1

INTRODUCTION

This book is all about image and video compression. Chapter 1 simply introduces the overall ideas behind data compression by way of pictorial and graphical examples to motivate the readers. Detailed discussions on various compression schemes appear in subsequent chapters. One of the goals of this book is to present the basic principles behind image and video compression in a clear and concise manner and develop the necessary mathematical equations for a better understanding of the ideas. A further goal is to introduce the popular video compression standards such as Joint Photographic Experts Group (JPEG) and Moving Picture Experts Group (MPEG) and explain the compression tools used by these standards. Discussions on semantics and data transportation aspects of the standards will be kept to a minimum. Although the readers are expected to have an introductory knowledge in college-level mathematics and systems theory, clear explanations of the mathematical equations will be given where necessary for easy understanding. At the end of each chapter, problems are given in an increasing order of difficulty to make the understanding firm and lasting.

In order for the readers of this book to benefit further, MATLAB codes for several examples are included. To run the M-files on your computers, you should install MATLAB software. Although there are other software tools such as C++ and Python to use, MATLAB appears to be more readily usable because it has a lot of built-in functions in various areas such as signal processing, image and video processing, wavelet transform, and so on, as well as simulation tools such as MATLAB Simulink. Moreover, the main purpose of this book is to motivate the readers to learn and get hands on experience in video compression techniques with easy-to-use software tools, which does not require a whole lot of programming skills. In the

Still Image and Video Compression with MATLAB By K. S. Thyagarajan.

remainder of the chapter, we will briefly describe various compression techniques with some examples.

1.1 WHAT IS SOURCE CODING?

Images and videos are moved around the World Wide Web by millions of users almost in a nonstop fashion, and then, there is television (TV) transmission round the clock. Analog TV has been phased out since February 2009 and digital TV has taken over. Now we have the cell phone era. As the proverb *a picture is worth a thousand words* goes, the transmission of these visual media in digital form alone will require far more bandwidth than what is available for the Internet, TV, or wireless networks. Therefore, one must find ways to format the visual media data in such a way that it can be transmitted over the bandwidth-limited TV, Internet, and wireless channels in real time. This process of reducing the image and video data so that it fits into the available limited bandwidth or storage space is termed *data compression*. It is also called *source coding* in the communications field. When compressed audio/video data is actually transmitted through a transmission channel, extra bits are added to it to counter the effect of noise in the channel so that errors in the received data, if present, could be detected and/or corrected. This process of adding additional data bits to the compressed data stream before transmission is called *channel coding*. Observe that the effect of reducing the original source data in source coding is offset to a small extent by the channel coding, which adds data rather than reducing it. However, the added bits by the channel coder are very small compared with the amount of data removed by source coding. Thus, there is a clear advantage of compressing data.

We illustrate the processes of compressing and transmitting or storing a video source to a destination in Figure 1.1. The source of raw video may come from a video camera or from a previously stored video data. The source encoder compresses the raw data to a desired amount, which depends on the type of compression scheme chosen. There are essentially two categories of compression—*lossless* and *lossy*. In a lossless compression scheme, the original image or video data can be recovered exactly. In a lossy compression, there is always a loss of some information about the

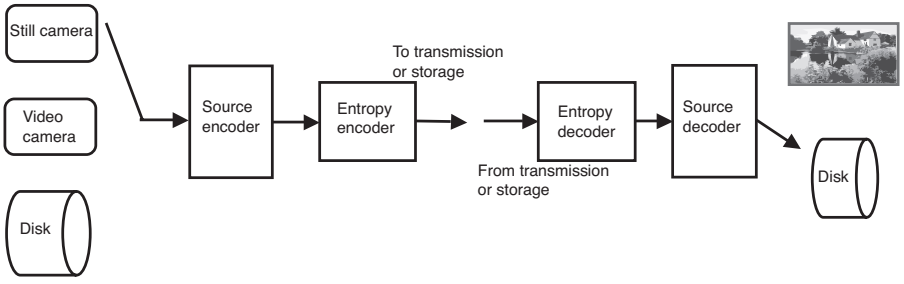


Figure 1.1 Source coding/decoding of video data for storage or transmission.

original data and so the recovered image or video data suffers from some form of distortion, which may or may not be noticeable depending on the type of compression used. After source encoding, the quantized data is encoded losslessly for transmission or storage. If the compressed data is to be transmitted, then channel encoder is used to add redundant or extra data bits and fed to the digital modulator. The digital modulator converts the input data into an RF signal suitable for transmission through a communications channel.

The communications receiver performs the operations of demodulation and channel decoding. The channel decoded data is fed to the entropy decoder followed by source decoder and is finally delivered to the sink or stored. If no transmission is used, then the stored compressed data is entropy decoded followed by source decoding as shown on the right-hand side of Figure 1.1.

1.2 WHY IS COMPRESSION NECESSARY?

An image or still image to be precise is represented in a computer as an array of numbers, integers to be more specific. An image stored in a computer is called a digital image. However, we will use the term image to mean a digital image. The image array is usually two dimensional (2D) if it is black and white (BW) and three dimensional (3D) if it is a color image. Each number in the array represents an intensity value at a particular location in the image and is called a picture element or pixel, for short. The pixel values are usually positive integers and can range between 0 and 255. This means that each pixel of a BW image occupies 1 byte in a computer memory. In other words, we say that the image has a grayscale resolution of 8 bits per pixel (bpp). On the other hand, a color image has a triplet of values for each pixel: one each for the red, green, and blue primary colors. Hence, it will need 3 bytes of storage space for each pixel. The captured images are rectangular in shape. The ratio of width to height of an image is called the aspect ratio. In standard-definition television (SDTV) the aspect ratio is 4:3, while it is 16:9 in a high-definition television (HDTV). The two aspect ratios are illustrated in Figure 1.2, where Figure 1.2a corresponds to an aspect ratio of 4:3 while Figure 1.2b corresponds to the same picture with an aspect ratio of 16:9. In both pictures, the height in inches remains the same,

