4th Class

2023-2024

Multimedia Data Compression

ضغط الوسائط المتعددة

أستاذ المادة : م.م. زينب علي

1. Image Compression

Historically, data compression was not one of the first fields of computer science. It seems that workers in the field needed the first 20 to 25 years to develop enough data before they felt the need for compression. Today, when the computer field is about 50 years old, data compression is a large and active field, as well as big business.

A digital image is a rectangular array of dots, or picture elements, arranged in m rows and n columns. The expression m × n is called the resolution of the image, and the dots are called pixels (except in the cases of fax images and video compression, where they are referred to as pels). The term "resolution" is sometimes also used to indicate the number of pixels per unit length of the image. Thus, dpi stands for dots per inch. For the purpose of image compression, it is useful to distinguish the following types of images:

1. A bi-level (or monochromatic) image. This is an image where the pixels can have one of two values, normally referred to as black and white.
2. A grayscale image.

3. A continuous-tone image. This type of image can have many similar colors (or grayscales). When adjacent pixels differ by just one unit, it is hard or even impossible for the eye to distinguish their colors. A continuous-tone image is obtained by taking a photograph with a digital camera, or by scanning a photograph or a painting. Figures 1 through 4 are typical examples of continuous-tone images.

4. A discrete-tone image (also called a graphical image or a synthetic image). This is normally an artificial image. Examples are an artificial object or machine, a page of text, a chart, a cartoon, or the contents of a computer screen. Figure 5 is a typical example of a discrete-tone image.

5. A cartoon-like image. This is a color image that consists of uniform areas. Each area has a uniform color but adjacent areas may have very different colors. This feature may be exploited to obtain excellent compression.

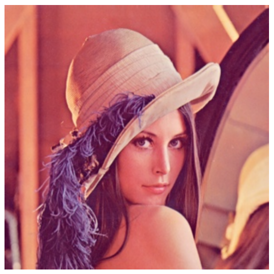
Fig. (1)

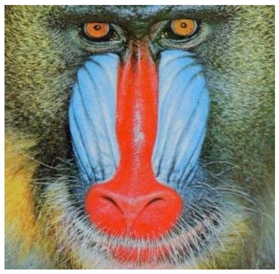
Fig. (2)

Fig. (3)

Fig. (4)

A continuous-tone images

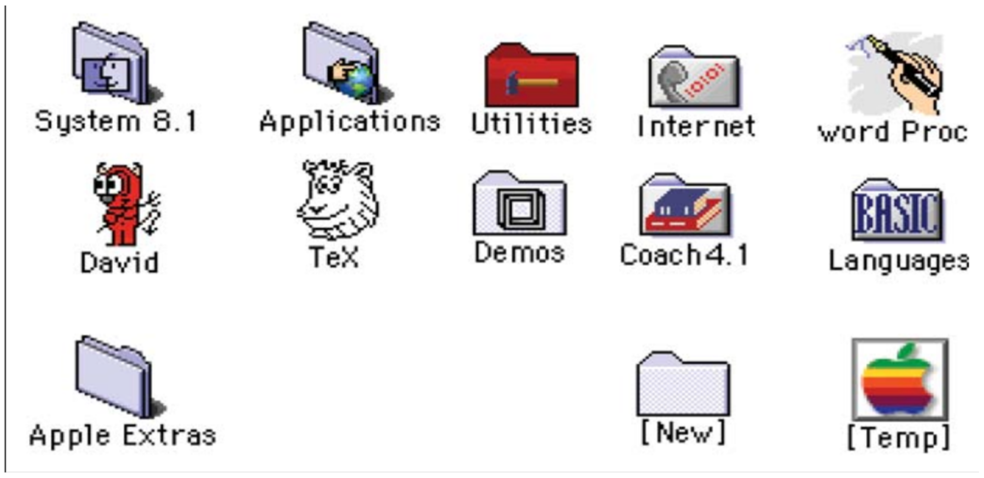

Fig. (5) A Discrete-Tone Image.

An image, after all, exists for people to look at, so, when it is compressed, it is acceptable to lose image features to which the eye is not sensitive. This is one of the main ideas behind the many lossy image compression methods.

In general, information can be compressed if it is redundant. It has been mentioned several times that data compression amounts to reducing or removing redundancy in the data. With lossy compression, however, we have a new concept, namely compressing by removing irrelevancy. An image can be lossy-compressed by removing irrelevant information even if the original image does not have any redundancy.

(Digitizing an image involves two steps: sampling and quantization. Sampling an image is the process of dividing the two-dimensional original image into small region of pixels. Quantization is the process of assigning an integer value to each pixel. Notice that digitizing sound involves the same two steps, with the difference that sound is one-dimensional.)

1.1 JPEG Compression

JPEG (Joint Photographic Experts Group) is an advance lossy/lossless compression method for color or grayscale still images (not videos). It does not handle bi-level (black and white) images very well. It also works best on continuous-tone images, where adjacent pixels have similar colors.

An important feature of JPEG is its use of many parameters, allowing the user to adjust the amount of the data lost (and thus also the compression ratio) over a very wide range. Often, the eye cannot see any image degradation even at compression factors of 10 or 20.

There are two operating modes, lossy (also called baseline) and lossless (which typically produces compression ratios of around 0.5). Most implementations support just the lossy mode. This mode includes progressive and hierarchical coding.

The main goals of JPEG compression are the following:

1. High compression ratios.
2. The use of many parameters.
3. Obtaining good results with any kind of continuous-tone image, regardless of image dimensions, color spaces, pixel aspect ratios, or other image features.
4. A sophisticated, but not too complex compression method, allowing software and hardware implementations on many platforms.
5. Several modes of operation:

(a) A sequential mode where each image component (color) is compressed in a single left-to-right, top-to-bottom scan;

(b) A progressive mode where the image is compressed in multiple blocks (known as "scans") to be viewed from coarse to fine detail;

(c) A lossless mode that is important in cases where the user decides that no pixels should be lost.

(d) A hierarchical mode where the image is compressed at multiple resolutions allowing lower-resolution blocks to be viewed without first having to decompress the following higher-resolution blocks.

The main JPEG compression steps are:

1. Color images are transformed from RGB a luminance/chrominance color space. The eye is sensitive to small changes in luminance but not in chrominance, so the chrominance part can later lose much data, and thus be highly compressed, without visually weaken the overall image quality much.

2. Color images are down-sampled by creating low-resolution pixels from the original ones (this step is used only when hierarchical compression is selected; it is always skipped for grayscale images). The down-sampling is not done for the luminance component. Since the luminance component is not touched, there is no noticeable loss of image quality.

3. The pixels of each color component are organized in groups of 8×8 pixels called data units, and each data unit is compressed separately. If the number of image rows or columns is not a multiple of 8, the bottom row and the rightmost column are duplicated as many times as necessary. The fact that each data unit is compressed separately is one of the drawbacks of JPEG. If the user asks for maximum compression, in the decompressed image Human manipulation of will appear due to differences between blocks. Figure 6 is an extreme example of this effect.

4. The discrete cosine transform (DCT) is then applied to each data unit to create an 8×8 map of frequency components. They represent the average pixel value and successive higher-frequency changes within the group. This prepares the image data for the step of losing information. Since DCT involves the function cosine, it must involve some loss of information due to the limited accuracy of computer arithmetic. This means that even without the main lossy step (step 5 below), there will be some loss of image quality, but it is normally small.

5. Each of the 64 frequency components in a data unit is divided by a separate number called its quantization coefficient (QC), and then rounded to an integer. This is where information is permanently lost. Large QCs cause more loss, so the high frequency components typically have larger QCs. Each of the 64 QCs is a JPEG parameter and can, in principle, be specified by the user. In practice, most JPEG implementations use the QC tables recommended by the JPEG standard for the luminance and chrominance image components (Table 1).

6. The 64 quantized frequency coefficients (which are now integers) of each data unit are encoded using a combination of RLE and Huffman coding.

7. The last step adds headers and all the required JPEG parameters, and outputs the result.

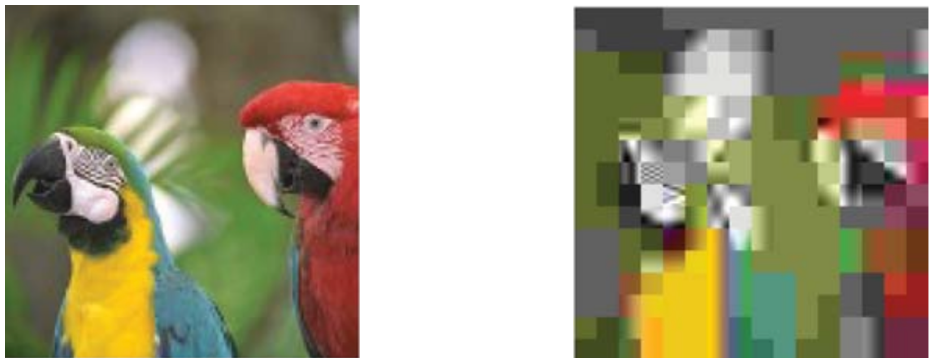
Fig. (6) JPEG Blocking Artifacts

The compressed file may be in one of three formats:

(1) the interchange format, in which the file contains the compressed image and all the tables needed by the decoder (mostly quantization tables and tables of Huffman codes).

(2) the abbreviated format for compressed image data, where the file contains the compressed image and may contain no tables (or just a few tables).

(3) the abbreviated format for table-specification data, where the file contains just tables, and no compressed image.

The second format makes sense in cases where the same encoder/decoder pair is used, and they have the same tables built in. The third format is used in cases where many images have been compressed by the same encoder, using the same tables. When those images need to be decompressed, they are sent to a decoder preceded by one file with table-specification data. Figure 7 shows JPEG diagram.

