Chapter 10
Image Compression

Introduction and Overview

- The field of image compression continues to grow at a rapid pace
- As we look to the future, the need to store and transmit images will only continue to increase faster than the available capability to process all the data

- ✔ Applications that require image compression are many and varied such as:

1. Internet,
2. Businesses,
3. Multimedia,
4. Satellite imaging,
5. Medical imaging

- Compression algorithm development starts with applications to two-dimensional (2-D) still images
- After the 2-D methods are developed, they are often extended to video (motion imaging)
- However, we will focus on image compression of single frames of image data

- *Image compression* involves reducing the size of image data files, while *retaining necessary information*
- Retaining necessary information depends upon the application
- Image segmentation methods, which are primarily a data reduction process, can be used for compression

- The reduced file created by the compression process is called the *compressed file* and is used to reconstruct the image, resulting in the *decompressed image*
- The original image, before any compression is performed, is called the *uncompressed image file*
- The ratio of the original, uncompressed image file and the compressed file is referred to as the *compression ratio*

✔ The compression ratio is denoted by:

$$Compression\ Ratio = \frac{Uncompressed\ file\ size}{Compressed\ file\ size} = \frac{SIZE_U}{SIZE_C}\ ;\ Often\ written\ as \rightarrow SIZE_U : SIZE_C$$

EXAMPLE 10.1.1: The original image is 256x256 pixels, single-band (grayscale), 8-bits per pixel. This file is 65,536 bytes (64k). After compression the image file is 6,554 bytes. The compression ratio is: $SIZE_U/SIZE_C$ = 65536/6554 = 9.999 $\approx$ 10. This can also be written as 10:1

This is called a "10 to 1 compression", a "10 times compression", or can be stated as "compressing the image to 1/10 its original size". Another way to state the compression is to use the terminology of *bits per pixel*. For an NxN image:

$$Bits\ per\ pixel = \frac{Number\ of\ bits}{Number\ of\ pixels} = \frac{(8)(Number\ of\ bytes)}{NxN}$$

EXAMPLE 10.1.2: Using the preceding example, with a compression ratio of 65,536/6,554 bytes, we want to express this as bits per pixel. This is done by first finding the number of pixels in the image: 256x256 = 65,536 pixels. We then find the number of bits in the compressed image file: (6,554 bytes)(8 bits/byte) = 52,432 bits. Now we can find the bits per pixel by taking the ratio: 52,432/65,536 = 0.8 bits/pixel

- The reduction in file size is necessary to meet the bandwidth requirements for many transmission systems, and for the storage requirements in computer databases
- Also, the amount of data required for digital images is enormous

EXAMPLE 10.1.3:

To transmit an RGB (color) 512x512, 24-bit (8-bit per pixel per color) image via modem at 56 kbaud (kilo-bits per second), it would take about:

$$\frac{(512 \times 512 \text{ pixels})(24 \text{ bits/pixel})}{(56 \times 1024 \text{ bits/second})} \approx 109 \text{ seconds} \approx 1.8 \text{ minutes}$$

- This number is based on the actual transmission rate being the maximum, which is typically not the case due to Internet traffic, overhead bits and transmission errors

- Additionally, considering that a web page might contain more than one of these images, the time it takes is simply too long
- For high quality images the required resolution can be much higher than the previous example

EXAMPLE 10.1.4:

To transmit a digitized color 35mm slide scanned at 4000x3000 pixels, and 24-bits, at 56 kbaud would take about:

$$\frac{(4000 \times 3000 pixels)(24 bits/pixel)}{(56 \times 1024 bits/\text{sec})} \approx 5022 \text{sec} \approx 83 \text{min}$$, too long to wait!

Example 10.1.5 applies maximum data rate to Example 10.1.4

EXAMPLE 10.1.5:

To transmit a digitized color 35mm slide scanned at 4000x3000 pixels, and 24-bits, at 3 Mbps would take about:

$$\frac{(4000 \text{ x } 3000 \text{ pixels})(24 \text{ bits/pixel})}{(3 \text{ x } 1024 \text{ x} 1024 \text{ bits/second})} \approx 91 \text{ seconds} \approx 1.5 \text{ minutes}$$

- Now, consider the transmission of video images, where we need multiple frames per second
- If we consider just one second of video data that has been digitized at 640x480 pixels per frame, and requiring 15 frames per second for interlaced video, then:

EXAMPLE 10.1.6: To transmit one second of interlaced video that has been digitized at 640x480 pixels:

$$\frac{(640 \times 480 \times 15\, frames / \mathbf{sec})(24 bits / pixel)}{3 \times 1024 \times 1024 bits / \mathbf{sec})} \approx 35\, \mathbf{sec}\, onds$$

- Waiting 35 seconds for one second's worth of video is not exactly real time!
- Even attempting to transmit uncompressed video over the highest speed Internet connection is impractical
- For example: The Japanese Advanced Earth Observing Satellite (ADEOS) transmits image data at the rate of 120 Mbps

- Applications requiring high speed connections such as high definition television, real-time teleconferencing, and transmission of multiband high resolution satellite images, leads us to the conclusion that *image compression is not only desirable but necessessary*

- *Key to a successful compression scheme* is *retaining necessary information*

- ✔ To understand "*retaining necessary information*", we must differentiate between *data* and *information*

1. *Data*:

- For digital images, *data* refers to the pixel gray level values that correspond to the brightness of a pixel at a point in space
- Data are used to convey information, much like the way the alphabet is used to convey information via words

2. *Information:*

- Information is an interpretation of the data in a meaningful way
- Information is an elusive concept; it can be application specific

- ✔ There are two primary types of image compression methods:

1. *Lossless compression methods:*

- Allows for the exact recreation of the original image data, and can compress complex images to a maximum 1/2 to 1/3 the original size – 2:1 to 3:1 compression ratios
- Preserves the data *exactly*

2. *Lossy compression methods:*

- Data loss, original image cannot be re-created exactly

- Can compress complex images 10:1 to 50:1 and retain high quality, and 100 to 200 times for lower quality, but acceptable, images

- Compression algorithms are developed by taking advantage of the redundancy that is inherent in image data

- Four primary types of redundancy that can be found in images are:

1. *Coding*
2. *Interpixel*
3. *Interband*
4. *Psychovisual redundancy*

1. *Coding redundancy*

- ✔ Occurs when the data used to represent the image is not utilized in an optimal manner

2. *Interpixel redundancy*

- ✔ Occurs because adjacent pixels tend to be highly correlated, in most images the brightness levels do not change rapidly, but change gradually

3. *Interband redundancy*

- ✔ Occurs in color images due to the correlation between bands within an image – if we extract the red, green and blue bands they look similar

4. *Psychovisual redundancy*

- ✔ Some information is more important to the human visual system than other types of information

- The key in image compression algorithm development is to determine the minimal data required to retain the necessary information
- The compression is achieved by taking advantage of the redundancy that exists in images
- If the redundancies are removed prior to compression, for example with a decorrelation process, a more effective compression can be achieved

- To help determine which information can be removed and which information is important, the *image fidelity criteria* are used
- These measures provide metrics for determining image quality
- It should be noted that the information required is application specific, and that, with lossless schemes, there is no need for a fidelity criteria

- Most of the compressed images shown in this chapter are generated with CVIPtools, which consists of code that has been developed for educational and research purposes
- The compressed images shown are not necessarily representative of the best commercial applications that use the techniques described, because the commercial compression algorithms are often *combinations* of the techniques described herein

✔ Compression System Model

- The compression system model consists of two parts:

1. The compressor
2. The decompressor

- The *compressor* consists of a preprocessing stage and encoding stage, whereas the *decompressor* consists of a decoding stage followed by a postprocessing stage

Figure 10.1-1: Compression System Model

a) Compression

b) Decompression

- Before encoding, preprocessing is performed to prepare the image for the encoding process, and consists of any number of operations that are application specific

- After the compressed file has been decoded, postprocessing can be performed to eliminate some of the potentially undesirable artifacts brought about by the compression process

- The compressor can be broken into following stages:

1. *Data reduction*: Image data can be reduced by gray level and/or spatial quantization, or can undergo any desired image improvement (for example, noise removal) process
2. *Mapping*: Involves mapping the original image data into another mathematical space where it is easier to compress the data

3. *Quantization*: Involves taking potentially continuous data from the mapping stage and putting it in discrete form

4. *Coding*: Involves mapping the discrete data from the quantizer onto a code in an optimal manner

- A compression algorithm may consist of all the stages, or it may consist of only one or two of the stages

Figure 10.1-2: The Compressor

- The decompressor can be broken down into following stages:

