

Multimedia

Module No: CM0340

© David Marshall 2013



Back

Close

About This Course

Aims of Module

- Single Module
 - Lectures — 2 Hours of Lectures weekly.
 - Tutorials
 - Labs — Multimedia Lab (C 2.10, Lab 4) 1 hour per student.
Weekly from week 2.

Assessment:

- Exam 80%
- Coursework 20%



Back

Close

- Lecturer

- Prof David Marshall
- Email: Dave.Marshall@cs.cf.ac.uk
- Office: S/2.20

Relationship with previous modules

- MATLAB will be used for examples and demos – basics covered in CM2202;
 - more practice in lab classes
- Difficult maths already covered in CM2202!



Back

Close

Course Material

<http://www.cs.cf.ac.uk/Dave/Multimedia/>

Course material will also be available on *Blackboard (Learning Central)*.

- PDFs of Slides (Colour)
- Coursework material.
- PDF — Additional Notes.
- HTML based notes
- Lots of Links to other material
- Always under Development — More to be added

Info also on Learning Central

- linked to above Web pages

Outline of Course

- Basic grounding in issue surrounding multimedia,
- Multimedia data:
 - Digital audio, graphics, images and video, etc.,
 - Underlying concepts and representations of sound, pictures and video,
 - Audio/Digital signal processing fundamentals — filtering, audio synthesis **Follows on from CM0268**
- Data compression — JPEG/GIF, MPEG video and MPEG Audio.
 - Core data compression algorithms in JPEG/MPEG etc.
- Transmission and Integration of media.
- Multimedia applications: e.g. content based retrieval.



Back

Close

Practical Work

Assessed Coursework

A small assessed practical programming “mini-project” based on **Multimedia digital audio synthesis/signal processing**.

Important Dates:

Hand Out: Week 3

Hand In: Week 10

MATLAB Programming Examples and Coursework

All module lecture/tutorial examples and the programming elements of the coursework will use **MATLAB**.



Back

Close

Outline of Module Delivery (1)

Lectures

- Focus on main theory of module.
- Lots of **Demos**:
 - Essential help for Assessed Coursework
 - MATLAB Examples explained in depth
 - **Interactive** — Questions and Answers please.
- Time:
 - Monday 3-4 pm
 - **Friday 12-1pm**

Outline of Module Delivery (2)

Tutorials

8

Multimedia
CM0340

- Revision of key aspects:
Filtering, Frequency Space (Fourier Transform).
- Focus on practical/programming elements of module prior to
Lab Class (Follows immediately after).
- Further Explanation of Lecture Demos.

All Lectures and Tutorial given by Lecturer

Tutorials: Weeks 1,2,4 and 8.



Back

Close

Outline of Module Delivery (3)

Lab Classes

- MATLAB programming help sessions
- Try out Lecture/Tutorial examples
- Extended reasoning and programming through Lab Worksheet Questions
- Build a solid basis for Assessed Coursework

Lab classes are in **C/2.10**, Weeks 2-10

Thursday 10-11AM

All lecture, tutorial and lab class material is examinable



Back

Close

Syllabus Outline

Topics in the module include the following:

- Introduction: Multimedia applications and requirements
- Multimedia data acquisition and formats: Audio, Graphics, Images and Video
- Audio/Video fundamentals including analog and digital representations, human perception, and audio/video equipment, applications.
- Digital Audio signal processing, Image/Video Processing.
- Digital Audio Synthesis: Basic audio synthesis techniques
- MIDI: Basic MIDI definitions, MIDI control of audio synthesis, MIDI and data compression (MPEG4)



Back

Close

Syllabus Outline (cont.)

- Audio and video compression
 - Lossy v. Lossless Compression
 - Information Theoretic Transform (Huffman Coding, Arithmetic Coding, LZW/GIF)
 - perceptual transform coders for audio/images/video (Fourier, DCT, Vector Quantization)
 - Image and video compression applications and algorithms:
JPEG, H.263, MPEG Video, MPEG Audio,
- Multimedia applications
 - Content based multimedia retrieval (audio & video)



Back

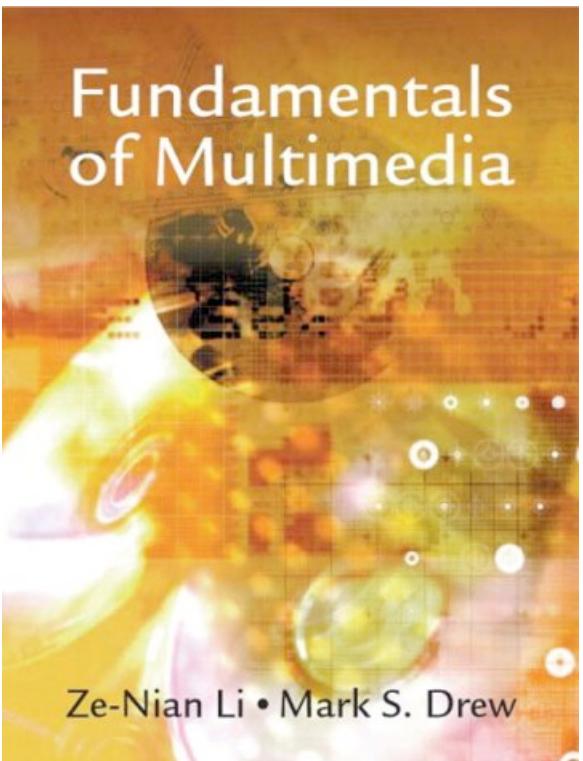
Close

Recommended Course Book

Fundamentals of Multimedia
Ze-Nian Li, Mark S. Drew
Prentice Hall, 2003
(ISBN: 0130618721)

*Decent coverage all
major aspects of the course
plus a lot more*

**No MATLAB Examples
Copies in library**

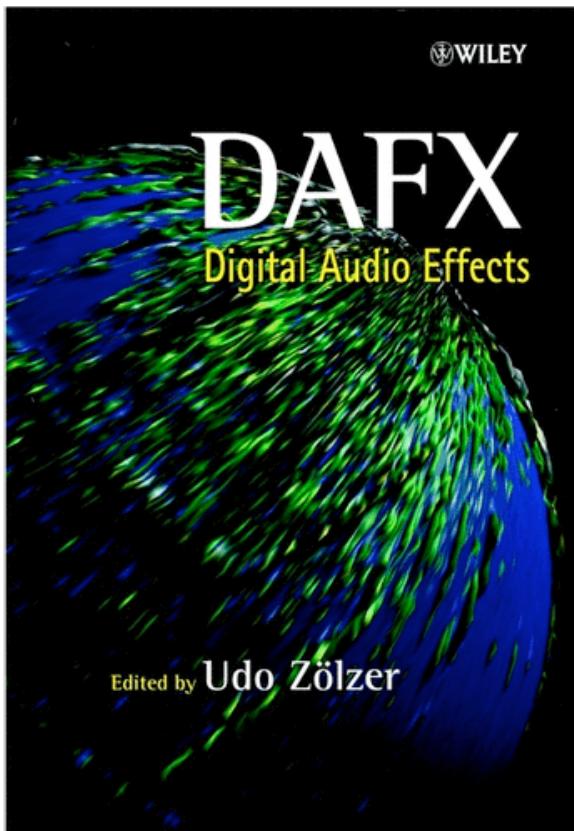


Other Texts Used In This Module: Practical MATLAB Based

DAFX: Digital Audio Effects
Udo Zolzer
John Wiley and Sons Ltd , 2002
(ISBN-13: 978-0471490784)

*Excellent coverage of audio
signal processing effects and
synthesis
plus a lot more*

All MATLAB examples
Expensive but copies in library



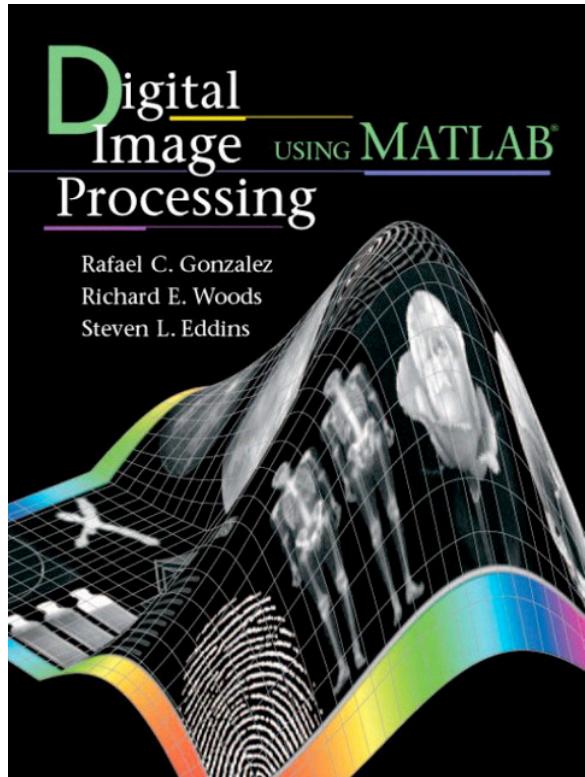
Other Texts Used In This Module: Practical MATLAB Based

Digital Image Processing Using
MATLAB

Rafael C. Gonzalez,
Richard E. Woods,
and Steven L. Eddins
Prentice Hall, 2004
(ISBN-13: 978-0130085191)

*Excellent coverage of Image
processing examples
plus a lot more*

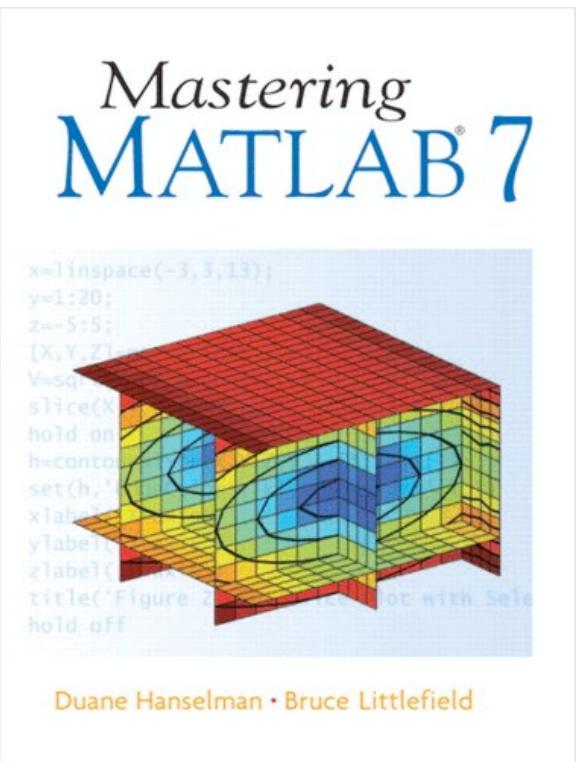
All MATLAB examples
Useful for CM0311 Image
Processing
Copies in library



Other Texts Used In This Module: Practical MATLAB Based

Mastering MATLAB
Duane C. Hanselman and Bruce
L. Littlefield
Prentice Hall, 2004
(ISBN-13: 978-0131857148)

*Excellent coverage of Basic
MATLAB programming
Copies in library*



Other Texts Used In This Module: Audio Synthesis

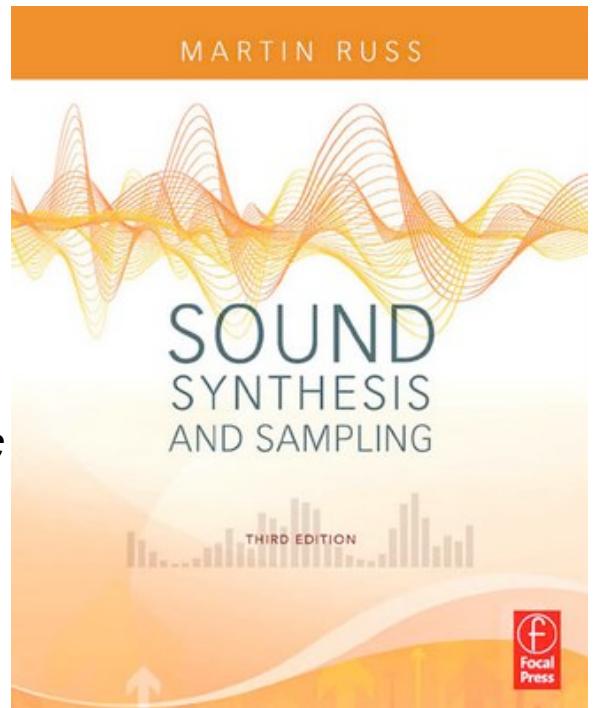
Sound Synthesis and Sampling
(Third Edition)

Martin Russ

Focal Press

(ISBN-13: 978-0240521053)

*Good coverage of basic
synthesis algorithms*



Copies in library

Other Texts Used In This Module: Compression Algorithms

Data Compression: The Complete Reference (Fourth Edition)

David Salomon

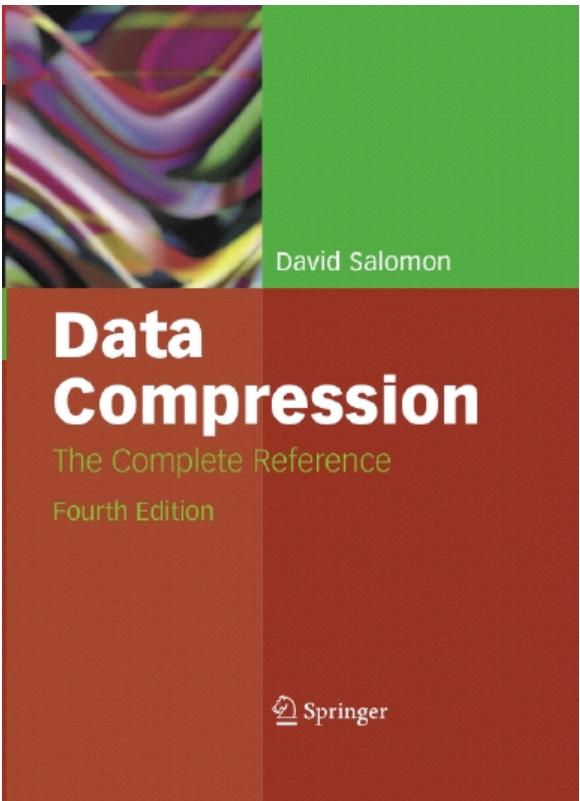
Springer-Verlag London, 2007

but (ISBN-13: 978-1846286025)

Comprehensive coverage of all compression algorithms and formats

Many more than covered in this course!

Expensive but Copies in library

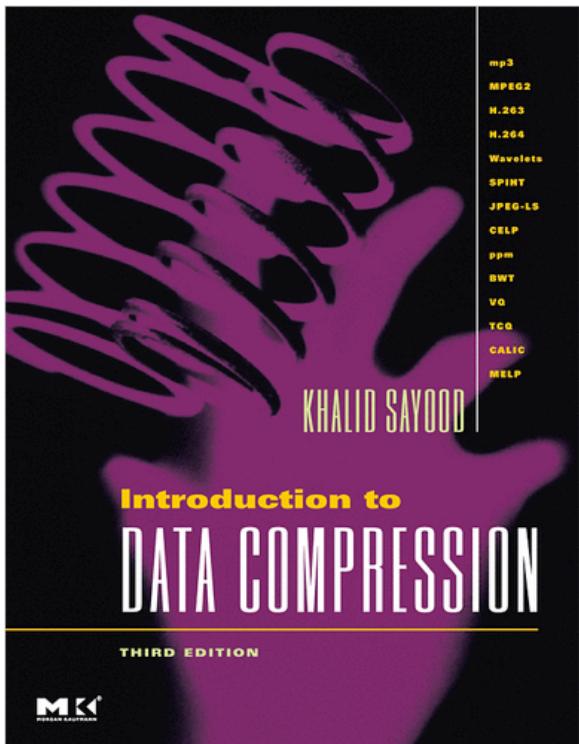


Other Texts Used In This Module: Compression Algorithms

Introduction to Data
Compression (3rd Edition)
Khalid Sayood
Morgan Kaufmann, 2005
(ISBN-13: 978-0126208627)

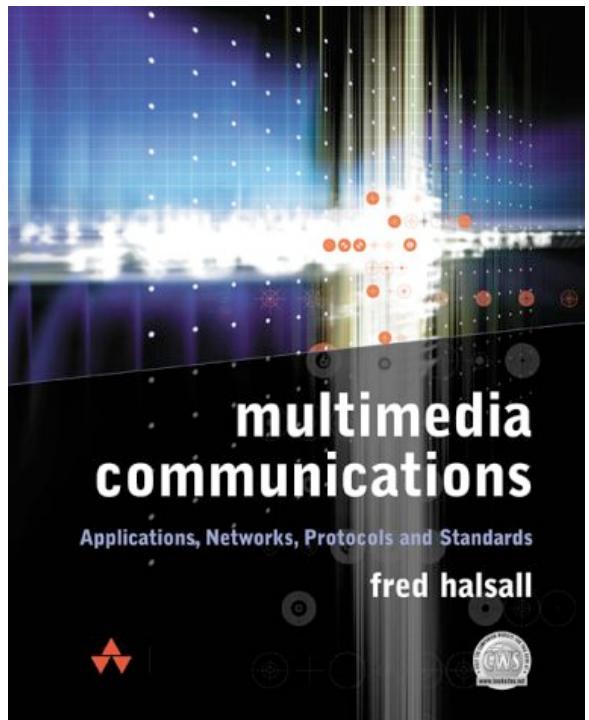
*Excellent coverage of all
compression algorithms and
formats*

Example code but not MATLAB
Copies in library



Other Good General Texts

Multimedia Communications:
Applications, Networks,
Protocols and Standards,
Fred Halsall,
Addison Wesley, 2000
(ISBN 0-201-39818-4)



Back

Close

Other Good General Texts

The following books are highly recommended reading:

Digital Audio

- A programmer's Guide to Sound, T. Kientzle, Addison Wesley, 1997 (ISBN 0-201-41972-6)
- Audio on the Web — The official IUMA Guide, Patterson and Melcher, Peachpit Press.
- The Art of Digital Audio, Watkinson, Butterworth-Heinmann.
- Synthesiser Basics, GPI Publications.
- Signal Processing: Principles and Applications, Brook and Wynne, Hodder and Stoughton.
- Digital Signal Processing, Oppenheim and Schafer, Prentice Hall.



Back

Close

Digital Imaging/Graphics/Video

- *Digital video processing*, A.M. Tekalp, Prentice Hall, 2005.
- *Encyclopedia of Graphics File Formats*, Second Edition by James D. Murray and William vanRyper, 1996, O'Reilly & Associates.

Data Compression

- *The Data Compression Book*, Mark Nelson, M&T Books, 1995.



Back

Close

Introduction to Multimedia

What is Multimedia?



Back

Close

What is Multimedia?

Multimedia can have many definitions these include:

(A computer system perspective)

Multimedia means that computer information can be represented through audio, video, and animation in addition to traditional media (i.e., text, graphics/drawings, images).

General Definition

A good general working definition for this module is:

Multimedia is the field concerned with the **computer controlled** integration of text, graphics, drawings, still and moving images (Video), animation, audio, and any other media where every type of information can be represented, stored, transmitted and processed **digitally**.

Multimedia Application Definition

A **Multimedia Application** is an application which uses a collection of multiple media sources e.g. text, graphics, images, sound/audio, animation and/or video.



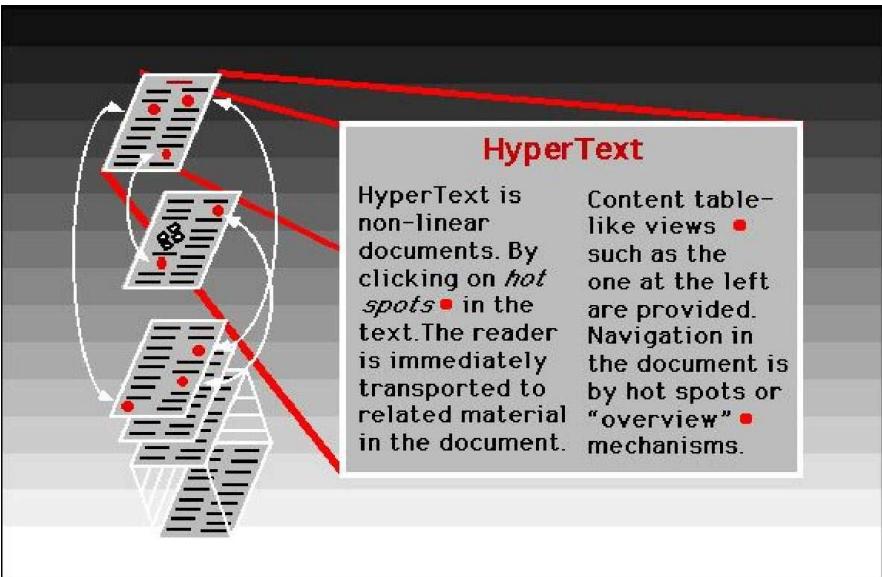
Back

Close

What is HyperText and HyperMedia?

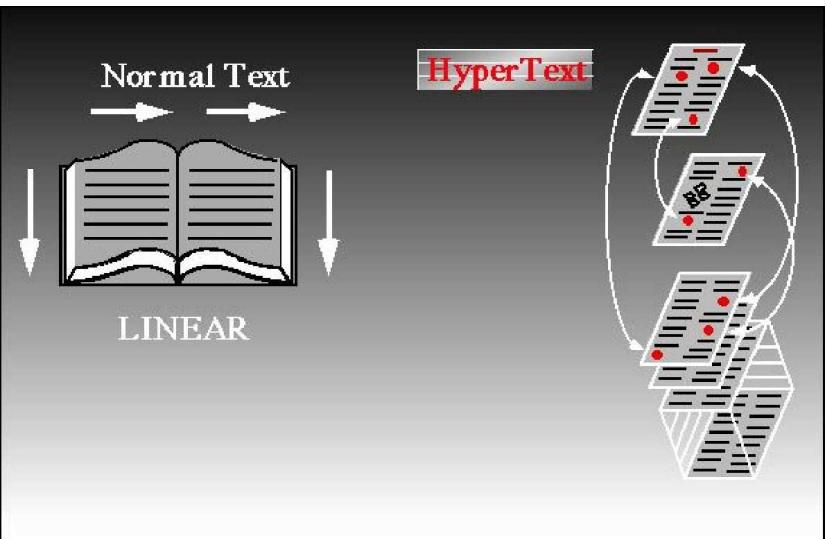
Hypertext is a text which contains links to other texts.

The term was invented by Ted Nelson around 1965.



HyperText Navigation

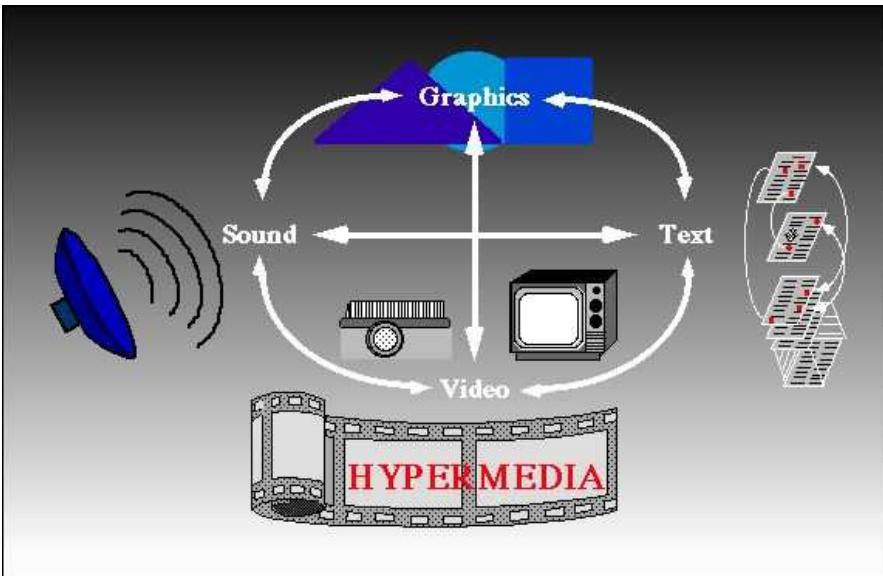
Traversal through pages of hypertext is therefore usually non-linear (as indicated below).



This has implications in layout and organisation of material — and depends a lot on the application at hand.

Hypermedia

HyperMedia is not constrained to be text-based. It can include other media, e.g., graphics, images, and especially the continuous media – sound and video.



Example Hypermedia Applications?

Example

Multimedia
CM0340

29



Back

Close

Example Hypermedia Applications?

- The World Wide Web (WWW) is the best example of a hypermedia application.
- Powerpoint
- Adobe Acrobat (or other PDF software)
- Adobe Flash
- Many Others?



Back

Close

Multimedia Applications

Examples of Multimedia Applications include:

- World Wide Web
- Multimedia Authoring, *e.g.* Adobe/Macromedia Director
- Hypermedia courseware
- Video-on-demand
- Interactive TV
- Computer Games
- Virtual reality
- Digital video editing and production systems
- Multimedia Database systems

Multimedia Systems

A **Multimedia System** is a system capable of processing multimedia data and applications.

A **Multimedia System** is characterised by the processing, storage, generation, manipulation and rendition of Multimedia information.



Back

Close

Characteristics of a Multimedia System

A **Multimedia system** has **four** basic characteristics:

- Multimedia systems must be **computer controlled**.
- Multimedia systems are **integrated**.
- The information they handle must be represented **digitally**.
- The interface to the final presentation of media is usually **interactive**.



Back

Close

Challenges for Multimedia Systems

- Distributed Networks
- Temporal relationship between data
 - Render different data at same time — continuously.
 - Sequencing within the media
playing frames in correct order/time frame in video
 - **Synchronisation** — inter-media scheduling

E.g. Video and Audio — Lip synchronisation is clearly important for humans to watch playback of video and audio and even animation and audio.

Ever tried watching an out of (lip) sync film for a long time?

Key Issues for Multimedia Systems

The key issues multimedia systems need to deal with here are:

- How to represent and store temporal information.
- How to strictly maintain the temporal relationships on play back/retrieval
- What process are involved in the above.
- Data has to represented **digitally** — Analog–Digital Conversion, Sampling etc.
- Large Data Requirements — bandwidth, storage,

Data compression is usually mandatory



Back

Close

Desirable Features for a Multimedia System

Given the above challenges the following feature a desirable (**if not a prerequisite**) for a Multimedia System:

Very High Processing Power — needed to deal with large data processing and real time delivery of media.

Special hardware commonplace.

Multimedia Capable File System — needed to deliver real-time media — e.g. Video/Audio Streaming.

Special Hardware/Software needed – e.g. RAID technology.

Data Representations — File Formats that support multimedia should be easy to handle yet allow for **compression/decompression** in **real-time**.



Back

Close

Efficient and High I/O — input and output to the file subsystem needs to be efficient and fast. Needs to allow for real-time recording as well as playback of data.
e.g. Direct to Disk recording systems.

Special Operating System — to allow access to file system and process data efficiently and quickly. Needs to support direct transfers to disk, real-time scheduling, fast interrupt processing, I/O streaming *etc.*

Storage and Memory — large storage units (of the order of hundreds of Tb if not more) and large memory (several Gb or more). Large Caches also required and high speed buses for efficient management.

Network Support — Client-server systems common as distributed systems common.

Software Tools — user friendly tools needed to handle media, design and develop applications, deliver media.



Back

Close

Components of a Multimedia System

Now let us consider the Components (Hardware and Software) required for a multimedia system:

Capture devices — Video Camera, Video Recorder, Audio Microphone, Keyboards, mice, graphics tablets, 3D input devices, tactile sensors, VR devices. Digitising Hardware

Storage Devices — Hard disks, CD-ROMs, DVD-ROM, etc

Communication Networks — Local Networks, Intranets, Internet, Multimedia or other special high speed networks.

Computer Systems — Multimedia Desktop machines, Workstations, MPEG/VIDEO/DSP Hardware

Display Devices — CD-quality speakers, HDTV, SVGA, Hi-Res monitors, Colour printers etc.



Back

Close

Applications

Examples of Multimedia Applications include:

- World Wide Web
- Hypermedia courseware
- Video conferencing
- Video-on-demand
- Interactive TV
- Groupware
- Home shopping
- Games
- Virtual reality
- Digital video editing and production systems

A Brief Look at Multimedia Data: Input and Format

Text and Static Data

- Source: keyboard, speech input, optical character recognition, data stored on disk.
- Stored and input character by character:
 - Storage of text is 1 byte per char / more bytes for Unicode.
 - For other forms of data (e.g. Spreadsheet files). May store format as text (with formatting) others may use binary encoding.
- Format: Raw text or formatted text e.g HTML, Rich Text Format (RTF), Word or a program language source (C, Pascal, etc..)
- Not temporal — BUT may have natural implied sequence e.g. HTML format sequence, Sequence of C program statements.
- Size Not significant w.r.t. other Multimedia data.

Graphics

- Format: constructed by the composition of primitive objects such as lines, polygons, circles, curves and arcs.
- Input: Graphics are usually generated by a graphics editor program (e.g. Illustrator) or automatically by a program (e.g. Postscript).
- Graphics are usually editable or revisable (unlike Images).
- Graphics input devices: keyboard (for text and cursor control), mouse, trackball or graphics tablet.
- graphics standards : OpenGL, PHIGS, GKS
- Graphics files usually store the primitive assembly
- Do not take up a very high storage overhead.



Back

Close

Images

- Still pictures which (uncompressed) are represented as a bitmap (a grid of pixels).
- Input: digitally scanned photographs/pictures or direct from a digital camera.
- Input: May also be generated by programs “similar” to graphics or animation programs.
- Stored at 1 bit per pixel (Black and White), 8 Bits per pixel (Grey Scale, Colour Map) or 24 Bits per pixel (True Colour)
- Size: a 512x512 Grey scale image takes up 1/4 MB, a 512x512 24 bit image takes 3/4 MB with no compression.
- This overhead soon increases with image size — modern high digital camera 10+ Megapixels \approx 29MB uncompressed!
- **Compression** is commonly applied.

Audio

- Audio signals are continuous analog signals.
- Input: microphones and then digitised and stored
- CD Quality Audio requires 16-bit sampling at 44.1 KHz
Even higher audiophile rates (e.g. 24-bit, 96 KHz)
- 1 Minute of Mono CD quality (uncompressed) audio requires 5 MB.
- 1 Minute of Stereo CD quality (uncompressed) audio requires 10 MB.
- Usually **compressed** (E.g. MP3, AAC, Flac, Ogg Vorbis).

Video

- Input: Analog Video is usually captured by a video camera and then digitised.
- There are a variety of video (analog and digital) formats
- Raw video can be regarded as being a series of single images. There are typically 25, 30 or 50 frames per second.
- *E.g.* A 512×512 size monochrome video images take $25 \times 0.25 = 6.25\text{MB}$ for a second to store uncompressed.
- Typical PAL digital video (720×576 pixels per colour frame)
 $\approx 1.24 \times 25 = 31\text{MB}$ for a second to store uncompressed.
- High Definition video on Blu-ray (up to $1920 \times 1080 = 2$ Megapixels per frame) $\approx 6.2 \times 25 = 155\text{MB}$ for a second to store uncompressed.
(There are higher possible frame rates!)
- Digital video **clearly needs** to be **compressed** for most times.

Summary: This Course is Essentially about Multimedia Data Compression

How can we compress data?

Lossy v Lossless :

Lossless : Ideal (e.g. zip, unix compress) not good enough for MM data!

Lossy : Throw away nonessential (perceptually less relevant) parts of the data stream

FILTER the data somehow.

Examples: MP3, JPEG, MPEG Video



Back

Close

Compression: Is there another way?

Compression via Synthesis :

Encode how to make (**synthesise**) the data

- can be done in many less bits in certain cases.

Examples: MP4 (Audio), MIDI, Vector Graphics (Flash),
MPEG Video.



Multimedia Data Basics

Multimedia systems/applications have to deal with the

- Generation of data,
- Manipulation of data,
- Storage of data,
- Presentation of data, and
- Communication of information/data

Lets consider some broad implications of the above



Back

Close

Discrete v Continuous Media

RECALL: Our Definition of Multimedia

- All data must be in the form of digital information.
- The data may be in a variety of formats:
 - text,
 - graphics,
 - images,
 - audio,
 - video.

Synchronisation

A majority of this data is large and the different media may need **synchronisation**:

- The data will usually have temporal relationships as an integral property.



Click here or image above to run movie

Static and Continuous Media

Static or Discrete Media — Some media is time independent:
Normal data, text, single images, graphics are examples.

Continuous media — Time dependent Media:
Video, animation and audio are examples.

Analog and Digital Signals

- Some basic definitions – [Studied HERE](#)
- Overviewing of technology — [Studied HERE](#)
- **More in depth study later.**



Back

Close

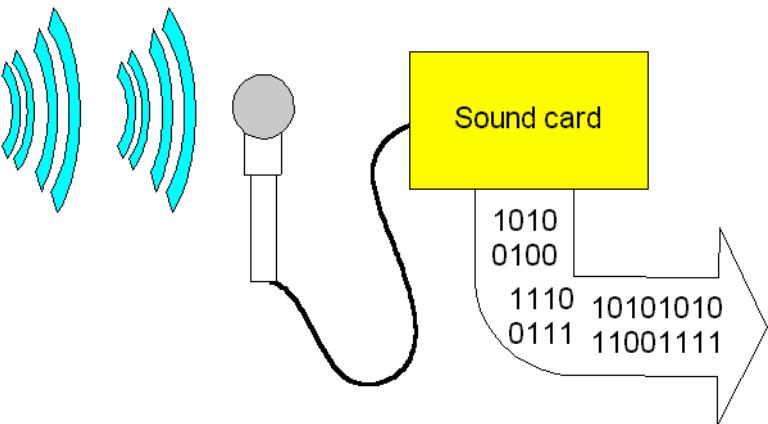
Analog and Digital Signal Conversion

The world we sense is full of analog signals:

- Electrical sensors convert the medium they sense into electrical signals
 - *E.g.* transducers, thermocouples (temperature sensor), microphones (acoustic sensor).
 - (usually) **continuous** signals
- **Analog**: continuous signals must be converted or **digitised** for computer processing.
- **Digital**: discrete digital signals that computer can readily deal with.

Analog-to-Digital Converter (ADC)

- Special hardware devices : **Analog-to-Digital** converters



Take analog signals from analog sensor (e.g. microphone)
and digitally sample data
(More details later)

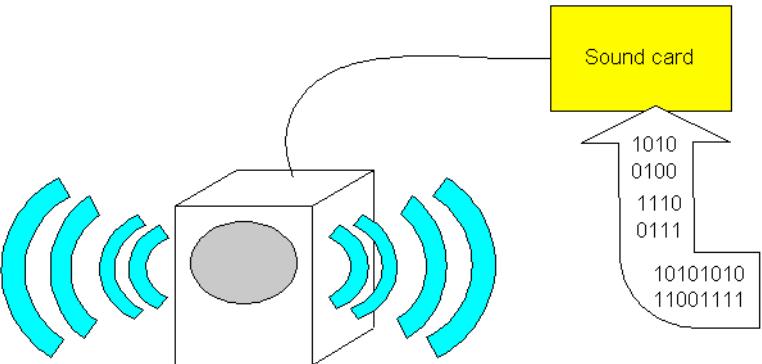


Back

Close

Digital-to-Analog Converter (DAC)

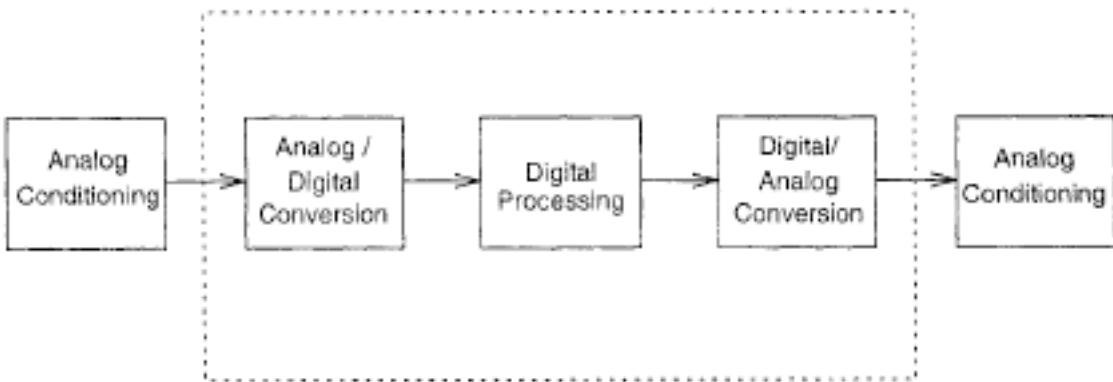
- Playback – a converse operation to *Analog-to-Digital*



- Takes digital signal, possibly after modification by computer (e.g. volume change, equalisation)
- Outputs an analog signal that may be played by analog output device (e.g. loudspeaker, CRT display)

Analog-to-Digital-to-Analog Pipeline (1)

- Begins at the conversion from the analog input and ends at the conversion from the output of the processing system to the analog output as shown:



(Strictly Analog-to-Digital-to-Analog is within the dotted box)



Back

Close

Analog-to-Digital-to-Analog Pipeline (2)

- Anti-aliasing filters (major part of *Analog Conditioning*) are needed at the input to remove frequencies above the sampling limit that would result in *aliasing*. **More later**
The anti-aliasing filter at the output removes the aliases that result from the sampling theorem.
- After the anti-aliasing filter, the analog/digital converter (ADC) quantises the continuous input into discrete levels.
- After digital processing, the output of the system is given to a digital/analog converter (DAC) which converts the discrete levels into continuous voltages or currents.
- This output must also be filtered with a low pass filter to remove the aliases from the sampling.
Subsequent processing can include further filtering, mixing, or other operations.
However, these shall not be discussed further in this course.



Back

Close

Multimedia Data: Input and format

How to capture and store each Media format?

Note that text and graphics (and some images) are mainly generated directly by computer/device (e.g. drawing/painting programs) and *do not* require digitising:

They are generated directly in some (usually binary) format.

- Printed text and some handwritten text can be scanned via Optical Character Recognition
- Handwritten text could also be digitised by electronic pen sensing
- Printed imagery/graphics can be flatbed scanned directly to image formats.

Text and Static Data

- Source: keyboard, speech input, optical character recognition, data stored on disk.



- Stored and input character by character:
 - Storage: 1 byte per character (text or format character), e.g. ASCII; more bytes for Unicode.
 - For other forms of data (e.g. Spreadsheet files). May store as text (with formatting, e.g. CSV – Comma-Separated Values) or may use binary encoding.

Text and Static Data (cont.)

- Formatted Text: Raw text or formatted text e.g HTML, Rich Text Format (RTF), Word or a program language source (C, Java, etc).
- Data **Not temporal** — **BUT** may have natural implied sequence e.g. HTML format sequence, Sequence of Java program statements.
- Size Not significant w.r.t. other Multimedia data formats.
- Compression: convenient to bundle files for archiving and transmission of larger files. *E.g. Zip, RAR, 7-zip.* General purpose compression programs may not work well for other media types: audio, image, video etc.

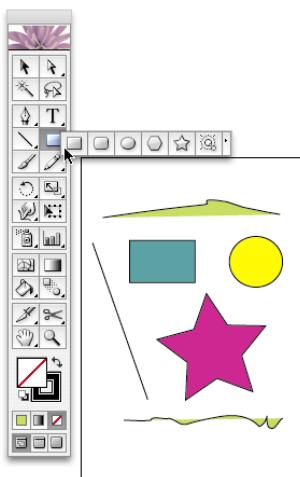
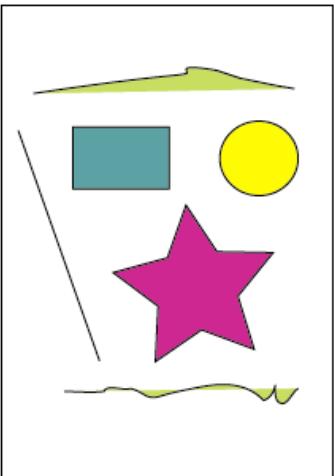


Back

Close

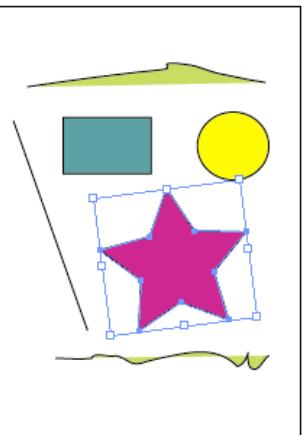
Graphics

- Format: constructed by the composition of primitive objects such as lines, polygons, circles, curves and arcs.
- Input: Graphics are usually generated by a graphics editor program (e.g. Illustrator, Freehand) or automatically by a program (e.g. Postscript).



Graphics (cont.)

- Graphics input devices: keyboard (for text and cursor control), mouse, trackball or graphics tablet.
- Graphics are usually selectable and editable or revisable (unlike images).



- Graphics files usually store the primitive assembly
- Do not take up a very high storage overhead.

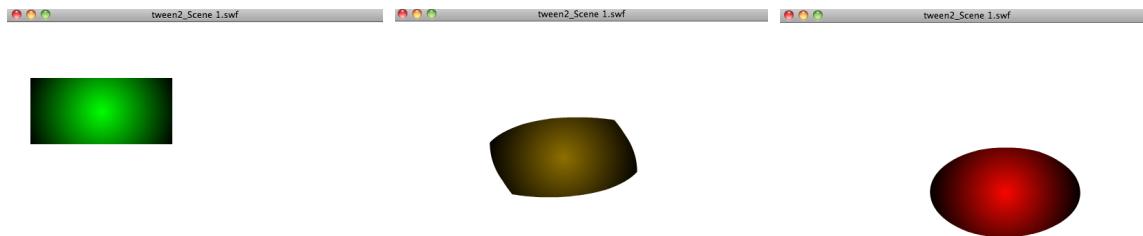


Back

Close

Graphics (cont.)

- Graphics standards : OpenGL - Open Graphics Library, a standard specification defining a *cross-language, cross-platform* API for writing applications that produce 2D/3D graphics.
- Animation: can be generated via a sequence of slightly changed graphics
 - 2D animation: e.g. Flash. Key frame interpolation.
tweening: motion; shape



Graphics (cont.)

- Animation (cont.)
 - 3D animation: e.g. Maya.
Change of shape/texture/position, lighting, camera
- Graphics animation is *compact*
 - suitable for network transmission (e.g. Flash).



© Gaia Dream Creation Inc. GaiaDreamCreation.com

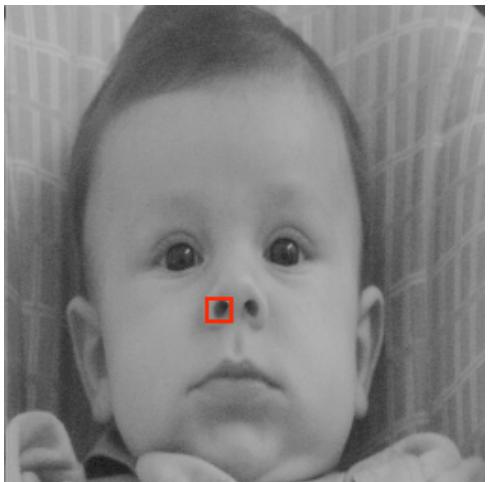


Back

Close

Images

- Still pictures which (uncompressed) are represented as a bitmap (a grid of pixels).



99	71	61	51	49	40	35	53	86	99
93	74	53	56	48	46	48	72	85	102
101	69	57	53	54	52	64	82	88	101
107	82	64	63	59	60	81	90	93	100
114	93	76	69	72	85	94	99	95	99
117	108	94	92	97	101	100	108	105	99
116	114	109	106	105	108	108	102	107	110
115	113	109	114	111	111	113	108	111	115
110	113	111	109	106	108	110	115	120	122
103	107	106	108	109	114	120	124	124	132

Images (cont.)

- Input: scanned for photographs or pictures using a digital scanner or from a digital camera.
- Input: May also be generated by programs similar to graphics or animation programs.
- Analog sources will require digitising.
- Stored at 1 bit per pixel (Black and White), 8 Bits per pixel (Grey Scale, Colour Map) or 24 Bits per pixel (True Colour)
- Size: a 512x512 Grey scale image takes up 1/4 MB, a 512x512 24 bit image takes 3/4 MB with no compression.
- This overhead soon increases with image size — modern high digital camera 10+ Megapixels \approx 29MB uncompressed!
- Compression is commonly applied.

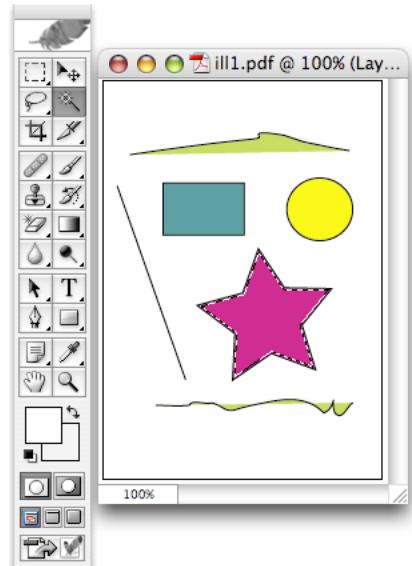
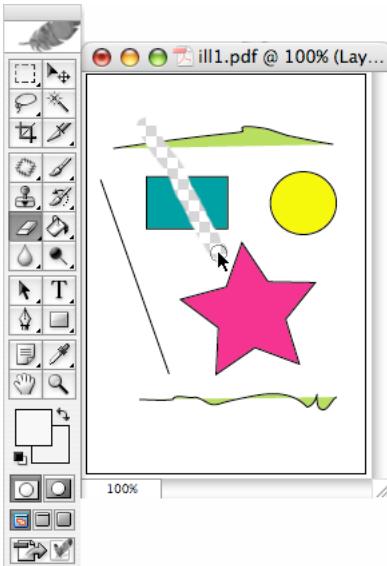
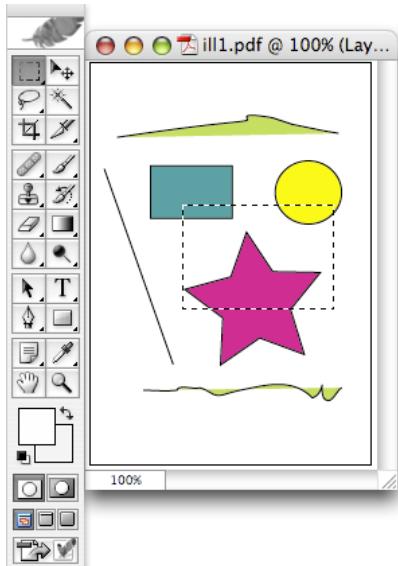


Back

Close

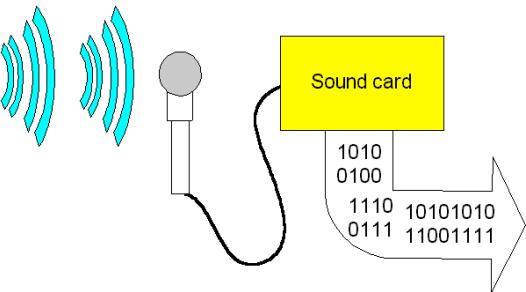
Images (cont.)

- Can usually only edit individual or groups of pixels in an image editing application, e.g. photoshop.



Audio

- Audio signals are continuous analog signals.
- Input: microphones and then digitised and stored



- CD Quality Audio requires 16-bit sampling at 44.1 KHz:
Even higher audiophile rates (e.g. 24-bit, 96 KHz)
- 1 Minute of Mono CD quality (uncompressed) audio = 5 MB.
Stereo CD quality (uncompressed) audio = 10 MB.
- Usually compressed (E.g. MP3, AAC, Flac, Ogg Vorbis)

Video

- Input: Analog Video is usually captured by a video camera and then digitised, although digital video cameras now essentially perform both tasks.
- There are a variety of video (analog and digital) formats (**more later**)
- Raw video can be regarded as being a series of single images. There are typically 25, 30 or 50 frames per second.



Click here or image above to run movie



Back

Close

Video (cont)

Video Size:

- A 512x512 size monochrome video images take

$$25 * 0.25 = 6.25\text{MB}$$

for a second to store uncompressed.

- Typical PAL digital video (720×576 pixels per colour frame)
 $\approx 1.2 \times 25 = 30\text{MB}$ for a second to store uncompressed.
- High Definition video on Blu-ray (up to $1920 \times 1080 = 2$ Megapixels per frame) $\approx 6 \times 25 = 150\text{MB}$ for a second to store uncompressed, i.e. 9GB for a minute to store uncompressed. (There are higher possible frame rates!)
- Digital video clearly needs to be compressed.

Multimedia Data Representation

Issues to be covered (Over next few lectures):

- Digital Audio
 - Digital Audio Synthesis
 - MIDI — Synthesis and Compression Control
 - Digital Audio Signal Processing/Audio Effects
- Graphics/Image Formats
 - Colour Representation/Human Colour Perception
- Digital Video
 - Chroma Subsampling

General Themes across all above

- Sampling/Digitisation
 - Sampling Artifacts — *Aliasing*
- Compression requirements
 - Data formats especially size
 - Human Perception → compression ideas

**Building up to full Multimedia
Compression Algorithms — following
lectures**



Digital Audio

What is Sound?

Multimedia
CM0340

72

Source — Generates Sound

- Air Pressure changes
- *Electrical* — Loud Speaker
- *Acoustic* — Direct Pressure Variations

Destination — Receives Sound

- *Electrical* — Microphone produces electric signal
- *Ears* — Responds to pressure **hear** sound (**more later**
(MPEG Audio))



Back

Close

Digitising Sound (Recap from CM0268)

- Microphone produces *analog* signal
- Computer like discrete entities

Need to convert Analog-to-Digital — Specialised Hardware

Also known as *Sampling*



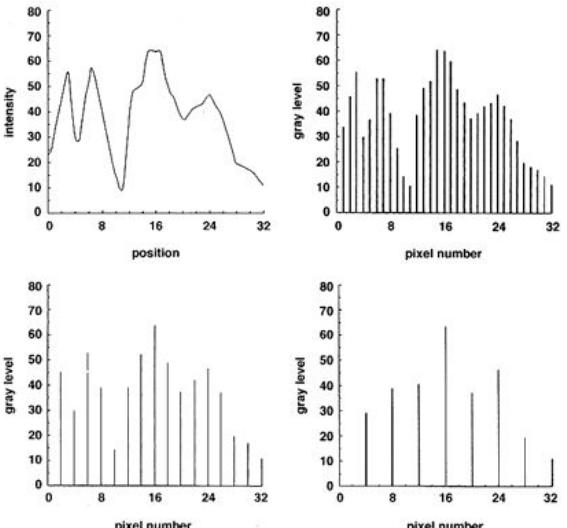
Back

Close

Digital Sampling

Sampling basically involves:

- Measuring the analog signal at regular discrete intervals
- Recording the value at these points



Computer Manipulation of Sound (Audio FX Preview)

Writing Digital Signal Processing routines range from being trivial to highly complex:

- Volume
- Cross-Fading
- Looping
- Echo/Reverb/Delay
- Filtering
- Signal Analysis



Back

Close

Sample Rates and Bit Size

How do we store each sample value (*Quantisation*)?

8 Bit Value (0-255)

16 Bit Value (Integer) (0-65535)

How many Samples to take?

11.025 KHz — Speech (Telephone 8 KHz)

22.05 KHz — Low Grade Audio
(WWW Audio, AM Radio)

44.1 KHz — CD Quality



Back

Close

Nyquist's Sampling Theorem

Sampling Frequency is Very Important in order to accurately reproduce a digital version of an Analog Waveform

Nyquist's Theorem:

The Sampling frequency for a signal must be **at least twice** the highest frequency component in the signal.



Back

Close

Common Audio Formats

- Popular audio file formats include
 - .au (*Origin: Unix, Sun*),
 - .aiff (*MAC, SGI*),
 - .wav (*PC, DEC*)
- Compression can be utilised in some of the above but is not **Mandatory**
- A simple and widely used (by above) audio compression method is Adaptive Delta Pulse Code Modulation (ADPCM).
 - Based on past samples, it predicts the next sample and encodes the difference between the actual value and the predicted value.
 - More on this later (Audio Compression)

Common Audio Formats (Cont.)

- Many formats linked to audio applications
- Most use some compression
- Common ones:
 - Sounblaster — .voc (Can use Silence Deletion ([More on this later \(Audio Compression\)](#)))
 - Protools/Sound Designer – .sd2
 - Realaudio — .ra.
 - Ogg Vorbis — .ogg
 - AAC , Apple, mp4 — [More Later](#)
 - Flac — .flac, [More Later](#)
 - Dolby AC coding — [More Later](#)
- **MPEG AUDIO** — [More Later \(MPEG-3 and MPEG-4\)](#)



Back

Close

Synthetic Sounds — reducing bandwidth?

- Synthesise sounds — hardware or software
- Client produces sound — only send parameters to control sound ([MIDI/MP4/HTML5 later](#))
- Many synthesis techniques could be used, For example:
 - FM (Frequency Modulation) Synthesis – used in low-end Sound Blaster cards, OPL-4 chip, Yamaha DX Synthesiser range popular in Early 1980's.
 - Wavetable synthesis – wavetable generated from sampled sound waves of real instruments
 - Additive synthesis — make up signal from smaller simpler waveforms
 - Subtractive synthesis — modify a (complex) waveform but taking out (Filtering) elements
 - Granular Synthesis — use small fragments of existing samples to make new sounds
 - Physical Modelling — model how acoustic sound is generated in software
 - Sample-based synthesis — record and play back recorded audio, often small fragments and audion processed.
- Most modern Synthesisers use a mixture of samples and synthesis.

Synthetic Sounds — Analogies with Vector Graphics

- Use more *high-level* descriptions to represent signals.
- Recorded sounds and digital images: regular sampling; large data size; difficult to modify
- Synthetic sounds and vector graphics: high level descriptions; small data size; easier to edit. **Conversion is needed before display – synthesis or rasterisation**
- Difference: 1D vs 2D

More on how synthesis works next



Back

Close

Digital Audio Synthesis + Effects

Some Practical Multimedia Digital Audio Applications

Multimedia
CM0340

82

Having considered the background theory to digital audio processing, let's consider some practical multimedia related examples:

- Digital Audio Synthesis — making some sounds
- Digital Audio Effects — changing sounds via some standard effects.
- MIDI — synthesis and effect control and **compression**



Back

Close

Digital Audio Synthesis

We have talked a lot about synthesising sounds.

Several Approaches:

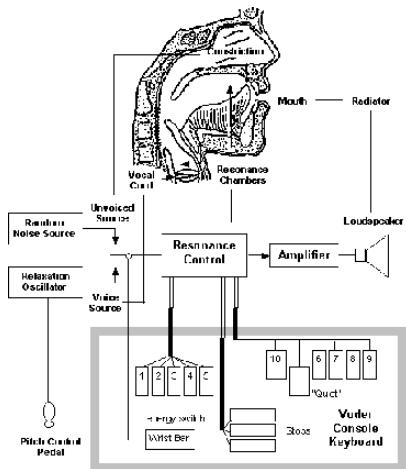
- Subtractive synthesis
- FM (Frequency Modulation) Synthesis
- Additive synthesis
- Sample-based synthesis
- Wavetable synthesis
- Granular Synthesis
- Physical Modelling

Subtractive Synthesis

Basic Idea: Subtractive synthesis is a method of subtracting overtones from a sound via sound synthesis, characterised by the application of an audio filter to an audio signal.

First Example: Vocoder — talking robot (1939).

