

# today: multimedia, streaming and the internet

course: Web Technology

Maarten Lamers

[www.maartenlamers.com/WT2006/](http://www.maartenlamers.com/WT2006/)

Leiden University

## **today**

about how computers process audio and video, how they are compressed, and how they are transferred

digital audio

audio compression

streaming audio, internet radio

video

video compression

before we start, refresh some high school stuff: bits, powers, and value ranges

## powers and combinations

suppose you have a 3-dial padlock,  
with 10 digits per dial (0 – 9)

Q: how many possible combinations  
are possible?

A: 1000 possible combinations (000 – 999)

$10 \times 10 \times 10 = 10^3$  (10 *to the power* 3) = 1000  
a lock with 3 dials of 10 numbers each has  $10^3$  possible  
combinations!

a lock with 4 dials of 10 numbers each has  $10^4$  possible  
combinations



## bits, powers, and value ranges

a bit can have 2 possible values: 0 and 1  
all possible combinations with 3 bits are:  
000, 001, 010, 011, 100, 101, 110, 111

a number built from 3 bits can express  $2^3$  possible values  
a number built from 8 bits can express  $2^8$  possible values  
a number built from 16 bits can express  $2^{16}$  possible values

$2^1 = 2$   
 $2^2 = 4$   
 $2^3 = 8$   
 $2^4 = 16$

$2^8 = 256$   
 $2^{10} = 1024$   
 $2^{16} = 65536$

a value expressed by 8 bits is also called a 'byte'

## bits, powers, and value ranges

Q: so, if you want to store a number with possible values 0 .. 200, how many bits do you need?

A: 8 bits, because  $2^8 = 256$ , so values 0 .. 255 are possible

to store a number with possible values 0 .. 10  
you need 4 bits, because  $2^3 = 8$  and  $2^4 = 16$

to store a number with possible values 0 .. 20000  
you need 15 bits, because  $2^{14} = 16384$  and  $2^{15} = 32768$

to store a number with possible values 0 .. 99  
you need 7 bits, because  $2^6 = 64$  and  $2^7 = 128$

## multimedia and streaming

multimedia: literally, two or more media

today we mean:

- continuous (real-time) media
- not necessarily more than one type

better term: *streaming media*

## audio and the human ear

wikipedia:

"Sound is the vibration of matter, capable of being perceived by the sense of hearing. ... When the vibrations reach our ears, they are converted into nerve impulses that are sent to our brains, allowing us to perceive the sound."

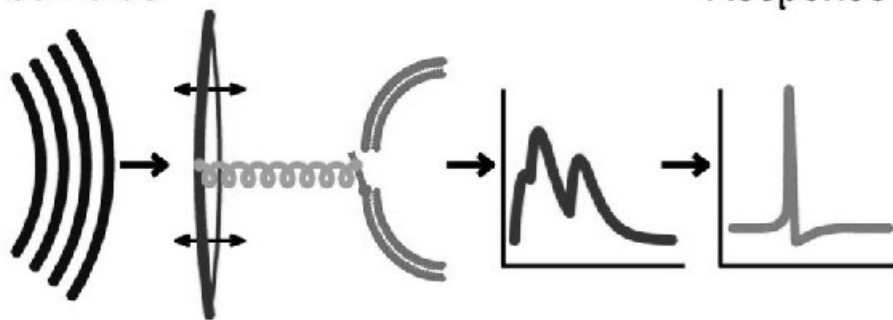
how fast the vibrations oscillate is what we call their *frequency*  
this value is expressed in *Hertz* (Hz)  
think of Hz (Hertz) as *waves per second* or *times per second*

the frequency range of the human ear runs from 20 Hz to 20,000 Hz, other frequencies we cannot hear

## audio and the human ear

Stimulus

Response



waves hit the eardrum; the cochlea transports them to auditory receptor cells; they convert certain frequencies to nerve impulses; these are perceived by your brain as sound

