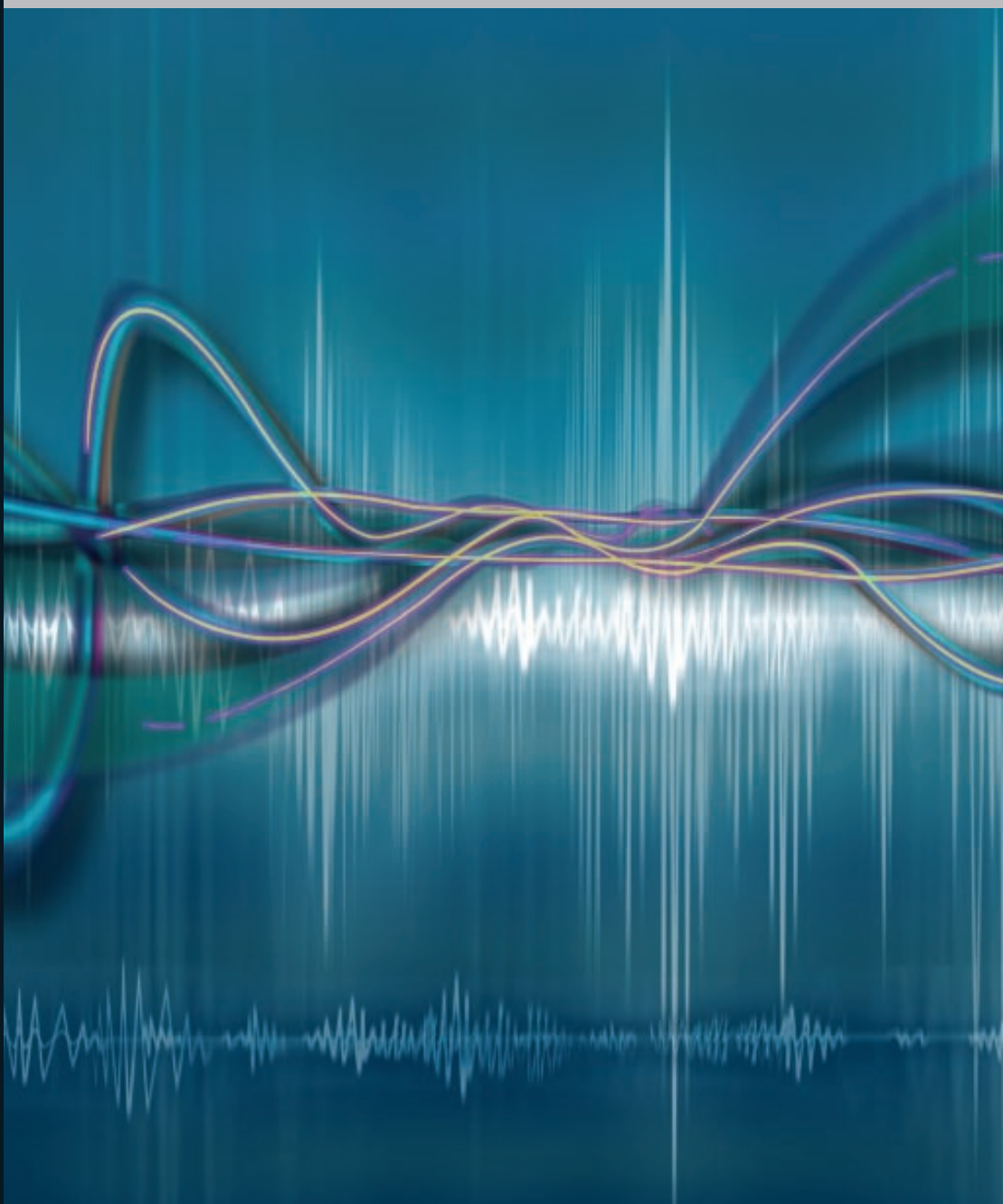
EURASIP Book Series on Signal Processing and Communications

High-Fidelity Multichannel Audio Coding

Dai Tracy Yang, Chris Kyriakakis, and C.-C. Jay Kuo



High-Fidelity Multichannel Audio Coding

EURASIP Book Series on Signal Processing and Communications
Editor-in-Chief: K. J. Ray Liu

Hindawi Publishing Corporation
410 Park Avenue, 15th Floor, #287 pmb, New York, NY 10022, USA
Nasr City Free Zone, Cairo 11816, Egypt
Fax: +1-866-HINDAWI (USA toll-free)

Cover Image: Mehau Kulyk/Science Photo Library

ISBN 977-5945-24-0

EURASIP Book Series on Signal Processing and Communications, Volume 2

High-Fidelity Multichannel Audio Coding

Dai Tracy Yang, Chris Kyriakakis, and C.-C. Jay Kuo

Hindawi Publishing Corporation
http://www.hindawi.com

Dedication

To Ruhua, Joshua, Junhui, and Zongduo
— Dai Tracy Yang

To Wee Ling, Anthony, and Alexandra
— Chris Kyriakakis

To Terri and Allison
— C.-C. Jay Kuo

Preface

Audio is one of the fundamental elements in multimedia signals. Audio signal processing has attracted attention from researchers and engineers for several decades. By exploiting unique features of audio signals and common features of all multimedia signals, researchers and engineers have been able to develop more efficient technologies to compress audio data. Although books on digital audio have been available some time, the subject of multichannel audio coding techniques has not yet been addressed in great detail.

With many years of teaching and research in the field of digital audio signal processing and digital audio compression, we see a need for an advanced audio coding book that covers recent developments in this field. When we started this book project, we had a smaller scope. Our objective was to present several innovative compression techniques for multichannel audio sources and publish it as a research monograph. However, after the first draft, we received valuable comments from our colleagues and anonymous reviewers. With their encouragement, we decided to extend the coverage of the book by including more background material to make it a senior undergraduate or a graduate level textbook on advanced audio coding techniques. Special thanks also go to Dr. Hongmei Ai for her valuable discussions and suggestions when we developed and tested our new audio coding algorithms.

This book includes three parts. The first part covers the basic topics on audio compression, such as quantization, entropy coding, psychoacoustic models, and sound quality assessment. The second part of the book highlights the current most prevalent low-bit-rate high-performance audio coding standard—MPEG-4 Audio. More emphasis is given to the audio standards that are capable of supporting multichannel signals, that is, MPEG Advanced Audio Coding (AAC), including the original MPEG-2 AAC specification, additional MPEG-4 toolsets, and the most recent aacPlus standard. The third part of this book introduces several innovative multichannel audio coding methods, which can further improve the coding performance and expand the available functionalities of MPEG AAC. This section is more suitable for graduate students and researchers.

Dai Tracy Yang, Chris Kyriakakis,
and C.-C. Jay Kuo
Los Angeles, CA
August 17, 2005

Contents

1 Introduction to digital audio

Digital audio pushes the development of many diverse engineering and manufacturing disciplines. Although the basic underlying concepts of digital audio have been well understood since the 1920s, mature technology for commercialized digital audio was not ready for 50 years until the 1970s. Because of the complex property of the digital audio, we would like to start our discussion from the basic topics. Particularly, the information about the digital audio codec, audio signal processing, and multichannel audio will be introduced in this chapter.

1.1. Digital audio coding

Before the introduction of digital audio, audio signals have been represented in analog form. Analog signals are often stored in audio tapes and disks. As compared with digital audio, analog audio has several main disadvantages.

(1) *Compression.* It is much more difficult to compress analog audio effectively, that is, minimizing the file size for storage and/or communication.

(2) *Rendering.* Analog audio is vulnerable to sound rendering. Each time we copy or process analog audio signals, we add noise to the original signals so that the quality of the sound degrades.

(3) *Quality enhancement.* Analog audio is broadcast by amplitude modulation (AM) and frequency modulation (FM) techniques today. The received audio quality is usually poor due to transmission channel noise. It is relatively difficult and expensive to apply sophisticated analog signal processing techniques for audio quality enhancement.

Representing audio signals in digital form allows us to achieve the above goals more easily. In this section, we will provide an overview on digital audio representation and coding.

1.1.1. Representing digital audio signals

The most prevalent audio representation is to store the sound as a set of amplitude samples, that is, to represent the volume of the sound as a changing function over time. The idea behind digital audio is to use numbers to represent the physical sound via an analog-to-digital (A/D) conversion process. Figures 1.1(a)

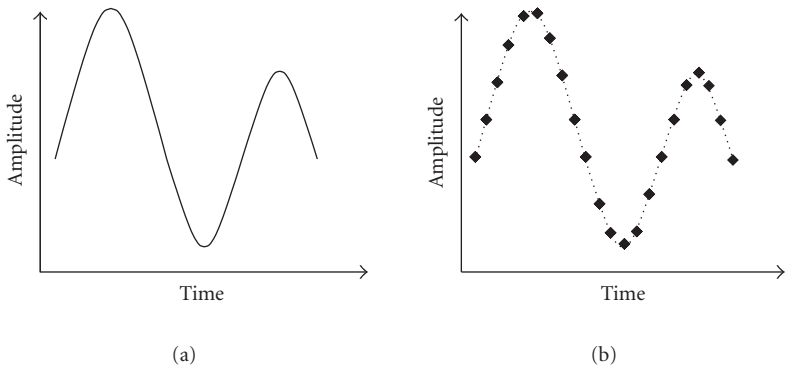


FIGURE 1.1. (a) Analog signal representation and (b) digital signal representation.

and 1.1(b) show a signal represented in analog and digital forms, respectively, where each diamond in Figure 1.1(b) represents one sample. The A/D conversion process involves sampling and quantization in two main steps as detailed below.

1.1.1.1. Sampling period and frequency

We need to store each sample's amplitude as a function of a discrete index. Furthermore, we should have the relative time information of these samples so that we know the proper time for a given amplitude of the sound to occur when digital signals are reproduced. Typically, the time interval between two adjacent samples remains constant. We call the rate at which each sample is extracted the *sampling frequency* or the *sampling rate*, which is described in terms of number of samples per second, or Hertz (Hz). Commonly adopted sampling frequencies include 8 kHz, 16 kHz, 22.05 kHz, 24 kHz, 32 kHz, 44.1 kHz, and 48 kHz. Generally speaking, the higher a sampling frequency is, the better the reproduction of the digital sound can achieve. Lower sampling rates are usually used in telephone quality sound or speech signals. Higher sampling rates are widely used in compact disk (CD) music signals.

1.1.1.2. Quantization and sample resolution

Sample resolution or bit depth determines how precisely the sample's amplitude is recorded or stored. An n-bit sample resolution allows 2^n different possible amplitude values. Intuitively, the larger a sample resolution is, the closer its digital signal value approaches that of the analog signal. A resolution of 16 bits per sample is the most commonly used sample resolution nowadays. However, 8-bit, 24-bit, and 32-bit resolutions are also available. Among all these bit resolutions, the 8-bit per sample case provides the lowest sound quality and is less popular than other bit resolutions.

1.1.1.3. Channels

Sound can be stored in more than one channel. Most of us have the experience that the same music played back from a home theater setting is more appealing than that played back from a CD player, let alone from a single speaker. Studies have shown that increasing the sampling frequency, the sample resolution, and the number of channels all improve the quality of the reproduced digital sound. However, increasing the number of channels gives more degree of quality improvement of reconstructed audio. More information about multichannel audio is given in Section 1.3.

1.1.2. Building blocks of digital audio codecs

Figure 1.2 shows a typical encoding and decoding process for a general digital audio codec. At the encoder side, a recording device first captures the analog signal from the sound source, then samples and digitizes the signal. The device that performs analog-to-digital conversion is called the analog-to-digital converter or A/D converter. Signals coming from the A/D converter $s(n)$ are referred to as uncompressed digital signals, which will be then processed by the encoder. The encoder is the compression engine, which performs digital signal processing and entropy coding. Depending on the desired compression result, the encoder can be designed with different complexity ranging from extremely simple to highly sophisticated. The output from the encoder is called bitstream, which contains compressed signals and cannot be understood by regular player without a matching decoding process.

The decoding part of a general digital audio codec is depicted in Figure 1.2(b). The decoder, which is the decompress engine, performs entropy decoding and digital signal processing. The output of the decompress engine $r(n)$ is uncompressed digital signals and can be played back by regular players. In order to listen to uncompressed signals $r(n)$, a digital-to-analog conversion or D/A conversion need to be involved to reconstruct the digital signal to physical sound.

1.1.3. Lossy compression and lossless compression

Any compression technique belongs to either lossy compression or lossless compression. The goal of lossless compression is to encode the data in a way such that the matching decoder is able to reconstruct an exact copy of the original signals that are input to the encoder. In other words, no information loss is acceptable when designing a lossless encoder and its decoding process should be an exact inverse of the encoding process. On the contrary, the lossy compression allows the encoder engine to discard unimportant information. Thus the reconstructed signal is only an approximation of the original signal. Take Figure 1.2 as an example, with lossless compression, we are able to generate the exact same digital signals from the decoder, that is, signals input to the encoder $s(n)$ is the same as the signal output from the decoder $r(n)$; whereas, with lossy compression, the decoded signal $r(n)$ and the original digital signal $s(n)$ are no longer identical.

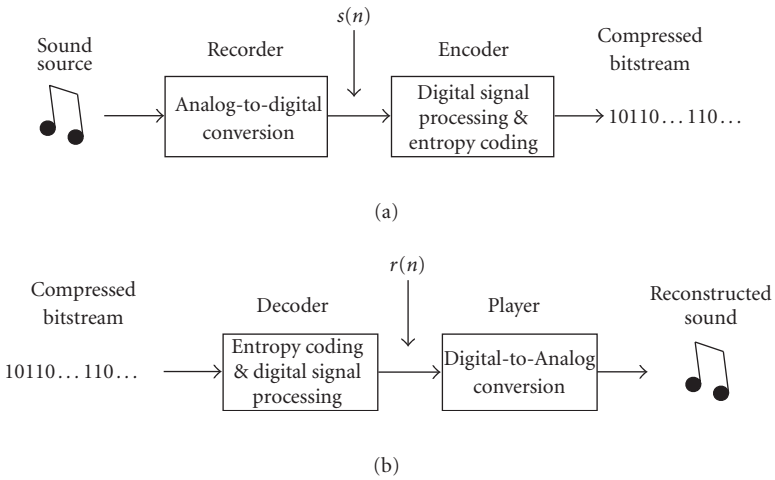


Figure 1.2. Block diagram of a typical digital audio codec: (a) encoding part and (b) decoding part.

In digital audio coding, a lossy codec is also called a perceptual codec because the design principle of a lossy audio codec is to remove the perceptually irrelevant or unimportant information as much as possible so that the decoder can generate a copy of sound file that is perceived as the identical or similar to the original audio clips by human ears. Most of today's popular audio coding techniques, such as mp3, AAC, and so forth, are perceptual audio codecs. Many lossless audio compression algorithms, such as Monkey Audio, DTS, Marian lossless, shorten, and so forth, are also available either on the market or can be freely distributed. The most cutting edge digital audio coding technology nowadays can achieve a compression ratio of 1:32 or higher, while the best lossless compression algorithm can hardly achieve a compression ratio of 1:3 for most sound source.

Looking back at the digital audio coding history, it is the lossy compression that dramatically changes the way that people think about the portable music. Fifty years ago, the only way for us to enjoy music at home is using gramophones, whose player and disk are both large and bulky. After tapes are introduced, we are able to carry the portable player and listen to songs while we are on the road. However, compared to the current start-of-art portable music players, for example ipod, these analog devices can never meet people's more and more demanding expectation. Thanks to the modern lossy audio compression techniques, without which no one is able to enjoy their music for several hours with a device fit in their palm.

1.2. Fundamentals of digital signal processing

1.2.1. Fourier transform

The audio signal is usually represented as a function of time when it is originally generated or recorded. However, during the audio coder design, it is convenient to

represent the same signal as a function of frequency. One important tool that allows us to transform functions of time into corresponding functions of frequency is called the *Fourier transform*. The analysis and synthesis formula of the Fourier transform for the continuous signals are given by

$$\begin{aligned} X(j\Omega) &= \int_{-\infty}^{\infty} x(t)e^{-j2\pi\Omega t}dt, \\ x(t) &= \int_{-\infty}^{\infty} X(j\Omega)e^{j2\pi\Omega t}d\Omega. \end{aligned} \tag{1.1}$$

For discrete signals, the Fourier transform pair is defined as below:

$$\begin{aligned} X(e^{j\omega}) &= \sum_{n=-\infty}^{\infty} x[n]e^{-j\omega n}, \\ x[n] &= \frac{1}{2\pi}\int_{-\pi}^{\pi} X(e^{j\omega})e^{j\omega n}d\omega. \end{aligned} \tag{1.2}$$

The Fourier transform pair provides complementary information about an audio signal and provides more flexibity in audio signal processing. For more thorough coverage on properties of Fourier transform and general digital signal processing techniques, we refer to [19, 95].

1.2.2. Sampling operation

The discrete-time representation of a continuous-time signal can be obtained by performing periodic sampling. That is, we set the value of a sampled signal $x[n]$ to the value of the continuous-time signal $x_c(t)$ at time $t = nT$:

$$x[n] = x_c(nT), \quad -\infty < n < \infty, \tag{1.3}$$

where T is called the *sampling period*, and its reciprocal $f_s = 1/T$ the *sampling frequency*, which is in the unit of samples per second. The system that implement the above mathematical operation is called an *ideal* continuous-to-discrete-time (C/D) converter. It is ideal in the sense that a practical C/D converter cannot sample a signal at time instance $t = nT$. However, this mathematical simplification allows us to focus on the main characteristics of a pure sampling process.

The ideal C/D converter can be decomposed into two stages as depicted in Figure 1.3(a), where the first stage is an impulse train modulator and the second stage is the conversion of the impulse train to a sequence. The effects of these two operations are shown in Figures 1.3(b) and 1.3(c). Figure 1.3(b) illustrates a continuous signal $x_c(t)$ being modulated by two impulse trains with different sampling periods. The corresponding output sequences of Figure 1.3(b) are shown in Figure 1.3(c).

To derive the frequency-domain relation between the input and output signals of an ideal C/D converter, let us first examine the relationship between $x_c(t)$

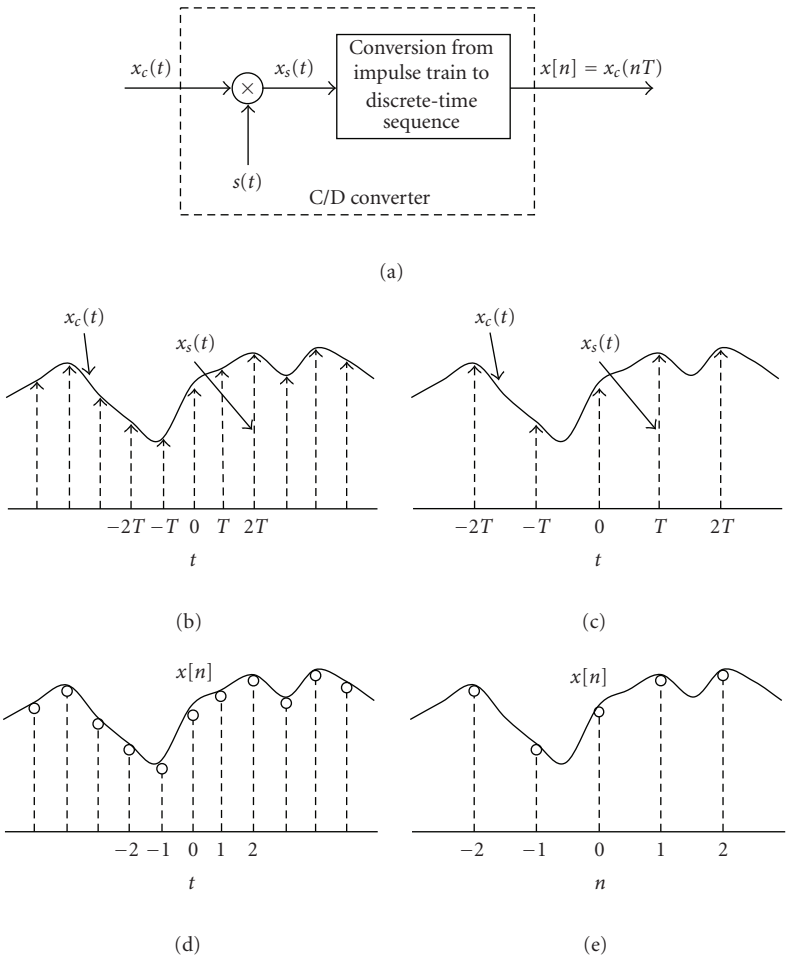


FIGURE 1.3. Sampling with a periodic impulse train followed by conversion to a discrete-time sequence: (a) the block diagram of the overall system, (b) $x_s(t)$ with two sampling rates T_1 and $2T_1$, where the envelope of the continuous signal $x_c(t)$ is represented by the dashed line, and (c) the output sequence with two different sampling rates.

and $x_s(t)$, where $x_s(t)$ is the impulse train modulated signal of $x_c(t)$. Since the modulation signal $s(t)$ is a periodic impulse train, that is

$$s(t) = \sum_{n=-\infty}^{\infty} \delta(t - nT), \tag{1.4}$$

where $\delta(t)$ is the unit impulse function or the Dirac delta function, we have

$$x_s(t) = x_c(t)s(t) = x_c(t) \sum_{n=-\infty}^{\infty} \delta(t - nT). \tag{1.5}$$

With the "shifting property" of the impulse function, we can express $x_s(t)$ as

$$x_s(t) = \sum_{n=-\infty}^{\infty} x_c(nT)\delta(t - nT). \tag{1.6}$$

Next, let us determine the Fourier transform of $x_s(t)$. From (1.5), we know that $x_s(t)$ is the product of $x_c(t)$ and $s(t)$. Therefore, the Fourier transform of $x_s(t)$ should be the convolution of the Fourier transforms of $x_c(t)$ and $s(t)$, that is,

$$X_s(j\Omega) = \frac{1}{2\pi} X_c(j\Omega) * S(j\Omega), \tag{1.7}$$

where $*$ denotes the convolution operation. It is known that the Fourier transform of a periodic impulse train is a periodic impulse train,

$$S(j\Omega) = \frac{2\pi}{T} \sum_{k=-\infty}^{\infty} \delta(\Omega - k\Omega_s), \tag{1.8}$$

where $\Omega_s = 2\pi/T$ is the sampling frequency in radians/s. Thus, we have

$$X_s(j\Omega) = \frac{1}{T} \sum_{k=-\infty}^{\infty} X_c(j\Omega - kj\Omega_s). \tag{1.9}$$

The above equation describes the relationship between the Fourier transforms of the original input signal $x_c(t)$ and the impulse train modulated signal $x_s(t)$ in the frequency domain.

1.2.3. Sampling theorem and aliasing

As indicated in (1.9), the Fourier transform of $x_s(t)$ is actually proportional to the summation of periodically repeated copies of Fourier transform of $x_c(t)$. The copies of $X_c(j\Omega)$ are first shifted by integer multiples of the sampling frequency before they are added together to produce the Fourier transform of $x_s(t)$. To shed light on the meaning of this important equation, let us consider an example where the input signal has a finite bandwidth. In Figure 1.4, we illustrate the frequency-domain effect after the time-domain sampling for a bandlimited input signal. The spectrum of the input signal is depicted in Figure 1.4(a), which has the highest nonzero frequency component Ω_N. The periodic impulse train $S(j\Omega)$ and the resulting modulated signal $X_s(j\Omega)$ are shown in Figures 1.4(b) and 1.4(c), respectively.

The sampling frequency is larger than twice of the signal's bandwidth in Figure 1.4(c). In Figure 1.4(d), we show the corresponding output signal $X_s(j\Omega)$ when the sampling frequency is less than twice of the signal's bandwidth. By comparing Figures 1.4(c) and 1.4(d), we see clearly that when

$$\Omega_s - \Omega_N > \Omega_N \qquad \text{or} \qquad \Omega_s > 2\Omega_N, \tag{1.10}$$

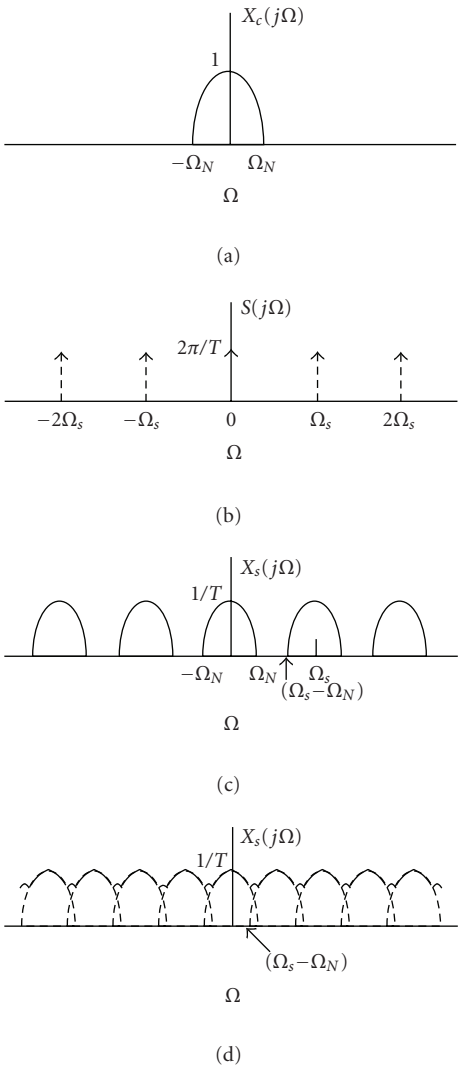


FIGURE 1.4. The effect in the frequency domain after sampling in the time domain: (a) the spectrum of a bandlimited input signal, (b) the spectrum of the sampling function, (c) the spectrum of the sampled signal with $\Omega_s > 2\Omega_N$, and (d) the spectrum of the sampled signal with $\Omega_s < 2\Omega_N$.

the shifted versions of $X_c(j\Omega)$ do not overlap with each other. Thus, by adding them to get the Fourier transform of $x_s(t)$, signals within the range of $[-\Omega_N, \Omega_N]$ is simply a $1/T$ scaled version of the original Fourier transformed signal $X_c(j\Omega)$. Consequently, the original continuous signal $x_c(t)$ can be recovered from $x_s(t)$ with an ideal lowpass filter.

The system to reconstruct the original signal from $x_s(t)$ is shown in Figure 1.5(a), where $H_r(j\Omega)$ is an ideal lowpass filter with gain T and cutoff frequency Ω_c

such that

$$\Omega_N < \Omega_c < (\Omega_s - \Omega_N). \tag{1.11}$$

Figures 1.5(b) and 1.5(c) show the frequency-domain signal of $x_c(t)$ and $x_s(t)$ when $\Omega_s > 2\Omega_N$. Since

$$X_r(j\Omega) = H_r(j\Omega)X_s(j\Omega), \tag{1.12}$$

it follows that

$$X_r(j\Omega) = X_c(j\Omega), \tag{1.13}$$

as depicted in Figure 1.5(e).

However, if inequality (1.10) does not hold, that is, $\Omega_s \leq 2\Omega_N$, shifted copies of $X_c(j\Omega)$ overlap with each other. By adding them together, there exists some interference between adjacent copies so that the spectrum of the original signal is modified. Consequently, the resulting signal $X_s(j\Omega)$ can no longer be used to recover the original signal $x_c(t)$ as illustrated in Figure 1.4(d). In this case, the reconstructed output $x_r(t)$ after passing $x_s(t)$ into the lowpass filter is a distorted version of the original input signal. This distorted version is called the *aliasing* of the original signal.

An example of aliasing is illustrated in Figure 1.6. The original signal

$$x_c(t) = \cos \Omega_0 t, \tag{1.14}$$

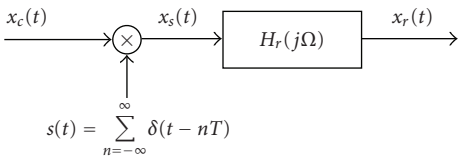
as shown in Figure 1.6(a), is sampled with sampling frequency Ω_s. Figures 1.6(b) and 1.6(c) depict the Fourier transform of $x_s(t)$ when $\Omega_0 < \Omega_s/2$ and $\Omega_0 > \Omega_s/2$. The corresponding Fourier transform of the lowpass filtered output for $\Omega_0 < \Omega_s/2 = \pi/T$ and $\Omega_0 > \pi/T$ are shown in parts (d) and (e), respectively, where the cutoff frequency of the lowpass filter Ω_c is equal to $\Omega_s/2$. In the nonaliasing case, as shown in parts (b) and (d), the reconstructed output $x_r(t)$ is

$$x_r(t) = \cos \Omega_0 t. \tag{1.15}$$

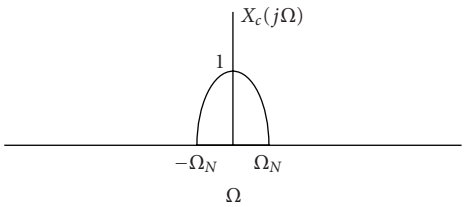
When aliasing happens as shown in parts (c) and (e), the reconstructed output is

$$x_r(t) = \cos (\Omega_s - \Omega_0) t. \tag{1.16}$$

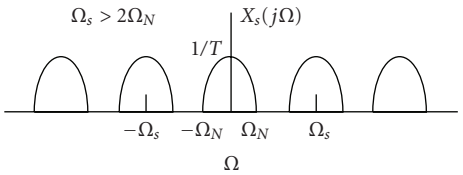
In other words, the higher-frequency signal $\cos \Omega_0 t$ is aliased to the lower-frequency signal $\cos (\Omega_s - \Omega_0)t$ after sampling and lowpass filtering.



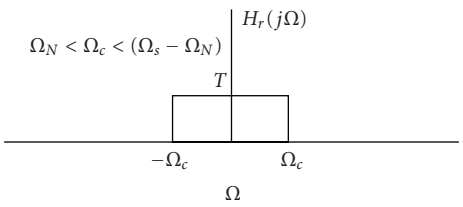
(a)



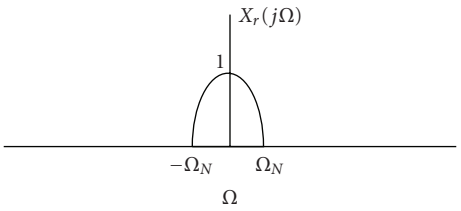
(b)



(c)



(d)



(e)

FIGURE 1.5. Reconstruction of a continuous-time signal from its samples using an ideal lowpass filter.

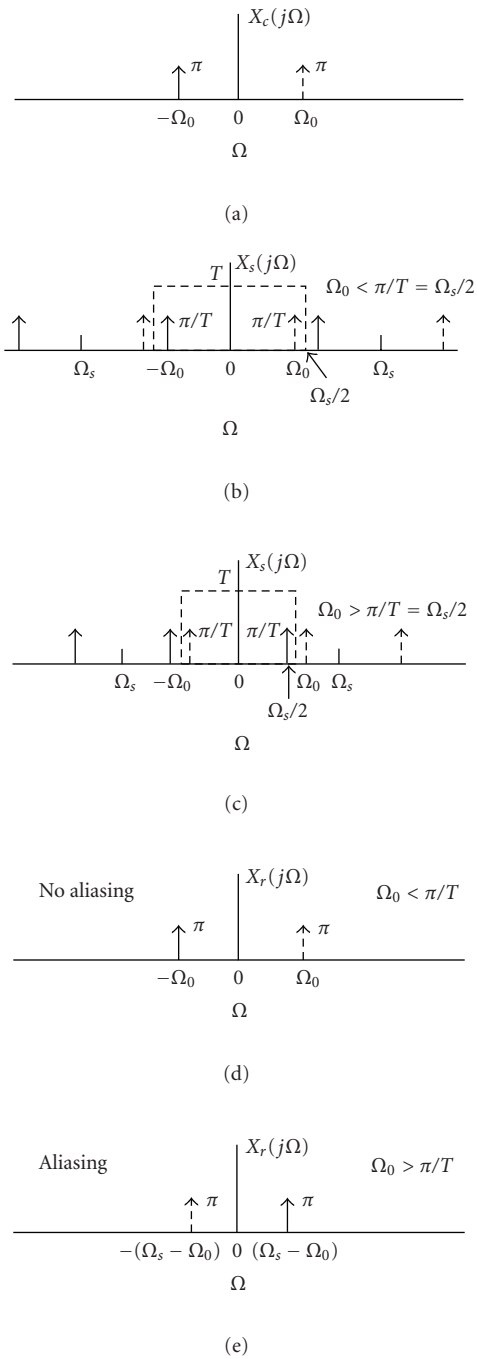


FIGURE 1.6. The effect of aliasing when sampling a cosine signal.

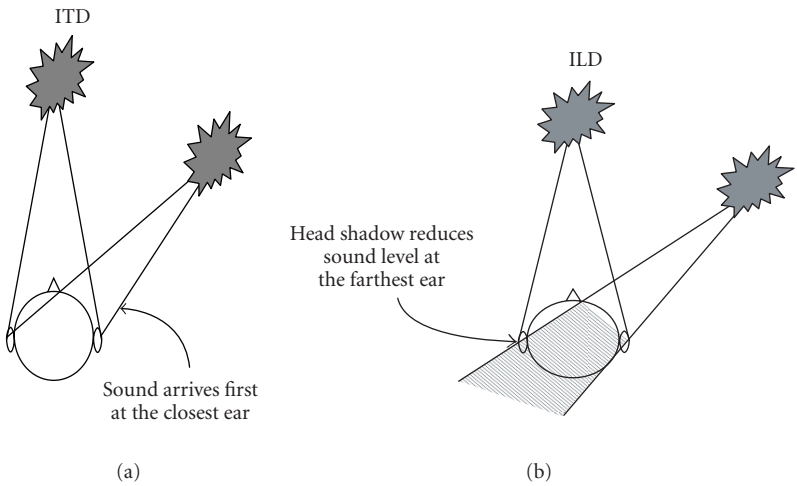


FIGURE 1.7. For low frequencies (typically below 1 kHz), the primary localization mechanism in the azimuth plane is ITD. The listener perceives small changes in the time of arrival of sound at the two ears and maps those changes into an angle of incidence. For higher frequencies, as the wavelength of sound becomes smaller than the head, localization is achieved through detection of level differences at the two ears caused by head shadowing.

The above discussion leads to the famous Nyquist sampling theorem as stated below.

NYQUIST SAMPLING THEOREM. *Let $x_c(t)$ be a bandlimited signal with*

$$X_c(j\Omega) = 0, \quad \text{for } |\Omega| > \Omega_N. \tag{1.17}$$

Then, $x_c(t)$ can be uniquely determined by its samples $x[n] = x_c(nT), n = 0, \pm 1, \pm 2, \ldots,$ if

$$\Omega_s = \frac{2\pi}{T} > 2\Omega_N. \tag{1.18}$$

The frequency Ω_N is called the Nyquist frequency, and the sampling frequency must exceed $2\Omega_N$ that is referred to as the Nyquist rate.

1.3. Multichannel audio

Audio technology has already reached the limits of human hearing for dynamic range with current digital audio systems operating at word lengths of up to 24 bits. The requirements for frequency response have been surpassed by moving to sampling rates past 48 kHz to 96 kHz and beyond. Future growth will inevitably come in the number of audio channels as human perception can hear many more directions than what current 5.1 channel surround sound systems provide. In this

section, we provide a review of the development of audio rendering from mono, to stereo, to 5.1 channel surround sound, and to 10.2 channel immersive audio. A great deal of the historical information is drawn from personal communications with Professor Tomlinson Holman and his presentations on "The history and future of surround sound" [54].

1.3.1. Perceptual cues

The reproduction of sound that is convincing enough to be indistinguishable from "the real thing" requires the audio rendering system to be capable of synthesizing as many of the necessary perceptual cues as possible. These cues include information about the location of each sound source in the recording space, as well as the interaction of each sound source with the acoustical elements of the environment.

The human sound localization process relies on analysis of signals presented at the two ears. Differences in intensity, time of arrival, and direction-dependent spectral changes caused by the outer ear and upper torso are the main elements that provide localization cues. The study of these cues started as early as 1882 [125] and identified two basic mechanisms that are responsible for source localization: (1) interaural time differences (ITDs) and (2) interaural level differences (ILDs). A later theory by Lord Rayleigh [122] proposed that ITD and ILD cues each operate in different wavelength regimes. For high frequencies (short wavelengths), the listener's head casts an acoustical shadow giving rise to a reduced sound level at the ear farthest from the sound source ILD (Figure 1.7(a)). For low frequencies (long wavelengths), the head is very small compared to the wavelength and localization is based on differences in the time of arrival of sound at the two ears ITD (Figure 1.7(b)). The two mechanisms of interaural time and level differences formed the basis of what became known as the duplex theory of sound localization.

These two basic principles for sound localization formed the basis of stereophony. It started with the work of Blumlein [14] who was the first to recognize that it was possible to locate a sound within a range of azimuth angles by using an appropriate combination of time and level differences. His work focused on the development of corresponding microphone techniques that would allow the recording of the amplitude and phase differences necessary for stereo reproduction. Fletcher, Snow, and Steinberg at Bell Laboratories [33, 119, 121] took a different approach. They examined the question "how many loudspeaker channels are required to create an exact representation of a sound scene?" Their theoretical findings showed that if an infinite number of microphones are used to capture a sound scene, then one can achieve perfect reproduction using an infinite number of loudspeakers, similar to the Huygens principle of secondary wavelets in optics. While this made for an interesting theoretical result, the Bell Labs researchers realized that practical implementations would require a significantly smaller number of channels. They showed that a three-channel system consisting of left, center, and right channels in the azimuth plane could represent the lateralization and depth of

the desired sound field with acceptable accuracy. Thus, stereo was born as a three-channel system. The first such stereophonic system was demonstrated in 1934 with the Philadelphia Orchestra conducted by Leopold Stokowski performing from the Academy of Music in Philadelphia for an audience in Washington, DC Constitution Hall over telephone lines.

Although the reproduction of localization cues is a critical component of audio rendering, it is by no means the only cue. When listening to a real sound source in a physical space, the listener perceives not only the direction of the source, but also its interaction with the acoustics of the space. One does not need visual reference to be able to tell that a violin is being reproduced on the stage of a large concert hall or in a small studio. In addition to the direct sound from the source, sound arrives at the listener's ears from an infinite number of directions as it interacts with complex acoustical surfaces in the room. The role of these reflections in the perception of the sound source depends on their direction of arrival and the amount of time they are delayed relative to the direct sound from the sound source. For example, in a concert hall, reflections arriving from the side walls at angles in the range of 40° to 80° and with a delay of 30 to 50 milliseconds are responsible for cues on the width of the space and the perceived width of the source in that space. Other reflections in the same time frame, but from the surfaces above the stage, provide cues about the depth of the stage. As reflections continue to bounce from different surfaces, progressively they longer paths to the listener. Once the arrival time exceeds a certain limit, the reflections are no longer discrete events, but rather become a statistical continuum with no perceived directionality. This component of the sound field (called the reverberant field) provides cues about the reverberation characteristics of the space.

From the description above, it is evident that a realistic rendering of sound sources in a physical space must contain not only the required localization cues provided by time and level differences of the direct sound, but also temporally and spatially distributed cues that are created by the interactions with the space. The limited number of channels in stereo and even 5.1 channel surround systems are not sufficient to reproduce all of these complex cues. This can only be achieved by increasing the number of audio channels.

1.3.2. Surround sound

The evolution of surround sound followed different paths in the consumer electronics and film industries. Film sound was historically where most of the innovations organized. The Fantasound system created by the Walt Disney company in the late 1930s used three screen channels, two side channels, a ceiling, and a back channel (played over two loudspeakers). It was the first multichannel system ever created, but it was only used in one film, "Fantasia."

In the 1950s, almost 20 years after Fantasia, another multichannel sound format was developed for Cinerama and later for Cinemascope. Film sound was already being reproduced over three front loudspeakers, but these new formats included an additional monophonic channel that was split to two loudspeakers

behind the audience and was known as the effects channel. This channel increased the sense of envelopment for the audience, but it also suffered from a serious limitation. Listeners seated on the center line with respect to the rear loudspeakers perceived "inside-the-head" localization similar to the effect of stereo images reproduced over headphones. Listeners seated off-center localized the channel to the effects loudspeaker that was closest to them as dictated by the law of the first-arriving wavefront, thus destroying the sense of envelopment desired [53]. The solution to these problems was found shortly after that by moving two effects channels reproduced over an array of loudspeakers along the sides of the theater to create a more diffuse sound field.

In the 1960s, home Hi-Fi systems began to spread, but they took a step back by using initially one and eventually two channels. This was due to the limitation of the delivery medium, the LP record, that was only capable of encoding two channels on its groove walls. The first attempt at consumer surround sound came in the early 1970s with Quad [113, 112]. In order to deliver the four channels in quadraphonic recordings over a two-channel phonograph record, an encoding and decoding scheme was developed that used 4:2:4 matrix encoding/decoding and relied on phase manipulation of the original stereo signals [88]. However the system did not take into account psychoacoustic considerations and instead used a loudspeaker placement that was symmetrical around the listener (two channels in front and two channels in the back). Later research [23] showed that human listeners can localize sounds very precisely in the front hemisphere and less precisely to the sides and in the rear hemisphere. As a result, for a given number of loudspeakers, it is always better to place more in the front and fewer in the back of the listener. These limitations and the presence of several competing formats in the consumer marketplace contributed to the demise of quadraphonic systems.

In the mid 1970s, Dolby introduced Dolby Stereo. It was based on the same quad matrix method for encoding four channels into two channels, but it used three front and one mono surround channels (split over two loudspeakers). In 1992, further enhancements were introduced through a new format that eliminated matrix-based encoding and decoding and provided five discrete channels (left, center, right, and two independent left and right surround channels) in a configuration known as stereo surround. A sixth, low-frequency enhancement (LFE) channel was introduced to add more headroom and to prevent the main speakers from overloading at low frequencies. The LFE channel was limited in bandwidth from 20120 Hz and so the system was named "5.1" by Tomlinson Holman (the LFE channel bandwidth is actually 0.005 of the full audio bandwidth of 20 Hz to 20 kHz). In 1995, this system was selected by the ATSC as the standard for HDTV in the US and shortly after that it also became the standard for DVD video.

1.3.3. Surround sound standards

In 1994, the Radiocommunication Assembly of the International Telecommunication Union (ITU) released a standards document entitled ITU-R BS.775-1 "Multichannel stereophonic sound with and without accompanying picture." In the

opening paragraph, the document states, the following: "Considering that it is widely recognized that a two-channel sound system has serious limitations and improved presentation is necessary . . . ."

The ITU document (http://www.itu.int/ITU-R/publications/index.html) went on to recommend one universal multichannel stereophonic sound system that is hierarchical and compatible with all broadcasting and recording standards. These include

(1) mono;
(2) mono with mono surround with the surround channel played over two loudspeakers (preferably decorrelated);
(3) two-channel stereo;
(4) two-channel stereo with one surround channel (split over two loudspeakers);
(5) two-channel stereo with two surround channels;
(6) three-channel stereo;
(7) three-channel stereo with one surround channel (split over two loudspeakers);
(8) three-channel stereo with two surround channels.

The requirements for each of the above systems include (i) control of the directional stability of the frontal sound image over a listening area larger than what is possible with two-channel stereo; (ii) the ambient sensation should be significantly enhanced over that provided by two-channel stereo; (iii) downward compatibility with sound systems that have a lower number of channels; (iv) upward compatibility with sound systems that have more loudspeakers than available signal channels; and (v) the audio quality after decoding must be "subjectively indistinguishable" from the reference for most types of program material.

For the case of five-channel surround sound, the standard recommends three front loudspeakers and two rear loudspeakers (Figure 1.8). The left (L) and right (R) front loudspeakers are to be placed at $\pm 30^\circ$ relative to the central listening position. The center (C) loudspeaker is to be at 0° and the surround loudspeakers (LS, RS) at an angle between 100° to 120°. All loudspeakers should be equidistant from the center listening position or compensated with time delay if that is not possible. The number of audio channels used for playback in this format is five with an optional low-frequency extension (LFE) signal channel that may be reproduced by a subwoofer loudspeaker.

This is an important standard that is widely adopted by the broadcast and recording industries. However, although in the title the document states that the standard is intended for program material with or without picture, it contains a recommendation that would seem to contradict that claim following: "It is not required that the side/rear loudspeakers should be capable of the prescribed image locations outside the range of the front loudspeakers." While this may be "rule" for film sound, future surround sound formats may be used to place sound at precise angular locations well outside the range of the front loudspeakers. In the next section, we describe such a future format that uses several more channels than the five prescribed in the ITU document.

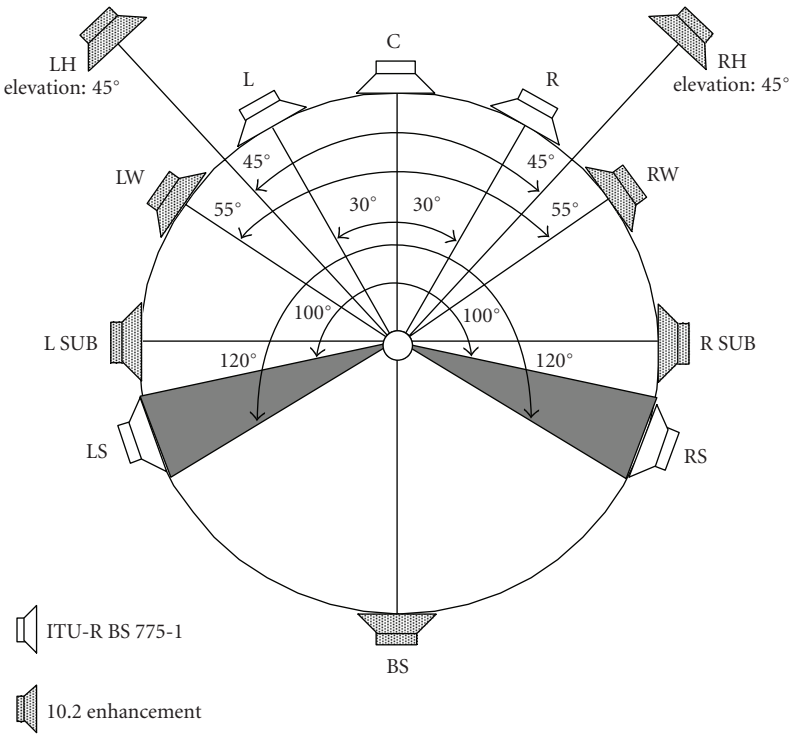


FIGURE 1.8. In the 10.2 channel surround sound system, additional wide, rear surround, and height loudspeakers are added to the standard five channels in the ITU recommendation. Also, two subwoofer channels are used on either side of the listening area. In addition to the bass from the main channels on the same side as each subwoofer, they also reproduce LFE information included in the program material.

1.3.4. A future surround sound system

Rebscher and Theile [105] examined the relationship between listening area width and the number of loudspeakers in the front plane. They showed that in order to keep the imaging distortion associated with shifts in the phantom images to less than 10%, the number of loudspeakers must increase as the width of the listening area increases. For example, at a listening distance of three picture heights, the listening area width is 0.25 m for two loudspeakers, 0.5 m for three loudspeakers, and 1.5 m for four loudspeakers. This work implies that the three front loudspeakers in today's 5.1 channel surround sound systems are barely adequate to produce accurate images. A listener that sits outside the 0.5 m listening area is subjected to high imaging distortion. Furthermore, the placement of the surround loudspeakers at $\pm 110^\circ$ was a compromise in order to make the system more universal for different types of program material. Theile showed in [105] that placing the surround loudspeakers directly to the side of the listener (at $\pm 90^\circ$) produced better envelopment, while placing them at $\pm 135^\circ$ allowed for better imaging in the back.

The only way around these problems is to increase the number of loudspeakers. A new multichannel audio system was developed in the Immersive Audio Laboratory of the Integrated Media Systems Center at the University of Southern California jointly with Professor Tomlinson Holman. The purpose of this system, called "10.2," was to go one step further in filling the perceptual gaps left by the 5.1 channel system.

In a monograph on subjective preferences and acoustical design, Ando [10] showed that controlling the direction, level, spectrum, and time of arrival of sidewall reflections in a concert hall is a key factor in a good hall design. He found that the preferred parameters over a wide range of listeners are $55^\circ \pm 20^\circ$ for the direction of the first sidewall reflection, while the level and spectrum are preferred to be the same as the direct sound. Based on these findings and on Theile's imaging distortion recommendations, we designed the 10.2 system to have a pair of "wide" channels (left wide (LW) and right wide (RW)) at $\pm 60^\circ$ relative to the central listening position (Figure 1.8).

In the 10.2 system, the surround loudspeaker compromise discussed above was addressed by adding a back surround loudspeaker at 180° directly behind the central listening position. This fills the envelopment gap between the two loudspeakers at $\pm 110^\circ$ and also allows for more precise placement of discrete sources in the rear hemisphere.

With eight loudspeaker channels in the horizontal plane, this system provides a significantly larger area over which imaging distortion is minimized. The remaining two channels are placed at a $\pm 45^\circ$ azimuth angle and are elevated to a 45° angle in the median plane. These height channels (left high (LH) and right high (RH)) are used to reproduce reflected sound from surfaces above the sound stage.

Content for this 10.2 channel immersive audio system is captured (or synthesized) using various microphone techniques depending on the venue and type of program material [92]. Classical music, for example, has been recorded in halls with very good acoustical characteristics using a combination of main microphone pairs, widely spaced outrigger microphones, choral or soloist microphones. In addition, microphones were placed in the back to capture the reflections in the hall and also in the proscenium above the orchestra to capture the information necessary for the height channels.

This new format and others that may come will inevitably utilize more channels than what current systems provide. Multichannel audio compression technology will have to be highly scalable in order to efficiently code the information in such program material. The novel compression methods described in the third part of this book are applicable not only to past mono, stereo, and surround formats, but also to future surround sound systems.

1.4. Outline of this book

This book consists of three parts and thirteen chapters. The first part of this book focuses on the basic knowledge of digital audio compression. Fundamental technologies on quantization, entropy coding, and psychoacoustics are covered from

Chapter 2 through Chapter 4. The second part of this book introduces the state-of-the-art MPEG4 audio coding standards. More space is devoted to the MPEG's multichannel audio coding standard, that is, advanced audio coding. The last part of this book discusses several new multichannel audio coding tools and is more suitable for readers in the advanced level.

2 Quantization

Quantization plays an important role in analog-to-digital conversion . It is usually achieved by approximating an input number with its nearest approximate value from a finite set of allowed values. The input value can be *analog*, that is, it may be of any value in a continuous range of possible amplitudes. The output value is always *digital*, that is, it is a uniquely specified value in the set $\{1, 2, 3, \ldots, N\}$, where N is the size of the set of output values. This chapter will present basic ideas of scalar quantization, vector quantization, and bit allocation theory. For more information about quantization, please refer to [38].

2.1. Scalar quantization

Scalar quantization maps an input value x into a finite set of output values y, that is, $Q : x \rightarrow y$. Figure 2.1 illustrates a scalar quantization by partitioning the real line into cells, where boundaries of these cells are called decision levels. If the input value x belongs to the ith cell, the output value of x is reconstruction value y_i. In other words, y_i is the only representative for all values that fall in the ith cell and y_i is called the ith codeword.

The sizes of cells can be the same or different as indicated in Figure 2.1. If all cells are of the same size, the resultant scheme is called the uniform quantization. Otherwise, it is called the nonuniform quantization. The uniform quantization is simple to implement. However, it may sacrifice the approximation performance if the probability distribution of the input value deviates from the uniform distribution significantly.

2.1.1. Uniform quantization

The uniform quantizer is the most commonly used scalar quantizer due to its simplicity. It is also known as a linear quantizer since its staircase input-output response lies along a straight line (with a unit slope). Two commonly used linear staircase quantizers are shown in Figure 2.2. They are called the midtread and the midrise staircase quantizers. In this figure, the horizontal axes represent the input value and the vertical axes represent the discrete output value. To design a uniform

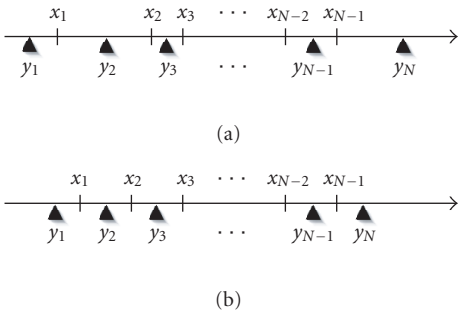


FIGURE 2.1. Illustration of a scalar quantizer by partitioning the real line.

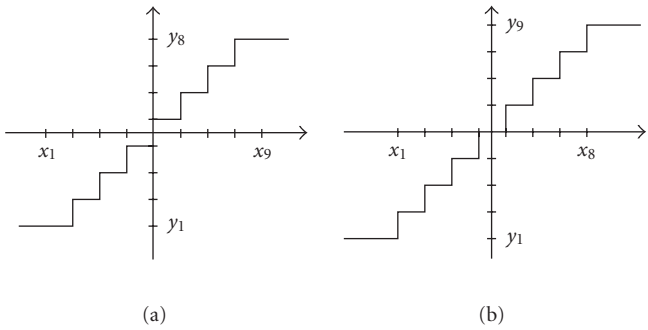


FIGURE 2.2. Linear staircase quantizers: (a) midrise staircase quantizer, (b) midtread staircase quantizer

quantizer, we have to find out the dynamic range of the input, determine the desired resolution of the output (in terms of the number of quantization levels), and select the proper type of quantizers (e.g., midtread or midrise).

From Figure 2.2, we have the following observation. The midrise quantizer does not have the zero output level and any input signal will be quantized into an even number of output steps. In contrast, the midtread quantizer is able to yield the zero output and has an odd number of output steps. If the output is represented by R bits, the midrise quantizer and the midtread quantizer will have 2^R and $2^R - 1$ different codes, respectively. Although the midtread quantizers output a smaller number of codewords, they generally generate a better quantization result for a given distribution of audio signal amplitudes by accommodating the zero output value.

It is difficult to handle an input of a very wide dynamic range. If the majority of input signal values are bounded by a range, say, from $-X_{\max}$ to $-X_{\max}$, we may want to clip values outside this range into this range. That is, any value smaller than $-X_{\max}$ is mapped to $-X_{\max}$ while any value larger than $X_{\max}$ is mapped to $X_{\max}$. The quantization error caused by clipping is often larger than the quantization error occurring in the regular cell, which may result in some noticeable artifact in

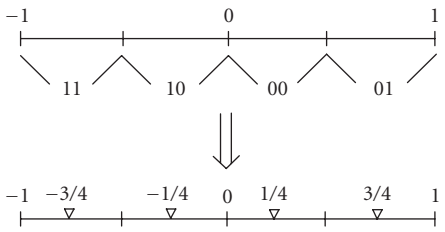


FIGURE 2.3. A two-bit uniform midrise quantizer.

high fidelity audio signal representation. Thus, there is a tradeoff between signal quality and the bit rate required for signal representation.

Once the bounded value $X_{\max}$ and the number of bits per codeword, R, are determined, the resolution Δ of the output can be computed accordingly: $\Delta = 2 \times X_{\max}/2^R$ and $\Delta = 2 \times X_{\max}/(2^R - 1)$ for the midrise quantizer and the midtread quantizer, respectively.

In the following two examples, we explain the quantization procedure of the midrise and the midtread quantizers when they are applied to input signals of amplitude between -1 and 1.

EXAMPLE (midrise quantizer). Let us consider a two-bit uniform midrise quantizer as illustrated in Figure 2.3. The amplitude of the input signal is shown at the left-hand side, which ranges from -1 to 1. For the two-bit quantizer, we can split the input range into 4 equal intervals with assigned codewords 01, 00, 10, and 11 as shown in Figure 2.3. Note that the first bit of the codeword is the same for input numbers with the same sign. In other words, the first bit is used to denote the sign while the remaining bits are used for the magnitude. At the decoder side, we should convert the 2-bit codes back to the signal sign and magnitude. This process is called dequantization. The dequantization process is often implemented by a simple table lookup. In this two-bit example, codes 01, 00, 10, and 11 are mapped to values of 0.75, 0.25, -0.25, and -0.75, respectively. Following the same principle, we can design midrise quantizers with more than two bits easily. A general procedure to map input signals onto R-bit uniform midrise quantizer codes and dequantize these codes back to signal amplitudes is shown in Algorithm 2.1.

EXAMPLE (midtread quantizer). A two-bit midtread uniform quantizer is illustrated in Figure 2.4, where the input range is divided into three parts. They are quantized in values of 01, 00 (or 10), and 11, respectively. At the decoder end, the dequantizer reconstructs codes 01, 00 (or 10), and 11 to 2/3, 0, and $-2/3$, respectively. The quantization and dequantization procedure for an R-bit uniform midtread quantizer is shown in Algorithm 2.2.

Quantization errors can be classified into two types: the round-off error and the clipping (or the overload) error. The round-off error occurs when the input signal with amplitudes within an interval is mapped into a single codeword. The wider this interval, the larger is the corresponding round-off error. We can estimate the round-off error if the probability distribution of the input signal is

Quantize:

$(s, m) = \text{quantize}(n, R)$

Input:

n: a number to be quantized

R: number of bits to be output

Output:

s: sign bit (1 bit)

m: magnitude bits ($R - 1$ bits)

$$s = \begin{cases} 0, & n \geq 0, \\ 1, & n < 0. \end{cases}$$

$$m = \begin{cases} 2^{R-1} - 1, & \text{when } |n| \geq 1, \\ \lfloor 2^{R-1} \times |n| \rfloor, & \text{elsewhere.} \end{cases}$$

Inverse-quantize:

$n = \text{inverse-quantize}(s, m)$

Input:

s: sign bit (1 bit)

m: magnitude bits ($R - 1$ bits)

Output:

n: output number

$$\text{sign} = \begin{cases} 1, & \text{if } s = 0, \\ -1, & \text{if } s = 1. \end{cases}$$

$|n| = (m + 0.5)/(2^{R-1})$

$n = \text{sign} \times |n|$

Algorithm 2.1. Quantization/inverse-quantization procedure for an R-bit uniform midrise quantizer.

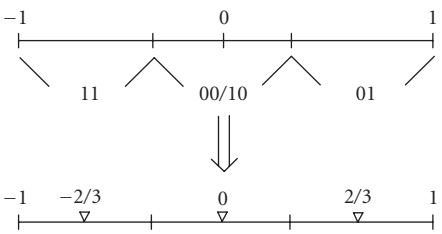


Figure 2.4. A two-bit uniform midtread quantizer.

available. The overload error is caused by input signals with an amplitude that is either too big or too small for the quantizer. The magnitude of this kind of input signals has to be clipped to the highest quantizer step. The overload error is associated with input signals with an amplitude greater than the quantizer's maximum amplitude $X_{\max}$. Thus, if we can find the probability distribution of the input signal amplitude, we can characterize the amount of overload error for a given quantizer.

Comparing the two-bit midrise quantizer and midtread quantizer examples given above, we see that the subinterval size of the midtread quantizer is larger than that of the midrise quantizer. Thus, if the input signal is uniformly distributed, the average quantization error of the midtread quantizer is larger than that of the midrise quantizer. However, when audio signals are being quantized, we often have quiet portions. Thus, the uniform distribution does not hold. It is observed in audio coding system design that a quantizer that can represent signals with zero amplitude perform better than the one that cannot. Due to this reason, the midtread quantizer is preferred.

Quantize:

$(s, m) = \text{quantize}\,(n, R)$

Input:

n: a number to be quantized

R: number of bits to be output

Output:

s: sign bit (1 bit)

m: magnitude bits ($R - 1$ bits)

$$s = \begin{cases} 0, & n \geq 0, \\ 1, & n < 0. \end{cases}$$

$$m = \begin{cases} 2^{R-1} - 1, & \text{when} |n| \geq 1, \\ \left\lfloor \dfrac{((2^R - 1) \times |n| + 1)}{2} \right\rfloor, & \text{elsewhere.} \end{cases}$$

Inverse-quantize:

$n = \text{inverse-quantize}\,(s, m)$

Input:

s: sign bit (1 bit)

m: magnitude bits ($R - 1$ bits)

Output:

n: output number

$$\text{sign} = \begin{cases} 1, & \text{if } s = 0, \\ -1, & \text{if } s = 1. \end{cases}$$

$$|n| = \frac{2 \times m}{(2^R - 1)}$$

$$n = \text{sign} \times |n|$$

Algorithm 2.2. Quantization/inverse-quantization procedure for an R-bit uniform midtread quantizer.

2.1.2. Nonuniform quantization

In contrast to uniform quantizers, we can design quantizers that divide the input range into unequal intervals, which are called nonuniform quantizers. If the input signal has a uniform distribution over the full dynamic range, the simple uniform quantization scheme offers the best performance. However, the probability of the input signal may not be uniform. Then, by dividing the full dynamic range into nonuniform spacing properly, we may either reduce the number of output bits with the same quantization error or reduce the amount of quantization errors with the same number of output bits. Two types of nonuniform quantizers that achieve robust performance for signals of a wide dynamic range are discussed below.

Example (compandor design). Figure 2.5 depicts a general approach to design a nonuniform quantizer. In the first stage, the input x is transformed by a nonlinear monotonically increasing function. The output of the first stage, that is, $y = G(x)$, is then uniformly quantized to $\hat{y}$. Finally, the inverse transform of the first stage is carried out in the last stage to produce the output $\hat{x} = G^{-1}(\hat{y})$. The nonlinear transform G is called a *compressor* and its inverse transform G^{-1} is called an *expandor*. These names come from the fact that G usually has a small slope for large amplitude inputs and therefore compresses (reduces the spread) large amplitude values while G^{-1} reverses this process and expands the large amplitudes. Combining the words "compressor" and "expandor," we get the name *compandor*.

One of the most important applications of this structure can be found in digital transmission of speech signals, where the compandor is implemented by simple diode resistor circuits. In digital speech telephony, two modified logarithmic compressor functions, the A-law and the μ-law, are widely adopted. The compressor

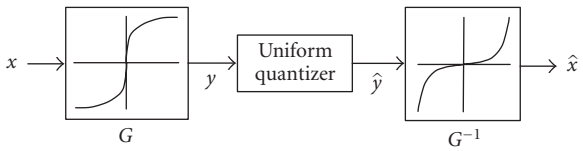


FIGURE 2.5. Design of nonuniform quantizers using the compandor approach.

function of the A-law is

$$G_A(x) = \begin{cases} \dfrac{A|x|}{1+\ln A}\,\mathrm{sgn}(x) & \text{for } 0 \le |x| \le \dfrac{V}{A} \\ \dfrac{V\left[1+\ln\left(A|x|/V\right)\right]}{1+\ln A}\,\mathrm{sgn}(x) & \text{for } \dfrac{V}{A} \le |x| \le V, \end{cases} \tag{2.1}$$

and the compressor function of the μ-law is

$$G_\mu(x) = V\frac{\ln\left(1+\mu|x|/V\right)}{\ln(1+\mu)}\,\mathrm{sgn}(x); \quad |x| \le V. \tag{2.2}$$

In the above compressor functions, A and μ are parameters that control the degree of compression by determining the ratio of the smallest to the largest step sizes. The widely adopted values of A and μ in practice are 87.6 and 255, respectively.

EXAMPLE (piecewise uniform quantization). Although the logarithmic compandors can be easily mathematically manipulated, they are difficult to implement due to their nonlinearities. Instead, uniform quantizers can be more accurately implemented using today's technology. Thus, we can design a piecewise linear compressor to approximate the smooth curves of the logarithmic or other compressors. A *piecewise uniform quantizer* is defined over several segments, each of which is a uniform quantizer. The quantization step size of different segments may be different. Similarly, the number of output levels can be different for different segments.

2.2. Vector quantization

Vector quantization (VQ) quantizes a vector of k dimension. Scalar quantization is a special and the simplest case of vector quantization, and only quantizes signals of one dimension. Unlike scalar quantization, VQ is usually only applied to signals that have already been digitized. In VQ, an input vector is compared to a set of predefined representative vectors. The index of the representative vector that minimizes a certain criterion will be used to represent the input vector. Since the amount of data needed for indices is much smaller than that of input vectors, a high compression rate can be achieved by VQ.

Mathematically, a vector quantizer Q that has size N and dimension k maps a vector in k-dimensional Euclidean space, R^k, into a finite set C containing N

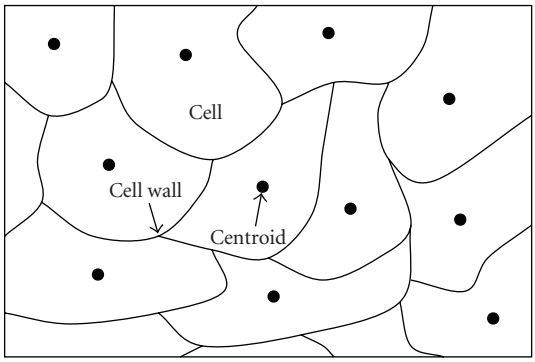


FIGURE 2.6. Example of 2D vector quantization.

representative points, which are called *code vectors* or *codewords*. In other words,

$$Q : R^k \longrightarrow C, \tag{2.3}$$

where $C = (y_1, y_2, \ldots, y_N)$ is the codebook with codewords $y_i \in R^k$ for each $i \in I \equiv \{1, 2, \ldots, N\}$. The *code rate* of a vector quantizer is defined as $r = (\log_2 N)/k$. The code rate, which is also known as resolution or rate, measures how many bits per vector component are needed to represent an input vector. If the codebook is well designed, the rate of a vector quantizer gives an indication on how accurately this vector quantizer can achieve.

