DOCUMENT
Anup Basu

Audio Image Video Data Graphics Objectives

Compression

Encryption

Network Communications

Decryption

Decompression

Client
site

Presentation of Information to client site

Multimedia

- two different types
-
  1) without any communication network
     i.e.) play a CD on computer, retrieve from local disk
  2) multimedia with communication

- types of problems very different

  e.g., –no security involved in non-network
      -compression may be less of an issue

Components of Multimedia Communication System

-info (media) sources
-compression/decompression technology
-encryption/decryption technology
-communication networks and protocols
-technologies and protocols for media synch

Components related to a Multimedia Communication System

-Multimedia databases
-User interfaces
-Media creation devices
-Communication with media creating devices
    -i.e. scanner

<u>Media Types</u>

-data (text)   -image   -graphics
-audio   -video

Compression
- Data → cannot afford to lose information
- Audio / image / video → can afford to lose information
- Graphics → maybe

<u>Some Compression Methods</u>

-RLE, Huffman encoding etc. for LOSSLESS
-Sub-band coding, CELP for Audio
-JPEG, Wavelet, fractal for Images
-H.263, MPEG-1, MPEG-2 for video
-MPEG-4, Emerging standard considering all media types, including graphics.

<u>Overview of Compression</u>

-Goal of compression is to reduce redundancies in information

-Types of redundancies
       -coding redundancies
       -spatio / temporal redundancy
       -perceptual redundancy (Human)

 Reducing coding redundancies is completely lossless
 Reducing spatio / temporal redundancies can be lossy
 Reducing perceptual redundancies is lossy

What kinds of events convey more information?
Events: 1) There was a car accident
             2) It snowed in Edmonton on January 1
             3) It snowed in Las Vegas on July 10
             4) A Jumbo Jet crashed

Event 3   contains the most because it's the rarest event
Event 4   eventful, but not as rare
Event 2   not that important, because it happens so often
Event 1   is almost not worth mentioning

Let P(E) = Probability of an event E occurring

$$\text{Info content in E} \quad \text{proportional to} \quad \frac{1}{P(E)}$$

Suppose b digits are used to code an Event or Symbol. Then, information content $= \log_b \dfrac{1}{P(E)}$

H = Entopy = Average Information content of a set of events (or symbols)

$E, E_2, E_3, \ldots, E_n$

$P(E_1),\qquad P(E_2),\qquad ....P(E_n)$

where $\qquad 0 \leq P(E_i) \leq 1$ & $\Sigma P(E_i) = 1$

$$H = \sum_{i=1}^{N} P(E_i) \times \text{Info content of } E_i = \Sigma P(E_i) \log P(E_i)$$

$$\boxed{H = \Sigma P(Ei) \log{}_b P(Ei)}$$

where b is base used for coding (e.g., 2 for binary)

When $P(E_i) = \dfrac{1}{N}$, for $i = 1, 2, ..., N$

the Entropy reaches its maximum value.

$$\boxed{\text{Hmax} = \log_b N}$$

where b is what base (i.e. 2 for binary)

For example, if base = 2 (i.e. Banary)

$$H_{max} = \log_2 N$$

e.g. 4 symbols:

code			Prob
00	A		¼
01	B		¼
10	C		¼
11	D		¼

Entropy $- \log_2 \not{c} = 2$ if binary representable

Usually during coding, some symbols appear more likely than others. In this case, it is possible to have variable length codes representing these symbols in order to reduce the average code length and get close to the <u>entropy</u>.

Huffman Coding

Idea is to create variable length codes depending on the probability of appearance of different symbols. Symbols that appear frequently got shorter codes.

The probability of a symbol (X) is nothing but the proportion of time X appears in the string of symbols to be coded. (P.344 in Image Processing text)

6 symbols: A B C D E F   P(A) = 0.4   P(B) = 0.3 P(C) = 0.1 P(D) = 0.1
P(E) = 0.06   P (F) = 0.04

Using fixed length codes, 000 = A, 001 = B, …101 = F

H= entropy = $-\Sigma$ P (E_i) x lg P (E_i) = 2.14

not efficient

for variable length codes, we use Huffman method:

Step 1 Source Reduction
-list symbols in order from largest to smallest probability
-we then combine successive two small + probability symbols, until 2 left

A	0.4	------- 0.4	------- 0.4	------- 0.4	------- 0.4
B	0.3	------- 0.3	------- 0.3	------- 0.3	0.6
C	0.1	------- 0.1	------- 0.1	------- 0.3	
D	0.1	------- 0.1	------- 0.2		
E	0.06	0.1			
F	0.04				

Step 2 Code Generation
In this step we work backwards, assigning variable length codes:

	(v)	(iv)	(iii)	(ii)	(i)
A	1	------- 1	------- 1	------- 1	------- 1
B	00	------- 00	------- 00	------- 00	0
C	011	------- 011	------- 011	------- 01	
D	0100	------- 0100	------- 010		
E	0100	0101			
F	01011				

Thus final Huffman codes will be:

A = 1   B = 00   C = 011   D = 0100   E = 01010   F = 01011

Average Huffman Code length = 2.2 bits/symbol

$$\underbrace{0.4 \times (1)}_{A} \quad \underbrace{+\ 0.3 \times (2)}_{B} + \underbrace{0.1 \times (3)}_{C} \quad \underbrace{+\ 0.1 \times (4)}_{D} \quad \underbrace{+\ 0.06 \times (5)}_{E} \quad \underbrace{+\ 0.04 \times (5)}_{F}$$

