CHAPTER 6

Audio compression in practice

In earlier chapters we have seen that digital sound is simply an array of numbers, where each number is a measure of the air pressure at a particular time. This representation makes it simple to manipulate sound on a computer by doing calculations with these numbers. We have seen examples of simple filters which change the sound in some preferred way like removing high or low frequencies or adding echo. We have also seen how an audio signal can be rewritten in terms of a multiresolution analysis which may be advantageous if we want to compress the data that represent the sound.

In this chapter we will very briefly review some practical aspects of sound processing. In particular we will briefly discuss how a sound may be represented in terms of trigonometric functions and review the most important digital audio formats.

6.1 Wavelet based compression

In chapter 10 in the Norwegian lecture notes, we described a way to decompose a set of points by comparing representations at different resolutions with the help of piecewise linear functions, a so-called *multiresolution analysis*. This illustrated the idea behind a family of functions called *wavelets* (although wavelets should have some additional properties that we have ignored to keep the presentation simple). In this section we will try to compress sound with the multiresolution analysis.

Example 6.1. Figure 6.1 shows an example with real audio data. The data in (a) were decomposed once which resulted in the coarser signal in (b) and the error signal in (c). Then all error values smaller than 0.02 set to zero which leads to the signal in (d). The reconstruction algorithm was then run with the data in (b) and (d); the result is shown in (e). Figure (f) shows the difference between the two signals.

The same computations were performed with a larger data set consisting of 300 001 points from the same song. Out of 150 000 error values 141 844 were smaller than 0.02 in absolute value. When these were set to 0 and the sound reconstructed with the perturbed data, the music still sounded quite good. The

