Chapter 3

IMAGE AND VIDEO COMPRESSION

Loss less techniques of image compression, gray codes, Two dimensional image transforms, JPEG, JPEG 2000, Predictive Techniques PCM and DPCM, Video compression and MPEG industry standard.

1. Introduction:

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.

The purpose of compression is to code the image data into a compact form, minimizing both the number of bits in the representation, and the distortion caused by the compression. The importance of image compression is emphasized by the huge amount of data in raster images: a typical gray-scale image of 512×512 pixels, each represented by 8 bits, contain 256 kilobytes of data. With the color information, the number of bytes is tripled. If we talk about video images of 25 frames per second, even a one second of color film requires approximately 19 megabytes of memory. Thus, the necessity for compression is obvious.

Image compression addresses the problem of reducing the amount of data required to represent a digital image. The underlying basis of the reduction process is the removal of redundant data. From a mathematical viewpoint, this amounts to transforming a 2-D pixel array into a statistically uncorrelated data set. The transformation is applied prior to storage or transmission of the image. At some later time, the compressed image is decompressed to reconstruct the original image or an approximation of it.

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. Each pixel in such an image is represented by one bit, making this the simplest type of image.

2. A ***grayscale*** **i**mage. A pixel in such an image can have one of the *n* values 0 through *n* − 1, indicating one of 2^n shades of gray (or shades of some other color). The value of *n* is normally

compatible with a byte size; i.e., it is 4, 8, 12, 16, 24, or some other convenient multiple of 4 or of 8. The set of the most-significant bits of all the pixels is the most-significant bitplane. Thus, a grayscale image has *n* bitplanes.

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. As a result, such an image may contain areas with colors that seem to vary continuously as the eye moves along the area. A pixel in such an image is represented by either a single large number (in the case of many grayscales) or three components (in the case of a color image). A continuous-tone image is normally a natural image (natural as opposed to artificial) and is obtained by taking a photograph with a digital camera, or by scanning a photograph or a painting.

4. A ***discrete-tone*** image (also called a graphical image or a synthetic image). This is normally an artificial image. It may have a few colors or many colors, but it does not have the noise and blurring of a natural image. Examples are an artificial object or machine, a page of text, a chart, a cartoon, or the contents of a computer screen. Artificial objects, text, and line drawings have sharp, well-defined edges, and are therefore highly contrasted from the rest of the image (the background). Adjacent pixels in a discretetone image often are either identical or vary significantly in value. Such an image does not compress well with lossy methods, because the loss of just a few pixels may render a letter illegible, or change a familiar pattern to an unrecognizable one.

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.

2. Introduction to image compression

The term data compression refers to the process of reducing the amount of data required to represent a given quantity of information. A clear distinction must be made between ***data*** and ***information***. They are not synonymous. In fact, ***data are the means by which information is conveyed***. Various amounts of data may be used to represent the same amount of information. That is, it contains data (or words) that either provide no relevant information or simply restate that which is already known. It is thus said to contain data redundancy.

Data redundancy is a central issue in digital image compression. It is e compression. It is not an abstract concept but a mathematically quantifiable entity. If $\mathbf{n_1}$ and $\mathbf{n_2}$ denote the number of information-carrying units in two data sets that represent the same information, the relative data redundancy $\mathbf{R_D}$ of the first data set (the one characterized by n_1) can be defined as,

$$R_D = 1 - \frac{1}{C_R}$$

Where C_R commonly called the **compression ratio**, as

$$C_R = \frac{n_1}{n_2}$$

For the case $n_2 = n_1$, $C_R = 1$ and $R_D = 0$, indicating that (relative to the second data set) the first representation of the information contains no redundant data. When $\mathbf{n_2 \ll n_1}$, $\mathbf{CR \rightarrow \infty}$ **and** $\mathbf{R_D \rightarrow -\infty}$ implying significant compression and **highly redundant data**. Finally, when $n_2 \gg \mathbf{n_1}$, $\mathbf{C_R \rightarrow 0}$ **and** $\mathbf{R_D \rightarrow -\infty}$, indicating that the second data set contains much more data than the original representation. In general, C_R and R_D lie in the open intervals (0, ∞) and (-∞, 1), respectively. A practical compression ratio, such as 10 (or 10:1), means that the first data set has 10 information carrying units (say, bits) for every 1 unit in the second or compressed data set. The corresponding redundancy of 0.9 implies that 90% of the data in the first data set is redundant.

In digital image compression, three basic data redundancies can be identified and exploited:

1. coding redundancy,
2. interpixel redundancy,
3. Psychovisual redundancy.

Data compression is achieved when one or more of these redundancies are reduced or eliminated.

