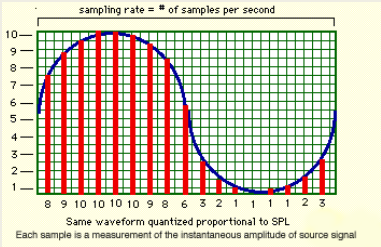
Sampled Audio

Sampled Audio

- We store digitised audio samples.
- This is the main audio data.
- In a file (e.g. .wav format) we also need some header information.
- We will **look** at the headers and data.
- Then **we will modify** the **audio data**.

Sampled Audio

- Sounds from the real world can be recorded and digitsed using an analog-to-digital converter (ADC).
- We take a sample of the instantaneous amplitude of the analog waveform.
- Samples are taken at a regular time interval. The rate of sample measurement is the sampling rate (frequency).
- Frequencies will be recreated later by playing back the sequential sample amplitudes at a specified rate.

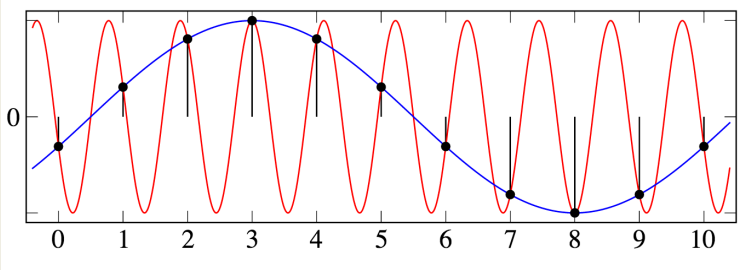


http://www.indiana.edu/~emusic/etext/digital_audio

Nyquist Theorem

- How often shall we take the sample?
- In 1928 Harry Nyquist from AT&T published a paper: "Certain Topics in Telegraph Transmission Theory."
- He presented a method for converting analog waveforms into digital signals for more accurate transmission over phone lines.
  - *If an analog signal were **band-limited** (i.e., had no frequencies higher than a specific band), it could be **captured** and **transmitted** in digital values and then **recreated** in an analog form on the receiving end.*
  - *He determined that the sampling rate would need to be **at least twice the highest frequency** to be reproduced.*
  - *For those interested in the mathematics, a copy of Shannon's proof can be found here.*

Aliasing



- Aliasing is an effect that causes different signals to become indistinguishable when sampled.
- It also refers to the distortion or artifact that results when the signal reconstructed from samples is different from the original continuous signal.

Aliasing (spatial and temporal)

http://users.wfu.edu/matthews/misc/DigPhotog/alias/

Temporal aliasing

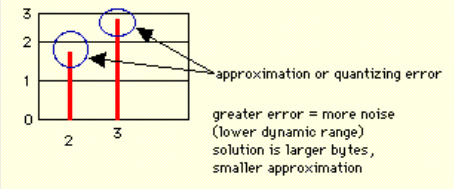
Sample rates and Bandwidth.

- The bandwidth of audio and video signals can be considered to be the highest frequency carried by the signal.
  - *In sound "crispness".*
  - *In vision "sharpness".*

8,000 Hz	Telephone, walkie-talkie
44,100 Hz	Audio CD
192,000 Hz	DVD-Audio

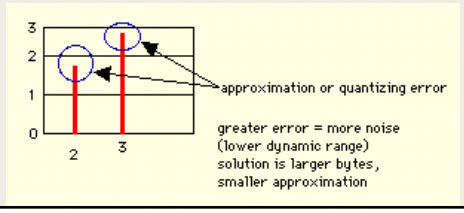
Quantisation and Coding

- Samples taken are then assigned numeric values that the computer or digital circuit can use in a process called **quantisation**.
- The number of available values is determined by the number of bits used for each sample, also called **bit depth** or **bit resolution**.
- When a sample is quantized, the instantaneous snapshot of its analog amplitude has to be rounded off to the nearest available digital value. This rounding-off process is called **approximation**.