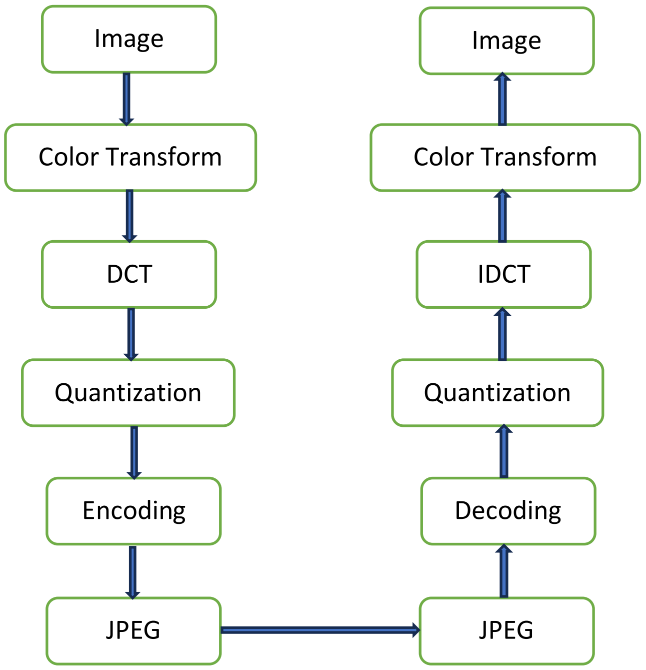

Fig. (7) JPEG Block Diagram

1.1.1 Luminance

Luminance is proportional to the power of the light source. It is similar to intensity, but the spectral composition of luminance is related to the brightness sensitivity of human vision.

The eye is very sensitive to small changes in luminance, which is why it is useful to have color spaces that use Y as one of their three parameters. A simple way to do this is to subtract Y from the Blue and Red components of RGB, and use the three components Y, $B - Y$, and $R - Y$ as a new color space. The last two components are called chroma. They represent color in terms of the presence or absence of blue (Cb) and red (Cr) for a given luminance intensity.

Y is defined to have a range of 16 to 235; Cb and Cr are defined to have a range of 16 to 240, with 128 equal to zero.

Transforming RGB to YCbCr is done by (note the small weight of blue):

$$\boxed{\begin{aligned} Y &= (77/256)R + (150/256)G + (29/256)B \\ Cb &= -(44/256)R - (87/256)G + (131/256)B + 128 \\ Cr &= (131/256)R - (110/256)G - (21/256)B + 128 \end{aligned}}$$

while the opposite transformation is

$$\boxed{\begin{aligned} R &= Y + 1.371(Cr - 128) \\ G &= Y - 0.698(Cr - 128) - 0.336(Cb - 128) \\ B &= Y + 1.732(Cb - 128) \end{aligned}}$$

When performing YCbCr to RGB conversion, the resulting RGB values have a nominal range of 16-235, with possible occasional values in 0-15 and 236-255.

1.1.2 The Discrete Cosine Transform

The JPEG committee elected to use the DCT because of its good performance, because it does not assume anything about the structure of the data and because there are ways to speed it up.

The JPEG standard calls for applying the DCT not to the entire image but to data units (blocks) of 8 × 8 pixels figure 8 shows Graphical illustration for (8*8) 2D DCT. The reasons for this are:

(1) Applying DCT to large blocks involves many arithmetic operations and is therefore slow. Applying DCT to small data units is faster.

(2) Experience shows that, in a continuous-tone image, correlations between pixels are short range. A pixel in such an image has a value (color component or shade of gray) that' s close to those of its near neighbors, but has nothing to do with the values of far neighbors. The JPEG DCT is therefore executed by Equation (1), duplicated here for n = 8.

$$G_{ij} = \frac{1}{4} C_i C_j \sum_{u=0}^{7} \sum_{v=0}^{7} p_{uv} \cos\left(\frac{(2u+1)i\pi}{16}\right) \cos\left(\frac{(2v+1)j\pi}{16}\right) \qquad \text{Eq. (1)}$$

$$\textit{where } i, j, u, v = 0 \ldots..7 \textit{ and the constant } C_i C_j \textit{ determined by}$$

$$C_f = \begin{cases} \frac{1}{\sqrt{2}}, & f = 0 \\ 1, & f > 0 \end{cases} \quad \textit{and } 0 \le \mathrm{i,j} \le 7.$$

The JPEG decoder works by computing the inverse DCT (IDCT), Equation (2), duplicated here for n = 8

$$p_{uv} = \frac{1}{4} \sum_{i=0}^{7} \sum_{j=0}^{7} C_i C_j G_{ij} \cos\left(\frac{(2u+1)i\pi}{16}\right) \cos\left(\frac{(2v+1)j\pi}{16}\right) \qquad \text{Eq. (2)}$$

$$\textit{where } i, j, u, v = 0 \ldots..7 \textit{ and the constant } C_i C_j \textit{ determined by}$$

$$C_f = \begin{cases} \frac{1}{\sqrt{2}}, & f = 0 \\ 1, & f > 0 \end{cases} \quad \textit{and } 0 \le \mathrm{i,j} \le 7.$$

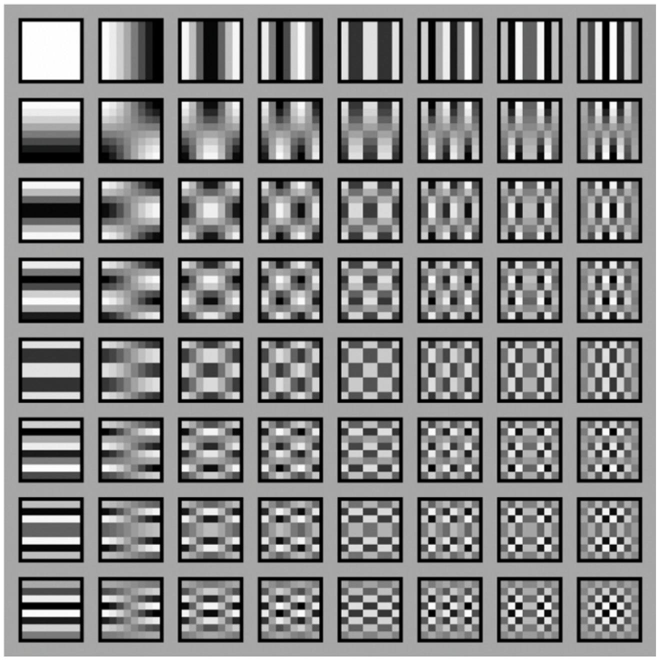
Fig.(8) Graphical illustration for (8*8) 2D DCT

The DCT is JPEG’s key to lossy compression. The unimportant image information is reduced or removed by quantizing the 64 DCT coefficients, especially the ones located toward the lower-right. If the pixels of the image are correlated, quantization does not degrade the image quality much. For best results, each of the 64 coefficients is quantized by dividing it by a different quantization coefficient (QC). All 64 QCs are parameters that can be controlled, in principle, by the user.

1.1.3 Quantization

After each 8×8 data unit of DCT coefficients Gij is computed, it is quantized. This is the step where information is lost (except for some unavoidable loss because of finite precision calculations in other steps). Each number in the DCT coefficients matrix is divided by the corresponding number from the particular “quantization table” used, and the result is rounded to the nearest integer.

Three such tables are needed, for the three color components. The JPEG standard allows for up to four tables, and the user can select any of the four for quantizing each color component. The 64 numbers that constitute each quantization table are all JPEG parameters. In principle, they can all be specified and get the desired performance by the user for maximum compression. In practice, few users have the patience or expertise to experiment with so many parameters, so JPEG software normally uses the following two approaches:

1. Default quantization tables. Two such tables, for the luminance (grayscale) and the chrominance components, are the result of many experiments performed by the JPEG committee. They are included in the JPEG standard and are reproduced here as Table (1). It is easy to see how the QCs in the table generally grow as we move from the upper left corner to the bottom right corner. This is how JPEG reduces the DCT coefficients with high spatial frequencies.

2. A simple quantization table Q is computed, based on one parameter R specified by the user. A simple expression such as $Qij = 1 + (i + j) \times R$ guarantees that QCs start small at the upper-left corner and get bigger toward the lower-right corner. Table (2) shows an example of such a table with $R = 2$.

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	104	113	92
49	64	78	87	103	121	120	101
72	92	95	98	112	100	103	99
Luminance							

17	18	24	47	99	99	99	99
18	21	26	66	99	99	99	99
24	26	56	99	99	99	99	99
47	66	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
99	99	99	99	99	99	99	99
Chrominance							

Table (1) Recommended Quantization Tables

1	3	5	7	9	11	13	15
3	5	7	9	11	13	15	17
5	7	9	11	13	15	17	19
7	9	11	13	15	17	19	21
9	11	13	15	17	19	21	23
11	13	15	17	19	21	23	25
13	15	17	19	21	23	25	27
15	17	19	21	23	25	27	29

Table 2: The Quantization Table 1 + (i + j) × 2

If the quantization is done correctly, very few nonzero numbers will be left in the DCT coefficients matrix, and they will typically be concentrated in the upper-left region. These numbers are the output of JPEG, but they are further compressed before being written on the output stream. In the JPEG literature this compression is called "entropy coding," and in the next Section shows in detail how it is done. Three techniques are used by entropy coding to compress the 8 × 8 matrix of integers:

1. The 64 numbers are collected by scanning the matrix in zigzags Figure 9. This produces a string of 64 numbers that starts with some nonzeros and typically ends with many consecutive (one after the other) zeros. Only the nonzero numbers are output (after further compressing them) and are followed by a special end-of block (EOB) code. This way there is no need to output the trailing zeros (we can say that the EOB is the run length encoding of all the trailing zeros).
2. The nonzero numbers are compressed using Huffman coding.
3. The first of those numbers (the DC coefficient) is treated differently from the others (the AC coefficients).

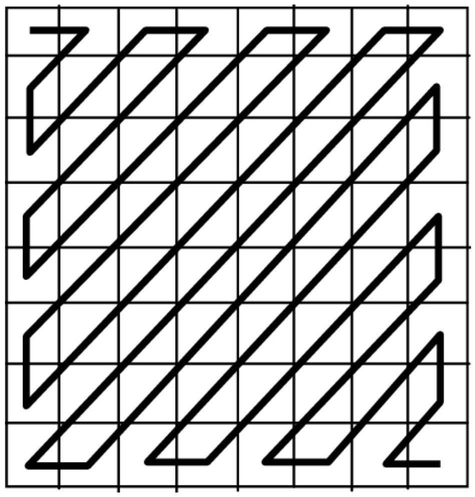
Fig. (9) zigzags 8*8

1.1.4 Coding

Each 8×8 matrix of quantized DCT coefficients contains one DC coefficient [at position (0,0), the top left corner] and 63 AC coefficients. The DC coefficient is a measure of the average value of the 64 original pixels, constituting the data unit. Experience shows that in a continuous-tone image, adjacent data units of pixels are normally correlated in the sense that the average values of the pixels in adjacent data units are close. We already know that the DC coefficient of a data unit is a multiple of the average of the 64 pixels constituting the unit.

This implies that the DC coefficients of adjacent data units don' t differ much. JPEG outputs the first one (encoded), followed by differences (also encoded) of the DC coefficients of consecutive data units.

