Chapter 9

Standards for audio and music representation

Giovanni De Poli

Version October 27, 2007

9.1 Digital audio compression

adapted from Noll (2000)

Audio compression or audio coding algorithms are used to obtain compact digital representations of high-fidelity (wideband) audio signals for the purpose of efficient transmission or storage. The central objective in audio coding is to represent the signal with a minimum number of bits while achieving transparent signal reproduction, i.e., generating output audio that cannot be distinguished from the original input, even by a sensitive listener (”golden ears”).

9.1.1 Bit Rate Reduction

Typical audio signal classes are telephone speech, wideband speech, and wideband audio, all of which differ in bandwidth, dynamic range, and in listener expectation of offered quality. The quality of telephone-bandwidth speech is acceptable for telephony and for some video telephony and video-conferencing services. Higher bandwidths (7 kHz for wideband speech) may be necessary to improve the intelligibility and naturalness of speech. Wideband (high fidelity) audio representation including multi-channel audio needs bandwidths of at least 20 kHz.

The conventional digital format for these signals is PCM, with sampling rates and amplitude resolutions (PCM bits per sample) as given in Table 9.1.1. The compact disc (CD) is todays de facto standard of digital audio representation. On a CD with its 44.1kHz sampling rate the resulting stereo net bit rate is $2 \times 44.1 \times 16 \times 1000 = 1.41$ Mb/s. However, the CD needs a significant overhead for a run length-limited line code, which maps 8 information bits into 14 bits, for synchronization and for error correction, resulting in a 49-bit representation of each 16-bit audio sample. Hence, the total stereo bit rate is $1.41 \times 49/16 = 4.32$ Mb/s. For archiving and processing of audio signals,

sampling rates of at least 96 kHz and amplitude resolutions of up to 24 bit per sample are used. The Digital Versatile Disk (DVD) with its capacity of 4.7 GB (single layer) or 8.5 GB (double layer) is the appropriate storage medium for such applications.

Audio signal	Frequency range in Hz	Sampling rate in kHz	PCM (bit/sample)	PCM bit rate (kbit/sec)
Telephone speech	300 -3,400	8	8	64
Wideband speech	50-7,000	16	8	128
Wideband audio (stereo)	10-20,000	48	2×16	2×768
CD	10-20,000	44.1	2×16	2×705.6

Table 9.1: Basic parameters for three classes of acoustic signals.

Although high bit rate channels and networks become more easily accessible, low bit rate coding of audio signals has retained its importance. The main motivations for low bit rate coding are the need to minimize transmission costs or to provide cost-efficient storage, the demand to transmit over channels of limited capacity such as mobile radio channels, and to support variable-rate coding in packet-oriented networks. For these reasons many researcher worked toward formulation of compression schemes that can satisfy simultaneously the conflicting demands of high compression ratios and transparent reproduction quality for high-fidelity audio signals, A *lossless* or noiseless coding system is able to reconstruct perfectly the samples of the original signal from the coded (compressed) representation. In contrast, a coding scheme incapable of perfect reconstruction from the coded representation is denoted *lossy*.

Basic requirements of low bit rate audio coders are first, to retain a high quality of the reconstructed signal with robustness to variations in spectra and levels. In the case of stereophonic and multi-channel signals spatial integrity is an additional dimension of quality. Second, robustness against random and bursty channel bit errors and packet losses is required. Third, low complexity and power consumption of the codecs are of high relevance. For example, in broadcast and playback applications, the complexity and power consumption of audio decoders used must be low, whereas constraints on encoder complexity are more relaxed. Additional network-related requirements are low encoder/decoder delays, robustness against errors introduced by cascading codecs, and a graceful degradation of quality with increasing bit error rates in mobile radio and broadcast applications. Finally, in professional applications, the coded bit streams must allow editing, fading, mixing, and dynamic range compression.

First proposals to reduce wideband audio coding rates have followed those for speech coding. Differences between audio and speech signals are manifold; however, audio coding implies higher sampling rates, better amplitude resolution, higher dynamic range, larger variations in power density spectra, stereophonic and multi-channel audio signal presentations, and, finally, higher listener expectation of quality. Indeed, the high quality of the CD with its 16-bit per sample PCM format has made digital audio popular. Speech and audio coding are similar in that in both cases quality is based on the properties of human auditory perception. On the other hand, speech can be coded very efficiently because a speech production model is available, whereas nothing similar exists for audio signals.

9.1.2 Auditory Masking and Perceptual Coding

9.1.2.1 Auditory Masking

The inner ear performs short-term critical band analyses where frequency-to-place transformations occur along the basilar membrane. The power spectra are not represented on a linear frequency scale but on limited frequency bands called critical bands. The auditory system can roughly be described as a band-pass filter-bank, consisting of strongly overlapping bandpass filters with bandwidths in the order of 50 to 100 Hz for signals below 500 Hz and up to 5000 Hz for signals at high frequencies. Twenty-five critical bands covering frequencies of up to 20 kHz have to be taken into account.

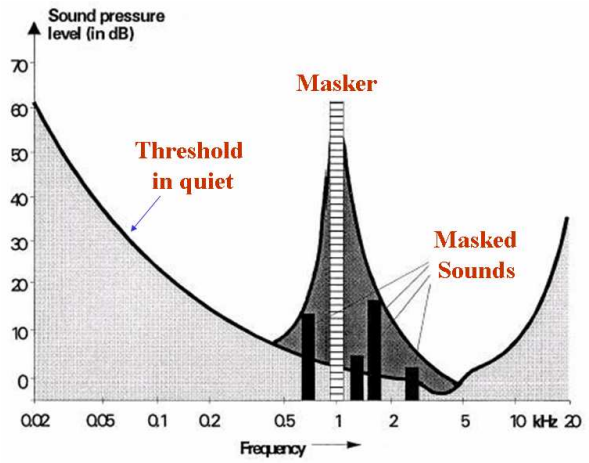

Figure 9.1: Threshold in quiet and masking threshold. Acoustical events in the shaded areas will not be audible.

Simultaneous masking is a frequency domain phenomenon where a low-level signal (the maskee) can be made inaudible (masked) by a simultaneously occurring stronger signal (the masker), if masker and maskee are close enough to each other in frequency, Such masking is greatest in the critical band in which the masker is located, and it is effective to a lesser degree in neighboring bands. A *masking threshold* can be measured below which the low-level signal will not be audible. This masked signal can consist of low-level signal contributions, quantization noise, aliasing distortion, or transmission errors. The masking threshold, in the context of source coding also known as threshold of just noticeable distortion (JND), varies with time. It depends on the sound pressure level (SPL), the frequency of the masker, and on characteristics of masker and maskee. Take the example of the masking threshold for the SPL = 60 dB narrowband masker in Fig. 9.1: around 1 kHz the four maskees will be masked as long as their individual sound pressure levels are below the masking threshold. The slope of the masking threshold is steeper towards lower frequencies, i.e., higher frequencies are more easily masked. It should be noted that the distance between masker and masking threshold is smaller in noise-masking-tone experiments than in tone-masking-noise experiments, i.e., noise is a better masker than a tone. In MPEG coders both thresholds play a role in computing the masking threshold.

Without a masker, a signal is inaudible if its sound pressure level is below the *threshold in quiet* which depends on frequency and covers a dynamic range of more than 60 dB as shown in the lower curve of Figure 9.1. The quiet (absolute) threshold is well approximated by the nonlinear function (see Fig. 9.3)

$$T_q(f) = 3.64(f/1000)^{-0.8} - 6.5^{-0.6(f/1000-3.3)^2} + 10^{-3}(f/1000)^4 \text{ dB SPL}$$

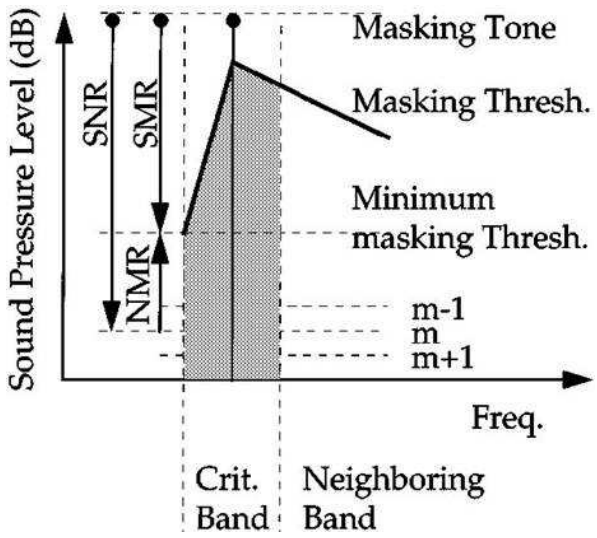

Figure 9.2: Masking threshold and signal-to-mask ratio (SMR). Acoustical events in the shaded areas will not be audible.