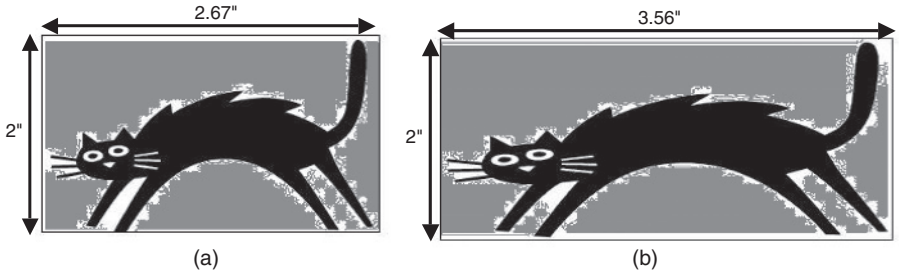


Figure 1.2 Aspect ratio: (a) 4:3 and (b) 16:9. The height is the same in both the pictures.

which means that the number of rows remains the same. So, if an image has 480 rows, then the number of pixels in each row will be $480 \times 4/3 = 640$ for an aspect ratio of 4:3. For HDTV, there are 1080 rows and so the number of pixels in each row will be $1080 \times 16/9 = 1920$. Thus, a single SD color image with 24 bpp will require $640 \times 480 \times 3 = 921{,}600$ bytes of memory space, while an HD color image with the same pixel depth will require $1920 \times 1080 \times 3 = 6{,}220{,}800$ bytes. A video source may produce 30 or more frames per second, in which case the raw data rate will be 221,184,000 bits per second for SDTV and 1,492,992,000 bits per second for HDTV. If this raw data has to be transmitted in real time through an ideal communications channel, which will require 1 Hz of bandwidth for every 2 bits of data, then the required bandwidth will be 110,592,000 Hz for SDTV and 746,496,000 Hz for HDTV. There are no such practical channels in existence that will allow for such a huge transmission bandwidth. Note that dedicated channels such as HDMI capable of transferring uncompressed data at this high rate over a short distance do exist, but we are only referring to long-distance transmission here. It is very clear that efficient data compression schemes are required to bring down the huge raw video data rates to manageable values so that practical communications channels may be employed to carry the data to the desired destinations in real time.

1.3 IMAGE AND VIDEO COMPRESSION TECHNIQUES

1.3.1 Still Image Compression

Let us first see the difference between *data compression* and *bandwidth compression*. Data compression refers to the process of reducing the digital source data to a desired level. On the other hand, bandwidth compression refers to the process of reducing the analog bandwidth of the analog source. What do we really mean by these terms? Here is an example. Consider the conventional wire line telephony. A subscriber's voice is filtered by a lowpass filter to limit the bandwidth to a nominal value of 4 kHz. So, the channel bandwidth is 4 kHz. Suppose that it is converted to digital data for long-distance transmission. As we will see later, in order to reconstruct the original analog signal that is band limited to 4 kHz exactly, sampling theory dictates that one should have at least 8000 samples per second. Additionally, for digital transmission each analog sample must be converted to a digital value. In telephony, each analog voice sample is converted to an 8-bit digital number using *pulse code modulation* (PCM). Therefore, the voice data rate that a subscriber originates is 64,000 bits per second. As we mentioned above, in an ideal case this digital source will require 32 kHz of bandwidth for transmission. Even if we employ some form of data compression to reduce the source rate to say, 16 kilobits per second, it will still require at least 8 kHz of channel bandwidth for real-time transmission. Hence, data compression does not necessarily reduce the analog bandwidth. Note that the original analog voice requires only 4 kHz of bandwidth. If we want to compress bandwidth, we can simply filter the analog signal by a suitable filter with a specified cutoff frequency to limit the bandwidth occupied by the analog signal.

Figure 1.3 Original cameraman picture.

Having clarified the terms data compression and bandwidth compression, let us look into some basic data compression techniques known to us. Henceforth, we will use the terms compression and data compression interchangeably. All image and video sources have redundancies. In a still image, each pixel in a row may have a value very nearly equal to a neighboring pixel value. As an example, consider the cameraman picture shown in Figure 1.3. Figure 1.4 shows the profile (top figure)

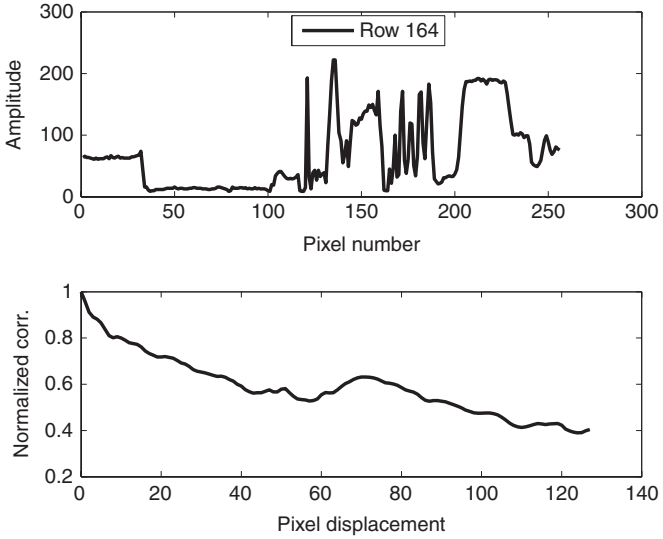


Figure 1.4 Profile of cameraman image along row number 164. The top graph shows pixel intensity, and the bottom graph shows corresponding normalized correlation over 128 pixel displacements.