1. *Decoding*: Takes the compressed file and reverses the original coding by mapping the codes to the original, quantized values

2. *Inverse mapping*: Involves reversing the original mapping process

3. *Postprocessing*: Involves enhancing the look of the final image

- This may be done to reverse any preprocessing, for example, enlarging an image that was shrunk in the data reduction process
- In other cases the postprocessing may be used to simply enhance the image to ameliorate any artifacts from the compression process itself

Figure 10.1-3: The Decompressor

- The development of a compression algorithm is highly application specific
- Preprocessing stage of compression consists of processes such as enhancement, noise removal, or quantization are applied
- The goal of preprocessing is to prepare the image for the encoding process by eliminating any irrelevant information, where *irrelevant* is defined by the application

- For example, many images that are for viewing purposes only can be preprocessed by eliminating the lower bit planes, without losing any useful information

Figure 10.1.4 Bit plane images

a) Original image

b) Bit plane 7, the most significant bit

c) Bit plane 6

Figure 10.1.4 Bit plane images (Contd)

d) Bit plane 5

e) Bit plane 4

f) Bit plane 3

Figure 10.1.4 Bit plane images (Contd)

g) Bit plane 2

h) Bit plane 1

i) Bit plane 0,
the least significant bit

- The mapping process is important because image data tends to be highly correlated
- Specifically, if the value of one pixel is known, it is highly likely that the adjacent pixel value is similar
- By finding a mapping equation that decorrelates the data this type of data redundancy can be removed

- *Differential coding:* Method of reducing data redundancy, by finding the difference between adjacent pixels and encoding those values

- The *principal components transform* can also be used, which provides a theoretically optimal decorrelation

- *Color transforms* are used to decorrelate data between image bands

Figure -5.6.1 Principal Components Transform (PCT)

a) Red band of a color image

b) Green band

c) Blue band

d) Principal component band 1

e) Principal component band 2

f) Principal component band 3

(c) Scott E Umbaugh,
SIUE 2005

- As the spectral domain can also be used for image compression, so the first stage may include mapping into the frequency or sequency domain where the energy in the image is compacted into primarily the lower frequency/sequency components

- These methods are all *reversible*, that is information preserving, although all mapping methods are not reversible

- *Quantization* may be necessary to convert the data into digital form (BYTE data type), depending on the mapping equation used

- This is because many of these mapping methods will result in floating point data which requires multiple bytes for representation which is not very efficient, if the goal is data reduction

- Quantization can be performed in the following ways:

1. *Uniform quantization:* In it, all the quanta, or subdivisions into which the range is divided, are of equal width

2. *Nonuniform quantization:* In it the quantization bins are not all of equal width

Figure 3.2-18: QUANTIZATION BINS

a) Uniform quantization bins: all bins are the same width. Values that fall within the same bin can be mapped to the low end(1), high end(2), or the middle(3).

b) Variable quantization bins are of different widths.

- Often, nonuniform quantization bins are designed to take advantage of the response of the human visual system

- In the spectral domain, the higher frequencies may also be quantized with wider bins because we are more sensitive to lower and midrange spatial frequencies and most images have little energy at high frequencies

- The concept of nonuniform quantization bin sizes is also described as a *variable bit rate*, since the wider quantization bins imply fewer bits to encode, while the smaller bins need more bits

- It is important to note that the quantization process is not reversible, so it is not in the decompression model and also some information may be lost during quantization

- The coder in the coding stage provides a one-to-one mapping, each input is mapped to a unique output by the coder, so it is a reversible process

- The code can be an *equal length code,* where all the code words are the same size, or an *unequal length code* with variable length code words

- In most cases, an unequal length code is the most efficient for data compression, but requires more overhead in the coding and decoding stages

LOSSLESS COMPRESSION METHODS

- ✔ No loss of data, decompressed image exactly same as uncompressed image
- ✔ Medical images or any images used in courts
- ✔ Lossless compression methods typically provide about a 10% reduction in file size for complex images

- Lossless compression methods can provide substantial compression for simple images
- However, lossless compression techniques may be used for both preprocessing and postprocessing in image compression algorithms to obtain the extra 10% compression

- The underlying theory for lossless compression (also called *data compaction*) comes from the area of communications and information theory, with a mathematical basis in probability theory

- One of the most important concepts used is the idea of *information content* and *randomness in data*

- ✔ *Information theory* defines information based on the probability of an event, knowledge of an unlikely event has more information than knowledge of a likely event
- ✔ For example:
  - The earth will continue to revolve around the sun; little information, 100% probability
  - An earthquake will occur tomorrow; more info. Less than 100% probability
  - A matter transporter will be invented in the next 10 years; highly unlikely – low probability, high information content

- This perspective on information is the *information theoretic definition* and should not be confused with our working definition that requires information in images to be useful, not simply novel

- *Entropy* is the measurement of the average information in an image

- The entropy for an *N x N* image can be calculated by this equation:

$$Entropy = -\sum_{i=0}^{L-1} p_i \log_2(p_i) \quad (in\ bits/pixel)$$

$$Where\ p_i = the\ probability\ of\ the\ i^{th}\ gray\ level = \frac{n_k}{N^2}$$

$$n_k = the\ total\ number\ of\ pixels\ with\ gray\ value\ k$$

$$L = the\ total\ number\ of\ gray\ levels\ (e.g.\ 256\ for\ 8\text{-}bits)$$

- This measure provides us with a theoretical minimum for the average number of bits per pixel that could be used to code the image

- It can also be used as a metric for judging the success of a coding scheme, as it is theoretically optimal

EXAMPLE 10.2.1:

Let L = 8, meaning there are 3 bits/pixel in the original image. Now, let's say the number of pixels at each gray level value is equal (they have the same probability), that is:

$$p_0 = p_1 = \ldots = p_7 = \frac{1}{8}$$

Now, we can calculate the entropy as follows:

$$Entropy = -\sum_{i=0}^{7} p_i \log_2(p_i) = -\sum_{i=0}^{7} \frac{1}{8} \log_2\left(\frac{1}{8}\right) = 3$$

This tells us that the theoretical minimum for lossless coding for this image is 3 bits per pixel. In other words, there is no code that will provide better results than the one currently used (called the natural code, since $000_2 = 0, 001_2 = 1, 010_2 = 2, \ldots, 111_2 = 7$). This example illustrates that the image with the most random distribution of gray levels, a uniform distribution, has the highest entropy

EXAMPLE 10.2.2:

Let = 8, thus we have a natural code with 3 bits per pixel in the original image. Now lets say that the entire image has a gray level of 2, so:

$$p_2 = 1, and\ p_0 = p_1 = p_3 = p_4 = p_5 = p_6 = p_7 = 0$$

And the entropy is:

$$Entropy = -\sum_{i=0}^{7} p_i \log_2(p_i) = -(1)\log_2(1) + 0 + \ldots + 0 = 0$$

This tells us the theoretical minimum for coding this image is 0 bits per pixel. Why is this? – Because the gray level value is known to be 2. To code the entire image we need only one value, this is called the certain event, it has a probability of 1

- The two preceding examples (10.2.1 and 10.2.2) illustrate the range of the entropy:

$$0 \le Entropy \le \log_2(L)$$

- The examples also illustrate the information theory perspective regarding information and randomness
- The more randomness that exists in an image, the more evenly distributed the gray levels, and more bits per pixel are required to represent the data

Figure 10.2-1 Entropy

a) Original image,
entropy = 7.032 *bpp*

b) Image after local histogram equalization,
block size 4, entropy = 4.348 *bpp*

c) Image after binary threshold,
entropy = 0.976 *bpp*

(c) Scott E Umbaugh,
SIUE 2005

Figure 10.2-1 Entropy (contd)

d) Circle with a radius of 32,
entropy = 0.283 *bpp*

f) Circle with a radius of 32,
and a linear blur radius of 64,
entropy = 2.030 *bpp*

e) Circle with a radius of 64,
entropy = 0.716 *bpp*

- Figure 10.2.1 depicts that a minimum overall file size will be achieved if a smaller number of bits is used to code the most frequent gray levels
- Average number of bits per pixel (*L*ength) in a coder can be measured by the following equation:

$$L_{ave} = \sum_{i=0}^{L-1} l_i \; p_i$$

$$\text{Where } l_i = \text{length in bits of the code for } i^{th} \text{ gray level}$$

$$p_i = \text{histogram-probability of } i^{th} \text{ gray level}$$

✔ Huffman Coding

- The Huffman code, developed by D. Huffman in 1952, is a *minimum length code*
- This means that given the statistical distribution of the gray levels (the histogram), the Huffman algorithm will generate a code that is as close as possible to the *minimum bound, the entropy*

- The method results in an *unequal* (or *variable*) *length code*, where the size of the code words can vary

- For complex images, Huffman coding alone will typically reduce the file by 10% to 50% (1.1:1 to 1.5:1), but this ratio can be improved to 2:1 or 3:1 by preprocessing for irrelevant information removal

- The Huffman algorithm can be described in five steps:

1. Find the gray level probabilities for the image by finding the histogram
2. Order the input probabilities (histogram magnitudes) from smallest to largest
3. Combine the smallest two by addition
4. GOTO step 2, until only two probabilities are left
5. By working backward along the tree, generate code by alternating assignment of 0 and 1

EXAMPLE 10.2.3:

We have an image with 2-bits per pixel, giving four possible gray levels. The image is 10 rows by 10 columns.