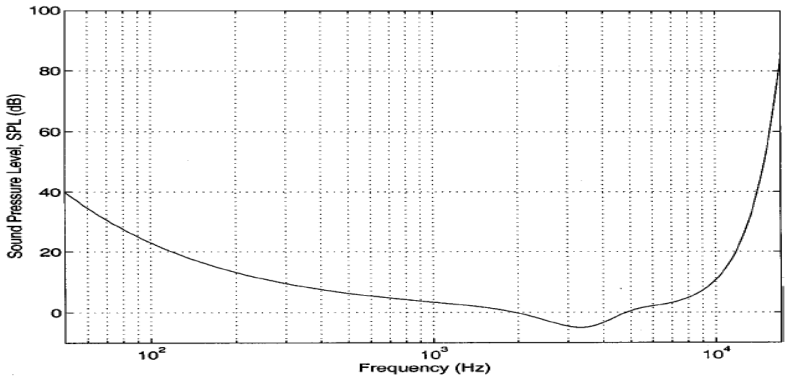

Figure 9.3: The absolute threshold of hearing in quiet. Across the audio spectrum, it quantifies the SPL required at each frequency such that an average listener will detect a pure tone stimulus in a noiseless environment. From (Painter-Spanias 2000).

The qualitative sketch of Fig. 9.2 gives a few more details about the masking threshold: a critical band, tones below this threshold (darker area) are masked. The distance between the level of the masker and the masking threshold is called Signal-to-Mask Ratio (SMR). Its maximum value is at the left border of the critical band (point A in Fig. 9.2), its minimum value occurs in the frequency range of the masker and is around 6dB in noise-masks-tone experiments. Assume a m-bit quantization of an audio signal. Within a critical band the quantization noise will not be audible as long as its signal to-noise ratio SNR is higher than its SMR. Noise and signal contributions outside the particular critical band will also be masked, although to a lesser degree, if their SPL is below the masking threshold.

Defining $SNR(m)$ as the signal-to-noise ratio resulting from an m-bit quantization, the perceivable distortion in a given subband is measured by the *Noise-to-Mask Ratio*:

$$NMR(m) = SMR - SNR(m) \quad \text{(in dB)}.$$

The noise-to-mask ratio $NMR(m)$ describes the difference in dB between the signal-to-mask ratio

and the signal-to-noise ratio to be expected from an m-bit quantization. The NMR value is also the difference (in dB) between the level of quantization noise and the level where a distortion may just become audible in a given subband. Within a critical band, coding noise will not be audible as long as $NMR(m)$ is negative.

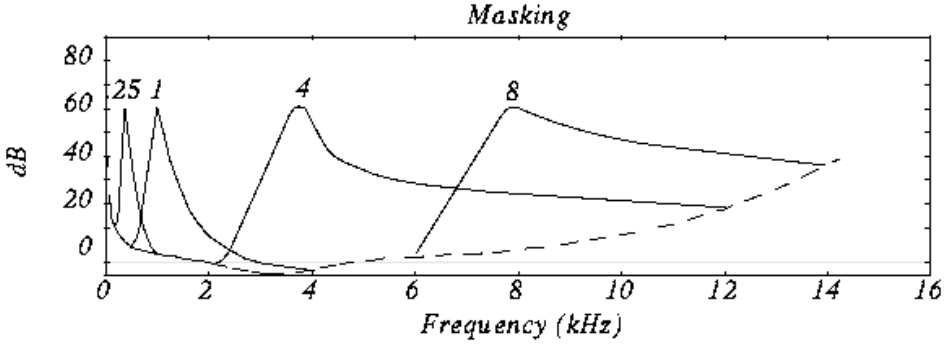

Figure 9.4: When many simultaneous maskers, each has its own masking threshold, and a global masking threshold can be computed.

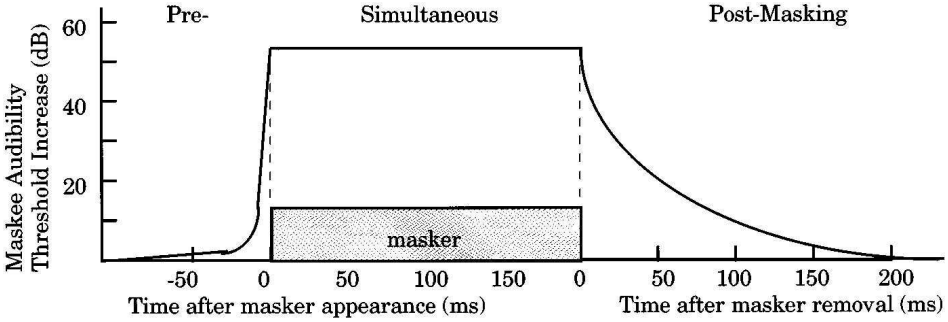

Figure 9.5: Temporal masking properties of the human ear. Backward (pre) masking occurs prior to masker onset and lasts only a few milliseconds; forward (post) masking may persist for more than 100 ms after masker removal.

We have just described masking by only one masker. If the source signal consists of many simultaneous maskers, each has its own masking threshold, and a *global masking threshold* can be computed that describes the threshold of just noticeable distortions as a function of frequency (see for example Fig. 9.4).

In addition to simultaneous masking, the time domain phenomenon of *temporal masking* plays an important role in human auditory perception. As shown in Fig. 9.5, masking phenomena extend in time beyond the window of simultaneous stimulus presentation. In other words, for a masker of finite duration, temporal masking occurs both prior to masker onset as well as after masker removal. The skirts on both regions are schematically represented in Fig. 9.5. Essentially, absolute audibility thresholds for masked sounds are artificially increased prior to, during, and following the occurrence of a masking signal. It may occur when two sounds appear within a small interval of time. Depending on the individual sound pressure levels, the stronger sound may mask the weaker one, even if the maskee precedes the masker (Fig. 9.5)! Temporal masking can help to mask pre-echoes caused by the spreading of a sudden large quantization error over the actual coding block. The duration within which pre-masking applies is signficantly less than one tenth of that of the post-masking which is in

the order of 50 to 200 ms. Both pre-and post masking are being exploited in MPEG/Audio coding algorithms. In Fig. 9.6 the net effect of simultaneous and temporal masking is shown.

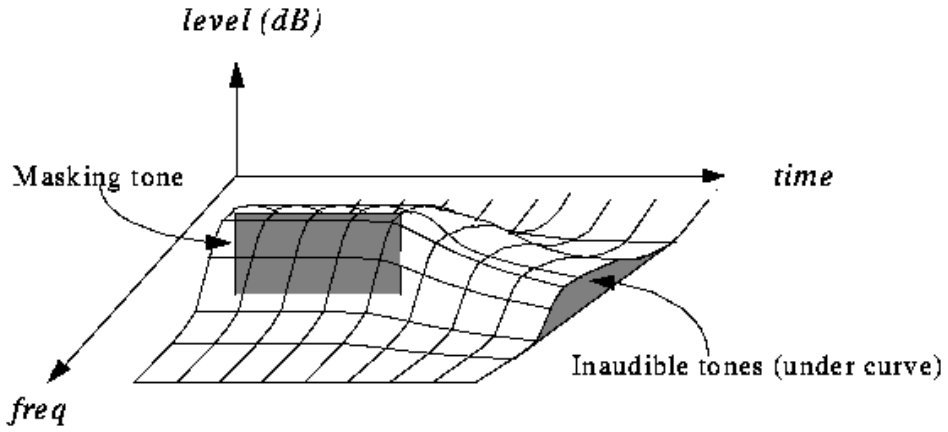

Figure 9.6: Net effect of simultaneous and temporal masking. Acoustical events under the surface will not be audible.

9.1.2.2 Perceptual Coding

Digital coding at high bit rates is dominantly waveform-preserving, i.e., the amplitude vs. time waveform of the decoded signal approximates that of the input signal. The difference signal between input and output waveform is then the basic error criterion of coder design. At lower bit rates, facts about the production and perception of audio signals have to be included in coder design, and the error criterion has to be in favour of an output signal that is useful to the human receiver rather than favouring an output signal that follows and preserves the input waveform. Basically, an efficient source coding algorithm will (1) remove *redundant* components of the source signal by exploiting correlations between its samples and (2) remove components that are *irrelevant* to the ear. Irrelevancy manifests itself as unnecessary amplitude or frequency resolution; portions of the source signal that are masked do not need to be transmitted.

The dependence of human auditory perception on frequency and the accompanying perceptual tolerance of errors can (and should) directly influence encoder designs; noise-shaping techniques can emphasize coding noise in frequency bands where that noise perceptually is not important. To this end, the noise shifting must be dynamically adapted to the actual short-term input spectrum in accordance with the signal-to-mask ratio which can be done in different ways. However, frequency weightings based on linear filtering, as typical in speech coding, cannot make full use of results from psychoacoustics. Therefore, in wideband audio coding, noise-shaping parameters are dynamically controlled in a more efficient way to exploit simultaneous masking and temporal masking.