Example: If the first three 8×8 data units of an image have quantized DC coefficients of 1118, 1114, and 1119, then the JPEG output for the first data unit is 1118 (Huffman encoded, see below) followed by the 63 (encoded) AC coefficients of that data unit. The output for the second data unit will be 1114 − 1118 = −4 (also Huffman encoded), followed by the 63 (encoded) AC coefficients of that data unit, and the output for the third data unit will be. 1119 − 1114 = 5 (also Huffman encoded), again followed by the 63 (encoded) AC coefficients of that data unit. This way of handling the DC coefficients is worth the extra trouble, because the differences are small.

Coding the DC differences is done with Table 3. Each row has a row number (on the left), the unary code for the row (on the right), and several columns in between. Each row contains greater numbers (and also more numbers) than its previous but not the numbers contained in previous rows.

Row i contains the range of integers $[-(2^{i} - 1), +(2^{i} - 1)]$ but is missing the middle range $[-(2^{i-1} - 1), +(2^{i-1} - 1)]$. Thus, the rows get very long, which means that a simple two dimensional array is not a good data structure for this table. In fact, there is no need to store these integers in a data structure, since the program can figure out where in the table any given integer x is supposed to reside by analyzing the bits of x.

The first DC coefficient to be encoded in our example is 1118. It resides in row 11 column 930 of the table (column numbering starts at zero), so it is encoded as 111111111110|01110100010 (the unary code for row 11, followed by the 11-bit binary value of 930). The second DC difference is −4. It resides in row 3 column 3 of Table 3 , so it is encoded as 1110|011 (the unary code for row 3, followed by the 3-bit binary value of 3).

Point 2 above has to do with the specific way the 63 AC coefficients of a data unit are compressed. It uses a combination of RLE and either Huffman or arithmetic coding. The idea is that the sequence of AC coefficients normally contains just a few nonzero numbers, with runs of zeros between them, and with a long run of trailing zeros. For each nonzero number x, the encoder:

(1) finds the number Z of consecutive zeros preceding x;

(2) finds x in Table 3 and prepares its numbers (R and C);

(3) the pair (R, Z) [that' s (R, Z), not (R, C)] is used as row and column numbers for Table 4; and

(4) the Huffman code found in that position in the table is concatenated to C (where C is written as an R-bit number) and the result is (finally) the code emitted by the JPEG encoder for the AC coefficient x and all the consecutive zeros preceding it.

0:	0										0
1:	-1	1									10
2:	-3	-2	2	3							110
3:	-7	-6	-5	-4	4	5	6	7	...		1110
4:	-15	-14	…	-9	-8	8	9	10	…	15	11110
5:	-31	-30	-29	…	-17	-16	16	17		31	111110
6:	-63	-62	-61	…	-33	-32	32	33		63	1111110
7:	-127	-126	-125	…	-65	-64	64	65		127	11111110
14:	-16383	– 16382	-16381	…	-8193	-8192	8192	8193		16383	111111111111110
15:	-32767	– 32766	-32765	…	-16385	-16384	16384	16385		32767	1111111111111110
16:	32768			…							1111111111111111

Table 3: Coding the Differences of DC Coefficients

The reader should notice the EOB code at position (0, 0) and the ZRL code at position (0,15). The former indicates end-of-block, and the latter is the code emitted for 15 consecutive zeros when the number of consecutive zeros exceeds 15. These codes are the ones recommended for the luminance AC coefficients of in specific Table. The EOB and ZRL codes recommended for the chrominance AC coefficients of in specific Table are 00 and 1111111010, respectively.

As an example consider the sequence

$$1118, 2, 0, -2, \underbrace{0, \dots, 0}_{13}, -1, 0, \dots.$$

The first AC coefficient 2 has no zeros preceding it, so Z = 0. It is found in Table 3 in row 2, column 2, so R=2 and C=2. The Huffman code in position(R,Z)=(2,0) of Table 4 is 01, so the final code emitted for 2 is 01|10. The next nonzero coefficient, −2, has one zero preceding it, so Z = 1. It is found in Table 3 in row 2, column 1, so R = 2 and C = 1. The Huffman code in position (R,Z) = (2,1) of Table 4 is 11011, so the final code emitted for 2 is 11011|01. The code is emitted for the last nonzero AC coefficient, −1 is R=1,C=0,Z=13 so (R,Z)=(1,13)=1110101 , so the final code emitted for −1 is 1110101|0.

Finally, the sequence of trailing zeros is encoded as 1010 (EOB), so the output for the above sequence of AC coefficients is 01101101110111010101010. We saw earlier that the DC coefficient is encoded as 111111111110|1110100010, so the final output for the entire 64-pixel data unit is the 46-bit number

$$\underbrace{1111111111100111010001}_{DC}\underbrace{0011}_{2}\underbrace{0110110}_{-2}\underbrace{111101010}_{-1}\underbrace{1010}_{EOB}.$$

These 46 bits encode one color component of the 64 pixels of a data unit. Let’s assume that the other two color components are also encoded into 46-bit numbers. If each pixel originally consists of 24 bits, then this corresponds to a compression factor of $64 \times 24/(46 \times 3) \approx 11.13$.

R \ Z	0	1	·········	15
0:	1010		······	11111111001(ZRL)
1:	00	1100	······	1111111111110101
2:	01	11011	······	1111111111110110
3:	100	1111001	······	1111111111110111
4:	1011	111110110	······	1111111111111000
5:	11010	11111110110		1111111111111001
:	:	:	:	:

Table 4: Coding AC Coefficients.

1.2 Progressive Image Compression

Progressive compression is an attractive choice when compressed images are transmitted over a communications line and are decompressed and viewed in real time. When such an image is received and is decompressed, the decoder can very quickly display the entire image in a low-quality format, and improve the display quality as more and more of the image is being received and decompressed. A user watching the image developing on the screen can normally recognize most of the image features after only 5-10% of it has been decompressed.

This should be compared to raster-scan image compression. When an image is raster scanned and compressed, a user normally cannot tell much about the image when only 5-10% of it has been decompressed and displayed. Images are supposed to be viewed by humans, which is why progressive compression makes sense even in cases where it is slower or less efficient than nonprogressive.

Perhaps a good way to think of progressive image compression is to imagine that the encoder compresses the most important image information first, then compresses less important information and appends it to the compressed stream, and so on. This explains why all progressive image compression methods have a natural lossy option; simply stop compressing at a certain point. The user can control the amount of loss by means of a parameter that tells the encoder how soon to stop the progressive encoding process. The sooner encoding is stopped, the better the compression ratio and the higher the data loss.

Another advantage of progressive compression becomes apparent when the compressed file has to be decompressed several times and displayed with different resolutions. The decoder can, in each case, stop the decompression when the image has reached the resolution of the particular output device used.

Progressive image compression has a connection with JPEG. JPEG uses the DCT to break the image up into its spatial frequency components, and it compresses the low-frequency components first. The decoder can therefore display these parts quickly, and it is these low-frequency parts that contain the principal image information. The high-frequency parts contain image details. Thus, JPEG encodes spatial frequency data progressively.

It is useful to think of progressive decoding as the process of improving image features over time, and this can be achieved in three ways:

1. Encode spatial frequency data progressively. An observer watching such an image being decoded sees the image changing from blurred to sharp. Methods that work this way typically feature medium speed encoding and slow decoding. This type of progressive compression is sometimes called SNR progressive or quality progressive.

2. Start with a gray image and add colors or shades of gray to it. An observer watching such an image being decoded will see all the image details from the start, and will see them improve as more color is continuously added to them. method that works this way normally features slow encoding and fast decoding.

3. Encode the image in layers, where early layers consist of a few large low-resolution pixels, followed by later layers with smaller higher-resolution pixels. A person watching such an image being decoded will see more detail added to the image over time. Such a method thus adds detail (or resolution) to the image as it is being decompressed. This way of progressively encoding an image is called pyramid coding or hierarchical coding. Most progressive methods use this principle. Figure 11 illustrates the three progressive methods mentioned here. It should be contrasted with Figure 10, which illustrates sequential decoding.

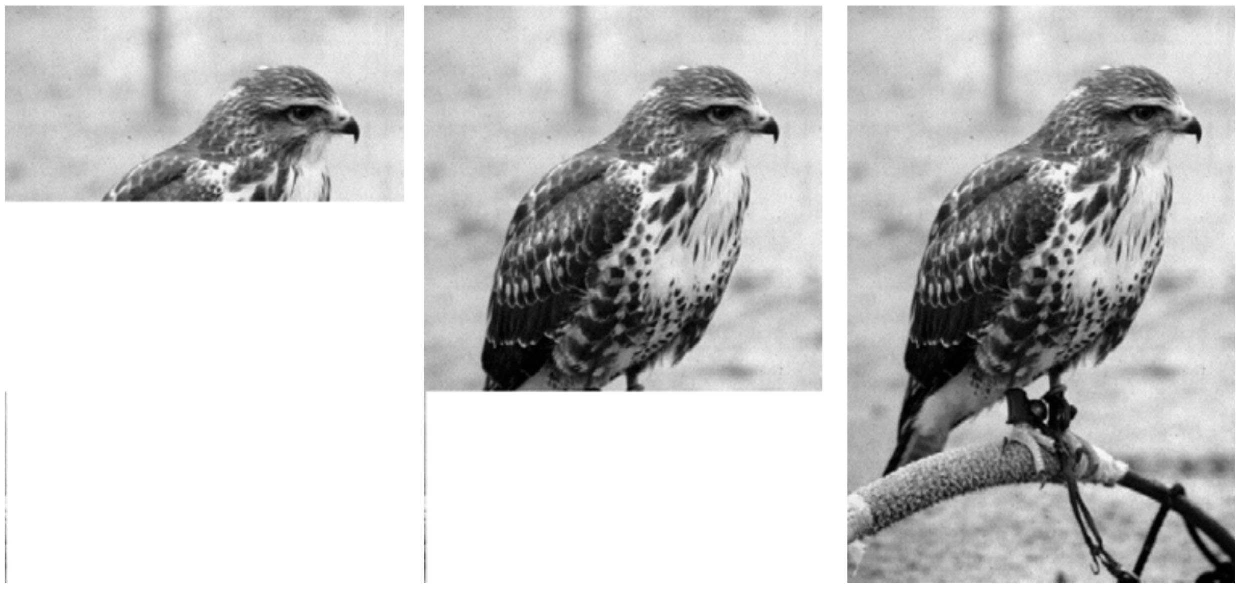
Fig. (10) Sequential Decoding.

