

The Video Data Type

Coding & Compression Basics

Kevin Jeffay

Department of Computer Science
University of North Carolina at Chapel Hill

jeffay@cs.unc.edu

September 7, 1999

<http://www.cs.unc.edu/~jeffay/courses/comp249f99>

1

The Video Data Type

Outline

- ◆ What is video?
 - » Video components
 - » Representations of video signals
 - » Color spaces

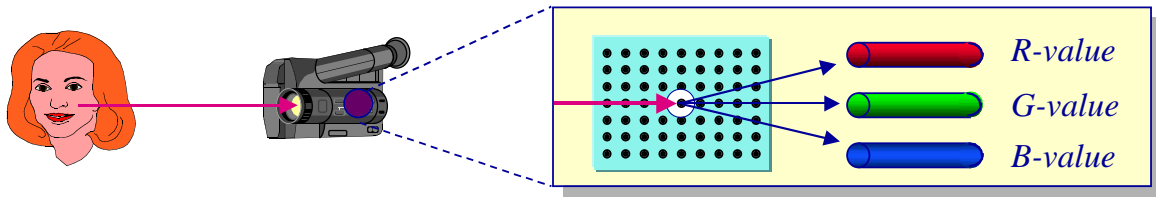
- ◆ Digital Video
 - » Coding

- ◆ Compression basics
 - » Simple compression
 - » Interpolation-based techniques
 - » Predictive techniques
 - » Transforms
 - » Statistical techniques

2

Video Basics

The components of video



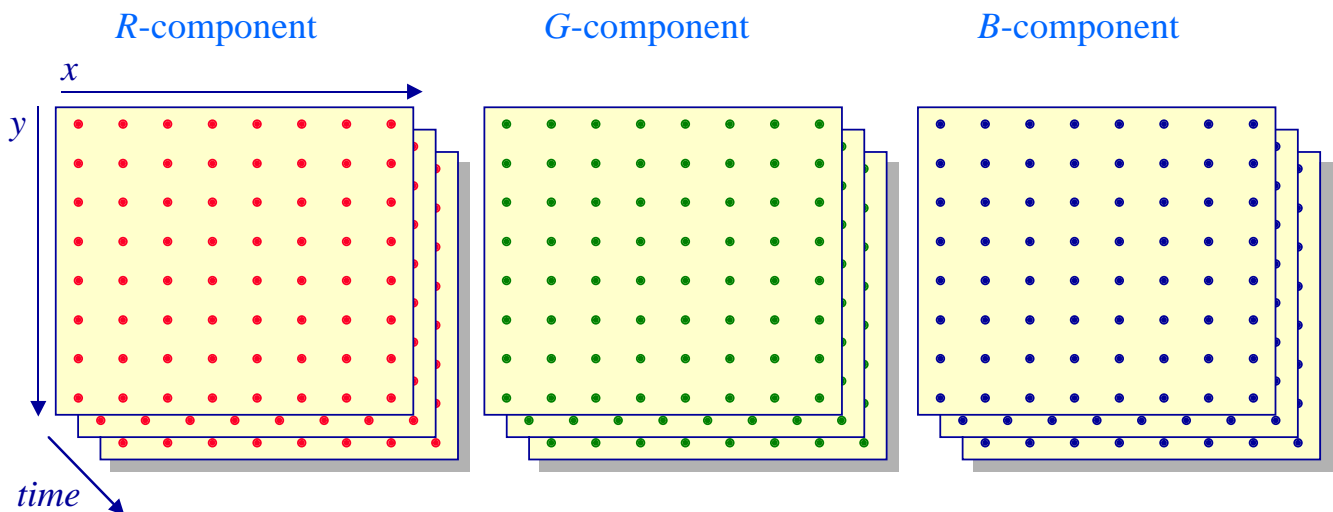
- ◆ Video deals with *absorbed* and *projected* light
 - » Cameras absorb light and monitors project light
- ◆ The primary colors in this domain are:
 - » red, green, and blue

