Data Communications Fundamentals: Analog and Digital Signals; Compression

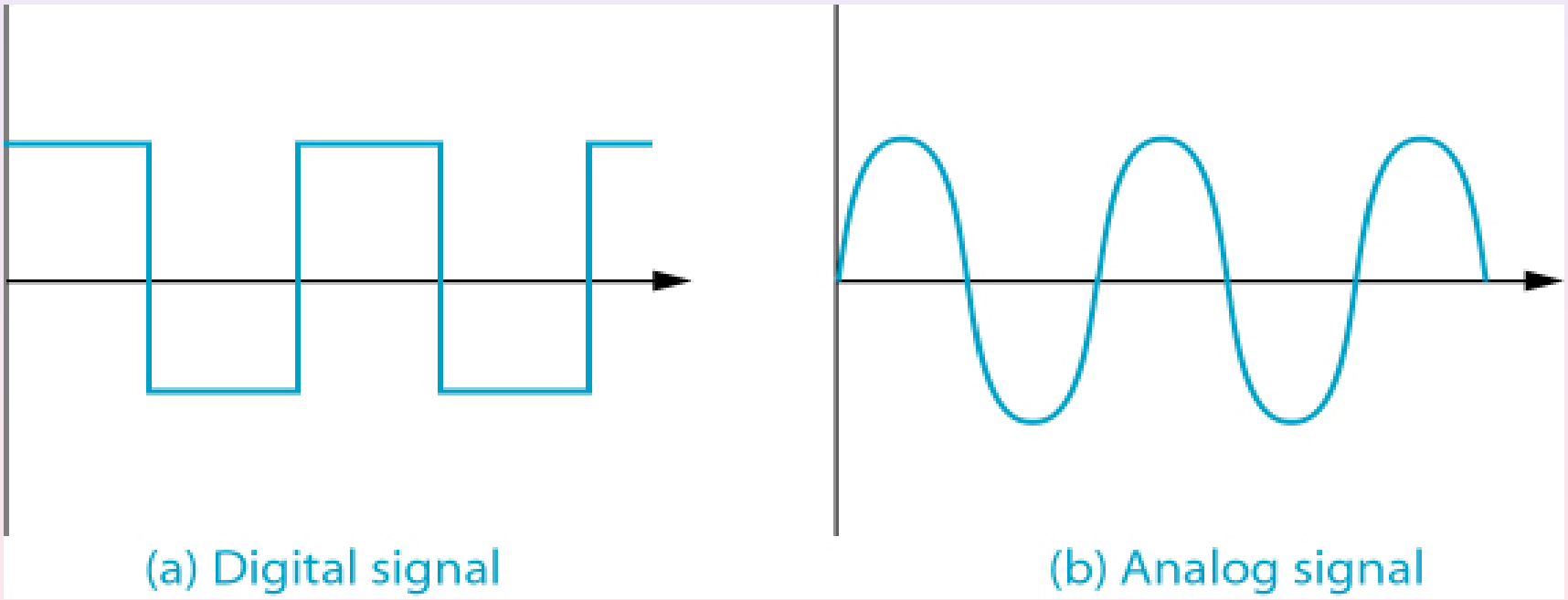
Cristian S. Calude Clark Thomborson

July 2010

Questions

1. How does symbolic data relate to electrical signals, microwaves, and light waves?
2. What does a 0 or a 1 actually look like as it travels through a wire, optical fibre, or space?
3. How many bits can a signal transmit per unit of time?
4. What effect does electrical interference (noise) have on data transmission?

Analog vs. digital signals



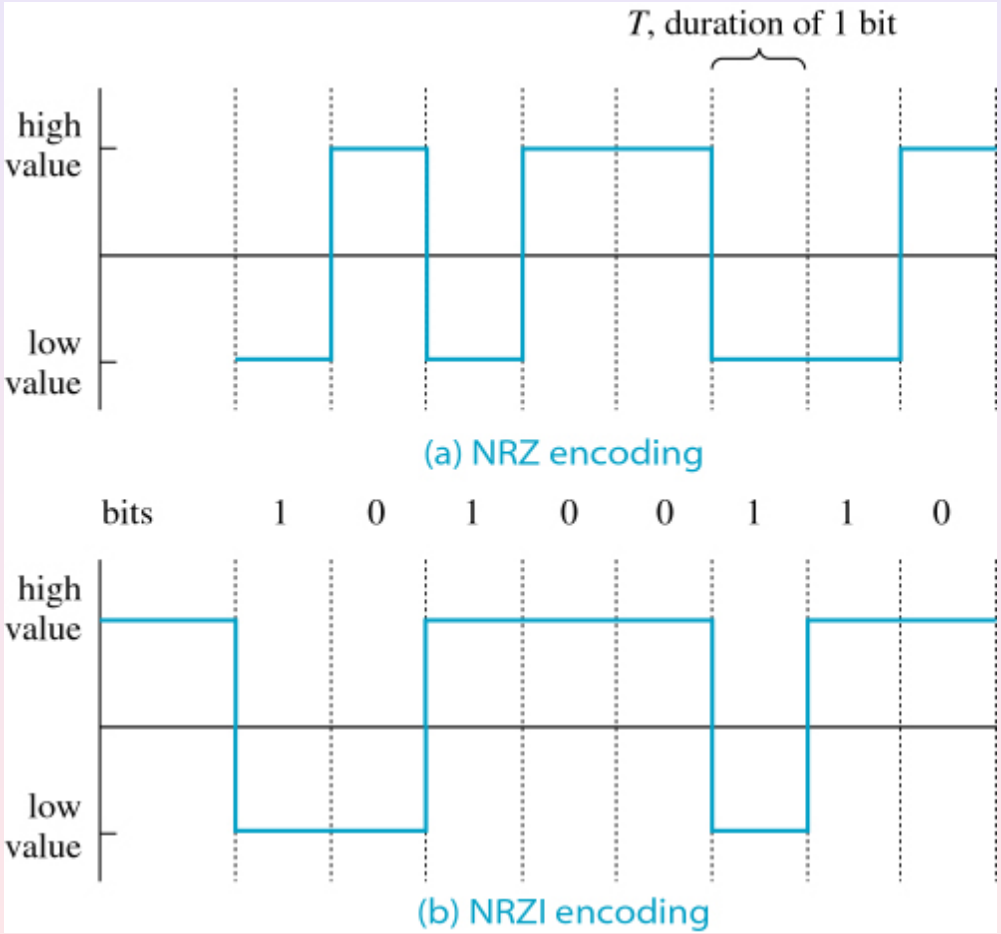
Encoding Bits in Analog Signals 1

Because digital signals can alternate between two constant values—say "high voltage" and "low voltage"—we simply associate 0 with one value and 1 with the other. The actual values are irrelevant.

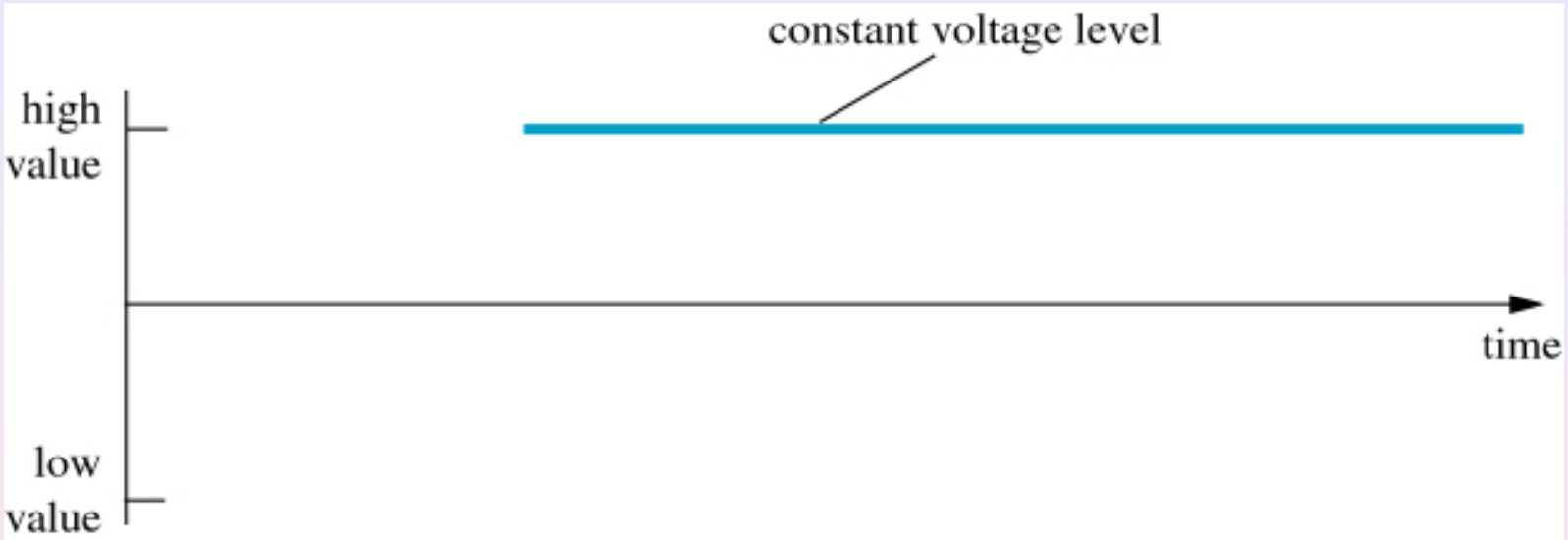
Non-Return to Zero (NRZ): a 0 is transmitted by raising the voltage level to high, and a 1 is transmitted using a low voltage. Alternating between high and low voltage allows for the transmission of any string of 0s and 1s.

Non-Return-to-Zero-Inverted Encoding (NRZI): a 0 is encoded as no change in the level. However a 1 is encoded depending on the current state of the line. If the current state is 0 [low] the 1 will be encoded as a high, if the current state is 1 [high] the 1 will be encoded as a low. Used in USB.

Example: NRZ and NRZI



Synchronisation Problem



What is being transmitted in NRZ? In NRZI? A string of 0s, in either case. But... how many?

NRZ and NRZI require a *clock signal* as well as a *data signal*. Sending the clock signal requires an additional connection with the same latency as the data signal, otherwise the clock will be *skewed* and the data will be decoded incorrectly.

