information is stored in the file?
  - Are there constraints on the sampling rate and/or bit depth?
  - Are there constraints on the number of channels?
  - Is extra information allowed, like timestamps or loop points?
- Is the file compressed?
  - Using a lossy procedure?
  - Using a lossless procedure?
- Is the format "open" or proprietary?
  - Is the code open source?
  - Does the format follow a widely accepted standard such as IMA or IFF?

Let's consider uncompressed files first. An audio file format that is described as ***linear PCM*** stores a sequence of audio samples encoded in linearly spaced quantization intervals, which means that each interval is the same size. (See Chapter 1.) An alternative is to use logarithmic quantization, where low amplitude samples are represented with more precision than are high amplitude samples. In other words, quantization intervals get progressively larger as the amplitude of the audio samples get larger. The advantage to this method is that you can make finer distinctions between amplitude levels at lower amplitudes, where the human ear is more sensitive to differences. A-law and μ-law encoding are examples of logarithmic quantization methods. The file extensions *.u*, *.al*, *.l8*, and *.l16* can be recognized, respectively, as 8-bit μ-law, 8-bit A-law, 8-bit linear PCM, and 16-bit linear PCM, all assumed to be at a sampling rate of 8000 Hz.

Raw files have nothing but sample values in them. There's no header to indicate the sampling rate, sample size, or type of encoding. If you try to open a *.raw* file in an audio processing program, you'll be asked to provide the necessary information.

Sometimes the file extension of an audio file uniquely identifies the file type, but this isn't always the case. Often it's necessary for the audio file player or decoder to read the header of the file to identify the file type. For example, the *.wav* extension is used for files in a variety of formats: uncompressed, A-law encoded, μ-law encoded, ADPCM encoded, or compressed with any codec. The header of the file specifies exactly which format is being used. WAV files are the standard audio format of Microsoft and IBM.

To facilitate file exchange, a standard format called IFF—the Interchange Format File—has been devised. IFF files are divided into chunks, each with its own header and data. Files with the *.wav* or *.aif* (Apple) extension are versions of the IFF format. They differ in that Microsoft/IBM use little-endian byte order while Macintosh uses big-endian. In little-endian byte ordering, the least significant byte comes first, and in big-endian the most significant byte comes first.

It is sometimes difficult to separate audio file formats from codecs. Codecs are compression/decompression algorithms that can be applied within a variety of file formats. For example, the fact that a file has the *.wav* file extension says nothing about whether or not the file is compressed. The file could be compressed with any one of a number of codecs, as specified in the file's header. However, if a file has the file extension *.mp3* or *.aac*, then you know that it has been compressed with the MP3 or AAC codec, respectively. Some codecs are proprietary (*e.g.*, WMA) and some are open source (*e.g.*, FLAC and Ogg Vorbis). Some provide lossy compression, some lossless.

Representative audio file formats and codecs are listed in Table 5.1. Your choice of audio file format depends on a number of factors. On what platforms do you expect to deliver the audio? What sampling rate and bit depth are appropriate to the type of audio you're

TABLE 5.1 Representative Audio File Formats

File Extension	File Type or Codec	Characteristics
.aac	Advanced Audio Coding	Lossy compression method designed as an improvement of *.mp3* files; used by iPods, cell phones, and portable PlayStations.
.aif	Audio Interchange File Format, Apple's standard wave format	Uncompressed PCM format supporting mono or stereo, 16-bit or 8-bit, and a wide range of sample rates; big-endian byte order.
.au	NeXT/Sun (Java)	Can be uncompressed or can use variants of CCITT μ-law, A-law, or G.721; supports mono or stereo, 16-bit or 8-bit, and a range of sampling rates when uncompressed; often used for distribution on the Internet and for inclusion in Java applications and applets.
.flac	Free Lossless Audio Codec, Xiph.Org Foundation	A free codec providing lossless compression at a ratio of about 1.5 : 1 or 2 : 1; popular for lossless compression of web-based audio; competitive with *.tta*.
.mp3	MPEG-1 Layer 3 audio	High compression rate with good quality; dominant web-based format for many years.
.ogg	Ogg Vorbis, Xiph.Org Foundation	Open, free codec; high compression rate with high fidelity; competitive with *.mp3*.
.raw	Raw files	Raw sample values with no header; when a raw file is opened, you're asked for the sampling rate, resolution, and number of channels.
.rm	RealMedia	Supports streamed audio, which allows you to begin listening to the audio without having to download the entire file first.
.tta	True Audio	A free codec providing lossless compression at a ratio of about 1.5 : 1 or 2 : 1; popular for lossless compression of web-based audio; competitive with *.flac.*
A-law or μ-law *.wav*	CCITT standard G.711 standard formats	8-bit per sample files created from 16-bit per sample using A-law or μ-law encoding, achieving the equivalent of about 13-bit dynamic range (about 78 dB).
DVI/IMA *.wav*	International Multimedia Association version of ADPCM *.wav* format with ADPCM compression	Uses a different (faster) method than Microsoft ADPCM; a good alternative to MPEG with fast decoding and good quality of compressed audio.
Microsoft ADPCM *.wav*	Microsoft	Uses ADPCM; yields 4-bit per channel compressed data for 4 : 1 compression ratio.
Windows PCM *.wav*	Microsoft	Standard Windows *.wav* format for uncompressed PCM audio; supports both mono and stereo at a variety of bit depths and sample rates; follows RIFF (Resource Information File Format) specification, allowing for extra user information to be saved with the file.
.wma or *.asf*	Microsoft Windows Media (audio)	Microsoft proprietary codec; allows you to choose quality settings, including constant bit rate (CBR) vs. variable bit rate (VBR) and lossy vs. lossless compression; competitive with *.aac*; uses *.asf* file extension if encapsulated in Advanced Systems Format; supports streaming audio.

recording? Not all file types offer all the choices you might want in sampling rate and bit depth. Remember to keep the file in an uncompressed form while you're working on it in order to maintain the greatest fidelity possible. But if file size is ultimately an issue, you need to consider what codec to use for the audio's final deliverable form. Will your user have the needed codec for decompression? Is lossy compression sufficient for the desired quality, or do you want lossless compression? All these decisions should become clearer as you work more with audio files and read more about audio file compression later in this chapter.

5.2 DYNAMICS PROCESSING

Dynamics processing is the process of adjusting the dynamic range of an audio selection, either to reduce or to increase the difference between the loudest and softest passages. An increase in amplitude is called ***gain*** or ***boost***. A decrease in amplitude is called ***attenuation*** or, informally, a ***cut***. Dynamics processing can be done at different stages while audio is being prepared, and by a variety of methods. The maximum amplitude can be limited with a hardware device during initial recording, gain can be adjusted manually in real-time with analog dials, and hardware compressors and expanders can be applied after recording. In music production, vocals and instruments can be recorded at different times, each on its own track, and each track can be adjusted dynamically in real-time or after recording. When the tracks are mixed down to a single track, the dynamics of the mix can be adjusted again. Finally, in the mastering process, dynamic range can be adjusted for the purpose of including multiple tracks on a CD and giving the tracks a consistent sound. In summary, audio can be manipulated through hardware or software; the hardware can be analog or digital; the audio can be processed in segments or holistically; and processing can happen in real-time or after recording.

The information in this section is based on digital dynamics processing tools: hard limiting, normalization, compression, and expansion. These tools alter the amplitude of an audio signal and therefore change its dynamics—the difference between the softest and the loudest part of the signal. Limiting sets a maximum amplitude. Normalization finds the maximum amplitude sample in the signal, boosts it to the maximum possible amplitude (or an amplitude chosen by the user), and boosts all other amplitudes proportionately. Dynamic compression decreases the dynamic range of a selection. (This type of compression has nothing to do with file size.) Dynamic expansion increases it.

ASIDE: You should be careful to distinguish among the following: *possible dynamic range* as a function of bit depth in digital audio; *actual dynamic range* of a particular piece of audio; and *perceived loudness* of a piece. The possible dynamic range for a piece of digital audio is determined by the bit depth in which that piece is encoded. In Chapters 1 and 4, we derived a formula that tells us that the possible dynamic range is equal to approximately $6 * n$ dB where n is the number of bits per sample. For CD quality audio, which uses 16-bit samples, this would be 96 dB. However, a given piece of music doesn't necessarily use that full possible dynamic range. The dynamic range of a piece is the difference between its highest amplitude and lowest amplitude sample. The overall perceived loudness of a piece, which is a subjective measurement, is related to the average RMS of the piece. The higher the average RMS, the louder a piece seems to the human ear. (RMS—root-mean-square—is explained in Chapter 4.)

The purpose of adjusting dynamic range is to improve the texture or balance of sound. The texture of music arises in part from its differing amplitude levels. Instruments and voices have their characteristic amplitude or dynamic range. The difference between peak level amplitude and average amplitude of the human voice, for example, is about 10 dB. In a musical composition, instruments and voices can vary in amplitude over time—a flute is played softly in the background, vocals emerge at medium amplitude, and a drum is suddenly struck at high amplitude. Classical music typically has a wide dynamic range. Sections of low amplitude are contrasted with impressive high amplitude sections full of instruments and percussion. You probably are familiar with Beethoven's Fifth Symphony. Think of the contrast between the first eight notes and what follows:

BUM BUM BUM BAH!
BUM BUM BUM BAH!
Then softer . . .

In contrast, "elevator music" or "Muzak" is intentionally produced with a small dynamic range. Its purpose is to lie in the background, pleasantly but almost imperceptibly.

Musicians and music editors have words to describe the character of different pieces that arise from their variance in dynamic range. A piece can sound "punchy," "wimpy," "smooth," "bouncy," "hot," or "crunchy," for example. Audio engineers train their ears to hear subtle nuances in sound and to use their dynamics processing tools to create the effects they want.

Deciding when and how much to compress or expand dynamic range is as much art as science. Compressing the dynamic range is desirable for some types of sound and listening environments and not for others. It's generally a good thing to compress the dynamic range of music intended for radio. You can understand why if you think about the way radio sounds in a car, which is where radio music is often heard. With the background noise of your tires humming on the highway, you don't want music that has big differences between the loudest and softest parts. Otherwise, the soft parts will be drowned out by the background noise. For this reason, radio music is dynamically compressed, and then the amplitude is raised overall. The result is that the sound has a higher average RMS, and overall it is perceived to be louder.

There's a price to be paid for dynamic compression. Some sounds—like percussion instruments or the beginning notes of vocal music—have a fast ***attack time***. The attack time of a sound is the time it takes for the sound to change amplitude. With a fast attack time, the sound reaches high amplitude in a sudden burst, and then it may drop off quickly. Fast-attack percussion sounds like drums or cymbals are called ***transients***. Increasing the perceived loudness of a piece by compressing the dynamic range and then increasing the overall amplitude can leave little ***headroom***—room for transients to stand out with higher amplitude. The entire piece of music may sound louder, but it can lose much of its texture and musicality. Transients give brightness or punchiness to sound, and suppressing them too much can make music sound dull and flat. Allowing the transients to be sufficiently loud without compromising the overall perceived loudness and dynamic range of a piece is one of the challenges of dynamics processing.

While dynamic compression is more common than expansion, expansion has its uses also. Expansion allows more of the potential dynamic range—the range made possible by the bit depth of the audio file—to be used. This can brighten a music selection.

Using downward expansion, it's possible to lower the amplitude of signals below the point where they can be heard. The point below which a digital audio signal is no longer audible is called the ***noise floor***. Say that your audio processing software represents amplitude in dBFS—decibels full scale—where the maximum amplitude of a sample is 0 and the minimum possible amplitude—a function of bit depth—is somewhere between 0 and $-\infty$. For 16-bit audio, the minimum possible amplitude is approximately -96 dBFS. Ideally, this is the noise floor, but in most recording situations there is a certain amount of low amplitude background noise that masks low amplitude sounds. The maximum amplitude of the background noise is the actual noise floor. If you apply downward expansion to an audio selection and you lower some of your audio below the noise floor, you've effectively

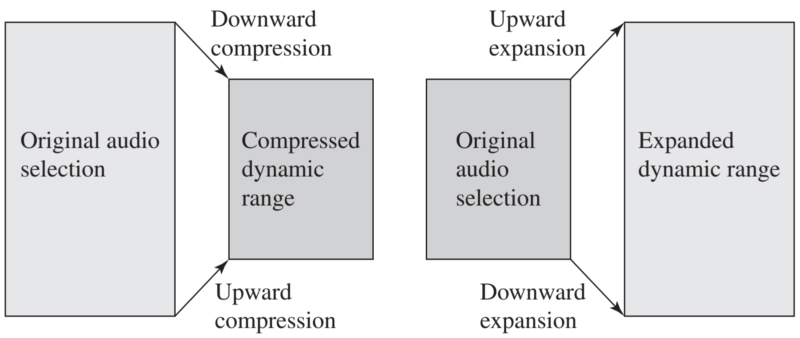


Figure 5.6 Types of dynamic range compression and expansion

lost it. (On the other hand, you could get rid of the noise within the music piece itself by downward expansion, moving the background below the −96 dB noise floor).

To understand how dynamic processing works, let's look more closely at the tools and the mathematics underlying dynamics processing, including hard limiting, normalization, compression, and expansion.

We've talked mostly about dynamic range compression in the examples above, but there are four ways to change dynamic range: downward compression, upward compression, downward expansion, and upward expansion, as illustrated in Figure 5.6. The two most commonly applied processes are downward compression and downward expansion. You have to look at your hardware or software tool to see what types of dynamics processing it can do. Some tools allow you to use these four types of compression and expansion in various combinations with each other.

- ***Downward compression*** lowers the amplitude of signals that are above a designated level, without changing the amplitude of signals below the designated level. It reduces the dynamic range.
- ***Upward compression*** raises the amplitude of signals that are below a designated level without altering the amplitude of signals above the designated level. It reduces the dynamic range.
- ***Upward expansion*** raises the amplitude of signals that are above a designated level, without changing the amplitude of signals below that level. It increases the dynamic range.
- ***Downward expansion*** lowers the amplitude of signals that are below a designated level without changing the amplitude of signals above this level. It increases the dynamic range.

Audio ***limiting***, as the name implies, limits the amplitude of an audio signal to a designated level. Imagine how this might be done in real-time during recording. If ***hard limiting*** is applied, the recording system does not allow sound to be recorded above a given amplitude. Samples above the limit are clipped. ***Clipping*** cuts amplitudes of samples to a given maximum and/or minimum level. If ***soft limiting*** is applied, then audio signals above the designated amplitude are recorded at lower amplitude. Both hard and soft limiting cause some distortion of the waveform.

Normalization is a process which raises the amplitude of audio signal values and thus the perceived loudness of an audio selection. Because normalization operates on an entire audio signal, it has to be applied after the audio has been recorded.

The normalization algorithm proceeds as follows:

- find the highest amplitude sample in the audio selection
- determine the gain needed in the amplitude to raise the highest amplitude to maximum amplitude (0 dBFS by default, or some limit set by the user)
- raise all samples in the selection by this amount

A variation of this algorithm is to normalize the RMS amplitude to a decibel level specified by the user. RMS can give a better measure of the perceived loudness of the audio. In digital audio processing software, predefined settings are sometimes offered with descriptions that are intuitively understandable—for example, "Normalize RMS to −10 dB (speech)."

Often, normalization is used to increase the perceived loudness of a piece after the dynamic range of the piece has been compressed, as described above in the processing or radio music. Normalization can also be applied to a group of audio selections. For example, the different tracks on a CD can be normalized so that they are at basically the same amplitude level. This is part of the mastering process.

Compression and expansion can be represented mathematically by means of a transfer function and graphically by means of the corresponding transfer curve. Digital audio processing programs sometimes give you this graphical view with which you can specify the type of compression or expansion you wish to apply. Alternatively, you may be able to type in values that indicate the compression or expansion ratio. The transfer function maps an input amplitude level to the amplitude level that results from compression or expansion. If you apply no compression or expansion to an audio file, the transfer function graphs as a straight line at a 45° angle, as shown in Figure 5.7. If you choose to raise the amplitude of the entire audio piece by a constant amount, this can also be represented by a straight line of slope 1, but the line crosses the vertical axis at the decibel amount by which all samples are raised. For example, the two transfer functions in Figure 5.7 show a 5 dB increase and a 5 dB decrease in the amplitude of the entire audio piece.

Supplements on dynamics processing:

interactive tutorial

worksheet

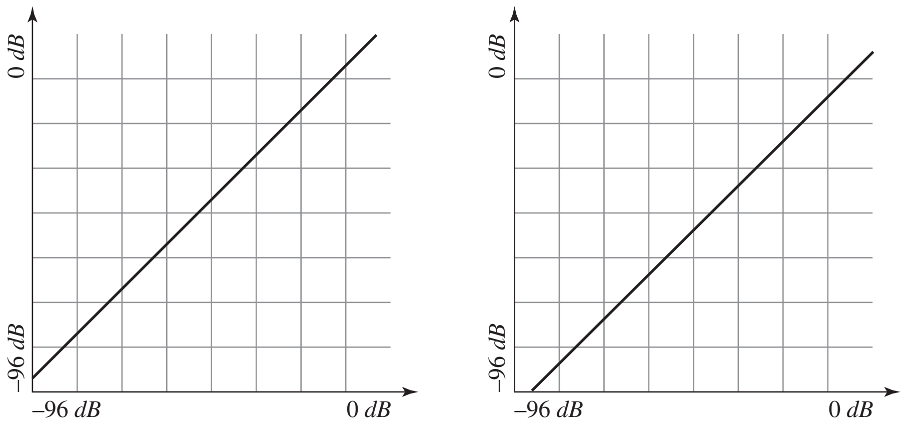
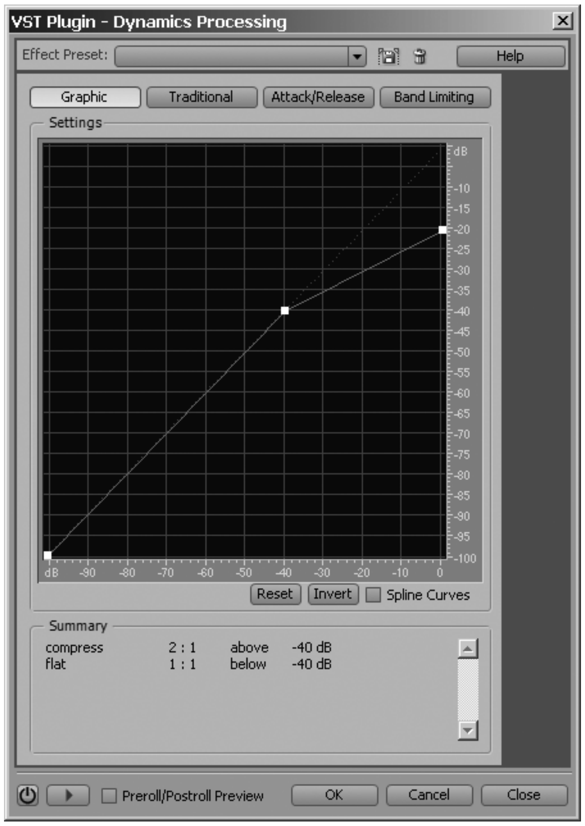


Figure 5.7 Linear transfer functions for 5 dB gain and 5 dB loss—no compression or expansion

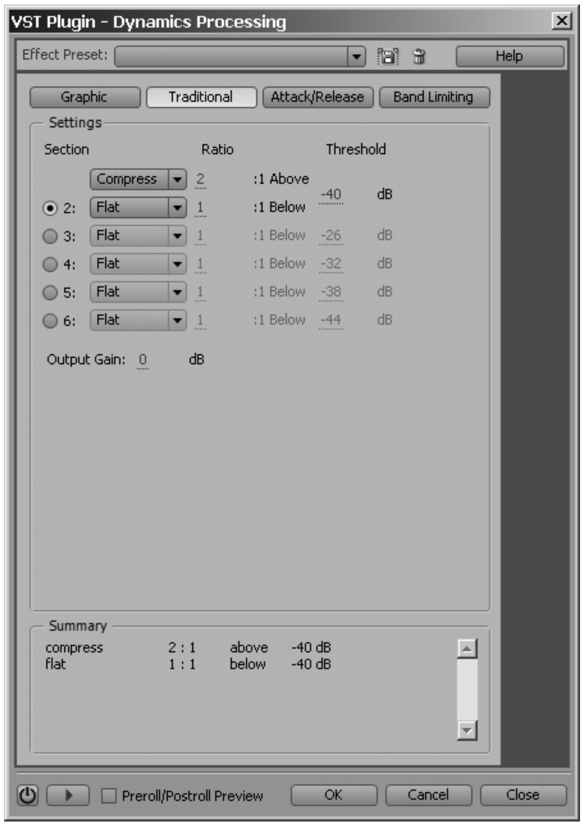
To apply downward compression, you designate a ***threshold***—that is, an amplitude above which you want the amplitude of the audio signal to be lowered. (For upward compression, amplitudes below the threshold would be raised.) Figure 5.8 shows the transfer



Summary

compress	2 : 1	above	-40 dB
flat	1 : 1	below	-40 dB

Figure 5.8 Graph of transfer function for downward compression (from Audition)



Section		Ratio		Threshold	
	Compress	2	:1 Above	-40	dB
● 2:	Flat	1	:1 Below		
○ 3:	Flat	1	:1 Below	-26	dB
○ 4:	Flat	1	:1 Below	-32	dB
○ 5:	Flat	1	:1 Below	-38	dB
○ 6:	Flat	1	:1 Below	-44	dB

Output Gain: 0 dB

Summary

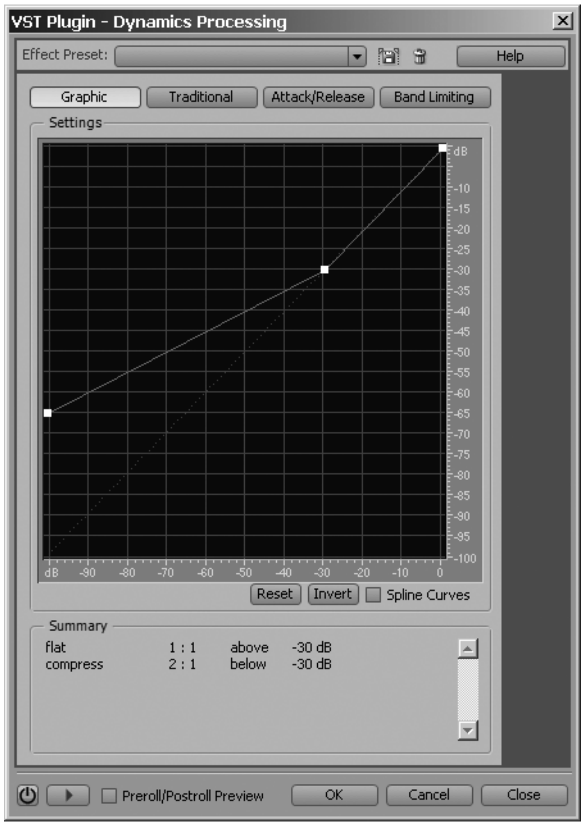
compress	2 : 1	above	-40 dB
flat	1 : 1	below	-40 dB

Figure 5.9 Downward compression, traditional view (from Audition)

function graph corresponding to downward compression where the rate of change of sample values higher than −40 dB is lowered by a 2 : 1 ratio. Figure 5.9 shows the traditional view. Compression above the threshold is typically represented as a ratio $a:b$. If you indicate that you want a compression ratio of $a:b$, then you're saying that, above the threshold, for each a decibels that the signal increases in amplitude, you want it to increase only by b decibels. For example, if you specify a dynamic range compression ratio of 2 : 1 above the threshold, then if the amplitude raises by 1 dB from one sample to the next, it will actually go up (after compression) by only 0.5 dB. Notice that, beginning at an input of −40 dB and continuing to the end, the slope of the line is $b/a = 1/2$.