3

Video Basics

The components of video transmission

- ◆ Video is a multi-dimensional signal

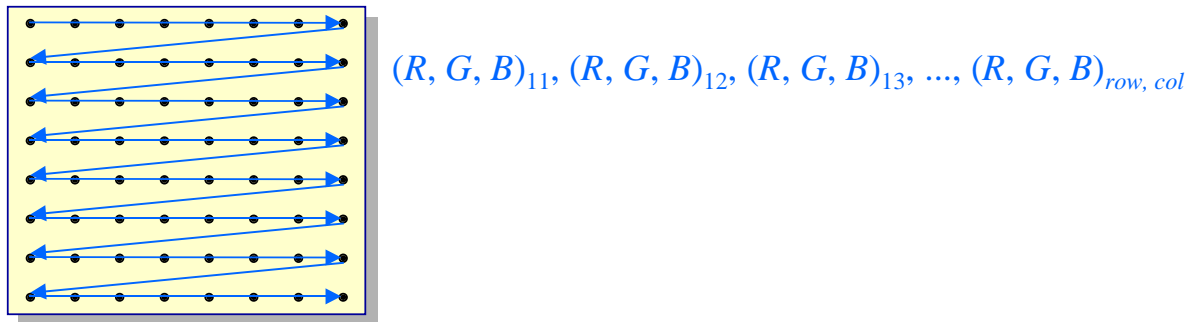


4

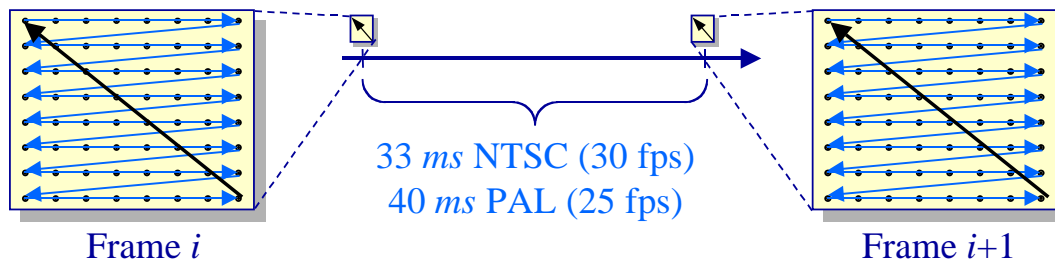
Video Basics

Video as a 1-dimensional signal

- ◆ Representation of a 2-dimensional image



- ◆ Representation of motion (3-dimensional images)



5

Video Basics

Resolution

- ◆ Television broadcast standards
 - » NTSC — 525 lines
 - » PAL — 625 lines
- ◆ Computer graphics standards
 - » VGA — 640x480
 - » SVGA — 1024x768
- ◆ Multimedia standards
 - » CIF — 352x288
 - » QCIF — 176x144
- ◆ Digital video standards
 - » CCIR 601 — 720x480
 - » HDTV — 1440x1152

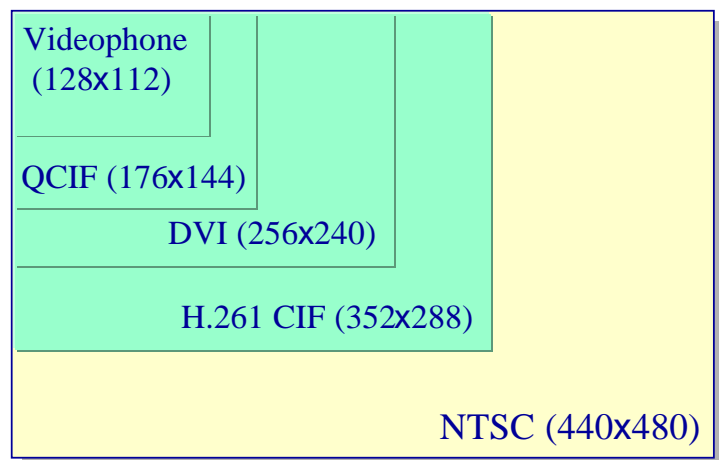


Image sizes
(in picture elements)

6

Video Basics

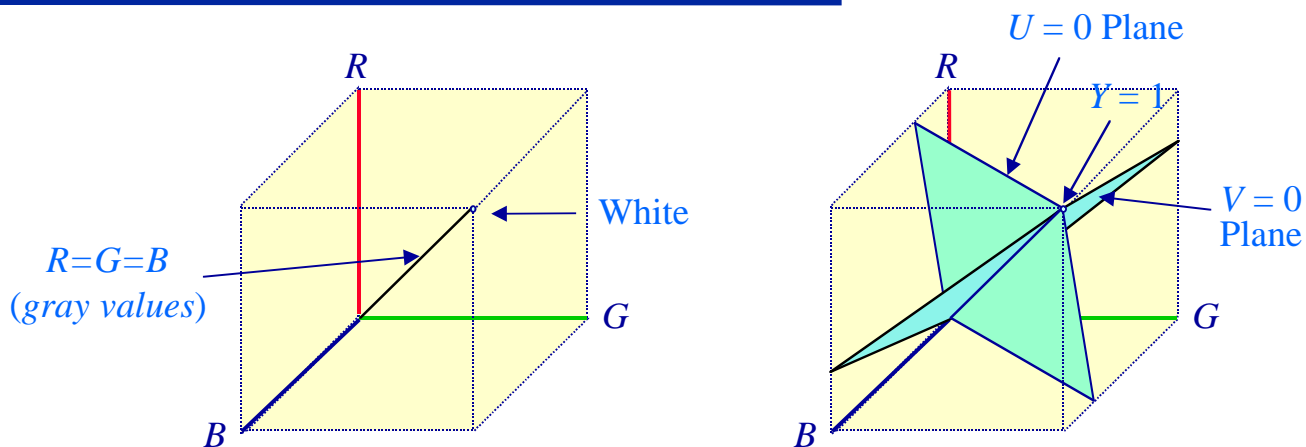
Color spaces

- ◆ *RGB* is not widely used for transmitting a signal between capture and display devices
 - » It's difficult to manage 3 separate inputs & outputs (and requires too much bandwidth)
- ◆ Composite formats are used instead
 - » Luminance (“*Y*”) — the brightness of the monochrome signal
 - » Chrominance — the coloring information
 - » Chrominance is typically represented by two “color difference” signals:
 - ❖ “*U*” and “*V*” (“*hue and tint*”) or
 - ❖ “*I*” and “*Q*” (“*saturation*” and “*color*”)