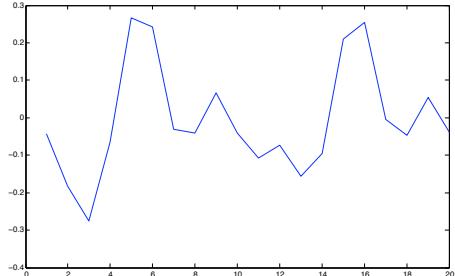
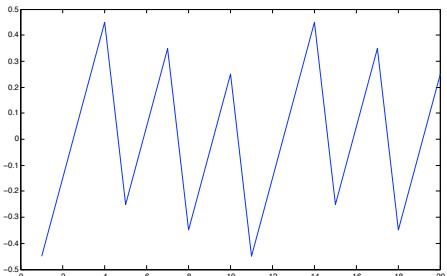
Popularised with Moog Synthesisers
1960-1970s



Subtractive synthesis: Simple Example

Simulating a bowed string

- Take the output of a sawtooth generator
- Use a low-pass filter to dampen its higher partials generates a more natural approximation of a bowed string instrument than using a sawtooth generator alone.



- [subtract_synth.m](#) MATLAB Code Example Here.



Back

Close

Subtractive Synthesis: A Human Example

We can regard the how humans make noises as subtractive synthesis:

Oscillator — the vocal cords act as the sound source and

Filter — the mouth and throat modify the sound.

- Saying or singing "ooh" and "aah" (at the same pitch.)
- Vocal chords are generating pretty much the same raw, rich in harmonic sound. Difference between the two comes from the filtering which we apply with the mouth and throat.
- Change of mouth shape varies the "cutoff frequency" of the filter, so removing (subtracting) some of the harmonics.
- The "aah" sound has most of the original harmonics still present,
- The "ooh" sound has most of them removed (or to be more precise, reduced in amplitude.)

Subtractive Synthesis: Another Human Example

A sweeping filter

"ooh"s to "aah"s again

- By gradually changing from "ooh" to "aah" and back again – simulate the "sweeping filter" effect
- Effect widely used in electronic music
- Basis of the "wahwah" guitar effect, so named for obvious reasons.
- We will see how we produce this effect in MATLAB code shortly.

Subtractive Synthesis: One More Human Example

Making Aeroplane Noise

Make a "ssh" sound — white noise

- Now "synthesise" a "jet plane landing" sound
- Should mostly by use mouth shape to filter the white noise into pink noise by removing the higher frequencies.
- The same technique (filtered white noise) can be used to electronically synthesise the sound of ocean waves and wind,
- Used in early drum machines to create snare drum and other percussion sounds.



Back

Close

Subtractive synthesis: Electronic Control

Three Basic elements:

Source signal : Common source signals on analog synths are square waves, pulse waves, sawtooth waves and triangle waves.

Modern digital and software synthesisers may include other, more complex waveforms or allow the user to upload arbitrary waveforms

Filtering : The cut-off frequency and resonance of the filter are controlled in order to simulate the natural timbre of a given instrument.

Amplitude Envelope : Further envelope control of signal amplitude (not subtractive synthesis but frequently used in practice). Also used with other synthesis techniques



Further Processing: ADSR Envelope

Basic Idea: Modulate some aspect of the instrument's sound often its volume over time.

Why is this needed? (used by many forms of synthesis):

When a mechanical musical instrument produces sound, the relative volume of the sound produced changes over time — The way that this varies is different from instrument to instrument

Examples:

Pipe Organ : when a key is pressed, it plays a note at constant volume; the sound dies quickly when the key is released.

Guitar : The sound of a guitar is loudest immediately after it is played, and fades with time.

Other instruments have their own characteristic volume patterns.

Also Note: While envelopes are most often applied to volume, they are also commonly used to control other sound elements, such as filter frequencies or oscillator pitches.



Back

Close

Further Processing: ADSR Envelope (Cont.)

Attack : How quickly the sound reaches full volume after the sound is activated (the key is pressed).

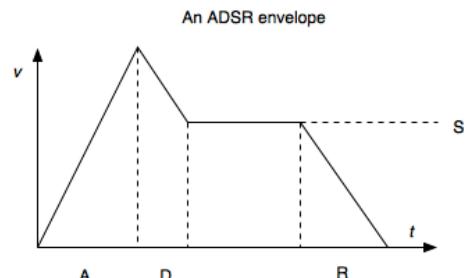
- For most mechanical instruments, this period is virtually instantaneous.
- For bowed strings or some popular synthesised "voices" that don't mimic real instruments, this parameter is slowed down. 'Slow attack' is commonly part of sounds — 'pads'.

Decay : How quickly the sound drops to the sustain level after the initial peak.

Sustain : The "constant" volume that the sound takes after decay until the note is released. Note that this parameter specifies a volume level rather than a time period.

Release How quickly the sound fades when a note ends (the key is released).

- Often, this time is very short. e.g. organ
- An example where the release is longer might be a bell ring, or a piano with the sustain pedal pressed.



Back

Close

Using MATLAB Filter Example: Subtractive Synthesis Example

The example for studying subtractive synthesis, [subtract_synth.m](#), uses the butter and filter MATLAB functions:

```
% simple low pas filter example of subtractive synthesis
Fs = 22050;
y = synth(440,2,0.9,22050,'saw');

% play sawtooth e.g. waveform
doit = input('\nPlay Raw Sawtooth? Y/[N]:\n\n', 's');
if doit == 'y',
    figure(1)
    plot(y(1:440));
    playsound(y,Fs);
end

%make lowpass filter and filter y
[B, A] = butter(1,0.04, 'low');
yf = filter(B,A,y);

[B, A] = butter(4,0.04, 'low');
yf2 = filter(B,A,y);

% play filtererd sawtooths
doit = ...
```



Back

Close

```
input('\nPlay Low Pass Filtered (Low order) ? Y/[N]:\n\n', 's');
if doit == 'y',
figure(2)
plot(yf(1:440));
playsound(yf,Fs);
end

doit = ...
input('\nPlay Low Pass Filtered (Higher order)? Y/[N]:\n\n', 's');
if doit == 'y',
    figure(3)
plot(yf2(1:440));
playsound(yf2,Fs);
end

%plot figures
doit = input('\nPlot All Figures? Y/[N]:\n\n', 's');
if doit == 'y',
figure(4)
plot(y(1:440));
hold on
plot(yf(1:440),'r+');
plot(yf2(1:440),'g-');
end
```



Back

Close

synth.m

The supporting function, **synth.m**, generates waveforms as we have seen earlier in this tutorial:

```
function y=synth(freq,dur,amp,Fs,type)
% y=synth(freq,dur,amp,Fs,type)
%
% Synthesize a single note
%
% Inputs:
% freq - frequency in Hz
% dur - duration in seconds
% amp - Amplitude in range [0,1]
% Fs - sampling frequency in Hz
% type - string to select synthesis type
%         current options: 'fm', 'sine', or 'saw'

if nargin<5
    error('Five arguments required for synth()');
end

N = floor(dur*Fs);
n=0:N-1;
if (strcmp(type,'sine'))
    y = amp.*sin(2*pi*n*freq/Fs);
```

```
elseif (strcmp(type,'saw'))  
  
T = (1/freq)*Fs; % period in fractional samples  
ramp = (0:(N-1))/T;  
y = ramp-fix(ramp);  
y = amp.*y;  
y = y - mean(y);  
  
elseif (strcmp(type,'fm'))  
  
t = 0:(1/Fs):dur;  
envel = interp1([0 dur/6 dur/3 dur/5 dur], [0 1 .75 .6 0], ...  
    0:(1/Fs):dur);  
I_env = 5.*envel;  
y = envel.*sin(2.*pi.*freq.*t + I_env.*sin(2.*pi.*freq.*t));  
  
else  
    error('Unknown synthesis type');  
end  
% smooth edges w/ 10ms ramp  
if (dur > .02)  
    L = 2*fix(.01*Fs)+1; % L odd  
    ramp = bartlett(L)'; % odd length  
    L = ceil(L/2);  
    y(1:L) = y(1:L) .* ramp(1:L);  
    y(end-L+1:end) = y(end-L+1:end) .* ramp(end-L+1:end);  
end
```

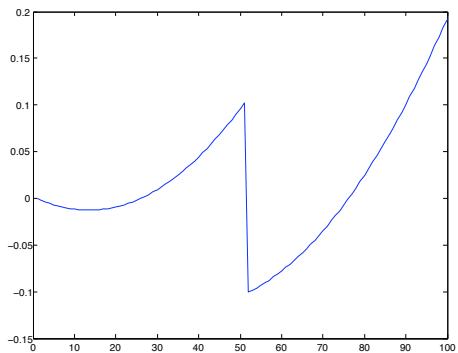


Back

Close

synth.m (Cont.)

Note the *sawtooth* waveform generated here has a non-linear up slope:



This is created with:

```
ramp = (0 : (N-1)) / T;  
y = ramp-fix(ramp);
```

`fix` rounds the elements of `X` to the nearest integers towards zero.

This form of sawtooth sounds slightly less harsh and is more suitable for audio synthesis purposes.

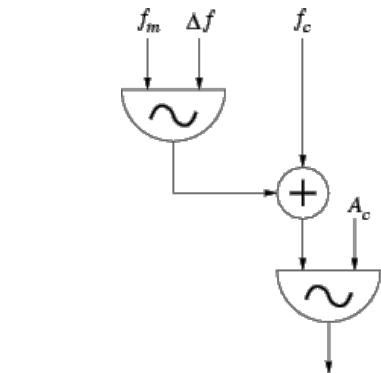
FM (Frequency Modulation) Synthesis

Basic Idea: Timbre of a simple waveform is changed by frequency modulating it with a frequency resulting in a more complex waveform — different-sounding.

Discovered by John Chowning at Stanford University in 1967-68,

Patented in 1975 and was later licensed to Yamaha.

Used in popular 1980s Yamaha Synthesisers: DX7, Casio CZ **still in use today**



FM (Frequency Modulation) Synthesis

(cont.)

- Radio broadcasts use FM in a different way
- FM synthesis is very good at creating both harmonic and inharmonic ('clang', 'twang' or 'bong' noises) sounds
 - For synthesizing harmonic sounds, the modulating signal must have a harmonic relationship to the original carrier signal.
 - As the amount of frequency modulation increases, the sound grows progressively more complex.
 - Through the use of modulators with frequencies that are non-integer multiples of the carrier signal (i.e., non harmonic), bell-like dissonant and percussive sounds can easily be created.

FM (Frequency Modulation) Synthesis

(cont.)

- Digital implementation — true analog oscillators difficult to use due to instability
- 1960s origin analog – FM discovered when vibrato sped up to the point that it was creating audible sidebands (perceived as a timbral change) rather than faster warbling (perceived as a frequency change).
- **DX synthesiser FM** - Where both oscillators use Sine waves and are "musically-tuned" frequencies generated from a keyboard



FM Synthesis: Mathematical Underpinnings Definitions

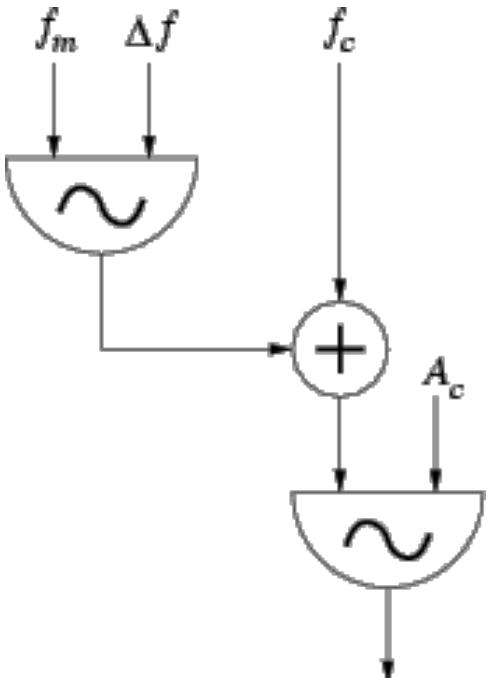
Oscillator : A device for generating waveforms

FM Frequency (Pitch) Modulation

: Where the frequency (pitch) of an oscillator (*the Carrier*) is modulated by another oscillator (*the Modulator*)

Carrier Frequency : The frequency of the oscillator which is being modulated

Modulator Frequency : The frequency of the oscillator which modulates the Carrier



FM Synthesis: Basic Frequency Modulation

Basic FM Equation:

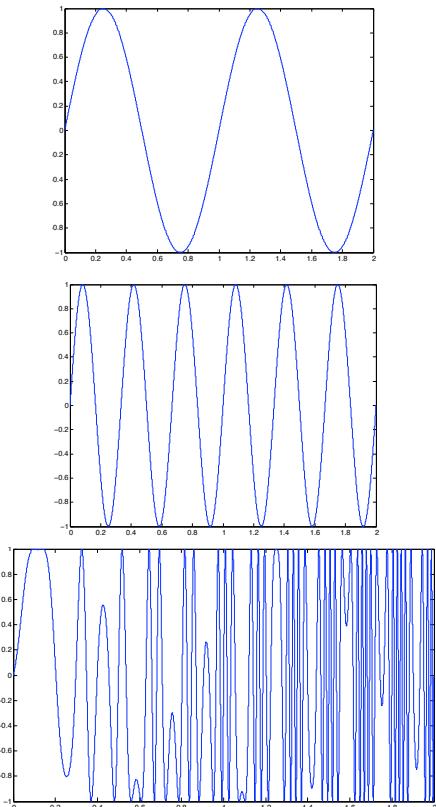
$$e = A \sin(\alpha t + I \sin \beta t)$$

A is the peak amplitude

e is the instantaneous amplitude of the modulated carrier

α and β are the respective carrier and modulator frequencies

I is the modulation index: the ratio of peak deviation to modulator frequency



MATLAB FM Example

MATLAB code to produce basic FM ([fm_eg.m](#),
see also [fm_eg_plot.m](#)):

```
% Signal parameters
fs = 22050;
T = 1/fs;
dur = 2.0;          % seconds
t = 0:T:dur;       % time vector

% FM parameters
fc = 440;          % center freq
fm = 30;
Imin = 0;
Imax = 20;
I = t.* (Imax - Imin)/dur + Imin;

y = sin(2*pi*fc*t + I.*sin(2*pi*fm*t));
plot(t(1:10000), y(1:10000));

playsound(y, fs);
```

FM Synthesis: Side Frequencies

The harmonic distribution of a simple sine wave signal modulated by another sine wave signal can be represented with Bessel functions

$$\begin{aligned} e = & A\{J_0 \sin \alpha t \\ & + J_1[\sin(\alpha + \beta)t - \sin(\alpha - \beta)t] \\ & + J_2[\sin(\alpha + 2\beta)t - \sin(\alpha - 2\beta)t] \\ & + J_3[\sin(\alpha + 3\beta)t - \sin(\alpha - 3\beta)t] \\ & \dots \} \end{aligned}$$

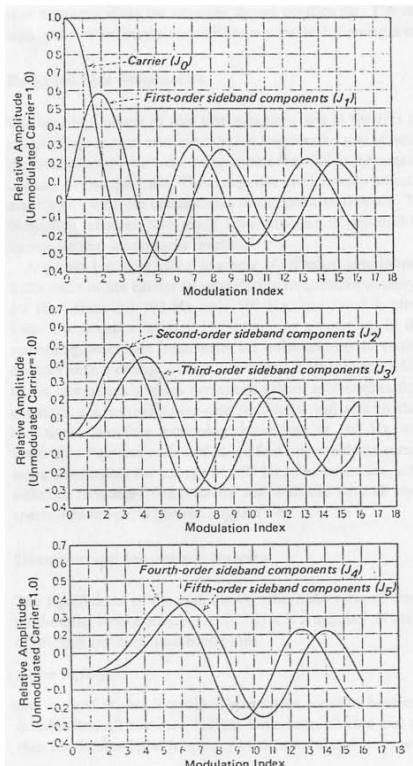
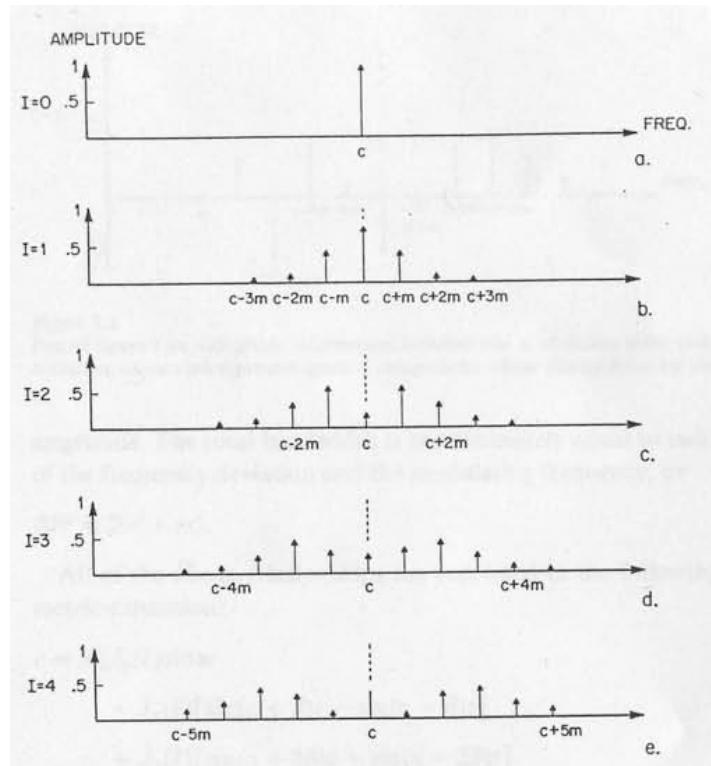
- Provides a basis for a simple mathematical understanding of FM synthesis.
- **Side Frequencies** produced and are related to modulation index, I
 - If $I > 1$ energy is *increasingly stolen* from the carrier but with constant modulation frequency.



Back

Close

FM Synthesis: Side Frequencies (Cont.)



FM Synthesis: Making Complex Sounds

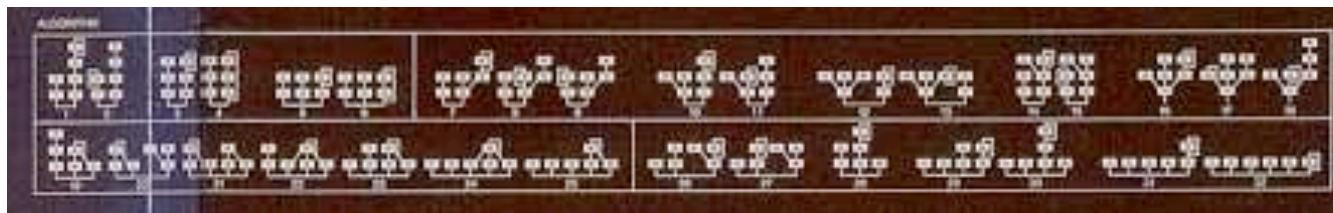
Operators and Algorithms

Operators are just Oscillators in FM Terminology.

- FM synths will have either 4 or 6 Operators.
- **Why so many Operators?**

Sounds from one Modulator and one Carrier aren't exactly that overwhelmingly complex

Algorithms are the preset combinations of routing available to you.



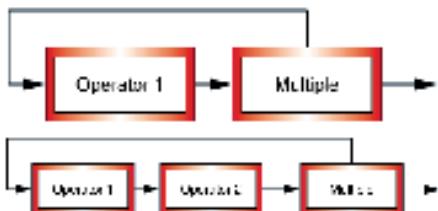
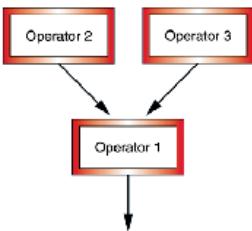
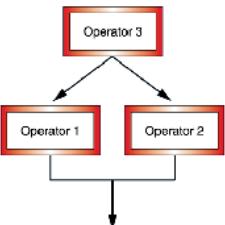
FM Synthesis: FM Algorithms

How to connect up Operators?

Multiple Carriers : One oscillator simultaneously modulates two or more carriers

Multiple Modulators : Two or more oscillators simultaneously modulate a single carrier

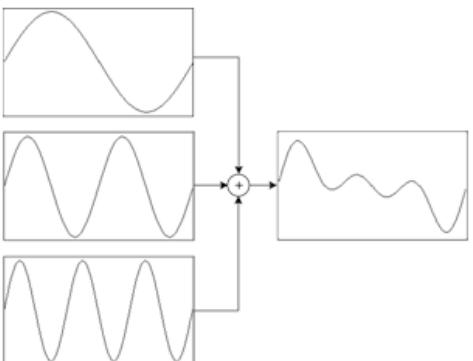
Feedback : Output of oscillator modulates the same oscillator



Additive synthesis

Basic Idea: Additive synthesis refers to the idea that complex tones can be created by the summation, or addition, of simpler ones.

- Frequency *mixing* is the essence of additive synthesis.
- Each of the frequency components (or partials) of a sound has its own amplitude envelope.
- This allows for independent behaviour of these components.
- Sources can be other forms of synthesis or samples.



Additive synthesis: Examples

Organs : Pipe organs or Hammond organs.

The concept of register-stops of organs = additive synthesis:

- complex timbres result from the addition of different components to the spectrum of a sound.
- Different pipe stops or tonewheel/drawbar settings



Telharmonium : An early giant electrical synthesiser (1900s):

- adds together the sounds from dozens of electro-mechanical tone generators to form complex tones.
- Important place in the history of electronic and computer music.

Modern Variants : Fairlight CMI, Synclavier, Kawai K5000 series, wavetable synthesis (more soon)

Additive synthesis: Basic Theory

- Basis: Fourier Theory
- Simply stated: a complex timbre that has been analysed into its sinusoidal components can then be reconstructed by means of additive synthesis.
- Additive synthesis has the advantage that the many micro-variations in the frequency and amplitude of individual partials, that make natural sounds so rich and lively, can be recreated.
- The disadvantage with this form of synthesis is its inefficiency in that a great deal of data must be specified to define a sound of any complexity of detail.
- Simple MATLAB Example: [additive_synth.m](#)



Back

Close

Sample-based synthesis

Basic Ideas: Similar to either subtractive synthesis or additive synthesis.

The **principal difference** is that the seed waveforms are sampled sounds or instruments instead of fundamental waveforms such as the saw waves of subtractive synthesis or the sine waves of additive synthesis.

Samplers, together with traditional Foley artists, are the mainstay of modern sound effects production.

Musical genres: Hip-hop, Trip-hop, Dance music, Garage, Jungle, Trance, Modern Electronica *invented* due to samplers.

Most music production now uses samplers.



Back

Close

Sample-based synthesis: Comparison with other Synthesis methods

- Advantages (over other methods of digital synthesis such as physical modelling synthesis (**more soon**) or additive synthesis): processing power requirements are much lower.
 - Nuances of the sound models are contained in the pre-recorded samples rather than calculated in real-time.
- Disadvantage: in order to include more detail, multiple samples might need to be played back at once
 - E.g. a trumpet might include a breath noise, a growl, and a looping soundwave used for continuous play
 - Reduces the polyphony as sample-based synthesizers rate their polyphony based on the number of multi-samples that can be played back simultaneously.



Back

Close

Sample-based synthesis: Examples

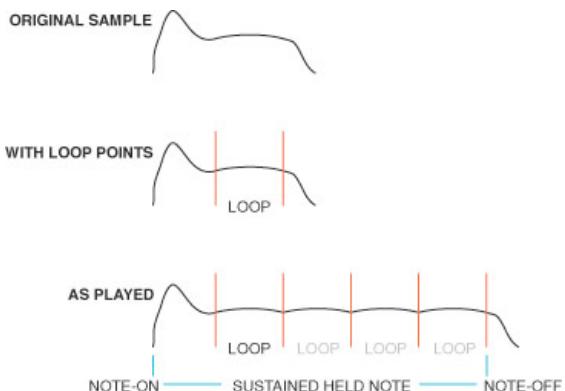
- Mellotron (analog tape) (1962)
- Computer Music Melodian (1976): Stevie Wonder's "Secret Life of Plants"
- CMI Fairlight (1979)
- NED Synclavier (1979).
- EMU Emulator series (1981)
- Akai S Series (1986)
- Korg M1 (1988): The M1 also introduced the "workstation" concept.
- Software Samplers (2005) : NI Kontakt, Steinberg Halion



Sample-based synthesis Basics: Looping

A sample-based synthesizer's ability to reproduce the nuances of natural instruments is determined primarily by its library of sampled sounds.

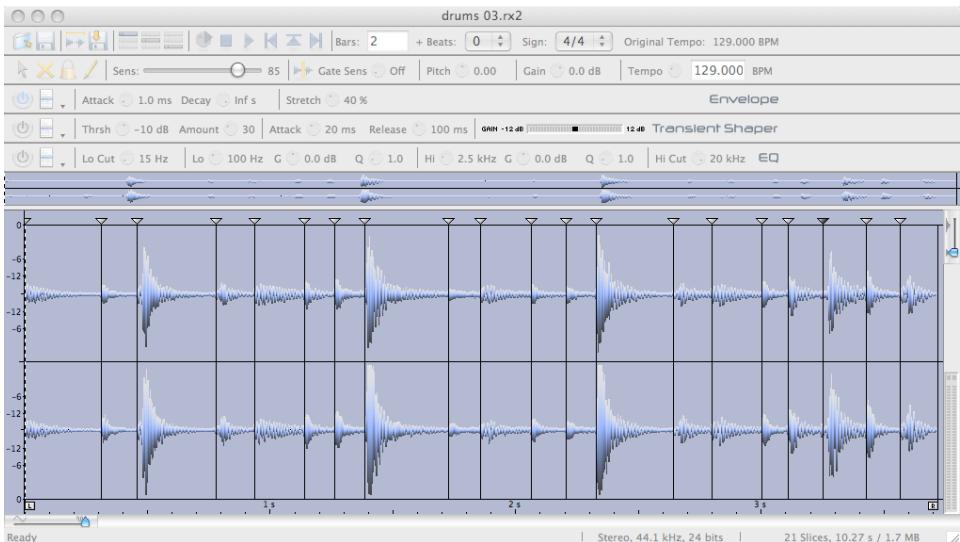
- Early days: computer memory expensive:
Samples had to be as **short** and as **few** as possible.
This was achieved by **looping a part of the sample**



- Problem: How to find looping points?

Finding looping points

- Simple idea: Find silence points (**zero (amplitude) crossings**) in sample.
E.g. Drum beats



Loop between these

- Alternative: Find portions in sample that have same audio content — pattern matching.
E.g. Sustaining musical instruments.

Sample-based synthesis Basics: Looping (cont)

Pitch control:

- Speed or slow up sample to change pitch (realism to only a few semitones in pitch change)
- Still need some sample across the range of the keyboard
- As memory became cheaper (and now with software based sample synthesis), it became possible to use **multisampling** — looping still used in individual samples.

Finishing off the loop:

- Early Days: Use a volume envelope curve to make the sound fade away.
- Today: Include tail off sample in data — triggered by note off.



Back

Close

Beat Slicing Algorithms Background: Altering Loops

Silence Points : Find silence points (**zero (amplitude) crossings**) in sample.

Snapping to silence points means that no nasty clicks in audio when joining audio together. (Too simple)

Beat Perception : *The human listening system determines the rhythm of music by detecting a pseudo periodical succession of beats*

- The more energy the sound transports, the louder the sound will seem.
- But a sound will be heard as a beat only if his energy is largely superior to the sounds energy history, that is to say if the brain detects a large variation in sound energy.
- Therefore if the ear intercepts a monotonous sound with sometimes big energy peaks it will detect beats,



Back

Close

Beat Slicing Algorithms Ideas (1):

Simple sound energy :

- Computing the **average** sound energy of the signal over a relatively large sample (around 1 second)
- Compute **instant** sound energy (around 5/100 second).
- Comparing average to the instant sound energy.
- We detect a beat only when the instant energy is larger than the local energy average.



Back

Close

Beat Slicing Algorithms Ideas (2):

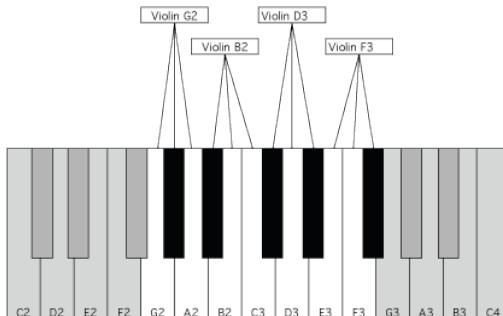
Frequency selected sound energy : More elaborate model:

Try to detect big sound energy variations in particular frequency subbands

- Apply Fourier Transform — separate beats according to their frequency sub-band.
- Apply energy analysis but in frequency space:
 - Compute Fourier Transform over 1024 samples.
 - Divide into around 32 sub-bands.
 - Compute the sound energy contained in each of the subbands
 - Compare it to the recent energy average corresponding to this subband.
 - If one or more subbands have an energy superior to their average we have detected a beat.
- For more details search web or see references at end of slides.

Sample-based Synthesis: Multisampling

- Non-pitched simple example: the concept of drum mapping — **see also general MIDI section later**
- **Need to preserve relationships between key notes**
- **Multisampling Basic Idea:**
 - Sample instrument at regular intervals to cover regions of several adjacent notes (*splits*) or for every note.
 - **Advantage:** provides a more natural progression from the lower to the higher registers



Sample-based Synthesis: Example Kontakt Sampler Multisample Keymap



◀◀

▶▶

◀

▶

Back

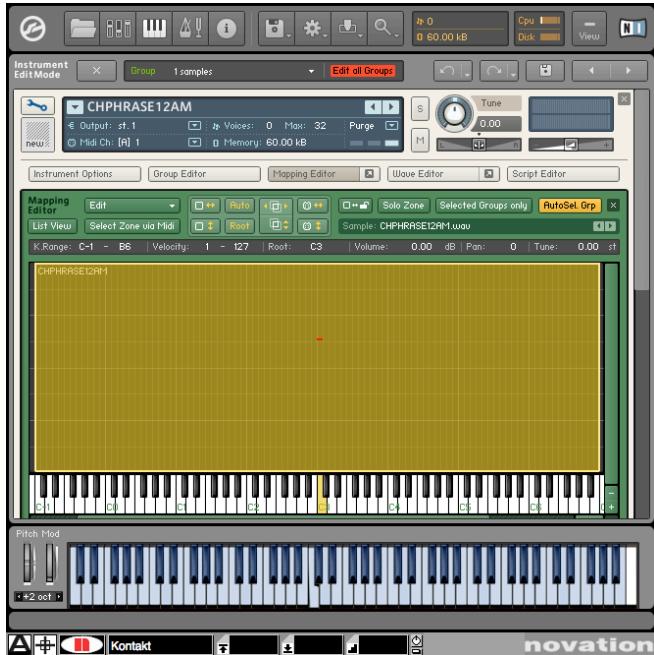
Close

Sample-based Synthesis: Velocity Layers

- When pluck a string or hit a drum or press a piano key, sound produced **depends** on how hard the action was.
- In software, this is measured by the velocity of a key press etc.
- Multisampling lays out samples vertically in keymap.
- **Velocity layers** layed out **horizontally**



Sample-based Synthesis: Example Velocity Layers (1)

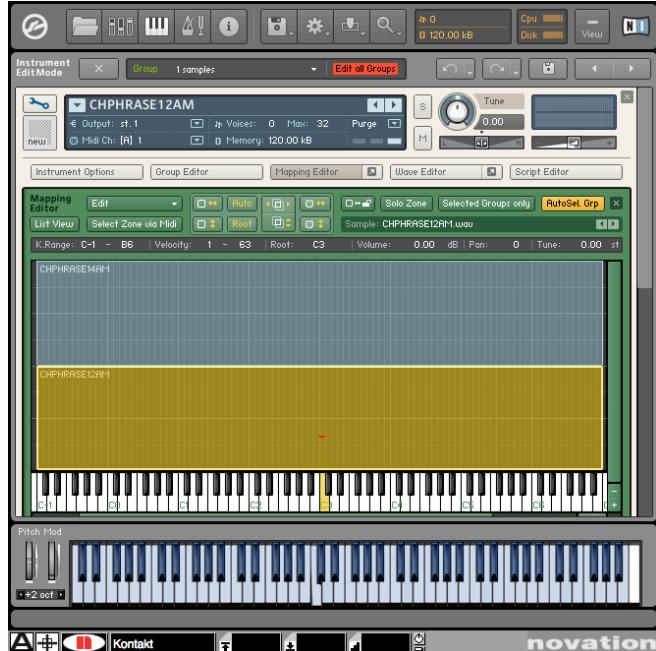


(Single key mapped) Single Velocity Layer — Only one type of sound played at any velocity.

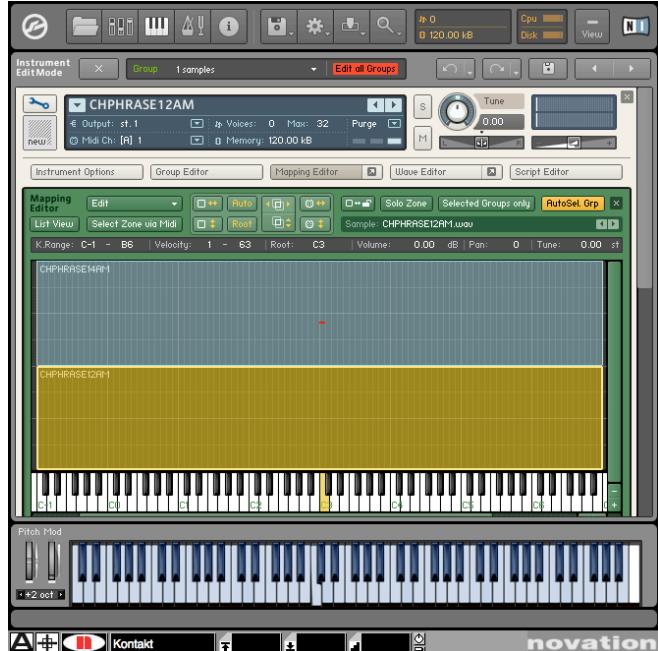
Volume output maybe controlled by velocity but not change in timbre of sound.

Sample-based Synthesis: Example Velocity Layers (2)

(Single key mapped) Dual Velocity Layer:

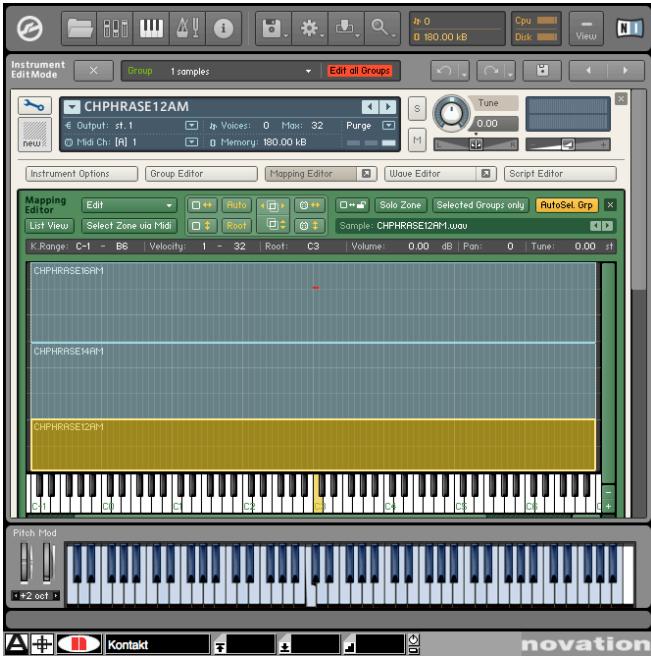


Sound one played at lower level velocity



Sound Two played at higher velocity

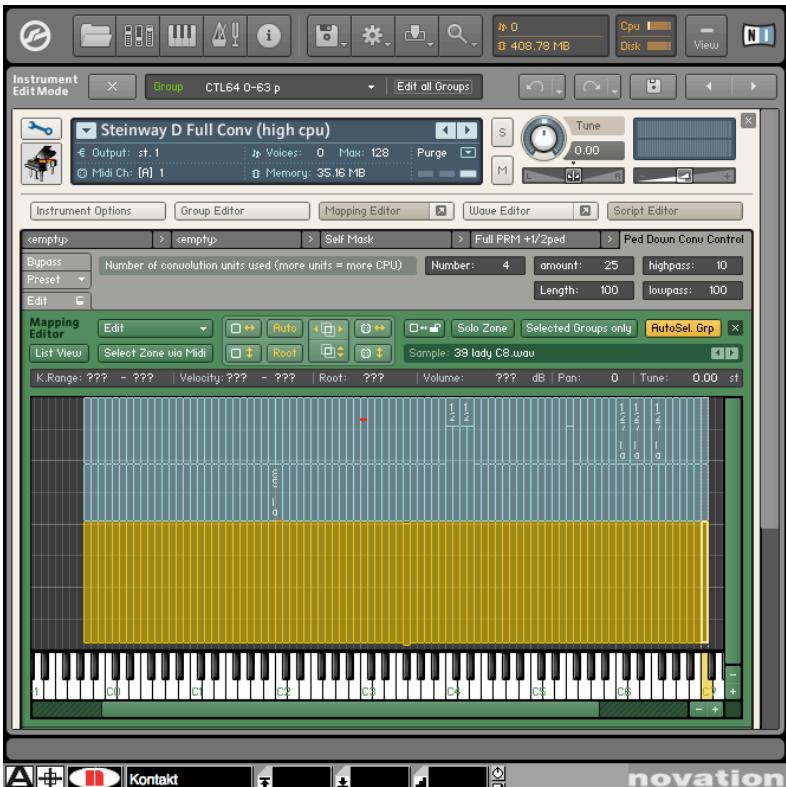
Sample-based Synthesis: Example Velocity Layers (3)



(Single key mapped) Triple Velocity Layer — Three type of sounds played according to velocity.

Here upper velocity sound is being played.

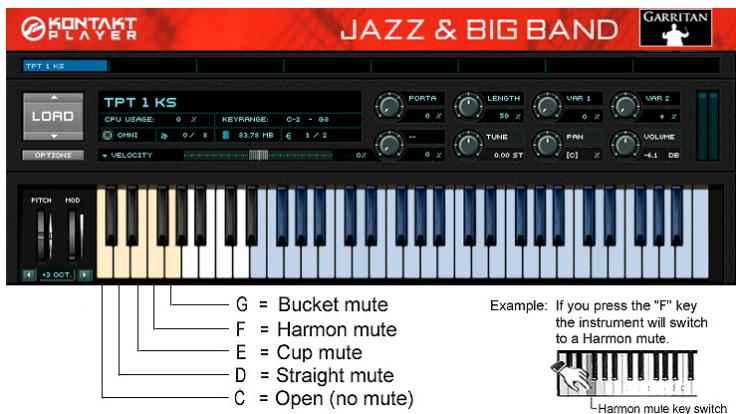
Sample-based Synthesis: Key Map and Velocity Layers



Most instruments are a combination of multisample key mapped and velocity layers

Sample-based synthesis Basics: Sample Keyswitching

- Instruments can make vastly different sounds depending how they are played
- Example: Trumpets (muted/not muted), violin (plucked, slow/fast up/down bow)
- For expressive performance samples can be **keyswitched**:
Use keys (usually lower keys outside of instrument range) to select appropriate sounds
 - Essentially banks of keymapped velocity layered samples



Advanced Software Based Sampling

- Sampling now seems to have very few limits
- Full orchestras and even choirs that can sing
- Can sing words too (Advanced Keyswitching).
- Programming script control over sampler (Kontakt 2 and above).



Wavetable synthesis

What is Wavetable music synthesis?

Similar to simple digital sine wave generation/additive synthesis but extended at least two ways.

- **Waveform lookup table** contains samples for not just a single period of a sine function but for a single period of a **more general waveshape**.
- Mechanisms exists for dynamically changing the waveshape as the musical note evolves:
thus generating a quasi-periodic function in time.

Not to be confused with common PCM sample buffer playback: soundcards



Back

Close

Wavetable synthesis: Examples

PPG Wave Series: Implementation of wavetable synthesis employed an array containing 64 pointers to individual single-cycle waves.

Waldorf Microwave: Next generation PPG.

Roland D-50 and Roland MT-32/variants:
 "Linear Arithmetic" synthesizers
 — combined complex sampled attack phases with less complex sustain/decay phases (basically a wavetable synthesizer with a 2-entry wave sequence table).

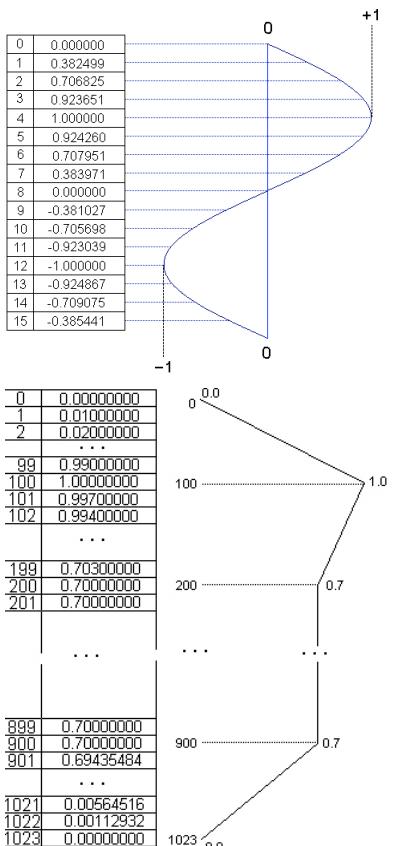
Sequential Circuits Prophet-VS,

Korg Wavestation: "Vector synthesis"
 — move through wavetables and sequences arranged on a 2-dimensional grid.



Wavetable Basics: Making Waves

- The sound of an existing instrument (a single note) is sampled and parsed into a circular sequence of samples or wavetables:
 - each having one period or cycle per wave;
 - A set of wavetables with user specified harmonic content can also be generated mathematically.
- At playback, these wavetables are used to fetch samples (table-lookup)
- However** the output waveform is not normally static and evolves slowly in time as one wavetable is mixed with another, creating a changing waveform via [ADSR Enveloping](#).
- Looping maybe used to slow, reverse wavetable evolution



Wavetable Basics: Practicalities

Put more simply, a wavetable synthesiser will store **two parts** of an instrument's sound.

- A sample of the attack section (e.g. the sound of the hammer hitting a piano string)
- A small segment of the sustain portion of the instrument's sound.

When triggered:

- The **attack** sample is played once immediately followed by a loop of the sustain segment.
- The endlessly looping segment is then enveloped to create a natural sounding decay (dying away of the sound).



Back

Close

Wavetable v. Sample Playback

- **Differs** from simple sample playback as
 - Output waveform is always generated in real time as the CPU processes the wave sequences
 - Waves in the tables are rarely more than 1 or 2 periods in length.



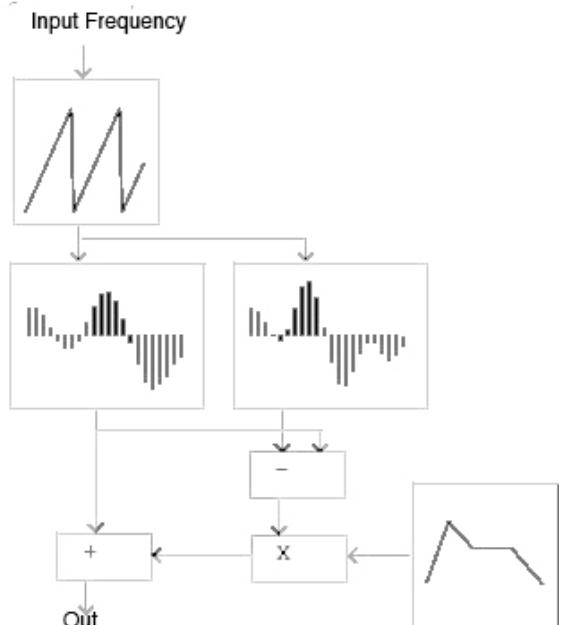
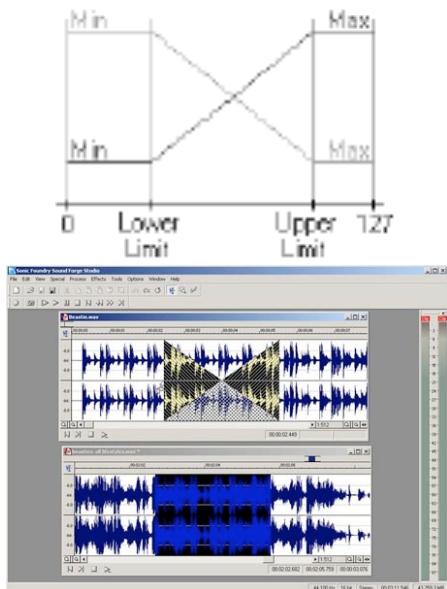
Back

Close

Wavetable synthesis: Dynamic Waveshaping (1)

Simplest idea: Linear crossfading

- Crossfade from one wavetable to the next sequentially.
- Crossfade = apply some envelope to smoothly merge waveforms.



Wavetable Synthesis Examples

Simple example — create one sine wave and one saw and then some simple cross-fading between the waves: [wavetable_synth.m](#)

```
% simple example of Wavetable synthesis

f1 = 440; f2 = 500; f3 = 620;
Fs = 22050;

%Create a single sine waves of frequencie f1
y1 = synth(f1,1/f1,0.9,Fs,'sine');

doit = input('\nPlay/Plot Raw Sine y1 looped for 10 ...
seconds? Y/[N]:\n\n', 's');
if doit == 'y',
figure(1)
plot(y1);
loopsound(y1,Fs,10*Fs/f1);
end

%Create a single Saw waves of frequencie f2
y2 = synth(f2,1/f2,0.9,Fs,'saw');

doit = input('\nPlay/Plot Raw saw y2 looped for 10 ...
```



Back

Close

```
seconds? Y/[N]:\n\n', 's');
if doit == 'y',
figure(2)
plot(y2);
loopsound(y2,Fs,10*Fs/f2);
end

%concatenate wave
ywave = [y1 , y2];

% Create Cross fade half width of wave y1 for xfade window
xfadewidth = floor(Fs/(f1*2));
ramp1 = (0:xfadewidth)/xfadewidth;
ramp2 = 1 - ramp1;

doit = input('\nShow Crossfade Y/[N]:\n\n', 's');
if doit == 'y',
figure(4)
plot(ramp1);
hold on;
plot(ramp2,'r');
end;

%apply crossfade centered over the join of y1 and y2
pad = (Fs/f1) + (Fs/f2) - 2.5*xfadewidth;
xramp1 = [ones(1,1.5*xfadewidth) , ramp2, zeros(1,pad)];
xramp2 = [zeros(1,1.5*xfadewidth) , ramp1, ones(1,pad)];
```



Back

Close

```
% Create two period waveforms to fade between
ywave2 = [y1 , zeros(1,Fs/f2)];;
ytemp = [zeros(1,Fs/f1), y2];

ywave = ywave2;
% do xfade

% add two waves together over the period modulated by xfade ramps
% (recall .* to multiply matrices element by element
% NOT MATRIX multiplication

ywave2 = xramp1.*ywave2 + xramp2.*ytemp;

doit = input ('\nPlay/Plot Additive Sines together? Y/[N]:\n\n', 's');
if doit == 'y',
figure(5)

subplot(4,1,1);
plot(ywave);

hold off

set(gca,'fontsize',18);
ylabel('Amplitude');
title('Wave 1');
set(gca,'fontsize',18);
```



Back

Close

```
subplot(4,1,2);
plot(ytemp);
set(gca,'fontsize',18);
ylabel('Amplitude');
title('Wave 2');
set(gca,'fontsize',18);
subplot(4,1,3);
plot(xramp1);
hold on
plot(xramp2,'r')
hold off
set(gca,'fontsize',18);
ylabel('Amplitude');
title('Crossfade Masks');
set(gca,'fontsize',18);
subplot(4,1,4);
plot(ywave2);
set(gca,'fontsize',18);
ylabel('Amplitude');
title('WaveTable Synthesis');
set(gca,'fontsize',18);
loopsound(ywave2,Fs,10*Fs/(f1 + f2));
end
```



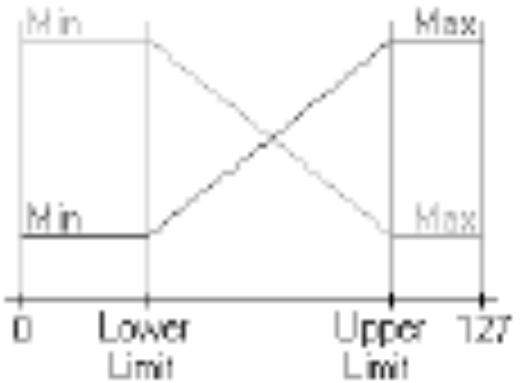
Back

Close

Wavetable synthesis: Dynamic Waveshaping (1)

Simplest idea: [Linear crossfading](#)

- Crossfade from one wavetable to the next sequentially.
- Crossfade = apply some envelope to smoothly merge waveforms.



Simple MATLAB Example: Linear Crossfading

Simple MATLAB wavetable/crossfading example:

wavetable_synth.m:

```
% Create Cross fade half width of wave y1 for xfade window
xfadewidth = floor(Fs/(f1*2));
ramp1 = (0:xfadewidth)/xfadewidth;
ramp2 = 1 - ramp1;

%apply crossfade centered over the join of y1 and y2
pad = (Fs/f1) + (Fs/f2) - 2.5*xfadewidth;
xramp1 = [ones(1,1.5*xfadewidth) , ramp2, zeros(1,pad)];
xramp2 = [zeros(1,1.5*xfadewidth) , ramp1, ones(1,pad)];
% Create two period waveforms to fade between
ywave2 = [y1 , zeros(1,Fs/f2)];
ytemp = [zeros(1,Fs/f1), y2];

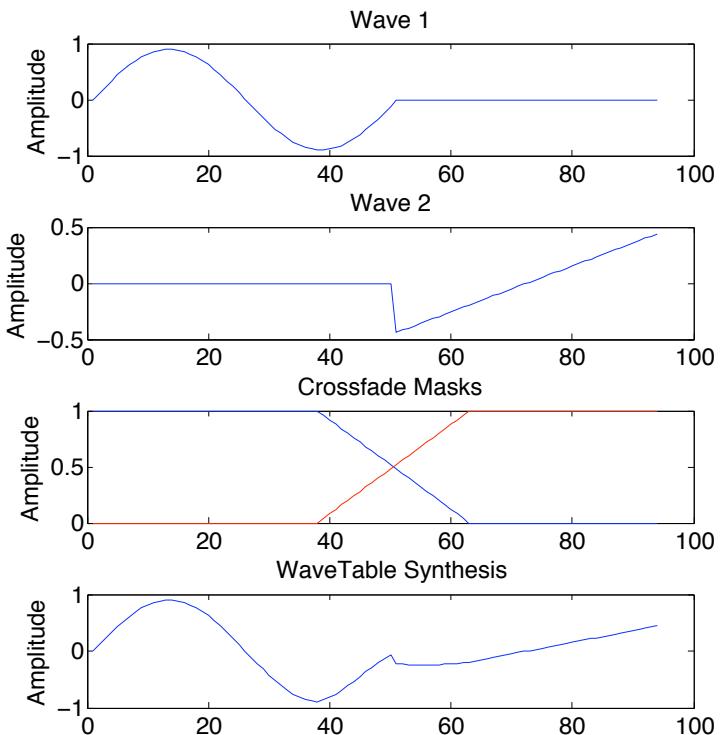
% do xfade
% add two waves together over the period modulated by xfade ramps
ywave2 = xramp1.*ywave2 + xramp2.*ytemp;
```



Back

Close

Simple MATLAB Example: Linear Crossfading (Cont.)

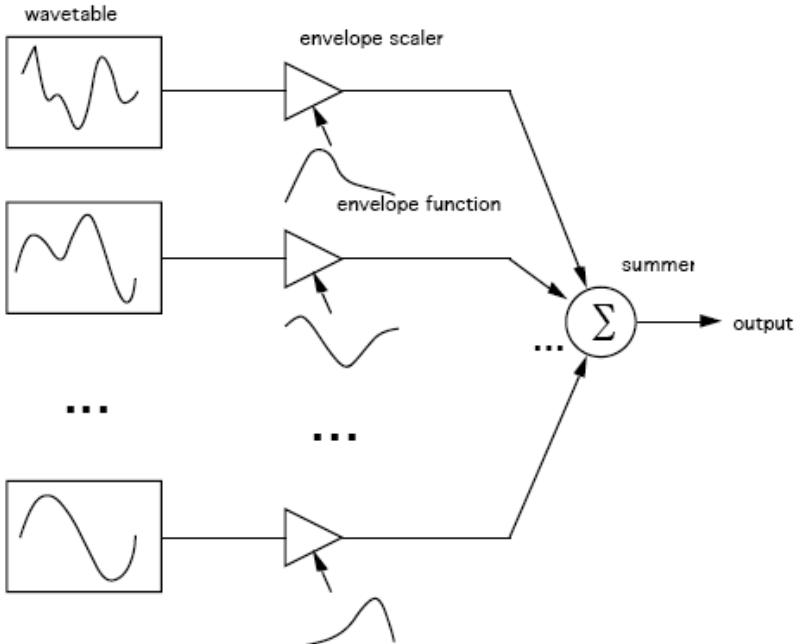


Note: This sort of technique is useful to create an ADSR envelope in MATLAB

Wavetable synthesis: Dynamic Waveshaping (2)

More sophisticated method: **Sequential Enveloping**

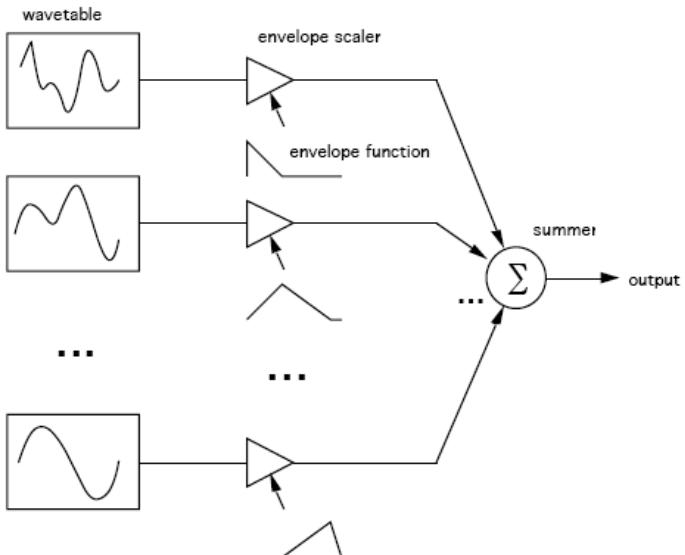
- Example below: two wavetables are being mixed at any one instance of time by moving envelope scale



Wavetable synthesis: Dynamic Waveshaping (3)

Linear Crossfading as Sequential Enveloping?

- The simple linear crossfading method can be thought of as a subclass of the more general basis mixing method where the envelopes are overlapping triangular pulse functions.



Wavetable synthesis: Advantages

- Well suited for synthesizing quasi-periodic musical tones because wavetable synthesis can be as compact in storage requirements
 - Amount of data being stored and used for this synthesis method is far less than just the PCM sample of same sound.
 - As general as additive synthesis but requires much less real-time computation.
- Wavetable synthesis takes advantage of the quasiperiodic nature of the waveform to remove redundancies and to reduce the data.
A bit more technical:
 - Precomputes the inverse Discrete Fourier Transform (DFT) of the waveform spectrum before playback
 - Rather than computing the inverse DFT in real-time as additive synthesis does.
 - Precomputed, real-time synthesis is reasonably simple to implement.



Back

Close

Granular Synthesis

"All sound is an integration of grains, of elementary sonic particles, of sonic quanta." -Iannis Xenakis, Greek Composer (1971).