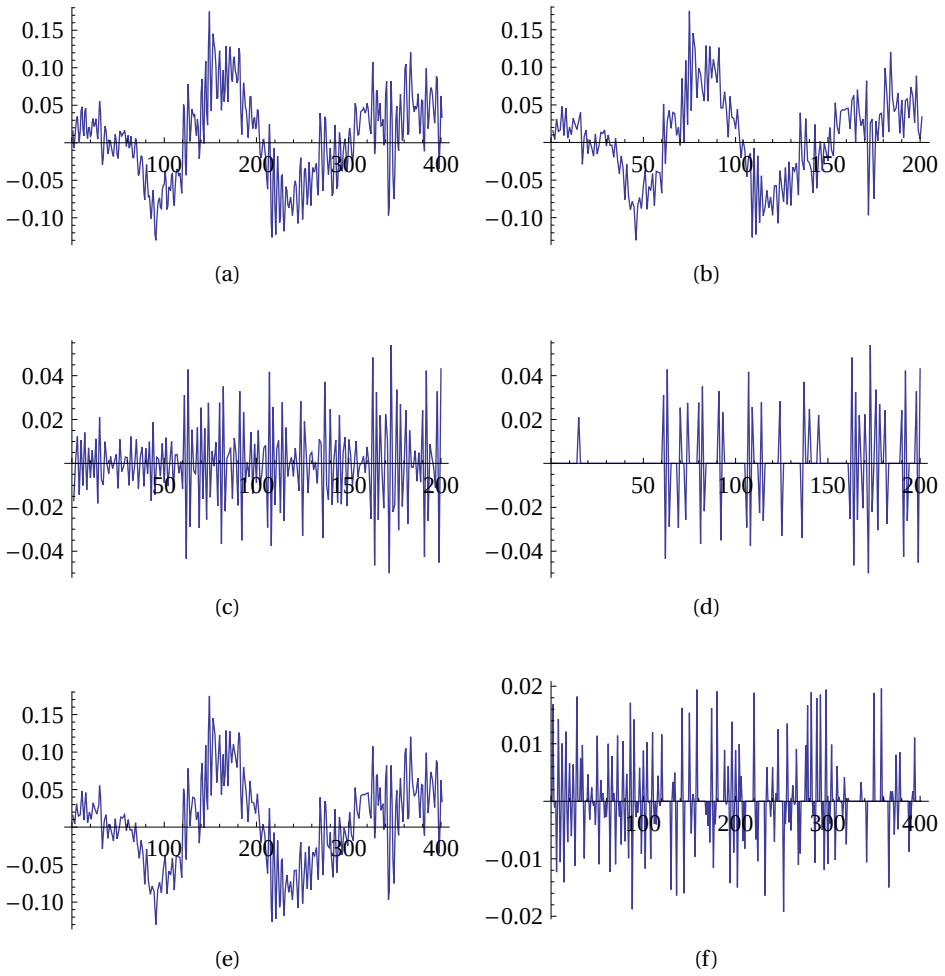

Figure 6.1. An example of data decomposition with the multiresolution analysis from chapter 10 of the Norwegian lecture notes. Figure (a) shows the piecewise linear interpolant to data taken from the song 'Here Comes the Sun' by the Beatles. Figure (b) shows the piecewise linear interpolant to every other data point, and figure (c) shows the error function. In (d) we have set all error values smaller than 0.02 to zero, altogether 148 out of 200 values. In (e) the data have been reconstructed from the functions in (c) and (d). Figure (f) shows the difference between the original data and the reconstructed data.

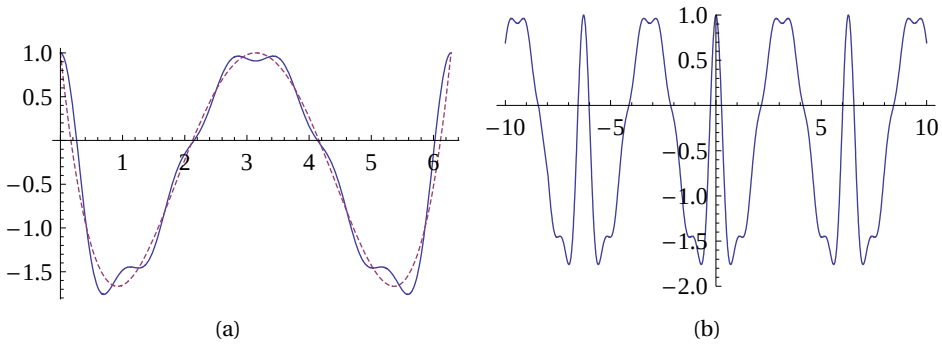

Figure 6.2. Trigonometric approximation of a cubic polynomial on the interval $[0, 2\pi]$. In (a) both functions are shown while in (b) the approximation is plotted on the interval $[-10, 10]$.

maximum difference between this perturbed signal and the original was just under 0.02.

To check the possibility of compression, the data were written to a file as 32-bit floating-point numbers and then compressed with `gzip`. This resulted in a file with 389 829 bytes. When the original data were written to file and compressed with `gzip`, the file size was 705 475 bytes. In other words, the sound has been compressed down to 55 % with a very simple algorithm, without too much deterioration in the sound quality. ∎

6.2 Fourier analysis and the DCT

As we have seen, multiresolution analysis and wavelets provide a convenient framework for compression of audio files. The theory behind wavelets was developed in the late 1980s and was not introduced into much commercial software until the late 1990s. Common audio formats like MP3, AAC and Ogg Vorbis were developed before this time and use a mathematical operation called the *Discrete Cosine Transform* (DCT) to transform the audio data to a form that lends itself well to compression. Before we consider the basics of the DCT, we need some background information.

A very common technique in mathematics is to approximate a general function by a function from some appropriate family. We have seen several examples of this, for example approximation by Taylor polynomials and approximation by piecewise linear functions. Another possibility is to approximate functions by combinations of trigonometric functions. This is a technique that is used in many different areas of mathematics and is usually referred to as *Fourier analysis*.