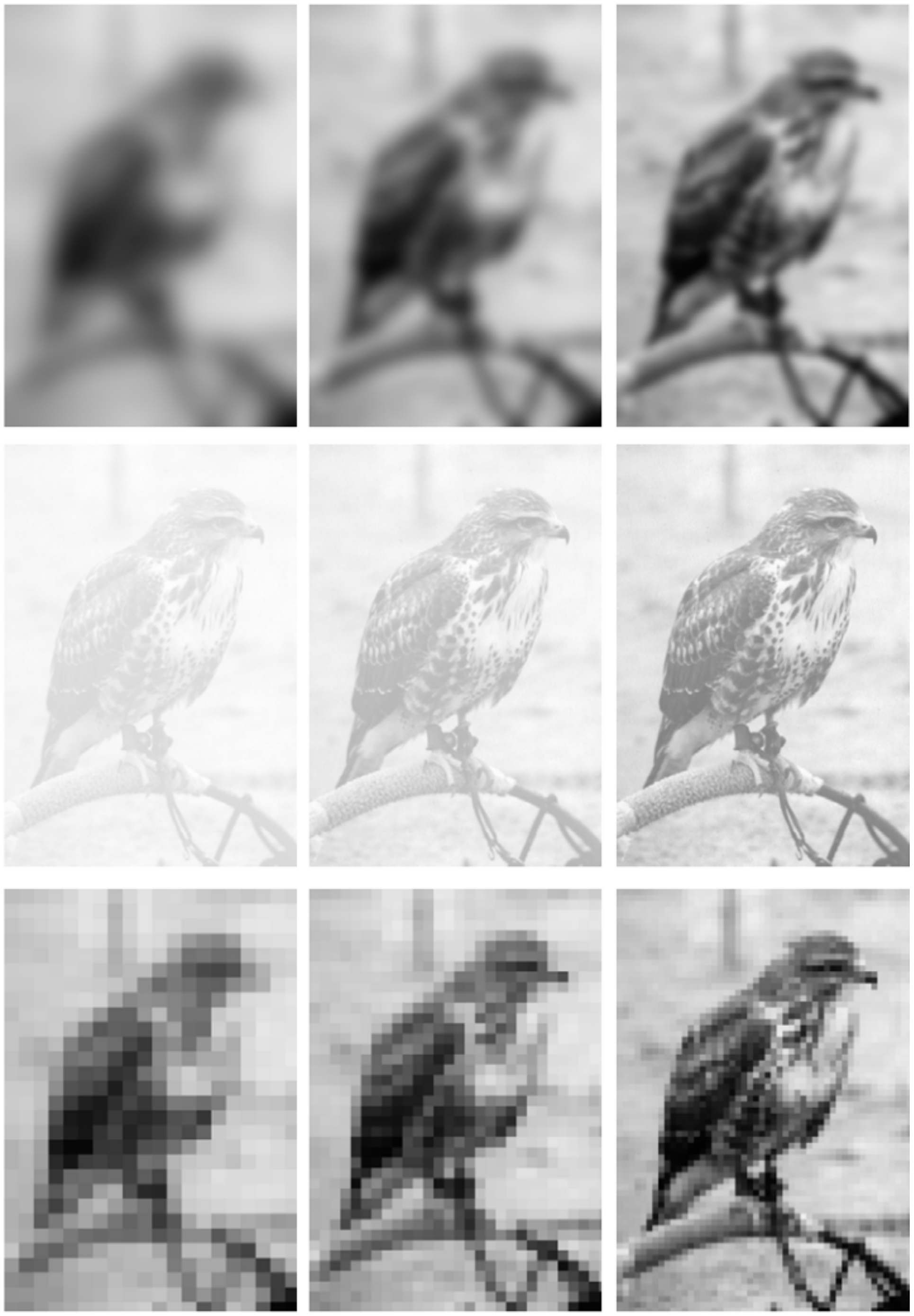

Fig. (11): Progressive Decoding.

Assuming that the image size is $2^n \times 2^n = 4^n$ pixels, the simplest method that comes to mind, when trying to implement progressive compression, is to calculate each pixel of layer $i - 1$ as the average of a group of 2×2 pixels of layer i. Thus, layer n is the entire image, layer $n - 1$ contains $2^{n-1} \times 2^{n-1} = 4^{n-1}$ large pixels of size 2×2, and so on, down to layer 1, with $4^{n-n} = 1$ large pixel, representing the entire image. If the image isn' t too large, all the layers can be saved in memory. The pixels are then written on the compressed stream in reverse order, starting with layer 1. The single pixel of layer 1 is the "parent" of the four pixels of layer 2, each of which is the parent of four pixels in layer 3, and so on. The total number of pixels in the pyramid is 33% more than the original number!

A simple way to bring the total number of pixels in the pyramid down to 4^n is to include only three of the four pixels of a group in layer i, and to compute the value of the 4th pixel using the parent of the group (from the preceding layer, $i - 1$) and its three siblings.

Example: Figure 12c shows a 4×4 image that becomes the third layer in its progressive compression. Layer two is shown in Figure 12b, where, for example, pixel 81.25 is the average of the four pixels 90, 72, 140, and 23 of layer three. The single pixel of layer one is shown in Figure 12a.

54.125

(a)

81.25	32.5
61.25	41.5

(b)

90	72	58	33
140	23	21	18
100	70	72	18
16	59	44	32

(c)

54

(d)

81	33
61	42

(e)

90	72	58	33
140	23	21	18
100	70	72	18
16	59	44	32

(f)

max 140

(g)

max 140	min 18
min 16	max 72

(h)

90	72	58	33
140	23	21	18
100	70	72	18
16	59	44	32

(i)

Fig. (12): Progressive Image Compression

The compressed file should contain just the numbers 54.125, 32.5,41.5,61.25, 72,23,140, 33,18,21, 18,32,44, 70,59,16

(properly encoded, of course), from which all the missing pixel values can easily be determined. The missing pixel 81.25, e.g., can be calculated from (x + 32.5 + 41.5 + 61.25)/4 = 54.125.

A small complication with this method is that averages of integers may be non-integers. If we want our pixel values to remain integers, we either have to lose precision or to keep using longer and longer integers.

Figure 12 d, e,f shows the results of rounding off our pixel values and thus losing some image information. The content of the compressed file in this case should be

54, 33, 42, 61, 72, 23, 140, 33, 18, 21, 18, 32, 44, 70, 59, 16. The first missing pixel, 81, of layer three can be determined from the equation (x + 33 + 42 + 61)/4 = 54, which yields the (slightly wrong) value 80.

One alternative approach is to select the maximum (or the minimum) pixel of a group as the parent. This has the advantage that the parent is identical to one of the pixels in the group. The encoder has to encode just three pixels in each group, and the decoder decodes three pixels and uses the parent as the fourth pixel, to complete the group. When encoding consecutive groups in a layer, the encoder should alternate between selecting the maximum and the minimum as parents, since always selecting the same creates progressive layers that are either too dark or too bright. Figure 12 g, h, i shows the three layers in this case.

The compressed file should contain the numbers 140, (0), 18, 72, 16, (3), 90, 72, 23, (2), 58, 33, 21, (0), 18, 32, 44, (3), 100, 70, 59,

where the numbers in parentheses are two bits each. They tell where (in what quadrant) the parent from the previous layer should go. Notice that quadrant numbering is $\begin{pmatrix} 0 & 1 \\ 3 & 2 \end{pmatrix}$.

Selecting the median of a group is a little slower than selecting the maximum or the minimum, but it improves the appearance of the layers during progressive decompression.

2. Video Compression

With the rapid advances in computers in the 1980s, 1990s and 2000s and so on came multimedia applications, where pictures and sound are combined in the same file. Such files tend to be large, which is why compressing them became a natural application.

Video compression is the process of reducing the size of a video file while maintaining its visual quality. This is typically done by removing redundant or unnecessary information from the video, such as repeating patterns or colors, and then encoding the remaining data in a more efficient way.

2.1 Digital Video

Digital video is, in principle, a sequence of images, called frames, displayed at a certain frame rate (so many frames per second, or fps) to create the illusion of animation. This rate, as well as the image size and pixel depth, depend heavily on the application. Surveillance cameras, for example, use the very low frame rate of five fps, while HDTV displays 25 fps.

Digital video has the following important advantages:

1. It can be easily edited. This makes it possible to produce special effects. The images of an actor in a movie can be edited to make him look young at the beginning and old later. Software for editing digital video is available for most computer platforms. Users can edit a video file and attach it to an email message, thereby creating email. Multimedia applications, where text, sound, still images, and video are integrated, are common today and involve the editing of video.

2. It can be stored on any digital medium. such as hard disks, removable cartridges, CD-ROMs, or DVDs. An error-correcting code can be added, if needed, for increased reliability. This makes it possible to duplicate a long movie or transmit it between computers without loss of quality (in fact, without a single bit getting corrupted).

3. It can be compressed. This allows for more storage (when video is stored on a digital medium) and also for fast transmission. Sending compressed video between computers makes video telephony possible, which, in turn, makes video conferencing possible. Transmitting compressed video also makes it possible to increase the capacity of television cables and thus add channels.

2.2 Video Compression

Video compression is based on two principles.

1- The spatial redundancy that exists in each frame.
2- The fact that most of the time, a video frame is very similar to its immediate neighbors. This is called temporal redundancy.

A typical technique for video compression should therefore start by encoding the first frame using a still image compression method. It should then encode each successive frame by identifying the differences between the frame and its predecessor, and encoding these differences. If a frame is very different from its predecessor (as happens with the first frame of a shot), it should be coded independently of any other frame.

A frame that is coded using its predecessor is called inter frame (or just inter), while a frame that is coded independently is called intra frame (or just intra).

Video compression is normally lossy. Encoding a frame F_i in terms of its predecessor F_{i-1} introduces some distortions. As a result, encoding the next frame F_{i+1} in terms of (the already distorted) F_i increases the distortion. Even in lossless video compression, a frame may lose some bits. This may happen during transmission or after a long shelf stay. If a frame F_i has lost some bits, then all the frames following it, up to the next intra frame, are decoded improperly, perhaps even leading to accumulated errors. This is why intra frames should be used from time to time inside a sequence, not just at its beginning. An intra frame is labeled I, and an inter frame is labeled P (for predictive).