2.1 Coding Redundancy

We know that how the gray-level histogram of an image can provide a great deal of insight into the construction of codes to reduce the amount of data used to represent it. Let us assume, that a discrete random variable $\mathbf{r_k}$ in the interval [0,1] represents the gray levels of an image and that each r_k occurs with probability $\mathbf{p_r(r_k)}$, which is given by,

$$p_r(r_k) = \frac{n_k}{n}$$

where L is the number of gray levels, n_k is the number of times that the *kth* gray level appears in the image, and n is the total number of pixels in the image. If the number of bits used to represent each value of r_k is $l(r_k)$, then the average number of bits required to represent each pixel is

$$L_{avg} = \sum_{k=0}^{L-1} l(r_k) p_r(r_k)$$

That is, the average length of the code words assigned to the various gray-level values is found by summing the product of the number of bits used to represent each gray level and the probability that the gray level occurs. Thus the total number of bits required to code an M X N image is $\mathbf{MNL_{avg}}$.

Assigning fewer bits to the more probable gray levels than to the less probable ones achieves data compression. This process commonly is referred to as **variable-length coding**. If the gray levels of an image are coded in a way that uses more code symbols than absolutely necessary to represent each gray level, the resulting image is said to contain **coding redundancy**. In general, coding redundancy is present when the codes assigned to a set of events (such as gray-level values) have not been selected to take full advantage of the probabilities of the events. It is almost always present when an image's gray levels are represented with a straight or natural binary code. In this case, the underlying basis for the coding redundancy is that images are typically composed of objects that have a regular and somewhat predictable morphology (shape) and reflectance, and are generally sampled so that the objects being depicted are much larger than the picture elements. The natural consequence is that, in most images, certain gray levels are more probable than others. A natural binary coding of their gray levels assigns the same number of bits to both the most and least probable values, thus failing to minimize $\mathbf{L_{avg}}$ and resulting in coding redundancy.

2.2 Interpixel Redundancy

Consider the images shown in Figs. 1(a) and (b). As Figs. 1(c) and (d) show, these images have virtually identical histograms. Note also that both histograms are **trimodal,** indicating the presence of three dominant ranges of gray-level values. Because the gray levels in these images are not equally probable, **variable-length coding** can be used to reduce the coding redundancy that would result from a straight or natural binary encoding of their pixels. The coding process, however, would not alter the level of correlation between the pixels within the images. In other words, the codes used to represent the **gray levels of each image have nothing to do with the correlation between pixels.** These correlations result from the structural or geometric relationships between the objects in the image.

These illustrations reflect another important form of data redundancy—one directly related to the interpixel correlations within an image. Because the value of any given pixel can be reasonably predicted from the value of its neighbors, the information carried by individual pixels is relatively small. Much of the visual contribution of a single pixel to an image is redundant; it could have been guessed on the basis of the values of its neighbors. A variety of names, including **spatial redundancy**, **geometric redundancy**, and **interframe redun**dancy, have been coined to refer to these interpixel dependencies. We use the term ***interpixel redundancy*** to encompass them all.