In Step 1 we find the histogram for the image. This is shown in Figure 10.2-2a, where we see that gray level 0 has 20 pixels, gray level 1 has 30 pixels, gray level 2 has 10 pixels, and gray level 3 has 40 pixels with the value. These are converted into probabilities by normalizing to the total number of pixels in the image.

Next, in step 2, the probabilities are ordered as in Figure 10.2-2b.

For step 3, we combine the smallest two by addition.

Step 4 repeats steps 2 and 3, where we reorder (if necessary) and add the two smallest probabilities as in Figure 10.2-2d. This step is repeated until only two values remain.

Figure 10.2-2: Huffman Coding Example

$g_0 = \frac{20}{100} = 0.2$

$g_1 = \frac{30}{100} = 0.3$

$g_2 = \frac{10}{100} = 0.1$

$g_3 = \frac{40}{100} = 0.4$

a) Step 1: Histogram

$g_3 \rightarrow 0.4$

$g_1 \rightarrow 0.3$

$g_0 \rightarrow 0.2$

$g_2 \rightarrow 0.1$

b) Step 2: Order

$0.4 \rightarrow 0.4$

$0.3 \rightarrow 0.3$

$0.2 \rightarrow 0.3$

0.1

c) Step 3: Add

$0.4 \rightarrow 0.4 \rightarrow 0.4$

$0.3 \rightarrow 0.3 \rightarrow 0.6$

$0.2 \rightarrow 0.3$

0.1

$0.4 \rightarrow 0.4 \rightarrow 0.6$

$0.3 \rightarrow 0.3 \rightarrow 0.4$

$0.2 \rightarrow 0.3$

0.1

d) Step 4: Reorder and add until only two values remain

EXAMPLE 10.2.3 (Contd):

Since we have only two left in our example, we can continue to step 5 where the actual code assignment is made. The code assignment is shown in Figure 10.2-3. We start on the right-hand side of this tree and assign 0's and 1's, working our way back to the original probabilities.

Figure 10.2-3a shows the first assignment of 0 and 1. A 0 is assigned to the 0.6 branch, and a 1 to the 0.4 branch. In Figure 10.2-3b, the assigned 0 and 1 are brought back along the tree, and wherever a branch occurs the code is put on both branches.

Now (Figure 10.2-3c), we assign the 0 and 1 to the branches labeled 0.3, appending to the existing code. Finally (Figure 10.2-3d), the codes are brought back one more level, and where the branch splits another assignment of 0 and 1 occurs (at the 0.1 and 0.2 branch).

Now we have the Huffman code for this image in Table 10-1.

Figure 10.2-3: Huffman Coding Example, Step 5

a) Assign 0 and 1 to the right-most probabilities

b) Bring 0 and 1 back along the tree

c) Append 0 and 1 to previously-added branches

d) Repeat the process until the original branch is labeled

TABLE 10-1

Original Gray Level (Natural Code)	Probability	Huffman code
g_0: 00_2	0.2	010_2
g_1: 01_2	0.3	00_2
g_2: 10_2	0.1	011_2
g_3: 11_2	0.4	1_2

EXAMPLE 10.2.4:

$$Entropy = -\sum_{i=0}^{3} p_i \log_2(p_i)$$

$$= -[(0.2)\log_2(0.2) + (0.3)\log_2(0.3) + (0.1)\log_2(0.1) + (0.4)\log_2(0.4)$$

$$\approx 1.846 \text{ bits/pixel}$$

(Note : $\log_2(x)$ can be found by taking $\log_{10}(x)$ and multiplying by 3.322)

$$L_{ave} = \sum_{i=0}^{L-1} l_i \; p_i$$

$$= 3(0.2) + 2(0.3) + 3(0.1) + 1(0.4)$$

$$= 1.9 \text{ bits/pixel} \quad \text{(Average length with Huffman code)}$$

- In the example, we observe a 2.0 : 1.9 compression, which is about a 1.05 compression ratio, providing about 5% compression

- From the example we can see that the Huffman code is highly dependent on the histogram, so any preprocessing to simplify the histogram will help improve the compression ratio

✔ Run-Length Coding

- Run-length coding (RLC) works by counting adjacent pixels with the same gray level value called the *run-length,* which is then encoded and stored

- RLC works best for binary, two-valued, images

- RLC can also work with complex images that have been preprocessed by thresholding to reduce the number of gray levels to two
- RLC can be implemented in various ways, but the first step is to define the required parameters
- Horizontal RLC (counting along the rows) or vertical RLC (counting along the columns) can be used

- In basic horizontal RLC, the number of bits used for the encoding depends on the number of pixels in a row

- If the row has 2^n pixels, then the required number of bits is *n*, so that a run that is the length of the entire row can be encoded

EXAMPLE 10.2.5:

A 256x256 image requires 8-bits, since $2^8 = 256$.

EXAMPLE 10.2.6:

A 512x512 image requires 9-bits, since $2^9 = 512$.

- The next step is to define a convention for the first RLC number in a row – does it represent a run of 0's or 1's?

EXAMPLE 10.2.7:

The image is an 8x8 binary image, which requires 3 bits for each run-length coded word. In the actual image file are stored 1's and 0's, although upon display the 1's become 255 (white) and the 0's are 0 (black). To apply RLC to this image, using horizontal RLC:

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

EXAMPLE 10.2.7 (contd):

The RLC numbers are:

First row: 8

Second row: 0, 4, 4

Third row: 1, 2, 5

Fourth row: 1, 5, 2

Fifth row: 1, 3, 2, 1, 1

Sixth row: 2, 1, 2, 2, 1

Seventh row: 0, 4, 1, 1, 2

Eighth row: 8

Note that in the second and seventh rows, the first RLC number is 0, since we are using the convention that the first number corresponds to the number of zeros in a run

- *Bitplane-RLC* : A technique which involves extension of basic RLC method to gray level images, by applying basic RLC to each bit-plane independently

- For each binary digit in the gray level value, an image plane is created, and this image plane (a string of 0's and 1's) is then encoded using RLC

Figure 10.2-4: Bit Plane Run-length Coding

b_3	b_2	b_1	b_0
0	0	0	0
0	0	0	1
0	0	1	0
. . .	. . .	. . .	. . .
0	1	1	0
1	1	1	1

a) 4 bits/pixel designation

b) bit-planes → b3 b2 b1 b0
0 1 1 0

- Typical compression ratios of 0.5 to 1.2 are achieved with complex 8-bit monochrome images
- Thus without further processing, this is not a good compression technique for complex images
- Bitplane-RLC is most useful for simple images, such as graphics files, where much higher compression ratios are achieved

- The compression results using this method can be improved by preprocessing to reduce the number of gray levels, but then the compression is *not lossless*

- With lossless bitplane RLC we can improve the compression results by taking our original pixel data (in *natural code*) and mapping it to a *Gray code* (named after Frank Gray), where adjacent numbers differ in only one bit

- As the adjacent pixel values are highly correlated, adjacent pixel values tend to be relatively close in gray level value, and this can be problematic for RLC

Figure 10.2-5: Gray Code

Decimal	4-bit Natural Code	4-bit Gray Code
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
9	1001	1101
10	1010	1111
11	1011	1110
12	1100	1010
13	1101	1011
14	1110	1001
15	1111	1000

a) Gray code versus natural code

b) The natural code transition of 7 to 8 changes all 4 bits

EXAMPLE 10.2.8:

In Figure 10.2-5 is shown the 4-bit Gray code and the natural binary code. The Gray code, by definition, only has one bit changing in adjacent codes. However, in, for example, the 7 to 8 transition with the natural code, all 4 bits change:

Natural Code	*Gray Code*
0111	*0100*
↓↓↓↓	↓↓↓↓
1000	*1100*

- When a situation such as the above example occurs, each bitplane experiences a transition, which adds a code for the run in each bitplane
- However, with the Gray code, only one bitplane experiences the transition, so it only adds one extra code word
- By preprocessing with a Gray code we can achieve about a 10% to 15% increase in compression with bitplane-RLC for typical images