In Fourier analysis, general functions are approximated by combinations of the functions

$$1, \sin x, \cos x, \sin 2x, \cos 2x, \sin 3x, \cos 3x, \sin 4x, \cos 4x, \ldots \tag{6.1}$$

A simple example is shown in figure 6.2. There a cubic polynomial has been approximated by a trigonometric function on the form

$$a_0 + b_1 \sin x + a_1 \cos x + b_2 \sin 2x + a_2 \cos 2x. \tag{6.2}$$

The coefficients have been determined in such a way that the error should be small on the interval $[0,1]$. This has been quite successful in that we can hardly differentiate between the two functions in figure 6.2a. In figure 6.2b the approximation has been plotted on the interval $[-10,10]$. From this plot we see that the approximation is periodic, as is the case for all trigonometric functions, and therefore very different from the polynomial as we come outside the interval $[0,1]$. The fact that the error was small in figure 6.2a was no accident; it can be proved that quite general functions can be approximated arbitrarily well by trigonometric functions.

Recall that $\sin kt$ corresponds to a pure tone of frequency $k/(2\pi)$. If we think of a certain sound as a function, the fact that any function can be expressed as a combination of the trigonometric functions in (6.1) means that any sound can be generated by combining a set of pure tones with different weights, just as in (6.2). So given a sound, it can always de decomposed into different amounts of pure tones be represented by suitable trigonometric functions.

Fact 6.2 (Fourier analysis)**.** *Any reasonable function f can be approximated arbitrarily well on the interval $[0,2\pi]$ by a combination*

$$a_0 + \sum_{k=1}^{N} (a_k \cos kt + b_k \sin kt), \tag{6.3}$$

where N is a suitable integer. This means that any sound can be decomposed into different amounts (the coefficients $\{a_k\}$ and $\{b_k\}$) of pure tones (the functions $\{\cos kt\}$ and $\{\sin kt\}$). This is referred to by saying that f has been converted to the frequency domain and the coefficients $\{a_k\}$ and $\{b_k\}$ are referred to as the spectrum of f.

When a given sound has been decomposed as in (6.3), we can adjust the sound by adjusting the coefficients. The trigonometric functions with large values of k correspond to high frequencies and the ones with small values of k correspond to low frequencies. We can therefore adjust the treble and bass by manipulating the appropriate coefficients.

There is an extensive theory for audio processing based on Fourier analysis that is studied in the field of *signal processing*.

6.2.1 The Discrete Cosine Transform

We are particularly interested in digital audio, i.e., sound given by values sampled from an audio signal at regular intervals. We have seen how digital sound can be compressed by using a multiresolution analysis, but this can also be done with ideas from Fourier analysis.

For the purpose of compressing sound it is not so common to use Fourier analysis as indicated above. Instead only cosine functions are used for decomposition.

Definition 6.3 (Discrete Cosine Transform (DCT))**.** *Suppose the sequence of numbers* $\boldsymbol{u} = \{u_s\}_{s=0}^{n-1}$ *are given. The DCT of* $\boldsymbol{u}$ *is given by the sequence*

$$v_s = \frac{1}{\sqrt{n}} \sum_{r=0}^{n-1} u_r \cos\Big(\frac{(2r+1)s\pi}{2n}\Big), \quad \text{for } s = 0, \ldots, n-1. \tag{6.4}$$

The new sequence $\boldsymbol{v}$ generated by the DCT tells us how much the sequence $\boldsymbol{u}$ contains of the different frequencies. For each $s = 0, 1, \ldots, n-1$, the function $\cos s\pi t$ is sampled at the points $t_r = (2r+1)/(2n)$ for $r = 0, 1, \ldots, n-1$ which results in the values

$$\cos\Big(\frac{s\pi}{2n}\Big), \quad \cos\Big(\frac{3s\pi}{2n}\Big), \quad \cos\Big(\frac{5s\pi}{2n}\Big), \quad \ldots, \quad \cos\Big(\frac{(2n-1)s\pi}{2n}\Big).$$

These are then multiplied by the u_r and everything is added together.

Plots of these values for $n = 6$ are shown in figure 6.3. We note that as s increases, the functions oscillate more and more. This means that v_0 gives a measure of how much constant content there is in the data, while v_5 gives a measure of how much content there is with maximum oscillation. In other words, the DCT of an audio signal shows the proportion of the different frequencies in the signal.