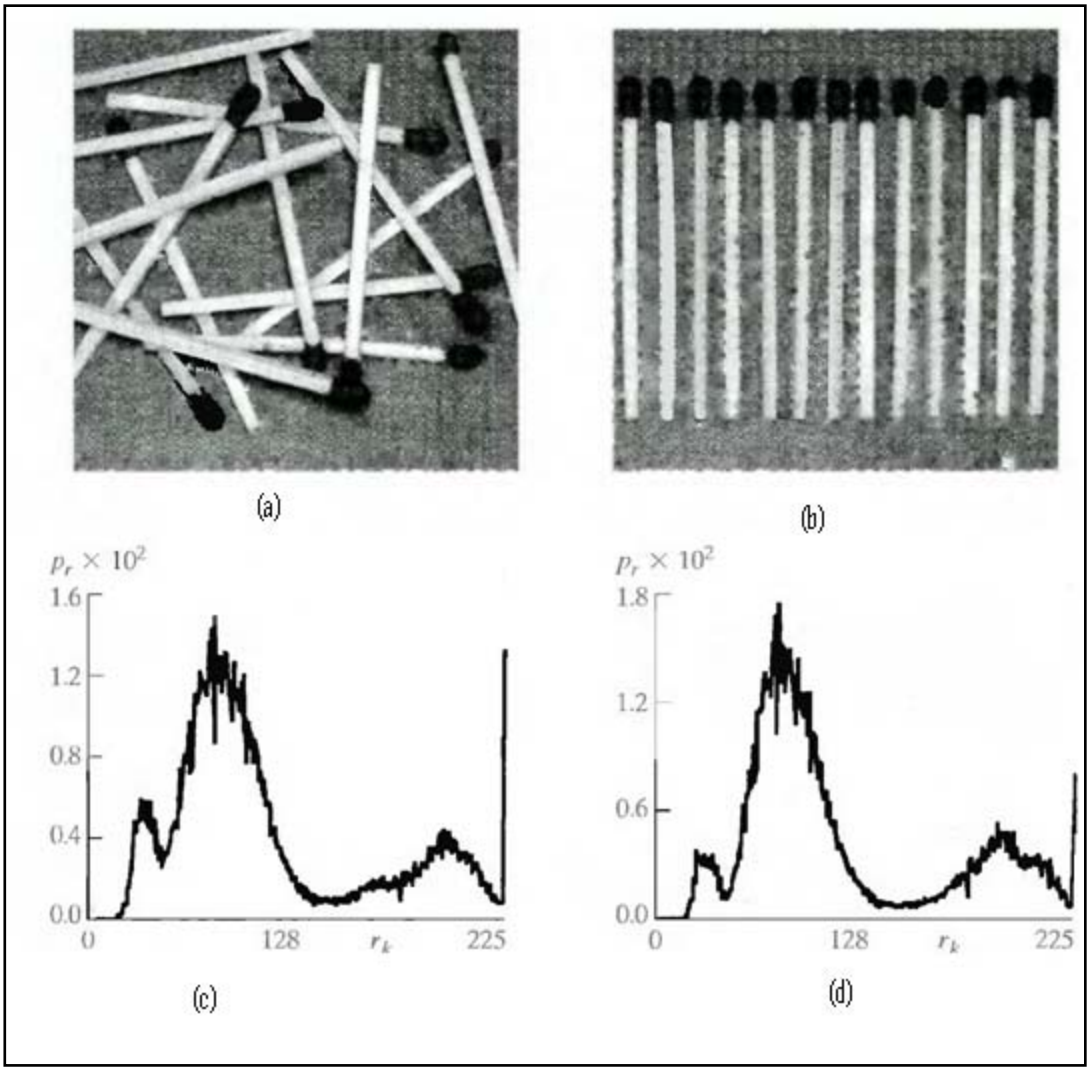


Figure 1: Two images (a) and (b) and their gray- level histograms (c) and (D)

In order to reduce the interpixel redundancies in an image, the 2-D pixel array normally used for human viewing and interpretation must be transformed into a more efficient (but usually "nonvisual") format. For example, the differences between adjacent pixels can be used to represent an image. Transformations of this type (that is, those that remove interpixel redundancy) are referred to as **mappings**. They are called **reversible mappings** if the original image elements can be reconstructed from the transformed data set.

2.3 Psychovisual Redundancy

We know that the brightness of a region, as perceived by the eye, depends on factors other than simply the light reflected by the region. For example, intensity variations (Mach bands) can be perceived in an area of constant intensity. Such phenomena result from the fact that the eye does

not respond with equal sensitivity to all visual information. Certain information simply has less relative importance than other information in normal visual processing. This information is said to be **psychovisually redundant**. It can be eliminated without significantly impairing the quality of image perception.

That psychovisual redundancies exist should not come as a surprise, because human perception of the information in an image normally does not involve quantitative analysis of every pixel value in the image. In general, **an observer searches for distinguishing features such as edges or textural regions** and mentally combines them into recognizable groupings. The brain then correlates these groupings with prior knowledge in order to complete the image interpretation process.

Psychovisual redundancy is fundamentally different from the redundancies discussed earlier. Unlike coding and interpixel redundancy, psychovisual redundancy is associated with real or quantifiable visual information. Its elimination is possible only because the information itself is not essential for normal visual processing. Since the elimination of psychovisually redundant data results in a loss of quantitative information, it is commonly referred to as **quantization.** This terminology is consistent with normal usage of the word, which generally means the mapping of a broad range of input values to a limited number of output values. As it is an **irreversible operation** (visual information is lost), quantization results in **lossy data compression**.

Improved gray-scale (IGS) quantization method recognizes the eye's inherent sensitivity to edges and breaks them up by adding to each pixel a pseudorandom number, which is generated from the low-order bits of neighboring pixels, before quantizing the result. Because the low-order bits arc fairly random, this amounts to adding a level of randomness, which depends on the local characteristics of the image, to the artificial edges normally associated with false contouring.

3. Approaches to Image Compression

Approach 1: This is appropriate for **bi-level images**. A pixel in such an image is represented by one bit. Applying the principle of image compression to a bi-level image therefore means that the immediate neighbors of a pixel *P* tend to be *identical* to *P*. Thus, it makes sense to use run-length encoding (**RLE**) to compress such an image. A compression method for such an image may scan it in raster order (row by row) and compute the lengths of runs of black and white pixels. The lengths are encoded by variable-size (prefix) codes and are written on the compressed stream. An example of such a method is **facsimile compression**.