and the corresponding correlation (bottom figure) of the cameraman picture along row 164. The MATLAB M-file for generating Figure 1.4 is listed below. Observe that the pixel values are very nearly the same over a large number of neighboring pixels and so is the pixel correlation. In other words, pixels in a row have a high correlation. Similarly, pixels may also have a high correlation along the columns. Thus, pixel redundancies translate to pixel correlation. The basic principle behind image data compression is to *decorrelate* the pixels and encode the resulting decorrelated image for transmission or storage. A specific compression scheme will depend on the method by which the pixel correlations are removed.

```
Figure1_4.m
%   Plots the image intensity profile and pixel correlation
%   along a specified row.
clear
close all
I = imread('cameraman.tif');
figure,imshow(I),title('Input image')
%
Row = 164; % row number of image profile
x = double(I(Row,:));
Col = size(I,2);
%
MaxN = 128; % number of correlation points to calculate
Cor = zeros(1,MaxN); % array to store correlation values
for k = 1:MaxN
    l = length(k:Col);
    Cor(k) = sum(x(k:Col) .* x(1:Col-k+1))/l;
end
MaxCor = max(Cor);
Cor = Cor/MaxCor;
figure,subplot(2,1,1),plot(1:Col,x,'k','LineWidth',2)
xlabel('Pixel number'), ylabel('Amplitude')
legend(['Row' ' ' num2str(Row)],0)
subplot(2,1,2),plot(0:MaxN-1,Cor,'k','LineWidth',2)
xlabel('Pixel displacement'), ylabel('Normalized corr.')
```

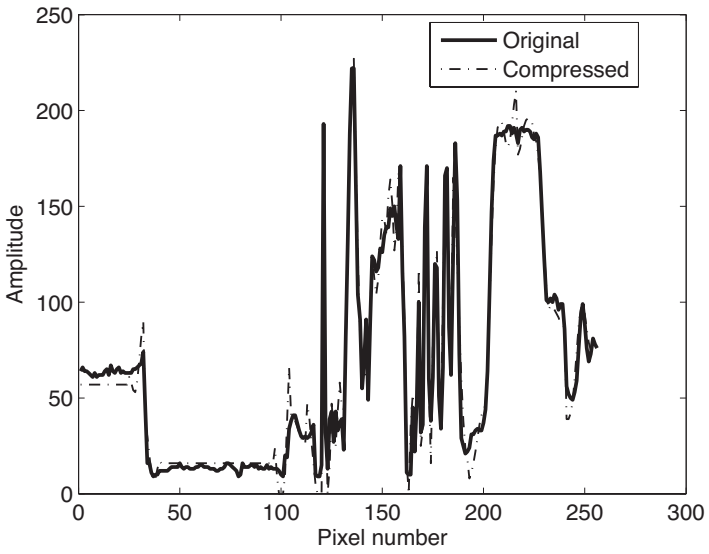
One of the earliest and basic image compression techniques is known as the *differential pulse code modulation* (DPCM) [1]. If the pixel correlation along only one dimension (row or column) is removed, then the DPCM is called one-dimensional (1D) DPCM or row-by-row DPCM. If the correlations along both dimensions are removed, then the resulting DPCM is known as 2D DPCM. A DPCM removes pixel correlation and requantizes the residual pixel values for storage or transmission. The residual image has a variance much smaller than that of the original image.

Further, the residual image has a probability density function, which is a double-sided exponential function. These give rise to compression.

The quantizer is fixed no matter how the decorrelated pixel values are. A variation on the theme is to use quantizers that adapt to changing input statistics, and therefore, the corresponding DPCM is called an adaptive DPCM. DPCM is very simple to implement, but the compression achievable is about 4:1. Due to limited bit width of the quantizer for the residual image, edges are not preserved well in the DPCM. It also exhibits occasional streaks across the image when channel error occurs. We will discuss DPCM in detail in a later chapter.

Another popular and more efficient compression scheme is known by the generic name *transform coding*. Remember that the idea is to reduce or remove pixel correlation to achieve compression. In transform coding, a block of image pixels is linearly transformed into another block of *transform coefficients* of the same size as the pixel block with the hope that only a few of the transform coefficients will be significant and the rest may be discarded. This implies that storage space is required to store only the significant transform coefficients, which are a fraction of the total number of coefficients and hence the compression. The original image can be reconstructed by performing the inverse transform of the reduced coefficient block. It must be pointed out that the inverse transform must exist for unique reconstruction. There are a number of such transforms available in the field to choose from, each having its own merits and demerits. The most efficient transform is one that uses the least number of transform coefficients to reconstruct the image for a given amount of distortion. Such a linear transform is known as the optimal transform where optimality is with respect to the minimum mean square error between the original and reconstructed images. This optimal image transform is known by the names *Karhunen–Loève transform* (KLT) or *Hotelling transform*. The disadvantage of the KLT is that the transform kernel depends on the actual image to be compressed, which requires a lot more side information for the receiver to reconstruct the original image from the compressed image than other *fixed* transforms. A highly popular fixed transform is the familiar *discrete cosine transform* (DCT). The DCT has very nearly the same *compression efficiency* as the KLT with the advantage that its *kernel* is fixed and so no side information is required by the receiver for the reconstruction. The DCT is used in the JPEG and MPEG video compression standards. The DCT is usually applied on nonoverlapping blocks of an image. Typical DCT blocks are of size 8×8 or 16×16. One of the disadvantages of image compression using the DCT is the *blocking* artifact. Because the DCT blocks are small compared with the image and because the average values of the blocks may be different, blocking artifacts appear when the zero-frequency (dc) DCT coefficients are quantized rather heavily. However, at low compression, blocking artifacts are almost unnoticeable. An example showing blocking artifacts due to compression using 8×8 DCT is shown in Figure 1.5a. Blockiness is clearly seen in flat areas—both low and high intensities as well as undershoot and overshoot along the sharp edges—see Figure 1.5b. A listing of M-file for Figures 1.5a,b is shown below.