Compared to entropy = 2.12 bits/symbol

$$-\ (\ \underbrace{0.4 \log_2 (0.4)}_{A} \underbrace{+0.3 \lg (0.3)}_{B} + \underbrace{0.1 \lg (.1)}_{C} + \underbrace{0.1 \lg (.1)}_{D} + \underbrace{0.06 \lg (.06)}_{E} + \underbrace{0.04 \lg (.04)}_{F}\ )$$

Note: codes are unique and identifiable <u>without</u> delimiters as individuals.

e.g.,   A = 1, and nothing else starts with 1
            B = 00, and nothing else starts with 00

Run Length Encoding

from Huffman,…

letters	prob	code	# of bits in code	total = 50	110 # of bits
A	04	1	1	20	20
B	0.3	00	2	15	30
C	0.1	011	3	5	15
D	0.1	0100	4	5	20
E	0.06	01010	5	3	15
F	0.04	01011	5	2	10

Suppose we have 50 letters in a text with probabilities

Using Huffman encoding:
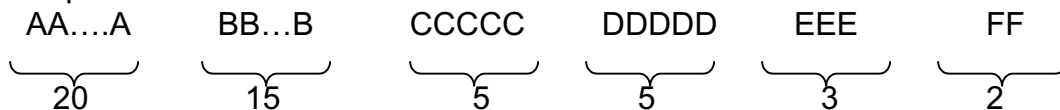
110/50 = 2.2 bits / symbol

Overhead for code:
        -need to store (or transmit) the codes and corresponding letters.

e.g. (1,A), (00,B), ….

Huffman (or other similar codes) by itself does not take into account the arrangement of letters ABCDEF in the text.

For example:

AA….A	BB…B	CCCCC	DDDDD	EEE	FF
20	15	5	5	3	2

5 bits can count up to 32
3 bits can represent 6 characters

Now, lets look at an alternate way of coding a string such as this.

    <u>20A</u>

    $5+3$ 8
    8×6 48 bits $\Rightarrow$ $48/50 = 0.96$ bits/symbol

Combining Run-length with Huffman

-Usually done for Binary code stream

011000110011110101…..Binary stream

(1,2), (2,1), (3,0), (2,1), (2,0), (4,1)…..
   1   2   3   2   2   4   3 Run Lengths

if we establish the first bit is 0, then it follows that it will alternate for each Run length thereafter.

0 1 2 3 2 2 4 } new stream

How can I use Huffman here??

We use Huffman encoding to code the run length

Using Huffman with Run Lengths

1. First, use run length to encode the bit stream

2. Second, use Huffman to encode the run lenths

Coding Text, e.g. Unix Compress

- Can use a dynamic dictionary (code book)

Option 1: Start with a default skeleton dictionary of frequently used words
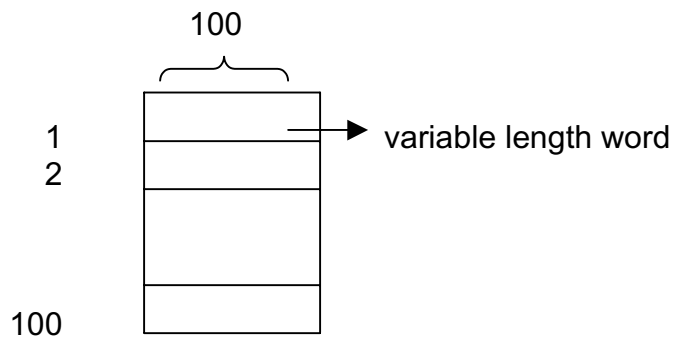   (context sensitive)

1	IS	
2	THE	
3	CAN	Default dictionary
	.	
	.	
	ANDRE	Dynamic dictionary
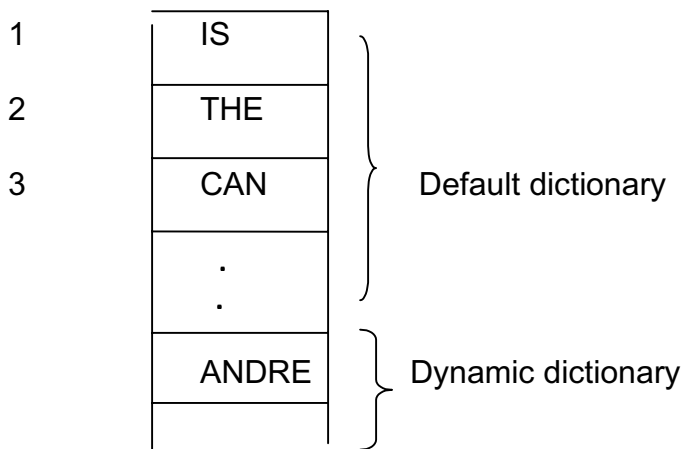
* Universal Method (note: a priori knowledge of source statistics is required)

♦ Messages are encoded as a sequence of addresses to words in the dictionary

- Repeating patterns become words in the dictionary

-Superior to run length encoding in most cases

```
              100
           ┌───────┐
  1        │       │──────► variable length word
  2        │       │
           │       │
           │       │
 100       └───────┘
```

-original algorithm was developed by Ziv & Lempel

-Practical implementation was done by Welch, so its called Lempel-Ziv-Welch (LZW) coding