A size N vector quantizer partitions the k-dimensional space R^k into N regions or cells, R_i for $i \in I$. The ith region R_i defined by

$$R_i = \{x \in R^k : Q(x) = y_i\}, \tag{2.4}$$

is called the *inverse image* or *preimage* of y_i based on mapping Q, and can also be denoted by $R_i = Q^{-1}(y_i)$. Since R_i is a partition of space R^k, we have

$$\bigcup_i R_i = R^k, \quad R_i \bigcap R_j = \varnothing \quad \text{for } i \neq j. \tag{2.5}$$

An unbounded cell is called an *overload cell*, while a bounded cell, that is, one having a finite k-dimensional volume, is called a *granular* cell. An example of how a 2D vector space is partitioned into cells is illustrated in Figure 2.6. The centroids of each region are depicted as black dots in the figure, and they may be used as codewords for input vectors that fall into the region.

A complete vector quantization/dequantization pair consists of two basic operations: the encoder and the decoder. The encoder $\mathcal{E}$ maps a vector in space R^k to an index in the set of I while the decoder $\mathcal{D}$ maps an index in the set I to a

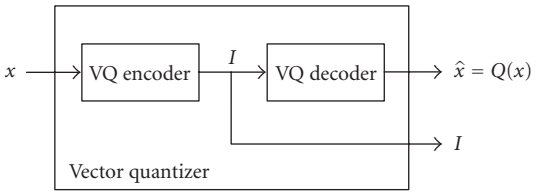


FIGURE 2.7. A vector quantizer as the cascade of an encoder and a decoder.

codeword in set C, that is,

$$\mathcal{E} : R^k \longrightarrow I, \qquad \mathcal{D} : I \longrightarrow R^k. \tag{2.6}$$

It is important to point out that it is the partition of the space that determines how the encoder should match an index to a given input vector. On the other hand, it is the given codebook that determines how the decoder should reconstruct the output vector from a given index. Therefore, the encoder's task is to identify which of the N regions of the k-dimensional space the input vector falls in and the decoder's task is simply doing a table lookup to find out which codeword corresponds to the index. The codebook of most practical vector quantizers contains sufficient information to characterize the partition so that the codebook is the only necessary data set in performing encoding and decoding since the encoder can implicitly determine the partition via the codebook.

The overall operation of VQ is a cascade of the encoding and the decoding operations as illustrated in Figure 2.7, that is,

$$Q(x) = \mathcal{D} \bullet \mathcal{E}(x) = \mathcal{D}(\mathcal{E}(x)). \tag{2.7}$$

In some cases, it is convenient to let the encoder generate both the index, i, and the corresponding codeword, $Q(x)$.

2.2.1. Nearest-neighbor quantizers

The *nearest-neighbor* vector quantizer, which is also called the *Voronoi* quantizer, is an important special class of vector quantizers that is of particular interest. Its special feature is that the partition is completely determined by the codebook and a distortion measure. The nearest-neighbor vector encoder is optimal in the sense of minimizing the average distortion for a given codebook. Such an encoder has an attractive advantage that no explicit storage of the geometrical description of cells is needed in the encoding process.

Let $d(x, y)$ be a distortion measure on the input and output vector space. The simplest and most commonly used measure is the *squared error distortion measure*,

$$d(x, y) = \|x - y\|^2 = \sum_{i=1}^{k} (x_i - y_i)^2, \tag{2.8}$$

which is the squared Euclidean distance between vectors x and y. The nearest-neighbor (NN) or Voronoi-vector quantizer is defined as one whose partition cells are given by

$$R_i = \{x : d(x, y_i) \leq d(x, y_j) \text{ all } j \in J\}. \tag{2.9}$$

In other words, with a nearest-neighbor vector encoder, each representative code vector y_i in cell R_i produces less distortion to any points x in cell R_i than any other code vector in other cells.

For a codebook containing N codewords, the encoding algorithm of the nearest-neighbor vector quantization is given below.

Nearest-neighbor encoding algorithm.

(1) Set $d = d_0$, $j = 1$, and $i = 1$.
(2) Compute $D_j = d(x, y_j)$.
(3) If $D_j < d$, set $D_j \to d$. Set $j \to i$.
(4) If $j < N$, set $j + 1 \to j$ and go to step 2.
(5) Stop. Result is index i.

The index i obtained from the nearest-neighbor vector encoder is associated with the desired codeword $C(x)$. The distortion generated by this encoder is $d(x, y_i)$. The above nearest-neighbor encoding algorithm performs an exhaustive search of the codebook to find the codeword that generates the smallest distortion. Such a search scheme is time consuming since we need to compare every codeword with the input vector. To accelerate the encoding process, some fast search techniques, which only search a subset of codewords, can be adopted. These fast search techniques generate results that have little performance loss as compared with that of the exhaustive search method.

2.2.2. Optimality of vector quantizers

Compared with scalar quantizers, vector quantizers may be able to improve the coding performance substantially. This is because the vector quantization has less geometric restrictions than the scale quantization. Taking the 2D space as an example, scale quantizers can quantize a 2D vector one dimension at a time so that the corresponding cells associated with scale quantizers are rectangular. Partitioning the 2D space into rectangles is generally not optimum. On the contrary, vector quantizers can partition the 2D space into cells of any shape, including the shape that optimizes performance.

Consider a two-dimensional random variable $X = (X_1, X_2)$, whose probability density function is a uniform distribution over the shaded regions within the unit square as shown in Figure 2.8(a), and the marginal distribution of X_1 and X_2 itself is also uniform over interval $(0, 1)$. However, the two random variables X_1 and X_2 are not statistically independent since the probability that $X_1 < 0.5$ given $X_2 = \alpha$ depends on whether α is less than or greater than 0.5. By using scalar quantization, an optimal partition method that uniformly quantize the space into

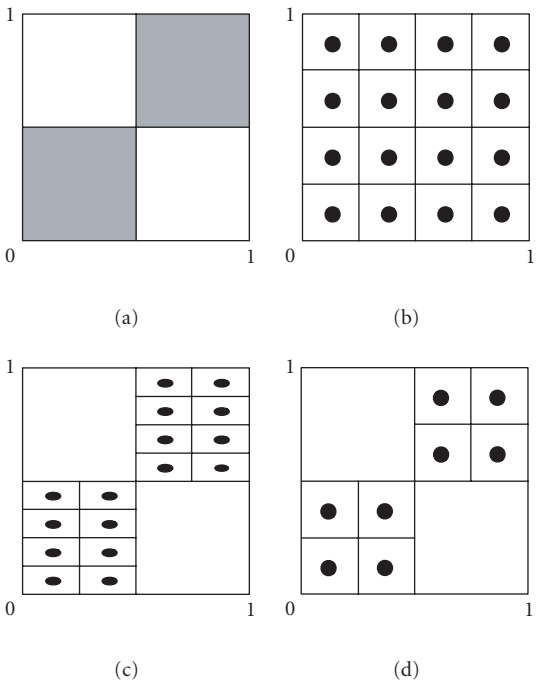


FIGURE 2.8. Comparison of several 2D quantization schemes: (a) 2D pdf, (b) scalar quantization, (c) VQ, (d) alternative VQ.

16 cells is shown in Figure 2.8(b) based on the fact that each component of the random variable X is uniformly distributed. On the other hand, using vector quantization, we can partition the space into 16 cells as shown in Figure 2.8(c). The worst-case quantization error between X and $Q(X)$ measured by Euclidean distance is $\sqrt{2}/8 \approx 0.18$ in case (b) and is $\sqrt{5}/16 \approx 0.13$ in case (c). In fact, the partition given by Figure 2.8(c) is superior to Figure 2.8(b) by any reasonable choice of performance measure. Therefore, in this example, vector quantizers can achieve smaller distortion than scale quantizers if the same numbers of bits are assigned for the index. Figure 2.8(d) shows an alternative partition by vector quantizers. It is clear that the average quantization error for case (d) is the same as for case (b). However, only 8 different codewords are assigned in case (d). This indicates that vector quantizers can achieve the same quantization error as scalar quantizers using a codebook of a smaller size.

Even for random variables that have components statistically independent of each other, vector quantization can still achieves better results than scalar quantization. Suppose that a 2D vector X is uniformly distributed on the square domain shown in Figure 2.9(a). A rectangular partition corresponding to the scalar quantization of each component is shown in Figure 2.9(b). Figure 2.9(c) shows a hexagonal partition by a vector quantizer. It is easy to verify that the worst-case error calculated by the Euclidean distance for the hexagonal partition is smaller

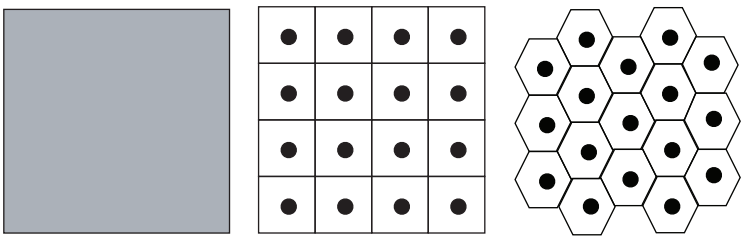


Figure 2.9. Comparison of two 2D partition schemes.

than for the square partition with the same number of partition cells. Since scale quantizers are restricted special cases of vector quantizers, vector quantization can always achieve better performance than or at least the same performance as scalar quantization.

2.2.3. Vector quantizer design

A VQ codebook is iteratively designed by choosing an initial codebook and then updating the partition based on the nearest-neighbor condition. The partition is continuously modified until an optimal codebook is obtained. This iteration process uses a set of training vectors which are representative of the actual vectors that the vector quantizer will work on.

The first important component in vector quantizer design is to select an initial codebook. If the initial codebook is properly chosen, the number of iterations for codebook refinement can be greatly reduced. A variety of techniques that generate initial codebook have been developed. One such technique is the *splitting algorithm*. It grows a large codebook from a small one and can produce increasingly larger codebooks of a fixed dimension. The splitting algorithm is given below. First, let us use only one vector to represent the whole training sequence. The centroid of the entire sequence would be the globally optimal solution. Thus, the optimal codebook of size 1 should have the only codeword equal to this centroid, denoted by y_0. The codebook of size 2 can be obtained from the codebook of size 1 by splitting y_0 into two codewords, y_0 and $y_0 + \epsilon$, where ϵ is a vector of a small Euclidean norm. One choice of ϵ is to make it proportional to the eigenvector corresponding to the largest eigenvalue of the covariance matrix of training vectors. Alternatively, we can choose ϵ to be proportional to the vector whose ith component is the standard deviation of the ith component of the training vectors. This codebook of size 2 can be refined iteratively to lead to two codewords that represent the entire training set with a good resolution. Suppose we have a good resolution codebook of size M. By splitting each of the M codewords into two codewords, we have $2M$ vectors. Starting from these $2M$ initial vectors, we can do some codebook refinement to produce a good resolution codebook of size $2M$.

The second important component in VQ codebook design is the codebook iterative refinement process. This is usually achieved by the generalized Lloyd algorithm, which is also known as the *k-means algorithm* in the field of pattern recognition or the LBG algorithm in the data compression literature[1]. Procedures of the generalized Lloyd algorithm and Lloyd iteration for codebook improvement are summarized below.

The generalized Lloyd algorithm.

(1) Start with an initial codebook C_1. Set $m = 1$ and set a threshold to be a small value.
(2) Use the codebook C_m and perform the Lloyd iteration to generate the improved codebook C_{m+1}.
(3) Compute the average distortion between C_m and C_{m+1}. If the distortion is smaller than the threshold, stop. Otherwise set $m + 1 \rightarrow m$ and go to Step 2.

The Lloyd iteration for codebook improvement.

(a) Given a codebook $C_m = \{y_i : i = 1, \ldots, N\}$, find the optimal partition of quantization cells using the nearest-neighbor condition, that is, find the nearest-neighbor cells

$$R_i = \{x : d(x, y_i) < d(x, y_j);\ \text{all } j \neq i\}. \tag{2.10}$$

If x yields a tie for distortion, for example, if $d(x, y_i) = d(x, y_j)$ for one or more $j \neq i$, then assign x to set R_j for which j is the smallest.

(b) Compute the centroid for each cell. The optimal codewords for these cells form the new codebook C_{m+1}, that is,

$$C_{m+1} = \{\text{cent}(R_i);\ i = 1, \ldots, N\}. \tag{2.11}$$

Various stopping criteria can be adopted to terminate the generalized Lloyd algorithm. One of the most common and effective criteria is to check if $(D_m - D_{m+1})/D_m$ is below a suitable threshold, where D_m represents the distortion between codebook C_m and C_{m-1}.

2.3. Bit allocation

An audio coding system usually consists of multiple quantizers, each of which is used to encode the audio signal within a short time interval. Different audio segments tend to have different characteristics and may require different accuracy of reproduction to achieve the best overall quality of the reconstructed audio with a given bit budget. The problem of distributing a proper number of bits into each quantizer is called the *bit allocation* problem.

[1]The name of LBG algorithm is more suitable to describe the splitting algorithm variation of the Lloyd algorithm.

2.3.1. Problem of bit allocation

Without loss of generality, let us consider a set of k random variables, $X_1, X_2, \ldots, X_k$, each of which has a zero mean and variance σ_i^2. We seek for a set of optimal quantizers Q_i $(i = 1, 2, \ldots, k)$, each of which uses b_i bits to quantize X_i. Suppose that $W_i(b_i)$ represent the error produced by quantizing X_i with b_i bits of resolution. The overall distortion of the whole system, D, can be defined as a function of the bit allocation vector $\mathbf{b} = (b_1, b_2, \ldots, b_k)$ as

$$D = D(\mathbf{b}) = \sum_{i=1}^{k} W_i(b_i). \tag{2.12}$$

Then, the bit allocation problem is to determine a set of optimal values for $b_1, b_2, \ldots, b_k$ subject to the condition that the sum of b_i is no larger than a fixed quota, B, of available bits. It is stated below.

The bit allocation problem. Find b_i for $i = 1, 2, \ldots, k$ to minimize $D(\mathbf{b}) = \sum_{i=1}^{k} W_i(b_i)$ subject to the constrain that $\sum_{i=1}^{k} b_i \leq B$.

Two basic approaches are available to find an optimal or near-optimal bit allocation. The first one is to find an algorithm that minimizes equation (2.12) over all possible choices of $\mathbf{b}$. However, this method is seldom used since it is computationally exhausted and the optimal distortion functions $W_i(b_i)$ are usually unknown. The second approach to minimize equation (2.12) is realized by invoking the high resolution quantization approximations[2]. Then, we can minimize $D(\mathbf{b})$ and generate the desired result, $b_1, b_2, \ldots, b_k$, using the Lagrangian technique.

2.3.2. Optimal bit allocation results

The goal of optimal bit allocation is to distribute available bits into quantizers such that the overall distortion is minimized. One of the pioneering solutions to the optimal bit allocation problem was proposed by Huang and Schultheiss [55]. In their algorithm, the correlated variables are first decorrelated using the Karhunen-Loève transform. The transformed random variables are then normalized by the corresponding variance so that the Max-Lloyd quantizer [84, 90], which is designed for unit variance Gaussian random variables, can be used for quantization. We discuss the classical optimal bit allocation solution to identically distributed normalized random variables. More information about other bit allocation methods can be found in [38, 114].

Optimal bit allocation for identically distributed normalized random variables. For a given set of identically distributed normalized random variables, the optimal bit

[2]High resolution refers to the case of quantization when the average distortion is much less than the input variance.

assignment using the high rate approximations is given by

$$b_i = \bar{b} + \frac{1}{2}\log_2 \frac{\sigma_i^2}{\rho^2}, \tag{2.13}$$

where

$$\bar{b} = \frac{B}{k} \tag{2.14}$$

is the average number of bits per component, k is the number of components, and

$$\rho^2 = \left(\prod_{i=1}^{k} \sigma_i^2 \right)^{1/k} \tag{2.15}$$

is the geometric mean of the variances of the random variables.

The optimal bit numbers calculated by (2.13) may contain zeros and negative values. In practice, this bit allocation rule is modified by replacing negative values by zeros. If an integer valued allocation is required, then we can adjust each non-integer allocation b_i to its nearest integer. However, these modifications can make the total number of assigned bit-exceeds the allocation quota, B. Then, further adjustment is needed to achieve an allocation result satisfying all requirements. The final integer-valued bit allocation vector $\mathbf{b}$ can be determined heuristically.

3 Entropy coding

Entropy coding has been studied extensively in the literature. Detailed discussions may be found in many books and papers. In this chapter, we will introduce three commonly used lossless coding algorithms, that is, Huffman coding, arithmetic coding, and QM coding. For further reading on entropy coding and data compression, we recommend [109, 110]. Before going to the detail of these algorithms, we go over some basic information theory concepts.

3.1. Introduction to information theory

The importance of information theory is that it provides a way to measure information quantitatively. With information theory, we can answer the question "how much information is included in this piece of data?" with a precise number! Quantifying information can be done based on the observation that a message's information content is equivalent to how surprising the message is. If I tell you something that you already know (e.g., "I had dinner yesterday"), I probably have not given you any information because usually everybody has dinner everyday. If I tell you something new (e.g., "I had dinner in a restaurant yesterday"), I have given you some information (the place where I had my dinner yesterday). If I tell you something that really surprises you (e.g., "I did not have dinner yesterday"), I have given you more information. Shannon [115] defined a quantity called *self-information*, which is defined as follows.

Suppose event E is a set of outcomes of a random experiment. Let $P(E)$ denote the probability that the event E will occur, then the self-information associated with E is given by

$$i(E) = \log_x \frac{1}{P(E)} = -\log_x P(E). \tag{3.1}$$

Since $\log(1) = 0$ and $-\log x$ increases as x decreases from one to zero, if the probability of an event is small, then its associated self-information is big; if the probability of an event is big, then the associated self-information is small.

Another important property of this mathematical definition of information is that the information obtained from the occurrence of two independent events

is equivalent to the sum of the information obtained from these two individual events. In other words, suppose E_1 and E_2 are two independent events. The self-information associated with the occurrence of both event E_1 and event E_2 is

$$i(E_1E_2) = \log_x \frac{1}{P(E_1E_2)}. \tag{3.2}$$

Since E_1 and E_2 are independent, that is,

$$P(E_1E_2) = P(E_1)P(E_2), \tag{3.3}$$

so

$$i(E_1E_2) = \log_x \frac{1}{P(E_1)P(E_2)} = \log_x \frac{1}{P(E_1)} + \log_x \frac{1}{P(E_2)} = i(E_1) + i(E_2). \tag{3.4}$$

The unit of information depends on the base of the log. The most commonly used unit is base 2 and the unit is called *bits*. Other units, such as *nats* and *hertleys*, use log-based e and 10, respectively.

Example. Let H and T be the outcomes of tossing a coin, and let $P(H)$, $P(T)$ be the probability of getting the head and tail. If the coin is fair, we have

$$i(H) = i(T) = 1 \text{ bit}, \tag{3.5}$$

because

$$P(H) = P(T) = \frac{1}{2}. \tag{3.6}$$

If the coin is not fair, then the information expected is different from the previous case. Suppose

$$P(H) = \frac{1}{4}, \qquad P(T) = \frac{3}{4}. \tag{3.7}$$

Then, we have

$$i(H) = 2 \text{ bits}, \qquad i(T) = 0.415 \text{ bits}. \tag{3.8}$$

From the mathematical point of view, the occurrence of a head conveys more information than the occurrence of a tail. As we will see later, this has certain consequences for how the information conveyed by these outcomes should be encoded.

Now let us look at another important definition in information theory, that is, *entropy*. Suppose there is a set of independent events E_i, which are sets of outcomes of some experiment S, such that

$$\bigcup E_i = S, \tag{3.9}$$

where S is the sample space, then the average self-information associated with the random experiment is given by

$$H = \sum P(E_i)\, i(E_i) = -\sum P(E_i) \log_x P(E_i). \tag{3.10}$$

The quantity given by H in the above equation is called the *entropy*. It was shown by Shannon that if the experiment is a source that outputs symbols E from a set $\mathcal{E}$, then the entropy gives the average number of binary symbols needed to encode the output of the source. Shannon also claimed that the upper limit that a lossless compression algorithm scheme can achieve is to encode the output of a source with an average number of bits (nats or hartleys) that are equal to the source entropy.

The symbol set $\mathcal{E}$ is often called the source *alphabet* while symbols are called *letters*. For a source $\mathcal{S}$ with alphabet $\mathcal{E} = \{1, 2, \ldots, k\}$ that generates sequence $\{X_1, X_2, \ldots\}$, the entropy of $\mathcal{S}$ can be computed by

$$H(\mathcal{S}) = \lim_{n\to\infty} \frac{1}{n} H_n, \tag{3.11}$$

where

$$\begin{aligned} H_n = -\sum_{i_1=1}^{i_1=k} \sum_{i_2=1}^{i_2=k} \cdots \sum_{i_n=1}^{i_n=k} & P(X_1 = i_1, X_2 = i_2, \ldots, X_n = i_n) \\ & \times \log P(X_1 = i_1, X_2 = i_2, \ldots, X_n = i_n), \end{aligned} \tag{3.12}$$

and where $\{X_1, X_2, \ldots\}$ is a sequence of length n from the source. If all elements in the sequence are independent and identically distributed, then we have

$$H_n = -n \sum_{i_1=1}^{i_1=k} P(X_1 = i_1) \log P(X_1 = i_1), \tag{3.13}$$

and the equation for the entropy becomes

$$H(\mathcal{S}) = -\sum P(X_1) \log P(X_1). \tag{3.14}$$

We call the quantity computed in (3.11) the real *entropy* of the source, while the quantity given by (3.14) as the *first-order entropy* of the source. Generally speaking, (3.11) and (3.14) are not identical for most sources.

Since the entropy for a physical source is normally unknown, it has to be estimated in most cases. The estimation of the entropy will be different if we adopt different assumptions about the structure of the source sequence. These assumptions are called the *model*. A *static* model is the one whose parameters do not change, while an *adaptive* model is the one whose parameters change or adapt to the changing characteristics of the input data.

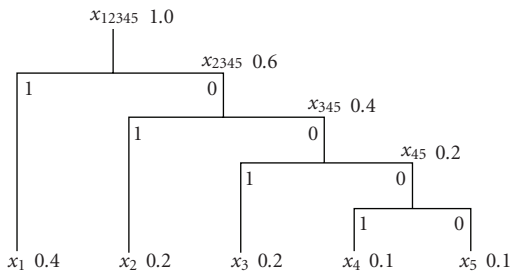


FIGURE 3.1. An example on how to build a Huffman tree and assign a Huffman code to each symbol.

3.2. Huffman coding

3.2.1. Huffman coding algorithm

Huffman coding is a commonly used data compression method, which was developed by Huffman [56]. Huffman codes belong to the set of prefix codes, which means that that no word in the entire codeword set is a prefix of another word from the same set.

The first step of building a Huffman code is to sort the probabilities of all alphabet symbols in a descending order and then construct a list of them. With this list, we can build a tree that has a symbol at every leaf, from the bottom up. This kind of tree is called *Huffman tree*. Now, we look at an example on how to build a Huffman tree.

EXAMPLE. Suppose there are five symbols x_1, x_2, x_3, x_4, and x_5. Their probabilities are shown in Figure 3.1 beside each symbol. We can build a Huffman tree by pairing them in the following order.

(1) Symbols x_4 and x_5 with the smallest probabilities are combined together and then replaced by a combined auxiliary symbol x_{45}, whose probability is equal to 0.2.

(2) Now, we have four symbols left, x_1, x_2, x_3, and x_{45}. Except for x_1 which has probability 0.4, all the other three symbols have probabilities equal to 0.2. We can just arbitrarily select two of them. Suppose x_3 and x_{45} are combined, then they can be replaced by another auxiliary symbol x_{345} with probability equal to 0.4.

(3) Now, we have three symbols left, x_1, x_2, and x_{345}. Their probabilities are 0.4, 0.2, and 0.4, respectively. Again, we have more than one choice to combine two of three symbols. We select x_2 and x_{345}, which are then combined and replaced by an auxiliary symbol x_{2345} with probability equal to 0.6.

(4) Finally, there are only two symbols left, x_1 and x_{2345}. We combine and replace them with x_{12345}, whose probability is equal to 1.

After the above four steps, we now have a complete Huffman tree as shown in Figure 3.1. To assign the codes, we arbitrarily assign a bit of 1 to the left branch, and a bit of 0 to the right branch, of every pair of branches and resulting codes 1, 01, 001, 0001, and 0000 to represent symbols x_1, x_2, x_3, x_4, and x_5, respectively.

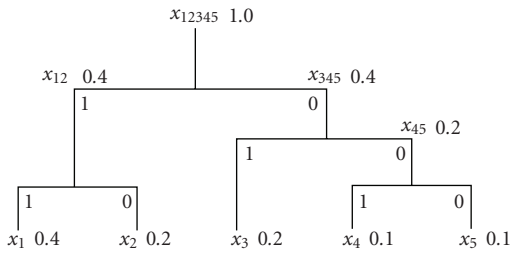


FIGURE 3.2. Another example of Huffman tree and Huffman codes.

The average size of this code is

$$\bar{l} = 0.4 \times 1 + 0.2 \times 2 + 0.2 \times 3 + 0.1 \times 4 + 0.1 \times 4 = 2.2 \text{ bits/symbol}. \quad (3.15)$$

With this example, we can summarize how to build a Huffman tree. At each stage, two symbols with the smallest probabilities should be selected, combined, and replaced by an auxiliary symbol, whose probability is equal to the sum of the two combined symbols. After the combination of two symbols, we have a symbol set with size equal to the size of the previous symbol set minus one. Treat the auxiliary symbol as a normal symbol and continue the selection, combination, and replacement as stated before until only one auxiliary symbol is left. The Huffman tree is now complete with the root representing the whole symbol set and probability equal to 1. We can then assign bits to each branch of the tree. Note that the assignment of bits to branches can be arbitrary and a different assignment gives a different Huffman code. However, the average size of the code will be the same for a given Huffman tree.

3.2.2. Variance of Huffman codes

In the previous example, there is some arbitrary selection when we build the Huffman tree. A different selection will end up with a different Huffman tree. Therefore, the Huffman tree is not unique. In the Huffman tree shown in Figure 3.1, we arbitrarily combined x_2 with x_{345}, since there were more than two symbols with the smallest probabilities. Figure 3.2 shows another selection and combination method using the same five-symbol set. With the Huffman tree and code assignment shown in Figure 3.2, we have a different Huffman code (11, 10, 01, 001, and 000). The average size of this code is

$$\bar{l} = 0.4 \times 2 + 0.2 \times 2 + 0.2 \times 2 + 0.1 \times 3 + 0.1 \times 3 = 2.2 \text{ bits/symbol}, \quad (3.16)$$

which is the same as the previous code.

It seems that the arbitrary decisions made in constructing the Huffman tree affect the individual codes but not the average size of the code. How about the

variance of the length of the codewords? The variance of code shown in Figure 3.1 is

$$v_1 = 0.4(1-2.2)^2 + 0.2(2-2.2)^2 + 0.2(3-2.2)^2 + 0.1(4-2.2)^2 + 0.1(4-2.2)^2 = 1.36, \tag{3.17}$$

while that of the code shown in Figure 3.2 is

$$v_2 = 0.4(2-2.2)^2 + 0.2(2-2.2)^2 + 0.2(2-2.2)^2 + 0.1(3-2.2)^2 + 0.1(3-2.2)^2 = 0.16. \tag{3.18}$$

v_2 is much smaller than v_1.

A code with smaller average length generates a smaller compressed bitstream. A code with smaller variance performs better when the encoder transmits the compressed stream as the code is being generated. Because, in such a case, although a variable length code is commonly used, the available transmission rate is generally fixed. Which means that the bits which can be transmitted in a given amount of time are constant. Since the output bit rate from an encoder using a variable length code varies all the time, a buffer needs to be used before bits of the compressed stream are entered into the real communication line. Thus, a code with smaller variance is preferable in this kind of situation, because it requires only a small buffer when doing the transmission. If the buffer cannot hold all output bits at a certain time, some bits will be lost and resulting in a corrupted bitstream at the receiver side.

Then, how can we construct a Huffman code that has smaller variance? We look at the Huffman tree in Figures 3.1 and 3.2 again. In the tree shown in Figure 3.1, symbol x_{345} is selected to be combined with x_2, whereas in the tree shown in Figure 3.2, x_1 is selected and combined with x_2. Therefore, we conclude that when there are more than one possible combination of two smallest-probability nodes, select the one that is closer to the beginning of the list and the one that has the lowest probability and combine them. By doing this, we combine symbols of low probability with ones of high probability resulting in a code with smaller total variance.

3.2.3. Huffman decoding

The Huffman codes have to be known before the decoder can start to do the decoding. The encoder can either directly write the Huffman code or the Huffman tree into the compressed stream, or just send the probabilities (or frequencies) of the symbol in the bitstream and let the decoder construct the Huffman code in the same way as the encoder does.

Once the Huffman codes are available, the decoding process is simple. Just start from the root of the Huffman tree and read bit by bit from the compressed stream. If the bit is one, follow the right branch, if it is zero, follow the left branch.

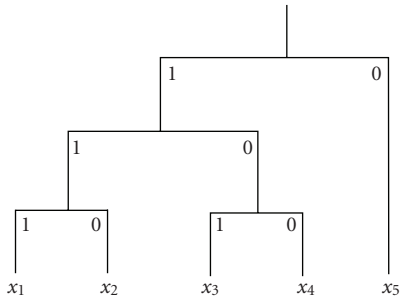


FIGURE 3.3. An example of Huffman decoding.

When a leaf of the tree is reached, the original symbol can be recovered by looking up the Huffman table.[1] After decoding one symbol, we should start from the root again and read the next bit. Continue this process until all the bitstreams are decoded.

Figure 3.3 is a Huffman tree with five-symbol alphabet. Suppose we now have a bitstream of 1001100111. Let us use this Huffman tree to decode the given bitstream. We should first start from the root, the first bit 1 indicates that we should follow the left branch. The second bit is 0, we then follow the right branch. With the third bit 0, we reach a leaf of the tree. And by looking up the Huffman table, we know that the code 100 corresponds to symbol x_4. So the decoder recovers the first symbol x_4 and then it starts from the root again. By following the same rule, we then decode symbols x_2, x_5, and x_1. After x_1 is reconstructed, we meet the end of the bitstream. Therefore, we know that the encoder has sent four symbols $x_4x_2x_5x_1$ to the decoder.

3.2.4. Adaptive Huffman coding

Both the encoder and the decoder need to know Huffman codes in order to proceed the Huffman coding. Thus, the first step of constructing Huffman codes is to sort the probabilities of symbols in the alphabet. When the probabilities of symbols are not available beforehand at the encoder side, we need to collect the statistics of the entire alphabet first. This means that we need to go over the string two times in performing the coding task. The first pass is to collect the symbol statistics by counting the frequencies of symbols while the second pass is to encode symbols with their associated codewords.

To simplify this procedure, Faller [30] and Gallager [37] developed an adaptive Huffman coding independently. With adaptive Huffman coding, we can build a Huffman tree based on the statistics of symbols that are already encountered, and then update the Huffman tree after each symbol entering the encoder or being

[1] A Huffman table is a table that shows the correspondence of the original symbol and its Huffman code.

decoded. Faller's and Gallager's algorithm was later improved by Knuth [77] and Vitter [129].

3.3. Arithmetic coding

3.3.1. Arithmetic coding algorithm

In [37], it has been shown that the code generated by Huffman coding has a rate smaller than $p_{\max} + 0.086$ of the entropy, where $p_{\max}$ is the maximum probability of all symbols. When the alphabet size is large, the value of $p_{\max}$ is generally small, so the code rate achieved by Huffman coding only has a small deviation from the entropy. However, when the alphabet size is small, $p_{\max}$ can be quite large and the coding efficiency of Huffman algorithm can become rather poor when compared with the entropy.

One of the reasons that prevent Huffman coding from further improving its performance is that it only assigns an integer number of bits to each symbol in the alphabet. One method to reduce the coding rate of Huffman algorithm is to combine two or more symbols as one auxiliary symbol then assign Huffman codes to these auxiliary symbols. However, a good symbol combination method is hard to determine and its performance is still not satisfactory in some cases. The arithmetic coding, which is different from Huffman algorithm, assigns only one (usually very long) code to the entire input stream. Whenever an input symbol is input and processed, it outputs one or more bits to the bitstream. The final number output from the arithmetic encoder can be viewed as the probability of the input stream.

The first step of arithmetic algorithm is to calculate or estimate the occurrence frequencies of symbols if the probabilities of each symbol in the alphabet are not known in advance. To achieve the best frequency estimation, we can read the entire input stream once before the real compression takes place.

Suppose we have three symbols x_1, x_2, and x_3, and their probabilities are $P_1 = 0.3$, $P_2 = 0.5$, and $P_3 = 0.2$, respectively. As shown in Figure 3.4, we divide the interval $[0, 1)$ into three subintervals whose sizes are proportional to the three probabilities. In this figure, subintervals assigned to x_1, x_2, and x_3 are [0,0.3), [0.3,0.8), and [0.8,1.0), respectively. To encode a string "$x_2x_2x_1x_3$," we start from the initial interval [0.0, 1.0). The first symbol x_2 reduces the initial interval to the range $[0.3, 0.8)$. Then again we divide the new interval into three subintervals whose size is proportional to its probability. After the second symbol x_2 is processed, the range reduces to [0.45, 0.7).[2] Similarly, we calculate the new subinterval and input symbol x_1 which makes the new interval [0.45, 0.525). Apply the same method after x_3 is processed: the final interval produced is [0.51, 0.525), which means that the probability of getting the stream of "$x_2x_2x_1x_3$" is between 0.51 and 0.525.

[2]The subinterval $[0.45, 0.7)$ is obtained from the interval $[0.3, 0.8)$ by $0.3 + (0.8 - 0.3) \times 0.3 = 0.45$ and $0.3 + (0.8 - 0.3) \times 0.8 = 0.7$.

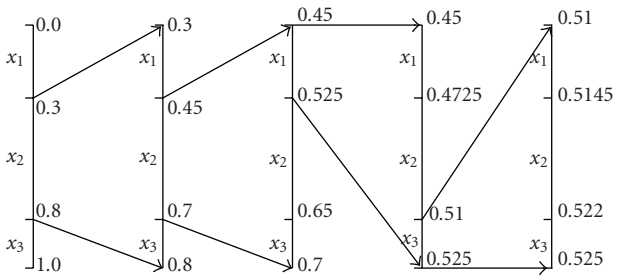


FIGURE 3.4. An example of arithmetic encoding.

Note that the final code output from the arithmetic encoder can be any number within the final range. Also, when the original interval is being divided into subintervals, the order of subintervals is immaterial. Different subinterval assignments will result in different final ranges. As long as the encoder and decoder have the same subinterval assignment, the decoder can always successfully decode the input stream.

With this example, we can easily understand the main steps of arithmetic coding.

Main steps of arithmetic coding.

(1) Define the initial range to be [0,1).

(2) For each symbol x in the input stream,

  (2.1) divide the interval into subintervals whose sizes are proportional to symbols' probabilities;

  (2.2) find the subinterval for symbol x and define it as the new interval.

(3) When all symbols in the input stream have been processed, output any number that is inside the final interval.

Below is a pseudocode to calculate the new interval after symbol x is processed:

$$\begin{aligned} \text{Range} &= \text{High} - \text{Low};\\ \text{High} &= \text{Low} + \text{Range} \times \text{HighLimit}(s);\\ \text{Low} &= \text{Low} + \text{Range} \times \text{LowLimit}(s); \end{aligned} \tag{3.19}$$

where LowLimit(x) and HighLimit(x) are the low and high limits of the original range of symbol x. For instance, the LowLimit(x_2) and HighLimit(x_2) in the previous example are 0.3 and 0.8. We start the encoding process by defining Low = 0.0 and High = 1.0. After processing each symbol, the interval becomes smaller. The output of the encoding is just a number representing the probability of the input stream. The average code size, which is calculated by dividing the size of the output (in bits) by the size of the input (in symbols), can be very small.

The principle of the arithmetic decoder is simple. Suppose the final code in the previous example is 0.52. The decoder first checks which original subinterval 0.52 lies in. Since it is within [0.3,0.8), the decoder knows that the first symbol is x_2. Then the decoder decodes the symbol x_2 and calculates the new number to

decode the second symbol by subtracting $\text{LowLimit}(x_2) = 0.3$ and then dividing by $\text{HighLimit}(x_2) - \text{LowLimit}(x_2) = 0.8 - 0.3 = 0.5$. The new number after the above operation is 0.44, which again falls into the range of [0.3, 0.8). So the decoder knows that the second symbol is x_2 as well. The decoder works on in the same manner and decodes symbol by symbol until all of them are recovered.

Below is the pseudocode to calculate the new number after symbol x is decoded:

$$\text{Number} = \frac{\text{Number} - \text{LowLimit}(x)}{\text{HighLimit}(x) - \text{LowLimit}(x)}. \tag{3.20}$$

Unlike Huffman code, which can just stop decoding when all bits are read from the bitstream, sometimes, it is hard for the arithmetic decoder to determine when to stop. One possibility is to ask the decoder to stop when the new calculated number becomes zero. However, book [109] gives an example to illustrate that this criterion could make the decoder stop decoding earlier than expected in some cases. One common method to solve the decoder's terminating problem is to let the encoder send the size of input stream at the beginning of the bitstream. Another solution is to add one special symbol EOF into the alphabet, and allocate a considerable small probability to EOF. By appending EOF at the end of the input stream, the decoder can stop decoding when it decodes an EOF symbol.

3.3.2. Implementation issues

The encoding and decoding process described in the previous section uses floating-point calculation and assumes that unlimited precision can be stored when doing the calculation. However, in real applications, this assumption is untrue and the floating-point arithmetic is slow and may have precision lost. Any practical implementation should only use integers, which should not be too long.

We consider using two four-decimal-digit long integers Low and High[3] to implement the arithmetic coder. Variables Low and High have the same meaning as described in the previous section, that is, they are the lower and higher limits of the current interval. Since we can only store four decimal digits, whenever the leftmost digits of Low and High become identical, we output the leftmost digit and update the new Low and High values by left shifting one digit and adding one more digit at the right. For the variable Low, a zero is added at the right. While for the variable High, a nine is added at the right. This process can be easier to understand if we think these two variables have the following format. Low is a number of $xxxx00, \ldots, 0$ and High is a number of $xxxx99, \ldots, 9$. Initially, Low contains 0000 and High contains 9999.

Now we look at an example to encode a sentence of "I␣AM␣A␣MAN" practically. The statistical model of all symbols in this example is listed in Table 3.1. When the symbols' frequency information is available, we divide the range [0,1)

[3] In practice, Low and High may be 16- or 32-bit long.

Table 3.1. Statistical model of five symbols (total CumFreq = 10).

Symbol	Freq.	Prob.	Range	CumFreq
A	5	5/10 = 0.5	[0.5, 1.0)	5
I	1	1/10 = 0.1	[0.4, 0.5)	4
M	2	2/10 = 0.2	[0.2, 0.4)	2
N	1	1/10 = 0.1	[0.1,0.2)	1
␣	1	1/10 = 0.1	[0.0, 0.1)	0

Table 3.2. Practical implementation of encoding "I␣AM␣A␣MAN."

Input symbol	Float Low & High	Integer Low & High	Output bit(s)	New integer Low & High
I	Low = $0 + (1 - 0) \times 0.4 = 0.4$	4000	4	0000
	High = $0 + (1 - 0) \times 0.5 = 0.5$	4999	4	9999
␣	Low = $0 + (1 - 0) \times 0.0 = 0.0$	0000	0	0000
	High = $0 + (1 - 0) \times 0.1 = 0.1$	0999	0	9999
A	Low = $0 + (1 - 0) \times 0.5 = 0.5$	5000	—	5000
	High = $0 + (1 - 0) \times 1.0 = 1.0$	9999	—	9999
M	Low = $0.5 + (1 - 0.5) \times 0.2 = 0.6$	6000	6	0000
	High = $0.5 + (1 - 0.5) \times 0.4 = 0.7$	6999	6	9999
␣	Low = $0 + (1 - 0) \times 0.0 = 0$	0000	0	0000
	High = $0 + (1 - 0) \times 0.1 = 0.1$	0999	0	9999
A	Low = $0 + (1 - 0) \times 0.5 = 0.5$	5000	—	5000
	High = $0 + (1 - 0) \times 1.0 = 1$	9999	—	9999
␣	Low = $0.5 + (1 - 0.5) \times 0.0 = 0.5$	5000	5	0000
	High = $0.5 + (1 - 0.5) \times 0.1 = 0.55$	5499	5	4999
M	Low = $0 + (0.5 - 0) \times 0.2 = 0.1$	1000	1	0000
	High = $0 + (0.5 - 0) \times 0.4 = 0.2$	1999	1	9999
A	Low = $0 + (1 - 0) \times 0.5 = 0.5$	5000	—	5000
	High = $0 + (1 - 0) \times 1.0 = 1$	9999	—	9999
N	Low = $0.5 + (1 - 0.5) \times 0.1 = 0.55$	5500	5500	—
	High = $0.5 + (1 - 0.5) \times 0.2 = 0.6$	5999	—	5999

into five subintervals as shown in the fourth column of Table 3.1. Table 3.2 describes all details of the implementation process. Column 1 indicates the symbol to be processed. Column 2 shows the floating-point value of Low and High variables. Column 3 shows the actual digits stored to represent Low and High. Column 4 lists the digit(s) output to the bitstream. If the leftmost digits of Low and high are identical, we output this identical digit. If all symbols have been processed, we output the four digits representing the current variable Low. Column 5 shows the updated four digits of Low and High to be used in coding the next symbol. The final output of this example is 4060515500.

The decoder starts with Low = 0000, High = 9999, and works in the opposite way. Initially, only the highest four digits of the compressed stream is read in, that

is, Number = 4060. Variables Low, High, and Number are updated at each step of the decoding process. Once the most significant digits of Low and High become the same, they are shifted out. At the same time, Number is also left shifted one digit. At each step, the index of the symbol is calculated. By searching the statistics model and comparing the index with the cumulative frequencies shown in Table 3.1, we can decode the current symbol.

Each time a symbol is decoded, we perform the following four steps.

(1) Calculate the symbol index:
index = $\lfloor((\text{Code} - \text{Low} + 1) \times 10 - 1)/(\text{High} - \text{Low} + 1)\rfloor$.

(2) Compare the index with the cumulative frequencies' column in Table 3.1. Suppose the index calculated in step 1 is 3. Since 3 is between the cumulative frequency 2 and 4 in the table, so symbol 'M' is decoded.

(3) Update variables Low and High according to

$$\begin{aligned} \text{Low} &= \frac{\text{Low} + (\text{High} - \text{Low} + 1)\text{LowCumFreq}[x]}{10}; \\ \text{High} &= \frac{\text{Low} + (\text{High} - \text{Low} + 1)\text{HighCumFreq}[x]}{10} - 1, \end{aligned} \tag{3.21}$$

where LowCumFreq[x] and HighCumFreq[x] are the cumulative frequencies of symbol x and of the symbol above x in Table 3.1.

(4) If the leftmost digits of Low and High are identical, left shift Low, High, and Number one digit. Set 0 as the lowest digit of Low, set 9 as the lowest digit of High, and let Number read in one more digit from the compressed stream.

The decoding steps for the previous example is listed below.

0. Initially set Low = 0000, High = 9999, and Number = 4060.
1. Index = $\lfloor[(4060 - 0 + 1) \times 10 - 1]/(9999 - 0 + 1)\rfloor = 4$. Decode symbol 'I.'
   Low = $0 + (9999 - 0 + 1) \times 4/10 = 4000$.
   High = $0 + (9999 - 0 + 1) \times 5/10 - 1 = 4999$.
   Shift 4 out, and update variables Low = 0000, High = 9999, and Number = 0605.
2. Index = $\lfloor[(0605 - 0 + 1) \times 10 - 1]/(9999 - 0 + 1)\rfloor = 0$. Decode symbol '␣.'
   Low = $0 + (9999 - 0 + 1) \times 0/10 = 0000$.
   High = $0 + (9999 - 0 + 1) \times 1/10 - 1 = 0999$.
   Shift 0 out, and update variables Low = 0000, High = 9999, and Number = 6051.
3. Index = $\lfloor[(6051 - 0 + 1) \times 10 - 1]/(9999 - 0 + 1)\rfloor = 6$. Decode symbol 'A.'
   Low = $0 + (9999 - 0 + 1) \times 5/10 = 5000$.
   High = $0 + (9999 - 0 + 1) \times 10/10 - 1 = 9999$.
4. Index = $\lfloor[(6051 - 5000 + 1) \times 10 - 1]/(9999 - 5000 + 1)\rfloor = 2$. Decode symbol 'M.'
   Low = $5000 + (9999 - 5000 + 1) \times 2/10 = 6000$.

High = 5000 + (9999 − 5000 + 1) × 4/10 − 1 = 6999.
Shift 6 out, and update variables Low = 0000, High = 9999, and Number = 0515.

5. Index = ⌊[(0515 − 0 + 1) × 10 − 1]/(9999 − 0 + 1)⌋ = 0. Decode symbol '␣.'
   Low = 0 + (9999 − 0 + 1) × 0/10 = 0000.
   High = 0 + (9999 − 0 + 1) × 1/10 − 1 = 0999.
   Shift 0 out, and update variables Low = 0000, High = 9999, and Number = 5155.
6. Index = ⌊[(5155 − 0 + 1) × 10 − 1]/(9999 − 0 + 1)⌋ = 5. Decode symbol 'A.'
   Low = 0 + (9999 − 0 + 1) × 5/10 = 5000.
   High = 0 + (9999 − 0 + 1) × 10/10 − 1 = 9999.
7. Index = ⌊[(5155 − 5000 + 1) × 10 − 1]/(9999 − 5000 + 1)⌋ = 0. Decode symbol '␣.'
   Low = 5000 + (9999 − 5000 + 1) × 0/10 = 5000.
   High = 5000 + (9999 − 5000 + 1) × 1/10 − 1 = 5499.
   Shift 5 out, and update variables Low = 0000, High = 4999, and Number = 1550.
8. Index = ⌊[(1550 − 0 + 1) × 10 − 1]/(4999 − 0 + 1)⌋ = 3. Decode symbol 'M.'
   Low = 0 + (4999 − 0 + 1) × 2/10 = 1000.
   High = 0 + (4999 − 0 + 1) × 4/10 − 1 = 1999.
   Shift 1 out, and update variables Low = 0000, High = 9999, and Number = 5500.
9. Index = ⌊[(5500 − 0 + 1) × 10 − 1]/(9999 − 0 + 1)⌋ = 5. Decode symbol 'A.'
   Low = 0 + (9999 − 0 + 1) × 5/10 = 5000.
   High = 0 + (9999 − 0 + 1) × 10/10 − 1 = 9999.
10. Index = ⌊[(5500 − 5000 + 1) × 10 − 1]/(9999 − 5000 + 1)⌋ = 1. Decode symbol 'N.'
    Low = 5000 + (9999 − 5000 + 1) × 1/10 = 5500.
    High = 5000 + (9999 − 5000 + 1) × 2/10 − 1 = 5999.
    Shift 5 out, and update variables Low = 5000, High = 9999. Since no more digits are available in the compressed stream, the decoding process is terminated.

3.3.3. Solving underflow problem

One problem may appear in the above arithmetic implementation. In the case when variables Low and High converge closely, they may not converge to numbers with identical most-significant digits. For example, Low and High may reach values 7999 and 8000. Since the leftmost digits are not the same, the previous algorithm will not shift, and operations afterwards will not be able to make the most significant digits identical. Thus no output digits will be generated until the last symbol is being processed. This kind of problem is called *underflow*.

Table 3.3. A ten-symbol alphabet and their counts.

Symbol	x_1	x_2	x_3	x_4	x_5	x_6	x_7	x_8	x_9	x_{10}
Count	12	14	14	2	5	1	2	20	14	9

(a)

Symbol	x_8	x_2	x_3	x_9	x_1	x_{10}	x_5	x_4	x_7	x_6
Count	20	14	14	14	12	9	5	2	2	1

(b)

Symbol	a_8	a_9	a_3	a_2	a_1	a_{10}	a_5	a_4	a_7	a_6
Count	20	15	14	14	12	9	5	2	2	1
LowCumFreq	74	59	45	31	19	10	5	3	1	0

(c)

To solve the underflow problem, we can rescale variables Low and High described below. If Low and High become 79xx and 80yy, respectively, rescaling should be done by squeezing out the second highest digits and adding a zero and nine at the lowest digit. In other words, Low and High will be set to 7xx0 and 8yy9. The algorithm may need to rescale several times before the most-significant digits become identical. The number of times rescaling is performed should be counted. Once the most significant digits of Low and High become equal, we should output this digit followed by a series of zeros (if they converged to 7) or nines (if they converged to 8). The number of zeros or nines should be the same as the number of rescaling occurrences.

3.3.4. Adaptive arithmetic coding

Same as Huffman coding, arithmetic coding can also be extended to an adaptive algorithm. We can easily understand how adaptive arithmetic coding works after looking at the following two features.

(1) In the encoding part, the Low and High variables are updated each time using the cumulative frequencies of the current symbol and the symbol above the current symbol. This suggests, as long as the encoder and decoder follow the same rule, that the symbol's frequency can be updated each time a previous symbol is processed.

(2) Similarly, the order of symbols' subinterval is not important. Even a reordering can be done, as long as the decoder agrees to do the same as what the encoder does.

With the nonadaptive arithmetic coding, we need to build the statistical model for the entire alphabet before the real encoding is performed. With the adaptive

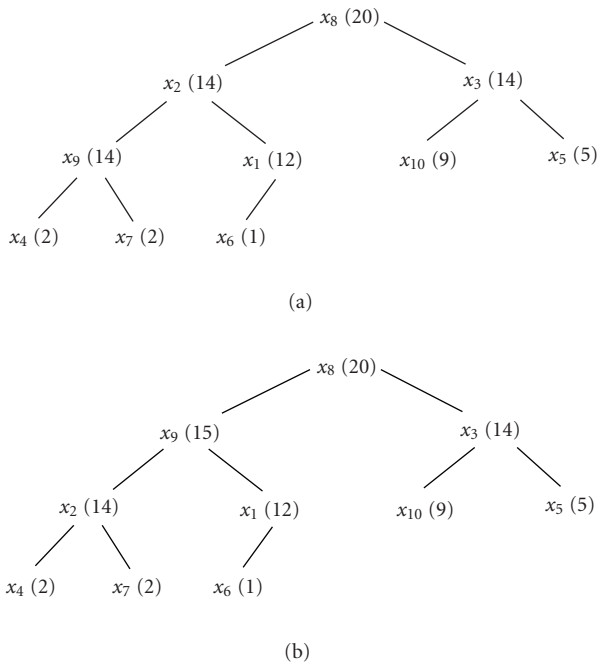


FIGURE 3.5. An example of adaptive arithmetic coding.

algorithm, we can calculate the symbols' frequencies and their accumulated frequencies on the run and complete the encoding and decoding based on the current statistical information. One important thing needs to be borne in mind when adaptive arithmetic encoding is performed and that is the frequencies of symbols should only be updated after, not before, each symbol has been processed. This is because the decoder needs to update the frequency information in exactly the same way as the encoder, and it will not be able to know the new symbol's occurrence before it has been processed.

A commonly adopted adaptive arithmetic model uses an array to keep track of the count of each symbol (occurrence frequency) and the corresponding cumulative frequencies. This array should be sorted in the order of symbol's count. Whenever a symbol is processed, after incrementing this symbol's occurrence frequency and updating the affected cumulative frequency, we should check the array and always make sure that the updated occurrences frequency are in the sorted order.

By using a balanced binary tree,[4] we can efficiently perform the search and update procedure. From the knowledge of data structure, if the alphabet has size n, we know that the height of the balanced binary tree is $\lceil \log_2 n \rceil$. The tree can

[4]A balanced binary tree is a complete binary tree where only some of the bottom-right nodes may be missing.

be arranged in a way such that any given node has its occurrence frequency larger than that of its children. A binary tree with the above property is called a *heap*.

We look at an example to see how the adaptive arithmetic coding works. Suppose we have ten-symbol alphabet, and their counts are listed in Table 3.3(a). Table 3.3(b) shows the same symbols set sorted by their counts.

Figure 3.5(a) illustrates a heap containing all symbols in the alphabet. In practice, this tree can be implemented by an array with the following rule:

(1) the first element of the array represents the root of the tree;
(2) the children of the ith element of the array are the $2i$th and $(2i + 1)$th elements in the array;
(3) the parent of the ith element of the array is the $\lfloor i/2 \rfloor$th element in the array.

The ith element of the array contains the symbol with the ith largest count. With a heap, the location of any array element in a tree can also be easily found. Below is an example that shows how to find the location of the tenth array element from the root.

(1) Dividing 10 by 2, we get the quotient equal to 5 and the remainder equal to 0. The quotient 5 means that the parent of the tenth element is the fifth element. The remainder 0 means that the tenth element is the left child of its parent.

(2) Now we need to find the location of the fifth element. So we divide 5 by 2, and get the quotient equal to 1 and the remainder equal to 1. The quotient 1 means that the parent of the fifth element is the second element. The remainder 1 means that the fifth element is the right child of its parent.

(3) Continue in this fashion. We divide 2 by 2 and get the quotient equal to 1 and the remainder equal to 0. The quotient 1 means that the parent of the second element is the first element, that is, the root, and the remainder 1 means that the second element is the right child of the root. When the root is found, the process stops.

Now we look at an example to illustrate how the tree and the array change when the next symbol comes in. Suppose x_9 is the next symbol to be processed. The count of symbol x_9 should be increased from 14 to 15. In order to keep the array in the sorted order, we check if the count of the parent of x_9 is still larger than that of x_9, if not, switch them. In our example, node x_2 should be switched with node x_9. Then after switching, we check again if the x_9's parent count is larger. We continue doing this until we reach the root of the tree. And we now have an updated tree as shown in Figure 3.5(b).

The array with each symbol's updated count and accumulative frequency are shown in Table 3.3(c). The LowCumFreq$[x_i]$ can be calculated by

$$\text{LowCumFreq}[x_{n-i}] = \begin{cases} 0, & i = 0, \\ \text{LowCumFreq}[x_{n-i+1}] + \text{count}[x_{n-i+1}], & 1 \le i < n. \end{cases} \tag{3.22}$$

And the HighCumFreq$[x_i]$ is equal to LowCumFreq$[x_{i-1}]$, with HighCumFreq$[x_0]$ equal to the sum of LowCumFreq$[x_0]$ and count$[x_0]$.

3.4. QM coding

3.4.1. QM encoder

QM coding is a successor of arithmetic coding and is also an adaptive algorithm. The most important feature of QM coder is that it only deals with binary symbols, that is, 0 and 1. Since there are only two different symbols in the alphabet, we can distinguish them by identifying one of them as the more probable symbol (MPS) and the other to be the less probable symbol (LPS). Each time the next bit is read in, the QM encoder determines whether or not it is an MPS in the current statistical model and then compresses it accordingly. To be more precise, the QM coder does not really code symbol 0 or 1, rather, it only codes whether the symbol is MPS or LPS. The decoder uses the same statistical model and tries to determine whether the next symbol is MPS or LPS. Each time a symbol is processed, we update the model and this update should be kept synchronized at both the encoder and decoder sides.

In QM coder, Qe is defined as the probability of the LPS. Therefore, the probability of the MPS is $(1 - Qe)$. From this definition, we expect Qe to be in the range of $[0, 0.5]$, and $(1 - Qe)$ to be in the range of $[0.5, 1]$. Let A be the initial probability interval. LPS and MPS divide the interval A into two subintervals with the width of LPS's subinterval equal to $A \times Qe$ and that of MPS's subinterval equal to $A \times (1 - Qe)$. This is shown in Figure 3.6.

The QM coder operates in a similar way as the conventional arithmetic coding. The output of QM encoder denoted by C can be considered as the bottom of the new interval each time a symbol is processed. In other words, whenever an MPS is input into the encoder, C is modified to be the bottom of the MPS subinterval, while if an LPS is input into the encoder, C is modified to be the bottom of the LPS subinterval. Accordingly, A is shrunk to the size of the corresponding subinterval. The probability of the input stream is within the range of $[C, C + A)$. And the width of the interval A becomes smaller and smaller as more and more bits are processed. Using the subinterval division method shown in Figure 3.6, we should update C and A according to the following rules:

(1) after MPS: C is unchanged, $A \rightarrow A(1 - Qe)$,

(2) after LPS: $C \rightarrow C + A(1 - Qe), A \rightarrow A \times Qe$.

With these rules, C is set to a point at the bottom of the MPS or the LPS subinterval and A is set to the new size of the subinterval.

Table 3.4 lists the values of A and C when four symbols (MPS, LPS, LPS, and MPS) are processed. In this simple example, we assume that Qe remains to be 0.5. In the real case, Qe may be changed each time a new bit is processed. After coding these four bits, C and A are updated to 0.625 and 0.0625, respectively. The probability of receiving a sequence containing these four bits is within the range of $[0.625, 0.625 + 0.0625)$.

The above example operates in the same way as a conventional arithmetic coder with a binary symbol set. QM coder actually works in a slightly different way so that a faster computation can be achieved. What makes the previous method inefficient is that it involves many multiplications and divisions. If these

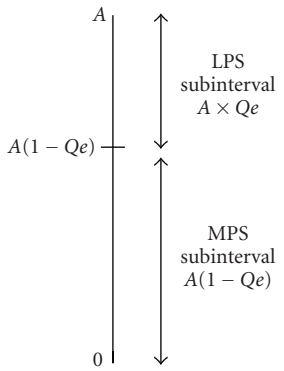


FIGURE 3.6. The probability interval of QM coder.

TABLE 3.4. Encoding four symbols with $Qe = 0.5$.

Symbol	C	A
Initially	0	1
s1 (MPS)	Unchanged (0)	$1 \times 0.5 = 0.5$
s2 (LPS)	$0 + 1 \times (1 - 0.5) = 0.5$	$0.5 \times 0.5 = 0.25$
s3 (LPS)	$0.5 + 0.25 \times (1 - 0.5) = 0.625$	$0.25 \times 0.5 = 0.125$
s4 (MPS)	Unchanged	$0.125 \times (1 - 0.5) = 0.0625$

multiplications and divisions can be replaced by additions, subtractions, and shifts, then the whole coding process can be much more efficient.

The QM coder initialized A to be the value of 1 and maintained it to be a number close to 1. This is done by doing *renormalization*. Whenever A becomes too small, we double its value. At the same time, C should be doubled too. In the real implementation, the doubling operation only involves a logical left shift, thus can be done fast. Now we need to find the criterion on when the renormalization should be performed. If A is doubled when it is just a little smaller than 1, say 0.9, then renormalizing A yields 1.8, which is closer to 2 than to 1. If the value of A is not normalized until it is below 0.5, then doubling it once still results in a value less than 1. A reasonable threshold to perform the renormalization is 0.75. Therefore, when A becomes 0.75 at a certain step, it will be renormalized to 1.5. If A reaches a value a little smaller than 0.75, it will be doubled to a number even closer to 1. However, if A reaches a value that is close to 0, it has to be renormalized several times. Consequently, renormalization makes A a number close to 1 and never larger than 1.5.

Since A is a value close to 1, then $A \times Qe$, which is a main multiplication involved in each update procedure, can be estimated simply by Qe itself. This approximation process changes the main rules of the QM encoder to

(1) after MPS: C is unchanged, $A \to A(1 - Qe) \approx A - Qe$,

(2) after LPS: $C \to C + A(1 - Qe) \approx C + A - Qe, A \to A \times Qe \approx Qe$.

TABLE 3.5. Renormalization added to Table 3.4.

Symbol	C	A	Renor. A	Renor. C
Initially	0	1	—	—
s1 (MPS)	0	$1 - 0.5 = 0.5$	1	0
s2 (LPS)	$0 + 1 + 0.5 = 1.5$	0.5	1	3
s3 (LPS)	$3 + 1 + 0.5 = 4.5$	0.5	1	9
s4 (MPS)	9	$1 - 0.5 = 0.5$	1	18

In the real implementation of QM coder, we need to choose an integer representation for A so that any value in the range of $[0, 1.5)$ can be represented as an integer. Suppose we are using a computer such that any integer can be stored in 16 bits. We can choose 16 zero bits to represent 0 and

$$2^{16} = 65536_{10} = 10000_{16} = 1\underbrace{0\ldots0}_{16}{}_2 \tag{3.23}$$

to represent 1.5. The hexadecimal representations of several important values in QM coding are listed below:

$$\begin{aligned} 0.25 &= \frac{21845}{2} = 10923_{10} = 2AAB_{16}, \\ 0.5 &= \frac{43690}{2} = 21845_{10} = 5555_{16}, \\ 0.75 &= \frac{1.5}{2} = 2^{15} = 32768_{10} = 8000_{16}, \\ 1 &= 0.75\frac{4}{3} = 43690_{10} = AAAA_{16}. \end{aligned} \tag{3.24}$$

Note that the choice of associating 1.5 with 2^{16} is arbitrary. As long as we can achieve accurate interval subdivision, different integer representations can be chosen.

If we include the renormalization, the main rules of the QM encoder is changed to

(1) after MPS: C is unchanged, $A \leftarrow A - Qe$, if $A < 8000_{16}$ renormalize A and C,

(2) after LPS: $C \leftarrow C + A + Qe, A \leftarrow Qe$, renormalize A and C.

Table 3.5 lists results of applying the new rules to the examples shown in Table 3.4.

Another important problem in need to be taken care of in the design of the QM coder is called *interval inversion*. This needs to be done when the subinterval allocated to the LPS becomes larger than that of MPS. This problem is a result of the approximation and may happen when Qe is close to 0.5. Table 3.6 illustrates an example of interval inversion. After three MPSs are encoded, the interval becomes 0.78. Since it is larger than the renormalization threshold 0.75,

Table 3.6. An example of interval inversion ($Qe = 0.46$).

Symbol	C	A	Renor. A	Renor. C
Initially	0	1	—	—
s1 (MPS)	0	$1 - 0.46 = 0.54$	1.08	0
s2 (MPS)	0	$1.08 - 0.46 = 0.62$	1.24	0
s3 (MPS)	0	$1.24 - 0.46 = 0.78$	—	—
s4 (MPS)	0	$0.78 - 0.46 = 0.32$	0.64	0

```
After MPS:
    C is unchanged;
    A = A − Qe;              (A is set to be the MPS's subinterval)
    if A < 8000_16           (check if reorganization needs to be done)
        if A < Qe            (check if inversion is needed)
            C = C + A;       (C updates to the bottom of LPS)
            A = Qe;          (A is set to be the LPS's subinterval)
        endif;
    renormalize A and C
    endif;

After LPS:
    A = A − Qe;              (A is set to be the MPS's subinterval)
    if A ≥ Qe                (if no interval exchange needs to be performed)
        C = C + A;           (C updates to the bottom of LPS)
        A = Qe;              (A is set to be the LPS's subinterval)
    endif;
    renormalize A and C;
```

Algorithm 3.1. The pseudocode for QM encoder.

no renormalization is performed. Then the subinterval allocated to MPS becomes $A - Qe = 0.78 - 0.46 = 0.32$, which is not as large as that of LPS, which is $Qe = 0.46$.

Whenever an interval inversion occurs, we need to swap the LPS and MPS so that the old MPS becomes LPS and the old LPS becomes MPS. This operation is called *conditional exchange*. Clearly, a conditional exchange needs to be performed when $A - Qe < Qe$. Since $Qe \leq 0.5$, $A - Qe < Qe \leq 0.5 < 0.75$, which means that the interval inversion only happens when a renormalization is needed. The pseudocode that combined all steps stated in this section is shown in Algorithm 3.1.

```
A = A − Qe;
if A > C
        if A ≥ 8000_16
            Decode: MPS;
        else
            if A ≥ Qe
                Decode: MPS;
            else
                Decode: LPS;
            endif;
             renormalize A and
C;
        endif;
   else
        if A ≥ Qe
            C = C − A;
            A = Qe;
            Decode: LPS;
        else
            C = C − A;
            A = Qe;
            Decode: MPS;
        endif;
        renormalize A and C;
   endif;
```

Algorithm 3.2. The pseudocode for QM decoder.

3.4.2. QM decoder

The QM decoder is a reverse process of the QM encoder. The rules to update C and A in QM decoder are given by

$$\begin{aligned} &\text{after MPS: } C \text{ is unchanged, } A \longleftarrow A(1-Qe) \approx A - Qe, \\ &\text{after LPS: } C \longleftarrow C - A(1-Qe) \approx C - A + Qe,\ A \longleftarrow A \times Qe \approx Qe. \end{aligned} \tag{3.25}$$

And the pseudocode for the QM decoder is shown in Algorithm 3.2.