Approach 2: Also for bi-level images. The principle of image compression tells us that the neighbors of a pixel tend to be similar to the pixel. We can extend this principle and conclude that if the current pixel has color *c* (where *c* is either black or white), then pixels of the same color seen in the past (and also those that will be found in the future) tend to have the same immediate neighbors.

This approach looks at *n* of the near neighbors of the current pixel and considers them an *n*-bit number. This number is the *context* of the pixel. In principle there can be 2*n* contexts, but because of image redundancy we expect them to be distributed in a nonuniform way. Some contexts should be common while others will be rare. This approach is used by **JBIG.**

Approach 3: Separate the grayscale image into n bi-level images and compress each with RLE and prefix codes. The principle of image compression seems to imply intuitively that two adjacent pixels that are similar in the grayscale image will be identical in most of the n bi-level images. This, however, is not true. An example of such a code is the **reflected Gray codes.**

Approach 4: Use the **context** of a pixel to predict its value. The context of a pixel is the values of some of its neighbors. We can examine some neighbors of a pixel P, compute an average A of their values, and predict that *P* will have the value *A*. The principle of image compression tells us that our prediction will be correct in most cases, almost correct in many cases, and completely wrong in a few cases. This is used in **MLP** method**.**

Approach 5: Transform the values of the pixels and encode the transformed values. Recall that compression is achieved by reducing or removing redundancy. The redundancy of an image is caused by the correlation between pixels, so transforming the pixels to a representation where they are **decorrelated** eliminates the redundancy. It is also possible to think of a transform in terms of the entropy of the image. In a highly correlated image, the pixels tend to have **equiprobable values**, which results in **maximum entropy**. If the transformed pixels are decorrelated, certain pixel values become common, thereby having large probabilities, while others are rare. This results in small entropy. Quantizing the transformed values can produce efficient lossy image compression.

Approach 6: The principle of this approach is to separate a continuous-tone color image into three grayscale images and compress each of the three separately; using approaches 3, 4, or 5. For a continuous-tone image, the principle of image

An important feature of this approach is to use a luminance chrominance color representation instead of the more common RGB. The advantage of the luminance chrominance color representation is that the eye is sensitive to small changes in luminance but not in

chrominance. This allows the loss of considerable data in the chrominance components, while making it possible to decode the image without a significant visible loss of quality.

Approach 7: A different approach is needed for discrete-tone images. Recall that such an image contains uniform regions, and a region may appear several times in the image. A good example is a screen dump. Such an image consists of text and icons. Each character of text and each icon is a region, and any region may appear several times in the image. A possible way to compress such an image is to scan it, identify regions, and find repeating regions. If a region *B* is identical to an already found region *A*, then *B* can be compressed by writing a pointer to *A* on the compressed stream. The **block decomposition method** (FABD) is an example of how this approach can be implemented.

Approach 8: Partition the image into parts (overlapping or not) and compress it by processing the parts one by one. Suppose that the next unprocessed image part is part number 15. Try to match it with parts 1–14 that have already been processed. If part 15 can be expressed, for example, as a combination of parts 5 (scaled) and 11 (rotated), then only the few numbers that specify the combination need be saved, and part 15 can be discarded. If part 15 cannot be expressed as a combination of already-processed parts, it is declared processed and is saved in raw format.

This approach is the basis of the various ***fractal*** methods for image compression. It applies the principle of image compression to image parts instead of to individual pixels. Applied this way, the principle tells us that "interesting" images (i.e., those that are being compressed in practice) have a certain amount of *self similarity*. Parts of the image are identical or similar to the entire image or to other parts.

4. Gray Codes and its significance for image compression

An image compression method that has been developed specifically for a certain type of image can sometimes be used for other types. Any method for compressing bi-level images, for example, can be used to compress grayscale images by separating the bitplanes and compressing each individually, as if it were a bi-level image. Imagine, for example, an image with 16 grayscale values. Each pixel is defined by four bits, so the image can be separated into four bi-level images. The trouble with this approach is that it violates the general principle of image compression. Imagine two adjacent 4-bit pixels with values $7 = 0111_2$ and $8 = 1000_2$. These pixels have close values, but when separated into four bitplanes, the resulting 1-bit pixels are different in every bitplane! This is because the binary representations of the consecutive integers 7 and 8 differ in all four bit positions. In order to apply any bi-level compression method to grayscale images, a binary

representation of the integers is needed where consecutive integers have codes differing by one bit only. Such a representation exists and is called ***reflected Gray code*** **(RGC**).