-Unix compress is a variation of this method

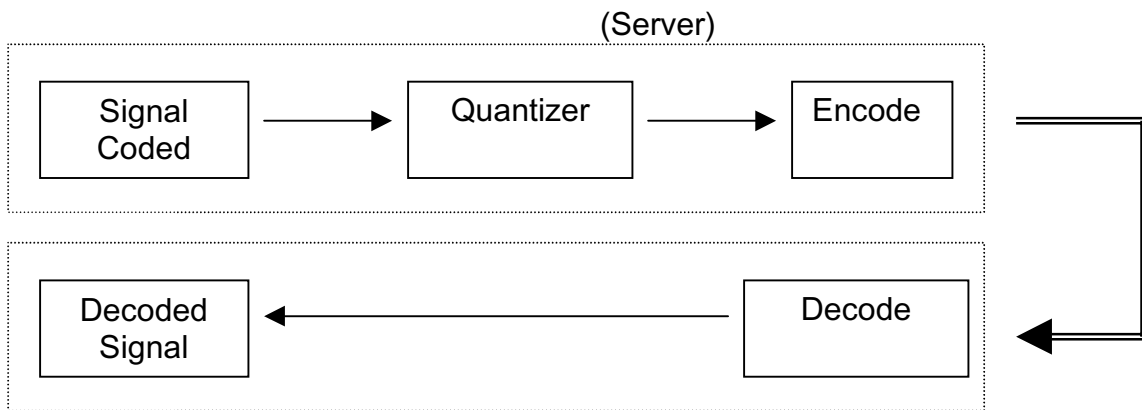(no guarantee that LZW will do better than Huffman)

General Models for Compression / Decompression

-they apply to symbols data, text, and to image but not video

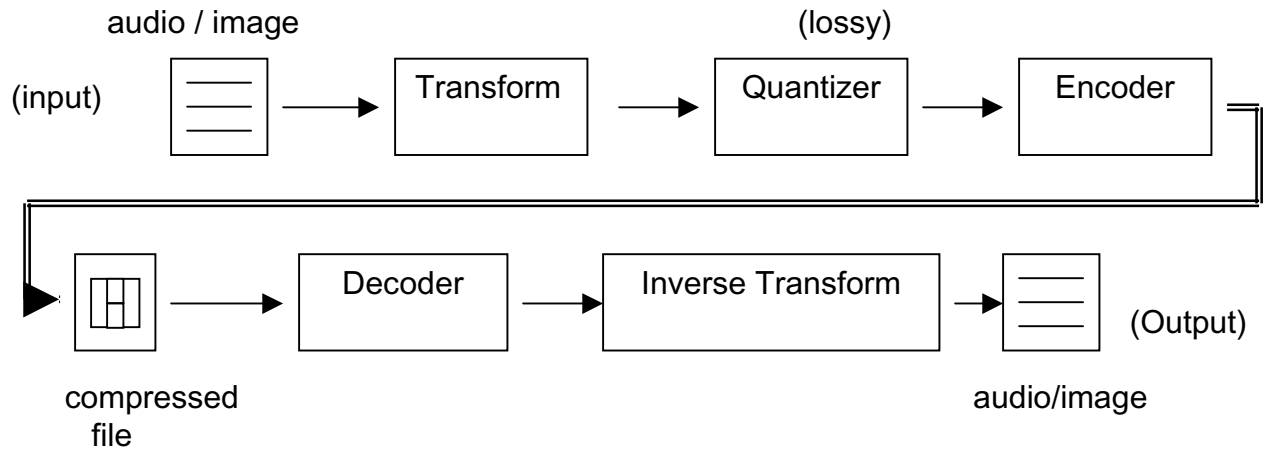1. Simplest model (Lossless encoding without prediction)

(server)

| Signal | → | Encode |

Transmit

(client)

| Signal | ← | Decode |
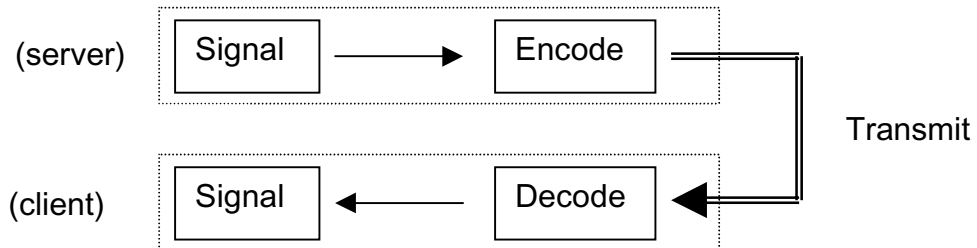
2. Lossy coding without prediction:

(Server)

| Signal Coded | → | Quantizer | → | Encode |

| Decoded Signal | ← | Decode |

Quantization: → Initial Data   0, 1, 2, 3 … 256
                Quantized Data 0, 1, 2, 3, 4, 8, 16, 32, 64, 128, 256

↳ Lossy

1

3. Transform Coding:

audio / image   (lossy)

(input) |≡| → | Transform | → | Quantizer | → | Encoder |

|⊞| → | Decoder | → | Inverse Transform | → |≡| (Output)

compressed   audio/image
file

One of the most popular transforms is called the discrete cosine transform (DCT)

In the frequency domain, we can have:
        -Fourier transform
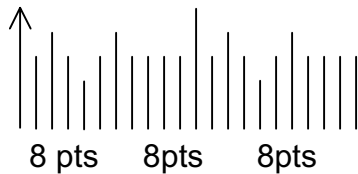   -Sin transform
        -Cosine transform

1 – D DCT:
- forward transformation

$$F(u) = \frac{c(u)}{2} \sum_{x=0}^{7} f(x) \cos\left[\frac{(2x+1)\,\pi\,u}{16}\right] \quad , u = 0, 1, 2 \ldots, 7$$