Figure 9.7 depicts the structure of a perception-based coder that exploits auditory masking. The encoding process is controlled by the *signal to mask ratio* (SMR), the ratio of short term signal power within each frequency band and the masking threshold. From the SMR the needed amplitude resolution (and hence the bit allocation and rate) in each frequency band is derived. Moreover the number of bits assigned to each band is adapted to short-term spectrum of the audio coding block on a block-by-block basis. Such a *dynamic bit allocation* is used in all recent audio coding algorithms, which assign bits in order to maximize the overall perceptual quality. Algoritm 9.1 presents the pseudocecode for

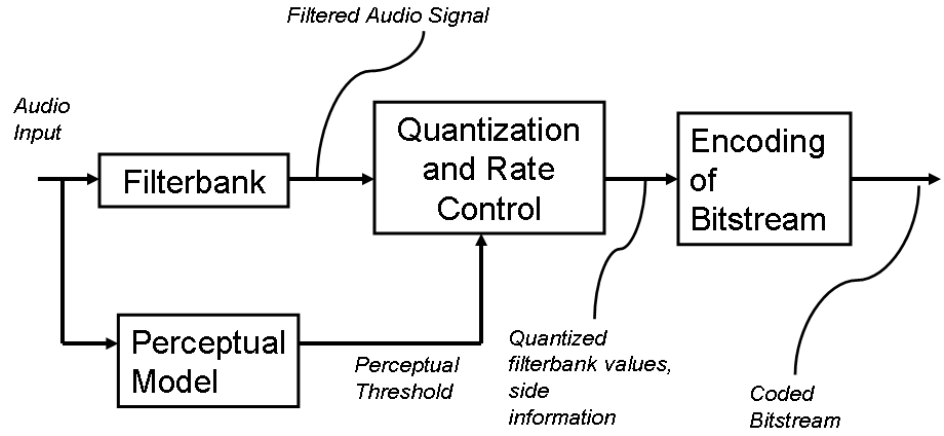

Figure 9.7: Block diagram of a generic perceptual coder.

Algorithm 9.1: PerceptualCoding

Data:
Result:
1 Use filters to divide the audio signal into frequency subbands;
2 Determine amount of masking for each band caused by nearby band using the psycho-acoustic model;
3 **if** *the power in a band is below masking threshold* **then**
4 don't encode it
5 **else**
6 determine no. of bits needed to represent the coefficient such that noise introduced by quantization is below the masking effect (one fewer bit of quantization introduces about 6 dB of noise)
7 Format bitstream;

the `PerceptualCoding` algorithm . Fig. 9.7 show the basic block diagram of a perceptual coding system. It consists of the following blocks:

- **Filter bank.** A filter bank is used to decompose the input signal into subsampled spectral components in time frequency domain. Together with the corresponding filter in the decoder, it forms as analysis/synthesis system.
- **Perceptual model** Using the time domain input signal and/or the output of the analysis filter bank, an estimate of the actual (time frequency dependent) masking threshold is computed using a perceptual model which implements known rules from psychoacoustics.
- **Quantization and coding.** The spectral components are quantized and coded with the aim of keeping the noise, which is introduced by quantization, below the masking threshold.
- **Encoding of beat stream.** A beat stream formatter is used to assemble the bitstream, which typically consists of the quantized and coded spectral components and some side information, e.g. bit allocation information.

The decoder (Fig. 9.8) simply reverses the formatting, then reconstructs the quantized subband values, and finally transforms the set of subband values into a time-domain audio signal.

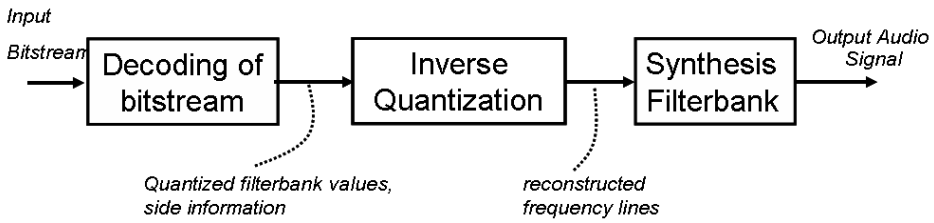

Figure 9.8: Block diagram of a generic perceptual decoder.

9.1.3 MPEG 1 Layer-3 coding

MPEG, a working group formally named as ISO/IEC JTC1/SC29/ WG11, but mostly known by its nickname, Moving Pictures Experts Group (MPEG), was set up by the ISO/IEC standardization body in 1988 to develop generic (i.e. useful for different applications) standards for the coded representation of moving pictures, associated audio and their combination. Since then, MPEG has undertaken the standardization of compression techniques for video and audio.

MPEG-1 Audio is an International Organization for Standardization (ISO) standard for high-fidelity audio compression. It is one part of a three-part compression standard. With the other two parts, video and systems, the composite standard addresses the compression of synchronized video and audio at a total bit rate of roughly 1.5 megabits per second. MPEG/audio compression is lossy; however, the MPEG algorithm can achieve transparent, perceptually lossless compression.

MPEG-1 Audio consists of three operating modes called *Layers*, with increasing complexity and performance, named Layer-1, Layer-2 and Layer-3. Layer-3, with the highest complexity, was designed to provide the highest sound quality at low bit-rates (around 128 kbit/s for a typical stereo signal). In general MP3 is appropriate for applications involving storage or transmission of mono or stereo music or other audio signals. Since it is implemented on virtually all digital audio devices playback is always ensured. Layer-3, or as it is mostly called nowadays `mp3`, is the most pervasive audio coding format for storage of music on PC platforms, and transmission of music over the Internet. Mp3 has created a new class of consumer electronics devices named after it, the mp3 player. It is found on almost all CD and DVD players and in an increasing number of car stereo systems and home stereo devices like networked home music servers. Additionally, Layer-3 finds wide application in satellite digital audio broadcast and on cellular phones.

9.1.3.1 The Psychoacoustic Model

The psychoacoustic model is the key component of the MPEG encoder that enables its high performance. The job of the psychoacoustic model is to analyze the input audio signal and determine where in the spectrum quantization noise will be masked and to what extent. The encoder uses this information to decide how best to represent the input audio signal with its limited number of code bits. The MPEG/audio standard provides two example implementations of the psychoacoustic model. Algorithm 9.2 (`Determine_Masking_Threshold`) outlines of the basic steps involved in the

psychoacoustic calculations for either model.

Algorithm 9.2: Determine_Masking_Threshold

Data: a frame (1152 samples) of the input signal
Result: Compute signal-to-mask ratio (SMR) for each subband
1 Time align audio data.
2 Convert audio to spectral domain
3 Partition spectral values into critical bands
4 Separate into tonal and non-tonal components
5 Apply spreading function
6 Find the minimum masking threshold for each subband
7 Calculate signal-to-mask ratio

1. *Time align audio data* The psychoacoustic model must account for both the delay of the audio data through the filter bank and a data offset so that the relevant data is centered within its analysis window. For example, when using psychoacoustic model two for Layer-1, the delay through the filter bank is 256 samples, and the offset required to center the 384 samples of a Layer-1 frame in the 512-point psychoacoustic analysis window is (512 384)/2 = 64 points. The net offset is 320 points to time align the psychoacoustic model data with the filter bank outputs.
2. *Convert audio to spectral domain* The psychoacoustic model uses a time-to-frequency mapping such as a 512-or 1,024-point Fourier trans form. A standard Hann weighting, applied to audio data before Fourier transformation, conditions the data to reduce the edge effects of the transform window. The model uses this separate and independent mapping instead of the filter bank outputs because it needs finer frequency resolution to calculate the masking thresholds.
3. *Partition spectral values into critical bands* To simplify the psychoacoustic calculations, the model groups the frequency values into perceptual quanta (Fig 9.9).
4. *Incorporate threshold in quiet* The model includes an empirically determined absolute masking threshold. This threshold is the lower bound for noise masking and is determined in the absence of masking signals.
5. *Separate into tonal and non-tonal components* The model must identify and separate the tonal and noise-like components of the audio signal because the noise-masking characteristics of the two types of signal are different.
6. *Apply spreading function* The model determines the noise-masking thresholds by applying an empirically determined masking or spreading function to the signal components.
7. *Find the minimum masking threshold for each subband* The psychoacoustic model calculates the masking thresholds with a higher-frequency resolution than provided by the filter banks. Where the filter band is wide relative to the critical band (at the lower end of the spectrum), the model selects the minimum of the masking thresholds covered by the filter band. Where the filter band is narrow relative to the critical band, the model uses the average of the masking thresholds covered by the filter band.
8. *Calculate signal-to-mask ratio* The psychoacoustic model takes the minimum masking threshold and computes the signal-to-mask ratio; it then passes this value to the bit (or noise) allocation section of the encoder.