The conclusion is that the **most-significant bitplanes of an image obey the principle of image compression more than the least-significant ones.** When adjacent pixels have values that differ by one unit (such as p and p+1), chances are that the least-significant bits are different and the most-significant ones are identical. Any image compression method that compresses bitplanes individually should therefore treat the least-significant bitplanes differently from the most-significant ones, or should use RGC instead of the binary code to represent pixels.. The bitplanes are numbered 8 (the leftmost or most-significant bits) through 1 (the rightmost or least-significant bits). It is obvious that the least-significant bitplane doesn't show any correlations between the pixels; it is random or very close to random in both binary and RGC. Bitplanes 2 through 5, however, exhibit better pixel correlation in the Gray code. Bitplanes 6 through 8 look different in Gray code and binary, but seem to be highly correlated in either representation.

Color images provide another example of using the same compression method across image types. Any compression method for grayscale images can be used to compress color images. In a color image, each pixel is represented by three color components (such as RGB). Imagine a color image where each color component is represented by one byte. A pixel is represented by three bytes, or 24 bits, but these bits should not be considered a single number. The two pixels 118/206/12 and 117/206/12 differ by just one unit in the first component, so they have very similar colors. Considered as 24-bit numbers, however, these pixels are very different, since they differ in one of their most significant bits. Any compression method that treats these pixels as 24-bit numbers would consider these pixels very different, and its performance would suffer as a result.

A compression method for grayscale images can be applied to compressing color images, but the color image should first be separated into three color components, and each component compressed individually as a grayscale image.

5. Error Metrics

Developers and implementers of lossy image compression methods need a standard metric to measure the quality of reconstructed images compared with the original ones. The better a reconstructed image resembles the original one, the bigger should be the value produced by this metric. Such a metric should also produce a dimensionless number, and that number should not be very sensitive to small variations in the reconstructed image.

A common measure used for this purpose is the ***peak signal to noise ratio*** (PSNR). Higher PSNR values imply closer resemblance between the reconstructed and the original images, but they do not provide a guarantee that viewers will like the reconstructed image. Denoting the pixels of the original image by P_i and the pixels of the reconstructed image by Q_i (where $1 \le i \le n$), we first define the ***mean square error*** (MSE) between the two images as

$$\boldsymbol{MSE = \frac{1}{n}\sum_{i=1}^{n}(P_i - Q_i)^2}$$

It is the average of the square of the errors (pixel differences) of the two images. The ***root mean square error*** (RMSE) is defined as the square root of the MSE, and the PSNR is defined as

$$\boldsymbol{PSNR = 20\ log_{10}\frac{max_i|P_i|}{RMSE}}$$

The absolute value is normally not needed, since pixel values are rarely negative. For a bi-level image, the numerator is 1. For a grayscale image with eight bits per pixel, the numerator is 255. For color images, only the luminance component is used. Greater resemblance between the images implies smaller RMSE and, as a result, larger PSNR. The PSNR is dimensionless, since the units of both numerator and denominator are pixel values. However, because of the use of the logarithm, we say that the PSNR is expressed in *decibels* (dB). The use of the logarithm also implies less sensitivity to changes in the RMSE. Notice that the PSNR has no absolute meaning. It is meaningless to say that a PSNR of, say, 25 is good. PSNR values are used only to compare the performance of different lossy compression methods or the effects of different parametric values on the performance of an algorithm.

Typical PSNR values range between 20 and 40. Assuming pixel values in the range [0, 255], an RMSE of 25.5 results in a PSNR of 20, and an RMSE of 2.55 results in a PSNR of 40. An RMSE of zero (i.e., identical images) results in an infinite (or, more precisely, undefined) PSNR. An RMSE of 255 results in a PSNR of zero, and RMSE values greater than 255 yield negative PSNRs.

A related measure is ***signal to noise ratio*** (SNR). This is defined as

$$\boldsymbol{SNR = 20\ log_{10}\frac{\sqrt{\frac{1}{n}\sum_{i=1}^{n}P_i^2}}{RMSE}}$$

The numerator is the root mean square of the original image.

Another relative of the PSNR is the ***signal to quantization noise ratio*** *(SQNR)*. This is a measure of the effect of quantization on signal quality. It is defined as

$$\boldsymbol{SQNR = 10\ log_{10}\frac{signal\ power}{quantization\ error}}$$

where the quantization error is the difference between the quantized signal and the original signal.

Another approach to the comparison of an original and a reconstructed image is to generate the difference image and judge it visually. Intuitively, the difference image is $D_i = P_i - Q_i$, but such an image is hard to judge visually because its pixel values D_i tend to be small numbers. If a pixel value of zero represents white, such a difference image would be almost invisible. In the opposite case, where pixel values of zero represent black, such a difference would be too dark to judge. Better results are obtained by calculating

$$\boldsymbol{D_i = a(P_i - Q_i) + b}$$

where a is a magnification parameter (typically a small number such as 2) and b is half the maximum value of a pixel (typically 128). Parameter a serves to magnify small differences, while b shifts the difference image from extreme white (or extreme black) to a more comfortable gray.