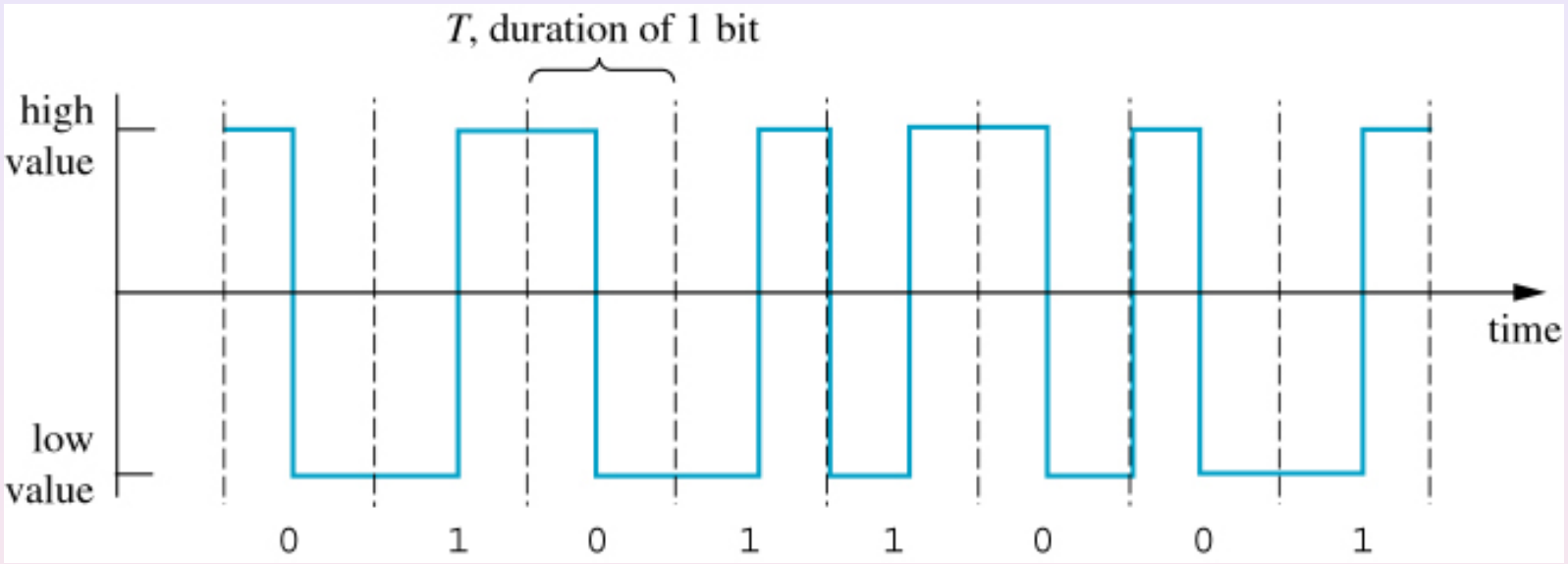
Self-synchronising Encoding 1

The **Manchester code** uses signal changes to keep the sending/receiving devices synchronised. It encodes 0 and 1 by changing the voltage:

0 is represented by a change from high to low and
1 is represented by a change from low to high.

Note: the signal will never be held constant longer than a single bit interval, no matter what data is being transmitted.

Analog signals 5



A Manchester signal can change levels twice every T seconds, and must change at least once. An NRZ or NRZI signal can change level at most once every T seconds: this is half the bandwidth of a Manchester signal. (Are there more efficient encodings of a clock signal?)

Modulation and Demodulation

The process of adding a data signal (e.g. a digital bitstream from a PC) to analog *carrier signal* is called **modulation**. The process of extracting the data from a modulated signal is called **demodulation**.

Frequency modulation is used in FM radio transmission. The analog audio signal (typically limited to 15 kHz) modulates the analog carrier signal (e.g. 101.4 MHz for National FM in the Auckland region).

A **modem**, short for modulator/demodulator, is a device that does both conversions – for digital data and an analog carrier.

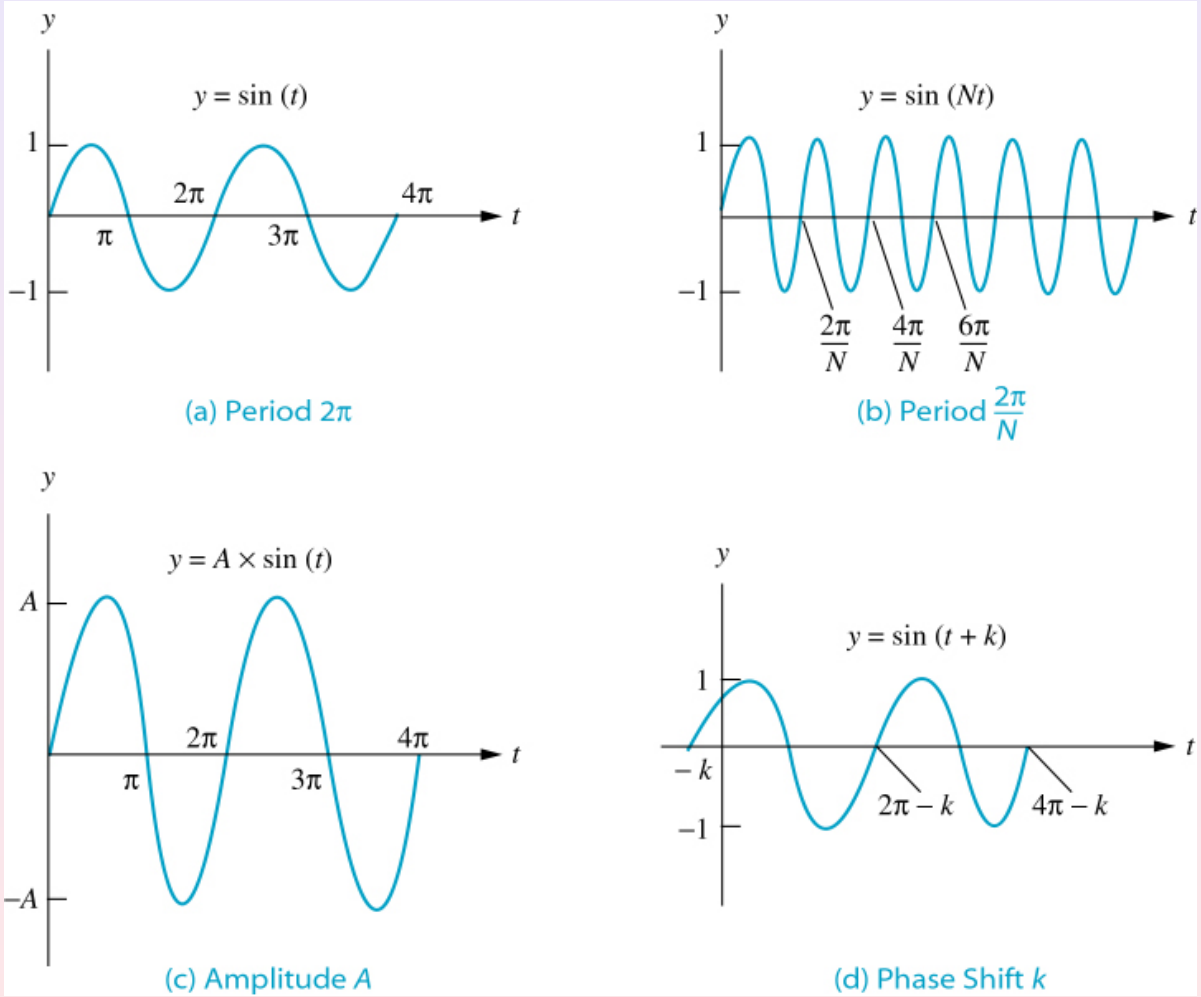
Understanding analog signals 1

A sine wave is the simplest analog signal. There are three ways to adjust a sine wave:

1. changing its frequency,
2. changing its amplitude,
3. changing its phase.

See `http://www.ltscotland.org.uk/5to14/resources/science/sound/index.asp`.

Understanding analog signals 2



Understanding analog signals 3

1. the **period**, p, is the time it takes to complete a pattern,
2. the **frequency**, f, is the number of times the signal oscillates per unit of time ((hertz) Hz if time is measured in seconds); $f = 1/p$,
3. the **amplitude** is the range in which the signal oscillates; the **peak amplitude** is the absolute value of signal's highest intensity),
4. the **phase** shift describes the position of the signal relative to time zero; it is obtained by adding or subtracting k from the sine argument ($\sin(t + k)$, $\sin(t - k)$).

Understanding analog signals 4

Here are some numerical examples with reference to the previous picture:

In (a) the period for $y = sin(t)$ is $p = 2\pi$,

In (b) the period for $y = sin(Nt)$ is $p = 2\pi/N$,

In (b) the frequency is $f = 1/p = N/2\pi$ Hz,

In (a), (b), and (d) the amplitude is $[-1, 1]$; and the peak amplitude is 1. In (c), the amplitude is $[-A, A]$ with a peak of A.

Understanding analog signals 5

A sine function can be written in the form:

$$s(t) = A\sin(2\pi ft + k),$$

where $s(t)$ is the instantaneous amplitude at time t, A is the peak amplitude, f is the frequency, and k is the phase shift.

These three characteristics fully describe a sine function.

Understanding analog signals 6

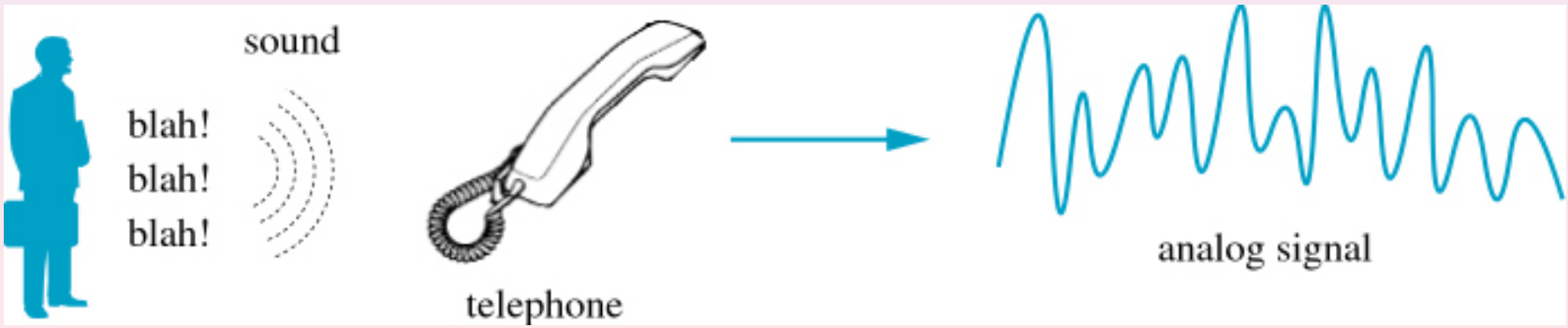
For electrical signals, the peak amplitude is measured in volts (V). The frequency is measured in hertz (Hz).

Unit	Equivalent	Unit	Equivalent
second (s)	1 s	hertz (Hz)	1Hz
millisecond (ms)	10^{-3} s	kilohertz (kHz)	10^{3} Hz
microsecond (μs)	10^{-6} s	megahertz (MHz)	10^{6} Hz
nanosecond (ns)	10^{-9} s	gigahertz (GHz)	10^{9} Hz
picosecond (ps)	10^{-12} s	terahertz (THz)	10^{12} Hz

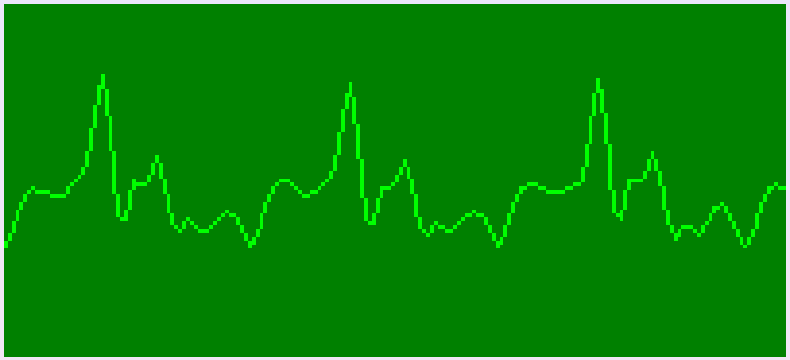
Understanding analog signals 7

Sound can be *transduced* by a microphone, or by our ears, into an electrical (analog) signal. The audio signal from a microphone has the following relationships to our aural perceptions:

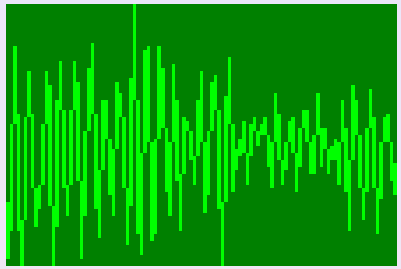
- the amplitude of the audio signal correlates with our perception of volume, and
- the frequency correlates with our perception of pitch.