A block scheme of the MPEG 1 Layer-3 encoder is shown in Fig. 9.10. Notice that a hybrid filter bank is introduced to increase frequency resolution and thereby better approximate critical band

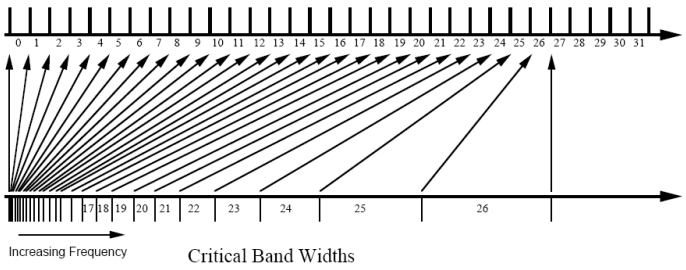

Figure 9.9: MPEG/Audio filter bandwidths versus critical bandwidths.

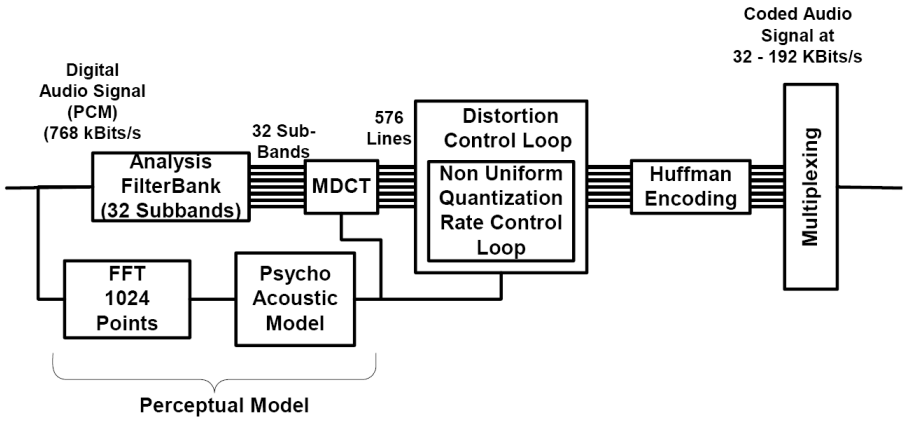

Figure 9.10: Block diagram of the MPEG 1 Layer-3 encoder.

behavior. The hybrid filter bank is constructed by following each subband filter with an adaptive Modified Digital Cosine Transform (MDCT). This practice allows for higher frequency resolution and pre-echo control. Use of an 18-point MDCT, for example, improves frequency resolution to 41.67 Hz per spectral line. Bit allocation and quantization of the spectral lines are realized in a nested loop procedure that uses both nonuniform quantization and Huffman coding. The inner loop adjusts the nonuniform quantizer step sizes for each block until the number of bits required to encode the transform components falls within the bit budget. The outer loop evaluates the quality of the coded signal (analysis-by-synthesis) in terms of quantization noise relative to the JND thresholds.

9.1.3.2 The Problem of Stereo Coding

There are several new issues introduced when the issue of stereophonic reproduction is introduced:

- *The problem of Binarual Masking Level Depression (BLMD).* At lower frequencies, $f < 3000$ Hz, the Human Auditory System is able to take the phase of interaural signals into account. This can lead to the case where, for instance, a noise image and a tone image can be in different places. This can reduce the masking threshold by up to 20dB in extreme cases. BMLD can create a situation whereby a signal that was the same as the original in a monophonic setting sounds substantially distorted in a stereophonic setting. Two good, efficient monophonic coders do NOT make one good efficient stereo coder.

- *The problem of image distortion or elimination.* In addition to BLMD issues, a signal with a

distorted high-frequency envelope may sound transparent in the monophonic case, but will not in general provide the same imaging effects in the stereophonic case.

Both the low-frequency BLMD and the high-frequency envelope effects behave quite similarly in terms of stereo image impairment or noise unmasking, when we consider signal envelope at high frequencies or waveforms themselves at low frequencies. The effect is not as strong between 500Hz and 2 kHz.

In order to control the imaging problems in stereo signals, several methods must be used. The MPEG/audio compression algorithm supports two types of stereo redundancy coding: intensity stereo coding and middle/side (M/S) stereo coding. Both forms of redundancy coding exploit the above seen perceptual weakness of the ear. Psychoacoustic results show that, within the critical bands covering frequencies above approximately 2 kHz, the ear bases its perception of stereo imaging more on the temporal envelope of the audio signal than its temporal fine structure. All layers support intensity stereo coding. Layer-3 also supports M/S stereo coding.

In *intensity stereo mode* or MPEG-1 Layer-1,2 *joint stereo mode*, the encoder codes some upper-frequency filter bank outputs with a single summed signal rather than send independent codes for left (L) and right (R) channels for each of the 32 filter bank outputs; i.e. the relative intensities of the L and R channels are used to provide high-frequency imaging information. Usually, one signal ($L + R$, typically) is sent, with two gains, one for L and one for R. The intensity stereo decoder reconstructs the left and right channels based only on independent left-and right-channel scale factors. With intensity stereo coding, the spectral shape of the left and right channels is the same within each intensity-coded filter bank signal, but the magnitude is different. Intensity stereo methods do not guarantee the preservation of the envelope of the signal for high frequencies. For lower quality coding, intensity stereo is a useful alternative to M/S stereo coding, We may think of intensity stereo as the coder equivalent of a pan-pot.

A psychoacoustic model that takes account of BMLD and envelope issues must be included. BMLD is best calculated and handled in the M/S paradigm. The *M/S stereo mode* is mid/side, or mono/stereo coding. It encodes the signals for left (L) and right (R) channels in certain frequency ranges as middle (M) and side (S) channels. where M and S are defined as:

$$\begin{aligned} M &= L + R \\ S &= L - R \end{aligned}$$

The normalization of $1/2$ is usually done on the encoding side. In this mode, the encoder uses specially tuned techniques to further compress side-channel signal. A good stereo coder uses both M/S and L/R coding methods, as appropriate. M/S, while very good for some signals, creates either a false noise image or a substantial overcoding requirement for other signals. An M/S coder provides a great deal of redundancy elimination when signals with strong central images are present, or when signals with a strong surround component are present. Finally, an M/S coder provides better signal recovery for signals that have matrixed information present, by preserving the M and S channels preferentially to the L and R channels when one of M or S has the predominant energy.

9.1.3.3 Discussion on MPEG Layer-3

MPEG Layer-3 emerged as the main tool for Internet audio delivery. Considering the reasons, the following factors were definitely helpful.

- *Open standard* MPEG is defined as an open standard. The specification is available (for a fee) to everybody. While there are a number of patents covering MPEG Audio encoding and decoding,

all patent-holders have declared that they will license the patents on fair and reasonable terms to everybody. Public example source code is available to help implementers to understand the text of the specification. As the format is well defined, no problems with interoperability of equipment or software from different vendors have been reported except from some rare incomplete implementations.

- *Availability of encoders and decoders* DSP-based hardware and software encoders and decoders have been available for a number of years driven at first by the demand for professional use in broadcasting.

- *Supporting technologies* While audio compression is viewed as a main enabling technology, other evolving technologies contributed to the MP3 boom, such as: the widespread use of computer sound cards; computers becoming powerful enough to run software audio decoders and even encoders in real-time; fast Internet access for universities and businesses; the availability of CD-ROM and CD-Audio writers.

- *Normative versus Informative* A very important property of the MPEG standards is the principle of minimizing the amount of normative elements in the standard. In the case of MPEG Audio, this led to the fact that only the data representation, i.e. the format of the compressed audio, and the decoder are normative. Even the decoder is not specified in a bit-exact fashion. Instead, formulae are given for most parts of the algorithm, and compliance is defined by a maximum deviation of the decoded signal from a reference decoder, implementing the formulae with double-precision arithmetic accuracy. This enables decoders running both on floating-point and fixed-point architectures. Depending on the skills of the implementers, fully-compliant high-accuracy Layer-3 decoders can be constructed with down to 20-bit arithmetic wordlength, without using double-precision calculations.

In short, MPEG Layer-3 had the luck to be the right technology available at the right time. In the meantime, research on perceptual audio coding progressed, and codecs with better compression efficiency became available. Of these, MPEG-2 Advanced Audio Coding (AAC) was developed as the successor of MPEG-1 Audio. Other proprietary audio coding schemes were also introduced, claiming a higher performance than MP3.