- Another way to extend basic RLC to gray level images is to include the gray level of a particular run as part of the code
- Here, instead of a single value for a run, two parameters are used to characterize the run
- The pair (G,L) correspond to the gray level value, G, and the run length, L
- This technique is only effective with images containing a small number of gray levels

EXAMPLE 10.2.9:

Given the following 8x8, 4-bit image:

$$\begin{bmatrix} 10 & 10 & 10 & 10 & 10 & 10 & 10 & 10 \\ 10 & 10 & 10 & 10 & 10 & 12 & 12 & 12 \\ 10 & 10 & 10 & 10 & 10 & 12 & 12 & 12 \\ 0 & 0 & 0 & 10 & 10 & 10 & 0 & 0 \\ 5 & 5 & 5 & 0 & 0 & 0 & 0 & 0 \\ 5 & 5 & 5 & 10 & 10 & 9 & 9 & 10 \\ 5 & 5 & 5 & 4 & 4 & 4 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

EXAMPLE 10.2.9 (contd):

The corresponding gray levels pairs are as follows:

First row: 10,8

Second row: 10,5 12,3

Third row: 10,5 12,3

Fourth row: 0,3 10,3 0,2

Fifth row: 5,3 0,5

Sixth row: 5,3 10,2 9,2 10,1

Seventh row: 5,3 4,3 0,2

Eighth row: 0,8

These numbers are then stored in the RLC compressed file as:

10,8,10,5,12,3,10,5,12,3,0,3,10,3,0,2,5,3,0,5,5,3,10,2,9,2,10,1,5,3,4,3,0,2,0,8

- The decompression process requires the number of pixels in a row, and the type of encoding used
- Standards for RLC have been defined by the International Telecommunications Union-Radio (ITU-R, previously CCIR)
- These standards use horizontal RLC, but postprocess the resulting RLC with a Huffman encoding scheme

- Newer versions of this standard also utilize a two-dimensional technique where the current line is encoded based on a previous line, which helps to reduce the file size

- These encoding methods provide compression ratios of about 15 to 20 for typical documents

✔ Lempel-Ziv-Welch Coding

- The Lempel-Ziv-Welch (LZW) coding algorithm works by encoding strings of data, which correspond to sequences of pixel values in images

- It works by creating a string table that contains the strings and their corresponding codes

- The string table is updated as the file is read, with new codes being inserted whenever a new string is encountered
- If a string is encountered that is already in the table, the corresponding code for that string is put into the compressed file
- LZW coding uses code words with more bits than the original data

For Example:

- With 8-bit image data, an LZW coding method could employ 10-bit words
- The corresponding string table would then have $2^{10} = 1024$ entries
- This table consists of the original 256 entries, corresponding to the original 8-bit data, and allows 768 other entries for string codes

- The string codes are assigned during the compression process, but the actual string table is not stored with the compressed data

- During decompression the information in the string table is extracted from the compressed data itself

- For the GIF (and TIFF) image file format the LZW algorithm is specified, but there has been some controversy over this, since the algorithm is patented by Unisys Corporation

- Since these image formats are widely used, other methods similar in nature to the LZW algorithm have been developed to be used with these, or similar, image file formats

- Similar versions of this algorithm include the *adaptive Lempel-Ziv*, used in the UNIX compress function, and the *Lempel-Ziv 77* algorithm used in the UNIX gzip function

✔ Arithmetic Coding

- *Arithmetic coding* transforms input data into a single floating point number between 0 and 1

- There is not a direct correspondence between the code and the individual pixel values

- As each input symbol (pixel value) is read the precision required for the number becomes greater

- As the images are very large and the precision of digital computers is finite, the entire image must be divided into small subimages to be encoded

- Arithmetic coding uses the probability distribution of the data (histogram), so it can theoretically achieve the maximum compression specified by the entropy

- It works by successively subdividing the interval between 0 and 1, based on the placement of the current pixel value in the probability distribution

EXAMPLE 10.2.10:

Given a 16x16, 2-bit image with the histogram shown in Figure 10.2-6a, we can define an arithmetic coding probability table shown in Figure 10.2-6b.

The probability values are the ratio of the specific gray level value to the total number of pixels in the image (in this case 16x16 = 256).

The initial subinterval specifies how the 0 to 1 interval is divided based on the distribution, where the width of the subinterval is equal to the probability, and the subinterval starts where the previous one stops.

Figure 10.2-6: Arithmetic Coding

a) Histogram

Pixel Value	Probability	Initial Sub-interval
0	64/256 = 1/4	0 - 1/4
1	128/256 = 1/2	1/4 - 3/4
2	32/256 = 1/8	3/4 - 7/8
3	32/256 = 1/8	7/8 - 1

b) Probability table

c) Coding process for '0,0,3,1'

EXAMPLE 10.2.10 (contd):

In Figure 10.2-6c, the actual arithmetic coding process is illustrated, with an example pixel value sequence of: 0,0,3,1.

Starting on the left, the initial 0 to 1 interval is subdivided, based on the probability distribution.

Next, the first pixel value '0' is coded by extracting the subinterval corresponding to the '0' and subdividing it again, based on the same relative distribution.

This process is repeated for each pixel value in the sequence until a final interval is determined, in this case from 58/1024 to 62/1024, or 0.056640625 to 0.060546875.

Any value within this subinterval, such as 0.057 or 0.060, can be used to represent this sequence of gray level values.

- In practice, this technique may be used as part of an image compression scheme, but is impractical to use alone

- It is one of the options available in the JPEG standard

Lossy Compression Methods

- Lossy compression methods are required to achieve high compression ratios with complex images
- They provides tradeoffs between image quality and degree of compression, which allows the compression algorithm to be customized to the application

Figure 10.3-1: Lossy Image Compression

a) Original image

b) JPEG compression, 10:1 ratio

c) JPEG compression, 48:1 ratio

d) Wavelet/vector quantization compression, 36:1 ratio

- With more advanced methods, images can be compressed 10 to 20 times with virtually no visible information loss, and 30 to 50 times with minimal degradation
- Newer techniques, such as *JPEG2000*, can achieve reasonably good image quality with compression ratios as high as 100 to 200
- *Image enhancement* and *restoration techniques* can be combined with lossy compression schemes to improve the appearance of the decompressed image

- In general, a higher compression ratio results in a poorer image, but the results are highly image dependent – application specific

- Lossy compression can be performed in both the *spatial* and *transform* domains. *Hybrid methods* use both domains.

✔ Gray-Level Run Length Coding

- The RLC technique can also be used for lossy image compression, by reducing the number of gray levels, and then applying standard RLC techniques

- As with the lossless techniques, preprocessing by Gray code mapping will improve the compression ratio

Figure 10.3-2 Lossy Bitplane Run Length Coding

Note: No compression occurs until reduction to 5 bits/pixel

a) Original image, 8 bits/pixel, 256 gray levels

b) Image after reduction to 7 bits/pixel, 128 gray levels, compression ratio 0.55, with Gray code preprocessing 0.66

Figure 10.3-2 Lossy Bitplane Run Length Coding (contd)

c) Image after reduction to 6 bits/pixel, 64 gray levels, compression ratio 0.77, with Gray code preprocessing 0.97

d) Image after reduction to 5 bits/pixel, 32 gray levels, compression ratio 1.20, with Gray code preprocessing 1.60

Figure 10.3-2 Lossy Bitplane Run Length Coding (contd)

e) Image after reduction to 4 bits/pixel, 16 gray levels, compression ratio 2.17, with Gray code preprocessing 2.79

f) Image after reduction to 3 bits/pixel, 8 gray levels, compression ratio 4.86, with Gray code preprocessing 5.82

Figure 10.3-2 Lossy Bitplane Run Length Coding (contd)

g) Image after reduction to 2 bits/pixel,
4 gray levels, compression ratio 13.18,
with Gray code preprocessing 15.44

h) Image after reduction to 1 bit/pixel,
2 gray levels, compression ratio 44.46,
with Gray code preprocessing 44.46

- A more sophisticated method is *dynamic window-based RLC*
- This algorithm relaxes the criterion of the runs being the same value and allows for the runs to fall within a gray level range, called the *dynamic window range*
- This range is dynamic because it starts out larger than the actual gray level window range, and maximum and minimum values are narrowed down to the actual range as each pixel value is encountered

- This process continues until a pixel is found out of the actual range

- The image is encoded with two values, one for the run length and one to approximate the gray level value of the run

- This approximation can simply be the average of all the gray level values in the run

EXAMPLE 10.3.1:

Given the following pixel values in sequence:

65 67 66 64 63 68 70

and a window range of 5.