Granular synthesis:

- Sound synthesis method that operates on the microsound time scale.
- Based on the same principles as sampling/wavetable synthesis but often includes analog technology as well.
- **Difference** Samples are not used directly to make usual sounds:
 - Split in small pieces of around 1 to 50 ms (milliseconds) in length, **the grains**.
 - Multiple grains may be layered on top of each other all playing at different speed, phase and volume.



Back

Close

Granular Synthesis: Soundscape

Result is no single tone, but a soundscape!

- Often a cloud, that is subject to manipulation
- Unlike any natural sound and also unlike the sounds produced by most other synthesis techniques.
- By varying the waveform, envelope, duration, spatial position, and density of the grains many different sounds can be produced.

Granular Synthesis: Is this music?

- Usable as music or soundscapes (ambient)
- Usable as Sound effects
- Usable to alter sample speed while preserving the original pitch/tempo information —**pitch/tempo synchronous granular synthesis**
- Usable as Raw material for further processing by other synthesis or DSP effects.
- The range of effects that can be produced include amplitude modulation, time stretching, stereo or multichannel scattering, random reordering, disintegration and morphing.

Granular Synthesis: Background

Strong Physics Background:

- Quantum physics has shown that sound can be atomically reduced to physical particles
- Physical form of sound was first envisioned by the Dutch scientist Isaac Beeckman (1618):
"Sound travels through the air as globules of sonic data."
- Denis Gabor (1947) proposed the idea of a grain as the quantum of sound and more recently
- Xenakis (1971) first musical use of granular synthesis — a reel to reel tape recorder, a razor blade, sticky tape, and a lot of time.
- Curtis Roads (1988), digital granular synthesis
- Barry Truax (1990) real-time granular synthesis composition
Riverrun, Buy the CD!



Back

Close

Granular Synthesis: Implementations

- Software:

Many implementations nowadays:

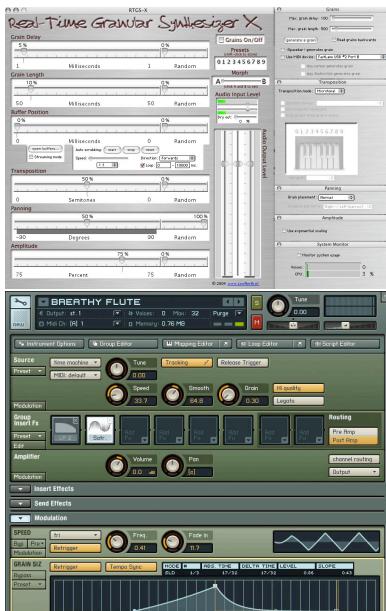
Programmable : Csound,
MATLAB,
routines:

Standalone : SuperCollider,
Granulab, RTGS X.

DAW plug-ins standalone :
VSTis etc.

Modern Music Samplers :
Native Instruments' Kontakt,
Intakt...., others

- Hardware: Korg Kaos Pad.



Granular Synthesis: What is a grain?

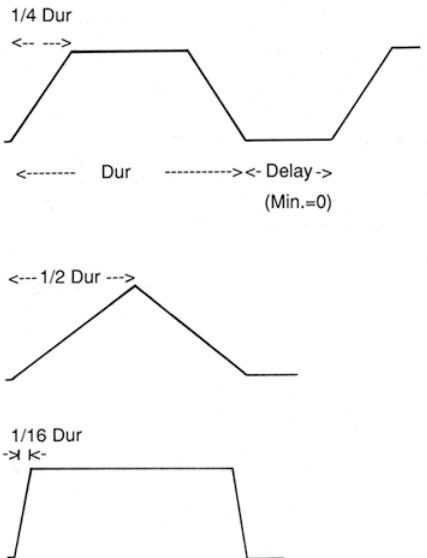
- A grain is a **small piece of sonic data**
- Usually have a duration \approx 10 to 50 ms.
- The grain can be broken down into smaller components:

Envelope — used so no distortion and crunching noises at the beginning and end of the sample.

The shape of the envelope has a significant effect on the grain though.

- For a sampled sound, a short linear attack and decay prevent clicks being added to the sound.
- Changing the slope of the grain envelope Changes the resulting spectrum, Sharper attacks producing broader bandwidths, just as with very short grain durations.

Contents — The audio: derived from any source, basic waves or samples



GRAIN ENVELOPES

Granular Synthesis: Making Sounds

Sounds made by the generation of thousands of short sonic grains:

- Combined linearly to form large scale audio events,
- 3 Possible combinations:

Quasi-synchronous granular synthesis

Asynchronous granular synthesis

Pitch/Tempo-synchronous granular synthesis — Preserve
Pitch/Tempo whilst altering sample playback speed
E.g. Intakt, Kontakt.

- The characteristics of the grains are definable which combine to give the characteristics of the overall sound.



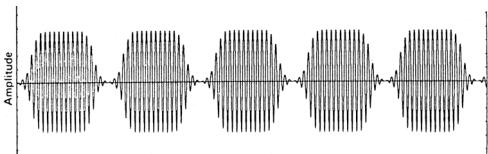
Back

Close

Granular Synthesis: Making Sounds (Cont.)

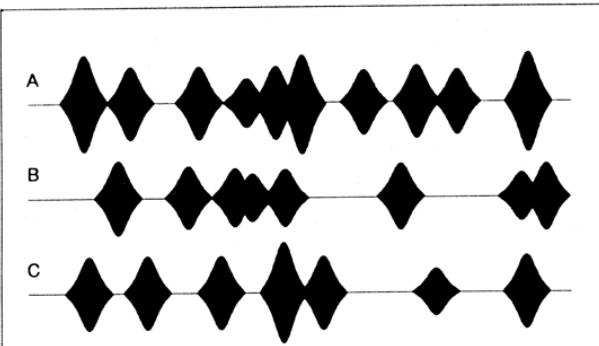
Quasi-synchronous granular synthesis:

- A grain stream of equal duration grains, produces Amplitude Modulation with grain durations less than 50 ms.

Multimedia
CM0340

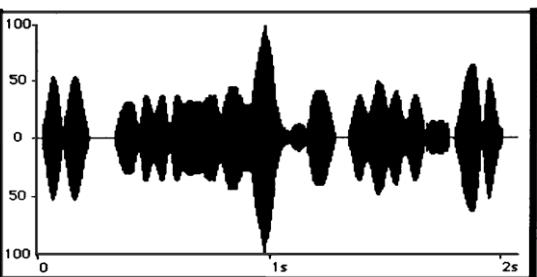
151

- Grain streams with variable delay time between grains — the sum of which resembles asynchronous granular synthesis.

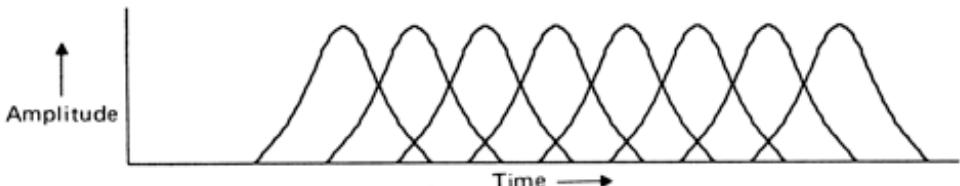


Granular Synthesis: Making Sounds (Cont.)

Asynchronous granular synthesis: Grains are distributed stochastically with no quasi regularity.



Pitch-synchronous granular synthesis: Overlapping grain envelopes designed to be synchronous with the frequency of the grain waveform, thereby producing fewer audio artifacts.



MATLAB Granular Synthesis Example

Simple MATLAB Example: granulation.m

```
% granulation.m
[filename,path] = uigetfile({'*.wav;*.waV;', 'Wav Files'; ...
    '.*', 'All files (*.*)'}, ...
    'Select a sound file');
if isequal(filename,0) | isequal(path,0)
cd(savedir);
return;
end
filenamepath = [path filename];
[x, fs] = wavread(filenamepath);

figure(1)
plot(x);

doit = input('\nPlay Original Wav file? Y/[N]:\n\n', 's');

if doit == 'y',
    playsound(x,fs);
end
```

MATLAB Granular Synthesis Example (Cont.)

```
Ly=length(x);    y=zeros(Ly,1);                      %output signal  
  
timex = Ly/fs;  
  
% Constants  
nEv=400; maxL=fs*0.02;  minL=fs*0.01;  Lw=fs*0.01;  
% Initializations  
L = round((maxL-minL)*rand(nEv,1))+minL;      %grain length  
initIn = ceil((Ly-maxL)*rand(nEv,1));           %init grain  
initOut= ceil((Ly-maxL)*rand(nEv,1));           %init out grain  
a = rand(nEv,1);                                %ampl. grain  
endOut=initOut+L-1;  
% Do Granular Synthesis  
for k=1:nEv,  
    grain=grainLn(x,initIn(k),L(k),Lw);  
    figure(2)  
    plot(grain);  
    y(initOut(k):endOut(k))=y(initOut(k):endOut(k))+ grain;  
end  
  
figure(3)  
plot(y)  
  
doit = input('\nPlay Granular Synthesised Wave? Y/[N]:\n\n', 's');  
if doit == 'y',  
    playsound(y,fs);  
end
```

Physical Modelling

Physical modelling synthesis is the **synthesis of sound by using a mathematical model — set of equations and algorithms to simulate a physical source of sound.**

- Sound is then generated using parameters that describe the physical materials used in the instrument and the user's interaction with it,
- For example, by plucking/bowing a string, or covering toneholes on a flute, clarinet etc.
- For example, to model the sound of a drum, there would be a formula for how striking the drumhead injects energy into a two dimensional membrane.



Back

Close

Physical Modelling: Examples

Hardware : Yamaha VL1 (1994), Roland COSM, Many since.

Software : Arturia Moog, Saxlab, Arturia Brass, PianoTeq

Examples of physical modelling algorithms:

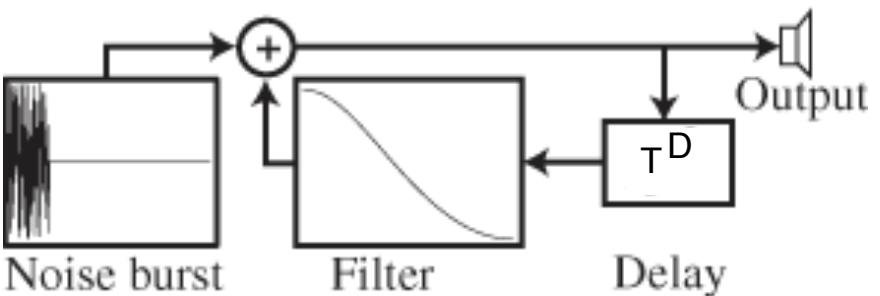
- Karplus-Strong strong synthesis (1971)
- Digital waveguide synthesis (1980s)
- Formant synthesis (1950s)



Physical Modelling: Karplus-Strong Algorithm Basics

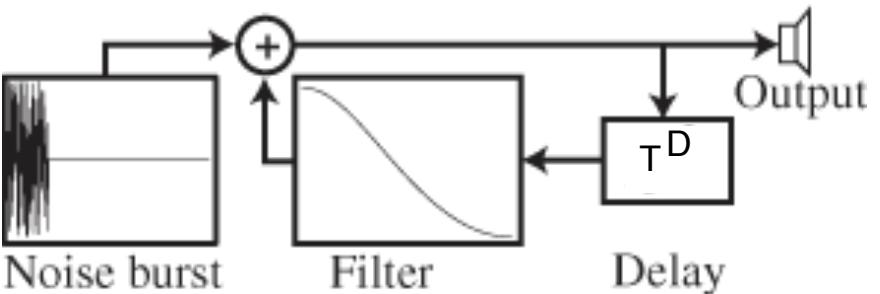
Simple Algorithm: *Makes a musical sound from noise*

- Loops a short noise burst through a filtered delay line to simulate the sound of a hammered or plucked string or some types of percussion.



- Feedback, Filtering and delay.
- Essentially subtractive synthesis technique based on a feedback loop similar to that of a comb filter.

Physical Modelling: Karplus-Strong Algorithm More Details



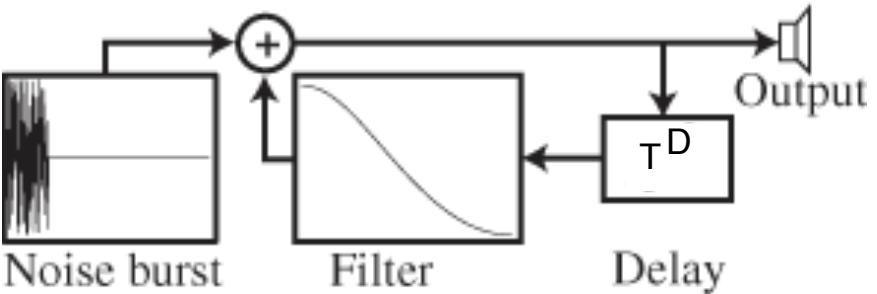
- Input: A burst of white noise, L samples long, (can use other signal).
- Output signal and feedback into a delay line.
- Output of the delay line is fed through a filter -gain of the filter must be less than 1 at all frequencies, usually a first order lowpass filter
- Filtered output is simultaneously mixed back into the output and fed back into the delay line.

Physical Modelling: Karplus-Strong Algorithm Tuning

- Period of the resulting signal is the period of the delay line plus the average group delay of the filter;
- **Fundamental frequency** is the reciprocal of the period.
- Required delay D for a given fundamental frequency F_1 is therefore calculated as:

$$D = \frac{F_s}{F_1}$$

where F_s is the sampling frequency.



Physical Modelling: Karplus-Strong Algorithm MATLAB Example

MATLAB Karplus-Strong Algorithm: [karplus.m](#) ([Source](#))

```
% ***** Constants and Other Parameters *****
fs = 44100; % sampling rate
N = 80000; % length of vector to compute
D = 200; % delay line (or wavetable) length

% ***** Simple String Attenuation Filter *****
b = -0.99*[0.5 0.5];
z = 0;

% ***** Initialize delay lines *****
y = zeros(1,N); % initialize output vector
dline = 2 * rand(1, D) - 1.0;
ptr = 1;

figure(1); subplot(3,1,1);plot(dline);set(gca,'fontsize',18);
title('Original delayline');

subplot(3,1,2);plot(dline);set(gca,'fontsize',18);
title('Filter delayline step n');

loopsound(dline,fs,fs/D);

subplot(3,1,3); plot(y); title('Waveform Step n');set(gca,'fontsize',18);
```



Back

Close

```
figure(1);
% ***** Run Loop Start *****
for n = 1:N,
    y(n) = dline(ptr);
    [dline(ptr), z] = filter(b, 1, y(n), z);
    % Increment Pointers & Check Limits
    ptr = ptr + 1;
    if ptr > D
        ptr = 1;
    end
    if mod(n,2000) == 0
        subplot(3,1,2);plot(dline)
        str = sprintf('Filter delayline step %d',n);
        title(str);
        subplot(3,1,3); plot(y);
        str = sprintf('Waveform Step %d',n);
        title(str);
        figure(1);
    end
end

% Scale soundfile if necessary
max(abs(y))
if max(abs(y)) > 0.95
    y = y./ (max(abs(y))+0.1);
    disp('Scaled waveform');
end

figure(2);clf;plot(y); title('Final Step');set(gca,'fontsize',18);
playsound(y',fs);
```



Back

Close

MIDI

What is MIDI?

- **No Longer** Exclusively the Domain of Musicians.
- Midi provides a very low bandwidth alternative on the Web:
 - transmit musical and
 - certain sound effects data
- also now used as a *compression control language (modified)*
 - See MPEG-4 Section soon

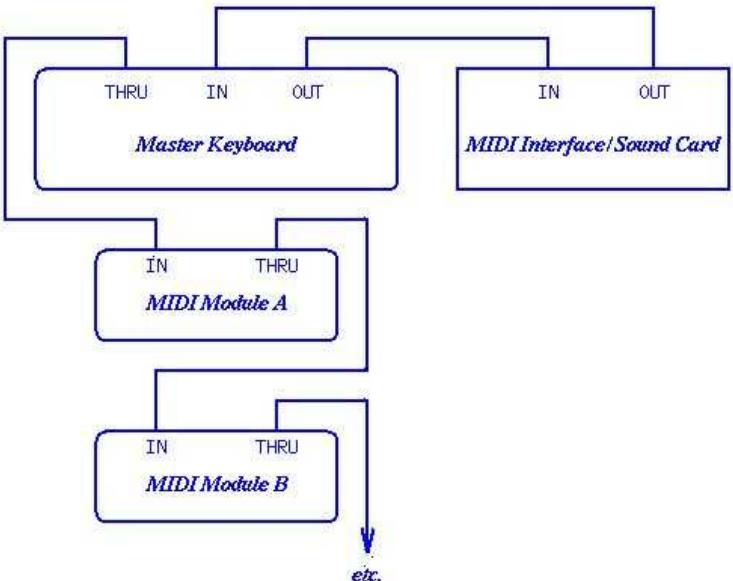
MIDI on the Web

Very Low Bandwidth (few 100K bytes)

- The responsibility of producing sound is moved to the client:
 - Synthesiser Module
 - Sampler
 - Soundcard
 - Software Generated
- Most Web browsers can deal with MIDI.
 - Available as plugins (e.g. [Quicktime](#)) and (as of 2013) as [Web MIDI API](#) in [HTML 5](#) — ([More Soon](#))

Definition of MIDI:

A protocol that enables computers, synthesisers, keyboards, and other musical devices to communicate with each other.



Brief History of MIDI:

Midi is now 30 Years old (as of 2012/3).

Multimedia
CM0340

165

However MIDI is still very much alive and kicking.

- Old meets new: iPad plays old Commodore Sequencer!
- Brief History: [BBC News Web Article](#)
- The protocol is still evolving: [High Definition MIDI](#) in Pipeline (2013). ([More soon](#))

Not bad for a 30 Year Old Hi-Tec Media Protocol!



Back

Close

Components of a MIDI System

Synthesiser/Sampler:

- It is a sound generator (various pitch, loudness, tone colour)
- Can use a variety of synthesis or Sample-based synthesis to make sound.
- A good (musician's) synthesiser often has a microprocessor, keyboard, control panels, memory, etc.
- For our purposes we define a synthesiser as the **tone generation unit**.
- It has one or more MIDI INs and MIDI OUTs and/or USB/Firewire connectivity
- Can be software based these days so virtual midi connections.



Sequencer:

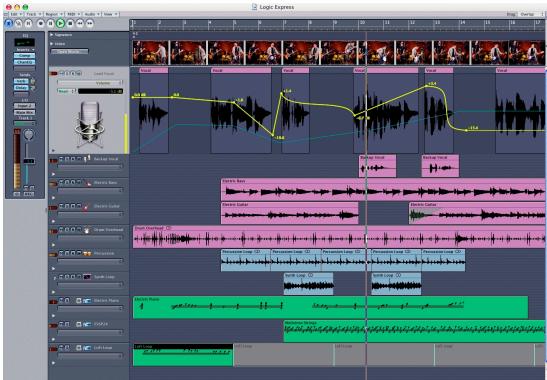
- It can be a stand-alone unit or a software program for a personal computer. (It used to be a storage server for MIDI data. Nowadays it is more a software *music editor* on the computer.)
- It has one or more MIDI INs and MIDI OUTs and/or USB/Firewire connectivity
- If sofware based virtual midi connections



Components of a MIDI System (Cont.)

Computer:

- Heart of a MIDI system
- Controls the scheduling, synchronisation and recording of all data.
- Sequencer usually software based and now part of larger applications that control all aspects of Audio and Midi — Digital Audio Workstation packages such as Cubase, Logic, Sonar, Live, Reason.
- Nowadays, includes many software synthesisers/samplers to make sounds in real time.
- Real time effects
- Control of Video also integral these days.



Components of a MIDI System (Cont.)

Midi Control Input Devices:

- Usually a Keyboard with additional control: sustain, pitch bend, modulation, aftertouch and other controllers
- Can be another musical device e.g. Customised Guitar, Wind Controller
- Can be just a bunch of controllers.
- Can be even more strange: Motion Capture, or virtual input or mind control!!



Components of a MIDI System (Cont.)

Midi Interfaces:

Midi devices (still) need to connect to computer with some interface

- Midi Interface — USB or Firewire
- Often functionality bundled with Keyboard or controller
- Audio Interface via USB or Firewire common
- Even Wireless Keyboards



Components of a MIDI System (Cont.)

Midi Control Output Devices:

- Not just making sounds
 - MIDI controls other things
 - Lighting
 - Robotics
 - Even Pat Metheny and his Musical Robot Band: Orchestrion!!
 - Video Systems e.g. Video DJing
 - MPEG4 Compression — More soon
 - Even Hamster Control!!!
 - Lots of other applications
- For a full range of MIDI I/o Controllers check out <http://www.synthzone.com/ctrlr.htm>



Basic MIDI Concepts

Track:

- Track in sequencer is used to organize the recordings.
- Tracks can be turned on or off on recording or playing back.

Channel:

- MIDI channels are used to separate information in a MIDI system.
- There are 16 MIDI channels in one '*cable*'.
- Channel numbers are coded into each MIDI message.

Timbre:

- The quality of the sound, e.g., flute sound, cello sound, etc.
- Multitimbral – capable of playing many different sounds at the same time (e.g., piano, brass, drums, etc.)



Back

Close

Basic MIDI Concepts (Cont.)

Pitch:

- The Musical note that the instrument plays

Voice:

- Voice is the portion of the synthesiser that produces sound.
- synthesisers can have many (12, 20, 24, 36, etc.) voices.
- Each voice works independently and simultaneously to produce sounds of different timbre and pitch.

Patch:

- The control settings that define a particular timbre.



Back

Close

Hardware Aspects of MIDI

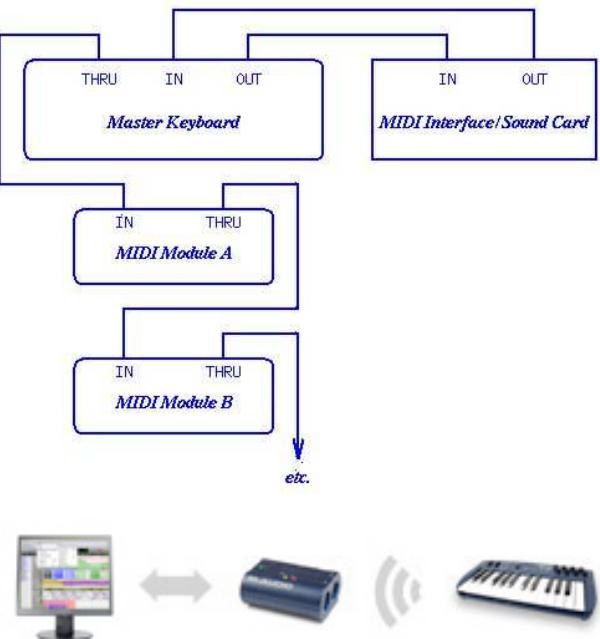
MIDI connectors:

– Three 5-pin ports found on the back of every MIDI unit

- **MIDI IN**: the connector via which the device receives all MIDI data.
- **MIDI OUT**: the connector through which the device transmits all the MIDI data it generates itself.
- **MIDI THROUGH**: the connector by which the device echoes the data receives from MIDI IN.

– As mentioned previously many modern interfaces connect via USB/Firewire

- Many devices bypass direct MIDI IN/OUT/THROUGH and have a direct (or possibly even wireless) USB/Firewire connection to the computer.



MIDI Messages

MIDI messages are used by MIDI devices to communicate with each other.

MIDI messages are very low bandwidth:

- Note On Command
 - Which Key is pressed
 - Which MIDI Channel (what sound to play)
 - 3 Hexadecimal Numbers
- Note Off Command Similar
- Other command (program change) configure sounds to be played.

Structure of MIDI messages:

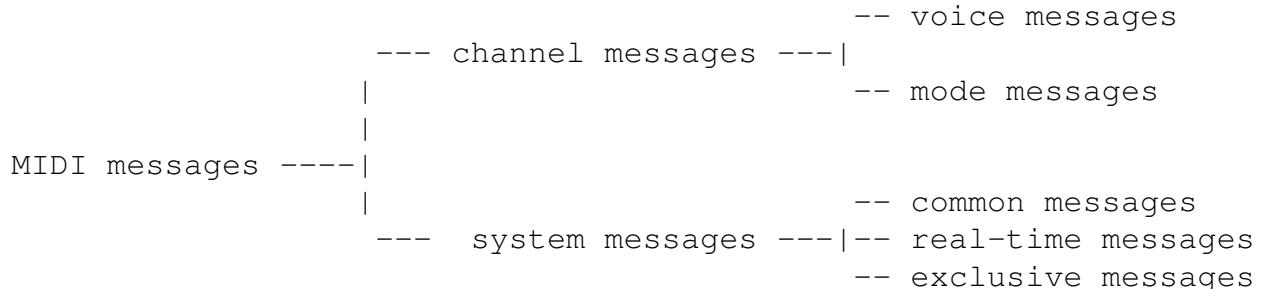
- MIDI message includes a status byte and up to two data bytes.
- Status byte
 - The most significant bit of status byte is set to 1.
 - The 4 low-order bits identify which channel it belongs to (four bits produce 16 possible channels).
 - The 3 remaining bits identify the message.
- The most significant bit of data byte is set to 0.



Back

Close

Classification of MIDI messages:



Midi Channel messages:

- messages that are transmitted on individual channels rather than globally to all devices in the MIDI network.

Channel voice messages:

- Instruct the receiving instrument to assign particular sounds to its voice
- Turn notes on and off
- Alter the sound of the currently active note or notes



Back

Close

Midi Channel Control Messages

Voice Message	Status Byte	Data Byte1	Data Byte2
Note off	8x	Key number	Note Off velocity
Note on	9x	Key number	Note on velocity
Polyphonic Key Pressure	Ax	Key number	Amount of pressure
Control Change	Bx	Controller number	Controller value
Program Change	Cx	Program number	None
Channel Pressure	Dx	Pressure value	None
Pitch Bend	Ex	MSB	LSB

Notes: 'x' in status byte hex value stands for a channel number.



Back

Close

Midi Command Example

A Note On message is followed by two bytes, one to identify the note, and one to specify the velocity.

To play:

- Note number 80 (HEX 50)
- With maximum velocity (127 (Hex 7F))
- On channel 13 (Hex C),

The MIDI device would send these three hexadecimal byte values:

9C 50 7F



Back

Close

Midi Channel mode messages:

- Channel mode messages are a special case of the Control Change message (Bx (Hex) or 1011nnnn (Binary)).
- The difference between a Control message and a Channel Mode message, is in the first data byte.
 - Data byte values 121 through 127 have been reserved in the Control Change message for the channel mode messages.
 - Channel mode messages determine how an instrument will process MIDI voice messages.



Back

Close

System Messages:

- System messages carry information that are not channel specific, Examples:
 - Timing signal for synchronization,
 - Positioning information in pre-recorded MIDI sequences, and
 - Detailed setup information for the destination device
 - Setting up sounds, Patch Names etc.



Back

Close

Midi System Real-time Messages

- These messages are related to synchronization/timing etc.

System Real-Time Message

Timing Clock

Start Sequence

Continue Sequence

Stop Sequence

Active Sensing

System Reset

Status Byte

F8

FA

FB

FC

FE

FF



Back

Close

System common messages

- These contain the following (unrelated) messages

System Common Message	Status Byte	Number of Data Bytes
MIDI Timing Code	F1	1
Song Position Pointer	F2	2
Song Select	F3	1
Tune Request	F6	None



Back

Close

Midi System exclusive messages

- Messages related to things that cannot be standardized:
 - System dependent creation of sound
 - System dependent organisation of sounds
(Not General Midi Compliant? (more soon))
- An addition to the original MIDI specification.
- Just a stream of bytes
 - all with their high bits set to 0,
 - bracketed by a pair of system exclusive start and end messages:
F0 — Sysex Start
F7 — Sysex End
 - Format of message byte stream system dependent.



Back

Close

General MIDI (GM)

Problem: Midi Music may not sound the same everywhere?

Basic GM Idea:

- **MIDI + Instrument Patch Map + Percussion Key Map** –> a piece of MIDI music sounds (more or less) the same anywhere it is played
 - Instrument patch map is a standardised list consisting of 128 instruments (patches).
Same instrument type sounds similar if not identical sound
 - Percussion map specifies 47 percussion sounds.
Same Drum type sounds on keyboard map
 - Key-based percussion is always transmitted on MIDI channel 10 (Default)
Can be transmitted on other channels as well



Back

Close

Requirements for General MIDI Compatibility

- Support all 16 channels — [Default standard Multitimbral MIDI Specification](#)
- Each channel can play a different instrument/program — [multitimbral](#)
- Each channel can play many notes — [polyphony](#)
- Minimum of 24 (usually much higher 64/128) fully dynamically allocated voices — [shared across all channels](#)



Back

Close

General MIDI Instrument Patch Map

Prog No. Instrument

Prog No. Instrument

(1-8 PIANO)

- | | |
|---|------------------|
| 1 | Acoustic Grand |
| 2 | Bright Acoustic |
| 3 | Electric Grand |
| 4 | Honky-Tonk |
| 5 | Electric Piano 1 |
| 6 | Electric Piano 2 |
| 7 | Harpsichord |
| 8 | Clav |

(9-16 CHROM PERCUSSION)

- | | |
|----|---------------|
| 9 | Celesta |
| 10 | Glockenspiel |
| 11 | Music Box |
| 12 | Vibraphone |
| 13 | Marimba |
| 14 | Xylophone |
| 15 | Tubular Bells |
| 16 | Dulcimer |

(17-24 ORGAN)

- | | |
|----|------------------|
| 17 | Drawbar Organ |
| 18 | Percussive Organ |
| 19 | Rock Organ |
| 20 | Church Organ |
| 21 | Reed Organ |
| 22 | Accordion |
| 23 | Harmonica |
| 24 | Tango Accordion |

(25-32 GUITAR)

- | | |
|----|------------------------|
| 25 | Acoustic Guitar(nylon) |
| 26 | Acoustic Guitar(steel) |
| 27 | Electric Guitar(jazz) |
| 28 | Electric Guitar(clean) |
| 29 | Electric Guitar(muted) |
| 30 | Overdriven Guitar |
| 31 | Distortion Guitar |
| 32 | Guitar Harmonics |

(33-40 BASS)

- | | |
|----|-----------------------|
| 33 | Acoustic Bass |
| 34 | Electric Bass(finger) |
| 35 | Electric Bass(pick) |
| 36 | Fretless Bass |
| 37 | Slap Bass 1 |
| 38 | Slap Bass 2 |
| 39 | Synth Bass 1 |
| 40 | Synth Bass 2 |

(41-48 STRINGS)

- | | |
|----|--------------------|
| 41 | Violin |
| 42 | Viola |
| 43 | Cello |
| 44 | Contrabass |
| 45 | Tremolo Strings |
| 46 | Pizzicato Strings |
| 47 | Orchestral Strings |
| 48 | Timpani |



Back

Close

(49-56 ENSEMBLE)		(57-64 BRASS)	
49	String Ensemble 1	57	Trumpet
50	String Ensemble 2	58	Trombone
51	SynthStrings 1	59	Tuba
52	SynthStrings 2	60	Muted Trumpet
53	Choir Aahs	61	French Horn
54	Voice Oohs	62	Brass Section
55	Synth Voice	63	SynthBrass 1
56	Orchestra Hit	64	SynthBrass 2
(65-72 REED)		(73-80 PIPE)	
65	Soprano Sax	73	Piccolo
66	Alto Sax	74	Flute
67	Tenor Sax	75	Recorder
68	Baritone Sax	76	Pan Flute
69	Oboe	77	Blown Bottle
70	English Horn	78	Skakuhachi
71	Bassoon	79	Whistle
72	Clarinet	80	Ocarina
(81-88 SYNTH LEAD)		(89-96 SYNTH PAD)	
81	Lead 1 (square)	89	Pad 1 (new age)
82	Lead 2 (sawtooth)	90	Pad 2 (warm)
83	Lead 3 (calliope)	91	Pad 3 (polysynth)
84	Lead 4 (chiff)	92	Pad 4 (choir)
85	Lead 5 (charang)	93	Pad 5 (bowed)
86	Lead 6 (voice)	94	Pad 6 (metallic)
87	Lead 7 (fifths)	95	Pad 7 (halo)
88	Lead 8 (bass+lead)	96	Pad 8 (sweep)



Back

Close

(97-104 SYNTHE EFFECTS)

- | | | | |
|-----|-------------------|-----|----------|
| 97 | FX 1 (rain) | 105 | Sitar |
| 98 | FX 2 (soundtrack) | 106 | Banjo |
| 99 | FX 3 (crystal) | 107 | Shamisen |
| 100 | FX 4 (atmosphere) | 108 | Koto |
| 101 | FX 5 (brightness) | 109 | Kalimba |
| 102 | FX 6 (goblins) | 110 | Bagpipe |
| 103 | FX 7 (echoes) | 111 | Fiddle |
| 104 | FX 8 (sci-fi) | 112 | Shanai |

(113-120 PERCUSSIVE)

- | | | | |
|-----|----------------|-----|-------------------|
| 113 | Tinkle Bell | 121 | Guitar Fret Noise |
| 114 | Agogo | 122 | Breath Noise |
| 115 | Steel Drums | 123 | Seashore |
| 116 | Woodblock | 124 | Bird Tweet |
| 117 | Taiko Drum | 125 | Telephone Ring |
| 118 | Melodic Tom | 126 | Helicopter |
| 119 | Synth Drum | 127 | Applause |
| 120 | Reverse Cymbal | 128 | Gunshot |

(105-112 ETHNIC)

- | | |
|-----|----------|
| 105 | Sitar |
| 106 | Banjo |
| 107 | Shamisen |
| 108 | Koto |
| 109 | Kalimba |
| 110 | Bagpipe |
| 111 | Fiddle |
| 112 | Shanai |

(121-128 SOUND EFFECTS)

- | | |
|-----|-------------------|
| 121 | Guitar Fret Noise |
| 122 | Breath Noise |
| 123 | Seashore |
| 124 | Bird Tweet |
| 125 | Telephone Ring |
| 126 | Helicopter |
| 127 | Applause |
| 128 | Gunshot |



Back

Close

General MIDI Percussion Key Map

MIDI Key	Drum Sound	MIDI Key	Drum Sound
35	Acoustic Bass Drum	59	Ride Cymbal 2
36	Bass Drum 1	60	Hi Bongo
37	Side Stick	61	Low Bongo
38	Acoustic Snare	62	Mute Hi Conga
39	Hand Clap	63	Open Hi Conga
40	Electric Snare	64	Low Conga
41	Low Floor Tom	65	High Timbale
42	Closed Hi-Hat	66	Low Timbale
43	High Floor Tom	67	High Agogo
44	Pedal Hi-Hat	68	Low Agogo
45	Low Tom	69	Cabasa
46	Open Hi-Hat	70	Maracas
47	Low-Mid Tom	71	Short Whistle
48	Hi-Mid Tom	72	Long Whistle
49	Crash Cymbal 1	73	Short Guiro
50	High Tom	74	Long Guiro
51	Ride Cymbal 1	75	Claves
52	Chinese Cymbal	76	Hi Wood Block
53	Ride Bell	77	Low Wood Block
54	Tambourine	78	Mute Cuica
55	Splash Cymbal	79	Open Cuica
56	Cowbell	80	Mute Triangle
57	Crash Cymbal 2	81	Open Triangle
58	Vibraslap		

35	Acoustic Bass Drum	59	Ride Cymbal 2
36	Bass Drum 1	60	Hi Bongo
37	Side Stick	61	Low Bongo
38	Acoustic Snare	62	Mute Hi Conga
39	Hand Clap	63	Open Hi Conga
40	Electric Snare	64	Low Conga
41	Low Floor Tom	65	High Timbale
42	Closed Hi-Hat	66	Low Timbale
43	High Floor Tom	67	High Agogo
44	Pedal Hi-Hat	68	Low Agogo
45	Low Tom	69	Cabasa
46	Open Hi-Hat	70	Maracas
47	Low-Mid Tom	71	Short Whistle
48	Hi-Mid Tom	72	Long Whistle
49	Crash Cymbal 1	73	Short Guiro
50	High Tom	74	Long Guiro
51	Ride Cymbal 1	75	Claves
52	Chinese Cymbal	76	Hi Wood Block
53	Ride Bell	77	Low Wood Block
54	Tambourine	78	Mute Cuica
55	Splash Cymbal	79	Open Cuica
56	Cowbell	80	Mute Triangle
57	Crash Cymbal 2	81	Open Triangle
58	Vibraslap		

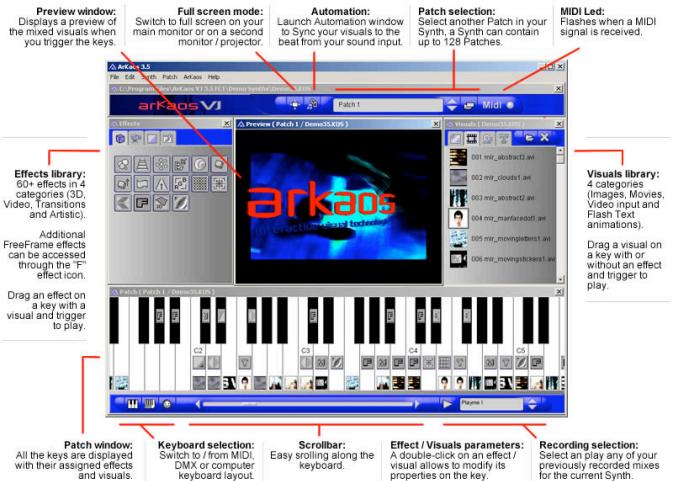
MIDI Percussion Key Mapping

- Each key is essentially a switch
- No **Pitch** information relevant — usually
- Can be extended to control other stuff e.g. **Video DJ (VJ)** application

F#4	Mute Triangle	A3	Open Triangle
F#4	Mute Cuica	G4	Open Cuica
F#4	Low Wood Block	F#4	Low Wood Block
D#4	Clevers	D4	Long Guiro
D#4	Short Guiro	C4	Long Whistle
B3	Short Whistle	B3	Short Whistle
A#3	Marracas	A3	Cabasa
G#3	Low/gogoll	G3	High Agogo
F#3	Low Timbale	F3	High Timbale
E3	Low Conga	E3	Low Conga
D#3	Open Hi Conga	D3	Mute Hi Conga
C#3	Low Bongo	C3	Hi Bongo
B3	Ride Cymbal 1	B3	Ride Cymbal 2
A#2	Vibraslap	A#2	Crash Cymbal 2
G#2	Crash Cymbal	G2	Splash Cymbal
F#2	Tambourine	F2	Ride Bell
E2	Chinese Cymbal		
D#2	Ride Cymbal 1	D2	High Tom
C#2	Crash Cymbal 1	C2	Mid Tom
B3	Open Hi Hat	B1	Cowbell
A#1	Hand Clap	A1	Low Tom
G#1	Side Stick	G1	Pedal Hi Hat
F#1	Acoustic Snare	F1	High Floor Tom
E1	Bass Drum	E1	Low Floor Tom
D#1		D#1	
C#1		C#1	
B2	Acoustic Bass Drum	B2	

Pitch	Instrument
C1	Bass Drum
C#1	Rim
D1	Snare 1
E1	Snare 2
F#1	Hi-hat Closed
G#1	Hi-hat Pedal
A#1	Hi-Hat Open
A1	Tom Low
C2	Tom Mid
D2	Tom High
D#2	Ride
C#2	Crash
C-2	Sound 1

ArKaos VJ MIDI



Limitations of Conventional MIDI

- Limited Number of Channels and Controllers
- Limited resolution in data values
 - Most midi numbers are 8-bit

Solutions:

- Some MIDI manufacturer utilities two midi data values to allow for large range of values

E.g. Use values as Least and Most Significant Bytes: **16 bit range**

- Open Sound Control (OSC) — been around a while, MIDI still rules?
- High Definition MIDI — fixes the above and adds more features.



Back

Close

Digital Audio, Synthesis, Midi and Compression — MPEG 4 Structured Audio

- We have seen the need for compression already in Digital Audio — Large Data Files
- Basic Ideas of compression (next lecture) used as integral part of audio format — MP3, real audio etc.
- Mpeg-4 audio — actually combines compression synthesis and midi to have a massive impact on compression.
- Midi, Synthesis encode what note to play and how to play it with a small number of parameters
 - Much greater reduction than simply having some encoded bits of audio.
- Responsibility to create audio delegated to generation side.

MPEG 4 Structured Audio

A newer standard than MP3 Audio — which we study in detail later

Multimedia
CM0340

195

MPEG-4 covers the the whole range of digital audio:

- From very low bit rate speech
- To full bandwidth high quality audio
- Built in anti-piracy measures
- **Structured Audio**
- Relation to MIDI so we study MPEG 4 audio here



Back

Close

Structured Audio Tools

MPEG-4 comprises of 6 *Structured Audio tools* are:

SAOL – the Structured Audio Orchestra Language

SASL – the Structured Audio Score Language

SASBF – the Structured Audio Sample Bank Format

Set of MIDI semantics — describe how to control SAOL with MIDI

Scheduler – describe how to take the above parts and create sound

AudioBIFS – part of BIFS, which lets you make audio soundtracks in MPEG-4 using a variety of tools and effects-processing techniques

SAOL

(Structured Audio Orchestra Language)

- Pronounced “[sail](#)”
- The central part of the Structured Audio toolset.
- A new software-synthesis language
- A language for describing synthesisers, a program, or instrument
- Specifically designed it for use in MPEG-4.
- Not based on any particular method of synthesis – supports many underlying synthesis methods.



Back

Close

SAOL Synthesis Methods

- Any known method of synthesis can be described in SAOL (Open Support).
 - FM synthesis,
 - physical-modeling synthesis,
 - Sample-based synthesis,
 - granular synthesis,
 - subtractive synthesis,
 - FOF synthesis, and
 - hybrids of all of these in SAOL.

SASL (Structured Audio Score Language)

- A very simple language to control the synthesisers specified by SAOL instruments.
- A SASL program, or score, contains instructions that tell SAOL:
 - what notes to play,
 - how loud to play them,
 - what tempo to play them at,
 - how long they last, and how to control them
- Similar to MIDI
 - doesn't suffer from MIDI's restrictions on temporal resolution or bandwidth.
 - more sophisticated controller structure



Back

Close

SASL (Structured Audio Score Language) (Cont.)

- Lightweight Scoring Language: Does not support:
 - looping,
 - sections,
 - repeats,
 - expression evaluation,
 - some other things.
 - most SASL scores will be created by automatic tools



Back

Close

SASBF

(Structured Audio Sample Bank Format)

- A format for efficiently transmitting banks of sound samples
- Used in wavetable, or sample-based synthesis.
- Partly compatible with the MIDI Downloaded Sounds (DLS) format
- The most active participants in this activity are EMu Systems (sampler manufacturer) and the MIDI Manufacturers Association (MMA).



Back

Close

MPEG-4 MIDI Semantics

SASL can be controlled by

- SASL Scripts
- MIDI
- Scores in MPEG-4

Reasons to use MIDI:

- MIDI is today's most commonly used representation for music score data,
- Many sophisticated authoring tools (such as sequencers) work with MIDI.

MPEG-4 Midi Control

- MIDI syntax external to MPEG-4 Structured Audio standard
- Use MIDI Manufacturers Association's standard.
- *Redefines* the some semantics for MPEG-4.
- The new semantics are carefully defined as part of the MPEG-4 specification.



Back

Close

MPEG-4 Scheduler

- The main body of the Structured Audio definition.
- A set of carefully defined and somewhat complicated instructions
- Specify how SAOL is used to create sound when it is driven by MIDI or SASL.



Back

Close

AudioBIFS

- BIFS is the MPEG-4 *Binary Format for Scene Description*.
- Describes how the different "objects" in a structured media scene fit together:
 - MPEG-4 consists also of the video clips, sounds, animations, and other pieces of multimedia
 - Each have special formats to describe them.
 - Need to put the pieces together
 - BIFS lets you describe how to put the pieces together.



Back

Close

AudioBIFS (Cont.)

- AudioBIFS is designed for specifying the mixing and post-production of audio scenes as they're played back.
- For example,
 - we can specify how the voice-track is mixed with the background music, and
 - that it fades out after 10 seconds and
 - this other music comes in and has a nice reverb on it.
- Extended version of VRML: capabilities for
 - streaming and
 - mixing audio and video data
- Very advanced sound model.

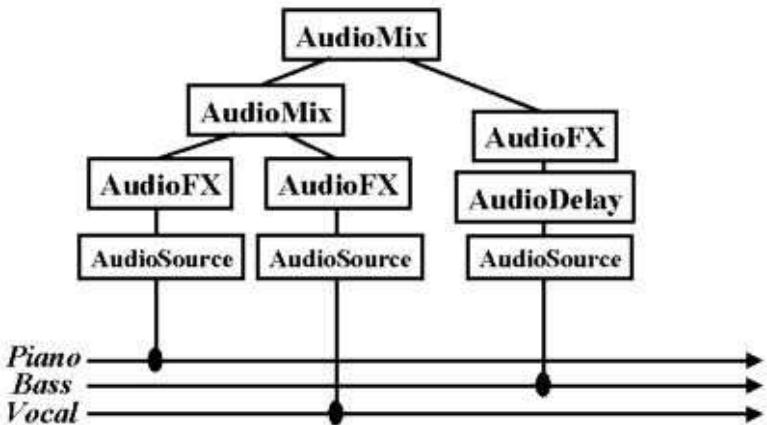


Back

Close

AudioBIFS (Cont.)

How a simple sound is created from three elementary sound streams:



HTML 5 MIDI

A new Web MIDI API (2013)¹:

- Part of general web audio development of HTML 5

The Web MIDI API specification

- Defines a means for web developers to manipulate and access MIDI devices
 - Midi Input and Output to hardware (outboard) and software.
 - Audio Synthesis available in Browser.
 - Total Web-Midi Control.
 - JavaScript Programming.

¹Support of Web MIDI API is not that well developed. Not all browsers support it. See [here](#) for an example of how to install



Back

Close

Some HTML 5 MIDI Examples: Moog Doodle

The first app was the Google Doodle for the [Mini Moog](#).



- Uses Web Audio/MIDI API to recreate a [Moog Synthesiser](#).
- Subtractive Sythesis on Web — [code here](#).
- [Celebrated Bob Moog's 78th Birthday](#).
- [Spawned a whole community](#).

Some HTML 5 MIDI Examples: MIDI Controlled Subtractive Synthesis

A fully fledged MIDI controlled Subtractive Synthesiser

Multimedia
CM0340

210



- Configurable MIDI input — needs JAVA²
- Polyphonic Synthesiser

²see [for details how run this](#)



Back

Close

Some Other HTML 5 MIDI Examples

Drum Machine :

Web Audio Drumming

Wavetable Synthesis :

Controllable Wavetables

Granular Synthesis :

Simple Granular Synth

More Examples :

webaudiodeemos.appspot.com

Some More Examples :

jazz-soft.net/demo

Digital Audio Effects

Having learned to make basic sounds from basic waveforms and more advanced synthesis methods lets see how we can add some digital audio effects. These may be applied:

- As part of the audio creation/synthesis stage — to be subsequently filtered, (re)synthesised
- At the end of the *audio chain* — as part of the production/mastering phase.
- Effects can be applied in different orders and sometimes in a *parallel* audio chain.
- The order of applying the same effects can have drastic differences in the output audio.
- Selection of effects and the ordering is a matter for the sound you wish to create. There is no absolute rule for the ordering.



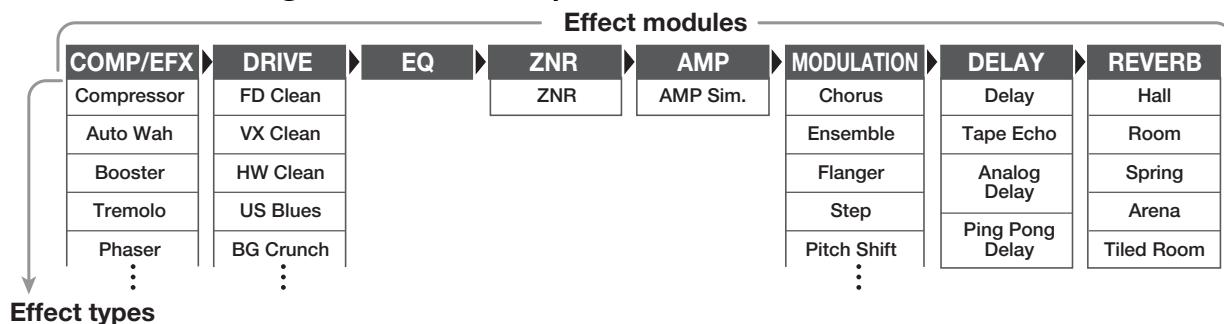
Back

Close

Typical Guitar (and other) Effects Pipeline

Some ordering is *standard* for some audio processing, *E.g.*:
 Compression → Distortion → EQ → Noise Redux → Amp Sim →
 Modulation → Delay → Reverb

Common for guitar effects pedal.





Multimedia
CM0340

214

◀◀

▶▶

◀

▶

Back

Close

Classifying Effects

Audio effects can be classified by the way do their processing:

Basic Filtering — Lowpass, Highpass filter etc., Equaliser: see
[CM2202 Notes](#)

Time Varying Filters — **Wah-wah, Phaser**

Delays — **Vibrato, Flanger, Chorus, Echo**

Modulators — **Ring modulation, Tremolo, Vibrato**

Non-linear Processing — Compression, Limiters, **Distortion**,
Exciters/Enhancers

Spacial Effects — Panning, Reverb (see [CM2202 Notes](#)), Surround
Sound

Some of the above **studied here**.³

³See [these notes](#) for further information. This additional information may be **useful for Coursework** but is **NOT examinable**



Time-varying Filters

Some common effects are realised by simply time varying a filter in a couple of different ways:

Wah-wah — A bandpass filter with a time varying centre (resonant) frequency and a small bandwidth. Filtered signal mixed with direct signal.

Phasing — A notch filter, that can be realised as set of cascading IIR filters, again mixed with direct signal.

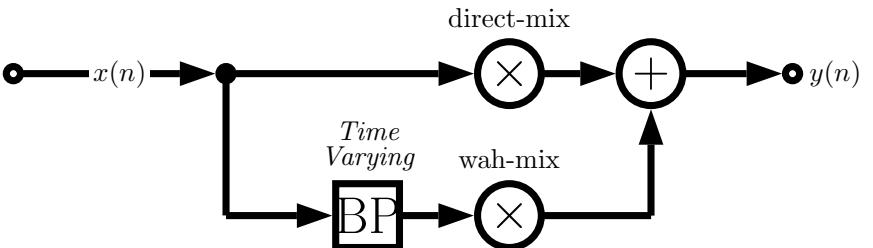


Back

Close

Wah-wah Example

The signal flow for a wah-wah is as follows:



where BP is a time varying frequency bandpass filter.

- A *phaser* is similarly implemented with a notch filter replacing the bandpass filter.
- A variation is the *M-fold wah-wah* filter where M tap delay bandpass filters spread over the entire spectrum change their centre frequencies simultaneously.
- A **bell effect** can be achieved with around a hundred M tap

delays and narrow bandwidth filters



Back

Close

Time Varying Filter Implementation: State Variable Filter

In our audio application of time varying filters we now want independent control over the cut-off frequency and damping factor of a filter.

(Borrowed from analog electronics) we can implement a **State Variable Filter** to solve this problem.

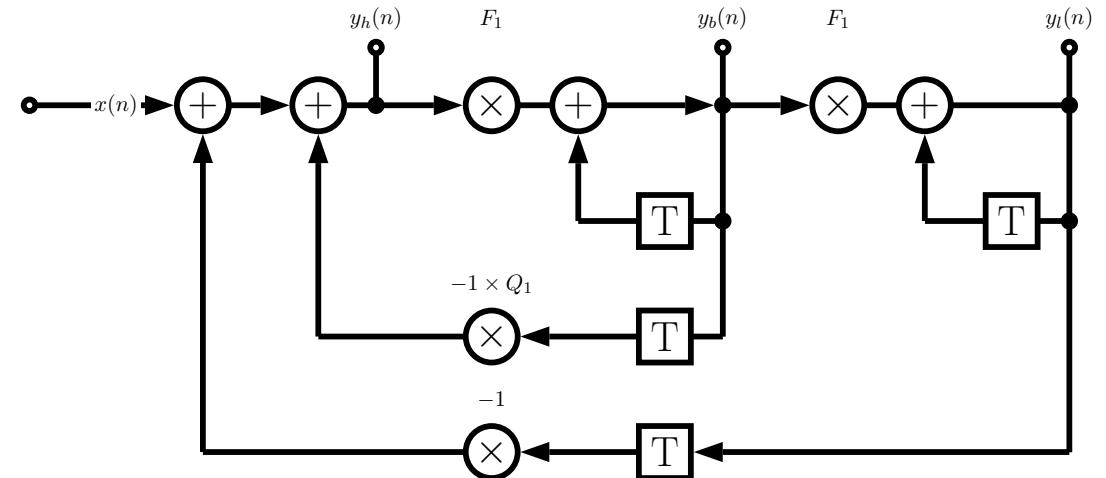
- One further advantage is that we can **simultaneously** get lowpass, bandpass and highpass filter output.



Back

Close

The State Variable Filter



where:

$x(n)$ = input signal

$y_l(n)$ = lowpass signal

$y_b(n)$ = bandpass signal

$y_h(n)$ = highpass signal



Back

Close

The State Variable Filter Algorithm

The algorithm difference equations are given by:

$$\begin{aligned}y_l(n) &= F_1 y_b(n) + y_l(n - 1) \\y_b(n) &= F_1 y_h(n) + y_b(n - 1) \\y_h(n) &= x(n) - y_l(n - 1) - Q_1 y_b(n - 1)\end{aligned}$$

with tuning coefficients F_1 and Q_1 related to the cut-off frequency, f_c , and damping, d :

$$F_1 = 2 \sin(\pi f_c / f_s), \quad \text{and} \quad Q_1 = 2d$$



Back

Close

MATLAB Wah-wah Implementation

We simply implement the State Variable Filter with a variable frequency, f_c . The code listing is [wah_wah.m](#):

```
% wah_wah.m      state variable band pass
%
% BP filter with narrow pass band, Fc oscillates up and
% down the spectrum
% Difference equation taken from DAFX chapter 2
%
% Changing this from a BP to a BR/BS (notch instead of a bandpass) converts
% this effect to a phaser
%
% yl(n) = F1*yb(n) + yl(n-1)
% yb(n) = F1*yh(n) + yb(n-1)
% yh(n) = x(n) - yl(n-1) - Q1*yb(n-1)
%
% vary Fc from 500 to 5000 Hz

infile = 'acoustic.wav';

% read in wav sample
[ x, Fs, N ] = wavread(infile);
```



Back

Close

```
%%%%%% EFFECT COEFFICIENTS %%%%%%  
%%%%% %%%%%% %%%%%% %%%%%% %%%%%% %%%%%% %%%%%% %%%%%% %%%%%% %%%%%% %%%%%%
```

```
% damping factor
```

```
% lower the damping factor the smaller the pass band
```

```
damp = 0.05;
```

```
% min and max centre cutoff frequency of variable bandpass filter
```

```
minf=500;
```

```
maxf=3000;
```

```
% wah frequency, how many Hz per second are cycled through
```

```
Fw = 2000;
```

```
%%%%%% %%%%%% %%%%%% %%%%%% %%%%%% %%%%%% %%%%%% %%%%%% %%%%%% %%%%%% %%%%%%
```

```
% change in centre frequency per sample (Hz)
```

```
delta = Fw/Fs;
```

```
% create triangle wave of centre frequency values
```

```
Fc=minf:delta:maxf;
```

```
while(length(Fc) < length(x) )
```

```
    Fc= [ Fc (maxf:-delta:minf) ];
```

```
    Fc= [ Fc (minf:delta:maxf) ];
```

```
end
```

```
% trim tri wave to size of input
```



Back

Close

```
Fc = Fc(1:length(x));  
  
% difference equation coefficients  
% must be recalculated each time Fc changes  
F1 = 2*sin((pi*Fc(1))/Fs);  
% this dictates size of the pass bands  
Q1 = 2*damp;  
  
yh=zeros(size(x)); % create empty out vectors  
yb=zeros(size(x));  
yl=zeros(size(x));  
  
% first sample, to avoid referencing of negative signals  
yh(1) = x(1);  
yb(1) = F1*yh(1);  
yl(1) = F1*yb(1);  
  
% apply difference equation to the sample  
for n=2:length(x),  
    yh(n) = x(n) - yl(n-1) - Q1*yb(n-1);  
    yb(n) = F1*yh(n) + yb(n-1);  
    yl(n) = F1*yb(n) + yl(n-1);  
    F1 = 2*sin((pi*Fc(n))/Fs);  
end  
  
%normalise  
maxyb = max(abs(yb));
```



Back

Close

```
yb = yb/maxyb;  
  
% write output wav files  
wavwrite(yb, Fs, N, 'out_wah.wav');  
  
figure(1)  
hold on  
plot(x,'r');  
plot(yb,'b');  
title('Wah-wah and original Signal');
```

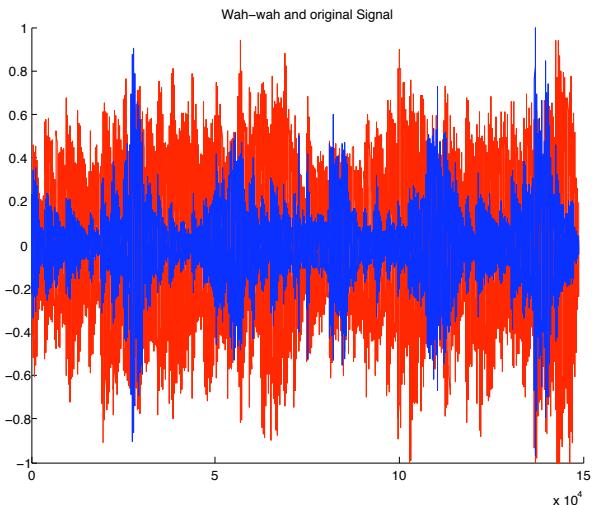


Back

Close

Wah-wah MATLAB Example (Cont.)