(source: wikipedia.org)

## **audio and the human ear**

the ear can sense sound variations lasting only a few milliseconds

the eye does not notice changes in light level that last only a few milliseconds

abrupt variation of a signal of only a few milliseconds affects perceived sound quality more than it affects perceived image quality

if this variation is unwanted: *jitter*

## **digital conversion of audio waves**

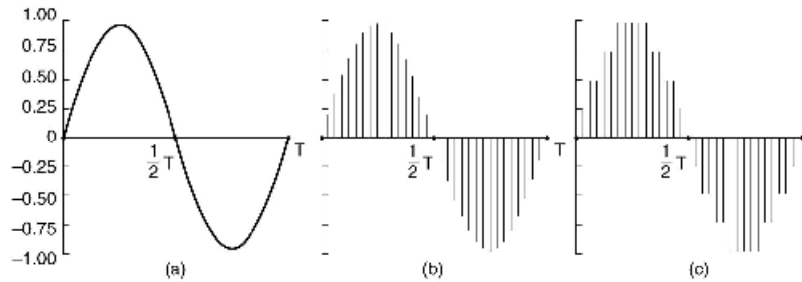
audio waves can be represented by taking measurements of the signal's amplitude ('height') at regular intervals.

this process is called *sampling*: the repetitive measuring of something in small time steps

the rate at which you take the measurements is called the *sampling rate* or *sampling frequency*

when you convert analog measured values to discrete (digital) measurement levels, this process is called *quantization*

## sampling and quantization example



- (a) a wave over time period  $T$   
(b) *sampling* the wave every  $\Delta T$  seconds  
(c) *quantizing* each sample to 4 bits ( $2^4 = 16$  possible values)

## quantization

digital representations of samples are never exact!  
the samples in figure (c) allow only 9 values, from -1.00 to 1.00 in steps of 0.25

e.g. 8-bit quantization allows for 256 ( $2^8$ ) distinct values

*quantization noise*: error introduced by expressing samples using a finite number of values

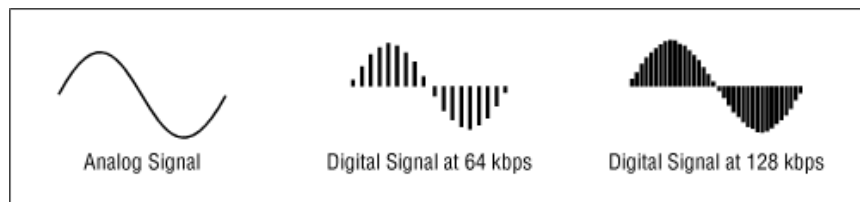
if quantization noise is too large, the ear detects it

## sampling and quantization

obviously, how good the digital result represents the original signal depends on:

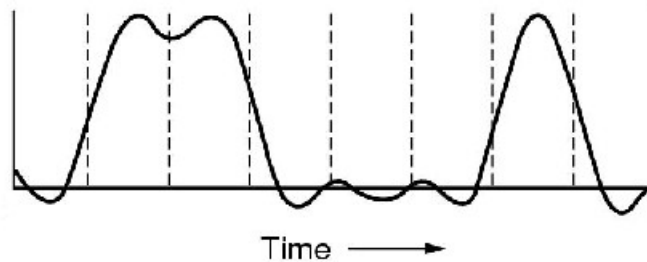
- the sampling rate
- the number of bits used to express the samples

together they determine the number of bits per second that you use to digitally represent a signal



## different frequencies

usually, a sound wave is not a pure sine wave but a linear sum of sine waves with different frequencies (the *frequency components*)



## **Nyquist theorem**

*Nyquist theorem* (1924):

if the highest frequency component present in a signal is of frequency  $f$ , then it is sufficient to take samples at frequency  $2f$

for example:

if a telephone line transmits audio signals up to 3000 Hz (3 KHz), then its signal can sufficiently be sampled at a rate of 6000 Hz (6 KHz). All the detail in the signal can then be captured by the sampling process.