- inverse transform

$$c(u) = \begin{cases} \tfrac{1}{2} & \text{for } u = 0 \\ 1 & \text{for } u > 0 \end{cases}$$

$$f(x) = \sum_{u=0}^{7} \frac{c(u)}{2} F(u) \cos\left[\frac{(2x+1)\,u}{16}\right] \quad , x = 0, 1, 2 \ldots 7$$

8 pts   8pts   8pts

2

<u>2-D DCT</u> [Works on 8 x 8 image blocks]

- forward:

$$F(u,v) = \frac{2}{N} \, C(u) \, c(v) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} f(x,y) \quad \cos\left[\frac{u\Pi(2x+1)}{2N}\right] \cos\left[\frac{v\Pi(2y(1)}{2N}\right]$$

- inverse

$$f(x,y) = \frac{2}{N} = \sum_{u=0}^{N-1} \sum_{v=0}^{N-1} F(u,v) \, v(u) \, c(v) \quad \cos\left[\frac{u\Pi(2x+1)}{2N}\right] \cos\left[\frac{v\Pi(2y(1)}{2N}\right]$$

$$F(0,0) = \frac{1}{1} \sum \sum f(x,y) = N \text{ (average Grey level of 8 x 8 block)}$$

- lower u,v values represent lower frequencies or slow transition (smooth variations) ina signal. Human perception is more sensitive to changes in smooth variations, so we use more precise quantization at lower u v numbers and less precise with higher u,v levels; the higher u, v values represent sudden changes in a 1-D signal or sharp edges in a 2-D image

-compression is achieved by specifying 8 x 8 quantization table, which usually has larger values for higher frequencies (i.e. higher (u,v))

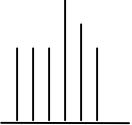-default quantization tables are created taking number perception into account

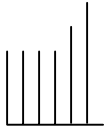-however, you can choose your own quantization tables.

<u>Some other classes of Transforms</u>

-wavelets (to be discussed in labs. + programming assignment)
-Gabor filters

f (*x,y*) usually 0 to 255 for 8 bit gray scale

4. Predictive Coding

1-D Digital Audio

(Given Signal)   Prediction: last sample point

(Predicted signal)

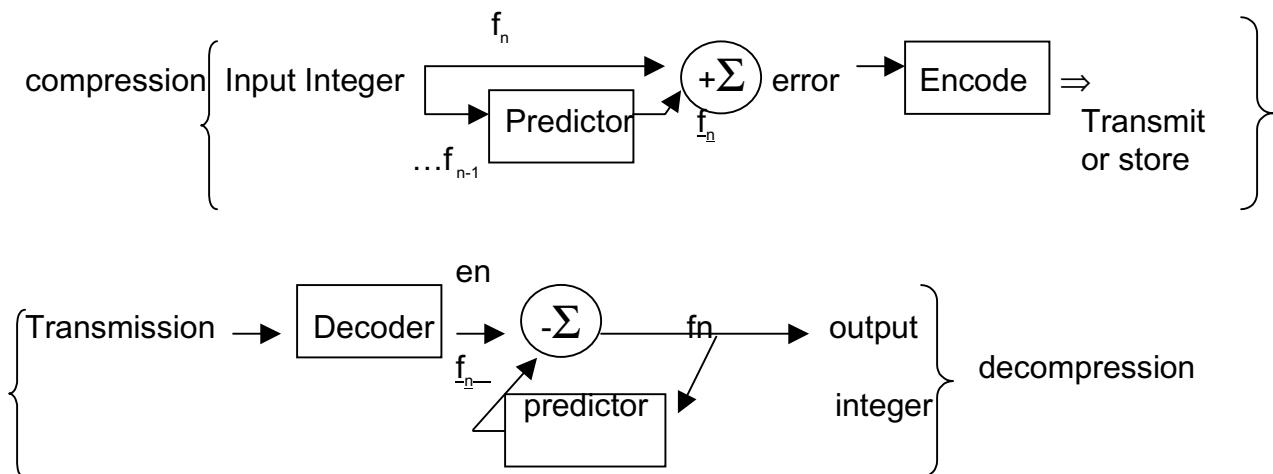(error after pred)

 Predictive coding types 1)lossless   2) lossy

lossless -> does not quantize error in prediction

lossy -> quantize errors in prediction

LOSSLESS MODEL:

$$f_n$$

compression $\{$ Input Integer $\longrightarrow$ $+\Sigma$ error $\rightarrow$ Encode $\Rightarrow$

Predictor $\quad \underline{f_n}$

$...f_{n-1}$

Transmit
or store $\}$

en

Transmission $\rightarrow$ Decoder $\rightarrow$ $-\Sigma$ fn $\rightarrow$ output

$\underline{f_n}$

predictor

integer $\}$ decompression

The most effective AUDIO codes use a form of linear prediction (LP). Some popular ones are:

    -CELP (Code Excited Linear Predictions)
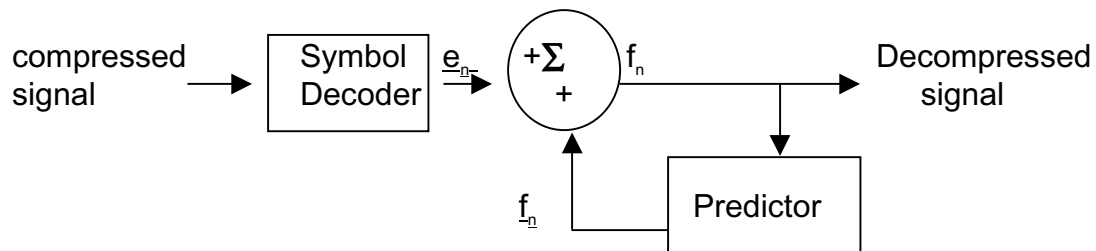
Lossy Predictive Coding:

$\underline{f}_n$: estimate of f_n

$e_n = f_n - \underline{f}_n$ = error in estimate

Coder

f_n $\quad>$ $\quad +\Sigma \quad e_n$ $\quad$ Quantizer $\quad \underline{e}_n$ $\quad$ Symbol Encoder $\quad$ Compressed signal

$\underline{f}_n$ $\quad$ Predictor $\quad \underline{f}_n \quad \Sigma \; + \atop +$

Decoder

compressed signal $\quad$ Symbol Decoder $\quad \underline{e}_n \quad +\Sigma \atop + \quad f_n \quad$ Decompressed signal
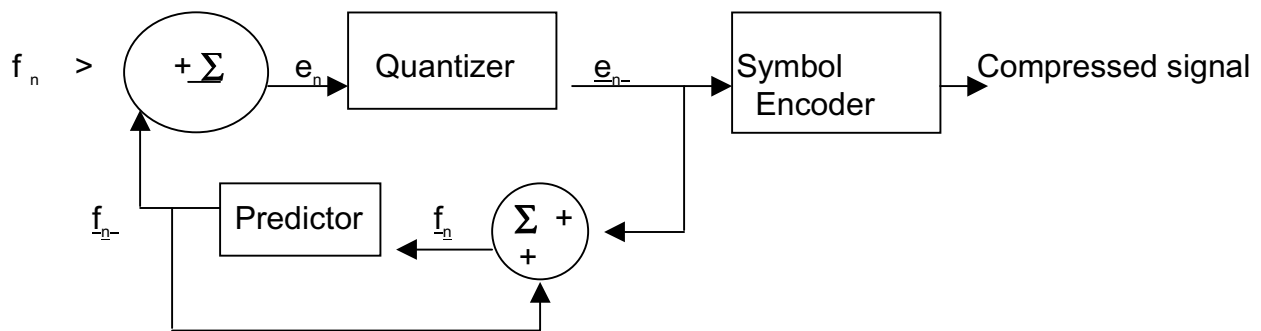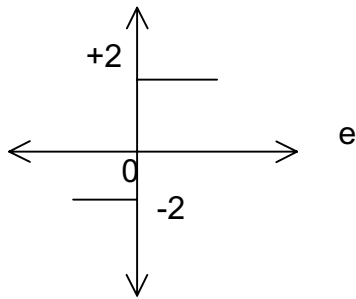
$\underline{f}_n$ $\quad$ Predictor

One of the simplest lossy predictive coding methods is known as "Delta Modulation"

1) Uses a simple predictor like $\quad \underline{f}_n = X (f_{n-1})$

2) Quantization is a simple Delta function with 2 levels depending on the error.

e quanticed error

$+2$

0

-2

e

$\underline{e}_n = \begin{cases} +2 & \text{for } e_n \geq 0 \\ -2 & \text{otherwise} \end{cases}$

Example

f_n : 10, 9, 15, 13, 6, 12, 15, 16, 18, 24, ….

lossless predictive coding $\underline{f}_n = f_{n-1}$, i.e., x = 1

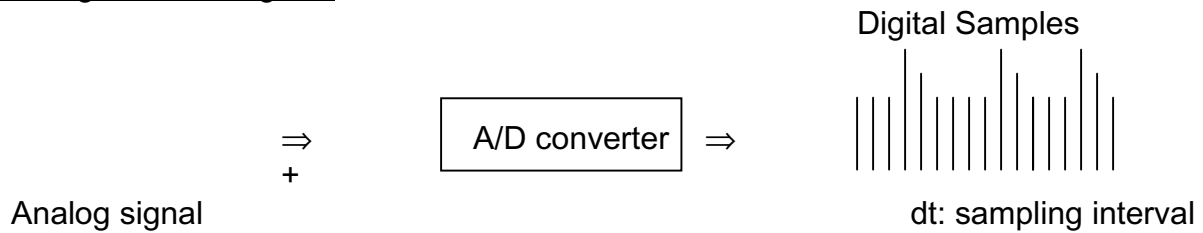$e_n = f_n - \underline{f}_n = f_n - f_{n-1}$

e_n = 1, 6, -2, -7, 6, 1, 2, 6, …

these e_n's are then encoded using the symbol encoder

lossy predictive coding using diagram at the top (Delta mode)

$\underline{f}_{n} = d\ f_{n-1} = f_{n-1}$ $e_n = f_n - \underline{f}_n = \boxed{f_n - f_{n-1}}$

$\underline{e}_n$:

Coding of Audio Signals

Digital Samples

Analog signal ⇒ + | A/D converter | ⇒

dt: sampling interval

- number of levels for each Digital Sample depends on the number of bits / sample of the A/D converter

-A/D can be 8 bits / sample, 10, 12, etc. …

256 level   1024 levels   1096 levels

dt: The sampling interval depends on the width of the frequency interval that you wish to digitize.

$$dt \leq \frac{1}{\text{width of freq. interval}} \quad : \text{Sampling theorem}$$

So that you can reconstruct audio exactly, in a given frequency band, from the digital samples.