Once the DCT of $\boldsymbol{u}$ has been computed, we can analyse the frequency content of the signal. If we want to reduce the bass we can decrease the v_s-values with small indices and if we want to increase the treble we can increase the v_s-values with large indices. In a typical audio signal there will be most information in the lower frequencies, and some frequencies will be almost completely absent, i.e., some of the v_s-values will be almost zero. This can exploited for compression: We change the small v_s-values a little bit and set them to 0, and then

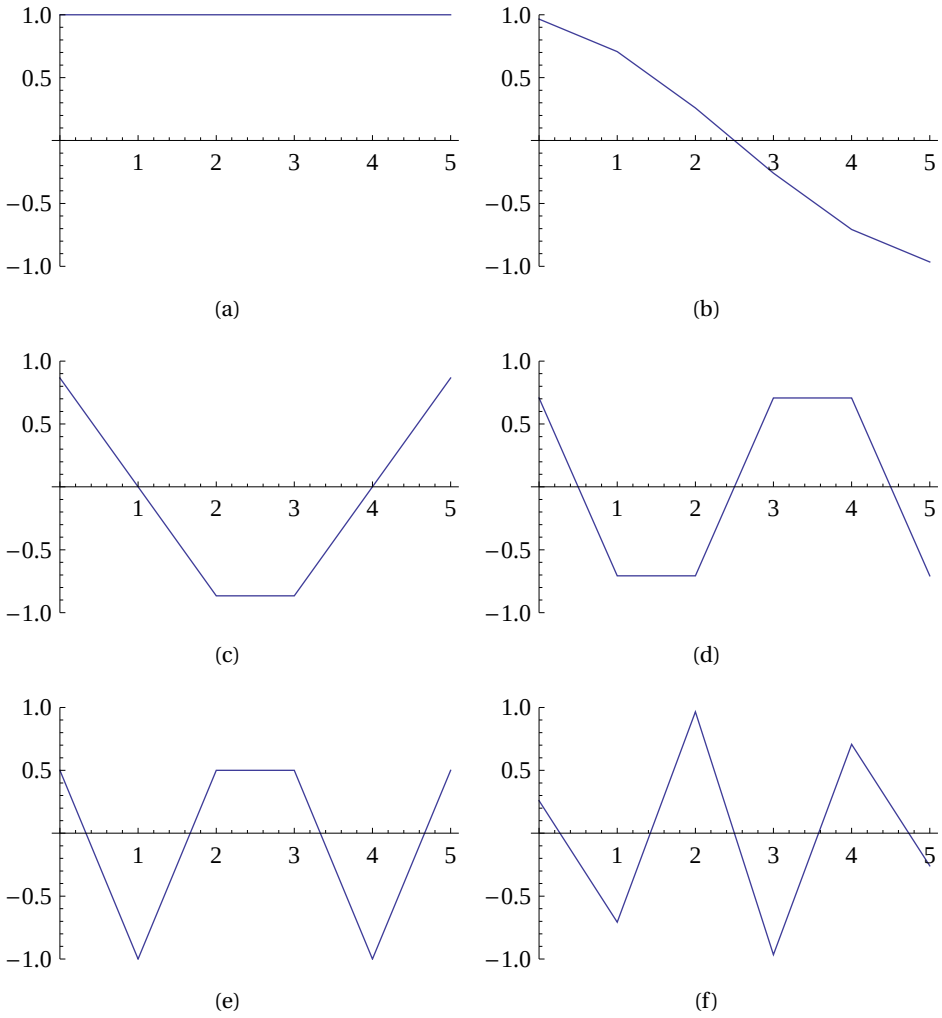

Figure 6.3. The 6 different versions of the cos function used in DCT for $n = 6$. The plots show piecewise linear functions, but this is just to make the plots more readable: Only the values at the integers $0, \ldots, 5$ are used.