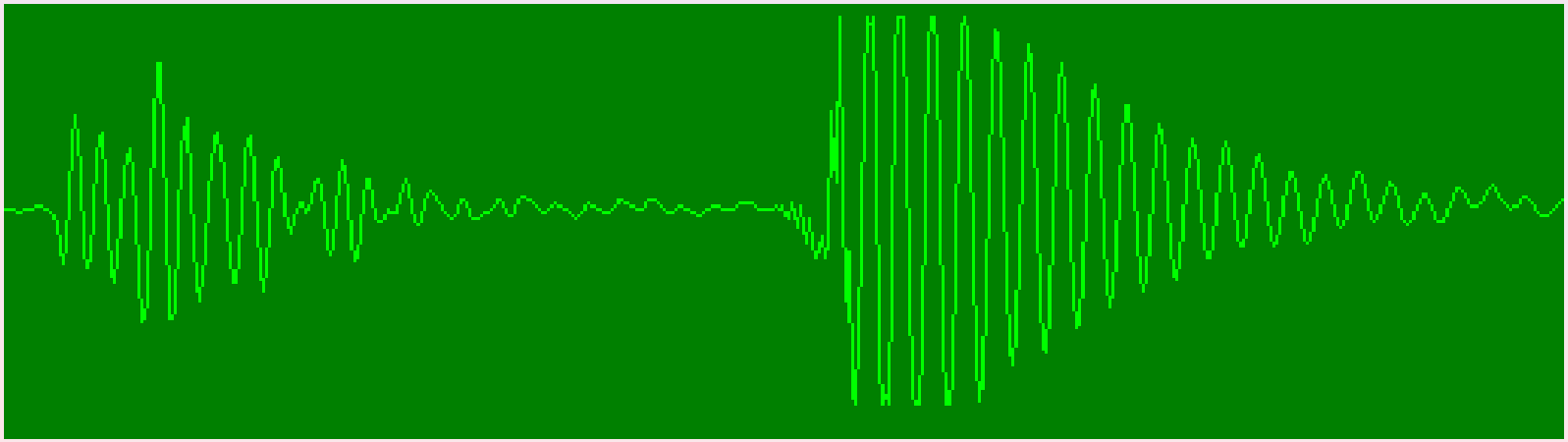
Understanding analog signals 8



Trumpet



White noise



Tongue click

Understanding analog signals 9

A single sine function is not useful for data communications.

We need to change one or more of its characteristics—amplitude, frequency and phase shift—to encode a signal.

How do we decode a signal from a sine wave carrier that has been modulated by shifts in amplitude, frequency and/or phase?

Answer: *Fourier Analysis*. The French mathematician Jean Baptiste Fourier proved that any periodic function can be expressed as an infinite sum of sine and cosine functions of varying amplitudes, frequencies and phase shifts—a **Fourier series**.

Understanding analog signals 10

Any periodic signal $x(t)$ with period P can be expressed as a Fourier series:

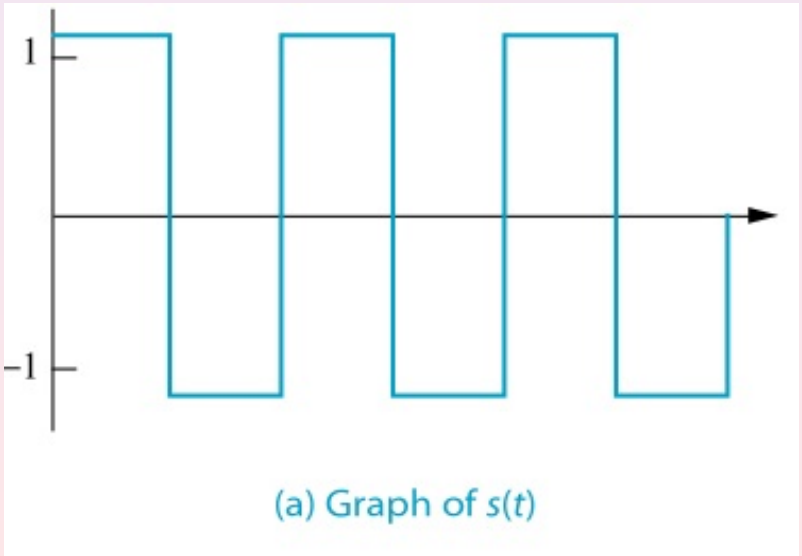
$$x(t) = \frac{a_0}{2} + \sum_{i=1}^{\infty} \left[a_i \cos \frac{2\pi it}{P} + b_i \sin \frac{2\pi it}{P} \right]$$

The coefficients $(\vec{a}, \vec{b})$ are uniquely determined by this equation, and are called the *Fourier transform* of $x(t)$.

Understanding analog signals 11

For example, consider a unit-amplitude square wave $s(t)$ with period 2π:

$$s(t) = \begin{cases} 1, & \text{if } t \in [0, \pi) \cup [2\pi, 3\pi) \cup [4\pi, 5\pi) \cup \ldots, \\ -1, & \text{if } t \in [\pi, 2\pi) \cup [3\pi, 4\pi) \cup [5\pi, 6\pi) \cup \ldots, \end{cases}$$



(a) Graph of $s(t)$

Understanding analog signals 12

Because the square wave $s(t)$ is periodic, it can be written as a Fourier series:

$$s(t) = \frac{a_0}{2} + \sum_{i=1}^{\infty} \left[a_i \cos \frac{2\pi i t}{P} + b_i \sin \frac{2\pi i t}{P} \right]$$

The values of P and the coefficients a_i and b_i are not complicated expressions (but would be hard to guess ;-):

$$P = 2\pi, \quad a_i = 0, \quad b_i = \begin{cases} 0, & \text{if } i \text{ is even} \\ \frac{4}{\pi i}, & \text{if } i \text{ is odd} \end{cases}$$

So a square wave is

$$s(t) = \sum_{i=1,3,5,\ldots} \frac{4}{\pi i} \sin(it).$$

Understanding analog signals 13

Let's look at the Fourier analysis of the square wave more carefully:

$$s(t) = \sum_{i=1,3,5,\ldots} \frac{4}{\pi i} \sin(it) = \frac{4}{\pi} \sin(t) + \frac{4}{3\pi} \sin(3t) + \cdots$$

Each term in this series can be expressed in our general form for sine waves:

$$\frac{4}{i\pi} \sin(it) = A \sin(2\pi f t i + k),$$

where the amplitude, frequency and phase shift are

$$A = \frac{4}{i\pi}, \quad f = \frac{1}{2\pi}, \quad k = 0.$$

Understanding analog signals 14

A square wave is thus a sum of sine functions with frequencies $f, 3f, 5f, \ldots$ and amplitudes $\frac{4}{\pi}, \frac{4}{3\pi}, \frac{4}{5\pi}, \ldots$.

$$s(t) = \frac{4}{\pi}\sin(2\pi ft) + \frac{4}{3\pi}\sin[2\pi(3f)t] + \frac{4}{5\pi}\sin[2\pi(5f)t] + \cdots$$

The term with frequency f is called the **fundamental frequency**; the term with frequency $3f$ is called the **third harmonic**; the term with frequency $5f$ is called the **fifth harmonic**; etc. These are all **odd harmonics**.

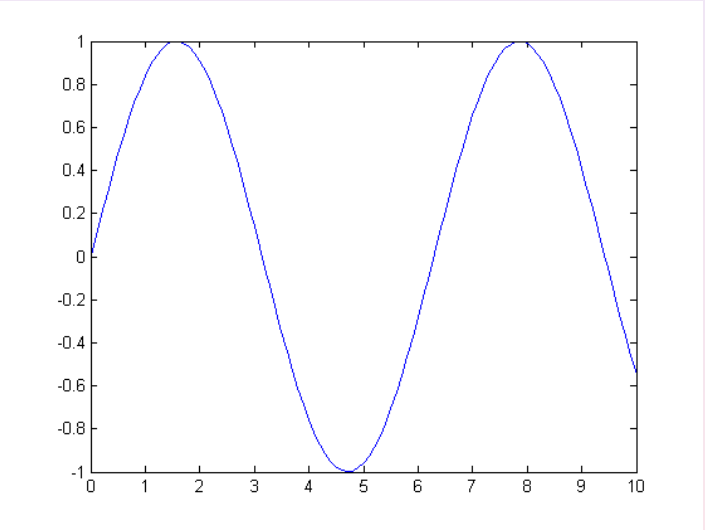
We can compute a finite number n of Fourier coefficients efficiently, in $O(n \log n)$ floating-point multiplications and additions, using an algorithm called the *fast Fourier transform*.

Understanding analog signals 15

The Fourier series expansion for a square-wave is made up of a sum of odd harmonics. We show this graphically using MATLAB®.

We start by forming a time vector running from 0 to 10 in steps of 0.1, and take the sine of all the points. Let's plot this fundamental frequency.

```
t = 0:.1:10;
y = sin(t);
plot(t,y);
```

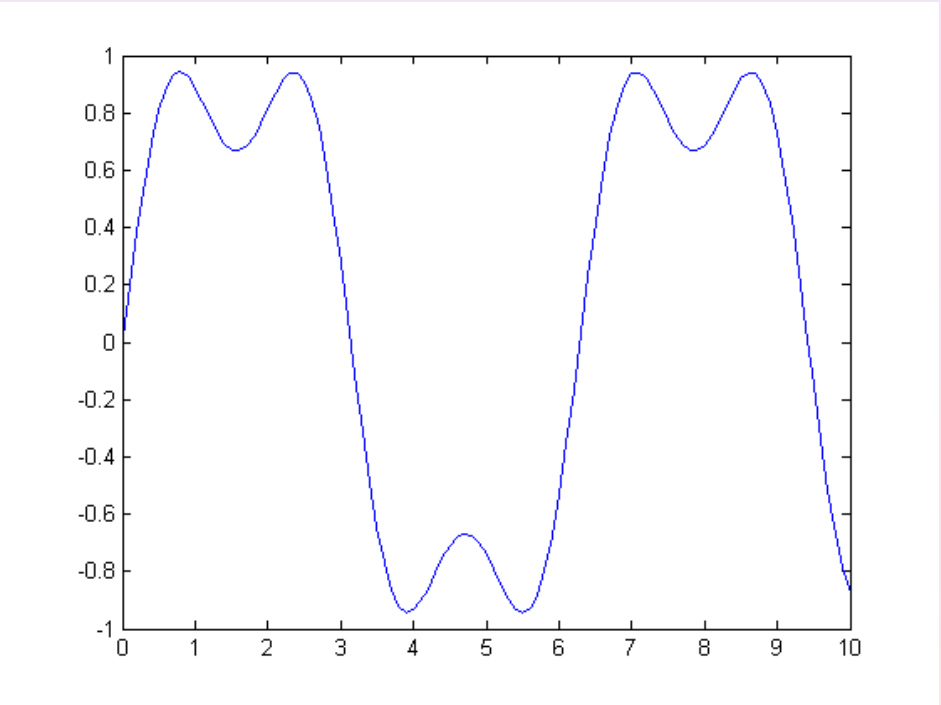


Text and graphics on this series of slides is from `http://www.mathworks.com/products/matlab/demos.html?file=/products/demos/shipping/matlab/xfourier.html`

Understanding analog signals 16

Now add the third harmonic to the fundamental, and plot it.