Quality Considerations: However, the pure compliance of an encoder with an MPEG audio standard does not guarantee any quality of the compressed music. Audio quality differs between different items, depending on basic parameters including, of course, the bit-rate of the compressed audio and the sophistication of different encoders, even if they work with the same set of basic parameters. To gain more insight into the level of quality possible with MP3, let us first have a look at typical artefacts associated with perceptual audio coders.

- *Common types of artefacts* Unlike analogue hi-fi equipment or FM broadcasting, perceptual encoders exhibit sound deficiencies when run at too low bit-rates or with the wrong parameters. These so-called artefacts are in most cases different from usual noise or distortion signals. Perceptual audio coding schemes such as MPEG Layer-3 introduce an error signal that can be described as a time-varying error at certain frequencies, which is not constrained to the harmonics of the music signal. The resulting music signal may sound: distorted, but not like harmonic distortions; noisy, but with the noise introduced only in a certain frequency range; rough, with the roughness often being very objectionable as the error may change its characteristics about every 24 ms.

- *Loss of bandwidth* If an encoder does not find a way to encode a block of music data with the required fidelity within the limits of the available bit-rate, it runs out of bits. This may lead to the deletion of some frequency lines, typically affecting the high-frequency content. Compared to a constant bandwidth reduction, such an effect becomes more objectionable if the effective bandwidth changes frame-by-frame (e.g. every 24 ms).

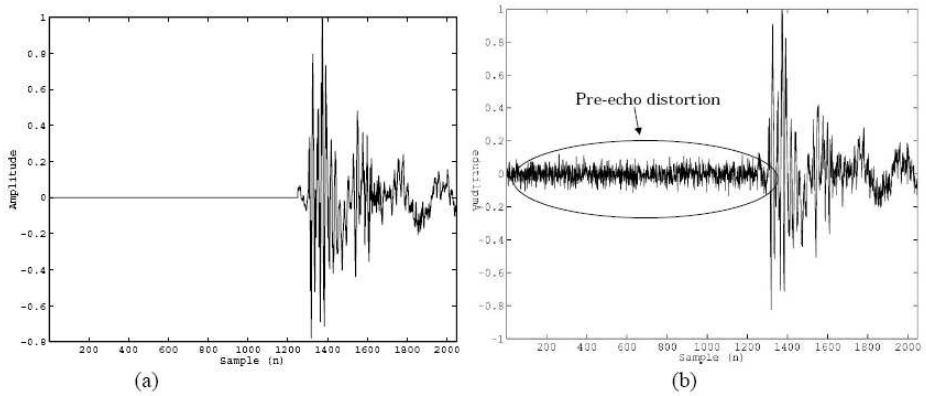

Figure 9.11: Pre-Echo Example: (a) Uncoded Castanets. (b) Transform Coded Castanets, 2048-Point Block Size.

- *Pre-echoes* Pre-echoes are very common artefacts, in the case of perceptual audio coding schemes using high-frequency resolution (see Fig. 9.11). The name pre-echo, although somewhat misleading, nicely describes the artefact, which is a noise signal occurring even before the music event that causes such noise.

  To understand the origin of Bitstream in pre-echoes, let us consider the decoder of a perceptual coding system (see Fig. 9.8). The reconstructed frequency lines are combined by the synthesis filterbank, consisting of a modulation matrix and a synthesis window. The quantization error introduced by the encoder can be seen as a signal added to the original frequency lines, with a length in time that is equal to the length of the synthesis window. Thus, reconstruction errors are spread over the full window length. If the music signal contains a sudden increase in signal energy (e.g. a castanet attack), the quantization error is increased as well. If such an attack occurs well within the analysis window, its error signal will be spread within the full synthesis window, preceding the actual cause for its existence in time (see Fig. 9.11). If such a pre-noise signal extends beyond the pre-masking period of the human ear, it becomes audible and is called *pre-echo*. There are a number of techniques to avoid audible pre-echoes, including changing the window lenght (Fig. 9.12), variable bit-rate coding or a local increase in the bit-rate to reduce the amplitude of the pre-echo. In general, these artefacts belong to the most difficult to avoid category.

- *Roughness, double-speak* Especially at low bit-rates and low sampling frequencies, there is a mismatch between time resolution of the coder and the time structure of some signals. This effect is most noticeable on speech signals and when listening via headphones. As a single voice tends to sound like it has been recorded twice and then overlaid, this effect is sometimes called *double-speak*.

Using Perceptual Audio Coding: Perceptual audio coding is intended for final delivery applications. It is not advisable for principle recording of signals, or in cases where the signal will be

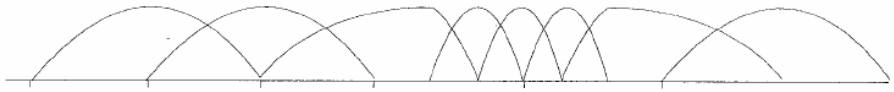

Figure 9.12: Changing the window length to match the signal properties of the input helps in avoiding pre-echoes. A long window is used to maximize coding gain and achieve good channel separation during segments identified as stationary, and a short window is used to localize time-domain artifacts when pre-echoes are likely.

processed heavily *after* the coding is applied. Perceptual audio coding is applicable where the signal will not be reprocessed, equalized, or otherwise modified before the final delivery to the consumer. If we are in a situation where we must do multiple encodings, we should avoid it to the extent possible and use a high bit rate for all but the final delivery bitstream. Finally, perceptual coding of audio is a very powerful technique for the final delivery of audio signals in situations where the delivery bit rate is limited, and/or when the storage space is small.

9.2 MIDI representation of music

Music is widely used in multimedia applications, so we require a media type for music to focus on the computers musical capabilities. We can distinguish two kinds of music representation, i.e. Operational vs. Symbolic. *Operational representations* specify exact timings for music and physical descriptions of the sounds to be produced, while *symbolic representations* use descriptive symbolism to describe the form of the music and allow great freedom in the interpretation. Both types are described as structural representations, since instead of representing music by audio samples there is information about the internal structure of the music

To illustrate the structural representations, we present MIDI, a widely used protocol allowing the connection of computers and musical equipment, an operational representation. The original Musical Instrument Digital Interface (MIDI) specification defined a physical connector and message format for connecting devices and controlling them in ”real time”. A few years later Standard MIDI Files were developed as a storage format so performance information could be recalled at a later date. The three parts of MIDI are often just referred to as ”MIDI”, even though they are distinctly different parts with different characteristics. Almost all recordings today uses MIDI as a key enabling technology for recording music. MIDI is also used in live performances to control stage lights and effects pedals.

The MIDI Message specification (or ”MIDI Protocol”) is probably the most important part of MIDI. Though originally intended just for use with the MIDI DIN transport as a means to connect two keyboards, MIDI messages are now used inside computers and cell phones to generate music, and transported over any number of professional and consumer interfaces (USB, FireWire, etc.) to a wide variety of MIDI-equipped devices. There are different message groups for different applications, only some of which will be explained here.

The final part of MIDI is the Standard MIDI File (and variants), which is used to distribute music playable on MIDI players of both the hardware and software variety. All popular computer platforms can play MIDI files (`*.mid`) and there are thousands of web sites offering files for sale or even for free. Anyone can make a MIDI file using commercial (or free) software that is readily available, and many people do, with a wide variety of results. The quality of a specific MIDI file can depend on how well it was created, and how accurately the synthesizer plays the file: not all synthesizers are the

same, and unless the one used for listening at is similar to that of the file composer, what is heard may not be at all what he or she intended.

9.2.1 MIDI Messages

MIDI started out as a music description language in digital (binary) form. It was designed for use with keyboard-based musical instruments, so the message structure is oriented to performance events, such as picking a note and then striking it, or setting typical parameters available on electronic keyboards. MIDI messages are used by MIDI devices to communicate with each other (Fig. 9.13). Every MIDI connection is a one-way connection from the MIDI Out connector of the sending device to the MIDI In connector of the receiving device. Each such connection can carry a stream of MIDI messages, with most messages representing a common musical performance event or gesture. All of those messages include *channel* number. There are 16 possible channels in the protocol. The channels are used to separate ”voices” or ”instruments”, somewhat like tracks in a multi-track mixer. The ability to multiplex 16 channels onto a single wire makes it possible to control several instruments at once using a single MIDI connection. When a MIDI instrument is capable of producing several independent sounds simultaneously (a multitimbral instrument), MIDI channels are used to address these sections independently. This should not be confused with ”polyphonic”; the ability to play several notes simultaneously in the same ”voice”.