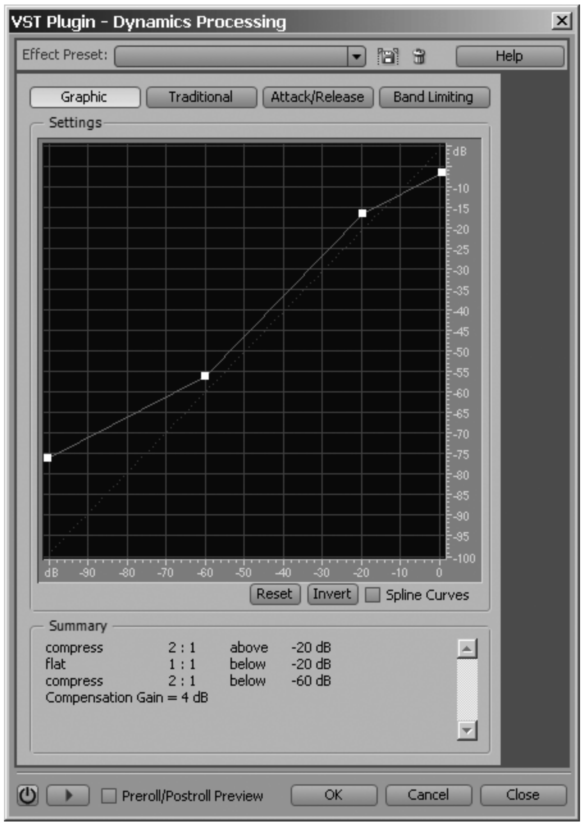
Often, a gain makeup is applied after downward compression. You can see in Figure 5.9 that there is a place to set Output Gain. The Output Gain is set to 0 in the figure. If you set the output gain to a value g dB greater than 0, this means that after the audio selection is compressed, the amplitudes of all samples are increased by g dB. Gain makeup can also be done by means of normalization, as described above. The result is to increase the perceived loudness of the entire piece. However, if the dynamic range has been decreased, the perceived difference between the loud and soft parts is reduced.

Upward compression is accomplished by indicating that you want compression of sample values that are *below* a certain threshold, or decibel limit. For example, Figure 5.10 shows how you indicate that you want samples that are below −30 dB to be compressed by



```
flat        1 : 1   above   -30 dB
compress    2 : 1   below   -30 dB
```

Figure 5.10 Upward compression by 2 : 1 below −30 dB (from Audition)



```
compress    2 : 1   above   -20 dB
flat        1 : 1   below   -20 dB
compress    2 : 1   below   -60 dB
Compensation Gain = 4 dB
```

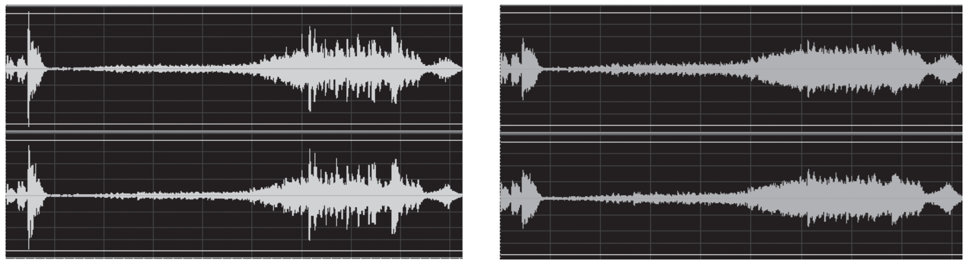
Figure 5.11 Downward and upward compression (from Audition)

a ratio of 2 : 1. If you look at the graph, you can see that this means that sample values will *get larger*. For example, a sample value of −80 dB becomes −54 dB after compression. This may seem counterintuitive at first, since you may think of compressing something as making it smaller. But remember that it is the dynamic range, not the sample values themselves, that you are compressing. If you want to compress the dynamic range by changing values that are below a certain threshold, then you have to make them larger, moving them toward the higher amplitude values at the top. This is what is meant by upward compression.

With some tools, it's possible to achieve both downward and upward compression with one operation. Figure 5.11 shows the graph for downward compression above −20 dB, no compression between −20 and −60 dB, and upward compression below −60 dB. To this, an output gain of 4 dB is added. An audio file before and after such dynamics processing is shown in Figure 5.12. The dynamic range has been reduced by both downward and upward compression.

Sometimes, normalization is used after dynamic range compression. If we downward and upward compress the same audio file and follow this with normalization, we get the audio file pictured in Figure 5.13.

It is also possible to compress the dynamic range at both ends, by making high amplitudes lower and low amplitudes higher. Following is an example of expanding the dynamic range by "squashing" at both the low and high amplitudes. The compression is performed



(a) Uncompressed audio (b) Compressed audio

Figure 5.12 Audio file before and after dynamics processing

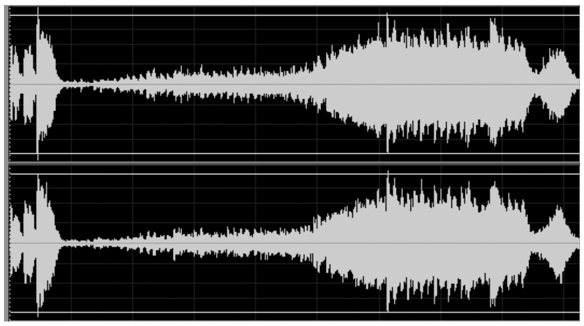
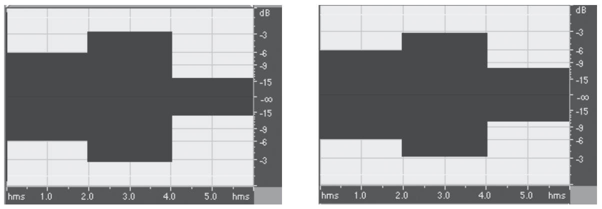


Figure 5.13 Downward and upward compression followed by normalization

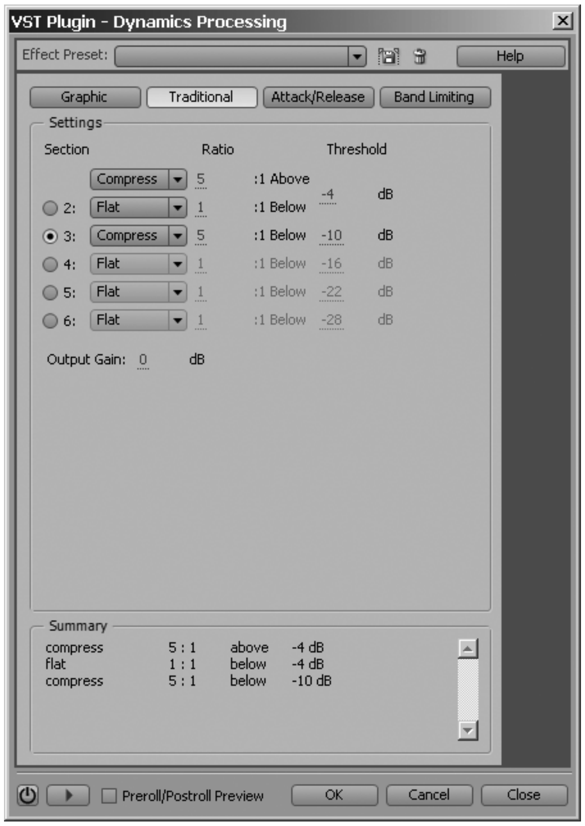
on an audio file that has three single-frequency tones at 440 Hz. The amplitude of the first is −5 dB, the second is −3 dB, and the third is −12 dB. Values above −4 dB are made smaller (downward compression). Values below −10 dB are made larger (upward compression). The settings are given in Figure 5.15. The audio file before and after compression is shown in Figure 5.14a and Figure 5.14b. (The three sine waves appear as solid blocks because the view is too far out to show detail. You can see only the amplitudes of the three waves.)

There's one more part of the compression and expansion process to consider. The ***attack*** of a dynamics processor is defined as the time between the appearance of the first sample value beyond the threshold and the full change in amplitude of samples beyond the threshold. Thus, attack relates to how quickly the dynamics processor *initiates* the compression or



(a) Before compression (b) After compression

Figure 5.14 Audio file (three consecutive sine waves of different amplitudes) before and after dynamic compression



Section		Ratio		Threshold	
	Compress	5	:1 Above	-4	dB
○ 2:	Flat	1	:1 Below		
● 3:	Compress	5	:1 Below	-10	dB
○ 4:	Flat	1	:1 Below	-16	dB
○ 5:	Flat	1	:1 Below	-22	dB
○ 6:	Flat	1	:1 Below	-28	dB

Output Gain: 0 dB

```
compress     5 : 1    above    -4 dB
flat         1 : 1    below    -4 dB
compress     5 : 1    below    -10 dB
```

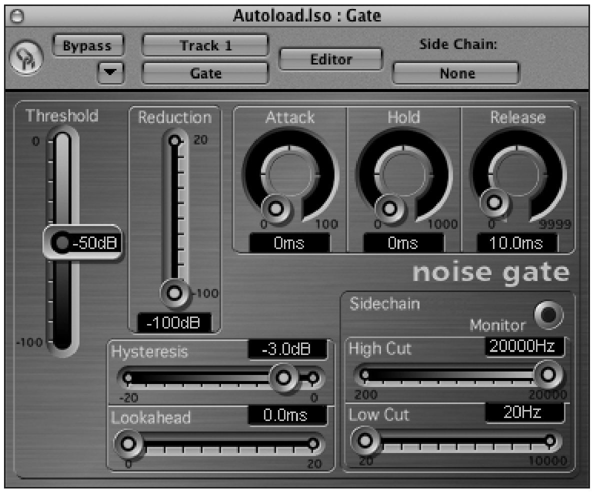
Figure 5.15 Compression of dynamic range at both high and low amplitudes (from Audition)

expansion. When you downward compress the dynamic range, a slower attack time can sometimes be more gradual and natural, preserving transients by not immediately lowering amplitude when the amplitude suddenly, but perhaps only momentarily, goes above the threshold. The ***release*** is the time between the appearance of the first sample that is *not* beyond the threshold (before processing) and the cessation of compression or expansion.

5.3 AUDIO RESTORATION

Background noise arises from a variety of sources during audio recording: the whir of a spinning disk drive, noises in the environment like air conditioners or wind, inadvertent taps on the microphone, tape hiss (if the audio is being copied from an analog tape), or a puff of air when a "p" is spoken too close to the mic, for example. Three basic types of audio restoration are used to alleviate these problems: noise gating, noise reduction, and click and pop removal.

The operation of a ***noise gate*** is very simple. (See Figure 5.16.) A noise gate serves as a block to signals below a given amplitude threshold. When samples fall below the threshold, the gate closes and the samples are not passed through. When the samples rise above

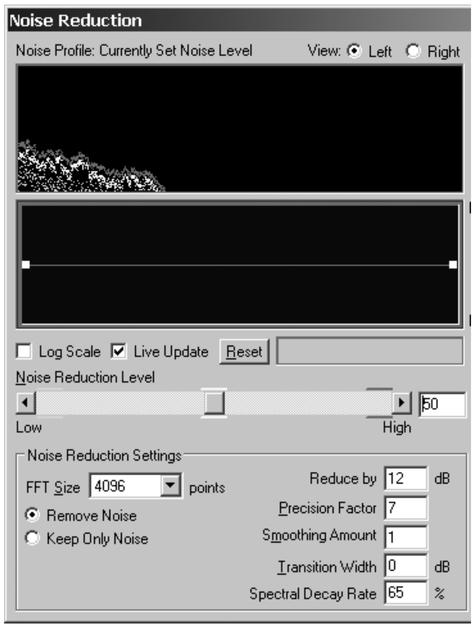


Threshold -50dB
Reduction -100dB
Attack 0ms
Hold 0ms
Release 10.0ms
Hysteresis -3.0dB
Lookahead 0.0ms
High Cut 20000Hz
Low Cut 20Hz

Figure 5.16 Interface to a noise gate (from Logic Pro)

the threshold, the gate opens. Some noise gates allow you to indicate the ***reduction level***, which tells the amplitude to which you want the below-threshold samples to be reduced. Often, reduction is set at the maximum value, completely eliminating the signals below the threshold. The ***attack*** time indicates how quickly you want the gate to open when the signal goes above the threshold. If you want to preserve transients like sudden drum beats, then you would want the attack to be short so that the gate opens quickly for these. A lookahead feature allows the noise gater to look ahead to anticipate a sudden rise in amplitude and open the gate shortly before the rise occurs. Some instruments like strings fade in slowly, and a short attack time doesn't work well in this case. If the attack time is too short, then at the moment the strings go above the threshold, the signal amplitude will rise suddenly. The ***release*** time indicates how quickly you want the gate to close when the signal goes below the threshold. If a musical piece fades gradually, then you want a long release time to model the release of the instruments. Otherwise, the amplitude of the signal will drop suddenly, ruining the decrescendo. Some noise gaters also have a ***hold*** control, indicating the minimum amount of time that the gate must stay open. A ***hysteresis*** control may be available to handle cases where the audio signal hovers around the threshold. If the signal keeps moving back and forth around the threshold, the gate will open and close continuously, creating a kind of ***chatter***, as it is called by audio engineers. The hysteresis control indicates the difference between the value that caused the gate to open (call it n) and the value that will cause it to close again (call it m). If $n - m$ is large enough to contain the fluctuating signal, the noise gate won't cause chatter.

Noise reduction tools can eliminate noise in a digital audio file after the file has been recorded. The first step is to get a profile of the background noise. This can be done by selecting an area that should be silent, but that contains a hum or buzz. The noise reduction tool does a spectral analysis of the selected area in order to determine the frequencies in the noise and their corresponding amplitude levels. Then the entire signal is processed in sections. The frequencies in each section are analyzed and compared to the profile, and if these sections contain frequency components similar to the noise, these



Noise Profile: Currently Set Noise Level

View: [✓ Left] Right

[ ] Log Scale [✓] Live Update

Noise Reduction Level: 50

Noise Reduction Settings

FFT Size: 4096 points

[✓ Remove Noise] Keep Only Noise

Reduce by: 12 dB

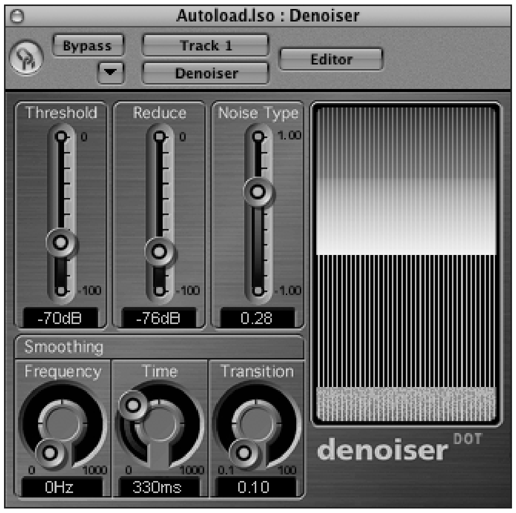
Precision Factor: 7

Smoothing Amount: 1

Transition Width: 0 dB

Spectral Decay Rate: 65 %

Figure 5.17 Interface of a noise reduction tool (from Audition)



Threshold: -70dB

Reduce: -76dB

Noise Type: 0.28

Smoothing

Frequency: 0Hz

Time: 330ms

Transition: 0.10

Figure 5.18 Interface of a noise reduction tool (from Logic Pro)

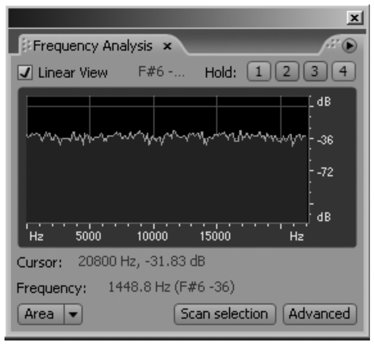
can be eliminated below certain amplitudes. Noise reduction is always done at the risk of changing the character of the sound in unwanted ways. Music is particularly sensitive to changes in its frequency components. A good noise reduction tool can analyze the harmonic complexity of a sound to distinguish between music and noise. A sound segment with more complex harmonic structure is probably music and therefore should not be altered. In any case, it is often necessary to tweak the parameters experimentally to get the best results.

Some noise reduction interfaces, such as the one in Figure 5.17, give a graphical view of the frequency analysis. The graph in the top window is the noise profile. The horizontal axis moves from lower to higher frequencies, while the vertical axis shows amplitude. The original signal is in one color (the data points at the top of the graph), the amount of noise reduction is in a second color (the data points at the next level down in amplitude), and the noise is in a third color (the data points at the bottom of the graph). The main setting is the amount of noise reduction. Some tools allow you to specify how much to "reduce by" (Figure 5.17) and some say exactly the level to "reduce to" (Figure 5.18).

You can see from the interface in Figure 5.17 that the noise profile is done by Fourier analysis. In this interface, the size of the FFT of the noise profiler can be set explicitly. Think about what the profiler is doing—determining the frequencies at which the noise appears, and the corresponding amplitudes of these frequencies. Once the noise profile has been made, the frequency spectrum of the entire audio file can be compared to the noise profile, and sections that match the profile can be eliminated. With a larger FFT size, the noise profile's frequency spectrum is divided into a greater number of frequency components—that is, there is greater frequency resolution. This is a good thing, up to a point, because noise is treated differently for each frequency component. For example, your audio sample may have a −70 dB background hum at 100 Hz. If the frequency resolution isn't fine enough, the noise reducer may have to treat frequencies between 80 and 120 Hz all the same way, perhaps doing more harm than good. On the other hand, it's also possible to set the FFT size too high because there is a tradeoff between frequency resolution and time resolution. The higher the frequency resolution, the lower the time resolution. If the FFT size is set too high, time slurring can occur, manifested in the form of reverberant or echo-like effects.

Noise reducers often allow you to set parameters for smoothing. Time smoothing adjusts the attack and release times for the noise reduction. Frequency smoothing adjusts the extent to which noise that is identified in one frequency band affects amplitude changes in neighboring frequency bands. Transition smoothing sets a range between amplitudes that are considered noise and those that are not considered noise.

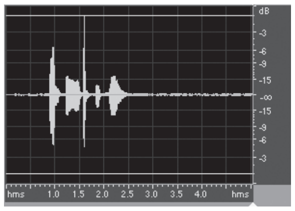
Noise reducers can also be set to look for certain types of noise. ***White noise*** is noise that occurs with equal amplitude (or relatively equal) at all frequencies (Figure 5.19). Another way to say this is that white noise has equal energy in frequency bands of the same size. (Intuitively, you can think of the energy as the area under the curve—the "colored" area—for a frequency band.) ***Pink noise*** has equal energy in octave bands. Recall from Chapter 4 that as you move from one octave to the same note in the next higher octave, you double the frequency. Middle C on the piano has a frequency of about 261.6 Hz, and the next higher C has a frequency of 2 * 261.6, which is about 523 Hz. In pink noise, there is an equal amount of noise in the band from 100 to 200 Hz as in the band from 1000 to 2000 Hz because each band is one octave wide. Human perception is logarithmic in that our sensitivity to the loudness of a signal drops off logarithmically as frequency increases. Thus, pink noise is perceived by humans to have equal loudness at all frequencies. If you know the character of the noise in your audio file, you may be able to set the noise reducer to eliminate this particular type of noise.



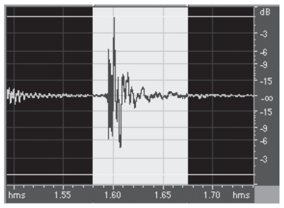
Cursor: 20800 Hz, -31.83 dB
Frequency: 1448.8 Hz (F#6 -36)

Figure 5.19 White noise with energy equally distributed across frequency spectrum (from Audition)

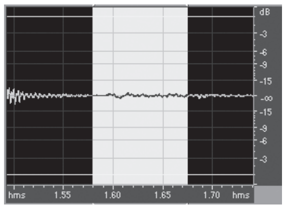
A click or pop eliminator can look at a selected portion of an audio file, detect a sudden amplitude change, and eliminate this change by interpolating the sound wave between the start and end point of the click or pop. Figure 5.20 shows how a waveform can be altered by click removal.



An unwanted click in
an audio file



Zooming in and selecting
the click



The click removed

Figure 5.20 Click removal (from Audition)

5.4 DIGITAL AUDIO FILTERS AND RELATED PROCESSING

5.4.1 Types of Digital Audio Filters

A digital audio filter is a linear system that changes the amplitude or phase of one or more frequency components of an audio signal. Picture a filter as a black box. The sound goes in one end with certain frequency and phase characteristics, and it comes out the other end altered. While an analog filter uses hardware components to operate on the physical properties of a wave, a digital filter—whether it is implemented in hardware or software—uses mathematical algorithms to alter sample values. A digital audio filter can be applied to either analog or digital audio signals. If it is applied to an analog signal, then there is an analog-to-digital converter at the input end and a digital-to-analog converter at the output end.

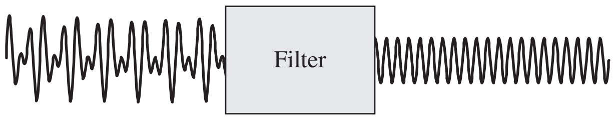


Figure 5.21 Audio filter

Digital audio filters are applied for a variety of reasons. They can be used to separate and analyze frequency components in order to identify the source of sounds. For example, undersea sounds could be detected and analyzed to determine the marine life in the area. Filters can also be used for restoration of a poorly recorded or poorly preserved audio recording, like an old, scratched vinyl record disk that is being digitized. Digital filters are the basis for common digital audio processing tools for equalization, reverb, and special effects.