We know that an encoder should not use any information that is not available to the decoder, but video compression is special because of the large quantities of data involved. We usually don' t mind if the encoder is slow, but the decoder has to be fast. A typical case is video recorded on a hard disk or on a DVD, to be played back. The encoder can take minutes or hours to encode the data. The decoder, however, has to play it back at the correct frame rate (so many frames per second), so it has to be fast. This is why a typical video decoder works in parallel. It has several decoding circuits working simultaneously on several frames.

A frame that is encoded based on both past and future frames is labeled B (for bidirectional).

The idea of a B frame is so useful that most frames in a compressed video presentation may be of this type. We therefore end up with a sequence of compressed frames of the three types I, P, and B. An I frame is decoded independently of any other frame. A P frame is decoded using the preceding I or P frame. A B frame is decoded using the preceding and following I or P frames. Figure 13a shows a sequence of such frames in the order in which they are generated by the encoder (and input by the decoder).

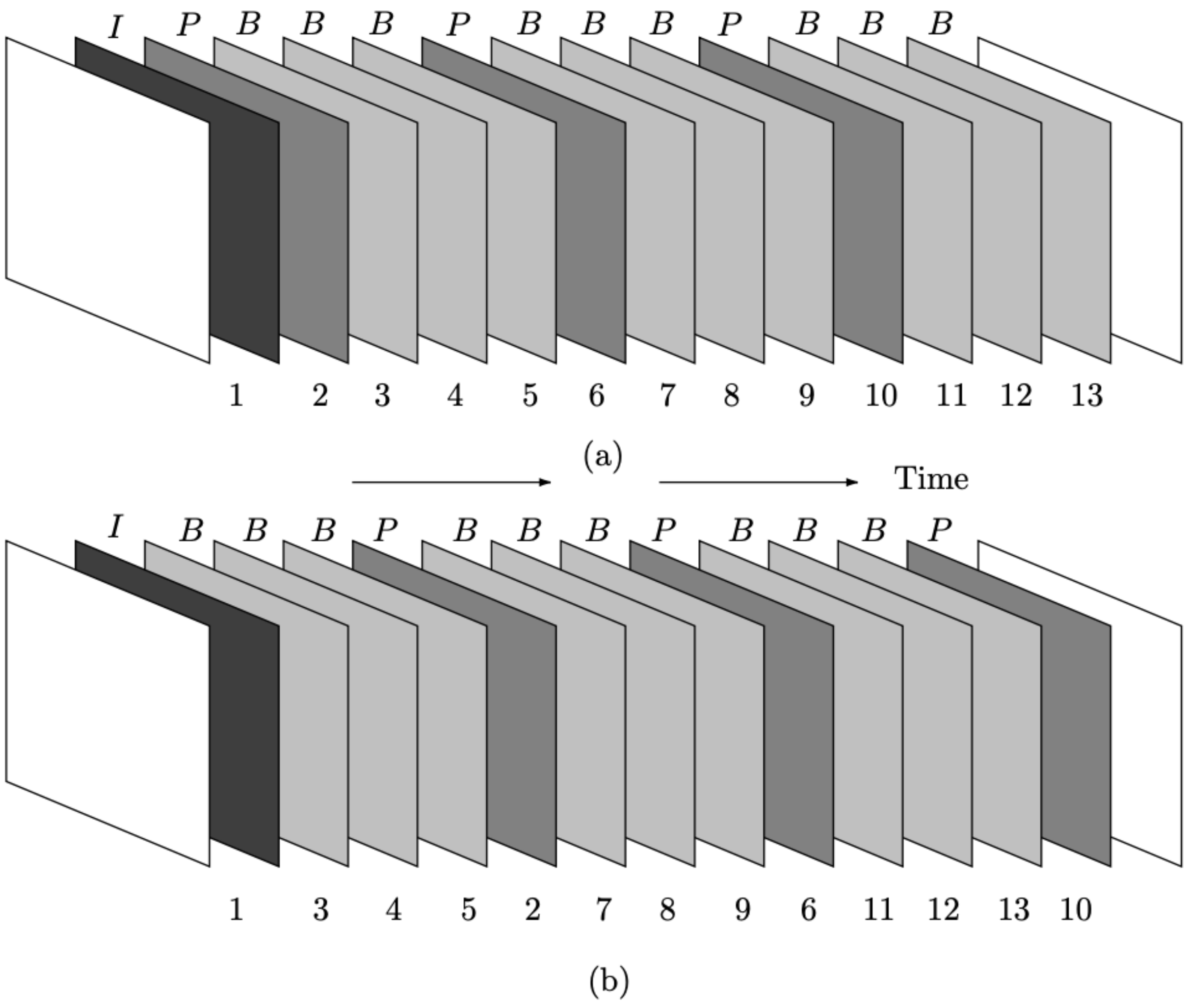

Fig. (13): (a) Coding Order. (b) Display Order

Figure 13b shows the same sequence in the order in which the frames are output by the decoder and displayed. The frame labeled 2 should be displayed after frame 5, so each frame should have two time stamps, its coding time and its display time.

We start with a few intuitive video compression methods.

1- Subsampling: The encoder selects every other frame and writes it on the compressed stream. This yields a compression factor of 2. The decoder inputs a frame and duplicates it to create two frames.

2- Differencing: A frame is compared to its predecessor. If the difference between them is small (just a few pixels), the encoder encodes the pixels that are different by writing three numbers on the compressed stream for each pixel: its image coordinates(r,c), and the difference between the values of the pixel in the two frames. If the difference between the frames is large, the current frame is written on the output in raw format.

A lossy version of differencing looks at the amount of change in a pixel. If the difference between the intensities of a pixel in the preceding frame and in the current frame is smaller than a certain (user controlled) threshold, the pixel is not considered different.

3-Block Differencing: This is a further improvement of differencing. The image is divided into blocks of pixels, and each block B in the current frame is compared with the corresponding block P in the preceding frame. If the blocks differ by more than a certain amount, then B is compressed by writing its image coordinates, followed by the values of all its pixels (expressed as differences) on the compressed stream.

4- Motion Compensation: Anyone who has watched movies knows that the difference between consecutive frames is small because it is the result of moving the scene, the camera, or both between frames. This feature can therefore be exploited to achieve better compression. If the encoder discovers that a part P of the preceding frame has been rigidly moved to a different location in the current frame, then P can be compressed by writing the following three items on the compressed stream: its previous location, its current location, and information identifying the boundaries of P.

In principle, such a part can have any shape. In practice, we are limited to equal- size blocks (normally square but can also be rectangular). The encoder scans the current frame block by block. For each block B it searches the preceding frame for an identical block C (if compression is to be lossless) or for a similar one (if it can be lossy). Finding such a block, the encoder writes the difference between its past and present locations on the output. This difference is of the form

$$(C_x - B_x, C_y - B_y) = (\Delta_x, \Delta_y)$$

so it is called a motion vector. Figure 13a,b shows a simple example where the sun and trees are moved rigidly to the right (because of camera movement) while the child moves a different distance to the left (this is scene movement).

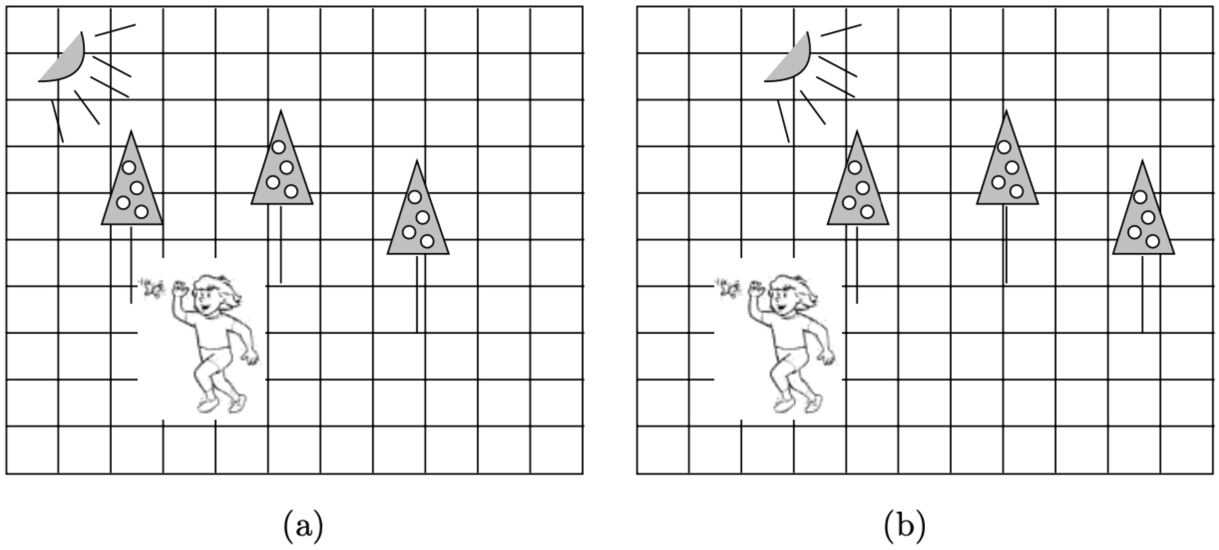

Fig. (13): Motion Compensation.

Motion compensation is effective if objects are just translated, not scaled or rotated. Serious changes in illumination from frame to frame also reduce the effectiveness of this method. In general, motion compensation is lossy. The following are the main aspects of motion compensation in detail:

a) Frame Segmentation: The current frame is divided into equal-size nonoverlapping blocks. The blocks may be squares or rectangles. The latter choice assumes that motion in video is mostly horizontal, so horizontal blocks reduce the number of motion vectors without degrading the compression ratio. The block size is important, because large blocks reduce the chance of finding a match, and small blocks result in many motion vectors. In practice, block sizes that are integer powers of 2, such as 8 or 16, are used, since this simplifies the software.

Search Threshold: Each block B in the current frame is first compared to its counterpart C in the preceding frame. If they are identical, or if the difference between them is less than a preset threshold, the encoder assumes that the block hasn’t been moved.

Block Search: This is a time-consuming process, and so has to be carefully designed. If B is the current block in the current frame, then the previous frame has to be searched for a block identical to or very close to B. The search is normally restricted to a small area (called the search area) around B, defined by the maximum displacement parameters dx and dy. These parameters specify the maximum horizontal and vertical distances, in pixels, between B and any matching block in the previous frame. If B is a square with side b, the search area will contain (b + 2dx)(b + 2dy) pixels (Figure 14), overlapping b×b squares.