7

Video Basics

Color spaces



- ◆ *NTSC* video
 - » $Y = 0.30R + 0.59G + 0.11B$
 - » $I = 0.60R - 0.28G - 0.32B$
 - » $Q = 0.21R - 0.52G + 0.31B$
- ◆ *PAL* video/Digital recorders
 - » $Y = 0.3R + 0.6G + 0.1B$
 - » $U = (B - Y) \times 0.493$
 - » $V = (R - Y) \times 0.877$

8

Video Basics

Digital video

- ◆ Sample an analog representation of video (*RGB* or *YUV*) & quantize
 - » Two dimensions of video are already discretized
 - » Sample in the horizontal direction according to the resolution of the media

- ◆ 8-bits per component per sample is common
 - » 24 bits per picture element (pixel)

- ◆ Storage/transmission requirements
 - » NTSC — $440 \times 480 \times 30 \times 24 = 152 \times 10^6$ bits/sec (19 MB/s or 24 bits/pixel (bpp))

9

The Video Data Type

Outline

- ◆ What is video?
 - » Video components
 - » Representations of video signals
 - » Color spaces

- ◆ Digital Video
 - » Coding

- ◆ Compression basics
 - » Simple compression
 - » Interpolation-based techniques
 - » Predictive techniques
 - » Transforms
 - » Statistical techniques

10

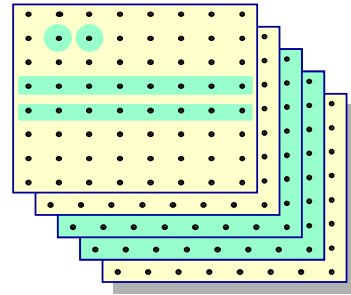
Digital Video

Compression Techniques

- ◆ Do we really need every “bit” of a video stream?
 - » Not if redundancy exists
 - » Not if we can't perceive the effect of eliminating the bit

- ◆ Eliminating redundancy
 - » Spatial redundancy
 - » Temporal redundancy

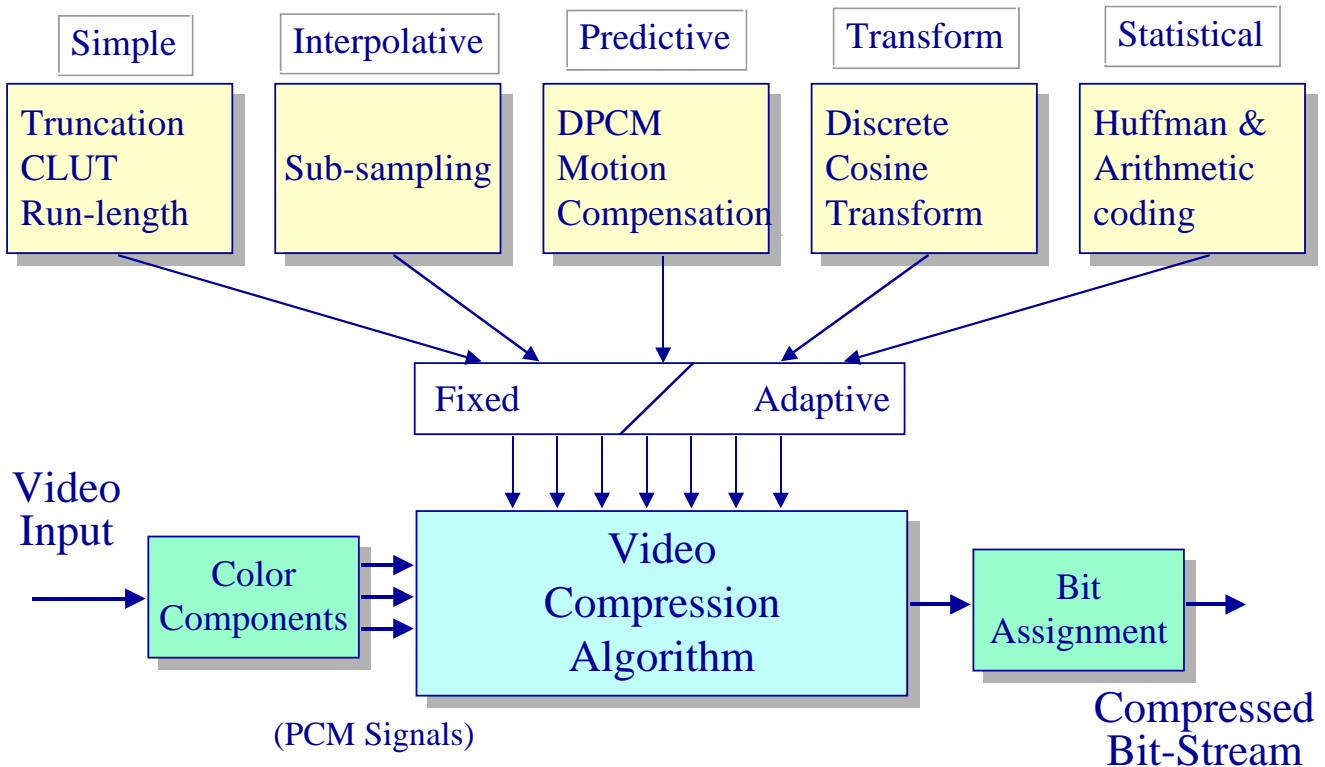
- ◆ Eliminating imperceptible detail
 - » Coding
 - » Domain transformation