Quantisation and Coding

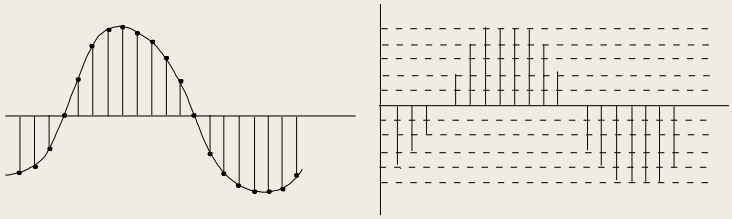
- The smaller the number of bits used per sample, the greater the distances the analog values need to be rounded off to. The difference between the analog and the digital value is called **quantising error**.
- Bit depth affects **dynamic range**, or the amplitude difference between the digital noise floor and the loudest possible sound before distortion.
- The CD standard of 16-bit samples, with 65,536 values for quantizing, provide the theoretical playback system optimum of a 96 dB dynamic range.



Sampled Audio

- Also, we must decide on some level of quantisation or bits per sample nbits.
- Total bit rate is therefore
  - *nbits x f_s bits per second.*
- After deciding on some coding scheme, we store the numeric representation of the samples sequentially.

3 stages of digitisation



Sampling Quantisation

1011, 1100, 1101, 0000, 0010, 0100, 0101,
0101, 0101, 0101, 0100, 0010, 0000,
1101, 1100, 1011, 1011, 1011, 1011, 1100

Coding

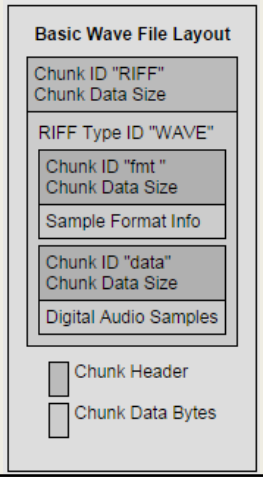
The “wav” format

- The Resource Interchange File Format (RIFF) is a container format for storing data in tagged chunks.
- Microsoft uses it for storing wav data
- Consists of a “RIFF” header.
- Broken into chunks and subchunks.
- Chunks (and subchunks) have:
  - *Label of chunk*
  - *Size of chunk*
  - *Then the chunk itself (body)*