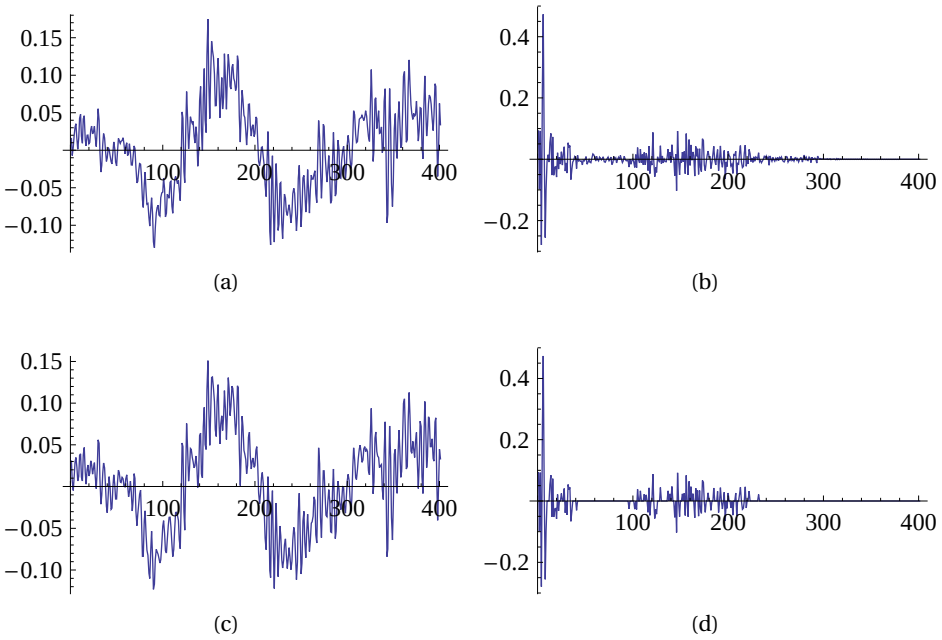

Figure 6.4. The signal in (a) is a small part of a song (the same as in figure 6.1b). The plot in (b) shows the DCT of the signal. In (c), all values of the DCT that are smaller than 0.02 in absolute value have been set to 0, a total of 309 values. In (c) the signal has been reconstructed from these perturbed values of the DCT. Note that all signals are discrete; the values have been connected by straight lines to make it easier to interpret the plots.

store the signal by storing the DCT-values. This leaves us with one challenge: How to convert the perturbed DCT-values back to a normal signal. It turns out that this can be accomplished with formulas very similar to the DCT itself.

Theorem 6.4 (Inverse Discrete Cosine Transform)**.** *Suppose that the sequence* $\boldsymbol{v} = \{v_s\}_{s=0}^{n-1}$ *is the DCT of the sequence* $\boldsymbol{u} = \{u_r\}_{r=0}^{n-1}$ *as in* (6.4)*. Then* $\boldsymbol{u}$ *can be recovered from* $\boldsymbol{v}$ *via the formulas*

$$u_r = \frac{1}{\sqrt{n}}\left(v_0 + 2\sum_{s=1}^{n-1} v_s \cos\left(\frac{(2r+1)s\pi}{2n}\right)\right), \quad \text{for } r = 0, \ldots, n-1. \tag{6.5}$$

We now see the resemblance with multiresolution analysis. The DCT provides another way to rewrite a signal in an alternative form where many values become small or even 0, and this can then be exploited for compression.

Example 6.5. Let us test a naive compression strategy similar to the one in ex-

ample 6.1 where we just replace the multiresolution decomposition and reconstruction formulas with the DCT and its inverse. The plots in figure 6.4 illustrate the principle. A signal is shown in (a) and its DCT in (b). In (d) all values of the DCT with absolute value smaller than 0.02 have been set to zero. The signal can then be reconstructed with the inverse DCT of theorem 6.4; the result of this is shown in (c). The two signals in (a) and (b) visually look almost the same even though the signal in (c) can be represented with less than 25 % of the information present in (a).

The larger data set from example 6.1 consists of 300 001 points. We compute the DCT and set all values smaller than a suitable tolerance to 0. It turns out that we need to use a tolerance as small as 0.004 to get an error in the signal itself that is comparable to the corresponding error in example 6.1. With a tolerance of 0.04, a total of 142 541 values are set to zero. When we then reconstruct the sound with the inverse DCT, we obtain a signal that differs at most 0.019 from the original signal which is about the same as in example 6.1. We can store the signal by storing a `gzip`'ed version of the DCT-values (as 32-bit floating-point numbers) of the perturbed signal. This gives a file with 622 551 bytes, which is 88 % of the `gzip`'ed version of the original data.