Faster sampling is not needed, according to Nyquist!

## **example: the telephone system**

sampling frequency

8 KHz (125 microseconds per sample)

Nyquist says: as a result, frequencies above 4 KHz are lost  
(human speech is typically 600 Hz – 6000 Hz)

quantization

8-bit samples in Europe

7-bit samples in USA and Japan (7 bits data, 1 bit control)

data rates, or *bandwidth*, in bits per second (bps):

64.000 bps in Europe

56.000 bps in USA/Japan

### **example: audio compact discs**

sampling rate: 44100 Hz

quantization: 16-bit ( $2^{16} = 65536$  possible values per sample)

Nyquist says: frequencies above 22050 Hz are lost

using only 16 bits per sample also introduces some quantization noise, but it is not easy for humans to hear this

bandwidth:

705.6 Kbps (16 \* 44100, mono) or 1.4 Mbps (stereo)

transmitting uncompressed CD quality stereo sound in real time requires very fast communication!

*bandwidth* is the number of *bits per second* (bps) required or available for transmission!

### **audio compression**

1.4 Mbps for transmission of audio is a lot of bandwidth!  
compression of the signal is needed to make transmission over the internet practical

various audio compression algorithms were developed  
a compression/decompression algorithm is also called a *codec*  
also called (*en-*)*coding* and *decoding*

MP3 is a very powerful and well known one

there are two main types of compression:

- *waveform coding*
- *perceptual coding*

## waveform coding

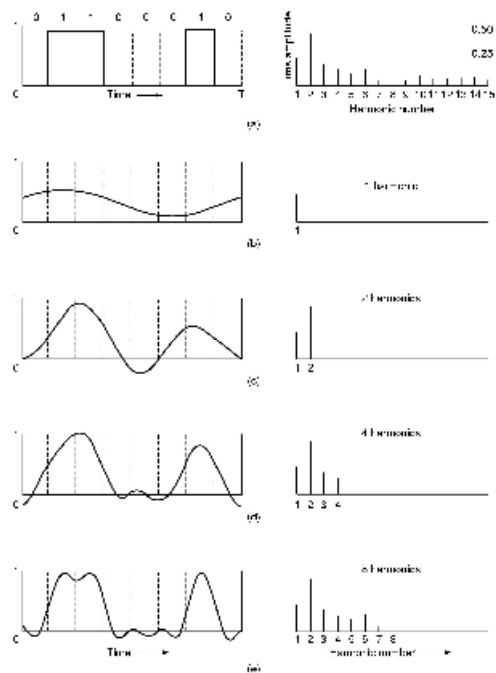
goal: reproduce the waveform after decoding as accurately as possible, using as few bits as possible in the coded data

a popular method to do this:

1. transform signal mathematically into its frequency components (sine-waves at different frequencies) by a 'Fourier transform'
2. encode the amplitude of each component in a minimal way

## waveform coding

example signal over time (left) and its Fourier amplitudes or powers (right)



## perceptual coding

goal: encode a signal in such a way that it *sounds* the same to the human listener (even though the waveform looks quite different on an oscilloscope)

based on science of psychoacoustics (how people perceive sound)  
exploits flaws in the human auditory system

'MP3' audio compression is a type of perceptual coding

the key idea of perceptual coding in audio is that some sounds can be *masked* by other sounds!