(a)



(b)

Figure 1.5 (a) Cameraman image showing blocking artifacts due to quantization of the DCT coefficients. The DCT size is 8 × 8. (b) Intensity profile along row number 164 of the image in (a).

```
% Figure1_5.m
% Example to show blockiness in DCT compression
% Quantizes and dequantizes an intensity image using
% 8x8 DCT and JPEG quantization matrix
close all
clear
I = imread('cameraman.tif');
figure,imshow(I), title('Original Image')
%
fun = @dct2; % 2D DCT function
N = 8; % block size of 2D DCT
T = blkproc(I,[N N],fun);  % compute 2D DCT of image using NxN blocks
%
Scale = 4.0; % increasing Scale quntizes DCT coefficients heavily
% JPEG default quantization matrix
jpgQMat = [16 11 10 16  24  40  51  61;
           12 12 14 19  26  58  60  55;
           14 13 16 24  40  57  69  56;
           14 17 22 29  51  87  80  62;
           18 22 37 56  68 109 103  77;
           24 35 55 64  81 194 113  92;
           49 64 78 87 103 121 120 101;
           72 92 95 98 121 100 103  99];
Qstep = jpgQMat * Scale; % quantization step size
%   Quantize and dequantize the coefficients
for k = 1:N:size(I,1)
    for l = 1:N:size(I,2)
        T1(k:k+N-1,l:l+N-1) = round(T(k:k+N-1,l:l+N-1)./ Qstep).*Qstep;
    end
end
% do inverse 2D DCT
fun = @idct2;
y = blkproc(T1,[N N],fun);
y = uint8(round(y));
figure,imshow(y), title('DCT compressed Image')
% Plot image profiles before and after compression
ProfRow = 164;
figure,plot(1:size(I,2),I(ProfRow,:),'k','LineWidth',2)
hold on
plot(1:size(I,2),y(ProfRow,:),'-.k','LineWidth',1)
title(['Intensity profile of row ' num2str(ProfRow)])
xlabel('Pixel number'), ylabel('Amplitude')
%legend(['Row' ' ' num2str(ProfRow)],0)
legend('Original','Compressed',0)
```

A third and relatively recent compression method is based on *wavelet transform*. As we will see in a later chapter, wavelet transform captures both long-term and short-term changes in an image and offers a highly efficient compression mechanism. As a result, it is used in the latest versions of the JPEG standards as a compression tool. It is also adopted by the SMPTE (Society of Motion Pictures and Television Engineers). Even though the wavelet transform may be applied on blocks of an image like the DCT, it is generally applied on the full image and the various wavelet

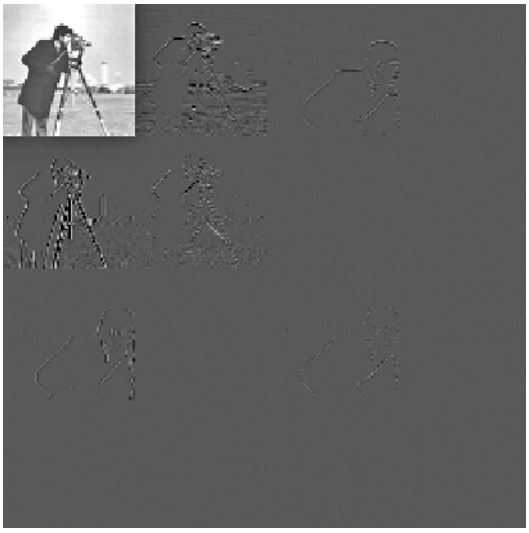
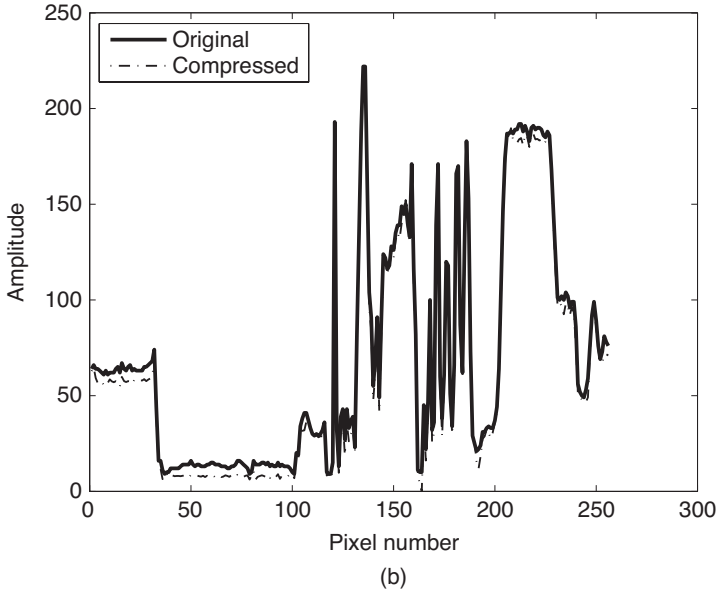


Figure 1.6 A two-level 2D DWT of cameraman image.