3.4.3. Probability estimation

As mentioned before, QM coder is an adaptive binary arithmetic coder. The statistical model of QM coder needs to be updated in order to reflect the up-to-date probability estimation. The intuitive method to estimate symbols' frequencies is

Table 3.7. An example of QM coder's probability estimation.

Qe state	Hex *Qe*	Dec *Qe*	Decr LPS	Incr MPS	MPS exch	*Qe* state	Hex *Qe*	Dec *Qe*	Decr LPS	Incr MPS	MPS exch
0	0AC1	0.50409	0	1	1	15	0181	0.07050	2	1	0
1	0A81	0.49237	1	1	0	16	0121	0.05295	2	1	0
2	0A01	0.46893	1	1	0	17	00E1	0.04120	2	1	0
3	0901	0.42206	1	1	0	18	00A1	0.02948	2	1	0
4	0701	0.32831	1	1	0	19	0071	0.02069	2	1	0
5	0681	0.30487	1	1	0	20	0059	0.01630	2	1	0
6	0601	0.28143	1	1	0	21	0053	0.01520	2	1	0
7	0501	0.23456	2	1	0	22	0027	0.00714	2	1	0
8	0481	0.21112	2	1	0	23	0017	0.00421	2	1	0
9	0441	0.19940	2	1	0	24	0013	0.00348	3	1	0
10	0381	0.16425	2	1	0	25	000B	0.00201	2	1	0
11	0301	0.14081	2	1	0	26	0007	0.00128	3	1	0
12	02C1	0.12909	2	1	0	27	0005	0.00092	2	1	0
13	0281	0.11737	2	1	0	28	0003	0.00055	3	1	0
14	0241	0.10565	2	1	0	29	0001	0.00018	2	0	0

to initialize *Qe* to 0.5 and update it by counting the number of bits 0 and 1 that have been processed so far. For instance, if 100 bits have been processed so far, and among them 80 are ones and 20 are zeros, then 1 is the current MPS with estimated probability equal to 0.8 and 0 is the current LPS with estimated probability equal to 0.2. Note that at the encoder side, the statistical model should be updated after, not before, a symbol has been encoded. Because, otherwise, the decoder would not be able to mirror the same probability estimation as the encoder and it could not correctly decode the symbols thereafter. Although this method produces good results, it is inefficient and is never used in real implementation since *Qe* has to be updated frequently (ideally after each processed symbol) and there are divisions involved in the calculation.

The method adopted by the QM coder to do the probability estimation is based on a table of preset *Qe* values. Initially, *Qe* is set to 0.5 and is only modified after renormalization, not after processing each symbol. Table 3.7 gives an example on how to perform this process. (Note that this is just an illustration. In real compression, QM coder may use a much more complex table. For example, the probability estimation of JPEG is prepared based on Nayesian statistics.) The initial state of *Qe*, that is, state 0, set *Qe* to be $0AC1_{16}$, which is close to 0.5. Suppose, at state k, we have "Decr LPS" and "Incr MPS" values equal to d and i. This means, at the *Qe*'s current state k, that if an LPS renormalization occurs first, we should update the *Qe* to state $k - d$; while if an MPS renormalization occurs first, we should update the *Qe* to state $k+i$. Notice that there is also a column labeled "MPS exch" in Table 3.7. This column shows the conditional exchange information of the MPS and LPS.

Compared with the intuitive probability estimation method, this table lookup approach is much simpler and faster: the *Qe* value can be found in the table by incrementing or decrementing the current state and no division is needed. Moreover, the statistical model does not have to be updated frequently.

4 Introduction to psychoacoustics

One of the key elements in the development of reduced bit rate audio is the understanding and application of psychoacoustics. All of the current perceptual audio coders achieve high compression rates by exploiting the fact that signal information that cannot be detected by even a well-trained listener can be discarded. Several psychoacoustic principles, such as the absolute hearing threshold and masking, have been incorporated in codec design. In this chapter, we discuss some of the basic principles of psychoacoustics that are commonly used in perceptual audio coding.

4.1. Perception of loudness

While it is a relatively simple task to measure the sound pressure level of a sound source, it is quite a challenging task to relate the measured level to a perceived level by human listeners. The perceptual quantity that describes the magnitude of the source as experienced by a listener is called *loudness*. The difficulty in making a direct correlation between measured sound pressure level (SPL) and perceived loudness lies in the fact that there are multiple mechanisms responsible for producing the sensation. For example, it is common for people listening to reproduced music to observe changes in loudness when equalization is applied to the signal thus altering the balance between low and high frequencies.

There are several aspects of loudness that relate to perceptual audio codecs. One has to deal with the minimum threshold of hearing and how that varies with frequency. Another deals with how the hearing mechanism determines differences in loudness level when other parameters (e.g., spectrum or duration) are kept constant.

First, let us begin by defining the sound pressure level (SPL)

$$L = 20 \log \left(\frac{p}{p_0} \right) \text{dB(SPL)}, \tag{4.1}$$

in which p is the pressure produced by a sound source and p_0 is the reference pressure of 20 μPa that corresponds to the minimum audible threshold of human

hearing for a 1 kHz tone. It is also useful to describe SPL in terms of sound intensity. Following the notation in [44] we can write the instantaneous intensity of sound as

$$I(t) = p(t)u(t), \tag{4.2}$$

in which $p(t)$ is the instantaneous sound pressure and $u(t)$ is the fluid velocity. For a plane wave, the fluid velocity is related to pressure by

$$u(t) = \frac{p(t)}{(\rho c)}, \tag{4.3}$$

in which ρ is the density (in the case of air $\rho = 1.29\,\text{kg/m}^3$) and c is the speed of sound in air. From the above two equations, the intensity can be written as a function of pressure as:

$$I(t) = \frac{p^2(t)}{(\rho c)}. \tag{4.4}$$

Using equation (4.4) we can calculate the reference intensity corresponding to the reference pressure of 20 μPa.

$$I_0 = \frac{p_0^2}{(\rho c)} = \frac{p_0^2}{\left[(1.29\,\text{kg/m}^3)(343.6\,\text{m/s})\right]} = 0.93 \times 10^{-12}\,\text{W/m}^2, \tag{4.5}$$

which is often approximated (to account for temperature variations in the speed of sound) to $I_0 = 10^{-12}\,\text{W/m}^2$. The sound pressure level of a sound source with intensity I, is then given by:

$$L = 10 \log\left(\frac{I}{I_0}\right) \text{dB(SPL)}. \tag{4.6}$$

The dependence of loudness on SPL has been studied and is now an international standard [57]. It was found to relate the perceptual magnitude H to the physical sound intensity I by a power law

$$H = kI^{0.6}, \tag{4.7}$$

which holds over a range of SPL levels from about 40 dB SPL to 120 dB SPL and shows a slight deviation at lower SPL levels. Note that the power law exponent of 0.6 corresponds to a loudness doubling for every 10 dB increase in intensity. So a perceptual "doubling" is not the same as an electrical signal doubling.

Perhaps the most important characteristic of loudness perception for the design of audio codecs is this dependence of loudness on frequency. This relationship was investigated by Fletcher and Munson [34] over headphones and later Robinson and Dadson [106] for a free field (Figure 4.1). A different sound intensity is

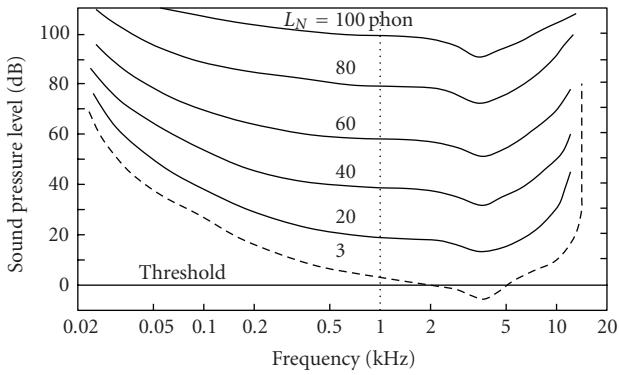


FIGURE 4.1. Equal loudness contours for pure tones. The curves show the sound pressure level required at each frequency to achieve a perceived loudness equal to a 1 kHz tone. Note that the curves tend to become flatter across frequency as the absolute loudness level is raised [148].

required at each frequency in order to produce the same level of perceived loudness. Human listeners are much less sensitive to low frequencies and thus a much higher SPL level is required to produce the same perceived loudness as that of a sound at high frequencies. The frequency-dependent intensity (or SPL) levels that give rise to the same perceived loudness are known as *equal loudness levels*. In fact, because these levels also depend on the absolute intensity, they form a family of curves known as the *equal loudness contours* (Figure 4.1). Each curve corresponds to the SPL at which a 1 kHz tone is perceived to be equally loud as a tone at another frequency. For example, on the 20 phon curve, a sound at 50 Hz must be 30 dB louder in sound pressure level in order to be perceived as having the same loudness as the reference tone at 1 kHz. Although the loudness level is measured in dB, it is distinguished from SPL and it is designated by a unit called the *phon*. The lowest curve in the figure is called the *minimum audible field (MAF)* and represents the hearing threshold averaged over a large population sample.

At low levels, the equal loudness curves resemble the shape of the MAF curve, but as the absolute loudness level increases the curves become flatter. This shows that the difference in balance between low and high frequencies decreases at high levels.

4.2. Masking

Another basic characteristic of human hearing that is exploited in perceptual audio coding is that of masking. It is relatively simple to observe this phenomenon in everyday life by simply noticing that a very loud sound (the masker signal) can prevent another sound (the maskee signal) from being heard. For example, a person near running water has difficulty hearing another person talking to them because the water acts as the masker signal. From studies of the ear physiology, it is known that the physical mechanism for producing sound cues in the brain begins with electrical discharges at nerve endings on the basilar membrane. While

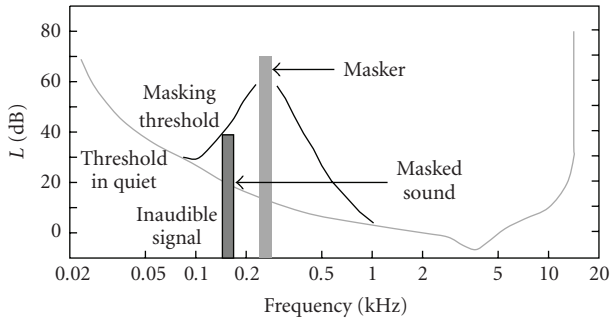


FIGURE 4.2. The masker signal causes a threshold shift that locally raises the minimum audible threshold. Any other signal with frequency components within the raised threshold range will not be audible and will therefore be ignored by a perceptual codec.

this discharge is occurring, the nerves involved cannot be stimulated by another sound. Thus if another sound happens to require some subset of the same nerves, it will be *masked*. The first published observation of this phenomenon is by Mayer [91] in 1876. As described by Fletcher [34], Mayer observed that low-frequency sounds had a very different masking effect from high-frequency sounds. A more general definition of masking includes not just the case when the maskee signal is not audible, but also when the masker produces a reduction in the perceived loudness of the maskee. This phenomenon is called *partial masking* [111].

The first systematic experiments to determine the effects of masking and the role of parameters such as spectrum, level, and duration of the masking signal were conducted by Fletcher (for a complete review see [34]). Starting with pure tones, a masker tone was generated and kept at a constant level while signal tones of different frequencies were increased in intensity from their threshold of audibility level until they just became perceptible in the presence of the masker. The level difference from the audibility threshold without the masker represents the threshold shift (Figure 4.2). This effect of masking is particularly relevant to perceptual audio codecs particularly because the threshold shift varies linearly with the masker level [45].

Fletcher's experiments verified Mayer's observations that high frequencies are masked more than low frequencies. This implies that for complex sounds that contain multiple tones, masking will be manifest as an apparent "boost" in low frequencies because the high frequencies are more easily masked (and thus not perceived as intense). What is interesting about this low-frequency boost is that it only occurs in monaural masking experiments. If the masker and maskee signals are presented to each ear separately, this apparent interference does not occur.

4.2.1. Frequency masking

The masking of complex sounds that consist of multiple pure tones can be explained by "complex addition" of the effects produced by individual tones. That is

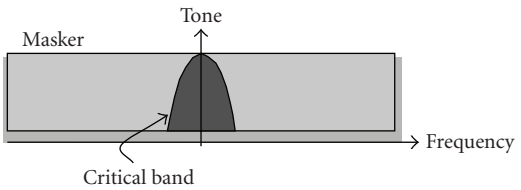


FIGURE 4.3. With a broadband masker, only the energy in the critical band around the pure tone gives rise to masking.

to say, the masking not only combines the effects of each tone, but also presents the effects of the sum and difference of the individual tones.

The spectrum of the masker signal also plays a critical role in masking. A pure tone (single line spectrum) can act as a masker. As Mayer showed [91], low frequency tones can mask higher frequencies; however, high frequencies are not good maskers of lower frequencies. Later studies by Wegel and Lane [133] verified this and went further to show that as the intensity of the masker signal is raised, masking spreads in the direction of higher frequencies, but not towards lower frequencies. The effect is called *upward spread of masking*. Furthermore, the width of the masking pattern at low frequencies is much wider than the width at higher frequencies. This implies that low frequency maskers have an effect over a much wider range of frequencies.

The human ear has the ability to act as a spectrum analyzer and analyze the frequency content in a signal using filter banks known as the auditory filters. The width of these auditory filters varies with the frequency around which they are centered. In Fletcher's work [28], it was shown that the masking threshold of a tone increased with the bandwidth of the masking noise. However, after a certain width of the masking noise there was no further increase in the masking effect. This suggested that each auditory filter has a certain critical bandwidth and Fletcher theorized at the threshold and the signal power and noise power would be equal within the critical bandwidth range. He defined a critical band as the ratio of signal to noise power, which when expressed in dB corresponds to the difference between the signal and the masking noise (Figure 4.3).

4.2.2. Temporal masking

So far we have discussed masking that arises due to the audibility threshold shift that a human perceives when a masking sound is present at the same time. It is also possible to create a masking effect when the masker and signal have occurred at different times. This is called *temporal masking* and can manifest itself as *backward masking* in which the signal is generated before the masker, or *forward masking* in which the signal comes after the masker (Figure 4.4).

In backward masking, the masker causes a rise in the threshold for signals that arrived *before* it. This may seem counterintuitive, but it has been demonstrated for time gaps on the order of 25 ms [31]. The level of the masker must be significantly higher than the signal for backward masking. Although the signal arrives at the ear

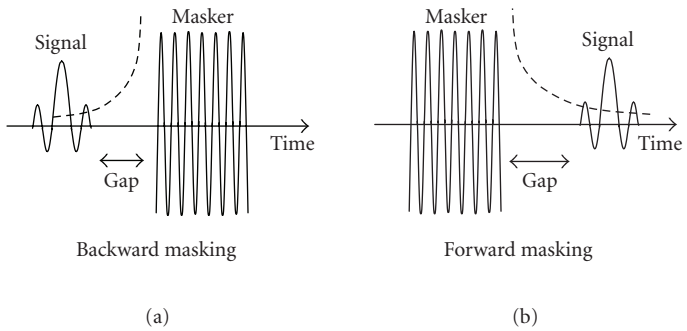


FIGURE 4.4. Temporal masking can occur in both the forward and backward directions.

earlier, the higher loudness masker is processed first thus giving rise to backward masking.

The most common type of temporal masking is forward masking. This effect happens over much longer time intervals between the signal and the masker that can reach 200 ms [31]. The amount of masking as well as the decay of the masking effect are directly proportional to the time gap between the masker and the signal.

4.2.3. Interaural masking

Most of the early masking experiments were monaural in nature with both the masker and the signal rendered in the same ear. Masking is also produced in a binaural situation with the masker in one ear and the signal in the other. As the intensity of the masker is raised it reaches a level at which it can mask a signal at the opposite (contralateral) ear. Although the reasons are not fully understood, it is hypothesized that the masker interacts with the signal much later in the hearing signal path, within the central nervous system at a point where binaural signal representation is maintained [149].

Interaural masking is a much weaker effect compared to monaural masking. It rises by a fraction of a dB as the masker level rises and decreases when there is a gap between the masker and the signal. Contrary to monaural masking, however, the effect is stronger for higher frequencies than lower frequencies. Zwislocki [149] showed that this type of masking correlates highly with the firing rate of neurons of the lower auditory nervous system.

5 Subjective evaluation of audio codecs

5.1. Introduction

Advances in audio codec performance can be evaluated and compared by using certain objective measures such as signal-to-noise ratio or the mask-to-noise ratio (MNR) [4] that is defined as the ratio of the masking threshold with respect to the energy of the coding error. Although these measures can give some indication of the relative performance among coding schemes, none are capable of accurately representing the preferences and sensitivities of the human ear to artifacts or the ability to judge the "transparency" of a codec. Because all low-bit-rate audio codecs discard audio signal data by definition, subjective evaluation is the only accepted method to evaluate both the audibility and "level of annoyance" of the inevitable artifacts that occur as the encoding bit rates are reduced.

In order to systematically assess the quality of compressed audio program material, it is critical to establish a methodology that removes as many variables from the subjective listening experiments as possible. This allows only those differences or artifacts that are due to the compression algorithms to present themselves to human listeners that must also be trained in the tasks they are to perform.

In this chapter, we describe the basic elements of the methodologies used in subjective listening evaluations, many of which have been codified in international standards. These elements include very precise specifications of the listening room environment, the type of program material, the methods of sound reproduction, the selection and training of listening panels, and the statistical analysis models that will yield meaningful results.

5.2. Listening environment specifications

It is critical to perform subjective evaluations in an environment in which the effects of room acoustics and distortions from the playback system have been minimized. This permits the listeners to attribute any audible artifacts to the coding algorithms rather than other unknown and uncontrolled parameters. For example, reproduction of program material over loudspeakers in a highly reverberant room will give rise to low intelligibility and masking from the multiple reflections that will arrive at the listeners ears after the desired signal from the loudspeakers.

One way to address the issue of room acoustics is to specify the use of headphones for sound reproduction. In a study conducted by the Consultative Committee for International Radio (CCIR), it was found that the proper conditions for headphone reproduction were achieved when the headphones were equalized for a flat response in a diffuse field [4, 7]. The headphone requirements are also further specified by the findings of Professor Gunther Theile which later led to an ITU standard (ITU-R BS.708) [20] and have also been specified by the IEC [6].

For listening evaluations using loudspeakers, a set of guidelines has been put forth by the European Broadcasting Union (EBU) [27] and also by the ITU (ITU-R BS.1116). The ITU requirements have been described in detail in other publications (see, e.g., [16]), so here we will provide an overview of the EBU recommendations.

The placement of listeners and loudspeakers should be such that the acoustical center of each loudspeaker in the reproduction system is at least 1.2 m from the floor. Furthermore, the height of the acoustical center from each loudspeaker should coincide with the seated ear height. This should be achieved by raising or lowering the loudspeakers rather than by tilting. If tilting is necessary, it should not exceed 10° in order to avoid off-axis effects in the loudspeaker frequency response particularly in the region of the crossover frequencies. In order to minimize boundary effects from nearby walls, the loudspeakers should be placed at a 1 m minimum distance from the surrounding walls. There also should not be any obstructing objects or surfaces between the loudspeakers and the listening area. The loudspeakers must be placed in a circle centered at the main listening position within the listening area. For experiments that involve the evaluation of sound imaging, the time-delay differences among loudspeakers must not exceed 100 microseconds.

The recommended listening room physical characteristics specified by the EBU begin with a minimum floor area of 40 m^2 (30 m^2 for a sound control room). The volume should not exceed 300 m^3. Even more important than the floor area and the volume are the required room dimension ratios that will give rise to an even distribution of normal modes. The range of recommended room dimension ratios are given below:

$$\begin{aligned} 1.1\left(\frac{w}{h}\right) \leq \frac{l}{h} \leq 4.5\frac{w}{h} - 4, \\ l < 3h, \\ w < 3h, \end{aligned} \tag{5.1}$$

in which l is length (largest floor dimension, regardless of orientation), w is width (shortest floor dimension, regardless of orientation), and h is height. Finally, ratios of l, w, and h that are ±5% of an integer number should be avoided.

The quality of the direct sound from the loudspeakers to the listening area is influenced by the characteristics of the loudspeakers. A set of minimum

requirements has been set forth by the EBU, but it is important to note that not all loudspeakers that meet these minimum requirements will be suitable for all types of subjective evaluations. A separate evaluation of the loudspeakers should be performed before conducting the listening tests.

This evaluation begins with a measurement of the loudspeaker free-field response that is usually taken in an anechoic chamber. The frequency response should be measured using a pink noise (constant energy per octave) test signal. The amplitude envelope of this signal decays as $1/f$, where f is the frequency in Hz. Measurements are to be taken in 1/3-octave bands on the main axis. The acceptable variation of the loudspeaker frequency response is ± 2 dB in the range from 40 Hz to 16 kHz. In addition to the on-axis measurements, frequency response must also be measured at $\pm 10^\circ$ and $\pm 30^\circ$ and the response at these angles must not vary from the on-axis response by more than ± 3 dB and ± 4 dB, respectively.

The directional radiation pattern of the loudspeakers must also be properly designed. In the horizontal plane, the polar pattern is measured with pink noise in standard octave spacing in the range from 250 Hz to 16 kHz. The maximum radiation must always be in the on-axis direction for all frequencies. The directivity index D of the loudspeaker is defined as the ratio of the on-axis power divided by the power delivered on-axis for a pure omnidirectional source. The allowable values for the directivity index are $4 \leq D \leq 12$ dB.

The EBU further specifies the allowable harmonic distortion in the system. This is typically measured with a sinusoidal signal that produces a sound pressure level of 90 dB in the range from 250 Hz to 2 kHz. The harmonic distortion levels within this range cannot be above 3% for the frequency range from 40 Hz to 250 Hz and 1% for the frequency range from 250 Hz to 16 kHz.

During the presentation of the program material, the maximum sound pressure level for continuous operation (defined as a minimum of 10 minutes) must not exceed 108 dB when measured with a sound level meter set to flat, RMS (slow) at a distance of 1 m from the sound source. The maximum system noise level is measured with a shortened input and must not exceed 10 dB (A-weighted) at a distance of 1 m.

With regard to the acoustical characteristics of the listening room, there are requirements for early reflections and the reverberant field. For early reflections that arrive at the main listening position within 15 milliseconds after the direct sound, the levels of the reflected sound should be at least -10 dB below those of the direct sound. These are spectral levels for the frequency range from 1 kHz to 8 kHz. Stricter requirements were found by Olive and Toole [94] who showed that the level of reflected sound must be below -15 dB for reflections to be inaudible relative to the direct sound.

The requirement for the reverberation field that arrives after the first 15 milliseconds is that it is diffuse in nature in order to avoid effects such as flutter echoes that would interfere with the listening evaluation of the program material. The reverberation time T_{60} of the listening room is defined as the time required for the direct sound from the source to decay to a level of -60 dB. The reverberation time

is a function of frequency and is typically reported for frequencies between 125 Hz and 4 kHz. The EBU recommendation is for T_{60}, to be in the range of 0.2–0.4 seconds.

The calibration of the sound rendering system must be performed to a prespecified frequency-response target curve. The EBU recommendation is that the 1/3-octave mean level must lie within a band that is bounded by ±3 dB from 50 Hz to 2 kHz and is allowed to decay by 1 dB per octave from 2 kHz to 16 kHz.

The listening levels must be set using pink noise at a level of −18 dBFS. The system gain is then adjusted so that the sound pressure level at the listening position is 85–10 log(n) dB (A-weighted), where n is the number of loudspeaker channels in the playback system. Level differences among channels must not exceed 1 dB.

The room background noise must not exceed the curve for noise rating (NR) of 10. Noise rating curves were developed by the International Standards Organization to specify the acceptable noise levels in indoor environments. At 500 Hz, the NR-10 allowable noise level is 15 dB SPL (A-weighted).

5.3. Testing methodology

The prevailing notion among psychophysicists is that the most reliable subjective evaluation results come from trained listeners. Although it was earlier thought by some that better statistics are obtained by randomly selecting listeners, the literature has shown that this is not the case. Listeners can be trained very effectively and the evaluations performed after training are more statistically meaningful [21, 81, 150]. Furthermore, it is much more efficient to process data from a smaller group of trained listeners that will give the same (or better) statistical confidence in the results.

A systematic study of listeners' ability to discriminate was performed by the Communications Research Center [43]. The results of that study showed that a triple-stimulus method in which a listener is allowed to select from three conditions was the most effective. Condition A is called the "uncoded reference" and conditions "B" and "C" are the hidden reference and coded versions randomly ordered so that the listener does not know which among "B" and "C" is which.

The evaluation procedure typically involves an initial training phase and a subsequent blind rating phase. During the training phase, listeners are given ample opportunity to explore and identify the possible types of artifacts that can arise in audio coding. This process can take several hours and it includes examples with specific coding artifacts that will later be included in the blind testing phase. The training phase presents listeners with a similar sequence of "A," "B," and "C" stimuli like what they will later experience in the formal evaluation phase. In this sequence, "A" is always the reference sound and "B" is always the coded version. This is done in order to provide listeners with an immediate and direct comparison. In the training phase, "B" will be chosen randomly to be either the coded version or the hidden reference.

The blind rating phase follows the training phase. Here the goal is to have listeners delivering numerical scores under double-blind conditions that prevent both the person administering the trials and the listeners from knowing the ordering of sounds and the placement of the hidden references in each trial. It is important during this phase to provide the listeners with the ability to freely switch among the three conditions with a simple interface that does not introduce any bias or artifacts due to switching. There are no time limits on making a decision and the listeners can repeat the samples as many times as they wish.

The rating scale most commonly used is the CCIR 5-grade impairment scale [5] that breaks down the scores into 5 categories: 5 "imperceptible"; 4 "perceptible but not annoying"; 3 "slightly annoying"; 2 "annoying"; 1 "very annoying." During the training phase, the listeners are explicitly exposed to all of these conditions that are described in detail. Listeners are also instructed to provide continuous decimal scores ranging between these categories based on their perception of the severity of the artifacts.

5.4. Data analysis after subjective listening tests

5.4.1. Mean

A measure of the central or the representative overall value is one of several items of interest in a set of observations. Although values such as the median, the mode, and the midpoint of the range of values are occasionally considered, the most widely used measure is the average, which is also known as the arithmetic mean, or simply the mean. The mean value can be obtained by dividing the total value of all observations by the number of observations as

$$\bar{y} = \frac{\sum_i Y_i}{n} = \frac{Y.}{n}, \tag{5.2}$$

where $\bar{y}$ represents the mean value of variable y, and n is the total number of observations in the sample, and $Y.$ represents the sum of the observations. Some sample data of a population is listed in Table 5.1. The mean of the data in this table is

$$\bar{y} = \frac{Y.}{n} = \frac{1795}{4} = 448.75, \tag{5.3}$$

and is an estimation of the unknown mean of the population μ.

5.4.2. Variance

When data are obtained via experiments, we are continually confronted with variability among observations. Among various available measures of this variability, one of the primary interest in statistical analysis is known as the *variance*. The

Table 5.1. A sample data in an experiment. (Sum = 1795 = $\sum_i Y_i = Y.$)

Number	Data
1	$415 = Y_1$
2	$485 = Y_2$
3	$435 = Y_3$
4	$460 = Y_4$

Table 5.2. Variance calculation using deviations from the mean. ($Y. = 1795$, $\sum_i Y_i - \bar{y} = 0.00$, $\sum_i Y_i - \bar{y}^2 = 2768.7500 = \sum_i y_i^2$, $\bar{y} = Y./n = 1795/4 = 448.75$, $s^2 = \sum_i Y_i - \bar{y}^2/n - 1 = \sum y_i^2/n - 1 = 2768.7500/3 = 922.92$.)

Y_i	$Y_i - \bar{y}$	$(Y_i - \bar{y})^2 = y_i^2$
415	−33.75	1139.0625
485	36.25	1314.0625
435	−13.75	189.0625
460	11.25	126.5625

variance in a sample or a group of observations is equal to the sum of the squares of mean-removed values divided by the number of observations minus one. It has been shown that the variance calculated in this way is an unbiased estimation, while the division by the total number of the observation results in a biased estimate of the population variance. The formula that calculates the variance in any sample set of data is given by

$$s^2 = \frac{\sum_i Y_i - \bar{y}^2}{n-1}, \tag{5.4}$$

where s^2 represents the variance of the observations in the sample and n is the number of observations. The value s^2 is an estimate of the population variance σ^2. The mean-removed sample value $Y_i - \bar{y}$ is often written as y_i. And symbol $\bar{Y}$ is frequently used to represent the mean since $\sum_i y_i = 0$.

Based on sample values shown in Table 5.1, Table 5.2 lists operations involved in calculating the variance using (5.4). The estimate of the population variance σ^2 is equal to 922.92. The calculation of the variance using (5.4) is relatively easy if only a few observations are involved. However, if the experiment contains a large number of observations, this calculation is quite cumbersome. Equation (5.4) can be simplified by

$$\frac{\sum_i (Y_i - \bar{y})^2}{n-1} = \frac{\sum_i Y_i{}^2 - Y.^2/n}{n-1}. \tag{5.5}$$

The right-hand side of the equation is much easier to be used with a large number of observations. By using (5.5), we obtain the same value for the variance of

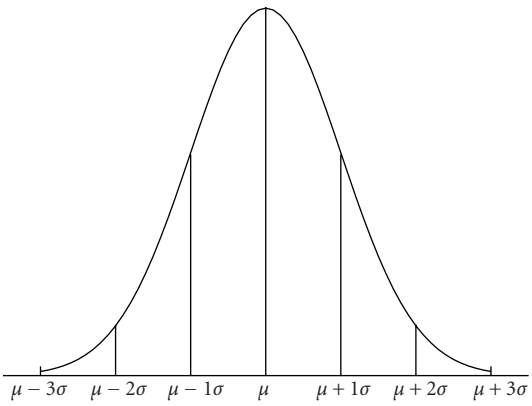


FIGURE 5.1. Areas of the normal curve.

observations as calculated in Table 5.2:

$$s^2 = \frac{\sum_i Y_i^2 - Y_\cdot^2/n}{n-1} l = \frac{808,275 - (1795)^2/4}{4-1} = \frac{808,275.00 - 805,506.25}{3} = \frac{2768.75}{3} = 922.92. \tag{5.6}$$

5.4.3. Standard deviation

From the notation of s^2 or σ^2, we know that the variance is the square value of another variable, which is referred to as the *standard deviation*. The standard deviation is simply the square root of the variance, and can be expressed as

$$s = \sqrt{\frac{\sum_i Y_i^2 - Y_\cdot^2/n}{n-1}}, \tag{5.7}$$

where s represents the standard deviation of a sample from the population and is an estimation of parameter σ. For the example under discussion, we have

$$s = \sqrt{2464.58} = 49.64. \tag{5.8}$$

If the data of interest follow a normal distribution, then the probability function of population observations should have the shape as shown in Figure 5.1. In other words, if one were able to plot the values of all individuals in a population, their frequency distribution would have the shape of a bell. This means that a larger portion of the observations would have values close to mean; and the number of observations would continuously decrease as their values get further away from the mean value. Statistical theory tells us that 68.26% of the variates or observations would have values within $\mu \pm 1\sigma$, 95.46% of the variates or observations would have values within $\mu \pm 2\sigma$, and 99.73% of the variates or observations would

have values within $\mu \pm 3\sigma$. This can be stated in a different way. That is, the area under the curve between $\mu \pm 0.674\sigma$ would include 50% of the observations, between $\mu \pm 1.965\sigma$ would include 95% of the observations, and between $\mu \pm 2.576\sigma$ would include 99% of the observations. Supposing that the estimate of the population's mean and standard deviation equals 448.75 and 49.64, respectively ($\mu = 448.75$ and $\sigma = 49.64$), we would expect that 50% of the observations fall between 415.29 and 482.21, 95% of the observations fall between 351.21 and 546.29, and 99% of the observations fall between 320.88 and 576.62.

5.4.4. Standard error of the mean

When data samples of a population are available, we can calculate the mean of each sample and we would find that there is variability among these sample means just as we had variability among individual observations in a single sample. If a large number of sample means were calculated, the mean of these sample means can be used to estimate the population's parameter μ. Similar to the distribution of the original population, the mean of samples would also be distributed in a normal curve. However, the variation among the means would be smaller than σ_y because the mean of a sample deviates less from the overall mean than some members of the sample. The variance among a group of sample means can be calculated by

$$s_{\bar{y}}{}^2 = \frac{\sum_{i=1}^{k} \left(\bar{y}_i - \bar{y}_.\right)^2}{k-1}, \tag{5.9}$$

where $s_{\bar{y}}{}^2$ represents the variance of a group of sample means, $\bar{y}_i$ represents an individual sample mean, $\bar{y}_.$ represents the mean of all sample means, and k is the number of means included. The value of $s_{\bar{y}}{}^2$ can be used as an estimation of $\sigma_{\bar{y}}{}^2$.

When we have only one sample mean and wish to estimate the variance to be expected in a distribution of several means, we will have

$$\sigma_{\bar{y}}{}^2 = \frac{s^2}{n}, \tag{5.10}$$

where s^2 is the variance calculated from the observations within the sample, and n is the number of observations in this sample. The standard deviation of the mean $s_{\bar{y}}$ is the square root of the variance of the means. The standard deviation of a statistics value such as a mean, whether calculated as the square root of $s_{\bar{y}}{}^2$ in (5.9) or estimated as $\sigma_{\bar{y}}{}^2$ in (5.10), is usually referred to as the standard error of the statistics (standard error of the mean in our case). The term standard deviation provides information on the variation among individual observations. In our sample, the standard error of the mean is equal to

$$s_{\bar{y}} = \frac{s}{\sqrt{n}} = \frac{28.39}{\sqrt{4}} = 14.20. \tag{5.11}$$

From the above equation, we find that the standard error of the mean is inversely proportional to the square root of the sample size. The standard error of

the mean decreases as the sample size increases. In a similar way as the standard deviation for a distribution of individual observations, the standard error of the mean serves the same purpose for the distribution of sample means. The standard error of the mean is an estimation of the $\sigma_{\bar{y}}$. And within range of $\mu \pm 1\sigma_{\bar{y}}$, there are 68.26% of a population of means and within range of $\mu \pm 1.96\sigma_{\bar{y}}$, there are 95% of a population of means. In statistics, the standard error of the mean is frequently adopted to indicate the amount of expected variation with continued sampling of a population of means.

5.4.5. Confidence interval

The confidence interval of a statistic can be developed by using the standard error. It is a range between upper and lower limits, which are referred to as *confidence limits*. The confidence interval is commonly associated with a parameter at a selected level of probability. Also, confidence limits are functions of a t value for a given level of probability and the standard error of the statistics value.

Table 5.3 lists some t values. These values are derived from the t distribution, which was originally developed by William S. Gossett (Student, 1908) and later modified by R. A. Fisher (1926).[1] Value t is defined as

$$t = \frac{\bar{y} - \mu}{s_{\bar{y}}}, \tag{5.12}$$

where $\bar{y}$, $s_{\bar{y}}$, and μ are the sample mean, the standard deviation, and the population's parameter, respectively. The t distribution has a symmetric curve and becomes closer to the normal curve in form as the degrees of freedom increase. Normal distribution produces the same value as t distribution when the degree of freedom is equal to ∞.

The t distribution can be used as a utility in many statistical procedures. Its application with regard to statistics other than the mean, and their standard errors, can be found in [104].

Supposing that we wish to find the 95% confidence interval about the mean of a set of observations, we need to first calculate the confidence limits by the following equations:

$$\begin{aligned} \text{Lower confidence limit} &= -t_{0.05}s_{\bar{y}}, \\ \text{Upper confidence limit} &= +t_{0.05}s_{\bar{y}}, \end{aligned} \tag{5.13}$$

where values of t can be found in the table of the t distribution, that is, Table 5.3. If the number of observations in the sample is n, then value $T_{0.05}$ can be found under the column headed 0.05 and in the $(n-1)$th row. Then the confidence

[1] W. S. Gossett (1876–1937) was a brewer and statistician who published under the name of Student. R. A. Fisher (1890–1962) made outstanding contributions in many areas of statistical theory and application and was considered as one of the pioneers in the field of statistics.

Table 5.3. Distribution of t: two-tailed tests.

	Probability of a larger value of t, sign is ignored								
df	0.500	0.400	0.300	0.200	0.100	0.050	0.020	0.010	0.001
1	1.000	1.376	1.963	3.078	6.314	12.706	31.821	63.657	636.619
2	0.816	1.061	1.386	1.886	2.920	4.303	6.965	9.925	31.598
3	0.765	0.978	1.250	1.638	2.353	3.182	3.541	5.841	12.941
4	0.741	0.941	1.190	1.533	2.132	2.776	3.747	4.604	8.610
5	0.727	0.920	1.156	1.476	2.015	2.571	3.365	4.032	6.859
6	0.718	0.906	1.134	1.440	1.943	2.447	3.143	3.707	5.959
7	0.711	0.896	1.119	1.415	1.895	2.365	2.998	3.499	5.405
8	0.706	0.889	1.108	1.397	1.860	2.306	2.896	3.355	5.041
9	0.703	0.883	1.100	1.383	1.833	2.262	2.821	3.250	4.781
10	0.700	0.879	1.093	1.372	1.812	2.228	2.764	3.169	4.587
11	0.697	0.876	1.088	1.363	1.796	2.201	2.718	3.106	4.437
12	0.695	0.873	1.083	1.356	1.782	2.179	2.681	3.055	4.318
13	0.694	0.870	1.079	1.350	1.771	2.160	2.650	3.012	4.221
14	0.692	0.868	1.076	1.345	1.761	2.145	2.624	2.977	4.140
15	0.691	0.866	1.074	1.341	1.753	2.131	2.602	2.947	4.073
16	0.690	0.865	1.071	1.337	1.746	2.120	2.583	2.921	4.015
17	0.689	0.863	1.069	1.333	1.740	2.110	2.567	2.898	3.965
18	0.688	0.862	1.067	1.330	1.734	2.101	2.552	2.878	3.922
19	0.688	0.861	1.066	1.328	1.729	2.093	2.539	2.861	3.883
20	0.687	0.860	1.064	1.325	1.725	2.086	2.528	2.845	3.850
21	0.686	0.859	1.063	1.323	1.721	2.080	2.518	2.831	3.819
22	0.686	0.858	1.061	1.321	1.717	2.074	2.408	2.819	3.792
23	0.685	0.858	1.060	1.319	1.714	2.069	2.500	2.807	3.767
24	0.685	0.857	1.059	1.318	1.711	2.064	2.492	2.797	3.745
25	0.684	0.856	1.058	1.316	1.708	2.060	2.485	2.787	3.725
26	0.684	0.856	1.058	1.315	1.706	2.056	2.479	2.779	3.707
27	0.684	0.855	1.057	1.314	1.703	2.052	2.473	2.771	3.690
28	0.683	0.855	1.056	1.313	1.701	2.048	2.467	2.763	3.674
29	0.683	0.854	1.055	1.311	1.699	2.045	2.462	2.756	3.659
30	0.683	0.854	1.055	1.310	1.697	2.042	2.457	2.750	3.646

interval about the population mean is

$$\bar{y} - t_{0.05} s_{\bar{y}} \leq \mu \leq \bar{y} + t_{0.05} s_{\bar{y}}. \tag{5.14}$$

If there are only four observations, then the 95% confidence interval for the mean would be from

$$448.75 - (3.182) \times (24.82) \quad \text{to} \quad 448.75 + (3.182) \times (24.82), \tag{5.15}$$

or from

$$369.77 \quad \text{to} \quad 527.73. \tag{5.16}$$

With the above calculation, we can claim that we feel 95% confident that the population mean μ lies between 369.77 and 527.73. The interval is commonly written as

$$\bar{y} \pm t_{0.05} s_{\bar{y}}. \tag{5.17}$$

In our example, this would be

$$448.75 \pm 78.98. \tag{5.18}$$

6 MPEG-4 audio coding tools

6.1. Introduction to MPEG-4 audio

MPEG-4 audio [58] is currently one of the most prevalent audio coding standards. It combines many different types of audio coding into one integrated system. The following integration has been provided in MPEG-4 audio:

(1) low bit rate with high quality compression,
(2) synthetic with natural sound processing,
(3) speech with audio coding,
(4) single-channel with multichannel audio configuration,
(5) traditional with interactive and virtual-reality content analysis.

By putting those sophisticated coding tools and the novel flexible framework for audio processing into standard, researchers and developers of the MPEG-4 audio have provided the world with new technologies for digital audio management.

MPEG-4 audio distinguishes itself from any other audio standards previously created by ISO/IEC or other groups by the fact that it standardized audio tools for every application that requires the use of advanced sound compression, synthesis, manipulation, or playback instead of any single application. As a single set of tools, MPEG-4 audio has the nature of interoperability. Different systems can easily share data and development tools if the same MPEG-4 audio toolset has been used. For example, a real-time voice communication system and a voicemail indexing and retrieval system can share data and development tools without any difficulties if the same MPEG-4 speech coding toolset is used in both systems.

MPEG-4 audio includes several subparts, each of which specifies a state-of-the-art coding tools in a certain domain. However, MPEG-4 audio is not a simple summation of its subparts. By integrating audio tools with the rest of MPEG-4 standards, MPEG-4 audio enables new opportunities in object-based audio coding, interactive presentation, dynamic soundtracks, and other sorts of new media.

MPEG-4 audio provides many different concepts than other previous MPEG audio standards. A brief overview that highlights the difference between MPEG-4 audio and MPEG-1, and MPEG-2 audio and MPEG-AAC is listed below.

(1) *Low bit rate coding.* Audio standards defined in MPEG-1 and MPEG-2 were focused on developing the perceptually lossless or near-lossless coding of high-quality audio without making much constrain on the

bit rate budget. MPEG-4 provides new and the state-of-the-art tools for low-bit-rate audio and speech coding. It has tested and standardized tools that have the ability to achieve the same high-quality sound at a much lower bit rate that is suitable for transmitting audio via Internet, digital radio, or other bandwidth-limited delivery.

(2) *Coarse-grain and fine-grain scalability.* In order to meet the requirements of different channels and applications, MPEG-4 provides bit rate and bandwidth scalability options within a single bitstream. Both coarse-grain scalability and fine-grain scalability are available in speech and audio coding.

(3) *Robustness to channel errors.* Error resilient tools are added for some MPEG-4 audio/speech codecs. With these tools, the error-resilient bitstream can be reconstructed with improved perceived quality when it is corrupted during transmission. Error protection tools are provided in MPEG-4 to work together with error resilient tools to improve the bitstream's error robustness. These tools can be used in various speech and audio coding systems operating over error-prone channels such as wireless network or digital audio broadcasting.

(4) *No standard for transport.* In all of the MPEG-4 tools for audio and visual coding, constructing a sequence of *access units* that contain the compressed data is the last step in the coding standard. How to convert the individually coded objects into a bitstream that contains a number of multiplexed substreams is specified in MPEG-4 systems.

Because MPEG-4 audio targets a broad range of applications, whose delivery requirements are too wide to be easily characterized with a single solution, it is hard to standardize a single *interface* (the delivery multimedia interface format, or DMIF) that describes the transport layer capabilities and how the transport, multiplex, and demultiplex functions communicate and work with each other in the encoders and decoders. Therefore, MPEG-4 audio does not set a standard mechanism for the transport of its stream over a channel. The MPEG-4 system bitstream specification together with the use of DMIF allows MPEG-4 transmission functions to work in a much more sophisticated way than previous MPEG standards. Besides the sophisticated transport functionality, a single MPEG-4 audio stream is provided as a private, not normative transport mechanism for applications with less delivery requirements, such as object-based coding and some other functions in MPEG-4 systems.

(5) *Object-based coding standard with multiple tools.* In previous MPEG audio standards, a single toolset is provided to be used for various applications when different configurations are set. Whereas in MPEG-4, several toolsets are defined, each of which targets a different function. Although these toolsets have no particular relationship to each other, they can be used together for various applications. How to configure these toolsets is specified in different MPEG-4 audio profiles.

Another advantage of using MPEG-4 audio over previous MPEG audio standard is that the concept of *soundtrack* is much more flexible in MPEG-4. This is because MPEG-4 can transmit more than one *audio object* when using multiple tools, while previous MPEG standards only transmit one single piece of content. In MPEG-4, when multiple tools are used together, MPEG-4 *audio composition* system will be adopted to create a single soundtrack from the audio substreams. During this process, several components, such as the speaker configuration, the user interaction, the terminal capability, and so forth need to be adjusted to produce the final soundtrack.

(6) *Synthetic sound capability.* When natural sound is being coded, existing signals are first compressed by a server, then transmitted in the channel and finally decompressed at the receiver. This kind of sound coding called natural speech and audio coding is the main or only target of many existing standards for sound compression. Besides the common natural sound coding, MPEG-4 is capable in providing a very-low-bit-rate but still very high-quality coding by standardizing a novel synthetic sound description, where synthetic speech and synthetic music, are transmitted and then *synthesized* into sound at the receiver.

Same as the previous MPEG standards, MPEG-4 does not standardize the encoding algorithms. Which means as long as the bitstream is compliant with the MPEG-4 standard or is recognizable and decodable by the standard decoder, end-users can make their own decisions in creating bitstreams with whatever method they think is the best. In fact, how to automatically convert natural sound into synthetic or multiobject descriptions is still an open problem. The most common current solutions to these problems still require human interaction in hand authoring the content stream, which is a similar process as current schemes for MIDI-based and multichannel mixdown authoring of soundtracks.

The rest of this chapter will give more detailed overview of MPEG-4 audio.

6.2. MPEG-4 audio tools

The MPEG-4 audio provides two types of tools, the *natural* sound coding tools and the audio *synthesis* tools. The *natural* sound coding can be used in encoding, transmitting, and decoding human speech for telephony, personal communication, and surveillance applications. The audio *synthesis* tools are used for very-low-bit-rate description, transmission, and synthesis of synthetic speech, music, and other sounds.

(1) *Natural sound coding tools.*

(i) *Speech* tools are used for the encoding, transmission, and decoding of human speech.

(ii) *Audio* tools are used for the encoding, transmission, and decoding of recorded music, environmental sound, and other audio soundtracks.

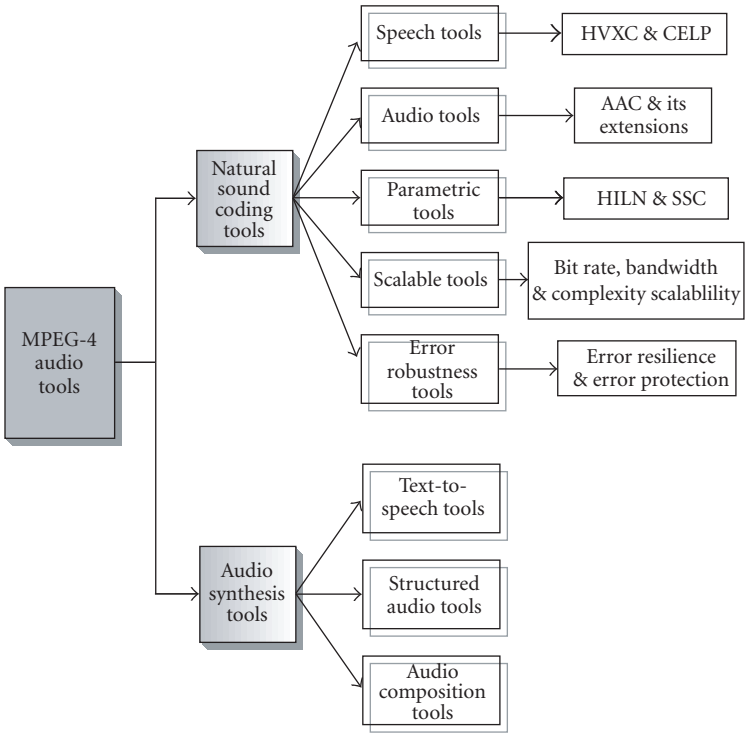


Figure 6.1. MPEG-4 audio tools.

(iii) *Parametric* tools are used for extremely low-bit-rate coding of general audio signals and the low-cost speech and pitch modification.

(iv) *Scalability* tools are used for creating bitstreams that can be transmitted, without recording, at several different bit rates.

(v) *Error robustness* tools are used at both encoding and decoding processes to reconstruct the sound with better perceptual quality when bitstreams are transmitted in error-prone networks.

(2) *Audio synthesis tools*.

(i) *Text-to-speech* tools are used as an interface of the text-to-speech synthesis systems and provide very-low-bit-rate operation and built-in connection with facial animation in low-bit-rate video-teleconferencing applications.

(ii) *Structured audio* tools are used to provide general description of synthetic sound and the creation of synthetic sound in the decoding terminal.

(iii) *Audio composition* tools are used for object-based coding, interactive functionality, and audiovisual synchronization.

In the following subsections, we will describe each type of these tools in more detail.

6.2.1. MPEG-4 natural sound coding tools

6.2.1.1. Speech coding tools

Natural speech sound coded at bit rates ranging between 2 and 24 kbps is covered by the MPEG-4 speech coding toolset. For variable bit rate coding, coding at even lower bit rate, such as average bit rate of 1.2 kbps, is also supported. The target applications of the MPEG-4 speech coder range from mobile and satellite communications, to internet telephony, to packaged media and speech databases. In other words, MPEG-4 speech coder is capable of meeting a wide range of requirements in bit rates, sound quality, as well as functionality. Two basic techniques are used in speech coding: one is called HVXC (harmonic vector excitation coding) and the other is called CELP (code-excited linear prediction).

MPEG-4 HVXC is a parametric speech coding algorithm and allows very-low bit-rate compression. It operates at fixed bit rates between 2.0 kbps and 4.0 kbps, using a bit-rate scalable scheme. In variable bit-rate modes, it also supports even lower bit rate, typically 1.2–1.7 kbps. At 8 kHz sampling rate and 100–3800 Hz band, HVXC can provide communications quality to near-toll-quality[1] speech. In addition, HVXC has the powerful functionality to independently change the speed and/or the pitch during decoding, which can be used to fast access speech databases.

MPEG-4 CELP coding technique is a well-known coding algorithm with new functionality. Conventionally, working in a single bit rate, CELP coders compress speech signals and are optimized for specific applications. Other than supporting compression as one of its functionalities, MPEG-4 makes the basic CELP coder contribute in multiple applications. CELP becomes a building block in MPEG-4 bit rate and bandwidth scalable tool and enables the MPEG-4 audio capability to generate bitstreams at any arbitrary bit rate. Two sampling rates, 8 and 16 kHz, are supported by MPEG-4 CELP. The associated bandwidths for 8 kHz sampling rate are 100–3800 Hz and 50–7000 Hz for 16 kHz sampling rate.

The silence compression tool in CELP encoder reduces the compressed stream to an even lower average bit rate. A voice activity detector is used in the encoder so that regions with normal speech activity and those with silence or background noise can be distinguished. The CELP coding, same as MPEG-4 version 1, is used for normal speech activity; a silence insertion descriptor (SID) is transmitted at a lower bit rate for other periods. At the decoder side, the SID signals the comfort noise generator (CNG) to generate a comfortable background noise rather than just insert a silent period. The amplitude and spectral shape of the comfortable noise is determined by the energy and LPC parameters. Since these parameters can be sent as an optional part of the SID, they can be updated from time to time as required.

[1]Toll quality is the speech quality of public switched telephone networks, and communication quality is that of 2G phone networks.

The MPEG-4 speech coding toolset has been proved to work efficiently while achieving good performance by MPEG committee by conducting extensive verification tests in practical listening conditions.

6.2.1.2. General audio coding tools

MPEG-4 general audio coding tools offer a wide bit rate range of coding capability. Bit rate from as low as 6 kbps up to several hundred kbps per audio channel for mono-channel, two-channel, and multichannel signals, are covered in MPEG-4. MPEG-2 AAC [60], which is able to provide high-quality compression, was adopted in MPEG-4 general audio coding standard as base coding techniques. Several new coding modules and improvements are added on top of MPEG-2 AAC to become the so-called MPEG-4 AAC standard. The MPEG-4 general audio (GA) coder uses several advanced coding blocks, including a perceptual filter bank, a sophisticated masking model, noise-shaping techniques, channel coupling, noiseless coding, and bit-allocation, to achieve the maximum compression within the constraints of providing the highest possible quality. The MPEG-4 general audio coding utilized the state-of-the-art psychoacoustic coding standards developed by MPEG, which is one of the most important factors that makes the AAC achieve its goal of low bit rate and high performance.

According to the MPEG standard verification tests, the MPEG-4 AAC toolset is capable of providing "indistinguishable quality[2]" when working at 64 kbit/s/channel or even higher bit rate. For lower bit rates from 6 kbps up to 64 kbps per channel, a modification to the AAC quantization module is developed in MPEG-4. This tool, called TwinVQ, enables content authors to generate high quality complex sound at extremely low bit rate range.

In order to develop a high quality general audio coding algorithm that is suitable for real-time bidirectional communication, based on MPEG-2/4 advanced audio coding (AAC), MPEG-4 version 2 specifies a low-delay audio coder. Unlike speech coding schemes, the low-delay audio coder can handle general audio signal types, including music, with high performance and low delay. It supports input signals with 48 kHz sampling rate and adopts a frame length of 512 or 480 samples, instead of 1024 or 960 samples used in the standard MPEG-2/4 AAC. The size of the window used in the analysis and synthesis filter bank is reduced by a factor of 2. In addition, no block switching is enabled in order to avoid the "look-ahead" delay due to the block switching decision. In case of transient signals, window shape switching is provided so that pre-echo artifacts can be reduced. As for the window type, a sine window is used for nontransient parts of the signal, while a so-called low overlap window is used for transient signals. Bit reservoir is rarely used during encoding in order to reach the desired target delay. In some extreme cases, no bit reservoir is used at all. It has been shown by the verification tests that only a moderate degradation in compression performance occurs due to reducing the coding delay.

[2]Indistinguishable quality, which is defined by the European Broadcasting Union, refers to the sound quality that is perceptually the same as that of original input signals.

The BSAC tool is built on top of original AAC coding tools with a new noiseless coding module. BSAC can achieve scalability in steps of 1 kbps per audio channel, that is, 2 kbps steps for a stereo signal. It generates bitstream with one base layer and many small enhancement layer. The base layer contains the audio data for the first layer and other side information. The enhancement layers contain audio data to improve the performance and other specific side information for the corresponding layer. The fine-grain scalability is accomplished by a bit-slicing scheme when doing the data quantization and transmission. The spectral data is first grouped into frequency bands after the quantization. Data in each group is represented in binary form. Then bits are processed in slices according to their significance one group after the other. In other words, the most significant bits (MSB) of all quantized values in a group are processed first, followed by the corresponding next significant bits, and so on. All bit-slices are then compressed by an arithmetic coding scheme to further remove the redundancy. Different arithmetic coding models are provided to cope with the different statistics of the bit-slices. Bandwidth scalability is achieved by assigning bit-slices of the different frequency bands to the enhancement layer in a special way, so that with an increasing number of enhancement layers, the decoder can reconstruct the audio signal with more refinement bits and in higher frequency bands.

The scalability of the fine-grain-scalable tool has been shown to perform well over a wide range of rates. At the highest rate, it can perform almost the same as the AAC main profile at the same rate. When working in a lower bit rate mode, the scalability function requires a moderate overhead relative to AAC main profile operating at the same rate.

MPEG-4 high efficiency AAC (HEAAC) is the newest extension of AAC and is targeting to compress music with high quality at bit rate lower than 64 kbps for each channel. It continues to use the traditional AAC to code input signals low frequency components, while adopting the spectral band replication (SBR) technology to take care of the high-frequency components. The HEAAC significantly reduced the required bit rate while maintaining similar perceptual quality of the reconstructed sound.

6.2.1.3. Parametric audio coding

MPEG-4 parametric audio coding tool has two distinguished features. One is its high compression ratio and the other is its convenient postprocessing capability. The parametric audio encoder is capable of compressing general audio signals with extremely low bit rate, while the parametric audio decoder can perform pitch and playback speed modification without the need for an expensive processing unit. Combined with the speech and audio coding tools of MPEG-4 version 1, MPEG-4 version 2 parametric audio coding provides improved overall coding efficiency for applications of object-based coding and allows selection and/or switching between different coding techniques.

The harmonic and individual lines plus noise (HILN) technique is the first tool introduced in the MPEG-4 parametric audio coding. It uses a parametric

representation of the audio signal and can code general audio signals at bit rates of 4 kbps and above. Separating or decomposing the input signal into different audio objects is the basic idea of the HILN technique. After decomposition, the coder can describe each audio object by using source models and then representing each object by model parameters. Three kinds of object models, that is, sinusoids, harmonic tones, and noise, are utilized in the HILN coder.

From the knowledge about the speech coding, where specialized source models based on the speech generation process in the human vocal tract are applied, we know that source models have more advantages than the waveform coding method in very low bit rate coding schemes. The most challenging part in the parametric audio codec design is to find a good set of source models. HILN introduces an advanced source models and successfully handles the parametric coding of the general sound signals.

Similar to the general audio coding tool, the parametric audio coding tool also adopts the perception model in order to select and transmit the most important parameters so that sound of the best perceptual quality can be reconstructed with minimum amount of bit resources. Similar to the general audio coding tool, in HILN, the output from the psychoacoustic model is used to control the quantization of the frequency and amplitude parameters. Then, the LPC model known from speech coding is used to describe the harmonic tone and the spectral envelope of the noise. Correlation between parameters of one frame and between consecutive frames can be reduced by parameter prediction. The quantized parameters are finally compressed by an entropy coder and then multiplexed together to form a single bitstream. Since the signal is described in terms of frequency and amplitude parameters in HILN, the speed and pitch change functionality can be performed by simple parameter modification in the decoder.

The MPEG-4 parametric speech coder (HVXC) can be combined with the HILN parametric audio coder to form an integrated parametric coder that supports a wider range of signals, bit rates, and speed and pitch change. With a preprocessing module that performs speech/music classification at the beginning of the encoding, it is possible to automatically select the HVXC for speech signals and the HILN for music signals. MPEG-4 version 2 standard describes this automatic HVXC/HILN switching and the corresponding classification tool in its informative part.

The HILN coding has been shown to achieve comparable performance with other MPEG-4 coding technology operating at similar bit rates while providing the additional capability of independent audio signal pitch and speed change during decoding. It has also been demonstrated that the scalable HILN coder provides comparable audio quality to that of a fixed-rate HILN coder at the same bit rate.

A newer tool in MPEG-4 parametric audio coding toolset is called sinusoidal coding (SSC). For a mono-input sound, the MPEG-4 SSC decomposes the signal into three objects: transients, sinusoids, and noise. Transients represent signals that dynamically change in the time domain, sinusoids represent the components with deterministic frequency characteristics, and noise represents the rest signals that are localized neither in temporal nor in spectral domain. If the input sound is

stereo sound, then SSC uses another set of parameters to capture and reconstruct the stereo sound image. Similar to HILN tool, SSC can also compress the complex sound with extremely low bit rate and is capable of performing pitch and temporal scaling at low cost. Moreover, its stereo image reconstruction capability makes it an even better tool to handle and generate more appealing sound than the HILN parametric coding tool. In fact, the stereo parameters can be combined with other audio/speech coding method as an stereo enhancement tool. By doing this, the core codec is only responsible for encoding and decoding the monophonic representation of the original stereo sound and the SSC stereo parameters can be used to enhance the monophonic signal and generate a copy of perceptually similar stereo image at the decoder end.

6.2.1.4. Audio scalability tools

Many bitstreams in MPEG-4 have scalability functionality in one manner or another. We will discuss several types of scalability in the standard below.

Scalability in bit rate means that a bitstream contains sub-bitstreams corresponding to lower bit rate encoding and the total compressed stream is a combination of basic bitstream and several additional bitstreams. The basic bitstream is able to reconstruct the input audio into reasonable quality signals. Each additional bitstream refines the output signal so that a better quality sound can be recovered. Making a scalable bitstream involves either reorganizing the nonscalable bitstream or even redesigning the whole coding process. In MPEG-4, scalability is available for each natural audio coding scheme, as well as combinations of different natural audio coding schemes.

As a particular case of bit rate scalability, bandwidth scalability means that part of a bitstream representing only partial or lower frequency spectrum can be transmitted and reconstructed independently in the receiver. The CELP speech coder provides two-layer bandwidth scalability, where the base layer presents the narrow-band speech signals and can be extended to wide-band speech signals by the enhancement layer. MPEG-4 general audio coding tool is capable of offering a much more flexible bandwidth control and providing more layers and finer grain of bandwidth scalability. This is because the general audio codec is a transform-based coder and all major signal processing tools in the encoder are performed in frequency domain.

Complexity scalability at the encoder side means that by working in different complexities, the encoder can generate valid and meaningful bitstreams corresponding to different output audio qualities. For example, the wideband CELP coder offers two modules. One is the high quality module and the other is the low complexity excitation module. Content listeners have choices to select either significant lower encoder complexity or optimized coding quality.

Complexity scalability at the decoder side means that a given bitstream can be decoded by decoders of different levels of complexity corresponding to different quality of the reconstructed audio signals. One of the subtypes of the decoder complexity scalability worth to be mentioned is called *graceful degradation*. When

operating in this mode, the decoder can dynamically monitor the available resources, and it switches to lower complexity decoder when resources are limited. For example, in MPEg-4 structured audio, the content author can provide two or more different algorithms on how to synthesize keyboard sounds. At the decoder side, depending on available resources, the content itself can decide which one to use to perform the synthesis.

6.2.1.5. Error robustness tools

When transmitting bitstreams through error-prone channels, the error robustness tools are extremely helpful in improving the final performance. There are two types of error robustness tools. One is codec-specific error resilience tools and the other is common error protection tools.

The MPEG-4 parametric speech coding has integrated error resilient tool, that is, the error resilient (ER) HVXC. The ER tool is available for both fixed-bit-rate modes (2.0–4.0 kbps) and variable-bit-rate modes (< 2.0 kbps, < 4.0 kbps) in either a scalable or a nonscalable scheme. The MPEG-4 version 1 HVXC supports maximum variable bit rate of 2.0 kbps while version 2 ER HVXC supports maximum variable bit rate of 4.0 kbps. In the variable bit rate modes, nonspeech parts are signaled by unvoiced signals and are only allocated a smaller number of bits so that the average bit rate can be reduced. At 8 kHz sampling rate, ER HVXC is capable of producing communications quality to near-toll-quality speech in the 100–3800 Hz bandwidth. An even lower average bit rate is achievable when the coder is operating in the variable bit rate mode. Speech signal coded by variable bit rate mode at typical bit rate of 1.5 kbps in average and at typical bit-rate of 3.0 kbps in average has the same quality as that coded by 2.0 kbps fixed rate and 4.0 kbps fixed rate, respectively. The capability to do the pitch and speech change is available during decoding in all modes. The ER HVXC also defines error sensitivity classes to be used with the EP tool. When transmitting through error-prone channel, such as mobile communication channels, the error concealment functionality is available during decoding. Therefore, the ER HVXC speech coder has a wide application area, including mobile and satellite communications, Internet telephony, packaged media, and speech databases.

MPEG-4 AAC has integrated a set of error resilience tools to improve its error robustness. These error resilient tools reduce the perceived quality degradation of the reconstructed audio signal that is caused by corrupted bits in the bitstream. The following three tools are provided in AAC to improve its error robustness in several areas:

(1) virtual codebook tool (VCB11),
(2) reversible variable length coding tool (RVLC),
(3) huffman codeword reordering tool (HCR).

Improved error robustness capabilities for all MPEG-4 audio coding tools are achieved by the payload syntax of the error resilient bitstream, which also enables advanced channel coding techniques so that special needs of different coding tools

can be met. MPEG-4 version 2 requires that each object type have the error resilient bitstream payload syntax.

All MPEG-4 audio version 2 audio objects are provided with the error protection tool (EP tool), which has a flexible configuration and can be applied to a wide range of channel error conditions.

The distinguished features of the EP tool are listed below.

(1) A set of error correcting/detecting codes, with wide- and small-step scalability in performance and in redundancy, is provided.
(2) A generic and bandwidth-efficient error protection framework that covers both fixed-length and variable-length frame streams is provided.
(3) An unequal error protection (UEP) configuration control with low overhead is provided.

MPEG-4 audio version 2 coding algorithms classify each bitstream field according to its error sensitivity. Then the bitstream is divided into several classes, each of which is separately protected by the EP tool, such that more error-sensitive parts are protected more strongly.

6.2.2. MPEG-4 audio synthesis tools

6.2.2.1. Text-to-speech interface

Text-to-speech (TTS) starts to play an important role in various multimedia application areas and is becoming a rather common media type. For example, a multimedia content with narration can be easily created by using TTS functionality without recording natural speech sound. However, without MPEG-4 TTS functionality, there was no way for a multimedia content provider to easily give instructions to an unknown TTS system. MPEG-4 standardizes a single common TTS interface, which enables the speech information to be transmitted in the international phonetic alphabet (IPA), or in a written form of any language.