## (1) temporal masking

after a loud sound stops, a quiet sound will be inaudible for a short period of time by humans

because the ear turns down its sensitivity (*gain*) when the loud sound starts and it takes some time to turn it up again

after a powerful sound stops in some frequency band, we can omit some other frequencies from encoding for some time interval, based on our knowledge of its temporal masking properties

## (2) frequency masking

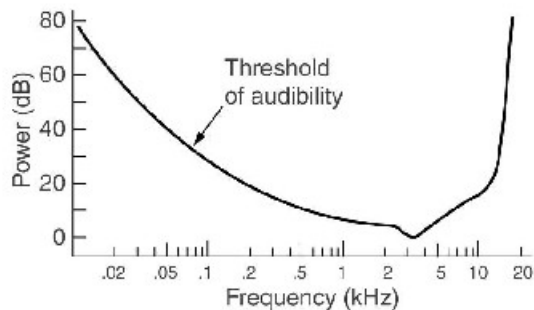
the ability of a loud sound in one frequency band to hide a simultaneous softer sound in another frequency band, that would otherwise be audible

a.k.a. the *masking effect*

MP3 encoding uses frequency masking:

it does not encode frequency band used by quiet sound when it is masked by another loud sound

## threshold of human audibility

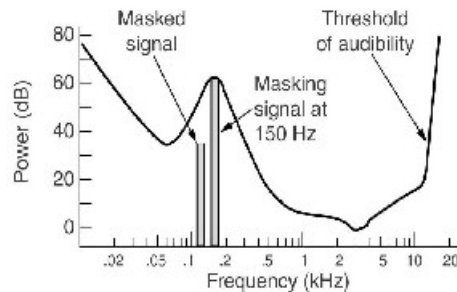


(a) threshold of human audibility as a function of frequency

it is unnecessary to encode any frequency whose power falls below the threshold of audibility

for example, a signal with power of 20 dB at 100 Hz can be omitted from the encoded signal with no perceptible loss of quality

## frequency masking: the masking effect



(b) in this example, a loud signal at 150 Hz raises the threshold of audibility for frequencies near it.

As a result, the 125 Hz signal is now under the threshold of audibility.

the 125 Hz signal can be completely omitted from the encoded signal without loss of perceptible quality

idea: by keeping track of which signals are being masked by more powerful 'nearby' signals, we can omit their frequencies from the encoded signal, saving bits!

## MPEG audio layers

MPEG audio: Motion Picture Experts Group standard  
it has three audio encoding variants, called *layers*  
each variant has additional optimizations

layer 1:  
basic scheme (used in DCC digital tapes)

layer 2:  
advanced bit allocation

layer 3, better known as *MP3*:  
add hybrid filters, non-uniform quantization, Huffman encoding, etcetera

Maarten says: MP3 is quite complex

## **MP3 encoding**

the general idea:

1. transform audio signal to its frequency components
2. encode only the unmasked frequencies (the ones we can actually hear), using as few bits as possible

typical compression ratio for MP3 is 1:12

sampling is done at 32 KHz, 44.1 KHz or 48 KHz

## **MP3 encoding, overview**

1. choose an output bit-rate (bandwidth) according to acceptable loss in quality and signal to noise ratio – lower ratio means higher bit-rate;  
e.g. 96kbps for stereo rock, 128kbps for a piano concerto
2. process samples in groups of 1152 (~ 26 msecs)
3. transform each group into its main 32 frequency components
4. determine masked freqs (using some psychoacoustic model)
5. further split the 32 freq bands into finer freq bands

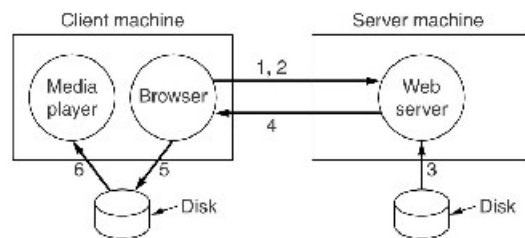
(continued)

## MP3 encoding, overview

6. divide the available bits (see step 1) among the freq bands
  1. more bits go to bands with the most unmasked power
  2. fewer bits go to bands with less unmasked power
  3. no bits are spent on any of the masked bands
7. encode the resulting frequency band powers using Huffman encoding (assigns short codes to common numbers, and longer codes to uncommon numbers; like Morse code does: E = \".", T = \"-", Q = \"-.-", Z = \"-...")
8. various other techniques are applied such as noise reduction, anti-aliasing, exploiting inter-channel redundancy (when applicable)

## music on the web: the simplest way

the simplest way to implement clickable music on the web



1. Establish TCP connection
2. Send HTTP GET request
3. Server gets file from disk
4. File sent back
5. Browser writes file to disk
6. Media player fetches file block by block and plays it

media players such as RealOne Player, WinAmp Player, ...