Recall that when we used the wavelet transform in example 6.1, we ended up with a file size of 389 829, which is considerably less than what we obtained here, without the error being much bigger. From this simple example, wavelets therefore seem promising for compression of audio. ∎

The approach to compression that we have outlined in the above examples is essentially what is used in practice. The difference is that commercial software does everything in a more sophisticated way and thereby get better compression rates.

Fact 6.6 (Basic idea behind audio compression)**.** *Suppose a digital audio signal* $\boldsymbol{u}$ *is given. To compress* $\boldsymbol{u}$*, perform the following steps:*

1. *Rewrite the signal* $\boldsymbol{u}$ *in a new format where frequency information becomes accessible.*
2. *Remove those frequencies that only contribute marginally to human perception of the sound.*
3. *Store the resulting sound by coding the remaining frequency information with some lossless coding method.*

All the compression strategies used in the commercial formats that we re-

view below, use the strategy in fact 6.6. In fact they all use a modified version of the DCT in step 1 and a variant of Huffman coding in step 3. Where they vary the most is probably in deciding what information to remove from the signal. To do this well requires some knowledge of human perception of sound.

6.3 Psycho-acoustic models

In the previous sections, we have outlined a simple strategy for compressing sound. The idea is to rewrite the audio signal in an alternative mathematical representation where many of the values are small, set the smallest values to 0, store this perturbed signal and compress it with a lossless compression method.

This kind of compression strategy works quite well, and is based on keeping the difference between the original signal and the compressed signal small. However, in certain situations a listener will not be able to perceive the sound as being different even if this difference is quite large. This is due to how our auditory system interprets audio signals and is referred to as *psycho-acoustic* effects.

When we hear a sound, there is a mechanical stimulation of the ear drum, and the amount of stimulus is directly related to the size of the sample values of the digital sound. The movement of the ear drum is then converted to electric impulses that travel to the brain where they are perceived as sound. The perception process uses a Fourier-like transformation of the sound so that a steady oscillation in air pressure is perceived as a sound with a fixed frequency. In this process certain kinds of perturbations of the sound are hardly noticed by the brain, and this is exploited in lossy audio compression.

The most obvious psycho-acoustic effect is that the human auditory system can only perceive frequencies in the range 20 Hz – 20 000 Hz. An obvious way to do compression is therefore to remove frequencies outside this range, although there are indications that these frequencies may influence the listening experience inaudibly.

Another phenomenon is *masking effects*. A simple example of this is that a loud sound will make a simultaneous quiet sound inaudible. For compression this means that if certain frequencies of a signal are very prominent, most of the other frequencies can be removed, even when they are quite large.

These kinds of effects are integrated into what is referred to as a *psycho-acoustic* model. This model is then used as the basis for simplifying the spectrum of the sound in way that is hardly noticeable to a listener, but which allows the sound to be stored with must less information than the original.

6.4 Digital audio formats

Digital audio first became commonly available when the CD was introduced in the early 1980s. As the storage capacity and processing speeds of computers increased, it became possible to transfer audio files to computers and both play and manipulate the data. However, audio was represented by a large amount of data and an obvious challenge was how to reduce the storage requirements. Lossless coding techniques like Huffman and Lempel-Ziv coding were known and with these kinds of techniques the file size could be reduced to about half of that required by the CD format. However, by allowing the data to be altered a little bit it turned out that it was possible to reduce the file size down to about ten percent of the CD format, without much loss in quality.

In this section we will give a brief description of some of the most common digital audio formats, both lossy and lossless ones.

6.4.1 Audio sampling — PCM

The basis for all digital sound is sampling of an analog (continuous) audio signal. This is usually done with a technique called Pulse Code Modulation (PCM). The audio signal is sampled at regular intervals and the sampled values stored in a suitable number format. Both the sampling rate and the number format varies for different kinds of audio. For telephony it is common to sample the sound 8000 times per second and represent each sample value as a 13-bit integer. These integers are then converted to a kind of 8-bit floating-point format with a 4-bit mantissa. Telephony therefore generates 64 000 bits per second.