Amount of Data (w/o any compression) is determined by number of bits per sample and sampling interval
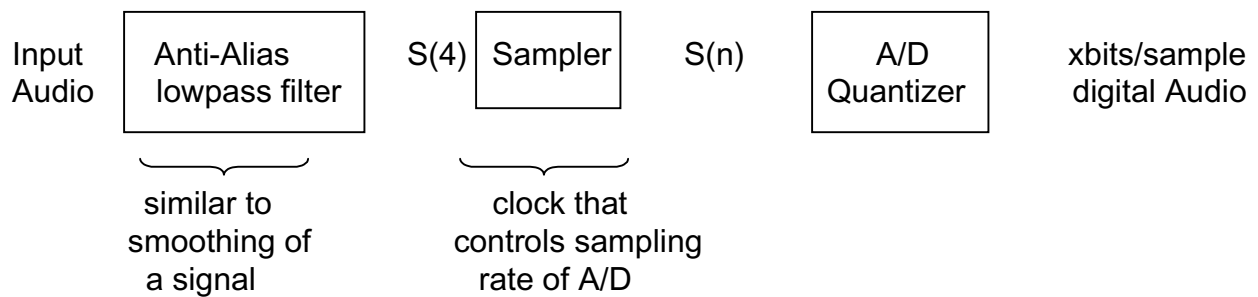
e.g.) 8 bits/sample and 8,000 samples / sec

→   64,000 bits/sec of uncompressed data

7

Time Domain Waveform Coding

·PCM – Pulse Code Modulation
·APCM – Adaptive Pulse Code Modulation

PCM is simply an anti alias lowpss filter which can be applied before the A/D conversion process.

| Input Audio | Anti-Alias lowpass filter | S(4) | Sampler | S(n) | A/D Quantizer | xbits/sample digital Audio |

similar to
smoothing of
a signal

clock that
controls sampling
rate of A/D

Adaptive PCM

-Adaptive Quantizer Pulse Code Modulation

·Variable step-size, which varies depending on short-term audio amplitude

·stepsize is proportional to average short-term amplitude

·two ways of implementing this

      a) variable quantizer

      b) variable gain

Differential Pulse Code Modulation (DPCM)

$\underline{f}(n)$:   estimate of $f(n)$

$e(n)$ = $f(n) - \underline{f(n)}$

$e(n)$ is quantized using a fixed quantizer

Adaptive Differential Pulse Code Modulation (ADPCM)

The quantizer stepsize is ADAPTED depending on the level of the amplitude (or the variations of the signal over a local window) etc.

-lower levels (or smaller variations)
    $\Rightarrow$ smaller stepsize in Quantizer

-large variations (higher levels)
    $\Rightarrow$ larger stepsize in Quantizer

Details of DPCM
    -first step is to find a "good" predictor
i.e.) $\underline{f}_n$ to estimate f_n, based on past values $f_{n-1}f_{n-2}\ldots$

$$\boxed{e_n = f_n - \underline{f}_n}$$ We will restrict ourselves to only linear predictors

$$\underline{f}_n = \Sigma \, x_i f_{n-i}, \quad \Sigma \, x_i \leq 1 \quad\quad 0 < X_i < 1$$

To compute the best predictor, with m coefficients, we need to find "optimal" values of X_i's $l = 1, \ldots m$

Optimizing a certain criterion. The criterion usually used is "minimum means square error"

$$E(e^2_n) = E\left\{ (f_n - \underline{f}_n)^2 \right\} = E\left\{ (f_n - \Sigma \, x_i f_{n-i})^2 \right\} \quad\quad \text{------ (*)}$$

Idea is to find X_i's minimizing (*)

To do this, we take derivative WRT X_i and set it equal to zero.

$$\frac{d}{dx_1}\left[f_n - \Sigma\, x_i f_{n-I}\right]^2 = -2\, f_{n-1}\left[f_n - \Sigma\, x_u f_{n-i}\right]$$

$$E\left[\frac{d}{dx_i}\left(f_n - \Sigma\, x_i f_{n-I}\right)^2\right] = E\left[2\, f_{n-1}\left(f_n - \Sigma\, x_i f_{n-I}\right)\right] = 0$$

$$-2\left[E\,(f_n f_{n-1}) \quad - \quad E\left(\sum_{n-1}^{m} X_i f_{n-1}\, f_{n-I}\right) \quad = 0\right.$$

$$\Sigma\,(f_n f_{n-1}) \quad = \quad \sum_{i=1}^{m} x_i\; E\,(f_{n-1}, f_{n-i})$$

Partial, WRT L_2, gives

$$E\,(f_n f_{n-2}) \quad = \quad \sum_{n-1}^{m} X_i\, \Sigma\,(f_{n-1}\, f_{n-i})$$

$$E\,(f_n f_{n-m}) \quad = \quad \sum_{1-1}^{m} X_i\, E\,(f_{n-m}\, f_{n-i})$$

or, can be expressed in vector form as:

$$
\begin{vmatrix} E\,(f_n f_{n-1}) \\ E\,(f_n f_{n-2}) \\ \\ E\,(f_n f_{n-m}) \end{vmatrix}
=
\begin{vmatrix} E\,(f_{n-1} f_{n-1}), & E\,(f_{n-1} f_{n-2}), & \dots & E(f_{n-1} f_{n-m}) \\ E\,(f_{n-2} f_{n-1}), & E\,(f_{n-2} f_{n-2}), & \dots & E(f_{n-2} f_{n-m}) \\ \\ E\,(f_{n-m} f_{n-1}), & E\,(f_{n-m} f_{n-2}), & \dots & E(f_{n-m} f_{n-m}) \end{vmatrix}
\times
\begin{vmatrix} X_1 \\ X_2 \\ \\ X_m \end{vmatrix}
$$

i.e. $P = R\; X$; where P & X are vectors and R is a matrix.