11

Digital Video

Compression Techniques



Video Compression

Issues

- ◆ Bandwidth requirements of resulting stream
 - » Bits per pixel (bpp)
- ◆ Image quality
- ◆ Compression/decompression speed
 - » Latency
 - » Cost
 - » Symmetry
- ◆ Robustness
 - » Tolerance of errors and loss
- ◆ Application requirements
 - » Live video
 - » Stored video

13

Simple Image Compression

Truncation

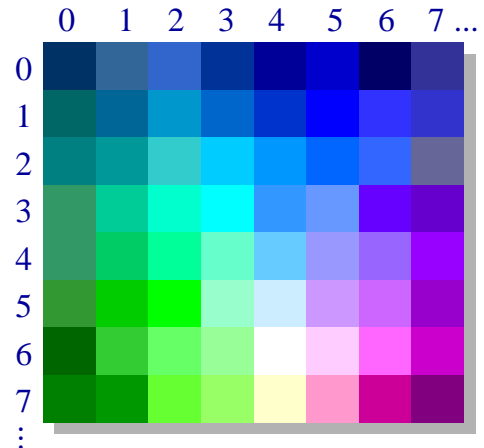
- ◆ Reducing the number of bits per pixel
 - » Throw away the least significant bits of each sample value
- ◆ Example
 - » Go from *RGB* at 8 bits/component sample (8:8:8) to 5 bits (5:5:5)
 - ❖ Go from 24 bpp to 15 bpp
 - ❖ This gives “acceptable results”
 - » Go from *YUV* at 8 bits/component sample 6:5:5 (16 bpp)
- ◆ Advantage — simple!

14

Simple Compression Schemes

Color-table lookup (CLUT)

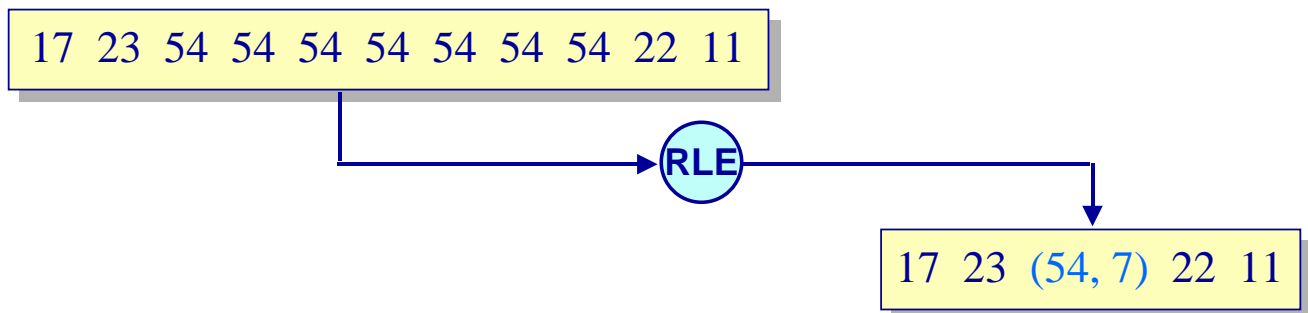
- ◆ Quantize coarser in the color domain
 - » Pixel values represent indices into a color table
 - » Tables can be optimized for individual images
- ◆ Entries in color table stored at “full resolution” (e.g. 24 bits)
- ◆ Example:
 - » 8-bit indices (256 colors) gives
 $(440 \times 480) \times 8 + (24 \times 256) = 1.7 \times 10^6$ bits/sec



15

Simple Compression Schemes

Run-length encoding



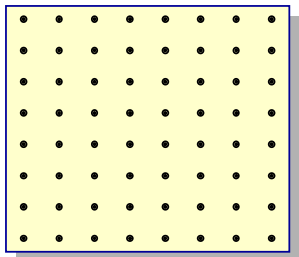
- ◆ Replace sequences of pixel components with identical values with a pair (*value, count*)
- ◆ Works well for computer-generated images, cartoons. works less well for natural video
- ◆ Also works well with CLUT encoded images
(i.e., multiple techniques may be effectively combined)

