Video signals

LOSSLESS CODING

Lossless coding

The goal of lossless image compression is to represent an image signal with the smallest possible number of bits without loss of *any* information, thereby speeding up transmission and minimizing storage requirements.

The number of bits representing the signal is typically expressed as an average bit rate (average number of bits per sample for still images, and average number of bits per second for video).

Lossy compression

The goal of lossy compression is to achieve the best possible fidelity given an available communication or storage bit-rate capacity, or to minimize the number of bits representing the image signal subject to some allowable loss of information.

In this way, a much greater reduction in bit rate can be attained as compared to lossless compression, which is necessary for enabling many real-time applications involving the handling and transmission of audiovisual information.

Why coding?

Coding techniques are crucial for the effective transmission or storage of data-intensive visual information.

In fact, a single uncompressed color image or video frame with a medium resolution of 500 x 500 pixels would require 100 s for transmission over an ISDN (Integrated Services Digital Network) link having a capacity of 64,000 bit& (64 Kbps).

The resulting delay is intolerably large, considering that a delay as small as 1-2 s is needed to conduct an interactive “slide show,” and a much smaller delay (of the order of 0.1 s) is required for video transmission or playback.

How lossless is possible?

Lossless compression is possible because, in general, there is significant redundancy present in image signals.

This redundancy is proportional to the amount of correlation among the image data samples.

For example, in a natural still image, there is usually a high degree of spatial correlation among neighboring image samples.

Also, for video, there is additional temporal correlation among samples in successive video frames.

In color images there is correlation, known as spectral correlation, between the image samples in the different spectral components.

Lossy vs lossless

In lossless coding, the decoded image data should be identical both quantitatively (numerically) and qualitatively (visually) to the original encoded image.

Although this requirement preserves exactly the accuracy of representation, it often severely limits the amount of compression that can be achieved to a compression factor of 2 or 3.

In order to achieve higher compression factors,perceptually lossless coding methods attempt to remove redundant as well as perceptually irrelevant information;

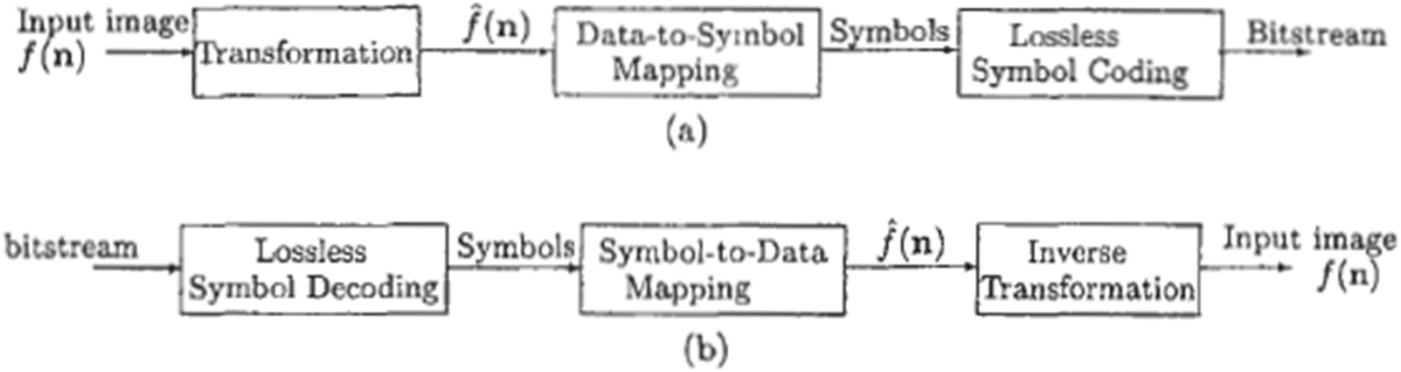
These methods require that the encoded and decoded images be only visually, and not necessarily numerically, identical.

So, why lossless?

Although a higher reduction in bit rate can be achieved with lossy compression, there exist several applications that require lossless coding, such as the compression of digital medical imagery and facsimile transmissions of bitonal images.

These applications triggered the development of several standards for lossless compression, including the lossless JPEG standard, facsimile compression standards and the JBIG compression standard.

Basics of Lossless Image Coding



The encoder (a) takes as input an image and generates as output a compressed bit stream.

The decoder (b) takes as input the compressed bit stream and recovers the original uncompressed image.

Different lossless approaches

Lossless compression is usually achieved by using variable length codewords, where the shorter codewords are assigned to the symbols that occur more frequently.

This variable-length codeword assignment is known as ***variable-length coding*** (VLC) and also as ***entropy coding.***

Entropy coders, such as Huffman and arithmetic coders, attempt to minimize the average bit rate (average number of bits per symbol) needed to represent a sequence of symbols, based on the probability of symbol occurrence.

An alternative way to achieve compression is to code ***variable-length strings*** of symbols using fixed-length binary codewords.

This is the basic strategy behind dictionary (Lempel-Ziv) codes.

Huffman coding

Huffman hit upon the idea of using a frequency-sorted binary tree and quickly proved this method the most efficient.