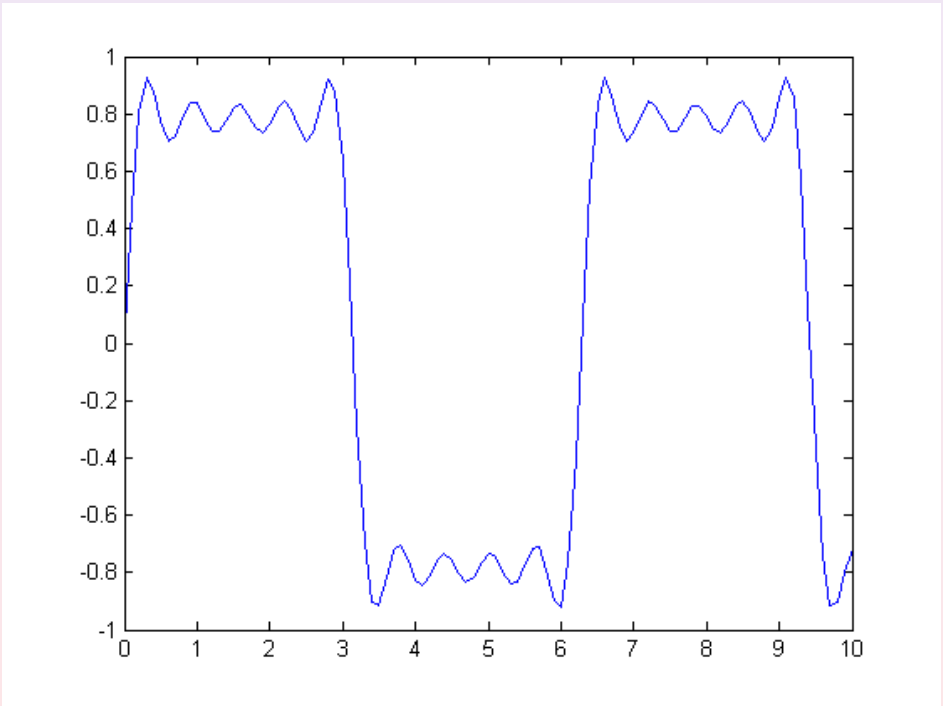
```
y = sin(t) + sin(3*t)/3; plot(t,y);
```



Understanding analog signals 17

Now use the first, third, fifth, seventh, and ninth harmonics.

```
y = sin(t) + sin(3*t)/3 + sin(5*t)/5 +
sin(7*t)/7 + sin(9*t)/9; plot(t,y);
```

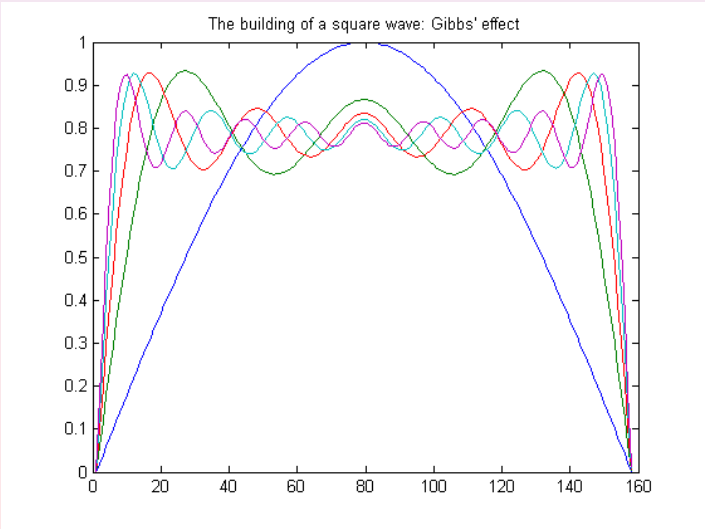


Understanding analog signals 18

For a finale, we will go from the fundamental to the 19th harmonic, creating vectors of successively more harmonics, and saving all intermediate steps as the rows of a matrix.

These vectors are plotted on the same figure to show the evolution of the square wave. Note that Gibbs' effect says that it will never really get there.

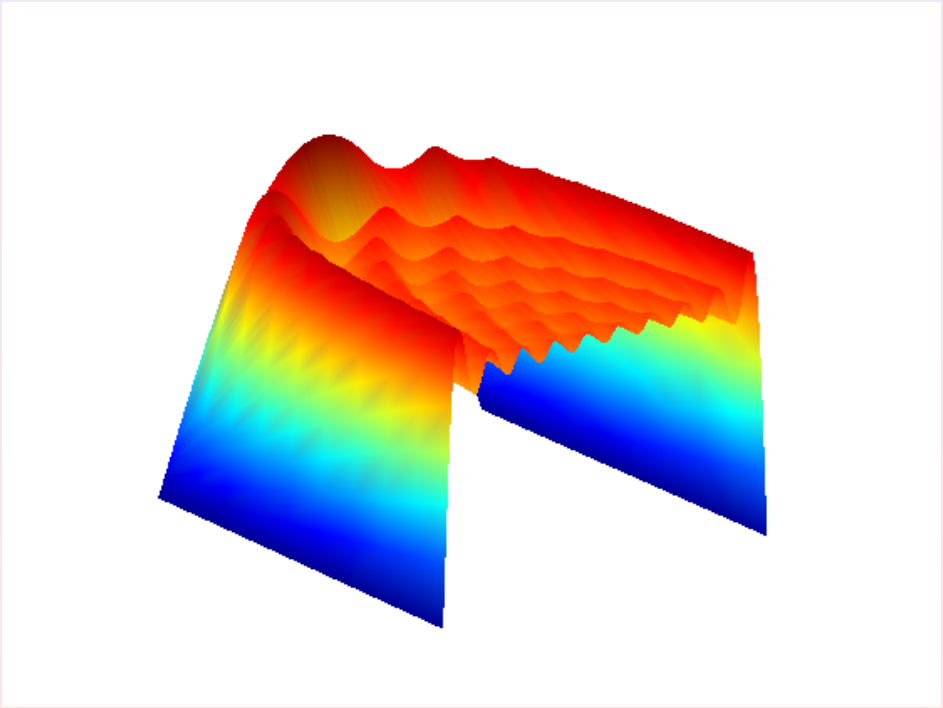
```
t = 0:.02:3.14; y =
zeros(10,length(t));
x = zeros(size(t));
for k=1:2:19
    x = x + sin(k*t)/k;
    y((k+1)/2,:)  = x;
end
plot(y(1:2:9,:)')
title('The building of a
square wave:  Gibbs'' effect')
```



Understanding analog signals 19

Here is a 3-D surface representing the gradual transformation of a sine wave into a square wave.

```
surf(y); shading interp axis off ij
```



Understanding analog signals 20

The fast Fourier transform can be used in digitised versions of analog signal-processing devices.

- **Filters** attenuate certain frequencies while allowing others to pass. The Bass control on a stereo system is an adjustable *lowpass filter* which limits the amount of low-frequency sound in the output. The Treble control is an adjustable *highpass filter*. Stereo equalisers have many adjustable *bandpass filters*.
- **Tuners** extract one modulated signal from a signal which has been modulated by many signals. Tuners are necessary in radio and TV receivers, to select one station or channel from all of the ones that are currently being broadcast.

Bit rate 1

The **bit rate** describes the information-carrying capacity of a digital channel, and is measured in bits per second (b/s).

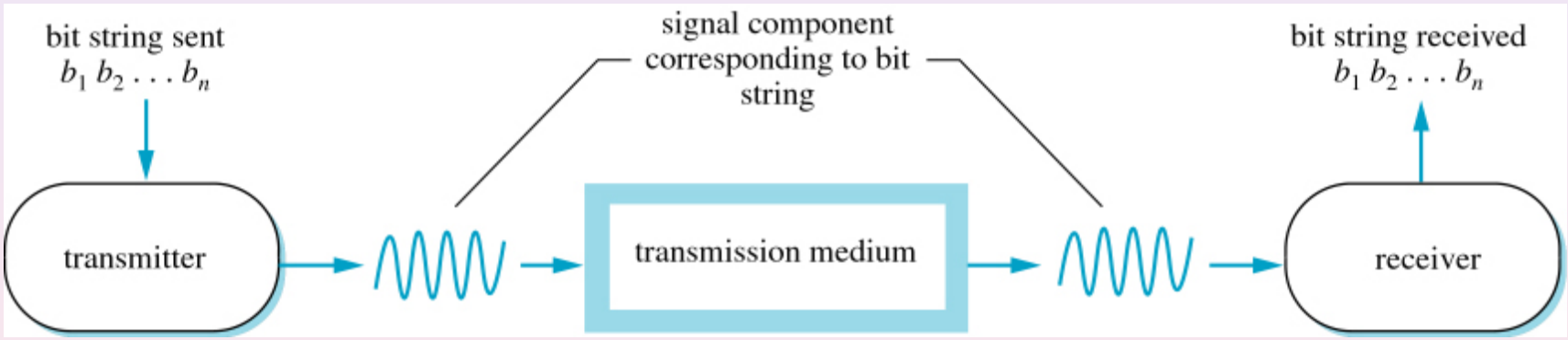
The range of frequencies in a channel is called its *bandwidth*.

Roughly:

a higher-bandwidth channel has a higher bit rate.

(We will develop a more refined understanding, in a moment...)

Bit rate 2



Sending data via signals

Bit rate 3

The bit rate $R = f_s n$ is the product of the frequency f_s at which symbols are sent, and the number of bits per symbol n.

Note that this equation is trivial, by a dimensional analysis:

$$\frac{\text{bits}}{\text{second}} = \frac{\text{bits} \cdot \text{symbol}}{\text{second} \cdot \text{symbol}} \quad (1)$$

$$= \frac{\text{bits}}{\text{symbol}} \cdot \frac{\text{symbols}}{\text{second}} \quad (2)$$

As a shorthand for symbols/second, we write *baud* – after Émile Baudot, the inventor of the Baudot code. The abbreviation is Bd, and its units are symbols per second.

Bit rate 4

Nyquist theorem. *In a distortion-free transmission, the baud rate f_s is at most twice the maximum frequency f of the medium.*

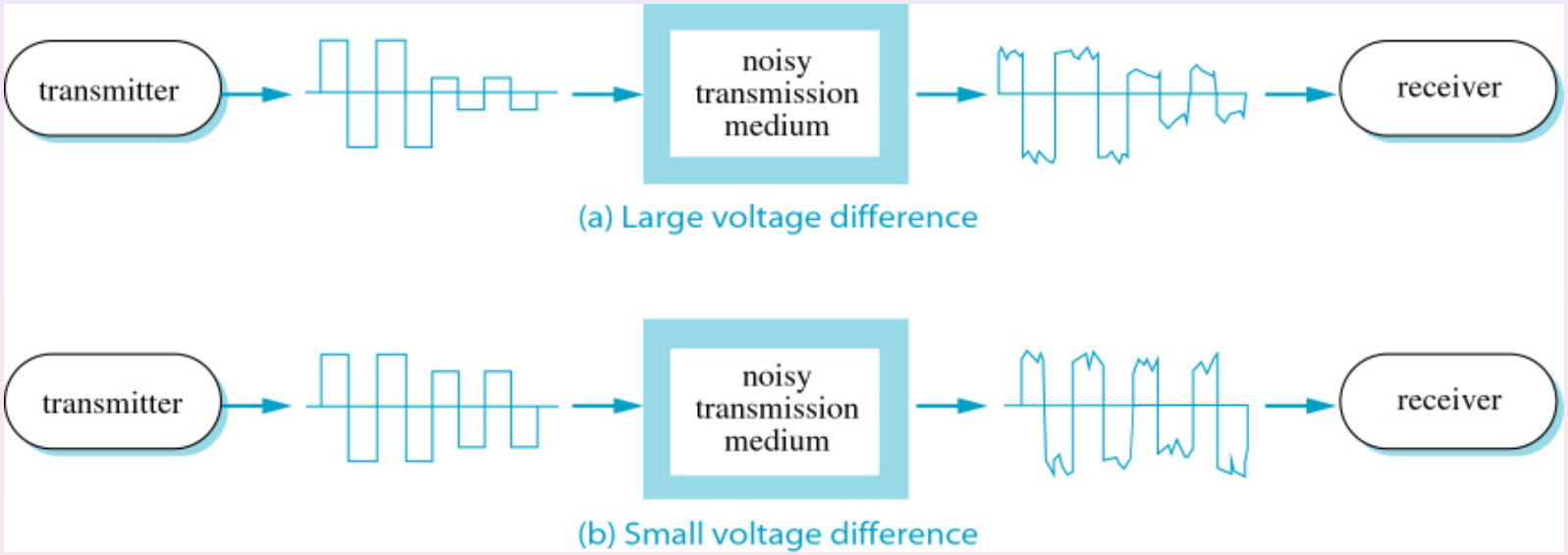
Since the baud rate f_s is upper-bounded by $2f$, the bit rate R is at most $2fn$ when each symbol carries n bits.