The classical CD-format samples the audio signal 44 100 times per second and stores the samples as 16-bit integers. This works well for music with a reasonably uniform dynamic range, but is problematic when the range varies. Suppose for example that a piece of music has a very loud passage. In this passage the samples will typically make use of almost the full range of integer values, from $-2^{15}-1$ to 2^{15}. When the music enters a more quiet passage the sample values will necessarily become much smaller and perhaps only vary in the range -1000 to 1000, say. Since $2^{10} = 1024$ this means that in the quiet passage the music would only be represented with 10-bit samples. This problem can be avoided by using a floating-point format instead, but very few audio formats appear to do this.

Newer formats with higher quality are available. Music is distributed in various formats on DVDs (DVD-video, DVD-audio, Super Audio CD) with sampling rates up to 192 000 and up to 24 bits per pixel. These formats also support surround sound (up to seven channels as opposed to the two stereo channels on CDs).

Both the number of samples per second and the number of bits per sample influence the quality of the resulting sound. For simplicity the quality is often measured by the number of bits per second, i.e., the product of the sampling rate and the number of bits per sample. For standard telephony we saw that the bit rate is 64000 bits per second or 64 kb/s. The bit rate for CD-quality stereo sound is $44100 \times 2 \times 16$ bits/s = 1411.2 kb/s. This quality measure is particularly popular for lossy audio formats where the uncompressed audio usually is the same (CD-quality). However, it should be remembered that even two audio files in the same file format and with the same bit rate may be of very different quality because the encoding programs me be of different quality.

All the audio formats mentioned so far can be considered raw formats; it is a description of how the sound is digitised. When the information is stored on a computer, the details of how the data is organised must be specified, and there are several popular formats for this.

6.4.2 Lossless formats

The two most common file formats for CD-quality audio are AIFF and WAV, which are both supported by most commercial audio programs. These formats specify in detail how the audio data should be stored in a file. In addition, there is support for including the title of the audio piece, album and artist name and other relevant data. All the other audio formats below (including the lossy ones) also have support for this kind of additional information.

AIFF. *Audio Interchange File Format* was developed by Apple and published in 1988. AIFF supports different sample rates and bit lengths, but is most commonly used for storing CD-quality audio at 44 100 samples per second and 16 bits per sample. No compression is applied to the data, but there is also a variant that supports lossless compression, namely AIFF-C.

WAV. *Waveform audio data* is a file format developed by Microsoft and IBM. As AIFF, it supports different data formats, but by far the most common is standard CD-quality sound. WAV uses a 32-bit integer to specify the file size at the beginning of the file which means that a WAV-file cannot be larger than 4 GB. Microsoft therefore developed the W64 format to remedy this.

Apple Lossless. After Apple's iPods became popular, the company in 2004 introduced a lossless compressed file format called Apple Lossless. This format is used for reducing the size of CD-quality audio files. Apple has not published the algorithm behind the Apple Lossless format, but most of the details have been

worked out by programmers working on a public decoder. The compression phase uses a two step algorithm:

1. When the nth sample value x_n is reached, an approximation y_n to x_n is computed, and the error $e_n = x_n - y_n$ is stored instead of x_n. In the simplest case, the approximation y_n would be the previous sample value x_{n-1}; better approximations are obtained by computing y_n as a combination of several of the previous sample values.

2. The error e_n is coded by a variant of the *Rice algorithm*. This is an algorithm which was developed to code integer numbers efficiently. It works particularly well when small numbers are much more likely than larger numbers and in this situation it achieves compression rates close to the entropy limit. Since the sample values are integers, the step above produces exactly the kind of data that the Rice algorithm handles well.

FLAC. *Free Lossless Audio Code* is another compressed lossless audio format. FLAC is free and open source (meaning that you can obtain the program code). The encoder uses an algorithm similar to the one used for Apple Lossless, with prediction based on previous samples and encoding of the error with a variant of the Rice algorithm.