*major drawback:* the entire file must be transmitted before music starts to play!

for typical 4 Mb MP3 music file at 56 kbps, this is 10 minutes!

## music on the web: using a meta file

method to avoid this problem:

the file linked to the song title is not the actual music file, but a *metafile*: a very short file just naming the music. For example:

```
rtsp://joes-audio-server/song-0025.mp3
```

browser downloads metafile, starts the associated helper application, and hands it the scratch file (just as it would with an audio file!)

helper-app contacts `joes-audio-server` and asks for the song  
browser is not "in the loop" anymore

often, the music server is not the web server

most often, it is not even an HTTP server, but a specialized media server using for example RTSP (Real-Time Streaming Protocol) instead

## streaming

transfer of data (usually 'multimedia') so that it can be played as a steady and continuous stream

client can start rendering (displaying, playing) the data before the entire file has been transmitted

uses buffering to store data that is received more quickly than required

buffering is needed also to compensate for possible later delays in transmission

use compression on the data before transmitting it

- compress on server before transmitting
- decompress after receiving on client

## **streaming: UDP instead of TCP**

use UDP (User Datagram Protocol) instead of TCP

- for streaming, timeliness is more important than reliability!
- lower overhead than TCP
- allows for multicasting

UDP is also a standard Transport Layer protocol, running on top of IP, just like TCP

it does not provide reliability; TCP does

as a result, it is faster (less processing, no acknowledgments)

real-time music transmissions rarely use TCP

with TCP an error and retransmission of packets might introduce an unacceptable long gap in the music (*jitter*)

## **streaming and reliability**

the actual transmission of the stream is done with a protocol such as *Real-time Transport Protocol* (RTP)

RTP is layered on top of UDP and unreliable (packets may be lost)

it is up to the player to deal with this!

## interleaving

technique to make error handling easier in streaming audio

suppose packets contain 220 audio samples

each packet represents 5 msecs of music

then a lost packet means a 5 msecs gap in the music

this is undesirable for the listener...

*interleaving:*

send all *odd* samples for a 10 msec interval in one packet

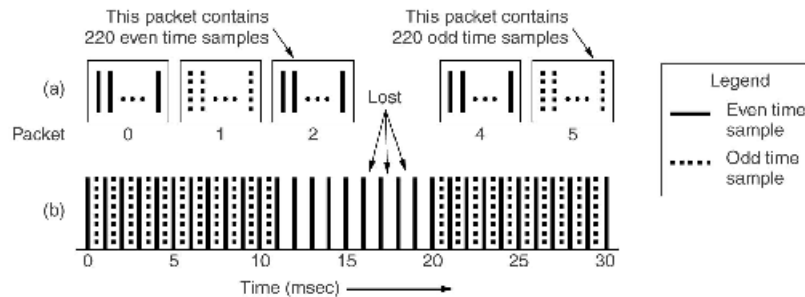
send *even* samples for same 10 msec interval in next packet