Digital audio filters can be divided into two categories based on the way they're implemented: ***FIR*** (***finite-impulse response***) and ***IIR*** (***infinite-impulse response***) ***filters***. Mathematically, FIR and IIR filters can be represented as a convolution operation. Let's look at the FIR filter first. The FIR filter is defined as follows:

KEY EQUATION

Let $\boldsymbol{x}(n)$ be a digital audio signal of L samples for $0 \leq n \leq L - 1$. Let $\boldsymbol{y}(n)$ be the audio signal after it has undergone an FIR filtering operation. Let $\boldsymbol{h}(n)$ be a convolution mask operating as an FIR filter where N is the length of the mask. Then an ***FIR filter function*** is defined by

$$\boldsymbol{y}(n) = \boldsymbol{h}(n) \otimes \boldsymbol{x}(n) = \sum_{k=0}^{N-1} \boldsymbol{h}(k)\,\boldsymbol{x}(n-k)$$

where $\boldsymbol{x}(n - k) = 0$ if $n - k < 0$

Equation 5.1

ASIDE: $\boldsymbol{h}(n)$ goes by different names, depending on your source. It can be called the ***convolution mask***, the ***impulse response***, the ***filter***, or the ***convolution kernel***.

Note that in this section, we will use notation that is standard in the literature. Instead of using subscripts for discrete functions to emphasize that they are arrays, as in $\boldsymbol{x}_n$, we use $\boldsymbol{x}(n)$ and $\boldsymbol{y}(n)$. These functions are in the time domain. (To emphasize this, we could call it $\boldsymbol{x}(Tn)$ where T is the interval between time samples, but $\boldsymbol{x}(n)$ captures the same information with T implicit.) $\otimes$ is the convolution operator.

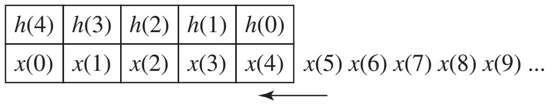
Consider what values convolution produces and, procedurally, how it operates.

$$\boldsymbol{y}(0) = \boldsymbol{h}(0)\boldsymbol{x}(0)$$
$$\boldsymbol{y}(1) = \boldsymbol{h}(0)\boldsymbol{x}(1) + \boldsymbol{h}(1)\boldsymbol{x}(0)$$
$$\boldsymbol{y}(2) = \boldsymbol{h}(0)\boldsymbol{x}(2) + \boldsymbol{h}(1)\boldsymbol{x}(1) + \boldsymbol{h}(2)\boldsymbol{x}(0)$$

In general, for $n \geq N, \boldsymbol{y}(n) = \boldsymbol{h}(0)\boldsymbol{x}(n) + \boldsymbol{h}(1)\boldsymbol{x}(n-1) + \cdots + \boldsymbol{h}(N-1)\ \boldsymbol{x}(n - N + 1)$

We already covered convolution in Chapter 3 as applied to two-dimensional digital images. Convolution works in basically the same way here, as pictured in Figure 5.22. $\boldsymbol{h}(n)$ can be thought of as a convolution mask that is moved across the sample values. The values in the mask serve as multipliers for the corresponding sample values. To compute succeeding values of $\boldsymbol{y}(n)$, the samples are shifted left and the computations are done again. With digital images, we applied a two-dimensional mask. For digital sound, convolution is applied in only one dimension, the time domain. You may notice that the mask is "flipped"

Filter on top, samples on bottom



$h(4)$ $h(3)$ $h(2)$ $h(1)$ $h(0)$
$x(0)$ $x(1)$ $x(2)$ $x(3)$ $x(4)$ $x(5)$ $x(6)$ $x(7)$ $x(8)$ $x(9)$ ...

At time $n = 4$, $y(4)$ is computed.
Filter size is $N = 5$

Figure 5.22 Visualizing the order of coefficients in a filter relative to the samples

relative to the sample values. That is, $\boldsymbol{x}(0)$ is multiplied by $\boldsymbol{h}(N-1)$, and $\boldsymbol{x}(N-1)$ is multiplied by $\boldsymbol{h}(0)$.

Equation 5.1 describes an FIR filter. The essence of the filtering operation is $\boldsymbol{h}(n)$, which is just a vector of multipliers to be applied successively to sample values. The multipliers are usually referred to as coefficients, and the number of coefficients is the ***order of a filter***. Engineers also call the coefficients ***taps*** or ***tap weights***. The central question is this: How do you determine what the coefficients $\boldsymbol{h}(n)$ should be, so that $\boldsymbol{h}(n)$ changes a digital signal in the way that you want? This is the area of digital filter design, an elegant mathematical approach that allows you to create filters to alter the frequency spectrum of a digital signal with a great amount of control. We'll return to this later in the chapter.

Now let's consider IIR filters. To describe an IIR, we need a mask of infinite length, given by the following equation:

KEY EQUATION

Let $\boldsymbol{x}(n)$ be a digital audio signal of L samples for $0 \leq n \leq L-1$. Let $\boldsymbol{y}(n)$ be the audio signal after it has undergone an IIR filtering operation. Let $\boldsymbol{h}(n)$ be a convolution mask operating as an IIR filter where N is the length of the forward filter and M is the length of the feedback filter. Then ***the infinite form of the IIR filter function*** is defined by

$$\boldsymbol{y}(n) = \boldsymbol{h}(n) \otimes \boldsymbol{x}(n) = \sum_{k=0}^{\infty} \boldsymbol{h}(k)\boldsymbol{x}(n-k)$$

where $\boldsymbol{x}(n-k) = 0$ if $n - k < 0$ and k is theoretically infinite.
An equivalent form, called ***the recursive form***, is

$$\boldsymbol{y}(n) = \boldsymbol{h}(n) \otimes \boldsymbol{x}(n) = \sum_{k=0}^{N-1} a_k \boldsymbol{x}(n-k) - \sum_{k=1}^{M} b_k \boldsymbol{y}(n-k)$$

Equation 5.2

Again, we use notation that is standard in the literature.

Notice that $\boldsymbol{y}(n)$ depends on present and past input samples as well as on past outputs. The dependence on past outputs is a kind of feedback. Intuitively, you can understand the sense in which the recursive equation is equivalent to an infinite summation when you consider that once you have the first output, you'll continue infinitely to have more outputs. This is because $\boldsymbol{y}(i)$ is used to create a value $\boldsymbol{y}(j)$ where $j > i$.

The recursive form of Equation 5.2 expands to a convolution of the form

$$\boldsymbol{y}(n) = a_0\boldsymbol{x}(n) + a_1\boldsymbol{x}(n-1) + a_2\boldsymbol{x}(n-2) + \cdots + b_1\boldsymbol{y}(n-1) + b_2\boldsymbol{y}(n-2) + b_3\boldsymbol{y}(n-3) + \cdots$$

If you have coefficients $\boldsymbol{h}(n)$ for an FIR filter or coefficients a_k and b_k for an IIR filter, you can perform a filter as a convolution. The task, then, is to understand how to derive these coefficients based on specifications for the desired filter. We'll see how to do this in later in the chapter.

The main advantage of IIR filters is that they allow you to create a sharp cutoff between frequencies that are filtered out and those that are not. More precisely, FIR filters require larger masks and thus more arithmetic operations to achieve an equivalently sharp cutoff as compared to what could be achieved with an IIR filter. Thus, FIR filters generally require more memory and processing time. A second advantage of IIR filters is that they can be designed from equivalent analog filters. FIR filters do not have analog counterparts. On the other hand, FIR filters have an important advantage in that they can be constrained to have a ***linear phase response***. In a filter with linear phase response, phase shifts for frequency components are proportional to the frequency. Thus, harmonic frequencies are shifted by the same proportions so that harmonic relationships are not distorted. Clearly, linear phase response is important for music. Finally, FIR filters are not as sensitive to noise resulting from low bit depth and round-off error.

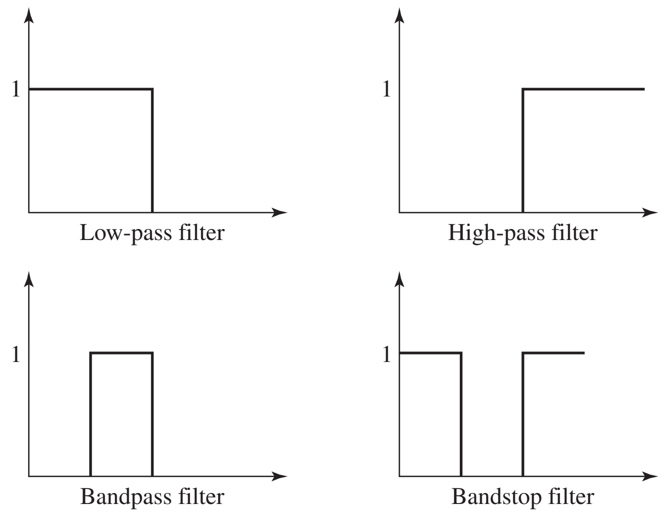
5.4.2 Impulse and Frequency Response

The convolution mask $\boldsymbol{h}(n)$ for an FIR or IIR filter is sometimes referred to as the ***impulse response***. Let's examine the origin of the term.

FIR and IIR filters are examples of linear systems. In the digital audio realm, a system operates on and changes the values of an audio signal, which is just a vector of sample values. A system is a ***linear system*** if it has the properties of homogeneity and additivity. With the property of homogeneity, a change in the input signal's amplitude results in a corresponding change in the output signal's amplitude. With the property of additivity, if the system takes $x(i)$ as input to yield output $y(i)$ and takes $x(j)$ as input to yield output $y(j)$, then $x(i) + x(j)$ yields output $y(i) + y(j)$.

A ***unit impulse***, also called a ***delta function***, is a vector of values where the 0th value in the vector is 1 and all the rest are 0. The FIR and IIR filtering functions as defined in Equation 5.1 and Equation 5.2 are linear systems. The output to either system is $\boldsymbol{h}(n)$ when the delta function is the input. This is why $\boldsymbol{h}(n)$ is called the impulse response; it is the response you get from the FIR or IIR function when an impulse is sent through it. That is, if you do a convolution on a vector of samples that has the shape of a delta function—a 1 followed by 0s—and you use $\boldsymbol{h}(n)$ as the convolution mask, you'll get the mask $\boldsymbol{h}(n)$ as the output.

A counterpart to the impulse response is the ***frequency response***. By convention, if the impulse response is denoted $\boldsymbol{h}(n)$, then the frequency response is denoted $\boldsymbol{H}(z)$. We'll explain why this is so later in the chapter, but for now let's just consider the concept of a frequency response and its corresponding graph. A ***frequency response graph*** describes how a filter acts on an audio signal. In the ideal, it looks like the graphs in Figure 5.23. Each of the four graphs depicts a different filter in the abstract. The low-pass filter retains low-frequency components and eliminates higher frequencies. The high-pass filter does the opposite. The



Horizontal axis:	Frequency component
Vertical axis:	Fraction of frequency component retained in filtered signal

Figure 5.23 Frequency response graphs

bandpass filter retains frequency components across a particular band. The bandstop filter eliminates frequencies in a particular band. All of these graphs are idealized. In reality, it isn't possible to filter out frequency components with perfect accuracy. However, idealized graphs like this can describe the general behavior of a filter.

5.4.3 Filters and Related Tools in Digital Audio Processing Software

The filters and filter-related tools that you'll find in audio processing hardware and software generally go by these names:

- ***band filters***
  - ***low-pass filter***—retains only frequencies below a given level.
  - ***high-pass filter***—retains only frequencies above a given level.
  - ***bandpass filter***—retains only frequencies within a given band.
  - ***bandstop filter***—eliminates all frequencies within a given band.
- ***IIR filters*** (also called ***scientific filters***)—Bessel, Butterworth, Chebyshev1, and Chebyshev2 filters, all four of which are infinite-impulse response filters that can give fine-tuned control over the frequency bands that are permitted to pass through and the way in which phase is affected.
- ***comb filters*—**add delayed versions of a wave to itself, resulting in phase cancellations that can be perceived as echo. Phase cancellations eliminate frequency components when two sine waves that are out-of-phase with each other are summed. Thus, the frequency response has the shape of a comb (Figure 5.24).
- ***convolution filters*—**FIR filters that can be used to add an acoustical environment to a sound file—for example, mimicking the reverberations of a concert hall.

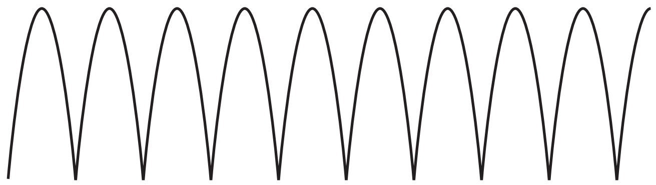


Figure 5.24 Frequency response of a comb filter

- ***graphic equalizer***—gives a graphical view that allows you to adjust the gain of frequencies within an array of bands.
- ***parametric equalizer***—similar to a graphic equalizer but with control over the width of frequency bands relative to their location.
- ***crossover***—splits an input signal into several output signals, each within a certain frequency range, so that each output signal can be directed to a speaker that handles that range of frequencies best.

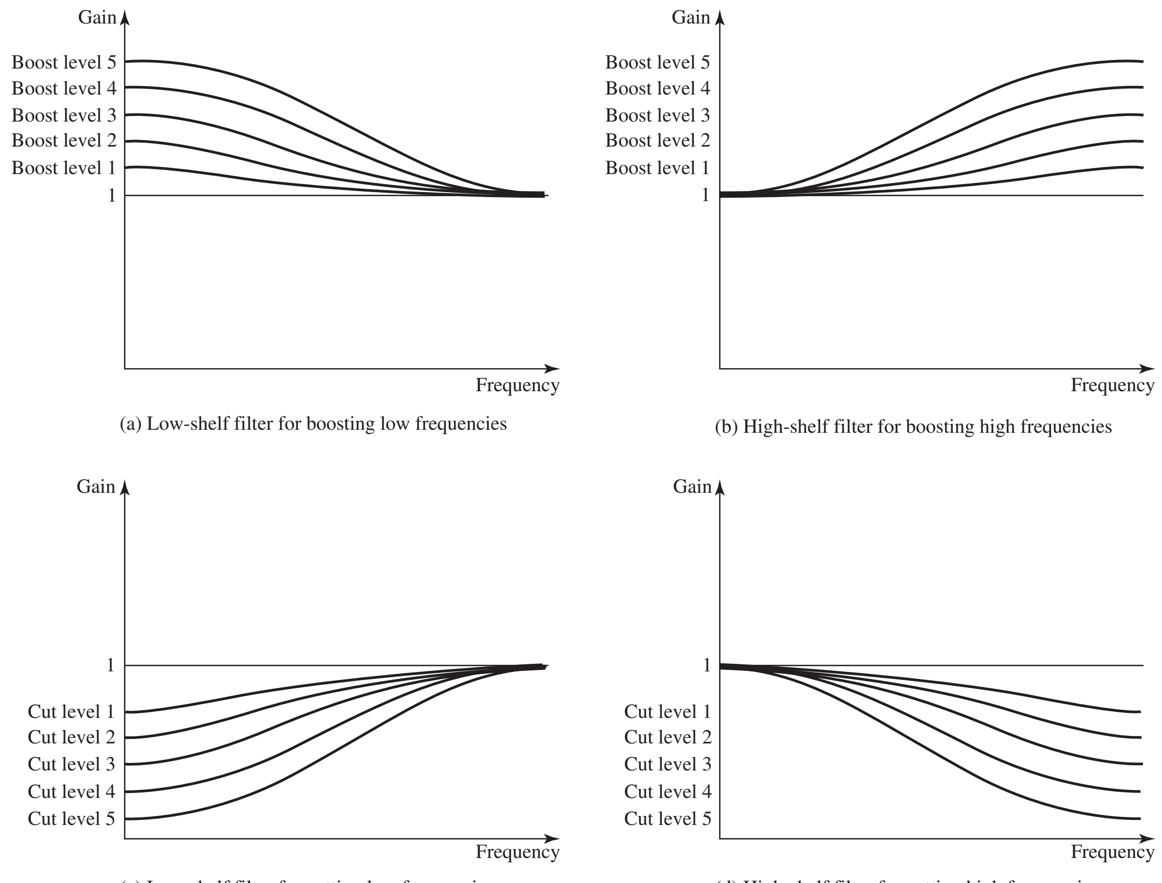
We'll look at some of these tools more closely in the context of digital processing software, and then return to the mathematics that makes them work.

5.4.4 Equalization

Equalization (***EQ***) is the process of selectively boosting or cutting certain frequency components in an audio signal. In digital audio processing, EQ can be used during recording to balance instruments, voices, and sound effects; it can be used in post-processing to restore a poorly recorded signal, apply special effects, or achieve a balance of frequencies that suits the purpose of the audio; or it can be applied when audio is played, allowing frequencies to be adjusted to the ear of the listener.

EQ is a powerful and useful audio processing tool, but it must be applied carefully. When you change the relative amplitudes of frequency components of an audio piece, you run the risk of making it sound artificial. The environment in which music is recorded puts an acoustic signature on the music, adding resonant frequencies to the mix. The frequency spectra of instruments and voices are also complex. Each instrument and voice has its own timbre, characterized by its frequency range and overtones. Separating the frequency components of different instruments and voices is next to impossible. You may think that by lowering the high-frequency components you're only affecting the flutes, but really you may also be affecting the overtones of the oboes. The point is that generally, it's better to get the best quality, truest sound when you record it rather than relying on EQ to "fix it in post-processing." Nevertheless, there are times when a good equalizer is an invaluable tool, one that is frequently used by audio engineers.

You've probably already used EQ yourself. The bass/treble controls on a car radio are a simple kind of equalizer. These ***tone controls***, as they are sometimes called, allow you to adjust the amplitude in two broad bands of frequencies. The bass band is approximately 20 to 200 Hz while the treble is approximately 4 to 20 kHz. Figure 5.25 shows four ***shelving filters***. Frequency is on the horizontal axis and gain is on the vertical axis. The gain level marked with a 1 indicates the level at which the filter makes no change to the input signal. The other lines indicate five levels to which you can boost or cut frequencies, like five discrete levels you might have for both bass and treble on your car's tone controls. A negative gain cuts the amplitude of a frequency. A shelving filter for cutting or boosting low frequencies is called a ***low-shelf filter***. A ***high-shelf filter*** boosts or cuts high frequencies. Shelving filters are similar to low- and high-pass filters except that they boost or cut frequencies up to a certain ***cutoff***



(a) Low-shelf filter for boosting low frequencies

(b) High-shelf filter for boosting high frequencies

(c) Low-shelf filter for cutting low frequencies

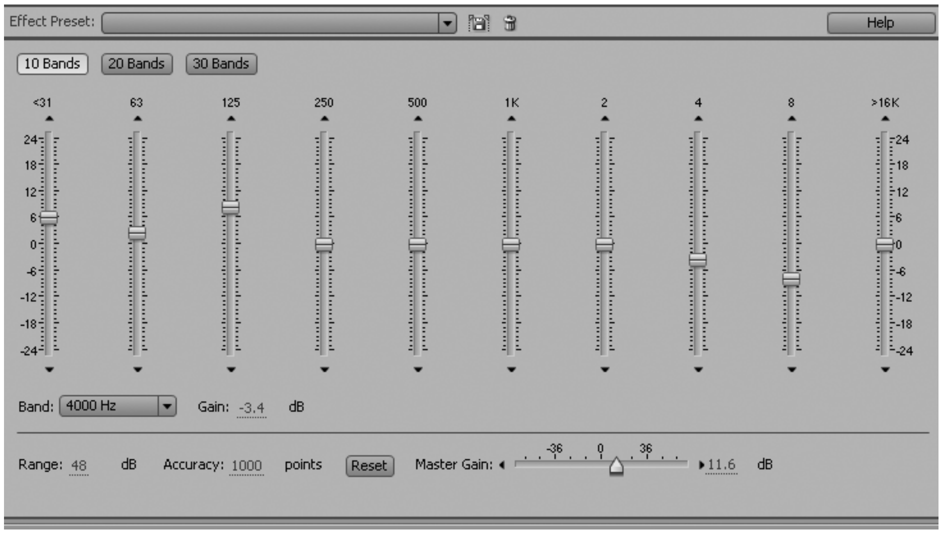
(d) High-shelf filter for cutting high frequencies

Figure 5.25 Shelving filters

frequency (or starting at a cutoff frequency) after which (or before which) they allow the frequencies to pass through unchanged. Low- and high-pass filters, in contrast, completely block frequencies lower or higher than a given limit, as we'll see later in this chapter.

Consider the low-shelf filter for boosting low frequencies in Figure 5.25a. For any of the five boost levels, as frequencies increase, the amount of gain decreases such that higher frequencies, above the cutoff level, are not increased at all.

You may also be familiar with more fine-tuned control of EQ in the form of a ***graphic equalizer*** on your home stereo. A graphic equalizer divides the frequency spectrum into bands and allows you to control the amplitude for these bands individually, usually with sliders. Digital audio processing tools have their equivalent of graphic equalizers, and with an interface that looks very much like the interface to the analog graphic equalizer on your home stereo (Figure 5.26). With graphic equalizers, you can select the number of frequency bands. Typically, there are ten frequency bands that are proportionately divided by octaves rather than by frequency levels in Hz. Recall that if two notes (*i.e.*, frequencies) are an octave apart, then the higher note has twice the frequency of the lower one. Thus, the starting point of each band of the graphic equalizer in Figure 5.26 is twice that of the previous band. If you ask for 20 bands, then the bands are separated by half an octave and thus are spaced by a factor of $2^{\frac{1}{2}} = \sqrt{2}$. If the first band starts at 31.5, then the next starts at $31.5 * \sqrt{2} \approx 44$ Hz, the next starts at $44 * \sqrt{2} \approx 63$ Hz, and so forth. If you divide the



Band: 4000 Hz Gain: -3.4 dB

Range: 48 dB Accuracy: 1000 points Master Gain: 11.6 dB

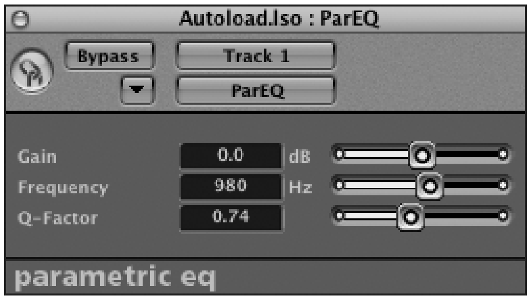
Figure 5.26 Graphic equalizer (from Audition)

spectrum into 30 bands, the bands are separated by a third of an octave and thus are spaced by a factor of $2^{\frac{1}{3}} = \sqrt[3]{2} \approx 1.26$.