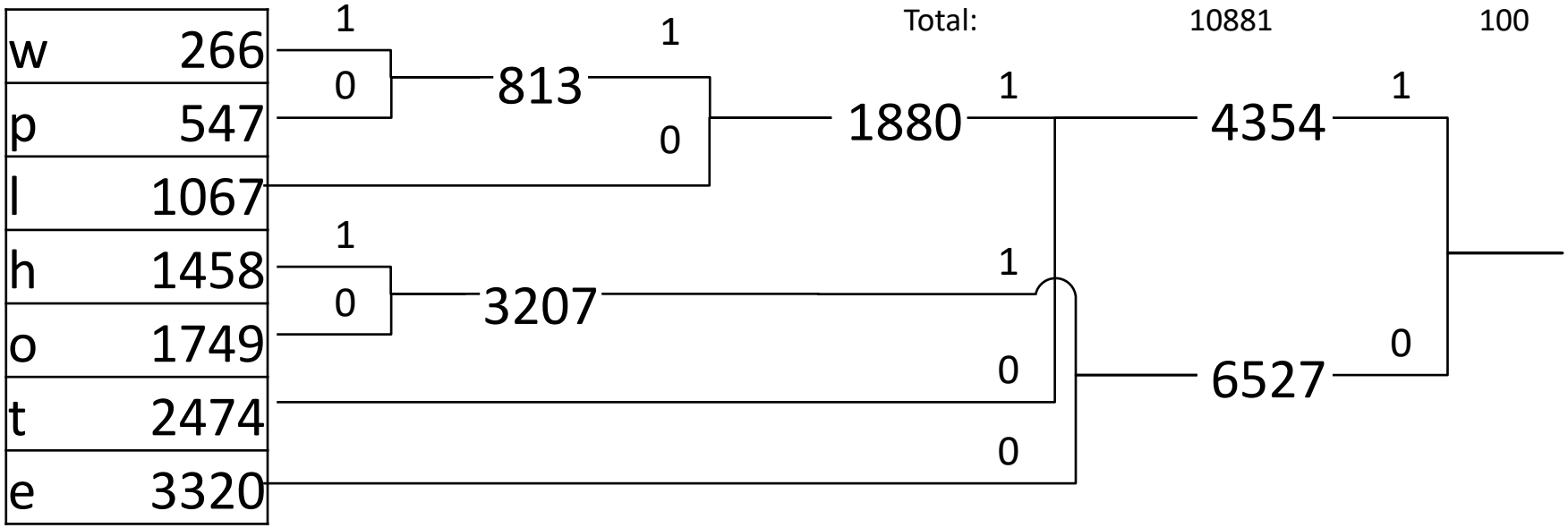
1. Take the two least probable symbols in the alphabet
   (longest codewords, equal length, differing in last digit)

2. Combine these two symbols into a single symbol, and repeat.

Huffman coding

Character	Binary Code
e	00
h	011
l	110
o	010
p	1110
t	10
w	1111

Character	Number of Occurrences (n)	Percentage (p)
e	3320	30.5119
h	1458	13.3995
l	1067	9.8061
o	1749	16.0739
p	547	5.0271
t	2474	22.7369
w	266	2.4446
Total:	10881	100



Huffman drawbacks

Huffman coding and arithmetic coding require *a priori* knowledge of the source symbol probabilities or of the source statistical model.

In some cases, a sufficiently accurate source model is difficult to obtain, especially when several types of data (such as text, graphics, and natural pictures) are intermixed.

Perceptually lossless coding

Perceptual-based algorithms attempt to discriminate between signal components that are and are not detected by the human receiver.

They exploit the spatiotemporal masking properties of the human visual system and establish thresholds of *justnoticeable distortion* based on psychophysical contrast masking phenomena.

The interest is in bandlimited signals because of the fact that visual perception is mediated by a collection of individual mechanisms in the visual cortex, denoted *channels* or *filters,* that are selective in terms of frequency and orientation.

Perceptually lossless coding

Neurons respond to stimuli above a certain contrast.

The necessary contrast to provoke a response from the neurons is defined as the *detection threshold.*

The inverse of the detection threshold is the *contrast sensitivity.*

Contrast sensitivity varies with frequency (including spatial frequency, temporal frequency, and orientation) and can be measured using *detection experiments.*

Perceptually lossless

Perceptually lossless image compression (a) Original Lena image, 8 bpp; (b) decoded Lena image, 0.361 bpp. The perceptual thresholds are computed for a viewing distance equal to *6* times the image height.

JPEG CODING

JPEG Image Compression

JPEG: Joint Photographic Experts Group.

JPEG: compression *algorithm*, NOT a file format. The original JPEG format was JFIF and later SPIFF. Current popular choices include C-Cube JFIF; Adobe TIFF/JPEG.

Key features

Both sequential and progressive modes of encoding are permitted.These modes refer to the manner in which quantized DCT coefficients are encoded. In sequential coding, the coefficients are encoded on a block-by-block basis in a single scan that proceeds from left to right and top to bottom.

In contrast, in progressive encoding only partial information about the coefficients is encoded in the first scan followed by encoding the residual information in successive scans.

Low complexity implementations in both hardware and software are feasible.

Key features

All types of images, regardless of source, content, resolution, color formats, etc., are permitted.