The MPEG-4 TTS package, called hybrid/multilevel scalable TTS interface, is not a conventional TTS system. It extends the conventional TTS system by adding the prosodic (e.g., time-dependent variation of pitch) information on top of the input text. With the help of the prosodic information, which is extracted from the natural speech, the decoder can generate much higher-quality and perceptually less artificial synthetic speech. Moreover, in MPEG-4 TTS, the interface and its bitstream format is highly scalable in terms of this additional information. For example, if part of the prosodic information, for example, some parameters, is not available at the decoder side, MPEG-4 TTS system is still able to generate these missing parameters by rule.

MPEG-4 TTS system does not specify its normative algorithms in speech synthesis and text-to-phoneme translation. However, in order to meet the goal that underlies the MPEG-4 TTS interface, a decoder should make full use of all the provided information according to the user's requirements level.

Besides an interface to text-to-speech synthesis systems, a joint coding method for phonemic information and facial animation parameters and other animation parameters are specified by MPEG-4, so that a single bitstream may be used to

control both the text-to-speech interface and the facial animation visual object decoder. Thus, MPEG-4 TTS has extended its functionalities from conventional TTS to natural speech coding and its application areas, and from simple TTS to audio presentation with TTS and motion picture dubbing with TTS.

6.2.2.2. Structured audio

MPEG-4 structured audio (the SA coder) refers to the toolset that achieves the general audio synthesis capability. It provides general description of synthetic sound and the normative creation of synthetic sound in the decoding terminal. By using these tools, high-quality stereo sound can be transmitted at bit rates from 0 kbps (no continuous cost) to 2–3 kbps for extremely complex and expressive sound.

Instead of being specified by a particular method of synthesis, the synthesis methods are described by a flexible language in SA coder. By adopting this technique, MPEG-4 structured audio not only provides a flexible encoding approach, but also ensures the compatibility during the decoding processing. This is because at the encoder side, by using this flexible language to specify the synthesis method, the available synthesis techniques are not limited to those provided during the standardization process, but can also be extended by any other future MPEG-4 SA tool users. On the other hand, since the creation of synthetic sound from structured descriptions is normative in MPEG-4, any terminal will be able to create the same sound with the SA decoder.

A set of *instrument* modules that can create audio signals under the control of a *score* is utilized to transmit the synthetic audio. An instrument refers to a small network composed of some signal-processing primitives that control how the sound is generated according to some algorithm. A single structured audio bitstream may consist of several different instruments. A score is a set of commands arranged in time. It determined at what specific time, how each instrument should make its contribution to the overall music performance.

Wavetables refer to the efficient transmission of sound samples. Its use in sampling synthesis is accomplished by interoperably working with the MIDI manufacturers association downloaded sounds level 2 (DLS-2) standard. Either by itself or in conjunction with other kinds of synthesis using the more general-purpose tools, the DLS-2 standard, which is normatively referenced by the structured audio standard, provides a simple and popular technique of wavetable synthesis in MPEG-4 structured audio soundtracks. The popular MIDI (musical instrument digital interface) control format can be adopted to replace, or in addition to, scores in SASL (structured audio score language) for controlling synthesis in order to further enable interoperability with existing content and authoring tools. By including the MIDI standards, MPEG-4 structured audio is a representation of the current technique for synthetic sound description (MIDI-based wavetable synthesis) and the future general-purpose algorithmic synthesis. This unified solution makes the resulting standard capable of solving problems not only in very-low-bit-rate coding, but also in many other applications, such as karaoke systems, video games, virtual environments, and so forth.

6.2.2.3. Audio composition tools

Same as the visual composition tools, the audio composition tools are specified in the MPEG-4 systems standard. *Audio composition* is an object-based approach, which makes use of multiple individual "audio objects" and mixes different techniques to create a single soundtrack. At the encoder side, it treats each object individually and uses different coding tool for each of them. The instructions on how to mix these multiple objects into one soundtrack are also included in the bitstream. At the decoder end, audio objects are first decoded separately after they are received, and then they are mixed together to a single soundtrack by related instructions. Finally, the composed mixed file can be played back to the listeners.

By treating each audio object individually, some more sophisticated client-side interaction can be enabled at the decoding terminal by the end user. For instance, when listeners want to focus more on the foreground speech content, they can simply mute the background music if the compound soundtrack is processed by MPEG-4 audio composition tool. On the other hand, this operation would require much more sophisticated technology or may be simply not possible if the speech and the background music were coded in one audio track.

Another advantage of using audio composition tools is that the bit rate of the final bitstream can be reduced if each audio object is encoded individually by its best encoding method. As an example, suppose a person is making a speech in a reverberant environment over background music. If we wish to transmit this sound with high fidelity, we can first represent this sound as a compound signal processing, a speech signal passing through a reverberator and then mixed with a synthetic music. The speech signal can be encoded by CELP speech coder at 16 kbps and the synthetic music can be taken care of by the SA tool at 2 kbps. The description of the reverberation and the stereo mixdown can be transmitted by a small amount of overhead (only a few hundreds of bytes as a fixed cost) in the bitstream. Therefore, the total bit rate of this bitstream is less than 20 kbps. On the other hand, if this soundtrack is treated as one single object, only a general audio codec can encode it with high quality due to its complex nature. Currently, the most state-of-art general audio codec, such as MPEG-4 HEAAC, would require 48 kbps to reconstruct this complex sound with similar perceptual quality.

In MPEG-4 systems, a tool called AudioBIFS, which is a subset of MPEG-4 binary format for scenes (BIFS), allows content authors to describe sound scenes using this object-based framework. We can perform multiple sources mixing and combining, as well as the interactive control provided for their combination. This method also allows sample-resolution control over mixing. Dynamic download of custom signal-processing routines makes it possible for the content author to exactly request a particular, normative, digital filter, reverberator, or other effects-processing routine. Finally, the description of virtual reality and other 3D sound material can be accomplished by an interface to terminal-dependent methods of 3D audio spatialization.

7 MPEG advanced audio coding

MPEG advanced audio coding (AAC) and its successors are currently the most powerful members in the MPEG family for high-quality, multichannel, digital audio compression. MPEG AAC provides a various selection of operating modes and configurations, and therefore supports a wide range of applications from low bit-rate Internet audio to multichannel broadcasting services. In this chapter, we will introduce the main features of the AAC multichannel coding system.

7.1. Introduction to advanced audio coding

As part of the MPEG-2 standard (ISO/IEC 13818-7) [60], MPEG-2 AAC was finalized in 1997. Initially the MPEG-2 AAC was called the MPEG-2 *non-backward compatible* (NBC) coding [15]. The standardization of MPEG-2 NBC started after the completion of the MPEG-2 multichannel audio standard in 1994 [59]. At that time, the MPEG audio subgroup wished to develop a new multichannel coding standard that allows an even higher quality other than the first MPEG-2 multichannel audio standard, which is backward compatible with MPEG-1. The development goal of this new multichannel audio coding algorithm is to achieve *indistinguishable quality* according to the EBU definition [11] at the bit rate of 384 kbps or lower for five full-bandwidth channel signals, without any backward compatibility constrain.

MPEG-4 AAC, which is also referred to as MPEG-4 general audio (T/F) coder [58], added several new audio tools on top of MPEG-2 AAC. Some of the new tools borrow ideas from speech coding to even lower the encoding bit rate while maintaining the similar sound quality. Others modified certain existing MPEG-2 AAC coding blocks to enable a few desirable features and provide more functionalities. With a typical bit rate of 64 kbps per channel, MPEG-4 AAC can achieve near-transparent coding for CD-quality original input sound. Even at very low bit rates down to 16 kbps, MPEG-4 AAC still exhibits excellent performance.

A scalable audio coding standard is also included in MPEG-4 audio. MPEG-4 version 1 provides large-step scalability, which integrated several existing MPEG-4 audio coding schemes to deliver a scalable audio bitstream. AAC is adopted as part of this large-step scalable audio coding architecture to provide enhancement layer bitstream. A fine-grain scalable (FGS) audio framework was developed in MPEG-4

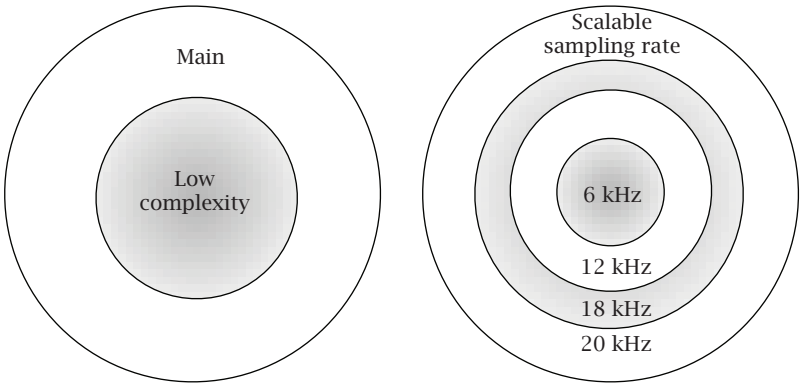


FIGURE 7.1. Three MPEG-2 AAC profiles [60].

audio version 2 [65]. The new FGS audio codec is called bit-sliced arithmetic coding (BSAC). It replaces the AAC's quantization and entropy coding module to enable the decoder to reconstruct the audio signal more and more accurately.

MPEG-4 high-efficiency AAC (HE AAC) is the next generation of AAC standard, which utilizes AAC as the core coder and extends the higher-frequency component and stereo information with the help of a set of limited parameters. Since this new algorithm is a model-based parametric extension to AAC, it further reduces the bit rate and is capable to reproduce a CD-quality stereo sound at 24 kbps, which is about 64 times smaller than the original file.

In this chapter, audio tools in all AAC-related standards including MPEG-2 AAC, MPEG-4 AAC, MPEG-4 BSAC, and MPEG-4 HEAAC will be discussed in detail.

7.2. MPEG-2 AAC

7.2.1. Overview of MPEG-2 AAC

7.2.1.1. Profiles

MPEG-2 AAC provides three profiles that can be selected by end-users according to their desired complexity and quality requirements. These three profiles are main profile, low-complexity (LC) profile, and scalable sampling rate (SSR) profile, and their relationships are shown in Figure 7.1.

(1) *Main profile*. AAC main profile, abbreviated as AAC main, provides the best audio quality at any given bit rate among all three profiles. All coding tools except the gain control are enabled in AAC main. Thus both the memory requirement and the computational complexity of this configuration are higher than the other two profiles. However, the main profile AAC decoder is more powerful and is capable to decode a bitstream generated by a low-complexity AAC encoder.

(2) *Low-complexity profile*. AAC low-complexity profile, abbreviated as AAC LC, does not use the gain control and prediction tools. Moreover, the temporal

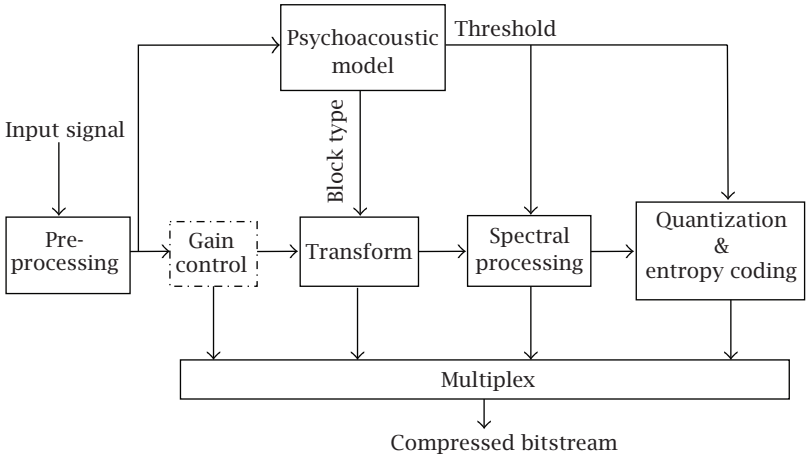


FIGURE 7.2. The block diagram of the AAC encoder.

noise shaping (TNS) filter has a lower order than that of the AAC main codec. With this configuration, the AAC codec consumes considerably lower memory and processing power requirements. However, degradation of the reconstructed sound quality is not obvious. In fact, among all three profiles, AAC LC is the most popular profile adopted in industry.

(3) *Scalable sampling rate profile*. AAC scalable sampling rate profile, abbreviated as AAC SSR, is capable to provide a frequency scalable signal. The gain control is activated when the AAC codec is operating in this profile. Except the gain control module, other audio tools are operating at the same configuration as the AAC LC profile. Therefore, the memory requirement and the computational complexity of the AAC SSR codec are also significantly lower than that of the AAC main codec.

7.2.1.2. High-level block diagrams

Figures 7.2 and 7.3 are high-level block diagrams of AAC encoder and decoder, respectively.

At the encoder side, input signals are first processed by the preprocessing module if it is necessary. Depending on the required bit rate and the sampling frequency of the input signal, the preprocessing module downsamples the original signal so that better quality can be achieved with the limited bit budget. If the input signal has out-of-range amplitude, it will also be processed in this module. If the encoder is operating in SSR profile, signals are then processed by the gain control. Otherwise, they directly go to the transform block to map the time-domain signal to frequency domain and obtain spectral data. A serial of spectral processing is followed to remove irrelevant and unimportant information. Finally, the processed spectral data is quantized and noiseless coded. Throughout the entire encoding process, the output from the psychoacoustic model, such as block type

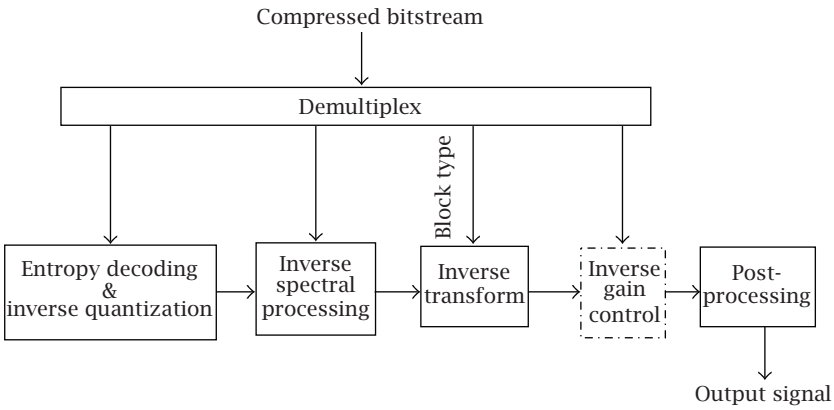


FIGURE 7.3. The block diagram of the AAC decoder.

and masking thresholds, is used as a guide line to control each encoding module. The compressed spectral data as well as that side information generated in each coding module are all multiplexed in the bitstream.

When the decoder receives a compressed bitstream, it first demultiplexes it. Compressed spectral data is then decoded by entropy decoder and inverse quantizer. Side information extracted from the bitstream is used to control the inverse spectral processing, inverse transform, and inverse gain control if activated. If the original sound input to the encoder has out-of-range amplitude, then the post-processing module restores the reconstructed signal back to its original amplitude range.

7.2.2. Psychoacoustic model

Psychoacoustic model plays the most important role in the perceptual audio coding. The output of this model controls almost every major coding block in the AAC encoder. The main task of this psychoacoustic model is to calculate masking thresholds. Any quantization noise added in the later quantization block is imperceptible if its energy is smaller than the masking thresholds. Ideally, if all encoding errors are masked by the signal itself, then the reconstructed sound is perceptually indistinguishable from the original input. However, if the available bit budget does not allow all spectral data to meet masking thresholds, psychoacoustic model needs to guide the quantizer so that the minimum perceptual noise will be added by loosening the psychoacoustic requirements.

7.2.3. Gain control

The gain control block, which is only activated in the SSR profile, includes a polyphase quadrature filter (PQF), gain detectors, and gain modifiers. The PQF filterbank separates the input signal to four equal-width frequency bands. For example, if input sound has 48 kHz sampling rate, then four-band output from the PQF

filter contains signals of 0–6 kHz, 6–12 kHz, 12–18 kHz, and 18–24 kHz. Hence, the bandwidth scalability can be achieved by discarding one or more upper band signals. The distinct feature of this structure allows the decoder to reconstruct a reduced bandwidth signal with lower computational complexity. The gain detector gathers information from the output of the PQF filter and let the gain modifier know which one of these four bands needs amplitude adjustment and how it should be done.

The inverse gain control module in the decoder consists of gain compensators and inverse polyphase quadrature filter (IPQF). Four gain compensators corresponding to four equal-width frequency bands are required if the full-bandwidth signal needs to be reconstructed. Each gain compensator recovers gain control data then restores the original signal dynamics. The resulting outputs after the gain compensation are combined and synthesized by the IPQF to generate final PCM data.

7.2.4. Transform

7.2.4.1. Modified discrete cosine transform

The conversion between time-domain signals as the input of the encoder or the output of the decoder and their corresponding frequency representations is a fundamental component of the MPEG AAC audio coder. This conversion, which is also called time-to-frequency transform, is carried out by a forward modified discrete cosine transform (MDCT) in the encoder, and an inverse modified discrete cosine transform (IMDCT) in the decoder. During encoding, each block of time samples, either 2048 samples for regular long windows or 256 samples for short windows, consists of 50% old signals from previous frame and 50% new signals from current frame. In other words, each block of input samples is overlapped by 50% with the immediately preceding block and the following block. From the MDCT transform function equation (7.1), we know that sequence X_{ik} is odd-symmetric, that is, the ith coefficient has the same amplitude as the $(N - i - 1)$th coefficient. Therefore, the overlapping will not increase the data rate after MDCT is performed. At the decoder side, spectral signals from two consecutive frames are also overlapped by 50% before they are added to reconstruct the time-domain data. Both MDCT and IMDCT adopt a technique called time-domain aliasing cancellation (TDAC) [103]. More information about TDAC and the window-overlap-add process can be found in [103].

The analytical expression for MDCT is given by

$$X_{ik} = \frac{1}{N} \sum_{n=0}^{N-1} x_{in} \cos\left(\frac{2\pi}{N}(n + n_0)\left(k + \frac{1}{2}\right)\right), \quad 0 \le k \le N - 1. \tag{7.1}$$

The analytical expression for IMDCT is

$$x_{in} = \sum_{k=0}^{N-1} X_{ik} \cos\left(\frac{2\pi}{N}(n + n_0)\left(k + \frac{1}{2}\right)\right), \quad 0 \le n \le N - 1, \tag{7.2}$$

where

$$\begin{aligned} n &= \text{sample index}, \\ N &= \text{transform block length}, \\ i &= \text{block index}, \\ n_0 &= \frac{N/2+1}{2}. \end{aligned} \tag{7.3}$$

7.2.4.2. Window function

Before MDCT is performed, each block of samples is first multiplied by an appropriate window function. In other words, MDCT is applied on the modulated signal to generate the frequency-domain spectral data. Similarly, a window function is applied on the spectral signals before they go through the inverse MDCT transform. The window performed before MDCT is called analysis window and the window performed before IMDCT is called synthesis window. In order to have perfect reconstruction, the analysis window $w_{\text{ana}}(i)$ and the synthesis window $w_{\text{syn}}(i)$ should satisfy the following condition:

$$w_{\text{ana}}[i] \times w_{\text{syn}}[i] + w_{\text{ana}}[N-M+i] \times w_{\text{syn}}[N-M+i] = 1, \quad \text{for } i = 1, 2, \ldots, N, \tag{7.4}$$

where N is the window length and M is the number of overlapped samples. If the synthesis window is the same as the analysis window, that is, $w_{\text{ana}}[i] = w_{\text{syn}}[i] = w[i]$, the above condition can be simplified to

$$\begin{aligned} w^2[i] + w^2[N-M+i] &= 1, \quad \text{for } i = 1, \ldots, M, \\ w^2[i] &= 1, \quad \text{for } i = M+1, \ldots, N-M. \end{aligned} \tag{7.5}$$

Two window functions, the sine window and the *Kaiser-Bessel-derived* (KBD) window [32], are provided in AAC coder. Compared with the sine window, the KBD window has a better frequency-selectivity property but a wider mainlobe. Depending on the characteristics of the input signal, the encoder can select the optimum window shape.

7.2.4.3. Block type and block switching

For stationary signals, long blocks of 2048 spectra are processed one at a time. However, for transient signals, this will produce noticeable noise after the quantization. Because the spectra quantization is performed in frequency domain, the quantization noise will extend to more than a few milliseconds when spectral signals are transformed back to time domain. For transient signals, these spread errors may not be masked by the nearby signal and present a perceptible artifact. This phenomenon is called preecho effect. To control the preecho, transient signals should be encoded with shorter transform blocks. However, short transform

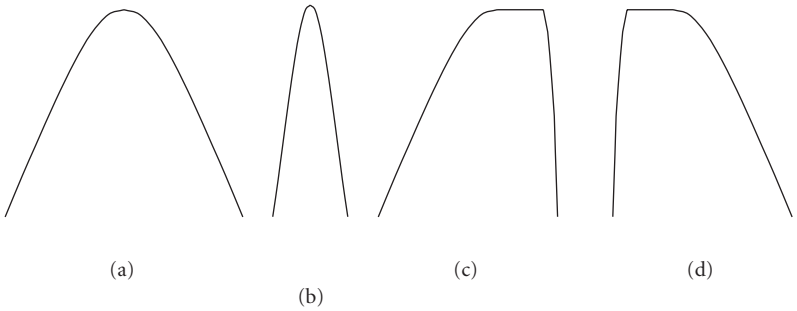


FIGURE 7.4. Different window types in AAC. (a) Long window. (b) Short window. (c) Long start window. (d) Long stop window.

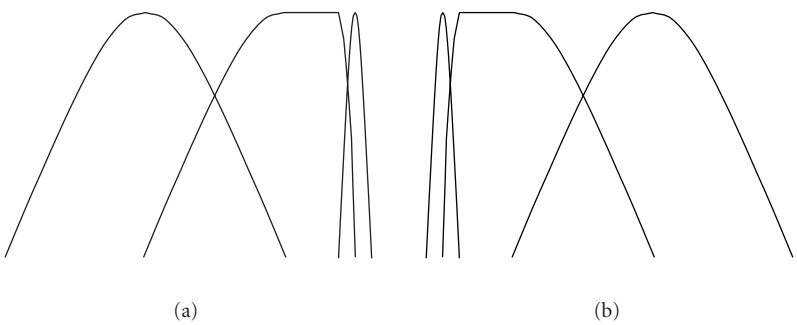


FIGURE 7.5. Window switch in AAC. (a) Long-to-short window switch. (b) Short-to-long window switch.

length results in inefficient coding for stationary signals. Thus, in order to achieve good performance for both regular signals and transient signals, a better solution to determine the window length needs to be reached.

AAC tackles this problem by adopting two block lengths and allowing block switching when different signal type is detected. For steady-state signals, long block is used to increase the coding efficiency. When transient signals appear, short block, which is one eighth of the long block length, is adopted to minimize the preecho effect. The transition between two different block types is carefully handled so that no aliasing would occur if no quantization is involved. Figure 7.4 shows four different window types used in AAC, where (a) is a long window, (b) is a short window, (c) is a long start window, and (d) is a long stop window. The criterion to design the transition window is that the first half of the transition window should always be the same type as the last part of the previous window. Therefore, if a short window is required after several long windows, then the last window before the short window starts should be a long start window. Similarly, when short window is followed by long windows, the first window after the short window should be a long stop window. The long-to-short and short-to-long window switches are illustrated in Figure 7.5.

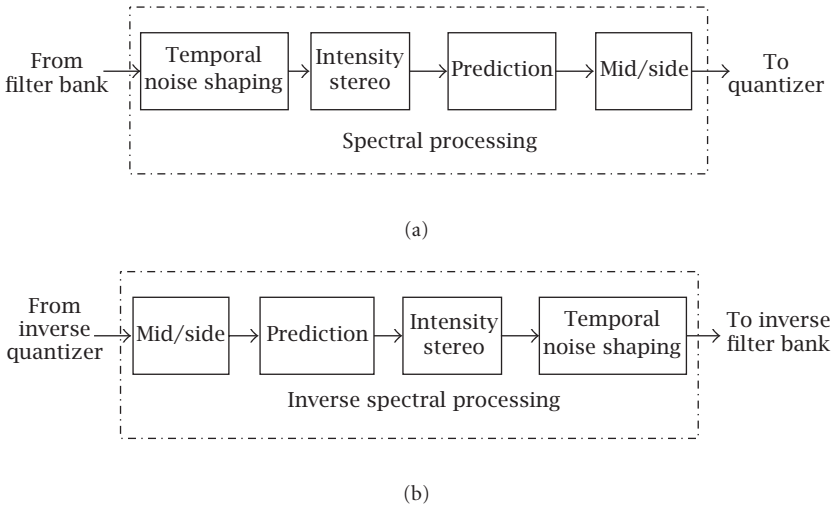


Figure 7.6. Spectral processing and inverse spectral processing in AAC [18].

7.2.5. Spectral processing

Figure 7.6 shows all coding tools provided in spectral processing and inverse spectral processing. Four separate blocks, temporal noise shaping (TNS), intensity stereo (IS), prediction, and mid/side (M/S) coding, are included in spectral processing module. We will discuss each of them one by one in this section.

7.2.5.1. Temporal noise shaping

Temporal noise shaping is a tool that further improves the coding performance when transient and pitched input signals are present. In Section 7.2.4.3, we discussed the block switch method. By switching to short length window, the AAC encoder is capable to reduce the preecho effect. However, the result of block switching alone does not achieve satisfactory coding result. This is because only one masking threshold is used to control the quantization noise in each block, while signals in one block may still vary considerably. Coding is especially difficult if there are temporal mismatches between the masking threshold and the quantization noise (preecho). To alleviate the above problem, AAC introduces a new concept, the temporal noise shaping tool, to the perceptual audio coding. With TNS, the encoder is capable to control the fine temporal structure of the quantization noise even within a filterbank window.

The basic idea of TNS is to do frequency-domain prediction. Regular time-domain prediction is efficient to code signals with "unflat" spectrum, such as a sine wave. While the frequency-domain prediction is a good choice to compress signals with "unflat" time structure, such as transient signals. Similarly as the time-domain intra-channel prediction method that increases coder's spectral

resolution, the prediction over frequency enhances the coder's temporal resolution. In addition, frequency-domain prediction helps the noise shaping so that the quantization noise can be put under the actual signal's masking threshold.

The TNS coding module is a noise-shaping filter [49]. Linear predictive coding (LPC) is adopted in the filter. The order of the filter depends on what profile the codec is working on. The higher the filter's order is, the higher the memory requirement and the computational complexity. Moreover, TNS filtering does not have to be applied to the entire spectrum. And different prediction filters can be applied to different frequency regions. In MPEG-2 AAC, only necessary frequency regions have active TNS filtering module.

At the decoder side, the TNS coding module is an inverse TNS filtering block and is inserted before the frequency-domain spectral data is converted back to the time-domain signals by the synthesis filterbank.

If we consider the encodering filterbank and the adaptive (TNS) prediction filter to be a compound continuously signal adaptive filterbank, then according to the characteristics of the input signal, the behavior of this type of adaptive composite filterbank switches between a high-frequency resolution filterbank (for stationary signals) and a high-time resolution filterbank (for transient signals) dynamically [48]. Thus, adding the TNS tool significantly improves the overall coding performance, especially for speech stimuli.

7.2.5.2. Joint stereo coding

Joint stereo coding has been proven extremely valuable for compressing high-quality stereophonic (or multichannel) audio signals at low bit rates. Based on the binaural psychoacoustic theory, joint stereo coding tool was developed to significantly reduce the bit rate for multichannel audio signals to a rate much lower than that required for coding of multiple input channels independently.

Figure 7.7 is a simple illustration of how joint stereo coding is applied on a five-channel audio signal. Signals in each input channel are independently processed through some coding module, such as transform, psychoacoustic model, and so forth, Then they will be processing by a joint stereo coding block. The outputs of the joint stereo coding are easier to be compressed than signals in original input channels. Some other processing tools, such as quantization and noiseless coding, are carried out on signals in the modified channels independently. Finally, bits are multiplexed to form the bitstream. Two techniques are included in the joint stereo coding module. One is called mid/side (M/S) stereo coding (also known as "sum/difference coding"), the other is called intensity stereo coding. Similar as TNS coding, both techniques in joint stereo coding can be selectively applied on different frequency regions. Brief introductions on M/S stereo coding and intensity stereo coding will be listed below. More detailed discussion on joint stereo coding in MPEG-2 AAC can be found in [74].

M/S stereo coding. In MPEG-2 AAC, M/S stereo coding is applied to signals at lower-frequency region in each channel pair of the multichannel audio source,

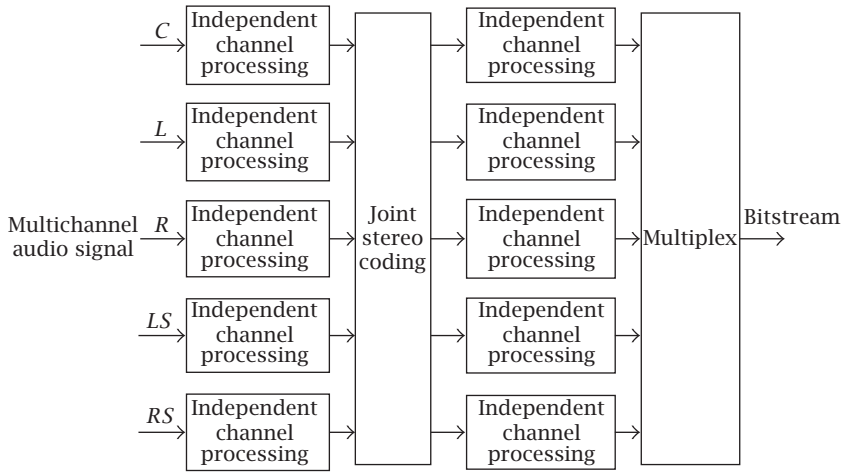


FIGURE 7.7. Illustration of a multichannel joint stereo coder.

such as the left and right channel pair, left surround and right surround channel pair, an so forth. In other words, lower-frequency signals belonging to any pair of channels that are arranged symmetrically on the left/right listener axis will go through the M/S stereo coding block in MPEG-2 AAC. With M/S stereo coding, the signal in left and right channels $L(i)$ and $R(i)$ will be replaced by $(L(i)+R(i))/2$ and $(L(i)-R(i))/2$. New signals are referred to as the middle (M) and side (S) channels.

Depending on input signals' characteristics, M/S can be enabled and disabled, not only from frame to frame but also from subband to subband. A single bit representing the M/S stereo on or off state needs to be sent to the bitstream so that the decoder can reconstruct the original channel accordingly.

Intensity stereo coding. The idea of intensity stereo coding has been utilized widely for stereophonic and multichannel coding under other names, such as dynamic crosstalk and channel coupling, in the past [15]. Based on the fact that the perception of high frequency sound components relies mainly on the analysis of their energy-time envelopes [13] rather than signals themselves, the encoder can transmit only a single set of spectral values and share them among several audio channels while still achieving excellent sound quality. In addition to shared spectral values, some side information, such as energy envelopes, needs to be included in the bitstream so that the decoder can recover signal's original energy level.

Two general mechanisms are provided in MPEG-2 AAC's intensity stereo coding block. The first one, called intensity stereo coding, is applied to higher-frequency components of channel pair elements. With intensity stereo coding, the signal in left channel $L(i)$ and right channel $R(i)$ will be replaced by $(L(i)+R(i))\times\sqrt{E_l/E_r}$ and 0, respectively, where E_l and E_r represent the subband energies of left and right channels. The implementation of intensity stereo coding is quite straightforward and covers most of the common needs with little overhead.

The second mechanism in intensity stereo coding, which is called AAC coupling-channel element, allows better control of coding parameters by relaxing the constrain on the channel pair concept. Coupling channels can be considered as an extended intensity stereo coding, where channel spectral values can be shared not only between channel pair elements but also between different channel pairs. Moreover, they can be used to downmix additional sound objects into stereo audio files.

7.2.5.3. Prediction

In MPEG-2 AAC main profile, a second-order backward prediction is adopted. The advantage of backward prediction over forward prediction is that it uses previous samples instead of future samples to predict the current sample. Therefore, no additional side information needs to be sent into the bitstream. For each spectral sample, the quantized spectral values at the same frequency region in previous two frames are used. If the prediction results in a coding gain for a certain subband, that is, coding the prediction residue consumes less bits than coding original signals, then the prediction will be set active for this subband. In order to signal the predictor's on or off state, a one-bit flag is required for each subband. When all subbands' prediction state has been checked, the prediction efficiency for the entire frame also needs to be checked. In other words, in cases when bits saved by enabling some of the subbands' prediction block cannot compensate the overall required side information, the prediction of this frame shall be completely disabled and only one-bit flag needs to be included in the bitstream.

Although both TNS and prediction modules use predictor, two major differences exist between these two techniques. Unlike TNS coding module, where prediction is performed over adjacent frequency samples within a frame, the MPEG-2 AAC prediction coding module implements prediction over samples in different frames. The TNS tool provides a temporal resolution enhancement, whereas prediction tool increases the temporal scope of the perceptual codec. The second difference between these two tools is that TNS filtering is performed over signals with any block type and is especially effective for transient signals. However, prediction of spectral coefficients over time is only suitable for stationary signals and no prediction over time is enabled for signals with short window type.

In real applications, the encoder and decoder may be running on different systems that exhibit different behaviors. This small difference between encoder and decoder may accumulate as the prediction over time goes further and further. To prevent significant diverge between encoder and decoder, the predictor should be reset once after a while. In MPEG-2 AAC, predictors of all coefficients are separated to several groups. Predictor reset is performed one group at a time. By doing this, resetting of all predictors can be achieved without considerably affecting the overall prediction gain.

The MPEG-2 prediction tool is only available for main profile encoder and decoder because of the high storage and computational power requirement. In fact, not all coefficients in all frequency range have active predictor even in the

main profile codec. And none of coefficients corresponding to higher frequency range performs any prediction calculation in any circumstances.

7.2.6. Quantization

All spectral processing blocks we discussed above as well as the entropy coding module have perfect inverse procedures, which means signals going through these modules can be precisely reconstructed if no intermediate computational error is involved. On the contrary, the effect of quantization module is irreversible and is crucial to the overall compression performance. The ultimate goal of quantization in AAC is twofold. The first goal, which is the same as the quantization module for compression of any other multimedia signals, is to reduce the necessary bit so that after the entropy coding, the bit requirement is under the target bit limit. The second goal, which is unique for audio coding, is to control the quantization noise so that psychoacoustic requirement can be met or no perceptual noise should be introduced after the quantization.

AAC encoder takes care of quantization in two steps. One is the quantization for spectral samples. The other is the quantization for the scale-factor band.

(i) *Quantization for spectral samples.* A nonuniform power-law quantizer

$$s_q(i) = \operatorname{sign}\left(s(i)\right) \times \operatorname{round}\left[\left(\frac{|s(i)|}{\sqrt[4]{2}^{sf}}\right)^{0.75} - \alpha\right] \tag{7.6}$$

is adopted, where $s(i)$ and $s_q(i)$ represent the original and quantized spectral samples, function round(x) returns the nearest integer value that is close to value x, α is a small constant, and sf represents the quantization parameter for the scale-factor band that sample i resides. With a nonuniform quantizer, the increase of the signal-to-noise ratio with the increasing signal energy is significantly lower than that of a linear quantizer. In fact, the feature of having the built-in noise shaping depending on the amplitude of coefficients is the main advantage of using a nonuniform quantizer. Huffman codes are used to code the quantized coefficients output from the quantizer. Several Huffman tables based on different probability models are available for any spectrum. The details of the lossless coding process is described in Section 7.2.7.

(ii) *Quantization for scale-factor bands.* Beside the use of a nonuniform quantizer, an additional method to shape the quantization noise is required in order to fulfill the psychoacoustic demands. In AAC, the encoder uses the amplification for each individual groups of spectral coefficients (scale-factor bands). If we can shape the quantization noise in units similar to that of the critical bands of the human auditory system, then the psychoacoustic requirements can be fulfilled more efficiently. This is done in AAC quantization block by adding a parameter sf for each scale-factor band as shown in (7.6). Since the decoder needs to perform the inverse quantization, the side information about parameters in each scale-factor band needs to be included in the bitstream. The first scale-factor parameter for

band #0, which is called *global gain*, is coded by a fixed-length PCM code. Parameters for all the rest scale-factor bands are differentially Huffman encoded.

Ideally, quantization block should be designed so that both the bit rate and the psychoacoustic requirement are fulfilled. However, in real encoding, there are many cases that one or both of these requirement cannot be satisfied. When this happens, the encoding algorithm should come up with a solution that either increases the available bit or loosens the psychoacoustic requirement. In fact, different implementation may approach this problem differently. And how it is carried out substantially affects the overall coding performance. In AAC, the target bit limit for each frame is usually the average bit per frame, which can be calculated from the desired bit rate and the sampling frequency of input signals. In addition, a bit reservoir is available for each frame so that an extra variable bit distribution is allowed between consecutive frames on a short-time basis. However, there is no standard procedure in MPEG AAC on how the psychoacoustic requirement should be loosened when the bit limit cannot be achieved. This is because MPEG AAC did not standardize the encoder part; as long as the compressed bitstream is compliant with the syntax as given in [60], any encoder design is allowed. In paper [15], Bosi et al. show a double-nested iteration loop to control the quantization. Although this method generates fairly good results and has been adopted in the MPEG AAC reference software, it is time-consuming and computationally exhausted. Based on the reference code, different quantization module [83] is proposed to modify the double loop and significantly improve the encoding speed.

7.2.7. Entropy coding

Each time, a set of 1024 quantized spectral coefficients is sent to the entropy coding module to further reduce the bit rate. The goal of this module is to represent all quantized samples as efficiently as possible without introducing new errors. In the beginning, a noiseless dynamic range compression may be applied to the spectral data. Since dynamic range compression requires sending side information into the bitstream, it is only activated when a net saving of bits can be achieved. Beside dynamic range compression, three other techniques, that is, sectioning, Huffman coding, grouping, and interleaving, are included in the entropy coding blocks.

7.2.7.1. Sectioning

Since each scale-factor band may exhibit different statistics, different Huffman tables could be used for each scale-factor band. However, a better way is to group several adjacent scale-factor bands into one section and let them share a common Huffman codebook so that the side information required to represents the Huffman codebook index can be reduced. There are a number of choices for sectioning, the encoder can perform a search for different sectioning parameters and find the one that gives the best overall bit reduction results.

Table 7.1. Huffman codebooks used in AAC.

Codebook index	Dimension	Max amplitude	Signed/unsigned
0	—	0	—
1	4	1	Signed
2	4	1	Signed
3	4	2	Unsigned
4	4	2	Unsigned
5	2	4	Signed
6	2	4	Signed
7	2	7	Unsigned
8	2	7	Unsigned
9	2	12	Unsigned
10	2	12	Unsigned
11	2	16(ESC)	Unsigned

7.2.7.2. Huffman coding

Twelve predefined Huffman codebooks are provided in AAC. The index of Huffman codebook together with the maximum absolute value of the quantized coefficients that can be represented by each Huffman codebook and the number of coefficients in each n-tuple for each codebook are shown in Table 7.1. This table indicates that both 2-dimensional and 4-dimensional codebooks are available. Among codebooks #1 through #10, for each maximum absolute value, two codebooks are provided. Although both of them can be used for encoder, only the best one, which is often the one that represents a probability distribution close to the source, should be chosen for final encoding.

Huffman codebooks #0 and #11 are two special tables. Codebook #0 indicates a section with all coefficients equal to zero. By using this codebook, only the codebook index 0 needs to be included in the bitstream, neither the scale-factor parameter nor the coefficients' codeword needs to be sent to the decoder. This codebook is especially useful for input signals with bandlimited feature. Codebook #11 is used to represent quantized coefficients that have an absolute value greater than or equal to 16. For each such coefficients, an escape code is used to indicate the excess value portion of the codeword. The largest absolute value that codebook #11 can represent is 8191. The sign bit of the coefficient should be appended to the codeword as necessary.

7.2.7.3. Grouping and interleaving

For the case of eight short windows, where the set of 1024 coefficients is actually a matrix of 8 by 128 frequency coefficients, a *grouping* and *interleaving* method is introduced to achieve higher coding gain. The purpose of grouping and interleaving is to rearrange the order of the quantized spectral in such a way that coefficients of similar magnitude are sorted into contiguous region thus minimum overhead

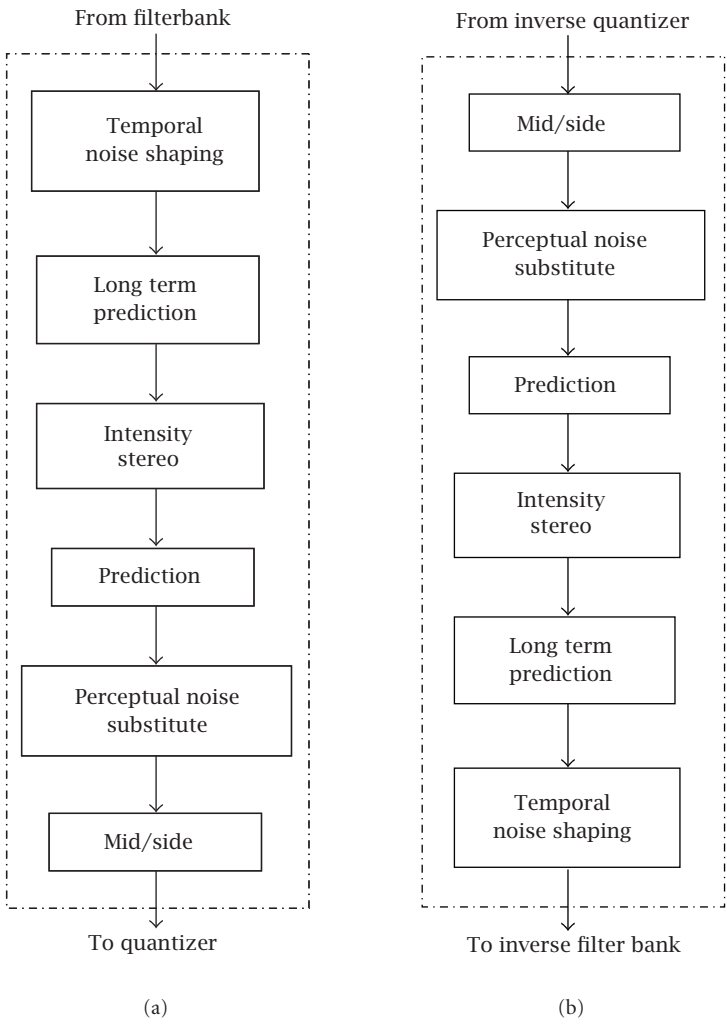


FIGURE 7.8. Spectral processing block for MPEG-4 AAC's (a) encoder and (b) decoder [58].

consumes the limited bit budget. Grouping is a strategy that selects contiguous short windows to form groups. Within each group, scale-factor parameters among all scale-factor bands can be shared, whereas interleaving changes the order of scale-factor bands, windows, and coefficients within a group and is especially useful for bandlimited signals. Both grouping and interleaving enhance the coding efficiency for short block signals.

7.3. New features in MPEG-4 AAC

MPEG-4 AAC has several new tools to enhance the coding efficiency. Figures 7.8(a) and 7.8(b) illustrate the spectral processing block for MPEG-4 AAC's encoder

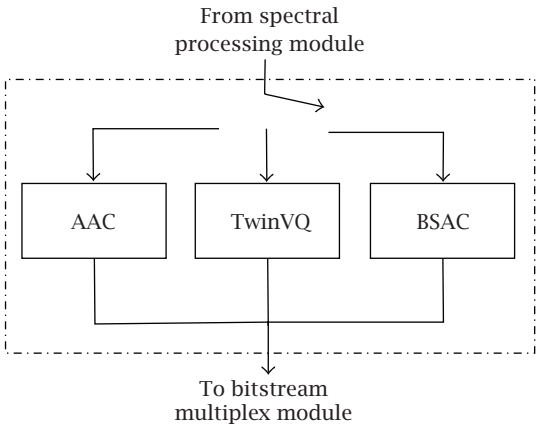


FIGURE 7.9. MPEG-4 AAC's quantization and entropy coding module [58].

and decoder. Two new coding blocks, *long-term prediction* (LTP) and *perceptual noise substitute* (PNS) are included in MPEG-4 AAC's spectral processing module. Figure 7.9 shows MPEG-4 AAC's quantization and entropy coding module, where a *transform-domain weighted interleave vector quantization* (TwinVQ) and a *fine-grain scalable* (FGS) bit-sliced arithmetic coding (BSAC) tool are added as optional coding kernels. Based on MPEG-2 AAC, two new functionalities, *low-delay* AAC (AAC-LD) coder and the *error-resilience*, are introduced by modifying one of original AAC's coding blocks. All above new coding tools and new functionalities will be addressed in following sections.

7.3.1. Perceptual noise substitution

Most of audio coding tools in MPEG-2 AAC's spectral processing module belong to waveform coding method, which means they have perfect inverse processing blocks that can reconstruct an exact copy of input signals. Contrary to the waveform coding method, the model-based (*vocoder*) technique, which does not have a matching perfect inverse processor, produces intermediate sound quality at much lower bit rate. Instead of trying to reproduce a waveform that is similar as input signals, the model-based coding tries to generate a perceptually similar sound as output. The perceptual noise substitution (PNS) tool [51] is one of this kind of nonwaveform coding methods and has been widely used in speech coding.

From the psychoacoustic study, it is found that the subjective sensation stimulated by a noise-like signal is determined by the input signal's spectral and temporal fine structure rather than its actual waveform. Based on this observation, a parametric coding tool, PNS is introduced to the MPEG-4 AAC spectral processing module. The encoding of PNS includes two steps.

(1) *Noise detection.* For input signals in each frame, the encoder performs some analysis and determines if the spectral data in a scale-factor band belongs to noise component.

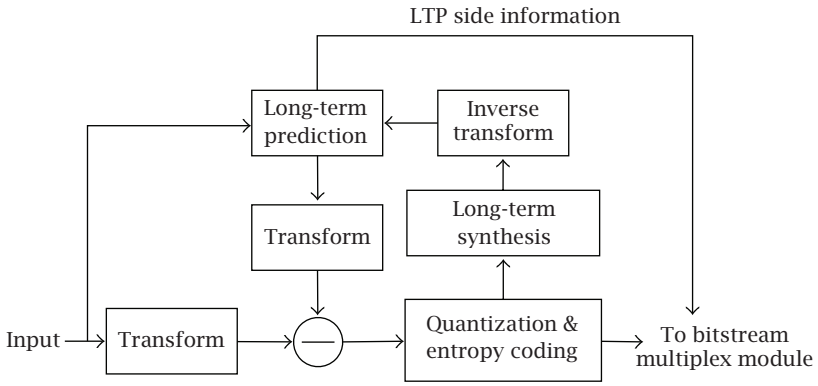


Figure 7.10. Encoding block diagram of MPEG-4 long-term prediction [58].

(2) *Noise compression.* All spectral samples in the noise-like scale-factor bands are excluded from the following quantization and entropy coding module. Instead, only a PNS flag and the energy of these samples are included in the bitstream.

The decoding of PNS is much simpler. When a PNS flag is signaled, the decoder replace the actual spectral coefficients with some pseudorandom numbers. The energy of the substitute random numbers needs to be adjusted according to PNS parameters.

The perceptual noise shaping technique significantly reduces the bit requirement for the noise-like components. It is verified that, at low bit rates, the PNS tool has the ability to enhance the coding efficiency for complex musical signals while maintaining similar memory and computational requirements.

7.3.2. Long-term prediction

MPEG-4 AAC's long-term prediction (LTP) is another nonwaveform tool that borrows idea from the speech coding. It is an efficient tool to reduce signals' redundancy between successive frames, especially for signals that have clear pitch property, which is true for speech signals and many other sound signals. Unlike MPEG-2 AAC's backward prediction tool, LTP is a forward-adaptive scheme. Although side information has to be sent to the decoder, LTP achieves similar compression result while only requiring half of the RAM storage and computational complexity as that of MPEG-2 AAC's prediction tool. Moreover, it is less sensitive to errors if spectral coefficients are not correctly reconstructed in previous frames due to transmission failure.

Figure 7.10 depicts a simple encoder block diagram of an MPEG-4 AAC coder with LTP. For simplicity, the psychoacoustic model is excluded in the figure. For each frame, the long-term prediction is carried out based on quantized spectral coefficients. The predicted signals and the original signals are both transformed into frequency domain. Errors of these signals are calculated and then quantized,

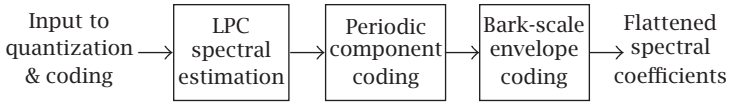


Figure 7.11. Spectral normalization in TwinVQ kernel.

coded, and sent to the bitstream. Side information including predictor control flags and predictor parameters also needs to be sent to the decoder.

In MPEG-4 AAC, long-term prediction is defined as a separate profile. However, the LTP bitstream syntax is backward compatible with that of MPEG-2 AAC LC profile. In other words, any regular nonscalable MPEG-2 AAC LC bitstream is decodable by an MPEG-4 AAC LTP decoder. On the other hand, any MPEG-4 AAC LTP bitstream can be decoded by an MPEG-2 AAC LC decoder, when LTP is not activated.

Different from MPEG-2 backward-adaptive prediction tool, LTP is not restricted to be used for long windows. It can also be applied to signals with eight-short window cases. Testing results have shown that LTP tool not only provides good coding gain for stationary harmonic signals, but also improves the coding performance to a certain degree for nonharmonic tonal signals.

7.3.3. TwinVQ

TwinVQ [69, 70] is an additional profile provided by MPEG-4 general audio coder. It can be used to replace the MPEG-2 AAC's quantization and coding kernel. Based on the vector quantization scheme, this TwinVQ tool is capable of providing good coding performance at extremely low bit rates (down to 6 kbps) for general audio signals.

In the TwinVQ kernel, two steps are performed in the process of quantizing and coding of the spectral coefficients. The first step is to normalize spectral coefficients, where the spectral coefficients' values are scaled into a specified target amplitude range. The second step is to quantize and code these normalized or flattened spectral coefficients by means of perceptually weighted vector quantization.

7.3.3.1. Normalize spectral coefficients

The normalization process flattens the spectral coefficients' amplitude to a desired range by performing scaling and extracting parameters that represent the signal's characteristic. After this process, the spectral coefficients become more flattened across the frequency axis than before. The extracted parameters need to be quantized and transmitted as side information to the decoder.

Three tools as shown in Figure 7.11 are associated with the spectral normalization step. They are LPC spectral estimation, periodic component coding, and Bark-scale envelope coding.

(i) *LPC spectral estimation.* An LPC model is adopted to estimate the overall coarse spectral envelope of the input signal, which is then used to normalize

the amplitude of the spectral coefficients. The LPC envelope parameters are transformed to the line spectral pair (LSP) representation so that they can be more efficiently coded.

(ii) *Periodic component coding.* Periodic peak components, which correspond to harmonic peaks in the spectrum, are coded for frames using long windows. The encoder estimates the pitch, that is, the fundamental signal frequency, and extracts a number of periodic peak components from LPC output coefficients. Side information about the pitch value and the average gain of the components needs to be sent to the bitstream.

(iii) *Bark-scale envelope coding.* By using a spectral envelope based on the Bark-related AAC scalefactor bands, the encoder can further flatten the resulting coefficients from previous steps. The vector quantization with interframe prediction is then used to quantize these envelope values.

7.3.3.2. Perceptual weighted VQ

The weighted vector quantization is carried out in the second phase of TwinVQ to code the flattened spectral coefficients. This process is illustrated in Figure 7.12 and can be separated into three stages: interleaving, weighting, and VQ.

(i) *Interleaving.* Flattened spectral coefficients are first separated into groups of subvectors. Generally speaking, coefficients corresponding to lower-frequency range require more bits in quantization than those coefficients corresponding to the higher-frequency range. Therefore, if consecutive coefficients are grouped together, the resulting subvectors would be unbalanced and are hard to be compressed. By interleaving the spectral coefficients in frequency, subvectors with comparable properties in terms of required accuracy can be grouped together. Consequently, it is possible to allocate constant bits for each subvector.

(ii) *Weighting.* A weighting factor, which is controlled by the psychoacoustic model [69], is introduced to each subvector before the final vector quantization is performed. By applying this adaptive weighting process, the quantization noise can be shaped perceptually to reduce possible noticeable artifact in the reconstructed sound.

(iii) *Vector quantization.* The VQ scheme is used to quantize the interleaved subvectors. By finding the best codebook indices, the encoder can minimize the quantization distortion. In order to achieve the perceptual control of the quantization distortion, a weighted distortion measure is applied during the codebook index selection for each subvector.

7.3.4. Low-delay AAC

Despite the fact that the MPEG-4 AAC provides excellent coding performance for general audio signals at low bit rates, its long algorithmic coding delay prevents it from being used by many real-time applications. For example, with a 24 kHz sampling rate input audio source, when coded at 24 kbps, MPEG-4 AAC's theoretical coding delay is about 110 milliseconds, and may have additional delay as

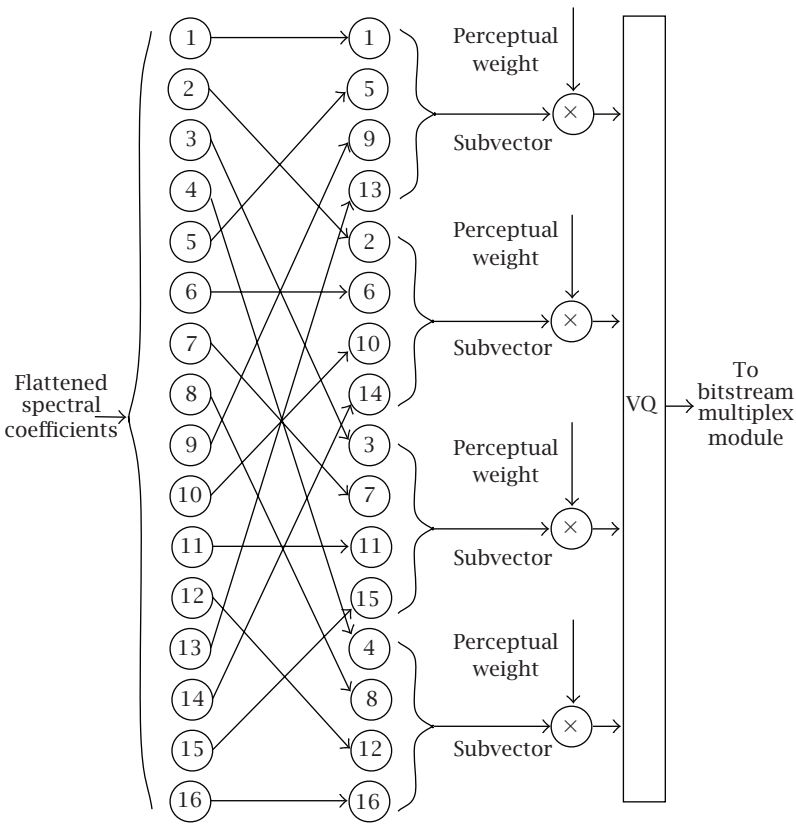


FIGURE 7.12. Weighted vector quantization [58].

long as 210 milliseconds due to the use of bit reservoir. This is certainly not tolerable by real-time bidirectional communications.

A *low-delay audio coding mode* (AAC-LD), which is derived from the MPEG-4 AAC LTP audio object type [9], is developed in MPEG-4 audio version 2 to enable coding of general audio signals with an algorithmic delay down to 20 milliseconds. Compared with the standard MPEG-4 AAC MAIN audio object type, although some limitation exists in the coding strategy due the low-delay modification, the low-delay AAC requires only about 8 kbps/ch additional bit rate to achieve the same reconstructed audio quality.

Based on the MPEG-4 AAC LTP coding scheme, the following four modifications have been made to achieve the low-delay functionality.

(i) *Reduced window size.* Instead of using window size of 1024 or 960 samples in standard MPEG-4 AAC, the low-delay mode uses a frame length of only 512 or 480 samples. In other words, the window size used in the analysis and synthesis filterbanks in AAC-LD is reduced by a factor of two.

(ii) *Disable window switching.* The regular window type switching is disabled in AAC-LD so that the *look-ahead* delay can be avoided. In order to eliminate

the annoying preecho artifact in the case of transient signals, only TNS tool is employed together with the window shape adaptation.

(iii) *Modification of window shape.* In order to achieve the optimum TNS performance and to reduce the MDCT's temporal aliasing effect, a low overlap window is applied for transient signals. The shape of the low overlap window is derived from the combination of a long start window and a long stop window. This low overlap window replaced the KBD window for transient signals. For nontransient signal parts, a sine window is used.

(iv) *Minimizing the use of bit reservoir.* In order to reach the desired target delay, the use of the bit reservoir is minimized or disabled at the encoder.

7.3.5. Error-resilient tools

In order to enhance the performance over error-prone transmission channels, MPEG-4 audio version 2 added several *error-resilient* (ER) tools to the AAC codec. On top of the error-resilient bitstream, some *error-protection* (EP) tools can be added to further improve the bitstream's error robustness. Despite these error-resilient and error-protection tools, errors may still occur, in which case *error-concealment* tools can be activated to recover or smooth the reconstructed sound. MPEG-4 audio does not standardize the error-concealment techniques and the error-protection tools are common for all MPEG-4 audio codecs. Therefore, only the AAC related error-resilient tools will be discussed below.

Three tools are introduced in the AAC ER scheme.

(1) *Virtual codebook (VCB11).* From Section 7.2.7, we know that Huffman codebook #11 is used to code those scale-factor bands that contain spectral coefficients with maximum absolute values greater than 16. When bit error occurs for these parts of bitstream, it usually leads to annoying perceived distortions. This is because large amplitude coefficients have more perceptual impact than coefficients with smaller amplitude values. Therefore, more accurate reconstruction of these coefficients results in better reconstructed sound quality. MPEG-4 AAC introduces 16 virtual codebooks, all of which refer to the same codes as codebook #11, but provides 16 difference limitations on the spectral values. By doing this, bit errors corresponding to large spectral coefficients, which lead to serious perceptual artifacts, can be properly detected and recovered.

(2) *Reversible variable-length coding (RVLC).* The RVLC tool uses symmetric codewords. When error happens in the middle of the bitstream, if the regular codewords are used, then most of the rest data in the current frame cannot be recovered. However, with the reversible variable-length codes, the decoder can start backward decoding from the end, then the percentage of the corrupted data can be reduced and result in better quality of the reconstructed sound. Note that in order to make the bitstream backward decodable, both the length of the scalefactor data and the value of the last scalefactor need to be sent to the decoder. Moreover, not all RVLC achieves the same coding performance. More information on how to construct the reversible length codes can be found in [123].

(3) *Huffman codeword reordering (HCR)* [120]. One of the biggest impacts of bitstream error is the loss of synchronization or the error propagation. Once the synchronization is lost, the decoder cannot perform as expected until the synchronization is regained. The HCR tool is developed based on the exploration that some Huffman codewords can be placed at known regular positions within the bitstream so that even in the presence of bit errors a proper synchronization can be regained at these positions. The idea of HCR is implemented by defining a set of regular positions in the bitstream and each of them is filled with a *priority codeword* (PCW). Between two PCWs are all remaining non-PCWs. For better results, more important coefficients are placed nearer to the PCW, so that they are less likely to be affected by the previous errors.

The above three error-resilient tools are provided for regular AAC quantization and coding module. Error-resilient tools for BSAC will be covered in Section 7.3.6.2. For the TwinVQ scheme, since the vector quantization uses only fixed-length codewords, it is much more robust to the bit error than variable-length codewords. Therefore, MPEG-4 AAC provides no additional error-resilient tools for TwinVQ.

7.3.6. MPEG-4 scalable audio coding tools

In the traditional perceptual audio coding design, each compression bitstream corresponds to one specific bit rate, which needs to be provided at the beginning of the encoding. Different bitstream coded at different bit rate usually has no meaningful overlap. This kind of traditional codec design is not appropriate for applications where target bit rate is not available beforehand or is varying depending on the channel capacity or the transmission conditions. In order to solve this problem, MPEG-4 audio standardized scalable audio coding algorithms, where the target bit rate of the transmission and the decoding of the bitstream can be adapted to dynamically varying requirements, such as the instantaneous transmission channel capacity or the decoder computational resources. MPEG-4 audio version 1 has scalability functionality in terms of large-step scalable coding. This functionality is integrated into the AAC coding framework and is capable to generate a single bitstream which adapts to different channel characteristic. In MPEG-4 audio version 2, the *fine-grain scalability* (FGS) mode based on bit-sliced arithmetic coding (BSAC) is introduced. This is the first time that fine-grain scalability is provided in MPEG-4 audio. In the previous MPEG standards, such kind of scalability is only available for video coding.

Scalable audio codecs are typically accomplished by the concept of *hierarchical embedded coding*, which means the bitstream generated by a scalable audio codec is composed of several partial bitstreams that can be independently decoded and then combined to form a meaningful decoding result. In other words, decoding the first several subsets of a hierarchical full bitstream will result in a valid decoded signal but corresponding to lower quality.

Figure 7.13 illustrates a hierarchical embedded coding scheme corresponding to a three-layer coding mechanism. The encoder #1 (*base layer* coder) is the first

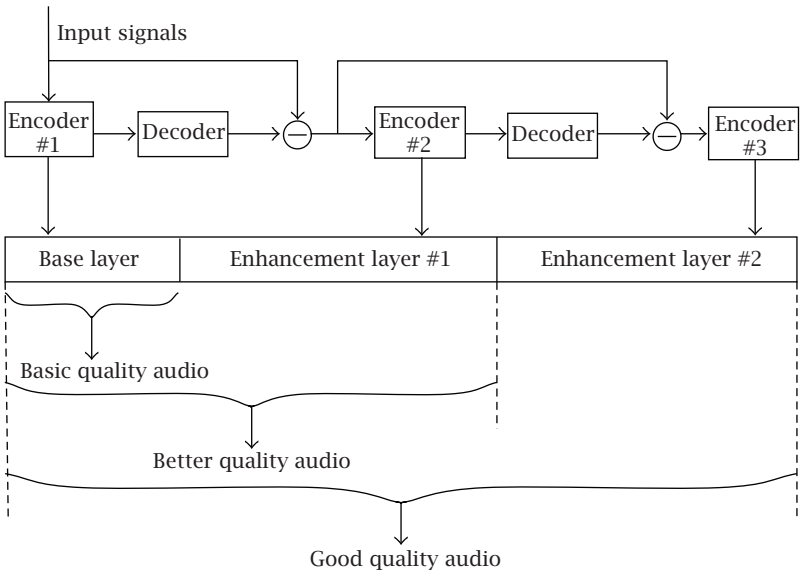


FIGURE 7.13. Hierarchical embedded coding scheme [58].

coder involved in the scheme. It codes the input audio signal into a bitstream delivering the audio with the basic perceptual quality. The coding error of the base layer coder is calculated by a matching decoder and then subtracted from the original signal. The residual data is coded by an encoder #2 (*enhancement layer* coder #1), and then output as the second part of the scalable bitstream. Decoding of base-layer and enhancement-layer #1 subbitstreams gives a better quality sound. Similarly, the encoder #3 (enhancement-layer coder #2) codes the residue signal resulting from encoders #1 and #2. This additional subbitstream enables the decoder to generate a good copy of the original sound.

In fact, more encoders can be included in this hierarchical embedded coding scheme to generate more enhancement layers and finer coding scalability. In MPEG-4 audio version 1, the concept of scalability is achieved by a limited number of layers (typically 2 to 4), while MPEG-4 audio version 2 includes a scalable coder that contains considerably more layers. Therefore, MPEG-4 audio version 1's scalable coding mechanism is referred to as *coarse-grain scalability* tool and MPEG-4 audio version 2's scalable coding mechanism is referred to as *fine-grain scalability* tool.