The first value is called the reference value (in this case = 65). A dynamic window range is then defined that has:

MINIMUM = reference - (window length -1) and

MAXIMUM = reference + (window length - 1)

In this case the dynamic window is: [65 - (5 - 1)] to [65 + (5 - 1]) = 61 to 69.

The next value encountered, 67, is used to adjust this range. The range based on this value alone is from 63 to 71. The new dynamic range is based on the intersection of the range from this new value with the previous range, so the new range is 63 to 69. This process continues until the value of 68 is encountered. At this point the range has been narrowed down to 63 to 67, so the 68 is out of range. This run is then encoded as:

RUN LENGTH = 5

GRAY LEVEL = (65+67+66+64+63)/5 = 65

Figure 10.3-3: Dynamic Window Range RLC

a) Original image

b) Window length = 10
compression = 4:1

c) Error image of (b),
multiplied to show detail

Figure 10.3-3, continued

d) Window length = 35
compression = 8:1

e) Error image of (d),
multiplied to show detail

f) Window length = 60
compression = 12.6:1

g) Error image of (f),
multiplied to show detail

- This particular algorithm also uses some preprocessing to allow for the run-length mapping to be coded so that a run can be any length and is not constrained by the length of a row

✔ Block Truncation Coding

- *Block truncation coding* (BTC) works by dividing the image into small subimages and then reducing the number of gray levels within each block

- The gray levels are reduced by a *quantizer* that adapts to local statistics

- The levels for the quantizer are chosen to minimize a specified error criteria, and then all the pixel values within each block are mapped to the quantized levels
- The necessary information to decompress the image is then encoded and stored
- The basic form of BTC divides the image into N * N blocks and codes each block using a two-level quantizer

- The two levels are selected so that the mean and variance of the gray levels within the block are preserved
- Each pixel value within the block is then compared with a threshold, typically the block mean, and then is assigned to one of the two levels
- If it is above the mean it is assigned the high level code, if it is below the mean, it is assigned the low level code

- If we call the high value *H* and the low value *L*, we can find these values via the following equations:

$$H = m_b + \sigma_b \sqrt{\frac{n^2 - q}{q}}$$

$$L = m_b - \sigma_b \sqrt{\frac{q}{n^2 - q}}$$

Where the block size is $n \times n$

$b =$ *the current block*

$$m_b = \text{the block mean} = \frac{1}{n^2} \sum_{I(r,c) \in b} I(r,c)$$

$$\sigma_b = \text{the block variance} = \sqrt{\frac{1}{n^2} \sum_{I(r,c) \in b} [I(r,c)]^2 - m_b^2}$$

$q =$ *the number of values in the block* $\geq m_b$

- If $n = 4$, then after the H and L values are found, the 4x4 block is encoded with four bytes

- Two bytes to store the two levels, H and L, and two bytes to store a bit string of 1's and 0's corresponding to the high and low codes for that particular block

Figure 10.3-4: Basic Block Truncation Coding

EXAMPLE 10.3.2:

Given the following 4x4 subimage, apply basic BTC and find the resulting values.

$$\begin{bmatrix} 12 & 16 & 15 & 17 \\ 13 & 16 & 17 & 17 \\ 4 & 4 & 35 & 35 \\ 42 & 42 & 12 & 12 \end{bmatrix}$$

$$m_b = \frac{1}{n^2} \sum_{I(r,c) \in b} I(r,c) = \frac{1}{16}\left[12+16+15+17+13+16+17+17+4+4+35+35+42+42+12+12\right]$$

$$= 19.3125$$

$$\sigma_b = \sqrt{\frac{1}{n^2} \sum_{I(r,c) \in b} [I(r,c)]^2 - m_b^2}$$

$$= \sqrt{\frac{1}{16}\left[12^2+16^2+15^2+17^2+13^2+16^2+17^2+17^2+4^2+4^2+35^2+35^2+42^2+42^2+12^2+12^2\right] - (19.3125)^2}$$

$$\approx 11.85$$

EXAMPLE 10.3.2 (contd):

There are 4 pixels values greater than the mean, so q = 4.

$$H = m_b + \sigma_b \sqrt{\frac{n^2 - q}{q}} = 19.3125 + 11.85\sqrt{\frac{16-4}{4}} \approx 40$$

$$L = m_b - \sigma_b \sqrt{\frac{q}{n^2 - q}} = 19.3125 - 11.85\sqrt{\frac{4}{16-4}} \approx 13$$

Now, find the bit string by using 0 for values less than the mean and 1 for values greater than the mean:

$$\begin{bmatrix} 12 & 16 & 15 & 17 \\ 13 & 16 & 17 & 17 \\ 4 & 4 & 35 & 35 \\ 42 & 42 & 12 & 12 \end{bmatrix} \Rightarrow \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 \end{bmatrix} \Rightarrow (0000000000111100_2)$$

The high value, H, and the low value, L, will be stored along with the bit string. The subimage, when decompressed will be:

$$\begin{bmatrix} 13 & 13 & 13 & 13 \\ 13 & 13 & 13 & 13 \\ 13 & 13 & 40 & 40 \\ 40 & 40 & 13 & 13 \end{bmatrix}$$

- This algorithm tends to produce images with blocky effects

- These artifacts can be smoothed by applying enhancement techniques such as median and average (lowpass) filters

Figure 10.3-5: Block Truncation Coding (BTC)

a) Original image

b) Block truncation coded image, compression = 4:1

c) Error image of (b), histogram stretched to show detail

Figure 10.3-5, continued

d) Image (b) post-processed with a 3x3 median filter

e) Image (b) post-processed with a 3x3 averaging filter

- The *multilevel BTC algorithm,* which uses a 4-level quantizer, allows for varying the block size, and a larger block size should provide higher compression, but with a corresponding decrease in image quality

- With this particular implementation, we get decreasing image quality, but the compression ratio is fixed

Figure 10.3-6: Multilevel Block Truncation Coding

a) Original image

b) Multilevel BTC,
block size = 8x8,
compression = 3.99:1

c) Error image of (b),
multiplied to show detail

Figure 10.3-6, continued

d) Multilevel BTC,
block size = 16x16,
compression = 3.99:1

e) Error image of (d)
multiplied to show detail

f) Multilevel BTC,
block size = 32x32,
compression = 3.99:1

g) Error image of (f)
multiplied to show detail

✔ Vector Quantization

- *Vector quantization (VQ)* is the process of mapping a vector that can have many values to a vector that has a smaller (quantized) number of values

- For image compression, the vector corresponds to a small subimage, or block

EXAMPLE 10.3.3:

Given the following 4x4 subimage:

$$\begin{bmatrix} 65 & 70 & 71 & 75 \\ 71 & 70 & 71 & 81 \\ 81 & 80 & 81 & 82 \\ 90 & 90 & 91 & 92 \end{bmatrix}$$

This can be re-arranged into a 1-D vector by putting the rows adjacent as follows:

[row1 row2 row3 row4] = [65 70 71 75 71 70 71 81 81 80 81 82 90 90 91 92]

- VQ can be applied in both the spectral or spatial domains

- Information theory tells us that better compression can be achieved with vector quantization than with scalar quantization (rounding or truncating individual values)

- Vector quantization treats the entire subimage (vector) as a single entity and quantizes it by reducing the total number of bits required to represent the subimage

- This is done by utilizing a *codebook*, which stores a fixed set of vectors, and then coding the subimage by using the index (address) into the codebook

EXAMPLE 10.3.4:

Given an 8-bit, 256x256 image, we devise a vector quantization scheme that will encode each 4x4 block with one of the vectors in a codebook of 256 entries. We determine that we want to encode a specific subimage with vector number 122 in the codebook. For this subimage we then store the number 122 as the index into the codebook. Then when the image is decompressed, the vector at the 122 address in the codebook, is used for that particular subimage. This is illustrated in Figure 10.3-7. This will require 1 byte (8-bits) to be stored for each 4x4 block, providing a data reduction of 16 bytes for a 4x4 block to 1 byte, or 16:1

- In the example we achieved a 16:1 compression, but note that this assumes that the codebook is not stored with the compressed file

Figure 10.3-7: Quantizing with a Codebook

a) Original 256x256 image divided into 4x4 blocks

b) Codebook with 256 16-byte entries

a	b	c	d
e	f	g	h
i	j	k	l
m	n	o	p

c) A subimage decompressed with vector #122

- However, the codebook will need to be stored unless a generic codebook is devised which could be used for a particular type of image, in that case we need only store the name of that particular codebook file

- In the general case, better results will be obtained with a codebook that is designed for a particular image