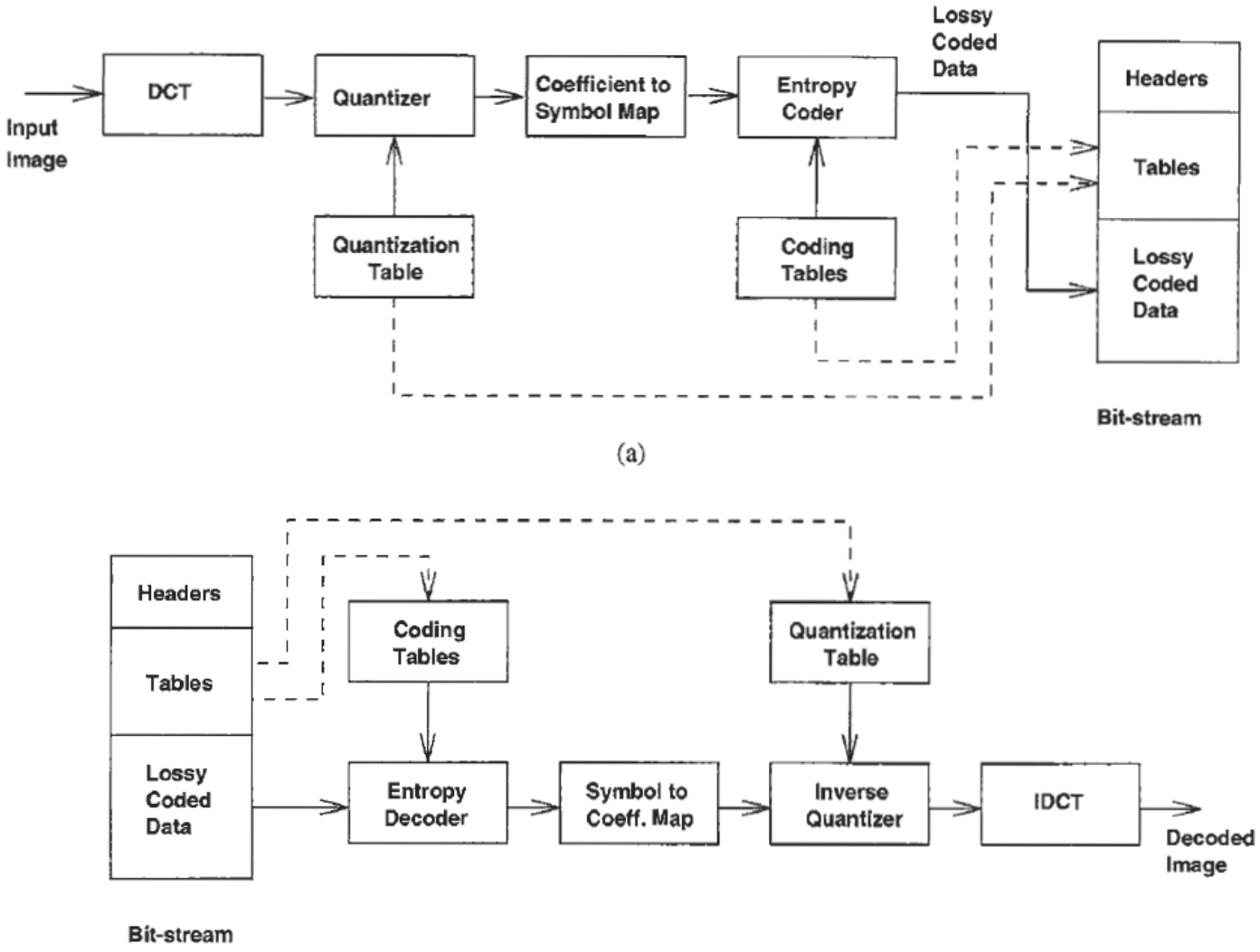
A graceful tradeoff in bit rate and quality is offered, except at very low bit rates.

A hierarchical mode with multiple levels of resolution is allowed.

Bit resolution of 8-12 bits is permitted.

A recommended file format, JPEG File Interchange Format (JFIF), enables the exchange of JPEG bit streams among a variety of platforms.

Gray scale codec scheme



Steps in JPEG Compression

1. (Optionally) If the color is represented in RGB mode, translate it to YUV.

2. Divide the file into 8 X 8 blocks.

3. Transform the pixel information from the spatial domain to the frequency domain with the Discrete Cosine Transform.

4. Quantize the resulting values by dividing each coefficient by an integer value and rounding off to the nearest integer.

5. Look at the resulting coefficients in a zigzag order. Do a run-length encoding of the coefficients ordered in this manner. Follow by Huffman coding.

Step 1a: Converting RGB to YUV

YUV color mode stores color in terms of its luminance (brightness) and chrominance (hue).

The human eye is less sensitive to chrominance than luminance.

YUV is not required for JPEG compression, but it gives a better compression rate.

RGB vs. YUV

It's simple arithmetic to convert RGB to YUV. The formula is based on the relative contributions that red, green, and blue make to the luminance and chrominance factors.

There are several different formulas in use depending on the target monitor. For example:

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

$$U = -0.1687 * R - 0.3313 * G + 0.5 * B + 128$$

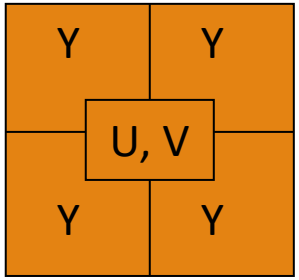
$$V = 0.5 * R - 0.4187 * G - 0.813 * B + 128$$

Step 1b: Downsampling

The chrominance information can (optionally) be downsampled.

The notation 4:1:1 means that for each block of four pixels, you have 4 samples of luminance information (Y), and 1 each of the two chrominance components (U and V).

MCU – minimum coded unit



Step 2: Divide into 8 X 8 blocks

Note that with YUV color, you have 16 pixels of information in each block for the Y component (though only 8 in each direction for the U and V components).

If the file doesn't divide evenly into 8 X 8 blocks, extra pixels are added to the end and discarded after the compression.

The values are shifted "left" by subtracting 128.

(See JPEG Compression for details.)

Step 3: Apply the DCT transform

In DCT coding, each component **of** the image is subdivided into blocks of 8 x 8 pixels.