7.3.6.1. Coarse-grain scalable tool

The general architectures of an MPEG-4 two-layer large-step scalable audio encoder and decoder are illustrated in Figures 7.14 and 7.15, respectively. This encoder configuration contains a non-AAC base-layer coder and an enhancement layer based on AAC coding modules. A narrowband speech coder, such as the MPEG-4 CELP coder, can be adopted as the non-T/F base-layer coder. This coder

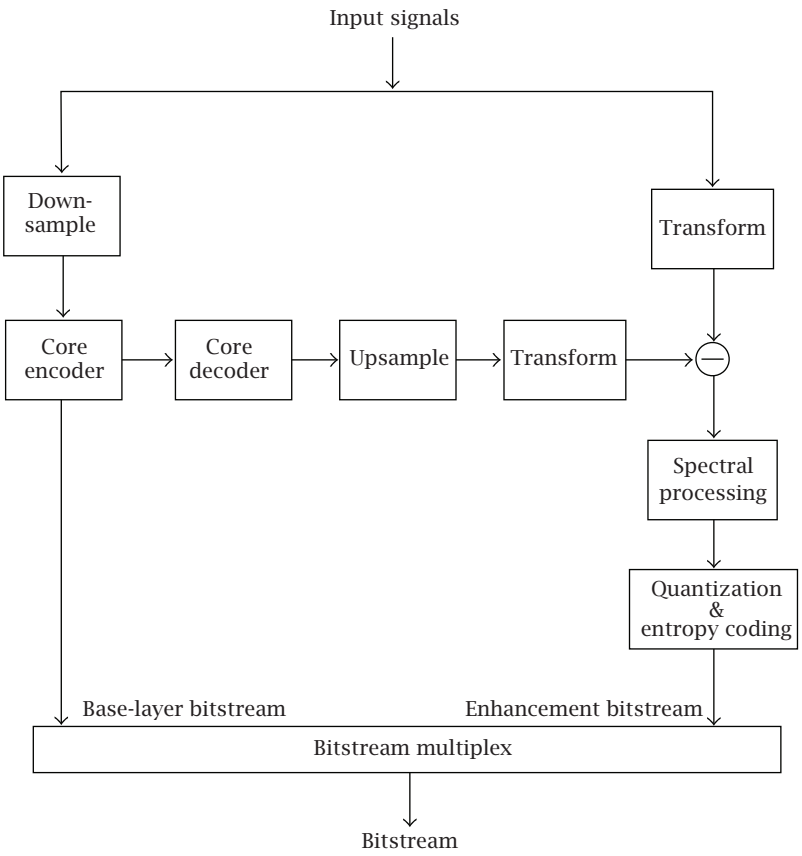


FIGURE 7.14. General architecture of an MPEG-4 two-layer large-step scalable audio encoder [58].

is called a *core coder* [41] and often operates at a lower sampling rate than the subsequent enhancement-layer coder. A more detailed description of how these scalable encoding and decoding components work is listed below.

(i) Encoding.

(1) The input audio signal is first downsampled and processed by the core coder. The base-layer bitstream occupies the first portion of the entire scalable coding output. This bitstream is then decoded, upsampled, and fed into the AAC time-to-frequency transform block.

(2) In the right path of the figure, the input signal is transformed into frequency domain via AAC's MDCT filterbank. These spectral coefficients are then subtracted by the corresponding coefficients calculated from the first step and produce residual coding error signals.

(3) Error coefficient output from the second step is then processed by the AAC spectral processing module and the quantization and

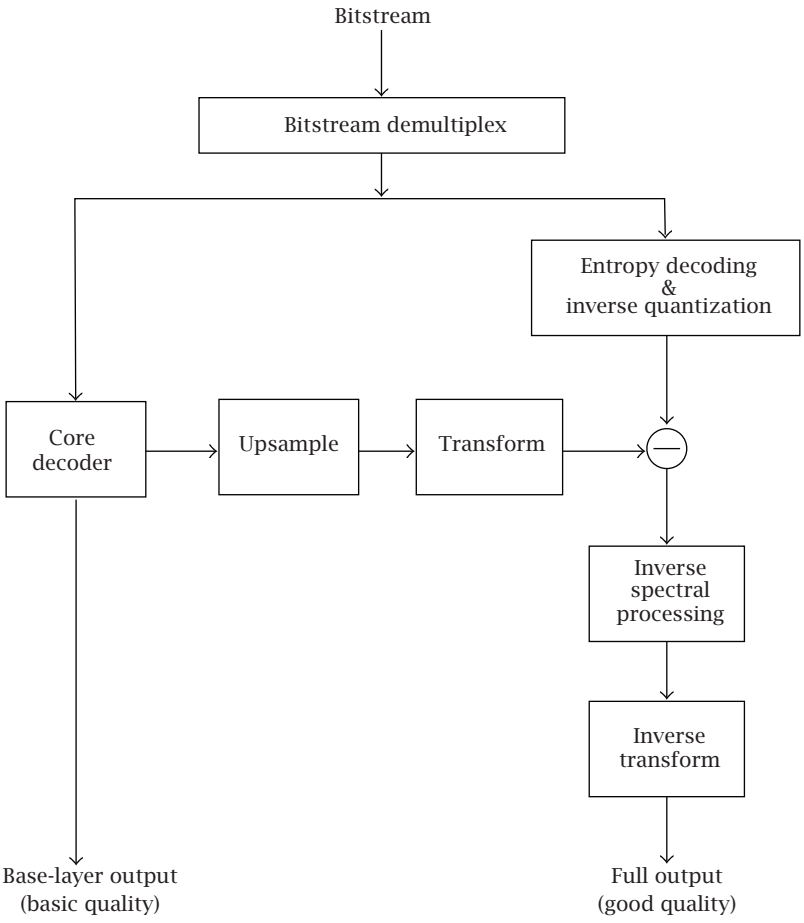


FIGURE 7.15. General architecture of an MPEG-4 two-layer large-step scalable audio decoder [58].

coding block. The bitstream generated from the right path of the encoding block diagram constructs the second portion of the entire scalable coding output, which enables the decoder to reconstruct a better copy of the original audio signal.

(ii) Decoding.

(1) The decoder first decomposes the bitstream into two different substreams, each of which is further processed by two different decoders.

(2) The core stream need to be decoded before the enhancement stream. The output from the core decoder produces a reconstructed sound of basic perceptual quality. If more layers need to be reconstructed, the output from the core decoder is upsampled and then transformed to the frequency domain.

(3) The enhancement substream is entropy decoded and inverse quantized to generate the residue signal, which is then added to the signal generated from the base-layer. After that, the combined spectral coefficients are processed by the inverse spectral processing module before they are inverse transformed into the time domain. The final output from the right enhancement path is the reconstructed sound with good perceptual quality.

More stages of refinement (enhancement layers) can be performed according to the hierarchical embedded coding scheme shown in Figure 7.13. In MPEG-4 large-step scalability tool, since the AAC kernel is adopted to provide coefficients' refinement, it is more efficient to calculate the error signals in frequency domain rather than in time domain as shown in Figures 7.14 and 7.15. In fact, the core coder does not have to be a non-AAC speech codec. The base-layer coder can be also a filterbank-based coder as the enhancement coder. For example, we can choose TwinVQ as the base codec. The advantage of choosing the same transform-based core coder as the enhancement coder is that no sampling conversion is required during the entire encoding and decoding process [47]. Thus, the coding structure of this kind of large-step scalable coder is even simpler than what is illustrated in Figures 7.14 and 7.15.

The scalable tool we described above achieves SNR scalability. Using narrowband speech codec as core coder can also achieve bandwidth scalability. Beside SNR/bandwidth scalability functionalities, the MPEG-4 scalable coder also offers a possible channel (mono-to-stereo) scalable feature, which means the core coder only takes care of the mono or downmixed signals, while the enhancement coder is capable to extend a monophonic signal into a stereophonic signal and improve the final listening effect.

Although MPEG-4 coarse-grain scalability tool is flexible in choosing different coding module, two restrictions apply in this coder design process. The first restriction is that the narrowband CELP speech coder can only be used as the core coder. In other words, no nontransform-based codec is allowed for the enhancement part. The other restriction is that TwinVQ coding kernel can only be adopted in the enhancement layer if the previous layer is also based on TwinVQ.

Large-step scalable tool is included in MPEG-4 audio as a separate profile, parallel with AAC-LC or AAC-LTP, and so forth. In addition to the standard 1024 frame size, the AAC coder in the MPEG-4 large-step scalability configuration also supports a smaller frame size of 960 so that it can better accommodate the core speech coder's frame size.

7.3.6.2. Fine-grain scalable tool

The large-step scalable audio framework is an efficient scalable configurations if only a few enhancement layers are involved. An alternative coding tool that achieves good performance for fine-grain-scalability (FGS) in MPEG-4 audio is specified in MPEG-4 audio version 2 [59]. The new codec is called BSAC [75, 98]. It is built based on the modified version of the MPEG-4 AAC tools with a new quantization and noiseless coding scheme.

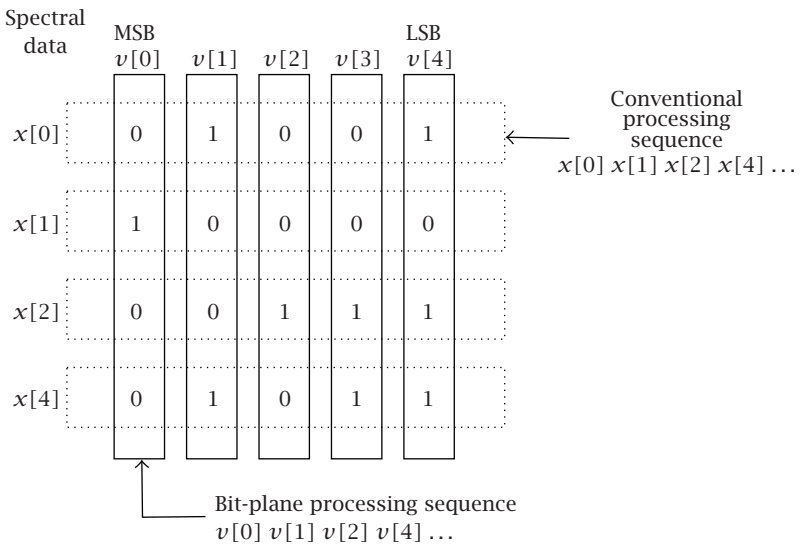


FIGURE 7.16. Conventional processing and bit-plane scanning process [100].

BSAC is capable to produce scalable bitstream in steps at approximately 1 kbps per audio channel. Since side information needs to be included in each refinement layer, the use of small steps scalability actually degrades the overall coding efficiency if full bit rate needs to be provided. BSAC handles this problem by replacing the original AAC's quantization and Huffman coding module with the bit-plane scanning and arithmetic coding. When working at high bit rates (about 48 to 64 kbps/ch), BSAC FGS achieves performance comparable to a nonscalable AAC coder with only a marginal loss in coding efficiency. At lower bit rates, the performance of the BSAC decreases progressively.

Figure 7.16 shows the difference between the conventional scanning process and the bit-plane scanning processing. Traditionally, coefficients are processed sequentially one after the other, that is, no bit information about the next coefficient is transmitted if all bit information about the previous coefficient has not finished. In bit-plane scanning processing, bits of coefficients are separated to several groups according to the bit position, that is, all the most significant bits (MSB) of all coefficients are in one group and all the least significant bits (LSB) of all coefficients are in another group, and so forth. In this manner, the coefficients' amplitude can be reconstructed more and more accurately as more and more layers are decoded. If we call the conventional scanning method as the horizontal scanning process, then we can call the bit-plane scanning method as the vertical scanning process. In the horizonal scanning processing, within each scanning path, bits are processed in the order of the bit plan, whereas in the vertical scanning process, within each scanning path, bits are processed in the sequential order. Therefore, the bit-plane coding is virtually a similar processing method as the conventional coding.

In BSAC, a sign/magnitude format is used for each quantized spectral coefficients. A maximum length of 13 bits can be used to represent the binary integers of any magnitudes (absolute values). Starting from the most significant bit (MSB) plane and continuing to the less significant bit plane, bits of magnitudes are processed in slices according to their significance in BSAC. The sign bit of a coefficient is sent following the first "1" bit encountered in the magnitude of the coefficient.

A context-based arithmetic coder is adopted to encode the vertical scanned bits. In context-based arithmetic coder, different coding models with different probability tables are provided. According to the statistics of each bit slice, the best matching coding model is selected so that a minimum redundant bitstream can be produced.

The side information, the scale factors, and the first bit slices of spectral data corresponding to a certain frequency range are included in the BSAC base-layer. Depending on the bit rate of the base-layer, reconstructed base-layer audio may have different bandwidth. As more and more numbers of enhancement layers are received by the decoder, quantized spectral data is refined by more and more LSB information, that is, more bit slices. Meanwhile, the bandwidth of the reconstructed audio is increased as more scale factors and spectral data bits in higher-frequency bands become available to the decoder. By operating in this fashion, the decoder may stop decoding almost at any bit in a bitstream and achieve a fine-grain scalability.

An optional segmented binary arithmetic (SBA) coding tool is provided in MPEG-4 audio to improve the BSAC error-resilient capability. In the error-resilient BSAC mode, the whole bitstream is composed by several segments. And in order to reduce overhead, each segment contains data from adjacent enhancement layers. The arithmetic coder is reinitialized at the beginning of each segment and terminated at its end. In this way, the potential error can be confined in the range of one segment and will not propagate across segment boundaries.

7.4. MPEG-4 high-efficiency AAC

During the March 2003 MPEG meeting, a new standard called MPEG-4 high-efficiency AAC (HE-AAC) is finalized. HE-AAC is a combination of MPEG AAC and the spectral band replication (SBR) bandwidth extension amendment. In fact, HE-AAC works as a two-layer scalable audio codec. The MPEG-AAC works as the core coder to process the lower half of the spectral coefficients, while the SBR coding module takes care of extending the bandlimited signal to the full-bandwidth by only limited additional bits. With the SBR coding tool, the HE-AAC is able to achieve the similar quality sound at a much lower bit rate. More specifically, HE-AAC can reconstruct CD-quality stereo at 48 kbps and 5.1 channel surround sound at 128 kbps.

MPEG-4 HE-AAC coder is a backward compatible extension of the previous MPEG-4 AAC standard. The HE-AAC decoder is capable to process the bitstream generated by both MPEG-4 AAC and HE-AAC encoders. On the other hand, the

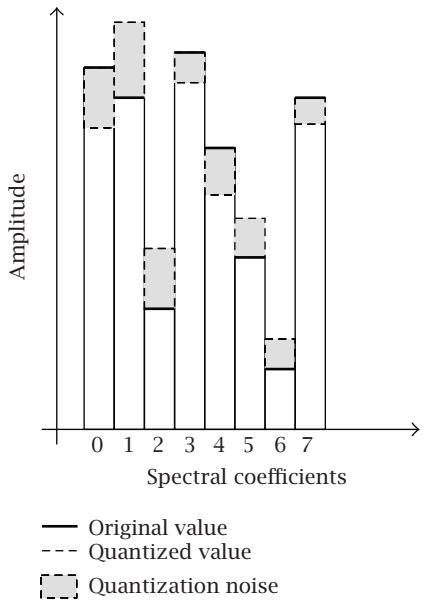


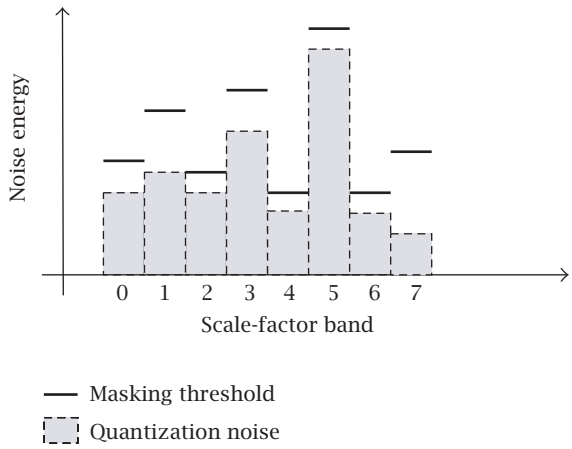
Figure 7.17. Spectral coefficients before and after quantization.

MPEG-4 AAC decoder can reconstruct the lower half-bandwidth signal in an HE-AAC bitstream. By cutting the bit rate to half of the MPEG AAC, HE-AAC is more suitable for Internet content delivery and new applications in the markets of mobile and digital broadcasting.

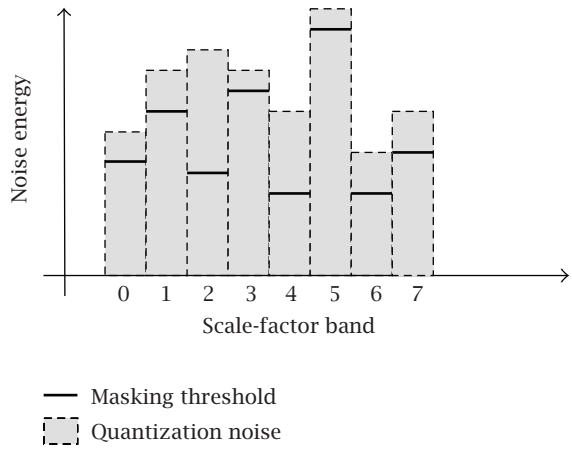
7.4.1. Background of SBR technology

The goal of the development of spectral band replication is to design a generic method so that the performance of the current perceptual audio codecs, such as MPEG layer-3 (mp3) and MPEG AAC, can be significantly enhanced.

The quantization block of any perceptual coder introduces noise, which is called quantization noise. Only when the quantization noise satisfies the psychoacoustic requirement can the perceptual coder really achieve the transparent audio quality. Figure 7.17 is an illustration of how quantization noise occurs. And Figures 7.18(a) and 7.18(b) show the relationship between the noise level and the masking threshold for two AAC encoders. Figure 7.18(a) reveals that quantization noise in all scale-factor bands has lower energy than the masking threshold. Therefore, the first AAC encoder is capable to reproduce the sound of perceptually indistinguishable quality as the original input. However, the quantization noise introduced by the second AAC encoder cannot be masked by its own signal. In other words, the energy of the noise is larger than the masking threshold, which is the typical result of a perceptual coder when the bit budget is too low and is incapable of handling the input sound transparently.



(a)



(b)

FIGURE 7.18. Quantization noise versus masking thresholds [25]. (a) Noise level is smaller than masking threshold. (b) Noise level is larger than the masking threshold.

The bitstream generated by the second AAC encoder delivers poor sound quality with noticeable artifacts. Since perceptual artifacts in the reconstructed sound is more annoying than a perceptually noise-free bandlimited sound, the typical way to solve the problem for the second AAC encoder is to downsample the input signal so that the limited bit budget can be used to encode the lower-frequency coefficients transparently. Although intensity stereo coding can also be applied to reduce the bit requirement of the original signal by combining some information in both channels, the bit reduction by intensity stereo is usually

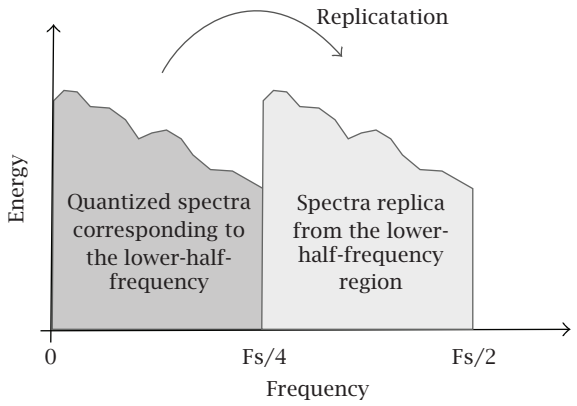


FIGURE 7.19. Generate the high-frequency component by replicating the low-frequency part [25].

insufficient and generates unsatisfactory stereo sound. Nevertheless, the spectral band replication technology tackles this problem in a more efficient way. While the core coder processes coefficients corresponding lower half of the frequency region, SBR extracts some parameters from the full-bandwidth original signal so that the decoder can generate a good quality sound of the same bandwidth as that of the original one.

7.4.2. Basic principle of SBR technology

The SBR technique is derived from the hybrid waveform/parametric method. It is observed in many cases the higher-frequency parts depend considerably on the lower-frequency parts and therefore they can be approximately reconstructed based on the information of low-frequency signal. In other words, there is no need to code and transmit the high-frequency signal as accurately as the low-frequency signal. Instead, only a small amount of SBR control data needs to be included in the bitstream so that the receiver can optimally reconstruct the high-frequency part.

Figures 7.19 and 7.20 illustrate the basic principle of SBR technology. The first step of the SBR enhancement is to replicate the lower-frequency spectra. However, the simple high-frequency generation by copying the low-frequency information is not sufficient for the accurate high-frequency reconstruction. Therefore, the second step of the SBR enhancement is to adjust the high-frequency component results from the first step. In fact, the high-frequency adjustment plays an important role in correctly shaping the high-frequency component and the SBR control information that need to be used to perform this adjustment constitutes a considerable amount of the SBR bitstream.

There are cases when there is little relationship between the low- and high-frequency parts, the above reconstruction method is not able to deliver the desired results. SBR technology takes care of this kind of situation by some additional tools and still can achieve excellent coding efficiency.

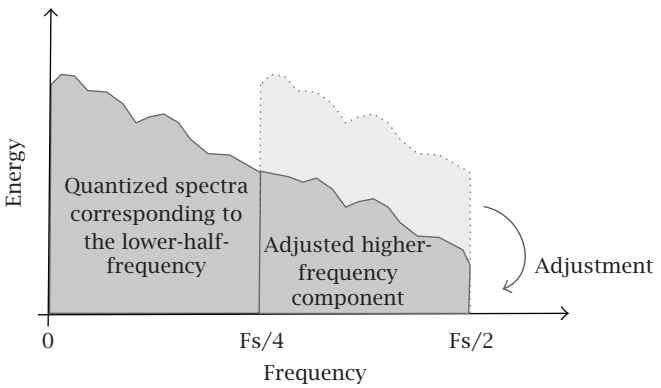


FIGURE 7.20. High-frequency component adjustment [25].

The reason that SBR is capable to significantly reduce the bit rate while still maintaining the performance of the perceptual codec can be summarized as follows.

(1) *High-frequency part.* In traditional perceptual audio codecs, the waveform coding of the high-frequency signals usually requires a considerable amount of bits in encoding. By relaxing the waveform coding requirement, the SBR technology, which is a hybrid waveform/parametric method, extracts only a small amount of side information to recreate the high-frequency component at the receiver side. Thus, a significant bit rate reduction can be achieved.

(2) *Low-frequency part.* Since the core coder only needs to handle the waveform coding of the low-frequency components, it can operate with a comparatively higher precision. In fact, the core coder can even work at a different sampling rate from the desired output sampling rate to ensure its optimal performance. This is because the SBR technology is capable of converting the sampling rate of the core codec into the desired output sampling rate.

7.4.3. More technical details on high-efficiency AAC

Figures 7.21 and 7.22 illustrate the encoder and decoder block diagrams of HE-AAC. The HE-AAC algorithm is generally to be used as a dual-rate coding system, where the core AAC codec operates at the half of the original sampling rate, and the SBR tool operates at the original sampling rate, which means the core AAC codec only takes care of handling spectral data in the lower half of the original frequency range. The spectral data in the higher half of the original frequency range are no longer waveform coded. Instead, they are synthesized by the model-based SBR technology.

SBR encoder is responsible for generating all necessary parameters so that the SBR decoder is able to reconstruct the higher-frequency part as effectively as possible. A serial of coding blocks is included in the SBR encoder. They are analysis quadratic mirror filterbank (QMF), transient detection, time and frequency

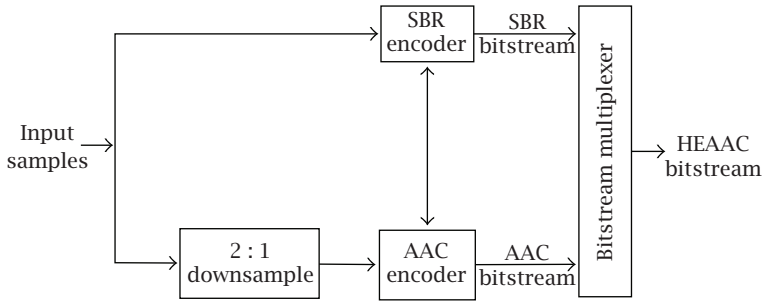


Figure 7.21. Encoder block diagram of HE-AAC.

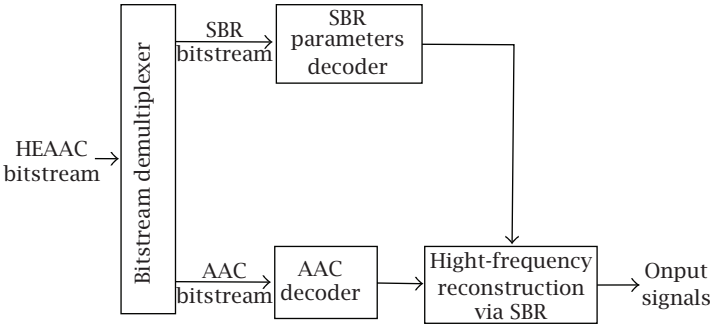


Figure 7.22. Decoder block diagram of HE-AAC.

(T/F) grid generation, envelope estimation, additional control parameters extraction and entropy coding of SBR parameters.

(i) *Analysis QMF filterbank*. The complex QMF analysis is calculated based on the original input signal and generates oversampled values. The reason that SBR encoder chooses to use a noncritically sampled QMF filterbank is to avoid aliasing at the decoder side.

(ii) *Transient detection*. Since transient signals mainly reside in the high-frequency region, while the SBR decoder generates high band signals based on the decoded lower band signals that present less transient components, the transient-related information has to be included in the SBR bitstream.

(iii) *T/F grid generation and envelope estimation*. Based on the characteristics of the current input signal segments, the SBR encoder estimates the spectral envelope for a suitable time and frequency resolution. For a given input segment, the best suited time and frequency resolutions of the spectral envelopes are selected.

(iv) *Additional control parameters extraction*. In cases when simple frequency patching and envelop adjustment cannot satisfy the coding performance, some special control parameters need to be available at the SBR decoder side. These parameters include tone-to-noise ratio, noise-floor energy, and sinusoidal component, and so forth.

(v) *Entropy coding of SBR parameters*. All parameters extracted from the previous SBR coding blocks are finally sent to the entropy coding part in order to further reduce the redundancy.

The HE-AAC decoder consists of the core AAC decoder and the SBR decoder. The AAC decoder works in the same way as MPEG-4 AAC decoder. Since only the spectral data corresponding to the lower half of the frequency range is encoded by the AAC at the encoder side, the core AAC decoder in HE-AAC decoder can only reconstruct the spectrum in low-frequency range. The corresponding higher-frequency components are synthesized by the SBR decoder.

The SBR decoder performs the following procedures after the bitstream demultiplexer outputs the SBR bitstream.

(i) *Entropy decoding*. The matching entropy decoder recovers all SBR parameters.

(ii) *High-frequency generation*. Higher-half-frequency components are first estimated by frequency patch based on decoded values of spectral data in low-frequency part.

(iii) *Envelope adjustment*. The envelope of the high-frequency part needs to be adjusted to match the high-frequency envelope of the original signal.

(iv) *Including additional high-frequency component*. Additional SBR control parameters are then used to further modify the high-frequency component so that more perceptual similar signals can be reconstructed.

(v) *QMF synthesis*. Signals generated from all previous steps are then synthesized by QMF filterbank to reconstruct the final time-domain output.

8 Introduction to new audio coding tools

8.1. Motivation and overview

Ever since the beginning of the twentieth century, the art of sound coding, transmission, recording, mixing, and reproduction has been constantly evolving. Starting from the monophonic technology, technologies on multichannel audio have been gradually extended to include stereophonic, quadraphonic, 5.1 channels, and more. Compared with traditional mono or stereo audio, multichannel audio provides end users with a more compelling experience and becomes increasingly more appealing to music producers. Consequently, an efficient coding scheme is needed for multichannel audio storage and transmission, and this subject has attracted a lot of attention recently.

Among several existing multichannel audio compression algorithms, Dolby AC-3, and MPEG advanced audio coding (AAC) are the two most prevalent perceptual digital audio coding systems. Dolby AC-3 is the third generation of digital audio compression systems from Dolby Laboratories, and has been adopted as the audio standard for high definition television (HDTV) systems. It is capable of providing indistinguishable audio quality at 384 kbit/s for 5.1 channels [8]. AAC is currently the most powerful multichannel audio coding algorithm in the MPEG family. It can support up to 48 audio channels and provide perceptually lossless audio at 320 kbit/s for 5.1 channels [18]. In general, these low-bit-rate multichannel audio compression algorithms not only utilize transform coding to remove statistical redundancy within each channel, but also take advantage of the human auditory system to hide lossy coding distortions.

8.1.1. Redundancy inherent in multichannel audio

Despite the success of AC-3 and AAC, not much effort has been made in reducing interchannel redundancy inherent in multichannel audio. The only technique used in AC-3 and AAC to eliminate redundancy across channels is called "Joint Coding," which consists of intensity/coupling and mid/side (M/S) stereo coding. Coupling is adopted based on the psychoacoustic evidence that, at high frequencies (above approximately 2 kHz), the human auditory system localizes sound primarily based on envelopes of critical-band-filtered signals that reach human ears,

rather than the signals themselves [24, 126]. M/S stereo coding is only applied to lower frequency coefficients of channel-pair-elements (CPEs). Instead of direct coding of original signals in the left and right channels, it encodes the sum and the difference of signals in two symmetric channels [15, 73].

Our experimental results show that high correlation is very likely to be present between every pair of channels besides CPE in all frequency regions, especially for those multichannel audio signals that are captured and recorded in a real space [137]. Since neither AAC nor AC-3 exploits this property to reduce redundancy, none of them can efficiently compress this kind of multichannel audio content. On the other hand, if the input multichannel audio signals presented to the encoder module have little correlation between channels, the same bit rate encoding would result in higher reconstructed audio quality. Therefore, a better compression performance can be achieved if interchannel redundancy can be effectively removed via a certain kind of transform together with redundancy removal techniques available in the existing multichannel audio coding algorithms. One possibility to reduce the cross-channel redundancy is to use interchannel prediction [36] to improve the coding performance. However, a recent study [79] argues that this kind of technique is not applicable to perceptual audio coding.

8.1.2. Quality-scalable single compressed bitstream

As the world is evolving into the information era, media compression for a pure storage purpose is far less than enough. The design of a multichannel audio codec that takes the network transmission condition into account is also important. When a multichannel audio bitstream is transmitted through a heterogeneous network to multiple end users, a quality-scalable bitstream would be much more desirable than the nonscalable one.

The quality scalability of a multichannel audio bitstream makes it possible that the entire multichannel sound can be played at various degrees of quality for end users with different receiving bandwidths. To be more precise, when a single quality-scalable bitstream is streamed to multiple users over the Internet via multicast, some lower priority packets can be dropped, and a certain portion of the bitstream can be transmitted successfully to reconstruct different quality multichannel sound according to different users' requirement or their available bandwidth. This is called the multicast streaming [134]. With nonscalable bitstreams, the server has to send different users different unicast bitstreams. This is certainly a waste of resources. Not being considered for audio delivery over heterogenous networks, the bitstream generated by most existing multichannel audio compression algorithms, such as AC-3 or AAC, is not scalable by nature.

8.1.3. Embedded multichannel audio bitstream

Similar to the quality-scalable single audio bitstream mentioned in the previous section, the most distinguishable property of an embedded multichannel audio

compression technique lies in its network transmission applications. In the scenario of audio coding, the embedded code contains all lower rate codes "embedded" at the beginning of the bitstream. In other words, bits are ordered according to their importance, and the decoder can reconstruct audio progressively. With an embedded codec, an encoder can terminate the encoding at any point, thus allowing a target rate or a distortion metric to be met exactly. Typically, some target parameters, such as the bit count, is monitored in the encoding process. When the target is met, the encoding simply stops. Similarly, given a bitstream, the decoder can cease decoding at any point and produces reconstructions corresponding to all lower-rate encodings. The property of being able to terminate the encoding or decoding of an embedded bitstream at any specific point is extremely useful in systems that are either rate- or distortion-constrained [116].

MPEG-4 version-2 audio coding supports fine grain bit rate scalability [98, 61, 65, 66, 47] in its generic audio coder (GAC). It has a bit-sliced arithmetic coding (BSAC) tool, which provides scalability in the step of 1 kbit/s per audio channel for mono or stereo audio material. Several other scalable mono or stereo audio coding algorithms [145, 128, 117] were proposed in recent years. However, not much work has been done on progressively transmitting multichannel audio sources. Most existing multichannel audio codecs, such as AAC or AC-3, can only provide fixed-bit-rate perceptually loseless coding at about 64 kbit/s/ch. In order to transfer high quality multichannel audio through a network of a time-varying bandwidth, an embedded audio compression algorithm is highly desirable.

8.1.4. Error-resilient scalable audio bitstream

Current coding techniques for high quality audio mainly focus on coding efficiency, which makes them extremely sensitive to channel errors. A few bit errors may lead to a long period of error propagation and cause catastrophic results, including making the reconstructed sound file with unacceptable perceptual quality and the crash of the decoder. The desired audio bitstream transmitted over the network should be the one with error resiliency, which means that the audio bitstream should be designed to be robust to channel errors, that is, make the error impact as small as possible. During the period when packets are corrupted or lost, the decoder should be able to perform error concealment and allow the output audio to be reproduced at acceptable quality. Compared with existing work on image, video or speech coding, the amount of work on error-resilient audio is relatively small. Techniques on robust coding and error concealment for compressed speech signals have been discussed for years. However, since speech and audio signals have different applications, straightforwardly applying these techniques to audio does not generate satisfactory results in general.

8.2. Audio coding improvements

Based on the current status of multichannel audio compression discussed in previous sections, we propose three audio coding algorithms which are all built upon

the MPEG AAC basic coding structure in this book. The first one is called modified AAC with Karhunen-Loève transform (MAACKLT). The second one is called progressive syntax-rich multichannel audio codec (PSMAC). The third one is called error-resilient scalable audio coding (ERSAC). Major contributions of this part of the book are summarized below.

8.2.1. Interchannel redundancy removal approach

As mentioned in Section 8.1.1, not much effort has been made in reducing interchannel redundancy inherent in multichannel audio sources. In our research, we carefully observe the interchannel correlation present in multichannel audio materials and propose an effective channel redundancy removal approach. Specific contributions along this direction are listed below.

(1) *Observation of channel correlation*. Based on our observation, most of the multichannel audio materials of interest exhibit two types of channel correlation. The first type only shows high correlation between CPEs, but little correlation between other channel pairs. The other type shows high correlation among all channels.

(2) *Proposal of an effective interchannel redundancy removal approach*. An interchannel decorrelation method via KLT is adopted in the preprocessing stage to remove the redundancy inherent in original multichannel audio signals. Audio channels after KLT show little correlation between channels.

(3) *Study of efficiency of the proposed interchannel decorrelation method*. Experimental results show that the KLT preprocessing approach not only significantly decorrelates input multichannel audio signals but also considerably compacts the signal energy into the first several eigen-channels. This provides strong evidence of KLT's data compaction capability. Moreover, the energy compaction efficiency increases with the number of input channels.

(4) *Frequency domain KLT versus time domain KLT*. It is observed that applying KLT to frequency domain signals achieves a better performance than directly applying KLT to time domain signals as shown by experimental results in Section 9.8. Thus, intrachannel signal decorrelation and energy compaction procedures should be performed after time-domain signals are transformed into the frequency domain via MDCT in the AAC encoder.

(5) *Temporal adaptive KLT*. A multichannel audio program is often comprised of different periods, each of which has its unique spectral signature. In order to achieve the highest information compactness, the decorrelation transform matrix must adapt to the characteristics of different periods. Thus, a temporal-adaptive KLT method is proposed, and the trade-off between the adaptive window size and the overhead bit rate is analyzed.

(6) *Eigen-channel compression*. Since signals in decorrelated eigen-channels have different characteristics from signals in original physical channels, MPEG AAC coding blocks are modified accordingly so that they are more suitable to compress audio signals in eigen-channels.

8.2.2. Audio concealment and channel transmission strategy for heterogeneous network

Based on the results of our KLT preprocessing approach, the advantage of this method is further explored. Once signals in original physical channels are transformed into independent eigen-channels, the energy accumulates much faster as the number of channels increases. This implies, when transmitting data of a fixed number of channels with our algorithm, more information content will be received in the decoder side and better quality of the reconstructed multichannel audio can be achieved. Possible channel transmission and recovery strategies for MPEG AAC and our algorithm are studied and compared. The following two contributions have been made in this work.

(1) *Channel importance sequence*. It is desirable to reorganize the bitstream such that bits of more important channels are received at the decoder side first for audio decoding. This should result in best audio quality given a fixed amount of received bits. According to the channel energy and, at the same time, considering the sound effect caused by different channels, the channel importance for both original physical channels and KL-transformed eigen-channels is studied. A channel transmission strategy is determined according to this channel importance criterion.

(2) *Audio concealment and channel scalable decoding*. When packets belonging to less important channels are dropped, an audio concealment strategy must be enforced in order to reconstruct a full multichannel audio. A channel-scalable decoding method based on this audio concealment strategy for bitstreams generated by AAC and the proposed algorithm is proposed. Experimental results show that our algorithm has a much better scalable capability and can reconstruct multichannel audio of better quality, especially at lower bit rates.

8.2.3. Quantization efficiency for adaptive Karhunen-Loève transform

The quantization method for the Karhunen-Loève transform matrix and the discussion on the temporal adaptive KLT method in the MAACKLT algorithm are relatively simple. Some questions arise with respect to improving the efficiency of the quantization scheme. For example, can we improve the coding performance by reducing the overhead involved in transmitting the KLT matrix? If the number of bits required to quantize each KLT matrix is minimized, can we achieve much better interchannel decorrelation efficiency if the KLT matrix is updated much more frequently? Having these questions in mind, we investigate the impact of different quantization methods and their efficiency on the adaptive Karhunen-Loève transform. The following two areas are addressed in this research.

(1) *Scalar quantizer versus the vector quantizer*. The coding efficiency, the bit requirement of the scalar quantizer, and the vector quantizer are carefully analyzed. Although a scalar quantizer applied to the KLT matrix gives much better interchannel decorrelation efficiency, vector quantization methods that have

a smaller bit requirement achieve a better performance in terms of the final MNR values of the reconstructed sound file.

(2) *Long versus short temporal adaptive period for KLT*. We study how to choose a moderately long temporal adaptive period so that the optimal trade-off between the interchannel decorrelation efficiency and the overhead bit rate can be achieved.

8.2.4. Progressive syntax-rich multichannel audio codec design

Inspired by progressive image coding and the MPEG AAC system, a novel progressive syntax-rich multichannel audio compression algorithm is proposed in this book. The distinctive feature of the multichannel audio bitstream generated by our embedded algorithm is that it can be truncated at any point and still reconstruct a full multichannel audio, which is extremely desirable in the network transmission. The novelty of this algorithm includes the following.

(1) *Subband selection strategy*. A subband selection strategy based on the mask-to-noise ratio (MNR) is proposed. An empirical MNR threshold is used to determine the importance of a subband in each channel so that the most sensitive frequency region can be reconstructed first.

(2) *Layered coefficient coding*. A dual-threshold strategy is adopted in our implementation. At each layer, the MNR threshold is used to determine the subband significance, the coefficient magnitude threshold is used to determine coefficient significance. According to these selection criteria, within each selected subband, transformed coefficients are layered quantized and transmitted into the bitstream so that a coarse-to-fine multiprecision representation of these coefficients can be achieved at the decoder side.

(3) *Multiple context lossless coding*. A context-based QM coder is used in the lossless coding part in the proposed algorithm. Six classes of contexts are carefully selected in order to increase the coding performance of the QM coder.

(4) *Three user-defined profiles*. Three user-defined profiles are designed in this codec. They are MNR progressive, random access, and channel enhancement. With these profiles, PSMAC algorithm provides end users versatile functionalities.

8.2.5. Error-resilient scalable audio coding

In order to improve the performance of the PSMAC algorithm when its bitstream is transmitted over erroneous channels, we extend its error-free codec to an error-resilient scalable audio coding (ERSAC) over WCDMA channels by reorganizing the bitstream and modifying the noiseless coding part. The distinctive features of the ERSAC algorithm are presented below.

(1) *Unequal error protection*. Compared with the equal error protection scheme, the unequal error protection method gives higher priority to critical bits and therefore better protection. It offers an improved perceived signal quality at the same channel signal-to-noise ratio.

(2) *Adaptive segmentation*. In order to minimize the error propagation effect, the bitstream is dynamically segmented into several variable length segments such that it can be resynchronized at the beginning of each segment even when error happens. Within each segment, bits can be independently decoded. In this way, errors can be confined to one segment, and will not propagate and affect the decoding of neighboring segments.

(3) *Frequency interleaving*. To further improve the error-resilience, bits belonging to the same time period but different frequency region are divided into two groups and sent in different packets. Therefore, even when the packets that contain bits for the header or the data part of the base layer are corrupted, not all information for the same time position is lost so that the end user is still able to reconstruct a poorer version of the sound with some frequency component missing.

9 Interchannel redundancy removal and channel-scalable decoding

9.1. Introduction

In this chapter,[1] we present a new algorithm called MAACKLT, which stands for modified advanced audio coding with Karhunen-Loève transform (KLT). In MAACKLT, a 1D temporal-adaptive KLT is applied in the preprocessing stage to remove interchannel redundancy. Then, decorrelated signals in the KL transformed channels, called eigen-channels, are compressed by a modified AAC main profile encoder module. Finally, a prioritized eigen-channel transmission policy is enforced to achieve quality scalability.

In this work, we show that the proposed MAACKLT algorithm provides a coarse-grain scalable audio solution. That is, even if packets of some eigen-channels are dropped completely, a slightly degraded yet full-channel audio can still be reconstructed in a reasonable fashion without any additional computational cost.

To summarize, we focus on two issues in this research. First, the proposed MAACKLT algorithm exploits interchannel correlation existing in audio data to achieve a better coding gain. Second, it provides a quality-scalable multichannel audio bitstream which can be adaptive to networks of time-varying bandwidth. The rest of this chapter is organized as follows. Section 9.2 summarizes the interchannel decorrelation scheme and its efficiency. Section 9.3 discusses the temporal adaptive approach. Section 9.4 describes the eigen-channel coding method and its selective transmission policy. Section 9.5 demonstrates the audio concealment strategy at the decoder end when the bitstream is partially received. The system overview of the complete MAACKLT compression algorithm is provided in Section 9.6. The computational complexity of MAACKLT is compared with that of MPEG AAC in Section 9.7. Experimental results are shown in Section 9.8. Finally, concluding remarks are given in Section 9.9.

9.2. Interchannel redundancy removal

9.2.1. Karhunen-Loève transform

For a given time instance, removing interchannel redundancy would result in a significant bandwidth reduction. This can be done via an orthogonal transform

[1] Part of this chapter represents works published before, see [137, 136, 142].

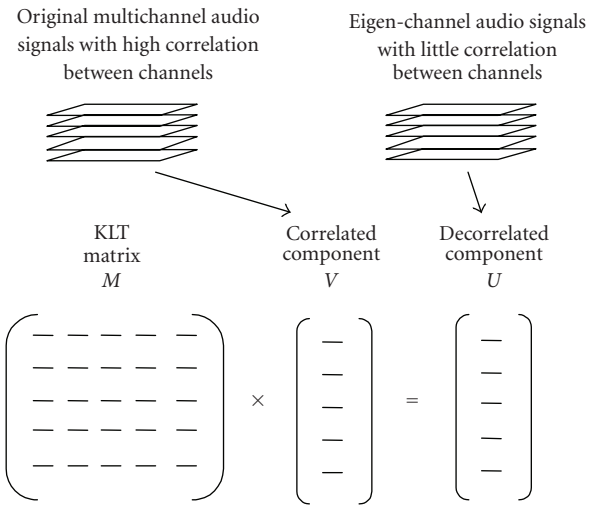


FIGURE 9.1. Interchannel decorrelation via KLT.

$MV = U$, where V and U denote the vector whose n elements are samples in original and transformed channels, respectively. Among several commonly used transforms, including the discrete cosine transform (DCT), the fourier transform (FT), and the Karhunen-Loève transform (KLT), the signal-dependent KLT is adopted in the preprocessing stage because it is theoretically optimal in decorrelating signals across channels. If M is the KLT matrix, the transformed channels are called the eigen-channels. Figure 9.1 illustrates how KLT is performed on multichannel audio signals, where the columns of the KLT matrix is composed of eigenvectors calculated from the covariance matrix C_V associated with original multichannel audio signals V.

Suppose that an input audio signal has n channels. Then, we can form an $n \times n$ KLT matrix M consisting of n eigenvectors of the cross-covariance matrix associated with these n channels. Let $V(i)$ denote the vector whose n elements are the ith sample value in channel $1, 2, \ldots, n$, that is ,

$$V(i) = [x_1, x_2, \ldots, x_n]^T, \quad i = 1, 2, \ldots, k, \tag{9.1}$$

where x_j is the ith sample value in channel j $(1 \le j \le n)$, k represents the number of samples in each channel, and $[*]^T$ represents the transpose of $[*]$. The mean vector μ_V and covariance matrix C_V are defined as

$$\begin{aligned} \mu_V &= E[V] = \frac{\sum_{i=1}^{k} V(i)}{k}, \\ C_V &= E\left[(V - \mu_V)(V - \mu_V)^T\right] = \frac{\sum_{i=1}^{k} [V(i) - \mu_V][V(i) - \mu_V]^T}{k}. \end{aligned} \tag{9.2}$$

The KLT matrix M is $M = [m_1, m_2, \ldots, m_n]^T$, where $m_1, m_2, \ldots, m_n$ are eigenvectors of C_V. The covariance of KLT signals is

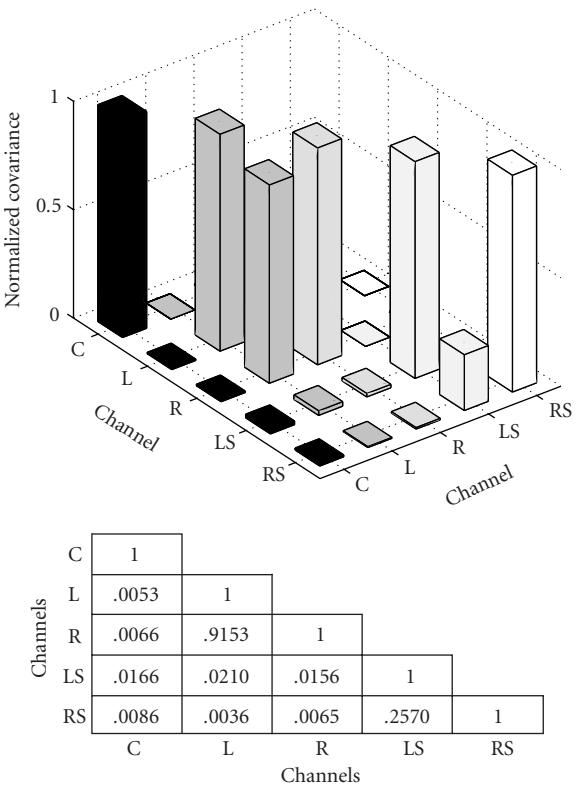
$$\begin{aligned} E[(U-\mu_U)(U-\mu_U)^T] &= E[(MV - M\mu_V)(MV - M\mu_V)^T] \\ &= ME[(V-\mu_V)(V-\mu_V)^T]M^T \\ &= MC_VM^T \\ &= \begin{bmatrix} \lambda_1 & 0 & \cdots & 0 \\ 0 & \lambda_2 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_n \end{bmatrix}, \end{aligned} \tag{9.3}$$

where $\lambda_1, \lambda_2, \ldots, \lambda_n$ are eigenvalues of C_V. Thus, the transform produces statistically decorrelated channels in the sense of having a diagonal covariance matrix for transformed signals. Another property of KLT, which can be used in the reconstruction of audio of original channels, is that the inverse transform matrix of M is equal to its transpose. Since C_V is real and symmetric, the matrix is formed by normalized eigenvectors which are orthonormal. Therefore, we have $V = M^T U$ in reconstruction. From KL expansion theory [46], we know that selecting eigenvectors associated with the largest eigenvalues can minimize the error between original and reconstructed channels. This error becomes zero if all eigenvectors are used. KLT is thus optimum in the least-square-error sense.

9.2.2. Evidence for interchannel decorrelation

Multichannel audio sources can be roughly classified into three categories. Those belonging to class I are mostly used in broadcasting, where signals in one channel may be completely different from the other. Either broadcasting programs are different from channel to channel, or the same program is broadcast but in different languages. Samples of audio sources in class I normally contain relatively independent signals in each channel and present little correlation among channels. Therefore, this type of audio source will not fall into the scope of high quality multichannel audio compression discussed here.

The second class of multichannel audio sources can be found in most film soundtracks, which are typically in the format of 5.1 channels. Most of this kind of program material has a symmetry property among CPEs and presents high correlation in CPEs, but little correlation across CPEs and single-channel elements (SCEs). Almost all existing multichannel audio compression algorithms such as AAC and Dolby AC-3 are mainly designed to encode audio material that belongs to this category. Figure 9.2 shows the normalized covariance matrix generated from one sample audio of class II, where the normalized covariance matrix is derived from the cross-covariance matrix by multiplying each coefficient with the



Channels	C	L	R	LS	RS
C	1				
L	.0053	1			
R	.0066	.9153	1		
LS	.0166	.0210	.0156	1	
RS	.0086	.0036	.0065	.2570	1

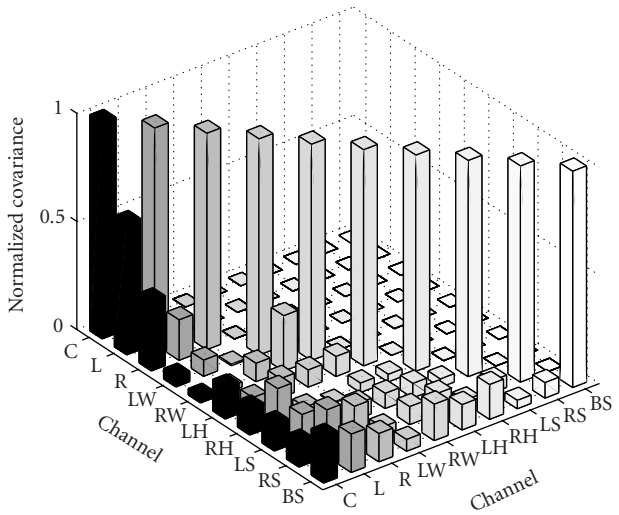
Channels

FIGURE 9.2. Absolute values of elements in the lower triangular normalized covariance matrix for five-channel "Herre."

reciprocal of the square root of the product of their individual variance. Since the magnitude of nondiagonal elements in a normalized covariance matrix provides a convenient and useful measure for the degree of interchannel redundancy, it is used as a correlation metric throughout the chapter.

A third emerging class of multichannel audio sources consists of material recorded in a real space with multiple microphones that capture acoustical characteristics of that space. Audio of class III is becoming more prevalent with the introduction of consumer media such as DVD-audio. This type of audio signals has considerably larger redundancy inherent among channels especially adjacent channels as graphically shown in Figure 9.3, which corresponds to the normalized covariance matrix derived from a test sequence named "Messiah." As shown in the figure, a large degree of correlation is presented between not only CPEs (e.g., left/right channel pair and left-surround/right-surround channel pair) but also SCE (e.g., the center channel) and any other channels.

The work presented in this research will focus on improving the compression performance for multichannel audio sources that belong to classes II and III. It will be demonstrated that the proposed MAACKLT algorithm not only achieves



Channels	C	L	R	LW	RW	LH	RH	LS	RS	BS
C	1									
L	.6067	1								
R	.3231	.1873	1							
LW	.0484	.0745	.0064	1						
RW	.0298	.0236	.0854	.2564	1					
LH	.1464	.0493	.0887	.0791	.0921	1				
RH	.1235	.1655	.0310	.0031	.0439	.0356	1			
LS	.1260	.1189	.0407	.0724	.0705	.0671	.0384	1		
RS	.1148	.2154	.1827	.0375	.0826	.0952	.0147	.0570	1	
BS	.2156	.1773	.1305	.0540	.1686	.1173	.1606	.0383	.0714	1

Channels

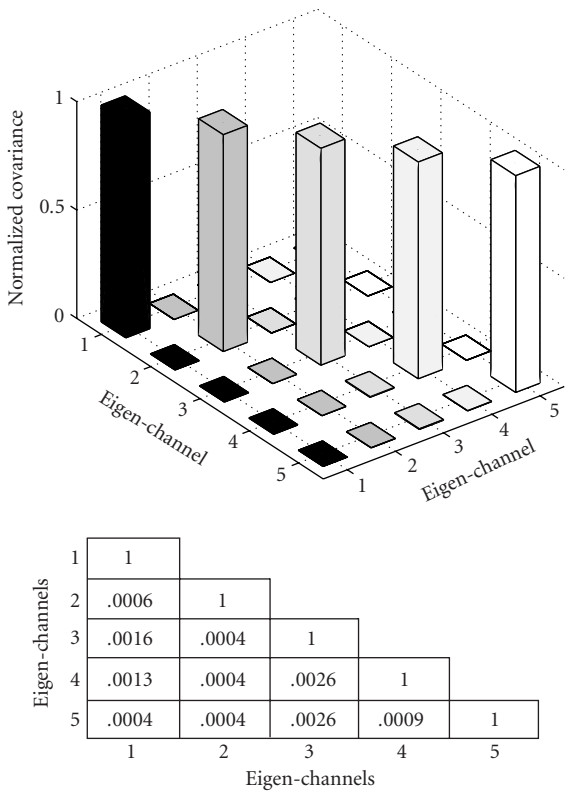
FIGURE 9.3. Absolute values of elements in the lower triangular normalized covariance matrix for ten-channel "Messiah."

good results for class III audio sources, but also improves the coding performance to a certain extent for class II audio sources compared with original AAC.

Two test data sets are used to illustrate the decorrelation effect of KLT. One is a class III ten-channel audio piece called "Messiah."[2] It is a piece of classical music recorded live in a concert hall. Another one is a class II five-channel audio piece called "Herre,"[3] which is a piece of pop music and was used in MPEG-2 AAC standard (ISO/IEC 13818-7) conformance work. These test sequences are chosen because they contain a diverse range of frequency components played by several

[2]The ten channels include center (C), left (L), right (R), left wide (LW), right wide (RW), left high (LH), right high (RH), left surround (LS), right surround (RS), and back surround (BS). They were obtained by mixing signals from 16 microphones placed in various locations in a concert hall.

[3]The five channels include C, L, R, LS, and RS.



Eigen-channels	1	2	3	4	5
1	1				
2	.0006	1			
3	.0016	.0004	1		
4	.0013	.0004	.0026	1	
5	.0004	.0004	.0026	.0009	1

Eigen-channels

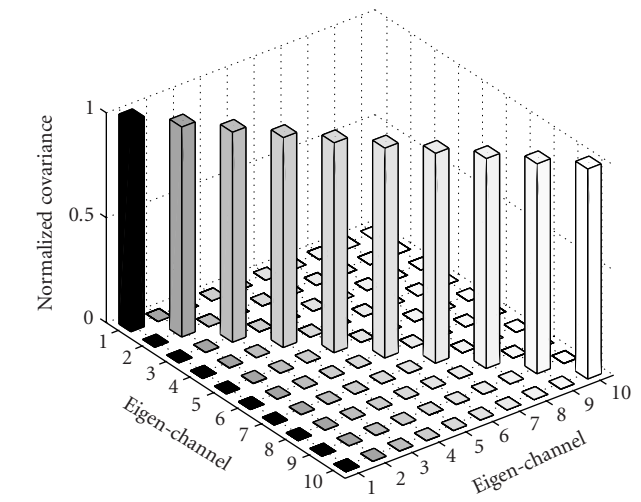
FIGURE 9.4. Absolute values of elements in the lower triangular normalized covariance matrix after KLT for five-channel "Herre."

different instruments so that they are very challenging for interchannel decorrelation and subsequent coding experiments. In addition, they provide good samples for result comparison between original AAC and the proposed MAACKLT algorithm.

Figures 9.4 and 9.5 show absolute values of elements in the lower triangular part of the normalized cross-covariance matrix after KLT for five-channel set "Herre" and ten-channel set "Messiah." These figures clearly indicate that KLT method achieves a high degree of decorrelation. Note that the nondiagonal elements are not exactly zeros because we are dealing with an approximation of KLT during calculation. We predict that by removing redundancy in the input audio with KLT, a much better coding performance can be achieved when encoding each channel independently, which will be verified in later sections.

9.2.3. Energy compaction effect

The KLT preprocessing approach not only significantly decorrelates the input multichannel audio signals but also considerably compacts the signal energy into the



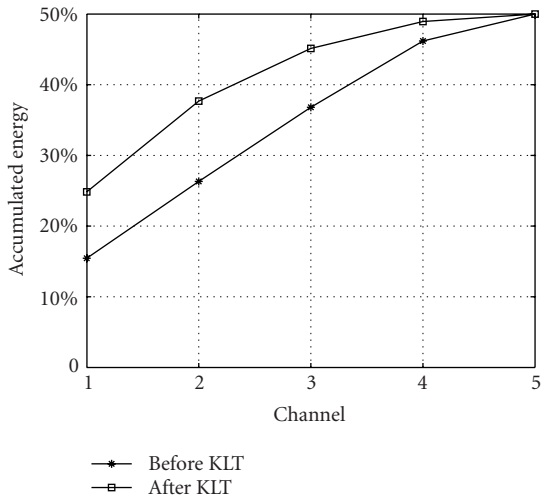
Eigen-channels	1	2	3	4	5	6	7	8	9	10
1	1									
2	.0012	1								
3	.0006	.0009	1							
4	.0001	.0014	.0010	1						
5	.0004	.0019	.0014	.0003	1					
6	.0001	.0007	.0006	.0005	.0005	1				
7	.0005	.0017	.0012	.0000	.0005	.0011	1			
8	.0004	.0016	.0009	.0002	.0005	.0007	.0005	1		
9	.0002	.0009	.0008	.0004	.0001	.0006	.0001	.0007	1	
10	.0001	.0013	.0008	.0005	.0002	.0010	.0001	.0001	.0010	1

Eigen-channels

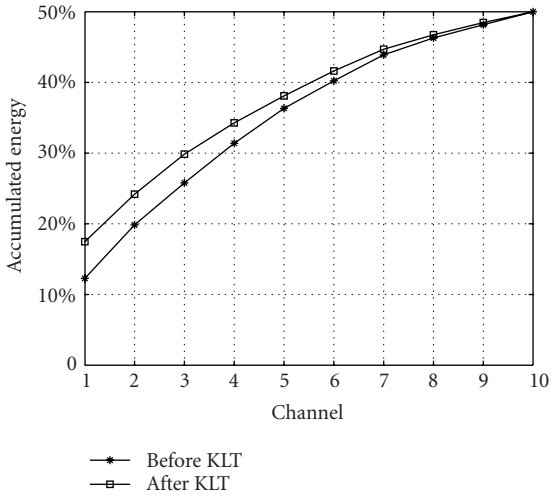
FIGURE 9.5. Absolute values of elements in the lower triangular normalized covariance matrix after KLT for ten-channel "Messiah."

first several eigen-channels. Figures 9.6(a) and (b) show how energy is accumulated with an increased number of channels for original audio channels and decorrelated eigen-channels. As clearly shown in these two figures, energy accumulates much faster in the case of eigen-channels than original channels, which provides a strong evidence of data compaction of KLT. It implies that, when transmitting data of a fixed number of channels with the proposed MAACKLT algorithm, more information content will be received at the decoder side, and better quality of reconstructed multichannel audio can be achieved.

Another convenient way to measure the amount of data compaction can be obtained via eigenvalues of the cross-covariance matrix associated with the KL transformed data. In fact, these eigenvalues are nothing else but variances of eigen-channels, and the variance of a set of signals reflects its degree of jitter, or the information content. Figures 9.7(a) and (b) are plots of variances of eigen-channels associated with the "Messiah" test set consisting of ten and five channels, respectively.



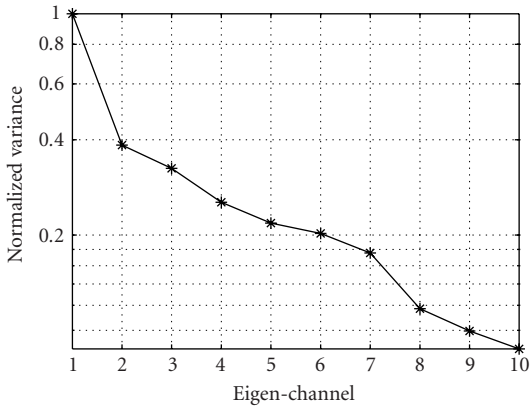
(a)



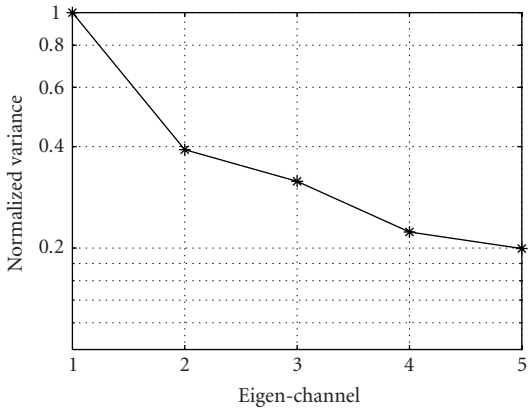
(b)

FIGURE 9.6. Comparison of accumulated energy distribution for (a) five-channel "Herre" and (b) ten-channel "Messiah."

As shown in the figures, the variance drops dramatically with the order of eigenchannels. The steeper the variance drop is, the more efficient is the energy compaction achieved. These experimental results also show that the energy compaction efficiency increases with the number of input channels. The area under the variance curve reflects the amount of information to be encoded. As illustrated from



(a)



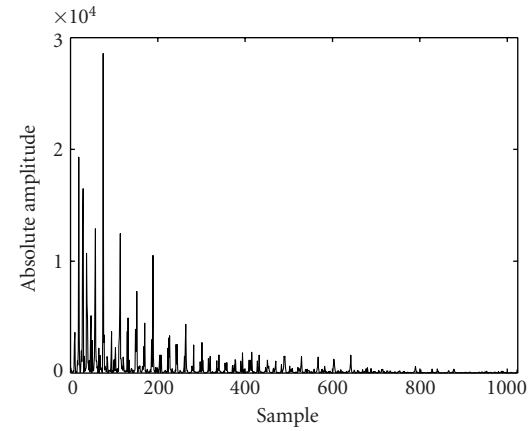
(b)

FIGURE 9.7. Normalized variances for (a) ten-channel "Messiah" and (b) five-channel "Messiah," where the vertical axis is plotted in the log scale.

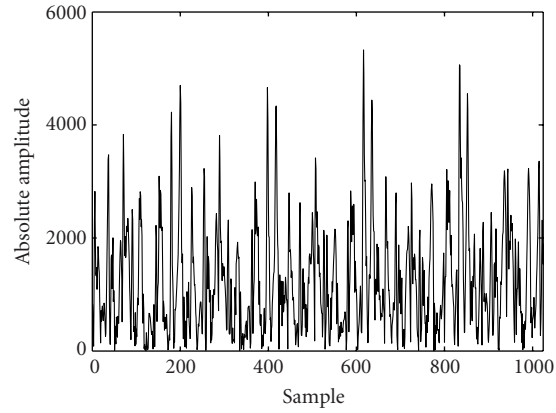
these two figures, this particular area is substantially much smaller for the ten-channel set than that of the five-channel set. As the number of input channels decreases, the final compression performance of MAACKLT tends to be more influenced by the coding power of the AAC main profile encoder.

9.2.4. Frequency-domain versus time-domain KLT

In all previous discussions, we considered only the case of applying KLT to time-domain signals across channels. However, it is also possible to apply the interchannel decorrelation procedure after time-domain signals are transformed into the



(a)



(b)

FIGURE 9.8. (a) Frequency-domain and (b) time-domain representations of the center channel from "Herre."

frequency-domain via modified discrete cosine transform (MDCT) in the AAC encoder.

One frame of the audio signal from the center channel of "Herre" in the frequency-domain and in the time-domain are shown in Figures 9.8(a) and (b), respectively. The energy compaction property can be clearly seen from the simple comparison between the time-domain and the frequency-domain plots. Generally speaking, applying KLT to frequency-domain signals achieves a better performance than directly applying KLT to time-domain signals. In addition, a certain degree of delay and reverberant sound copies may exist in time-domain signals among different channels, which is especially true for class III multichannel

audio sources. The delay and reverberation effects affect the time-domain KLT's decorrelation capability, however, they may not have that much impact on frequency-domain signals. Figures 9.9 and 9.10 show absolute values of off-diagonal nonredundant elements for normalized covariance matrices generated from frequency- and time-domain KLTs with test audio "Herre" and "Messiah," respectively. Clearly, the frequency-domain KLT has a much better interchannel decorrelation capability than that of the time-domain KLT. This implies that applying KLT to frequency-domain signals should lead to a better coding performance, which will be verified by experimental results shown in Section 9.8. Any result discussed hereafter will focus on frequency-domain KLT method unless otherwise mentioned.

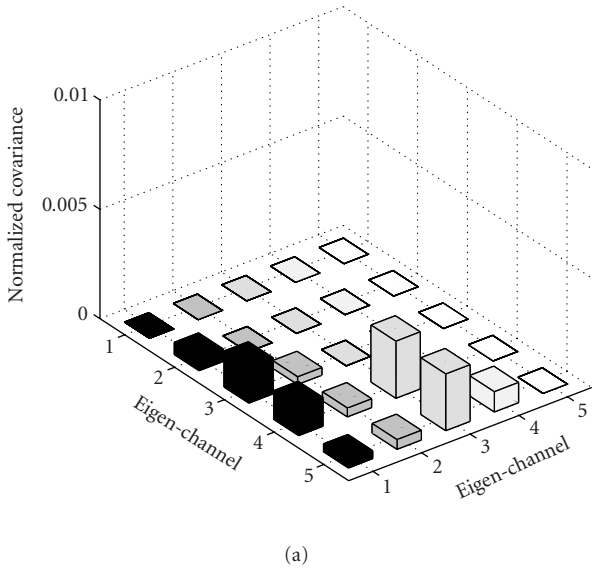
9.3. Temporal adaptive KLT

A multichannel audio program may comprise different periods, each of which has its unique spectral signature. For example, a piece of music may begin with a piano prelude followed by a chorus. In order to achieve the highest information compactness, the decorrelation transform matrix should be adaptive to the characteristics of different periods. In this section, we present a temporal-adaptive KLT approach, in which the covariance matrix (and, consequently, the corresponding KLT matrix) is updated from time to time. Each adaptation period is called a block.