Figure 5.26 shows the interface to a graphic equalizer in an audio processing program. The Accuracy setting relates to the length of the filter used to achieve EQ. If you're equalizing only high frequencies, you can use a smaller Accuracy number. To adjust low frequencies, you need a larger Accuracy number. A larger filter has more terms in the filtering equation and thus yields better frequency resolution. This is analogous to the significance of window size in the discrete Fourier transform, as discussed in Chapter 4. The bigger the window size, the more frequency components are computed by the DFT. Greater frequency resolution is necessary when you're working with the low frequency bands because these bands are narrower than the high frequency ones. The cost of a larger filter, however, is increased processing time.

A ***parametric EQ*** is more flexible for equalization in that it allows you to focus on individual bands of frequencies. Rather than having a number of fixed bandwidths, as in the graphic equalizer, with the parametric equalizer you can set the center point of a band and its bandwidth and adjust the frequencies in this band as you desire. An example of a simple digital parametric equalizer interface is shown in Figure 5.27. This EQ operates on only one band at a time.

Parametric EQs are based on bandpass and bandstop filters. A bandpass filter allows frequencies in a certain band to pass through and filters out frequencies below and above the band.



Gain 0.0 dB
Frequency 980 Hz
Q-Factor 0.74

Figure 5.27 Parametric equalizer (from Logic Pro)

Ideally, the unwanted frequencies would be filtered out entirely, but in reality this isn't possible. Thus, the frequency response graph for a bandpass filter looks more like the bell curve in Figure 5.28. This type of filter is sometimes called a ***peaking filter***. If you set the gain to a positive number, the peak is pointed upward and you create a bandpass filter around your central frequency. If you set the gain to a negative number, the peak is pointed downward and you create a bandstop filter. A very narrow bandstop filter is also called a ***notch filter***. The ***Q-factor*** (also called ***quality factor*** or simply ***Q***) determines how steep and wide the peaking curve is. The higher the Q-factor, the higher the peak in relation to the width of the frequency band.

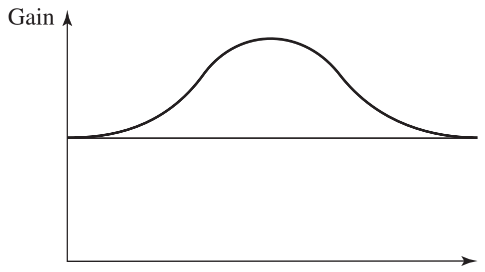


Figure 5.28 Peaking filter

Q-factor is a term adopted from physics and electronics. In physical and electrical systems, Q-factor measures the rate at which a vibrating system dissipates its energy, a process called ***damping***. More precisely, it is the number of cycles required for the energy to fall off by a factor of 535. (The rate 535 is actually $e^{2\pi}$, a number chosen because it simplified related computations.) The Q-factor of an inductor is the ratio of its inductance to its resistance at a given frequency, which is a measure of its efficiency. Q-factors are also used in acoustics to describe how much a surface resonates. A surface with a high Q-factor resonates more than one with a low Q-factor. Instruments and sound speakers have Q-factors. A bell is an example of a high-Q system, since it resonates for a long time after it is struck. Although a high Q-factor seems to imply better quality, the right Q-factor depends on the situation. If your speakers' Q-factor is too high, they may prolong the normal decay of the instruments, or "ring" if a signal suddenly stops, making the music sound artificial or distorted.

A graph can be drawn to depict how a system resonates, with frequency on the horizontal axis and energy on the vertical axis. The same type of graph is used to depict a peaking filter. It is essentially a frequency response graph, depicting which bands of frequencies are filtered out or left in, except that it plots energy (rather than amplitude) against frequency. This is shown in Figure 5.29. The peaking filter graph is parameterized by its Q-factor, a high Q-factor corresponding to a steep peak. Formally, Q-factor can be defined as follows:

KEY EQUATION

Given the graph of a peaking filter, let f_{width} be the width of the peak measured at a point that is $\frac{1}{2}$ times the peak's height, and let f_{center} be the frequency at the geometric center of the peak, both in Hz. Then the Q-factor, Q, is defined as

$$Q = \frac{f_{center}}{f_{width}}$$

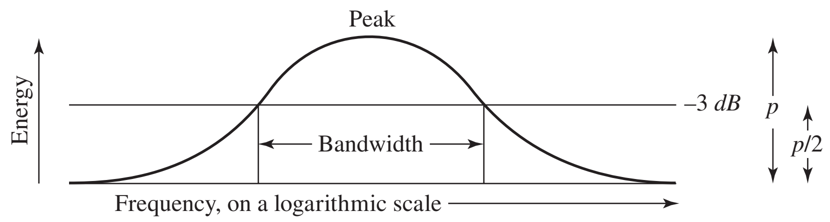


Figure 5.29 Resonance graph of energy plotted against frequency

In Figure 5.29, frequency is shown in a logarithmic scale. On this graph, the center frequency f_{center} is in the geometric center of the peak, and the bandwidth f_{width} is measured at a point that is half the peak height, called the ***−3 dB point***. Note that the geometric center is $(\text{min*max})^{\frac{1}{2}}$ rather than $\left(\frac{\text{min} + \text{max}}{2}\right)$. On a frequency response graph of the filter that plots amplitude (rather than energy) against frequency, the bandwidth would be measured at a point that is $\frac{1}{\sqrt{2}}$ times the peak height.

When defined in terms of octaves, the definition of Q becomes the following:

KEY EQUATION

Let n be the bandwidth of a peaking filter in octaves. Then the filter's Q-factor, Q, is defined as

$$Q = \frac{\sqrt{2^n}}{2^n - 1}$$

Given the Q-factor, you can compute the bandwidth of the filter.

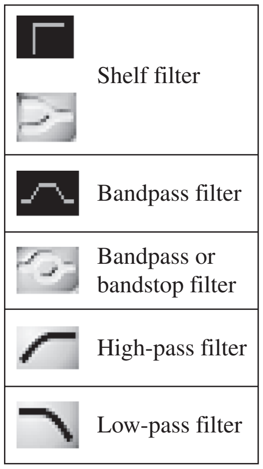
KEY EQUATION

Let Q be the Q-factor of a peaking filter. Then the bandwidth of the filter, n, is given by

$$n = 2\log_2\left(\frac{1}{2Q} + \sqrt{\left(\frac{1}{2Q}\right)^2 + 1}\right)$$

When you use a parametric EQ, you may not need the equations above to compute a precise bandwidth. The main point to realize is that the higher the Q, the steeper the peak of the band. The parametric EQ pictured in Figure 5.27 allows Q to be set in the range of 0.10 to 10, the former being a fairly flat peak and the latter a steep one. Some EQ interfaces actually show you the graph and allow you to "pull" on the top of the curve to adjust Q, or you can type in numbers or control them with a slider.

Different types of filters—low-pass, high-pass, shelving, and bandpass—can be grouped into one EQ tool, sometimes called a ***paragraphic equalizer***. Figure 5.31, Figure 5.32,



	Shelf filter
	Bandpass filter
	Bandpass or bandstop filter
	High-pass filter
	Low-pass filter

Figure 5.30 Icons for filters

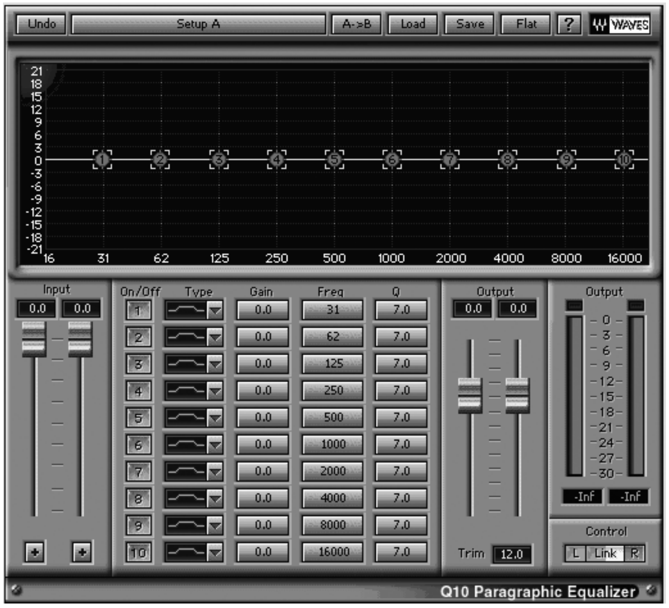


Figure 5.31 A paragraphic equalizer (from Waves Native Power Pack)

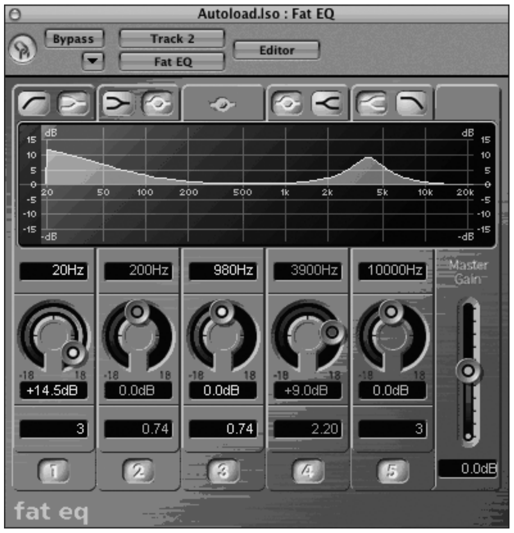
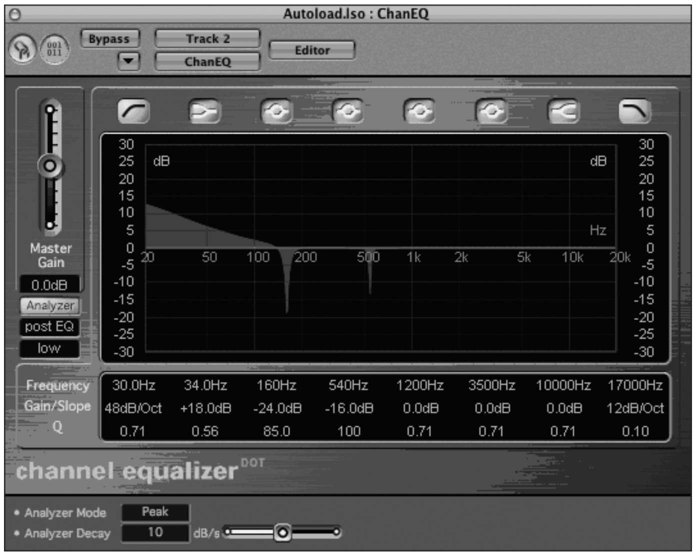


Figure 5.32 An EQ that combines high-pass, bandpass/stop, shelving, and low-pass filters (from Logic Pro)

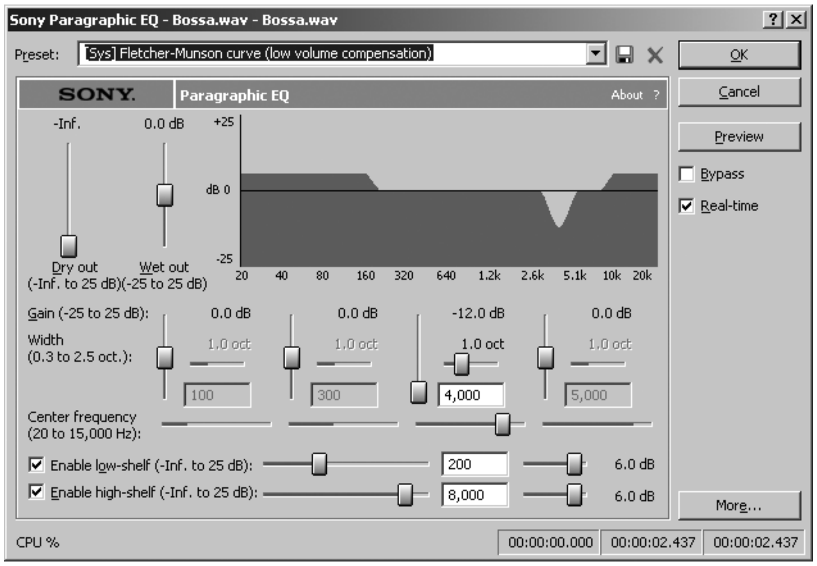
Figure 5.33, and Figure 5.34 show four paragraphic EQs. Note that frequency is on a logarithmic scales in these figures. Common icons for filters that are used in audio processing software are shown in Figure 5.30.

When different filters are applied to different frequency bands, as is the case with a paragraphic EQ, the filters are applied in parallel rather than serially. The audio signal is input



Frequency	30.0Hz	34.0Hz	160Hz	540Hz	1200Hz	3500Hz	10000Hz	17000Hz
Gain/Slope	48dB/Oct	+18.0dB	-24.0dB	-16.0dB	0.0dB	0.0dB	0.0dB	12dB/Oct
Q	0.71	0.56	85.0	100	0.71	0.71	0.71	0.10

Figure 5.33 Another EQ that combines high-pass, bandpass/stop, shelving, and low-pass filters (from Logic Pro)



Preset: [Sys] Fletcher-Munson curve (low volume compensation)

Dry out (-Inf. to 25 dB): -Inf.
Wet out (-25 to 25 dB): 0.0 dB

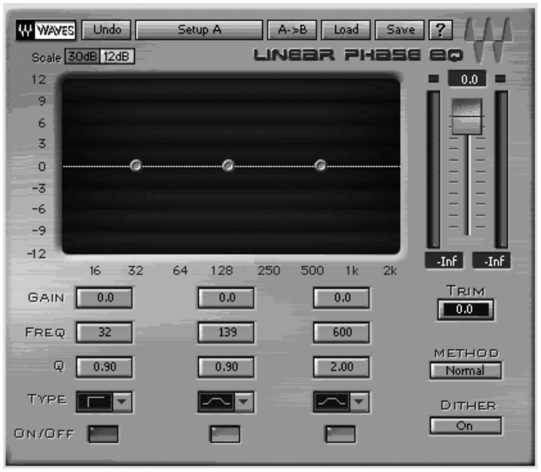
Gain (-25 to 25 dB):	0.0 dB	0.0 dB	-12.0 dB	0.0 dB
Width (0.3 to 2.5 oct.):	1.0 oct	1.0 oct	1.0 oct	1.0 oct
Center frequency (20 to 15,000 Hz):	100	300	4,000	5,000

☑ Enable low-shelf (-Inf. to 25 dB): 200 6.0 dB
☑ Enable high-shelf (-Inf. to 25 dB): 8,000 6.0 dB

00:00:00.000 00:00:02.437 00:00:02.437

Figure 5.34 A paragraphic EQ with wet and dry control (from Sound Forge)

separately into each of the filters, and the outputs of the filters are combined after processing. A parallel rather than serial configuration is preferable because any phase distortions introduced by a single filter will be compounded by a subsequent phase distortion when filters are applied serially. If phase linearity is important, a linear phase EQ can be applied. Linear phase filters will not distort the harmonic structure of music because if there is a phase shift at a certain frequency, the harmonic frequencies will be shifted by the same amount. A linear phase filter is pictured in Figure 5.35.



GAIN 0.0 0.0 0.0
FREQ 32 139 600
Q 0.90 0.90 2.00
TRIM 0.0
METHOD Normal
DITHER On

Figure 5.35 A linear phase EQ (from Waves Native Power Pack)

Some EQs allow you to set the master gain, which boosts or cuts the total audio signal output after the EQ processing. You may also be able to control the amount of wet and dry signal added to the final output. A ***wet signal*** is an audio signal that has undergone processing. A ***dry signal*** is unchanged. A copy of the dry signal can be retained, and after the signal has been processed and made wet, varying amounts of wet and dry can be combined for the final audio output. A paragraphic EQ with wet and dry controls is shown in Figure 5.34.

5.4.5 Comb Filters, Delay, Reverb, and Convolution Filters

A type of filter of particular interest is the ***comb filter***. A comb filter is created when an earlier audio sample is added to a later one, causing certain resonant frequencies to be eliminated, as if they are "combed out" at regular frequency intervals. Comb filters can be either nonrecursive FIR filters or recursive IIR filters. A simple nonrecursive comb filter is described by $\boldsymbol{y}(n) = \boldsymbol{x}(n) + g\boldsymbol{x}(n - m)$ where m is the amount of the delay in numbers of samples and $|g| \leq 1$. A simple recursive comb filter is described by $\boldsymbol{y}(n) = \boldsymbol{x}(n) + g\boldsymbol{y}(n - m)$ where m is the amount of the delay in numbers of samples and $|g| < 1$. This filter is recursive in that you feed back the output $g\boldsymbol{y}(n - m)$ into a later output.

You should be able to get an intuitive understanding of what these filters do. In a nonrecursive filter, a fraction of an earlier sample is added to a later one, but there's no feedback. The result would be simple repetition of part of the sound. The distance from the original sound is controlled by m. But what if the result of a previous computation is fed back into a later one, as is the case with a recursive comb filter? A single earlier sound would continue to affect later sounds, but with decreasing intensity (because each time its effect is scaled down again by the coefficient g, which we assume is less than 1). That would sound more like an echo.

Filters are often represented by flow diagrams. The flow diagram for the nonrecursive and recursive filters are shown in Figure 5.36. z^{-m} symbolizes a delay of m samples in the term $g\boldsymbol{x}(n - m)$ or $g\boldsymbol{y}(n - m)$. (You'll understand this notation even better after you read the section on z-transforms.)

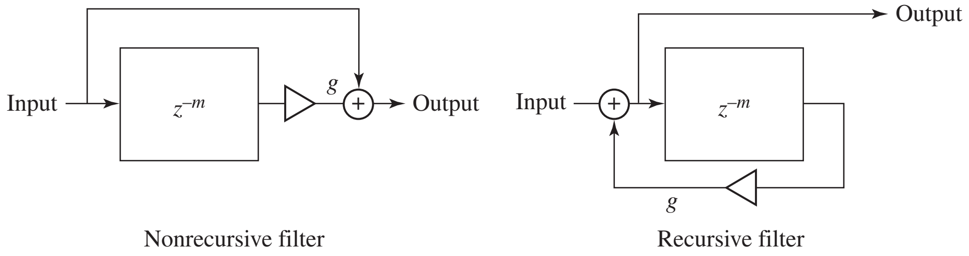


Figure 5.36 Flow diagrams of simple nonrecursive and recursive filters

The comb filters given above can be used to create simple delay effects. Changing the coefficients, changing the delay *m*, and cascading filters can create different effects similar to real-world echoes and reverberation. The phase characteristics of these filters, however, are not very natural, and thus a single filter like those above isn't the best for creating realistic echo and reverberation. What is unnatural is that the phases of all the frequencies aren't changed sufficiently relative to each other. This isn't how real audio echoes and reverberations happen.

Imagine that you stand in the middle of a room and play a note on a musical instrument. You hear the initial sound, and you also hear reflections of the sound as the sound waves move through space, bounce off the walls and objects in the room, and repeatedly come back to your ears until the reflections finally lose all their energy and die away. The way in which the sound waves reflect depends on many factors—the size of the room, the materials with which it's built, the number of objects, the heat and humidity, and so forth. Furthermore, different frequencies are reflected or absorbed differently, low frequencies being absorbed less easily than high. (Have you ever sat beside another car at a traffic light and heard only the booming bass of the other person's stereo?)

Another type of filter, the ***all-pass filter***, is often used as a building block of reverberation effects. Adding the all-pass filter to a chain or bank of comb filters helps to make the reflections sound like they're arriving at different times. The all-pass filter doesn't change the frequencies of the wave on which it acts. It only changes the phase, and it does so in a way that "smears" the harmonic relationships more realistically. A simple all-pass filter is defined by $\boldsymbol{y}(n) = -g\boldsymbol{x}(n) + \boldsymbol{x}(n - m) + g\boldsymbol{y}(n - m)$ where m is the number of samples in the delay and $|g| < 1$. The flow diagram is shown in Figure 5.37.

Different echo and reverberation effects can be created by combining the above filters in various ways—in series, in parallel, or nested. One of the first designs that combined filters in this way was made by Manfred Schroeder. Filter designs for reverberation have gone beyond this simple configuration, but the diagram in Figure 5.38 serves to illustrate how the simple filters can be put together for more realistic sound than is possible with a single filter.

Digital audio processing programs offer tools for delay, multi-tap delay, chorus, flange, and reverberation. More sophisticated algorithms can be found in audio effects plug-ins

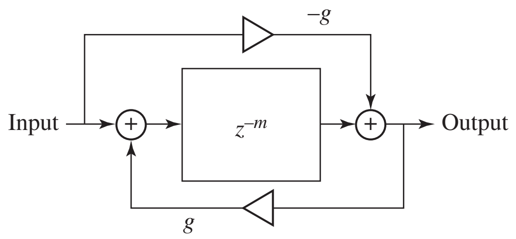


Figure 5.37 Flow diagram for an all-pass filter

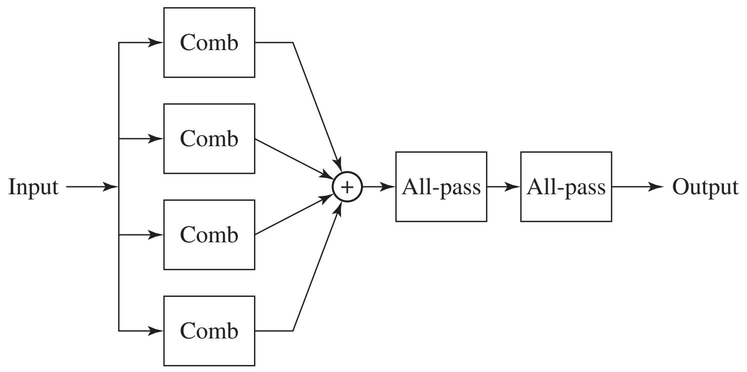
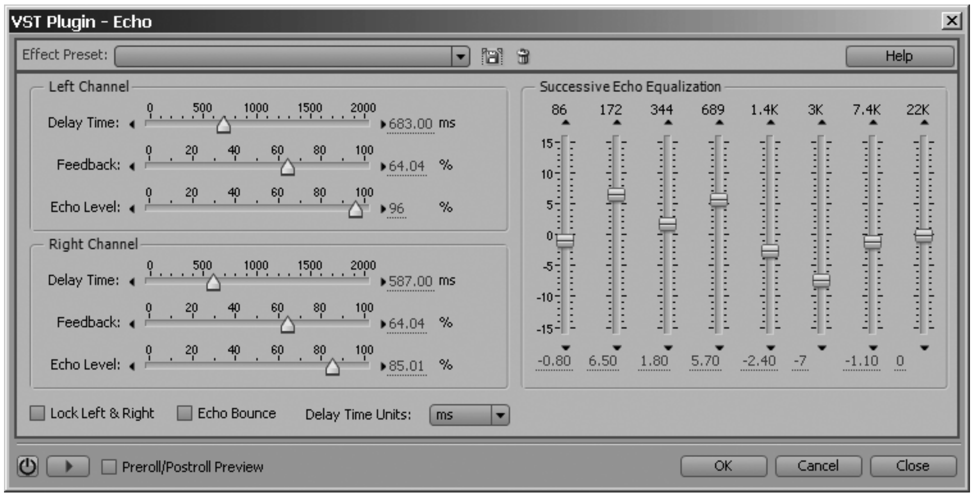


Figure 5.38 Schroeder's Filter

that you can add to your basic software. ***Reverb*** (short for ***reverberation***) is one of the most useful effects for giving vibrance and color to sound. It can simulate not only how music and voice sound in different listening spaces, but also how it is produced by particular instruments, microphones, loudspeakers, and so forth. Reverb can also enhance the timbre of instruments and voices, making a recording that is otherwise flat and dull sound interesting and immediate. Let's look at how these effects are presented to you in audio processing software. The distinction between the different delay-based effects is not a strict one, and the methods of implementation are mostly transparent to the user, but the following discussion will give you an idea of what to expect.