coefficients are quantized according to their types. A two-level discrete wavelet transform (DWT) of the cameraman image is shown in Figure 1.6 to illustrate how the 2D wavelet transform coefficients look like. Details pertaining to the levels and subbands of the DWT will be given in a later chapter. The M-file to implement multilevel 2D DWT that generates Figure 1.6 is listed below. As we will see in a later chapter, the 2D DWT decomposes an image into one approximation and many detail coefficients. The number of coefficient subimages corresponding to an L-level 2D DWT equals $3 \times L + 1$. Therefore, for a two-level 2D DWT, there are seven coefficient subimages. In the first level, there are three detail coefficient subimages, each of size $\frac{1}{4}$ the original image. The second level consists of four sets of DWT coefficients—one approximation and three details, each $\frac{1}{16}$ the original image. As the name implies the approximation coefficients are lower spatial resolution approximations to the original image. The detail coefficients capture the discontinuities or edges in the image with orientations in the horizontal, vertical, and diagonal directions. In order to compress, an image using 2D DWT we have to compute the 2D DWT of the image up to a given level and then quantize each coefficient subimage. The achievable quality and compression ratio depend on the chosen wavelets and quantization method. The visual effect of quantization distortion in DWT compression scheme is different from that in DCT-based scheme. Figure 1.7a is the cameraman image compressed using 2D DWT. The wavelet used is called Daubechies 2 (db2 in MATLAB) and the number of levels used is 1. We note that there are no blocking effects, but there are patches in the flat areas. We also see that the edges are reproduced faithfully as evidenced in the profile (Figure 1.7b). It must be pointed out that the amount of quantization applied in Figure 1.7a is not the same as that used for the DCT example and that the two examples are given only to show the differences in the artifacts introduced by the two schemes. An M-file listing to generate Figures 1.7a,b is shown below.

(a)



(b)

Figure 1.7 (a) Cameraman image compressed using one-level 2D DWT. (b) Intensity profile of image in Figure 1.8a along row number 164.

```
% Figure1_6.m
% 2D Discrete Wavelet Transform (DWT)
% Computes multi-level 2D DWT of an intensity image
close all
clear
I = imread('cameraman.tif');
figure,imshow(I), title('Original Image')
L = 2; % number of levels in DWT
[W,B] = wavedec2(I,L,'db2'); % do a 2-level DWT using db2 wavelet
% declare level-1 subimages
w11 = zeros(B(3,1),B(3,1));
w12 = zeros(B(3,1),B(3,1));
w13 = zeros(B(3,1),B(3,1));
% declare level-2 subimages
w21 = zeros(B(1,1),B(1,1));
w22 = zeros(B(1,1),B(1,1));
w23 = zeros(B(1,1),B(1,1));
w24 = zeros(B(1,1),B(1,1));
% extract level-1 2D DWT coefficients
offSet11 = 4*B(1,1)*B(1,2);
offSet12 = 4*B(1,1)*B(1,2)+B(3,1)*B(3,2);
offSet13 = 4*B(1,1)*B(1,2)+2*B(3,1)*B(3,2);
for c = 1:B(2,2)
    for r = 1:B(2,1)
        w11(r,c) = W(offSet11+(c-1)*B(3,1)+r);
        w12(r,c) = W(offSet12+(c-1)*B(3,1)+r);
        w13(r,c) = W(offSet13+(c-1)*B(3,1)+r);
    end
end
% extract level-2 2D DWT coefficients
offSet22 = B(1,1)*B(1,2);
offSet23 = 2*B(1,1)*B(1,2);
offSet24 = 3*B(1,1)*B(1,2);
for c = 1:B(1,2)
    for r = 1:B(1,1)
        w21(r,c) = W((c-1)*B(1,1)+r);
        w22(r,c) = W(offSet22+(c-1)*B(1,1)+r);
        w23(r,c) = W(offSet23+(c-1)*B(1,1)+r);
        w24(r,c) = W(offSet24+(c-1)*B(1,1)+r);
    end
end
% declare output array y to store all the DWT coefficients
%y = zeros(261,261);
y = zeros(2*B(1,1)+B(3,1),2*B(1,2)+B(3,2));
y(1:B(1,1),1:B(1,1))=w21;
y(1:B(1,1),B(1,1)+1:2*B(1,1))=w22;
y(B(1,1)+1:2*B(1,1),1:B(1,1))=w23;
y(B(1,1)+1:2*B(1,1),B(1,1)+1:2*B(1,1))=w24;
%
y(1:B(3,1),2*B(1,1)+1:261)=w11;
y(2*B(1,1)+1:261,1:129)=w12;
y(2*B(1,1)+1:261,2*B(1,1)+1:261)=w13;
figure,imshow(y,[]),title([num2str(L) '-level 2D DWT'])


% Figure1_7.m
% An example to show the effect of quantizing the 2D DWT
% coefficients of an intensity image along with intensity
% profile along a specified row
```

```
%
close all
clear
I = imread('cameraman.tif');
figure,imshow(I), title('Original Image')
% do a 1-level 2D DWT
[W,B] = wavedec2(I,1,'db2');
w11 = zeros(B(1,1),B(1,2));
w12 = zeros(B(1,1),B(1,2));
w13 = zeros(B(1,1),B(1,2));
w14 = zeros(B(1,1),B(1,2));
%
offSet12 = B(1,1)*B(1,2);
offSet13 = 2*B(1,1)*B(1,2);
offSet14 = 3*B(1,1)*B(1,2);
% quantize only the approximation coefficients
Qstep = 16;
for c = 1:B(1,2)
    for r = 1:B(1,1)
        W((c-1)*B(1,1)+r) = floor(W((c-1)*B(1,1)+r)/Qstep)*Qstep;
        %{
        W(offSet12+(c-1)*B(1,1)+r) = floor(W(offSet12+(c-1)*B(1,1)+r)/8)*8;
        W(offSet13+(c-1)*B(1,1)+r) = floor(W(offSet13+(c-1)*B(1,1)+r)/8)*8;
        W(offSet14+(c-1)*B(1,1)+r) = floor(W(offSet14+(c-1)*B(1,1)+r)/8)*8;
        %}
    end
end
% do inverse 2D DWT
y = waverec2(W,B,'db2');
figure,imshow(y,[])
% plot profile
ProfRow = 164;
figure,plot(1:size(I,2),I(ProfRow,:),'k','LineWidth',2)
hold on
plot(1:size(I,2),y(ProfRow,:),'-.k','LineWidth',1)
title(['Profile of row ' num2str(ProfRow)])
xlabel('Pixel number'), ylabel('Amplitude')
%legend(['Row' ' ' num2str(ProfRow)],0)
legend('Original','Compressed',0)
```