6. Image Transforms

An image can be compressed by transforming its pixels (which are correlated) to a representation where they are ***decorrelated*****.** Compression is achieved if the new values are smaller, on average, than the original ones. Lossy compression can be achieved by quantizing the transformed values. The decoder inputs the transformed values from the compressed stream and reconstructs the (precise or approximate) original data by applying the inverse transform. The transforms discussed in this section are *orthogonal*.

The term *decorrelated* means that the transformed values are independent of one another. As a result, they can be encoded independently, which makes it simpler to construct a statistical model. An image can be compressed if its representation has redundancy. The redundancy in images stems from pixel correlation. If we transform the image to a representation where the pixels are decorrelated, we have eliminated the redundancy and the image has been fully compressed.

6.1 Orthogonal Transforms

Image transforms are designed to have two properties:

1. to reduce image redundancy by reducing the sizes of most pixels and
2. to identify the less important parts of the image by isolating the various frequencies of the image.

We intuitively associate a frequency with a wave. Water waves, sound waves, and electromagnetic waves have frequencies, but pixels in an image can also feature frequencies. Figure 2 shows a small, 5×8 bi-level image that illustrates this concept. The top row is uniform, so we can assign it zero frequency. The rows below it have increasing pixel frequencies as measured by the number of color

changes along a row. The four waves on the right roughly correspond to the frequencies of the four top rows of the image.

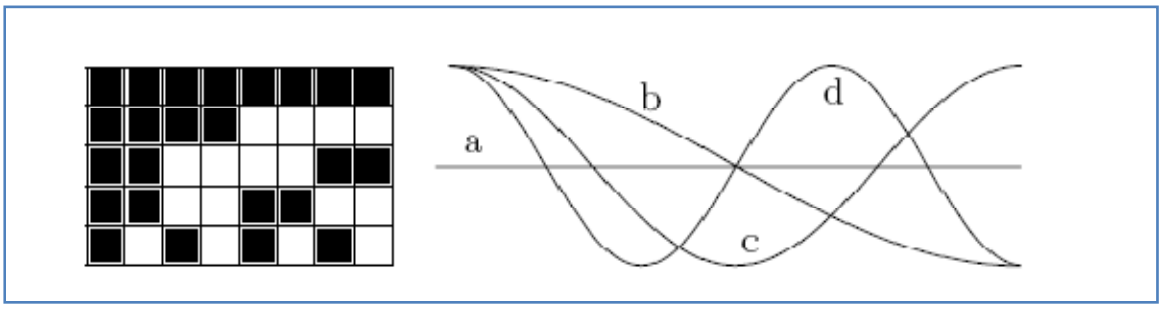


Figure 2: Image frequencies

Image frequencies are important because of the following basic fact: ***Low frequencies correspond to the important image features, whereas high frequencies correspond to the details of the image, which are less important.*** Thus, when a transform isolates the various image frequencies, pixels that correspond to high frequencies can be quantized heavily, whereas pixels that correspond to low frequencies should be quantized lightly or not at all. This is how a transform can compress an image very effectively by losing information, but only information associated with unimportant image details.

Practical image transforms should be fast and preferably also simple to implement. This suggests the use of ***linear transforms***. In such a transform, each transformed value (or transform coefficient) c_i is a weighted sum of the data items (the pixels) d_j that are being transformed, where each item is multiplied by a weight w_{ij}. Thus, $\boldsymbol{C_i = \sum_j d_j w_{ij}}$ for $i, j = 1, 2, \ldots, n$. For $n = 4$, this is expressed in matrix notation:

$$\begin{pmatrix} c_1 \\ c_2 \\ c_3 \\ c_4 \end{pmatrix} = \begin{pmatrix} w_{11} & w_{12} & w_{13} & w_{14} \\ w_{21} & w_{22} & w_{23} & w_{24} \\ w_{31} & w_{32} & w_{33} & w_{34} \\ w_{41} & w_{42} & w_{43} & w_{44} \end{pmatrix} \begin{pmatrix} d_1 \\ d_2 \\ d_3 \\ d_4 \end{pmatrix}$$

For the general case, we can write **C** =**W.D**. Each row of **W** is called a "basis vector." The only quantities that have to be computed are the weights w_{ij} . The guiding principles are as follows:

1. ***Reducing redundancy***. The first transform coefficient c_1 can be large, but the remaining values $c_2, c_3, \ldots$ should be small.