then a missing packet means loss of *every other* sample for 10 msecs

msecs

the missing values can be interpolated ('guessed') by the player

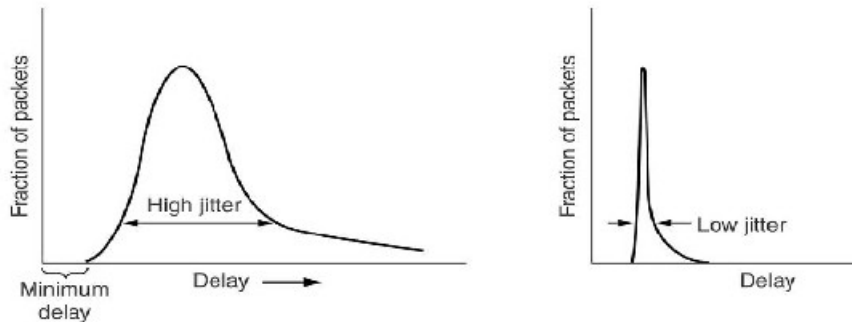
## interleaving



when packets carry alternate samples, a lost packet reduces the temporal resolution rather than creating a gap in time

## jitter

for audio/video streaming, the amount of time required for packet delivery is not very important  
however, variation in time between packets arriving, is!  
it leads to *jitter* in the streamed signal  
caused e.g. by network congestion, or routing issues  
high jitter gives uneven quality and is a nasty thing



## buffering

all streaming A/V systems start with buffering 10-15 secs

*pull server* : as long as there is room in the buffer, the player keeps sending requests for additional data to the server  
keeps the buffer as full as possible  
disadvantage: many unnecessary data requests

*push server* : player sends one request and lets the server keep pushing data

## push server

two possibilities:

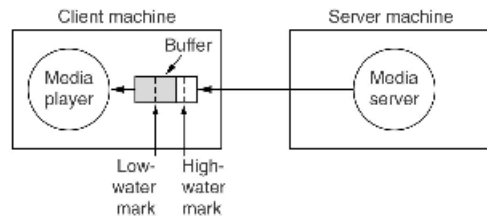
1. server runs at normal playback speed:  
simple scheme, because no messages are required either way between the server and media player
2. server pumps out data faster than is needed by player
  - advantage: server can catch up if it gets behind later
  - problem: potential for buffer overruns (buffer getting too full)
  - solution: define a *low-water mark* and *high-water mark* in the buffer

in both cases, some data is buffered before playback starts

## using water-marks

media player plays from a buffer, not directly from network  
server pumps out data until high-water mark in buffer is reached

then the media player tells server to pause while the server pauses, the buffer is emptied by playback  
when low-water mark is reached, the player tells server to start transmitting again etcetera ...



RTSP (Real-Time Streaming Protocol) can be used to control a push-server

the data stream itself is usually transmitted using RTP (Real-time Transport Protocol)

## **live internet radio**

*live* means: it is always broadcast at exactly the rate it is generated and played back

when *not live*, a stored audio file can be streamed out by the server at greater rate

live radio stations usually have many, many simultaneous listeners, whereas streaming audio is usually point-to-point

for this reason, live internet radio *should use* multicasting with both the RTP and RTSP protocols,

however...

## **live internet radio, in practice**

in practice data feed is sent over TCP point-2-point connection!  
this introduces problems: lost packets timing out,  
retransmissions, redundancy, etcetera  
resulting in jitter!

reasons for using TCP unicasting instead of RTP multicasting  
few ISPs support multicasting  
RTP is less well known than TCP  
many people listen to internet radio at work from behind a  
firewall that blocks all non-HTTP transmissions



video

## **video**

the human eye does *not* notice changes in light levels lasting only a few milliseconds, as we do with audio  
a flashed image is retained by the retina for some milliseconds before it starts decaying  
as a result vision sampling can be done less rapidly than audio sampling

flashing a sequence of images at 50 Hz (images /sec) or more is perceived by the human eye as smooth motion  
even at a rate of 25 images/sec, people perceive smooth motion, although flickering may be perceived

each screen-image in video is called a *frame*

### **analog systems: TV**

to obtain a frame, the scene is scanned by an electron beam in lines, transforming it to an analog voltage-over-time signal  
after the frame is scanned, the beam retraces to start scanning the next frame

the voltage (intensity) over time is broadcast to TV sets  
your TV set rebuilds the image on the screen in the same way, also scanning with an electron beam