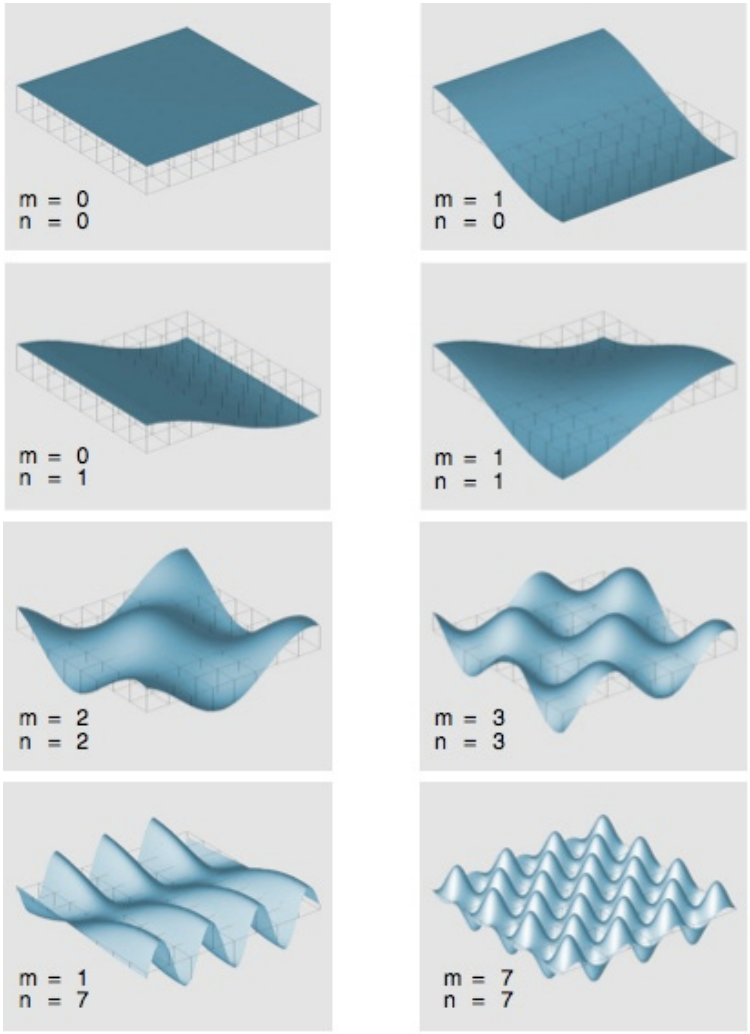
A two-dimensional DCT is applied to each block of data to obtain an 8 x 8 array of coefficients.

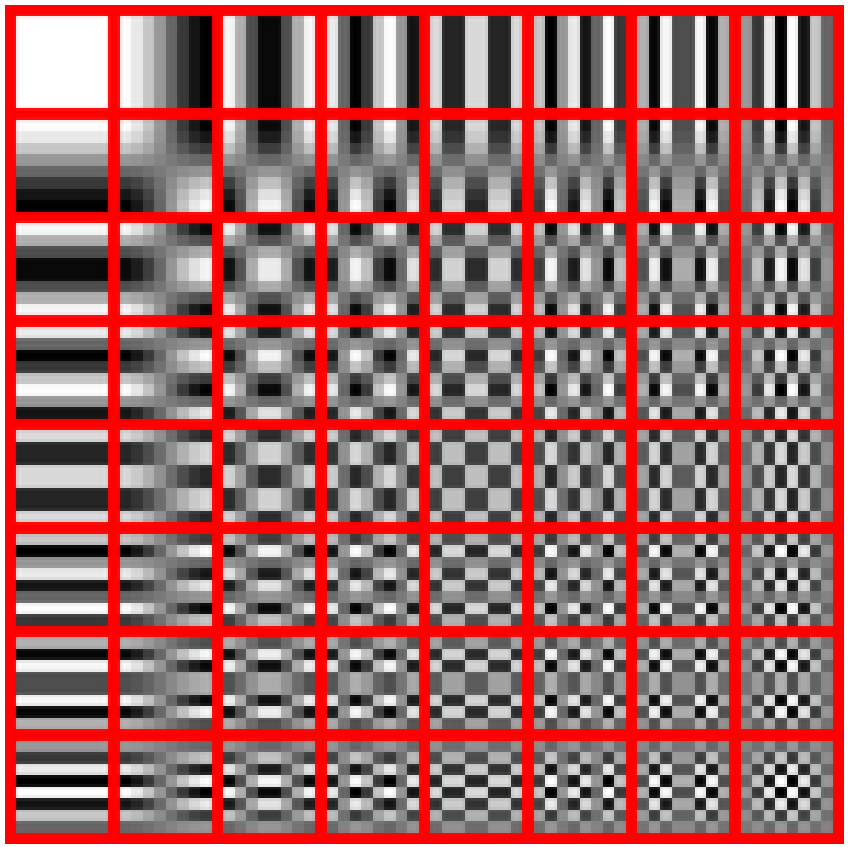
If $\boldsymbol{x}[m, n]$ represents the image pixel values in a block, then the DCT is computed for each block of the image data as follows:

$$X[u, v] = \frac{C[u]C[v]}{4} \sum_{m=0}^{7} \sum_{n=0}^{7} x[m, n] \cos \frac{(2m+1)u\pi}{16} \times \cos \frac{(2n+1)v\pi}{16}, \quad 0 \le u, v \le 7,$$

$$C[u] = \begin{cases} \frac{1}{\sqrt{2}} & u = 0, \\ 1 & 1 \le u \le 7. \end{cases}$$

DCT representation





Mathematical representation

IDCT

$$x[m, n] = \sum_{u=0}^{7} \sum_{v=0}^{7} \frac{C[u]C[v]}{4} X[u, v] \cos \frac{(2m+1)u\pi}{16} \times \cos \frac{(2n+1)v\pi}{16}, \quad 0 \leq m, n \leq 7.$$

Step 4: Quantize the Coefficients Computed by the DCT

The DCT is lossless in that the reverse DCT will give you back exactly your initial information (ignoring the rounding error that results from using floating point numbers.)

The values from the DCT are initially floating-point.

They are changed to integers by quantization.

Step 4: Quantization

Quantization involves dividing each coefficient by an integer between 1 and 255 and rounding off.

The quantization table is chosen to reduce the precision of each coefficient to no more than necessary.

The quantization table is carried along with the compressed file.

Quantizer

If we wish to recover the original image exactly from the DCT coefficient array, then it is necessary to represent the DCT coefficients with high precision.

Such a representation requires a large number of bits.

In lossy compression the DCT coefficients are mapped into a relatively small set of possible values that are represented compactly by defining and coding suitable symbols.