Telephone connections have a maximum frequency f of 3300 Hz.

bits/symbol (n)	number of symbols (2^n)	max bits/second (R)
1	2	6,600
2	4	13,200
3	8	19,800
4	16	26,400

Noisy channels 1

Many channels are **noisy**. Note the voltage difference between the first and last part of the transmitted signal



Sending data via signals

Noisy channels 2

The **signal-to-noise (S/N) ratio or SNR** is measured by the ratio S/N, where S is the signal power and N is the noise power. A clear signal has a large SNR, and a distorted (noisy) signal has a low SNR. Since the measured values of S/N vary hugely, we typically report their logarithms rather than their absolute values:

$$\mathrm{SNR}_{\mathrm{dB}} = \log_{10}(S/N)\,\mathrm{bels}$$

where the unit of measurement is called the **bel**. The more familiar **decibel** is defined by

$$1\,\mathrm{dB} = 0.1\,\mathrm{bel}.$$

Decibels can also be used to measure sound intensity, relative to some baseline level (N). Typically, a 3 dB change is barely perceptible. See `http://en.wikipedia.org/wiki/Weber%E2%80%93Fechner_law`.

Noisy channels 3

If our SNR is reported as 25 dB, this is 2.5 bels, i.e. we have

$$\mathrm{SNR}_{\mathrm{dB}} = \log_{10}(S/N) = 2.5\,\mathrm{bels}$$

This implies

$$S/N = 10^{2.5}$$

or

$$S = 10^{2.5} \times N = 100\sqrt{10} \times N \approx 316N$$

Noisy channels 4

The American scientist Claude Shannon, inventor of the classical theory of information, refined Nyquist's theorem by taking into account the channel's noise:

Shannon's theorem. *In a noisy transmission,*

$$\text{bit rate (in b/s)} \leq \text{bandwidth (in Hz)} \times \log_2(1 + S/N)$$

The quantity $\log_2(1 + S/N)$ is the maximum number of bits that can be transmitted per cycle on this channel.

Note that decreasing the signal-to-noise ratio on any channel will decrease its information-carrying capacity. Somewhat surprisingly, a channel with any non-zero signal strength has some capacity (because its SNR must be greater than zero).

Noisy channels 5

Telephones carry audio frequencies in the range 300 Hz to 3300 Hz: this is a bandwidth of 3000 Hz. A good telephone connection has a signal-to-noise ratio of 35 dB. So,

$$3.5 \text{ bels} = \log_{10}(S/N) \text{ bels},$$

$$S = 10^{3.5} \times N \approx 3162N.$$

Using Shannon's theorem we get:

$$\begin{aligned} \text{bit rate} &= \text{bandwidth} \times \log_2(1 + S/N) \\ &= 3000 \times \log_2(1 + 3162) \text{ b/s} \\ &\approx 3000 \times 11.63 \text{ b/s} \\ &\approx 34880 \text{ b/s} \end{aligned}$$

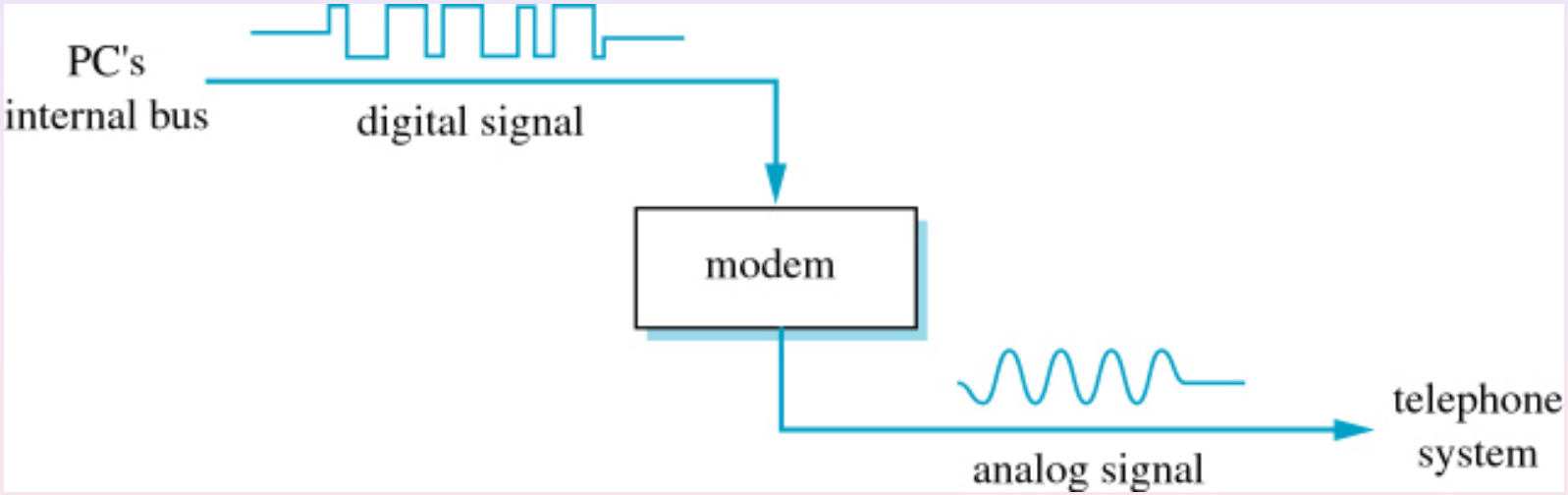
Noisy channels 6

How can a 56 kb/s modem possibly achieve its maximum data rate on a 3000 Hz analog phone line with a S/N of 35 dB? (Did Shannon make a mistake?)

Answer: Modern telephone systems use 64 kb/s digital signalling on their long-distance connections, and the downlink on a V.90 modem is able to interpret these digital signals when it receives data from your ISP. Your downlink is a channel with an 8 kBd signalling rate and 8 bits/symbol; the bandwidth of this channel is 4 kHz, not the 3 kHz of an analog voice connection. The uplink on a V.90 modem uses the 3 kHz analog voice channel, transmitting data at 33.6 kb/s if your phone line isn't noisy, and transmitting at a lower bitrate if you have a noisy line.

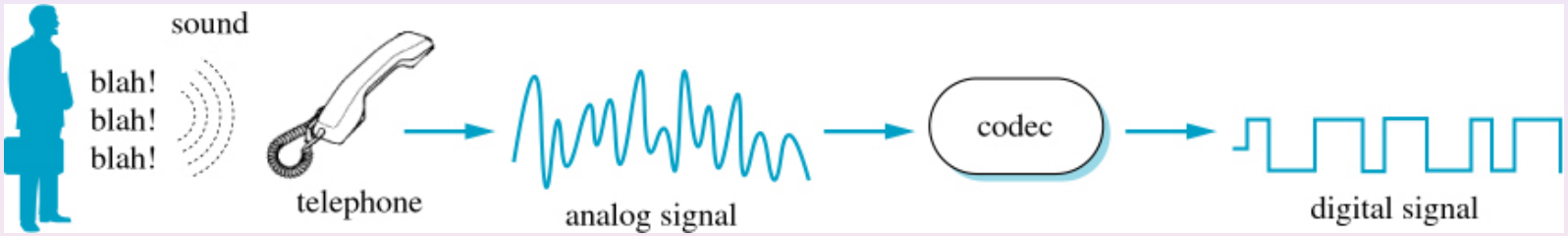
V.92 modems can upload at 56 kb/s, but as far as I know, no NZ ISP provides a V.92 dialup service.

Digital to analog conversion 1



Computer data transmitted over telephone lines

Digital to analog conversion 2



Voice information transmitted through a digital connection, using a **codec** (coder/decoder)

Digital to analog conversion 3

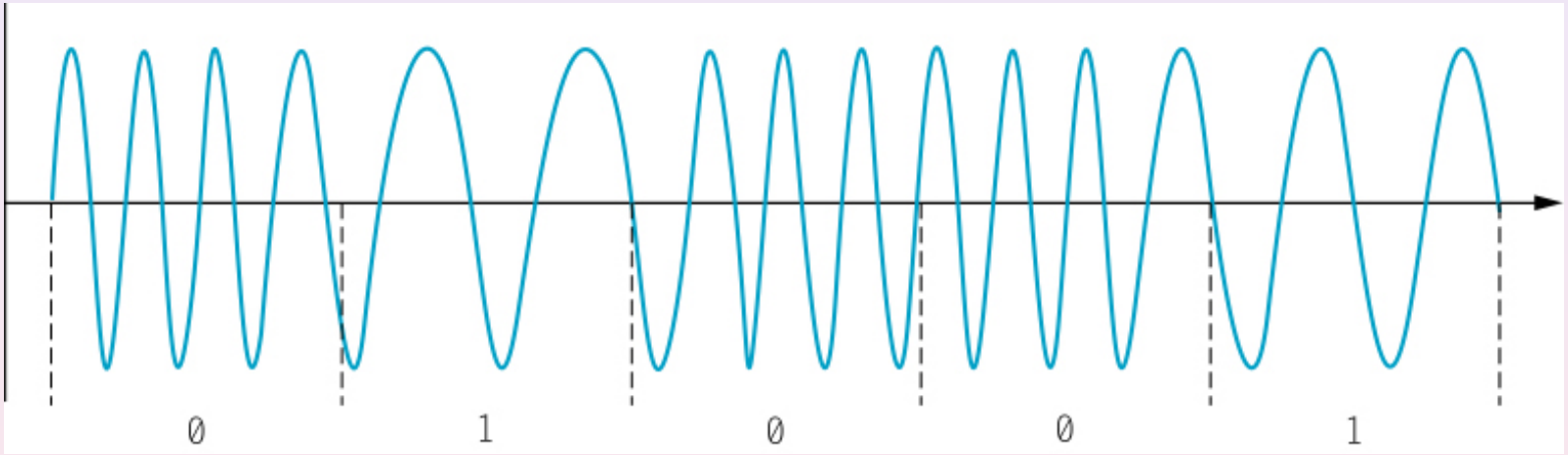
There are three main ways to encode a digital signal as an analog signal, corresponding to the three parameters in a sine wave. We can modulate

1. by frequency, for example by **frequency shift keying (FSK)**,
2. by amplitude, for example by **amplitude shift keying (ASK)**, or
3. by phase modulation, for example **phase shift keying (PSK)**.

We can also use combined methods, such as **quadrature amplitude modulation (QAM)** which modulates both amplitude and phase.

Digital to analog conversion 4

FSK or **frequency modulation (FM)** assigns a digital 0 to one analog frequency and a 1 to another.