Some audio processing programs make no distinction between delay and echo, while others have separate tools for them. If the features are separated, then the ***delay effect*** is less realistic, possibly implemented with nonrecursive delay units that create simple repetition of the sound rather than trailing echoes. From the interface in an audio processing program, you're able to set the delay time and the amount of wet and dry signal to be added together. The ***multi-tap delay effect*** compounds the effects of multiple delays, a ***tap*** being a single delay. When an echo effect is considered separate from delay, ***echo effect*** is more realistic in that the spectrum of the signal is changed. That is, a special filter may be used so that the


VST Plugin - Echo

Effect Preset:

Left Channel
Delay Time: 683.00 ms
Feedback: 64.04 %
Echo Level: 96 %

Right Channel
Delay Time: 587.00 ms
Feedback: 64.04 %
Echo Level: 85.01 %

Successive Echo Equalization

86	172	344	689	1.4K	3K	7.4K	22K
-0.80	6.50	1.80	5.70	-2.40	-7	-1.10	0

Lock Left & Right
Echo Bounce
Delay Time Units: ms
Preroll/Postroll Preview

Figure 5.39 Echo effect (from Audition)

phases of echoed sounds are changed to simulate the way different frequencies are reflected at different rates. The echo effect uses feedback, and thus interfaces to echo tools generally allow you to control this parameter.

The ***chorus effect*** takes a single audio signal—presumably a voice—and makes it sound like multiple voices singing or speaking in unison. You can choose how many voices you want, and the chorus effect delays the different copies of the signal one from the other. This isn't enough, however, for a realistic effect. When people sing together, they sing neither in perfect unison nor at precisely the same pitch. Thus, the frequencies of the different voices are also modulated according to your settings. Too much modulation makes the chorus sound off-pitch as a group, but the right amount makes it sound like a real chorus.

The ***flange effect*** uses a comb filter with a variable delay time that changes while the music is played, varying between about 0 and 20 milliseconds. You can picture the teeth of the comb filter moving closer together and then farther apart as the delay time changes. Rather than creating an echo, the flange effect continuously boosts some frequencies, cuts some frequencies, and filters others out completely, but which frequencies are affected which way changes continuously over time, which is the distinguishing characteristic of the flange effect. The resulting sound is described variously as "swooshy," "warbly," "wah wah" and so forth, depending on the audio clip to which the effect is applied and the parameters that are set. Parameters that a flange tool will probably allow you to set include the initial and final delay time and the amount of feedback.

Reverb can alter an audio signal so that it sounds like it comes from a particular acoustical space. Let's consider how reverb can be used to simulate the way sound travels in a room, as depicted in Figure 5.40. Imagine that a sound is emitted in a room—an instantaneous impulse. The line marked *direct sound* indicates the moment when the sound of the impulse first reaches the listener's ears, having traveled directly and with no reflections. In the meantime, the sound wave also propagates toward the walls, ceilings, and objects in the room and is reflected back. The moments when these first reflections reach the listener's ears are labeled ***first-order reflections***. When the sound reflects off surfaces, it doesn't reflect perfectly. It's not like a billiard ball striking the side of a billiard table, where the angle of incidence equals the angle of reflection. Some of the sound is diffused, meaning that a wave reflects back at multiple angles. The first-order reflections then strike surfaces again, and these are the ***secondary reflections***. Because of the diffusion of the sound, there are even more reflections at a higher

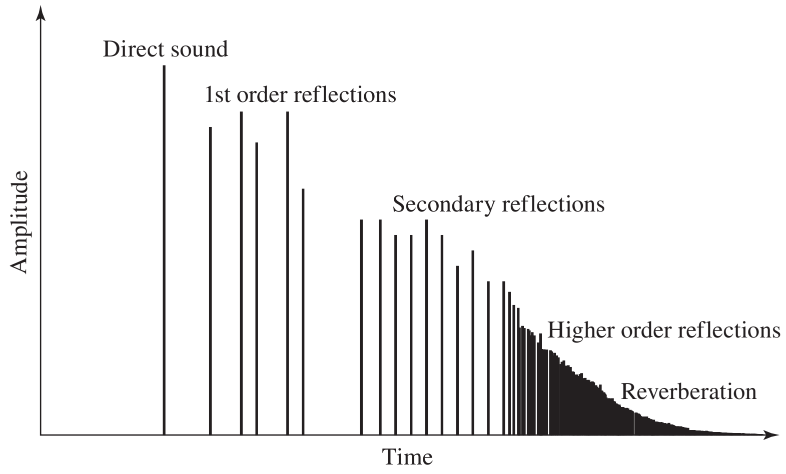


Figure 5.40 Sound reflections in an acoustical space

order than there were at a lower order. However, each time the waves reflect, they lose some of their energy to heat, as shown by the decreasing amplitude of higher-order reflections. Eventually, there can be so many reflected waves that there is barely any time between their moments of arrival to the listener's ear. This blur of reflections is perceived as reverb.

Two basic strategies are used to implement reverb effects. The first is to try to model the physical space or the acoustical properties of an instrument or recording device. For rooms, this involves analyzing the size of the room, the angles and spacing of walls and ceilings, the objects in the room, the building and decorating materials, the usual temperature and humidity, the number of people likely to be in the room, and so forth. One standard measure of an acoustical space is its reverberation time—the time it takes for a sound to decay by 60 dB from its original level. Based on the analysis of the space, the space's effect on sound is simulated by arranging multiple recursive and all-pass filters in serial, in parallel, or in nested configurations. The many possible arrangements of the filters along with the possible parameter settings of delay time and coefficients allow for a wide array of designs.

A second method, called the ***impulse response method***, is something you could try experimentally yourself. This method is applied to modeling the way things sound in a particular listening space. The idea is to test the acoustics of a space by recording how a single quick impulse of sound reverberates, and then using that recording to make your own sound file reverberate in the same manner. Ideally, you would go into the listening room and generate a pure, instantaneous impulse of sound that contains all audible frequencies, recording this sound. In practice, you can't create an ideal impulse, but you can do something like clap your hands, pop a balloon, or fire a starter pistol. This recording is your impulse response—a filter—that can be convolved with the audio file to which you're applying the effect. This technique is sometimes called ***convolution reverb***. There are some hitches in this if you try it experimentally. The impulse you generate may be noisy. One way to get around this is to use a longer sound to generate the impulse—a sine wave that sweeps through all audible frequencies over several seconds. Then this recorded sound is deconvolved so that, hypothetically, only the reverberations from the room remain, which can be used as the impulse response. (Deconvolution begins with the output that would result from applying this filter and determines the input signal.)

Trying to generate your own impulse response file is interesting and illustrative, but for more polished results you have recourse to hundreds of libraries of impulse files (sometimes called ***IRs***) available commercially or free on the web. These IRs simulate the reverberance of concert halls, echo chambers, electric guitars, violins, specialized mics—just about any sound environment, instrument, voice, or audio equipment you can think of. In fact, if you have reverberation software that takes IRs as input, you can use any file at all, as long as it's of the expected format (*e.g.*, WAV or AIFF). You can get creative and add the resonance of your own voice, for example, to a piece of music.

5.5 THE RELATIONSHIP BETWEEN CONVOLUTION AND THE FOURIER TRANSFORM

It can be shown that the Fourier transform is just another side of the coin of convolution. Let's think about how filtering might be done with the Fourier transform and see how this relates to filtering as convolution. What you're trying to do is the following: Let $\boldsymbol{x}(n)$ be a digital audio signal. Let $\boldsymbol{X}(z)$ be the discrete Fourier transform of $\boldsymbol{x}(n)$. (The notation used in this section is standard in digital signal processing.) You want to filter $\boldsymbol{X}(z)$, creating output $\boldsymbol{Y}(z)$, such that $\boldsymbol{Y}(z)$ has the frequency components at the desired levels. Thus, you want to find some $\boldsymbol{H}(z)$ such that $\boldsymbol{Y}(z) = \boldsymbol{H}(z)\boldsymbol{X}(z)$. Note that $\boldsymbol{Y}(z)$ is the audio signal represented in the frequency domain. Thus, the inverse discrete Fourier transform of $\boldsymbol{Y}(z)$ will give you $\boldsymbol{y}(n)$, the filtered signal in the time domain. In summary, the steps are these:

- take the discrete Fourier transform of digital audio signal $\boldsymbol{x}(n)$, which gives $\boldsymbol{X}(z)$
- describe the specifications of your filter
- from these specifications, find $\boldsymbol{H}(z)$ such that $\boldsymbol{Y}(z) = \boldsymbol{H}(z)\,\boldsymbol{X}(z)$ and $\boldsymbol{Y}(z)$ has the desired frequency components
- perform the multiplication $\boldsymbol{H}(z)\boldsymbol{X}(z)$ to get $\boldsymbol{Y}(z)$
- take the inverse discrete Fourier transform of $\boldsymbol{Y}(z)$, which gives $\boldsymbol{y}(n)$, your filtered signal represented in the time domain

The difficult part of the process above is finding $\boldsymbol{H}(z)$, just as the difficult part of designing a convolution filter is finding $\boldsymbol{h}(n)$. Actually, the two processes amount to the same thing. Performing $\boldsymbol{Y}(z) = \boldsymbol{H}(z)\boldsymbol{X}(z)$ and then taking the inverse discrete Fourier transform of $\boldsymbol{Y}(z)$ to get $\boldsymbol{y}(n)$ is equivalent to doing a convolution operation described as $\boldsymbol{y}(n) = \boldsymbol{h}(n) \otimes \boldsymbol{x}(n) = \sum_{k=0}^{N} \boldsymbol{h}(k)\boldsymbol{x}(n-k)$. The point here is that convolving in the time domain is equivalent to multiplying in the frequency domain. This is expressed in the ***convolution theorem***:

Let $\boldsymbol{H}(z)$ be the discrete Fourier transform of a convolution filter $\boldsymbol{h}(n)$, and let $\boldsymbol{X}(z)$ be the discrete Fourier transform of a digital audio signal $\boldsymbol{x}(n)$. Then $\boldsymbol{y}(n) = \boldsymbol{h}(n) \otimes \boldsymbol{x}(n)$ is equivalent to the inverse discrete Fourier transform of $\boldsymbol{Y}(z)$, where $\boldsymbol{Y}(z) = \boldsymbol{H}(z)\boldsymbol{X}(z)$.

In fact, doing multiplication in the frequency domain and then inverting to the time domain is more computationally efficient than doing convolution in the time domain (if the Fast Fourier transform is used for the implementation).

The bottom line is that to design a particular filter yourself, you still have to determine either $\boldsymbol{h}(n)$ or $\boldsymbol{H}(z)$.

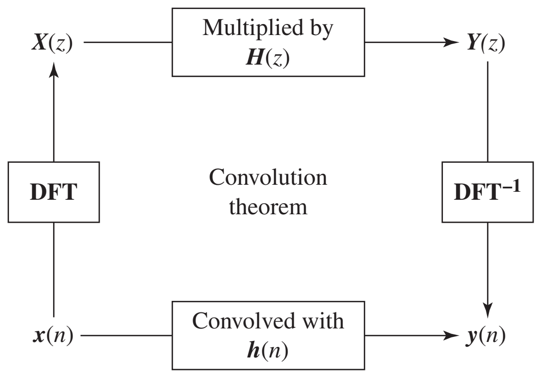


Figure 5.41 Equivalent operations in time and frequency domains

5.6 DESIGNING AND IMPLEMENTING YOUR OWN FILTERS

5.6.1 FIR Filter Design

Let's turn now to the design of FIR filters. At the end of this section, we give an algorithm for creating an FIR filter. You could apply this algorithm without understanding how it works. If you do want to understand how it works, you should read on through this section to look at the underlying mathematics, or you can skip ahead to the algorithm and try to learn by application.

The convolution mask $\boldsymbol{h}(n)$ is also called the ***impulse response***, representing a filter in the time domain. Its counterpart in the frequency domain, the ***frequency response*** $\boldsymbol{H}(z)$, is also sometimes referred to as the ***transfer function***. The relationship between the impulse response and the frequency response of a digital audio filter is precisely the relationship between $\boldsymbol{h}(n)$ and $\boldsymbol{H}(z)$ described in the previous section.

A frequency response graph can be used to show the *desired* frequency response of a filter you are designing. For example, the frequency response of an ideal low-pass filter is shown in Figure 5.42. In the graph, angular frequency is on the horizontal axis. The vertical axis represents the fraction of each frequency component to be permitted in the filtered signal. This figure indicates that frequency components between $-\omega_c$ and ω_c are to be left unchanged, while all other frequency components are to be removed entirely. ω_c is the angular cutoff frequency. It can be assumed that the angular frequency in this graph is normalized. This is done by mapping the Nyquist angular frequency—which is half the sampling frequency—to π. That is, if f_{samp} is the sampling frequency in Hz and f_c is the cutoff frequency in Hz, then f_c, normalized, is $\frac{f_c}{f_{samp}}$. In angular units, this gives us the normalized $\omega_c = 2\pi\frac{f_c}{f_{samp}}$. It makes sense to normalize in this way because the only frequencies that can be validly digitized at a sampling frequency of f_s are frequencies between 0 and $\frac{f_s}{2}$. Note also that normalization implies that the cutoff frequency ω_c must be less than π.

This frequency response is ideal in that it displays a completely sharp cutoff between the frequencies that are removed and those that are not changed. In reality, it isn't possible to design such a filter, but the ideal frequency response graph serves as a place to begin. The first

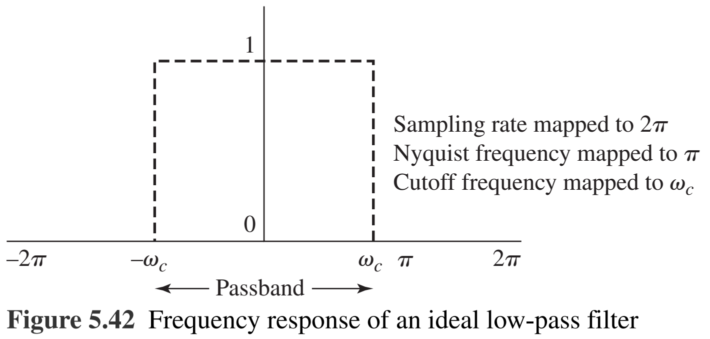

1
0
Sampling rate mapped to 2π
Nyquist frequency mapped to π
Cutoff frequency mapped to ω_c
-2π
$-\omega_c$
ω_c
π
2π
Passband

Figure 5.42 Frequency response of an ideal low-pass filter

ASIDE: A sinc function is the product of a sine function and a monotonically decreasing function. Its basic form is

$$\mathrm{sinc}(x) = \left\{ \begin{array}{ll} \dfrac{\sin(x)}{x} & \textit{for } x \neq 0 \\ 1 & \textit{for } x = 0 \end{array} \right\}$$

The sinc function is the Fourier transform of the rectangle function, and vice versa.

step in FIR filter design is to determine what ideal impulse response corresponds to the ideal frequency response.

The frequency response graph shown in Figure 5.42 is an example of a ***rectangle function***, which has a value of 1 in a certain finite range and 0s everywhere else. This graph represents an idealized version of how we'd like our filter to behave. You can think of it as an idealized form of $\boldsymbol{H}(z)$. Then what would be the corresponding ideal impulse response, an idealized $\boldsymbol{h}(n)$?

It can be shown that the inverse Fourier transform of a rectangle function in the frequency domain is a ***sinc function*** in the time domain, shown in Figure 5.43. The rectangle and sinc functions arise frequently in digital signal processing and are convenient because of the relationship they have with each other through the Fourier transform.

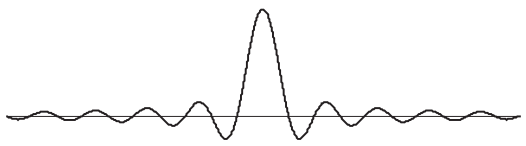


Figure 5.43 Sinc function

ASIDE: If you compare this ideal impulse response equation with Equation 4.5, you should note the following relationships:

function	$h_{ideal}(n)$	$f(t)$
time	n	t
frequency	ω	nω
freq. range	$-\pi\ldots\pi$*	$-\infty\ldots\infty$
base frequency	infinitesimal	ω
freq. domain function	$H_{ideal}(\omega)$	F_n
periodic?	no	yes
continuous?	yes	yes

*Actually, it is $-\infty\ldots\infty$, but we normalize frequencies so that $H_{ideal}(\omega) = 0$ when $|\omega| \geq$, so the integral is 0 outside the range $-\pi\ldots\pi$.

Let's see how this works mathematically. Recall the relationship between the frequency response and the impulse response. The former is the Fourier transform of the latter. In other words, the inverse Fourier transform of the ideal frequency response $H_{ideal}(\omega)$ gives us the ideal impulse response $h_{ideal}(n)$. (An integral rather than a summation is used because the function is not assumed to be periodic.)

$$h_{ideal}(n) = \frac{1}{2\pi}\int_{-\pi}^{\pi} H_{ideal}(\omega)e^{i\omega n}d\omega$$

This equation can be simplified as follows:

$$h_{ideal}(n) = \frac{1}{2\pi}\int_{-\pi}^{\pi} H_{ideal}(\omega)e^{i\omega n}d\omega$$

Step 1:
$$= \frac{1}{2\pi}\int_{-\omega_c}^{\omega_c} e^{i\omega n}d\omega$$

Step 2:
$$= \frac{1}{2\pi}\int_{-\omega_c}^{\omega_c} \cos(\omega n) + i\sin(\omega n)d\omega$$

Step 3:
$$= \frac{\sin(\omega_c n)}{\pi n}$$

Step 4:
$$= \frac{\sin(2\pi f_c n)}{\pi n} \quad \text{for } -\infty \leq n \leq \infty, n \neq 0 \text{ and } 2f_c \text{ for } n = 0$$

In short, we get

$$h_{ideal}(n) = \frac{\sin(2\pi f_c n)}{\pi n} \quad \text{for } -\infty \leq n \leq \infty, n \neq 0 \text{ and } 2f_c \text{ for } n = 0$$

Equation 5.3

In step 1, the integral is limited to the range of $-\omega_c$ to ω_c because $H_{ideal}(\omega)$ is equal to 1 between $-\omega_c$ and ω_c, and 0 everywhere else under the assumption that $\omega_c < \pi$. Step 2 does a substitution using Euler's identity. (See Chapter 4.) Step 3 does the integration. Notice that the sine term falls out because its integral is –cosine, which is an even function, and the negatives cancel the positives. In step 4, we use the substitution $\omega_c = 2\pi f_c$, from the relationship between angular frequency and frequency in Hz. The case where $n = 0$ results from taking the limit as n goes to 0, by l'Hôpital's rule. We have derived a form of the sinc function, as expected. The importance of this derivation is that Equation 5.3 gives you the ideal impulse response based on your desired cutoff frequency f_c. By a similar process, it is possible to derive the equations for ideal high-pass and bandpass filters, which are given in Table 5.2.

TABLE 5.2 Equations for Ideal Impulse Responses for Standard Filters, Based on Cutoff Frequency f_c and Band Edge Frequencies f_1 and f_2

Type of filter	$h_{ideal}(n), n \neq 0$	$h_{ideal}(0)$
Low-pass	$\frac{\sin(2\pi f_c n)}{\pi n}$	$2f_c$
High-pass	$-\frac{\sin(2\pi f_c n)}{\pi n}$	$1 - 2f_c$
Bandpass	$\frac{\sin(2\pi f_2 n)}{\pi n} - \frac{\sin(2\pi f_1 n)}{\pi n}$	$2(f_2 - f_1)$
Bandstop	$\frac{\sin(2\pi f_1 n)}{\pi n} - \frac{\sin(2\pi f_2 n)}{\pi n}$	$1 - 2(f_2 - f_1)$

But we're still in the realm of the ideal. We don't yet have a workable FIR filter. The problem is that a sinc function goes on infinitely in the positive and negative directions, as shown in Figure 5.43. The amplitude continues to diminish in each direction but never goes to 0. We can't use this ideal, infinite sinc function as an impulse response (*i.e.*, a convolution mask) for an FIR filter. Think about the problem of trying to apply a convolution mask like this in real-time. To compute a new value for a given audio sample, we'd need to know all past sample values on to infinity, and we'd need to predict what future sample values are on to infinity.

The next step in FIR filter design is to take this ideal, infinite sinc function and modify it to something that is realizable as a convolution mask of finite length. However, a consequence of the modification is that you can't achieve an *ideal* frequency response. Modifying the ideal impulse response by making it finite creates ripples in the corresponding frequency response.