The quantization unit performs this task of a many-to-one mapping of the DCT coefficients, so that the possible outputs are limited in number.

A key feature of the quantized DCT coefficients is that many of them are zero, making them suitable for efficient coding.

Coefficient-to-Symbol Mapping Unit

The quantized DCT coefficients are mapped to new symbols to facilitate a compact representation in the symbol coding unit that follows.

The symbol definition unit can also be viewed as part of the symbol coding unit. However, it is shown here as a separate unit to emphasize the fact that the definition of symbols to be coded is an important task.

An effective definition of symbols for representing AC coefficients in JPEG is the “runs” of zero coefficients followed by a nonzero terminating coefficient.

For representing DC coefficients, symbols are defined by computing the difference between the DC coefficient in the current block and that in the previous block.

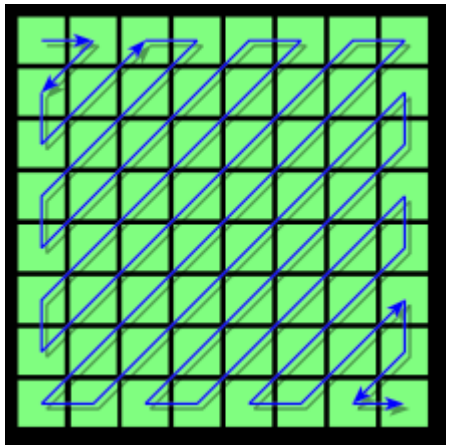
Step 5: Arrange in “zigzag” order

This is done so that the coefficients are in order of increasing frequency.

The higher frequency coefficients are more likely to be 0 after quantization.

This improves the compression of run-length encoding.

Do run-length encoding and Huffman coding.



JPEG vs. GIF

Color depth:

JPEG stores full color info (24 bits/pixel).

GIF stores only 8 bits/pixel (intended for inexpensive color displays)

Edge preservation:

JPEG tends to blur sharp edges (esp. in high compression)

GIF does very well with ‘graphics’ images (e.g. line drawings)

JPEG vs. GIF (cont.)

B&W imagery:

JPEG is not suitable for two-tone imagery

GIF is lossless for gray scale images (up to 256 gray values)

Compression performance

JPEG:

- 10:1 to 20:1 compression without visible loss (effective storage requirement drops to 1-2 bits/pixel)
- 30:1 to 50:1 compression with small to moderate visual deterioration
- 100:1 compression for low quality applications

GIF:

- 3:1 compression by reducing color space to 8 bits
- LZW coding may improve compression up to 5:1

BLOCK TRUNCATION CODE

Block Truncation Coding

Statistical and structural methods have been developed for image compression

the former being based on the principles of source coding with emphasis on the algebraic structure of the pixels in an image, whereas the latter methods exploit the geometric structure of the image.

Basics of BTC

The basic BTC algorithm is a lossy fixed length compression method that uses a Q-level quantizer to quantize a local region of the image.

The quantizer levels are chosen such that a number of the moments of a local region in the image are preserved in the quantized output.

In its simplest form, the objective of BTC is to preserve the sample mean and sample standard deviation of a gray-scale image.

Additional constraints can be added to preserve higher-order moments. For this reason BTC is a block adaptive moment preserving quantizer.

BTC

The first step of the algorithm is to divide the image into nonoverlapping rectangular regions. For the sake of simplicity we let the blocks be square regions of size *n* x *n,* where *n* is typically **4.**

For a two-level (1 bit) quantizer, the idea is to select two luminance values to represent each pixel in the block.

These values are chosen such that the sample mean and standard deviation of the reconstructed block are identical to those of the original block.

BTC

An *n* x *n* bit map is then used to determine weather a pixel luminance value is above or below a certain threshold.

In order to illustrate how BTC works, we will let the sample mean of the block be the threshold;

- a “1” would then indicate if an original pixel value is above this threshold, and
- *“0”* if it is below.

Since BTC produces a bit map to represent a block, it is classified as a binary pattern image coding method