The output from the above code is (red plot is original audio):



Click here to hear: [original audio](#), [wah-wah filtered audio](#).



Back

Close

Delay Based Effects

Many useful audio effects can be implemented using a delay structure:

- Sounds reflected off walls
 - In a cave or large room we hear an echo and also reverberation takes place – this is a different effect — **see later**
 - If walls are closer together repeated reflections can appear as parallel boundaries and we hear a modification of sound colour instead.
- Vibrato, Flanging, Chorus and Echo are examples of delay effects



Back

Close

Basic Delay Structure

We build basic delay structures out of some very basic FIR and IIR filters:

- We use *FIR* and *IIR comb filters*
- Combination of FIR and IIR gives the **Universal Comb Filter**

FIR Comb Filter

This simulates a single delay:

- The input signal is delayed by a given time duration, τ .
- The delayed (processed) signal is added to the input signal some amplitude gain, g
- The difference equation is simply:

$$y(n) = x(n) + gx(n - M) \quad \text{with } M = \tau/f_s$$

- The transfer function is:

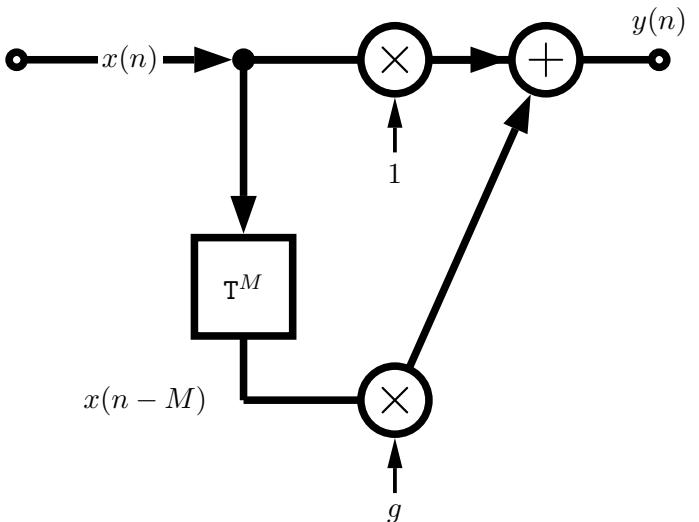
$$H(z) = 1 + gz^{-M}$$



Back

Close

FIR Comb Filter Signal Flow Diagram



FIR Comb Filter MATLAB Code

fircomb.m:

```
x=zeros(100,1);x(1)=1; % unit impulse signal of length 100  
  
g=0.5; %Example gain  
  
Delayline=zeros(10,1); % memory allocation for length 10  
  
for n=1:length(x);  
    y(n)=x(n)+g*Delayline(10);  
    Delayline=[x(n);Delayline(1:10-1)];  
end;
```



Back

Close

IIR Comb Filter

This simulates a single delay:

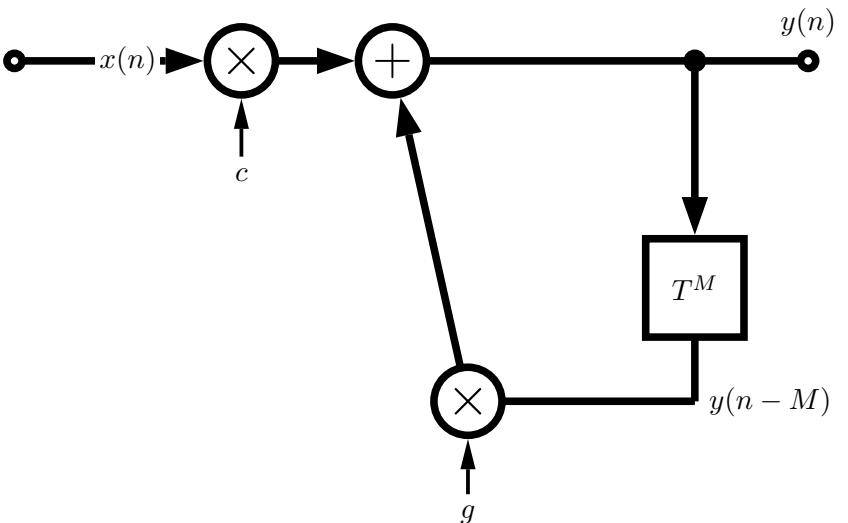
- Simulates **endless reflections** at both ends of cylinder.
- We get an endless series of responses, $y(n)$ to input, $x(n)$.
- The input signal circulates in delay line (delay time τ) that is fed back to the input..
- Each time it is fed back it is attenuated by g .
- Input sometime scaled by c to **compensate** for high amplification of the structure.
- The difference equation is simply:

$$y(n) = Cx(n) + gy(n - M) \quad \text{with } M = \tau/f_s$$

- The transfer function is:

$$H(z) = \frac{c}{1 - gz^{-M}}$$

IIR Comb Filter Signal Flow Diagram



IIR Comb Filter MATLAB Code

iircmb.m:

```
x=zeros(100,1);x(1)=1; % unit impulse signal of length 100  
  
g=0.5;  
  
Delayline=zeros(10,1); % memory allocation for length 10  
  
for n=1:length(x);  
    y(n)=x(n)+g*Delayline(10);  
    Delayline=[y(n);Delayline(1:10-1)];  
end;
```



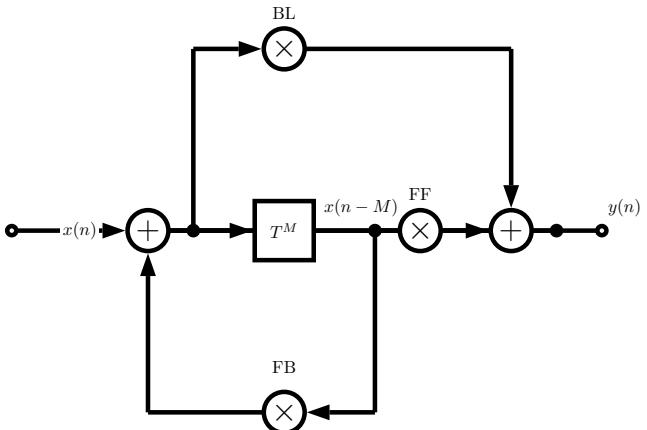
Back

Close

Universal Comb Filter

The combination of the FIR and IIR comb filters yields the **Universal Comb Filter**:

- Basically this is an **allpass filter** with an M sample delay operator and an additional multiplier, FF.



- Parameters: FF = feedforward, FB = feedbackward, BL = blend

Universal Comb Filter Parameters

Universal in that we can form any comb filter, an allpass or a delay:

	BL	FB	FF
FIR Comb	1	0	g
IIR Comb	1	g	0
Allpass	a	$-a$	1
delay	0	0	1



Universal Comb Filter MATLAB Code

unicomb.m:

```
x=zeros(100,1);x(1)=1; % unit impulse signal of length 100  
  
BL=0.5;  
FB=-0.5;  
FF=1;  
M=10;  
  
Delayline=zeros(M,1); % memory allocation for length 10  
  
for n=1:length(x);  
    xh=x(n)+FB*Delayline(M);  
    y(n)=FF*Delayline(M)+BL*xh;  
    Delayline=[xh;Delayline(1:M-1)];  
end;
```



Back

Close

Vibrato - A Simple Delay Based Effect

- **Vibrato** — Varying the time delay periodically
- If we vary the distance between an observer and a sound source (*cf. Doppler effect*) we hear a change in pitch.
- Implementation: A Delay line and a low frequency oscillator (LFO) to vary the delay.
- Only listen to the delay — no forward or backward feed.
- Typical delay time = 5–10 Ms and LFO rate 5–14Hz.

Vibrato MATLAB Code

vibrato.m function, Use vibrato_eg.m to call function:

```
function y=vibrato(x,SAMPLERATE,Modfreq,Width)

ya_alt=0;
Delay=Width; % basic delay of input sample in sec
DELAY=round(Delay*SAMPLERATE); % basic delay in # samples
WIDTH=round(Width*SAMPLERATE); % modulation width in # samples
if WIDTH>DELAY
    error('delay greater than basic delay !!!');
    return;
end;

MODFREQ=Modfreq/SAMPLERATE; % modulation frequency in # samples
LEN=length(x); % # of samples in WAV-file
L=2+DELAY+WIDTH*2; % length of the entire delay
Delayline=zeros(L,1); % memory allocation for delay
y=zeros(size(x)); % memory allocation for output vector
```

```
for n=1:(LEN-1)
    M=MODFREQ;
    MOD=sin(M*2*pi*n);
    ZEIGER=1+DELAY+WIDTH*MOD;
    i=floor(ZEIGER);
    frac=ZEIGER-i;
    Delayline=[x(n);Delayline(1:L-1)];
    %---Linear Interpolation-----
    y(n,1)=Delayline(i+1)*frac+Delayline(i)*(1-frac);
    %---Allpass Interpolation-----
    %y(n,1)=(Delayline(i+1)+(1-frac)*Delayline(i)-(1-frac)*ya_alt);
    %ya_alt=ya(n,1);
end
```

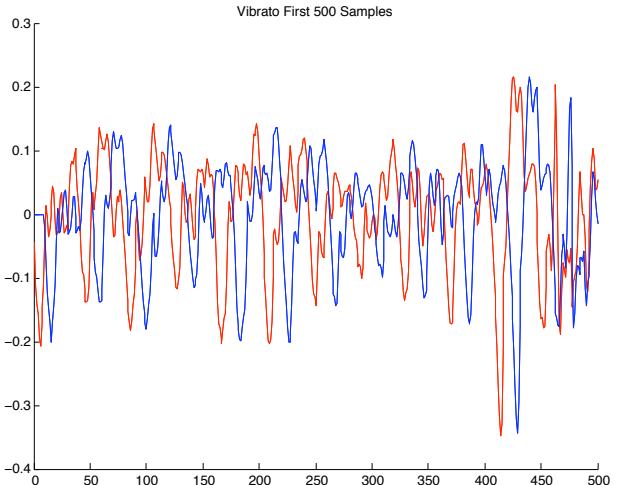


Back

Close

Vibrato MATLAB Example (Cont.)

The output from the above code is (red plot is original audio):



Click here to hear: [original audio](#), [vibrato audio](#).



Back

Close

Comb Filter Delay Effects: Flanger, Chorus, Slapback, Echo

- A few popular effects can be made with a comb filter (FIR or IIR) and some modulation
- Flanger, Chorus, Slapback, Echo same basic approach but *different sound* outputs:

Effect	Delay Range (ms)	Modulation
Resonator	0 ... 20	None
Flanger	0 ... 15	Sinusoidal (≈ 1 Hz)
Chorus	10 ... 25	Random
Slapback	25 ... 50	None
Echo	> 50	None

- Slapback (or doubling) — quick repetition of the sound,
Flanging — continuously varying LFO of delay,
Chorus — **multiple copies** of sound delayed by small random delays



Back

Close

Flanger MATLAB Code

flanger.m:

```
% Creates a single FIR delay with the delay time oscillating from
% Either 0-3 ms or 0-15 ms at 0.1 - 5 Hz

infile='acoustic.wav';
outfile='out_flanger.wav';

% read the sample waveform
[x,Fs,bits] = wavread(infile);

% parameters to vary the effect %
max_time_delay=0.003; % 3ms max delay in seconds
rate=1; %rate of flange in Hz

index=1:length(x);

% sin reference to create oscillating delay
sin_ref = (sin(2*pi*index*(rate/Fs)))';

%convert delay in ms to max delay in samples
max_samp_delay=round(max_time_delay*Fs);

% create empty out vector
y = zeros(length(x),1);
```

```
% to avoid referencing of negative samples
y(1:max_samp_delay)=x(1:max_samp_delay);

% set amp suggested coefficient from page 71 DAFX
amp=0.7;

% for each sample
for i = (max_samp_delay+1):length(x),
    cur_sin=abs(sin_ref(i));      %abs of current sin val 0-1
    % generate delay from 1-max_samp_delay and ensure whole number
    cur_delay=ceil(cur_sin*max_samp_delay);
    % add delayed sample
    y(i) = (amp*x(i)) + amp*(x(i-cur_delay));
end

% write output
wavwrite(y,Fs,outfile);
```

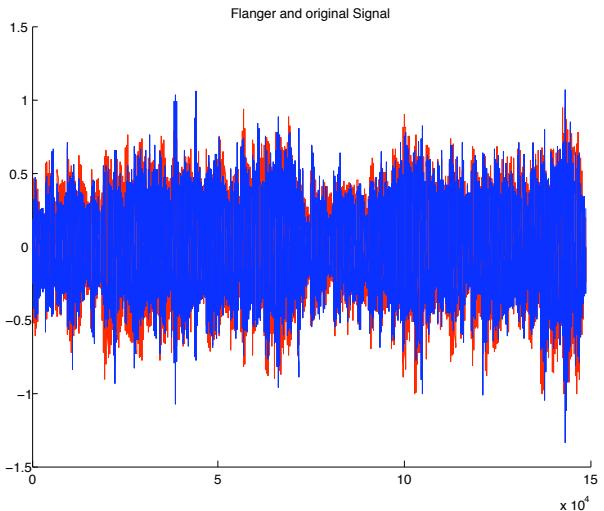


Back

Close

Flanger MATLAB Example (Cont.)

The output from the above code is (red plot is original audio):



Click here to hear: [original audio](#), [flanged audio](#).



Back

Close

Modulation

Modulation is the process where parameters of a sinusoidal signal (amplitude, frequency and phase) are modified or varied by an audio signal.

We have met some example effects that could be considered as a class of modulation already:

Amplitude Modulation — Wah-wah, Phaser

Frequency Modulation — Audio synthesis technique (**Already Studied**)

Phase Modulation — Vibrato, Chorus, Flanger

We will now introduce some Modulation effects.



Back

Close

Ring Modulation

Ring modulation (RM) is where the audio *modulator* signal, $x(n)$ is multiplied by a sine wave, $m(n)$, with a *carrier* frequency, f_c .

- This is very simple to implement digitally:

$$y(n) = x(n).m(n)$$

- Although audible result is easy to comprehend for simple signals things get more complicated for signals having numerous partials
- If the modulator is also a sine wave with frequency, f_x then one hears the sum and difference frequencies: $f_c + f_x$ and $f_c - f_x$, for example.
- When the input is *periodic* with at a fundamental frequency, f_0 , then a spectrum with amplitude lines at frequencies $|k f_0 \pm f_c|$
- Used to create robotic speech effects on old sci-fi movies and can create some odd almost non-musical effects if not used with care. (Original speech)

MATLAB Ring Modulation

Two examples, a sine wave and an audio sample being modulated by a sine wave, [ring_mod.m](#)

```
filename='acoustic.wav';  
  
% read the sample waveform  
[x,Fs,bits] = wavread(filename);  
  
index = 1:length(x);  
  
% Ring Modulate with a sine wave frequency Fc  
Fc = 440;  
carrier= sin(2*pi*index*(Fc/Fs))';  
  
% Do Ring Modulation  
y = x.*carrier;  
  
% write output  
wavwrite(y,Fs,bits,'out_ringmod.wav');
```

Click here to hear: [original audio](#), [ring modulated audio](#).



Back

Close

MATLAB Ring Modulation: Two sine waves

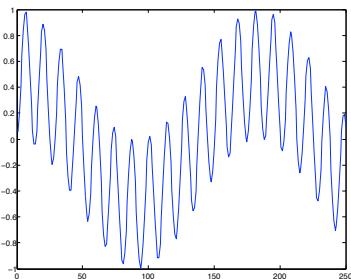
```
% Ring Modulate with a sine wave frequency Fc
Fc = 440;
carrier= sin(2*pi*index*(Fc/Fs))';

%create a modulator sine wave frequency Fx
Fx = 200;
modulator = sin(2*pi*index*(Fx/Fs))';

% Ring Modulate with sine wave, freq. Fc
y = modulator.*carrier;

% write output
wavwrite(y,Fs,bits,'twosine_ringmod.wav');
```

Output of Two sine wave ring modulation ($f_c = 440$, $f_x = 380$)



Click here to hear: [Two RM sine waves \(\$f_c = 440\$, \$f_x = 200\$ \)](#)

Amplitude Modulation

Amplitude Modulation (AM) is defined by:

$$y(n) = (1 + \alpha m(n)).x(n)$$

- Normalise the peak amplitude of $M(n)$ to 1.

- α is *depth of modulation*

$\alpha = 1$ gives maximum modulation

$\alpha = 0$ turns off modulation

- $x(n)$ is the audio **carrier** signal

- $m(n)$ is a low-frequency oscillator **modulator**.

- When $x(n)$ and $m(n)$ both sine waves with frequencies f_c and f_x respectively we have **three** frequencies: carrier, difference and sum: $f_c, f_c - f_x, f_c + f_x$.

Amplitude Modulation: Tremolo

A common audio application of AM is to produce a **tremolo** effect:

- Set modulation frequency of the sine wave to below 20Hz

The MATLAB code to achieve this is [tremolo1.m](#)

```
% read the sample waveform
filename='acoustic.wav';
[x,Fs,bits] = wavread(filename);

index = 1:length(x);

Fc = 5;
alpha = 0.5;

trem=(1+ alpha*sin(2*pi*index*(Fc/Fs)))';
y = trem.*x;

% write output
wavwrite(y,Fs,bits,'out_tremolo1.wav');
```

Click here to hear: [original audio](#), [AM tremolo audio](#).



Back

Close

Tremolo via Ring Modulation

If you ring modulate with a triangular wave (or try another waveform) you can get tremolo via RM, [tremolo2.m](#)

```
% read the sample waveform
filename='acoustic.wav';
[x,Fs,bits] = wavread(filename);

% create triangular wave LFO
delta=5e-4;
minf=-0.5;
maxf=0.5;

trem=minf:delta:maxf;
while(length(trem) < length(x) )
    trem=[trem (maxf:-delta:minf)];
    trem=[trem (minf:delta:maxf)];
end

%trim trem
trem = trem(1:length(x))';

%Ring mod with triangular, trem
y= x.*trem;

% write output
wavwrite(y,Fs,bits,'out_tremolo2.wav');
```

Click here to hear: [original audio](#), [RM tremolo audio](#).

Non-linear Processing

Non-linear Processors are characterised by the fact that they create (intentional or unintentional) harmonic and inharmonic frequency components not present in the original signal.

Three major categories of non-linear processing:

Dynamic Processing: control of signal envelop — aim to minimise harmonic distortion Examples: Compressors, Limiters

Intentional non-linear harmonic processing: Aim to introduce strong harmonic distortion. Examples: Many electric guitar effects such as distortion

Exciters/Enhancers: add additional harmonics for subtle sound improvement.



Back

Close

Overdrive, Distortion and Fuzz

Distortion plays an important part in electric guitar music, especially rock music and its variants.

Distortion can be applied as an effect to other instruments including vocals.

Overdrive — Audio at a low input level is driven by higher input levels in a non-linear curve characteristic

Distortion — a wider tonal area than overdrive operating at a higher non-linear region of a curve

Fuzz — complete non-linear behaviour, harder/harsher than distortion



Back

Close

Overdrive

For overdrive, **Symmetrical soft clipping** of input values has to be performed. A simple three layer ***non-linear soft saturation*** scheme may be:

$$f(x) = \begin{cases} 2x & \text{for } 0 \leq x < 1/3 \\ \frac{3-(2-3x)^2}{3} & \text{for } 1/3 \leq x < 2/3 \\ 1 & \text{for } 2/3 \leq x \leq 1 \end{cases}$$

- In the lower third the output is liner — multiplied by 2.
- In the middle third there is a non-linear (quadratic) output response
- Above 2/3 the output is set to 1.



Back

Close

MATLAB Overdrive Example

The MATLAB code to perform symmetrical soft clipping is,
[symclip.m](#):

```
function y=symclip(x)
% y=symclip(x)
% "Overdrive" simulation with symmetrical clipping
% x      - input
N=length(x);
y=zeros(1,N); % Preallocate y
th=1/3; % threshold for symmetrical soft clipping
           % by Schetzen Formula
for i=1:N,
    if abs(x(i))< th, y(i)=2*x(i);end;
    if abs(x(i))>=th,
        if x(i)> 0, y(i)=(3-(2-x(i)*3).^2)/3; end;
        if x(i)< 0, y(i)=-(3-(2-abs(x(i))*3).^2)/3; end;
    end;
    if abs(x(i))>2*th,
        if x(i)> 0, y(i)=1;end;
        if x(i)< 0, y(i)=-1;end;
    end;
end;
```

MATLAB Overdrive Example (Cont.)

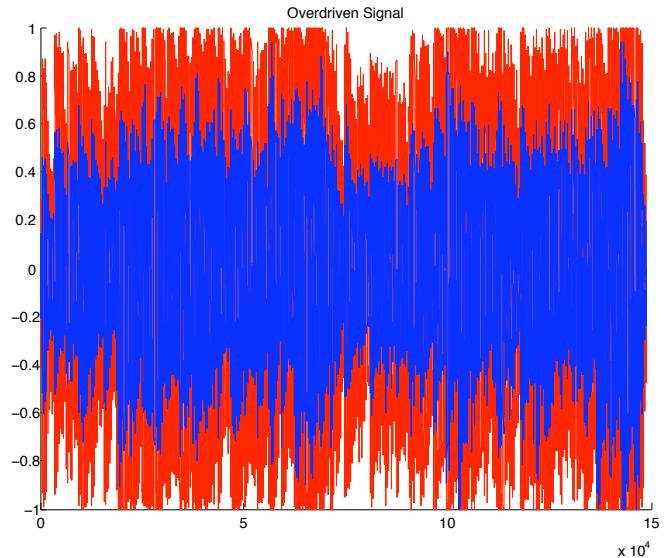
An **overdriven signal** looks like this , [overdrive_eg.m](#):

```
% read the sample waveform
filename='acoustic.wav';
[x,Fs,bits] = wavread(filename);

% call symmetrical soft clipping
% function
y = symclip(x);

% write output
wavwrite(y,Fs,bits, ...
    'out_overdrive.wav');

figure(1);
hold on
plot(y,'r');
plot(x,'b');
title('Overdriven Signal');
```



Click here to hear: [original audio](#), [overdriven audio](#).

Distortion/Fuzz

A non-linear function commonly used to simulate distortion/fuzz is given by:

$$f(x) = \frac{x}{|x|} (1 - e^{\alpha x^2 / |x|})$$

- This a non-linear exponential function:
- The gain, α , controls level of distortion/fuzz.
- Common to mix part of the distorted signal with original signal for output.



Back

Close

MATLAB Fuzz Example

The MATLAB code to perform symmetrical soft clipping is,
fuzzexp.m:

```
function y=fuzzexp(x, gain, mix)
% y=fuzzexp(x, gain, mix)
% Distortion based on an exponential function
% x      - input
% gain   - amount of distortion, >0->
% mix    - mix of original and distorted sound, 1=only distorted
q=x*gain/max(abs(x));
z=sign(-q).* (1-exp(sign(-q).*q));
y=mix*z*max(abs(z))/max(abs(z))+(1-mix)*x;
y=y*max(abs(y))/max(abs(y));
```

Note: function allows to **mix** input and fuzz signals at output



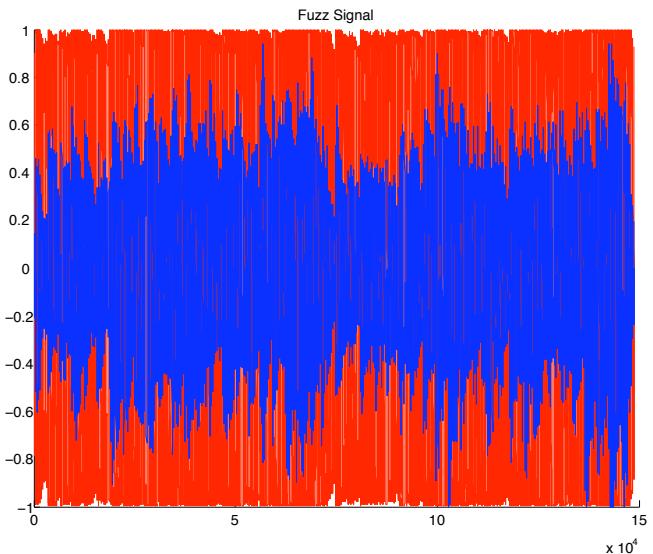
Back

Close

MATLAB Fuzz Example (Cont.)

An **fuzzed up signal** looks like this , fuzz_eg.m:

```
filename='acoustic.wav';  
  
% read the sample waveform  
[x,Fs,bits] = wavread(filename);  
  
% Call fuzzexp  
gain = 11; % Spinal Tap it  
mix = 1; % Hear only fuzz  
y = fuzzexp(x,gain,mix);  
  
% write output  
wavwrite(y,Fs,bits,'out_fuzz.wav');
```



Click here to hear: original audio, Fuzz audio.

Graphic/Image File Formats

Common graphics and image file formats:

- <http://www.dcs.ed.ac.uk/home/mxr/gfx/> — comprehensive listing of various formats.
- See *Encyclopedia of Graphics File Formats* book in library
- Most formats incorporate *compression*, including **lossless** or **lossy**
- Graphics, video and audio compression techniques in next Chapter.



Back

Close

Graphic/Image Data Structures

"A picture is worth a thousand words, but it uses up three thousand times the memory."

- A digital image consists of many picture elements, termed **pixels**.
- The number of pixels determine the quality of the image (**resolution**).
- Higher resolution always yields better quality.
- A *bit-map* representation stores the graphic/image data in the same manner that the computer monitor contents are stored in video memory.

Monochrome/Bit-Map Images



Figure 1: Sample Monochrome Bit-Map Image

- Each pixel is stored as a single bit (0 or 1)
- A 640×480 monochrome image requires 37.5 KB of storage.
- *Dithering* is often used for displaying monochrome images

Gray-scale Images



Figure 2: Example of a Gray-scale Bit-map Image

- Each pixel is usually stored as a byte (value between 0 to 255)
- A dark pixel may have a value of 10; a bright one may be 240
- A 640 x 480 greyscale image requires over 300 KB of storage.

Dithering

- Dithering is often used when converting greyscale images to monochrome ones e.g. for printing
- The main strategy is to replace a **pixel value** (from 0 to 255) by a **larger pattern** (e.g. 4×4) such that the number of printed dots approximates the greyscale level of the original image
- If a pixel is replaced by a 4×4 array of dots, the intensities it can approximate from 0 (no dots) to 16 (full dots).
- Given a 4×4 dither matrix e.g.

$$\begin{pmatrix} 0 & 8 & 2 & 10 \\ 12 & 4 & 14 & 6 \\ 3 & 11 & 1 & 9 \\ 15 & 7 & 13 & 5 \end{pmatrix}$$

we can re-map pixel values from 0–255 to a new range 0–16 by dividing the value by (256/17) (and rounding down to integer).



Back

Close

Dithering (cont.)

- A simple approach: replace each pixel by a 4×4 dots (monochrome pixels). If the remapped intensity is $>$ the dither matrix entry, put a dot at the position (set to 1) otherwise set to 0.
- Note that the size of the dithered image may be much larger. Since each pixel is replaced by 4×4 array of dots, the image becomes **16 times** as large.
- To keep the image size: an **ordered dither** produces an output pixel with value 1 iff the remapped intensity level **just at the pixel position** is greater than the corresponding matrix entry.



24-bit Colour Images



Figure 3: Example of 24-Bit Colour Image

- Each pixel is represented by three bytes (e.g., RGB)
- Supports $256 \times 256 \times 256$ possible combined colours (16,777,216)
- A 640×480 24-bit colour image would require 921.6 KB of storage
- Some colour images are 32-bit images,
 - the extra byte of data for each pixel is used to store an *alpha* value representing special effect information



Back

Close

8-bit Colour Images



Figure 4: Example of 8-Bit Colour Image

- One byte for each pixel
- Supports 256 out of the millions possible
- Acceptable colour quality
- Requires Colour Look-Up Tables (**LUTs**)
- A 640×480 8-bit colour image requires 307.2 KB of storage (the same as 8-bit greyscale)



Back

Close

Colour Look-Up Tables (LUTs)

- Store only the index of the colour LUT for each pixel
- Look up the table to find the colour (RGB) for the index
- LUT needs to be built when converting 24-bit colour images to 8-bit: grouping similar colours (each group assigned a colour entry)
- Possible for **palette animation** by changing the colour map

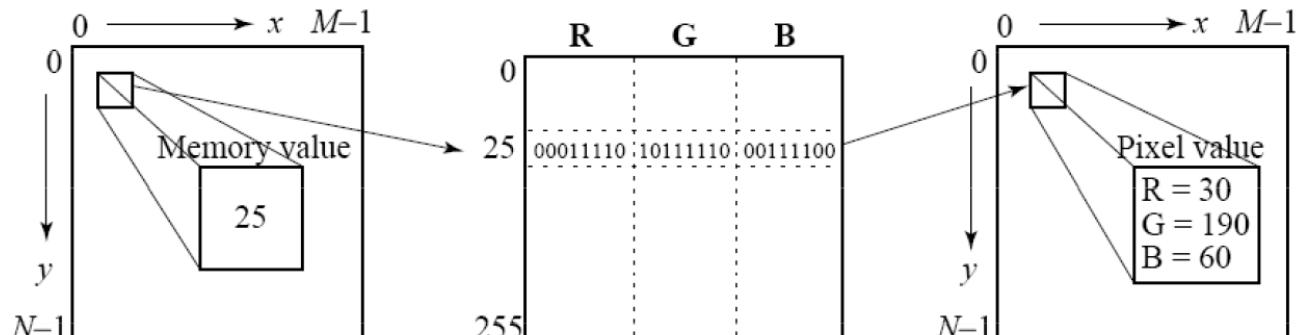


Figure 5: Colour LUT for 8-bit Colour Images

Standard System Independent Formats

GIF (GIF87a, GIF89a)

- Graphics Interchange Format (GIF) devised by the UNISYS Corp. and CompuServe, initially for transmitting graphical images over phone lines via modems
- Uses the Lempel-Ziv Welch algorithm (a form of Huffman Coding), modified slightly for image scan line packets (line grouping of pixels) so **lossless** — **Algorithm Soon**
- Limited to only 8-bit (256) colour images, suitable for images with few distinctive colours (e.g., graphics drawing)
- Supports *interlacing*
- GIF89a: supports **simple animation, transparency index etc.**



Back

Close

JPEG

- A standard for photographic image compression created by the Joint Photographic Experts Group
- Takes advantage of limitations in the human vision system to achieve high rates of compression
- **Lossy** compression which allows user to set the desired level of quality/compression
- **Algorithm Soon** — Detailed discussions in next chapter on compression.



Back

Close

TIFF

- Tagged Image File Format (TIFF), stores many different types of images (e.g., monochrome, greyscale, 8-bit & 24-bit RGB, etc.) → tagged
- Developed by the Aldus Corp. in the 1980's and later supported by the Microsoft
- TIFF is a lossless format (when not utilising the new JPEG tag which allows for JPEG compression)
- It does not provide any major advantages over JPEG and is not as user-controllable it appears to be declining in popularity



Back

Close

PNG

- PNG stands for Portable Network Graphics, meant to supersede GIF standard
- Features of PNG
 - Support up to **48 bits** per pixel – more accurate colours
 - Support description of **gamma-correction** and **alpha-channel** for controls such as transparency
 - Support progress display through 8×8 blocks.

Postscript/Encapsulated Postscript

- A typesetting language which includes text as well as vector/structured graphics and bit-mapped images
- Used in several popular graphics programs (Illustrator, FreeHand)
- Does not provide compression, files are often large
- Although able to link to external compression applications

System Dependent Formats

Microsoft Windows: BMP

- A system standard graphics file format for Microsoft Windows
- Used in Many PC Graphics programs, Cross-platform support
- It is capable of storing 24-bit bitmap images

Macintosh: PAINT and PICT

- PAINT was originally used in MacPaint program, initially only for 1-bit monochrome images.
- PICT format was originally used in MacDraw (a vector based drawing program) for storing structured graphics
- Still an underlying Mac format



Back

Close

X-window: XBM

- Primary graphics format for the X Window system
- Supports 24-bit colour bitmap
- Many public domain graphic editors, e.g., *xv*
- Used in X Windows for storing icons, pixmaps, backdrops, etc.



Back

Close

Basics of Colour: Image and Video

Light and Spectra

- Visible light is an electromagnetic wave in the 400nm - 700 nm range.
- Most light we see is not one wavelength, it's a combination of many wavelengths (Fig. 6).

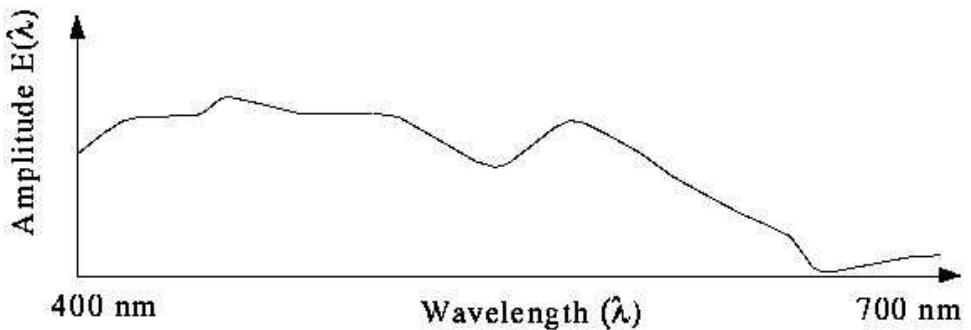
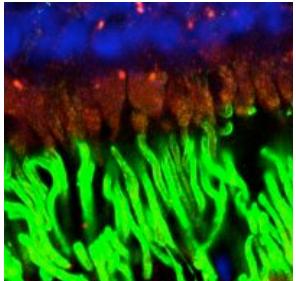
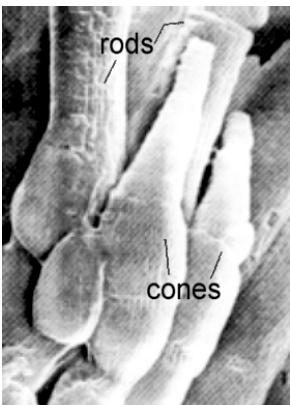
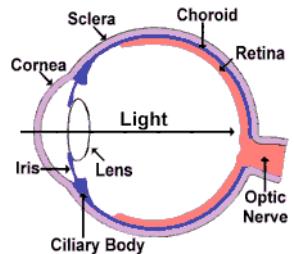


Figure 6: Light Wavelengths

- The profile above is called a *spectra*.

The Human Eye

- The eye is basically similar to a camera
- It has a lens to focus light onto the Retina of eye
- Retina full of **neurons**
- Each neuron is either a *rod* or a *cone*.
- Rods are not sensitive to colour.



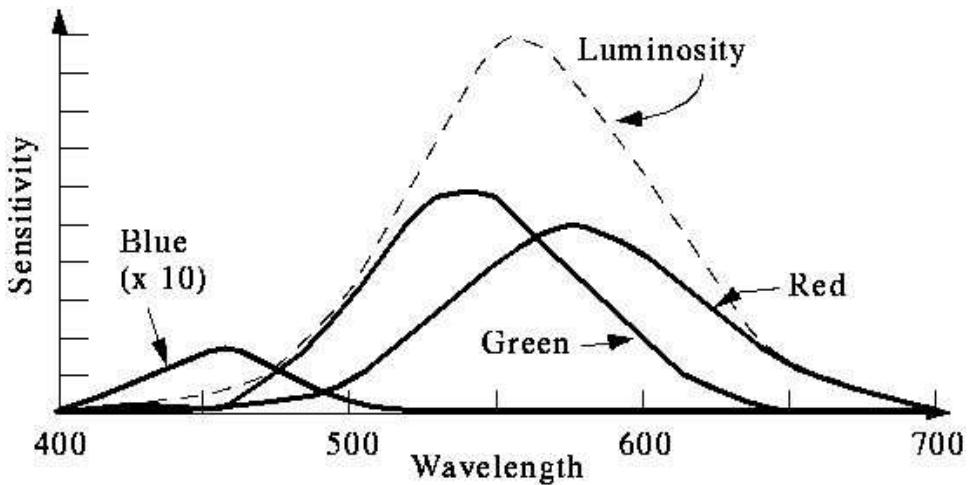


Figure 7: Cones & Luminous-efficiency Function of the Human Eye

- Human eyes can't tell lights with different spectra as long as they have the same stimuli.



Back

Close



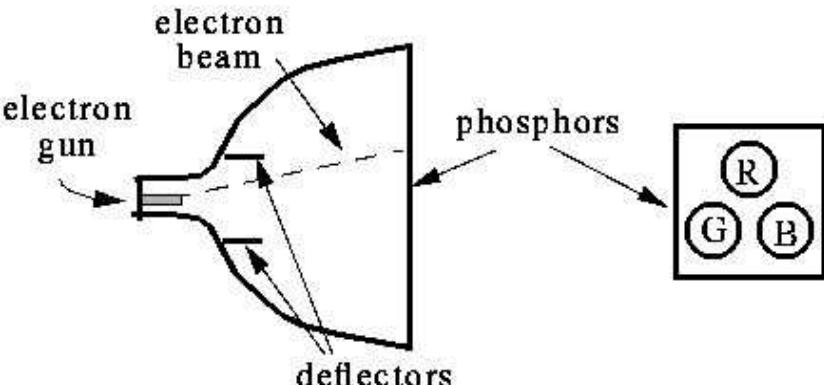
Figure 8: Original Color Image

- Colour Space is made up of Red, Green and Blue intensity components



Red, Green, Blue (RGB) Image Space

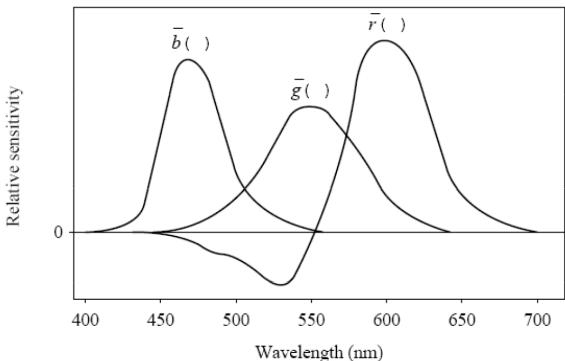
Red, Green, Blue (RGB) Respective Intensities



- CRT displays have three phosphors (RGB) which produce a combination of wavelengths when excited with electrons.
- The *gamut* of colours is all colours that can be reproduced using the three primaries
- The gamut of a colour monitor is smaller than that of color models, E.g. CIE (LAB) Model — **see later.**

Colour-Matching Functions

- To match a given colour with the three **colour primaries**, a subject is asked to separately adjust the brightness of the three primaries until the resulting light match most closely to the desired colour
- The amounts of R/G/B the subject selects to match each single-wavelength light forms the **color-matching curves**:



CIE Chromaticity Diagram

Does a set of primaries exist that span the space with only positive coefficients?

- Yes, but not the pure colours.
- In 1931, the CIE defined three standard primaries (X , Y , Z) . The Y primary was intentionally chosen to be identical to the luminous-efficiency function of human eyes.
- All visible colours are in a *horseshoe* shaped cone in the X-Y-Z space. Consider the plane $X+Y+Z=1$ and project it onto the X-Y plane, we get the *CIE chromaticity diagram* as shown overleaf.

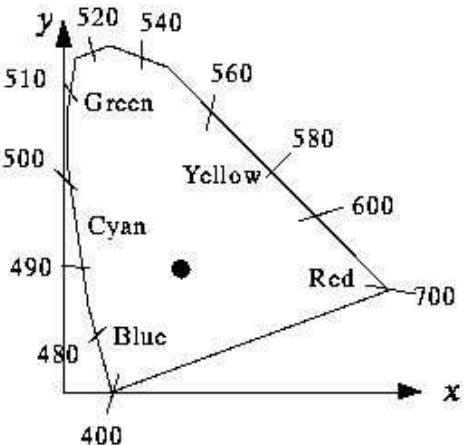
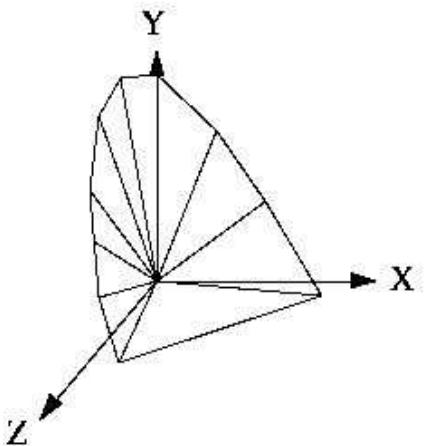


Back

Close

CIE Chromaticity Diagram (Cont.)

- CIE chromaticity diagram:



- The edges represent the *pure* colours (sine waves at the appropriate frequency)
- White (a blackbody radiating at 6447 kelvin) is at the *dot*
- When added, any two colours (points on the CIE diagram) produce a point on the line between them.

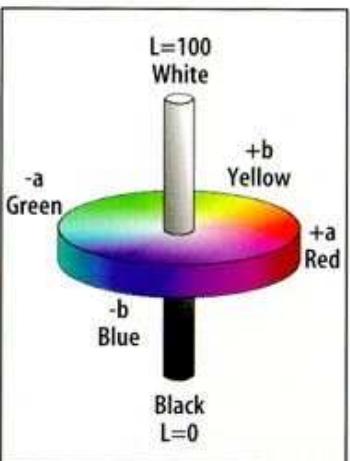
L*a*b* (Lab) Colour Model

- A refined CIE model, named CIE L*a*b* in 1976

- **Luminance:** L

Chrominance: a – ranges from green to red, b – ranges from blue to yellow

- Used by *Photoshop*



Lab model



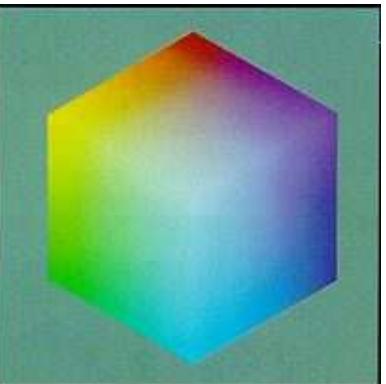
Original Color Image



L, A, B Image Intensities

Colour Image and Video Representations

- Recap: A black and white image is a 2-D array of integers.
- Recap: A colour image is a 2-D array of (R,G,B) integer triplets. These triplets encode how much the corresponding phosphor should be excited in devices such as a monitor.
- Example is shown:



Beside the RGB representation, YIQ and YUV are the two commonly used in video.

YIQ Colour Model

- YIQ is used in colour TV broadcasting, it is downward compatible with B/W TV.
- Y (luminance) is the CIE Y primary.

$$Y = 0.299R + 0.587G + 0.114B$$

- the other two vectors:

$$I = 0.596R - 0.275G - 0.321B \quad Q = 0.212R - 0.528G + 0.311B$$

- The YIQ transform:

$$\begin{bmatrix} Y \\ I \\ Q \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ 0.596 & -0.275 & -0.321 \\ 0.212 & -0.528 & -0.311 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

- I is red-orange axis, Q is roughly orthogonal to I.
- Eye is most sensitive to Y, next to I, next to Q. In NTSC, 4 MHz is allocated to Y, 1.5 MHz to I, 0.6 MHz to Q.



Back

Close



Original Color Image



Y, I, Q Image Intensities

YUV Color Model

- Established in 1982 to build digital video standard
- Video is represented by a sequence of fields (odd and even lines). Two fields make a frame.
- Works in PAL (50 fields/sec) or NTSC (60 fields/sec)
- Uses the Y, U, V colour space

$$Y = 0.299R + 0.587G + 0.114B, U = B - Y, V = R - Y$$

- The YUV Transform:

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.299 & -0.587 & 0.886 \\ 0.701 & -0.587 & -0.114 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$



Back

Close

YCrCb (CCIR 601) Colour Model

- Similar to YUV
- YUV is normalised by a scaling

$$Cb = (B - Y)/1.772$$

$$Cr = (R - Y)/1.402$$

$$\begin{bmatrix} Y \\ Cr \\ Cb \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.169 & -0.331 & 0.500 \\ 0.500 & -0.419 & -0.081 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$



Original Color Image



Y, Cr, Cb Image Intensities

The CMY Colour Model

- Cyan, Magenta, and Yellow (CMY) are complementary colours of RGB (Fig. 9). They can be used as *Subtractive Primaries*.
- CMY model is mostly used in printing devices where the colour pigments on the paper absorb certain colours (e.g., no red light reflected from cyan ink).

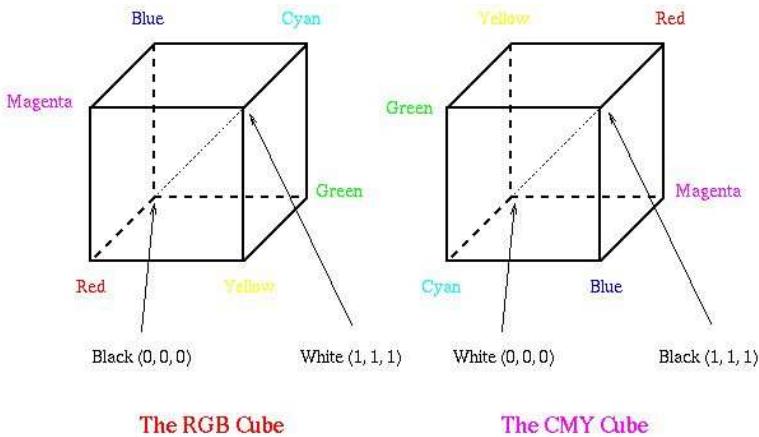
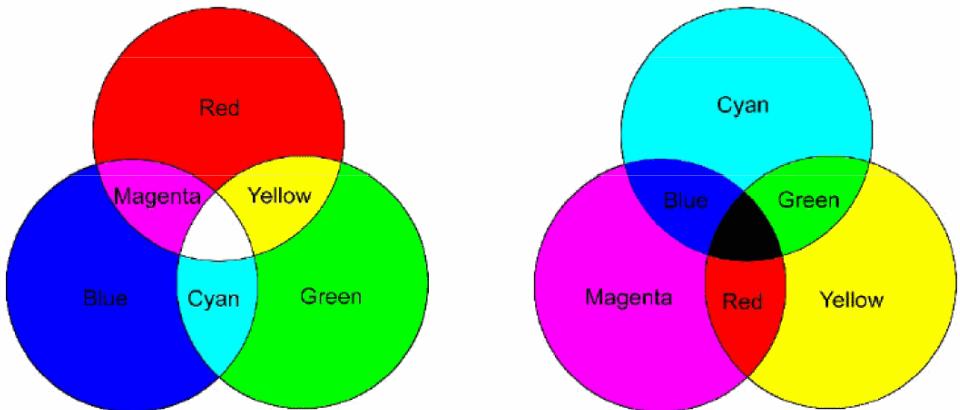


Figure 9: The RGB and CMY Cubes

The CMY Colour Model (cont.)

- Combinations of **additive** and **subtractive** colours.



Conversion between RGB and CMY

E.g., convert **White** from (1, 1, 1) in RGB to (0, 0, 0) in CMY.

$$\begin{bmatrix} C \\ M \\ Y \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

$$\begin{bmatrix} R \\ G \\ B \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix} - \begin{bmatrix} C \\ M \\ Y \end{bmatrix}$$

CMYK Color Model

- Sometimes, an alternative CMYK model (K stands for *Black*) is used in colour printing (e.g., to produce darker black than simply mixing CMY). where

$$K = \min(C, M, Y),$$

$$C = C - K,$$

$$M = M - K,$$

$$Y = Y - K.$$

CMYK Colour Space



Original Color Image



C, M, Y, K image Intensities

Summary of Colour

- Colour images are encoded as triplets of values.
- Three common systems of encoding in video are RGB, YIQ, and YCrCb.
- Besides the hardware-oriented colour models (i.e., RGB, CMY, YIQ, YUV), HSB (Hue, Saturation, and Brightness, e.g., used in Photoshop) and HLS (Hue, Lightness, and Saturation) are also commonly used.
- YIQ uses properties of the human eye to prioritise information. Y is the black and white (luminance) image, I and Q are the colour (chrominance) images. YUV uses similar idea.
- YUV/YCrCb is a standard for digital video that specifies image size, and decimates the chrominance images (for 4:2:2 video) — **more soon**.



Back

Close

Basics of Video

Types of Colour Video Signals

- **Component video** – each primary is sent as a separate video signal.
 - The primaries can either be RGB or a luminance-chrominance transformation of them (e.g., YIQ, YUV).
 - Best colour reproduction
 - Requires more bandwidth and good synchronisation of the three components
- **Composite video** – colour (chrominance) and luminance signals are mixed into a single carrier wave. Some interference between the two signals is inevitable.
- **S-Video** (Separated video, e.g., in S-VHS) – a compromise between component analog video and the composite video. It uses two lines, one for luminance and another for composite chrominance signal.



Back

Close

NTSC Video

- 525 scan lines per frame, 30 frames per second (or be exact, 29.97 fps, 33.37 msec/frame)
- Aspect ratio 4:3
- Interlaced, each frame is divided into 2 fields, 262.5 lines/field
- 20 lines reserved for control information at the beginning of each field
 - So a maximum of 485 lines of visible data
 - Laser disc and S-VHS have actual resolution of \approx 420 lines
 - Ordinary TV – \approx 320 lines



Back

Close

NTSC Video Colour and Analog Compression

- Colour representation:

- NTSC uses YIQ colour model.
- Composite = $Y + I \cos(Fsc t) + Q \sin(Fsc t)$,
where Fsc is the frequency of colour subcarrier
- Basic Compression Idea

Eye is most sensitive to Y, next to I, next to Q.

- This is STILL Analog Compression:

In NTSC,

- * 4 MHz is allocated to Y,
- * 1.5 MHz to I,
- * 0.6 MHz to Q.

- Similar (easier to work out) Compression (Part of) in digital compression — **more soon**

PAL Video

- 625 scan lines per frame, 25 frames per second (40 msec/frame)
- Aspect ratio 4:3
- Interlaced, each frame is divided into 2 fields, 312.5 lines/field
- Colour representation:
 - PAL uses YUV (YCrCb) colour model
 - composite =
$$Y + 0.492 \times U \sin(Fsc t) + 0.877 \times V \cos(Fsc t)$$
 - In PAL, 5.5 MHz is allocated to Y, 1.8 MHz each to U and V.

MATLAB Colour functions

Example MATLAB's image processing toolbox colour space functions:

Colormap manipulation :

`colormap` — Set or get colour lookup table

`rgbplot` — Plot RGB colormap components

`cmpermute` — Rearrange colours in colormap.

Colour space conversions :

`hsv2rgb/rgb2hsv` — Convert HSV values/RGB colour space

`lab2double/lab2uint16/lab2uint8` — Convert Lab colour values to double etc.

`ntsc2rgb/rgb2ntsc` — Convert NTSC (**YIQ**)/RGB colour values

`ycbcr2rgb/rgb2ycbcr` — Convert YCbCr/RGB colour



Back

Close

Chroma Subsampling

Chroma subsampling is a method that stores color information at lower resolution than intensity information.

Why is this done? — **COMPRESSION**

- Human visual system (HVS) more sensitive to variations in brightness than colour.
- So devote more bandwidth to Y than the color difference components Cr/I and Cb/Q.
 - HVS is less sensitive to the position and motion of color than luminance
 - Bandwidth can be optimised by storing more luminance detail than color detail.
- Reduction results in almost no perceivable visual difference.



Back

Close

How to Chroma Subsample?

Use **color difference components**. The signal is divided into:

luma (Y) component and

Chroma — two color difference components which we subsample in some way to reduce its bandwidth

How to subsample for chrominance?

The subsampling scheme is commonly expressed as a three part ratio (e.g. 4:2:2):

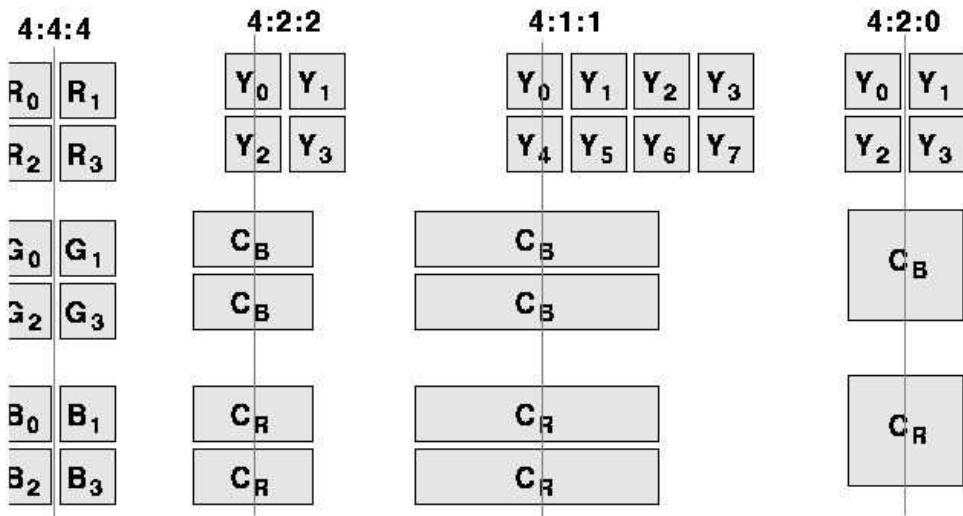


Chroma Subsample 3 Part Ratio Explained

Each part of the three part ratio is respectively:

- 1: Luma (Y) or Red (R) — horizontal sampling reference
(originally, as a multiple of 3.579 MHz in the NTSC analog television system — rounded to 4)
- 2: Cr/I/G — horizontal factor (relative to first digit)
- 3: Cb/Q/B – horizontal factor (relative to first digit), except when **zero**.
 - **Zero** indicates that Cb (Q/B) horizontal factor is equal to second digit, **and**,
 - Both Cr (I/G) and Cb (Qb) are subsampled 2:1 vertically.