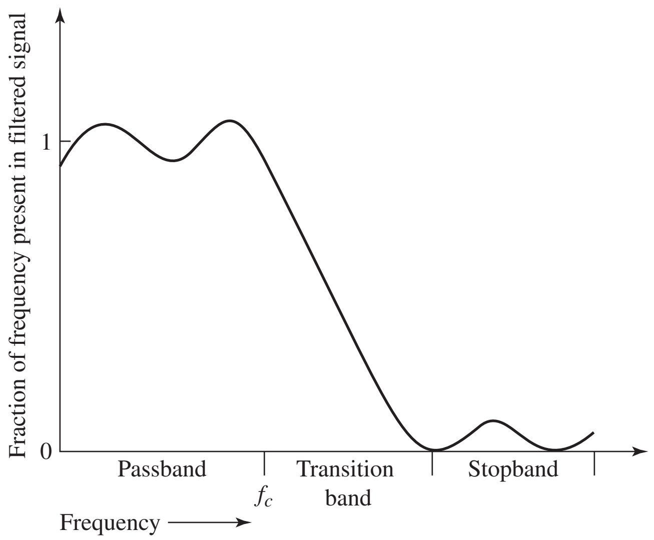


Figure 5.44 Frequency response of a realistic low-pass filter

Now let's consider a realistic frequency response graph. Figure 5.44 shows the kind of frequency response graph you'll encounter in the literature and documentation on digital audio filters. Such a graph could represent the specifications of a filter design, or it could represent the behavior of an existing filter. As with the graph of the ideal frequency response, the horizontal axis corresponds to frequency and the vertical axis gives the fraction of the frequency that will remain in the audio file after it is filtered, corresponding to attenuation. In Figure 5.44, frequency is given in Hz rather than radians, and only the positive portion need be shown since the graph is implicitly symmetrical around 0, but otherwise the graph is the same in nature as the ideal frequency response graph in Figure 5.42. As before, we consider the frequency components only up to one-half the sampling rate, which is the Nyquist frequency, since no other frequencies can be validly sampled. In frequency response graphs, units on the vertical axis are sometimes shown in decibels. Measured in decibels, attenuation is equal to $20 \log_{10}\left(\frac{a_{out}}{a_{in}}\right) dB$, where a_{in} is the amplitude of frequency component f before the filter is applied, and a_{out} is the amplitude after the filter is applied. In the graph shown, the units are normalized to range from 0 to 1, indicating the fraction by which each frequency component is attenuated. In the case of normalization, attenuation is measured as $\frac{a_{out}}{a_{in}}$.

Compare the realistic frequency response graph to an ideal one. In a realistic graph, the ***passband*** is the area from 0 up to the cutoff frequency f_c and corresponds to the frequencies the filter tries to retain. Ideally, you want frequencies up to f_c to pass through the filter unchanged, but in fact they may be slightly attenuated in either the positive or negative direction, shown as a ***ripple***, fluctuations in the frequency magnitude. The ***stopband*** corresponds to the frequencies the filter attenuates or filters out. In the realistic graph, the stopband has ripples like the passband, indicating that the unwanted frequencies are not filtered out perfectly. The ***transition band*** lies in between. Unlike the transition in an ideal filter, in a real filter the ***transition region rolloff***—the slope of the curve leading to the transition band—is not infinitely steep.

When you design an FIR filter, you specify certain parameters that define an acceptable, though not ideal, frequency response. For example, you could specify:

- for a low-pass filter, the cutoff frequency marking the end of the passband (f_c above)
- for a low-pass filter, transition width, the maximum acceptable bandwidth from the end of the passband to the beginning of the stopband
- for a stopband or bandpass filter, f_1 and f_2, the beginning and end frequencies for the stopband or passband
- passband deviation, the maximum acceptable range of attenuation in the passband due to rippling
- stopband deviation, the maximum acceptable range of attenuation in the stopband due to rippling

So how do you realize a workable FIR filter based on your design specifications? This takes us back to the ideal impulse response functions we derived in Table 5.2. One way to design an FIR filter is to take an ideal impulse response and multiply it by a windowing function $w(n)$. The purpose of the windowing function is to make the impulse response finite. However, making the impulse response finite results in a frequency response that is less than ideal. It has ripples in the passband or stopband, like those shown in Figure 5.44. Thus, you need to select a windowing function that minimizes the ripples in the passband or stopband and/or creates your desired cutoff slope.

The simplest windowing function is rectangular. It has the value 1 in some finite range across the infinite impulse response and 0s everywhere else. Multiplying by this window simply truncates the infinite impulse response on both sides. The disadvantage of a rectangular windowing function is that it has the effect of providing only limited attenuation in the stopband. Four other commonly used windowing functions are the triangular, Hanning, Hamming, and Blackman windows. The triangular, Hamming, Hanning, and Blackman windows are tapered smoothly to 0 at each end. The effect is that when you multiply a sinc function by one of these windowing functions, the resulting frequency response has less of a ripple in the stopband than would be achieved if you multiply by a rectangular window. The disadvantage is that the transition band will generally be wider.

ASIDE: These are essentially the same Hanning, Hamming, and Blackman function as discussed in Chapter 4, through they look a little different in form. The functions in Chapter 4 were expressed as continuous functions going from 0 to T. In Table 5.3 they are expressed as discrete functions going from $-\frac{1}{2}N$ to $\frac{1}{2}N$. This shift of values symmetric around 0 results in terms being added rather than subtracted in the function.

TABLE 5.3 Windowing Functions

1 Rectangular windowing function	$w(n) = 0.5 + 0.5\cos\left(\frac{2\pi n}{N}\right)$ Hanning windowing function
$w(n) = 0.54 + 0.46\cos\left(\frac{2\pi n}{N}\right)$ Hamming windowing function	$w(n) = 0.42 + 0.5\cos\left(\frac{2\pi n}{N-1}\right) + 0.08\cos\left(\frac{4\pi n}{N-1}\right)$ Blackman windowing function

The process discussed in this section is called the ***windowing method of FIR filter design***. In summary, the steps are given in Algorithm 5.1. The algorithm is for a low-pass filter, but a similar one could be created for a high-pass, bandpass, or bandstop filter simply by replacing the function with the appropriate one from Table 5.2 and including the needed parameters. One parameter needed for all types of filters is the order of the filter, N (*i.e.*, the number of coefficients). The order of the filter will have an effect on the width of the transition band. Call the width of the transition band b. Assume that the Nyquist frequency in Hz is normalized to 0.5 and that b is on this scale. Then the relationship between b and N is given by $b = 4/N$. Thus, if you know the width you'd like for your transition bandwidth, you can determine an appropriate order for the filter. A higher-order filter gives a sharper cutoff between filtered and unfiltered frequencies, but at the expense of more computation.

ALGORITHM 5.1

```
algorithm FIR_low_pass filter
/*Input: f_c, the cutoff frequency for the lowpass filter, in Hz
         f_samp, the sampling frequency of the audio signal to be filtered, in Hz
         N, the order of the filter; assume N is odd
  Output: a low-pass FIR filter in the form of an N-element array */
{
  /*Normalize f_c and $\omega$_c so that $\pi$ is equal to the Nyquist angular frequency*/
  f_c = f_c/f_samp
  $\omega$_c = 2*$\pi$*f_c
  middle = N/2      /*Integer division, dropping remainder*/
  /*Create the filter using the low-pass filter function from Table 5.2*/
  for n = –N/2 to N/2

    if (n = 0) fltr(middle) = 2*f_c
    else fltr(n+middle) = sin(2*$\pi$*_c*n)/($\pi$ *n)
  /*Multiply the elements of fltr by a windowing function chosen from Table 5.3.
  We use the Hanning window function*/
  for n = 0 to N-1

      fltr(n) = fltr(n) * (0.5 + 0.5*cos((2*$\pi$*(n-middle))/N))
}
```

Supplement on windowing method for FIR filter design:

mathematical modeling worksheet

We've described the windowing method of FIR filter design because it gives an intuitive sense of the filters and related terminology and is based on the mathematical relationship between the impulse and frequency responses. However, it isn't necessarily the best method. Two other methods are the optimal method and the frequency sampling method. The optimal method is easy to apply and has the advantage of distributing the ripples more evenly over the passband and stopband, called ***equiripple***. The advantage of the frequency sampling method is that it allows FIR filters to be implemented both nonrecursively and recursively and thereby leads to computational efficiency. Another important implementation issue is the effect of quantization error on filter performance. We refer you to the references for details on these filter design approaches and issues.

In this section, we've given you the basic mathematical knowledge with which you could design and implement your own software FIR filters. Of course you don't always have to design filters from scratch. Signal processing tools such as MATLAB allow you to design filters at a higher level of abstraction, but to do so, you still need to understand the terminology and processes described here. Even when you use the predesigned filters that are available in audio processing hardware or software, it helps to understand how the filters operate in order to choose and apply them appropriately.

5.6.2 The Z-Transform

For a complete understanding of IIR and FIR filters, you need to be familiar with the ***z-transform***. Terminology related to z-transforms can be used in filter design and in descriptions of predesigned filters. The manner in which frequencies are altered by a filter can be represented conveniently by means of z-transforms. Another application of z-transforms is to analyze the effects of quantization error on digital filter performance.

Consider a sequence of discrete values $\boldsymbol{x}(n)$ for $n = 0, 1, \ldots$. The z-transform of this sequence is defined as $\boldsymbol{X}(z) = \sum_{n=0}^{\infty}\boldsymbol{x}(n)z^{-n}$ where z is a complex variable. This defines a ***one-sided z-transform***, with values of n going from 0 to ∞. A full z-transform sums from $-\infty$ to ∞, but the one-sided transform works for our purposes.

Notice the naming convention that $\boldsymbol{x}(n)$ is transformed to $\boldsymbol{X}(z)$, $\boldsymbol{y}(n)$ is transformed to $\boldsymbol{Y}(z)$, $\boldsymbol{h}(n)$ is transformed to $\boldsymbol{H}(z)$, and so forth.

First, think of the z-transform in the abstract. It doesn't really matter what $\boldsymbol{x}(n)$ corresponds to in the real world. Try applying the definition to $\boldsymbol{x}(n) = [5, -2, 3, 6, 6]$, assuming that $\boldsymbol{x}(n) = 0$ for $n > 4$. (Henceforth, we'll assume $\boldsymbol{x}(n) = 0$ if no value is specified for $\boldsymbol{x}(n)$. Array positions are numbered from 0.)

$$\boldsymbol{X}(z) = 5 - 2z^{-1} + 3z^{-2} + 6z^{-3} + 6z^{-4}$$

You can see that $\boldsymbol{X}(z)$ is a function of the complex variable z.

It's no coincidence that the functions are called $\boldsymbol{H}(z)$, $\boldsymbol{X}(z)$, and $\boldsymbol{Y}(z)$ as they were in the previous sections. This is because you can turn the z-transform into something more familiar if you understand that it is a generalization of the discrete Fourier transform. Let's see how this works.

In general, $\boldsymbol{X}(z)$ is a function that runs over all complex numbers z. However, consider what you get if you specifically set $z = e^{i\theta}$ and apply the z-transform to a vector of length N. This gives you

$$\text{for } 0 \le k < N, \boldsymbol{X}(z_k) = \sum_{n=0}^{N-1}\boldsymbol{x}(n)e^{-i\theta n} = \sum_{n=0}^{N-1}\boldsymbol{x}(n)e^{\frac{-i2\pi kn}{N}} \quad \text{for } z = e^{i\theta} \quad \text{and} \quad \theta = \frac{2\pi k}{N}$$

Equation 5.4

Equation 5.4 is the discrete Fourier transform of $\boldsymbol{x}(n)$. The subscript k indicates that the summation is performed to determine each kth frequency component. We will omit the k when it is not important to the discussion, but you should keep in mind that the summation is performed for every frequency component that is computed. In the context of this equation, we are considering discretely spaced frequency components. The N frequency components are spaced by $\frac{2\pi}{N}$, and the kth frequency component is frequency $\theta = \frac{2\pi k}{N}$. Thus, although

$\boldsymbol{X}(z)$ is defined over the whole complex plane, when we perform the discrete Fourier transform of $\boldsymbol{x}(n)$, we are only evaluating $\boldsymbol{X}(z)$ at N specific points on the complex plane. These points represent different frequency components, and they all lie evenly spaced around the unit circle.

We already observed that convolution in the time domain is equivalent to multiplication in the frequency domain in the sense that performing $\boldsymbol{y}(n) = \boldsymbol{h}(n) \otimes \boldsymbol{x}(n)$ is equivalent to performing $\boldsymbol{Y}(z) = \boldsymbol{H}(z)\boldsymbol{X}(z)$ and then taking the inverse Fourier transform of $\boldsymbol{Y}(z)$. So why do you need z-transforms, since you can focus on a special case of the z-transform, which is the Fourier transform? The reason is that expressing things using the notation of a z-transform is more convenient mathematically because it helps to lay bare how a given filter will behave. An IIR filter is defined as a convolution by $\boldsymbol{y}(n) = \boldsymbol{h}(n) \otimes \boldsymbol{x}(n)$. The equivalent relation in terms of the z-transform is $\boldsymbol{Y}(z) = \boldsymbol{H}(z)\boldsymbol{X}(z)$, from which it follows that

$$\boldsymbol{H}(z) = \frac{\boldsymbol{Y}(z)}{\boldsymbol{X}(z)}$$

This form of $\boldsymbol{H}(z)$, expressed as $\dfrac{\boldsymbol{Y}(z)}{\boldsymbol{X}(z)}$, is referred to as a ***transfer function***. Note that this form is closely related to the recursive form of an IIR filter.

KEY EQUATION

Let $\boldsymbol{y}(n) = \boldsymbol{h}(n) \otimes \boldsymbol{x}(n) = \sum_{k=0}^{N-1} a_k \boldsymbol{x}(n-k) - \sum_{k=1}^{M} b_k \boldsymbol{y}(n-k)$ be an IIR filter, as defined in Equation 5.2. Let $\boldsymbol{H}(z) = \dfrac{\boldsymbol{Y}(z)}{\boldsymbol{X}(z)}$ be the transfer function for this filter. Then

$$\boldsymbol{H}(\boldsymbol{z}) = \frac{a_0 + a_1 z^{-1} + a_2 z^{-2} + \cdots}{1 + b_1 z^{-1} + b_2 z^{-2} + \cdots} = \frac{a_0 z^N + a_1 z^{N-1} + \cdots}{z^N + b_1 z^{N-1} + \cdots}$$

We'll see in the next section that the transform function gives us a convenient way of predicting how a filter will behave, or designing an IIR filter so that it behaves the way we want it to.

5.6.3 Zero-Pole Diagrams and IIR Filter Design

Previously, we looked at graphs of the frequency response, calling the function $\boldsymbol{H}(z)$—for example, Figure 5.23. More precisely, this is a graph of the frequency magnitude response. Recall that the discrete Fourier transform yields frequency components as complex numbers. The magnitude of a complex number $a + bi$ is equal to $\sqrt{a^2 + b^2}$. The frequency response of a filter can also be visualized by plotting $\boldsymbol{H}(z)$ on the complex number plane. For a complex number $a + bi$, the horizontal axis of the complex number plane represents the real-number component, a, while the vertical axis represents the coefficient of the imaginary component, b. This is pictured in Figure 5.45. $\overrightarrow{a}$, $\overrightarrow{b}$, and $\overrightarrow{r}$ are vectors, while a, b, and r denote the magnitudes of these vectors. Consider a circle of radius $r = 1$ around

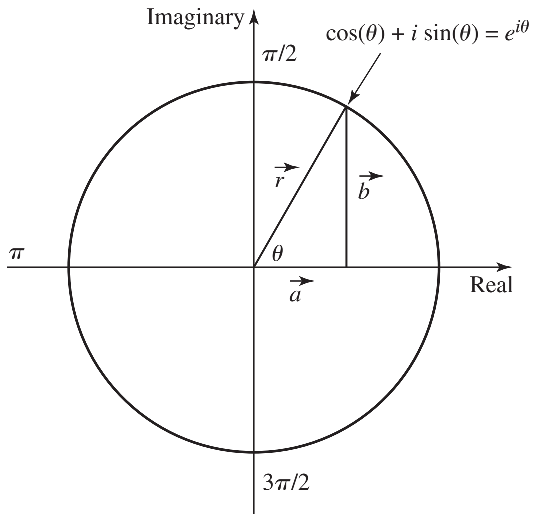


Figure 5.45 A circle with radius 1 on the complex-number plane

the origin. If $r = 1$, then the points that lie on this circle correspond to the complex numbers $e^{i\theta}$. We can see this from simple geometry and vector addition.

$$\begin{aligned} a &= r|\cos\theta| = |\cos\theta| \\ b &= r|\sin\theta| = |\sin\theta| \\ \vec{a} &= (\cos\theta, 0) \\ \vec{b} &= (0, \sin\theta) \\ \vec{r} &= (\cos\theta, \sin\theta) \end{aligned}$$

The point $(\cos\theta, \sin\theta)$ corresponds to $\cos\theta + i\sin\theta$ in the complex number plane. By Euler's identity, this is $e^{i\theta}$.

Why do we want to trace a circle on the complex number plane using the point $e^{i\theta}$? Because this circle is exactly the circle on which we evaluate the z-transform for each frequency component. By Equation 5.4, the kth frequency component, $\boldsymbol{H}(z_k)$, is obtained by evaluating the discrete Fourier transform at $z = e^{i\theta} = e^{i\frac{2\pi k}{N}}$. Since $\boldsymbol{H}(z)$ yields the frequency response, evaluating $\boldsymbol{H}(z_k)$ tells us how much the kth frequency component is attenuated by the filter.

We know that $\boldsymbol{H}(z) = \dfrac{\boldsymbol{Y}(z)}{\boldsymbol{X}(z)}$. Notice that as $\boldsymbol{Y}(z)$ gets closer to 0, $\boldsymbol{H}(z)$ gets smaller. As $\boldsymbol{X}(z)$ gets closer to 0, $\boldsymbol{H}(z)$ gets larger. The places where $\boldsymbol{Y}(z) = 0$ are called the ***zeros*** of $\boldsymbol{H}(z)$. The places where $\boldsymbol{X}(z) = 0$ are called the ***poles*** of $\boldsymbol{H}(z)$.

The zeros and poles for a given filtering function can be placed on this same graph. Placing the zeros and poles on the graph makes it possible to get a general picture of how a filter affects the digital signal it operates on, without having to evaluate $\boldsymbol{H}(z_k)$ for every k. To see how the filter defined by $\boldsymbol{H}(z)$ will affect a digital signal, you determine its zeros and poles from $\boldsymbol{H}(z) = \dfrac{\boldsymbol{Y}(z)}{\boldsymbol{X}(z)}$ and plot them on the zero-pole graph. Once you know where the zeros and poles are, you can see how frequency components are affected by the filter. For example, you may be able to see that high frequencies, beyond a certain point, are attenuated while frequencies less than this cutoff are unchanged.

This may seem confusing at first. You need $\frac{\boldsymbol{Y}(z)}{\boldsymbol{X}(z)}$ to get a filter's zeros and poles. But $\boldsymbol{Y}(z)$ is the z-transform of $\boldsymbol{y}(n)$, and $\boldsymbol{y}(n)$ is the output of the filter. How can you determine $\boldsymbol{Y}(z)$ without actually running the filter? Actually, you do it algebraically. Let's try a simple example of an FIR filter to see how you might get values for the zeros and poles, and from these you can predict the overall effect of the filter.

Let $\boldsymbol{h}(n) = [1, -0.5]$. Then its z-transform, $\boldsymbol{H}(z)$, is $1 - 0.5z^{-1}$. We can divide this into a numerator and denominator by multiplying by $\frac{z}{z}$. This gives us $\boldsymbol{H}(z) = \frac{z - 0.5}{z}$. Thus, $\boldsymbol{H}(z)$ has a zero at $z = 0.5$ and a pole at $z = 0$.

Another way to get a zero and a pole associated with $\boldsymbol{h}(n) = [1, -0.5]$ is as follows. The difference equation form of this convolution is

$$\boldsymbol{y}(n) = \sum_{k=0}^{1} \boldsymbol{h}(k)\boldsymbol{x}(n - k) = \boldsymbol{x}(n) - 0.5\boldsymbol{x}(n - 1)$$

The z-transform has a property called the ***delay property*** whereby the z-transform of $\boldsymbol{x}(n - 1)$ equals $z^{-1}\boldsymbol{X}(z)$. The z-transform also has the ***linearity property,*** which states that if two functions are equal, then their z-transforms are equal, and the z-transform of a sum of terms is equal to the sum of the z-transforms of the terms. Thus, if we take the z-transform of both sides of the equation $\boldsymbol{y}(n) = \boldsymbol{x}(n) - 0.5\boldsymbol{x}(n - 1)$, we get

$$\boldsymbol{Y}(z) = \boldsymbol{X}(z) - 0.5z^{-1}\boldsymbol{X}(z)$$

Dividing both sides by $\boldsymbol{X}(z)$ yields

$$\frac{\boldsymbol{Y}(z)}{\boldsymbol{X}(z)} = \boldsymbol{H}(z) = 1 - 0.5z^{-1} = \frac{z - 0.5}{z}$$

Thus, we have a zero at $z = 0.5$ (where the numerator equals 0) and a pole at $z = 0$ (where the denominator equals 0).

Once you know the zeros and poles, you can plot them on the complex plane in a graph called a ***zero-pole diagram***. From this you can derive information about the frequency response of the filter. At point $z = 0.5$, there is no imaginary component, so this point lies on the horizontal axis. The pole is at the origin. This is shown in Figure 5.46.

To predict the filter's effect on frequencies, we look at each kth point—called it P_k—around the unit circle, where P_0 falls on the x-axis. The angle formed between P_k, the origin, and P_0 is θ (which is equal to $\frac{2\pi k}{N}$). Each such angle corresponds to a frequency component of a signal being altered by the filter. Since the Nyquist frequency has been normalized to π, we consider only the frequencies between 0 and π, in the top semicircle. Let d_{zero} be the distance between P_k and the zero of the filter, and let d_{pole} be the distance between P_k and the pole. The frequency response $H(z)$ will be larger or smaller in proportion to $\frac{d_{zero}}{d_{pole}}$. In this case, because the pole of the filter is at the origin, d_{pole} is the distance

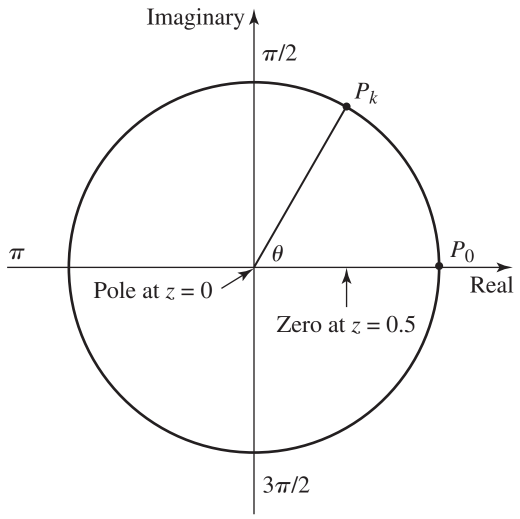


Figure 5.46 A zero-pole diagram

from the origin to the unit circle, which is always 1, so the size of $\frac{d_{zero}}{d_{pole}}$ depends entirely on d_{zero}. As d_{zero} gets larger, the frequency response gets larger, and d_{zero} gets increasingly large as you move from $\theta = 0$ to $\theta = \pi$, which means the frequency response gets increasingly large. Thus, the zero-pole diagram represents a high-pass filter in that it attenuates low frequencies more than high frequencies.