So, what should X be??? $\qquad X = R^{-1} P$

In practice these computations are not practical in reasonable time, thus simplifications are made and global coefficients are computed based on a priori information.

Basic Concepts for Multimedia Synchronization

Media Types
        -Audio   video 30 frames / sec
        -Video   1   1   1   1   1   1   1
        -Text   1/30 1/30 1/20 1/40 1/15 …

                Time in sec between display of consecutive frames

Jitter: measure of variation of actual appearance of a frame from its expected location.

-People are most sensitive to jitter in sound.

Simple strategy for synchronization is to have one of the media types as master and the others as slaves.

e.g., Audio is master. For each and packet you'll have audio in formation, video packet number and text packet number.

| Audio | video # |
| | text # |

A strategy for reducing jitter is to use a buffer. The input packets are added to a buffer from which multimedia is played out.

The larger the buffer, the greater the delay in the playback. Thus buffering may not be acceptable for real-time applications such as video conferencing.

JPEG Image Compression Standard

-standards are created by an international body of "experts" in a field

-CCITT was the initial body that developed the JPEG standard
-based on D.C.T.
-JTCL/ST../…   JPEG2000 → use wavelets
-Currently standards are being developed in the MPEG-4 and MPEG-7 areas

Motion Encoding

H.261,H.263,   MPEG-1,   MPEG-2

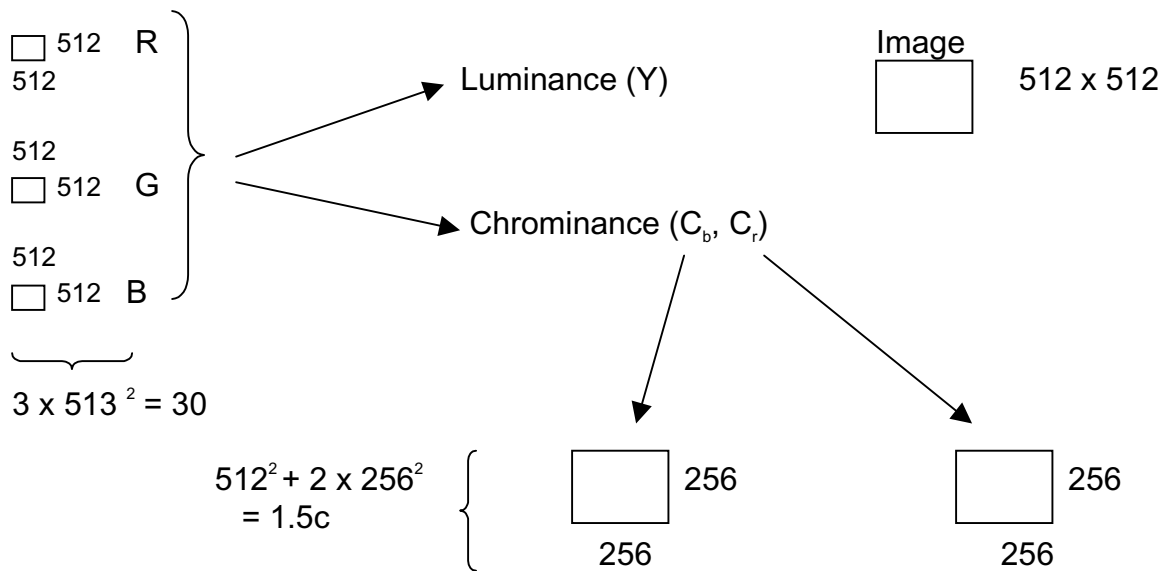## Multimedia (Encompassing Many Types of Media)

MPEG-4

Motivation

Why did the JPEG committee decide to choose certain options in their standard?

1) As size of image increases, transforms become more and more expensive. So, what sizes do we make the window and what transformation algorithm to use?

The JPEG committee chose to use an 8 x 8 DCT because MSE was not reduced significantly for larger windows (sub-image size) and DCT performed better than other well known transforms.

2)   Human Eyes (Visual System) are more sensitive to luminance (brightness) changes rather than chrominance (color) changes.

Thus, JPEG has options for reducing the resolution of the chominance part, compared to the luminance part.

512 $\square$ 512 R

512

512 $\square$ 512 G

512

512 $\square$ 512 B

$3 \times 513^2 = 30$

Luminance (Y)

Chrominance (C_b, C_r)

Image $\square$ 512 x 512

$512^2 + 2 \times 256^2$ = 1.5c

$\square$ 256 / 256

$\square$ 256 / 256

As long as luminance is left unchanged, the chrominance change is less noticeable.

This saves 50% because

$3 \times 512^2 = 3 S$

initial

$512^2 + (2 \times 256^2) = 1.5 S$

final

3) Human eyes are more sensitive to loss at lower frequencies than loss at higher frequencies.