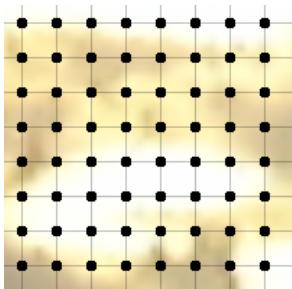
Chroma Subsampling Examples



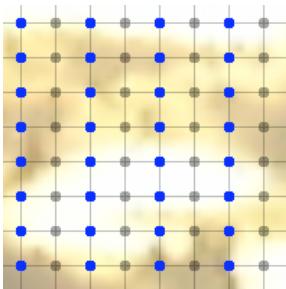
- 4:4:4 — no subsampling in any band — equal ratios.
- 4:2:2 → Two chroma components are sampled at half the sample rate of luma, horizontal chroma resolution halved.
- 4:1:1 → Horizontally subsampled by a factor of 4.
- 4:2:0 → Subsampled by a factor of 2 in both the horizontal and vertical axes

Chroma Subsampling: How to Compute?

- Simply different frequency sampling of digitised signal
- Digital Subsampling: For 4:4:4, 4:2:2 and 4:1:1
 - Perform 2x2 (or 1x2, or 1x4) chroma subsampling
 - Subsample horizontal and, where applicable, vertical directions
 - *i.e.* Choose every second, fourth pixel value.

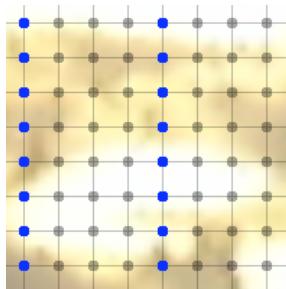


4:4:4



4:2:2

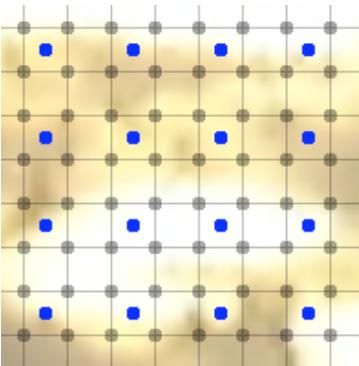
Subsampling



4:1:1

Chroma Subsampling: How to Compute? (Cont.)

- For 4:2:0, Cb and Cr are effectively centred vertically halfway between image rows.:
 - Break the image into 2x2 pixel blocks and
 - Stores the **average** color information for each 2x2 pixel group.



4:2:0 Subampling

Chroma Subsampling in MATLAB

The MATLAB function `imresize()` readily achieves all our subsampling needs:

`IMRESIZE` Resize image.

`IMRESIZE` resizes an image of any type using the specified interpolation method. Supported interpolation methods include:

'nearest' --- (default) nearest neighbour interpolation
'bilinear' bilinear interpolation

`B = IMRESIZE(A,M,METHOD)` returns an image that is M times the size of A. If M is between 0 and 1.0, B is smaller than A. If M is greater than 1.0, B is larger than A.

`B = IMRESIZE(A,[MROWS MCOLS],METHOD)` returns an image of size MROWS-by-MCOLUMNS.

After MATLAB colour conversion to YUV/YIQ:

- Use `nearest` for 4:2:2 and 4:2:1 and scale the `MCOLS` to half or quarter the size of the image.
- Use `bilinear` (to average) for 4:2:0 and set scale to half.

See next Lab worksheet



Back

Close

Digital Chroma Subsampling Errors (1)

This sampling process introduces two kinds of errors:

1. A minor problem is that colour is typically stored at only half the horizontal and vertical resolution as the original image — *subsampling*.

This is not a real problem:

- Recall: The human eye has lower resolving power for colour than for intensity.
- Nearly all digital cameras have lower resolution for colour than for intensity, so there is no high resolution colour information present in digital camera images.



Back

Close

Digital Chroma Subsampling Errors (2)

2. Another issue: The subsampling process demands two conversions of the image:

- From the original RGB representation to an intensity+colour (YIQ/YUV) representation , and
- Then back again (YIQ/YUV → RGB) when the image is displayed.
- Conversion is done in integer arithmetic — some round-off error is introduced.
 - This is a much smaller effect,
 - But (slightly) affects the colour of (typically) one or two percent of the pixels in an image.



Back

Close

Aliasing in Images

Stair-stepping — Stepped or jagged edges of angled lines,
e.g., at the slanted edges of letters.

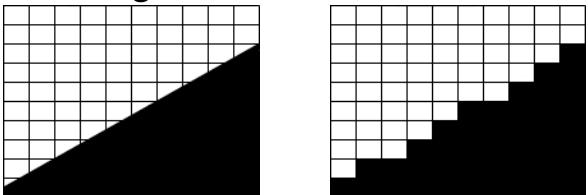


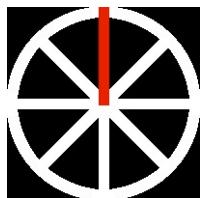
Image Zooming — changing resolution or not acquiring image in adequate resolution, e.g. digital zoom on cameras, digital scanning.
(see [zoom_alias.m](#))



Explanation: Simply Application of Nyquist's Sampling Theorem:
Zooming in by a factor n divides the sample resolution by n

Aliasing in Video

Temporal aliasing - e.g., rotating wagon wheel spokes apparently reversing direction, (see [aliasing_wheel.m](#) + [spokesR.gif](#)):



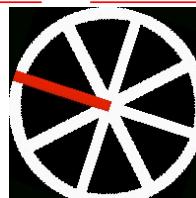
Frame 1



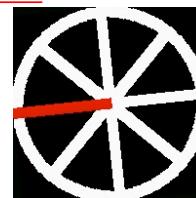
Frame2



Frame3



Frame4



Frame5

Below Nyquist Video

At Nyquist Video

Above Nyquist Video

Raster scan aliasing — e.g., twinkling or strobing effects on sharp horizontal lines, (see [raster_aliasing.m](#) + [barbara.gif](#)):

Strobing Alias Video

Strobing Alias Frequency Distributions Video

Interlacing aliasing — Some video is interlaced, this effectively halves the sampling frequency. e.g.: [Interlacing Aliasing effects](#)

Image Aliasing — Stair-stepping/Zooming aliasing effects as images.

Explanation: Simply Application of Nyquist's Sampling Theorem

Compression: Basic Algorithms

Recap: The Need for Compression

Raw Video, Image and Audio files can be very large:

Uncompressed Audio

1 minute of Audio:

<i>Audio Type</i>	<i>44.1 KHz</i>	<i>22.05 KHz</i>	<i>11.025 KHz</i>
<i>16 Bit Stereo</i>	10.1 Mb	5.05 Mb	2.52 Mb
<i>16 Bit Mono</i>	5.05 Mb	2.52 Mb	1.26 Mb
<i>8 Bit Mono</i>	2.52 Mb	1.26 Mb	630 Kb

Uncompressed Images:

<i>Image Type</i>	<i>File Size</i>
512 x 512 Monochrome	0.25 Mb
512 x 512 8-bit colour image	0.25 Mb
512 x 512 24-bit colour image	0.75 Mb

Video

Can also involve: Stream of audio **plus** video imagery.

Raw Video – Uncompressed Image Frames, 512x512 True Colour, 25 fps, 1125 MB Per Min

HDTV — **Gigabytes** per minute uncompressed (1920 × 1080, true colour, 25fps: 8.7GB per min)

- Relying on higher bandwidths is not a good option — M25 Syndrome.
- Compression **HAS TO BE** part of the representation of audio, image and video formats.



Back

Close

Classifying Compression Algorithms

What is Compression?

E.g.: Compression ASCII Characters EIEIO

$\underbrace{01000101}_{E(69)} \underbrace{01001001}_{I(73)} \underbrace{01000101}_{E(69)} \underbrace{01001001}_{I(73)} \underbrace{01001111}_{O(79)} = 5 \times 8 = 40 \text{ bits}$

The Main aim of Data Compression is find a way to use less bits per character, E.g.:

$\underbrace{xx}_{E(2\text{bits})} \underbrace{yy}_{I(2\text{bits})} \underbrace{xx}_{E(2\text{bits})} \underbrace{yy}_{I2\text{bits}} \underbrace{zzz}_{O(3\text{bits})} = \underbrace{(2 \times 2)}_{2 \times E} + \underbrace{(2 \times 2)}_{2 \times I} + \underbrace{3}_{O} = 11$
bits

Note: We usually consider character sequences here for simplicity. Other token streams can be used — e.g. Vectorised Image Blocks, Binary Streams.



Back

Close

Compression in Multimedia Data

Compression basically employs redundancy in the data:

- Temporal — in 1D data, 1D signals, Audio etc.
- Spatial — correlation between neighbouring pixels or data items
- Spectral — correlation between colour or luminescence components.
This uses the frequency domain to exploit relationships between frequency of change in data.
- Psycho-visual — exploit perceptual properties of the human visual system.

Lossless v Lossy Compression

Compression can be categorised in two broad ways:

Lossless Compression — after decompression gives an exact copy of the original data

Examples: Entropy Encoding Schemes (Shannon-Fano, Huffman coding), arithmetic coding,LZW algorithm used in GIF image file format.

Lossy Compression — after decompression gives ideally a ‘close’ approximation of the original data, in many cases perceptually lossless but a byte-by-byte comparision of files shows differences.

Examples: Transform Coding —
FFT/DCT based quantisation used in JPEG/MPEG differential encoding, vector quantisation



Back

Close

Why do we need Lossy Compression?

- Lossy methods for **typically** applied to high resolution audio, image compression
- **Have to be employed** in video compression (apart from special cases).

Basic reason:

- ***Compression ratio*** of lossless methods (e.g., Huffman coding, arithmetic coding, LZW) is not high enough.



Back

Close

Lossless Compression Algorithms

- Repetitive Sequence Suppression
- Run-Length Encoding (RLE)
- Pattern Substitution
- Entropy Encoding
 - Shannon-Fano Algorithm
 - Huffman Coding
 - Arithmetic Coding
- Lempel-Ziv-Welch (LZW) Algorithm



Back

Close

Lossless Compression Algorithms: Repetitive Sequence Suppression

- Fairly straight forward to understand and implement.
- Simplicity is their downfall: **NOT best compression ratios**.
- Some methods have their applications, *e.g. Component of JPEG, Silence Suppression*.



Back

Close

Simple Repetition Suppression

If a sequence a series on n successive tokens appears

- Replace series with a token and a count number of occurrences.
 - Usually need to have a special *flag* to denote when the repeated token appears

For Example:

we can replace with:

894 f.32

where f is the flag for zero.

Simple Repetition Suppression: How Much Compression?

Compression savings depend on the content of the data.

Applications of this simple compression technique include:

- Suppression of zero's in a file (*Zero Length Suppression*)
 - Silence in audio data, Pauses in conversation *etc.*
 - Bitmaps
 - Blanks in text or program source files
 - Backgrounds in simple images
- Other regular image or data tokens



Back

Close

Lossless Compression Algorithms: Run-length Encoding (RLE)

This encoding method is frequently applied to graphics-type images (or pixels in a scan line) — simple compression algorithm in its own right.

It is also a component used in **JPEG compression pipeline**.

Basic RLE Approach (e.g. for images):

- Sequences of image elements X_1, X_2, \dots, X_n (Row by Row)
- Mapped to pairs $(c_1, l_1), (c_2, l_2), \dots, (c_n, l_n)$
where c_i represent image intensity or colour and l_i the length of the i th run of pixels
- (Not dissimilar to zero length suppression above).



Run-length Encoding Example

Original Sequence (1 Row):

11112223333311112222

can be encoded as:

(1, 4), (2, 3), (3, 6), (1, 4), (2, 4)

How Much Compression?

The savings are dependent on the data: In the **worst case** (**Random Noise**) encoding is more heavy than original file:
2*integer rather than 1* integer if original data is integer vector/array.

MATLAB example code:

[rle.m](#) (run-length encode) , [rld.m](#) (run-length decode)



Back

Close

Lossless Compression Algorithms: Pattern Substitution

This is a simple form of statistical encoding.

Here we substitute a frequently repeating pattern(s) with a code.

The code is shorter than the pattern giving us compression.

A simple Pattern Substitution scheme could employ predefined codes



Back

Close

Simple Pattern Substitution Example

For example replace all occurrences of pattern of characters 'and' with the predefined code '&'.

So:

and you and I

Becomes:

& you & I

Similar for other codes — commonly used words



Back

Close

Token Assignment

More typically tokens are assigned to according to frequency of occurrence of patterns:

- Count occurrence of tokens
- Sort in Descending order
- Assign some symbols to highest count tokens

A predefined symbol table may be used *i.e.* assign code i to token T . (**E.g. Some dictionary of common words/tokens**)

However, it is more usual to dynamically assign codes to tokens.

The entropy encoding schemes **below** basically attempt to decide the optimum assignment of codes to achieve the best compression.



Back

Close

Lossless Compression Algorithms

Entropy Encoding

Multimedia
CM0340

333

- Lossless Compression frequently involves some form of **entropy encoding**
- Based on **information theoretic techniques**.



Back

Close

Basics of Information Theory

According to Shannon, the **entropy** of an information source S is defined as:

$$H(S) = \eta = \sum_i p_i \log_2 \frac{1}{p_i}$$

where p_i is the probability that symbol S_i in S will occur.

- $\log_2 \frac{1}{p_i}$ indicates the amount of information contained in S_i , i.e., the number of bits needed to code S_i .
- For example, in an image with uniform distribution of gray-level intensity, i.e. $p_i = 1/256$, then
 - The number of bits needed to code each gray level is 8 bits.
 - The entropy of this image is 8.



Back

Close

The Shannon-Fano Algorithm — Learn by Example

This is a basic information theoretic algorithm.

A simple example will be used to illustrate the algorithm:

A finite token Stream:

ABBAAAACDEAAABBBDDEEEAAA.....

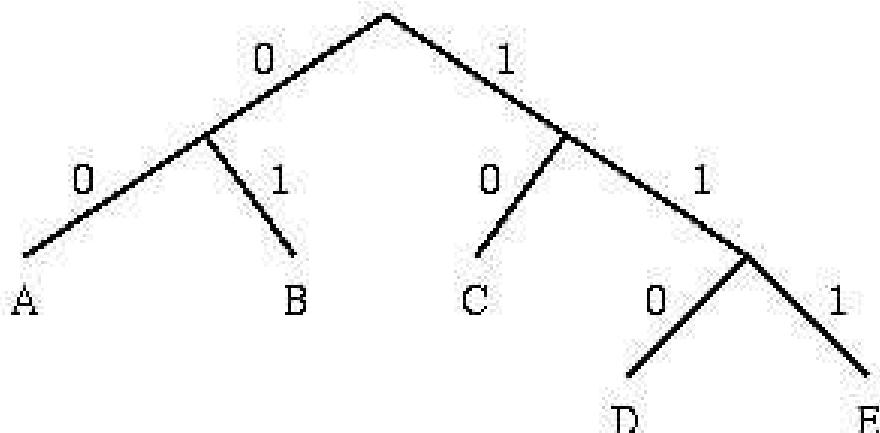
Count symbols in stream:

Symbol	A	B	C	D	E
<hr/>					
Count	15	7	6	6	5



Encoding for the Shannon-Fano Algorithm:

- A top-down approach
 1. Sort symbols (Tree Sort) according to their frequencies/probabilities, e.g., ABCDE.
 2. Recursively divide into two parts, each with approx. same number of counts.



3. Assemble code by depth first traversal of tree to symbol node

Symbol	Count	$\log(1/p)$	Code	Subtotal (# of bits)
A	15	1.38	00	30
B	7	2.48	01	14
C	6	2.70	10	12
D	6	2.70	110	18
E	5	2.96	111	15
TOTAL (# of bits) :				89

4. Transmit Codes instead of Tokens

- Raw token stream 8 bits per (39 chars) token = 312 bits
- Coded data stream = 89 bits



Back

Close

Shannon-Fano Algorithm: Entropy

In the above example:

Multimedia
CM0340

338

$$\begin{aligned}\text{Ideal_entropy} &= (15 * 1.38 + 7 * 2.48 + 6 * 2.7 \\&\quad + 6 * 2.7 + 5 * 2.96) / 39 \\&= 85.26 / 39 \\&= 2.19\end{aligned}$$

Number of bits needed for Shannon-Fano Coding is: $89 / 39 = 2.28$



Back

Close

Huffman Coding

- Based on the frequency of occurrence of a data item (pixels or small blocks of pixels in images).
- Use a lower number of bits to encode more frequent data
- Codes are stored in a **Code Book** — as for Shannon (previous slides)
- Code book constructed for each image or a set of images.
- Code book **plus** encoded data **must** be transmitted to enable decoding.



Back

Close

Encoding for Huffman Algorithm:

- A bottom-up approach
1. Initialization: Put all nodes in an OPEN list, keep it sorted at all times (e.g., ABCDE).
 2. Repeat until the OPEN list has only one node left:
 - (a) From OPEN pick two nodes having the lowest frequencies/probabilities, create a parent node of them.
 - (b) Assign the sum of the children's frequencies/probabilities to the parent node and insert it into OPEN.
 - (c) Assign code 0, 1 to the two branches of the tree, and delete the children from OPEN.
 3. Coding of each node is a top-down label of branch labels.

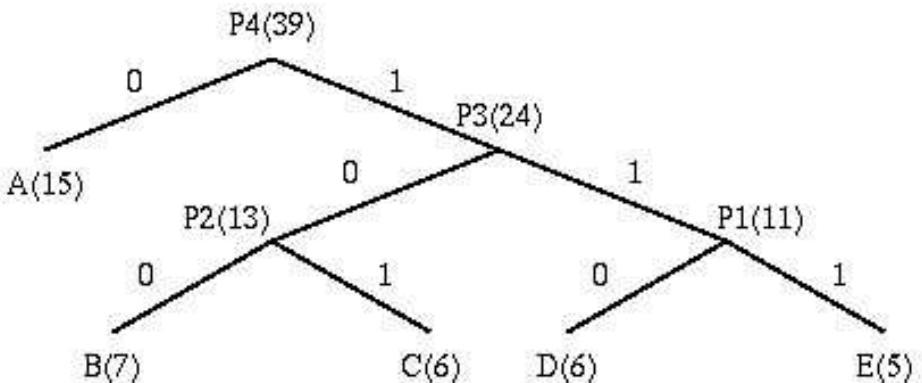


Back

Close

Huffman Encoding Example:

ABBAAAACDEAAABBBDDEEEA..... (Same as Shannon-Fano E.g.)



Symbol	Count	$\log(1/p)$	Code	Subtotal (# of bits)
A	15	1.38	0	15
B	7	2.48	100	21
C	6	2.70	101	18
D	6	2.70	110	18
E	5	2.96	111	15
TOTAL (# of bits):				87

Huffman Encoder Analysis

The following points are worth noting about the above algorithm:

- Decoding for the above two algorithms is trivial as long as the coding table/book is sent before the data.
 - There is a bit of an overhead for sending this.
 - But negligible if the data file is big.
- **Unique Prefix Property**: no code is a prefix to any other code (all symbols are at the leaf nodes) → great for decoder, unambiguous.
- If prior statistics are available and accurate, then Huffman coding is very good.



Back

Close

Huffman Entropy

In the above example:

Multimedia
CM0340

343

$$\begin{aligned}\text{Ideal_entropy} &= (15 * 1.38 + 7 * 2.48 + 6 * 2.7 \\&\quad + 6 * 2.7 + 5 * 2.96) / 39 \\&= 85.26 / 39 \\&= 2.19\end{aligned}$$

Number of bits needed for Huffman Coding is: $87 / 39 = 2.23$



Back

Close

Huffman Coding of Images

In order to encode images:

- Divide image up into (typically) 8x8 blocks
- Each block is a symbol to be coded
- Compute Huffman codes for set of block
- Encode blocks accordingly
- In **JPEG**: Blocks are DCT coded first before Huffman may be applied ([More soon](#))

Coding image in blocks is common to all image coding methods

MATLAB Huffman coding example:

[huffman.m](#) (Used with JPEG code later),

[huffman.zip](#) (Alternative with tree plotting)



Back

Close

Arithmetic Coding

- A widely used entropy coder
- Also used in **JPEG — more soon**
- Only problem is it's speed due possibly complex computations due to large symbol tables,
- Good compression ratio (better than Huffman coding), entropy around the Shannon Ideal value.

Why better than Huffman?

- **Huffman coding etc.** use an integer number (k) of bits for each symbol,
 - hence k is never less than 1.
- Sometimes, e.g., when sending a 1-bit image, compression **becomes impossible**.



Back

Close

Decimal Static Arithmetic Coding

- Here we describe basic approach of Arithmetic Coding
- Initially basic static coding mode of operation.
- Initial example **decimal coding**
- Extend to Binary and then machine word length later

Basic Idea

The idea behind arithmetic coding is

- To have a probability line, 0–1, and
- Assign to every symbol a range in this line based on its probability,
- The higher the probability, the higher range which assigns to it.

Once we have defined the ranges and the probability line,

- Start to encode symbols,
- Every symbol defines where the output floating point number lands within the range.



Back

Close

Simple Basic Arithmetic Coding Example

Assume we have the following token symbol stream

BACA

Therefore

- A occurs with **probability 0.5**,
- B and C with **probabilities 0.25**.



Back

Close

Basic Arithmetic Coding Algorithm

Start by assigning each symbol to the probability range 0–1.

- Sort symbols highest probability first

Symbol	Range
A	[0.0, 0.5)
B	[0.5, 0.75)
C	[0.75, 1.0)

- The first symbol in our example stream is B

We now know that the code will be in the range 0.5 to 0.74999 . . .

Range is not yet unique

- Need to narrow down the range to give us a unique code.

Basic arithmetic coding iteration

- Subdivide the range for the first token given the probabilities of the second token then the third etc.

Subdivide the range as follows

For all the symbols:

```
range = high - low;  
high = low + range * high_range of the symbol being coded;  
low = low + range * low_range of the symbol being coded;
```

Where:

- `range`, keeps track of where the next range should be.
- `high` and `low`, specify the output number.
- Initially `high = 1.0`, `low = 0.0`



Back

Close

Back to our example

The second symbols we have

(now `range` = 0.25, `low` = 0.5, `high` = 0.75):

Symbol	Range
BA	[0.5, 0.625)
BB	[0.625, 0.6875)
BC	[0.6875, 0.75)

Third Iteration

We now reapply the subdivision of our scale again to get for our third symbol

(range = 0.125, low = 0.5, high = 0.625):

Symbol	Range
BAA	[0.5, 0.5625)
BAB	[0.5625, 0.59375)
BAC	[0.59375, 0.625)

Fourth Iteration

Subdivide again

(`range` = 0.03125, `low` = 0.59375, `high` = 0.625):

Symbol	Range
BACA	[0.59375, 0.60937)
BACB	[0.609375, 0.6171875)
BACC	[0.6171875, 0.625)

So the (Unique) output code for BACA is any number in the range:

[0.59375, 0.60937).

Decoding

To **decode** is essentially the opposite

- We compile the table for the sequence given probabilities.
- Find the range of number within which the code number lies and carry on

Binary static algorithmic coding

This is very similar to above:

- **Except** we us **binary fractions**.

Binary fractions are simply an extension of the binary systems into fractions much like decimal fractions.



Back

Close

Binary Fractions — Quick Guide

Fractions in **decimal**:

Multimedia
CM0340

357

$$0.1 \text{ decimal} = \frac{1}{10^1} = 1/10$$

$$0.01 \text{ decimal} = \frac{1}{10^2} = 1/100$$

$$0.11 \text{ decimal} = \frac{1}{10^1} + \frac{1}{10^2} = 11/100$$

So in **binary** we get

$$0.1 \text{ binary} = \frac{1}{2^1} = 1/2 \text{ decimal}$$

$$0.01 \text{ binary} = \frac{1}{2^2} = 1/4 \text{ decimal}$$

$$0.11 \text{ binary} = \frac{1}{2^1} + \frac{1}{2^2} = 3/4 \text{ decimal}$$



Back

Close

Binary Arithmetic Coding Example

- Idea: Suppose alphabet was X, Y and token stream:

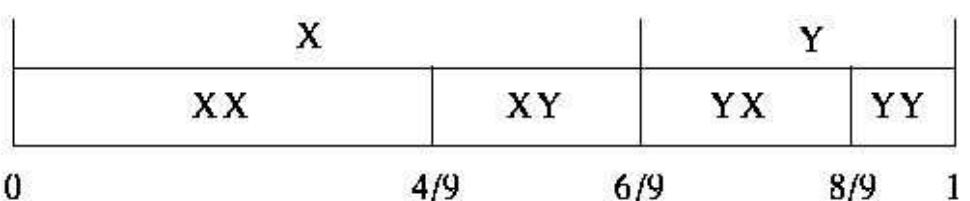
XXY

Therefore:

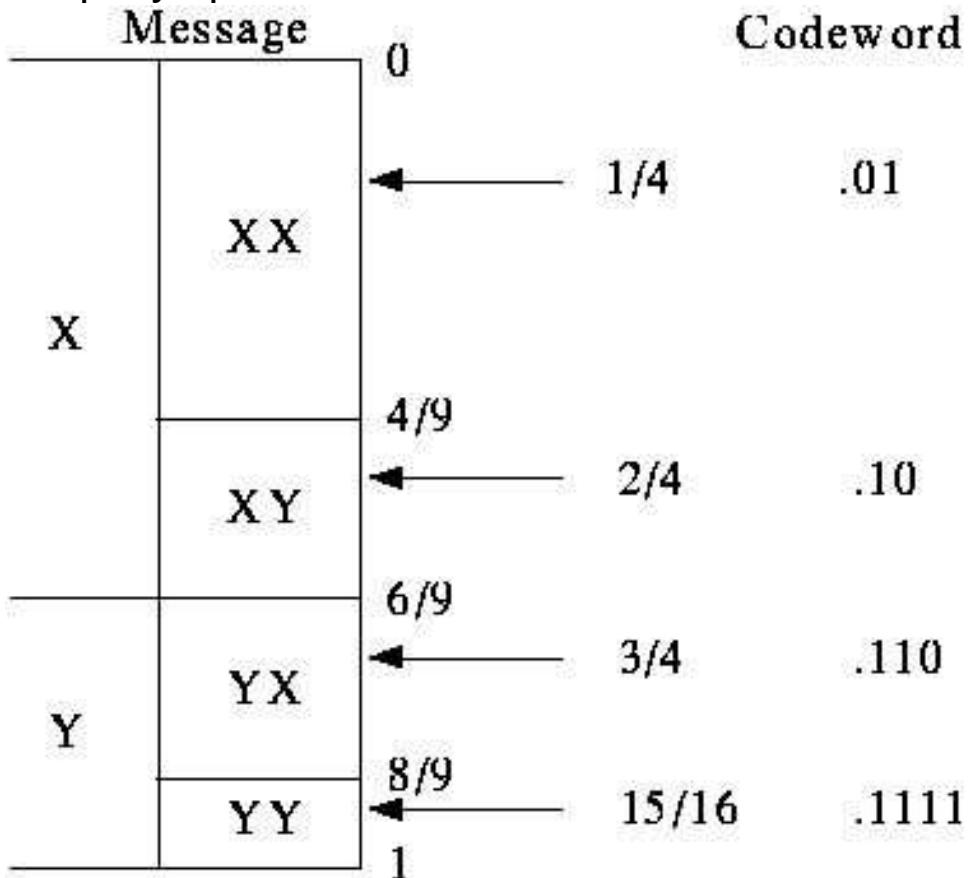
$$\text{prob}(X) = 2/3$$

$$\text{prob}(Y) = 1/3$$

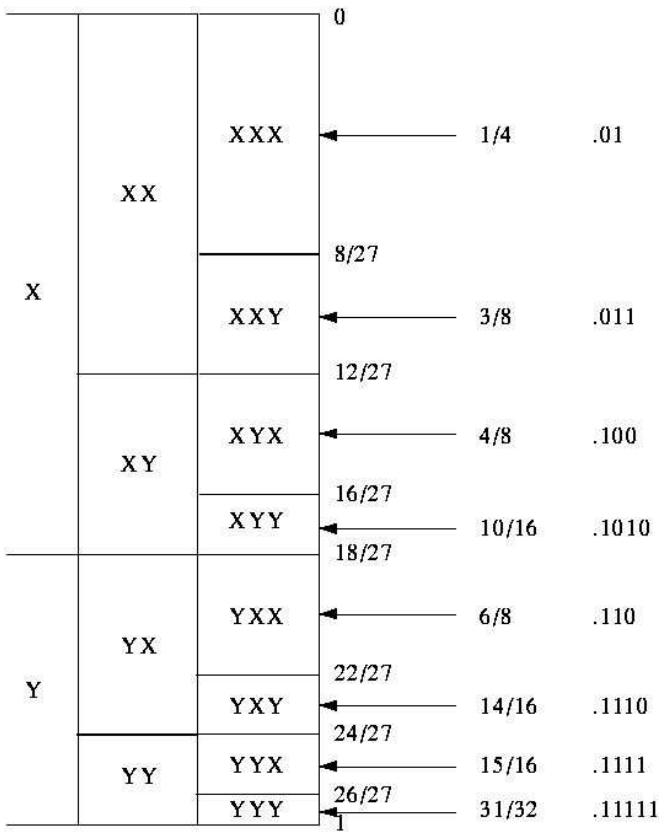
- If we are only concerned with encoding length 2 messages, then we can map all possible messages to intervals in the range [0..1]:



- To encode message, just send enough bits of a binary fraction that uniquely specifies the interval.



- Similarly, we can map all possible length 3 messages to intervals in the range [0..1]:



Implementation Issues

FPU Precision

- Resolution of the number we represent is limited by FPU precision
- Binary coding extreme example of rounding
- Decimal coding is the other extreme — theoretically no rounding.
- Some FPUs may use up to 80 bits
- As an example let us consider working with 16 bit resolution.



Back

Close

16-bit arithmetic coding

We now encode the range 0–1 into 65535 segments:

0.000	0.250	0.500	0.750	1.000
0000h	4000h	8000h	C000h	FFFFh

If we take a number and divide it by the maximum (FFFFh) we will clearly see this:

$$0000h: 0 / 65535 = 0.0$$

$$4000h: 16384 / 65535 = 0.25$$

$$8000h: 32768 / 65535 = 0.5$$

$$C000h: 49152 / 65535 = 0.75$$

$$FFFFh: 65535 / 65535 = 1.0$$



The operation of coding is similar to what we have seen with the binary coding:

- Adjust the probabilities so the bits needed for operating with the number aren't above 16 bits.
- Define a new interval
- The way to deal with the infinite number is
 - to have only loaded the 16 first bits, and when needed shift more onto it:
1100 0110 0001 000 0011 0100 0100 ...
 - work only with those bytes
 - as new bits are needed they'll be shifted.



Back

Close

Memory Intensive

What about an alphabet with 26 symbols, or 256 symbols, ...?

- In general, number of bits is determined by the size of the interval.
- In general, (from entropy) need $-\log p$ bits to represent interval of size p .
- Can be memory and CPU intensive

MATLAB Arithmetic coding examples:

[Arith06.m](#) (Version 1),

[Arith07.m](#) (Version 2)

Lempel-Ziv-Welch (LZW) Algorithm

- A very common compression technique.
- Used in GIF files (LZW), Adobe PDF file (LZW),
UNIX compress (LZ Only)
- Patented — LZW not LZ.

Basic idea/Example by Analogy:

Suppose we want to encode the Oxford Concise English dictionary which contains about 159,000 entries.

Why not just transmit each word as an 18 bit number?



Back

Close

LZW Constructs Its Own Dictionary

Problems:

- Too many bits per word,
- Everyone needs a dictionary,
- Only works for English text.

Solution:

- Find a way to build the dictionary adaptively.
- Original methods (LZ) due to Lempel and Ziv in 1977/8.
- Quite a few variations on LZ.
- Terry Welch improvement (1984), **Patented LZW Algorithm**
 - LZW introduced the idea that only the **initial dictionary** needs to be transmitted to enable **decoding**:
The decoder is able to **build** the **rest** of the table from the **encoded sequence**.



Back

Close

LZW Compression Algorithm

The LZW Compression Algorithm can summarised as follows:

```
w = NIL;  
while ( read a character k )  
{   if wk exists in the dictionary  
    w = wk;  
    else  
    { add wk to the dictionary;  
      output the code for w;  
      w = k;  
    }  
}  
}
```

- Original LZW used dictionary with 4K entries, first 256 (0-255) are ASCII codes.



Back

Close

LZW Compression Algorithm Example:

Input string is " ^WED^WE^WEE^WEB^WET".

w	k	output	index	symbol
NIL	^			
^	W	^	256	^W
W	E	W	257	WE
E	D	E	258	ED
D	^	D	259	D^
^	W			
^W	E	256	260	^WE
E	^	E	261	E^
^	W			
^W	E			
^WE	E	260	262	^WEE
E	^			
E^	W	261	263	E^W
W	E			
WE	B	257	264	WEB
B	^	B	265	B^
^	W			
^W	E			
^WE	T	260	266	^WET
T	EOF	T		

- A 19-symbol input has been reduced to 7-symbol plus 5-code output. Each code/symbol will need more than 8 bits, say 9 bits.

- Usually, compression doesn't start until a large number of bytes (e.g., > 100) are read in.

LZW Decompression Algorithm

The LZW Decompression Algorithm is as follows:

```
read a character k;  
output k;  
w = k;  
while ( read a character k )  
/* k could be a character or a code. */  
{  
    entry = dictionary entry for k;  
    output entry;  
    add w + entry[0] to dictionary;  
    w = entry;  
}
```

Note (Recall):

LZW decoder only needs the **initial dictionary**:

The decoder is able to **build** the **rest** of the table from the **encoded sequence**.



Back

Close

LZW Decompression Algorithm Example:

Input string is

" ^WED<256>E<260><261><257>B<260>T "

w	k	output	index	symbol
^	^			
^	W	W	256	^W
W	E	E	257	WE
E	D	D	258	ED
D	<256>	^W	259	D^
<256>	E	E	260	^WE
E	<260>	^WE	261	E^
<260>	<261>	E^	262	^WEE
<261>	<257>	WE	263	E^W
<257>	B	B	264	WEB
B	<260>	^WE	265	B^
<260>	T	T	266	^WET

MATLAB LZW Code

[norm2lzw.m](#): LZW Encoder

[lzw2norm.m](#): LZW Decoder

[lzw_demo1.m](#): Full MATLAB demo

[More Info on MATLAB LZW code](#)



Back

Close

Lossy Compression: Source Coding Techniques

Source coding is based on changing the content of the original signal.

Also called *semantic-based coding*

Compression rates may be high but at a price of loss of information. Good compression rates may be achieved with source encoding with (occasionally) **lossless** or (mostly) little **perceivable** loss of information.

There are three broad methods that exist:

- Transform Coding
- Differential Encoding
- Vector Quantisation

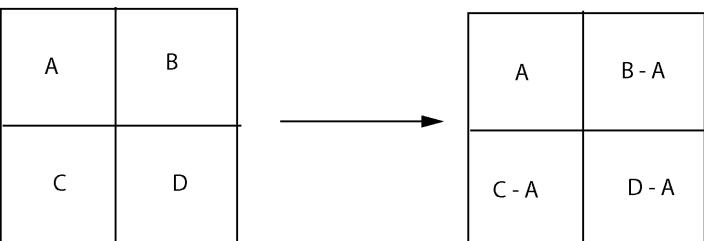


Transform Coding

A simple transform coding example

A Simple Transform Encoding procedure maybe described by the following steps for a 2x2 block of monochrome pixels:

1. Take top left pixel as the base value for the block, pixel A.
2. Calculate three other transformed values by taking the difference between these (respective) pixels and pixel A, i.e. **B-A, C-A, D-A**.
3. Store the base pixel and the differences as the values of the transform.



$$X_0 = A$$

$$X_1 = B - A$$

$$X_2 = C - A$$

$$X_3 = D - A$$

and the inverse transform is:

$$A_n = X_0$$

$$B_n = X_1 + X_0$$

$$C_n = X_2 + X_0$$

$$D_n = X_3 + X_0$$



Back

Close

Compressing data with this Transform?

Exploit redundancy in the data:

- Redundancy transformed to values, X_i .
- Compress the data by using fewer bits to represent the differences — *Quantisation*.
 - I.e if we use 8 bits per pixel then the 2x2 block uses 32 bits
 - If we keep 8 bits for the base pixel, X_0 ,
 - Assign 4 bits for each difference then we only use 20 bits.
 - Better with an average 5 bits/pixel



Back

Close

Example

Consider the following 4x4 image block:

120	130
125	120

then we get:

$$X_0 = 120$$

$$X_1 = 10$$

$$X_2 = 5$$

$$X_3 = 0$$

We can then compress these values by taking less bits to represent the data.



Back

Close

Inadequacies of Simple Scheme

- It is **Too Simple** — not applicable to slightly more complex cases
- Needs to operate on larger blocks (typically 8x8 min)
- Simple encoding of differences for large values will result in loss of information
 - Poor losses possible here 4 bits per pixel = values 0-15 unsigned,
 - Signed value range: $-8 - 7$ so either quantise in larger step value or massive overflow!!

Practical approaches: use more complicated transforms e.g.
DCT ([see later](#))



Differential Transform Coding Schemes

- **Differencing** is used in some compression algorithms:
 - Later part of JPEG compression
 - Exploit static parts (e.g. background) in MPEG video
 - Some speech coding and other simple signals
 - **Good** on repetitive sequences
- **Poor** on highly varying data sequences
 - e.g interesting audio/video signals

MATLAB Simple Vector Differential Example

[diffencodevec.m](#): Differential Encoder

[diffdecodevec.m](#): Differential Decoder

[diffencodevecTest.m](#): Differential Test Example



Back

Close

Differential Encoding

Simple example of transform coding mentioned earlier and instance of this approach.

Here:

- The difference between the actual value of a sample and a prediction of that values is encoded.
- Also known as **predictive encoding**.
- Example of technique include: differential pulse code modulation, delta modulation and adaptive pulse code modulation — differ in prediction part.
- Suitable where successive signal samples do not differ much, but are not zero. **E.g.** Video — difference between frames, some audio signals.

Differential Encoding Methods

- Differential pulse code modulation (DPCM)

Multimedia
CM0340

380

Simple prediction (also used in JPEG):

$$f_{predict}(t_i) = f_{actual}(t_{i-1})$$

I.e. a simple Markov model where current value is the predict next value.

So we simply need to encode:

$$\Delta f(t_i) = f_{actual}(t_i) - f_{actual}(t_{i-1})$$

If successive sample are close to each other we only need to encode first sample with a large number of bits:



Back

Close

Simple Differential Pulse Code Modulation Example

Actual Data: 9 10 7 6

Predicted Data: 0 9 10 7

$\Delta f(t)$: +9, +1, -3, -1.

MATLAB Complete (with quantisation) DPCM Example

[dpcm_demo.m.m](#): DPCM Complete Example

[dpcm.zip.m](#): DPCM Support Files



Back

Close

Differential Encoding Methods (Cont.)

- **Delta modulation** is a special case of DPCM:
 - Same predictor function,
 - Coding error is a **single bit or digit** that indicates the current sample should be increased or decreased by a step.
 - Not Suitable for rapidly changing signals.
- **Adaptive pulse code modulation**

Fuller Temporal/Markov model:

- Data is extracted from a function of a series of previous values
- **E.g.** Average of last n samples.
- Characteristics of sample better preserved.



Back

Close

Frequency Domain Methods

Another form of Transform Coding

Transformation from one domain —time (e.g. 1D audio, video:2D imagery over time) or Spatial (e.g. 2D imagery) domain to the **frequency** domain via

- **Discrete Cosine Transform (DCT)**— Heart of **JPEG** and **MPEG Video**, (alt.) MPEG Audio.
- **Fourier Transform (FT)** — **MPEG Audio**

Theory already studied earlier



Back

Close

RECAP — Compression In Frequency Space

How do we achieve compression?

- Low pass filter — ignore high frequency noise components
- Only store lower frequency components
- High Pass Filter — Spot Gradual Changes
- If changes to low Eye does not respond so ignore?



Back

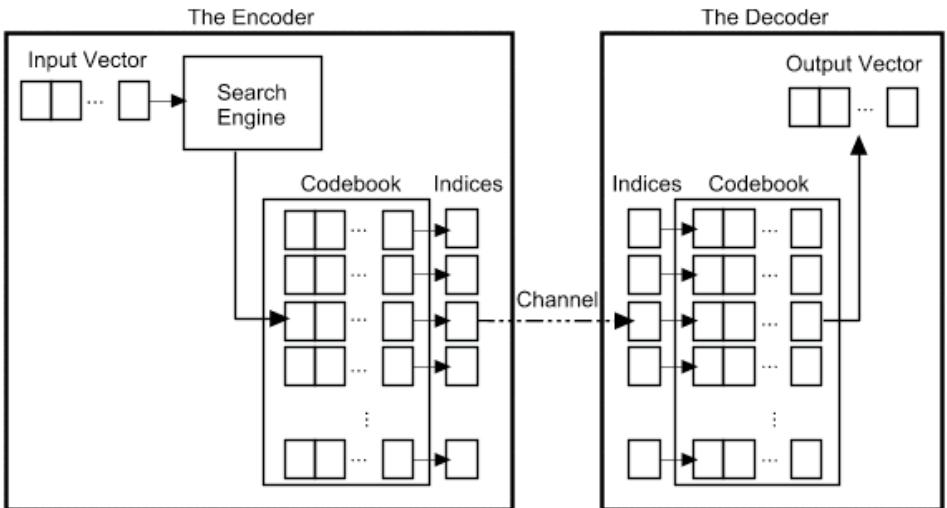
Close

Vector Quantisation

The basic outline of this approach is:

- Data stream divided into (1D or 2D square) blocks — **vectors**
- A table or **code book** is used to find a pattern for each block.
- Code book can be dynamically constructed or predefined.
- Each pattern for block encoded as a look value in table
- Compression achieved as data is effectively subsampled and coded at this level.
- Used in MPEG4, Video Codecs (Cinepak, Sorenson), Speech coding, Ogg Vorbis.

Vector Quantisation Encoding/Decoding



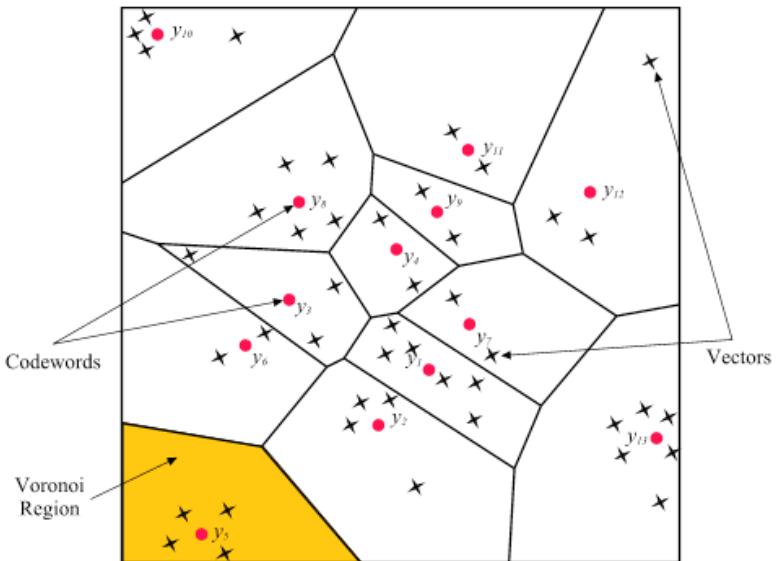
- **Search Engine:**

- Group (Cluster) data into vectors
 - Find closest code vectors
- On decode output need to *unblock* (smooth) data

Vector Quantisation Code Book Construction

How to cluster data?

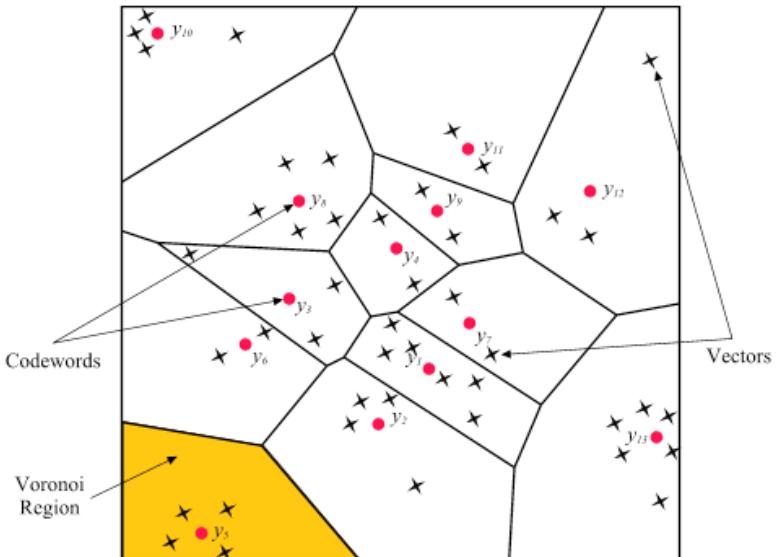
- Use some clustering technique,
e.g. K-means, Voronoi decomposition
Essentially cluster on some closeness measure, minimise
inter-sample variance or distance.



Vector Quantisation Code Book Construction

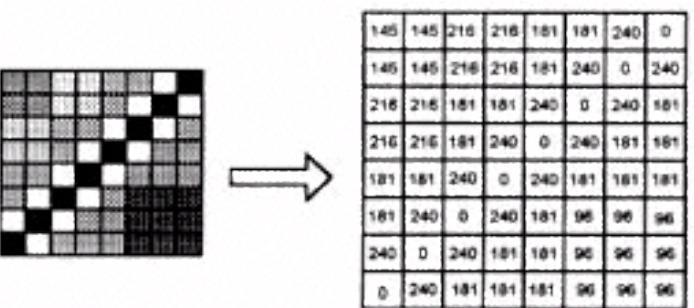
How to code?

- For each cluster choose a mean (median) point as representative code for all points in cluster.

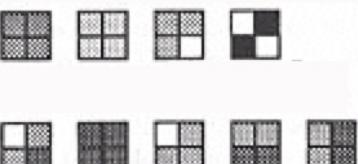


Vector Quantisation Image Coding Example

- A small block of images and intensity values



- Consider Vectors of 2x2 blocks, and only allow 8 codes in table.
- 9 vector blocks present in above:

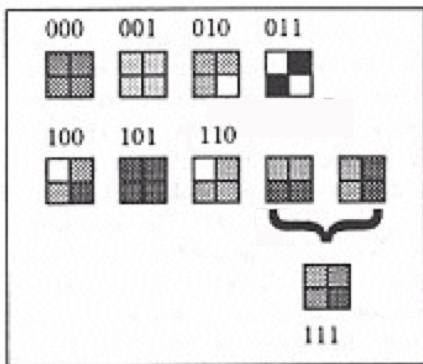


Vector Quantisation Image Coding Example (Cont.)

- 9 vector blocks, so **only one** has to be **vector quantised** here.
- Resulting code book for above image

Multimedia
CM0340

390



MATLAB EXAMPLE: vectorquantise.m



Back

Close

Moving into the Frequency Domain

Frequency domains can be obtained through the transformation from one (**Time** or **Spatial**) domain to the other (**Frequency**) via

- Discrete Cosine Transform (DCT)— Heart of **JPEG** and **MPEG Video**, (alt.) MPEG Audio. (**New**)
- Fourier Transform (FT) — **MPEG Audio** (**See Tutorial 2** —**Recall From CM0268 and**)

Note: We mention some image (and video) examples in this section with DCT (in particular) but also the FT is commonly applied to filter multimedia data.



Back

Close

Recap: What do frequencies mean in an image?

- Large values at **high** frequency components then the data is changing rapidly on a short distance scale.
e.g. a page of text
- Large **low** frequency components then the large scale features of the picture are more important.
e.g. a single fairly simple object which occupies most of the image.



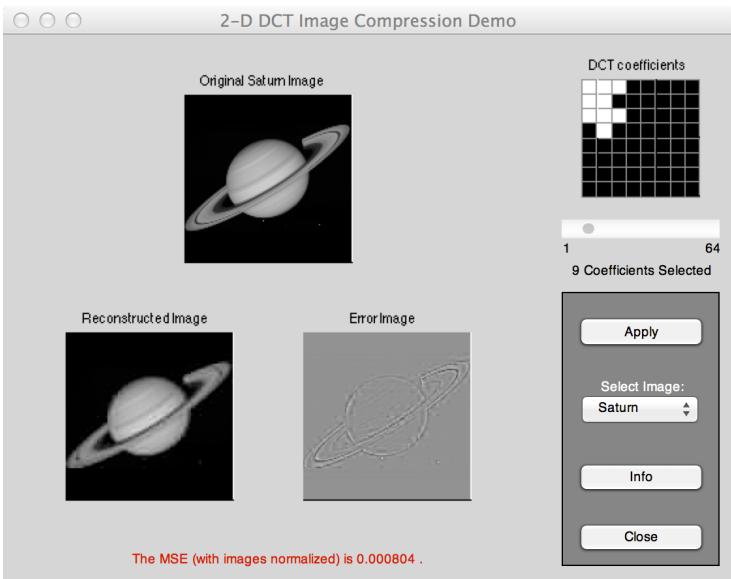
Back

Close

The Road to Compression

How do we achieve compression?

- Low pass filter — ignore high frequency noise components
 - Only store lower frequency components
- High Pass Filter — Spot Gradual Changes
 - If changes to low Eye does not respond so ignore?



Low Pass Image Compression Example: [dctdemo.m](#)

MATLAB demo, [dctdemo.m](#), (uses DCT (**see very soon**) to

- Load an image
- **Low Pass Filter** in frequency (DCT) space
- **Tune** compression via a single slider value to select n coefficients
- **Inverse DCT**, subtract input and filtered image to see **compression artefacts**.



Back

Close

Recap: Fourier Transform

The tool which converts a spatial (real space) description of audio/image data into one in terms of its frequency components is called the **Fourier transform**

The new version is usually referred to as the **Fourier space description** of the data.

We then essentially process the data:

- *E.g.* for **filtering** basically this means attenuating or setting certain frequencies to zero

We then need to convert data back to real audio/imagery to use in our applications.

The corresponding **inverse** transformation which turns a Fourier space description back into a real space one is called the **inverse Fourier transform**.



The Discrete Cosine Transform (DCT)

Relationship between DCT and FFT

DCT (Discrete Cosine Transform) is actually a *cut-down* version of the Fourier Transform or the Fast Fourier Transform (FFT):

- Only the **real** part of FFT
- Computationally simpler than FFT
- DCT — Effective for Multimedia Compression
- DCT **MUCH** more commonly used (than FFT) in Multimedia Image/Video Compression — **more later**
- Cheap MPEG Audio Variant — **more later**

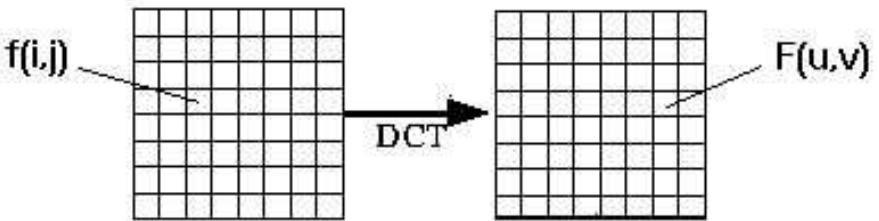


Back

Close

Applying The DCT

- Similar to the discrete Fourier transform:
 - it transforms a signal or image from the spatial domain to the frequency domain
 - DCT can approximate lines well with fewer coefficients



- Helps separate the image into parts (or spectral sub-bands) of differing importance (with respect to the image's visual quality).

1D DCT

For N data items 1D DCT is defined by:

$$F(u) = \left(\frac{2}{N}\right)^{\frac{1}{2}} \sum_{i=0}^{N-1} \Lambda(u). \cos\left[\frac{\pi \cdot u}{2 \cdot N}(2i + 1)\right] f(i)$$

and the corresponding **inverse** 1D DCT transform is simple $F^{-1}(u)$, i.e.:

$$\begin{aligned} f(i) &= F^{-1}(u) \\ &= \left(\frac{2}{N}\right)^{\frac{1}{2}} \sum_{u=0}^{N-1} \Lambda(u). \cos\left[\frac{\pi \cdot u}{2 \cdot N}(2i + 1)\right] F(u) \end{aligned}$$

where

$$\Lambda(\xi) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for } \xi = 0 \\ 1 & \text{otherwise} \end{cases}$$

2D DCT

For a 2D N by M image 2D DCT is defined :

$$\begin{aligned} F(u, v) &= \left(\frac{2}{N}\right)^{\frac{1}{2}} \left(\frac{2}{M}\right)^{\frac{1}{2}} \sum_{i=0}^{N-1} \sum_{j=0}^{M-1} \Lambda(\textcolor{red}{u}) \cdot \Lambda(\textcolor{red}{v}) \cdot \\ &\quad \cos\left[\frac{\pi \cdot u}{2 \cdot N}(2i+1)\right] \cos\left[\frac{\pi \cdot v}{2 \cdot M}(2j+1)\right] \cdot f(i, j) \end{aligned}$$

and the corresponding **inverse** 2D DCT transform is simple $F^{-1}(u, v)$, i.e.:

$$\begin{aligned} f(i, j) &= F^{-1}(u, v) \\ &= \left(\frac{2}{N}\right)^{\frac{1}{2}} \sum_{u=0}^{N-1} \sum_{v=0}^{M-1} \Lambda(u) \cdot \Lambda(\textcolor{red}{v}) \cdot \\ &\quad \cos\left[\frac{\pi \cdot u}{2 \cdot N}(2i+1)\right] \cdot \cos\left[\frac{\pi \cdot v}{2 \cdot M}(2j+1)\right] \cdot F(u, v) \end{aligned}$$

where

$$\Lambda(\xi) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for } \xi = 0 \\ 1 & \text{otherwise} \end{cases}$$

Performing DCT Computations

The basic operation of the DCT is as follows:

- The input image is N by M;
- $f(i,j)$ is the intensity of the pixel in row i and column j ;
- $F(u,v)$ is the DCT coefficient in row u_i and column v_j of the DCT matrix.
- For JPEG image (and MPEG video), the DCT input is usually an 8 by 8 (or 16 by 16) array of integers.

This array contains each image window's respective colour band pixel levels;

Compression with DCT

- For most images, much of the signal energy lies at low frequencies;
 - These appear in the upper left corner of the DCT.
- Compression is achieved since the lower right values represent higher frequencies, and are often small
 - Small enough to be neglected with little visible distortion.