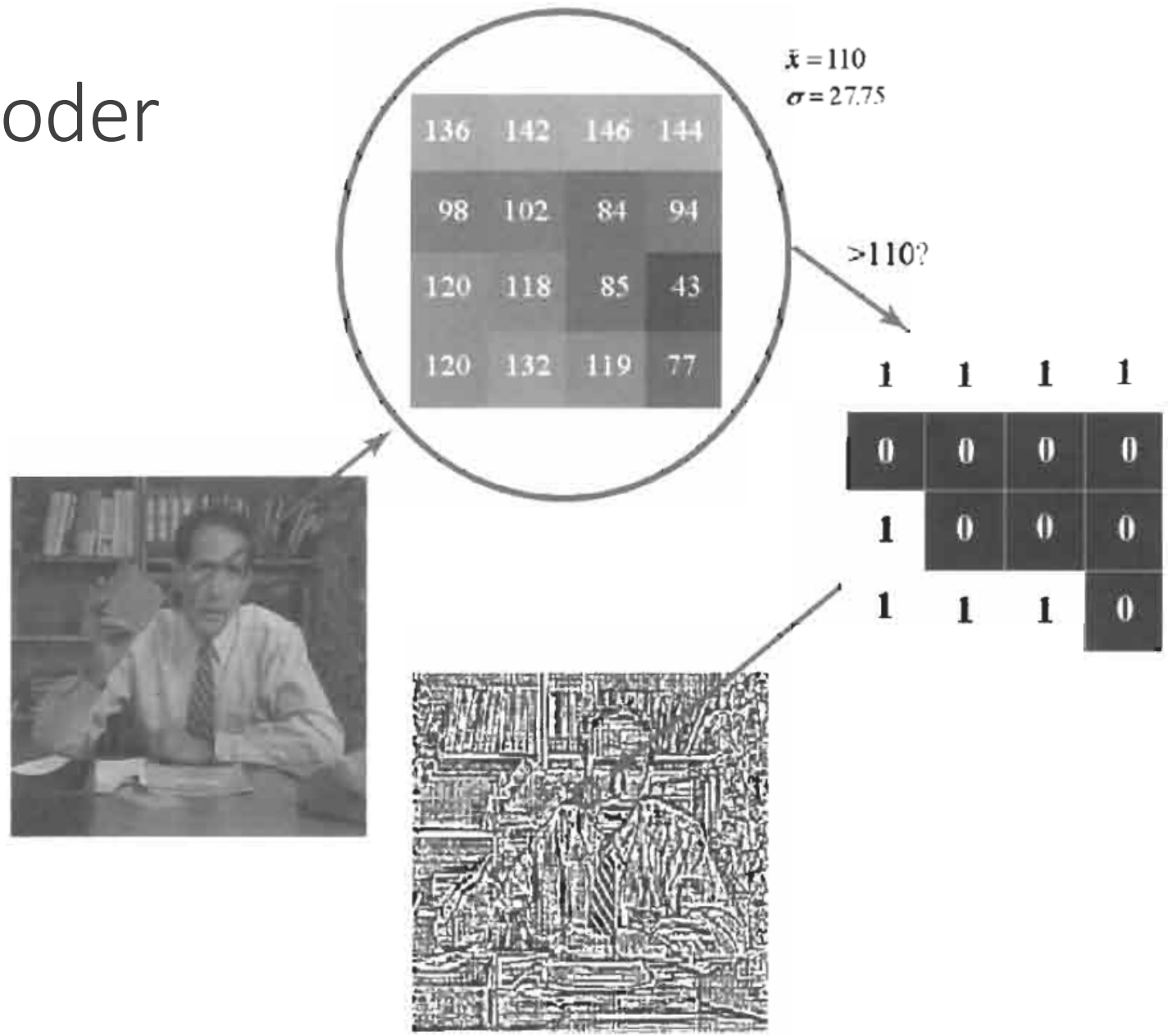
BTC

By knowing the bit map for each block, the decompression/reconstruction algorithm knows whether a pixel is brighter or darker than the average.

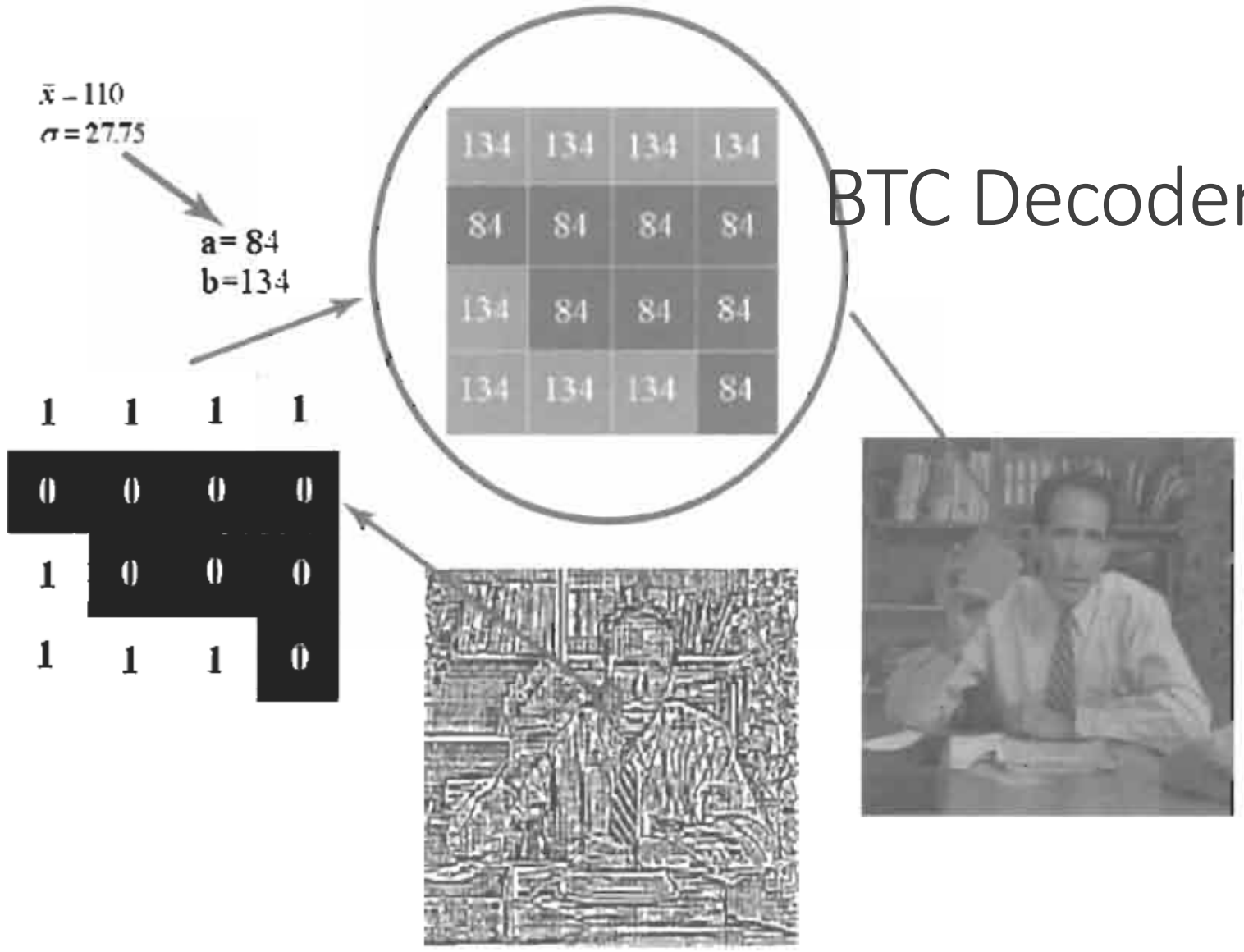
Thus, for each block two gray-scale values, ***a*** and ***b,*** are needed to represent the two regions.