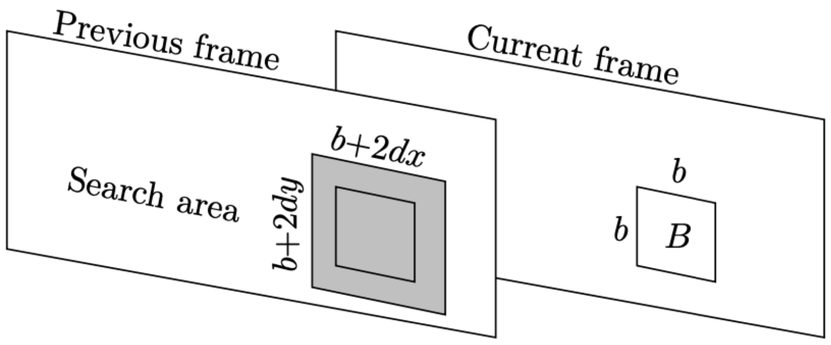

Fig. (14): Search Area

Distortion Measure: This is the most sensitive part of the encoder. The distortion measure selects the best match for block B. It has to be simple and fast, but also reliable.

The mean absolute difference (or mean absolute error) calculates the average of the absolute differences between a pixel B_{ij} in B and its counterpart C_{ij} in a candidate block C:

$$\frac{1}{b^2}\sum_{i=1}^{b}\sum_{j=1}^{b}|B_{ij} - C_{ij}|$$

The smallest distortion (say, for block C_k) is examined. If it is smaller than the search threshold, then Ck is selected as the match for B. Otherwise, there is no match for B, and B has to be encoded without motion compensation. A block in the current frame match nothing in the preceding frame if imagine a camera panning from left to right. New objects will enter the field of view from the right all the time. A block on the right side of the frame may therefore contain objects that did not exist in the previous frame.

Motion Vector Correction: Once a block C has been selected as the best match for B, a motion vector is computed as the difference between the upper-left corner of C and the upper-left corner of B.

Coding Motion Vectors: A large part of the current frame (perhaps close to half of it) may be converted to motion vectors, which is why the way these vectors are encoded is important; it must also be lossless. Two properties of motion vectors help in encoding them: (1) They are correlated and (2) their distribution is nonuniform.

2.2 MPEG Compression

Moving Pictures Experts Group (MPEG) is a method for video compression, which involves the compression of digital images and sound, as well as synchronization of the two. There currently are several MPEG standards.

MPEG uses its own vocabulary. An entire movie is considered a video sequence. It consists of pictures, each having three components, one luminance (Y) and two chrominance (Cb and Cr).

The luminance component contains the black-and- white picture, and the chrominance components provide the color hue and saturation. Each component is a rectangular array of samples, and each row of the array is called a raster line. A pel is the set of three samples.

The input to an MPEG encoder is called the source data, and the output of an MPEG decoder is the reconstructed data. The MPEG decoder has three main parts, called layers, to decode the audio, the video, and the system data.

2.3.1 MPEG-1 Main Component

MPEG uses I, P, and B pictures, as discussed before. They are arranged in groups, where a group can be open or closed. The pictures are arranged in a certain order, called the coding order, but (after being decoded) they are output and displayed in a different order, called the display order. In a closed group, P and B pictures are decoded only from other pictures in the group. In an open group, they can be decoded from pictures outside the group. Different regions of a B picture may use different pictures for their decoding.

MPEG-1 was originally developed as a compression standard for interactive video on CDs and for digital audio broadcasting. It turned out to be a technological succeeded but a visionary failure. On the one hand, not a single design mistake was found during the implementation of this complex algorithm and it worked as expected. On the other hand, interactive CDs and digital audio broadcasting have had little commercial success, so MPEG-1 is used today for general video compression. One aspect of MPEG-1 that was supposed to be minor, namely MP3, has grown out of proportion and is commonly used today for audio .MPEG-2, on the other hand, was specifically designed for digital television and this standard has had enormous commercial success.

The basic building block of an MPEG picture is the macroblock (Figure 15). It consists of a 16×16 block of luminance (grayscale) samples (divided into four 8×8 blocks) and two 8 × 8 blocks of the matching chrominance samples. The MPEG compression of a macroblock consists mainly in passing each of the six blocks through a discrete cosine transform, which creates decorrelated values, then quantizing and encoding the results. It is very similar to JPEG compression, the main differences being that different quantization tables and different code tables are used in MPEG for intra and nonintra, and the rounding is done differently.

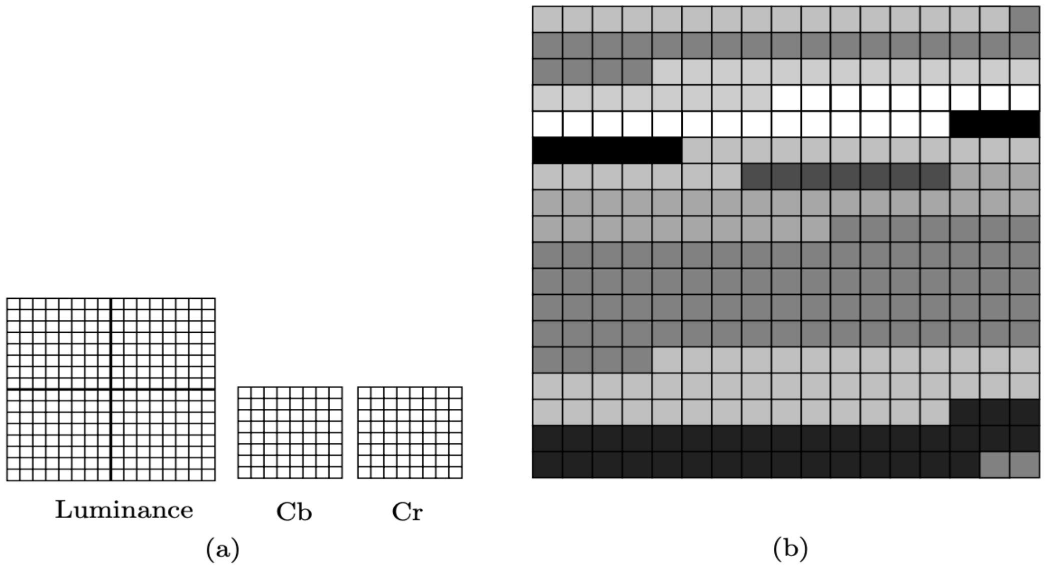

Fig. (15): (a) A Macroblock. (b) A Possible Slice Structure.

A picture in MPEG is organized in slices, where each slice is a adjacent set of macroblocks (in raster order) that have the same grayscale (i.e., luminance component). The concept of slices makes sense because a picture may often contain large uniform areas, causing many adjacent macroblocks to have the same grayscale. (Figure 15b) shows a hypothetical MPEG picture and how it is divided into slices. Each square in the picture is a macroblock. Notice that a slice can continue from scan line to scan line.

In Figure 15b consists of 18×18 macroblocks, and each macroblock constitutes six 8×8 blocks of samples. The total number of samples is therefore 18×18×6×8×8 = 124416.

When a picture is encoded in nonintra mode (i.e., it is encoded by means of another picture, normally its predecessor), the MPEG encoder generates the differences between the pictures, then applies the DCT to the differences. In such a case, the DCT does not contribute much to the compression, because the differences are already decorrelated. Nevertheless, the DCT is useful even in this case, since it is followed by quantization, and the quantization in nonintra coding can be quite deep.

The precision of the numbers processed by the DCT in MPEG also depends on whether intra or nonintra coding is used. MPEG samples in intra coding are 8-bit unsigned integers, whereas in nonintra they are 9-bit signed integers. This is because a sample in nonintra is the difference of two unsigned integers, and may therefore be negative. In intra coding, rounding is done in the normal way, to the nearest integer, whereas in nonintra, rounding is done by truncating a noninteger to the nearest smaller integer. (Figure 16) shows the results graphically. Notice the wide interval around zero in nonintra coding. This is the so-called dead zone.

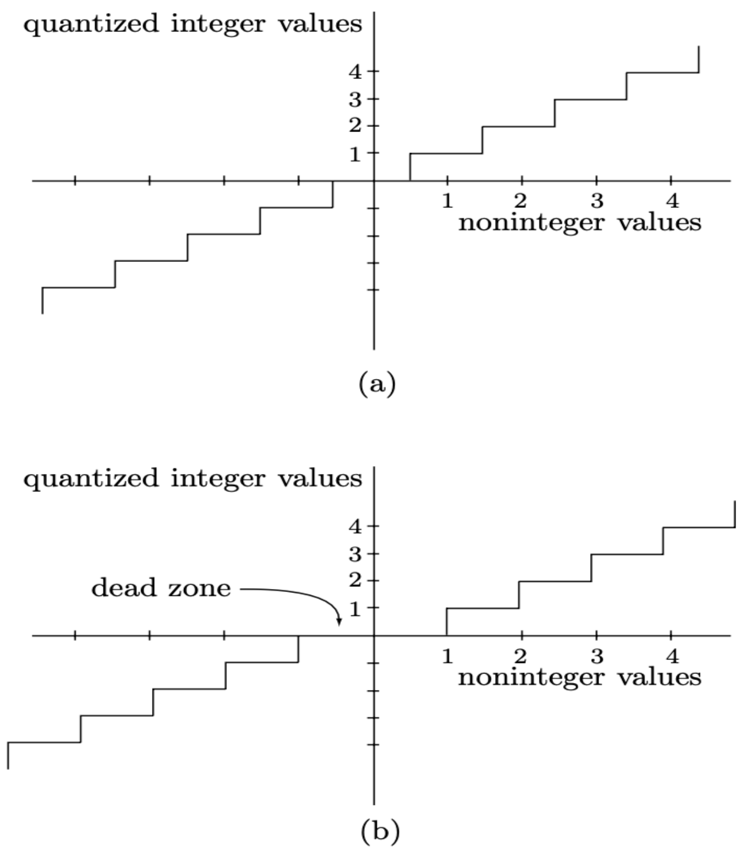

Fig. (16):Rounding of Quantized DCT Coefficients. (a)For Intra Coding. (b) For Nonintra Coding.

2.3.2 MPEG-4

MPEG-4 is a new standard for audiovisual data. Although video and audio compression is still a central feature of MPEG-4, this standard includes much more than just compression of the data. As a result, MPEG-4 is huge and this section can only describe its main features.

The MPEG-4 project started in May 1991 and initially aimed at finding ways to compress multimedia data to very low bitrates with minimal distortions. Many proposals were accepted for the many facets of MPEG-4, and the first version of MPEG-4 was accepted and approved in late 1998. The formal description was published in 1999 with many amendments that keep coming out.