2. ***Isolating frequencies***. The first transform coefficient c_1 should correspond to zero pixel frequency, and the remaining coefficients should correspond to higher and higher frequencies.

The key to determining the weights w_{ij} is the fact that our data items d_j are not arbitrary numbers but pixel values, which are ***nonnegative and correlated.***

This choice of w_{ij} satisfies the first requirement: ***to reduce pixel redundancy by means of a transform.*** In order to satisfy the second requirement, the weights w_{ij} of row i should feature frequencies that get higher with i. Weights w_{1j} should have zero frequency; they should all be +1's. Weights w_{1j} should have one sign change; i.e., they should be +1, +1, . . . + 1,−1,−1, . . . ,−1. This continues until the last row of weights w_{nj} should have the highest frequency +1,−1, +1,−1, . . . , +1,−1. The mathematical discipline of vector spaces coins the term "basis vectors" for our rows of weights.

In addition to isolating the various frequencies of pixels d_j, this choice results in basis vectors that are ***orthogonal.*** The basis vectors are the rows of matrix **W**, which is why this matrix and, by implication, the entire transform are also termed orthogonal. These considerations are satisfied by the orthogonal matrix

$$\begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix}$$

The first basis vector (the top row of **W**) consists of all 1's, so its frequency is zero. Each of the subsequent vectors has two +1's and two −1's, so they produce small transformed values, and their frequencies (measured as the number of sign changes along the basis vector) get higher. It is also possible to modify this transform to conserve the energy of the data vector. All that's needed is to multiply the transformation matrix **W** by the scale factor 1/2. Another advantage of **W** is that it also performs the ***inverse transform***.

6.2 Two-Dimensional Transforms

Given two-dimensional data such as the 4X4 matrix

$$\begin{pmatrix} 5 & 6 & 7 & 4 \\ 6 & 5 & 7 & 5 \\ 7 & 7 & 6 & 6 \\ 8 & 8 & 8 & 8 \end{pmatrix}$$

where each of the four columns is highly correlated, we can apply our simple one dimensional transform to the columns of **D**. The result is,

$$\boldsymbol{C'} = \boldsymbol{W} \cdot \boldsymbol{D} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix} \cdot \begin{pmatrix} 5 & 6 & 7 & 4 \\ 6 & 5 & 7 & 5 \\ 7 & 7 & 6 & 6 \\ 8 & 8 & 8 & 8 \end{pmatrix} = \begin{pmatrix} 26 & 26 & 28 & 23 \\ -4 & -4 & 0 & -5 \\ 0 & 2 & 2 & 1 \\ -2 & 0 & -2 & -3 \end{pmatrix}$$

Each column of **C'** is the transform of a column of **D**. Notice how the top element of each column of **C'** is dominant, because the data in the corresponding column of **D** is correlated. Notice also that the rows of **C'** are still correlated. **C'** is the first stage in a two-stage process that produces the two-dimensional transform of matrix **D**. The second stage should transform each *row* of **C'**, and this is done by multiplying **C'** by the transpose $\mathbf{W}^T$. Our particular **W**, however, is symmetric, so we end up with $\mathbf{C} = \mathbf{C'}.\mathbf{W}^T = \mathbf{W}.\mathbf{D}.\mathbf{W}^T = \mathbf{W}.\mathbf{D}.\mathbf{W}$ or

$$\boldsymbol{C} = \begin{pmatrix} 26 & 26 & 28 & 23 \\ -4 & -4 & 0 & -5 \\ 0 & 2 & 2 & 1 \\ -2 & 0 & -2 & -3 \end{pmatrix} \cdot \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \\ 1 & -1 & 1 & -1 \end{pmatrix} = \begin{pmatrix} 103 & 1 & -5 & 5 \\ -13 & -3 & -5 & 5 \\ 5 & -1 & -3 & -1 \\ -7 & 3 & -3 & -1 \end{pmatrix}$$

The elements of **C** are decorrelated. The top-left element is ***dominant***. It contains most of the total energy of the original **D**. The elements in the top row and the leftmost column are somewhat large, while the remaining elements are smaller than the original data items. The double-stage, two-dimensional transformation has reduced the correlation in both the horizontal and vertical dimensions. As in the one-dimensional case, excellent compression can be achieved by quantizing the elements of **C**, especially those that correspond to higher frequencies (i.e., located toward the bottom-right corner of **C**).

This is the essence of orthogonal transforms. The important transforms are:

1. ***The Walsh-Hadamard transform***: is fast and ***easy*** to compute (it requires only additions and subtractions), but its performance, in terms of energy compaction, is ***lower*** than that of the DCT.

2. ***The Haar transform***: is a ***simple, fast*** transform. It is the simplest wavelet transform.