- These are obtained from the sample mean and sample standard deviation of the block, and
- they are stored together with the bit map.

BTC encoder


$\bar{x} = 110$
$\sigma = 27.75$
136 142 146 144
98 102 84 94
120 118 85 43
120 132 119 77
>110?
1 1 1 1
0 0 0 0
1 0 0 0
1 1 1 0

BTC Decoder


$\bar{x}$ – 110
σ = 27.75
a= 84
b=134
134 134 134 134
84 84 84 84
134 84 84 84
134 134 134 84
1 1 1 1
0 0 0 0
1 0 0 0
1 1 1 0

BTC

the image was compressed from 8 bits per pixel to **2** bits per pixel (bpp).

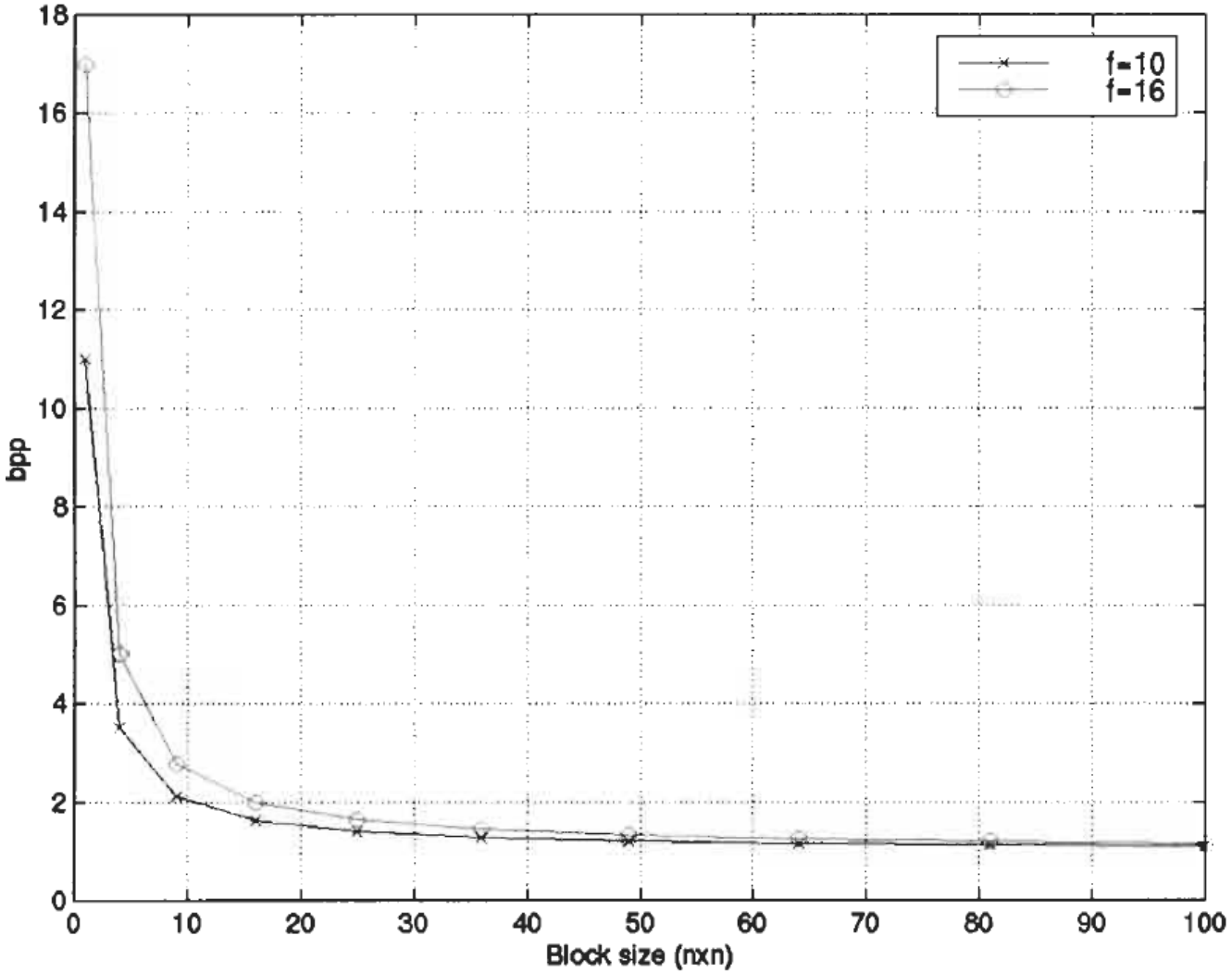
This is done because BTC requires **16** bits for the bit map, 8 bits for the sample mean, and 8 bits for the sample standard deviation.

Thus, the entire **4** x **4** block requires **32** bits, and hence the data

rate is **2** bpp.

From this example it is easy to understand how a smaller data rate can be achieved by selecting a bigger block size, or by allocating fewer bits for the sample mean or the sample standard deviation

Data rate vs. block size.