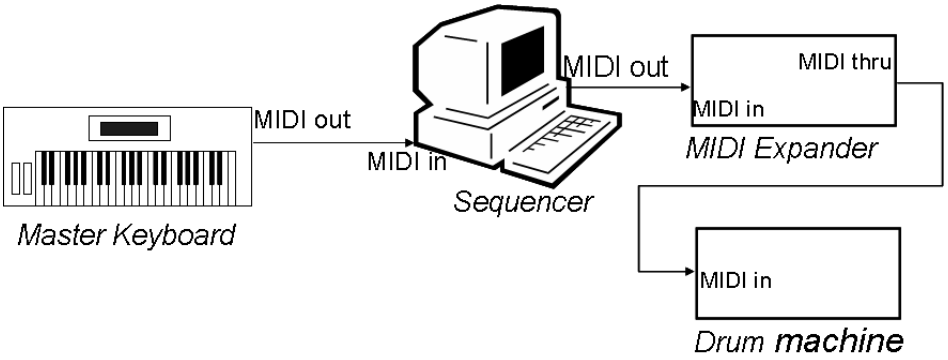

Figure 9.13: Example of a connection of MIDI devices.

MIDI specifies asynchronous serial communication using current-loop signalling at 31.25 Kbit/sec. There is 1 start bit, 8 data bits, and 1 stop bit; thus, each data byte takes 10 serial bits, and is sent in 320 microseconds. Since the majority of MIDI messages need three bytes, only about a thousand messages per second can be transmitted.

A MIDI message can consist of from one to several thousand bytes of data. The receiving instrument knows how many bytes to expect from the value of the first byte of the message. This byte is known as the status byte, the others are data bytes. Status bytes always have the most significant bit equal to 1 and it is used to inform the receiver as to what to do with incoming data. Many of the commands include the channel number (0-15) as the 4 low-order bits of the status byte. The 3 remaining bits identify the message. The most significant bit of data byte is set to 0 and thus actual data values are limited to numbers less than 128. This restricts many things in the MIDI universe, such as the number of presets.

There are a number of different types of MIDI messages. At the highest level, MIDI messages are classified as being either Channel Messages or System Messages. *Channel messages* are those which

apply to a specific Channel, and the Channel number is included in the status byte for these messages. *System messages* are not Channel specific, and no Channel number is indicated in their status bytes.

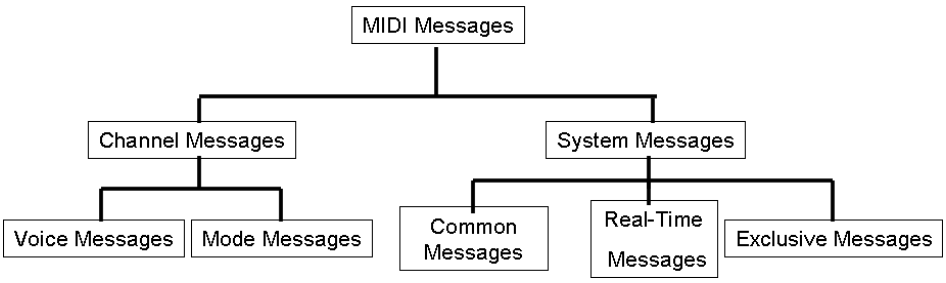

Figure 9.14: Organization of MIDI messages.

9.2.1.1 Channel messages

Channel Messages may be further classified as being either Channel Voice Messages, or Mode Messages. Channel Voice Messages carry musical performance data, and these messages comprise most of the traffic in a typical MIDI data stream. The messages in this category are the Note On, Note Off, Polyphonic Key Pressure, Channel Pressure, Pitch Bend Change, Program Change, and the Control Change messages. Channel Mode messages affect the way a receiving instrument will respond to the Channel Voice messages.

Voice Message	Status Byte	Mess. size	Data Byte1	Data Byte2
Note off	0x8c	2	Key number	Note Off velocity
Note on	0x9c	2	Key number	Note On velocity
Polyphonic Key Press.	0xAc	2	Key number	Amount of pressure
Control Change	0xBc	2	Controller	Controller value
Program Change	0xCc	1	Program num.	—
Channel Pressure	0xDc	1	Pressure val.	—
Pitch Bend	0xEc	2	bend LSB	bend MSB

Table 9.2: MIDI channel voice messages: c indicates the channel number.

Note On and Note Off In MIDI systems, the activation of a particular note and the release of the same note are considered as two separate events. When a key is pressed on a MIDI keyboard instrument or MIDI keyboard controller, the keyboard sends a *Note On* message on the MIDI OUT port. The keyboard may be set to transmit on any one of the sixteen logical MIDI channels, and the status byte (0x9c) for the Note On message will indicate the selected Channel number c. The Note On status byte is followed by two data bytes, which specify *note number* (indicating which note was pressed) and *key velocity* (how hard the key was pressed). The note number is used in the receiving synthesizer to select which note should be played, and the velocity is normally used to control the amplitude of the note. The note numbers start with 0 representing the lowest C. Middle C is note number 60. So, a MIDI note number of 69 is used for A440 tuning (i.e. the A note above middle C). For example to play note number 80 (= 0x50) with maximum velocity 127 (= 0x7F) on channel 14 (=

0xD), the MIDI device would send these three hexadecimal byte values: 0x9D 0x50 0x7F.
A tone with note number (MIDI pitch) p has frequency

$$f = 440 \times 2^{(p-69)/12} \text{ Hz} = 440 \times s^{p-69} \text{ Hz}$$

and a note with frequency f Hz has MIDI pitch

$$p = 69 + 12 \log_2(f/440)$$

where $s = \sqrt[12]{2} \approx 1.059$ is the semitone frequency ratio. Notice that frequencies, that are not in the tempered musical scale, can not be represented directly in MIDI.

When the key is released, the keyboard instrument or controller will send a *Note Off* message (status byte 0x8c). The Note Off message also includes data bytes for the note number and for the velocity with which the key was released. The Note Off velocity information is normally ignored. A velocity of zero in a Note On event is a Note Off event. This manner of thinking, requiring separate actions to start and stop a note, greatly simplifies the design of receiving instruments (the synthesizer does not have to keep time). Note Off is actually not used very much. Instead, MIDI allows for a shorthand, known as running status: a MIDI message can be sent without its Status byte (i.e., just its data bytes are sent) as long as the previous, transmitted message had the same Status.

Controllers with continuous values

Aftertouch. Some MIDI keyboard instruments have the ability to sense the amount of pressure which is being applied to the keys while they are depressed. This pressure information, commonly called "aftertouch", may be used to control some aspects of the sound produced by the synthesizer (vibrato, for example). If the keyboard has a pressure sensor for each key, then the resulting "polyphonic aftertouch" information would be sent in the form of *Polyphonic Key Pressure* messages (0xAc). These messages include separate data bytes for key number and pressure amount.

It is currently more common for keyboard instruments to sense only a single pressure level for the entire keyboard. This "Channel aftertouch" information is sent using the *Channel Pressure* message (0xDc), which needs only one data byte to specify the pressure value. Since the musician can be continually varying his pressure, devices that generate Polyphonic Key Pressure and Channel Pressure typically send out many such messages while the musician is varying his pressure.

Pitch Bend. The *Pitch Bend Change* message (0xEc) is normally sent from a keyboard instrument in response to changes in position of the pitch bend wheel. The pitch bend information is used to modify the pitch of sounds being played on a given Channel. The Pitch Bend message includes two data bytes to specify the pitch bend value. Two bytes are required to allow fine enough resolution to make pitch changes resulting from movement of the pitch bend wheel seem to occur in a continuous manner rather than in steps.

Control Change There is a group of commands called *Control Changes* (0xBc), that set a particular controller's value. A controller is any switch, slider, knob, etc, that implements some function (usually) other than sounding or stopping notes. Each command has two parts, defining which control to change and what to change it to. Controllers are numbered from 0 to 121, and some of them have defined purposes; e.g. control n. 1 = Modulation wheel; 2 = Breath controller; 4 = Foot controller, etc. The values 122-127 are reserved for special mode messages which affect the way a synthesizer

responds to MIDI data. A controller usually has a single data byte, giving a range of 0-127 as the value. This is rather coarse, so the controllers from 32 to 63 are reserved to give extra precision to those assigned from 0 to 31.

Program Change The sound of a synthesizer is determined by the connections between the modules and settings of the module controls. Very few current models allow repatching of the digital subroutines that substitute for modules, but they have hundreds of controls to set. The settings are just numbers, and are stored in the synthesizer memory. A particular group of settings is called a Patch, Preset, Voice, or Tone for different brands, but the official word is program. The *Program Change* message (0xBc) is used to specify the program (type of instrument) which should be used to play sounds on a given Channel. This message needs only one data byte which specifies the new program number.

9.2.1.2 System Messages

The preceding messages are Channel Voice Messages which apply only to instruments set to the specified channel. *System Messages* apply to all machines and carry information that is not channel specific, such as timing signal for synchronization, positioning information in pre-recorded MIDI sequences, and detailed setup information for the destination device.

System Real-Time Message	Status Byte
Timing Clock	0xF8
Start Sequence	0xFA
Continue Sequence	0xFB
Stop Sequence	0xFC
Active Sensing	0xFE
System Reset	0xFF

Table 9.3: MIDI System Real-Time Messages.