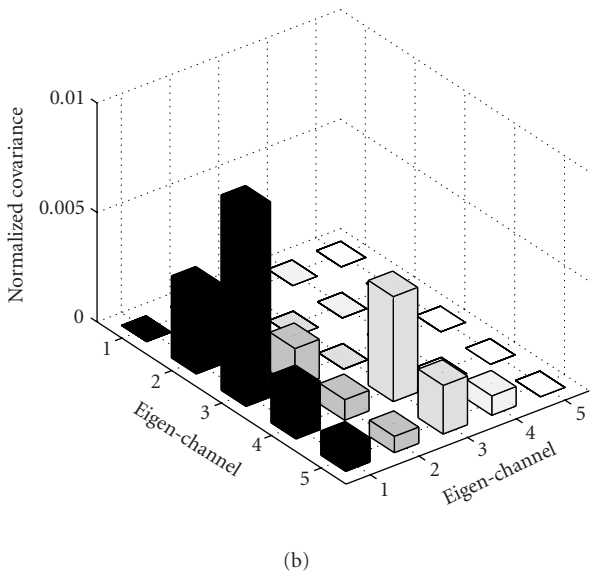
Figure 9.11 shows the variance of each eigen-channel of one nonadaptive and two temporal-adaptive approaches for test set "Messiah." Compared with the nonadaptive method, the adaptive method achieves a smaller variance for each eigen-channel. Furthermore, the shorter the adaptation period, the higher is the interchannel decorrelation achieved. The only drawback of the temporal-adaptive approach over the nonadaptive approach goes to the overhead bits, which have to be transmitted to the decoder so that the multichannel audio can be reconstructed to its original physical channels. Due to the increase of the block number, the shorter the adaptation period, the larger is the overhead bit rate. The trade-off between this block size and the overhead bit rate will be discussed below.

Since the inverse KLT has to be performed at the decoder side, the information of the transform matrix should be included in the coded bitstream. As mentioned before, the inverse KLT matrix is the transpose of the forward KLT matrix, which is composed by eigenvectors of the cross-covariance matrix. To reduce the overhead bit rate, elements of the covariance matrix are included in the bitstream instead of those of the KLT matrix since the covariance matrix is real and symmetric and we only have to send the lower (or higher) triangular part that contains nonredundant elements. As a result, the decoder also has to calculate eigenvectors of the covariance matrix before the inverse KLT can be performed.

Only one covariance matrix has to be coded for the nontemporal-adaptive approach. However, for the temporal-adaptive approach, every covariance matrix must be coded for each block. Assume that n channels are selected for simultaneous interchannel decorrelation, and that the adaptation period is K seconds, that



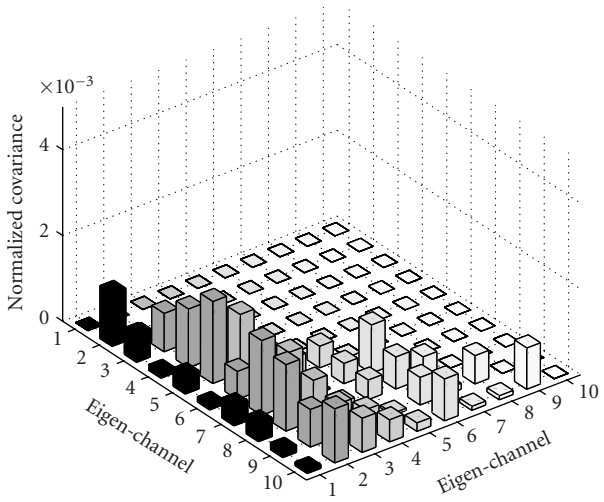
(a)



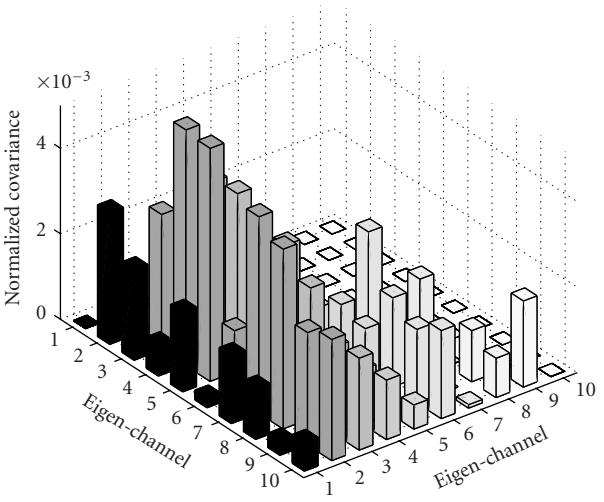
(b)

FIGURE 9.9. Absolute values of off-diagonal elements for the normalized covariance matrix after (a) frequency-domain and (b) time-domain KLTs with test audio "Herre."

is, each block contains K seconds of audio. The size of the covariance matrix is $n \times n$, and the number of nonredundant elements is $n \times (n+1)/2$. In order to reduce the overhead bit rate, the floating-point covariance matrix is quantized to 16 bits per element. Therefore, the total bit requirement for each covariance matrix is



(a)



(b)

FIGURE 9.10. Absolute values of off-diagonal elements for the normalized covariance matrix after (a) frequency-domain and (b) time-domain KLTs with test audio "Messiah."

$8n \times (n+1)$ bits, and the overhead bit rate r_{overhead} is

$$r_{\text{overhead}} = \frac{8n \times (n+1)}{nK} = \frac{8(n+1)}{K} \tag{9.4}$$

in bit per second per channel (bit/s/ch). The above equation suggests that the

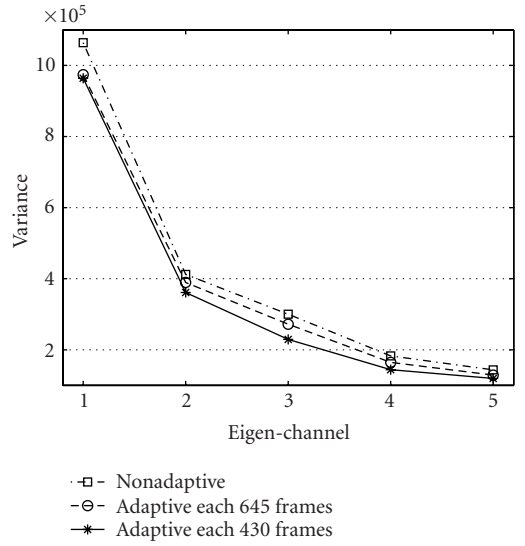


FIGURE 9.11. Decorrelation efficiency of temporal adaptive KLT.

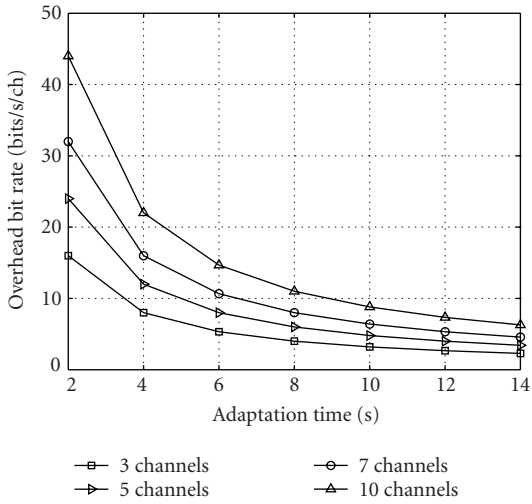


FIGURE 9.12. The overhead bit rate versus the number of channels and the adaptation period.

overhead bit rate increases approximately linearly with the number of channels. The overhead bit rate is, however, inversely proportional to the adaptation time (or the block size).

Figure 9.12 illustrates the overhead bit rate for different channel numbers and block sizes. The optimal adaptation time is around 10 seconds, since shorter adaptation time dramatically increases the overhead bit rate. Extensive experimental

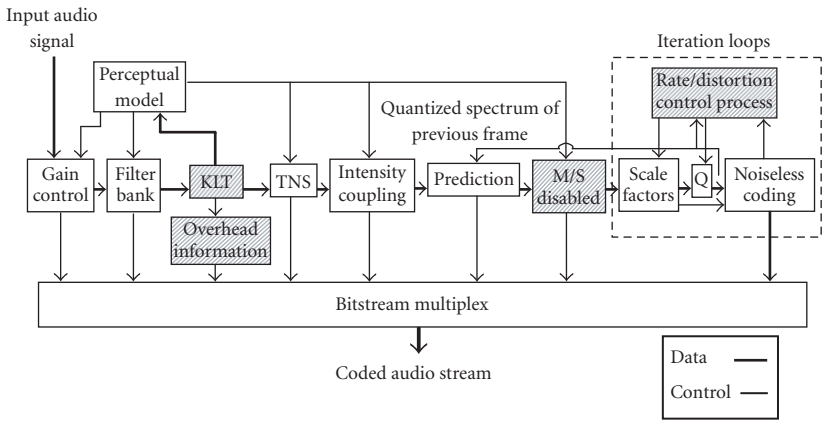


Figure 9.13. The modified AAC encoder block diagram.

results [138] suggest that, when shorter adaptation time is adopted, the improvement of decorrelation efficiency is not sufficient to compensate for coding performance degradation due to the excessive overhead bit rate.

9.4. Eigen-channel coding and transmission

9.4.1. Eigen-channel coding

The main profile of the AAC encoder is modified to compress audio signals in decorrelated eigen-channels. The detailed encoder block diagram is given in Figure 9.13, where the shaded parts represent coder blocks that are different from the original AAC algorithm.

The major difference between Figure 9.13 and the original AAC encoder block diagram is the KLT block added after the filter bank. When the original input signals are transformed into frequency domain, the cross-channel KLT is performed to generate the decorrelated eigen-channel signals. Masking thresholds are then calculated based on the KL transformed signals in the perceptual model. The KLT related overhead information is sent into the bitstream afterwards.

The original AAC is typically used to compress class II audio sources. Its M/S stereo coding block is specifically used for symmetric CPEs. It encodes the mean and difference of CPEs instead of two independent SCEs, which reduces redundancy existing in symmetric channel pairs. In the proposed algorithm, since interchannel decorrelation has been performed at an earlier stage and audio signals after KLT are from independent eigen-channels with little correlation between any channel pairs, the M/S coding block is no longer needed. Thus, the M/S coding block of the AAC main profile encoder is disabled.

The AAC encoder module originally assigns an equal amount of bits to each input channel. However, since signals into the iteration loops are no longer the original multichannel audio in the new system, the optimality of the same strategy

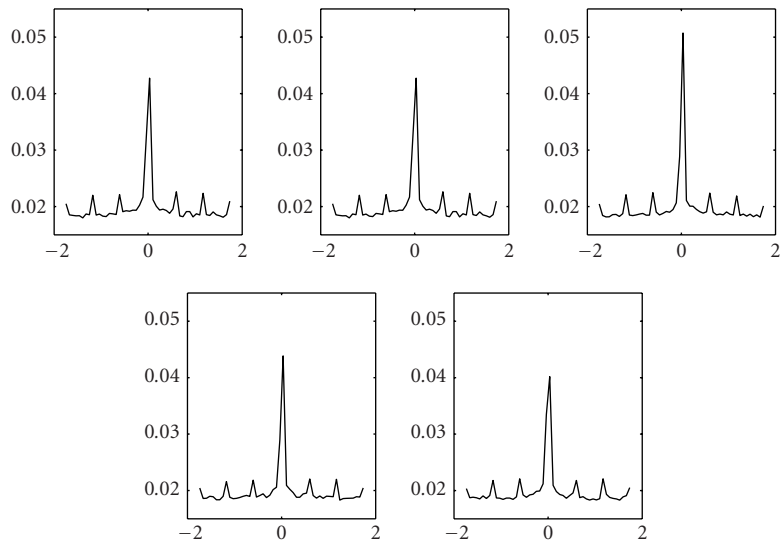


FIGURE 9.14. The empirical probability density functions of normalized signals in five eigen-channels generated from test audio "Herre," where X axes represent the value of normalized random variable and Y axes represent the corresponding probability.

has to be investigated. Experimental results indicate that the compression performance will be strongly influenced by the bit assignment scheme for decorrelated eigen-channels.

According to the bit allocation theory [38], the optimal bit assignment for identically distributed normalized random variables under the high rate approximations while without nonnegativity or integer constraints on the bit allocations is

$$b_i = \bar{b} + \frac{1}{2}\log_2 \frac{\sigma_i^2}{\rho^2}, \tag{9.5}$$

where $\bar{b} = B/k$ is the average number of bits per parameter, k is the number of parameters, and $\rho^2 = (\prod_{i=1}^{k} \sigma_i^2)^{1/k}$ is the geometric mean of the variances of the random variables. A normalized random variable of X is

$$\bar{X} = \frac{X - E(X)}{\mathrm{std}(X)}, \tag{9.6}$$

where $E(X)$ and $\mathrm{std}(X)$ represent the mean and the standard deviation of X, respectively. It is verified by experimental data that the normalized probability density functions of signals in eigen-channels are almost identical. They are given in Figures 9.14 and 9.15. This optimal bit allocation method is adopted for rate/distortion control processing when encoding eigen-channel signals.

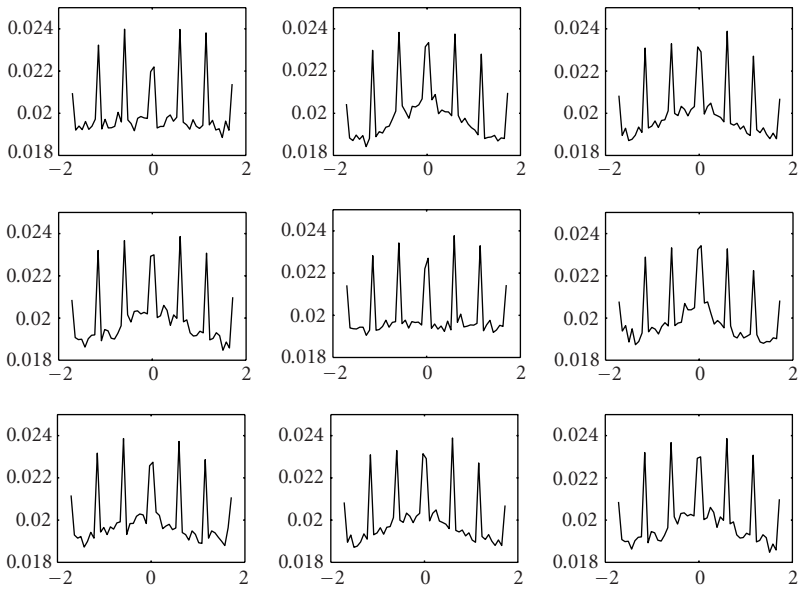


FIGURE 9.15. The empirical probability density functions of normalized signals in the first nine eigen-channels generated from test audio "Messiah," where X axes represent the value of normalized random variable and Y axes represent the corresponding probability.

9.4.2. Eigen-channel transmission

Figures 9.6(a) and (b) show that the signal energy accumulates faster in eigen-channel form than original multichannel form. This implies that, with a proper channel transmission and recovery strategy, as well as transmitting the same number of eigen-channels and original multichannels, the eigen-channel approach should result in a higher quality reconstructed audio since more energy is transmitted.

It is desirable to reorganize the bitstream so that bits of more important channels can be received at the decoder side first for audio decoding. This should result in the best audio quality given a fixed amount of received bits. When this reorganized audio bitstream is transmitted over a heterogeneous network, for those users with a limited bandwidth, the network can drop packets belonging to less important channels.

The first instinct about the metric of channel importance would be the energy of the audio signal in each channel. However, this metric does not work well in general. For example, for some multichannel audio sources, especially those belonging to class II; since those are reproduced in a music studio artificially, the side channel which normally does not contain the main melody may even have larger energy than the center channel. Based on our experience with multichannel audio, loss or significant distortion of the main melody in the center channel would be much more annoying than loss of melodies in side channels. In other words,

the location of channels also plays an important role. Therefore, for a regular 5.1 channel configuration, the order of channel importance from that of highest to lowest importance should be:

(1) center channel,
(2) L/R channel pair,
(3) LS/RS channel pair,
(4) low frequency channel.

Between channel pairs, their importance can be determined by their energy values. This rule is adopted in experiments below.

After KLT, eigen-channels are no longer the original physical channels, and sounds in different physical channels are mixed in every eigen-channel. Thus, spatial dependency of eigen-channels is less trivial. We observe from experiments that although it is true that one eigen-channel may contain sounds from more than one original physical channel, there still exists a close correspondence between eigen-channels and physical channels. To be more precise, audio of eigen-channel one would sound similar to that of the center channel, audio of eigen-channels two and three would sound similar to that of the L/R channel pair and so forth. Therefore, if eigen-channel one is lost in transmission, we would end up with a very distorted center channel. Moreover, it happens that, sometimes, eigen-channel one may not be the channel with a very large energy and could be easily discarded if the channel energy is adopted as the metric of channel importance. Thus, the channel importance of eigen-channels should be similar to that of physical channels. That is, eigen-channel one should correspond to the center channel, eigen-channel two and three should correspond to the L/R channel pair, and eigen-channel four and five should correspond to the LS/RS channel pair. Within each channel pair, the importance is still determined by their energy values.

9.5. Audio concealment for channel-scalable decoding

Consider the scenario that an AAC-coded multichannel bitstream is transmitted in a heterogeneous network such as the Internet. For end users that do not have enough bandwidth to receive full channel audio, some packets have to be dropped. In this section, we consider the bitstream of each channel as a one minimum unit for audio reconstruction. When the bandwidth is not sufficient, we may drop bitstreams of a certain number of channels to reduce the bit rate. This is called channel-scalable decoding, which has an analogy in MPEG video coding, that is, dropping B frames while keeping only I and P frames.

For an AAC channel pair, the M/S stereo coding block will replace low frequency coefficients in symmetric channels to be their sum and difference at the encoder, that is,

$$\begin{aligned} \mathrm{spec}_l[i] &\longleftarrow \frac{\mathrm{spec}_l[i] + \mathrm{spec}_r[i]}{2}, \\ \mathrm{spec}_r[i] &\longleftarrow \frac{\mathrm{spec}_l[i] - \mathrm{spec}_r[i]}{2}, \end{aligned} \tag{9.7}$$

where $\text{spec}_l[i]$ and $\text{spec}_r[i]$ are the ith frequency-domain coefficient in the left and right channels of the channel pair, respectively.

The intensity coupling coding block will replace high frequency coefficients of the left channel with a value proportional to the envelope of the sound signal in the symmetric channel, and set the value of right channel high frequency coefficients to zero, that is,

$$\begin{aligned} \text{spec}_l[i] &\longleftarrow (\text{spec}_l[i] + \text{spec}_r[i]) \times \sqrt{\frac{E_l[\text{sfb}]}{E_s[\text{sfb}]}}, \\ \text{spec}_r[i] &\longleftarrow 0, \end{aligned} \tag{9.8}$$

where $E_l[\text{sfb}]$, $E_r[\text{sfb}]$, and $E_s[\text{sfb}]$ are, respectively, energy values of the left channel, the right channel and the sum of left and right channels of the scale factor band that sample i belongs to. Values of $E_l[\text{sfb}]/E_r[\text{sfb}]$ are included in the coded bitstream as scaling factors.

At the decoder end, the low frequency coefficients of the left and right channel are reconstructed via

$$\begin{aligned} \text{spec}_l[i] &\longleftarrow \text{spec}_l[i] + \text{spec}_r[i], \\ \text{spec}_r[i] &\longleftarrow \text{spec}_l[i] - \text{spec}_r[i]. \end{aligned} \tag{9.9}$$

For high frequency coefficients, audio signals in the left channel will remain the same as they are received from the bitstream, while those in the right channel will be reconstructed via

$$\text{spec}_r[i] = f(\text{scale}) \times \text{spec}_l[i], \tag{9.10}$$

where $f(\text{scale})$ is a function of the scaling factor.

When packets of one channel of a channel pair are dropped, we drop frequency coefficients of the right channel while keeping all other side information including scaling factors. Therefore, what we receive at the decoder side are just coefficients in the left channel. For low frequency coefficients, they correspond to the mean value of the original frequency coefficient in the left and right channels. For high frequency coefficients, they correspond to the energy envelope of the symmetric channel. That is, we have

$$\begin{aligned} \text{spec}_l[i] &\longrightarrow \frac{\text{spec}_l[i] + \text{spec}_r[i]}{2}, \\ \text{spec}_r[i] &\longrightarrow 0, \end{aligned} \tag{9.11}$$

for the low frequency part and

$$\begin{aligned} \text{spec}_l[i] &\longrightarrow (\text{spec}_l[i] + \text{spec}_r[i]) \times \sqrt{\frac{E_l[\text{sfb}]}{E_s[\text{sfb}]}}, \\ \text{spec}_r[i] &\longrightarrow 0, \end{aligned} \tag{9.12}$$

for the high frequency part.

Note that since scaling factors are contained in the received bitstream, reconstruction of high frequency coefficients in the right channel will remain the same as in the original AAC when data of all channels are received. Therefore, only low frequency coefficients in the right channel need to be recovered. The strategy used to reconstruct these coefficients is just to let values of right channel coefficients equal the values of received left channel coefficients. This is nothing else but the mean value of coefficients in the original channel pair, that is,

$$\text{spec}_r[i] = \text{spec}_l[i] \longrightarrow \frac{\text{spec}_l[i] + \text{spec}_r[i]}{2}. \tag{9.13}$$

Audio concealment for the proposed eigen-channel coding scheme is relatively simple. All coefficients in dropped channels will be set to 0, then a regular decoding process is performed to reconstruct full multichannel audio. For the situation where packets of two or more channels are dropped, the reconstructed dropped channel may have a much smaller energy than other channels after inverse KLT. In order to get better reconstructed audio quality, an energy boost up process can be enforced so that the signal in each channel will have a similar amount of energy.

To illustrate that the proposed algorithm MAACKLT has a better quality-degradation property than AAC (via a proper audio concealment process described in this section), we perform experiments with lossy channels where packets are dropped in a coded bitstream in Section 9.8.

9.6. Compression system overview

The block diagram of the proposed compression system is illustrated in Figure 9.16. It consists of four modules: (1) data partitioning, (2) Karhunen-Loève transform, (3) dynamic range control, and (4) the modified AAC main profile encoder. In the data partitioning module, audio signals in each channel are partitioned into sets of nonoverlapping intervals, that is, blocks. Each block contains K frames, where K is a predefined value. Then, data in each block are sequentially fed into the KLT module to perform interchannel decorrelation. In the KLT module, multichannel block data are decorrelated to produce a set of statistically independent eigen-channels. The KLT matrix consists of eigenvectors of the cross-covariance matrix associated with the multichannel block set. The covariance matrix is first estimated and then quantized into 16 bits per element. The quantized covariance coefficients will be sent to the bitstream as the overhead. Note that the KLT decorrelation module will add a certain amount of delay in the encoder and the decoder. For the nontemporal-adaptive method, the delay can be as long as the length of the input audio. For the temporal-adaptive method, this delay can be reduced to that of the block length.

As shown in Figure 9.1, eigen-channels are generated by multiplication of the KLT matrix and the block data set. Therefore, after the transform, the sample value in eigen-channels may have a larger dynamic range than that of original channels. To avoid any possible data overflow in the upcoming compression module, data

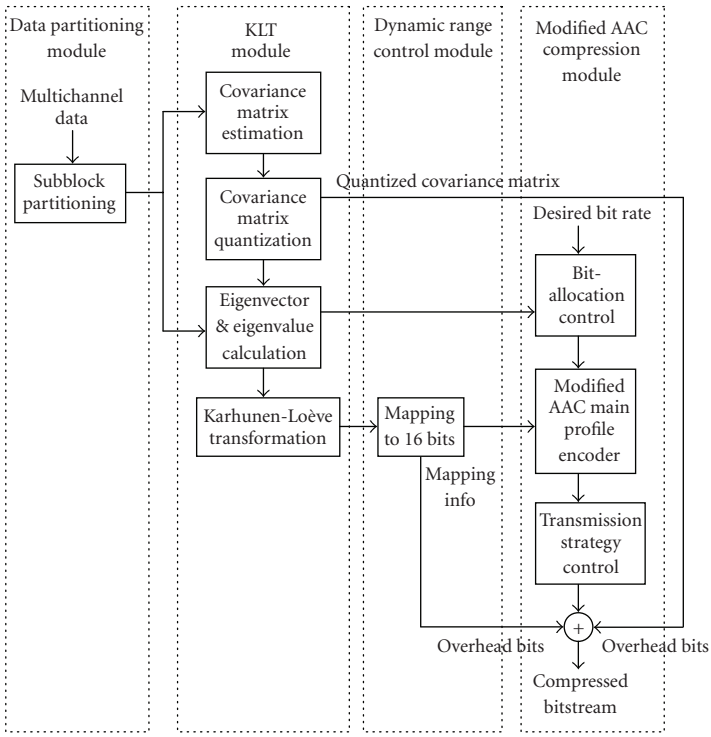


Figure 9.16. The block diagram of the proposed MAACKLT encoder.

in eigen-channels are rescaled in the dynamic range control module so that the sample value input to the modified AAC encoder module does not exceed the dynamic range of that in regular 16-bit PCM audio files. This rescaling information will also be sent to the bitstream as the overhead.

Signals in decorrelated eigen-channels are compressed in the next module by a modified AAC main profile encoder. The AAC main profile encoder is modified in our algorithm so that it is more suitable in compressing the audio signal in eigen-channels. To enable channel-scalability, a transmission strategy control block is adopted in this module right before the compressed bitstream is formed.

The block diagram of the decoder is shown in Figure 9.17. The mapping information and the covariance matrix together with the coded information for eigen-channels are extracted from the received bitstream. If data of some eigen-channels are lost due to the network condition, the eigen-channel concealment block will be enabled. Then, signal values in eigen-channels will be reconstructed by the AAC main profile decoder. The mapping information is used to restore a 16-bit dynamic range of the decoded eigen-channel back to its original range. The inverse KLT matrix can be calculated from the extracted covariance matrix via transposing its eigenvectors. Then, inverse KLT is performed to generate the reconstructed

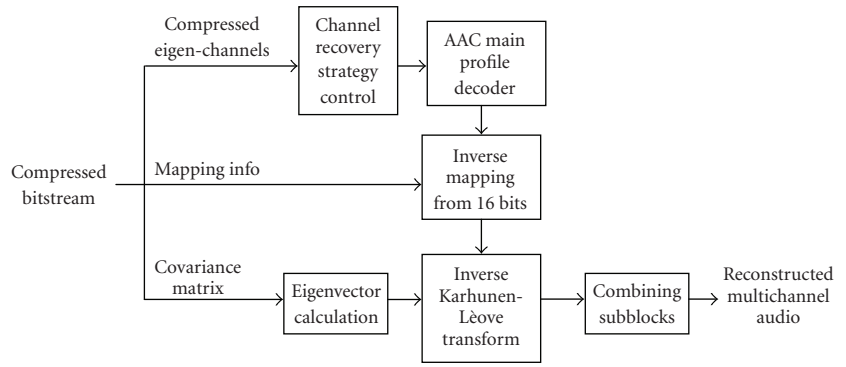


Figure 9.17. The block diagram of the proposed MAACKLT decoder.

multichannel block set. These block sets are finally combined together to produce the reconstructed multichannel audio signals.

9.7. Complexity analysis

Compared with the original AAC compression algorithm, the additional computational complexity required by the MAACKLT algorithm mainly comes from the KLT preprocessing module, which includes the generation of the cross-covariance matrix, calculation of its eigenvalues and eigenvectors, and matrix multiplication required by KLT.

Table 9.1 illustrates the running time of MAACKLT and AAC for both the encoder and the decoder at a typical bit rate of 64 kbit/s/ch, where "*n*-sec AP" means the MAACKLT algorithm with a temporal adaptation period of *n* seconds while "NonA" means a nonadaptive MAACKLT algorithm. The input test audio signals are 20-second ten-channel "Messiah" and 8-second five-channel "Herre." The system used to generate the above result is a Pentium III 600 PC with 128M RAM.

These results indicate that the coding time for MAACKLT is still dominated by the AAC compression and decompression part. When the optimal 10-second temporal adaptation period is used for test audio "Messiah," the additional KLT computational time is less than 7% of the total encoding time at the encoder side while the MAACKLT algorithm only takes about 26.8% longer than that of the original AAC at the decoder side. The MAACKLT algorithm with a shorter adaptation period will take a little bit more time in encoding and decoding since more KLT matrices need to be generated. Note also that we have not made any attempt to optimize our experimental codes. A much shorter amount of encoding/decoding time of MAACKLT is expected if the source code for the KLT preprocessing part is carefully rewritten to optimize the performance.

In channel-scalable decoding, when packets belonging to less important channels are dropped during transmission in the heterogeneous network, the audio concealment part adds a negligible amount of additional complexity in the

Table 9.1. Comparison of computational complexity between MAACKLT and AAC.

Encoding					
Time used (s)	MAACKLT		AAC	Extra	
				Time	Percent
Messiah	1-sec AP	344.28		26.15	8.2%
Messiah	5-sec AP	340.43		22.30	7.0%
Messiah	10-sec AP	339.44		21.31	6.7%
Messiah	NonA	337.62	318.13	19.49	6.1%
Herre	NonA	112.15	101.23	10.92	10.8%
Decoding					
Time used (s)	MAACKLT		AAC	Extra	
				Time	Percent
Messiah	1-sec AP	16.92		4.62	37.6%
Messiah	5-sec AP	16.04		3.74	30.4%
Messiah	10-sec AP	15.60		3.30	26.8%
Messiah	NonA	14.66	12.30	2.36	19.2%
Herre	NonA	2.75	2.42	0.33	13.6%

MAACKLT decoder. The decoding time remains about the same as that of regular bit rate decoding at 64 kbit/s/ch when all packets are received at the decoder side.

9.8. Experimental results

9.8.1. Multichannel audio coding

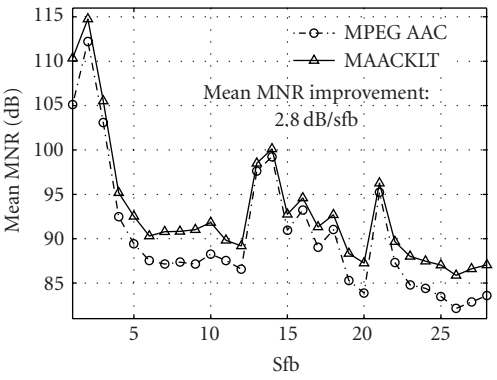
The proposed MAACKLT algorithm has been implemented and tested under the PC Windows environment. We supplemented an interchannel redundancy removal block and a channel transmission control block to the basic source code structure of MPEG-2 AAC [60, 62]. The proposed algorithm is conveniently parameterized to accommodate various input parameters, such as the number of audio channels, the desired bit rate, and the window size of temporal adaptation, and so forth.

We have tested the coding performance of the proposed MAACKLT algorithm by using three ten-channel set audio data "Messiah," "Band,"[4] and "Herbie"[5] and one five-channel set audio data "Herre" at a typical rate of 64 kbit/sec/ch. "Messiah" and "Band" are class III audio files, while "Herbie" and "Herre" are class II audio files. Figures 9.18(a) and (b) show the mean mask-to-noise-ratio (MNR) comparison between the original AAC[6] and the MAACKLT scheme for

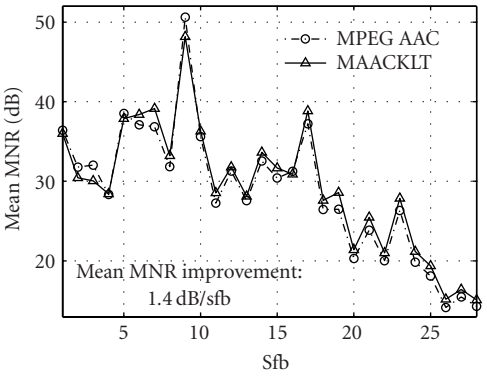
[4]"Band" is a rock band music recorded live in a football field.

[5]"Herbie" is a piece of music played by an orchestra.

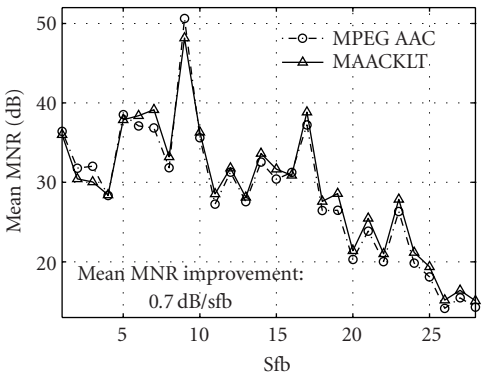
[6]All audio files generated by AAC in this section are processed by the AAC main profile codec.



(a)



(b)



(c)

FIGURE 9.18. The MNR comparison for (a) ten-channel "Herbie" using frequency-domain KLT, (b) five-channel "Herre" using frequency-domain KLT, and (c) five-channel "Herre" using time-domain KLT.

the ten-channel set "Herbie" and the five-channel set "Herre," respectively. The mean MNR values in these figures are calculated via

$$\text{mean MNR}_{\text{sfb}} = \frac{\sum_{\text{channel}} \text{MNR}_{\text{channel,sfb}}}{\text{number of channels}}, \tag{9.14}$$

where sfb represents the "scale factor band." The mean MNR improvement shown in these figures are calculated via

$$\text{mean MNR improvement} = \frac{\sum_{\text{sfb}} (\text{mean MNR}_{\text{sfb}}^{\text{MAACKLT}} - \text{mean MNR}_{\text{sfb}}^{\text{AAC}})}{\text{number of sfb}}. \tag{9.15}$$

Experimental results shown in Figure 9.18(a) and (b) are generated by using the frequency-domain nonadaptive KLT method. These plots clearly indicate that MAACKLT outperforms AAC in the objective MNR measurement for most scale factor bands and achieves mean MNR improvement of more than 1 dB for both test audio. It implies that, compared with AAC, MAACKLT can achieve a higher compression ratio while maintaining similar indistinguishable audio quality. It is worthwhile to mention that no software optimization has been performed for any codec used in this section and all coder blocks adopted from AAC have not been modified to improve the performance of our codec.

Figure 9.18(c) shows the mean MNR comparison between AAC and MAACKLT with the time-domain KLT method using five-channel set "Herre." Compared with the result shown in Figure 9.18(b), we confirm that frequency-domain KLT achieves a better coding performance than time-domain KLT.

The experimental result for the temporal-adaptive approach for ten-channel set "Messiah" is shown in Figure 9.19. This result verifies that a shorter adaptive period leads to a more efficient decorrelation of the multichannel signal but sacrifices the coding performance by adding the overhead in the bitstream. On the other hand, if the covariance matrix is not updated frequently enough, interchannel redundancy cannot be removed to the largest extent. As shown in the figure, to compromise between these two constraints, the optimal adaptation period for "Messiah" is around 10 seconds.

9.8.2. Audio concealment with channel-scalable coding

As described in Section 9.5, when packets of one channel from a channel pair are lost, we can conceal the missing channel at the decoder side. Experimental results show that the quality of the recovered channel pair with the AAC bitstream is much worse than that of the MAACKLT bitstream when it is transmitted under the same network conditions.

Take the test audio "Herre" as an example. If one signal of the L/R channel pair is lost, the reconstructed R channel using the AAC bitstream has obvious

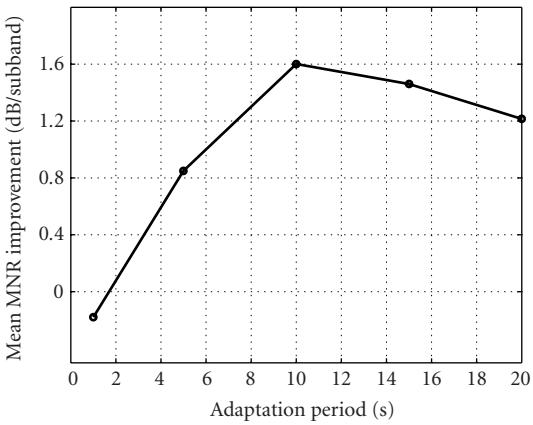


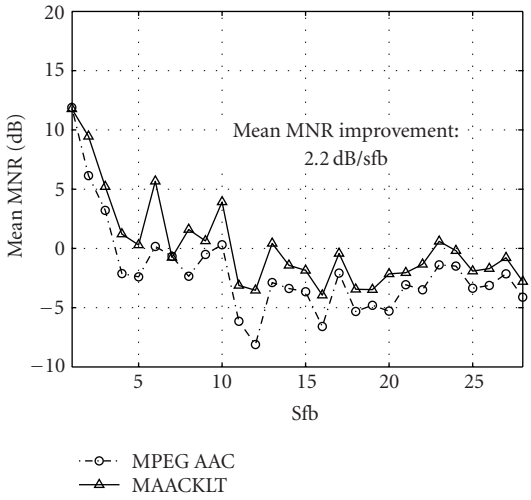
FIGURE 9.19. The mean MNR improvement over AAC for temporal-adaptive KLT applied to the coding of ten-channel "Messiah," where the overhead information is included in the overall bit rate calculation.

distortion and discontinuity in several places while the reconstructed R channel by using the MAACKLT bitstream has little distortion and is much smoother. If one signal of the LS/RS channel pair is lost, the reconstructed RS channel using the AAC bitstream has larger noise in the first one to two seconds in comparison with that of MAACKLT. The corresponding MNR values are compared in Figures 9.20(a) and (b) when AAC and MAACKLT are used; missing channels are concealed, when packets of one channel from L/R and LS/RS channel pairs are lost. We see clearly that MAACKLT achieves better MNR values than AAC for about 2 dB per scale factor band for both cases.

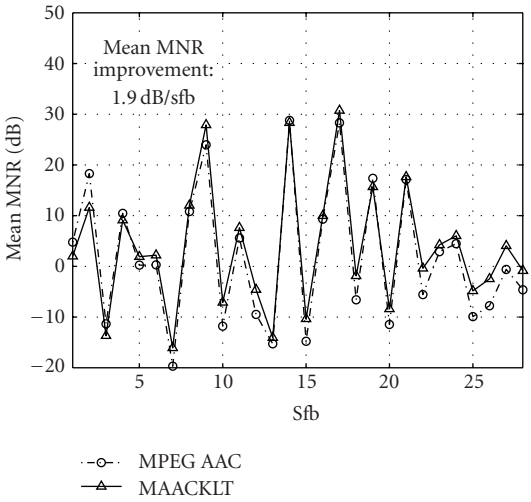
For a typical 5.1 channel configuration, when packets of more than two channels are dropped, which implies that at least one channel pair's information is lost, some lost channels can no longer be concealed from the received AAC bitstream. In contrast, the MAACKLT bitstream can still be concealed to obtain a full 5.1 channel audio with poorer quality. Although the recovered channel pairs do not sound exactly the same as the original ones, a reconstructed full multichannel audio would give the listener a much better acoustical effect than a three- or mono-channel audio.

Take the five-channel "Messiah," which includes C, L, R, LS, and RS channels, as an example. At the worst case, when packets of four channels are dropped and only data of the most important channel are received at the decoder side, the MAACKLT algorithm can still recover five-channel audio. Compared with the original sound, the recovered LS and RS channels lost most of the reverberant sound effect. Since eigen-channel one does not contain much reverberant sound, the MAACKLT decoder can hardly recover these reverberant sound effects in the LS and RS channels.

Similar experiments were also performed using test audio "Herre." However, the advantage of MAACKLT over AAC is not as obvious as that of test audio



(a)



(b)

FIGURE 9.20. MNR comparison for five-channel "Herre" when packets of one channel from the (a) L/R and (b) LS/RS channel pairs are lost.

"Messiah." The reason can be easily found out from the original covariance matrix as shown in Figure 9.2. It indicates that little correlation exists between SCE and CPE for class II test audio such as "Herre." Thus, once one CPE is lost, little information can be recovered from other CPEs or SCEs.

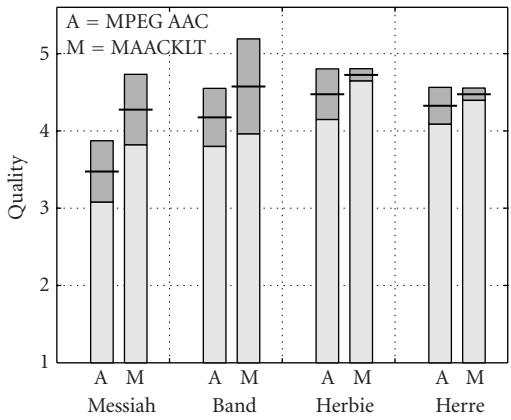


FIGURE 9.21. Subjective listening test results.

9.8.3. Subjective listening test

In order to further confirm the advantage of the proposed algorithm, a formal subjective listening test according to ITU recommendations [1, 2, 3] was conducted in an audio lab to compare the coding performance of the proposed MAACKLT algorithm and that of the MPEG AAC main profile codec. At the bit rate of 64 kbit/s/ch, the reconstructed sound clips are supposed to have perceptual quality similar to that of the original ones. This implies that the difference between MAACKLT and AAC would be so small that nonprofessionals can hardly hear it. Thus, instead of inviting a large number of nonexpert listeners, four well-trained professionals, who have no knowledge of either of two algorithms, participated in the listening test [3]. During the test, for each test sound clip, the subjects listened to three versions of the same sound clip, that is, the original one followed by two processed ones (one by MAACKLT and one by AAC in random order). The subjects were allowed to listen to these files as many times as possible until they were comfortable to give scores to the two processed sound files for each test material.

The five-grade impairment scale given in Recommendation ITU-R BS. 1284 [2] was adopted in the grading procedure and utilized for final data analysis. Four multichannel audio materials, that is, "Messiah," "Band," "Herbie," and "Herre," are all used in this subjective listening test. According to ITU-R BS. 1116-1 [1], audio files selected for the listening test are of short duration (10 to 20 seconds long), so all test files coded by MAACKLT are generated by nonadaptive frequency-domain KLT method.

Figure 9.21 shows the listening test results, where bars represent the score given to each test material coded at 64 kbit/s/ch. The dark shaded area on the top of each bar represents the 95% confidence interval, where the middle line shows the mean value and the other two lines at the boundary of the dark shaded area represent the upper and lower confidence limits [104]. Figure 9.21 indicates that the proposed MAACKLT algorithms outperforms MPEG AAC in all four test pieces,

and indicates statistically significant improvements for the "Messiah" and "Band" pieces.

Besides the 95% confidence interval comparison, it is possible to analyze the obtained listening test results with the sign-test [22, 85] as shown below. Table 9.2 lists listening test results in terms of signs +, 0, and −, which represent cases where the score given to the sound file processed by MAACKLT is higher than, the same as or worse than that of AAC, respectively. It is assumed that the score given by each listener to each sound file is independent of all other scores so that these test results can be viewed as 16 independent experiments. The total number S of sign + can be viewed as a random variable having a binomial distribution with parameters n and p, where n is the number of experiments and p is the probability of getting sign +. We consider the following null hypothesis (H_0) and its counter hypothesis (H_1):

$$\begin{aligned} &H_0 : \text{the proposed MAACKLT algorithm is no better than AAC,} \\ &H_1 : \text{the proposed MAACKLT algorithm is better than AAC.} \end{aligned} \tag{9.16}$$

Under the null hypothesis, the probability that a listener gives a higher score to the sound file processed by MAACKLT should be smaller than or equal to 0.5 (i.e., $p \leq 0.5$).

The strict sign test. In the strict sign test, all experimental results are taken into account, that is, $n = 16$. Because $p \leq 0.5$ (under H_0) and

$$\begin{aligned} (1-p)^{16-i}p^i &\leq \left[\frac{(16-i)(1-p)+ip}{16}\right]^{16} = \left[\frac{16-i+(2i-16)p}{16}\right]^{16} \\ &\leq \left[\frac{16-i+(2i-16)/2}{16}\right]^{16} = 0.5^{16}, \quad \forall i \geq 8, \end{aligned} \tag{9.17}$$

where the first inequality is based on the fact that the arithmetic mean is greater than or equal to the geometric mean. The probability of getting a result as shown in Table 9.2 or getting a result which is even more favorable to our proposed algorithm under H_0 is

$$P[S \geq 13] = \sum_{13 \leq i \leq 16} \binom{16}{i} (1-p)^{16-i}p^i \leq \sum_{13 \leq i \leq 16} \binom{16}{i} 0.5^{16} = \frac{697}{65536} \approx 0.01. \tag{9.18}$$

The above inequality indicates that, under null hypothesis H_0, the probability of getting a listening test result that is so favorable (as given in Table 9.2) or even more favorable to the proposed algorithm would be as small as 1%. Thus, it is concluded that the null hypothesis H_0 is rejected in favor of H_1 at the level of 0.01.

Table 9.2. Performance comparison of AAC and MAACKLT.

	Listener #1	Listener #2	Listener #3	Listener #4
Messiah	+	+	+	+
Band	+	−	+	+
Herbie	+	0	+	+
Herre	+	+	0	+

The loose sign test. In the loose sign test, the experiment that has sign 0 in Table 9.2 is not counted. Then, we have a total of 14 effective experimental results, that is, $n = 14$. The probability of getting a result as shown in Table 9.2 or getting a result which is even more favorable to our proposed algorithm under the null hypothesis H_0 is

$$P[S \geq 13] = \sum_{13 \leq i \leq 14} \binom{14}{i} (1-p)^{14-i} p^i \leq \sum_{13 \leq i \leq 14} \binom{14}{i} 0.5^{14} = \frac{15}{16384} < 0.001. \tag{9.19}$$

This means that, if the null hypothesis is true, the probability of getting a listening test result so favorable or even more favorable to the proposed algorithm would be as small as 0.1%. In other words, the null hypothesis H_0 is rejected in favor of H_1 at the level of 0.001.

Based on the strict sign test or the loose sign test shown above, we reject the null hypothesis with a comfortable degree of confidence and conclude that the proposed algorithm MAACKLT has a better performance than that of AAC.

9.9. Conclusion

We presented a new channel-scalable high-fidelity multichannel audio compression scheme called MAACKLT based on the existing MPEG-2 AAC codec. This algorithm explores the inter and intrachannel correlation in the input audio signal and allows channel-scalable decoding. The compression technique utilizes KLT in the preprocessing stage to remove the interchannel redundancy, then compresses the resulting relatively independent eigen-channel signals with a modified AAC main profile encoder module, and finally uses a prioritized transmission policy to achieve quality scalability. The novelty of this technique lies in its unique and desirable capability to adaptively vary the characteristics of the interchannel decorrelation transform as a function of the covariance of a certain period of music and its ability to reconstruct different quality audio with single bitstream. It achieves a good coding performance especially for the input audio source whose channel number goes beyond 5.1. In addition, it outperforms AAC according to both objective (MNR measurement) and subjective (listening) tests at the typical low-bit-rate of 64 kbit/s/ch while maintaining a similar computational complexity for

both encoder and decoder modules. Moreover, compared with AAC, the channel-scalable property of MAACKLT enables users to conceal full multichannel audio of reasonable quality without any additional cost.

9.10. Appendix: Karhunen-Loève expansion

9.10.1. Definition

The discrete-time *Karhunen-Loève expansion* can be described as shown below.

Karhunen-Loève expansion. Suppose $\mathbf{u}(n)$ is an M-by-1 vector that represents a data sequence drawn from a wide-sense stationary process with mean equals zero and correlation matrix equals $\mathbf{R}$. Let M vectors $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_M$ denote eigenvectors of the matrix $\mathbf{R}$. Thus, vector $\mathbf{u}(n)$ may be represented by a linear combination of these eigenvectors as follows:

$$\mathbf{u}(n) = \sum_{i=1}^{M} c_i \mathbf{q}_i, \tag{9.20}$$

where, $c_i(n)$ are coefficients of the expansion, which have zero-mean and are uncorrelated to each other. Coefficients $c_i(n)$ can be calculated by the inner product of $\mathbf{q}_i$ and $\mathbf{u}(n)$, that is,

$$c_i(n) = \mathbf{q}_i^H \mathbf{u}(n), \quad i = 1, 2, \ldots, M. \tag{9.21}$$

Equation (9.21) shows the analysis part of the Karhunen-Loève expansion and indicates the relationship between the coefficient $c_i(n)$ and the input vector $\mathbf{u}(n)$. Whereas equation (9.20) shows the "synthesis" part of the expansion and illustrates how to reconstruct the original input vector $\mathbf{u}(n)$ from $c_i(n)$. Assuming that all eigenvectors $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_M$ have the unit length, these eigenvectors form an orthonormal set. The definition of the expansion coefficient in equation (9.21) can be derived directly from equation (9.20). Similarly, given equation (9.21), equation (9.20) can also be easily derived.

9.10.2. Features and properties

The expansion coefficients $c_i(n)$ are random variables and have the following desirable properties:

$$E[c_i(n)] = 0, \quad i = 1, 2, \ldots, M, \tag{9.22}$$

$$E[c_i(n) c_j^*(n)] = \begin{cases} \lambda_i, & i = j, \\ 0, & i \neq j. \end{cases} \tag{9.23}$$

Equation (9.22) tells us that all coefficients of the expansion have a mean value equal to zero. Equation (9.23) indicates that coefficients of the expansion are uncorrelated and that each one of them has a mean-square value equal to its corresponding eigenvalue. The first property can be easily calculated from the coefficient's definition shown in equation (9.21) given the fact that the input random vector $\mathbf{u}(n)$ is assumed to have the zero mean in the beginning. The derivation of the second property is given as below:

$$\begin{aligned} E[c_i(n)c_j^*(n)] &= E\Big[\left(\mathbf{q}_i^H\mathbf{u}(n)\right)\left(\mathbf{q}_j^H\mathbf{u}(n)\right)^H\Big] \\ &= \mathbf{q}_i^H E[\mathbf{u}(n)\mathbf{u}(n)^H]\mathbf{q}_j \\ &= \mathbf{q}_i^H\mathbf{R}\mathbf{q}_j \\ &= \lambda_j\mathbf{q}_i^H\mathbf{q}_j \\ &= \begin{cases} \lambda_i, & i = j, \\ 0, & i \neq j. \end{cases} \end{aligned} \tag{9.24}$$

There is also a physical interpretation of the above discrete-time Karhunen-Loève expansion. Let the eigenvectors $\mathbf{q}_1, \mathbf{q}_2, \ldots, \mathbf{q}_M$ be the M orthogonal coordinates of an M-dimensional space. Then the random vector $\mathbf{u}(n)$ can be represented by a set of its projections $c_1(n), c_2(n), \ldots, c_M(n)$ onto these axes. From equation (9.20), we know

$$\sum_{i=1}^{M} |c_i(n)|^2 = |\mathbf{u}(n)|^2, \tag{9.25}$$

where $|\mathbf{u}(n)|$ represents the Euclidean norm of $\mathbf{u}(n)$. The above equation suggests that the energy measured along the ith coordinate of the original vector $\mathbf{u}(n)$ is given by the Euclidean norm of the ith coefficient $c_i(n)$. Moreover, the mean value of this energy is equal to the ith eigenvalue, that is,

$$E[|c_i(n)|^2] = \lambda_i, \quad i = 1, 2, \ldots, M. \tag{9.26}$$

which is a direct result of equation (9.23).

10 Adaptive Karhunen-Loève transform and its quantization efficiency

10.1. Introduction

Based on today's most distinguished multichannel audio coding system, a modified advanced audio coding with Karhunen-Loève transform (MAACKLT) method is proposed to perceptually losslessly compress a multichannel audio source in Chapter 9. This method utilizes the Karhunen-Loève transform (KLT) in the preprocessing stage for the powerful multichannel audio compression tool, that is, MPEG advanced audio coding (AAC), to remove interchannel redundancy and further improve the coding performance. However, as described in Chapter 9, each element of the covariance matrix, from which the KLT matrix is derived, is scalar quantized to 16 bits. This results in a 240 bits overhead for each KLT matrix for typical five channel audio contents. Since the bit budget is the most precious resource in the coding technique, every effort must be made to minimize the overhead due to the additional preprocessing stage while maintaining a similar high-quality coding performance. Moreover, the original MAACKLT algorithm did not fully explore the KLT temporal adaptation effect.

In this research, we investigate the KLT decorrelation efficiency versus the quantization accuracy and the temporal KLT adaptive period. Extensive experiments on the quantization of the covariance matrix by using scalar and vector quantizers have been performed. Based on these results, the following two interesting points are concluded.

(1) Coarser quantization can dramatically degrade the decorrelation capability in terms of the normalized covariance matrix of decorrelated signals. However, the degradation of decoded multichannel audio quality is not as obvious.

(2) Shorter temporal adaptation of KLT will not significantly improve the decorrelation efficiency whilst it will considerably increase the overhead. Thus, a moderately long adaptation time is a good choice.

It is shown in this work that, with vector quantization, we can reduce the overhead from more than 200 bits to less than 3 bits per KLT while maintaining comparable coding performance. Even with scalar quantization, a much lower overhead bit rate can still generate decoded audio with comparable quality. Our experimental results indicate that although a coarser quantization of the covariance matrix

gives a poorer decorrelation effect, reduction of bits in the overhead is able to compensate this degradation to result in a similar coding performance in terms of the objective MNR measurement.

The rest of this chapter[1] is organized as follows. In Section 10.2, we introduce vector quantization and its application to the MAACKLT algorithm. In Sections 10.3 and 10.4, we explore how the quantization method and the temporal adaptive scheme affect the KLT decorrelation efficiency and the coding performance by applying scalar and vector quantizers to encode the KLT matrix with a range of different bit rates. In Section 10.5, we examine computational complexity issues. Some experimental results are presented in Section 10.6. Finally concluding remarks are given in Section 10.7.

10.2. Vector quantization

The MAACKLT algorithm described in Chapter 9 dealt only with scalar quantization of the covariance matrix. If the input audio material is short or if the KLT matrix is updated more frequently, the overhead that results from transmitting the covariance matrix will increase significantly, which will degrade the coding performance of the MAACKLT algorithm to a certain extent. To alleviate this problem, we have to resort to a look-up-table (LUT) approach. Here, a stored table of precalculated covariance matrices is searched to find the one that approximates the estimated covariance matrix of the current block of the input audio. This approach yields substantial savings in the overhead bit rate since only pointers to the table, instead of the entire covariance matrix itself, will be transmitted to the receiver.

Vector quantization (VQ) [38, 97, 29] provides an excellent choice to implement the LUT idea. By vector quantization, we identify a set of possible vectors both at the encoder and the decoder side. They are called the codebook. The VQ encoder pairs up each source vector with the closest matching vector (i.e., "codeword") in the codebook, thus "quantizing" it. The actual encoding is then simply a process of sequentially listing the identity of codewords that match most closely with vectors making up the original data. The decoder has a codebook identical to the encoder, and the decoding is a trivial matter of piecing together the vectors whose identity has been specified. Vector quantizers in this work consider the entire set of nonredundant elements of each covariance matrix as an entity, or a vector. The identified codebook should be general enough to include the characteristics of different types of multichannel audio sources. Since VQ allows for direct minimization of the quantization distortion, it results in smaller quantization distortion than scalar quantizers (SQ) at the same bit rate. In other words, VQ demands a smaller number of bits for source data coding while keeping the quantization error similar to that achieved with a scalar quantizer.

Four different five-channel audio pieces (each containing center (C), left (L), right (R), left surround (LS), and right surround (RS) channels), are used to generate more than 80,000 covariance matrices. Each covariance matrix is treated as

[1] Part of this chapter represents work published before, see [138].

one training vector **X**, which is composed of fifteen nonredundant elements of the covariance matrix as shown below:

$$\mathbf{X} = \begin{matrix} x_1 & & & & \\ x_2 & x_3 & & & \\ x_4 & x_5 & x_6 & & \\ x_7 & x_8 & x_9 & x_{10} & \\ x_{11} & x_{12} & x_{13} & x_{14} & x_{15}, \end{matrix} \tag{10.1}$$

where $x_1, x_2, \ldots, x_{15}$ are elements in the lower triangular part of the covariance matrix. During the codebook generation procedure, the generalized Lloyd algorithm (GLA) was run on the training sequence by using the simple square error distortion measurement, that is,

$$d(\mathbf{X}, Q(\mathbf{X})) = \sum_{i=1}^{15} [\mathbf{X} - Q(\mathbf{X})]^2, \tag{10.2}$$

where $Q(\mathbf{X})$ represents the quantized value of **X**. The same distortion measurement is used with the full searching method during the encoding procedure.

10.3. Efficiency of KLT decorrelation

The magnitudes of nondiagonal elements in a normalized covariance matrix provide a convenient metric to measure the degree of interchannel correlation. The normalized covariance matrix is derived from the cross-covariance matrix by multiplying each coefficient with the reciprocal of the square root of the product of their individual variance, that is,

$$C_N(i, j) = \frac{C(i, j)}{\sqrt{C(i, i) \times C(j, j)}}, \tag{10.3}$$

where $C_N(i, j)$ and $C(i, j)$ are elements of the normalized covariance matrix and the cross-covariance matrix in row i and column j, respectively.

Tables 10.1 and 10.2 show the absolute values of nonredundant elements (i.e., elements in only the lower- or the upper-triangle) of the normalized covariance matrix calculated from original signals and KLT decorrelated signals, respectively, where no quantization is performed during the KLT decorrelation. From these tables, we can easily see that KLT reduces the interchannel correlation from around the order of 10^{-1} to the order of 10^{-4}.

In order to investigate how the decorrelation efficiency is affected by various quantization schemes, a sequence of experiments, including SQ and VQ with a different number of bits per element/vector, was performed. Table 10.3 shows the absolute values of nonredundant elements of the normalized covariance matrix calculated from KLT decorrelated signals, where each element of the covariance

Table 10.1. Absolute values of nonredundant elements of the normalized covariance matrix calculated from original signals.

1				
5.36928147e-1	1			
3.26056331e-1	1.02651220e-1	1		
1.17594877e-1	8.56662289e-1	5.12340667e-3	1	
7.46899187e-2	1.33213668e-1	1.15962389e-1	6.55651089e-2	1

Table 10.2. Absolute values of nonredundant elements of the normalized covariance matrix calculated from KLT decorrelated signals.

1				
1.67971275e-4	1			
2.15059591e-4	1.01530173e-3	1		
4.19255484e-4	4.03864289e-4	2.56863610e-4	1	
3.07486032e-4	4.23535476e-4	3.48484672e-4	5.20389082e-5	1

Table 10.3. Absolute values of nonredundant elements of the normalized covariance matrix calculated from scalar quantized KLT decorrelated signals.

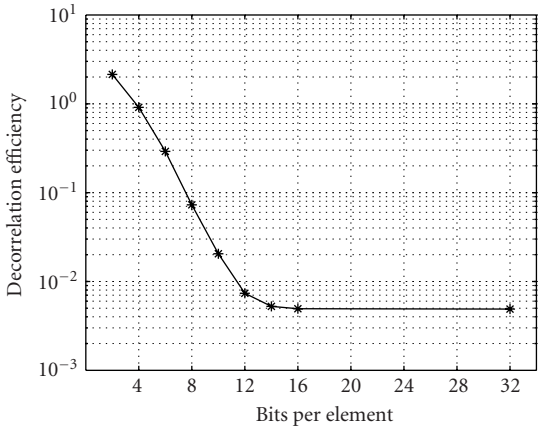
1				
1.67971369e-4	1			
2.15059518e-4	1.01530166e-3	1		
4.19255341e-4	4.03863772e-4	2.56863464e-4	1	
3.07486076e-4	4.23536876e-4	3.48484820e-4	5.20396538e-5	1

matrix is scalar quantized into 32 bits. Compared with Table 10.2, values in Table 10.3 are almost identical to those in Table 10.2 with less than 0.0001% distortion per element. This suggests that, with a 32 bits per element scalar quantizer, we can almost faithfully reproduce the covariance matrix with negligible quantization error.

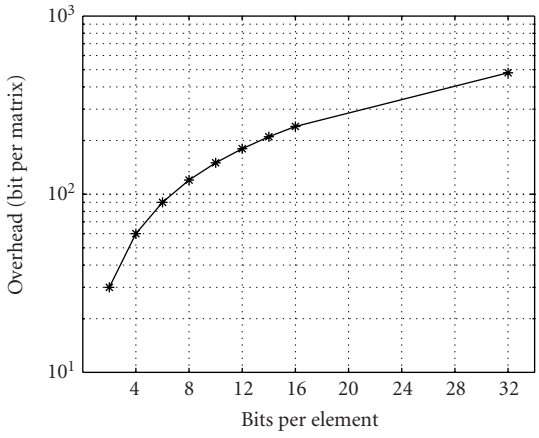
Figures 10.1 and 10.2 illustrate how decorrelation efficiency and the corresponding overhead changes with SQ and VQ, respectively. It is observed that simple mean square error (MSE) measurement is not a good choice when evaluating the decorrelation efficiency. A better measure is the average distortion D, which is the summation of magnitudes of lower triangular elements of the normalized covariance matrix, that is,

$$D = \sum_{i=2}^{N} \sum_{j=1}^{i-1} \left| C_N(i,j) \right|, \tag{10.4}$$

where C_N is the normalized covariance matrix of signals after KLT decorrelation.



(a)



(b)

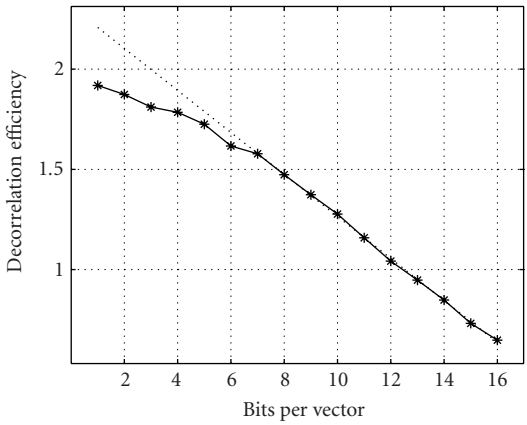
FIGURE 10.1. (a) The decorrelation efficiency and (b) the overhead bit rate versus the number of bits per element in SQ.

The overhead in terms of bits per KLT matrix is calculated via

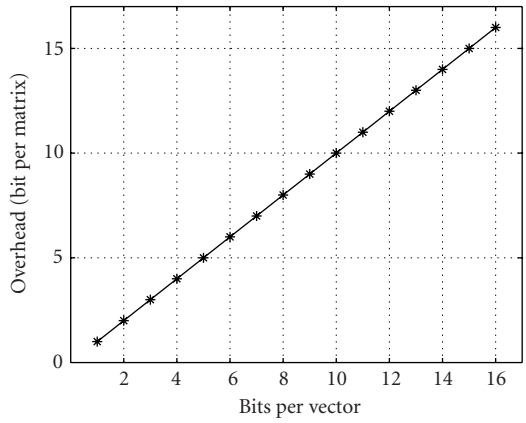
$$OH_s = B_s \times N, \tag{10.5}$$

$$OH_v = B_v, \tag{10.6}$$

where B_s denotes the number of bits per element for SQ, N is the number of nonredundant elements per matrix, and B_v denotes the number of bits per codeword for VQ (recall that one KLT matrix is quantized into one codeword). For five-channel audio material, N is equal to 15.



(a)



(b)

FIGURE 10.2. (a) The decorrelation efficiency and (b) the overhead bit rate versus the number of bits per vector in VQ.

Figure 10.1(a) suggests that there is no significant degradation in decorrelation efficiency when the number of bits is reduced from 32 bits per element to 14 bits per element. However, further reduction in the number of bits per element will result in a dramatic increase of distortion D given in equation (10.4). From equation (10.5), we know that the overhead increases linearly as the number of bits per element increases with a gradient equal to N. This is confirmed by Figure 10.1(b), in which the overhead is plotted as a function of the number of bits per element in the logarithmic scale. Compared with Figure 10.1(a), we observe that the overhead OH increases much more rapidly than the decrease of the distortion D when the number of bits per element increases from 14 to 32. It

Table 10.4. Decorrelation results with SQ.

Bit/element	D	Overhead (bit/matrix)	Ave MNR (dB/sb)
2	2.14	30	N/A[a]
4	9.13e-1	60	56.56
6	2.91e-1	90	56.37
8	7.29e-2	120	56.02
10	2.05e-2	150	56.08
12	7.36e-3	180	56.00
14	5.24e-3	210	55.93
16	4.93e-3	240	55.91
32	4.89e-3	480	55.84

[a]Using 2 bits per element, which quantizes each element into values of either 0 or ±1, leads to problems in later compression steps.

indicates that when transmitting the covariance matrix with a higher bit rate, the improvement of decorrelation efficiency is actually not sufficient to compensate the loss due to a higher overhead rate.