Back

Close

Computational Issues (1)

- Image is partitioned into 8×8 regions — The DCT input is an 8×8 array of integers. **Why 8×8 ?**
- An 8 point DCT would be:

$$F(u, v) = \frac{1}{4} \sum_{i,j} \Lambda(\textcolor{red}{u}) \cdot \Lambda(\textcolor{red}{v}) \cdot \cos \left[\frac{\pi \cdot u}{16} (2i + 1) \right] \cdot \cos \left[\frac{\pi \cdot v}{16} (2j + 1) \right] f(i, j)$$

where

$$\Lambda(\xi) = \begin{cases} \frac{1}{\sqrt{2}} & \text{for } \xi = 0 \\ 1 & \text{otherwise} \end{cases}$$

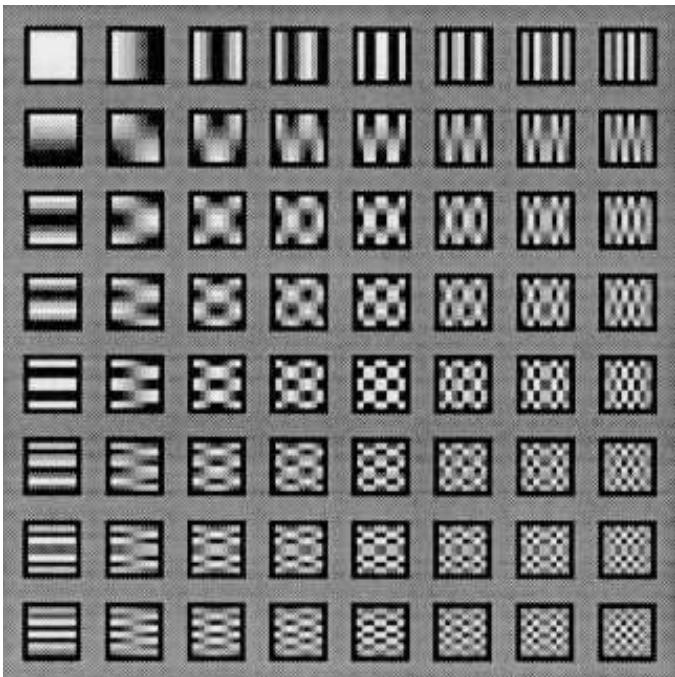
- The output array of DCT coefficients contains integers; these can range from -1024 to 1023.

Computational Issues (2)

- Computationally easier to implement and more efficient to regard the DCT as a set of **basis functions**
 - Given a known input array size (8×8) can be precomputed and stored.
 - Computing values for a convolution mask (8×8 window) that get applied
 - * Sum values x pixel the window overlap with image apply window across all rows/columns of image
 - The values as simply calculated from DCT formula.

Computational Issues (3)

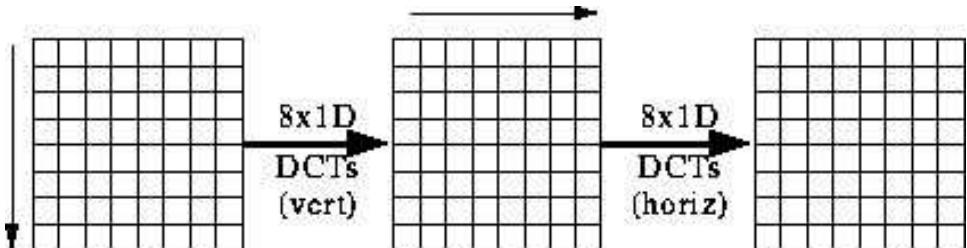
Visualisation of DCT basis functions



See MATLAB demo, [dctbasis.m](#), to see how to produce these bases.

Computational Issues (4)

- Factoring reduces problem to a series of 1D DCTs
(No need to apply 2D form directly):
 - apply 1D DCT (Vertically) to Columns
 - apply 1D DCT (Horizontally) to resultant Vertical DCT above.
 - or alternatively Horizontal to Vertical.



Computational Issues (5)

- The equations are given by:

$$G(i, v) = \frac{1}{2} \sum_i \Lambda(v) \cdot \cos \left[\frac{\pi \cdot v}{16} (2j + 1) \right] f(i, j)$$

$$F(u, v) = \frac{1}{2} \sum_i \Lambda(u) \cdot \cos \left[\frac{\pi \cdot u}{16} (2i + 1) \right] G(i, v)$$

- Most software implementations use fixed point arithmetic. Some fast implementations approximate coefficients so all multiplies are shifts and adds.

Filtering in the Frequency Domain: Some more examples

FT and DCT methods pretty similar:

- DCT has less data overheads — no complex array part
- FT captures more frequency 'fidelity' (e.g. Phase).

Low Pass Filter

*Example: Frequencies above the Nyquist Limit,
Noise:*

- The idea with noise smoothing is to reduce various spurious effects of a local nature in the image, caused perhaps by
 - noise in the acquisition system,

- arising as a result of transmission of the data, for example from a space probe utilising a low-power transmitter.



Back

Close

Recap: Frequency Space Smoothing Methods

Noise = High Frequencies:

- In audio data many spurious peaks in over a short timescale.
- In an image means there are many rapid transitions (over a short distance) in intensity from high to low and back again or vice versa, as faulty pixels are encountered.
- **Not all high frequency data noise though!**

Therefore noise will contribute heavily to the high frequency components of the image when it is considered in Fourier space.

Thus if we reduce the high frequency components — **Low-Pass Filter**, we should reduce the amount of noise in the data.



Back

Close

(Low-pass) Filtering in the Fourier Space

We thus create a new version of the image in Fourier space by computing

$$G(u, v) = H(u, v)F(u, v)$$

where:

- $F(u, v)$ is the Fourier transform of the original image,
- $H(u, v)$ is a filter function, designed to reduce high frequencies, and
- $G(u, v)$ is the **Fourier transform of the improved image**.
- Inverse Fourier transform $G(u, v)$ to get $g(x, y)$ our **improved image**

Note: Discrete Cosine Transform approach identical, sub. FT with DCT

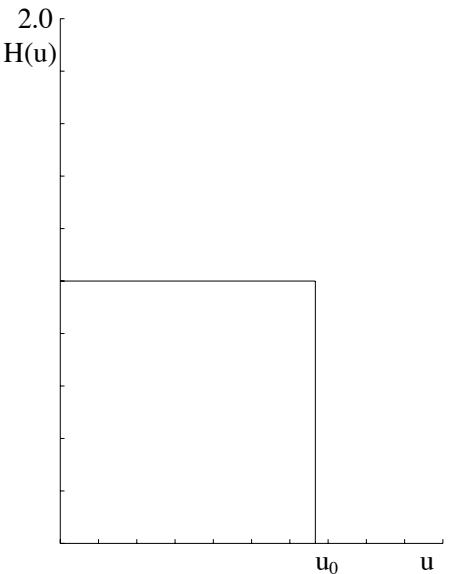


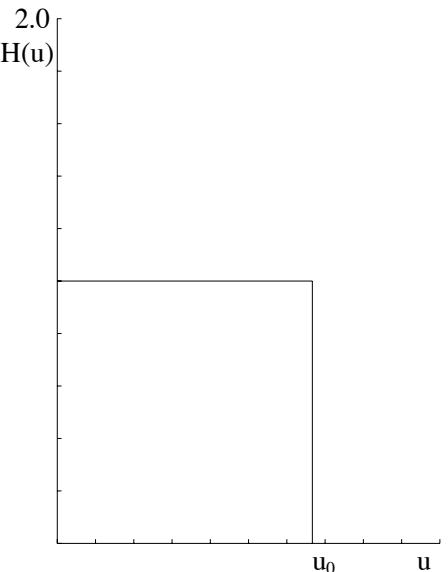
Back

Close

Ideal Low-Pass Filter

The simplest sort of filter to use is an *ideal low-pass filter*, which in one dimension appears as :





Ideal Low-Pass Filter (Cont.)

This is a top hat function which is 1 for u between 0 and u_0 , the *cut-off frequency*, and zero elsewhere.

- So All frequency space information above u_0 is thrown away, and all information below u_0 is kept.
- A **very simple** computational process.

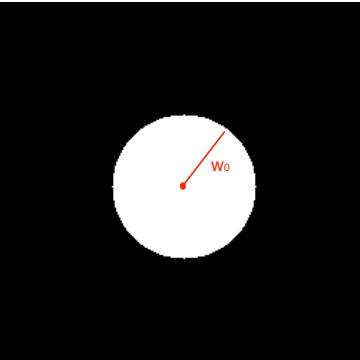
Ideal 2D Low-Pass Filter

The two dimensional analogue of this is the function

$$H(u, v) = \begin{cases} 1 & \text{if } \sqrt{u^2 + v^2} \leq w_0 \\ 0 & \text{otherwise,} \end{cases}$$

where w_0 is now the cut-off frequency.

Thus, all frequencies inside a radius w_0 are kept, and all others discarded.



Not So Ideal Low-Pass Filter?

The problem with this filter is that as well as the noise:

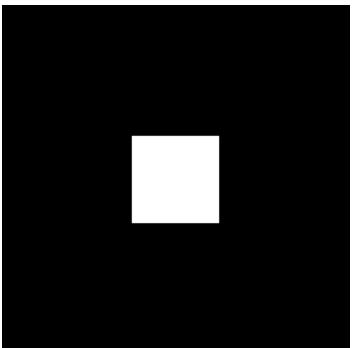
- In audio: plenty of other high frequency content
- In Images: edges (places of rapid transition from light to dark) also significantly contribute to the high frequency components.

Thus an ideal low-pass filter will tend to *blur* the data:

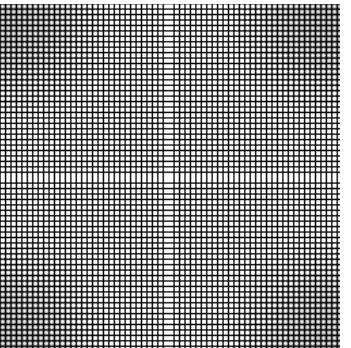
- High audio frequencies become muffled
- Edges in images become blurred.

The lower the cut-off frequency is made, the more pronounced this effect becomes in *useful data content*

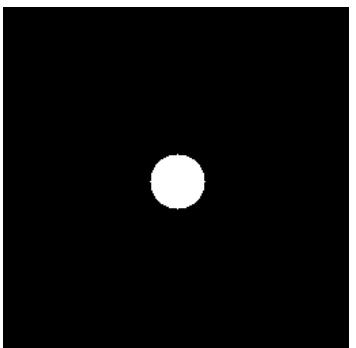
Ideal Low Pass Filter Example 1



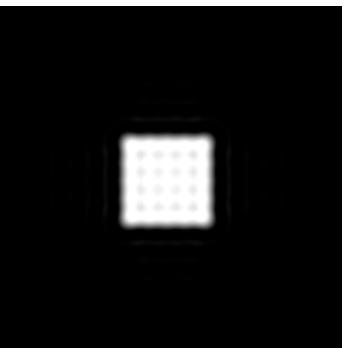
(a) Input Image



(b) Image Spectra



(c) Ideal Low Pass Filter



(d) Filtered Image

Ideal Low-Pass Filter Example 1 MATLAB Code

low pass.m:

```
% Create a white box on a black background image
M = 256; N = 256;
image = zeros(M,N)
box = ones(64,64);
%box at centre
image(97:160,97:160) = box;

% Show Image

figure(1);
imshow(image);

% compute fft and display its spectra

F=fft2(double(image));
figure(2);
imshow(abs(fftshift(F)));
```



Ideal Low-Pass Filter Example 1 MATLAB Code (Cont.)

```
%compute Ideal Low Pass Filter
u0 = 20; % set cut off frequency
```

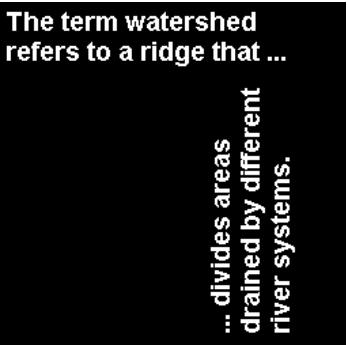
```
u=0:(M-1);
v=0:(N-1);
idx=find(u>M/2);
u(idx)=u(idx)-M;
idy=find(v>N/2);
v(idy)=v(idy)-N;
[V, U]=meshgrid(v, u);
D=sqrt(U.^2+V.^2);
H=double(D<=u0);
```

```
% display
figure(3);
imshow(fftshift(H));
```

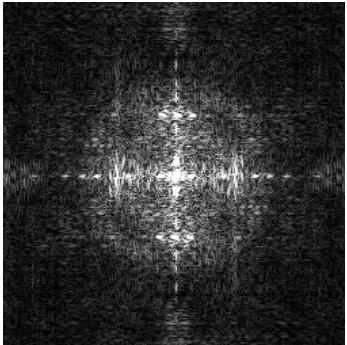
```
% Apply filter and do inverse FFT
G=H.*F;
g=real(ifft2(double(G)));
```

```
% Show Result
figure(4);
imshow(g);
```

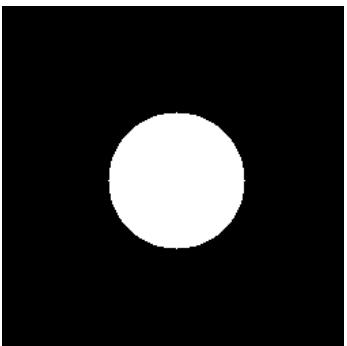
Ideal Low-Pass Filter Example 2



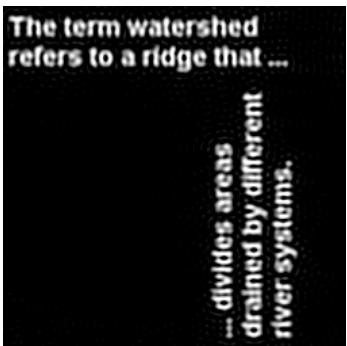
(a) Input Image



(b) Image Spectra



(c) Ideal Low-Pass Filter



(d) Filtered Image

Ideal Low-Pass Filter Example 2 MATLAB Code

lowpass2.m:

```
% read in MATLAB demo text image
image = imread('text.png');
[M N] = size(image)

% Show Image

figure(1);
imshow(image);

% compute fft and display its spectra

F=fft2(double(image));
figure(2);
imshow(abs(fftshift(F))/256);
```



Ideal Low-Pass Filter Example 2 MATLAB Code (Cont.)

```
%compute Ideal Low Pass Filter
u0 = 50; % set cut off frequency

u=0:(M-1);
v=0:(N-1);
idx=find(u>M/2);
u(idx)=u(idx)-M;
idy=find(v>N/2);
v(idy)=v(idy)-N;
[V, U]=meshgrid(v, u);
D=sqrt(U.^2+V.^2);
H=double(D<=u0);

% display
figure(3);
imshow(fftshift(H));

% Apply filter and do inverse FFT
G=H.*F;
g=real(ifft2(double(G)));

% Show Result
figure(4);
imshow(g);
```

Low-Pass Butterworth Filter

Another filter sometimes used is the *Butterworth low pass filter*.

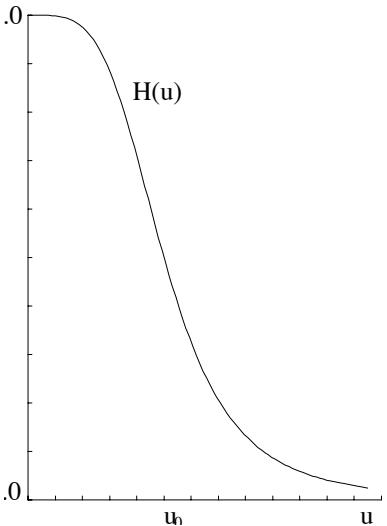
In the 2D case, $H(u, v)$ takes the form

$$H(u, v) = \frac{1}{1 + [(u^2 + v^2)/w_0^2]^n},$$

where n is called the **order** of the filter.

Low-Pass Butterworth Filter (Cont.)

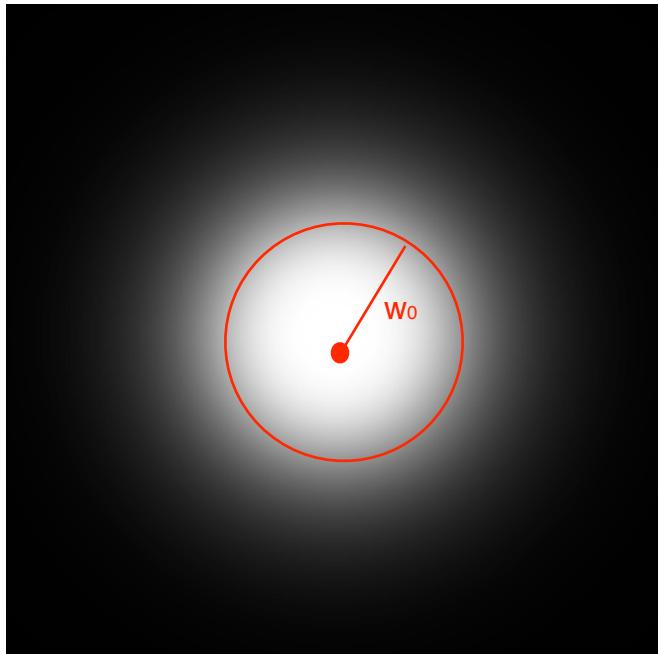
This keeps some of the high frequency information, as illustrated by the second order one dimensional Butterworth filter:



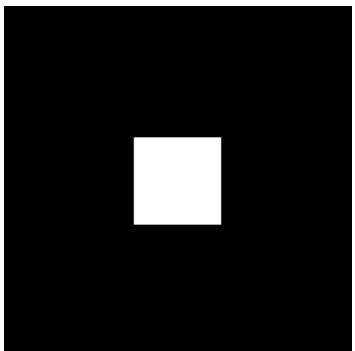
Consequently reduces the blurring.

Low-Pass Butterworth Filter (Cont.)

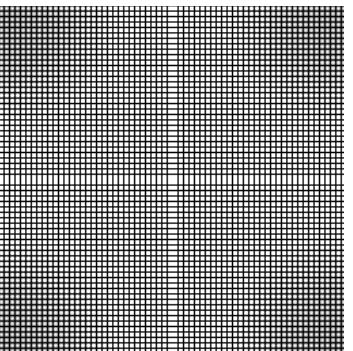
The 2D second order Butterworth filter looks like this:



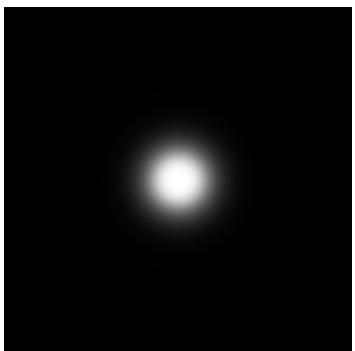
Butterworth Low Pass Filter Example 1



(a) Input Image



(b) Image Spectra



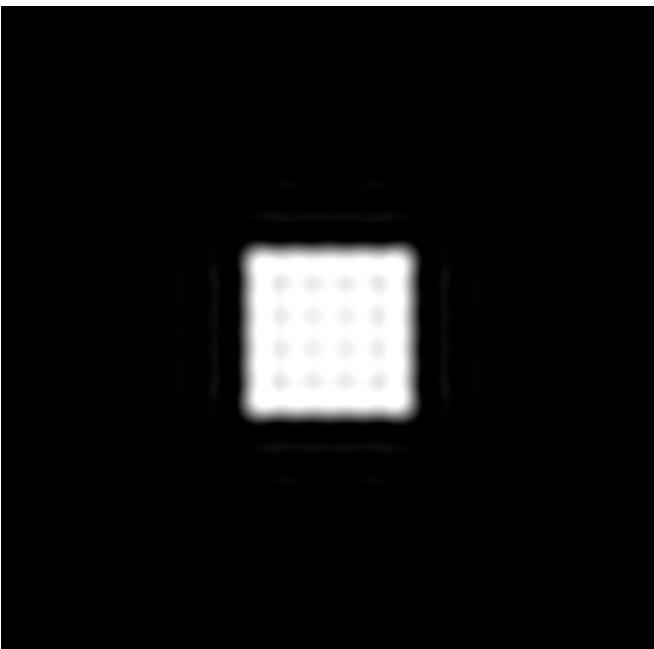
(c) Butterworth Low-Pass Filter



(d) Filtered Image

Butterworth Low-Pass Filter Example 1 (Cont.)

Comparison of Ideal and Butterworth Low Pass Filter:



Ideal Low-Pass



Butterworth Low Pass

Butterworth Low-Pass Filter Example 1 MATLAB Code

butterworth.m:

```
% Load Image and Compute FFT as in Ideal Low Pass Filter
% Example 1
.....
% Compute Butterworth Low Pass Filter
u0 = 20; % set cut off frequency

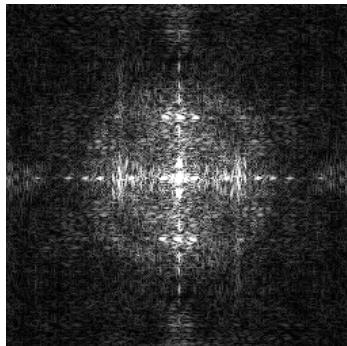
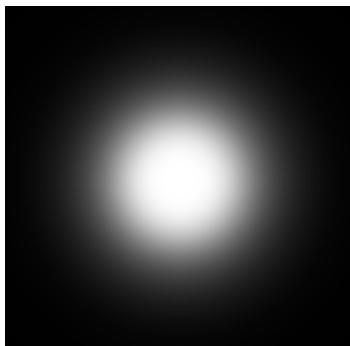
u=0:(M-1);
v=0:(N-1);
idx=find(u>M/2);
u(idx)=u(idx)-M;
idy=find(v>N/2);
v(idy)=v(idy)-N;
[V,U]=meshgrid(v,u);

for i = 1: M
    for j = 1:N
        %Apply a 2nd order Butterworth
        UVw = double((U(i,j)*U(i,j) + V(i,j)*V(i,j)) / (u0*u0));
        H(i,j) = 1/(1 + UVw*UVw);
    end
end
% Display Filter and Filtered Image as before
```

Butterworth Low-Pass Butterworth Filter Example 2

The term watershed
refers to a ridge that ...
... divides areas
drained by different
river systems.

(a) Input Image



(b) Image Spectra

The term watershed
refers to a ridge that ...
... divides areas
drained by different
river systems.

(c) Butterworth Low-Pass Filter

(d) Filtered Image

Butterworth Low-Pass Filter Example 2 (Cont.)

Comparison of Ideal and Butterworth Low-Pass Filter:

The term watershed
refers to a ridge that ...

... divides areas
drained by different
river systems.

Ideal Low Pass

The term watershed
refers to a ridge that ...

... divides areas
drained by different
river systems.

Butterworth Low Pass

Butterworth Low Pass Filter Example 2 MATLAB Code

butterworth2.m:

```
% Load Image and Compute FFT as in Ideal Low Pass Filter
% Example 2
.....
% Compute Butterworth Low Pass Filter
u0 = 50; % set cut off frequency

u=0:(M-1);
v=0:(N-1);
idx=find(u>M/2);
u(idx)=u(idx)-M;
idy=find(v>N/2);
v(idy)=v(idy)-N;
[V,U]=meshgrid(v,u);

for i = 1: M
    for j = 1:N
        %Apply a 2nd order Butterworth
        UVw = double((U(i,j)*U(i,j) + V(i,j)*V(i,j)) / (u0*u0));
        H(i,j) = 1/(1 + UVw*UVw);
    end
end
% Display Filter and Filtered Image as before
```

Other Filters

High-Pass Filters — opposite of low-pass, select high frequencies, attenuate those **below** u_0

Band-pass — allow frequencies in a range $u_0 \dots u_1$ attenuate those outside this range

Band-reject — opposite of band-pass, attenuate frequencies within $u_0 \dots u_1$ **select** those **outside** this range

Notch — attenuate frequencies in a narrow bandwidth around cut-off frequency, u_0

Resonator — amplify frequencies in a narrow bandwidth around cut-off frequency, u_0

Other filters exist that are a combination of the above



Back

Close

Convolution

Several important audio and optical effects can be described in terms of convolutions.

- In fact the above Fourier filtering is applying convolutions of low pass filter where the equations are Fourier Transforms of real space equivalents.
- deblurring — high pass filtering
- reverb — **see CM0268**.

1D Convolution

Let us examine the concepts using 1D continuous functions.

The convolution of two functions $f(x)$ and $g(x)$, written $f(x) * g(x)$, is defined by the integral

$$f(x) * g(x) = \int_{-\infty}^{\infty} f(\alpha)g(x - \alpha) d\alpha.$$

1D Convolution Example

For example, let us take two top hat functions of the type described earlier.

Let $f(\alpha)$ be the top hat function shown:

$$f(\alpha) = \begin{cases} 1 & \text{if } |\alpha| \leq 1 \\ 0 & \text{otherwise,} \end{cases}$$

and let $g(\alpha)$ be as shown in next slide, defined by

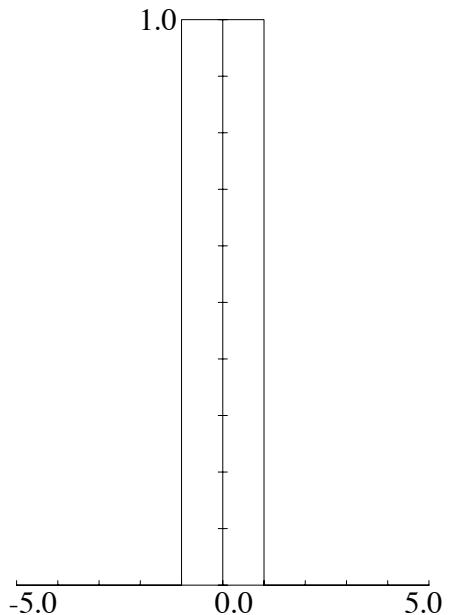
$$g(\alpha) = \begin{cases} 1/2 & \text{if } 0 \leq \alpha \leq 1 \\ 0 & \text{otherwise.} \end{cases}$$



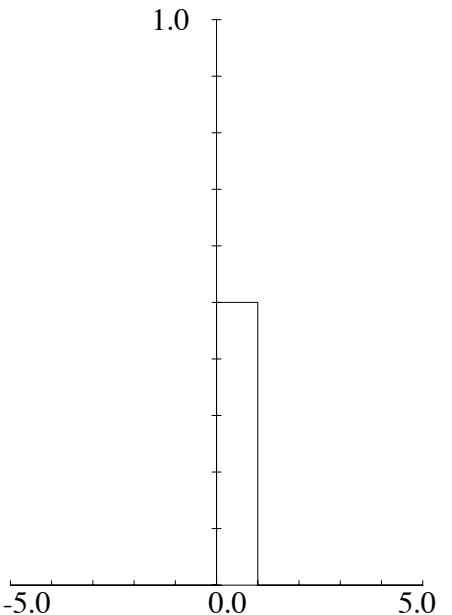
Back

Close

1D Convolution Example (Cont.)



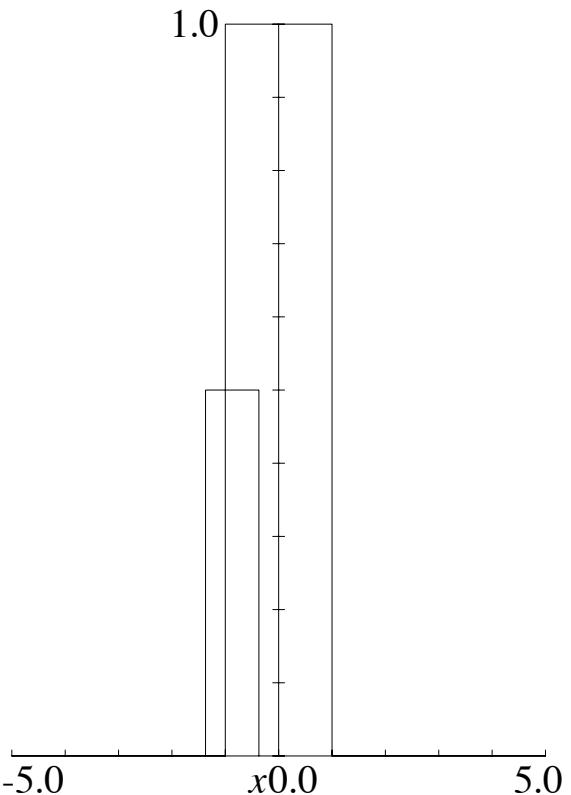
$$f(\alpha) = \begin{cases} 1 & \text{if } |\alpha| \leq 1 \\ 0 & \text{otherwise,} \end{cases}$$



$$g(\alpha) = \begin{cases} 1/2 & \text{if } 0 \leq \alpha \leq 1 \\ 0 & \text{otherwise.} \end{cases}$$

1D Convolution Example (Cont.)

- $g(-\alpha)$ is the reflection of this function in the vertical axis,
- $g(x - \alpha)$ is the latter shifted to the right by a distance x .
- Thus for a given value of x , $f(\alpha)g(x - \alpha)$ integrated over all α is the area of overlap of these two top hats, as $f(\alpha)$ has unit height.
- An example is shown for x in the range $-1 \leq x \leq 0$ opposite

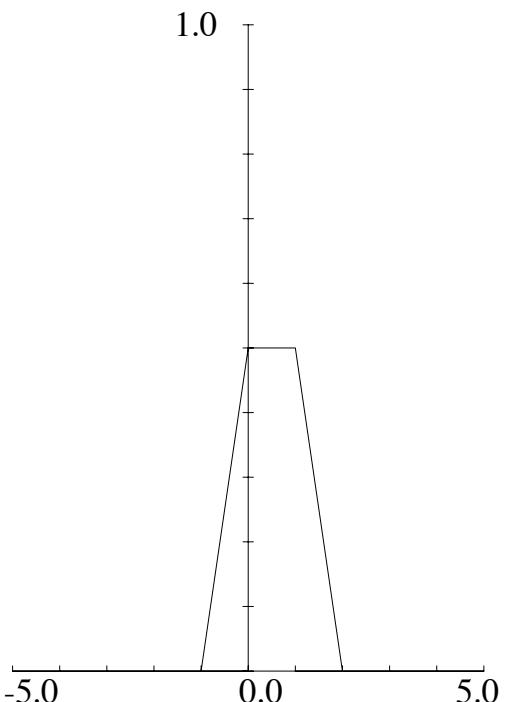


1D Convolution Example (cont.)

If we now consider x moving from $-\infty$ to $+\infty$, we can see that

- For $x \leq -1$ or $x \geq 2$, there is no overlap;
- As x goes from -1 to 0 the area of overlap steadily increases from 0 to $1/2$;
- As x increases from 0 to 1 , the overlap area remains at $1/2$;
- Finally as x increases from 1 to 2 , the overlap area steadily decreases again from $1/2$ to 0 .
- Thus the convolution of $f(x)$ and $g(x)$, $f(x) * g(x)$, in this case has the form shown on next slide

1D Convolution Example (cont.)

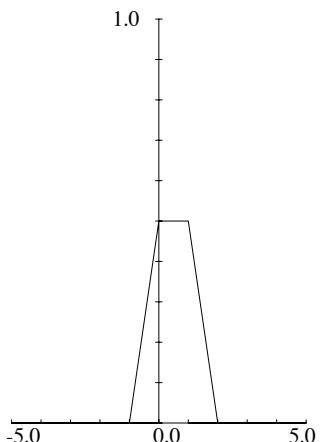


Result of $f(x) * g(x)$

1D Convolution Example (cont.)

Mathematically the convolution is expressed by:

$$f(x) * g(x) = \begin{cases} (x+1)/2 & \text{if } -1 \leq x \leq 0 \\ 1/2 & \text{if } 0 \leq x \leq 1 \\ 1-x/2 & \text{if } 1 \leq x \leq 2 \\ 0 & \text{otherwise.} \end{cases}$$



Fourier Transforms and Convolutions

One major reason that Fourier transforms are so important in image processing is the **convolution theorem** which states that:

*If $f(x)$ and $g(x)$ are two functions with Fourier transforms $F(u)$ and $G(u)$, then the Fourier transform of the convolution $f(x) * g(x)$ is simply the **product** of the **Fourier transforms** of the **two functions**, $F(u)G(u)$.*

Recall our Low Pass Filter Example (MATLAB CODE)

```
% Apply filter  
G=H.*F;
```

Where F was the Fourier transform of the image, H the filter

Computing Convolutions with the Fourier Transform

E.g.:

- To apply some reverb to an audio signal, **example later**
- To compensate for a less than ideal image capture system:

To do this **fast convolution** we simply:

- Take the Fourier transform of the audio/imperfect image,
- Take the Fourier transform of the function describing the effect of the system,
- Multiply by the effect to apply effect to audio data
- To remove/compensate for effect: Divide by the effect to obtain the Fourier transform of the ideal image.
- Inverse Fourier transform to recover the new audio/ideal image.

This process is sometimes referred to as **deconvolution**.

Compression: Images (JPEG)

What is JPEG?

- **JPEG: Joint Photographic Expert Group** — an international standard since 1992.
- Works with colour and greyscale images
- Up to **24 bit colour** images (**Unlike GIF**)
- Target **photographic** quality images (**Unlike GIF**)
- Suitable for many applications e.g., satellite, medical, general photography...



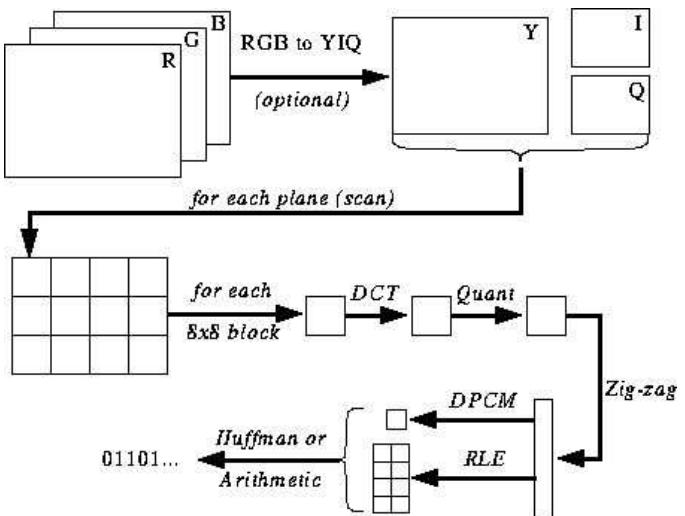
Back

Close

Basic JPEG Compression Pipeline

JPEG compression involves the following:

- Encoding



- Decoding – Reverse the order for encoding

Major Coding Algorithms in JPEG

The Major Steps in JPEG Coding involve:

- Colour Space Transform and subsampling (YIQ)
- DCT (Discrete Cosine Transformation)
- Quantisation
- Zigzag Scan
- DPCM on DC component
- RLE on AC Components
- Entropy Coding — Huffman or Arithmetic

We have met most of the algorithms already:

- JPEG exploits them in the compression pipeline to achieve maximal overall compression.

Quantisation

Why do we need to quantise:

- To throw out bits from DCT.
- *Example:* $(101101)_2 = 45$ (6 bits).

Truncate to 4 bits: $(1011)_2 = 11$.

Truncate to 3 bits: $(101)_2 = 5$.

- Quantisation error is the main source of **Lossy Compression**.
- **DCT itself is not Lossy**
- How we **throw away bits** in **Quantisation Step** is **Lossy**



Quantisation Methods

Uniform quantisation

- Divide by constant N and round result ($N = 4$ or 8 in examples on previous page).
- Non powers-of-two gives fine control (e.g., $N = 6$ loses 2.5 bits)



Back

Close

Quantisation Tables

- In JPEG, each $F[u,v]$ is divided by a constant $q(u,v)$.
- Table of $q(u,v)$ is called *quantisation table*.
- Eye is most sensitive to low frequencies (upper left corner), less sensitive to high frequencies (lower right corner)
- JPEG Standard defines 2 default quantisation tables, one for luminance (below), one for chrominance. *E.g Table below*

16	11	10	16	24	40	51	61
12	12	14	19	26	58	60	55
14	13	16	24	40	57	69	56
14	17	22	29	51	87	80	62
18	22	37	56	68	109	103	77
24	35	55	64	81	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99

Quantization Tables (Cont)

- Q: How would changing the numbers affect the picture

E.g., if we doubled them all?

Quality factor in most implementations is the **scaling factor** for default quantization tables.

- **Custom quantization tables** can be put in image/scan header.

JPEG Quantisation Example

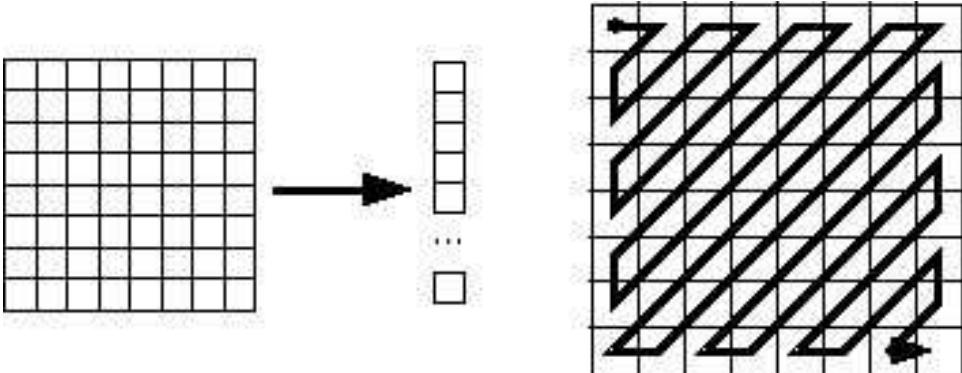
- **JPEG Quantisation Example (Java Applet)**



Zig-zag Scan

What is the purpose of the Zig-zag Scan:

- To group low frequency coefficients in top of vector.
- Maps 8×8 to a 1×64 vector



Differential Pulse Code Modulation (DPCM) on DC Component

- Another encoding method is employed
- DPCM on the DC component.
- Why is this strategy adopted:
 - DC component is large and varies, but often close to previous value (like lossless JPEG).
 - Encode the difference from previous 8x8 blocks – DPCM



Back

Close

Run Length Encode (RLE) on AC Components

Yet another simple compression technique is applied to the AC component:

- 1x63 vector (AC) has lots of zeros in it
- Encode as $(skip, value)$ pairs, where $skip$ is the number of zeros and $value$ is the next non-zero component.
- Send $(0,0)$ as end-of-block sentinel value.



Back

Close

Huffman (Entropy) Coding

DC and AC components finally need to be represented by a smaller number of bits

(Arithmetic coding also supported in place of Huffman coding):

- (Variant of) Huffman coding: Each DPCM-coded DC coefficient is represented by a pair of symbols :

([Size](#), [Amplitude](#))

where [Size](#) indicates number of bits needed to represent coefficient and

[Amplitude](#) contains actual bits.

- [Size only](#) Huffman coded in JPEG:

– [Size](#) does not change too much, generally smaller [Sizes](#) occur frequently (= **low entropy** so is suitable for coding,

– [Amplitude](#) can change widely so coding **no real benefit**



Huffman (Entropy) Coding (Cont)

- Example **Size** category for possible **Amplitudes**:

Size	Typical Huffman Code for Size	Amplitude
0	00	0
1	010	-1, 1
2	011	-3, -2, 2, 3
3	100	-7...-4, 4...7
4	101	-15...-8, 8...15
.	.	.
.	.	.

- Use *ones complement* scheme for negative values: i.e 10 is binary for 2 and 01 for -2 (bitwise inverse). Similarly, 00 for -3 and 11 for 3.



Back

Close

Huffman Coding DC Example

- *Example:* if DC values are 150, -6, 5, 3, -8
- Then 8, 3, 3, 2 and 4 bits are needed respectively.
Send off sizes as Huffman symbol, followed by actual values in bits.

$(8_{huff}, 10010110), (3_{huff}, 001), (3_{huff}, 101), (2_{huff}, 11), (4_{huff}, 0111)$

where $8_{huff} \dots$ are the Huffman codes for respective numbers.

- Huffman Tables can be custom (sent in header) or default.



Back

Close

Huffman Coding on AC Component

AC coefficient are **run-length encoded (RLE)**

- RLE pairs (`Runlength`, `Value`) are Huffman coded as with DC **only** on `Value`.
- So we get a triple: (`Runlength`, `Size`, `Amplitude`)
- However, `Runlength`, `Size` allocated 4-bits each and put into a single byte with is then **Huffman coded**. Again , `Amplitude` is **not** coded.
- So only two symbols transmitted per RLE coefficient:

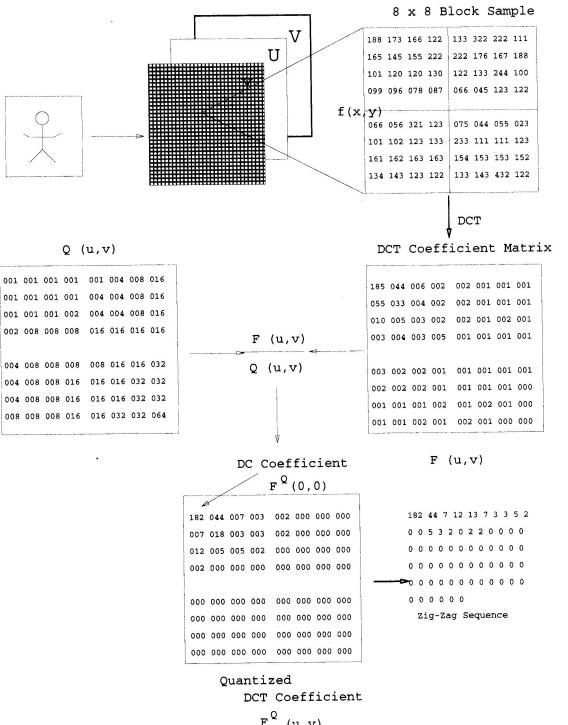
$(RLESIZE_{\text{byte}}, \text{Amplitude})$



Back

Close

Example JPEG Compression



Another Enumerated Example

139	144	149	153	155	155	155	155	235.6	-1.0	-12.1	-5.2	2.1	-1.7	-2.7	1.3	16	11	10	16	24	40	51	61
144	151	153	156	159	156	156	156	-22.6	-17.5	-6.2	-3.2	-2.9	-0.1	0.4	-1.2	12	12	14	19	26	58	60	55
150	155	160	163	158	156	156	156	-10.9	-9.3	-1.6	1.5	0.2	-0.9	-0.6	-0.1	14	13	16	24	40	57	69	56
159	161	162	160	160	159	159	159	-7.1	-1.9	0.2	1.5	0.9	-0.1	0.0	0.3	14	17	22	29	51	87	80	62
159	160	161	162	162	155	155	155	-0.6	-0.8	1.5	1.6	-0.1	-0.7	0.6	1.3	18	22	37	56	68	109	103	77
161	161	161	161	160	157	157	157	1.8	-0.2	1.6	-0.3	-0.8	1.5	1.0	-1.0	24	35	55	64	81	104	113	92
162	162	161	163	162	157	157	157	-1.3	-0.4	-0.3	-1.5	-0.5	1.7	1.1	-0.8	49	64	78	87	103	121	120	101
162	162	161	161	163	158	158	158	-2.6	1.6	-3.8	-1.8	1.9	1.2	-0.6	-0.4	72	92	95	98	112	100	103	99

(a) source image samples

(b) forward DCT coefficients

(c) quantization table

15	0	-1	0	0	0	0	0	240	0	-10	0	0	0	0	0	144	146	149	152	154	156	156	156
-2	-1	0	0	0	0	0	0	-24	-12	0	0	0	0	0	0	148	150	152	154	156	156	156	156
-1	-1	0	0	0	0	0	0	-14	-13	0	0	0	0	0	0	155	156	157	158	158	157	156	155
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	160	161	161	162	161	159	157	155
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	163	163	164	163	162	160	158	156
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	163	164	164	164	162	160	158	157
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	160	161	162	162	161	159	158	158
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	158	159	161	161	162	161	159	158

(d) normalized quantized coefficients

(e) denormalized quantized coefficients

(f) reconstructed image samples



Back

Close

JPEG Example MATLAB Code

The JPEG algorithm may be summarised as follows,

im2jpeg.m (Encoder) jpeg2im.m (Decoder)

mat2huff.m (Huffman coder)

```
m = [16 11 10 16 24 40 51 61           % JPEG normalizing array
      12 12 14 19 26 58 60 55           % and zig-zag reordering
      14 13 16 24 40 57 69 56           % pattern.
      14 17 22 29 51 87 80 62
      18 22 37 56 68 109 103 77
      24 35 55 64 81 104 113 92
      49 64 78 87 103 121 120 101
      72 92 95 98 112 100 103 99] * quality;

order = [1 9 2 3 10 17 25 18 11 4 5 12 19 26 33 ...
          41 34 27 20 13 6 7 14 21 28 35 42 49 57 50 ...
          43 36 29 22 15 8 16 23 30 37 44 51 58 59 52 ...
          45 38 31 24 32 39 46 53 60 61 54 47 40 48 55 ...
          62 63 56 64];                         ...

[xm, xn] = size(x);                      % Get input size.
x = double(x) - 128;                     % Level shift input
t = dctmtx(8);                           % Compute 8 x 8 DCT matrix

% Compute DCTs of 8x8 blocks and quantize the coefficients.
y = blkproc(x, [8 8], 'P1 * x * P2', t, t');
y = blkproc(y, [8 8], 'round(x ./ P1)', m);
```



Back

Close

```
y = im2col(y, [8 8], 'distinct'); % Break 8x8 blocks into columns
xb = size(y, 2); % Get number of blocks
y = y(order, :); % Reorder column elements

eob = max(y(:)) + 1; % Create end-of-block symbol
r = zeros(numel(y) + size(y, 2), 1);
count = 0;
for j = 1:xb % Process 1 block (col) at a time
    i = max(find(y(:, j))); % Find last non-zero element
    if isempty(i) % No nonzero block values
        i = 0;
    end
    p = count + 1;
    q = p + i;
    r(p:q) = [y(1:i, j); eob]; % Truncate trailing 0's, add EOB,
    count = count + i + 1; % and add to output vector
end

r((count + 1):end) = []; % Delete unusued portion of r

y = struct;
y.size = uint16([xm xn]);
y.numblocks = uint16(xb);
y.quality = uint16(quality * 100);
y.huffman = mat2huff(r);
```



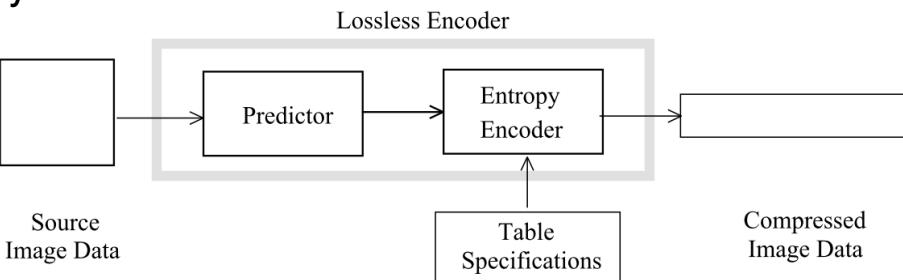
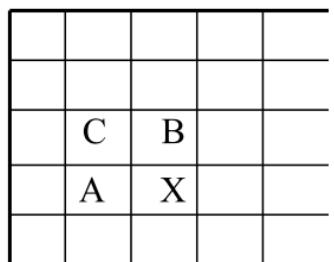
Back

Close

Further Information

Further standards:

- Lossless JPEG: Predictive approach for lossless compression (why?), not widely used



- JPEG 2000: ISO/IEC 15444

- Based on **wavelet** transform, instead of DCT, no 8×8 blocks, less artefacts
- Often better compression ratio, compared with JPEG



Back

Close

Further Information

References:

- <http://www.jpeg.org>
- [Online JPEG Tutorial](#)
- [The JPEG Still Picture Compression Standard](#)
- [The JPEG 2000 Still Image Compression Standard](#)



Back

Close

Compression:

Video Compression (MPEG and others)

We need to compress video (more so than audio/images) in practice since:

1. Uncompressed video (and audio) data are huge.

In HDTV, the bit rate easily **exceeds 1 Gbps**. — big problems for storage and network communications.

E.g. HDTV: 1920 x 1080 at 30 frames per second, 8 bits per YCbCr (PAL) channel = **1.5 Gbps**.

2. Lossy methods have to be employed since the **compression ratio** of lossless methods (e.g., Huffman, Arithmetic, LZW) is not high enough for image and video compression.



Back

Close

Not the complete picture studied here

Much more to MPEG — Plenty of other tricks employed.

We only concentrate on some basic principles of video compression:

- Earlier H.261 and MPEG 1 and 2 standards.
with a brief introduction of ideas used in new standards such as **H.264 (MPEG-4 Advanced Video Coding)**.

Compression Standards Committees

Image, Video and Audio Compression standards have been specified and released by two main groups since 1985:

ISO - International Standards Organisation: JPEG, MPEG.

ITU - International Telecommunications Union: H.261 — 264.

Compression Standards

Whilst in many cases one of the groups have specified separate standards there is some crossover between the groups.

For example:

- JPEG issued by ISO in 1989 (but adopted by ITU as ITU T.81)
- MPEG 1 released by ISO in 1991,
- H.261 released by ITU in 1993 (based on CCITT 1990 draft).
CCITT stands for **Comité Consultatif International Téléphonique et Télégraphique** (International Telegraph and Telephone Consultative Committee) whose parent organisation is ITU.
- H.262 is alternatively better known as MPEG-2 released in 1994.
- H.263 released in 1996 extended as H.263+, H.263++.
- MPEG 4 release in 1998.
- H.264 releases in 2002 to lower the bit rates with comparable quality video and support wide range of bit rates, and is now part of MPEG 4 (Part 10, or AVC – Advanced Video Coding).

How to compress video?

Basic Idea of Video Compression:

Motion Estimation/Compensation

- Spatial Redundancy Removal – Intraframe coding (JPEG)
NOT ENOUGH BY ITSELF?
- Temporal — Greater compression by noting the temporal coherence/incoherence over frames. Essentially we note the difference between frames.
- Spatial and Temporal Redundancy Removal – Intraframe and Interframe coding (H.261, MPEG)



Back

Close

Simple Motion Estimation/Compensation Example

Things are much more complex in practice of course.

Which Format to represent the compressed data?

- Simply based on Differential Pulse Code Modulation (DPCM).



Back

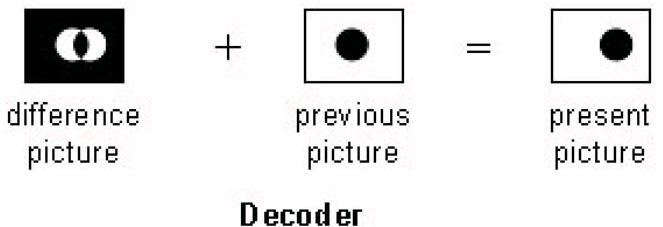
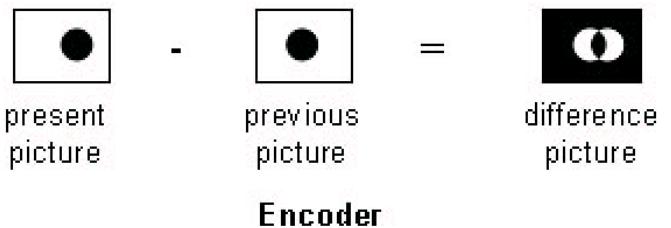
Close

Simple Motion Example (Cont.)

Consider a simple image (block) of a moving circle.

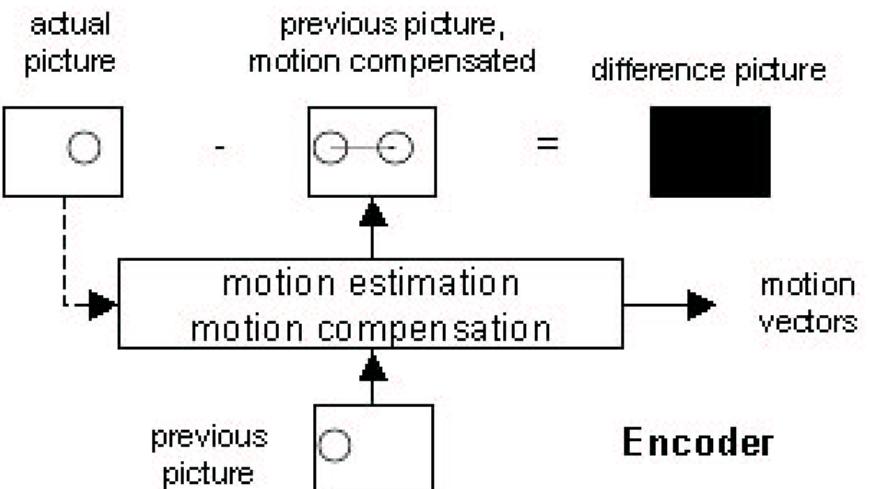
Lets just consider the difference between 2 frames.

It is simple to encode/decode:

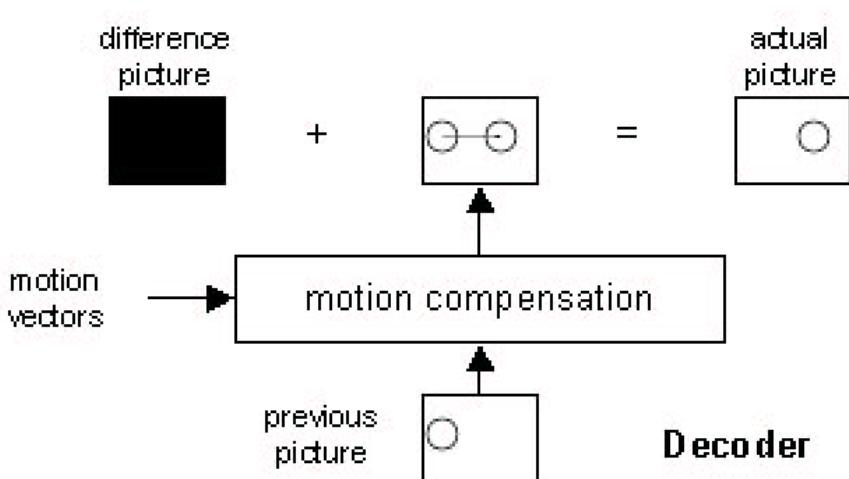


Now lets Estimate Motion of blocks

We will examine methods of estimating motion vectors in due course.



Decoding Motion of blocks



Why is this a better method than just frame differencing?



Back

Close

How is Block Motion used in Compression?

Block Matching:

- MPEG-1/H.261 is done by using block matching techniques,

For a certain area of pixels in a picture:

- find a good estimate of this area in a previous (or in a future) frame, within a specified search area.

Motion compensation:

- Uses the motion vectors to compensate the picture.
- Parts of a previous (or future) picture can be reused in a subsequent picture.
- Individual parts spatially compressed — JPEG type compression



Back

Close

Any Overheads?

- Motion estimation/compensation techniques reduces the video bitrate significantly
but
- Introduce extras computational complexity and delay (?),
 - Need to buffer reference pictures - backward and forward referencing.
 - Reconstruct from motion parameters

Lets see how such ideas are used in practice.



Back

Close

H.261 Compression

The basic approach to H. 261 Compression is summarised as follows:

H. 261 Compression has been specifically designed for video telecommunication applications:

- Developed by CCITT in 1988-1990
- Meant for videoconferencing, videotelephone applications over ISDN telephone lines.
- Baseline ISDN is 64 kbits/sec, and integral multiples ($px64$)



Back

Close

Overview of H.261

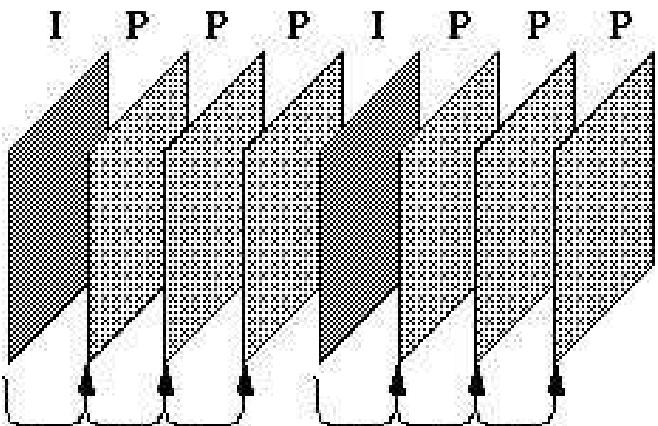
- Frame types are CCIR 601 CIF (Common Intermediate Format) (352x288) and QCIF (176x144) images with 4:2:0 subsampling.
- Two frame types:
Intraframes (I-frames) and **Interframes (P-frames)**
- I-frames use basically JPEG — **but** YUV (YCrCb) and **larger** DCT windows, **different** quantisation
- I-frames provide us with a refresh accessing point — **Key Frames**
- P-frames use **pseudo-differences** from previous frame (predicted), so frames depend on each other.