EXAMPLE 10.3.5:

If we include the codebook in the compressed file from the previous example, the compression ratio will not be quite as good. For every 4x4 block we will have 1 byte. This gives us:

$$\left(\frac{256\ pixels}{4\ pixels/block}\right)\left(\frac{256\ pixels}{4\ pixels/block}\right)=4096\ blocks$$

At 1 byte for each 4x4 block, this give us 4096 bytes for the codebook addresses. Now we also include the size of the codebook, 256x16:

$$4096+(256)(16)=8192\ bytes\ for\ the\ coded\ file$$

The original 8-bit, 256x256 image contained:

$$(256)(256)=65{,}536\ bytes$$

Thus, we obtain a compression of:

$$\frac{65{,}536}{8192}=8\rightarrow 8:1\ compression$$

In this case, including the codebook cut the compression in half, from 16:1 to 8:1

- A training algorithm determines which vectors will be stored in the codebook by finding a set of vectors that best represent the blocks in the image

- This set of vectors is determined by optimizing some error criterion, where the *error* is defined as the sum of the vector distances between the original subimages and the resulting decompressed subimages

- The standard algorithm to generate the codebook is the Linde-Buzo-Gray (*LBG*) algorithm, also called the *K-means* or the *clustering* algorithm:

Step 1: Given an arbitrary codebook, encode each input vector according to the nearest-neighbor criterion. Use a distance metric to compare all the input vectors to the encoded vectors, and then sum these errors (distances) to provide a distortion measure. If the distortion is small enough (less than a predefined threshold), quit. If not, go to step 2.

Step 2: For each codebook entry, compute the euclidean centroid of all the input vectors encoded into that specific codebook vector.

Step 3: Use the computed centroids as the new codebook, and go to step 1.

- The LBG algorithm, along with other iterative codebook design algorithms do not, in general, yield globally optimum codes
- These algorithms will converge to a local minimum in the error (distortion) space
- Theoretically, to improve the codebook, the algorithm is repeated with different initial random codebooks and the one codebook that minimizes distortion is chosen

- However, the LBG algorithm will typically yield "good" codes if the initial codebook is carefully chosen by subdividing the vector space and finding the centroid for the sample vectors within each division
- These centroids are then used as the initial codebook
- Alternately, a subset of the training vectors, preferably spread across the vector space, can be randomly selected and used to initialize the codebook

- The primary advantage of vector quantization is simple and fast decompression, but with the high cost of complex compression

- The decompression process requires the use of the codebook to recreate the image, which can be easily implemented with a look-up table (LUT)

- This type of compression is useful for applications where the images are compressed once and decompressed many times, such as images on an Internet site

- However, it cannot be used for real-time applications

Figure 10.3-8
Vector Quantization in the Spatial Domain

a) Original image

b) VQ with 4x4 vectors, and a codebook of 128 entries, compression ratio = 11.49

Figure 10.3-8
Vector Quantization in the Spatial Domain (contd)

c) VQ with 4x4 vectors, and a codebook of 256 entries, compression ratio = 7.93

d) VQ with 4x4 vectors, and a codebook of 512 entries, compression ratio = 5.09

Note: As the codebook size is increased the image quality improves and the compression ratio decreases.

Figure 10.3-9
Vector Quantization in the Transform Domain

Note: The original image is the image in Figure 10.3-8a

a) VQ with the discrete cosine transform, compression ratio = 9.21

b) VQ with the wavelet transform, compression ratio = 9.21

Figure 10.3-9
Vector Quantization in the Transform Domain (contd)

c) VQ with the discrete cosine transform, compression ratio = 3.44

d) VQ with the wavelet transform, compression ratio = 3.44

✔ Differential Predictive Coding

- *Differential predictive coding (DPC)* predicts the next pixel value based on previous values, and encodes the difference between predicted and actual value – the error signal
- This technique takes advantage of the fact that adjacent pixels are highly correlated, except at object boundaries

- Typically the difference, or error, will be small which minimizes the number of bits required for compressed file

- This error is then quantized, to further reduce the data and to optimize visual results, and can then be coded

Figure 10.3-10: Differential Predictive Coding (DPC)

- From the block diagram, we have the following:

$$\tilde{I} = \textit{the predicted next pixel value}$$

$$\hat{I} = \textit{the reconstructed pixel value}$$

$$e = I - \tilde{I} = \textit{error}$$

$$\hat{e} = \hat{I} - \tilde{I} = \textit{quantized error}$$

- The prediction equation is typically a function of the previous pixel(s), and can also include global or application-specific information

The theoretically optimum predictor that uses only the previous value is based on minimizing mean squared error between the original and the decompressed image, and is given by:

$$\tilde{I}(r,c+1) = \rho\hat{I}(r,c) + (1-\rho)\bar{I}(r,c)$$

where $\bar{I}(r,c)$ = *the average value for the image*

ρ = *the normalized correlation between pixel values*

For most images ρ is between 0.85 and 0.95. Once the next pixel value has been predicted, the error is calculated:

$$e(r,c+1) = I(r,c+1) - \tilde{I}(r,c+1)$$

This error signal is then quantized, such that:

$$\hat{e}(r,c+1) = \hat{I}(r,c+1) - \tilde{I}(r,c+1)$$

- This quantized error can be encoded using a lossless encoder, such as a Huffman coder

- It should be noted that it is important that the predictor uses the same values during both compression and decompression; specifically the reconstructed values and not the original values

Figure 10.3-11: DPC Example

a) Original image

b) DPC using original values in predictor, clipping to the maximum, 5 bits/pixel, normalized correlation .90

c) Error image of (b), multiplied to show detail

Figure 10.3-11, continued

d) DPC using reconstructed values in predictor, clipping to the maximum, 5 bits/pixel, normalized correlation .90

e) Error image of (d) multiplied to show detail

- The prediction equation can be one-dimensional or two-dimensional, that is, it can be based on previous values in the current row only, or on previous rows also
- The following prediction equations are typical examples of those used in practice, with the first being one-dimensional and the next two being two-dimensional:

$$\tilde{I}(r,c+1) = 0.97\ \hat{I}(r,c)$$

$$\tilde{I}(r,c+1) = 0.49\ \hat{I}(r,c) + 0.49\hat{I}(r-1,c+1)$$

$$\tilde{I}(r,c+1) = 0.74\ \hat{I}(r,c) + 0.74\hat{I}(r-1,c+1) - 0.49\hat{I}(r-1,c)$$

Figure 10.3-12: DPC Predictor Dimensions

a) One-dimensional predictor, based on current row only
x = current pixel

b) Two-dimensional predictor is based on current and previous row or rows

- Using more of the previous values in the predictor increases the complexity of the computations for both compression and decompression

- It has been determined that using more than three of the previous values provides no significant improvement in the resulting image

- The results of DPC can be improved by using an optimal quantizer, such as the *Lloyd-Max quantizer*, instead of simply truncating the resulting error

- The Lloyd-Max quantizer assumes a specific distribution for the prediction error

- Assuming a 2-bit code for the error, and a Laplacian distribution for the error, the Lloyd-Max quantizer is defined as follows:

ERROR RANGE		QUANTIZED VALUE
$0 \le e < 1.102\sigma$	$\rightarrow$	$+0.395\sigma$
$1.102\sigma \le e < \infty$	$\rightarrow$	$+1.81\sigma$
$-1.102\sigma \le e < 0$	$\rightarrow$	-0.395σ
$-1.102\sigma \le e < -\infty$	$\rightarrow$	-1.81σ

Where σ = the standard deviation of the error distribution

Figure 10.3-13: Lloyd-Max Quantizer

a) 2-bit Lloyd-Max quantizer with Laplacian error distribution

b) An example for σ = 3.63. The ±1.43 is typically rounded to ±1, and the ±6.57 is rounded to ±7

- For most images, the standard deviation for the error signal is between 3 and 15

- After the data is quantized it can be further compressed with a lossless coder such as Huffman or arithmetic coding

Figure 10.3-14: DPC Quantization

a) Original image

b) Lloyd-Max quantizer, using 2 bits/pixel, normalized correlation = 0.90, with standard deviation = 10

c) Truncation quantizer, using 2 bits/pixel, normalized correlation = 0.90

Figure 10.3-14, continued

d) Lloyd-Max quantizer, using 4 bits/pixel, normalized correlation = 0.90, with standard deviation = 10

e) Truncation quantizer, using 4 bits/pixel, normalized correlation = 0.90

Figure 10.3-15: DPC Quantization

a) Original image

b) Lloyd-Max quantizer, using 1 bit/pixel, normalized correlation = 0.90, with standard deviation = 10

c) Error image for (b)