Traditionally, methods for compressing video have been based on pixels. Each video frame is a rectangular set of pixels, and the algorithm looks for correlations between pixels in a frame and between frames. The compression paradigm adopted for MPEG-4, on the other hand, is based on objects. (The name of the MPEG-4 project was also changed at this point to "coding of audiovisual objects.") In addition to producing a movie in the traditional way with a camera or with the help of computer animation, an individual generating a piece of audiovisual data may start by defining objects, such as a flower, a face, or a vehicle, and then describing how each object should be moved and manipulated in successive frames. A flower may open slowly, a face may turn, smile, and fade, a vehicle may move toward the viewer and appear bigger. MPEG-4 includes an object description language that provides for a compact description of both objects and their movements and interactions.

An important feature of MPEG-4 is interoperability. This term refers to the ability to exchange any type of data, be it text, graphics, video, or audio. Obviously, interoperability is possible only in the presence of standards. All devices that produce data, deliver it, and consume (play, display, or print) it must obey the same rules and read and write the same file structures.

Because of the wide goals and rich variety of tools available as part of MPEG-4, this standard is expected to have many applications. The ones listed here are just a few important examples.

1. Streaming multimedia data over the Internet or over local-area networks. This is important for entertainment and education.

2. Communications, both visual and audio, between vehicles and/or individuals. This has military and law enforcement applications.

3. Broadcasting digital multimedia. This, again, has many entertainment and educational applications.

4. Context-based storage and retrieval. Audiovisual data can be stored in compressed form and retrieved for delivery or consumption.

5. Studio and television postproduction. A movie originally produced in English may be translated to another language by dubbing or subtitling.

6. Surveillance. Low-quality video and audio data can be compressed and transmitted from a surveillance camera to a central monitoring location over an inexpensive, slow communications channel. Control signals may be sent back to the camera through the same channel to rotate or zoom it in order to follow the movements of a suspect.

7. Virtual conferencing. This time-saving application is the favorite of busy executives.

2.3.3 H.261

In late 1984, the CCITT (currently the ITU-T) organized an expert group to develop a standard for visual telephony for ISDN services. The idea was to send images and sound between special terminals, so that users could talk and see each other. This type of application requires sending large amounts of data, so compression became an important consideration. The group eventually came up with a number of standards, known as the H series (for video) and the G series (for audio) recommendations, all operating at speeds of $p\times64$ Kbit/sec for $1 \le p \le 30$. These standards are summarized in Table 5.

Standard	Purpose
H.261	Video
H.221	Communications
H.230	Initial handshake
H.320	Terminal systems
H.242	Control protocol
G.711	Companded audio (64 Kbits/s)
G.722	High quality audio (64 Kbits/s)
G.728	Speech (LD-CELP @16kbits/s)

Table 5: The p×64 Standards.

Members of the p×64 also participated in the development of MPEG, so the two methods have many common elements. There is, however, an important difference between them. In MPEG, the decoder must be fast, since it may have to operate in real time, but the encoder can be slow. This leads to very asymmetric compression, and the encoder can be hundreds of times more complex than the decoder. In H.261, both encoder and decoder operate in real time, so both have to be fast. Still, the H.261 standard defines only the data stream and the decoder. The encoder can use any method as long as it creates a valid compressed stream. The compressed stream is organized in layers, and macroblocks are used as in MPEG. Also, the same 8×8 DCT and the same zigzag order as in MPEG are used. The intra DC coefficient is quantized by always dividing it by 8, and it has no dead zone. The inter DC and all AC coefficients are quantized with a dead zone.

Motion compensation is used when pictures are predicted from other pictures, and motion vectors are coded as differences. Blocks that are completely zero can be skipped within a macroblock, and variable-size codes that are very similar to those of MPEG (such as run-level codes), or are even identical (such as motion vector codes) are used. In all these aspects, H.261 and MPEG are very similar.

There are, however, important differences between them. H.261 uses a single quantization coefficient instead of an 8×8 table of QCs, and this coefficient can be changed only after 11 macroblocks. AC coefficients that are intra coded have a dead zone. The compressed stream has just four layers, instead of MPEG’s six. The motion vectors are always full-pel and are limited to a range of just ±15 pels. There are no B pictures, and only the immediately preceding picture can be used to predict a P picture.

3. Audio Compression

With the advent of powerful, inexpensive personal computers in the 1980s and 1990s came multimedia applications, where text, images, movies, and sound are stored in the computer, and can be uploaded, downloaded, displayed, edited, and played back. The storage requirements of sound are smaller than those of images or movies, but bigger than those of text. This is why audio compression has become important and has been the subject of much research and experimentation throughout the 1990s.

Two important features of audio compression are (1) it can be lossy and (2) it requires fast decoding. Text compression must be lossless, but images and audio can lose much data without a noticeable degradation of quality. Thus, there are both lossless and lossy audio compression algorithms. Often, audio is stored in compressed form and has to be decompressed in real-time when the user wants to listen to it. This is why most audio compression methods are asymmetric. The encoder can be slow, but the decoder has to be fast.

The sound means: An intuitive definition: Sound is the sensation detected by our ears and interpreted by our brain in a certain way.

A scientific definition: Sound is a physical disturbance in a medium. It propagates in the medium as a pressure wave by the movement of atoms or molecules.

Like any other wave, sound has three important attributes, its speed, amplitude, and period. The frequency of a wave is not an independent attribute; it is the number of periods that occur in one time unit (one second). The unit of frequency is the hertz (Hz). The speed of sound depends mostly on the medium it passes through, and on the temperature.

The human ear is sensitive to a wide range of sound frequencies, normally from about 20 Hz to about 22,000 Hz, depending on a person ’ s age and health. This is the range of audible frequencies.

The amplitude of sound is also an important property. We perceive it as loudness. The sensitivity of the human ear to sound level depends on the frequency. Experiments indicate that people are more sensitive to (and therefore more annoyed by) high-frequency sounds (which is why sirens have a high pitch)

3.1 Digital Audio

Digital audio is a technology that is used to record, store, manipulate, generate and reproduce sound using audio signals that have been encoded in digital form.

When sound is played into a microphone, it is converted into a voltage that varies continuously with time. Figure 18 shows a typical example of sound that starts at zero and oscillates several times. Such voltage is the analog representation of the sound. Digitizing sound is done by measuring the voltage at many points in time, translating each measurement into a number, and writing the numbers on a file. This process is called sampling. The sound wave is sampled, and the samples become the digitized sound. The device used for sampling is called an analog-to-digital converter (ADC).

The difference between a sound wave and its samples can be compared to the difference between an analog clock, where the hands seem to move continuously, and a digital clock, where the display changes abruptly every second.

Since the audio samples are numbers, they are easy to edit. However, the main use of an audio file is to play it back. This is done by converting the numeric samples back into voltages that are continuously fed into a speaker. The device that does that is called a digital-to-analog converter (DAC). Intuitively, it is clear that a high sampling rate would result in better sound reproduction, but also in many more samples and therefore bigger files. Thus, the main problem in audio sampling is how often to sample a given sound.

Figure 18a shows what may happen if the sampling rate is too low. The sound wave in the figure is sampled four times, and all four samples happen to be identical. When these samples are used to play back the sound, the result is silence. Figure 18b shows seven samples, and they seem to "follow" the original wave fairly closely. Unfortunately, when they are used to reproduce the sound, they produce the curve shown in dashed. There simply are not enough samples to reconstruct the original sound wave.

The solution to the sampling problem is to sample sound at a little over the Nyquist frequency (he minimum rate at which a signal can be sampled without introducing errors, which is twice the highest frequency present in the signal). The range of human hearing is typically from 16-20 Hz to 20,000-22,000 Hz. When sound is digitized at high fidelity, it should therefore be sampled at a little over the Nyquist rate of $2 \times 22000 = 44000$ Hz. This is why high-quality digital sound is based on a 44,100-Hz sampling rate. Anything lower than this rate results in distortions, while higher sampling rates do not produce any improvement in the reconstruction of the sound. We can consider the sampling rate of 44,100 Hz a lowpass filter, since it effectively removes all the frequencies above 22,000 Hz. Such a sampling rate guarantees true reproduction of the sound. This is illustrated in Figure 18c, which shows 10 equally-spaced samples taken over four periods. Notice that the samples can come from any point.

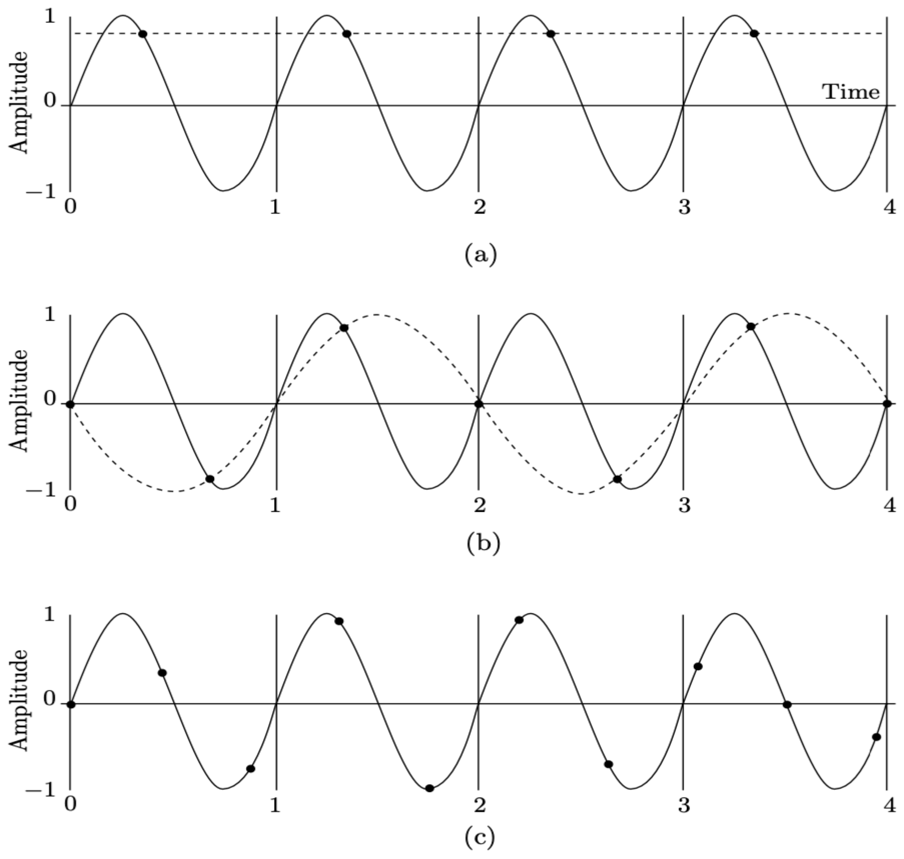

Fig (18): Sampling a Sound Wave

The second problem in sound sampling is the sample size. Each sample becomes a number, but how large should this number be? In practice, samples are normally either 8 or 16 bits, although some high-quality sound cards that are available for many computer platforms may optionally use 32-bit samples. Assuming that the highest voltage in a sound wave is 1 volt, an 8-bit sample can distinguish voltages as low as 1/256 ≈ 0.004 volt, or 4 millivolts (mv). A quiet sound, generating a wave lower than 4 mv, would be sampled as zero and played back as silence.