The minimum number of bits per element for SQ is 2, since we need one bit for the sign and at least one bit for the absolute value for each element. To further reduce the number of bits per element, VQ is used. Figure 10.2(a) illustrates that the average distortion D increases almost linearly when the number of bit per vector decreases from 16 bits per vector to 7 bits per vector, and then slows down when the bit per vector further decreases. Compared with Figure 10.1, it is verified that VQ results in smaller quantization distortion than SQ at any given bit rate. Figure 10.2(b) shows how the overhead varies with the number of bits per covariance matrix. Note that VQ reduces the overhead bit rate by more than a factor of N (which is 15 for five channel audio) with respect to SQ.

Tables 10.4 and 10.5 show the average distortion D, the overhead information and the average MNR value for SQ and VQ, respectively, where the average MNR value is calculated as below:

$$\text{mean MNR}_{\text{subband}} = \frac{\sum_{\text{channel}} \text{MNR}_{\text{channel,subband}}}{\text{number of channels}}, \tag{10.7}$$

$$\text{average MNR} = \frac{\sum_{\text{subband}} \text{mean MNR}_{\text{subband}}}{\text{number of subband}}. \tag{10.8}$$

The test audio material used to generate results in this section is a five channel performance of "Messiah" with the KLT matrix updated every second. As shown in Table 10.4, a fewer number of bits per element results in a higher distortion in exchange for a smaller overhead and, after all, a larger MNR value. Thus, although a smaller number of bits per element of the covariance matrix results in a larger

Table 10.5. Decorrelation results with VQ.

Bit/vector	D	Overhead (bit/matrix)	Ave MNR (dB/sb)
1	1.92	1	56.73
2	1.87	2	56.61
3	1.81	3	56.81
4	1.78	4	56.87
5	1.73	5	56.12
6	1.62	6	56.23
7	1.58	7	56.88
8	1.47	8	56.96
9	1.37	9	56.42
10	1.28	10	55.97
11	1.16	11	56.08
12	1.04	12	56.28
13	0.948	13	55.83
14	0.848	14	55.72
15	0.732	15	56.19
16	0.648	16	55.87

average distortion D, the decrease of the overhead bit rate actually compensates this distortion and improves the MNR value for the same coding rate.

A similar argument applies to the VQ case as shown in Table 10.5 with only one minor difference. That is, the MNR value is nearly a monotonic function for the SQ case while it moves up and down slightly in a local region (fluctuating within 1 dB per subband) for the VQ case. However, the general trend is the same, that is, a larger overhead in KLT coding degrades the final MNR value. We also noticed that even when using 1 bit per vector, vector quantization of the covariance matrix still gives good MNR results.

Our conclusion is that it is beneficial to reduce the overhead bit rate used in the coding of the covariance matrix of KLT, since a larger overhead has a negative impact on the rate-distortion trade-off.

10.4. Temporal adaptation effect

A multichannel audio program in general comprises of several different periods, each of which has its unique spectral signature. For example, a piece of music may begin with a piano preclude followed by a chorus. In order to achieve the highest information compactness, the decorrelation transform matrix must adapt to the characteristics of different sections of the program material. The MAACKLT algorithm utilizes a temporal-adaptive approach, in which the covariance matrix is updated frequently. On one hand, the shorter the adaptive time, the more efficient

the interchannel decorrelation mechanism. On the other hand, since the KLT covariance matrix has to be coded for audio decoding, a shorter adaptive time contributes to a larger overhead in bit rates. Thus, it is worthwhile to investigate the trade-off so that a good balance between this adaptive time and the final coding performance can be reached.

In Figure 10.3, we show the magnitude of the lower triangular elements of the normalized covariance matrix calculated from decorrelated signals by using different adaptive periods, where no quantization has been applied yet. These figures suggest that there is no significant improvement of the decorrelation efficiency when the KLT adaptive time decreases from 10 seconds to 0.05 second. As the overhead dramatically increases with the shorter adaptive time, the final coding performance may be degraded. In order to find the optimal KLT adaptive time, a thorough investigation is performed for both SQ and VQ.

First, we look at how adaptive time affects the overhead bit rate. Suppose n channels are selected for simultaneous interchannel decorrelation, the adaptive time is K seconds, that is, each subblock contains K seconds of audio, and M bits are transmitted to the decoder for each KLT. The overhead bit rate r_{overhead} is

$$r_{\text{overhead}} = \frac{M}{nK} \tag{10.9}$$

in bits per second per channel (bit/s/ch). This equation suggests that the overhead bit rate increases linearly with the number of bits used to encode and transmit the KLT matrix. The overhead bit rate is, however, inversely proportional to the number of channels and the adaptive time. If SQ is used in the encoding procedure, each nonredundant element has to be sent. For n channel audio material, the size of the covariance matrix is $n \times n$, and the number of nonredundant elements is $n \times (n+1)/2$. If B_s bits are used to quantize each element, the total bit requirement for each KLT is $n(n+1)B_s/2$. Thus the overhead bit rate $r^{\text{SQ}}_{\text{overhead}}$ for SQ is equal to

$$r^{\text{SQ}}_{\text{overhead}} = \frac{(n+1)B_s}{2K}. \tag{10.10}$$

The overhead bit rate $r^{\text{VQ}}_{\text{overhead}}$ for VQ is simpler. It is equal to

$$r^{\text{VQ}}_{\text{overhead}} = \frac{B_v}{nK}, \tag{10.11}$$

where B_v represents the number of bits used for each KLT covariance matrix.

The average MNR value (in dB) and the overhead bit rate (in the logarithm scale) versus the adaptive time for both SQ and VQ are shown in Figures 10.4(a) and (b), respectively. The test material is five-channel "Messiah," with 8 bits per element for SQ and 4 bits per vector for VQ for KLT decorrelation. The total coding

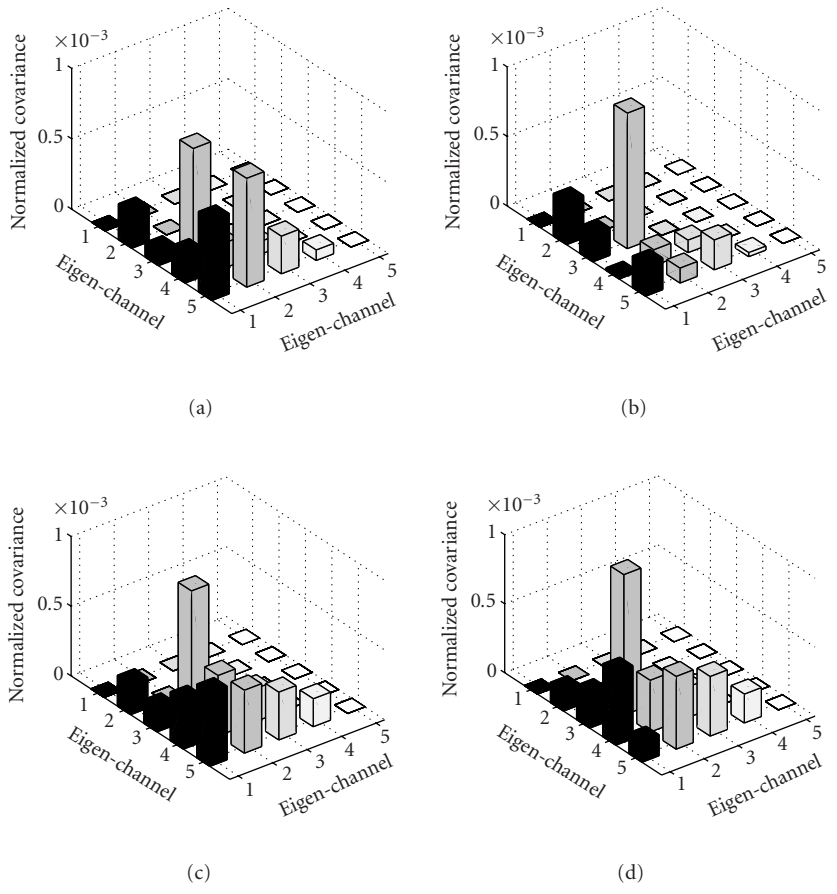
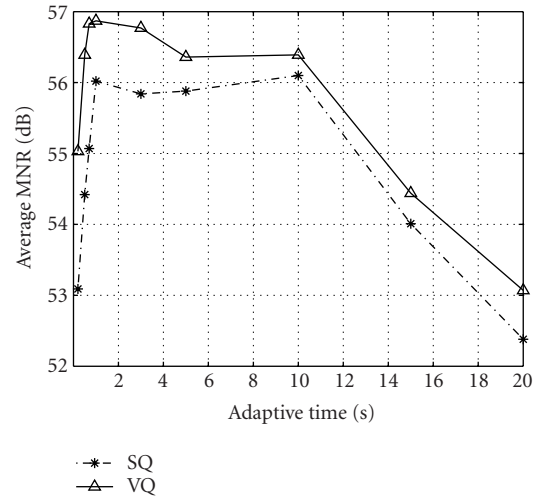


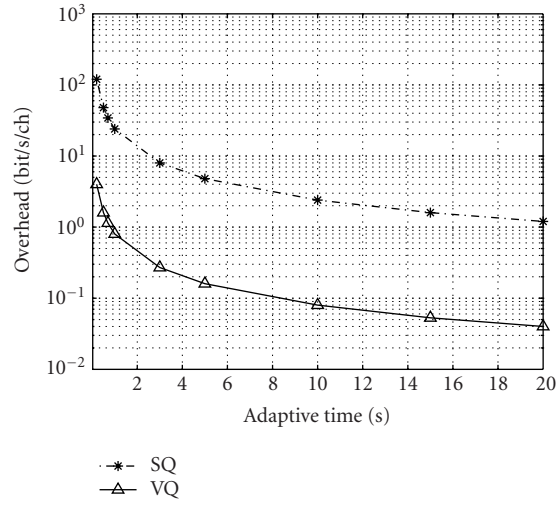
FIGURE 10.3. The magnitude of the lower triangular elements of the normalized covariance matrix calculated from decorrelated signals, where the adaptive time is equal to (a) 0.05, (b) 0.2, (c) 3, and (d) 10 seconds.

bit rate (including bits for the overhead and the content) is kept the same for all points in the two curves in Figure 10.4(a). We have the following observations from these figures.

First, for the SQ case, the average MNR value remains about the same with less than 0.3 dB variation per subband when the adaptive time varies from 1 to 10 seconds. However, when the adaptive time is decreased furthermore, the overhead effect starts to dominate, and the average MNR value decreases dramatically. On the other hand, when the adaptive time becomes longer than 10 seconds, the average MNR value also decreases, which implies that the coding performance is degraded if the KLT matrix is not updated frequently enough. For VQ, the changing pattern of the average MNR value versus the adaptive time is similar as that of SQ. However, compared with the scalar case, the average MNR value starts to



(a)



(b)

FIGURE 10.4. (a) Adaptive MNR results and (b) adaptive overhead bits for SQ and VQ for five-channel Messiah.

decline earlier at about 5 seconds. This is probably due to the effect that VQ gives less efficient decorrelation, so that more frequent adaptation of the KLT matrix will generate a better coding result. As shown in Figure 10.4(a), it is clear that the average MNR generated by using VQ is always better than that of SQ and the difference becomes significant when the overhead becomes the dominant factor for KLT adaptive time less than 1 second.

10.5. Complexity analysis

The main concern of a VQ scheme is its computational complexity at the encoder side. For each D dimension vector, we need $O(DS)$ operations to find the best matched codeword from a codebook of size S using the full search technique. For an n channel audio, each covariance matrix is represented by a vector of dimension $n(n+1)/2$. Thus, for each KLT, we need $O(n^2S)$ operations. While for the scalar case, the quantization of each element requires $O(1)$ operations, and for each covariance matrix, we need $O(n^2)$ operations. Suppose that the input audio is of L seconds long, and the KLT matrix is updated each K seconds. Then, there will be totally $\lceil L/K \rceil$ covariance matrices to be quantized.[2] This means we need

$$\begin{aligned} O(\lceil L/K \rceil n^2) &= O(Ln^2/K) \quad \text{for scalar quantization,} \\ O(\lceil L/K \rceil n^2 S) &= O(Ln^2 S/K) \quad \text{for vector quantization,} \end{aligned} \tag{10.12}$$

operations.

Thus, for a given test audio source, the complexity is inversely proportional to KLT adaptive time for either quantization scheme. For VQ, the complexity is also proportional to the codebook size. Compared with SQ, VQ requires more operations by a factor of S. To reduce the computational complexity, we should limit the codebook size and set the KLT adaptation time as long as possible while keeping the desired coding performance.

Experimental results shown in Section 10.3 suggest that a very small codebook size is usually good enough to generate the desired compressed audio. By choosing a small codebook size and keeping the KLT adaptation time long enough, we do not only limit the additional computational complexity, but also save the overhead bit requirement. At the decoder side, VQ demands just a table look-up procedure and its complexity is comparable to that of SQ.

10.6. Experimental results

We tested the modified MAACKLT method by using two five-channel audio sources "Messiah" and "Ftbl" at different bit rates varying from a typical rate of 64 kbit/s/ch to a very-low-bit-rate of 16 kbit/s/ch. Figures 10.5 and 10.6 show the mean MNR comparison between SQ and VQ for test audio "Messiah" and "Ftbl," respectively, where the KLT matrix adaptation time is set to 10 seconds. The mean MNR values in these figures are calculated by equation (10.7). In order to show the general result of scalar and vector cases, a moderate bit rate (i.e., 8 bits per element for SQ and 4 bits per vector for VQ) is adopted here. From these figures, we see that, compared with SQ, VQ generates comparable mean MNR results at all bit rates, and VQ even outperforms SQ at all bit rates for some test sequence such as "Messiah."

[2] $\lceil * \rceil$ represents the smallest integer which is greater than or equal to $*$.

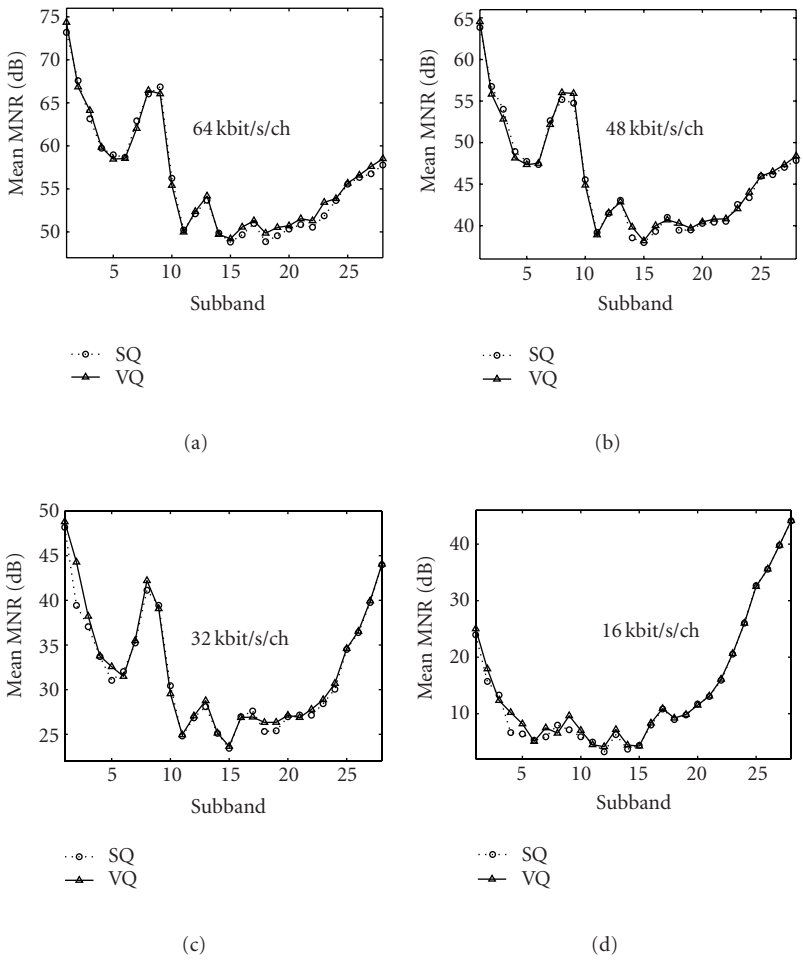


FIGURE 10.5. MNR result using test audio "Messiah" coded at (a) 64 kbit/s/ch, (b) 48 kbit/s/ch, (c) 32 kbit/s/ch, and (d) 16 kbit/s/ch.

10.7. Conclusion

To enhance the MAACKLT algorithm proposed earlier, we examined the relationship between coding of the KLT covariance matrix with different quantization methods, KLT decorrelation efficiency and the frequency of KLT information update extensively in this research. In particular, we investigated how different quantization methods affect the final coding performance by using objective MNR measurements. It is demonstrated that reducing the overhead bit rate generally provides a better trade-off for the overall coding performance. This can be achieved by adopting a smallest possible bit rate to encode the covariance matrix together

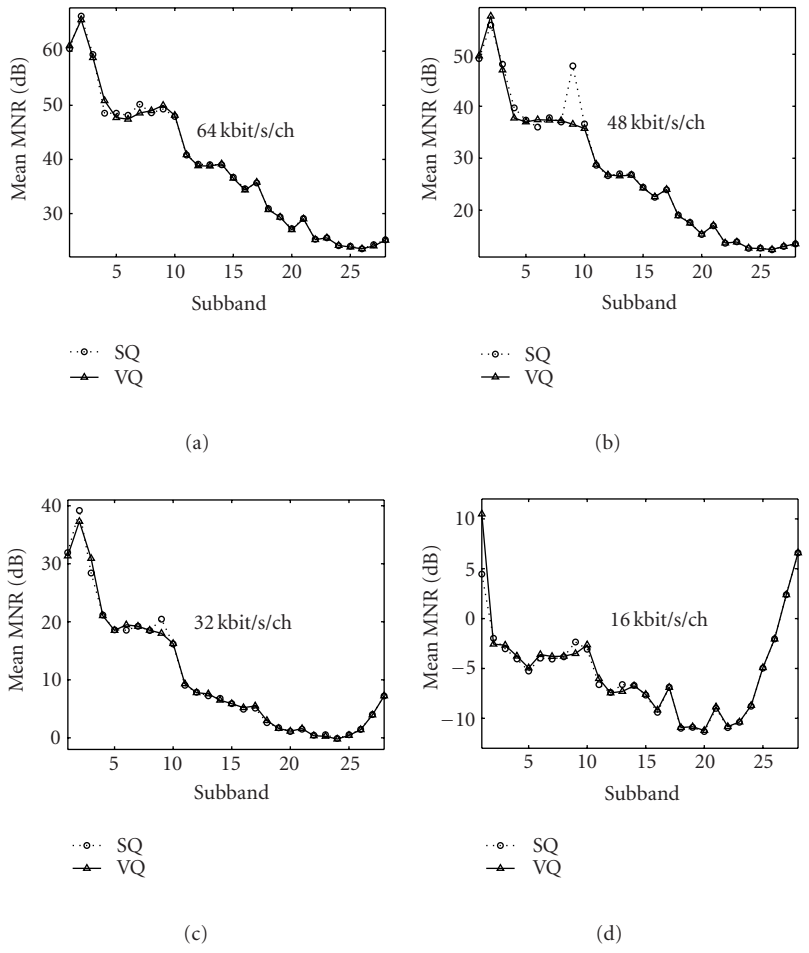


FIGURE 10.6. MNR result using test audio "Ftbl" coded at (a) 64 kbit/s/ch, (b) 48 kbit/s/ch, (c) 32 kbit/s/ch, and (d) 16 kbit/s/ch.

with moderately long KLT adaptation period to generate the desired coding performance. Besides, a small codebook size in VQ do not increase the computational complexity significantly.

11 Progressive syntax-rich multichannel audio codec

11.1. Introduction

Most of today's multichannel audio codecs can only provide bitstreams with a fixed bit rate, which is specified during the encoding phase. When this kind of bitstream is transmitted over variable bandwidth networks, the receiver can either successfully decode the full bitstream or ask the encoder site to retransmit a bitstream with a lower bit rate. The best solution to address this problem is to develop a scalable compression algorithm, which is able to transmit and decode the bitstream with a bit rate that can be adaptive to a dynamically varying environment (e.g., the instantaneous capacity of a transmission channel). This capability offers a significant advantage in transmitting contents over channels with a variable channel capacity or connections for which the available channel capacity is unknown at the time of encoding. To achieve this goal, a bitstream generated by scalable coding schemes consists of several partial bitstreams, each of which can be decoded on their own in a meaningful way. Therefore, transmission and decoding of a subset of the total bitstream will result in a valid decodable signal at a lower bit rate and quality.

MPEG-4 version-2 audio coding supports fine grain bit rate scalablility [98, 61, 65, 66, 47] in its generic audio coder (GAC). It has a bit-sliced arithmetic coding (BSAC) tool, which provides scalability in the step of 1 kbit/s per audio channel for mono or stereo audio material. Several other scalable mono or stereo audio coding algorithms [145, 128, 117] were proposed in recent years. However, not much work has been done on progressively transmitting multichannel audio sources. In this work, we propose a progressive syntax-rich multichannel audio codec (PSMAC) based on MPEG AAC. In PSMAC, the interchannel redundancy inherent in original physical channels is first removed in the preprocessing stage by using the Karhunen-Loève transform (KLT). Then, most coding blocks in the AAC main profile encoder are employed to generate spectral coefficients. Finally, a progressive transmission strategy and a context-based QM coder are adopted to obtain the fully quality-scalable multichannel audio bitstream. The PSMAC system not only supports fine grain bit rate scalability for the multichannel audio bitstream, but also provides several other desirable functionalities, such as random

access and channel enhancement, which have not been supported by other existing multichannel audio codecs.

Moreover, compared with the BSAC tool provided in MPEG-4 version-2 and most of other scalable audio coding tools, a more sophisticated progressive transmission strategy is employed in PSMAC. PSMAC does not only encode spectral coefficients from MSB to LSB and from low to high frequency so that the decoder can reconstruct these coefficients more and more precisely with an increasing bandwidth as the receiver collects increasingly more bits from the bitstream, but also utilizes the psychoacoustic model to control the subband transmission sequence so that the most sensitive frequency area is more precisely reconstructed. In this way, bits used to encode coefficients in the nonsensitive frequency area can be saved and used to encode coefficients in the sensitive frequency area. Because of this subband selection strategy, a perceptually more appealing audio can be reconstructed by PSMAC, especially at very-low-bit-rates such as 16 kbit/s/ch. The side information required to encode the subband transmission sequence is carefully handled in our implementation so that the overall overhead will not have significant impact on the audio quality even at very-low-bit-rates. Note that Shen *et al.* [117] proposed a subband selection rule to achieve progressive coding. However, Shen's scheme demands a large amount of overhead in coding the selection order.

Experimental results show that, when compared with MPEG AAC, the decoded multichannel audio generated by the proposed PSMAC's MNR progressive mode has comparable quality at high-bit-rates, such as 64 kbits/s/ch or 48 kbits/s/ch and much better quality at low-bit-rates, such as 32 kbits/s/ch or 16 kbits/s/ch. We also demonstrate that our PSMAC codec can provide better quality of single-channel audio when compared with MPEG-4 version-2 generic audio coder at several different bit rates.

The rest of this chapter[1] is organized as follows. Section 11.2 gives an overview of the proposed syntax-rich design. Sections 11.3 and 11.4 describe how the progressive quantization is employed in our system. Section 11.5 discusses some implementation issues. Section 11.6 illustrates the complete compression system. Some experimental results are shown in Section 11.7. Finally, conclusion remarks are given in Section 11.8.

11.2. Progressive syntax-rich codec design

In the proposed progressive syntax-rich codec, the following three user defined profiles are provided.

(1) *MNR progressive profile*. If the flag of this profile is on, it should be possible to decode the first N bytes of the bitstream per second, where N is a user specified value or a value that the current network parameters allowed.

(2) *Random access profile*. If the flag of this profile is present, the codec will be able to independently encode a short period of audio more precisely than other

[1]Part of this chapter represents work published before, see [139, 140, 135, 143].

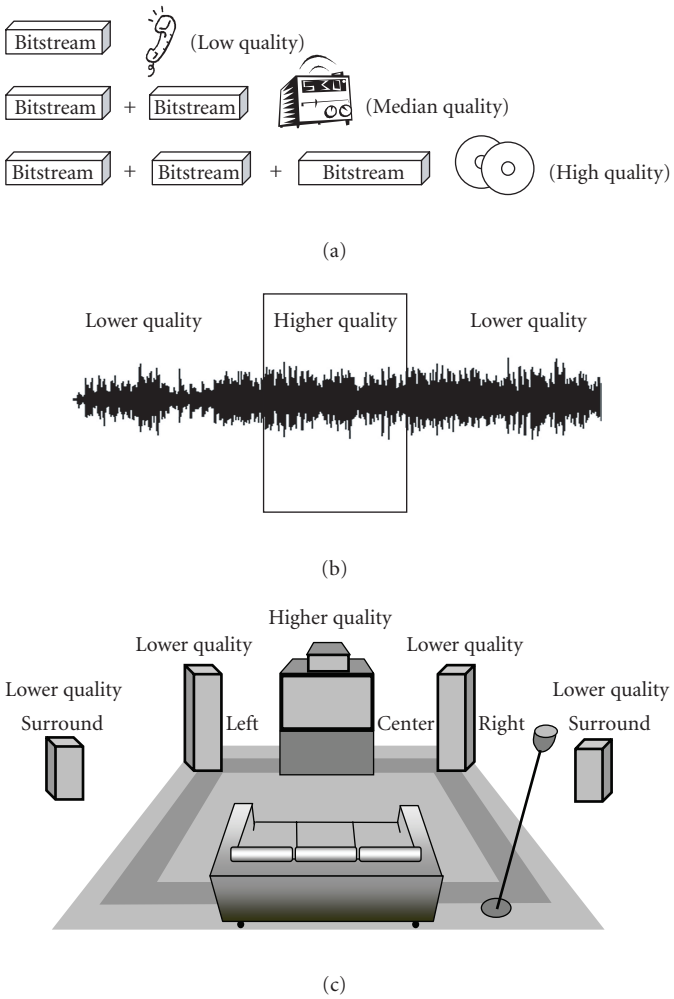


FIGURE 11.1. Illustration of three user-defined profiles: (a) the MNR progressive profile, (b) the random access profile, and (c) the channel enhancement with the enhanced center channel.

periods. It allows users to randomly access a certain part of audio that is more of interest to end users.

(3) *Channel enhancement profile*. If the flag of this profile is on, the codec will be able to independently encode an audio channel more precisely than other channels. Either these channels are of more interest to end users or the network situation does not allow the full multichannel audio bitstream to be received on time.

Figure 11.1 illustrates a simple example of three user defined profiles. Among all profiles, the MNR progressive profile is the default one. In the other two profiles,

that is, the random access and the channel enhancement, the MNR progressive feature is still provided as a basic functionality and decoding of the bitstream can be stopped at any arbitrary point. With these three profiles, the proposed codec can provide a versatile set of functionalities desirable in variable bandwidth network conditions with different user access bandwidths.

11.3. Scalable quantization and entropy coding

The major difference between the proposed progressive audio codec and other existing nonprogressive audio codecs such as AAC lies in the quantization block and the entropy coding block. The dual iteration loop used in AAC to calculate the quantization step size for each frame's and each channel's coefficients is replaced by a progressive quantization block. The Huffman coding block used in the AAC to encode quantized data is replaced by a context-based QM coder. This will be explained in detail below.

11.3.1. Successive approximation quantization

The most important component of the quantization block is called successive approximation quantization (SAQ). The SAQ scheme, which is adopted by most embedded wavelet coders for progressive image coding, is crucial to the design of embedded coders. The motivation for successive approximation is built upon the goal of developing an embedded code that is in analogy to finding an approximation of binary-representation to a real number [116]. Instead of coding every quantized coefficient as one symbol, SAQ processes the bit representation of coefficients via bit layers sliced in the order of their importance. Thus, SAQ provides a coarse-to-fine, multiprecision representation of the amplitude information. The bitstream is organized such that a decoder can immediately start reconstruction based on the partially received bitstream. As increasingly more bits are received, more accurate coefficients and higher quality multichannel audio can be reconstructed.

11.3.1.1. Description of the SAQ algorithm

SAQ sequentially applies a sequence of thresholds $T_0, T_1, \ldots, T_{N+1}$ for refined quantization, where these thresholds are chosen such that $T_i = T_{i-1}/2$. The initial threshold T_0 is selected such that $|C(i)| < 2T_0$ for all transformed coefficients in one subband, where $C(i)$ represents the ith spectral coefficient in the subband. To implement SAQ, two separate lists, the dominant list, and the subordinate list, are maintained both at the encoder and the decoder sides. At any point of the process, the dominant list contains the coordinates of those coefficients that have not yet been found to be significant. While the subordinate list contains magnitudes of those coefficients that have been found to be significant. The process that updates the dominate list is called the significant pass, and the process that updates the subordinate list is called the refinement pass.

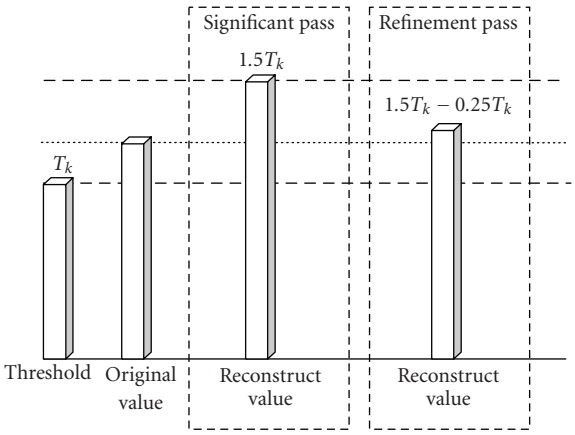


FIGURE 11.2. An example to show how the decoder reconstructs a single coefficient after one significant pass and one refinement pass.

In the proposed algorithm, SAQ is adopted as the quantization method for each spectral coefficient within each subband. This algorithm (for the encoder part) is listed below.

Successive approximation quantization (SAQ) algorithm.

(1) Initialization. For each subband, find out the maximum absolute value $C_{\max}$ for all coefficients $C(i)$ in the subband, and set the initial quantization threshold to be $T_0 = C_{\max}/2 + BIAS$, where $BIAS$ is a small constant.

(2) Construction of the significant map (significance identification). For each $C(i)$ contained in the dominant list, if $|C(i)| \geq T_k$, where T_k is the threshold of the current layer (layer k), add i to the significant map, remove i from the dominant list, and encode it with '1s,' where 's' is the sign bit. Moreover, modify the coefficient's value to

$$C(i) \longleftarrow \begin{cases} C(i) - 1.5 \times T_k, & \forall C(i) > 0, \\ C(i) + 1.5 \times T_k, & \text{otherwise.} \end{cases} \tag{11.1}$$

(3) Construction of the refinement map (refinement). For each $C(i)$ contained in the significant map, encode the bit at layer k with a refinement bit 'D' and change the value of $C(i)$ to

$$C(i) \longleftarrow \begin{cases} C(i) - 0.25 \times T_k, & \forall C(i) > 0, \\ C(i) + 0.25 \times T_k, & \text{otherwise.} \end{cases} \tag{11.2}$$

(4) Iteration. Set $T_{k+1} = T_k/2$ and repeat steps 2–4 for $k = 0, 1, 2, \ldots$.

At the decoder side, the decoder performs similar steps to reconstruct coefficients' values. Figure 11.2 gives a simple example to show how the decoder reconstructs a single coefficient after one significant pass and one refinement pass.

As illustrated in this figure, the magnitude of this coefficient is recovered to 1.5 times of the current threshold T_k after the significant pass, and then refined to $1.5T_k - 0.25T_k$ after the first refinement pass. As more refinement steps follow, the magnitude of this coefficient will approach its original value gradually.

11.3.1.2. Analysis of error reduction rates

The following two points have been observed before [78].

(1) The coding efficiency of the significant map is always better than that of the refinement map at the same layer.

(2) The coding efficiency of the significant map at the kth layer is better than that of the refinement map at the $(k-1)$th layer.

In the following, we would like to provide a formal proof by analyzing the error reduction capability due to the significant pass and the refinement pass, respectively.

First, we consider the error reduction capability for the bit-layer coding of coefficient $C(i)$, for all i, in the significant pass. Since the sign of each coefficient will be coded separately, we will assume $C(i) > 0$ in the discussion below without loss of generality. Suppose that $C(i)$ becomes significant at layer k. This means $T_k \le C(i) < T_{k-1} = 2T_k$ and its value is modified accordingly. Then, error reduction Δ_1 due to the coding of this bit can be found as

$$\Delta_1 = C(i) - |C(i) - 1.5 \times T_k|. \tag{11.3}$$

Note that, at any point of the process, the value of $|C(i)|$ is nothing else but the remaining coding error. Since $T_k \le C(i) < 2T_k$, $-0.5T_k < C(i) - 1.5T_k \le 0.5T_k$, we have $|C(i) - 1.5T_k| \le 0.5T_k$. Consequently,

$$\Delta_1 = C(i) - |C(i) - 1.5 \times T_k| \ge 0.5T_k. \tag{11.4}$$

Now, we calculate the error reduction for the bit-layer coding of coefficient $C(j)$, for all j, in the refinement pass. Similar to the previous case, we assume $C(j) > 0$. At layer k, suppose $C(j)$ is being refined, and its value is modified accordingly. The corresponding error reduction is

$$\Delta_2 = C(j) - |C(j) - 0.25 \times T_k|. \tag{11.5}$$

Two cases have to be considered.

(1) If $C(j) \ge 0.25T_k$,

$$\Delta_2 = C(j) - C(j) + 0.25T_k = 0.25T_k. \tag{11.6}$$

(2) If $C(j) < 0.25T_k$,

$$\Delta_2 = C(j) + C(j) - 0.25T_k = 2C(j) - 0.25T_k < 0.5T_k - 0.25T_k = 0.25T_k. \tag{11.7}$$

Thus, we conclude that

$$\Delta_2 = C(j) - |C(j) - 0.25 \times T_k| \le 0.25T_k < 0.5T_k \le \Delta_1. \tag{11.8}$$

Thus, the error reduction for a significant pass is always greater than that of the refinement pass at the same layer.

Similarly, at layer $(k-1)$, the error reduction for coefficient $C(j)$, for all j, caused by the refinement pass is

$$\Delta_3 = C(j) - |C(j) - 0.25 \times T_{k-1}| \le 0.25T_{k-1} = 0.5T_k \le \Delta_1, \tag{11.9}$$

which demonstrates that error reduction in the significant pass at layer k is actually greater than or equal to that of the refinement pass at layer $(k-1)$.

According to the above analysis, a refinement-significant map coding is proposed and adopted in our progressive multichannel audio codec. That is, the transmission of kth refinement map of subband i is followed immediately by the transmission of $(k+1)$th significant map of subband i.

11.3.1.3. Analysis of error bounds

Suppose the ith coefficient $C(i)$ has a value $T_0/2^{R+1} \le |C(i)| < T_0/2^R$. Then, its binary representation can be written as

$$\begin{aligned} C(i) &= \text{sign} \times \left[a_0\left(\frac{T}{2^0}\right) + a_1\left(\frac{T}{2^1}\right) + a_2\left(\frac{T}{2^2}\right) + \cdots \right] \\ &= \text{sign} \times \sum_{k=0}^{\infty} a_k\left(\frac{T}{2^k}\right), \end{aligned} \tag{11.10}$$

where $T = T_0/2^{R+1}$, T_0 is the initial threshold, and $a_0, a_1, a_2, \ldots$ are binary values (either 0 or 1).

In the SAQ algorithm, $C(i)$ is represented by:

$$\begin{aligned} C(i) &= \text{sign} \times \left[1.5a_0\left(\frac{T}{2^0}\right) + 0.5b_1\left(\frac{T}{2^1}\right) + 0.5b_2\left(\frac{T}{2^2}\right) + \cdots \right] \\ &= \text{sign} \times \left[1.5a_0\left(\frac{T}{2^0}\right) + 0.5\sum_{k=1}^{\infty} b_k\left(\frac{T}{2^k}\right) \right], \end{aligned} \tag{11.11}$$

where a_k and b_k are related via

$$a_k = 0.5(b_k + 1), \quad \forall k = 1, 2, 3, \ldots, \tag{11.12}$$

or

$$b_k = \begin{cases} 1, & a_k = 1, \\ -1, & a_k = 0, \end{cases} \quad \forall k = 1, 2, 3, \ldots. \tag{11.13}$$

Based on the first $M+1$ bits $a_0, a_1, a_2, \ldots, a_M$, the reconstructed value $R_1(i)$ by using the binary representation is

$$\begin{aligned} R_1(i) &= \text{sign} \times \left[a_0\left(\frac{T}{2^0}\right) + a_1\left(\frac{T}{2^1}\right) + a_2\left(\frac{T}{2^2}\right) + \cdots + a_M\frac{T}{2^M} \right] \\ &= \text{sign} \times \left[\sum_{k=0}^{M} a_k\left(\frac{T}{2^k}\right) \right]. \end{aligned} \tag{11.14}$$

Based on the first $M+1$ bits $a_0, b_1, b_2, \ldots, b_M$, the reconstructed value $R_2(i)$ by using SAQ is

$$\begin{aligned} R_2(i) &= \text{sign} \times \left[1.5a_0\left(\frac{T}{2^0}\right) + 0.5b_1\left(\frac{T}{2^1}\right) + 0.5b_2\left(\frac{T}{2^2}\right) + \cdots + 0.5b_M\frac{T}{2^M} \right] \\ &= \text{sign} \times \left[1.5a_0\left(\frac{T}{2^0}\right) + 0.5\sum_{k=1}^{M} b_k\left(\frac{T}{2^k}\right) \right]. \end{aligned} \tag{11.15}$$

Thus, the error introduced by the binary representation for this coefficient is

$$\begin{aligned} E_1(i) &= \left| C(i) - R_1(i) \right| = \left| \sum_{k=M+1}^{\infty} a_k\frac{T}{2^k} \right| \\ &\leq \sum_{k=M+1}^{\infty} \frac{T}{2^k} = \frac{T}{2^M}. \end{aligned} \tag{11.16}$$

Similarly, the error introduced by SAQ for this coefficient is

$$\begin{aligned} E_2(i) &= \left| C(i) - R_2(i) \right| = \left| 0.5 \times \sum_{k=M+1}^{\infty} b_k\frac{T}{2^k} \right| \\ &\leq 0.5 \sum_{k=M+1}^{\infty} \frac{T}{2^k} = \frac{T}{2^{M+1}}. \end{aligned} \tag{11.17}$$

We conclude that the upper bound of both error $E_1(i)$ caused by the binary representation and error $E_2(i)$ caused by SAQ are decaying exponentially when the incoming number of bits M is increasing linearly.

11.3.2. Context-based QM coder

The QM coder is a binary arithmetic-coding algorithm designed to encode data formed by a binary symbol set. It was the result of the effort by JPEG and JBIG committees, in which the best features of various arithmetic coders are integrated. The QM coder is a lineal descendent of the Q-coder, but significantly enhanced by

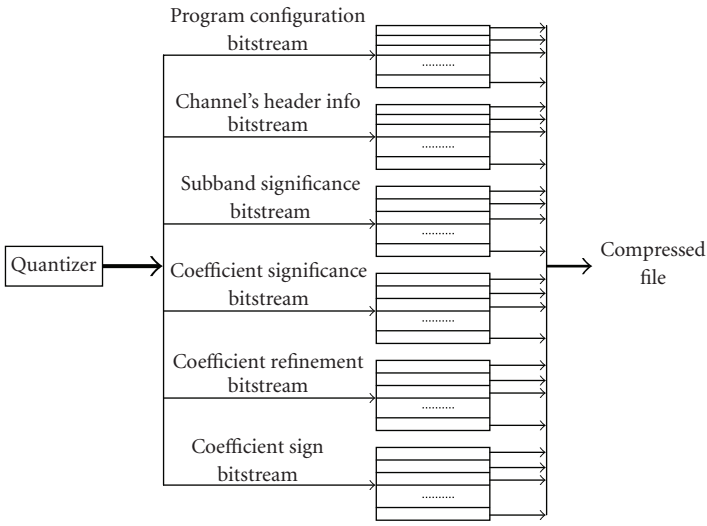


FIGURE 11.3. The adopted context-based QM coder with six classes of contexts.

improvements in the two building blocks, that is, interval subdivision and probability estimation [99]. Based on the Bayesian estimation, a state-transition table, which consists of a set of rules to estimate the statistics of the bitstream depending on the next incoming symbols, can be derived. The efficiency of the QM coder can be improved by introducing a set of context rules. The QM arithmetic coder achieves a very good compression result if the context is properly selected to summarize the correlation between coded data.

Six classes of contexts are used in the proposed embedded audio codec as shown in Figure 11.3. They are the general context, the constant context, the subband significance context, the coefficient significance context, the coefficient refinement context, and the coefficient sign context. The general context is used in the coding of the configuration information. The constant context is used to encode different channel's header information. As their names suggest, the subband significance context, the coefficient significance context, the coefficient refinement context, and the coefficient sign context are used to encode the subband significance, coefficient significance, coefficient refinement, and coefficient sign bits, respectively. These contexts are adopted because different classes of bits may have different probability distributions. In principle, separating their contexts should increase the coding performance of the QM coder.

11.4. Channel and subband transmission strategy

11.4.1. Channel selection rule

In the embedded multichannel audio codec, we should put the most important bits (in the rate-distortion sense) to the cascaded bitstream first so that the decoder

can reconstruct the optimal quality of multichannel audio given a fixed number of bits received. Thus, the importance of channels should be determined for an appropriate ordering of the bitstream. For the normal 5.1 channel configuration, it was observed in Chapter 9 that the channel importance order will be eigen-channel one, followed by eigen-channels two and three, and then followed by eigen-channels four and five. Between each channel pair, the importance is determined by their energy. This policy is used in this chapter.

11.4.2. Subband selection rule

In principle, any quality assessment of an audio channel can be either performed subjectively by employing a large number of expert listeners or done objectively by using an appropriate measuring technique. While the first choice tends to be an expensive and time-consuming task, the use of objective measures provides quick and reproducible results. An optimal measuring technique would be a method that produces the same results as subjective tests while avoiding all problems associated with the subjective assessment procedure. Nowadays, the most prevalent objective measurement is the mask-to-noise-ratio (MNR) technique, which was first introduced by Brandenburg [17] in 1987. It is the ratio of the masking threshold with respect to the error energy. In our implementation, the masking is calculated from the general psychoacoustic model of the AAC encoder. The psychoacoustic model calculates the maximum distortion energy which is masked by the signal energy, and outputs the signal-to-mask-ratio (SMR).

A subband is masked if the quantization noise level is below the masking threshold so the distortion introduced by the quantization process is not perceptible to human ears. As discussed earlier, SMR represents the human auditory response to the audio signal. If SNR of an input audio signal is high enough, the noise level will be suppressed below masking threshold and the quantization distortion will not be perceived. Since SNR can be easily calculated by

$$\mathrm{SNR} = \frac{\sum_i |S_{\mathrm{original}}(i)|^2}{\sum_i |S_{\mathrm{original}}(i) - S_{\mathrm{reconstruct}}(i)|^2}, \tag{11.18}$$

where $S_{\mathrm{original}}(i)$ and $S_{\mathrm{reconstruct}}(i)$ represent the ith original and the ith reconstructed audio signal value, respectively. Thus, MNR is just the difference of SNR and SMR (in dB), or

$$\mathrm{SNR} = \mathrm{MNR} + \mathrm{SMR}. \tag{11.19}$$

A side benefit of the SAQ technique is that an operational rate versus distortion plot (or, equivalently, an operational rate versus the current MNR value) for the coding algorithm can be computed online.

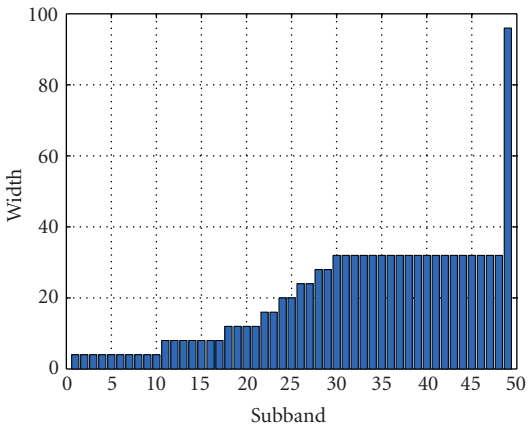


FIGURE 11.4. Subband's width distribution used in AAC for 44.1 kHz and 48 kHz sampling frequencies and long block frames.

The basic ideas behind choosing the subband selection rules are simple. They are

(1) the subband with a better rate deduction capability should be chosen earlier to enhance the performance,
(2) the subband with a smaller number of coefficients should be chosen earlier to reduce the computational complexity, if the rate reduction performances of two subbands are close.

The first rule implies that we should allocate more bits to those subbands with larger SMR values (or smaller MNR values). In other words, we should send out bits belonging to those subbands with larger SMR values (or smaller MNR values) first. The second rule tells us how to decide the subband scanning order. We know that for subband formation in MPEG AAC, the number of coefficients in each subband is nondecreasing with the increase of the subband number. Figure 11.4 shows the subband width distribution used in AAC for 44.1 kHz and 48 kHz sampling frequencies and long block frames. Thus, a sequential subband scanning order from the lowest number to the highest number is adopted in this work.

In order to save bits, especially at very-low-bit-rates, only information corresponding to lower subbands will be sent into the bitstream at the first layer. When the number of layers increase, increasingly more subbands will be added. Figure 11.5 shows how subbands are scanned for the first several layers. At the base layer, the priority is given to lower frequency signals, so that only subbands numbered up to L_B will be scanned. As the information of enhancement layers is added to the bitstream, the subband scanning upper limit increases (as indicated by values of L_{E1}, L_{E2}, and L_{E3} as shown in Figure 11.5) until it reaches the effective psychoacoustic upper bound of all subbands N. In our implementation, we choose $L_{E3} = N$, which means that all subbands are scanned after the third enhancement layer. Here, the subband scanning upper limits in different layers, that

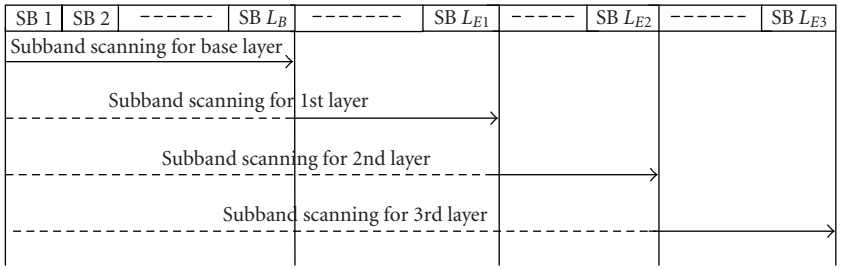


FIGURE 11.5. Illustration of the subband scanning rule, where the solid line with an arrow means all subbands inside this area are scanned, and the dashed line means only those nonsignificant subbands inside the area are scanned.

is, L_B, L_{E1}, and L_{E2} are empirically determined values that provide a good coding performance.

In PSMAC, a dual-threshold coding technique is proposed in the progressive quantization module. One is the MNR threshold, which is used in subband selection. The other is the magnitude threshold, which is used for coefficients' quantization in each selected individual subband. A subband that has its MNR value smaller than the current MNR threshold is called the significant subband. Similar to the SAQ process for coefficient quantization, two lists, that is, the dominant subband list and the subordinate subband list, are maintained in the encoder and the decoder. The dominant subband list contains the indices of those subbands that have not yet become significant, and the subordinate subband list contains the indices of those subbands that have already become significant. The process that updates the subband dominant list is called the subband significant pass, and the process that updates the subband subordinate list is called the subband refinement pass.

Different coefficient magnitude thresholds are maintained in different subbands. We would like to deal with the most important subbands first and get the best result with only a little amount of information from the resource. Moreover, since sounds in different subbands have different impacts on human ears according to the psychoacoustic model, it is worthwhile to consider each subband independently, rather than all subbands in one frame simultaneously.

We summarize the subband selection rule below.

Subband selection procedure.

(1) *MNR threshold calculation*. Determine empirically the MNR threshold value $T_{i,k}^{\mathrm{MNR}}$ for channel i at layer k. Subbands with smaller MNR value at the current layer are given higher priority.

(2) *Subband dominant pass*. For those subbands that are still in the dominant subband list, if subband j in channel i has the current MNR value $\mathrm{MNR}_{i,j}^{k} < T_{i,k}^{\mathrm{MNR}}$, add subband j of channel i into the significant map, remove it from the dominant subband list, send 1 to the bitstream, indicating this subband is selected. Then, apply SAQ to coefficients in this subband. For subbands that have $\mathrm{MNR}_{i,j}^{k} \geq T_{i,k}^{\mathrm{MNR}}$, send 0 to the bitstream, indicating this subband is not selected in this layer.

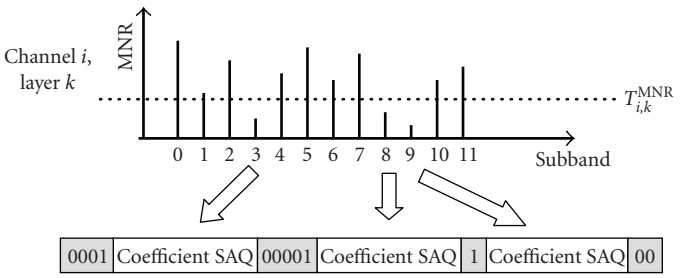


FIGURE 11.6. An example of the subband selection rule.

(3) *Subband refinement pass*. For subbands already in the subordinate list, do coefficient SAQ.

(4) *MNR values update*. Recalculate and update MNR values for selected subbands.

(5) Repeat steps 1–4 until the bitstream meets the target rate.

Figure 11.6 gives a simple example of the subband selection rule. Suppose that, at layer k, channel i has the MNR threshold equal to $T_{i,k}^{\mathrm{MNR}}$. In this example, among all scanned subbands, that is, subbands 0 to 11, only subbands 3, 8, and 9 have current MNR values less than $T_{i,k}^{\mathrm{MNR}}$. Therefore, according to rule 2, three 0 bits and one 1 bit are first sent into the bitstream indicating nonsignificant subbands 0, 1, 2, and significant subband 3. These subband selecting bits are represented in the shaded area in Figure 11.6. Similarly, subband selecting bits for subbands 4 to 11 are also illustrated in shaded areas. Coefficients' SAQ bits of significant subbands are sent immediately after each significant subband bit as shown in this example.

11.5. Implementation issues

11.5.1. Frame, subband, or channel skipping

As mentioned earlier, each subband has its own initial coefficient magnitude threshold. This threshold has to be included in the bitstream as the overhead so that the decoder can start to reconstruct these coefficients once the layered information is available. In our implementation, the initial coefficient magnitude threshold $T_{i,j}(0)$ for channel i and subband j will be truncated to the nearest power of 2 that is not less than $C_{i,j}^{\max}$, that is,

$$T_{i,j}(0) = 2^{p_{i,j}}, \quad p_{i,j} = \lceil \log_2 C_{i,j}^{\max} \rceil, \tag{11.20}$$

where $C_{i,j}^{\max}$ is the maximum magnitude for all coefficients in channel i and subband j.

In order to save bits, the maximum power $p_i^{\max} = \max(p_{i,j})$, for all j, for all subbands in channel i will be included in the bitstream at the first time when channel i is selected. A relative value of each subband's maximum power, that is,

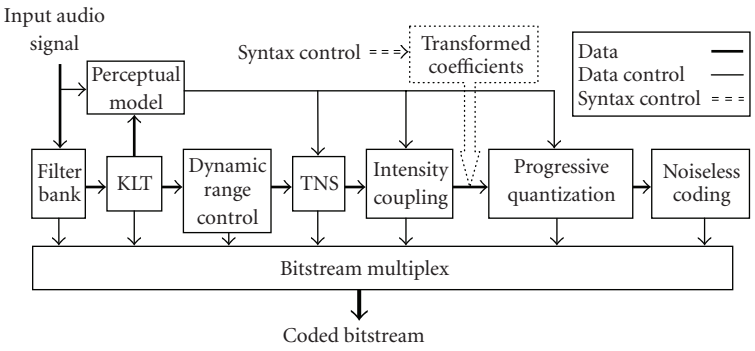


Figure 11.7. The block-diagram of the proposed PSMAC encoder.

the difference $\Delta p_{i,j} = p_i^{\max} - p_{i,j}$ between $p_i^{\max}$ and $p_{i,j}$, will be included in the bitstream at the first time when the selected subband becomes significant.

For a frame with a maximum value $C_{i,j}^{\max}$ equal to zero, that is, $\max(C_{i,j}^{\max}) = 0$, for all j, which means all coefficients in channel i in this frame have value 0, then a special indicator will be set to let the decoder know it should skip this frame. Similarly, if $C_{i,j}^{\max}$ has value 0, another special indicator is set to tell the decoder that it should always skip this subband. In some cases when the end user is only interested in reconstructing some channels, channel skipping can also be adopted.

11.5.2. Determination of the MNR threshold

At each layer, the MNR threshold for each channel is determined empirically. Two basic rules are adopted when calculating this threshold.

(1) The MNR threshold should allow a certain number of subbands to pass at each layer. Since the algorithm sends zero to the bitstream for each unselected subband which is still in the significant subband list, if the MNR threshold is so small that it allows too few subbands to pass, too many overhead bits will be generated. As a result, this will degrade the performance of the progressive audio codec.

(2) Adopt a maximum MNR threshold. If the MNR threshold calculated by using the above rule is greater than a predefined maximum MNR threshold $T_{\max}^{\mathrm{MNR}}$, then the current MNR threshold for channel i at kth layer $T_{i,k}^{\mathrm{MNR}}$ will be set to $T_{\max}^{\mathrm{MNR}}$. This is based on the assumption that a higher MNR value does not provide higher perceptual audio quality perceived by the human auditory system.

11.6. Complete description of PSMAC codec

The block diagram of a complete encoder is shown in Figure 11.7. The perceptual model, the filter bank, the temporal noise shaping (TNS), and the intensity blocks in our progressive encoder are borrowed from the AAC main profile encoder. The interchannel redundancy removal procedure via KLT is implemented after the input audio signals are transformed into the MDCT domain. Then, a dynamic range

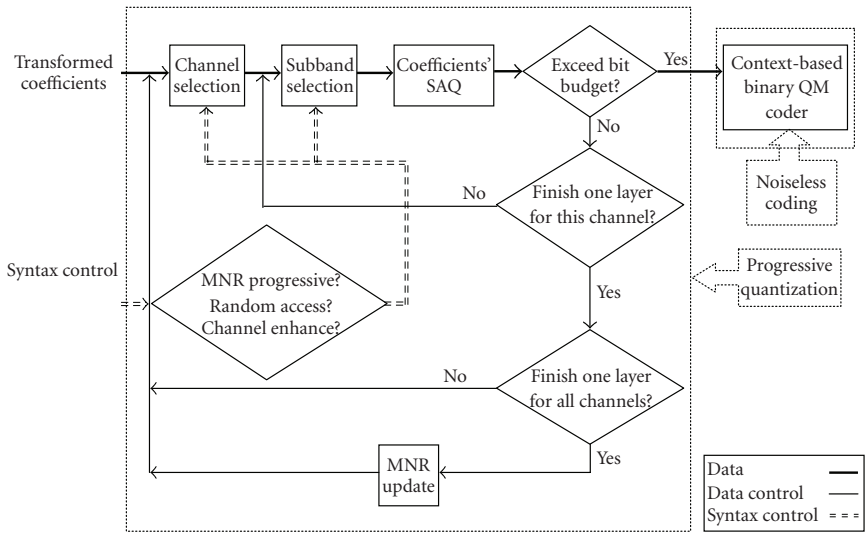


FIGURE 11.8. Illustration of the progressive quantization and lossless coding blocks.

control block follows to avoid any possible data overflow in later compression stages. Masking thresholds are then calculated in the perceptual model based on the KL transformed signals. The progressive quantization and lossless coding parts are finally used to construct the compressed bitstream. The information generated at the first several coding blocks will be sent into the bitstream as the overhead.

Figure 11.8 provides more details of the progressive quantization block. The channel and the subband selection rules are used to determine which subband in which channel should be encoded at this point, and then coefficients within this selected subband will be quantized via SAQ. The user defined profile parameter is used for the syntax control of the channel selection and the subband selection. Finally, based on several different contexts, the layered information together with all overhead bits generated during previous coding blocks will be losslessly coded by using the context-based QM coder.

The encoding process performed by using the proposed algorithm will stop when the bit budget is exhausted. It can cease at any time, and the resulting bitstream contains all lower rate coded bitstreams. This is called the fully embedded property. The capability to terminate the decoding of an embedded bitstream at any specific point is extremely useful in a coding system that is either rate-constrained or distortion-constrained.

11.7. Experimental results

The proposed PSMAC system has been implemented and tested. The basic audio coding blocks [62] inside the MPEG AAC main profile encoder, including the psychoacoustic model, filter bank, temporal noise shaping, and intensity/coupling,

Table 11.1. MNR comparison for MNR progressive profiles.

Bit rate (bit/s/ch)	Average MNR values (dB/subband/ch)			
	Herre		Messiah	
	AAC	PSMAC	AAC	PSMAC
16 k	−0.90	6.00	14.37	21.82
32 k	5.81	14.63	32.40	34.57
48 k	17.92	22.32	45.13	42.81
64 k	28.64	28.42	54.67	47.84

are still adopted. Furthermore, an interchannel removal block, a progressive quantization block, and a context-based QM coder block are added to construct the PSMAC audio codec.

Two types of experimental results are shown in this section. One is measured by an objective metric, that is, the mask-to-noise ratio (MNR), and the other is measured in terms of a subjective metric, that is, listening test score. It is worthwhile to mention that the coding blocks adopted from AAC have not been modified to improve the performance of the proposed PSMAC codec for fair comparison. Moreover, the test audio that produced the worst performance by the MPEG reference code was not selected in the experiment.

11.7.1. Results using MNR measurement

Two multichannel audio materials are used in this experiment to compare the performance of the proposed PSMAC algorithm with MPEG AAC's [62] main profile codec. One is a one-minute long ten-channel[2] audio material called "Messiah," which is a piece of classical music recorded live in a real concert hall. Another one is an eight-second long five-channel[3] music called "Herre," which is a piece of pop music and was used in the MPEG-2 AAC standard (ISO/IEC 13818-7) conformance work.

11.7.1.1. MNR progressive

The performance comparison of MPEG AAC and the proposed PSMAC for the normal MNR progressive mode are shown in Table 11.1. The average MNR shown in the table is calculated by equation (10.7) and (10.8).

Table 11.1 shows the MNR values for the performance comparison of the nonprogressive AAC algorithm and the proposed PSMAC algorithm when working in the MNR progressive profile. Values in this table clearly show that our codec outperforms AAC for both testing materials at lower bit rates and only has a small performance degradation at higher bit rates. In addition, the bitstream generated

[2]The ten channels include center (C), left (L), right (R), left wide (LW), right wide (RW), left high (LH), right high (RH), left surround (LS), right surround (RS), and back surround (BS).

[3]The five channels include C, L, R, LS, and RS.

Table 11.2. MNR comparison for random access and channel enhancement profiles.

Random access		Channel enhancement			
		Enhanced channel		Other channels	
Other area	Enhanced area	w/o enhance	w/ enhance	w/o enhance	w/ enhance
3.99	13.94	8.42	19.23	1.09	−2.19

by MPEG AAC only achieves an approximate bit rate and is normally a little bit higher than the desired one while our algorithm achieves a much more accurate bit rate in all carried out experiments.

11.7.1.2. Random access

The MNR result after the base layer reconstruction for the random access mode by using the test material "Herre" is shown in Table 11.2. When listening to the reconstructed music, we can clearly hear the quality difference between the enhanced period and the rest of the other period. The MNR value given in Table 11.2 verifies the above claim by showing that the mean MNR value for the enhanced period is about 10 dB per subband better than the rest of other periods. It is common that we may prefer a certain part of a music to others. With the random access profile, the user can individually access a period of music with better quality than others when the network condition does not allow a full high quality transmission.

11.7.1.3. Channel enhancement

The performance results using test material "Herre" for the channel enhancement mode is also shown in Table 11.2. Here, the center channel has been enhanced with enhancement parameter 1. Note that the total bit rate is kept the same for both codecs, that is, each has an average bit rate of 16 kbit/s/ch. Since we have to separate the quantization and the coding control of the enhanced physical channel as well as simplify the implementation, KLT is disabled in the channel enhancement mode. Compared with the normal MNR progressive mode, we find that the enhanced center channel has an average of more than 10 dB per subband MNR improvement, while the quality of other channels is only degraded by about 3 dB per subband.

When an expert subjectively listens to the reconstructed audio, the one with the center channel enhanced has a much better performance and is more appealing, compared with the one without channel enhancement. This is because the center channel of "Herre" contains more musical information than other channels and a better reconstructed center channel will give listeners better overall quality, which is basically true for most multichannel audio materials. Therefore, this experiment suggests that with a narrower bandwidth, audio generated by the channel enhancement mode of the PSMAC algorithm can provide the user a more compelling experience with either a better reconstructed center channel or a channel which is more interesting to a particular user.

11.7.2. Subjective listening tests

In order to further confirm the advantage of the proposed algorithm, a formal subjective listening test according to ITU recommendations [1, 2, 3] was conducted in an audio lab to compare the coding performance of PSMAC and that of the MPEG AAC main profile codec. The same group of listeners described in Section 9.8.3 participated in the listening test. During the test, for each test sound clips, the subjects listened to three versions of the same sound clips, that is, the original one followed by two processed ones (one by PSMAC and one by AAC in random order). The subjects were allowed to listen to these files as many times as possible until they were comfortable to give scores to the two processed sound files for each test material.

The five-grade impairment scale given in Recommendation ITU-R BS. 1284 [2] was adopted in the grading procedure and utilized for final data analysis. Besides "Messiah" and "Herre," two ten-channel audio materials called "Band" and "Herbie" were included in this subjective listening test, where "Band" is a live 16 microphone recording of a marching band playing a rock and roll song in a football field, and "Herbie" is a studio recording of a jazz piece remixed for 10.2 from 48 original tracks. According to ITU-R BS. 1116-1 [1], audio files selected for the listening test were of short duration, that is, 10 to 20 seconds long.

Figure 11.9 shows the score given to each test material coded at four different bit rates during the listening test for multichannel audio materials. The solid vertical line represents the 95% confidence interval, whereas the middle line shows the mean value and the other two lines at the boundary of the vertical line represent the upper and lower confidence limits [104]. It is clear from Figure 11.9 that at lower bit rate, such as 16 kbit/s/ch or 32 kbit/s/ch, the proposed PSMAC algorithm outperforms MPEG AAC in all four test materials. While at a higher bit rate, such as 48 kbit/s/ch or 64 kbit/s/ch, PSMAC achieves comparable or a little degraded subjective quality when compared with MPEG AAC.

To demonstrate that the PSMAC algorithm achieves excellent coding performance even for single-channel audio files, another listening test for mono sound was also carried out. Three single-channel single-instrument test audio materials, known as "GSPI," "TRPT," and "VIOO" were used in this experiment and the performance between the standard fine-grain scalable audio coder provided by MPEG-4 BSAC [61, 66] and the single-channel mode of the proposed PSMAC algorithm was compared.

Figure 11.10 shows the listening test results for the three single-channel audio materials. For cases where no confidence intervals are shown, this means that all four listeners happened to give the same score to the given sound clip. From this figure, we can clearly see that at lower bit rates, for example, 16 kbit/s/ch and 32 kbit/s/ch, our algorithm generates better sound quality for all test sequences. At higher bit rates, for example, 48 kbit/s/ch and 64 kbit/s/ch, our algorithm outperforms MPEG-4 BSAC for two out of three test materials and is only slightly worse for the "TRPT" case.

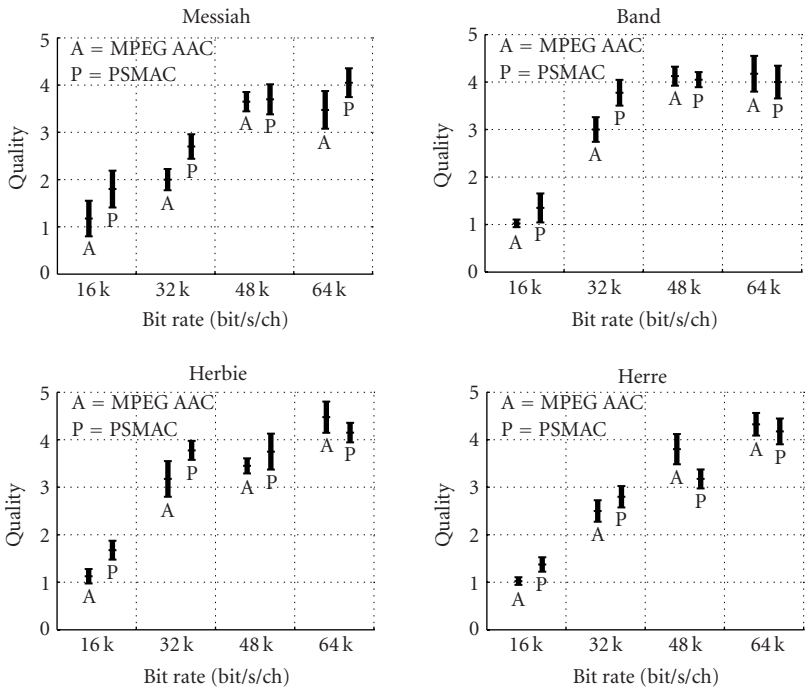


FIGURE 11.9. Listening test results for multichannel audio sources.