Figure 10.3-15, continued

d) Lloyd-Max quantizer, using 2 bits/pixel, normalized correlation = 0.90, with standard deviation = 10

e) Error image for (d)

f) Lloyd-Max quantizer, using 3 bit/pixel, normalized correlation = 0.90, with standard deviation = 10

g) Error image for (f)

Figure 10.3.15 DPC Quantization (contd)

h) Lloyd-Max quantizer, using 4 bits/pixel,
normalized correlation = 0.90,
with standard deviation = 10

i) Error image for (h)

j) Lloyd-Max quantizer, using 5 bits/pixel,
normalized correlation = 0.90,
with standard deviation = 10

k) Error image for (j)

✔ Model-based and Fractal Compression

- *Model-based* or *intelligent* compression works by finding models for objects within the image and using model parameters for the compressed file
- The techniques used are similar to computer vision methods where the goal is to find descriptions of the objects in the image

- The objects are often defined by lines or shapes (boundaries), so a Hough transform (Chap 4) may be used, while the object interiors can be defined by statistical texture modeling
- The model-based methods can achieve very high compression ratios, but the decompressed images often have an artificial look to them
- *Fractal methods* are an example of model-based compression techniques

- Fractal image compression is based on the idea that if an image is divided into subimages, many of the subimages will be self-similar

- *Self-similar* means that one subimage can be represented as a skewed, stretched, rotated, scaled and/or translated version of another subimage

- Treating the image as a geometric plane, the mathematical operations (skew, stretch, scale, rotate, translate) are called *affine transformations* and can be represented by the following general equations:

$$r' = k_1 r + k_2 c + k_3$$

$$c' = k_4 r + k_5 c + k_6$$

Where r' and c' are the new coordinates, and k_i are constants

- Fractal compression is somewhat like vector quantization, except that the subimages, or blocks, can vary in size and shape
- The idea is to find a good set of basis images, or fractals, that can undergo affine transformations, and then be assembled into a good representation of the image
- The fractals (basis images), and the necessary affine transformation coefficients are then stored in the compressed file

- Fractal compression can provide high quality images and very high compression rates, but often at a very high cost
- The quality of the resulting decompressed image is directly related to the amount of time taken in generating the fractal compressed image
- If the compression is done offline, one time, and the images are to be used many times, it may be worth the cost

- An advantage of fractals is that they can be magnified as much as is desired, so one fractal compressed image file can be used for any resolution or size of image
- To apply fractal compression, the image is first divided into non-overlapping regions that completely cover the image, called *domains*
- Then, regions of various size and shape are chosen for the basis images, called the *range* regions

- The range regions are typically larger than the domain regions, can be overlapping and do not cover the entire image

- The goal is to find the set affine transformations to best match the range regions to the domain regions

- The methods used to find the best range regions for the image, as well as the best transformations, are many and varied

Figure 10.3-16 Fractal Compression

a) Cameraman image compressed with fractal encoding, compression ratio = 9.19

b) Error image for (a)

Figure 10.3-16 Fractal Compression (contd)

c) Compression ratio = 15.65

d) Error image for (c)

Figure 10.3-16 Fractal Compression (contd)

e) Compression ratio = 34.06

f) Error image for (e)

Figure 10.3-16 Fractal Compression (contd)

g) A checkerboard,
compression ratio = 564.97

h) Error image for (g)

Note: Error images have been remapped for display so the background gray corresponds to zero, then they were enhanced by a histogram stretch to show detail

✔ Transform Coding

- *Transform coding,* is a form of block coding done in the transform domain
- The image is divided into blocks, or subimages, and the transform is calculated for each block

- Any of the previously defined transforms can be used, frequency (e.g. Fourier) or sequency (e.g. Walsh/Hadamard), but it has been determined that the discrete cosine transform (DCT) is optimal for most images

- The newer JPEG2000 algorithms uses the wavelet transform, which has been found to provide even better compression

- After the transform has been calculated, the transform coefficients are quantized and coded

- This method is effective because the frequency/sequency transform of images is very efficient at putting most of the information into relatively few coefficients, so many of the high frequency coefficients can be quantized to 0 (eliminated completely)

- This type of transform is a special type of mapping that uses spatial frequency concepts as a basis for the mapping

- The main reason for mapping the original data into another mathematical space is to pack the information (or energy) into as few coefficients as possible

- The simplest form of transform coding is achieved by *filtering* by eliminating some of the high frequency coefficients

- However, this will not provide much compression, since the transform data is typically *floating point* and thus 4 or 8 bytes per pixel (compared to the original pixel data at 1 byte per pixel), so quantization and coding is applied to the reduced data

- Quantization includes a process called *bit allocation*, which determines the number of bits to be used to code each coefficient based on its importance

- Typically, more bits are used for lower frequency components where the energy is concentrated for most images, resulting in a *variable bit rate* or *nonuniform quantization* and better resolution

EXAMPLE 10.3.6:

We have decided to use transform coding with a DCT on an image by dividing it into 4x4 blocks. The selected bit allocation can be represented by the following mask:

$$\begin{bmatrix} 8 & 6 & 4 & 1 \\ 6 & 4 & 1 & 0 \\ 4 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix}$$

Where the numbers in the mask are the number of bits used to represent the corresponding transform coefficients. The upper left corner corresponds to the zero-frequency coefficient, DC or average value, and the frequency increases to the right and down. This allows the lower frequencies less quantization (more resolution), since they have more bits to represent them. Because the number of bits varies with different coefficients, we have a *variable bit rate*. Additionally, more bits means finer quantization and fewer bits means coarser quantization resulting in *nonuniform quantization*

- Then a quantization scheme, such as Lloyd-Max quantization is applied
- As the zero-frequency coefficient for real images contains a large portion of the energy in the image and is always positive, it is typically treated differently than the higher frequency coefficients
- Often this term is not quantized at all, or the differential between blocks is encoded
- After they have been quantized, the coefficients can be coded using, for example, a Huffman or arithmetic coding method

- Two particular types of transform coding have been widely explored:

1. *Zonal coding*
2. *Threshold coding*

- These two vary in the method they use for selecting the transform coefficients to retain (using ideal filters for transform coding selects the coefficients based on their location in the transform domain)

1. *Zonal coding*

- It involves selecting specific coefficients based on maximal variance
- A zonal mask is determined for the entire image by finding the variance for each frequency component
- This variance is calculated by using each subimage within the image as a separate sample and then finding the variance within this group of subimages

Figure 10.3-17: Zonal Coding

I(r,c)

a) Divide the image into blocks

T(u,v)

a	b	c	d	e
f	g···			

b) Apply the transform to each block

c) Treating each transform block from T(u,v) as a separate sample, calculate the variance for each frequency component. Retain only the components with variance above a specified threshold.

1	1	1	1	0
1	1	1	0	0
1	1	0	0	0
1	0	0	0	0
0	0	0	0	0

d) Generate zonal masks;
1 = retain
0 = eliminate
A typical mask is shown.

- The *zonal mask* is a bitmap of 1's and 0', where the 1's correspond to the coefficients to retain, and the 0's to the ones to eliminate

- As the zonal mask applies to the *entire image,* only one mask is required

2. *Threshold coding*

- It selects the transform coefficients based on specific value
- A different threshold mask is required for each block, which increases file size as well as algorithmic complexity

- In practice, the zonal mask is often predetermined because the low frequency terms tend to contain the most information, and hence exhibit the most variance

- In this case we select a fixed mask of a given shape and desired compression ratio, which streamlines the compression process

- It also saves the overhead involved in calculating the variance of each group of subimages for compression and also eases the decompression process

- Typical masks may be square, triangular or circular and the cutoff frequency is determined by the compression ratio

Figure 10.3-18
Zonal Compression with DCT and Walsh Transforms

A block size of 64x64 was used, a circular zonal mask, and DC coefficients were not quantized

a) Original image, a view of St. Louis, Missouri, from the Gateway Arch

b) Results from using the DCT with a compression ratio = 4.27

c) Error image comparing the original and (b), histogram stretched to show detail

(c) Scott E Umbaugh, SIUE 2005

Figure 10.3-18
Zonal Compression with DCT and Walsh Transforms
(contd)

d) Results from using the DCT with a compression ratio = 14.94

e) Error image comparing the original and (d), histogram stretched to show detail,

Figure 10.3-18
Zonal Compression with DCT and Walsh Transforms
(contd)

f) Results from using the Walsh Transform (WHT) with a compression ratio = 4.27

g) Error image comparing the original and (f), histogram stretched to show detail

Figure 10.3-18
Zonal Compression with DCT and Walsh Transforms
(contd)

h) Results from using the WHT with a compression ratio = 14.94

i) Error image comparing the original and (h), histogram stretched to show detail