In contrast, with a 16-bit sample it is possible to distinguish sounds as low as 1/65536 ≈ 15 microvolt. We can think of the sample size as a quantization of the original audio data. Eight-bit samples are more coarsely quantized than 16-bit samples. As a result, they produce better compression but poorer audio reconstruction. Suppose that the sample size is one bit. Each sample has a value of either 0 or 1. We hear when these samples are played back each 0 would result in silence and each sample of 1, in the same tone. The result would be a nonuniform buzz. Such sounds were common on early personal computers.

3.2 Conventional Audio Compression Methods

conventional compression methods, such as RLE, statistical, and dictionary-based, can be used to losslessly compress sound files, but the results depend heavily on the specific sound. Some sounds may compress well under RLE but not under a statistical method. Other sounds may lend themselves to statistical compression but may expand when processed by a dictionary method. Here is how sounds respond to each of the three classes of compression methods.

RLE may work well when the sound contains long runs of identical samples. With 8-bit samples this may be common. Recall that the difference between the two 8-bit samples n and n + 1 is about 4 mv. A few seconds of uniform music, where the wave does not oscillate more than 4 mv, may produce a run of thousands of identical samples. With 16-bit samples, long runs may be rare and RLE, consequently, ineffective.

Statistical methods assign variable-size codes to the samples according to their frequency of occurrence. With 8-bit samples, there are only 256 different samples, so in a large audio file, the samples may sometimes have a flat distribution. Such a file will therefore not respond well to Huffman coding. With 16-bit samples there are more than 65,000 possible samples, so they may sometimes feature skewed probabilities (i.e., some samples may occur very often, while others may be rare). Such a file may therefore compress better with arithmetic coding.

Dictionary-based methods expect to find the same phrases again and again in the data. This happens with text, where certain strings may repeat often. Sound, however, is an analog signal and the particular samples generated depend on the precise way the ADC works. Signals may become samples of different sizes. This is why parts of speech that sound the same to us, and should therefore have become identical phrases, end up being digitized slightly differently, and go into the dictionary as different phrases, thereby reducing compression. Dictionary-based methods are not well suited for sound compression.

3.3 Lossy Sound Compression

It is possible to get better sound compression by developing lossy methods that take advantage of our perception of sound, and discard data to which the human ear is not sensitive. This is similar to lossy image compression, where data to which the human eye is not sensitive is discarded. In both cases we use the fact that the original information (image or sound) is analog and has already lost some quality when digitized. Losing some more data, if done carefully, may not significantly affect the played-back sound, and may therefore be indistinguishable from the original. We briefly describe two approaches, silence compression and companding.

The principle of silence compression is to treat small samples as if they were silence (i.e., as samples of 0). This generates run lengths of zero, so silence compression is actually a variant of RLE, suitable for sound compression. This method uses the fact that some people have less sensitive hearing than others, and will tolerate the loss of sound that is so quiet they may not hear it anyway. Audio files containing long periods of low-volume sound will respond to silence compression better than other files with high-volume sound. This method requires a user-controlled parameter that specifies the largest sample that should be terminate. Two other parameters are also necessary, although they may not have to be user-controlled. One specifies the shortest run length of small samples, typically 2 or 3. The other specifies the minimum number of consecutive large samples that should terminate a run of silence. For example, a run of 15 small samples, followed by two large samples, followed by 13 small samples may be considered one silence run of 30 samples, whereas the runs 15, 2, 13 may become two distinct silence runs of 15 and 13 samples, with nonsilence in between.

Companding (short for "compressing/expanding") uses the fact that the ear requires more precise samples at low amplitudes (soft sounds), but is more forgiving at higher amplitudes. A typical ADC used in sound cards for personal computers converts voltages to numbers linearly. If an amplitude a is converted to the number n, then amplitude 2a will be converted to the number 2n. A compression method using companding examines every sample in the sound file, and employs a nonlinear formula to reduce the number of bits devoted to it. For 16-bit samples, for example, a companding encoder may use a formula as simple as

$$mapped = 32767(2^{\frac{sample}{65536}} - 1) \qquad \text{Eq. (3. 1)}$$

to reduce each sample. This formula maps the 16-bit samples nonlinearly to 15-bit numbers (i.e., numbers in the range [0, 32767]) such that small samples are less affected than large ones. Table 6 illustrates the nonlinearity of this mapping. It shows eight pairs of samples, where the two samples in each pair differ by 100. The two samples of the first pair get mapped to numbers that differ by 34, whereas the two samples of the last pair are mapped to numbers that differ by 65. The mapped 15-bit numbers can be decoded back into the original 16-bit samples by the inverse formula

$$sample = 65536 * log_2(1 + \frac{mapping}{32767}) \qquad \text{Eq. (3.2)}$$

sample	mapping	diff	sample	mapping	diff
100	35	34	30000	12236	47
200	69		30100	12283	
1000	348	35	40000	17256	53
1100	383		40100	17309	
10000	3656	38	50000	22837	59
10100	3694		50100	22896	
20000	7719	43	60000	29040	65
20100	7762		60100	29105	

Table 6: 16-Bit sample Mapped to 15-Bit Numbers

Reducing 16-bit numbers to 15 bits doesn't produce much compression. Better compression can be achieved by substituting a smaller number for 32,767 in equations (3.1) and (3.2). A value of 127, for example, would map each 16-bit sample into an 8-bit one using 127 instead 32767, yielding a compression ratio of 0.5. However, decoding would be less accurate. A 16-bit sample of 60,100, for example, would be mapped into the 8-bit number 113, but this number would produce 60,172 when decoded by Equation (3.2). Even worse, the small 16-bit sample 1000 would be mapped into 1.35, which has to be rounded to number 1. When Equation (3.2) is used to decode a 1, it produces 742, significantly different from the original sample.

Companding is not limited to Equations (3.1) and (3.2). More sophisticated methods, such as μ-law and A-law, are commonly used and have been designated international standards.

3.4 μ-Law and A-Law Companding

μ-Law and A-Law Companding international standard known as G.711. They employ logarithm-based functions to encode audio samples for ISDN (integrated services digital network) digital telephony services, by means of nonlinear quantization. The ISDN hardware samples the voice signal from the telephone 8,000 times per second, and generates 14-bit samples (13 for A-law). The method of μ-law companding is used in North America and Japan, and A-law is used elsewhere. The two methods are similar; they differ mostly in their quantization’s.

The low amplitudes of speech signals contain more information than the high amplitudes. This is why use nonlinear quantization. Imagine an audio signal sent on a telephone line and digitized to 14-bit samples. The louder the conversation, the higher the amplitude, and the bigger the value of the sample. Since high amplitudes are less important, they can be coarsely quantized. If the largest sample, which is $2^{14} - 1 = 16,383$, is quantized to 255 (the largest 8-bit number), then the compression factor is 14/8 = 1.75. When decoded, a code of 255 will become very different from the original 16,383. We say that because of the coarse quantization, large samples end up with high quantization noise. Smaller samples should be finely quantized, so they end up with low quantization noise.

The μ-law encoder inputs 14-bit samples and outputs 8-bit codewords. The A- law inputs 13-bit samples and also outputs 8-bit codewords. The telephone signals are sampled at 8 kHz (8,000 times per second), so the μ-law encoder receives $8,000 \times 14 = 112,000$ bits/sec. At a compression factor of 1.75, the encoder outputs(112000/1.75=64,000 bits/sec).

Logarithms are slow to compute, so the μ-law encoder performs much simpler calculations that produce an approximation. The output specified by the G.711 standard is an 8-bit codeword whose format is shown in Figure 19.

P	S2	S1	S0	Q3	Q2	Q1	Q0

Fig. (19): G.711 μ-Law Codeword

				Q3	Q2	Q1	Q0					
0	0	0	1	0	1	0	1	1	0	0	0	1
12	11	10	9	8	7	6	5	4	3	2	1	0

Fig. (20): Encoding input sample -656

Bit P in Figure 19 is the sign bit of the output (same as the sign bit of the 14-bit signed input sample). Bits S2, S1, and S0 are the segment code, and bits Q3 through Q0 are the quantization code. The encoder determines the segment code by

(1) adding a bias of 33 to the absolute value of the input sample.
(2) determining the bit position of the most significant 1-bit among bits 5 through 12 of the input.

(3) subtracting 5 from that position. The 4-bit quantization code is set to the four bits following the bit position determined in step 2. The encoder ignores the remaining bits of the input sample, and it inverts (1's complements) the codeword before it is output.

We use the input sample -656 as an example. The sample is negative, so bit P becomes 1. Adding 33 to the absolute value of the input yields 689 = 0001010110001_2 (Figure 20). The most significant 1-bit in positions 5 through 12 is found at position 9.

The segment code is thus 9 − 5 = 4. The quantization code is the four bits 0101 at positions 8-5, and the remaining five bits 10001 are ignored. The 8-bit codeword (which is later inverted) becomes

P	S2	S1	S0	Q3	Q2	Q1	Q0
1	1	0	0	0	1	0	1

The μ-law decoder inputs an 8-bit codeword and inverts it. It then decodes it as follows:

1. Multiply the quantization code by 2 and add 33 (the bias) to the result.

2. Multiply the result by 2 raised to the power of the segment code.
3. Decrement the result by the bias.
4. Use bit P to determine the sign of the result.

Applying these steps to our example produces

1. The quantization code is 101_2 =5, so 5×2+33=43.
2. The segment code is 100_2 =4, so 43× 2^4 =688.
3. Decrement by the bias 688 − 33 = 655.
4. Bit P is 1, so the final result is −655. Thus, the quantization error (the noise) is 1; very small.