16

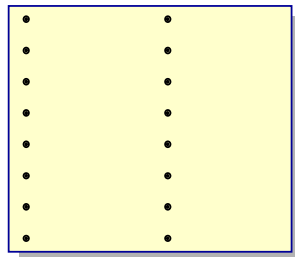
Interpolative Compression Schemes

Color sub-sampling

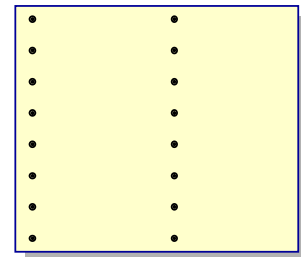
- ◆ Do not acquire chrominance component values at all sampling points
 - » Humans have poor acuity for color changes
 - » UV and IQ components were defined with this in mind
- ◆ Example: Color representation in digital tape recorders
 - » Subsampling by a factor of 4 horizontally is performed



Y component



U component



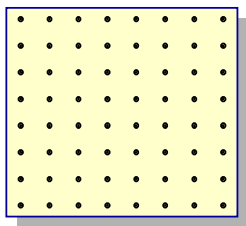
V component

17

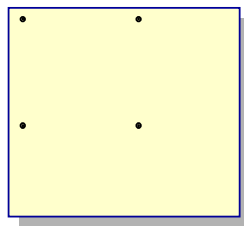
Interpolative Compression Schemes

Color sub-sampling

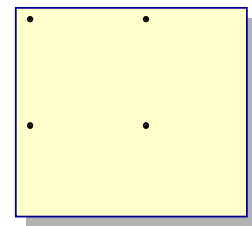
- ◆ Subsampling by a factor of 4 horizontally & vertically



Y component



U component



V component

- ◆ Interpolating between samples provides “excellent” results
 - » Chrominance still sampled at 8 bpp

18

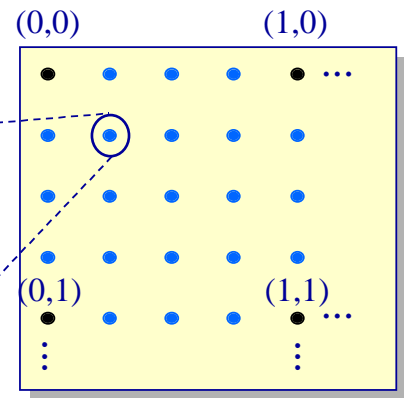
Interpolative Compression Schemes

Color sub-sampling

- ◆ Intermediate pixels either take on the value of nearest sampling point or their value is computed by interpolation

- ◆ Bi-linear interpolation:

$$U(1, 1) = U(0,0) \times 0.75 + U(1,0) \times 0.25 + U(0,1) \times 0.75 + U(1,1) \times 0.25$$

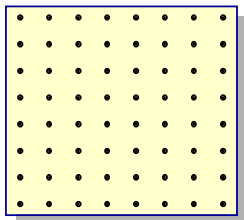


Sub-sampled
U or *V* component

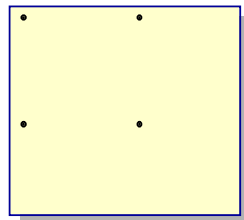
19

Interpolative Compression Schemes

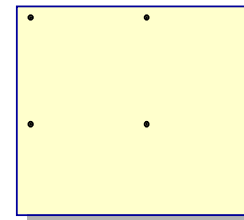
Color sub-sampling



Y component



U component



V component

- ◆ Storage/transmission requirements reduction:

- » Within a 4x4 pixel block:

$$\text{bpp} = \frac{(8 \text{ bpp luminance}) \times 16 \text{ samples} + (8 \text{ bpp chrominance}) \times 2}{16} = 9$$

- » A 62.5% reduction overall

20

Predictive Compression Schemes

Exploiting spatial & temporal redundancy

- ◆ Adjacent pixels are frequently similar
 - » Do pixel-by-pixel DPCM compression
 - ❖ Leads to smearing of high-contrast edges
 - » ADPCM — a little better, a little worse
 - ❖ Introduces “edge quantization” noise
- ◆ Motion Estimation — If the future is the similar to the past, encode only the difference between frames
 - » This assumes we can store a previous frame to compare with a future one