- One of the most commonly used image compression standards is primarily a form of transform coding
- The Joint Photographic Expert Group (JPEG) under the auspices of the International Standards Organization (ISO) devised a family of image compression methods for still images
- The original JPEG standard uses the DCT and 8x8 pixel blocks as the basis for compression

- Before computing the DCT, the pixel values are level shifted so that they are centered at zero

- EXAMPLE 10.3.7:

  A typical 8-bit image has a range of gray levels of 0 to 255. Level shifting this range to be centered at zero involves subtracting 128 from each pixel value, so the resulting range is from -128 to 127

- After level shifting, the DCT is computed
- Next, the DCT coefficients are quantized by dividing by the values in a quantization table and then truncated
- For color signals JPEG transforms the RGB components into the *YCrCb* color space, and subsamples the two color difference signals (*Cr* and *Cb*), since we perceive more detail in the luminance (brightness) than in the color information

- Once the coefficients are quantized, they are coded using a Huffman code
- The zero-frequency coefficient (DC term) is differentially encoded relative to the previous block

Figure 10.3-19 Original JPEG DCT Coefficient Quantization Tables. a) luminance (brightness) quantization table corresponding to the Y component of the color transform, b) chrominance (color) quantization table corresponding to the Cr and Cb components of the color transform. The zero frequency (DC) term is in the upper left corner, at (1,1), and the frequencies increase to the right and down. As the quantization coefficient increases, the quantization gets coarser. For many high frequency terms the quantization coefficient is large enough that, after dividing and truncating, they are zero.

a)

Mask	1	2	3	4	5	6	7	8
1	16	11	10	16	24	40	51	61
2	12	12	14	19	26	58	60	55
3	14	13	16	24	40	57	69	56
4	14	17	22	29	51	87	80	62
5	18	22	37	56	68	109	103	77
6	24	35	55	64	81	104	113	92
7	49	64	78	87	103	121	120	101
8	72	92	95	98	112	100	103	99

b)

Mask	1	2	3	4	5	6	7	8
1	17	18	24	47	99	99	99	99
2	18	21	26	66	99	99	99	99
3	24	26	56	99	99	99	99	99
4	47	66	99	99	99	99	99	99
5	99	99	99	99	99	99	99	99
6	99	99	99	99	99	99	99	99
7	99	99	99	99	99	99	99	99
8	99	99	99	99	99	99	99	99

These quantization tables were experimentally determined by JPEG to take advantage of the human visual system's response to spatial frequency which peaks around 4 or 5 cycles per degree

Figure 10.3-20: The Original DCT-based JPEG Algorithm

a) Original image

b) JPEG compression = 10:1

c) Error image for (b), multiplied by 8 to show detail

Figure 10.3-20, continued

d) JPEG compression = 20:1

e) Error image for (d), multiplied by 8 to show detail

f) JPEG compression = 30:1

g) Error image for (f), multiplied by 8 to show detail

Figure 10.3-21:The Original DCT-based JPEG Algorithm Applied to a Color Image

a) The original image

b) Compression ratio = 34.34

Figure 10.3-21:The Original DCT-based JPEG Algorithm Applied to a Color Image (contd)

c) Compression ratio = 57.62

d) Compression ratio = 79.95

Figure 10.3-21:The Original DCT-based JPEG Algorithm Applied to a Color Image (contd)

e) Compression ratio = 131.03

f) Compression ratio = 201.39

✔ Hybrid and Wavelet Methods

- Hybrid methods use both the spatial and spectral domains
- Algorithms exist that combine differential coding and spectral transforms for analog video compression

- For digital images these techniques can be applied to blocks (subimages), as well as rows or columns
- Vector quantization is often combined with these methods to achieve higher compression ratios
- The wavelet transform, which localizes information in both the spatial and frequency domain, is used in newer hybrid compression methods like the JPEG2000 standard

- The wavelet transform provides superior performance to the DCT-based techniques, and also is useful in progressive transmission for Internet and database use

- Progressive transmission allows low quality images to appear quickly and then gradually improve over time as more detail information is transmitted or retrieved

- Thus the user need not wait for an entire high quality image before they decide to view it or move on

- The wavelet transform combined with vector quantization has led to the development of experimental compression algorithms

- The general algorithm is as follows:

1. Perform the wavelet transform on the image by using convolution masks

2. Number the different wavelet bands from 0 to $N-1$, where N is the total number of wavelet bands, and 0 is the lowest frequency (in both horizontal and vertical directions) band

3. Scalar quantize the 0 band linearly to 8 bits

4. Vector quantize the middle bands using a small block size (e.g. 2x2). Decrease the codebook size as the band number increases

5. Eliminate the highest frequency bands

Figure 10.3-22: Wavelet/Vector Quantization Compression

a) Numbering for 7 bands

b) Numbering for 10 bands

- The example algorithms shown here utilize 10-band wavelet decomposition (Figure 10.3-22b), with the Daubecies 4 element basis vectors, in combination with the vector quantization technique

- They are called *Wavelet/Vector Quantization* followed by a number (WVQ#); specifically WVQ2, WVQ3 and WVQ4

- One algorithm (WVQ4) employs the PCT for preprocessing, before subsampling the second and third PCT bands by a factor of 2:1 in the horizontal and vertical direction

TABLE 10-2

Band Number	WVQ2		WVQ3		WVQ4 (PCT)	
	blocksize	# of bits	blocksize	# of bits	blocksiz e	# of bits
0	(scalar)	8	(scalar)	8	(scalar)	8
1, 2	2x2	8	2x2	8	2x2	8
3	2x2	6	2x2	6	2x2	6
4, 5	2x2	5	2x2	5	2x2	5
6	2x2	5	2x4	4	2x4	5
7, 8	2x4	4	X	0	2x4	5
9	X	0	X	0	X	0

- The table (10.2) lists the wavelet band numbers versus the three WVQ algorithms
- For each WVQ algorithm, we have a blocksize, which corresponds to the vector size, and the number of bits, which, for vector quantization, corresponds to the codebook size
- The lowest wavelet band is coded linearly using 8-bit scalar quantization

- Vector quantization is used for bands 1-8, where the number of bits per vector defines the size of the codebook
- The highest band is completely eliminated (0 bits are used to code them) in WVQ2 and WVQ4, while the highest three bands are eliminated in WVQ3
- For WVQ2 and WVQ3, each of the red, green and blue color planes are individually encoded using the parameters in the table

Figure 10.3-23: Wavelet/Vector Quantization (WVQ)Compression Example

a) Original image

b) WVQ2 compression ratio 10:1

c) Error of image (b)

Figure 10.3-23, continued

d) WVQ3 compression ratio 15:1

e) Error of image (d)

f) WVQ4 compression ratio 33:1

g) Error of image (f)

Figure 10.3.23 Wavelet/Vector Quantization (WVQ) Compression Example (contd)

h) WVQ4 compression ratio 36:1

i) Error of image (h)

- The JPEG2000 standard is also based on the wavelet transform
- It provides high quality images at very high compression ratios
- The committee that developed the standard had certain goals for JPEG2000

- The goals are as follows:

1. To provide better compression than the DCT-based JPEG algorithm
2. To allow for progressive transmission of high quality images
3. To be able to compress binary and continuous tone images by allowing 1 to 16 bits for image components

4. To allow random access to subimages

5. To be robust to transmission errors

6. To allow for sequentially image encoding

- The JPEG2000 compression method begins by level shifting the data to center it at zero, followed by an optional transform to decorrelate the data, such as a color transform for color images

- The one-dimensional wavelet transform is applied to the rows and columns, and the coefficients are quantized based on the image size and number of wavelet bands utilized

- These quantized coefficients are then arithmetically coded on a bitplane basis

Figure 10.3-24: The JPEG2000 Algorithm Applied to a Color Image

a) The original image

Figure 10.3-24: The JPEG2000 Algorithm Applied to a Color Image (contd)

b) Compression ratio = 130 ,
compare to Fig10.3-21e (next slide)

c) Compression ratio = 200,
compare to Fig10.3-21f

Figure 10.3-21:The Original DCT-based JPEG Algorithm Applied to a Color Image (contd)

e) Compression ratio = 131.03

f) Compression ratio = 201.39

Figure 10.3-24: The JPEG2000 Algorithm Applied to a Color Image (contd)

d) A 128x128 subimage cropped from the standard JPEG image and enlarged to 256x256 using zero-order hold

e) A 128x128 subimage cropped from the JPEG2000 image and enlarged to 256x256 using zero order hold

Note: The JPEG2000 image is much smoother, even with the zero-order hold enlargement