1.3.2 Video Compression

So far our discussion on compression has been on still images. These techniques try to exploit the *spatial correlation* that exists in a still image. When we want to compress video or sequence images we have an added dimension to exploit, namely, the temporal dimension. Generally, there is little or very little change in the spatial arrangement of objects between two or more consecutive *frames* in a video. Therefore, it is advantageous to send or store the differences between consecutive frames rather than sending or storing each frame. The difference frame is called the *residual* or *differential* frame and may contain far less details than the actual frame itself. Due to this reduction in the details in the differential frames, compression is achieved. To illustrate the idea, let us consider compressing two consecutive frames (frame 120 and frame 121) of a video sequence as shown in Figures 1.8a,b, respectively (see M-file listing shown below). The difference between frames 121 and 120 is shown

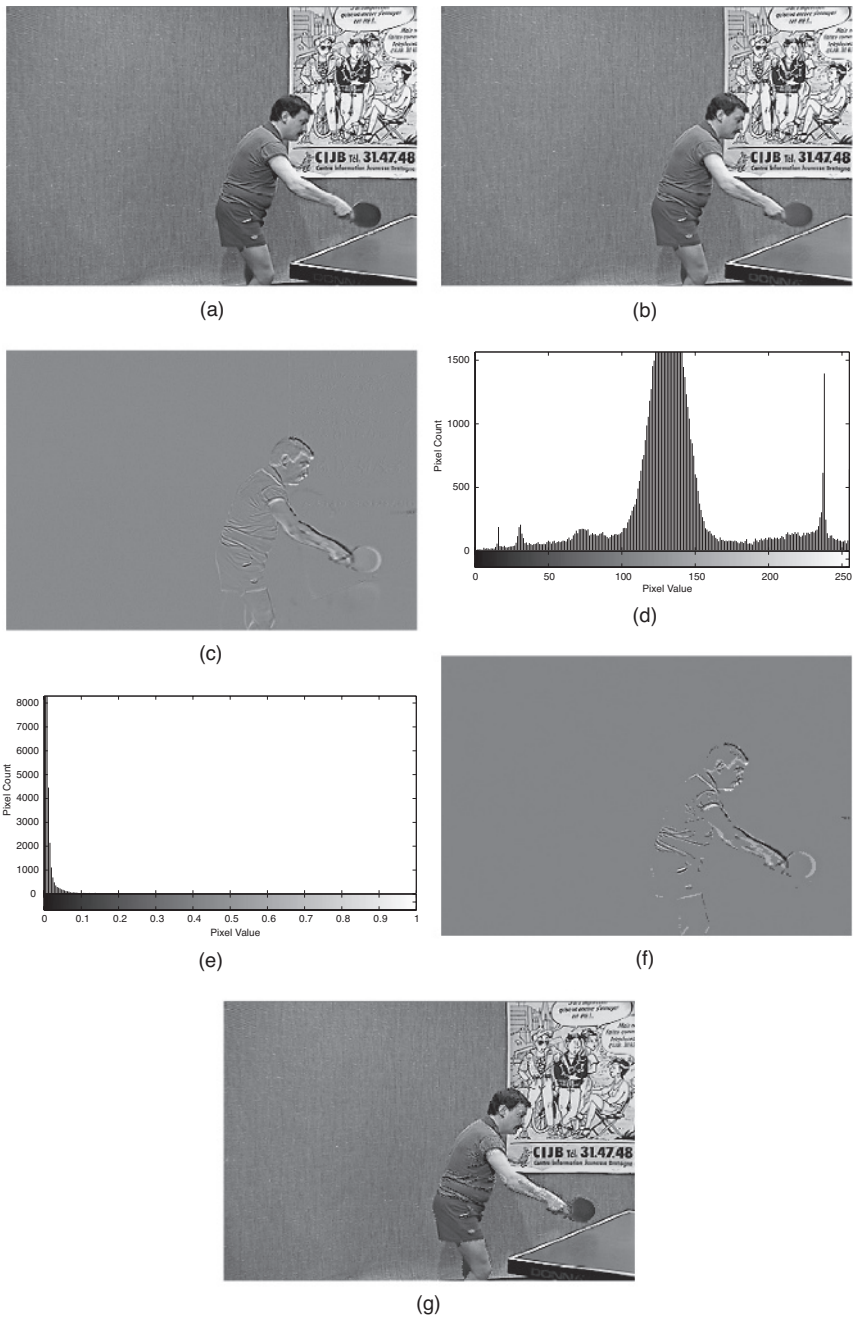


Figure 1.8 Compressing a video sequence: (a) frame 120 of a table tennis video sequence; (b) frame 121 of the video sequence; (c) difference between frames 121 and 120; (d) histogram of frame 121; (e) histogram of the difference of frames; (g) quantized difference frame; and (h) Reconstructed frame 121 by adding the quantized difference frame to frame 120.

in Figure 1.8c. The differential frame has a small amount of details corresponding to the movements of the hand and the racket. Note that stationary objects do not appear in the difference frame. This is evident from the histogram of the differential frame shown in Figure 1.8e, where the intensity range occupied by the differential pixels is much smaller. Compare this with the histogram of frame 121 in Figure 1.8d which is much wider. The quantized differential frame and the reconstructed frame 121 are shown in Figures 1.8f,g, respectively. We see some distortions in the edges due to quantization.