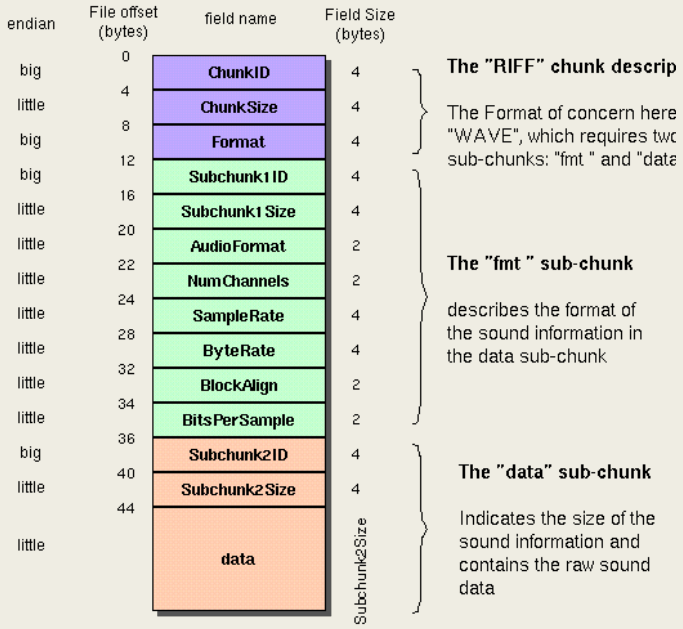
“wav” file always has

- *RIFF chunk*
- *Fmt subchunk*
- *Data subchunk*

http://www.sonicspot.com/guide/wavefiles.html

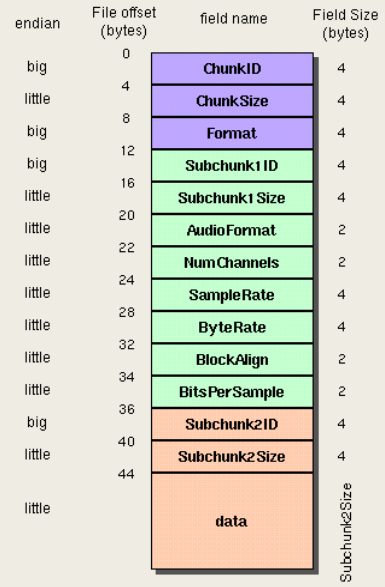


The “RIFF” header



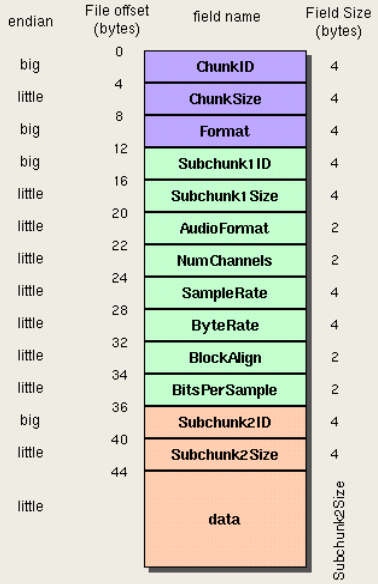
The “RIFF” header

- Byte 1 – 4 “**RIFF**”
  - *Type of file in ASCII*
- Byte 5 - 8
  - *Size of file from byte 9 onwards*
- Byte 9 – 12 “**WAVE**”
  - *Form of data in ASCII*
- Byte 13 – 16 “**fmt** ”
  - *Format in ASCII*
- *Byte 17 – 20*
  - *Size in bytes of the format infor.*
- Byte 21 – 22
  - *Code (1 represents Microsoft PCM)*



The “RIFF” header

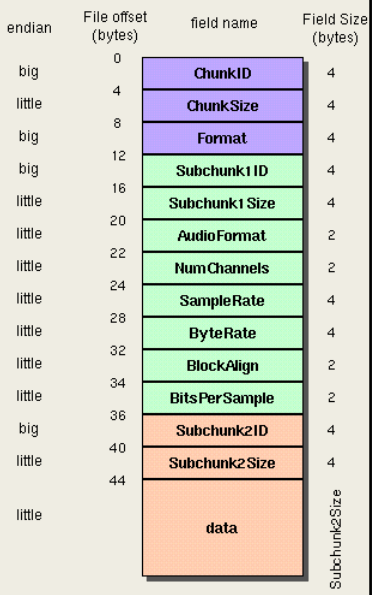
- Byte 23 – 24
  - Number of channels
- Byte 25 – 28
  - Samples per second
- Byte 29 – 32
  - Average bytes per second
- Byte 33 – 34
  - Block alignment
- Byte 35 – 36
  - Bits per sample
- Byte 37 - 40 “**data**”
  - Indicates the start of the data block
- Byte 41 – 44
  - Size in bytes of the data



An example

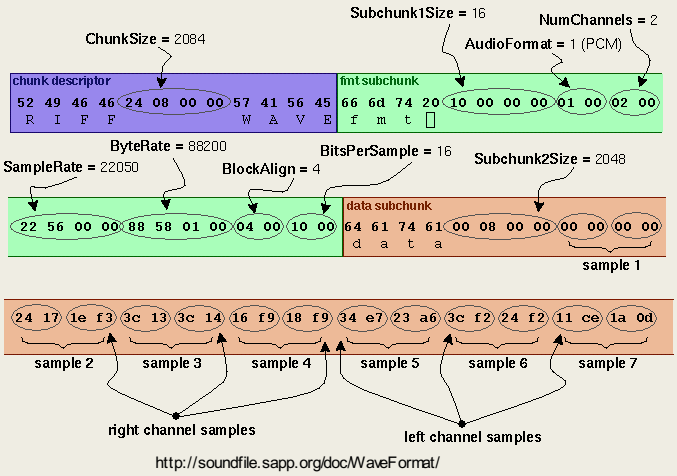
The first 72 bytes of a WAVE file with bytes shown as hex numbers:

52 49 46 46 24 08 00 00 57 41
56 45 66 6d 74 20 10 00 00 00
01 00 02 00 22 56 00 00 88 58
01 00 04 00 10 00 64 61 74 61
00 08 00 00 00 00 00 00 24 17
1e f3 3c 13 3c 14 16 f9 18 f9
34 e7 23 a6 3c f2 24 f2 11 ce
1a 0d



An example

The first 72 bytes of a WAVE file with bytes shown as hex numbers:
Here is the interpretation of these bytes as a WAVE sound file:



http://soundfile.sapp.org/doc/WaveFormat/

Block Alignment

BlockAlign == NumChannels * BitsPerSample/8
The number of bytes for one sample including all channels.

Sample 1		Sample 1	
Byte 1	Byte 2	Byte 3	Byte 4
Channel 0 Low byte	Channel 0 High byte	Channel 1 Low byte	Channel 1 High byte

Format codes for audio data

- 0x0001
  - *Microsoft pulse-code-modulation (PCM) format*
- 0x0101
  - *IBM m-law format*
- 0x0102
  - *IBM a-law format*
- 0x0103
  - *IBM AVC adaptive differential PCM format*

Other possible “subchunks”

- Fact chunk
  - *This gives information about the file which is particularly useful for other formats, for example where the data is compressed. In PCM this is not required.*
- Cue chunk
  - *The cue chunk identifies points in the data stream.*
- Playlist chunk
  - *Gives the order in which the data is to be played using the cue points.*
- Associated data chunks
  - *Other information that you might want stored such as text.*

Sound file handling using Matlab

- Use **wavread** to retrieve audio data.
  - *snd= wavread ('chord.wav')*
- Or more properly
  - *[snd , fs, bits]= wavread ('chord.wav')*
- **If you are programming in Matlab 2015b, use audioread instead.**
  - ***[snd , fs]= audioread('chord.wav')***

Sound file handling

- Audio data, specified as an m-by-1 column vector for single-channel (mono) audio, or an m-by-2 matrix for stereo playback, where m is the number of audio samples.
- If y is an m-by-2 matrix, then the first column corresponds to the left channel, and the second column corresponds to the right channel. Stereo playback is available only if your system supports it.

snd

606171x2 double

	1	2
190	-0.0522	0.0852
191	-0.0694	0.0839
192	-0.0798	0.0788
193	-0.0812	0.0760
194	-0.0774	0.0794
195	-0.0679	0.0845
196	-0.0492	0.0890
197	-0.0211	0.0907
198	0.0096	0.0874
199	0.0410	0.0803
200	0.0713	0.0754
201	0.0912	0.0742
202	0.0967	0.0718
203	0.0938	0.0660
204	0.0892	0.0588
205	0.0815	0.0506
206	0.0680	0.0442
207	0.0506	0.0373

Sound file handling using Matlab

- Use **wavwrite (audiowrite)** to save audio data.
  - *wavwrite (snd, 'chord.wav') (simplest)*
  - *See Matlab help for more options*
- Use **sound**(snd) to play sound.
- More properly sound (snd, fs)

Some manipulation

- Cut a sound
  - *Acut = A(start:end,:)*
- Join two sounds together
  - *Newsound = [A (start:end,:);B (start:end,:)] (Vertical concentration)*

Matlab Exercises

- Change the sampling frequency.
- Remove part of the sound using the Matlab : operator.
- Play one channel of the stereo sound using array indexing.