Supplement on z-transforms, zero-pole diagrams, and IIR filters:

interactive tutorial

Now let's try deriving and analyzing the zero-pole diagram of an IIR filter. Say that your filter is defined by the difference equation

$$\boldsymbol{y}(n) = \boldsymbol{x}(n) - \boldsymbol{x}(n-2) + 0.64\boldsymbol{y}(n-2)$$

This can be written as

$$\boldsymbol{y}(n) - 0.64\boldsymbol{y}(n-2) = \boldsymbol{x}(n) - \boldsymbol{x}(n-2)$$

The delay and linearity properties of the z-transform yield

$$\boldsymbol{Y}(z) - 0.64z^{-2}\boldsymbol{Y}(z) = \boldsymbol{X}(z) - z^{-2}\boldsymbol{X}(z)$$

The rest of the derivation goes as follows:

Supplement on creating FIR and IIR filters in MATLAB:

mathematical modeling worksheet

$$\boldsymbol{Y}(z)(1 - 0.64z^{-2}) = \boldsymbol{X}(z)(1 - z^{-2}) \Rightarrow$$
$$\frac{\boldsymbol{Y}(z)}{\boldsymbol{X}(z)} = \frac{1 - z^{-2}}{1 - 0.64z^{-2}} = \frac{z^2 - 1}{z^2 - 0.64} = \frac{(z+1)(z-1)}{(z+0.8i)(z-0.8i)} = \boldsymbol{H}(z)$$

The zeros are at $z = 1$ and $z = -1$, and the poles are at $z = 0.8i$ and $z = -0.8i$. (Note that if the imaginary part of z is nonzero, roots appear as conjugate pairs $a \pm ib$.) The zero-pole diagram is in Figure 5.47. This diagram is harder to analyze by inspection because we have more than one zero and more than one pole. To determine when the frequency response gets larger, we have to determine when $\boldsymbol{H}(z) = \frac{(z+1)(z-1)}{(z+0.8i)(z-0.8i)}$ grows, which is the

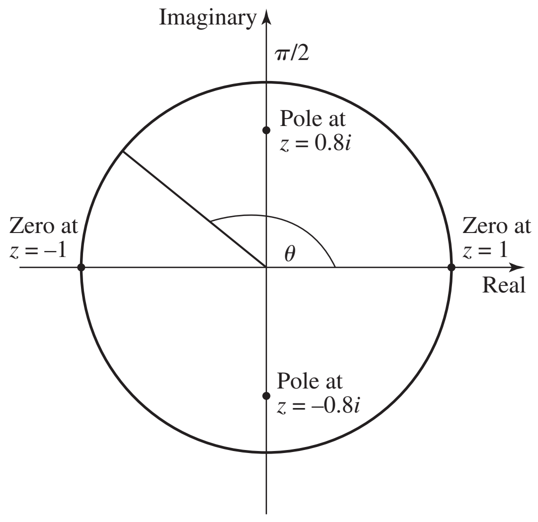


Figure 5.47 A zero-pole diagram for an IIR filter

Supplement on creating a filter via a transfer function:

mathematical modeling worksheet

same thing as determining how $(z + 1)(z - 1)$ gets large relative to $(z + 0.8i)(z - 0.8i)$. In fact, this diagram represents a bandpass filter, with frequencies near $\pi/2$ being the least attenuated.

We've looked at how you analyze an existing filter by means of its zero-pole diagram. It is also possible to design a filter using a zero-pole diagram—by placing the zeros and poles in relative positions such that they create the desired frequency response. This is feasible, however, only for simple filters.

The most commonly used design method for IIR filters is to convert analog filters into equivalent digital ones. This method relies on the z-transform as well as the equivalent transform in the continuous domain, the Laplace transform. We won't go into the details of this method of filter design. However, you should be aware of four commonly used types of IIR filters, the Bessel, Butterworth, Chebyshev, and elliptic filters.

Recall that FIR filters are generally phase linear, which means that phase shifts affect frequency components in a linear fashion. The result is that harmonic frequencies are shifted in the same proportions and thus don't sound distorted after filtering, which is important in music. In IIR filters, on the other hand, care must be taken to enforce phase linearity.

Bessel filters are the best IIR filters for ensuring a linear phase response. ***Butterworth filters*** sacrifice phase linearity so that they can provide a frequency response that is as flat as possible in the passband and stopband (*i.e.*, they minimize ripples). A disadvantage of Butterworth filters is that they have a wide transition region. ***Chebyshev filters*** have a steeper roll-off than Butterworth filters, the disadvantage being that they have more nonlinear phase response and more ripple. However, the amount of ripple can be constrained, with tradeoffs between ripple and roll-off or between ripple in the passband (Chebyshev 1 filters) and ripple in the stopband (Chebyshev 2 filters). ***Elliptic filters***, also called Cauer filters, have the worst phase linearity of the common IIR filters, but they have the sharpest roll-off relative to the number of coefficients and are equiripple in the passband and stopband. The choice of filter depends on the nature of your audio signal and how you want it to sound after filtering.

5.7 DIGITAL AUDIO COMPRESSION

5.7.1 Time-Based Compression Methods

Chapter 4 covered some methods that effectively reduce the size of audio files. Nonlinear quantization (*e.g.*, μ- or A-law encoding) reduces the amount of data in voice signals by quantizing high-amplitude signals more coarsely than low-amplitude ones. This works well because the human ear distinguishes more finely among relatively low amplitudes (1000 to 5000 Hz) than among high (10,000 to 20,000 Hz). Also, when the technique is applied to a voice signal such as a telephone transmission, the reduction in quality resulting from effectively reducing the bit depth isn't objectionable.

Differential pulse code encoding is another way of making an audio file smaller than it would be otherwise—by recording the difference between one sample and the next rather than recording the actual sample value. Variations of DPCM include ADPCM and delta modulation.

A-law encoding, μ-law encoding, delta modulation, and other variations of PCM are time-based methods. That is, there is no need to transform the data into the frequency domain in order to decide where and how to eliminate information. In a sense, these methods aren't really compression at all. Rather, they are ways of reducing the amount of data at the moment an audio signal is digitized, and they are often considered conversion techniques rather than compression methods.

The most effective audio compression methods require some information about the frequency spectrum of the audio signal. These compression methods are based on psychoacoustical modeling and perceptual encoding.

5.7.2 Perceptual Encoding

Perceptual encoding is based upon an analysis of how the ear and brain perceive sound, called ***psychoacoustical modeling***. Just as digital image compression takes advantage of visual elements to which the human eye is not very sensitive, perceptual encoding exploits audio elements that the human ear cannot hear very well. In the case of digital images, human eyes are 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 an audio signal. Then what *can* be filtered out of digital audio? What kinds of sounds do humans not hear very well, and in what conditions?

Psychoacoustical tests have shown that there is a great deal of nonlinearity in human sound perception. The pitch of frequencies is perceived in a nonlinear fashion, for example. Octaves are separated by frequencies that are multiples of each other, such that the width in Hz of a lower octave is much smaller than the width in Hz of a higher octave. The octave from middle C (called C4) to C5 ranges from 261.63 Hz to 523.25 Hz, while C5 to C6 ranges from 523.25 to 1046.50 Hz. But the distance from C4 to C5 subjectively sounds the same as the distance from C5 to C6.

Humans hear best in the 1000 to 5000 Hz range, the frequency range of human speech. If you have two tones—the first at, say, 100 Hz and the second at 1000 Hz, you have to play the 100 Hz tone louder than the 1000 Hz tone to make them equally loud. The same difference holds true between 1000 Hz and 10,000 Hz tones. Again, you have to play the higher frequency tone louder to make it sound as loud as the 1000 Hz tone, because the 1000 Hz tone is in the range where humans hear the best.

Human ability to distinguish between frequencies decreases nonlinearly from lower to higher frequencies. At low frequencies, we can distinguish between sound waves that are only a few Hz apart. At high frequencies, the frequencies must be a hundred or more Hz apart for us to hear the difference. The reason for this is that the ear is divided into frequency bands called ***critical bands***. When a sound wave arrives at the ear, it is perceived by a section—that is, a band—of the inner ear that responds to that wave's particular frequency. Within a small window of time, sound waves with neighboring frequencies—which must be sensed by the ear's same critical band—have an effect on each other in that the loudest frequency sound can overpower the others in the band. This phenomenon is called ***masking***. Critical bands are narrower for low- than for high-frequency sounds. Between 1 and 500 Hz, bands are about 100 Hz in width. The critical band at the highest audible frequency is over 4000 Hz wide. Think about the implications of relatively narrow bands at low frequencies: Narrow bands imply that that interference among frequencies occurs over a narrower range, which explains why we have greater frequency resolution at low frequencies.

The phenomenon of critical bands is one of the most important in perceptual encoding, since it gives us a basis for eliminating sound information that is not perceived anyway. The masking phenomenon is pictured in Figure 5.48. Recall from Chapter 1 that the limit below which you can't hear is called the threshold of hearing. Masking causes the threshold of hearing to change in the presence of a dominant frequency component in a band. For example, a 500 Hz frequency component is easy to hear, but when a 400 Hz tone occurs at about the same time and at sufficient amplitude, it can mask the 500 Hz component. Another way to say this is that 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 frequency is called a ***masking frequency*** or ***masking tone***. As shown in Figure 5.48, the threshold of hearing for all the frequencies within the critical band of the masking frequency are raised, indicated by the ***masking threshold*** in the figure. All frequencies that appear at amplitudes beneath the masking threshold will be inaudible. This graph represents the effect of just one masking tone within one critical band at one window in time. These effects happen continuously over time, over all the critical bands. The important point to note is that frequencies that are

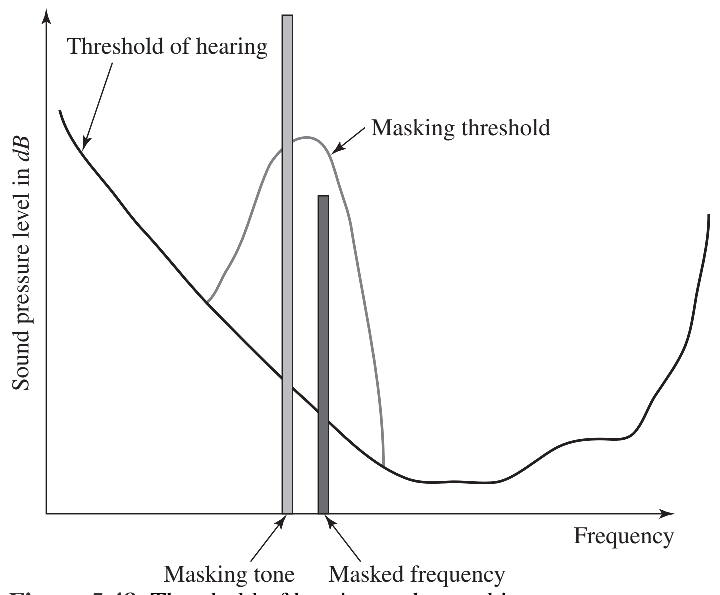


Figure 5.48 Threshold of hearing and a masking tone

masked out in an audio signal cannot be heard, so they do not need to be stored. Eliminating masked information results in compression.

Here's a sketch of how the masking phenomenon can be applied in compression: A small window of time called a ***frame*** is moved across a sound file, and the samples in that frame are compressed as one unit. Spectral analysis by means of a ***filter bank*** divides each frame into bands of frequencies; 32 is often the number of bands used. 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. This is done by determining the lowest possible bit depth that can be used for the band such that the resulting quantization noise 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 at the bit depth determined for each band.

In the sections that follow, we will use MPEG compression to illustrate how perceptual encoding is implemented. MPEG coding has been one of the dominant audio compression methods since the 1990s. Other important audio coding schemes exist that we won't examine here. For example, AC3-Dolby audio is a widely used audio codec that is used in DVD video, digital TV, and video playstations. Based on perceptual encoding, AC3-Dolby uses many of the techniques used in MPEG compression.

5.7.3 MPEG Audio Compression

5.7.3.1 Overview of MPEG

MPEG, an acronym for the ***Motion Picture Experts Group***, represents a family of compression algorithms for both digital audio and video. In Chapter 7, we'll cover MPEG video compression algorithms. In this section we focus only on the audio portion of MPEG.

Standardized by ISO/IEC, MPEG has been developed in a number of phases since the 1990s: MPEG-1 (1991), MPEG-2 (1994), MPEG-4 (1998), and MPEG-7 (2003). Each is targeted at a different aspect of multimedia communication. ***MPEG-1*** covers CD quality and audio suitable for video games. ***MPEG-2*** covers multichannel surround sound. ***MPEG-4*** covers a wide range of audio including simple voice, MIDI, high-end audiophile sound, and interactive systems. ***MPEG-7***, also called Multimedia Content Description, supports searching and filtering of multimedia data. The phases of the MPEG audio standard are also divided into three layers, ***Audio Layers I, II, and III***. The compression algorithms increase in sophistication from Layer I to Layer III, requiring more computation for encoding and decoding, but generally with better playback quality in proportion to the bit depth. The naming convention is a little confusing, but just remember that the Arabic numerals indicate phases and the Roman numerals indicate layers within a phase. (Unfortunately, not all sources adhere to this distinction.) Sometimes abbreviations are used. The well-known ***MP3*** audio file format is actually ***MPEG-1 Audio Layer III***.

ASIDE: The term ***version*** is sometimes used to distinguish MPEG-1, 2, 4, and 7, but this isn't the most appropriate term. Versions are generally implementations that are developed over time, each with improvements and additions over the previous. MPEG phases have different purposes, targeted at different systems and types of multimedia.

AAC compression, which stands for ***Advanced Audio Coding***, is not standardized as a layer, but it is available in both MPEG-2 and MPEG-4. AAC has been implemented in proprietary form as AT&T's a2b music and as Liquid Audio. It supports a wide range of sampling rates and number of channels with excellent quality at relatively low bit rates.

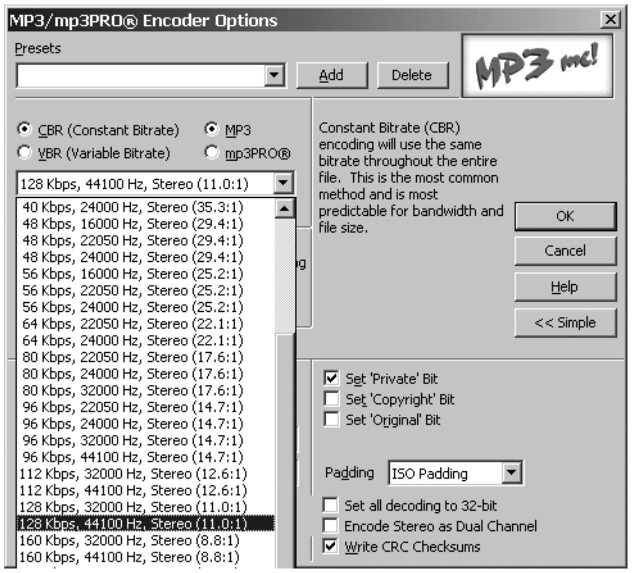
You may find yourself confused by the various bit rates that you'll see cited for different implementations of MPEG audio. The bit rate tells you how much data there is per second of compressed audio. Think about what might cause this number to be relatively larger. If you start with a higher sampling rate and bit depth, you're going to get a higher bit rate. For example, a sampling rate of 48 kHz results in a higher bit rate than a sampling rate of 44.1 kHz. Also, more stereo channels yield a higher bit rate. Finally, a compression algorithm that is less lossy gives a higher bit rate. A range of bit rates is associated with each MPEG audio layer because the layer's compression method can be applied to audio files of different sampling rates, either mono or stereo. In Table 5.4, for example, the lowest bit rate cited for MPEG-1 results from compressing 32 kHz mono signals. The highest bit rate in MPEG-1 Layer I (448 kb/s) corresponds to compressing a 48 kHz stereo file. The fact that the maximum bit rate in a layer gets smaller from Layer I to Layer III is a result of the greater complexity of the Layer III compression algorithm compared to the Layer I algorithm—that is, the Layer III can compress more without loss of quality, but it takes more time to do so.

TABLE 5.4 Phases and Layers of MPEG Audio Compression

Version	Data Rates	Applications	Channels	Sampling Rates Supported
MPEG-1		CD, DAT, ISDN, video games, digital audio broadcasting, music shared on the web, portable music players	mono or stereo	32, 44.1, and 48 kHz
Layer I	32–448 kb/s			
Layer II	32–384 kb/s			
Layer III	32–320 kb/s			
MPEG-2 LSF (Low Sampling Frequency)		audio files with low sampling frequency	mono or stereo	16, 22.05, and 24 kHz
Layer I	32–256 kb/s			
Layer II	8–160 kb/s			
Layer III	8–160 kb/s			
MPEG-2 Multichannel		multichannels, multilingual extensions	5.1 surround	32, 44.1, and 48 kHz
Layer I	max 448 kb/s			
Layer II	max 384 kb/s			
Layer III	max 32 kb/s			
AAC in MPEG-2 and MPEG-4 (Advanced Audio Coding)	8–384 kb/s	multichannels, music shared on the web, portable music players, cell phones	up to 48 channels	32, 44.1, and 48 kHz and other rates between 8 and 96 kHz

The amount of time needed for encoding and decoding has the greatest impact on real-time audio, such as for two-way voice conversations. In general, a delay of more than 10 milliseconds can be disturbing in a voice conversation. MPEG audio encoding delays are on the order of between 15 milliseconds for Layer I and 60 milliseconds for Layer III, dependent on the hardware and the software implementation.

The compression rates that you may see cited for different layers of MPEG give you the general picture that higher layers yield better compression. For example, MPEG-1 Layer 1 is cited as having a compression rate of about 4 : 1, Layer 2 as about 7 : 1, and Layer III as about 11 : 1. However, as with bit rates, compression rates don't mean much unless you consider what type of audio is being compressed. To get a sense of the compression rate, consider that uncompressed CD-quality stereo requires 32 bits per sample * 44,100 samples/s = 1.4112 Mb/s. Compressing CD-quality stereo to 128 kb/s using MP3 encoding yields a



[✓ CBR (Constant Bitrate)] [✓ MP3]
VBR (Variable Bitrate) mp3PRO®

128 Kbps, 44100 Hz, Stereo (11.0:1)

40 Kbps, 24000 Hz, Stereo (35.3:1)
48 Kbps, 16000 Hz, Stereo (29.4:1)
48 Kbps, 22050 Hz, Stereo (29.4:1)
48 Kbps, 24000 Hz, Stereo (29.4:1)
56 Kbps, 16000 Hz, Stereo (25.2:1)
56 Kbps, 22050 Hz, Stereo (25.2:1)
56 Kbps, 24000 Hz, Stereo (25.2:1)
64 Kbps, 22050 Hz, Stereo (22.1:1)
64 Kbps, 24000 Hz, Stereo (22.1:1)
80 Kbps, 22050 Hz, Stereo (17.6:1)
80 Kbps, 24000 Hz, Stereo (17.6:1)
80 Kbps, 32000 Hz, Stereo (17.6:1)
96 Kbps, 22050 Hz, Stereo (14.7:1)
96 Kbps, 24000 Hz, Stereo (14.7:1)
96 Kbps, 32000 Hz, Stereo (14.7:1)
96 Kbps, 44100 Hz, Stereo (14.7:1)
112 Kbps, 32000 Hz, Stereo (12.6:1)
112 Kbps, 44100 Hz, Stereo (12.6:1)
128 Kbps, 32000 Hz, Stereo (11.0:1)
[✓ 128 Kbps, 44100 Hz, Stereo (11.0:1)]
160 Kbps, 32000 Hz, Stereo (8.8:1)
160 Kbps, 44100 Hz, Stereo (8.8:1)

Constant Bitrate (CBR) encoding will use the same bitrate throughout the entire file. This is the most common method and is most predictable for bandwidth and file size.

[✓ Set 'Private' Bit]
Set 'Copyright' Bit
Set 'Original' Bit

Padding: ISO Padding

Set all decoding to 32-bit
Encode Stereo as Dual Channel
[✓ Write CRC Checksums]

Figure 5.49 MP3 compression bit rate options

compression rate of 11 : 1. But MP3 can also be compressed at 192 kb/s, which is a rate of about 7 : 1. You can see that to generate specific compression rates, you need to compare the initial bit rate with the final bit rate. To some extent, the final bit rate is your own choice. When you compress an audio file with an MPEG compressor, you get a list of bit rate options, as shown in Figure 5.49. The lower the bit rate, the more compression you get, but at the sacrifice of some sound fidelity. Notice that you can also choose between ***constant bit rate*** (***CBR***) and ***variable bit rate*** (***VBR***). CBR uses the same number of bits per second regardless of how complex the signal is. VBR uses fewer bits for passages with a smaller dynamic range, resulting overall in a smaller file.

The MPEG audio layer and bit rate that is best for a given application depends on the nature of the audio files being compressed and how high a bit rate the user can tolerate. For example, MPEG-1 Layer 2 (sometimes referred to as MP2) gives better quality sound than MP3 if you don't mind relatively higher bit rates, and it requires lower encoding delays. For these reasons, it is widely used for digital radio. MP3 compresses well and has been extremely popular for music files that are shared through the web. The usual MP3 bit rate of 128 or 192 kb/s creates reasonable-size files that don't take too much time to download and that have good quality for playback.

MPEG layers are backward compatible so that, for example, a Layer III decoder can decode a Layer I stream. MPEG-2 decoders are backward compatible with MPEG-1. Proprietary implementations of MPEG such as AT&T's a2b and Liquid Audio are not mutually compatible.

5.7.3.2 MPEG-1 Audio Compression

We focus on MPEG-1 compression because it illustrates perceptual encoding concepts well, and it can be explained understandably in a fairly short space. MPEG-2 and AAC use many of the same techniques as those described in this section, but with additional refinements to further improve the audio quality and handle other special applications.