Back

Close

H.261 Group of Pictures



- We typically have a group of pictures — one *I-frame* followed by several *P-frames* — a **group of pictures**
- Number of *P-frames* followed by each *I-frame* determines the size of GOP – can be fixed or dynamic. Why this can't be too large?

Intra Frame Coding

- Various lossless and lossy compression techniques use — like JPEG.
- Compression contained only within the current frame
- Simpler coding – Not enough by itself for high compression.
- Can't rely on intra frame coding alone not enough compression:
 - Motion JPEG (MJPEG) standard does exist — not commonly used.
 - So introduce idea of inter frame difference coding
- However, can't rely on inter frame differences across a large number of frames
 - So when Errors get too large: Start a new I-Frame

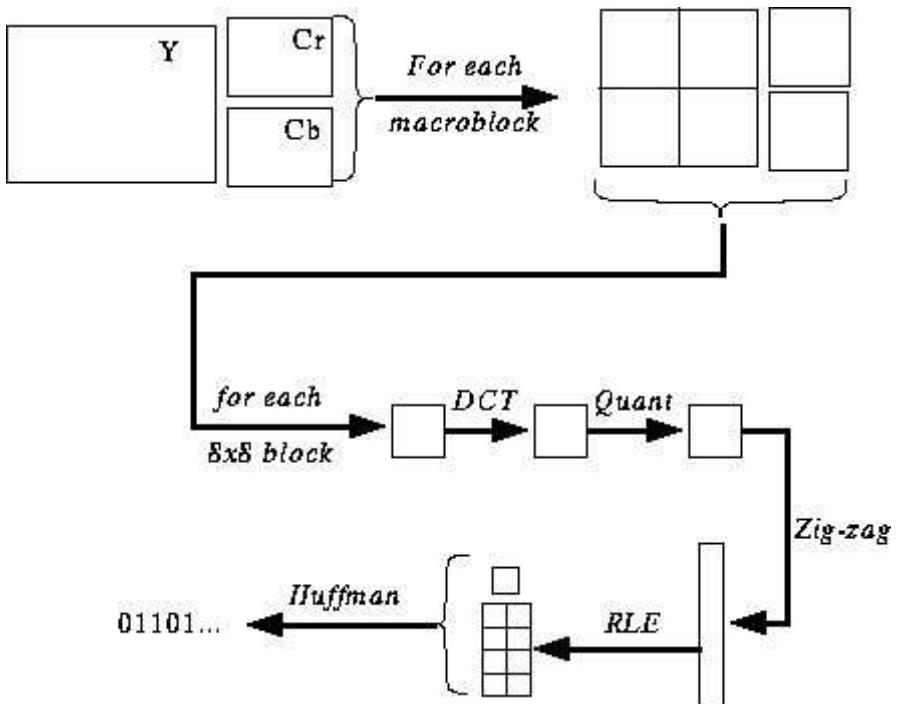


Back

Close

Intra Frame Coding (Cont.)

Intraframe coding is very similar to JPEG:



What are the differences between this and JPEG?

Intra Frame Coding (Cont.)

A basic Intra Frame Coding Scheme is as follows:

- Macroblocks are typically 16x16 pixel areas on Y plane of original image.
- A **macroblock** usually consists of 4 Y blocks, 1 Cr block, and 1 Cb block. (4:2:0 chroma subsampling)
 - Eye most sensitive to luminance, less sensitive to chrominance.
 - * We operate on a **more effective** color space: YUV (YCbCr) colour which we studied earlier.
 - Typical to use 4:2:0 macroblocks: one quarter of the chrominance information used.
- Quantization is by constant value for all DCT coefficients.
I.e., no quantization table as in JPEG.

Inter-frame (P-frame) Coding

- Intra frame limited to spatial basis relative to 1 frame
- Considerable more compression if the inherent temporal basis is exploited as well.

BASIC IDEA:

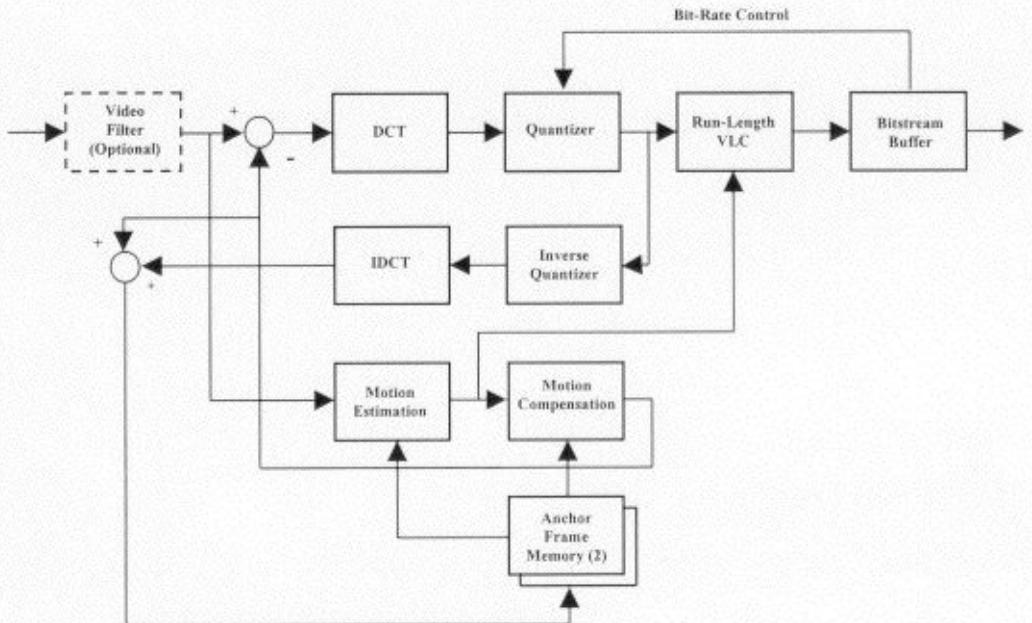
- Most consecutive frames within a sequence are very similar to the frames both before (and after) the frame of interest.
- Aim to exploit this redundancy.
- Use a technique known as **block-based motion compensated prediction**
- Need to use **motion estimation**
- Coding needs extensions for **Inter** but encoder can also supports an **Intra** subset.



Back

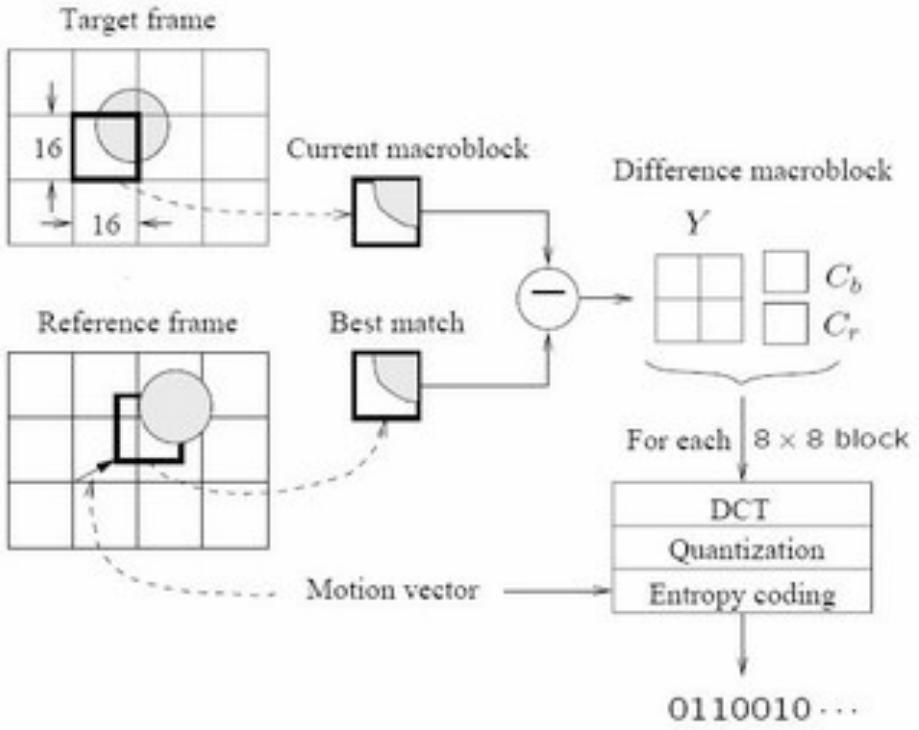
Close

Inter-frame (P-frame) Coding (Cont.)



Inter-frame (P-frame) Coding (Cont.)

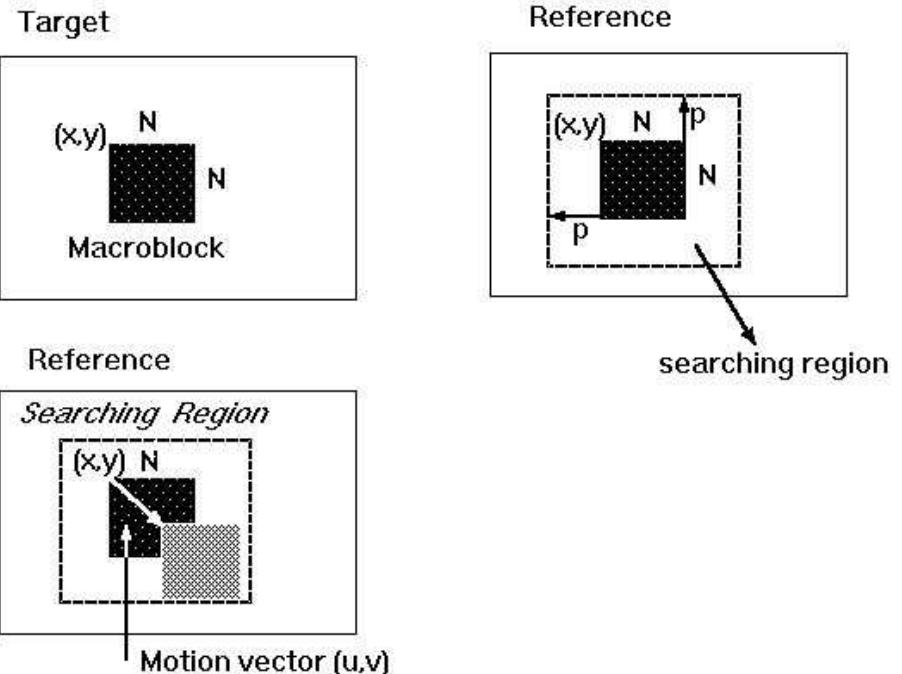
P-coding can be summarised as follows:



Motion Vector Search

So we know how to encode a P-block.

How do we find the motion vector?



Back

Close

Motion Estimation

- The temporal prediction technique used in MPEG video is based on motion estimation.

The basic premise:

- Consecutive video frames will be similar except for changes induced by objects moving within the frames.
- Trivial case of zero motion between frames — no other differences except noise, etc.),
- Easy for the encoder to predict the current frame as a duplicate of the prediction frame.
- When there is motion in the images, the situation is not as simple.



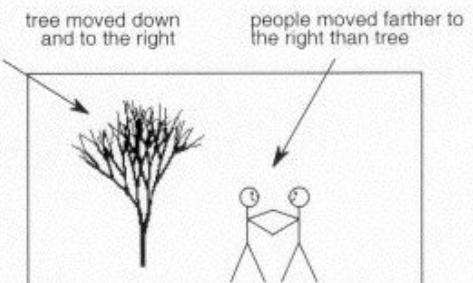
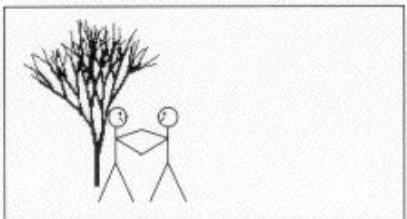
Back

Close

Example of a frame with 2 stick figures and a tree

The problem for motion estimation to solve is :

- How to adequately represent the changes, or differences, between these two video frames.



Solution:

A comprehensive 2-dimensional spatial search is performed for each luminance macroblock.

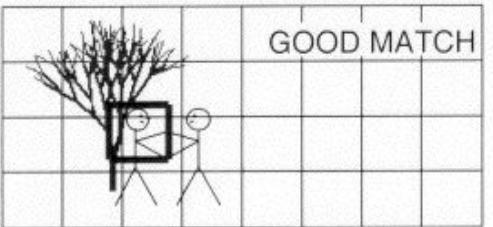
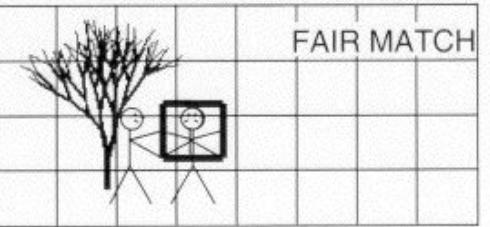
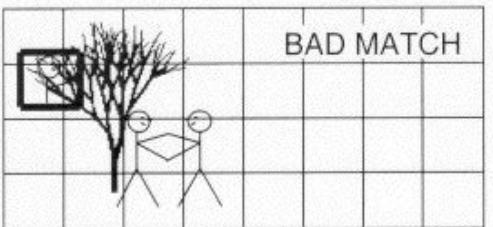
- Motion estimation is not applied directly to chrominance in MPEG
- MPEG does *not* define how this search should be performed.
- A detail that the system designer can choose to implement in one of many possible ways.
- Well known that a full, exhaustive search over a wide 2-D area yields the best matching results in most cases, but at extreme computational cost to the encoder.
- Motion estimation usually is the most computationally expensive portion of the video encoding.



Back

Close

Motion Estimation Example



Macroblock to be coded



Back

Close

Motion Vectors, Matching Blocks

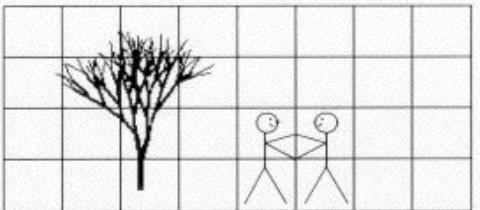
Previous figure shows an example of a particular macroblock from Frame 2 of earlier example, relative to various macroblocks of Frame 1:

- The top frame has a bad match with the macroblock to be coded.
- The middle frame has a fair match, as there is some commonality between the 2 macroblocks.
- The bottom frame has the best match, with only a slight error between the 2 macroblocks.
- Because a relatively good match has been found, the encoder assigns motion vectors to that macroblock,

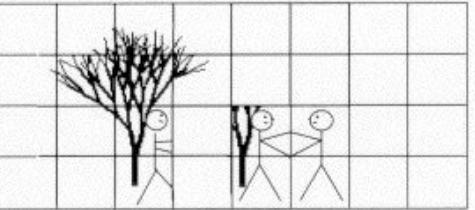


Back

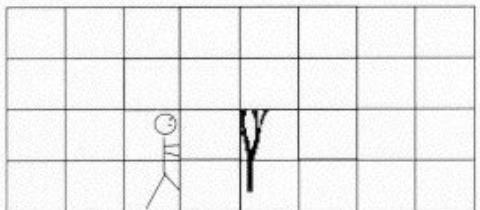
Close



Desired Picture



Minus Predicted Picture



Residual Error Picture
(Coded & Transmitted)

Final Motion Estimation Prediction (Cont.)

Previous figure shows how a potential predicted Frame 2 can be generated from Frame 1 by using motion estimation.

- The predicted frame is subtracted from the desired frame,
- Leaving a (hopefully) less complicated residual error frame which can then be encoded **much more efficiently** than **before** motion estimation.
- The more accurate the motion is estimated and matched, the more likely it will be that the residual error will approach zero — and the coding efficiency will be **highest**.



Back

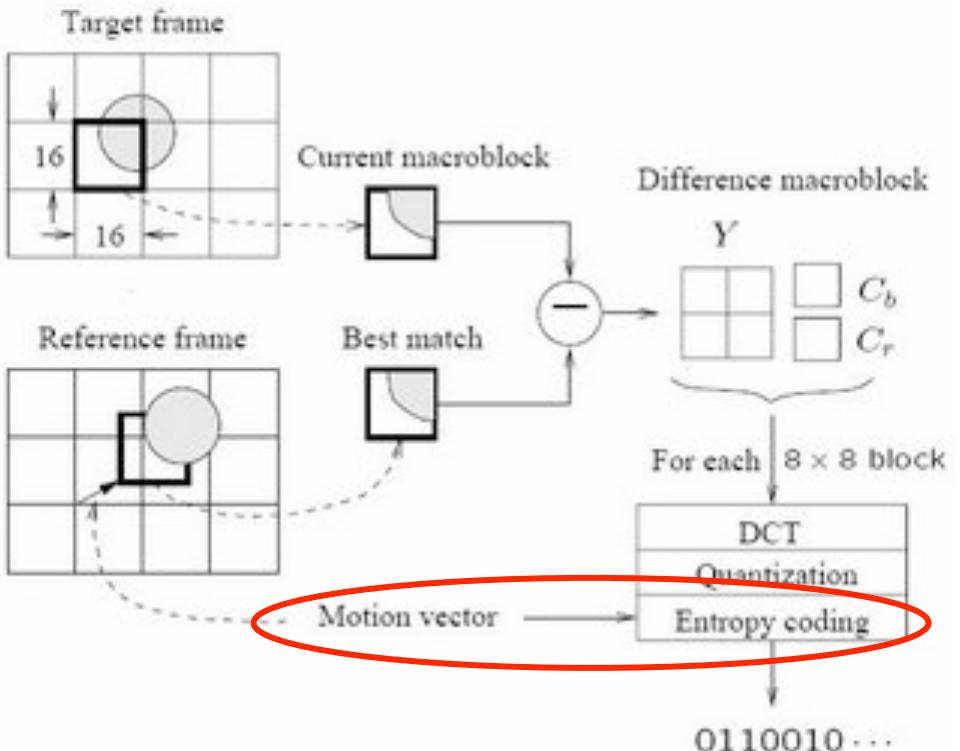
Close

Further coding efficiency

Differential Coding of Motion Vectors

- Motion vectors tend to be highly correlated between macroblocks:
 - The horizontal component is compared to the previously valid horizontal motion vector and
 - * Only the **difference** is coded.
 - Same difference is calculated for the **vertical** component
 - Difference codes are then described with a **variable length code** (e.g. Huffman) for **maximum** compression efficiency.

RECAP: P-Frame Coding Summary



What Happens if we can't find acceptable match?

P Blocks may not be what they appear to be?

If the encoder decides that no acceptable match exists then it has the option of

- Coding that particular macroblock as an intra macroblock,
- Even though it may be in a P frame!
- In this manner, high quality video is maintained at a slight cost to coding efficiency.



Back

Close

Estimating the Motion Vectors

So How Do We Find The Motion?

Basic Ideas is to search for Macroblock (MB)

- Within a $\pm n \times m$ pixel search window
- Work out **Sum of Absolute Difference (SAD)**
(or Mean Absolute Error (MAE) for each window but this is computationally more expensive)
- Choose window where SAD/MAE is a **minimum**.



Back

Close

Sum of Absolute Difference (SAD)

SAD is computed by:

For i = -n to +n

For j = -m to +m

$$SAD(i, j) = \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} |C(x+k, y+l) - R(x+i+k, y+j+l)|$$

- N = size of Macroblock window typically (16 or 32 pixels),
- (x, y) the position of the original Macroblock, C, and
- R is the **reference** region to compute the SAD.
- $C(x+k, y+l)$ – pixels in the macro block with upper left corner (x, y) in the Target.
- $R(x+i+k, y+j+l)$ – pixels in the macro block with upper left corner $(x+i, y+j)$ in the Reference.



Mean Absolute Error (MAE)

- Cost function is:

$$MAE(i, j) = \frac{1}{N^2} \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} |C(x + k, y + l) - R(x + i + k, y + j + l)|$$

$$MAE(i, j) = \frac{1}{N^2} \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} |C(x+k, y+l) - R(x+i+k, y+j+l)|$$

Search For Minimum

- Goal is to find a vector (u, v) such that SAD/MAE (u, v) is minimum
 - Full Search Method
 - Two-Dimensional Logarithmic Search
 - Hierarchical Search

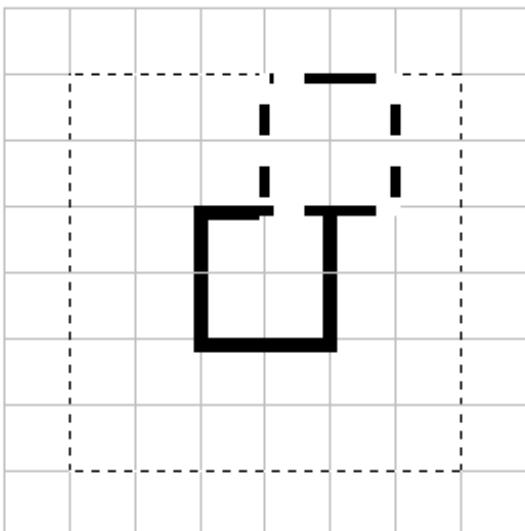


Back

Close

SAD Search Example

So for a $\pm 2 \times 2$ Search Area given by dashed lines and a 2×2 Macroblock window example, the *SAD* is given by bold dot dash line (near top right corner):



Full Search

- Search exhaustively the whole $(2n + 1) \times (2m + 1)$ window in the **Reference** frame.
- A macroblock centred at each of the positions within the window is compared to the macroblock in the *target* frame pixel by pixel and their respective SAD (or MAE) is computed.
- The vector (i, j) that offers the least SAD (or MAE) is designated as the motion vector (u, v) for the macroblock in the *target* frame.
- **Full search is very costly** — assuming each pixel comparison involves three operations (subtraction, absolute value, addition), the cost for finding a motion vector for a single macroblock is $(2n + 1) \cdot (2m + 1) \cdot N^2 \cdot 3 = O(nmN^2)$.



Back

Close

2D Logarithmic Search

- An approach takes several iterations akin to a binary search.
- Computationally **cheaper, suboptimal** but **usually effective**.
- Initially only nine locations in the search window are used as seeds for a SAD(MAE)-based search (marked as ‘1’).
- After locating the one with the minimal SAD(MAE), the centre of the new search region is moved to it and the step-size (“offset”) is reduced to half.
- In the next iteration, the nine new locations are marked as ‘2’ and this process repeats.
- If L iterations are applied, for altogether 9^L positions, only $9L$ positions are checked.

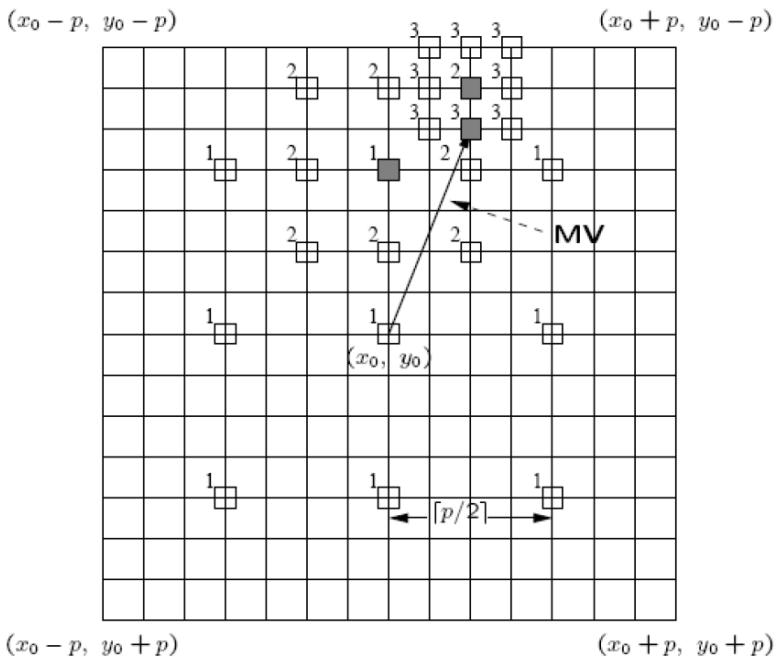


Back

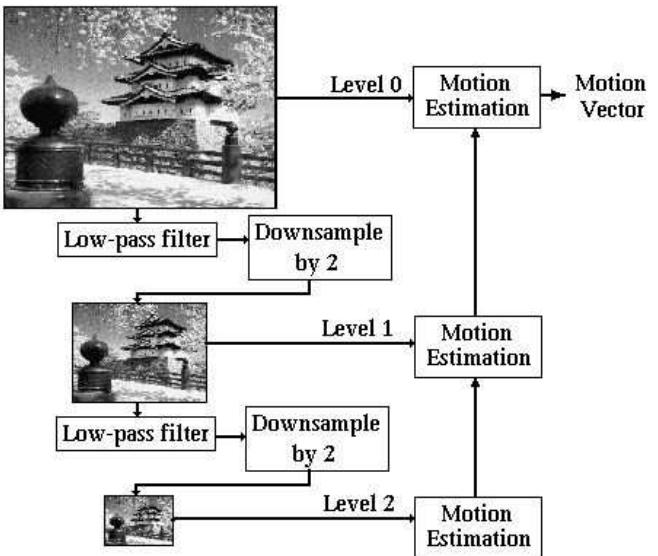
Close

2D Logarithmic Search (cont.)

2D Logarithmic Search for Motion Vectors



Hierarchical Motion Estimation:



1. Form several low resolution version of the target and reference pictures
2. Find the best match motion vector in the lowest resolution version.
3. Modify the motion vector level by level when going up



Back

Close

Selecting Intra/Inter Frame coding

Based upon the motion estimation a decision is made on whether INTRA or INTER coding is made.

Multimedia
CM0340

499

To determine INTRA/INTER MODE we do the following calculation:

$$MB_{mean} = \frac{\sum_{i=0,j=0}^{N-1} |C(i,j)|}{N^2}$$

$$A = \sum_{i=0,j=0}^{N-1} | C(i,j) - MB_{mean} |$$

If $A < (SAD - 2N)$ INTRA Mode is chosen.



Back

Close

MPEG Compression

MPEG stands for:

- **Motion Picture Expert Group** — established circa 1990 to create standard for delivery of audio and video
- MPEG-1 (1991). Target: VHS quality on a CD-ROM (320 x 240 + CD audio @ 1.5 Mbits/sec)
- MPEG-2 (1994): Target Television Broadcast
- MPEG-3: HDTV but subsumed into an extension of MPEG-2
- MPEG 4 (1998): Very Low Bitrate Audio-Visual Coding, later MPEG-4 Part 10(H.264) for wide range of bitrates and better compression quality
- MPEG-7 (2001) “Multimedia Content Description Interface”
- MPEG-21 (2002) “Multimedia Framework”

Three Parts to MPEG

- The MPEG standard had three parts:
 1. Video: based on H.261 and JPEG
 2. Audio: based on MUSICAM (Masking pattern adapted Universal Subband Integrated Coding And Multiplexing) technology
 3. System: control interleaving of streams



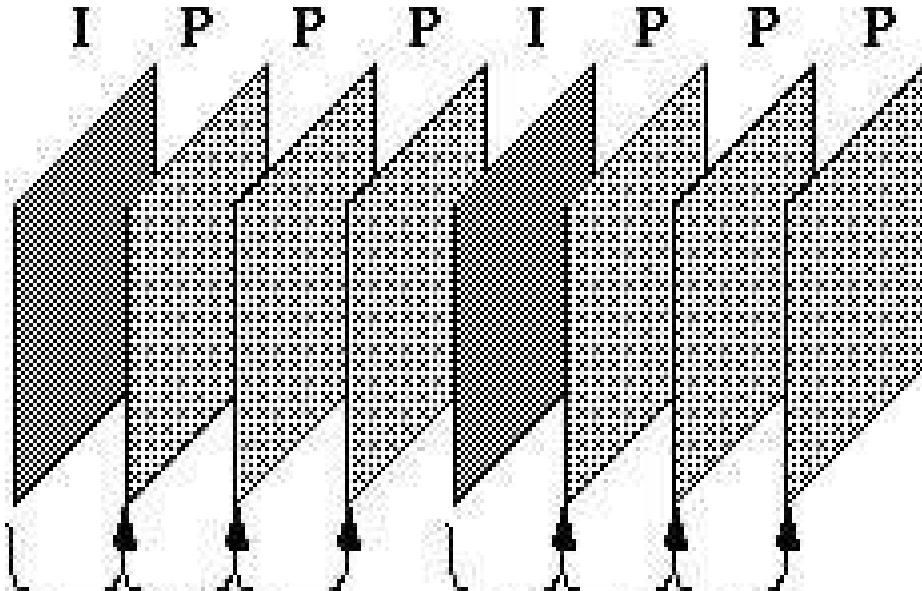
Back

Close

MPEG Video

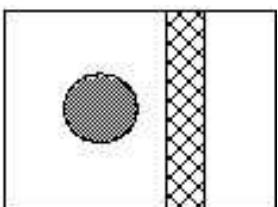
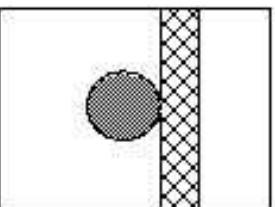
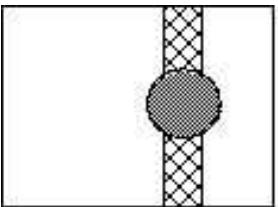
MPEG compression is essentially an attempt to overcome some shortcomings of H.261 and JPEG:

- Recall H.261 dependencies:



The Need for a Bidirectional Search

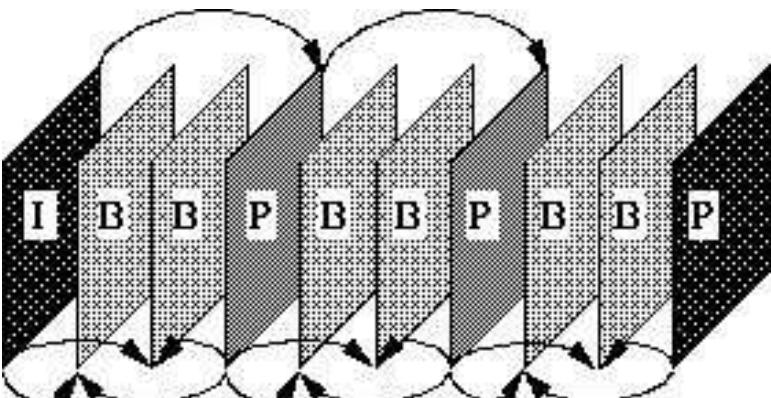
- The Problem here is that many macroblocks need information that is **not** in the reference frame.
- For example:



- Occlusion by objects affects differencing
- Difficult to track occluded objects etc.
- MPEG uses **forward/backward** interpolated prediction.

MPEG B-Frames

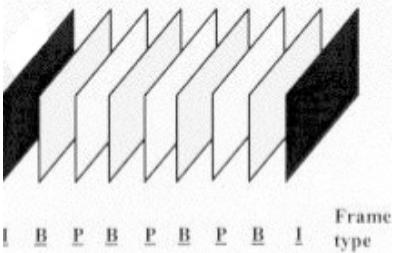
- The **MPEG solution** is to add a third frame type which is a **bidirectional** frame, or *B-frame*
- B-frames search for macroblock in *past* and *future* frames.
- Typical pattern is IBBPBBPBB IBBPBBPBB IBBPBBPBB
Actual pattern is up to encoder, and need not be regular.



Example I, P, and B frames

Consider a group of pictures that lasts for 6 frames:

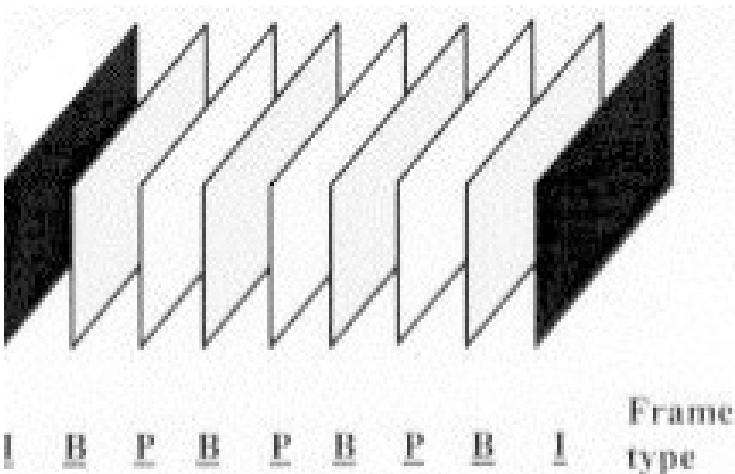
- Given: I,B,P,B,P,B,I,B,P,B,P,B,



- I frames are coded spatially only (as before in H.261).
- P frames are forward predicted based on previous I and P frames (as before in H.261).
- B frames are coded based on a forward prediction from a **previous** I or P frame, **as well** as a **backward prediction** from a **succeeding** I or P frame.

Example I, P, and B frames (Cont.)

- 1st B frame is predicted from the 1st I frame and 1st P frame.
- 2nd B frame is predicted from the 1st and 2nd P frames.
- 3rd B frame is predicted from the 2nd and 3rd P frames.
- 4th B frame is predicted from the 3rd P frame and the 1st I frame of the **next** group of pictures.

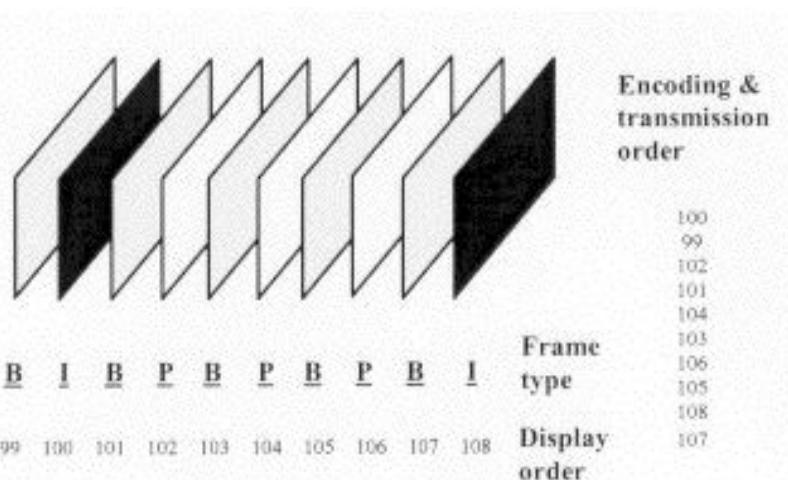


Backward Prediction Implications

Note: Backward prediction requires that the **future frames** that are to be used for **backward prediction** be

- Encoded and Transmitted **first**, *i.e.* **out of order**.

This process is summarised:



Backward Prediction Implications (Cont.)

Also NOTE:

- No defined limit to the number of consecutive B frames that may be used in a group of pictures,
- Optimal number is application dependent.
- Most broadcast quality applications however, have tended to use 2 consecutive B frames (I,B,B,P,B,B,P,) as the ideal trade-off between compression efficiency and video quality.
- MPEG suggests some standard groupings.



Back

Close

Advantage of the usage of B frames

- Coding efficiency.
- Most B frames use less bits.
- Quality can also be improved in the case of moving objects that reveal hidden areas within a video sequence.
- Better Error propagation: B frames are not used to predict future frames, errors generated will not be propagated further within the sequence.

Disadvantage:

- Frame reconstruction memory buffers within the encoder and decoder must be doubled in size to accommodate the 2 anchor frames.
- More delays in real-time applications.



Back

Close

MPEG-2, MPEG-3, and MPEG-4

- MPEG-2 target applications

Level	size	Pixels/sec	bit-rate (Mbits)	Application
Low	352 x 240	3 M	4	VHS tape equiv.
Main	720 x 480	10 M	15	studio TV
High	1440 x 1152	47 M	60	consumer HDTV
High	1920 x 1080	63 M	80	film production

- MPEG-2 differences from MPEG-1

1. Search on fields, not just frames.
2. 4:2:2 and 4:4:4 macroblocks
3. Frame sizes as large as 16383 x 16383
4. Scalable modes: Temporal, Progressive,...
5. Non-linear macroblock quantization factor
6. A bunch of minor fixes

MPEG-2, MPEG-3, and MPEG-4 (Cont.)

- MPEG-3: Originally for HDTV (1920 x 1080), got folded into MPEG-2
- MPEG-4: very low bit-rate communication (4.8 to 64 kb/sec).
Video processing

MATLAB MPEG Video Coding Code

[MPEGVideo](#) (DIRECTORY)

[MPEGVideo.zip](#) (All Files Zipped)



Back

Close

MPEG-4 Video Compression

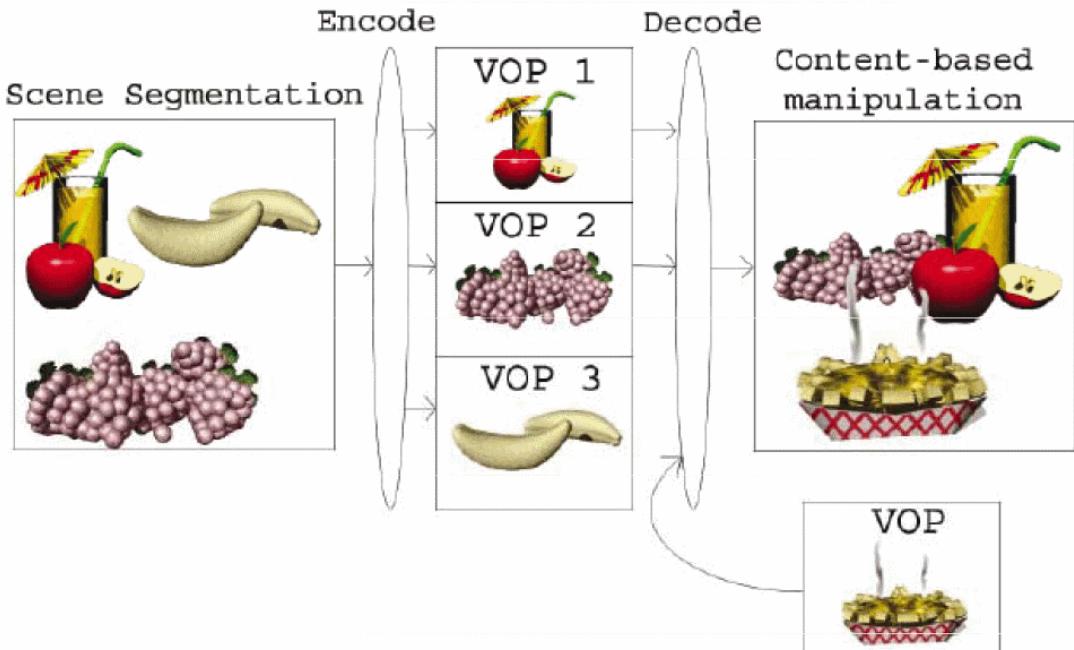
A newer standard than MPEG-2. Besides compression, great attention is paid to issues about **user interactivity**.

We look at **key ideas** here.

- **Object based coding**: offers higher compression ratio, also beneficial for digital video composition, manipulation, indexing and retrieval.
- **Synthetic object coding**: supports 2D mesh object coding, face object coding and animation, body object coding and animation.
- **MPEG-4 Part 10/H.264**: new techniques for improved compression efficiency.

Object based coding (1)

Composition and manipulation of MPEG-4 videos.



Object based coding (2)

Compared with MPEG-2, MPEG-4 is an entirely new standard for

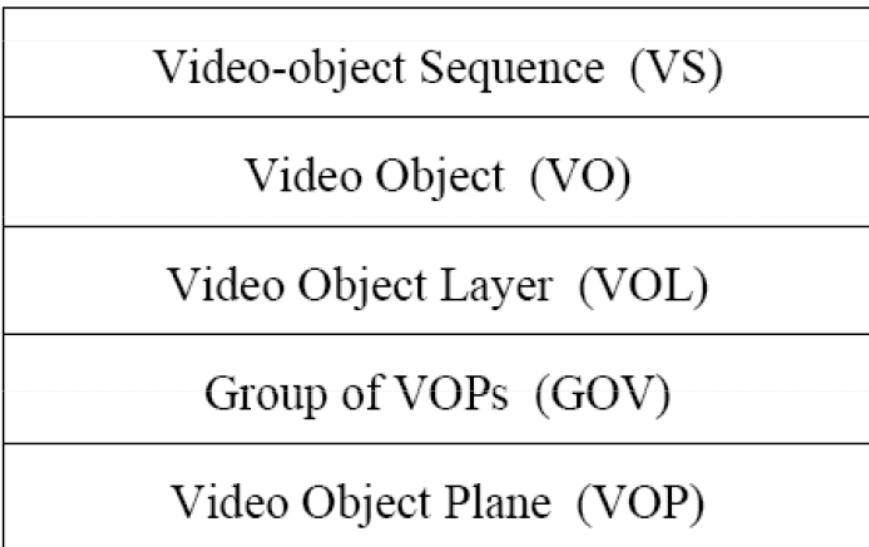
- **Composing** media objects to create desirable audiovisual scenes.
- **Multiplexing** and **synchronising** the bitstreams for these media data entities so that they can be transmitted with guaranteed **Quality of Service (QoS)**.
- **Interacting** with the audiovisual scene at the receiving end.

MPEG-4 provides a set of advanced coding modules and algorithms for audio and video compressions.

We have discussed MPEG-4 Structured Audio and we will focus on video here.

Object based coding (3)

The hierarchical structure of MPEG-4 visual bitstreams is very different from that of MPEG-2: it is very much **video object-oriented**:



Back

Close

Object based coding (4)

- **Video-object Sequence (VS)**: delivers the complete MPEG4 visual scene; may contain 2D/3D natural or synthetic objects.
- **Video Object (VO)**: a particular object in the scene, which can be of arbitrary (non-rectangular) shape corresponding to an object or background of the scene.
- **Video Object Layer (VOL)**: facilitates a way to support (multi-layered) scalable coding. A VO can have multiple VOLs under scalable (multi-bitrate) coding, or have a single VOL under non-scalable coding.
- **Group of Video Object Planes (GOV)**: groups of video object planes together (optional level).
- **Video Object Plane (VOP)**: a snapshot of a VO at a particular moment.



Back

Close

VOP-based vs. Frame-based Coding

- MPEG-1 and MPEG-2 do **not** support the VOP concert; their coding method is **frame-based** (also known as **block-based**).
- For block-based coding, it is possible that multiple potential matches yield small prediction errors. Some may not coincide with the real motion.
- For VOP-based coding, each VOP is of **arbitrary shape** and ideally will obtain a unique motion vector consistent with the actual object motion.

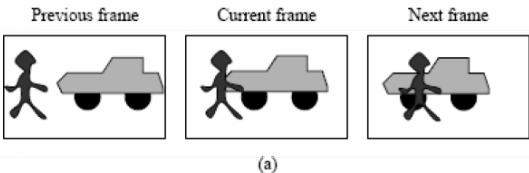
See the example in the next page:



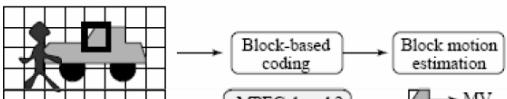
Back

Close

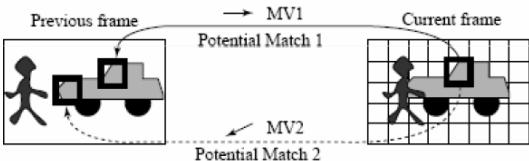
VOP-based vs. Frame-based Coding



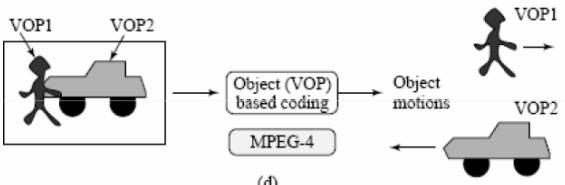
(a)



(b)



(c)



(d)

VOP-based Coding

- MPEG-4 VOP-based coding also employs Motion Compensation technique:
 - I-VOPs: **Intra-frame** coded VOPs.
 - P-VOPs: **Inter-frame** coded VOPs if only forward prediction is employed.
 - B-VOPs: **Inter-frame** coded VOPs if bi-directional predictions are employed.
- The new difficulty for VOPs: may have **arbitrary shapes**. Shape information must be coded in addition to the texture (luminance or chroma) of the VOP.



Back

Close

VOP-based Motion Compensation (MC)

- MC-based VOP coding in MPEG-4 again involves three steps:
 1. Motion Estimation
 2. MC-based Prediction
 3. Coding of the Prediction Error
- Only pixels within the VOP of the current (target) VOP are considered for matching in MC. To facilitate MC, each VOP is divided into macroblocks with 16×16 luminance and 8×8 chrominance images.



Back

Close

VOP-based Motion Compensation (MC)

- Let $C(x + k, y + l)$ be pixels of the MB in target in target VOP, and $R(x + i + k, y + j + l)$ be pixels of the MB in Reference VOP.
- A **Sum of Absolute Difference (SAD)** for measuring the difference between the two MBs can be defined as:

$$SAD(i, j) = \sum_{k=0}^{N-1} \sum_{l=0}^{N-1} |C(x + k, y + l) - R(x + i + k, y + j + l)| \cdot Map(x + k, y + l).$$

N — the size of the MB

$Map(p, q) = 1$ when $C(p, q)$ is a pixel within the target VOP
otherwise $Map(p, q) = 0$.

- The vector (i, j) that yields the **minimum SAD** is adopted as the motion vector (u, v) .



Back

Close

Coding of Texture and Shape

- Texture Coding (luminance and chrominance):
 - I-VOP: the gray values of the pixels in each MB of the VOP are directly coded using **DCT followed by VLC (Variable Length Coding)**, such as Huffman or Arithmetic Coding.
 - P-VOP/B-VOP: MC-based coding is employed — the **prediction error** is coded similar to I-VOP.
 - Boundary MBs need appropriate treatment. May also use improved Shape Adaptive DCT.



Back

Close

Coding of Texture and Shape (cont.)

- Shape Coding (shape of the VOPs)
 - Binary shape information: in the form of a binary map. A value ‘1’ (opaque) or ‘0’ (transparent) in the bitmap indicates whether the pixel is inside or outside the VOP.
 - Greyscale shape information: value refers to the **transparency** of the shape ranging from 0 (completely transparent) and 255 (opaque).
 - Specific encoding algorithms are designed to code in both cases.

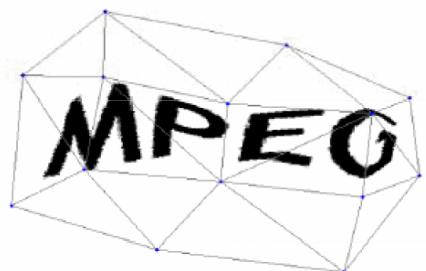


Back

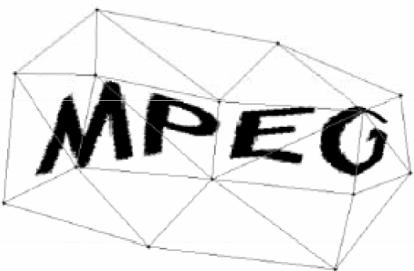
Close

Synthetic Object Coding: 2D Mesh

- 2D Mesh Object: a tessellation (or partition) of a 2D planar region using **polygonal patches**.
- Mesh based texture mapping can be used for 2D object animation.



(a)



(b)



Back

Close

Synthetic Object Coding: 3D Model

- MPEG-4 has defined special **3D models** for **face objects** and **body objects** because of the frequent appearances of human faces and bodies in videos.
- Some of the potential **applications**: teleconferencing, human-computer interfaces, games and e-commerce.
- MPEG-4 goes beyond wireframes so that the surfaces of the face or body objects can be shaded or texture-mapped.



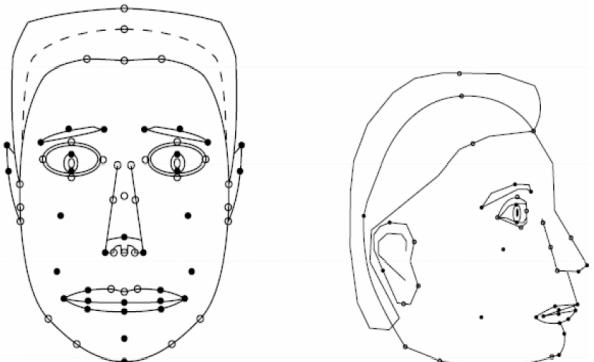
Back

Close

Synthetic Object Coding: Face Object

Face Object Coding and Animation

- MPEG-4 adopted a generic default face model, developed by VRML Consortium.
- **Face Animation Parameters (FAPs)** can be specified to achieve desirable animation.
- **Face Definition Parameters (FDPs)**: feature points better describe individual faces.



MPEG-4 Part 10/H.264

- Improved video coding techniques, identical standards: ISO MPEG-4 Part 10 (Advanced Video Coding / AVC) and ITU-T H.264.
- Preliminary studies using software based on this new standard suggests that H.264 offers up to 30-50% better compression than MPEG-2 and up to 30% over H.263+ and MPEG-4 advanced simple profile.
- H.264 is currently one of the leading candidates to carry **High Definition TV (HDTV)** video content on many applications, e.g. Blu-ray.
- Involves various technical improvements. We mainly look at improved inter-frame encoding.

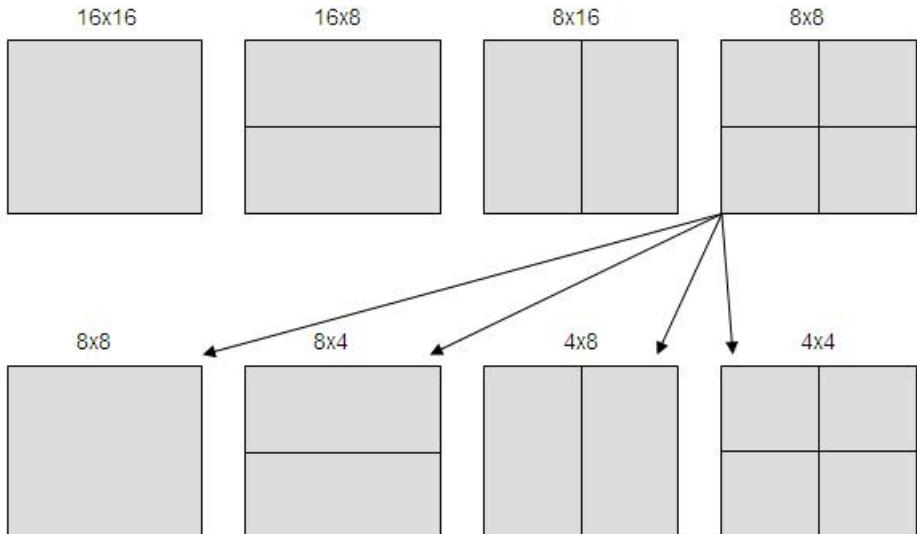


Back

Close

MPEG-4 AVC: Flexible Block Partition

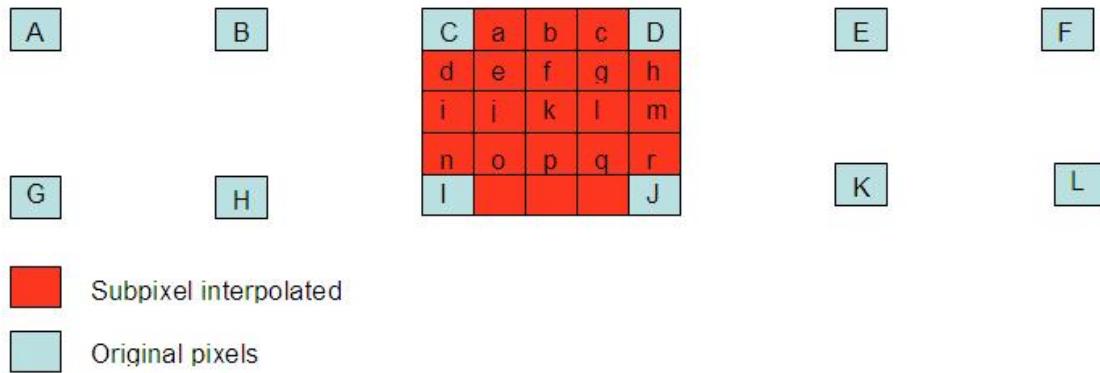
Macroblock in MPEG-2 uses 16×16 luminance values. MPEG-4 AVC uses a tree-structured motion segmentation down to 4×4 block sizes (16×16 , 16×8 , 8×16 , 8×8 , 8×4 , 4×8 , 4×4). This allows much more accurate motion compensation of moving objects.



MPEG-4 AVC: Up to Quarter-Pixel Resolution Motion Compensation

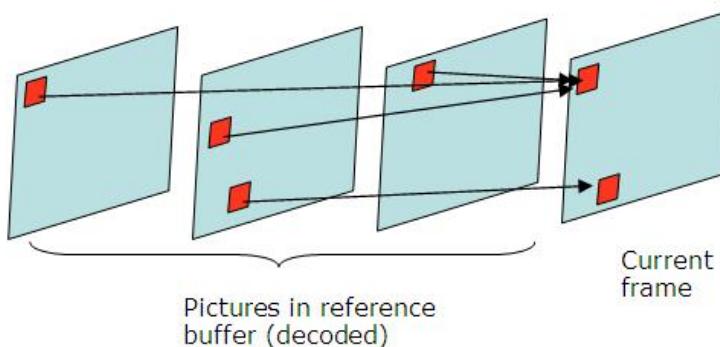
Motion vectors can be up to half-pixel or quarter-pixel accuracy.
Pixels at quarter-pixel position are obtained by bilinear interpolation.

- Improves the possibility of finding a block in the reference frame that better matches the target block.



MPEG-4 AVC: Multiple References

- Multiple references to motion estimation. Allows finding the best reference in 2 possible buffers (past pictures and future pictures) each contains up to 16 frames.
- Block prediction is done by a weighted sum of blocks from the reference picture. It allows enhanced picture quality in scenes where there are changes of plane, zoom, or when new objects are revealed.



Further Reading

- [Overview of the MPEG-4 Standard](#)
- [The H.264/MPEG4 AVC Standard and its Applications](#)

Compression: Audio Compression (MPEG and others)

As with video a number of compression techniques have been applied to audio.

Simple Audio Compression Methods

RECAP (Already Studied)

Traditional lossless compression methods (Huffman, LZW, etc.) usually don't work well on audio compression

- For the same reason as in image and video compression:
Too much change variation in data over a short time



Back

Close

Some Simple But Limited Practical Methods

- Silence Compression - detect the "silence", similar to run-length encoding (**seen examples before**)
- Differential Pulse Code Modulation (DPCM)

Relies on the fact that difference in amplitude in successive samples is small then we can used reduced bits to store the difference (**seen examples before**)



Back

Close

Simple But Limited Practical Methods (Cont.)

- Adaptive Differential Pulse Code Modulation (ADPCM)
 - e.g., in CCITT G.721 – 16 or 32 Kbits/sec.
 - (a) Encodes the difference between two consecutive signals but a refinement on DPCM,
 - (b) Adapts at quantisation so fewer bits are used when the value is smaller.
 - It is necessary to predict where the waveform is heading
 - > **difficult**
 - Apple had a proprietary scheme called ACE/MACE. Lossy scheme that tries to predict where wave will go in next sample. About 2:1 compression.

Simple But Limited Practical Methods (Cont.)

- Adaptive Predictive Coding (APC) typically used on Speech.
 - Input signal is divided into fixed segments (**windows**)
 - For each segment, some sample **characteristics** are computed, **e.g. pitch, period, loudness**.
 - These characteristics are used to predict the signal
 - Computerised talking (Speech Synthesisers use such methods) but low bandwidth:

Acceptable quality at 8 kbits/sec

Simple But Limited Practical Methods (Cont.)