6.4.3 Lossy formats

All the lossy audio formats described below apply a modified version of the DCT to successive groups (frames) of sample values, analyse the resulting values, and perturb them according to a psycho-acoustic model. These perturbed values are then converted to a suitable number format and coded with some lossless coding method like Huffman coding. When the audio is to be played back, this process has to be reversed and the data translated back to perturbed sample values at the appropriate sample rate.

MP3. Perhaps the best known audio format is MP3 or more precisely *MPEG-1 Audio Layer 3*. This format was developed by Philips, CCETT (Centre commun d'études de télévision et télécommunications), IRT (Institut für Rundfunktechnik) and Fraunhofer Society, and became an international standard in 1991. Virtually all audio software and music players support this format. MP3 is just a sound format and does not specify the details of how the encoding should be done. As a consequence there are many different MP3 encoders available, of varying quality. In particular, an encoder which works well for higher bit rates (high quality sound) may not work so well for lower bit rates.

MP3 is based on applying a variant of the DCT (called the Modified Discrete Cosine Transform, MDCT) to groups of 576 (in special circumstances 192) samples. These MDCT values are then processed according to a psycho-acoustic model and coded efficiently with Huffman coding.

MP3 supports bit rates from 32 to 320 kbit/s and the sampling rates 32, 44.1, and 48 kHz. The format also supports variable bit rates (the bit rate varies in different parts of the file).

AAC. *Advanced Audio Coding* has been presented as the successor to the MP3 format by the principal MP3 developer, Fraunhofer Society. AAC can achieve better quality than MP3 at the same bit rate, particularly for bit rates below 192 kb/s. AAC became well known in April 2003 when Apple introduced this format (at 128 kb/s) as the standard format for their iTunes Music Store and iPod music players. AAC is also supported by many other music players, including the most popular mobile phones.

The technologies behind AAC and MP3 are very similar. AAC supports more sample rates (from 8 kHz to 96 kHz) and up to 48 channels. AAC uses the MDCT, just like MP3, but AAC processes 1 024 samples at time. AAC also uses much more sophisticated processing of frequencies above 16 kHz and has a number of other enhancements over MP3. AAC, as MP3, uses Huffman coding for efficient coding of the MDCT values. Tests seem quite conclusive that AAC is better than MP3 for low bit rates (typically below 192 kb/s), but for higher rates it is not so easy to differentiate between the two formats. As for MP3 (and the other formats mentioned here), the quality of an AAC file depends crucially on the quality of the encoding program.

There are a number of variants of AAC, in particular AAC Low Delay (AAC-LD). This format was designed for use in two-way communication over a network, for example the Internet. For this kind of application, the encoding (and decoding) must be fast to avoid delays (a delay of at most 20 ms can be tolerated).

Ogg Vorbis. *Vorbis* is an open-source, lossy audio format that was designed to be free of any patent issues and free to use, and to be an improvement on MP3. At our level of detail Vorbis is very similar MP3 and AAC: It uses the MDCT to transform groups of samples to the frequency domain, it then applies a psycho-acoustic model, and codes the final data with a variant of Huffman coding. In contrast to MP3 and AAC, Vorbis always uses variable length bit rates. The desired quality is indicated with an integer in the range -1 (worst) to 10 (best). Vorbis supports a wide range of sample rates from 8 kHz to 192 kHz and up to

255 channels. In comparison tests with the other formats, Vorbis appear to perform well, particularly at medium quality bit rates.

WMA. *Windows Media Audio* is a lossy audio format developed by Microsoft. WMA is also based on the MDCT and Huffman coding, and like AAC and Vorbis, it was explicitly designed to improve the deficiencies in MP3. WMA supports sample rates up to 48 kHz and two channels. There is a more advanced version, WMA Professional, which supports sample rates up to 96 kHz and 24 bit samples, but this has limited support in popular software and music players. There is also a lossless variant, WMA Lossless. At low bit rates, WMA generally appears to give better quality than MP3. At higher bit rates, the quality of WMA Pro seems to be comparable to that of AAC and Vorbis.