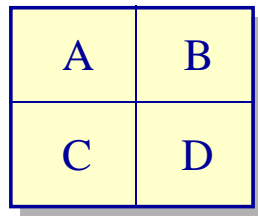
21

Transform-Based Compression

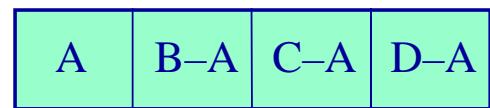
Exploiting redundancy in other domains

- ◆ A simple linear transformation

2 x 2 array of pixels



1-D array of differences



- » Encode differences with less precision

- ◆ Storage savings

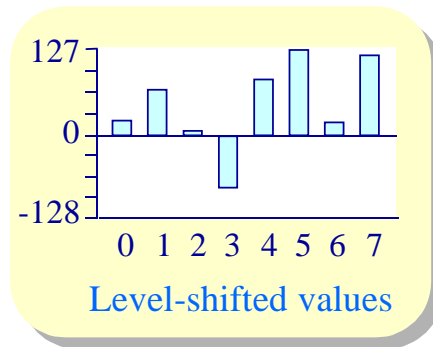
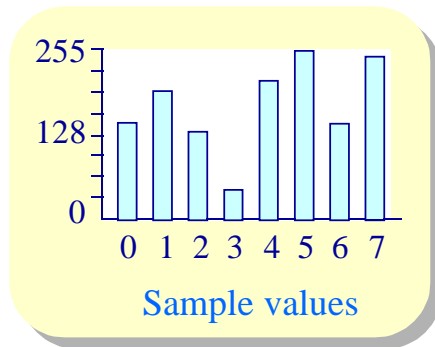
- » Original array: 4 pixels x 8 bpp = 32 bits
- » Transformed array: 8 bits + (3 pixels x 4 bpp) = 20 bits

22

Transform-Based Compression

The Discrete Cosine Transform (DCT)

- ◆ A transformation into the frequency domain
- ◆ Example: 8 adjacent pixel values (e.g., luminance)



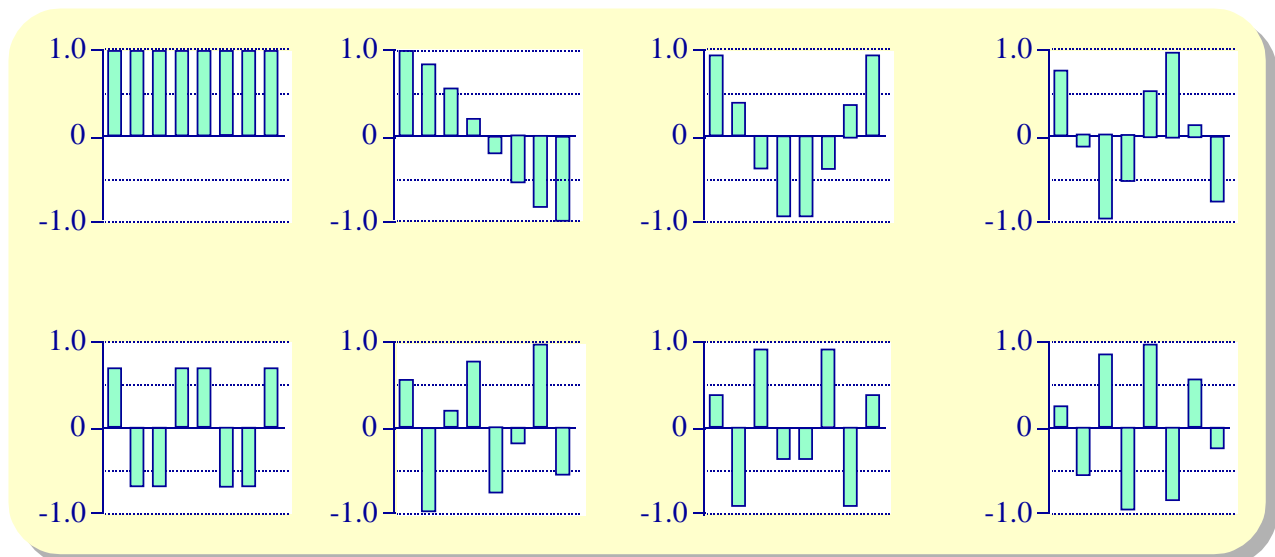
- ◆ What is the most compact way to represent this signal?

23

Transform-Based Compression

The Discrete Cosine Transform (DCT)

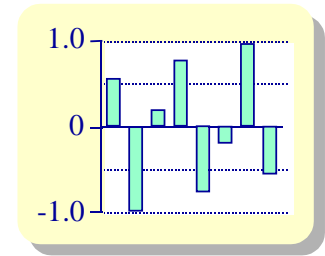
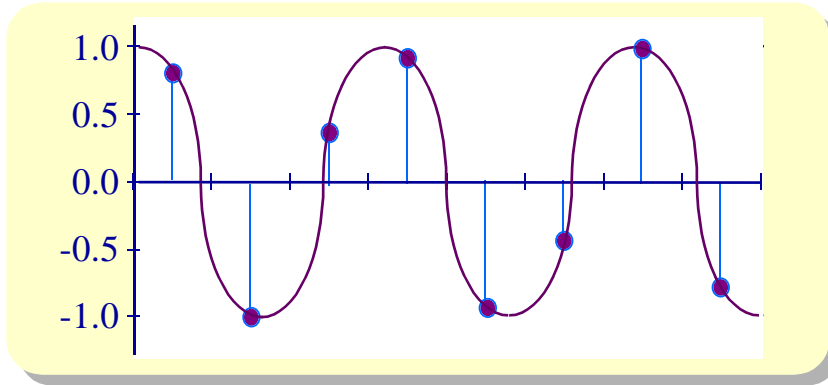
- ◆ Represent the signal in terms of a set of *cosine basis functions*