NTSC (Americas and Japan): 525 scan lines, 483 displayed, 30 frames/sec

PAL/SECAM (Europe): 625 scan lines, 576 displayed, 25 frames/sec

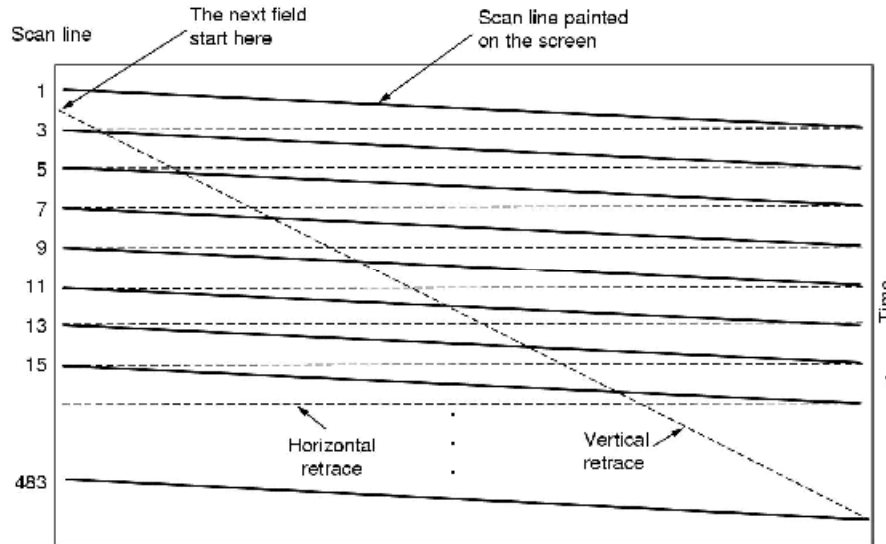
### **analog systems: TV**

in practice 25 frames/sec may be perceived as flickering  
a solution to this is *interlacing*: odd and even scan lines of one frame are alternated in two *fields*  
since this results in 50 frames/sec, no flickering  
non-interlaced television or video is called *progressive*

color TV uses three beams (red, green and blue) instead on single intensity (B&W) electron beam

TeleText uses the vertical retrace time to broadcast its data in  
HDTV (High Definition TV) doubles scan lines and uses 16:9 size, instead of standard 4:3

## interlacing scanning pattern (NTSC)



## digital video

sequence of frames, each consisting of a grid (matrix) of picture elements or *pixels*

pixels are typically 24 bits values (8 bits for R, B, and G each)

*24 bits per pixel*

$2^{24} = 16.7$  million colors possible

computer monitors rescan 75 times/second (Hz)

flickering is eliminated by repainting the same frame multiple times: the frame is shown again before your retina starts to lose it

## **digital video: quality and bitrate**

bandwidth in digital systems is expressed as bits-per-second (bps), or *bitrate*

suppose we're sending

- N frames / second (frame rate)
- P pixels per frame (screen size)
- D bits / pixels (color depth)

then the required bitrate is  $N * P * D$  bps

example: 25 frames of 1024\*768 (VGA) with 24 bit colors requires 471859200 bps (472 Mbps)

conclusion: we obviously need compression for transmitting video over the internet!

## **data compression**

improving the encoding of video data, so that it requires smaller bandwidth to transmit

often, data is encoded only once but decoded many times e.g. in the case of creating a DVD movie

as a result, encoding may be much more expensive than decoding (asymmetrical)

## **data compression: lossy vs lossless**

### *lossless* compression:

exact data can be reconstructed from the encoded data  
used when you cannot know which information inside the data  
is critical

Luckily, *Zip* compression of files is lossless

### *lossy* compression:

encoding/decoding need not be invertible: the original  
signal/data cannot be retrieved exactly from the encoded data!  
makes use of known properties of the data, or assumptions  
about properties of the data

perceptual coding is usually 'lossy'. MP3 certainly is!