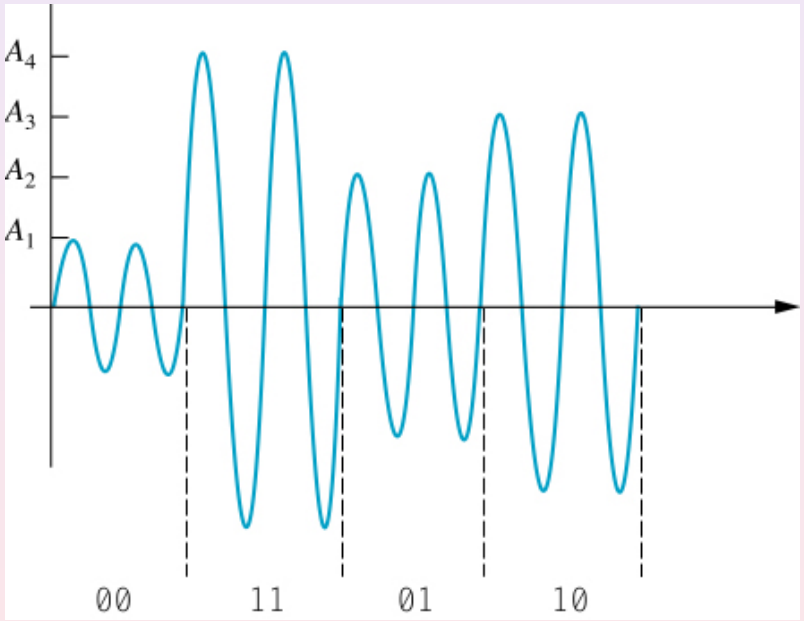


FSK, at one bit per symbol.

Note: we might use n-bit symbols, where each symbol is assigned one frequency in a set of 2^n frequencies.

Digital to analog conversion 5

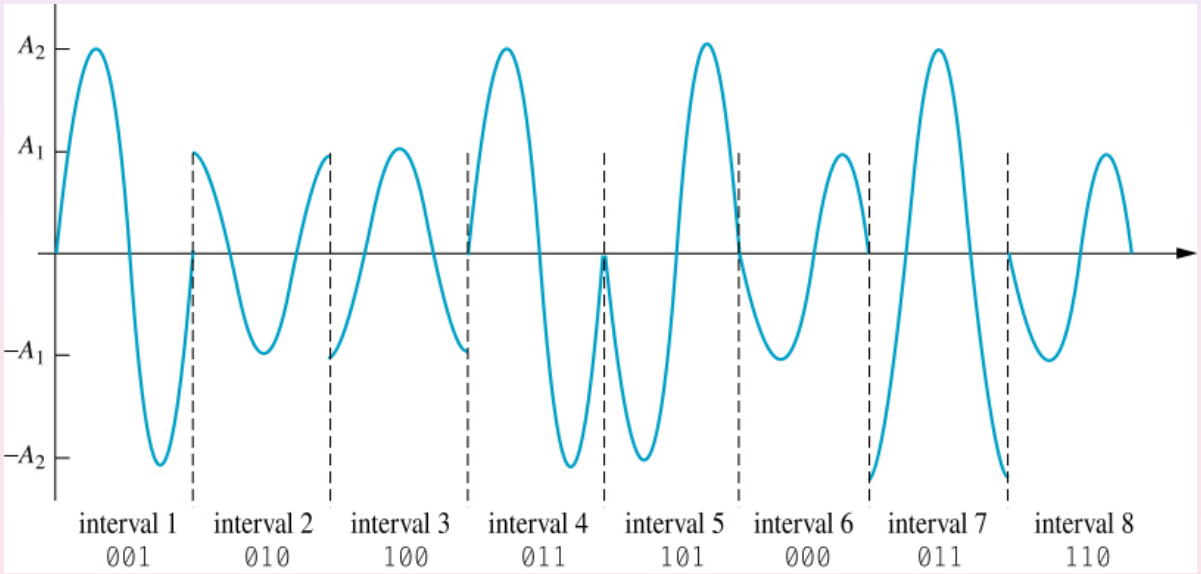
ASK or **amplitude modulation (AM)** assigns a digital '0' to one analog amplitude, a '1' to another amplitude, ... and (for AM at n bits per symbol) a '$2^n - 1$' to yet-another amplitude.



ASK, at two bits per symbol.

Digital to analog conversion 6

PSK or **phase modulation (PM)** assigns each bit-string of a fixed length to one analog phase shift. **Quadrature amplitude modulation (QAM)** is a combination of amplitude and phase shift.

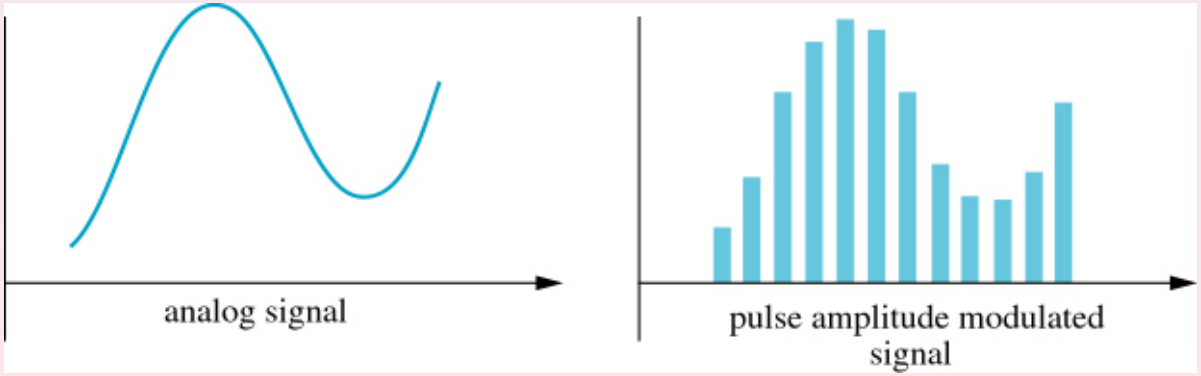


QAM with two amplitudes, four phases, and three bits per symbol.

Analog to digital conversion 1

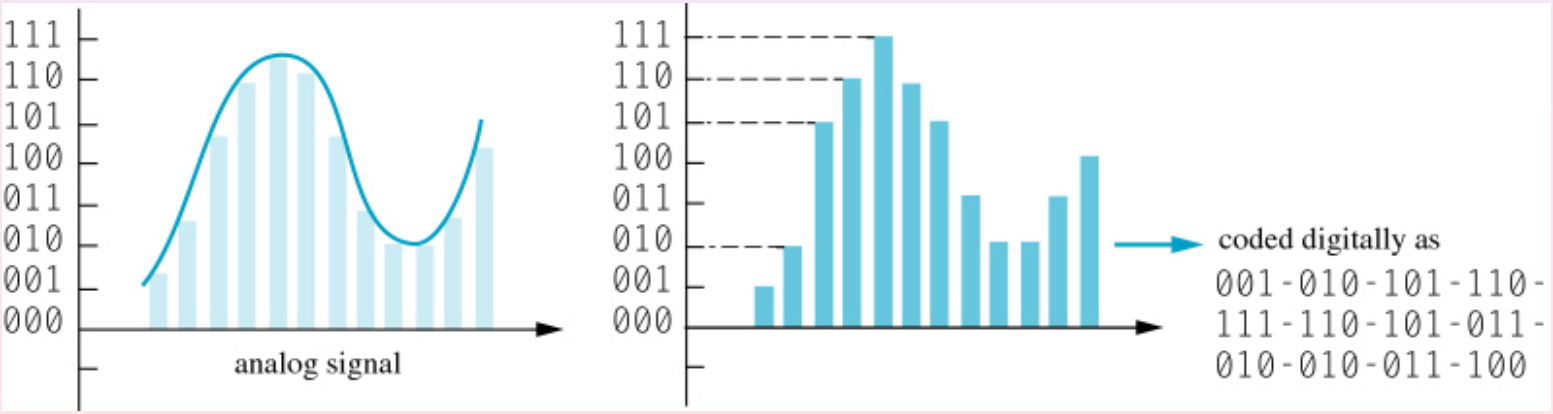
In principle, any digital to analog encoding can be decoded. However some analog-to-digital decodings are easier than others.

In the most obvious analog-to-digital decoding, we start by sampling an analog signal at regular intervals. When our samples are analog, we are modifying the analog signal by "flattening" all its high-frequency components. This modification process is called **pulse amplitude modulation (PAM)**.



Analog to digital conversion 2

PAM signals may seem digital, but they have analog amplitudes. In **pulse code modulation (PCM)**, we define a set of 2^n amplitudes, where n is the number of bits per symbol. The process of "rounding" an analog amplitude to its nearest available ("digital") approximation is called **quantisation**.



Digital telephone signalling (DS0, E0, J0) in North America, Europe, and Japan, is PCM with 8 bits per symbol at 8 kbaud.

Why compression? 1

Digital media, by using sophisticated compression algorithms, can have significant performance benefits over analog media.

Here is an example. In the (obsolescent) PAL broadcast standard for New Zealand television, the bandwidth is approximately 7 MHz. The SNR is roughly 20 dB, depending greatly on the location, design, and orientation of your antenna.

If we sample a PAL signal at the Nyquist rate of twice its bandwidth (14 MBd), and if we use the trivial digital encoding of 8 bits/sample for three channels (Red, Green, and Blue), then we will have a data rate of $3 \cdot 8 \cdot 14 = 336$ Mb/s, or 42 MB/s.

If we record all of these bytes for two hours ($= 2 \cdot 60 \cdot 60 = 7200$ seconds), we will have 302400 MB, or 302.4 GB. A digital video disk (DVD-5) can hold about 4.7 GB ...

Why compression? 2

A trivial form of data compression, for any analog signal, is to cut its bandwidth – we can use a low-pass filter to limit its high-frequency information. Analog PAL broadcasting, and VHS tapes, use this technique to compress the chrominance (colour) information in studio-quality TV recordings. The broadcast chroma is only about 2 MHz at 20 dB.

We could digitally sample the chroma signal (using PCM) at a rate of $4log_2(1 + 10^{20/2}) = 27$ Mb/s ≈ 3 MB/s. The luminance signal is about 9 MB/s, for a total of 12 MB/s.

This is more than a 3:1 compression of our naive 42 MB/s encoding. However we need to get a 40:1 compression, down to about 1 MB/s, in order to record 90 minutes of video on a DVD.

How do you reduce bits and still keep enough information?

Guiding principle of compression: Discard any information (such as high-frequency chroma) that is unimportant; retain any information that is essential to the "meaning".

Here is a simple example. Assume that you wish to email a large file consisting entirely of strings of capital letters. If the file has n characters each stored as an 8-bit ASCII code, then we need $8n$ bits.

However, we don't need all ASCII codes to code strings of capital letters: they use only 26 characters. We can make our own code with only 5-bit codewords ($2^5 = 32 > 26$), code the file using this coding scheme, send the encoded file via email, and finally decode it at the other end.

Big deal? The size of the file has decreased by $8n - 5n = 3n$, i.e. a 37.5% reduction.

Huffmann code 1

ASCII code is an 8-bit code which gives no preference in coding characters. However, some characters appear more frequently than others. A **frequency-dependent code** varies the lengths of codewords on frequency: more frequent characters have shorter codewords. An example of frequency-dependent code is the **Huffmann code code**.

Huffmann code 2

To illustrate assume that we have five characters, A-E, whose frequencies are as follows:

Letter	Frequency (%)
A	25
B	15
C	10
D	20
E	30

Huffmann code 3

To construct the Huffmann codeword for the string we follow the following algorithm:

1. To each character we associate a binary tree consisting of just one node. To each tree we assign the tree's *weight*. Initially, the weight of nodes is exactly its frequency: 0.25 for A, 0.15 for B, etc.
2. Calculate the two lightest-weight trees (choose any if there are more than two). Merge the two chosen trees into a single tree with a new root node whose left and right sub-trees are the two we chose. The weight of the new tree is the sum of the weights of the merged trees.
3. Repeat the procedure till one tree is left.