24

Transform-Based Compression

The Discrete Cosine Transform (DCT)



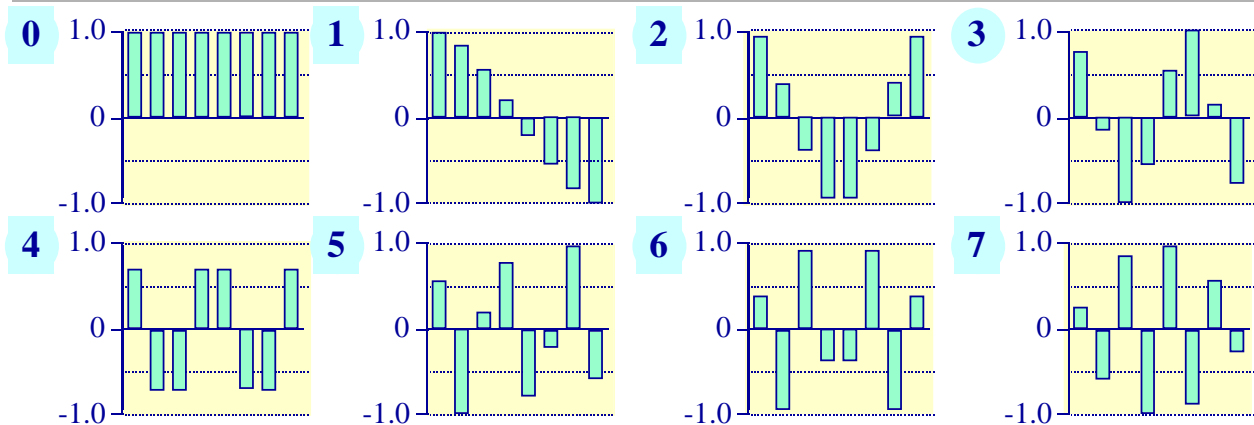
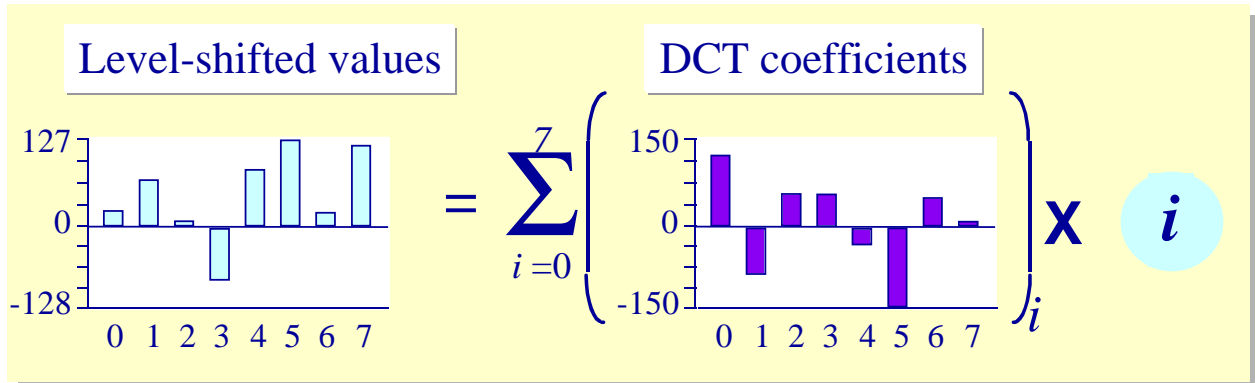
Sampled 2.5 Hz cosine wave

- ◆ The basis functions derive from sampling cosine functions of increasing frequency
 - » From 0-3.5 Hz
 - » Basis functions sampled at 8 discrete points

25

The Discrete Cosine Transform

Represent input as a sum of scaled basis functions



26

Transform-Based Compression

The Discrete Cosine Transform (DCT)

- ◆ The 1-dimensional transform:

$$F(\mu) = \frac{C(\mu)}{2} \sum_{x=1}^7 f(x) \cos \frac{(2x+1)\mu\pi}{16}$$

- » $F(\mu)$ is the DCT coefficient for $\mu = 0..7$
- » $f(x)$ is the x^{th} input sample for $x = 0..7$
- » $C(\mu)$ is a constant (equal to $2^{-0.5}$ if $\mu = 0$ and 1 otherwise)

- ◆ The 2-dimensional (spatial) transform:

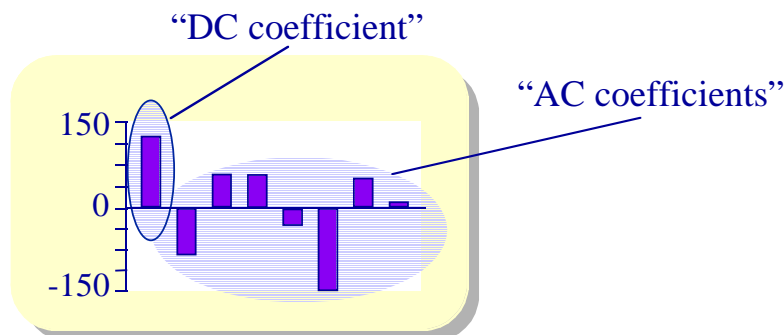
$$F(\mu, \nu) = \frac{C(\mu)C(\nu)}{2} \sum_{y=1}^7 \sum_{x=1}^7 f(x, y) \cos \frac{(2x+1)\mu\pi}{16} \cos \frac{(2y+1)\nu\pi}{16}$$

27

Transform-Based Compression

The Discrete Cosine Transform (DCT)

- ◆ DCT coefficients encode the spatial frequency of the input signal
 - » DC coefficient — zero spatial frequency (the “average” sample value)
 - » AC coefficients — higher spatial frequencies



- ◆ Claim: Higher frequency coefficients will be zero and can be ignored

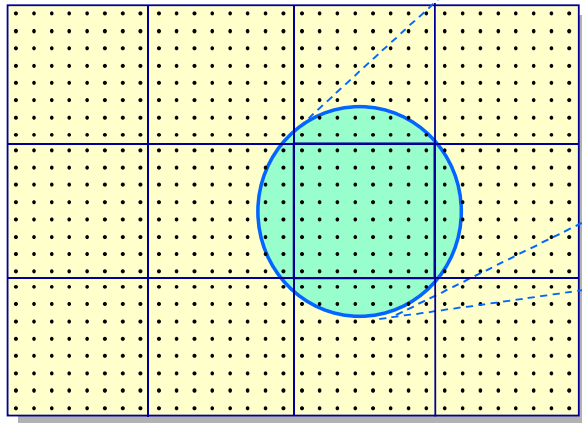
28

Transform-Based Compression

The two-dimensional DCT

- ◆ Apply the DCT in x and y dimensions simultaneously to 8×8 pixel blocks
 - » Code coefficients individually with fewer bits

Video Frame



172	-18	15	-8	23	-9	-14	19
21	-34	24	-8	-10	11	14	7
-9	-8	-4	6	-5	4	3	-1
-10	6	-5	4	-4	4	2	1
-8	-2	-3	5	-3	3	4	6
4	-2	-4	6	-4	4	2	-1
4	-3	-4	5	6	3	1	1
0	-8	-4	3	2	1	4	0

DCT Coefficients

29

Statistical Compression

Huffman coding

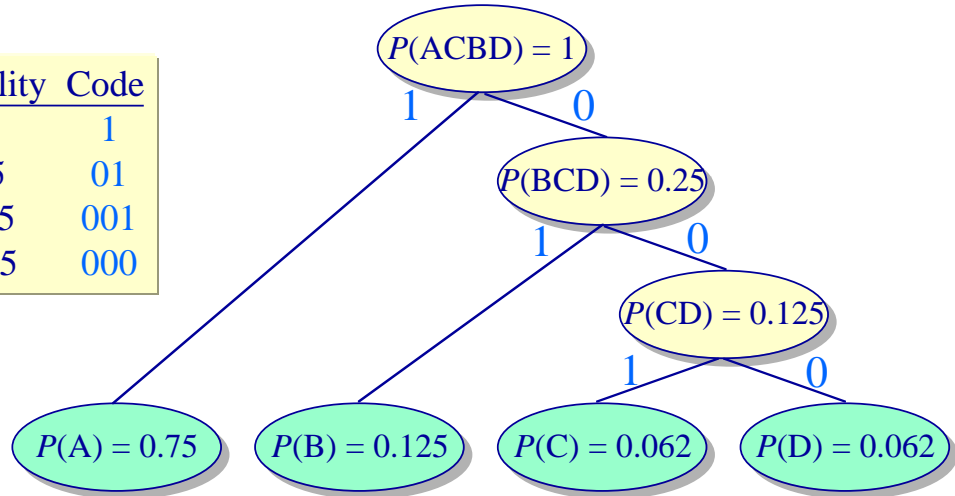
- ◆ Exploit the fact that not all sample values are equally likely
 - » Samples values are non-uniformly distributed
 - » Encode “common” values with fewer bits and less common values with more bits
- ◆ Process each image to determine the statistical distribution of sample values
 - » Generate a *codebook* — a table used by the decoder to interpret variable length codes
 - » Codebook becomes part of the compressed image

30

Statistical Compression

Huffman coding

Symbol	Probability	Code
A	0.75	1
B	0.125	01
C	0.0625	001
D	0.0625	000



- ◆ Order all possible sample values in a binary tree by combining the least likely samples into a sub-tree
- ◆ Label the branches of the tree with 1's and 0's
 - » Huffman code is the sequence of 1's and 0's on the path from the root to the leaf node for the symbol