00 01

10

8.8

8×8 $\Rightarrow$

lower quantization values

higher quantization values

13
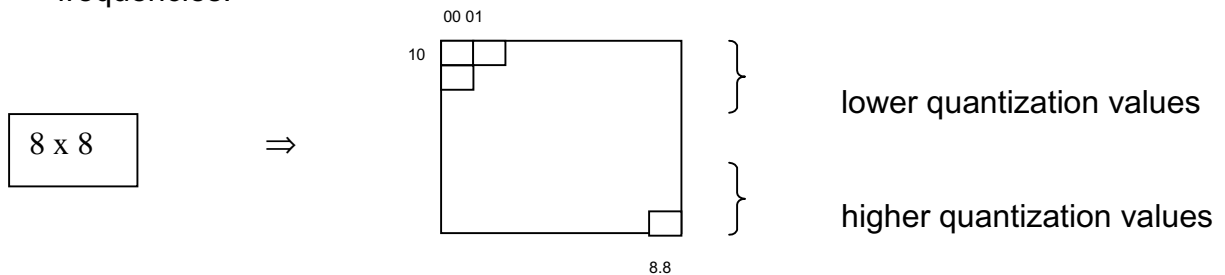
<u>JPEG Modes</u>:

1) Lossless

2) Baseline Sequential

> Every JPEG coder must support at least the Baseline standard
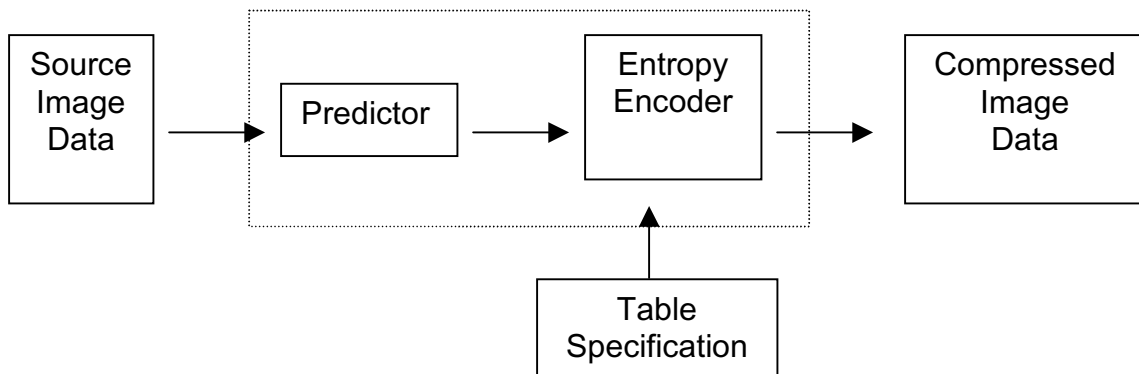
3) Progressive

4) Hierarchical

Methods 2, 3 and 4 are lossy transform coding based on DCT

Method 1 uses predictive coding → NO TRANSFORMATION

Progressive Mode: usually updates different frequency component (from LOW to HIGH) progressively.

Hierarchical Mode: create multi-resolution representation of images, and codes "Difference" between conseptive levels.

Lossless Mode:

```
┌──────────┐     ┌───────────┐     ┌───────────┐     ┌────────────┐
│ Source   │     │           │     │ Entropy   │     │ Compressed │
│ Image    │ ──▶ │ Predictor │ ──▶ │ Encoder   │ ──▶ │ Image      │
│ Data     │     │           │     │           │     │ Data       │
└──────────┘     └───────────┘     └───────────┘     └────────────┘
                                         ▲
                                   ┌───────────┐
                                   │ Table     │
                                   │ Specification │
                                   └───────────┘
```
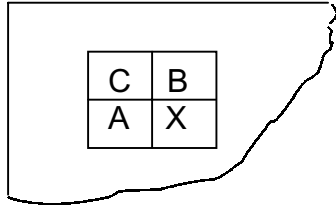
JPEG allows Huffman & Arithmetic encoding

Unlike audio, image compression must be 2 dimensional

Prediction:

-Want to predict value of X given values of A,B,C in the 2 x 2 neighborhood.

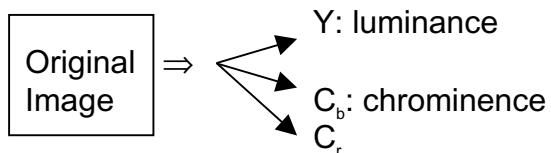| C	B
A	X

Predictors for lossless coding

Predictor Code	Prediction
0	No Prediction
1	A
2	B
3	C
4	A+B-C
5	A+[(B –C) /2]
6	B+[(A -C) /2]
7	(A+B) /2

example:

250	249
200	210

we can have positive as well as negative errors.

Baseline Sequential Mode:

Original Image $\Rightarrow$

Y: luminance

C_b: chrominence

C_r

Similar compression methods are used for all components, with the following differences:

(i)   The Chrominance resolution is usually reduced

(ii)   The quantization tables are different for the chrominance and luminance parts.

OUTLINE of Steps in the JPEG Baseline Compression Method:

1. Transform the (R,G,B) image into 3 channels based on Luminance (Y) & Chrominance (Cb, Cr).
2. Each channel is divided into 8x8 blocks; images are padded if necessary to make rows and columns multiples of 8.
3. Each 8x8 image block is transformed into 8x8 frequency blocks by using the Discrete Cosine Transform (DCT).
4. The DCTs are usually computed to 11-bit precision if the R, G, B colors have 8-bit precision. This extra precision in DCT computation is kept to compensate for loss of accuracy when the frequency coefficients are divided by a Quantization matrix.
5. The DCT coefficients are ordered in a zig-zag format, starting with the top left corner corresponding to F(0,0).
6. The DC coefficients (F(0,0)) are coded separately from the other coefficients (AC coefficients).
7. The DC coefficients are coded from one block to the next using a predictive coding strategy. [The DC coefficient in one block can be predicted using the DC coefficient in the previous block.]
8. The AC coefficients are quantized in the zig-zag order, then treating these values as a sequence of bits a Run-length Coding method is used with the run-lengths being coded using Huffman or Arithmetic coding.