- Linear Predictive Coding (LPC) fits signal to speech model and then transmits parameters of model as in APC.

Speech Model:

- Speech Model:

Pitch, period, loudness, vocal tract parameters (voiced and unvoiced sounds).

- Synthesised speech
- More prediction coefficients than APC – lower sampling rate
- Still sounds like a computer talking,
- Bandwidth as low as 2.4 kbits/sec.

Simple But Limited Practical Methods (Cont.)

- Code Excited Linear Predictor (CELP) does LPC, but also transmits error term.
 - Based on more sophisticated model of vocal tract than LPC
 - Better perceived speech quality
 - Audio conferencing quality at 4.8 kbits/sec.



Back

Close

Psychoacoustics or Perceptual Coding

Basic Idea: Exploit areas where the human ear is **less sensitive** to sound to achieve compression

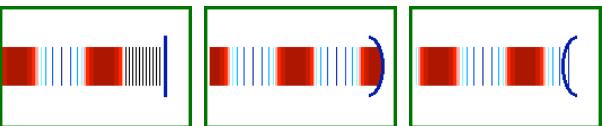
E.g. MPEG audio, Dolby AC.

How do we hear sound?



Sound Revisited

- Sound is produced by a vibrating source.
- The vibrations disturb air molecules
- Produce variations in air pressure: lower than average pressure, **rarefactions**, and higher than average, **compressions**. **This produces sound waves.**
- When a sound wave impinges on a surface (**e.g.** eardrum or microphone) it causes the surface to vibrate in sympathy:



- In this way **acoustic energy** is transferred from a source to a receptor.

Human Hearing

- Upon receiving the waveform the eardrum vibrates in sympathy
- Through a variety of mechanisms the acoustic energy is transferred to nerve impulses that the brain interprets as sound.

The ear can be regarded as being made up of 3 parts:

- The outer ear,
- The middle ear,
- The inner ear.



Back

Close

Human Ear

We consider:

- The function of the main parts of the ear
- How the transmission of sound is processed.

⇒ FLASH EAR DEMO (Lecture ONLY)

Click Here to Run Flash Ear Demo over the Web

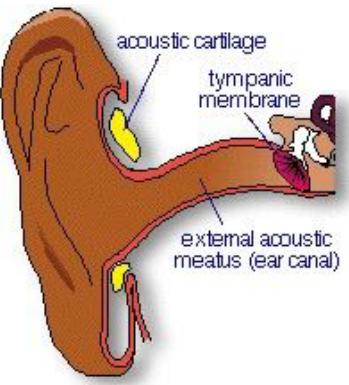
(Shockwave Required)



Back

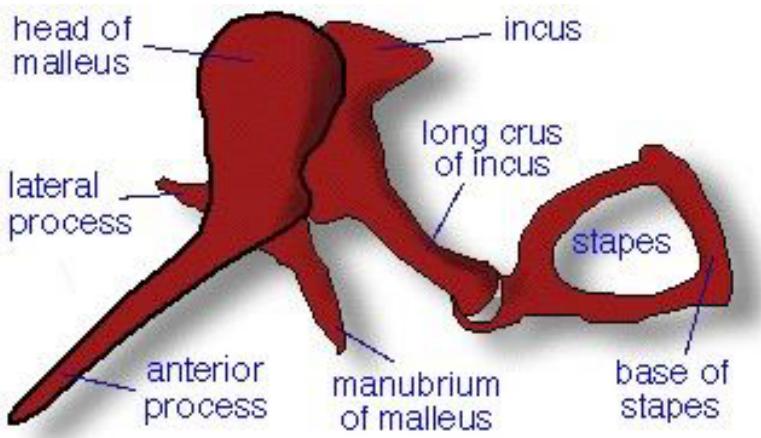
Close

The Outer Ear



- **Ear Canal:** Focuses the incoming audio.
- **Eardrum (Tympanic Membrane):**
 - Interface between the external and middle ear.
 - Sound is converted into mechanical vibrations via the middle ear.
 - Sympathetic vibrations on the membrane of the eardrum.

The Middle Ear



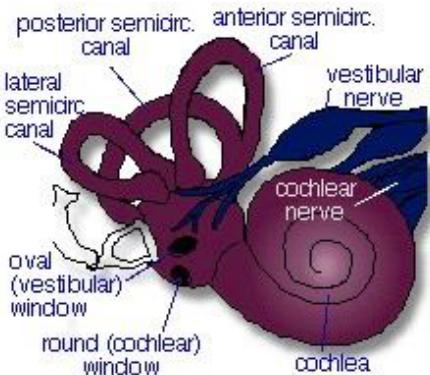
- 3 small bones, the **ossicles**:
Malleus, Incus, and Stapes.
- Form a system of levers which are linked together and driven by the eardrum
- Bones amplify the force of sound vibrations.



Back

Close

The Inner Ear

Multimedia
CM0340

544

The Cochlea:

- Transforms mechanical ossicle forces into hydraulic pressure,
- The cochlea is filled with fluid.
- Hydraulic pressure imparts movement to the cochlear duct and to the organ of Corti.
- Cochlea which is no bigger than the tip of a little finger!!

Semicircular canals

- Body's balance mechanism
- Thought that it plays no part in hearing.



Back

Close

How the Cochlea Works

- Pressure waves in the cochlea exert energy along a route that begins at the oval window and ends abruptly at the membrane-covered round window
- Pressure applied to the oval window is transmitted to all parts of the cochlea.

Stereocilia

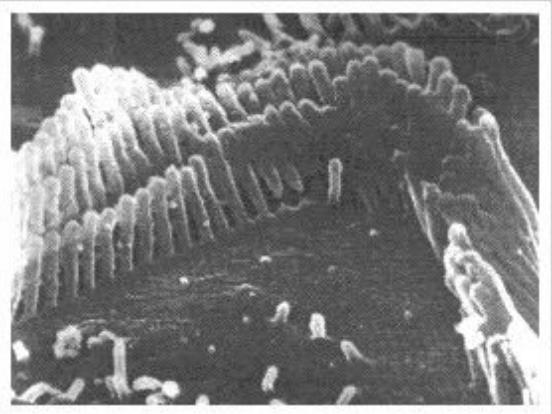
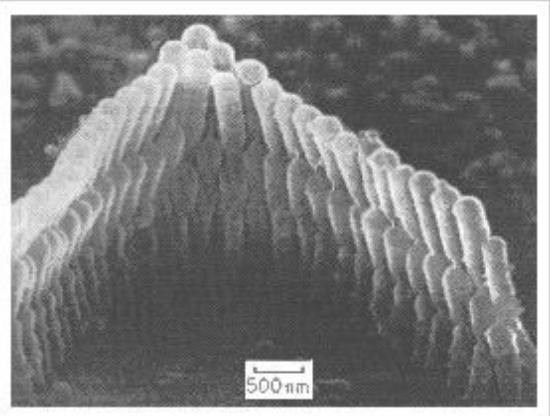
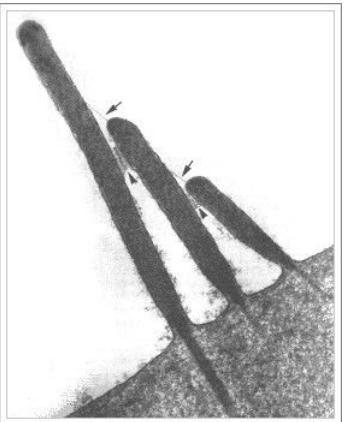
- Inner surface of the cochlea (**the basilar membrane**) is lined with over 20,000 hair-like nerve cells — **stereocilia**,
- One of the most critical aspects of hearing.



Back

Close

Stereocilia Microscope Images



◀◀

▶▶

◀

▶

Back

Close

Hearing different frequencies

- Basilar membrane is tight at one end, looser at the other
- High tones create their greatest crests where the membrane is tight,
- Low tones where the wall is slack.
- Causes resonant frequencies much like what happens in a tight string.
- Stereocilia differ in length by minuscule amounts
- they also have different degrees of resiliency to the fluid which passes over them.



Back

Close

Finally to nerve signals

- Compressional wave moves middle ear through to the cochlea
- Stereocilia will be set in motion.
- Each stereocilia sensitive to a particular frequency.
- Stereocilia cell will resonate with a larger amplitude of vibration.
- Increased vibrational amplitude induces the cell to release an electrical impulse which passes along the auditory nerve towards the brain.

In a process which is not clearly understood, the brain is capable of interpreting the qualities of the sound upon reception of these electric nerve impulses.



Back

Close

Sensitivity of the Ear

- Range is about 20 Hz to 20 kHz, most sensitive at 2 to 4 kHz.
- Dynamic range (quietest to loudest) is about 96 dB
- Approximate threshold of pain: 130 dB
- Hearing damage: > 90 dB (prolonged exposure)
- Normal conversation: 60-70 dB
- Typical classroom background noise: 20-30 dB
- Normal voice range is about 500 Hz to 2 kHz
 - Low frequencies are vowels and bass
 - High frequencies are consonants

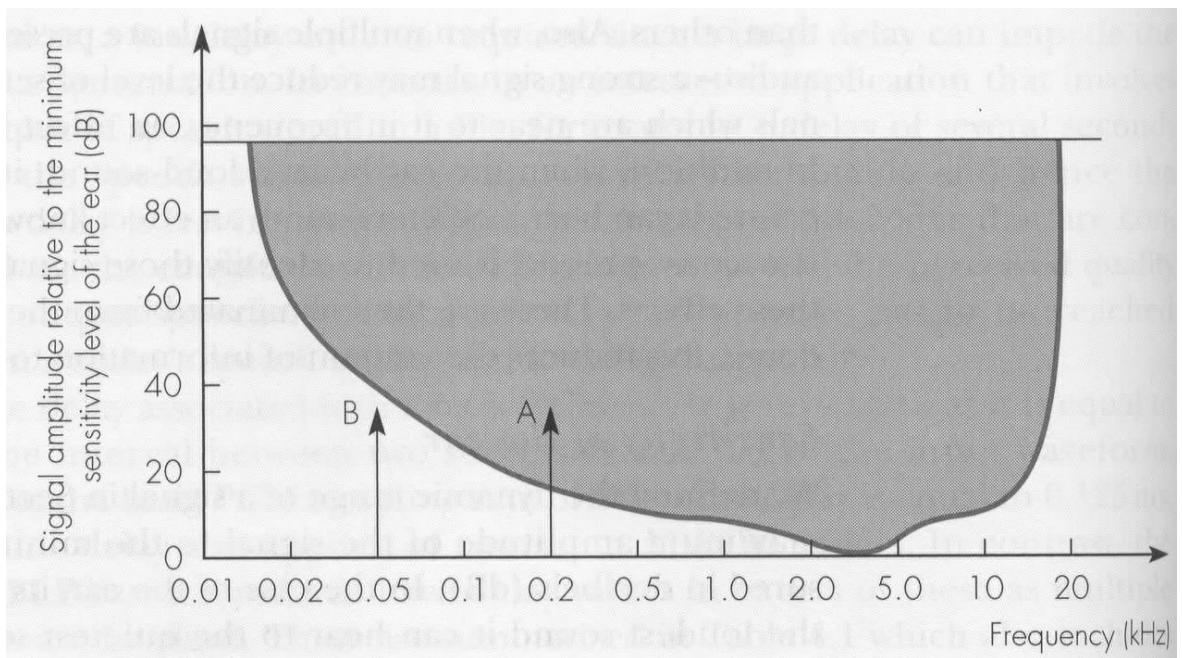


Back

Close

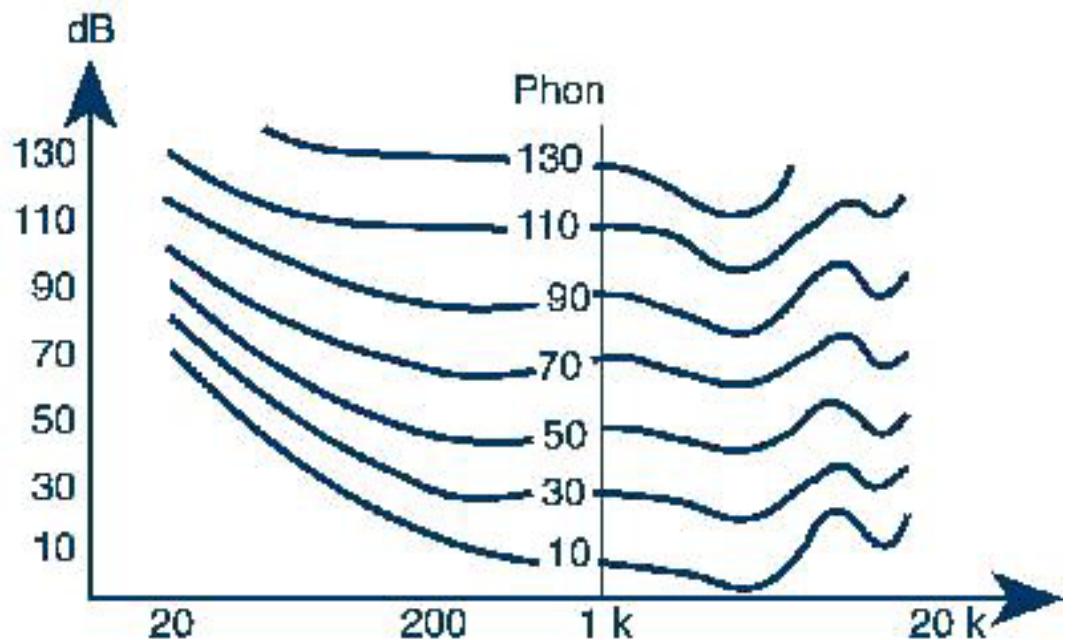
Question: How sensitive is human hearing?

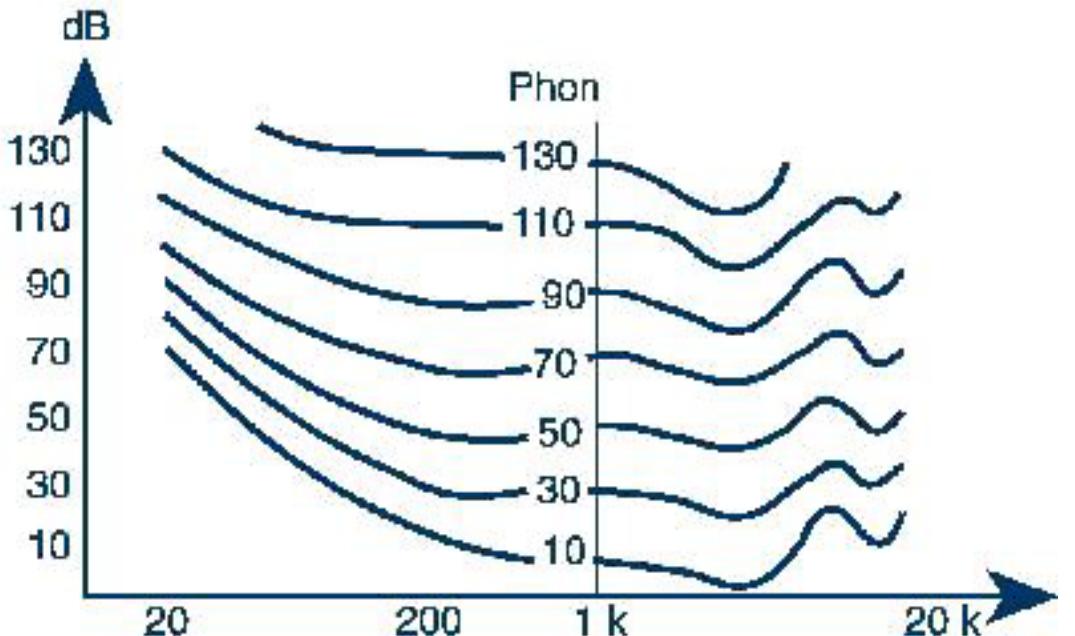
The sensitivity of the human ear with respect to frequency is given by the following graph.



Frequency dependence is also level dependent!

- Ear response is even more complicated.
- Complex phenomenon to explain.
- Illustration : Loudness Curves or Fletcher-Munson Curves:

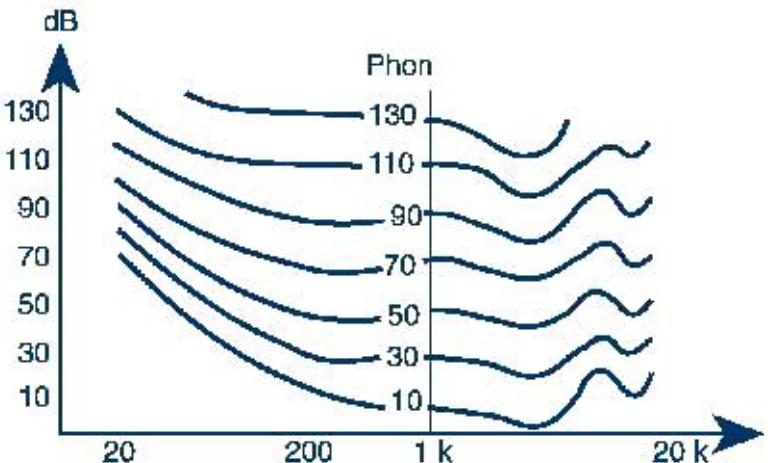




- Connecting pure tone stimuli producing the same perceived loudness ('Phons', in dB).

Perceptual Audio Demos⁴

⁴Also demos relevant for all this section here



- Curves indicate perceived loudness is a function of both the frequency and the level (sinusoidal sound signal)
- Equal loudness curves. Each contour:
 - Equal loudness
 - Express how much a sound level must be changed as the frequency varies, **to maintain a certain perceived loudness**



Back

Close

Physiological Implications

Why are the curves accentuated where they are?

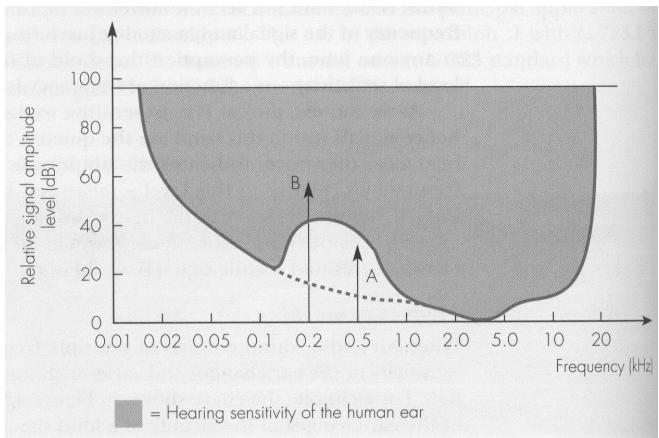
- Accentuates of frequency range to coincide with speech.
- Sounds like **p** and **t** have very important parts of their spectral energy within the accentuated range
- Makes them more easy to discriminate between.

The ability to hear sounds of the **accentuated** range (around a few kHz) is thus vital for speech communication.

Traits of Human Hearing

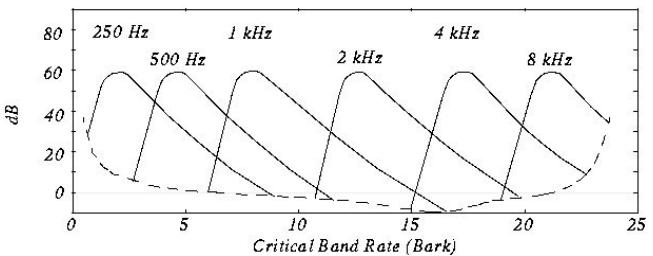
Frequency Masking

- Multiple frequency audio changes the sensitivity with the relative amplitude of the signals.
- If the frequencies are close and the amplitude of one is less than the other close frequency then the second frequency may not be heard.



Critical Bands

- Range of closeness for frequency masking depends on the frequencies and relative amplitudes.
- Each **band** where frequencies are masked is called **the Critical Band**
- Critical bandwidth for average human hearing varies with frequency:
 - Constant 100 Hz for frequencies less than 500 Hz
 - Increases (approximately) linearly by 100 Hz for each additional 500 Hz.
- Width of critical band is called a **bark**.



Critical Bands (cont.)

First 12 of 25 critical bands:

Band #	Lower Bound (Hz)	Center (Hz)	Upper Bound (Hz)	Bandwidth (Hz)
1	-	50	100	-
2	100	150	200	100
3	200	250	300	100
4	300	350	400	100
5	400	450	510	110
6	510	570	630	120
7	630	700	770	140
8	770	840	920	150
9	920	1000	1080	160
10	1080	1170	1270	190
11	1270	1370	1480	210
12	1480	1600	1720	240

What is the cause of Frequency Masking?

- The stereocilia are excited by air pressure variations, transmitted via outer and middle ear.
- Different stereocilia respond to different ranges of frequencies — the critical bands

Frequency Masking occurs because after excitation by one frequency further excitation by a less strong similar frequency of the same group of cells is not possible.

[Click here](#) to hear example of Frequency Masking.

See/Hear also: [Click here](#) (in the Masking section).



Back

Close

Temporal masking

After the ear hears a loud sound: **It takes a further short while before it can hear a quieter sound.**

Why is this so?

- **Stereocilia** vibrate with corresponding force of input sound stimuli.
- If the stimuli is strong then stereocilia will be in a high state of excitation and **get fatigued**.
- **Hearing Damage**: After extended listening to loud music or headphones this sometimes manifests itself with ringing in the ears and even temporary deafness.
- **Hearing Damage**: Prolonged exposure to noise permanently damages the **Stereocilia**.

Temporal masking Explained

Temporal Masking occurs because the hairs take time to settle after excitation to respond again.

- Any loud tone will cause the hearing receptors in the inner ear to become saturated and require time to recover.



Back

Close

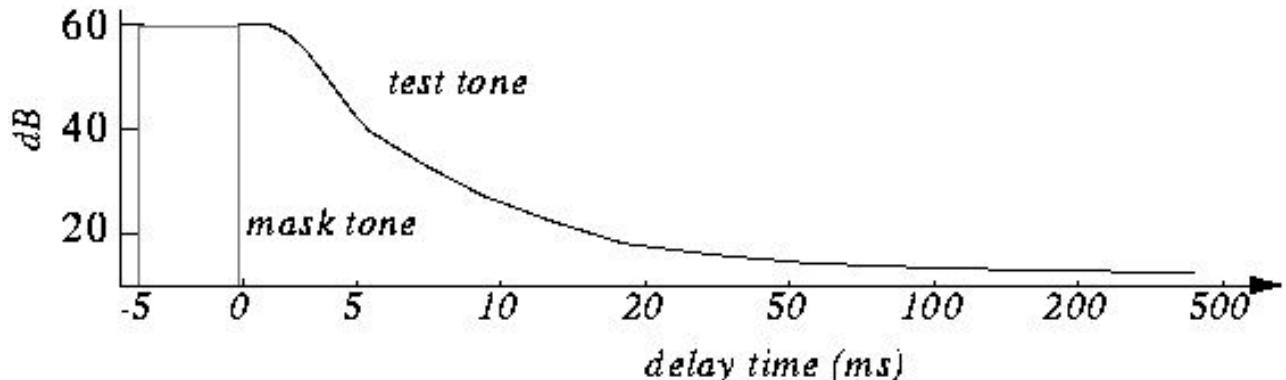
Example of Temporal Masking

- Play 1 kHz *masking tone* at 60 dB, plus a *test tone* at 1.1 kHz at 40 dB. Test tone can't be heard (it's masked).

Stop masking tone, then stop test tone after a short delay.

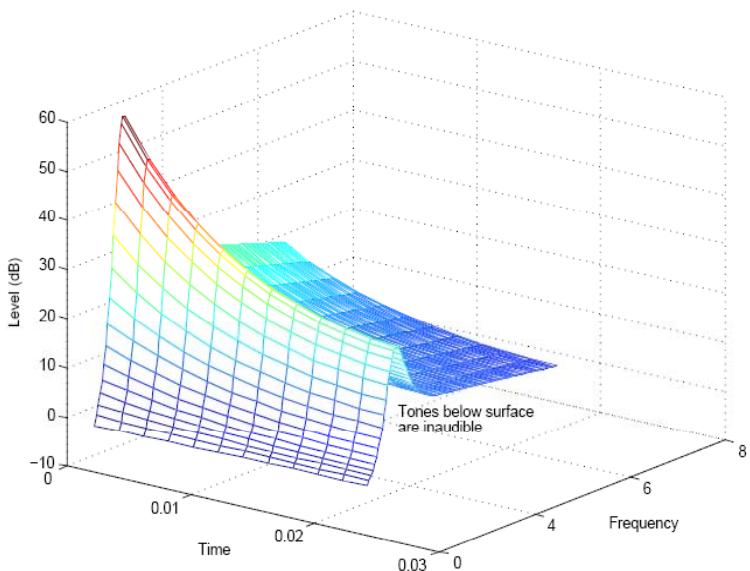
Adjust delay time to the shortest time that test tone can be heard (e.g., 5 ms).

Repeat with different level of the test tone and plot:



Example of Temporal Masking (Cont.)

- Try other frequencies for test tone (masking tone duration constant). Total effect of masking:



Compression Idea: How to Exploit?

- If we have a loud tone at, say at 1 kHz, then nearby quieter tones are masked.
- Best compared on **critical band scale** – range of masking is about 1 critical band
- Two factors for masking – **frequency masking** and **temporal masking**
- Question: How to use this for compression?

Two examples:

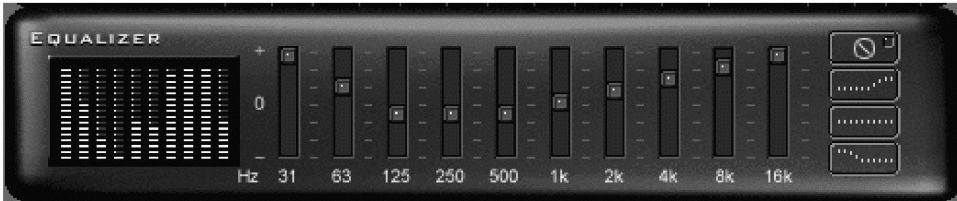
- **MPEG Audio**
- **Dolby**



How to compute?

We have met basic tools:

- Bank Filtering with **IIR/FIR Filters**
- **Fourier** and **Discrete Cosine Transforms**
 - Work in **frequency space**
- (Critical) **Band Pass Filtering** — Visualise a graphic equaliser



MPEG Audio Compression

- Exploits the psychoacoustic models above.
- **Frequency masking** is always utilised
- More complex forms of MPEG also employ **temporal masking**.

Basic Frequency Filtering Bandpass

MPEG audio compression basically works by:

- Dividing the audio signal up into a set of frequency subbands
- Use Filter Banks to achieve this.
- Subbands approximate **critical bands**.
- Each band quantised according to the **audibility of quantisation noise**.

Quantisation is the key to MPEG audio compression and is the reason why it is lossy.



Back

Close

How good is MPEG compression?

Although (data) lossy

MPEG claims to be **perceptually lossless**:

- Human tests (part of standard development), Expert listeners.
- 6-1 compression ratio, stereo 16 bit samples at 48 KHz compressed to 256 kbits/sec
- **Difficult**, real world examples used.
- **Under Optimal listening conditions** no statistically distinguishable difference between original and MPEG.



Back

Close

Basic MPEG: MPEG audio coders

- Set of standards for the use of video **with** sound.
- Compression methods or **coders** associated with audio compression are called **MPEG audio coders**.
- MPEG allows for a variety of different coders to employed.
- **Difference** in level of sophistication in applying perceptual compression.
- Different **layers** for levels of sophistication.



Back

Close

An Advantage of MPEG approach

Complex psychoacoustic modelling only in coding phase

- Desirable for real time (Hardware or software) decompression
- Essential for broadcast purposes.
- Decompression is independent of the psychoacoustic models used
- Different models can be used
- If there is enough bandwidth no models at all.



Back

Close

Basic MPEG: MPEG Standards

Evolving standards for MPEG audio compression:

- MPEG-1 is by the most prevalent
- So called [mp3](#) files we get off Internet are members of [MPEG-1 family](#).
- Standards now extends to MPEG-4 (structured audio) — [Earlier Lecture](#).

For now we concentrate on MPEG-1

Basic MPEG: MPEG Facts

- MPEG-1: 1.5 Mbits/sec for audio and video
About 1.2 Mbits/sec for video, 0.3 Mbits/sec for audio
(Uncompressed CD audio is
 $44,100 \text{ samples/sec} * 16 \text{ bits/sample} * 2 \text{ channels} > 1.4 \text{ Mbits/sec}$)
- Compression factor ranging from 2.7 to 24.
- MPEG audio supports sampling frequencies of 32, 44.1 and 48 KHz.
- Supports one or two audio channels in one of the four modes:
 1. Monophonic – single audio channel
 2. Dual-monophonic – two independent channels
(functionally identical to stereo)
 3. Stereo – for stereo channels that share bits, but not using joint-stereo coding
 4. Joint-stereo – takes advantage of the correlations between stereo channels

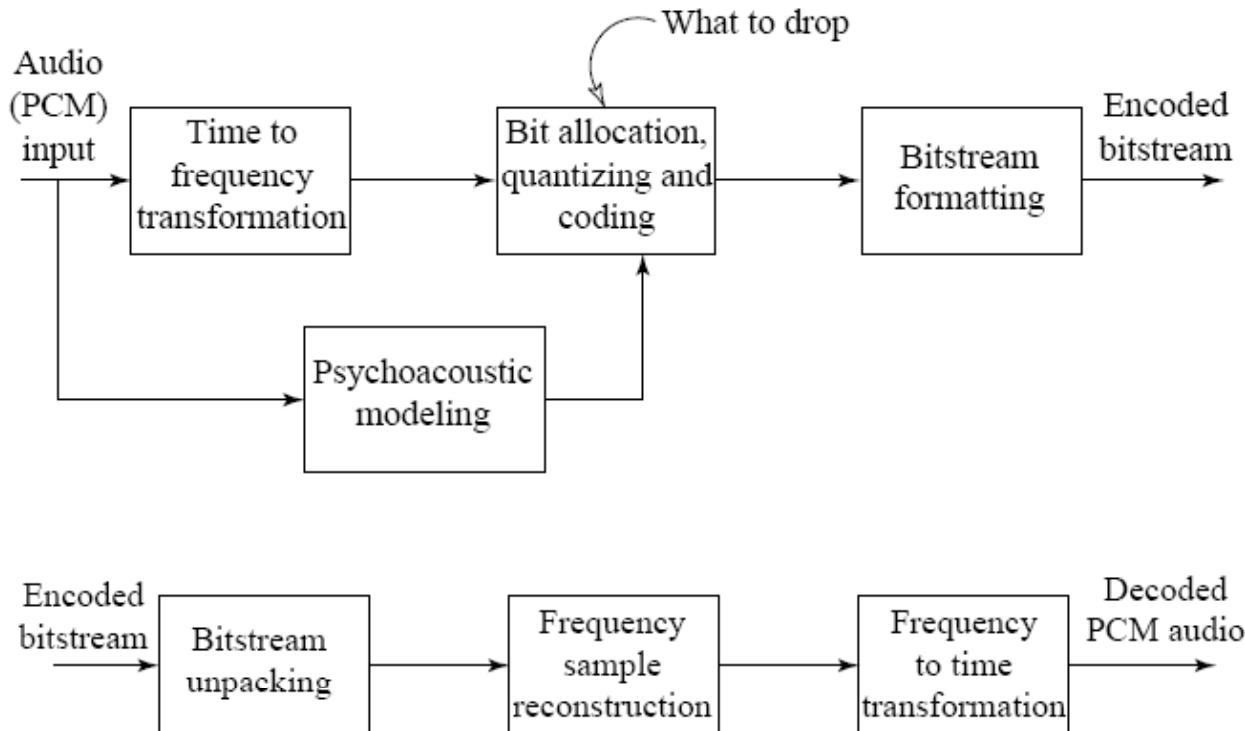


Back

Close

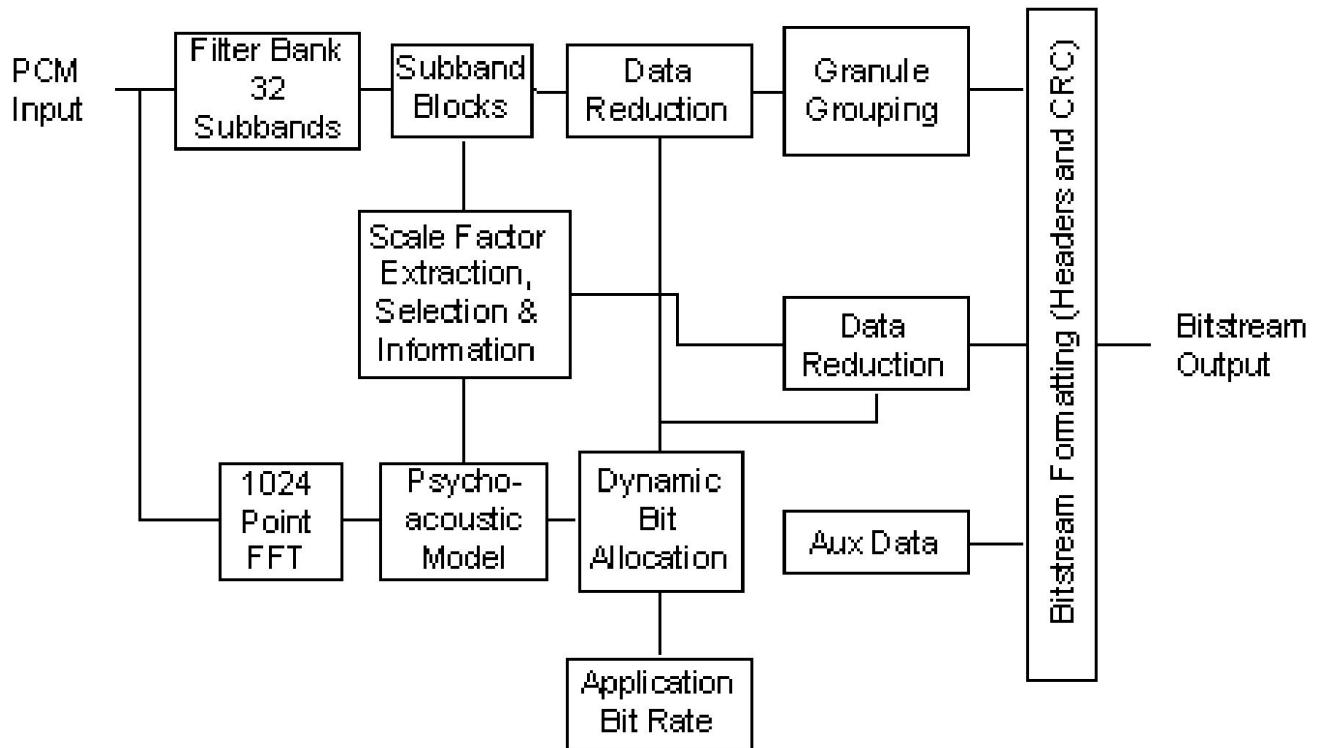
Basic MPEG-1 Encoding/Decoding algorithm

Basic MPEG-1 encoding/decoding maybe summarised as:



Basic MPEG-1 Compression algorithm (1)

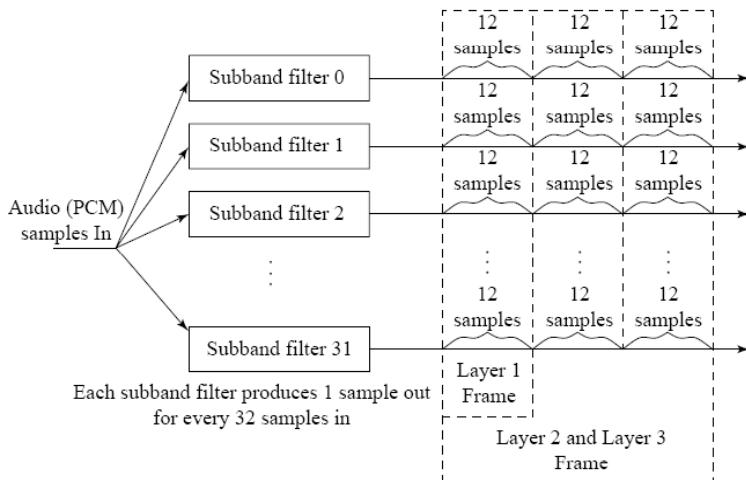
Basic encoding algorithm in more detail summarised below:



Basic MPEG-1 Compression algorithm (2)

The main stages of the algorithm are:

- The audio signal is first samples and quantised use PCM
 - Application dependent: Sample rate and number of bits
 - The PCM samples are then divided up into a number of **frequency subband** and compute **subband scaling factors**:



Basic MPEG-1 Compression algorithm (3)

Analysis filters

- Also called **critical-band filters**
- Break signal up into equal width subbands
- Use Filter Banks (modified with discrete cosine transform (DCT) Level 3)
- Filters divide audio signal into frequency subbands that approximate the 32 critical bands
- Each band is known as a *sub-band sample*.
- **Example:** 16 kHz signal frequency, Sampling rate 32 kbits/sec gives each subband a bandwidth of 500 Hz.
- Time duration of each sampled segment of input signal is time to accumulate 12 successive sets of 32 PCM (subband) samples, **i.e.** $32 \times 12 = 384$ samples.

Analysis filters (cont)

- In addition to filtering the input, analysis banks determine
 - Maximum amplitude of 12 subband samples in each subband.
 - Each known as the **scaling factor** of the subband.
 - Passed to *psychoacoustic model* and **quantiser blocks**



Back

Close

Basic MPEG-1 Compression algorithm (5)

Psychoacoustic modeller:

- Frequency Masking and may employ temporal masking.
- Performed concurrently with filtering and analysis operations.
- Uses Fourier Transform (FFT) to perform analysis.
- Determine amount of masking for each band caused by nearby bands.
- Input: set hearing thresholds and subband masking properties (model dependent) and scaling factors (above).



Back

Close

Basic MPEG-1 Compression algorithm (6)

Psychoacoustic modeller (cont):

- Output: a set of **signal-to-mask** ratios:
 - Indicate those frequencies components whose amplitude is below the audio threshold.
 - If the power in a band is below the masking threshold, don't encode it.
 - Otherwise, determine number of bits (from scaling factors) needed to represent the coefficient such that noise introduced by quantisation is below the masking effect (Recall that 1 bit of quantisation introduces about 6 dB of noise).

Basic MPEG-1 Compression algorithm (7)

Example of Quantisation:

- Assume that after analysis, the first levels of 16 of the 32 bands are:

Band	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
Level (db)	0	8	12	10	6	2	10	60	35	20	15	2	3	5	3	1

- If the level of the 8th band is 60 dB, then assume (according to model adopted) it gives a masking of 12 dB in the 7th band, 15 dB in the 9th.
Level in 7th band is 10 dB (< 12 dB), so ignore it.
Level in 9th band is 35 dB (> 15 dB), so send it.
→ Can encode with up to 2 bits (= 12 dB) of quantisation error.
- More on Bit Allocation soon.**



Back

Close

MPEG-1 Output bitstream

The basic output stream for a basic MPEG encoder is as follows:

Header	SBS Format	12x32 subband	Ancillary data
--------	------------	---------------	----------------

- **Header**: contains information such as the sample frequency and quantisation.,.
- **Subband sample (SBS) format**: Quantised scaling factors and 12 frequency components in each subband.
 - Peak amplitude level in each subband quantised using 6 bits (64 levels)
 - 12 frequency values quantised to 4 bits
- **Ancillary data**: Optional. Used, for example, to carry additional coded samples associated with special broadcast format (**e.g surround sound**)



Back

Close

Decoding the bitstream

- Dequantise the subband samples after demultiplexing the coded bitstream into subbands.
- **Synthesis bank** decodes the dequantised subband samples to produce PCM stream.
 - This essentially involves applying the inverse fourier transform (**IFFT**) on each substream and multiplexing the channels to give the PCM bit stream.



Back

Close

MPEG Layers

MPEG defines 3 levels of processing layers for audio:

- Level 1 is the basic mode,
- Levels 2 and 3 more advance (use temporal masking).
- Level 3 is the most common form for audio files on the Web
 - Our beloved MP3 files that record companies claim are bankrupting their industry.
 - Strictly speaking these files should be called **MPEG-1 level 3** files.

Each level:

- Increasing levels of sophistication
- Greater compression ratios.
- Greater computation expense



Back

Close

Level 1

- Best suited for bit rate bigger than 128 kbits/sec per channel.
- Example: Phillips Digital Compact Cassette uses Layer 1 192 kbits/sec compression
- Divides data into frames,
 - Each of them contains 384 samples,
 - 12 samples from each of the 32 filtered subbands as shown above.
- Psychoacoustic model only uses frequency masking.
- Optional Cyclic Redundancy Code (CRC) error checking.



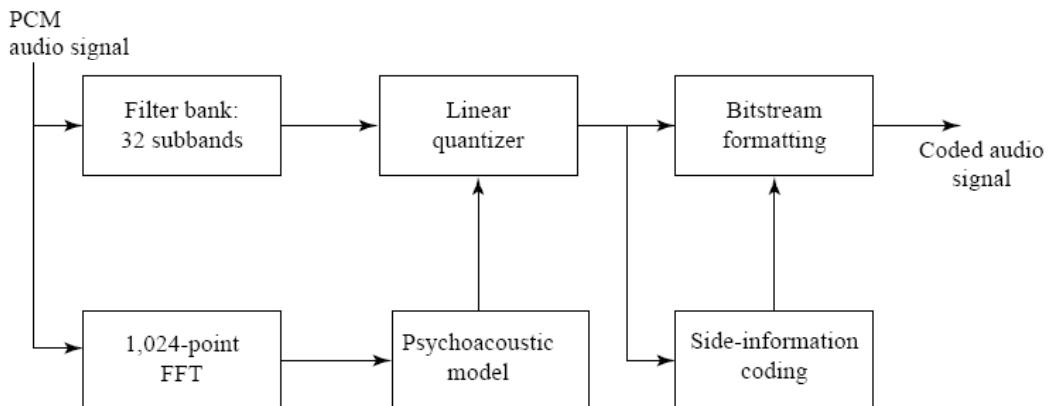
Back

Close

Level 1 (and Level 2) Audio Layers

Note: Mask Calculations done in parallel with subband filtering

- Accurate frequency decomposition via Fourier Transform.



Layer 2

- Targeted at bit rates of around 128 kbits/sec per channel.
- Examples: Coding of Digital Audio Broadcasting (DAB) on CD-ROM, CD-I and Video CD.
- Enhancement of level 1.
- Codes audio data in larger groups:
 - Use three frames in filter:
before, current, next, a total of 1152 samples.
 - This models a little bit of the temporal masking.
- Imposes some restrictions on bit allocation in middle and high subbands.
- More compact coding of scale factors and quantised samples.
- Better audio quality due to saving bits here so more bits can be used in quantised subband values

Layer 3

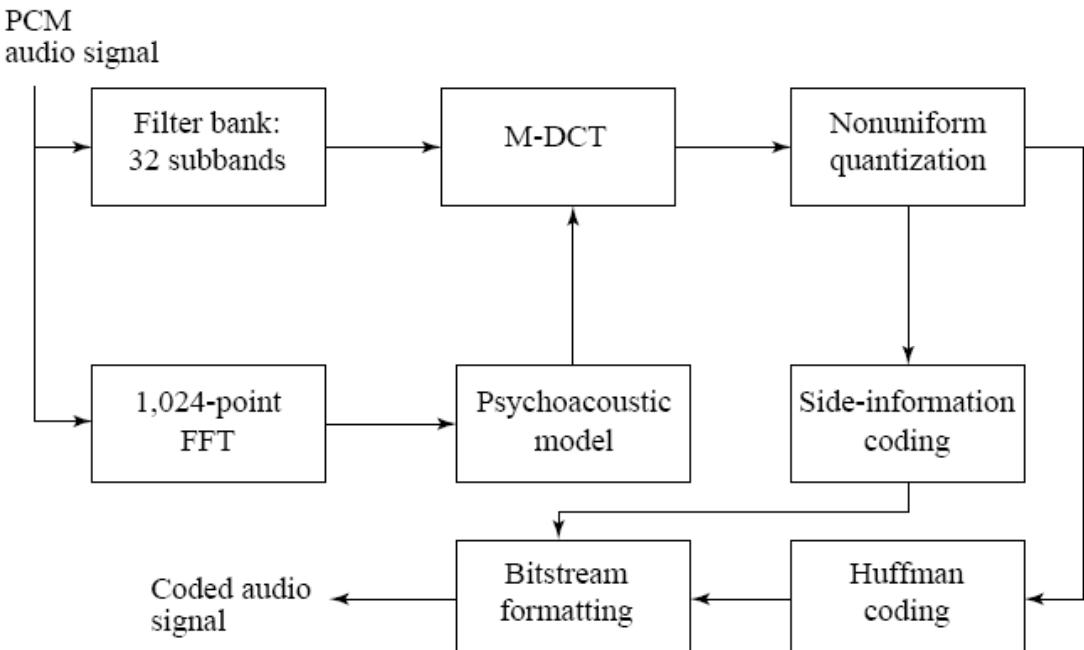
- Targeted at bit rates of 64 kbits/sec per channel.
- Example: audio transmission of ISDN or suitable bandwidth network.
- Much more complex approach.
- Psychoacoustic model includes temporal masking effects,
- Takes into account stereo redundancy.
- Better critical band filter is used (non-equal frequencies)
- Uses a modified DCT (MDCT) for lossless subband transformation.
- Two different block lengths: 18 (long) or 6 (short)
- 50% overlap between successive transform windows gives window sizes of 36 or 12 — **accounts for temporal masking**
- Greater frequency resolution accounts for poorer time resolution
- Uses Huffman coding on quantised samples for better compression.



Back

Close

Level 3 Audio Layers



Bit Allocation

- Process determines the number of code bits for each subband
- Based on information from the psychoacoustic model.



Back

Close

Bit Allocation For Layer 1 and 2

- Compute the mask-to-noise ratio (MNR) for all subbands:

$$MNR_{dB} = SNR_{dB} - SMR_{dB}$$

where

MNR_{dB} is the mask-to-noise ratio,

SNR_{dB} is the signal-to-noise ratio (SNR), and

SMR_{dB} is the signal-to-mask ratio from the psychoacoustic model.

- Standard MPEG lookup tables estimate SNR for given quantiser levels.
- Designers are free to try other methods SNR estimation.

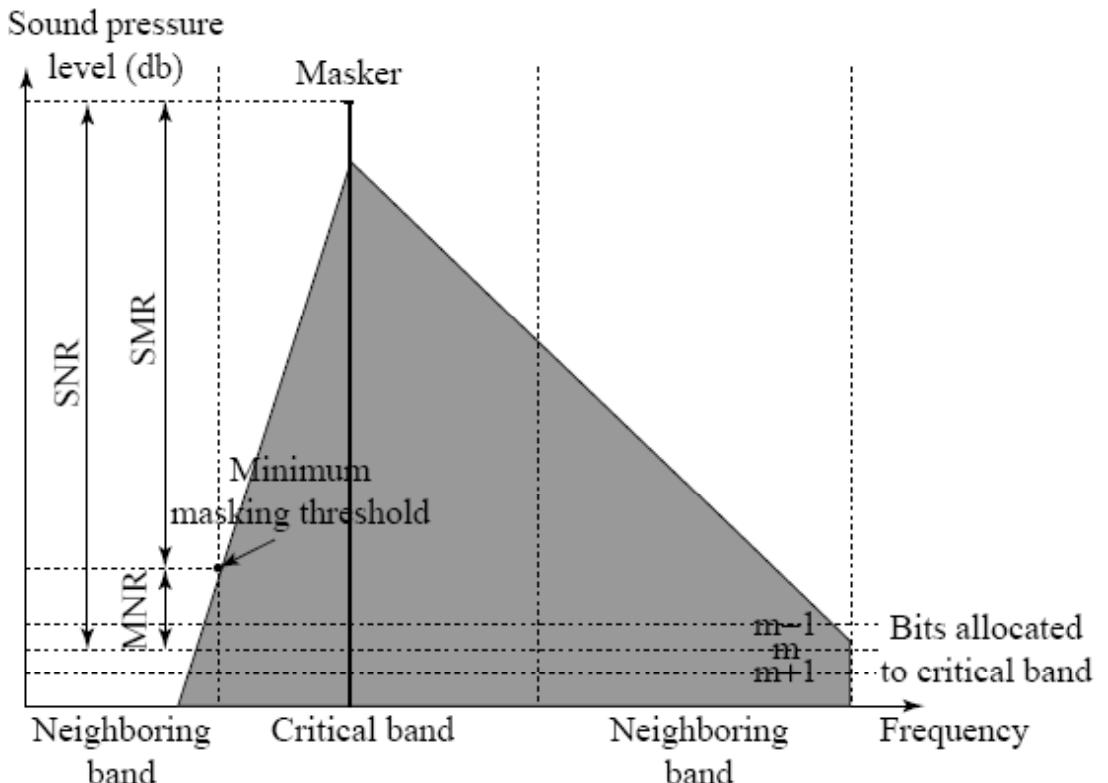


Bit Allocation For Layer 1 and 2 (cont.)

Once MNR computed for all the subbands:

- Search for the subband with the lowest MNR
- Increment code bits to that subband.
- When a subband gets allocated more code bits, the bit allocation unit:
 - Looks up the new estimate for SNR
 - Recomputes that subband's MNR.
- The process repeats until no more code bits can be allocated.

Bit Allocation For Layer 1 and 2 (cont.)



Bit Allocation For Layer 3

- Uses **noise allocation**, which employs Huffman coding.
- Iteratively varies the quantisers in an orderly way
 - Quantises the spectral values,
 - Counts the number of Huffman code bits required to code the audio data
 - Calculates the resulting noise in Huffman coding.
If there exist scale factor bands with more than the allowed distortion:
 - Encoder amplifies values in bands
 - **To effectively decreases** the quantiser step size for those bands.



Back

Close

Bit Allocation For Layer 3 (Cont.)

After this the process repeats. The process stops if any of these three conditions is true:

- None of the scale factor bands have more than the allowed distortion.
- The next iteration would cause the amplification for any of the bands to exceed the maximum allowed value.
- The next iteration would require all the scale factor bands to be amplified.

Real-time encoders include a time-limit exit condition for this process.



Back

Close

Stereo Redundancy Coding

Exploit redundancy in two couple stereo channels?

- Another perceptual property of the human auditory system
- Simply stated at low frequencies, the human auditory system can't detect where the sound is coming from.
 - So save bits and encode it mono.
- Used in MPEG-1 Layer 3.

Two types of stereo redundancy coding:

- Intensity stereo coding — all layers
- Middle/Side (MS) stereo coding — Layer 3 only stereo coding.

Intensity stereo coding

Encoding:

- Code some upper-frequency subband outputs:
 - A single summed signal instead of sending independent left and right channels codes
 - Codes for each of the 32 subband outputs.

Decoding:

- Reconstruct left and right channels
 - Based only on a single summed signal
 - 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 subband
- **But** the magnitude is different.

Middle/Side (MS) stereo coding

- Encodes the left and right channel signals in certain frequency ranges:
 - **Middle** — sum of left and right channels
 - **Side** — difference of left and right channels.
- Encoder uses specially tuned threshold values to compress the side channel signal further.

MATLAB MPEG Audio Coding Code

[MPEGAudio](#) (DIRECTORY)

[MPEGAudio.zip](#) (All Files Zipped)



Back

Close

Dolby Audio Compression

Application areas:

- FM radio Satellite transmission and broadcast TV audio (DOLBY AC-1)
- Common compression format in PC sound cards (DOLBY AC-2)
- High Definition TV standard **advanced television** (ATV) (DOLBY AC-3). *MPEG a competitor in this area.*

Differences with MPEG

- MPEG perceptual coders control quantisation accuracy of each subband by computing bit numbers for each sample.
- MPEG needs to store each quantise value with each sample.
- MPEG Decoder uses this information to dequantise:
forward adaptive bit allocation
- **Advantage of MPEG?**: no need for psychoacoustic modelling in the decoder due to store of every quantise value.
- **DOLBY**: Use **fixed bit rate allocation** for each subband.
 - No need to send with each frame — as in **MPEG**.
 - **DOLBY** encoders and decoder need this information.

Fixed Bit Rate Allocation

- Bit allocations are determined by known sensitivity characteristics of the ear.



Back

Close

Different Dolby standards

DOLBY AC-1 :

Low complexity psychoacoustic model

- 40 subbands at sampling rate of 32 kbits/sec or
- (Proportionally more) Subbands at 44.1 or 48 kbits/sec
- Typical compressed bit rate of 512 kbits per second for stereo.
- Example: FM radio Satellite transmission and broadcast TV audio



Back

Close

DOLBY AC-2 :

Variation to allow subband bit allocations to vary

- NOW Decoder needs copy of psychoacoustic model.
- Minimised encoder bit stream overheads at expense of transmitting encoded frequency coefficients of sampled waveform segment — known as the **encoded spectral envelope**.
- Mode of operation known as **backward adaptive bit allocation mode**
- High (hi-fi) quality audio at 256 kbits/sec.
- Not suited for broadcast applications:
 - encoder cannot change model without changing (remote/distributed) decoders
- Example: Common compression format in PC sound cards.

DOLBY AC-3 :

Development of AC-2 to overcome broadcast challenge

- Use **hybrid backward/forward adaptive bit allocation mode**
- Any model modification information is encoded in a frame.
- Sample rates of 32, 44.1, 48 kbits/sec supported depending on bandwidth of source signal.
- Each encoded block contains 512 subband samples, with 50% (256) overlap between successive samples.
- For a 32 kbits/sec sample rate each block of samples is of 8 ms duration, the duration of each encoder is 16 ms.
- Audio bandwidth (at 32 kbits/sec) is 15 KHz so each subband has 62.5 Hz bandwidth.
- Typical stereo bit rate is 192 kbits/sec.
- Example: High Definition TV standard **advanced television** (ATV). MPEG competitor in this area.

Further Reading

- A tutorial on MPEG audio compression
- AC-3: flexible perceptual coding for audio trans. & storage



Back

Close

Summary

What we have learned in this module?

- Different multimedia data and their representations
 - Audio
 - Image/Graphics
 - Video
- Colour model and perception
 - Different colour models (benefits and applications): RGB, CMYK, YUV, YIQ, L*a*b*...
 - Chroma subsampling (why and how)

Multimedia
CM0340

604



Back

Close

Fundamental Tools

- Nyquist theorem
- Fourier Transform and Discrete Cosine Transform:
Frequency Analysis
- Information theory and basic compression techniques:
 - Run-length encoding
 - Entropy encoding (Shannn-Fano, Huffman, arithmetic)
 - LZW
 - Predictive/Differential encoding
 - Frequency Domain Compression
 - Vector Quantisation



Back

Close

Audio

- Processing: Digital Audio Effects [CM0268]
- Synthesis: Additive, Subtractive, FM, Wavetable, Sample-based, Granular, Physical
- Compression:
 - Traits of human hearing system
 - MPEG Audio, Dolby AC
- Representation: MIDI, MPEG-4 Structured Audio



Back

Close

Image

- Representations: monochrome (dithering), index (LUT), greyscale, true-colour
- Chroma Subsampling (also used in Video)
- Compression:
 - What to exploit?
 - JPEG: true colour
 - GIF: index

- Compression: What to exploit?
 - Static intra-frames compression — JPEG like
 - Motion Compensation based Prediction (H.261)
 - Bidirectional (B) frames (MPEG-1/2)
 - Object-based coding, synthetic objects (MPEG-4)
 - Various technical improvements:
 - * flexible macroblock partitioning,
 - * subpixel motion estimation,
 - * multiple references etc. (MPEG-4 AVC)

Beyond the Material

- Problem-solving skills,
 - e.g. the development of various methods for video compression
- Identify the challenge: what is the real problem?
- Learn the nature: what to exploit?
- Understand the practice: e.g. encoder / decoder balance



Back

Close

The End



Back

Close