When objects move between successive frames, simple differencing will introduce large residual values especially when the motion is large. Due to relative motion of objects, simple differencing is not efficient from the point of view of achievable compression. It is more advantageous to determine or estimate the relative motions of objects between successive frames and compensate for the motion and then do the differencing to achieve a much higher compression. This type of prediction is known as *motion compensated prediction*. Because we perform motion *estimation* and *compensation* at the encoder, we need to inform the decoder about this motion compensation. This is done by sending motion vectors as side information, which conveys the object motion in the horizontal and vertical directions. The decoder then uses the motion vectors to align the blocks and reconstruct the image.

```
% Figure1_8.m
% generates a differential frame by subtracting two
% temporally adjacent intensity image frames
% quantizes the differential frame and reconstructs
% original frame by adding quantized differential frame
% to the other frame.
close all
clear
Frm1 = 'tt120.ras';
Frm2 = 'tt121.ras';
I = imread(Frm1); % read frame # 120
I1 = im2single(I); % convert from uint8 to float single
I = imread(Frm2); % read frame # 121
figure,imhist(I,256),title(['Histogram of frame ' num2str(121)])
xlabel('Pixel Value'), ylabel('Pixel Count')
I2 = im2single(I); % convert from uint8 to float single
clear I
figure,imshow(I1,[]), title([num2str(120) 'th frame'])
figure,imshow(I2,[]), title([num2str(121) 'st frame'])
%
Idiff = imsubtract(I2,I1); % subtract frame 120 from 121
figure,imhist(Idiff,256),title('Histogram of difference image')
xlabel('Pixel Value'), ylabel('Pixel Count')
figure,imshow(Idiff,[]),title('Difference image')
% quantize and dequantize the differential image
IdiffQ = round(4*Idiff)/4;
figure,imshow(IdiffQ,[]),title('Quantized Difference image')
y = I1 + IdiffQ; % reconstruct frame 121
figure,imshow(y,[]),title('Reconstructed image')
```

A video sequence is generally divided into scenes with scene changes marking the boundaries between consecutive scenes. Frames within a scene are similar and there is a high temporal correlation between successive frames within a scene. We may, therefore, send differential frames within a scene to achieve high compression. However, when the scene changes, differencing may result in much more details than the actual frame due to the absence of correlation, and therefore, compression may not be possible. The first frame in a scene is referred to as the *key frame*, and it is compressed by any of the above-mentioned schemes such as the DCT or DWT. Other frames in the scene are compressed using temporal differencing. A detailed discussion on video compression follows in a later chapter.

1.3.3 Lossless Compression

The above-mentioned compression schemes are lossy because there is always a loss of some information when reconstructing the image from the compressed image. There is another category of compression methods wherein the image decoding or reconstruction is exact, that is, there is no loss of any information about the original image. Although this will be very exciting to a communications engineer, the achievable *compression ratio* is usually around 2:1, which may not always be sufficient from a storage or transmission point of view. However, there are situations where lossless image and video compression may be necessary. Digital mastering of movies is done by lossless compression. After editing a movie, it is compressed with loss for distribution. In telemedicine, medical images need to be compressed losslessly so as to enable a physician at a remote site to diagnose unambiguously. In general, a lossless compression scheme considers an image to consist of a finite *alphabet* of discrete *symbols* and relies on the probability of occurrence of these symbols to achieve lossless compression. For instance, if the image pixels have values between 0 and 255, then the alphabet consists of 256 symbols one for each integer value with a characteristic probability distribution that depends on the source. We can then generate binary *codeword* for each symbol in the alphabet, wherein the code length of a symbol increases with its decreasing probability in a logarithmic fashion. This is called a *variable-length coding*. Huffman coding [2] is a familiar example of variable-length coding. It is also called *entropy coding* because the average code length of a large sequence approaches the entropy of the source. As an example, consider a discrete source with an alphabet $A = \{a_1, a_2, a_3, a_4\}$ with respective probabilities of $\frac{1}{8}$, $\frac{1}{2}$, $\frac{1}{8}$, and $\frac{1}{4}$. Then one possible set of codes for the symbols is shown in the table below. These are variable-length codes, and we see that no code is a prefix to other codes. Hence, these codes are also known as prefix codes. Observe that the most likely symbol a_2 has the least code length and the least probable symbols a_1 and a_3 have the largest code length. We also find that the average number of bits per symbol is 1.75, which happens to be the entropy of this source. We will discuss source entropy in detail in Chapter 5. One drawback of Huffman coding is that the entire codebook must be available at the decoder. Depending on the number of codewords, the amount of side information about the codebook to be transmitted may be very large and the coding efficiency may be reduced. If the number

Table 1.1 Variable-length codes

Symbol	Probability	Code
a_1	1/8	110
a_2	1/2	0
a_3	1/8	111
a_4	1/4	10

of symbols is small, then we can use a more efficient lossless coding scheme called *arithmetic coding*. Arithmetic coding does not require the transmission of codebook and so achieves a higher compression than Huffman coding would. For compressing textual information, there is an efficient scheme known as *Lempel–Ziv* (LZ) *coding* [3] method. As we are concerned only with image and video compression here, we will not discuss LZ method further.

With this short description of the various compression methods for still image and video, we can now look at the plethora of compression schemes in a tree diagram as illustrated in Figure 1.9. It should be pointed out that lossless compression is always included as part of a lossy compression even though it is not explicitly shown in the figure. It is used to losslessly encode the various quantized pixels or transform coefficients that take place in the compression chain.

1.4 VIDEO COMPRESSION STANDARDS

Interoperability is crucial when different platforms and devices are involved in the delivery of images and video data. If for instance images and video are compressed using a proprietary algorithm, then decompression at the user end is not feasible unless the same proprietary algorithm is used, thereby encouraging monopolization.