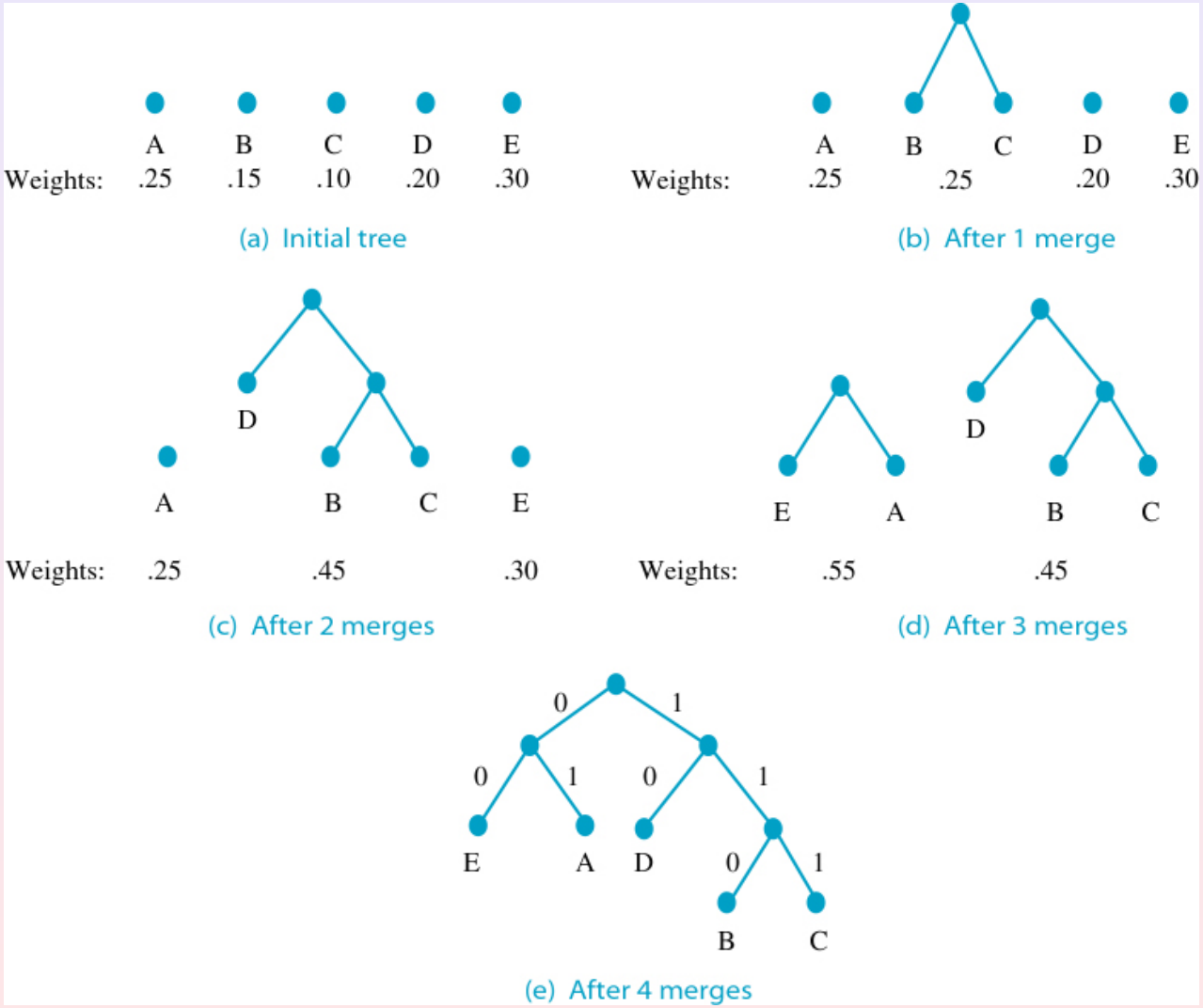
Huffmann code 4

When completed, each of the original nodes is a leaf in the final tree. Arcs are labelled with 0 and 1 as follows: we assign a 0 each time a left child pointer is followed and a 1 for each right child pointer.

As with any binary tree, there is a unique path from the root to any leaf. The path to any leaf defines the Huffmann code of the character on labelling the leaf.

Let us apply this procedure to the character strings A,B,C,D,E.

Huffmann code 5



Huffmann code 6

The Huffmann code is the following:

Letter	Frequency (%)	Code
A	25	01
B	15	110
C	10	111
D	20	10
E	30	00

Huffmann code 7

As one can see, the algorithm for constructing the Huffmann tree is very "tolerant", it gives a lot of flexibility. In the above example one could, for instance, choose putting D on the right-hand side of the merged tree in stage (c) and swap A and E in stage (d). The result will be the Huffmann code:

Letter	Frequency (%)	Code
A	25	00
B	15	100
C	10	101
D	20	11
E	30	01

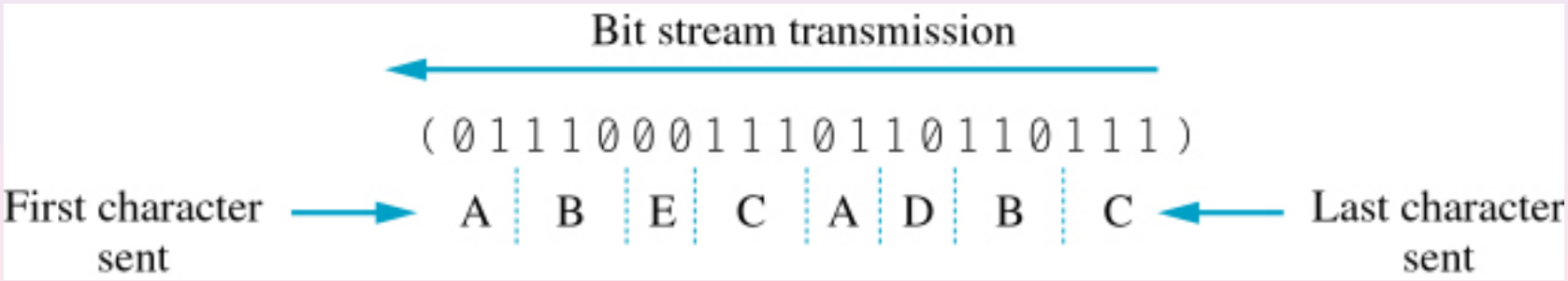
Huffmann code 8

Huffmann code properties:

- There are many Huffmann codes which can be associated to the same characters and weights. So, we should better speak of *a* Huffmann code instead of *the* Huffmann code.
- All Huffmann codes are, in general, variable-length, frequency-dependent, prefix codes. Consequently, Huffmann codes are uniquely decodable.

Huffmann code 9

Let us decode the codeword 01110001110110110111:



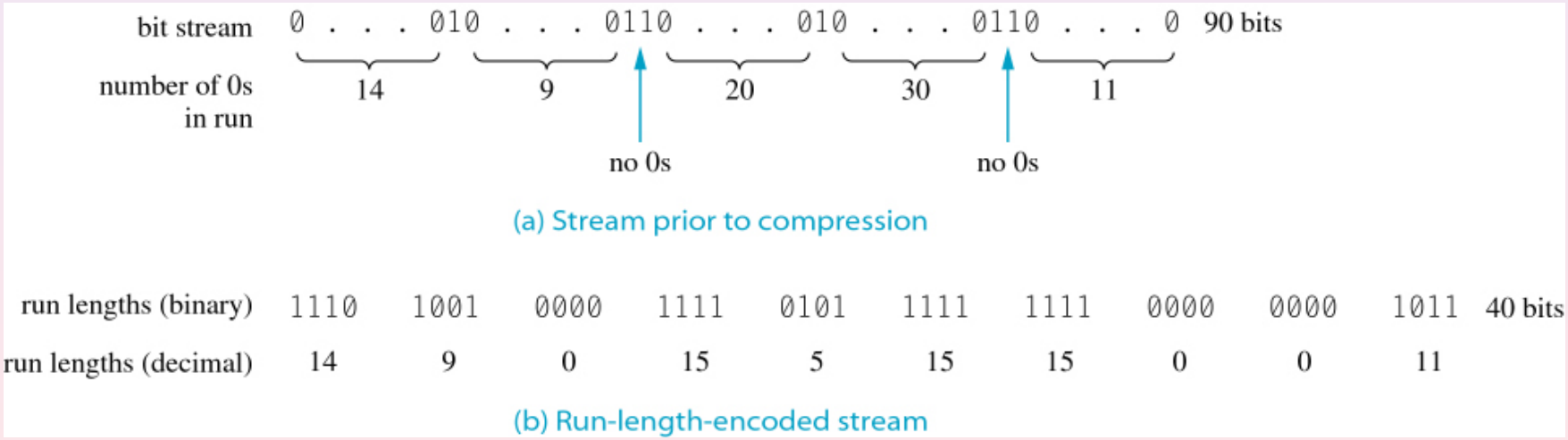
Run-length encoding 1

Huffmann codes are useful only in case we know the frequency of characters. Many items that travel the communications media, including binary files, fax data, and video signals, do not fall into this category.

Run-length encoding analyses bit-strings by looking for long runs of 0 or 1. Instead of sending all bits, *it sends only how many of them are in the run*. Fax transmission is well-served by this technique as potentially a large part of a page is white space, corresponding to a long run of 0s.

Run-length encoding 2

The sender transmits only the length of each run as a fixed-length integer; the receiver gets each length and generates the proper number of bits in the run, inserting the other bit in between. Here is an example:



This technique works well when there are many long 0 runs.

Run-length encoding 3

What about long runs of different bits or even characters?

Solution: send the actual character along with the run length. For example,

HHHHHHHUFFFFFFFFFFFFFFFYYYYYYY
YYYYYYYYYYYYYYYDGGGGGGGGGGG

can be sent as 7,H,1,U,14,F,20,Y,1,D,11,G.

Relative encoding 1

A single video image may contain little repetition, but *there is a lot of repetition over several images*.

Reason: a) a USA TV signal sends 30 pictures per second, b) each picture generally varies only slightly from the previous one.

The **relative encoding** or **differential encoding** works as follows:

- the first picture is sent and stored in a receiver's buffer;
- the second picture is compared with the first one and the encoding of differences are sent in a frame format; the receiver gets the frame and applies the differences to create the second picture;
- the second picture is stored in a receiver's buffer and the process continues.

Relative encoding 2

Here is an example of relative encoding:

5 7 6 2 8 6 6 3 5 6	5 7 6 2 8 6 6 3 5 6	5 7 6 2 8 6 6 3 5 6
6 5 7 5 5 6 3 2 4 7	6 5 7 6 5 6 3 2 3 7	6 5 8 6 5 6 3 3 3 7
8 4 6 8 5 6 4 8 8 5	8 4 6 8 5 6 4 8 8 5	8 4 6 8 5 6 4 8 8 5
5 1 2 9 8 6 5 5 6 6	5 1 3 9 8 6 5 5 7 6	5 1 3 9 7 6 5 5 8 6
5 5 2 9 9 6 8 9 5 1	5 5 2 9 9 6 8 9 5 1	5 5 2 9 9 6 8 9 5 1
First frame	Second frame	Third frame
	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0
	0 0 0 1 0 0 0 0 −1 0	0 0 1 0 0 0 0 1 0 0
	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0
	0 0 1 0 0 0 0 0 1 0	0 0 0 0 −1 0 0 0 1 0
	0 0 0 0 0 0 0 0 0 0	0 0 0 0 0 0 0 0 0 0
	Transmitted frame contains the encoded differences between the first and second frames.	Transmitted frame contains the encoded differences between the second and third frames.

Lempel-Ziv compression

The **Lempel-Ziv compression** method looks ahead in the input, finding the longest previously-transmitted string which is a prefix of the input. It transmits a reference to that previous string, thereby avoiding sending the same string more than once.

This technique—replacing an input string by an encoded reference to a prior string—is widely used:

1. UNIX compress command,
2. gzip on UNIX,
3. V.42bis compression standard for modems,
4. GIF (Graphics Interchange Format).

JPEG 1

Pixels are represented using 8 bits which can distinguish 256 shades of gray, ranging from white to black.