11.8. Conclusions

A progressive syntax-rich multichannel audio coding algorithm is presented in this research. This algorithm utilizes KLT in the preprocessing stage to remove interchannel redundancy inherent in the original multichannel audio source. Then, rules for channel selection and subband selection were developed and the SAQ process was used to determine the importance of coefficients and their layered information. At the last stage, all information was losslessly compressed by using the context-based QM coder to generate the final multichannel audio bitstream.

The distinct advantages of the proposed algorithm over most existing multichannel audio codecs not only lie in its progressive transmission property, which can achieve a precise rate control, but also in its rich-syntax design. Compared with the new MPEG-4 BSAC tool, PSMAC provides a more delicate subband selection strategy such that the information, which is more sensitive to the human ear, is reconstructed earlier and more precisely at the decoder side. It was shown by experimental results that PSMAC has a comparable performance to nonprogressive MPEG AAC at several different bit rates when using the multichannel test material, while PSMAC achieves better reconstructed audio quality than MPEG-4 BSAC tools when using single-channel test materials. Moreover, the advantage of the proposed algorithm over the other existing audio codec is more obvious at lower bit rates.

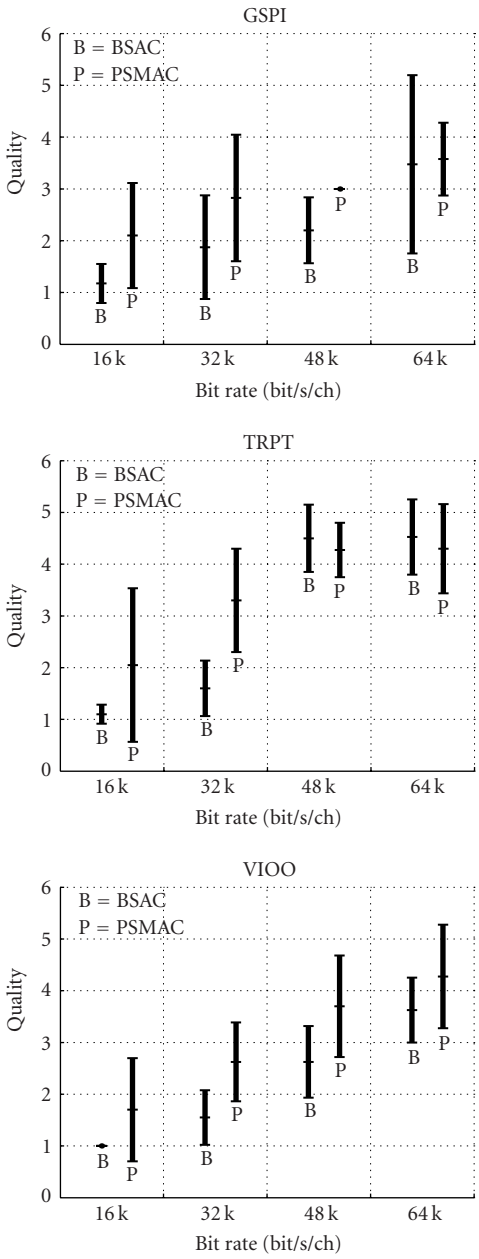


FIGURE 11.10. Listening test results for single-channel audio sources.

12 Error-resilient scalable audio codec design

12.1. Introduction

High quality audio communication becomes an increasingly important part of the global information infrastructure. Compared with speech, audio communication requires a larger volume of data being transmitted in a timely manner, and a highly efficient compression scheme for the storage and transmission of audio data is critical. Extensive research on audio coding has been conducted in both academia and industry for years. Several standards, including AC-3, MPEG-1, MPEG-2, and MPEG-4 [18, 8, 60, 58, 64], have been established in the past decade. Earlier standards, for example, MPEG-1, MPEG-2, or AC-3 were primarily designed for coding efficiency and they only allow a fixed bit rate coding structure. These algorithms are not ideal for audio delivery over noisy wireless IP networks with a time-varying bandwidth, since they do not take error resilience and VBR (variable bit rates) traffic into consideration.

Recent technological developments have led to several mobile systems aiming at personal communications services (PCS), supporting both speech and data transmission. Mobile uses usually communicate over wireless links characterized by lower bandwidths, higher transmission error rates, and more frequent disconnection in comparison to wired networks. To transmit high quality audio through an IP network with VBR traffic, a scalable audio compression algorithm, which is able to transfer audio signals from coarse to fine qualities progressively, is desirable. However, to achieve a good coding gain, most existing scalable techniques adopt variable-length coding in their entropy coding part, which makes the entire bitstream susceptible to channel noise. The traditional channel coding scheme only protects bits equally, without giving important bits higher protection, which results in a situation that a small bit error rate may lead to reconstructed audio with annoying distortion or even unacceptable perceptual quality. Since most audio compression standards and network protocols, such as the MPEG-4 version 2 audio codec, were designed for wired audio communications, they would not be effective if directly applied to the wireless case. A scalable bitstream with joint source-channel coding would be truly needed in a wireless audio streaming system.

MPEG-4 version 2 supports audio fine-grain scalability and error-robust coding [58, 64]. Its error resilient AAC coder does not have the progressive property. Its BSAC utilizes segmented binary arithmetic (SBA) coding to avoid error propagation within spectral data. However, this feature alone is not sufficient to protect the audio data in an effective manner over the wireless channel. Compared to work on error-resilient image/video coding, the number of papers on error resilient audio coding is relatively small. Data partitioning and reversible variable length codes were adopted by Zhou *et al.* in [146] to provide the error-resilient feature to the scalable audio codec in [145]. Based on the framework in [146], Wang *et al.* incorporated an unequal error protection scheme in [131]. In Chapter 11, we proposed a progressive high quality audio coding algorithm, which has been shown to outperform MPEG-4 version 2's scalable audio codec. In this work, we extend the error-free progressive audio codec to an error-resilient scalable audio codec (ERSAC) by reorganizing the bitstream and modifying its noiseless coding part. The proposed error-resilient scalable audio codec actually uses the MPEG advanced audio coding (AAC) as the baseline together with an error robust scalable transmission module, which is specifically designed for WCDMA channels.

In the proposed ERSAC codec, a dynamic segmentation scheme is first used to divide the audio bitstream into several variable-length segments. In order to achieve good error resiliency, the length of each segment is adaptively determined by the characteristics of WCDMA channels. The arithmetic coder and its probability table are reinitialized at the beginning of each segment, so that synchronization can be achieved at the decoder side even when error occurs. Bits within each segment are ordered in such a way that more important bits are placed near the synchronization point. In addition, an unequal error protection scheme is adopted to improve robustness of the final bitstream, where Reed-Solomon codes are used to protect data bits, and the parameters of each Reed-Solomon code is determined by the WCDMA channel condition. Moreover, a frequency interleaving technique is adopted when data packetization is performed so that the frequency information belonging to the same period is sent in different packets. In this way, even if some packets belonging to the header or the base layer are corrupted, we can still hear a poorer quality period of sound with some frequency component lost (unless packets corresponding to the same period of sound are corrupted at the same time). We test the performance of our algorithm using several single-channel audio materials under different error patterns of WCDMA channels. Experimental results show that the proposed approach has excellent error resiliency at a regular user bit rate of 64 kb/s.

The rest of this chapter[1] is organized as follows. Some characteristics of the WCDMA channel are summarized in Section 12.2. The layered audio coding structure is described in Section 12.3. Section 12.4 explains the detailed error-resilient technique in the proposed algorithm. Some experimental results are shown in Section 12.5 and concluding remarks are given in Section 12.6. Some discussions and future work directions are addressed in Section 12.7.

[1] Part of this chapter represents work published before, see [141].

12.2. WCDMA characteristics

The third generation (3G) mobile communication systems have been designed for effective wireless multimedia communication [52]. The wideband direct-sequence code division multiple access (WCDMA) technology has been adopted by the UMTS standard as the physical layer for air interface. WCDMA has the following characteristics.

(1) WCDMA is designed to be deployed in conjunction with GSM.
(2) The chip rate of 3.84 Mcps used leads to a carrier bandwidth of approximately 5 MHz.
(3) WCDMA supports highly variable user data rates; in other words, the concept of obtaining bandwidth on demand (BoD) is well supported.
(4) WCDMA supports two basic modes of operation: frequency division duplex (FDD) and time division duplex (TDD).
(5) WCDMA supports the operation of asynchronous base stations.
(6) WCDMA employs coherent detection on uplink and downlink signals based on the use of pilot symbols or common pilot.
(7) The WCDMA air interface has been crafted in such a way that advanced CDMA receiver concepts can be deployed by the network operator as a system option to increase capacity and/or coverage.

Two reference error-resilient simulation studies [67, 68] for the characterization of the radio channel performance of the 1.9 GHz WCDMA air interface were recently carried out by the ITU-Telecommunications Standardization Sector. In the study of 1998, which is referred to as study #1, only six simulation results for a fixed data bit rate of 64 kb/s were obtained. In the study of 1999, which is referred to as study #2, simulation results were extended to four different data bit rates, including 32 kb/s, 64 kb/s, 128 kb/s, and 384 kb/s. Study #2 also replaced convolutional codes by turbo codes so that an even better channel performance can be achieved. In this work, we only consider error-resilient coding for single-channel audio coded at 64 kb/s. The main characteristics of all error patterns corresponding to 64 kb/s contained in the two studies are listed in Table 12.1. Each error pattern file is 11,520,000 bits long corresponding to 3 minutes of data transmission. The bit error is a binary one. Within a byte the least significant bit is transmitted first.

12.3. Layered coding structure

12.3.1. Advantages of the layered coding

The most popular and efficient way to construct a scalable bitstream is to use the layered coding structure. When the information from the first and the most important layer called the base layer is successfully retrieved from the bitstream at the decoder side, a rough outline of the entire sound file can be recovered. When the information from progressively higher level layers, called enhancement layers, are successfully retrieved from the bitstream, a sound file with increasingly better

Table 12.1. Characteristics of WCDMA error patterns.

Study #	File #	File name	Mobile speed (km/h)	Average BER (b/s)
1	0	WCDMA-64 kb-005 hz-4	3	8.2e-5
1	1	WCDMA-64 kb-070 hz-4	40	1.2e-4
1	2	WCDMA-64 kb-211 hz-4	120	9.4e-5
1	3	WCDMA-64 kb-005 hz-3	3	1.4e-3
1	4	WCDMA-64 kb-070 hz-3	40	1.3e-3
1	5	WCDMA-64 kb-211 hz-3	120	9.7e-4
2	6	WCDMA_64 kb_50 kph_7e-04	50	6.6e-4
2	7	WCDMA_64 kb_50 kph_2e-04	50	1.7e-4
2	8	WCDMA_64 kb_3 kph_5e-04	3	5.1e-4
2	9	WCDMA_64 kb_3 kph_2e-04	3	1.6e-4
2	10	WCDMA_64 kb_3 kph_7e-05	3	7.2e-5
2	11	WCDMA_64 kb_3 kph_3e-06	3	3.4e-6
2	12	WCDMA_64 kb_50 kph_6e-05	50	6.0e-5
2	13	WCDMA_64 kb_50 kph_3e-06	50	3.4e-6

quality can be reconstructed. When the bitstream is transmitted over error-prone channels, such as the wired and/or wireless IP networks, the advantage of the layered coded bitstream is more notable than the fixed rate bitstreams. For a fixed rate bitstream, when an error occurs during transmission, the decoder can only reconstruct the period before the error and the period after the decoder regains the synchronization caused by the error. The resulting sound file may contain the lost period of several milliseconds to several seconds long, depending on how long it takes to regain synchronization at the decoder site. If the bitstream cannot be resynchronized, the data after the error may be completely lost, which results in a partially reconstructed sound file. However, when an error occurs during transmission of a layered coded bitstream, unless the error occurs in the base layer, the decoder can still recover the sound file, but has sound quality degradation in enhancement layers for some period of time. Experiments suggest that, even if it contains poorer quality for some period of time, the reconstruct sound file of full length would give listeners better sensation than a sound file with some completely lost periods.

12.3.2. Main features of scalable codec

The major difference between the proposed scalable codec design and the traditional fixed bit rate codec lies in the quantization module and the entropy coding module. The ERSAC algorithm inherits the basic idea of the progressive quantization and context-based QM coder in PSMAC algorithm in order to achieve the fine-grain bit rate scalability. In order to classify and protect bits according to their importance, bits are reordered so that bits which belong to the same priority are grouped together for easier protection.

Compared with PSMAC algorithm, the major modification in the progressive quantization module lies in how to transmit those subband significant bits. In PSMAC, bits which indicate the subband significance are sent together with the coefficient bits, while in the ERSAC algorithm, these bits are sent in the header. In PSMAC, the threshold used to determine the subband significance is MNR values and they are updated after each coding layer, while in ERSAC, SMR values are adopted for determination of the subband significance in every layer. Thus the subband selecting sequence might not be the same when using PSMAC and ERSAC algorithms for some input audio files. However, experiments show that the perceptual quality of the reconstructed sound files is quite similar when adopting these two slightly different subband selection rules.

In ERSAC, at layer i, $i \leq 3$, an empirical threshold T_i based on the signal-to-mask ratio (SMR) is calculated. Only those subbands whose SMR values are greater or equal to T_i will be selected and become significant. At the next layer, that is, layer $i + 1$, the SMR values of newly included subbands together with the remaining nonsignificant subbands in previous layers will be compared with T_{i+1}, and an updated significant subband list will be created.

Figure 12.1 provides an example to show how subbands are selected from layer 0 to layer 3. To better illustrate this procedure, L_0, L_1, L_2, and L_3 are set to 6, 10, 14, and 18, respectively, in this example. If the bit budget is not exhausted after the third enhancement layer, more layers can be encoded. All subbands will be included in the significant list from this point on. At the encoder, the subband significance information is included in the header, where a binary digit "1" represents a significant subband and "0" represents a nonsignificant subband. Thus, in the example given in Figure 12.1, the subband significance information bits should be

$$0101111011011001100101011. \tag{12.1}$$

Whenever the encoder finds a significant subband, it visits coefficients inside this subband, performs progressive quantization on coefficients, and then does the entropy coding. Here, we adopt the successive approximation quantization (SAQ) scheme to quantize the magnitude of coefficients and the context-based QM coder for the noiseless coding of all generated bits. Detailed description of coefficients' SAQ and context-based QM coder can be found in Chapter 11.

12.4. Error-resilient codec design

12.4.1. Unequal error protection

When a bitstream is transmitted over the network, errors may occur due to channel noise. Even with channel coding, errors can still be found in received audio data. In particular, for highly compressed audio signals, a small bit error rate can lead to highly annoying or perceptually unacceptable distortion. Instead of demanding an even lower bit error rate (BER), which is expensive to achieve in a wireless environment, the use of joint source and channel coders have been studied

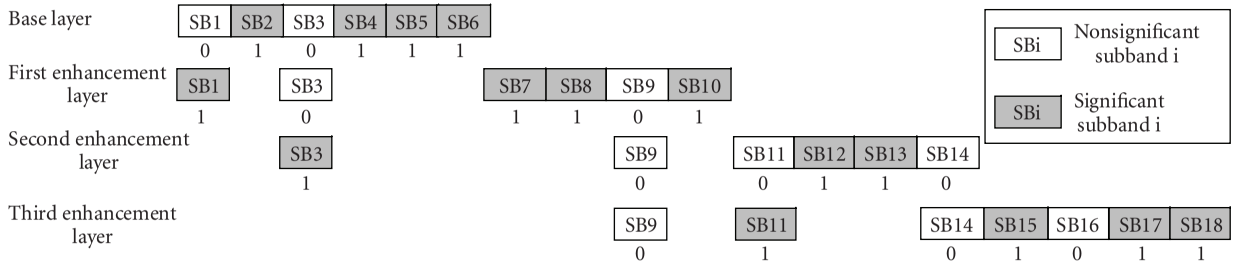


FIGURE 12.1. A simplified example of how subbands are selected from layers 0 to 3.

in [144, 118, 131] and shown to be promising in achieving good perceptual quality without complex processing.

Most coded audio bitstreams contain certain bits that are more sensitive to transmission errors than others in audio reconstruction. Unequal error protection (UEP) offers a mechanism to relate the transmission error sensitivity to the error protection capability. An UEP system typically has the same average transmission rate as a corresponding equal error protection (EEP) system but offers an improved perceived signal quality at the same channel signal-to-noise ratio. In order to prevent the complete loss of transmitted audio, the critical bits need to be well protected from channel errors. Examples of critical bits include the headers and the most significant bits of source symbols. The loss of header's information usually leads to catastrophic sound distortion while an error in the most significant bit results in higher degradation than that of others. Thus, high-priority bits need to be protected using channel coding or other methods. However, the redundancy due to channel coding reduces compression efficiency. A good trade-off between rates of the source coder and the channel coder has to be considered.

The study of error-correcting codes began in late 1940s. Among several error correcting codes [87, 82], such as Hamming, BCH, cyclic, and Reed-Muller codes, the Reed-Solomon code is chosen in our implementation because of its excellent performance on correcting burst errors, which is the most common case in wireless channel transmission. Reed-Solomon codes are block-based error correcting codes with a wide range of applications in digital communications and storage. The number and the type of errors that can be corrected by Reed-Solomon codes depend on code parameters. For a fixed number of data symbols, codes that can detect and correct a smaller number of bit errors have smaller parity check symbols, thus producing smaller redundancy bits. In this work, data in the compressed audio bitstreams are protected according to their error sensitivity classes. Experimental results suggest that errors in both headers and the base layer lead to unacceptable reconstructed audio quality, while the same amount of errors in the enhancement layers results in less perceptual distortion as the number of the enhancement layers increases. Therefore, bits in the header and the base layer are given the same highest priority, bits in the first enhancement layer are given the moderate priority, and bits in the second and higher enhancement layers are given the lowest priority during the error protection procedure.

To further determine the error correcting capability of Reed-Solomon codes used for each error sensitivity class, more detailed analysis is performed on all WCDMA error patterns. Since the Reed-Solomon code is a byte-based error correcting code, the mean and the standard deviation of the byte error rate are calculated for each error file. Files with similar byte error rate characteristics are combined into one group. All fourteen error patterns are finally divided into four groups, whose virtual mean and standard deviation values are then empirically determined. Based on these virtual statistical data, the target error correcting capability of the Reed-Solomon code is finally calculated by the following formula

$$\text{etarget}(g, c) = \text{mean}_{\text{byte}}(g) + f_{\text{byte}}(g, c) \times \text{std}_{\text{byte}}(g), \tag{12.2}$$

where $\text{etarget}(g, c)$, $\text{mean}_{\text{byte}}(g)$, and $\text{std}_{\text{byte}}(g)$ are the target error correcting ability (in percentage), the virtual mean, and the virtual standard deviation of the byte error rate for group g and error sensitive class c, respectively, and $f_{\text{byte}}(g, c)$ is a function of group number g and error sensitivity class number c.

12.4.2. Adaptive segmentation

Although the arithmetic coder has excellent coding efficiency and is adopted by almost all layer-coded source coding techniques, the arithmetic coder together with other variable-length codes are known to be highly susceptible to channel errors due to the synchronization loss at the decoder side, which leads to error propagation, the loss of some source symbols, and even the crash of the decoder. To prevent these undesirable results and, at the same time, to consider the redundancy generated by the UEP scheme, an adaptive segmentation strategy is developed in this work. That is, the generated bitstream is partitioned into several independent segments. By "independent" we mean that the entropy coding module at the decoder side can independently decode each segment. This can be achieved as follows. At the beginning of each segment, the arithmetic coder is restarted, its probability tables are refreshed, and some stuffing bits are appended at the end of each segment so that each segment is byte-aligned. In this way, several independent synchronization points are provided in the bitstream and errors can only propagate until the next synchronization point, which means that errors will be confined to their corresponding segments and will not affect the decoding of other segments.

The determination of the segment length is another issue to be addressed. Since the arithmetic coder has to be restarted and flushed for each segment, segments with a length too small will considerably degrade the entropy coding efficiency. Thus, a good trade-off between the coding performance and the error resilient capability should be studied. The use of the bit error rate as a parameter to determine the segment length provides an intuitive and straightforward solution. However, after some exploration, we find that the error distribution pattern should also be taken into consideration.

Experimental results show that error files with a similar bit error rate may have a quite different error distribution pattern. We introduce a concept called the error occurrence period, which is defined as the length (in bits) between two neighboring errors. Here, it is assumed that there are at least eight or more free bits between these two neighboring errors. The average error occurrence period and its standard deviation are calculated for each error pattern file. Files with similar characteristics are grouped together. Then, the virtual mean and the virtual standard deviation value of the error occurrence period are empirically determined for each group. Finally, the segment length is calculated via

$$\text{seglen}(g) = \text{mean}_{\text{occur}}(g) + f_{\text{occur}}(g) \times \text{std}_{\text{occur}}(g), \tag{12.3}$$

where $\text{seglen}(g)$, $\text{mean}_{\text{occur}}(g)$, and $\text{std}_{\text{occur}}(g)$ are the segment length, the virtual mean, and the virtual standard deviation of the error occurrence period for group

g, respectively, and $f_{\text{occur}}(g)$ is a function of group number g. Note that the group number g in equation (12.3) may not be the same as that in equation (12.2).

12.4.3. Frequency interleaving

Traditionally, all bits belonging to the same time position are packed together and sent into the bitstream. When errors happen in the global header or the data part of the base layer, the corresponding period of sound data cannot be reconstructed. Instead, it may have to be replaced by silence or other error concealment technology. Simple experiments show that substituting the corrupted period with silence in a reconstructed sound file generates an unpleasant sound effect. So far, there is no effective error concealment technology to recover the lost period in audio. In order to improve the performance in this situation, a novel frequency interleaving method is proposed and incorporated in the ERSAC algorithm. With this new method, bits corresponding to different frequency components in the same time position are divided into two groups. Then, even if some packets are corrupted during transmission, the decoder can still be able to reconstruct a poorer quality version of the sound with some frequency component missing.

Figure 12.2 depicts a simple example on how the frequency interleaving is implemented in ERSAC. In this figure, only significant subbands for each layer in Figure 12.1 are shown. Adjacent significant subbands are divided into different groups. Bits belonging to a different group will not be included in the same packet. This way, bits in different subbands, which correspond to different frequency intervals, are interleaved so that a perceptually better decoded sound file can be reconstructed when some packets are corrupted.

To show the advantage of the frequency interleaving method, a simple experiment is carried out. During the experiment, we artificially corrupt two packets for each test sound file. Both packets belong to the data part of the base layer in the bitstream. The corresponding time positions of these packets are chosen such that one packet contains information for a smooth period and the other one contains information for a period with rapid melody variations. With one packet lost in the base layer, coefficients corresponding to certain frequency areas cannot be reconstructed. Therefore, the reconstructed sound clips may contain a period with defects. However, the degree of the perceptual impairment differs a lot from sample to sample. Table 12.2 lists the experiment results for the frequency interleaving method.

We see from this table that, for some input sound files, such as the one named "VIOO," users can hardly detect the defect of the reconstructed file. For some other input sound files, such as the one named "TRPT," users are able to catch the defect in the smooth period, but the perceptual impairment is in the level of "perceived but not annoying." For input sound files like "GSPI," which has a wide range of frequency components, users can easily detect the defect that may be somewhat annoying. On the other hand, if no frequency interleaving is enforced, when corrupted packets reside in the header or the data part of the base layer, no information for the corresponding time period can be recovered and the time period can

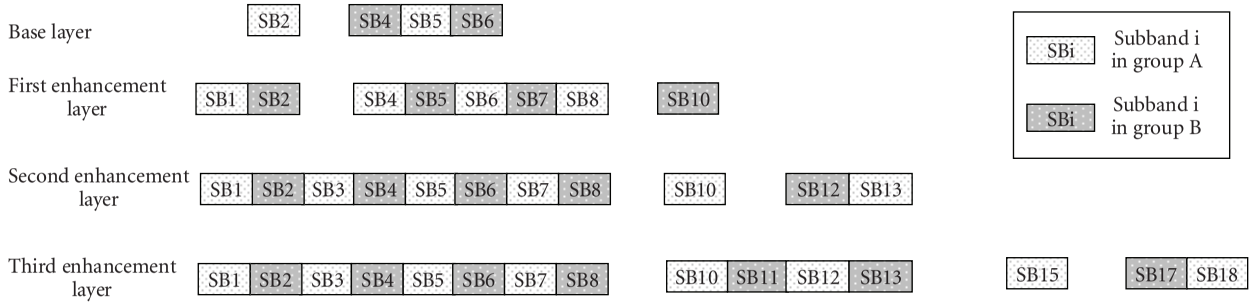


FIGURE 12.2. Example of frequency interleaving.

Table 12.2. Experimental results of the frequency interleaving method.

Affected area	Input file		
	VIOO	TRPT	GSPI
Smooth area	Noticeable only when listened to carefully	Noticeable but not annoying	Noticeable and somewhat annoying
Area with rapid melody variations	Hardly noticeable	Hardly noticeable	Noticeable but not annoying

only be played back by silence if there is no concealment technique involved. We can decisively conclude that a sound file with a sudden silence period inserted is much more annoying than the one constructed by the proposed frequency interleaving method.

12.4.4. Bitstream architecture

The bitstream architecture of the proposed algorithm is illustrated in Figure 12.3. The entire bitstream is composed of a global header and several layered information. Bits contained in lower layers represent information of perceptually more important coefficients' values. In other words, the bitstream contains all lower bit rate codes at the beginning of the bitstream so that it can provide different QoS to different end users. We look at the details of each layer. Within each layer, there are many variable-length segments, and each segment can be independently decoded. At the beginning of each segment, there is a segment header. These segment header bits are utilized to indicate the synchronization point. The data part within segments is partitioned into several packets. One packet is considered as a basic unit input into the Reed-Solomon coding block, where parity check bits are appended after data bits. At the end of each segment, there are some stuffing bits so that the whole segment is byte-aligned.

12.4.5. Error control strategy

When the end user receives a bitstream, the Reed-Solomon decoder is employed to detect and correct any possible errors that had occurred during channel transmission. Once an error that cannot be corrected is detected, its position information, such as the layer number, the segmentation number, and the packet number will be marked. If the concerned error occurs in the global header or the data part of the base layer, all bits belonging to the corresponding time position will not be reconstructed and this period of sound will be replaced with silence. Some error concealment strategy may be involved to make the final audio file sound smoother. However, if the same error occurs in the data part of any of the other enhancement layers, all bits belonging to the corresponding time position will not be used to refine the spectral data so that this error will not cause unpleasant distortions.

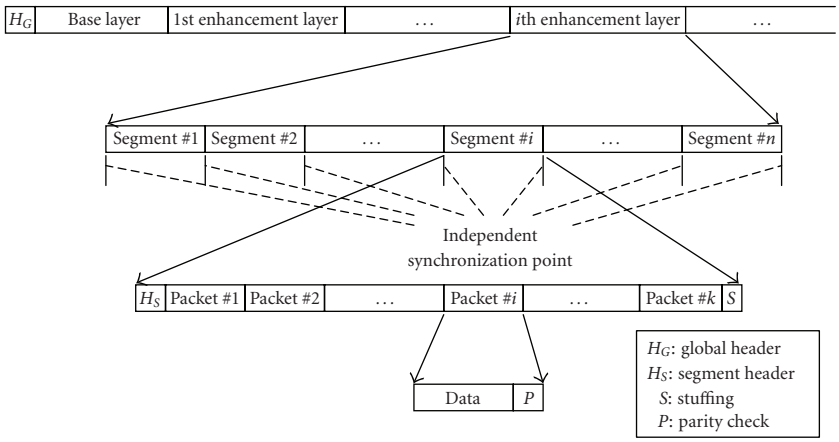


Figure 12.3. The bitstream architecture.

Experimental results show that errors that cannot be corrected in layer two or higher have little impact on the final audio file. Normal listeners can hardly perceive any impairment, if not listening carefully. If these errors are in layer one, normal listeners may perceive a little but not annoying impairment in the final audio file. There is another type of error that happens to bits belonging to the segment header. When this type of error occurs, it may cause the decoder to lose synchronization and stop decoding earlier than expected. In this scenario, the error affects more frames which can be well beyond one specific segment and the resulting audio file may correspond to a lower rate reconstructed audio file with poorer quality.

12.5. Experimental results

The proposed ERSAC system has been implemented and tested. The basic audio coding blocks [62] of the MPEG AAC main profile encoder, including the psychoacoustic model, filter bank, and temporal noise shaping are adopted to generate spectral data. An error-resilient progressive quantization block and a context-based QM coder block are added at the end to construct the error-robust scalable audio bitstream. Three single-channel sound files, that is, GSPI, TRPT, and VIOO, which are downloaded and processed from the MPEG sound quality assessment material (SQAM, http://www.tnt.uni-hannover.de/project/mpeg/audio/sqam/), are selected to test the coding performance of the proposed algorithm. The mask-to-noise ratio (MNR) values are adopted here as the objective sound quality measurement.

Figure 12.4 and Table 12.3 show the experimental results for three test materials using different WCDMA error pattern files, where the mean MNR and the average MNR values are calculated by equations (10.7) and (10.8).

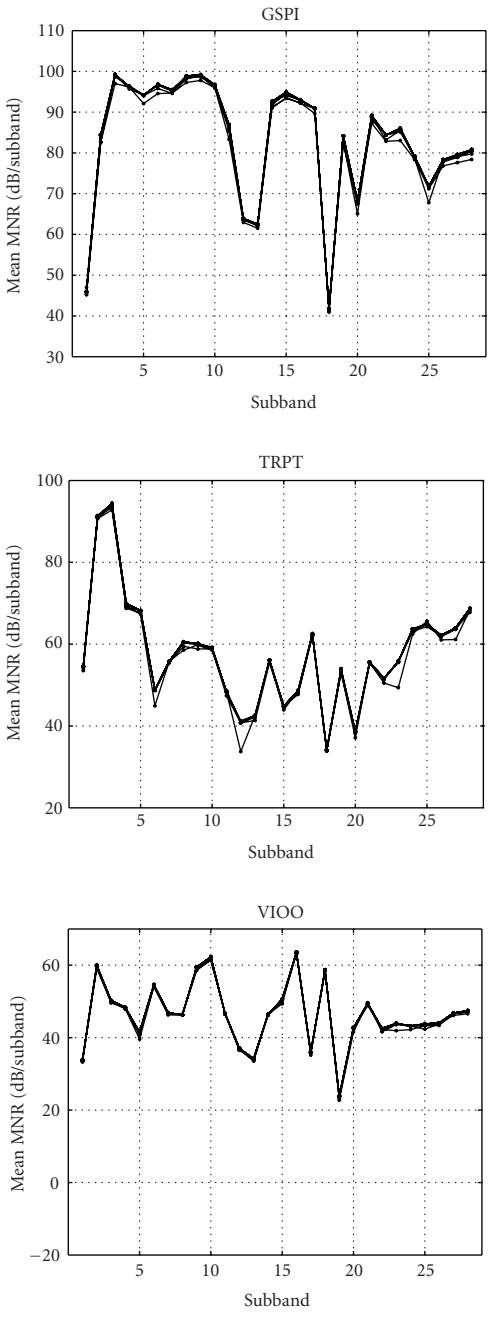


FIGURE 12.4. Mean MNR values of reconstructed audio files through different WCDMA channels.

Table 12.3. Average MNR values of reconstructed audio files through different WCDMA channels.

Error pattern file #	Error pattern file name	Ave. MNR (dB/subband)		
		GSPI	TRPT	VIOO
0	WCDMA-64 kb-005 hz-4	83.24	57.82	46.57
1	WCDMA-64 kb-070 hz-4	82.97	57.82	46.57
2	WCDMA-64 kb-211 hz-4	83.18	57.82	46.53
3	WCDMA-64 kb-005 hz-3	83.25	57.64	46.25
4	WCDMA-64 kb-070 hz-3	83.24	57.49	46.47
5	WCDMA-64 kb-211 hz-3	83.24	57.77	46.43
6	WCDMA_64 kb_50 kph_7e-04	82.72	57.24	46.38
7	WCDMA_64 kb_50 kph_2e-04	83.36	57.80	46.31
8	WCDMA_64 kb_3 kph_5e-04	81.96	56.82	46.23
9	WCDMA_64 kb_3 kph_2e-04	83.39	57.86	46.45
10	WCDMA_64 kb_3 kph_7e-05	83.39	57.86	46.61
11	WCDMA_64 kb_3 kph_3e-06	83.24	57.82	46.57
12	WCDMA_64 kb_50 kph_6e-05	83.39	57.86	46.58
13	WCDMA_64 kb_50 kph_3e-06	83.24	57.82	46.57

Based on results shown in Figure 12.4 and Table 12.3, we have the following observations.

(1) *No error*: there are no errors that cannot be corrected (in GSPI error pattern 0, 4, 5, 9, 10, 11, 12, 13; TRPT error pattern 0, 1, 2, 9, 10, 11, 12, 13; VIOO error pattern 0, 1, 10, 11, 13). This happens when either there is no error during the period when the bitstream is transmitted over the WCDMA channel or there are errors but they have been corrected by ERSAC's error detection scheme. Since more than half of the experiment cases belong to this category, it shows that the proposed ERSAC algorithm has an excellent error-resilient capability.

(2) *Error case 1*: error occurs in the global header and the data part of the base layer (none is observed in our experiment). When this happens, the decoder has no way to reconstruct the affected period of the sound file. Then, this period will be error concealed by the repetition of data in the previous period.

(3) *Error case 2*: error occurs in the segment header (none is observed in our experiment). When this happens, the decoder may lose synchronization and will not be able to continue decoding the proceeding bitstream, which means the decoder will stop refining all coefficients' values. If this happens in lower layers, for example, layer 0 or layer 1, the reconstructed audio should have poor quality and the end user may easily perceive the distortion. However, if this happens in higher layers, for example, layer 2 or higher, errors will not have big impact on the reconstructed sound file.

(4) *Error case 3*: error occurs in the data part of layer 1 or higher (observed in all remaining cases). When this happens, the decoder will stop refining coefficients in the affected period, and the reconstructed sound file has slightly degraded quality, which belongs to the perceptible distortion degree, but not annoying.

12.6. Conclusions

We presented an error-resilient scalable audio coding algorithm, which is an extension of our previous work on progressive audio compression. Compared with other existing audio codecs, this algorithm not only preserves the scalable property, but also incorporates an error-robust scheme specifically designed for WCDMA channels. Based on the characteristics of the simulated WCDMA channel, a joint source-channel coding method was developed in this work. The novelty of this technique lies in its unique unequal error protection, adaptive segmentation coding structures, and the frequency interleaving technique. Experimental results showed that the proposed ERSAC achieved a good performance using all simulated WCDMA error pattern files at a regular user bit rate of 64 kb/s.

12.7. Discussion and future work

12.7.1. Discussion

12.7.1.1. Frame interleaving

Error-resilient algorithms designed for image or video codec normally contain a block interleaving procedure, where bits belonging to adjacent areas are packed separately so that any propagated error within packets will not affect a large area. This is done because human eyes are more sensitive to low frequency components while less sensitive to high frequency components when errors are spread to a larger area. Similarly, the frame interleaving technique can also be considered for error-resilient audio codec design. However, unlike image or video, experimental results show that human ears are capable of catching spreading impairment in sound files. In fact, a longer evenly distorted period is less annoying and more tolerable than an unsmooth sound period with distortion frequently on and off. Therefore, no frame interleaving is adopted in the proposed algorithm; packets are just sent according to their original time sequence.

12.7.1.2. Error concealment

Several error concealment techniques were proposed for speech communications [40, 72, 132, 127, 107], such as SOLA, WSOLA, frame repetition, waveform substitution, and so forth. However, these methods are not suitable for high quality audio because of different applications for speech and audio. The main purpose of speech is communication while the main purpose of audio is entertainment. Thus, as long as people can understand, some noise in the background of speech is tolerable. However, this is certainly not the case for high quality audio. One common practice of existing speech error concealment methods is the addition of background noise. As a result, the error-concealed speech has good intelligibility while just having some additional noise in the background. However, adding noise in the audio file is normally not tolerated, which makes none of the available error concealment methods suitable for high quality audio.

12.7.2. Future work

In our current work, the ERSAC algorithm has only been implemented for single-channel material, and it can be extended to accommodate stereo or even multi-channel error-resilient codecs. Although only mono or stereo audio applications are needed in today's wireless communication systems, we can foresee the need of sending multichannel audio files over wired or wireless networks in the future. Thus, error-resilient multichannel audio coding is still a research topic. When input sound files with more than one channel are incorporated into the error-resilient codec, channel dependency should be taken into account, and could be utilized to develop an efficient error concealment strategy.

Bibliography

[1] Recommendation ITU-R BS. 1116-1, *Methods for the Subjective Assessment of Small Impairments in Audio Systems Including Multichannel Sound Systems*.

[2] Recommendation ITU-R BS. 1284, *Methods for the subjective assessment of sound quality—general requirements*.

[3] Recommendation ITU-R BS. 1285, *Pre-Selection Methods for the Subjective Assessment of Small Impairments in Audio Systems*.

[4] CCIR Document TG 10/2 6 (Rev. 2), *CCIR Listening Tests for the Assessment of Low Bit Rate Audio Coding Systems*, 1991.

[5] CCIR Recommendations 562-3, *Subjective Assessment of Sound Quality*, Recommendations of the CCIR **X** (1990), Part 1.

[6] IEC Publication 581, *High Fidelity Audio Equipment and Systems; Minimum Performance Requirements. Part 10: Headphones*, 1986.

[7] CCIR Recommendation 708, *Determination of Electro-Acoustical Properties of Studio Monitor Headphones*, Recommendations of the CCIR **X** (1990), Part 1.

[8] ATSC Document A/52, *Digital Audio Compression Standard (AC-3)*.

[9] E. Allamanche, R. Geiger, J. Herre, and T. Sporer, "MPEG-4 low delay audio coding based on the AAC codec," in *Proc. 106th Audio Engineering Society Convention*, Munich, Germany, May 1999, AES preprint 4929.

[10] Y. Ando, *Concert Hall Acoustics*, Springer, Berlin, Germany, 1985.

[11] D. Kirby and K. Watanabe, *Report on the formal subjective listening tests of MPEG-2 NBC multichannel audio coding*, ISO/IEC/SC29/WG11 N1419, Maceió, Reio de Janeiro, Brazil, November 1996.

[12] MPEG Audio, *MPEG-2 AAC Stereo Verification Test Results*, ISO/MPEG N2006, February 1998.

[13] J. Blauert, *Spatial Hearing*, MIT Press, Cambridge, Mass, USA, 1983.

[14] A. D. Blumlein, *Improvements in and relating to sound-transmission, sound-recording and sound-reproducing systems*, 1931, UK Patent 394325.

[15] M. Bosi, K. Brandenburg, S. Quackenbush, et al., "ISO/IEC MPEG-2 advanced audio coding," in *Proc. 101st Audio Engineering Society Convention*, Los Angeles, Calif, USA, November 1996, AES preprint 4382.

[16] M. Bosi and R. E. Goldberg, *Introduction to Digital Audio Coding and Standards*, Kluwer Academic, Norwell, Mass, USA, 2003.

[17] K. Brandenburg, "Evaluation of quality for audio encoding at low bit rates," in *Proc. 82nd Audio Engineering Society Convention*, London, UK, March 1987, AES preprint 2433.

[18] K. Brandenburg and M. Bosi, "ISO/IEC MPEG-2 advanced audio coding: overview and applications," in *Proc. 103rd Audio Engineering Society Convention*, New York, NY, USA, September 1997, AES preprint 4641.

[19] E. O. Brigham, *The Fast Fourier Transform*, Prentice-Hall, Englewood Cliffs, NJ, USA, 1973.
[20] ITU Recommendation BS.708, *Determination of the Electro-Acoustical Properties of Studio Monitor Headphones*, 1990.
[21] R. A. Campbell and A. M. Small, "Effect of practice and feedback on frequency discrimination," *Journal of the Acoustical Society of America*, vol. 35, pp. 1511–1514, 1963.
[22] X. Chen, *Advanced Mathematical Statistics*, Press of University of Science and Technology of China, Hefei, China, 1999.
[23] D. H. Cooper and T. Shiga, "Discrete-matrix multichannel stereo," *Journal of the Audio Engineering Society*, vol. 20, no. 5, pp. 346–360, 1972.
[24] M. Davis, "The AC-3 multichannel coder," in *Proc. 95th Audio Engineering Society Convention*, New York, NY, USA, October 1993, AES preprint 3774.
[25] M. Dietz, L. Liljeryd, K. Kjorling, and O. Kunz, "Spectral band replication, a novel approach in audio coding," in *Proc. 112th Audio Engineering Society Convention*, Munich, Germany, May 2002, AES preprint 5553.
[26] M. Dietz and S. Meltzer, *CT-aacPlus—a state-of-the-art audio coding scheme*, 2002, Coding Technologies, EBU Technical Review.
[27] EBU document Tech. 3276 (2nd edition), *Listening conditions for the assessment of sound programme material*, 1998.
[28] H. Fletcher, *Speech and Hearing in Communication*, Acoustical Society of America, Woodbury, NY, USA, 1995.
[29] W. H. Equitz, "A new vector quantization clustering algorithm," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 37, no. 10, pp. 1568–1575, 1989.
[30] N. Faller, "An adaptive system for data compression," in *Proc. Record of the 7th Asilomar conference on Circuits, Systems and Computers*, pp. 593–597, Pacific Grove, Calif, USA, November 1973.
[31] H. Fastl, "Temporal masking effects: I. Broad band noise masker," *Acustica*, vol. 35, no. 5, pp. 287–302, 1976.
[32] L. Fielder, M. Bosi, G. Davidson, M. Davis, C. Todd, and S. Vernon, "AC-2 and AC-3: low-complexity transform-based audio coding," in *Collected Papers on Digital Audio Bit Rate Reduction*, pp. 54–72, Audio Engineering Society, New York, NY, USA, 1995.
[33] H. Fletcher, "Auditory perspective—basic requirements," *Elect. Eng.*, vol. 53, pp. 9–11, 1934.
[34] H. Fletcher and W. A. Munson, "Loudness, its definition, measurement, and calculation," *Journal of the Acoustical Society of America*, vol. 5, no. 2, pp. 82–108, 1933.
[35] D. Frerichs, *New MPEG-4 High-efficiency AAC Audio*, Coding Technologies, White paper.
[36] H. Fuchs, "Improving joint stereo audio coding by adaptive inter-channel prediction," in *Proc. IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, pp. 39–42, New Paltz, NY, USA, October 1993.
[37] R. G. Gallager, "Variations on a theme by Huffman," *IEEE Trans. Inform. Theory*, vol. 24, no. 6, pp. 668–674, 1978.

[38] A. Gersho and R. M. Gray, *Vector Quantization and Signal Compression*, Kluwer Academic, Norwell, Mass, USA, 1991.

[39] G. H. Golub and C. F. Van Loan, *Matrix Computations*, Johns Hopkins University Press, Baltimore, Md, USA, 1983.

[40] D. J. Goodman, G. B. Lockhart, O. J. Wasem, and W.-C. Wong, "Waveform substitution techniques for recovering missing speech segments in packet voice communications," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 34, no. 6, pp. 1440–1448, 1986.

[41] B. Grill, "A bit rate scalable perceptual coder for MPEG-4 audio," in *Proc. 103rd Audio Engineering Society Convention*, New York, NY, USA, September 1997, AES preprint 4620.

[42] B. Grill and B. Teichmann, "Scalable joint stereo coding," in *Proc. 105th Audio Engineering Society Convention*, San Francisco, Calif, USA, September 1998, AES preprint 4851.

[43] T. Grusec, L. Thibault, and R. J. Beaton, "Sensitive methodologies for the subjective evaluation of high quality audio coding systems," in *Proc. Audio Engineering Society UK DSP Conference*, London, UK, September 1992.

[44] W. M. Hartmann, *Signals, Sound, and Sensation*, American Institute of Physics, Woodbury, NY, USA, 1997.

[45] J. E. Hawkins Jr. and S. S. Stevens, "The masking of pure tones and of speech by white noise," *Journal of the Acoustical Society of America*, vol. 22, no. 1, pp. 6–13, 1950.

[46] S. Haykin, *Adaptive Filter Theory*, Prentice-Hall, Englewood Cliffs, NJ, USA, 3rd edition, 1996.

[47] J. Herre, E. Allamanche, K. Brandenburg, et al., "The integrated filterbank-based scalable MPEG-4 audio coder," in *Proc. 105th Audio Engineering Society Convention*, San Francisco, Calif, USA, September 1998, AES preprint 4810.

[48] J. Herre and J. D. Johnston, "A continuously signal-adaptive filterbank for high quality perceptual audio coding," in *Proc. IEEE Workshop on Applications of Signal Processing to Audio and Acoustics*, New Paltz, NY, USA, October 1997.

[49] J. Herre and J. D. Johnston, "Exploiting both time and frequency structure in a system that uses an analysis/synthesis filterbank with high-frequency resolution," in *Proc. 103rd Audio Engineering Society Convention*, New York, NY, USA, September 1997, AES preprint 4519.

[50] J. Herre and J. D. Johnston, "Enhancing the performance of perceptual audio coders by using temporal noise shaping (TNS)," in *Proc. 101st Audio Engineering Society Convention*, Los Angeles, Calif, USA, November 1996, AES preprint 4384.

[51] J. Herre and D. Schulz, "Extending the MPEG-4 AAC codec by perceptual noise substitution," in *Proc. 104th Audio Engineering Society Convention*, Amsterdam, The Netherlands, May 1998, AES preprint 4720.

[52] H. Holma and A. Toskala, *WCDMA for UMTS, Radio Access for Third Generation Mobile Communications*, John Wiley & Sons, New York, NY, USA, revised edition, 2001.

[53] T. Holman, "Channel crossing," *Studio Sound*, pp. 40–42, February 1996.
[54] T. Holman, "The history and future of surround sound," in *Surround Professional Workshop*, Los Angeles, Calif, USA, December 2003.
[55] J.-Y. Huang and P. M. Schultheiss, "Block quantization of correlated Gaussian random variables," *IEEE Trans. Commun.*, vol. 11, no. 3, pp. 289–296, 1963.
[56] D. Huffman, "A Method for the construction of minimum redundancy codes," *Proc. IRE*, vol. 40, no. 9, pp. 1098–1101, 1952.
[57] International Standardization Organization (ISO), *Method for Calculating Loudness Level, R532*, ISO, New York, NY, USA, 1966.
[58] ISO/IEC, *Information Technology—Coding of Audio-Visual Objectis—Part 3. ISO/IEC IS 14496-3:2001*.
[59] ISO/IEC 13818-3:1994, *Generic Coding of Moving Pictures and Associated Audio Information—Part 3: Audio*, 1994.
[60] ISO/IEC JTC1/SC29/WG11 N1650, *IS 13818-7 (MPEG-2 Advanced Audio Coding, AAC)*.
[61] ISO/IEC JTC1/SC29/WG11 N2205, *Final Text of ISO/IEC FCD 14496-5 Reference Software*.
[62] ISO/IEC JTC1/SC29/WG11 N2262, *ISO/IEC TR 13818-5, Software Simulation*.
[63] ISO/IEC JTC1/SC29/WG11 N2425, *MPEG-4 Audio verification test results: Audio on Internet*.
[64] ISO/IEC JTC1/SC29/WG11 N2803, *Information Technology—Coding of Audio-Visual Objects—Part 3: Audio Amendment 1: Audio Extensions. ISO/IEC 14496-3:1999/AMD 1:2000*.
[65] ISO/IEC JTC1/SC29/WG11 N2803, *Text ISO/IEC 14496-3 Amd 1/FPDAM*.
[66] ISO/IEC JTC1/SC29/WG11 N4025, *Text of ISO/IEC 14496-5:2001*.
[67] ITU-T SG-16, *WCDMA Error Patterns at 64 kb/s*, June 1998.
[68] ITU-T SG-16, *WCDMA Error Patterns*, January 1999.
[69] N. Iwakami and T. Moriya, "Transform domain weighted interleave vector quantization (TwinVQ)," in *Proc. 101st Audio Engineering Society Convention*, Los Angeles, Calif, USA, November 1996, AES preprint 4377.
[70] N. Iwakami, T. Moriya, and S. Miki, "High-quality audio-coding at less than 64 kbit/s by using transform-domain weighted interleave vector quantization (TwinVQ)," in *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '95)*, vol. 5, pp. 3095–3098, Detroit, Mich, USA, May 1995.
[71] N. S. Jayant and P. Noll, *Digital Coding of Waveforms*, Prentice-Hall, Englewood Cliffs, NJ, USA, 1984.
[72] N. S. Jayant and S. Christensen, "Effects of packet losses in waveform coded speech and improvements due to an odd-even sample-interpolation procedure," *IEEE Trans. Commun.*, vol. 29, no. 2, pp. 101–109, 1981.

[73] J. D. Johnston and A. J. Ferreira, "Sum-difference stereo transform coding," in *Proc. IEEE International Conference on Acoustics, Speech, and Signal Processing (ICASSP '92)*, vol. 2, pp. 569–572, San Francisco, Calif, USA, March 1992.

[74] J. D. Johnston, J. Herre, M. Davis, and U. Gbur, "MPEG-2 NBC audio-stereo and multichannel coding methods," in *Proc. 101st Audio Engineering Society Convention*, Los Angeles, Calif, USA, November 1996, AES preprint 4383.

[75] S. Kim, S. Park, and Y. Kim, "Fine grain scalability in MPEG-4 audio," in *Proc. 111th Audio Engineering Society Convention*, New York, NY, USA, November–December 2001, AES preprint 5491.

[76] D. Kirby and K. Watanabe, "Formal subjective testing of the MPEG-2 NBC multichannel coding algorithm," in *Proc. 102nd Audio Engineering Society Convention*, Munich, Germany, March 1997, AES preprint 4383.

[77] D. E. Knuth, "Dynamic huffman coding," *Journal of Algorithms*, vol. 6, no. 2, pp. 163–180, 1985.

[78] H.-J. Wang and C.-C. Jay Kuo, *Multithreshold wavelet codec (MTWC)*, Doc. No. WG1N665, Sidney, Australia, November 1997.

[79] S.-S. Kuo and J. D. Johnston, "A study of why cross channel prediction is not applicable to perceptual audio coding," *IEEE Signal Processing Lett.*, vol. 8, no. 9, pp. 245–247, 2001.

[80] J. Lee, "Optimized quadtree for Karhunen-Loeve transform in multispectral image coding," *IEEE Trans. Image Processing*, vol. 8, no. 4, pp. 453–461, 1999.

[81] M. R. Leek and C. S. Watson, "Learning to detect auditory pattern components," *Journal of the Acoustical Society of America*, vol. 76, no. 4, pp. 1037–1044, 1984.

[82] S. Lin and D. J. Costello Jr., *Error Control Coding: Fundamentals and Applications*, Prentice-Hall, Englewood Cliffs, NJ, USA, 1983.

[83] C.-M. Liu, W.-C. Lee, and C.-T. Chien, "Bit allocation for advanced audio coding using bandwidth-proportional noise-shaping criterion," in *Proc. 6th International Conference on Digital Audio Effects (DAFx '03)*, London, UK, September 2003.

[84] S. P. Lloyd, "Least squares quantization in PCM," *IEEE Trans. Inform. Theory*, vol. 28, no. 2, pp. 129–137, 1982.

[85] R. A. Johnson and G. K. Bhattacharyya, *Statistics: Principles and Methods*, John Wiley & Sons, New York, NY, USA, 4th edition, 2001.

[86] J. MacQueen, "Some methods for classification and analysis of multivariate observations," in *Proc. 5th Berkeley Symposium on Mathematical Statistics and Probability*, vol. 1, pp. 281–297, Berkeley, Calif, USA, 1967.

[87] F. J. MacWilliams and N. J. A. Sloane, *The Theory of Error-Correcting Codes*, North-Holland, Amsterdam, The Netherlands, 1977.

[88] J. C. Makous and J. C. Middlebrooks, "Two-dimensional sound localization by human listeners," *The journal of the Acoustical Society of America*, vol. 87, no. 5, pp. 2188–2200, 1990.

[89] *SQAM—Sound Quality Assessment Material*, http://www.tnt.uni-hannover.de/project/mpeg/audio/sqam/.

[90] J. Max, "Quantizing for minimum distortion," *IRE Transactions on Information Theory*, vol. 6, no. 1, pp. 7–12, 1960.

[91] A. A. Mayer, "Researches in acoustics," *Philosophy Magazine*, vol. 11, pp. 500–507, 1876.

[92] A. Mouchtaris, S. S. Narayanan, and C. Kyriakakis, "Virtual microphones for multichannel audio resynthesis," *EURASIP Journal on Applied Signal Processing*, vol. 2003, no. 10, pp. 968–979, 2003, Special Issue on Digital Audio for Multimedia Communications.

[93] J. Ojanpera and M. Vaananen, "Long term predictor for transform domain perceptual audio coding," in *Proc. 107th Audio Engineering Society Convention*, New York, NY, USA, September 1999, AES preprint 5036.

[94] S. E. Olive and F. E. Toole, "The detection of reflections in typical rooms," *Journal of the Audio Engineering Society*, vol. 37, no. 7-8, pp. 539–553, 1989.

[95] A. V. Oppenheim and R. W. Schafer, *Discrete-Time Signal Processing*, Prentice-Hall, Upper Saddle River, NJ, USA, 2nd edition, 1999.

[96] T. Painter and A. Spanias, "Perceptual coding of digital audio," *Proc. IEEE*, vol. 88, no. 4, pp. 451–515, 2000.

[97] K. K. Paliwal and B. S. Atal, "Efficient vector quantization of LPC parameters at 24 bits/frame," *IEEE Trans. Speech Audio Processing*, vol. 1, no. 1, pp. 3–14, 1993.

[98] S.-H. Park, Y.-B. Kim, S.-W. Kim, and Y.-S. Seo, "Multi-layer bit-sliced bitrate scalable audio coding," in *Proc. 103rd Audio Engineering Society Convention*, New York, NY, USA, September 1997, AES preprint 4520.

[99] W. B. Pennebaker and J. L. Mitchell, *JPEG Still Image Data Compression Standard*, Van Nostrand Reinhold, New York, NY, USA, 1993.

[100] F. Pereira and T. Ebrahimi, Eds., *The MPEG-4 Book*, Prentice-Hall, Englewood Cliffs, NJ, USA, 2002.

[101] K. C. Pohlmann, *Principles of Digital Audio*, McGraw-Hill, New York, NY, USA, 4th edition, 2002.

[102] W. H. Press, B. P. Flannery, S. A. Teukolsky, and W. T. Vetterling, *Numerical Recipes in C: The Art of Scientific Computing*, Cambridge University Press, Cambridge, UK, 2nd edition, 1992.

[103] J. Princen and A. Bradley, "Analysis/Synthesis filter bank design based on time domain aliasing cancellation," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 34, no. 5, pp. 1153–1161, 1986.

[104] R. A. Damon Jr. and W. R. Harvey, *Experimental Design ANOVA, and Regression*, Harper & Row, New York, NY, USA, 1987.

[105] R. Rebscher and G. Theile, "Enlarging the listening area by increasing the number of loudspeakers," in *Proc. 88th Audio Engineering Society Convention*, Montreux, Switzerland, March 1990, AES preprint 2932.

[106] D. W. Robinson and R. S. Dadson, "A re-determination of the equal-loudness relations for pure tones," *British Journal of Applied Physics*, vol. 7, no. 5, pp. 166–181, 1956.

[107] S. Roucos and A. Wilgus, "High quality time-scale modification for speech," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '85)*, vol. 10, pp. 493–496, Tampa, Fla, USA, April 1985.

[108] J. Saghri, A. Tescher, and J. Reagan, "Practical transform coding of multispectral imagery," *IEEE Signal Processing Mag.*, vol. 12, no. 1, pp. 32–43, 1995.

[109] D. Salomon, *Data Compression: The Complete Reference*, Springer, New York, NY, USA, 2nd edition, 2000.

[110] K. Sayood, *Introduction to Data Compression*, Morgan Kaufmann, San Fransisco, Calif, USA, 2nd edition, 2000.

[111] B. Scharf, "Partial masking," *Acustica*, vol. 14, pp. 16–23, 1964.

[112] P. Scheiber, "Multidirectional sound system," U.S. Patent 3746792, 1973.

[113] P. Scheiber, "Quadrasonic sound system," U.S. Patent 3632886, 1973.

[114] G. M. Schuster and A. K. Katsaggelos, *Rate-Distortion Based Video Compression*, Kluwer Academic, Boston, Mass, USA, 1997.

[115] C. E. Shannon, "A mathematical theory of communication," *Bell System Tec. J.*, vol. 27, no. 3, pp. 379–423, 623–656, 1948.

[116] J. M. Shapiro, "Embedded image coding using zerotrees of wavelet coefficients," *IEEE Trans. Signal Processing*, vol. 41, no. 12, pp. 3445–3462, 1993.

[117] Y. Shen, H. Ai, and C.-C. J. Kuo, "A progressive algorithm for perceptual coding of digital audio signals," in *Proc. 33rd Annual Asilomar Conference on Signals, Systems, and Computers*, vol. 2, pp. 1105–1109, Pacific Grove, Calif, USA, October 1999.

[118] D. Sinha and C.-E. W. Sundberg, "Unequal error protection methods for perceptual audio coders," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '99)*, vol. 5, pp. 2423–2426, Phoenix, Ariz, USA, March 1999.

[119] W. B. Snow, "Basic principles of stereophonic sound," *SMPTE Journal*, vol. 61, pp. 567–589, 1953.

[120] R. Sperschneider, "Error resilient source coding with variable length codes and its application to MPEG advanced audio coding," in *Proc. 109th Audio Engineering Society Convention*, Los Angeles, Calif, USA, September 2000, AES preprint 5271.

[121] J. Steinberg, "Physical factors," *Bell System Technical Journal*, vol. 13, pp. 245–258, January 1934.

[122] J. W. Strutt and L. Rayleigh, "On the perception of sound direction," *Philos. Magazine*, vol. 13, pp. 214–232, 1907.

[123] Y. Takishima, M. Wada, and H. Murakami, "Reversible variable length codes," *IEEE Trans. Commun.*, vol. 43, no. 234, pp. 158–162, 1995.

[124] E. Terhardt, "Calculating virtual pitch," *Hearing Research*, vol. 1, no. 2, pp. 155–182, 1979.

[125] S. P. Thompson, "On the function of the two ears in the perception of space," *Philos. Magazine*, vol. 13, pp. 406–416, 1882.

[126] C. Todd, G. Davidson, M. Davis, L. Fielder, B. Link, and S. Vernon, "AC-3: flexible perceptual coding for audio transmission and storage," in *Proc.*

96th Audio Engineering Society Convention, Amsterdam, Holland, February–March 1994, AES preprint 3796.

[127] W. Verhelst and M. Roelands, "An overlap-add technique based on waveform similarity (WSOLA) for high quality time-scale modification of speech," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '93)*, vol. 2, pp. 554–557, Minneapolis, Minn, USA, April 1993.

[128] M. S. Vinton and E. Atlas, "A scalable and progressive audio codec," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '01)*, vol. 5, pp. 3277–3280, Salt Lake City, Utah, USA, May 2001.

[129] J. S. Vitter, "Design and analysis of dynamic Huffman codes," *Journal of the Association for Computing Machinery*, vol. 34, no. 4, pp. 825–845, 1987.

[130] R. Waal and R. Veldhuis, "Subband coding of stereophonic digital audio signals," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '91)*, vol. 5, pp. 3601–3604, Toronto, Canada, May 1991.

[131] G. Wang, Q. Zhang, W. Zhu, and J. Zhou, "Channel-adaptive error protection for scalable audio streaming over wireless Internet," in *Proc. IEEE Global Telecommunications Conference (Globecom '01)*, vol. 3, pp. 2045–2049, San Antonio, Tex, USA, November 2001.

[132] O. J. Wasem, D. J. Goodman, C. A. Dvorak, and H. G. Page, "The effects of waveform substitution on the quality of PCM packet communications," *IEEE Trans. Acoust., Speech, Signal Processing*, vol. 36, no. 3, pp. 342–348, 1988.

[133] R. L. Wegel and C. E. Lane, "The auditory masking of one pure tone by another and its probable relation to the dynamics of the inner ear," *Physical Review*, vol. 23, pp. 266–286, 1924.

[134] D. Wu, Y. T. Hou, and Y.-Q. Zhang, "Transporting real-time video over the Internet: challenges and approaches," *Proc. IEEE*, vol. 88, no. 12, pp. 1855–1877, 2000.

[135] D. Yang, H. Ai, and C.-C. J. Kuo, "Progressive multichannel audio codec (PMAC) with rich features," in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP '02)*, vol. 3, pp. 2717–2720, Orlando, Fla, USA, May 2002.

[136] D. Yang, H. Ai, C. Kyriakakis, and C.-C. J. Kuo, "Exploration of Karhunen-Loeve transform for multichannel audio coding," in *Digital Cinema and Microdisplays*, vol. 4207 of *Proceedings of SPIE*, pp. 89–100, Boston, Mass, USA, November 2000.

[137] D. Yang, H. Ai, C. Kyriakakis, and C.-C. J. Kuo, "An inter-channel redundancy removal approach for high-quality multichannel audio compression," in *Proc. 109th Audio Engineering Society Convention*, Los Angeles, Calif, USA, September 2000, AES preprint 5238.

[138] D. Yang, H. Ai, C. Kyriakakis, and C.-C. J. Kuo, "Adaptive Karhunen-Loeve transform for enhanced multichannel audio coding," in *Mathematics of Data/Image Coding, Compression, and Encryption IV, with Applications*, vol. 4475 of *Proceedings of SPIE*, pp. 43–54, San Diego, Calif, USA, July 2001.

[139] D. Yang, H. Ai, C. Kyriakakis, and C.-C. J. Kuo, "Embedded high-quality multichannel audio coding," in *Media Processors 2001*, vol. 4313 of *Proceedings of SPIE*, pp. 74–85, San Jose, Calif, USA, January 2001.

[140] D. Yang, H. Ai, C. Kyriakakis, and C.-C. J. Kuo, "Design of progressive syntax-rich multichannel audio codec," in *Media Processors 2002*, vol. 4674 of *Proceedings of SPIE*, pp. 121–132, San Jose, Calif, USA, January 2002.

[141] D. Yang, H. Ai, C. Kyriakakis, and C.-C. J. Kuo, "Error-resilient design of high-fidelity scalable audio coding," in *Digital Wireless Communications IV*, vol. 4740 of *Proceedings of SPIE*, pp. 53–63, Orlando, Fla, USA, April 2002.

[142] D. Yang, H. Ai, C. Kyriakakis, and C.-C. J. Kuo, "High-fidelity multichannel audio coding with Karhunen-Loeve transform," *IEEE Trans. Speech Audio Processing*, vol. 11, no. 4, pp. 365–380, 2003.

[143] D. Yang, H. Ai, C. Kyriakakis, and C.-C. J. Kuo, "Progressive syntax-rich coding of multichannel audio sources," *EURASIP Journal on Applied Signal Processing*, vol. 2003, no. 10, pp. 980–992, 2003, Special Issus on Digital Audio for Multimedia Communications.

[144] C. W. Yung, H. F. Fu, C. Y. Tsui, R. S. Cheng, and D. George, "Unequal error protection for wireless transmission of MPEG audio," in *Proc. IEEE International Symposium on Circuits and Systems (ISCAS '99)*, vol. 6, pp. 342–345, Orlando, Fla, USA, May–June 1999.

[145] J. Zhou and J. Li, "Scalable audio streaming over the internet with network-aware rate-distortion optimization," in *Proc. IEEE International Conference on Multimedia and Expo (ICME '01)*, pp. 567–570, Tokyo, Japan, August 2001.

[146] J. Zhou, Q. Zhang, Z. Xiong, and W. Zhu, "Error resilient scalable audio coding (ERSAC) for mobile applications," in *Proc. IEEE 4th Workshop on Multimedia Signal Processing*, pp. 307–312, Cannes, France, October 2001.

[147] T. Ziegler, A. Ehret, P. Ekstrand, and M. Lutzky, "Enhancing MP3 with SBR: features and capabilities of the new MP3PRO algorithm," in *Proc. 112th Audio Engineering Society Convention*, Munich, Germany, May 2002, AES preprint 5560.

[148] E. Zwicker and H. Fastl, *Psychoacoustics, Facts, and Models*, Springer, Berlin, Germany, 1990.

[149] J. Zwislocki, "A theory of central auditory masking and its partial validation," *Journal of the Acoustical Society of America*, vol. 52, pp. 644–659, 1972.

[150] J. Zwislocki, F. Maire, A. S. Feldman, and H. Rubin, "On the effects of practice and motivation on the threshold of audibility," *Journal of the Acoustical Society of America*, vol. 30, pp. 254–262, 1958.

Index