Moments preservation in the output

$$m_1 = \bar{x} = \frac{1}{k}\sum_{i=1}^{k} x_i,$$

$$m_2 = \frac{1}{k}\sum_{i=1}^{k} x_i^2,$$

$$\sigma^2 = m_2 - m_1^2.$$

$$\longrightarrow$$

$$km_1 = (k-q)a + qb$$

$$km_2 = (k-q)a^2 + qb^2.$$

$$\downarrow$$

$$a = m_1 - \sigma\sqrt{\frac{q}{k-q}},$$

$$b = m_1 + \sigma\sqrt{\frac{k-q}{q}}.$$

$$m_3 = \frac{1}{k}\sum_{i=1}^{k} x_i^3. \longrightarrow km_3 = (k-q)a^3 + qb^3$$

BTC compression

(a) (b) (c)

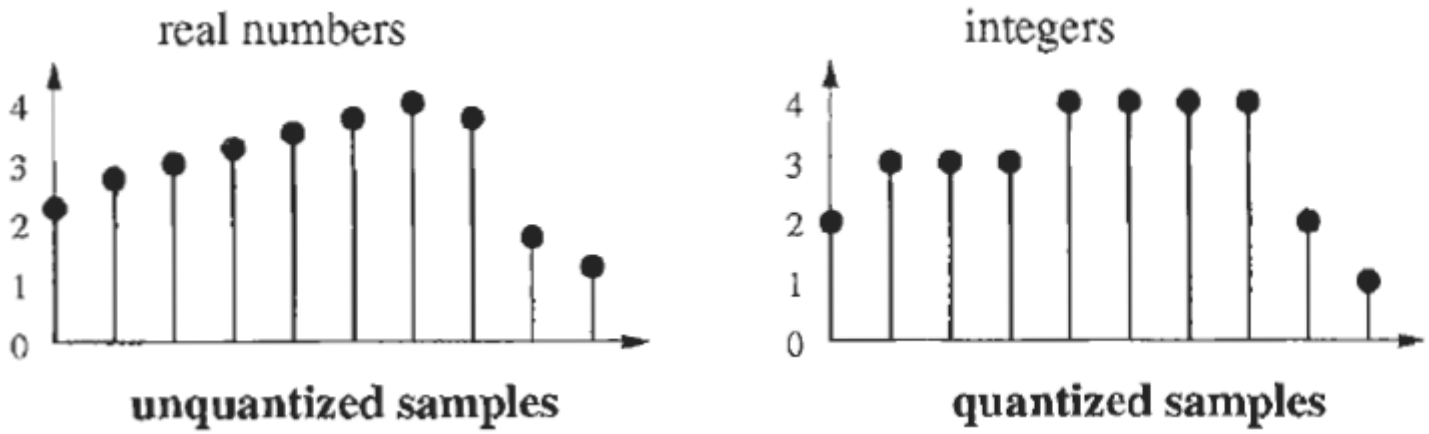
FIGURE 5 BTC with errors: (a) original image; (b) image compressed to 1.625 bpp; (c) performance of BTC in the presence of channel errors.

VECTOR QUANTIZATION

Quantization

Quantization is a field of study that has matured over the past few decades.

In simplest terms, quantization is a mapping of a large set of values to a smaller set of values. The concept is here illustrated



It shows on the left a sequence of unquantized samples with **am**plitudes assumed to be of infinite precision, and on the right that same sequence quantized to integer values.

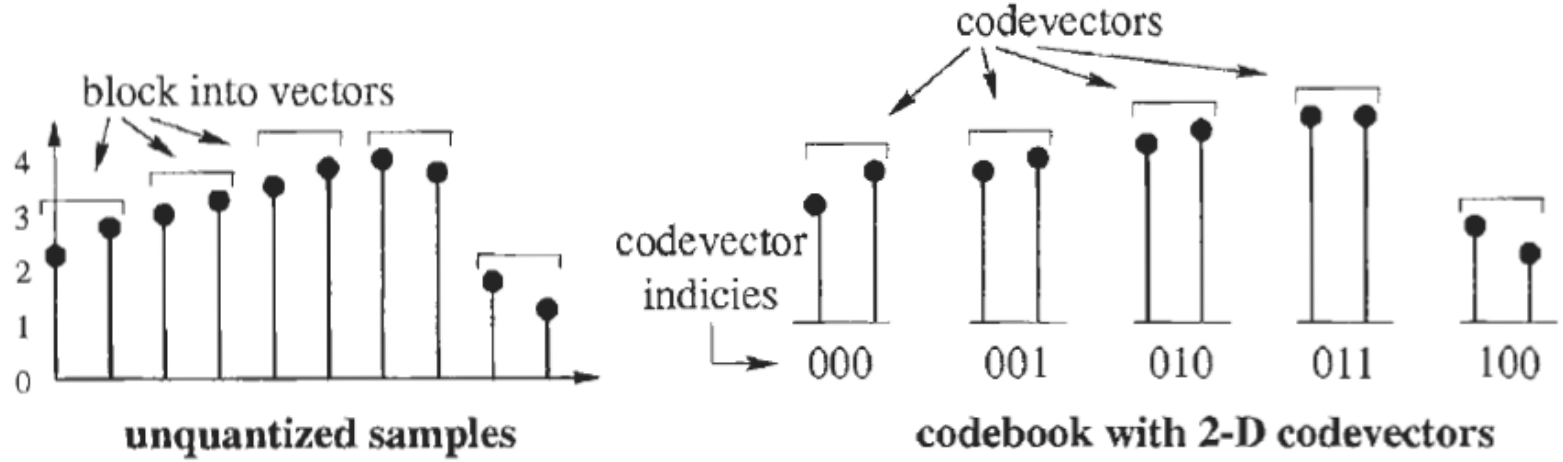
Vector quantization

Obviously, quantization is an irreversible process, since it involves discarding information.

If it is done wisely, the error introduced by the process can be held to a minimum.

The generalization of this notion is called ***vector quantization,*** commonly denoted VQ.

It too is a mapping from a large set to a smaller set, but it involves quantizing blocks of samples together.



Applications

The general concept of VQ can be applied to any type of digital data.

For a one-dimensional signal as illustrated in previous slide vectors can be formed by extracting contiguous blocks from the sequence.

For two-dimensional signals (i.e, digital images) vectors *can* be formed by taking 2-D blocks, such as rectangular blocks, and unwrapping them to form vectors.

Similarly, the same idea can be applied to 3-D data (i.e., video), color and multispectral data, transform coefficients, and so on.

Encoder and Decoder