For colour pictures we can use one byte to represent each of the three primary colours. The intensity of each colour can be adjusted according to the 8-bit value to produce the desired colour. With $3 \times 8 = 24$ bits we can describe $2^{24} = 16,777,216$ different colours.

An alternative method uses three codes as follows: one codeword Y for *luminance* (brightness), and two codewords (U, V) for *chrominance*. The U value is the Blue intensity minus Y, and V is the Red intensity minus Y.

In 4:2:2 Y'UV coding, the luminance is scaled non-linearly (gamma-corrected) into a Y' signal at four bits per pixel. The U and V components are just two bits per pixel, because the human eye is less sensitive to chroma than to luma.

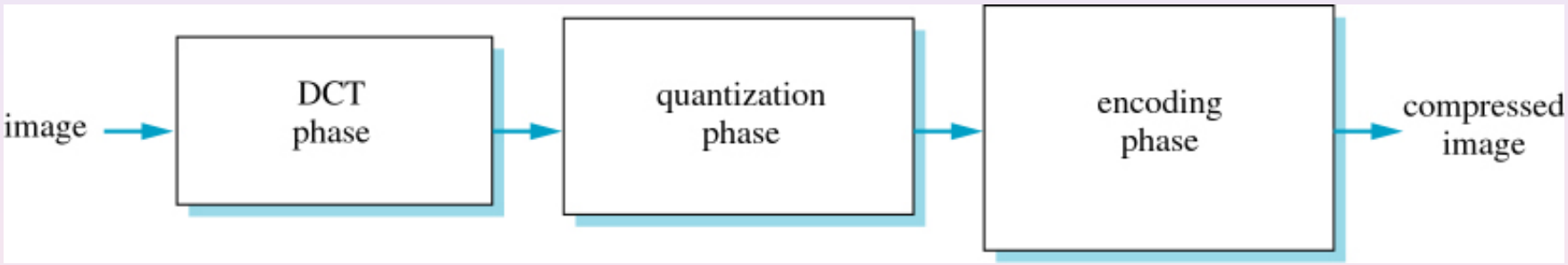
JPEG 3

JPEG is an acronym for the Joint Photographic Experts Group and **JPEG compression** is a **lossy** data compression method. This means that the image obtained after decompression may not coincide with the original image. Lossy compression is acceptable for images because of the inherent limitations of the human optical system.

There are three phases in a JPEG compression:

1. the discrete cosine transform (DCT),
2. quantisation,
3. encoding phase.

JPEG 4



JPEG three phases

JPEG 5

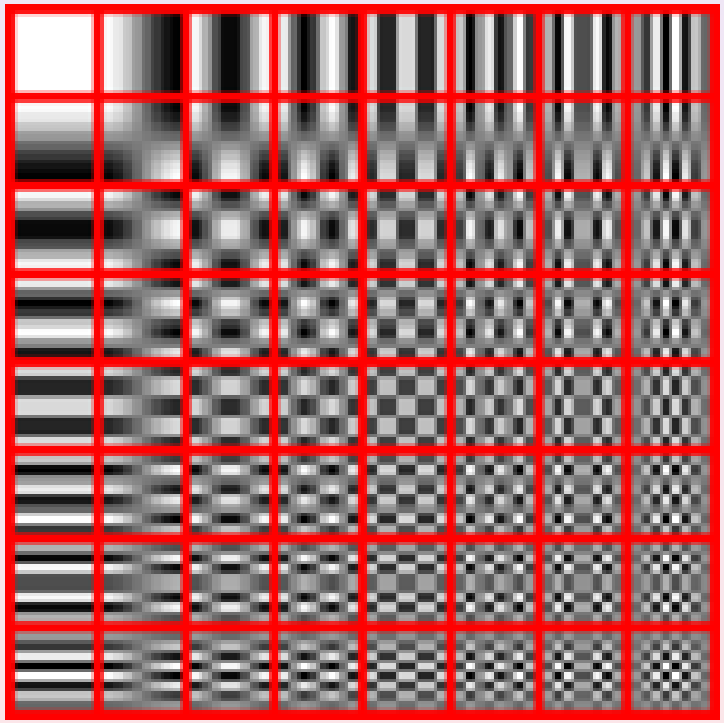
In the DCT phase, each component of the image is "tiled" into sections of 8×8 pixels each; dummy data fills incomplete blocks.

Usually, there are three components in a JPEG image, corresponding to the (Y', U, V) we discussed earlier. Other colour-spaces and grey-scale (a single 8-bit component) may be used.

The chroma components are usually downsampled, i.e. encoded at lower spatial resolution than the luma.

JPEG 6

Then, each tile in each component is converted to frequency space using a two-dimensional forward "discrete cosine transform", using the basis functions shown below.



picture source: http://en.wikipedia.org/wiki/JPEG

JPEG 7

The human eye can see small differences in brightness over a relatively large area, but is insensitive to brightness variation at high frequency.

JPEG attenuates the high frequency signal components by dividing each component G_{ij} in the frequency domain by a constant Q_{ij}, where the value of the constant is larger for the higher frequency components (i.e. the ones with larger i,j values). After division, the component is rounded to the nearest integer. This is the **quantisation phase**, in which the main lossy operation takes place.

As a result, in the **encoding** phase, many of the higher frequency components are rounded to zero, and many of the rest become small positive or negative numbers; they take many fewer bits to store.

JPEG 8

The compression ratio of JPEG depends greatly on the divisors used during the quantisation phase. These are controlled by a "Quality" parameter.

At 10:1 compression, it is difficult to distinguish the compressed image from the original one.

At 100:1 compression, a JPEG-compressed image is usually still recognisable but has many visual artifacts, especially near sharp edges.

http://en.wikipedia.org/wiki/JPEG

GIF

GIF (Graphics Interchange Format) compresses by: a) reducing the number of colours to 256, and b) trying to cover the range of colours in an image as closely as possible.

It replaces each 24-bit pixel value with an 8-bit index to a table entry containing the colour that matches the original "most closely". In the end, a variation of the Lempel-Ziv encoding is applied to the resulting bit values.

GIF files are lossy if the number of colours exceeds 256 and lossless otherwise.

MPEG and MP3 1

The **Moving Picture Experts Group (MPEG)** is a working group of ISO/IEC charged with the development of video and audio encoding standards.

The video codecs from MPEG use the discrete cosine transform techniques of the JPEG image compressor, and they also take advantage of redundancy between successive frames of video for "inter-frame compression". Differences between successive frames can be encoded very compactly, when the motion-prediction techniques are successful.

Note: MPEG holds patents, and charges license fees. In June 2002, China formed a working group to develop an alternative set of audio and video codecs. The group was successful, see `http://www.avs.org.cn`, however the H.263 (MPEG-2) and H.264 (MPEG-4 Part 10) codecs are still commonly used in China.

MPEG and MP3 2

MPEG has standardised the following compression formats and ancillary standards:

- MPEG-1: video and audio compression standard; it includes the popular Layer 3 (MP3) audio compression format
- MPEG-2: transport, video and audio standards for broadcast-quality television
- MPEG-4: expands MPEG-1 to support video/audio "objects", 3D content, low bit-rate encoding and support for Digital Rights Management

MPEG and MP3 3

MPEG-1 Audio Layer 3, better known as **MP3**, is a popular digital audio encoding, lossy compression format, and algorithm.

MP3 is based on the **psycho-acoustic model**, **auditory mask,** and **filter bank**.

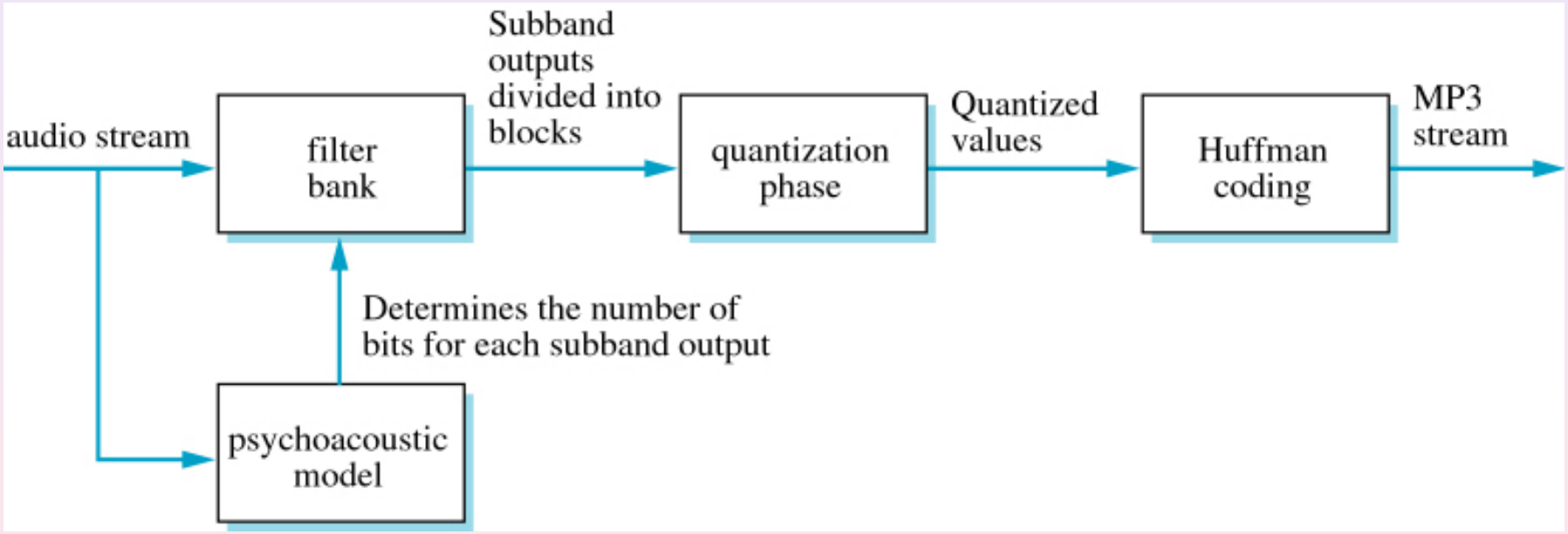
Psycho-acoustics is the study of the human auditory system to learn what we can hear and what sounds we can distinguish. In general we can hear sounds in the 20 Hz to 20 kHz range, but sounds with close frequencies (for example, 2000 Hz and 2001 Hz) cannot be distinguished.

MPEG and MP3 4

The auditory mask is the following phenomenon: if a sound with a certain frequency is strong, then we may be unable to hear a weaker sound with a similar frequency.

The filter bank is a collection of filters, each of which creates a stream representing signal components of a specified ranges. There is one filter for each of the many frequency ranges. Together they decompose the signal into **sub-bands**.

MPEG and MP3 5



MP3 encoding

MPEG4 1

The MPEG standards define audio and video codecs, as well as file formats.

The MP4 file format (MPEG-4 Part 14) is based on Apple's QuickTime container format. The file header indicates what codecs are used for video and audio, and it also gives values for important paramaters such as the video frame rate, horizontal and vertical resolution (i.e. number of pixels per line, and number of lines per frame), and colourspace.

MPEG4 2

The video codecs defined in MPEG-4 are all extensions of JPEG (or closely-related DCT-based techniques). Each video frame can be individually JPEG-encoded, however usually each JPEG-encoded frame (an "I-frame") is followed by a few frames which have been encoded, much more compactly, using codewords which refer to 8x8 image blocks in the preceding (or following) I-frame.

These video compressors are especially successful in video with stationary backgrounds, but are not as effective in scenes with a lot of differently-moving objects – such as the leaves on a tree, on a windy day. When compression ratios are high for video with high-frequency information, JPEG-like artifacts become apparent e.g. sharp edges are sometimes rendered at very low resolution (as 8x8 blocks).