System Real Time messages are used to synchronize all of the MIDI clock-based equipment within a system, such as sequencers and drum machines. To help ensure accurate timing, System Real Time messages are given priority over other messages. The System Real Time messages are the Timing Clock, Start, Continue, Stop, Active Sensing, and the System Reset message. The Timing Clock message is the master clock which sets the tempo for playback of a sequence. The Timing Clock message is sent 24 times per quarter note. The Start, Continue, and Stop messages are used to control playback of the sequence. The Active Sensing signal is used to help eliminate ”stuck notes” which may occur if a MIDI cable is disconnected during playback of a MIDI sequence.

System Common Message	Status Byte	Number of Data Bytes
MIDI Timing Code	0XF1	1
Song Position Pointer	0XF2	2
Song Select	0XF3	1
Tune Request	0XF6	None

Table 9.4: MIDI System Common Messages.

System common messages which are currently defined include: the MIDI Timing Code (0xF1), used for synchronization of MIDI equipment and other equipment such as audio or video tape machines; The Song Select message (0xF3), used with MIDI equipment, such as sequencers or drum machines, which can store and recall a number of different songs; the Song Position Pointer (0xF3), used to set a sequencer to start playback of a song at some point other than at the beginning; the Song Position Pointer (0xF2) followed by 2 data bytes containing a value related to the number of MIDI clocks which would have elapsed between the beginning of the song and the desired point in the song.

System exclusive messages are related to things that cannot be standardized and addition to the original MIDI specification. It is just a stream of bytes, all with their high bits set to 0, bracketed by a pair of system exclusive start and end messages (0xF0 and 0xF7). System Exclusive messages may be used to send data such as patch parameters or sample data between MIDI devices. Manufacturers of MIDI equipment may define their own formats for System Exclusive data. Manufacturers are granted unique identification (ID) which is included as part of the System Exclusive message.

A Simple MIDI Stream The hexadecimal bytes in Table 9.2.1.2 are a simple, typical stream of MIDI data. Each message must be sent at the real-time that the synthesizer is supposed to perform the action. Sophisticated sequencers will send the message slightly ahead of time to account for the duration of the message – one third of a millisecond per byte – and the response time of the particular synthesizer – perhaps several milliseconds. The perceptual improvement of accuracy under 5 or 10 milliseconds, to the untrained listener, is questionable.

MIDI bytes	Description
90 3C 40	Play middle C (note number 60, hexadecimal 3C) on channel 1 (the zero nibble of 90), at half the full velocity (velocity 64, hexadecimal 40).
43 40	Play note G above middle C (note number 67, hexadecimal 43) at velocity 64. Note that the status byte of 90 from the first message is still in effect, and did not have to be resent.
B9 07 33	Change the volume of MIDI channel 10 to 51 (hexadecimal 33). The MIDI volume controller number is 7.
B3 07 10	Change the volume of MIDI channel 8 to 16 (hexadecimal 10). Since the channel number is different than the previous message, running status could not be used.
90 3C 00	Stop playing middle C on channel 1. The last byte of zero indicates a ”key down velocity of 0” which indicates a ”note-off” event.
80 43 64	Stop playing the G above middle C (key number 67, hexadecimal 43). The key is released with a velocity of 64. Since very few, if any, synthesizers implement an interpretation of the note-off volume, this message is generally equivalent to 90 43 00.

Table 9.5: Example of a simple MIDI stream.

9.2.2 MIDI files

9.2.2.1 Standard MIDI Files

When MIDI messages are stored on disks, they are commonly saved in the Standard MIDI file (SMF) format, which is slightly different from native MIDI protocol, because the events are also time-stamped for playback in the proper sequence. Music delivered by MIDI files is the most common use of MIDI today. Just about every personal computer is now equipped to play Standard MIDI files. One reason for the popularity of MIDI files is that, unlike digital audio files (.wav, .aiff, etc.) or even compact discs or cassettes, a MIDI file does not need to capture and store actual sounds. Instead, the MIDI file can be just a list of events which describe the specific steps that a soundcard or other playback device must take to generate certain sounds. This way, MIDI files are very much smaller than digital audio files, and the events are also editable, allowing the music to be rearranged, edited, even composed interactively, if desired.

MIDI files are typically created using desktop/laptop computer-based sequencing software (or sometimes a hardware-based MIDI instrument or workstation) that organizes MIDI messages into one or more parallel "tracks" for independent recording and editing. In most but not all sequencers, each track is assigned to a specific MIDI channel and/or a specific General MIDI instrument patch. An SMF consists of one header chunk and one or more track chunks. There are three SMF formats; the format is encoded in the file header. Format 0 contains a single track and represents a single song performance. Format 1 may contain any number of tracks, enabling preservation of the sequencer track structure, and also represents a single song performance. Format 2 may have any number of tracks, each representing a separate song performance.

Many values are stored in a variable-length format which may use one or more bytes per value. Variable-length values use the lower 7 bits of a byte for data and the top bit to signal a following data byte. If the top bit is set to 1, then another value byte follows. Table 9.6 presents some examples to help demonstrate how variable length values are used.

Value		Variable length	
Hex	Bin	Hex	Bin
00	0000 0000	00	0000 0000
48	0100 1000	48	0100 1000
7F	0111 1111	7F	0111 1111
80	1000 0000	81 00	1000 0001 0000 0000
C8	1100 1000	81 48	1000 0001 0100 1000

Table 9.6: Examples of values and their variable-length equivalents.

Track events are used to describe all of the musical content of a MIDI file, from tempo changes to sequence and track titles to individual music events. Each event includes a delta time, event type and usually some event type specific data. The *event delta time* is defined by a variable-length value. It determines when an event should be played relative to the track's last event. A delta time of 0 means that it should play simultaneously with the last event. A track's first event delta time defines the amount of time to wait before playing this first event. Events unaffected by time are still preceded by a delta time, but should always use a value of 0 and come first in the stream of track events. Examples of this type of event include track titles and copyright information. The most important thing to remember about delta times is that they are relative values, not absolute times. The actual

time they represent is determined by a couple factors: the time division (defined in the MIDI header chunk) and the tempo (defined with a track event). If no tempo is defined, 120 beats per minute is assumed.

There are three types of events: MIDI Channel Events, System Exclusive Events and Meta Events. The first two types are represented as explained in the previous section. Meta Events include specifications for tempo, time and key signature, lyrics, sequence and track numbers, score markers, copyright notice.

As an example, MIDI Files for the following excerpt are shown in Table 9.7. A format 0 file is used, with all information intermingled. A resolution of 96 ticks per quarter note is used. A time signature of 4/4 and a tempo of 120, though implied, are explicitly stated.

Delta Time (decimal)	Event Code (hex)	Other Bytes (decimal)	Comment
0	FF 58	04 04 02 24 08	4 bytes: 4/4 time, 24 MIDI clocks/click, 8 32nd notes/24 MIDI clocks
0	FF 51	03 500000	3 bytes: 500,000 5sec per quarter-note
0	C0	5	Ch. 1, Program Change 5
0	C0	5	Ch. 1, Program Change 5
0	C1	46	Ch. 2, Program Change 46
0	C2	70	Ch. 3, Program Change 70
0	92	48 96	Ch. 3 Note On C2, forte
0	92	60 96	Ch. 3 Note On C3, forte
96	91	67 64	Ch. 2 Note On G3, mezzo-forte
96	90	76 32	Ch. 1 Note On E4, piano
192	82	48 64	Ch. 3 Note Off C2, standard
0	82	60 64	Ch. 3 Note Off C3, standard
0	81	67 64	Ch. 2 Note Off G3, standard
0	80	76 64	Ch. 1 Note Off E4, standard
0	FF 2F	00	Track End

Table 9.7: Examples of midi file events.

9.2.2.2 General MIDI

General MIDI (GM) is a response to a problem that arose with the popularity of the Standard MIDI file. As composers began exchanging compositions (and selling them) in SMF format, they discovered that pieces would change when played on different synthesizers. That's because the MIDI program commands simply provide a number for a preset. What sound you get on preset four is anybody's guess.

General MIDI is a specification for synthesizers which imposes several requirements beyond the more abstract MIDI standard. While MIDI itself provides a protocol which ensures that different instruments can interoperate at a fundamental level (e.g. that pressing keys on a MIDI keyboard will cause an attached MIDI sound module to play musical notes), GM goes further in two ways: it requires that all GM-compatible instruments meet a certain minimal set of features, such as being able

to play at least 24 notes simultaneously (polyphony), and it attaches certain interpretations to many parameters and control messages which were left unspecified in MIDI.