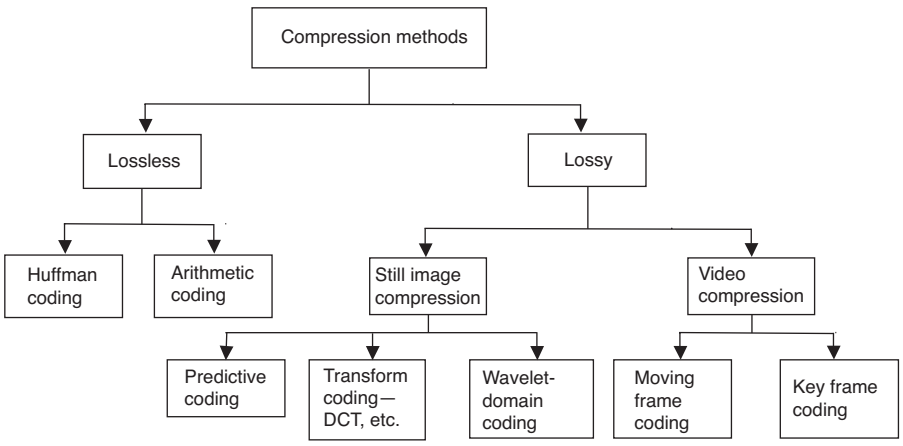


Figure 1.9 A taxonomy of image and video compression methods.

This, therefore, calls for a standardization of the compression algorithms as well as data transportation mechanism and protocols so as to guarantee not only interoperability but also competitiveness. This will eventually open up growth potential for the technology and will benefit the consumers as the prices will go down. This has motivated people to form organizations across nations to develop solutions to interoperability.

The first successful standard for still image compression known as JPEG was developed jointly by the International Organization for Standardization (ISO) and International Telegraph and Telephone Consultative Committee (CCITT) in a collaborative effort. CCITT is now known as International Telecommunication Union—Telecommunication (ITU-T). JPEG standard uses DCT as the compression tool for grayscale and true color still image compression. In 2000, JPEG [4] adopted 2D DWT as the compression vehicle.

For video coding and distribution, MPEG was developed under the auspicious of ISO and International Electrotechnical Commission (IEC) groups. MPEG [5] denotes a family of standards used to compress audio-visual information. Since its inception MPEG standard has been extended to several versions. MPEG-1 was meant for video compression at about 1.5 Mb/s rate suitable for CD ROM. MPEG-2 aims for higher data rates of 10 Mb/s or more and is intended for SD and HD TV applications. MPEG-4 is intended for very low data rates of 64 kb/s or less. MPEG-7 is more on standardization of description of multimedia information rather than compression. It is intended for enabling efficient search of multimedia contents and is aptly called *multimedia content description interface*. MPEG-21 aims at enabling the use of multimedia sources across many different networks and devices used by different communities in a transparent manner. This is to be accomplished by defining the entire multimedia framework as *digital items*. Details about various MPEG standards will be given in Chapter 10.

1.5 ORGANIZATION OF THE BOOK

We begin with image acquisition techniques in Chapter 2. It describes image sampling and quantization schemes followed by various color coordinates used in the representation of color images and various video formats. Unitary transforms, especially the DCT, are important compression vehicles, and so in Chapter 3, we will define the unitary transforms and discuss their properties. We will then describe image transforms such as KLT and DCT and illustrate their merits and demerits by way of examples. In Chapter 4, 2D DWT will be defined along with methods of its computation as it finds extensive use in image and video compression. Chapter 5 starts with a brief description of information theory and source entropy and then describes lossless coding methods such as Huffman coding and arithmetic coding with some examples. It also shows examples of constructing Huffman codes for a specific image source. We will then develop the idea of predictive coding and give detailed descriptions of DPCM in Chapter 6 followed by transform domain coding procedures for still image compression in Chapter 7. Chapter 7 also describes JPEG standard for still

image compression. Chapter 8 deals with image compression in the wavelet domain as well as JPEG2000 standard. Video coding principles will be discussed in Chapter 9. Various motion estimation techniques will be described in Chapter 9 with several examples. Compression standards such as MPEG will be discussed in Chapter 10 with examples.

1.6 SUMMARY

Still image and video sources require wide bandwidths for real-time transmission or large storage memory space. Therefore, some form of data compression must be applied to the visual data before transmission or storage. In this chapter, we have introduced terminologies of lossy and lossless methods of compressing still images and video. The existing lossy compression schemes are DPCM, transform coding, and wavelet-based coding. Although DPCM is very simple to implement, it does not yield high compression that is required for most image sources. It also suffers from distortions that are objectionable in applications such as HDTV. DCT is the most popular form of transform coding as it achieves high compression at good visual quality and is, therefore, used as the compression vehicle in JPEG and MPEG standards. More recently 2D DWT has gained importance in video compression because of its ability to achieve high compression with good quality and because of the availability of a wide variety of wavelets. The examples given in this chapter show how each one of these techniques introduces artifacts at high compression ratios.

In order to reconstruct images from compressed data without incurring any loss whatsoever, we mentioned two techniques, namely, Huffman and arithmetic coding. Even though lossless coding achieves only about 2:1 compression, it is necessary where no loss is tolerable, as in medical image compression. It is also used in all lossy compression systems to represent quantized pixel values or coefficient values for storage or transmission and also to gain additional compression.

REFERENCES

1. J. B. O'Neal, Jr., "Differential pulse code modulation (DPCM) with entropy coding," *IEEE Trans. Inf. Theory,* IT-21 (2), 169–174, 1976.
2. D. A. Huffman, "A method for the construction of minimum redundancy codes," *Proc. IRE*, 40, 1098–1101, 1951.
3. J. Ziv and A. Lempel, "Compression of individual sequences via variable-rate coding," *IEEE Trans. Inf. Theory*, IT-24 (5), 530–536, 1978.
4. JPEG, *Part I: Final Draft International Standard* (ISO/IEC FDIS15444–1), ISO/IEC JTC1/SC29/WG1N1855, 2000.
5. MPEG, URL for MPEG organization is http://www.mpeg.org/.