Matlab Exercises

- Change the volume by multiplication/division.
- Mix two sounds together. (Must be of the same length)
- Add echo to the sound, by starting a lower level version a bit later.
- Invent your own effects.

Download.wavetlan.com/SVV/Media/HTTP/http-wav.htm

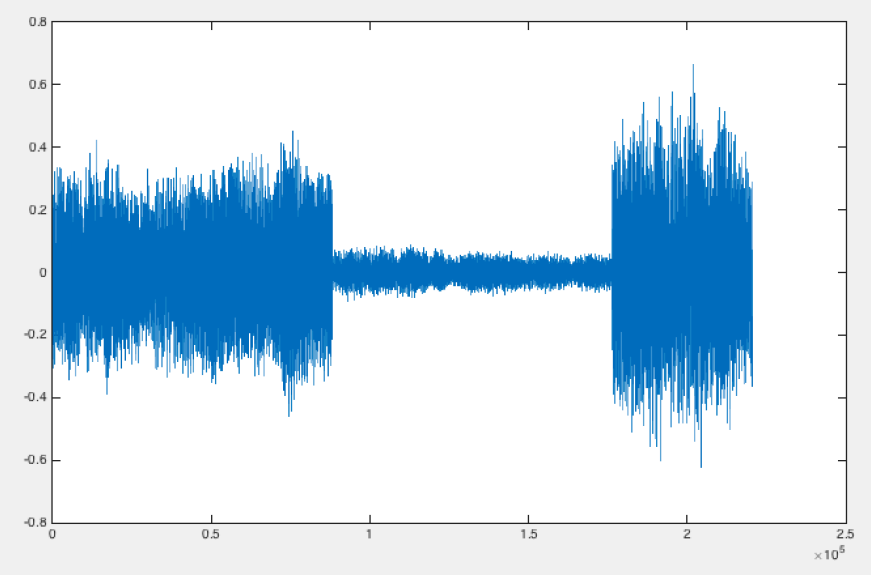
Matlab example code

```
clear all
clc
DATATYPE='double';
FILENAME='road.wav'; %input filename
SAVENAME='newroad.wav'; %output filename
[snd,FS,bits]=wavread(FILENAME); %use [snd,FS]=audioread(FILENAME) if you are on Matlab2015b
audiosize=size(snd); %Return audiosize as [Length in samples,number of channels]
leftchannel=snd(:,1); %Extract left channel
rightchannel=snd(:,2); %Extract right channel
START=1;
END=audiosize(1); %The size of audio file in number of samples

%UNCOMMENT THE FOLLOWING COMMAND TO ENABLE THE TEST
%sound(snd,FS); %Play original audio
%sound(snd,2*FS); %play original audio using twice the sample rate
%sound(leftcnannel,FS); %Play left channel (your PC will play left channel as stereo.
%forcemono=[leftchannel,zeros(size(leftchannel))];
%sound(forcemono,FS); Force to play only leftchannel in left speaker
%sound(snd(START:round(END/2)),FS); Play half of the original audio

firstcut=leftchannel(1:FS*2); %Extract two seconds of audio
secondcut=leftchannel(FS*3:FS*5)*0.2; %lower volumn
thirdcut=leftchannel(FS*6:FS*7)*2; %increase volumn
newedit=[firstcut;secondcut;thirdcut]; %Create a new edit by joining three cuts together
figure(1),plot(newedit);
%wavwrite(newedit,FS,SAVENAME); %Save the new edit to a file
sound(newedit,FS);

```



Mono playback

- Your mono sound (with one channel) may be populated to both channels by your computer for playback. Use to following command to force mono playback (i.e., define the second channel as zeros);
  - forcemono=[leftchannel,zeros(size(leftchannel))];

Lab exercises

http://homepages.udayton.edu/~hardierc/ECE203/sound.htm