The *Instrument patch map* is a standard program list consisting of 128 patch types. The patches are arranged into 16 ”families” of instruments, with each family containing 8 instruments. Program numbers 1-8 are piano sounds, 9-16 are chromatic percussion sounds, 17-24 are organ sounds, 25-32 are guitar sounds, etc.. For example among the 8 instruments within the Reed family, you will find Saxophone, Oboe, and Clarinet. Channel 10 is reserved for percussion under GM; this channel always sounds as percussion regardless of whatever program change numbers it may be sent, and different note numbers are interpreted as different instruments.

The *Percussion map* specifies 47 percussion sounds, e.g. A note-on with note number 402 will trigger a Electric Snare. Note that GM specifies which instrument or sound corresponds with each program/patch number, but it does not specify how these sounds are produced. Thus for example a Clarinet sound on two GM synthesizers which use different synthesis techniques may sound quite different. GM also specifies which operations should be performed by several controllers.

GM Standard makes it easy for musicians to put Program Change messages in their MIDI (sequencer) song files, confident that those messages will select the correct instruments on all GM sound modules, and the song file would therefore play all of the correct instrumentation automatically. Finally, musicians didn't have to worry that a snare drum part, for example, would be played back on a Cymbal. All of these standards help to ensure that MIDI Files play back properly upon setups of various equipment. GM is most important in the soundcards that plug into PCs. These allow game programmers to create MIDI based scores instead of including recorded sounds for the music cuts. On the other hand it greatly limits the possible freedom of the composer.

9.2.2.3 MIDI Timing

Any MIDI device which records data (e.g. a sequencer) needs timing information so that the data can be replayed at the correct rate. Those MIDI devices which do not record data (e.g. synthesizers and effects units) do not use timing information. Timing information can be transferred between MIDI devices via the MIDI link. There is no need for separate synchronization connections. MIDI provides two ways to count time: via MIDI Clock messages or via MIDI Timecode.

MIDI Clock signals are not simple pulses. Each clock message is a single byte. Clock messages are sent at the rate of 6 per semiquaver (a semiquaver = ”sixteenth note”), i.e. a rate of 24 per quarter note. So, a MIDI sequencer refers to a position in a sequence of events in terms of musical divisions such as beats and bars. The MIDI beat does not therefore occupy a fixed amount of time. Its duration depends on the speed (”tempo”) of the music.

This is in contrast with SMPTE timecode where the timing information relates to *absolute time* (measured in hours, minutes, seconds, frames). Both methods of determining position are useful in their respective applications but in the modern studio it is necessary to reconcile the two. Nowadays recording studios integrate MIDI systems and audio recording systems. This integration is achieved by relating MIDI timing data to timecode recorded onto tape.

MIDI timecode (MTC) were introduced as a way of converting SMPTE timecode into MIDI messages. MTC generates absolute time signals that synchronize SMPTE with MIDI devices. There are two kinds of MTC message: *quarter-frame messages*, which update the receiving device regularly when the transmitting device is running at normal speed, and *full-frame messages* are used during spooling, when updating at the quarter-frame rate would result in an excessive rate of data transmission. Timing information received from a tape recorder using conventional timecode (SMPTE or EBU) is thus distributed as MIDI timecode (MTC) around a MIDI system.

In addition to the synchronization of tape recorders and sequencers, MTC is increasingly being used in a variety of automated equipment. Mixing console automation is currently resulting in interesting developments in the integration of mixer control and MIDI sequencing. MTC can be used to control channel routing and muting, leaving the engineer free to experiment with levels, EQ, panning etc. A number of mixer settings can be prepared in advance away from the mixing console (prepared "off-line").

9.2.3 Discussion on MIDI representation

9.2.3.1 MIDI limitations

While MIDI performance capabilities has been a great boom for computer music software applications since 1984, it has limitations when compared to replicating authentic sounding real-time live musical performances.

Bandwidth limitations. MIDI messages are extremely compact, due to the low bandwidth of the connection, and the need for real-time accuracy. However, the serial nature of MIDI messages means that long strings of MIDI messages take an appreciable time to send, at times even causing audible delays, especially when dealing with dense musical information or when many channels are particularly active. Notice that every Note On messages takes about 1 ms to transmit. The protocol was designed to record the keyboard performances of one to four human keyboard players without much continuous controller manipulation. The bandwidth can be overwhelmed by a single virtuoso.

Performance nuances limitation. MIDI measures many variables between 0-127 (not note placement.). So the velocity of a kick drum is always between 0-127 which might not seem like a big deal but if you emulating real instruments something closer to infinity is necessary. Interpretative ingredients of music performances can not be easily included or controlled in MIDI files unless done with individual adjustments made to every note. Such a process is not practical because it is so time consuming with the specific attention directed to single notes with regards to attack, delay, sustaining, vibrato, pulse and other creative performance elements. Consequently, MIDI file performances can easily sound contrived, predictable musical expressiveness or mechanical rather than like live performances where the individual musical nuances are created for particular interpretations.

Music representation limitations. A more fundamental constraint of MIDI is its concept of music embodied in its design. It is biased toward popular songs (metered, in equal temperament) as played on a musical keyboard. Moreover MIDI lacks of representation of timbre. General MIDI mode table was designed for home entertainment and not for professional musicians: it includes only a tiny fractions of the timbres possible in computer music. MIDI concept of pitch is based on equal temperament. It is possible to detune a pitch with a Pitch Bend message, but it is a global operation applying to all notes on a channel. One of the justification of computer music is the ability to go beyond the pitch, timing and timbre limitations of traditional instruments.

9.2.3.2 Operation on music

In considering music representation, we can recognize several advantages over audio: music representation will be more compact than audio; it is portable and can be synthesized with the fidelity and complexity appropriate to the output devices used; while digital audio suffers from inherent noise, musical representations are noise free; many operations can be performed on music that would be infeasible or require extensive processing on audio.

- Playback and Synthesis. During audio playback, the listener has limited influence over the musical aspects of the performance, beyond changing the volume or processing the audio in some way. If music is produced by synthesis from a structural representation the listener can independently change pitch and tempo, increase or decrease individual instruments volumes or change the sounds they produce. Musical representations offer greater potential for interactivity than audio

- Timing. Structural representation makes timing of musical events explicit. The ability to modify tempo makes it possible to alter the timing of groups of musical events and adjust the synchronization of those events with other events (film, video, etc.)

- Editing and Composition. Basic editing allows the user to modify primitive events and notes; more complex editing operations operate on musical aggregates (chords, bars, etc.) to permit phrase-repetition, melody replacement and other such functions. Composition software simplifies the task of generating and combining or rearranging tracks, and prints the score

MIDI sequence editing A sequencer is a multitrack recorder for MIDI information. It allows you to record, overdub, and edit MIDI performances on different channels and play them back simultaneously, triggering sounds from your MIDI instrument. Sequencers are often included in many of today's keyboards, but the most popular way to record MIDI is by using computer-based sequencing software.

In addition to channels, MIDI sequencers use tracks and patterns as organizing concept. *Tracks* are arbitrary units of musical organization in a sequencer. It may refers to separate recording versions of a musical line, with the idea of selecting one track for the final mix at a later stage. Another way is refers to different layers of a composition, such as a track for each synthesizer voice, separate tracks for each controller etc..

Pattern orientation means that the data can be broken into subsequences within a sequence. A pattern can be looped, played back in combination with other patterns. Moreover a composer can add, delete, transpose or transform a pattern The same as note events.

For editing purposes, MIDI information can be displayed in several representations. An *event list* represent MIDI data in an alphanumeric listing, sorted in time order. It is used to fine tuning a sequence. In *Piano roll* representation, individual notes are laid down vertically, while the start time and duration of events are encoded as a horizontal line. Some sequencers can display MIDI data into a form of common music notation. To this purpose metronome information is useful and time quantization is necessary to take into account performers deviations from exact metrical time.

Contents

9 Standards for audio and music representation **9.1**

9.1 Digital audio compression 9.1
9.1.1 Bit Rate Reduction 9.1
9.1.2 Auditory Masking and Perceptual Coding 9.3
9.1.2.1 Auditory Masking 9.3
9.1.2.2 Perceptual Coding 9.6
9.1.3 MPEG 1 Layer-3 coding 9.8
9.1.3.1 The Psychoacoustic Model 9.8
9.1.3.2 The Problem of Stereo Coding 9.10
9.1.3.3 Discussion on MPEG Layer-3 . . . . . 9.11
9.2 MIDI representation of music 9.14
9.2.1 MIDI Messages 9.15
9.2.1.1 Channel messages 9.16
9.2.1.2 System Messages 9.18
9.2.2 MIDI files 9.20
9.2.2.1 Standard MIDI Files 9.20
9.2.2.2 General MIDI 9.21
9.2.2.3 MIDI Timing 9.22
9.2.3 Discussion on MIDI representation 9.23
9.2.3.1 MIDI limitations 9.23
9.2.3.2 Operation on music 9.23