None of the MPEG phases prescribe an encoding algorithm. Instead, they offer psychoacoustical models of audio compression. The MPEG-1, for example, proposes two psychoacoustical models, one more complex and offering higher compression than the other. What

ALGORITHM 5.2

```
algorithm MPEG-1_audio
/*Input: An audio file in the time domain
  Output: The same audio file, compressed*/
{
    Divide the audio file into frames
    For each frame {
        By applying a bank of filters, separate the signal into frequency bands.
        For each frequency band {
            Perform a Fourier transform to analyze the band's frequency spectrum
            Analyze the influence of tonal and nontonal elements (i.e., transients)
            Analyze how much the frequency band is influenced by neighboring bands
            Find the masking threshold and signal-to-mask ratio (SMR) for the band,
            and determine the bit depth in the band accordingly
            Quantize the samples from the band using the determined bit depth
            Apply Huffman encoding (optional)
        }
        Create a frame with a header and encoded samples from all bands
    }
}
```

is dictated by the standard is how the bit stream must look to the decoder. Implementers can create whatever ingenious algorithms they can imagine, as long as the compressed bit stream is in the format the decoder expects. The advantage to this is that algorithms can be refined and improved, and existing decoders will still work on the new implementations.

Algorithm 5.2 and Figure 5.50 show the basic steps common to most implementations of MPEG-1 compression, which is sufficient to give the flavor of MPEG compression in general. First, try to get an overview of the method, and then you can focus on some of the motivations and details of each step.

1. Divide the audio file into frames and analyze the psychoacoustical properties of each frame individually.

Motivation: Each frame covers a sequence of samples in a small window of time. To analyze how one frequency component might mask neighboring ones, it's necessary to look at samples that are close to each other in time. The masking phenomenon happens only when different frequencies are played at close to the same time.

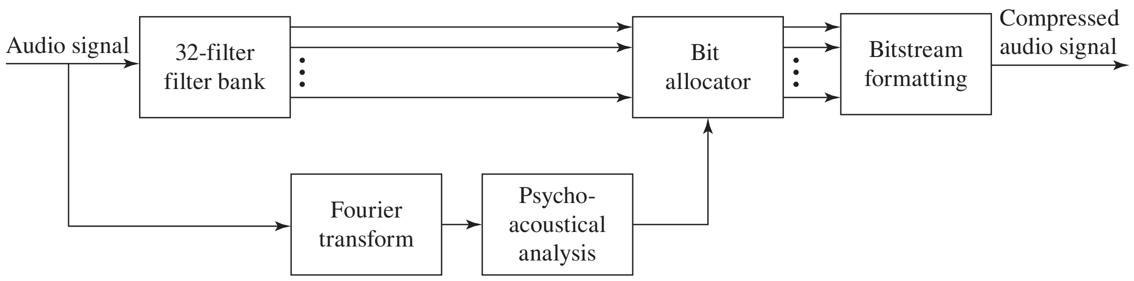


Figure 5.50 MPEG audio compression

Details: Frames can contain 384, 576, or 1152 samples, depending on the MPEG phase and layer. It may seem odd that these numbers are not powers of two. However, when frequency analysis is done on these samples, a larger window is placed around the sample, and the window *is* a power of two.

For the remainder of these steps, it is assumed that we're operating on an individual frame.

2. By applying a bank of filters, separate the signal into frequency bands.

Motivation: The samples are divided into frequency bands because the psychoacoustical properties of each band will be analyzed separately. Each filter removes all frequencies except for those in its designated band. Once the psychoacoustical properties of a band are analyzed, the appropriate number of bits to represent samples in that band can be determined.

Details: The use of filter banks is called ***subband coding***. Say that there are n filters in the filter bank. n copies of the signal are generated, and a copy is sent through each of the filters. (In MPEG-1, for example, $n = 32$.) Time-domain bandpass filters are applied such that each filter lets only a range of frequencies pass through. Note that the n frequency bands are still represented in the time domain—that is, they contain samples of audio consecutive points in time. Also, the amount of data does *not* increase as a result of dividing the frame into bands. If, for example, 384 samples entered the 32 filters, each filter produces 12 samples. There is no increase in data.

The previous section on FIR and IIR filters shows that bandpass filters are never ideal in that you cannot perfectly isolate the band of filters you want. The frequency response graph is never a perfect rectangle like the one pictured in Figure 5.42, with a sharply vertical cutoff between the desired frequencies and the filtered-out frequencies. Instead, the cutoff between the frequencies is a sloped line, called an ***attenuation slope***. For this reason, the frequency bands overlap a little. If this overlap is not accounted for, then there will be aliasing when the signal is reconstructed. ***Quadrature mirror filtering*** (***QMF***) is a technique that eliminates the aliasing. It does so by making the attenuation slope of one band a mirror image of the attenuation slope of the previous band so that the aliased frequencies cancel out when the signal is reconstructed. This is pictured in Figure 5.51.

In MPEG-1 Layers I and II, the frequency bands created by the filter banks are uniform in size. This means that their width doesn't match the width of the critical bands in

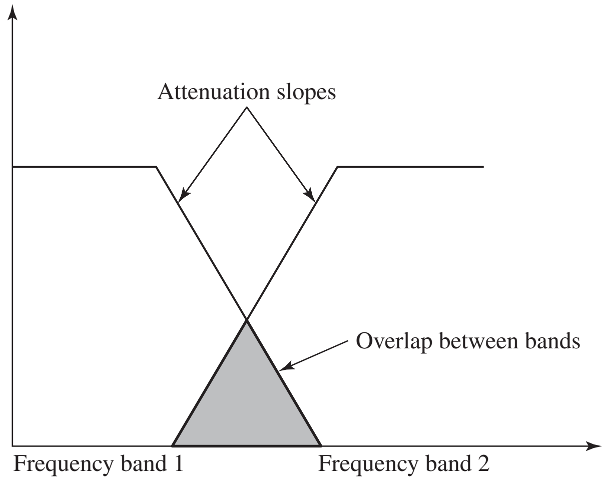


Figure 5.51 Quadrature mirror filter (QMF)

human hearing, which are wider at high frequencies. However, the way in which the frequency components of a band are analyzed can compensate somewhat for this disparity. In MP3 encoding, the ***modified discrete cosine transform*** (***MDCT***) is used to improve frequency resolution, particularly at low-frequency bands, thus modeling the human ear's critical bands more closely. Also, MP3 uses nonlinear quantization, where (like in μ-law encoding) the quantization intervals are larger for high amplitude samples.

3. Perform a Fourier transform on the samples in each band in order to analyze the band's frequency spectrum.

Motivation: The filters have already limited each band to a certain range of frequencies, but it's important to know exactly how much of each frequency component occurs in each band. This is the purpose of applying a Fourier transform. From the frequency spectrum, a masking curve can be produced for each band. Then the number of bits with which to encode each band can be chosen in a way that pays attention to the elevated noise floor produced from masking.

Details: Another copy of each frame, called a ***sidechain***, is created so that the Fourier transform can be applied. A 512 or 1024-sample window is used. You may notice that this is not the same size as the number of samples in a frame. In fact, the Fourier transform is done on a window that surrounds the frame samples. For example, if a frame has 384 samples, 1024 samples that encompass this window are used for the Fourier transform. If the frame has 1152 samples, two 1024-sample Fourier transforms are done, covering two halves of the frame.

4. Analyze the influence of tonal and nontonal elements in each band. (Tonal elements are simple sinusoidal components, such as frequencies related to melodic and harmonic music. Nontonal elements are transients like the strike of a drum or the clapping of hands.)

Motivation: It would not be good to allow masking to eliminate nontonal elements that punctuate an audio signal in meaningful ways. Also, if nontonal elements are not compressed properly, ringing or pre-echo effects can result.

Details: The longer the time window is in frequency analysis, the better the frequency resolution, but the worse the time resolution. The effect of poor time resolution is that sound elements that occur very close to one another in time might not be distinguished sufficiently, one from the other. This in turn can mean that transient signals are not properly identified. Layers II and III have a longer time window than Layer I, so they have better time resolution.

5. Determine how much each band's influence is likely to spread to neighboring frequency bands.

Motivation: It isn't sufficient to deal with bands entirely in isolation from each other, since there can be a masking effect between bands.

Details: Empirically determined spreading functions can be applied.

6. Find the masking threshold and *signal-to-mask ratio* (*SMR*) for each band, and determine the bit depth of each band accordingly.

Motivation: Within each band, a high amplitude frequency component can mask other frequency components. The precision with which samples need to be represented depends on the ratio of the loudest sample among them to the amplitude below which they can't be heard, even if they are present. This is in fact the definition of SMR: the ratio between the peak sound pressure level and the masking threshold is the SMR (Figure 5.52). This ratio determines how many bits are needed to represent samples within a band.

Details: The main idea you should get from this is that the masking phenomenon causes the noise floor within a band to be raised. The number of bits per sample varies from band

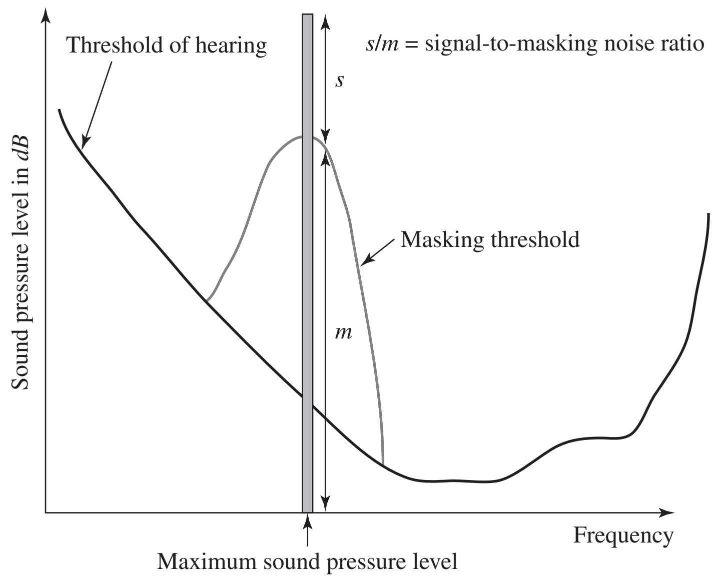

Threshold of hearing
s/m = signal-to-masking noise ratio
s
Masking threshold
m
Sound pressure level in dB
Frequency
Maximum sound pressure level

Figure 5.52 Signal-to-masking noise ratio

to band depending on how high the noise floor is. When the noise floor is high relative to the maximum sound pressure level within a band, then fewer bits are needed. Fewer bits create more quantization noise, but it doesn't matter if that quantization noise is below the masking threshold. The noise won't be heard anyway.

MP3 has interesting variation in its design, allowing for a ***bit reservoir***. In a constant bit rate (CBR) encoding, all frames have the same number of bits at their disposal, but not all frames need all the bits allocated to them. A bit reservoir scheme makes it possible to take unneeded bits from a frame with a low SMR and use them in a frame with a high SMR. This doesn't increase the compression rate (since we're assuming CBR), but it does improve the quality of the compressed audio.

7. Quantize the samples for the band with the appropriate number of bits, possibly following this with Huffman encoding.

Motivation: Some bit patterns occur more frequently than others. Huffman encoding allows the frequently occurring bit patterns to be encoded with fewer bits than patterns that occur infrequently.

Details: MPEG-1 Layers 1 and 2 use linear quantization, while MP3 uses nonlinear.

After quantization, the values can be scaled so that the full dynamic range offered by the bit depth is used. The scale factor then has to be stored in the frame.

In MP3 encoding, the frame data is reordered after quantization and divided into regions so that Huffman tables can be applied. These empirically generated tables reflect the probabilities of certain bit patterns in each region.

A refinement in the above algorithm is related to the compression of stereo information. There are a number of different ways to compress stereo. The ***joint stereo*** compression method allows one stereo channel to carry the information that is identical in both channels, while the other channel carries the differences. This is a lossless method that gives fairly good compression. The ***intensity stereo*** compression method replaces a left and right signal with one signal plus a vector representing directional information. This yields the highest stereo compression rate, but obviously it is a lossy method. ***Mid/side stereo coding***

Header (Sync word, phase and layer #*s*, bit rate, sampling frequency, etc.)	**CRC (Layers i and ii)**	**Bit allocation by band**	**Scale factor by band**	**Sample values**	**Extra info** (Artist, album, year, etc.)

Figure 5.53 MPEG bit stream

calculates a middle channel as (left + right)/2 and side channel as (left − right)/2, which then is encoded with fewer bits than straightforward stereo representation. Mid/side stereo coding has a less detrimental effect on phase information than intensity stereo does.

The MPEG standard defines how a compressed bit stream must look. The compressed signal is divided into frames. A sketch of an MPEG frame is shown in Figure 5.53. Each frame begins with a sync word to signal that this is the beginning of a new frame. The header then tells the bit rate, sampling frequency, mono or stereo mode, and so forth. CRC stands for ***cyclic redundancy check***, which is an error-checking algorithm to ensure that no errors have been inserted in transmission. Following the CRC, the number of bits used in each band is listed. Then the scale factor for each band is given. The largest part of the frame is devoted to the samples. Extra information may be added to a frame in the form of ID3 tags. ID3 is an informal standard for giving information about music files being compressed, including copyright, artist, title, album name, lyrics, year, and web links.

5.7.3.3 MPEG-2, AAC, and MPEG-4

The MPEG-2 audio standard is complicated by the fact that it is divided into three formats: ***MPEG-2 Multichannel***, ***MPEG-2 LSF*** (***low sampling frequency***), and ***MPEG-2 AAC*** (Advanced Audio Coding). The first two of these are further subdivided into Layers, as shown in Table 5.4.

MPEG-2 Multichannel, as the name indicates, has features to accommodate more than one channel—up to ***5.1 stereo surround sound***. This type of surround sound allows music to be played through three front speakers, two back speakers, and one subwoofer for bass tones. MPEG-2 Multichannel is backward compatible with MPEG-1 so that an audio file compressed with an MPEG-1 encoder can be played by an MPEG-2 decoder.

MPEG-2 LSF allows sampling rates of 16, 22.05, and 24 kHz. It is sometimes called MPEG-2.5. This format offers better frequency resolution, and hence better quality in the compressed file, if lower sampling rates can be used.

A third format—MPEG-2 AAC—was designed to take advantage of newly developed psychoacoustical models and corresponding compression techniques. Applying these techniques, however, meant that the AAC could not be backward compatible with MPEG-1. Thus, AAC was originally referred to as MPEG-2 NBC (not backward compatible), while MPEG-2 Multichannel was called MPEG-2 BC.

AAC handles multiple channels up to 5.1 surround sound, compressing down to a bit rate of 64 kb/s per channel, which yields 384 kb/s total. AAC achieves this bit rate with even better quality than MPEG-2 Multichannel. Listening tests show, for example, that a music file compressed as MPEG-2 AAC at, say 96 kb/s sounds better than the same one compressed as MP3 at 128 kb/s. Because of its good quality, AAC has gained popularity for use in portable music players and cell phones, two of the biggest markets for compressed audio files.

The AAC psychoacoustical model is too complicated to describe in detail in a short space. The main features include the following:

- The modified discrete cosine transform improves frequency resolution as the signal is divided into frequency bands.
- Temporal noise shaping (TNS) helps in handling transients, moving the distortion associated with their compression to a point in time that doesn't cause a pre-echo.
- Predictive coding improves compression by storing the difference between a predicted next value and the actual value. (Since the decoder uses the same prediction algorithm, it is able to recapture the actual value from the difference value stored.)
- Not all scale factors need to be stored since consecutive ones are often the same. AAC uses a more condensed method for storing scale factors.
- A combination of intensity coding and mid/side stereo coding is used to compress information from multiple channels.
- A model for low-delay encoding provides good perceptual encoding properties with the small coding delay necessary for two-way voice communication.

The AAC compression model is a central feature of both MPEG-2 and MPEG-4 audio. MPEG-4, however, is a broad standard that also encompasses DVD video, streaming media, MIDI, multimedia objects, text-to-speech conversion, score-driven sound synthesis, and interactivity.

We refer you to the sources at the end of this chapter and Chapter 4 if you're interested in more implementation details of AAC, MPEG-4, and the other compression models.

EXERCISES AND PROGRAMS

1. Say that the first 30 values of a digital audio file, $\boldsymbol{x}(n)$, are those given below, and you apply the nine-tap FIR filter, $\boldsymbol{h}(n)$, below. What is the value of the 10th digital sample after filtering?

$\boldsymbol{x}(n) = [138, 232, 253, 194, 73, -70, -191, -252, -233, -141, -4, 135, 230, 253, 196, 77, -66, -189, -251, -235, -144, -7, 131, 229, 253, 198, 80, -63, -186, -251]$

$\boldsymbol{h}(n) = [0.0788, 0.1017, 0.1201, 0.1319, 0.1361, 0.1319, 0.1201, 0.1017, 0.0788]$

2. Say that you have an IIR filter $\boldsymbol{y}(n) = \sum_{k=0}^{N-1} a_k \boldsymbol{x}(n-k) - \sum_{k=1}^{M} b_k \boldsymbol{y}(n-k)$ where $a = [0.389, -1.558, 2.338, -1.558, 0.389]$ and $b = [2.161, -2.033, 0.878, -0.161]$. Note that vector **a** is numbered starting at index 0 and vector **b** is numbered starting at index 1.
To find $\boldsymbol{h}(n)$ for this filter so that you can define it with

$$\boldsymbol{y}(n) = \boldsymbol{h}(n) \otimes \boldsymbol{x}(n) = \sum_{k=0}^{\infty} \boldsymbol{h}(k)\boldsymbol{x}(n-k),$$

you can evaluate a unit impulse run through the function $\boldsymbol{y}(n) = \sum_{k=0}^{N-1} a_k \boldsymbol{x}(n-k) - \sum_{k=1}^{M} b_k \boldsymbol{y}(n-k)$. Do this to get $\boldsymbol{h}(n)$. You'll have to give it a finite length. Try 32 taps. You might want to write a program to do the computation for you. Graph the impulse response that you get.

3. Say that you have a graphic equalizer that allows you to divide the frequency spectrum into 30 bands, each separated by a third of an octave. If the first band is at 31 Hz, where are the remaining 29 bands, in Hz?
4. For a peaking filter, assume that you want the center frequency of the curve defining the peak to be at 8000 Hz, and you want the bandwidth of the peak to be 1000 Hz.
   a. What is the Q-factor?
   b. What is the corresponding bandwidth of this filter, in octaves?
   c. What is the Q-factor of a peaking filter with a central frequency of 2000 Hz and a bandwidth of 1000 Hz?
   d. What is the corresponding bandwidth of the filter, in octaves?
   e. What is Q if you want the peaking filter to span one octave?
5. The step response of a continuous filter is the integral of the filter—the area under the curve that graphs the filter. Thus, the step response of a discrete filter would be the sum of the discrete points that define the filter. Create the impulse response of a low-pass filter, or find some values in another source. (See the online worksheets for how to create an FIR filter.) Then determine the corresponding step response and graph it. Describe what the step response tells you about the filter, with regard to the steepness of the step.
6. Dynamics processing, interactive tutorial and worksheet, online
7. Windowing method for FIR filter design, mathematical modeling worksheet, online
8. Z-transforms, zero-pole diagrams, and filters, interactive and worksheet online
9. Creating FIR and IIR filters with MATLAB.
10. Creating a filter via a transfer function, mathematical modeling worksheet, online

APPLICATIONS

1. Examine the specifications of your sound card (or the one you would like to have). What kind of input and output ports does it have? What is its signal-to-noise ratio? Does it handle MIDI? If so, how?
2. Examine the specifications of your microphone (or the one you would like to have). Is a dynamic mic, a condenser mic, or some other kind? Does your microphone require power? What frequencies does it handle best? Do you need different mics for different purposes?
3. If you have access to more than two microphones with significantly different characteristics, try making similar recordings with both of them. Listen to the recordings to see if you can hear the difference. Then look at the frequency spectrum of the recordings and analyze them. Are the results what you expect, based on the specifications of the mics?
4. Create a one-minute radio commercial for a hypothetical product, mixing voices, music, and sound effects. Record the commercial at your digital audio workstation using multitrack editing software. Record and edit the different instruments, voices, and so forth on different tracks. Experiment with filters, dynamics processing, and special effects. When you've finished, mix down the tracks, and save and compress the audio in an appropriate file type. Document and justify your steps and choices of sampling rate, bit depth, editing tools, compression, and final file type.
5. If you don't know already, find out what loops are. Find a source of free online loops. Create background music for a hypothetical computer game by combining and editing loops. (At the time of this writing, Acid Loops is an excellent commercial program for working with loops. Royalty Free Music is a source for free loops.) If you create a computer game when you get to Chapter 8, you can use your music in that game.

6. See if you can replicate the impulse response method of creating a convolution filter for reverb that makes an audio file sound like it was recorded in a certain acoustical space. (If you do a web search, you should be able to find descriptions of the details of this method. Look under "convolution reverb." For example, at the writing of this chapter, a paper entitled "Implementation of Impulse Response Measurement Techniques" by J. Shaeffer and E. Elyashiv could be found at www.acoustics.net/objects/pdf/IR-paper.pdf.)

Examine the features of your digital audio processing program or programs and try the exercises below with features that are available.

7. What file types are supported by your audio editing software? What compressions methods?
8. Make a simple audio file and intentionally put a click or pop in it. See if you can remove the click or pop with your audio editing program.
9. Examine the tools for dynamics processing offered in your audio editing program. Try compressing the dynamic range of an audio file. Then try expansion. Experiment with different attack and release times and listen to the differences.
10. Examine the types of filters offered in your audio editing program. See if your software has a graphic equalizer, parametric EQ, paragraphic EQ, notch filters, and/or convolution. Can you tell if they are implemented with FIR or IIR filters? What parameters do you have to set to use these filters?
11. Examine the types of reverb, echo, and delay effects available in your audio editing program and experiment with the effects.

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

REFERENCES

Print Publications

Coulter, Doug. *Digital Audio Processing*. Lawrence, KS: R & D Books, 2000.

Embree, Paul M., and Damon Danieli. *C++ Algorithms for Digital Signal Procssing*. 2nd ed. Upper Saddle River, NJ: Prentice Hall, 1999.

Noll, Peter. Sept. 1997. "MPEG Digital Audio Coding." *IEEE Signal Processing Magazine*, 14 (5): 59–81.

Pan, Davis. 1995. "A Tutorial on MPEG/Audio Compression." *IEEE Multimedia* 2 (2): 60–74.

Phillips, Dave. *The Book of Linux Music & Sound.* San Francisco: No Starch Press/Linux Journal Press, 2000.

Smith, Steven W. *The Scientist and Engineer's Guide to Digital Signal Processing*. San Diego: California Technical Publishing, 1997.

Weeks, Michael. *Digital Signal Processing Using MATLAB and Wavelets.* Hingham, MA: Infinity Science Press LLC, 2007.

See Chapter 4 for additional references on digital audio.

Websites

aRts-project.
http://www.arts-projects.org.

Audacity.
http://jackit.sourceforge.net.