## **JPEG photograph encoding**

standard for compression of continuous-tone still images  
(photographs)

by the Joint Photographic Experts Group

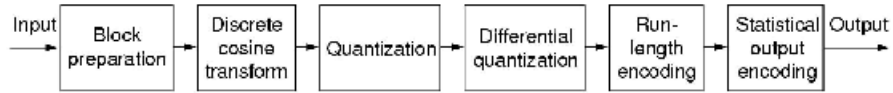
it is the basis for MPEG video encoding

JPEG has 4 modes, including *lossy sequential mode*

it is a very complicated encoding!

but reaches ~ 20:1 compression

## JPEG lossy sequential mode



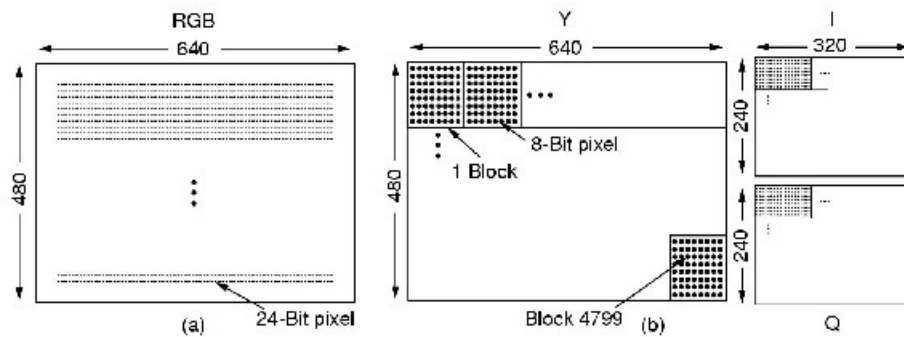
block preparation separates luminance from chrominance information and immediately lossy-compresses the latter (people respond less to chrominance than to luminance)

DCT is kind-of like a discrete Fourier transform in two spatial dimensions (this is also slightly lossy)

quantization is also lossy and removes the finest spatial structure; how much is up to each implementation and largely determines the rate of compression achieved

the last three steps are more easy...

## block preparation



(a) RGB input data  
24 bits-per-pixel

(b) after block preparation  
Y = luminance image  
Q,I = chrominance images

## **MPEG standards**

family of standards for coding audio and visual information (movies) in digital compressed format  
by the Motion Pictures Experts Group

compresses both audio and video  
video compression is largely based on JPEG encoding algorithm, which exploits *spatial* redundancy in images

also *temporal* (inter-frame) redundancy is exploited in MPEG compression

## **MPEG standards**

MPEG-1: video recorder quality (352x240 NTSC)  
1.2 Mbps bitrate (1 minute of MPEG-1 movie = 9 Mbytes)  
instead of 50.7 Mbps for uncompressed 24 bit/pixel RGB  
used as Video CD (VCD) standard  
its *Layer 3* audio format is known as *MP3*

MPEG-2: broadcast quality  
4 to 6 Mbps bitrate  
basis of the DVD standard and digital satellite TV

MPEG-4: medium resolution / videoconferencing  
low frame rates (10 frames/sec), low bandwidth (64 kbps)

## MPEG video

exploits:

spatial redundancy, by using JPEG compression

temporal redundancy (it assumes that most consecutive frames are almost identical)

temporal redundancy:

exploited by coding differences between consecutive frames

easy when background is stationary

but what if camera pans or zooms? then everything moves!

MPEG-1 uses concept of *macroblocks* that are matched between consecutive frames

how macroblocks are matched is not specified but left to each specific implementation

## macroblocks



three consecutive frames

**end of the course**

thanks for attending, and good luck with the exam!

no lab-assistance today

Thursday March 16, 10:00 – 13:00h instead

'affective computing' course in July