3. ***The Karhunen-Lo`eve transform***: is the best one theoretically, in the sense of ***energy compaction*** (or, equivalently, pixel decorrelation). However, ***its coefficients are not fixed***; ***they depend on the data to be compressed***. Calculating these coefficients (the basis of the transform) is slow, as is the calculation of the transformed values themselves. Since the coefficients are data dependent, they have to be included in the compressed stream. For these reasons and because the DCT performs almost as well, the KLT is not generally used in practice.

4. ***The discrete cosine transform (DCT):*** is important transform as efficient as the KLT in terms of energy compaction, but it uses ***a fixed basis***, independent of the data. There are also fast methods for calculating the DCT. This method is used by JPEG and MPEG audio.

The 1-D *discrete cosine transform* (DCT) is defined as

$$C(u)=\alpha(u)\sum_{x=0}^{N-1} f(x)\cdot\cos\left[\frac{(2x+1)u\pi}{2N}\right]$$

The input is a set of *n* data values (pixels, audio samples, or other data), and the output is a set of *n* DCT *transform coefficients* (or weights) C(u) . The first coefficient C(0) is called the ***DC coefficient,*** and the rest are referred to as the ***AC coefficients***. Notice that the coefficients are real numbers even if the input data consists of integers. Similarly, the coefficients may be positive or negative even if the input data consists of nonnegative numbers only.

Similarly, the inverse DCT is defined as

$$f(x)=\sum_{u=0}^{N-1}\alpha(u)C(u)\cdot\cos\left[\frac{(2x+1)u\pi}{2N}\right]$$

where

$$\alpha(u)=\begin{cases}\sqrt{1/N} & \text{for} \quad u=0\\ \sqrt{2/N} & \text{for} \quad u=1,2,\ldots,N-1\end{cases}$$

The corresponding 2-D DCT, and the inverse DCT are defined as

$$C(u,v)=\alpha(u)\alpha(v)\sum_{x=0}^{N-1}\sum_{y=0}^{N-1} f(x,y)\cdot\cos\left[\frac{(2x+1)u\pi}{2N}\right]\cdot\cos\left[\frac{(2y+1)v\pi}{2N}\right]$$

and

$$f(x,y)=\sum_{u=0}^{N-1}\sum_{v=0}^{N-1}\alpha(u)\alpha(v)C(u,v)\cdot\cos\left[\frac{(2x+1)u\pi}{2N}\right]\cdot\cos\left[\frac{(2y+1)v\pi}{2N}\right]$$

The advantage of DCT is that it can be expressed without complex numbers. 2-D DCT is also separable (like 2-D Fourier transform), i.e. it can be obtained by two subsequent 1-D DCT.

The important feature of the DCT, the feature that makes it so useful in data compression, is that it takes correlated input data and concentrates its energy in just the first few transform coefficients. If the input data consists of correlated quantities, then most of the ***N*** transform coefficients produced by the DCT are zeros or small numbers, and only a few are large (normally the first ones).

Compressing data with the DCT is therefore done by quantizing the coefficients. The small ones are quantized coarsely (possibly all the way to zero), and the large ones can be quantized finely to the nearest integer. After quantization, the coefficients (or variable-size codes assigned to the coefficients) are written on the compressed stream. Decompression is done by performing the inverse DCT on the quantized coefficients. This results in data items that are not identical to the original ones but are not much different.

In practical applications, the data to be compressed is partitioned into sets of N items each and each set is DCT-transformed and quantized individually. The value of N is critical. Small values of N such as 3, 4, or 6 result in many small sets of data items. Such a small set is transformed to a small set of coefficients where the energy of the original data is concentrated in a few coefficients, but there are only a few coefficients in such a set! Thus, there are not enough small coefficients to quantize. Large values of N result in a few large sets of data. The problem in such a case is that the individual data items of a large set are normally not correlated and therefore result in a set of transform coefficients where all the coefficients are large. Experience indicates that $N = 8$ is a good value, and most data compression methods that employ the DCT use this value of N.

7. JPEG

JPEG is a sophisticated lossy/lossless compression method for ***color or grayscale still images***. 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. JPEG is a compression method, not a complete standard for image representation. This is why it does not specify image features such as pixel aspect ratio, color space, or interleaving of bitmap rows. JPEG has been designed as a compression method for continuous-tone images.

The name JPEG is an acronym that stands for Joint Photographic Experts Group. This was a joint effort by the CCITT and the ISO (the International Standards Organization) that started in June 1987 and produced the first JPEG draft proposal in 1991. The JPEG standard has proved successful and has become widely used for image compression, especially in Web pages.

The main goals of JPEG compression are the following:

1. High compression ratios, especially in cases where image quality is judged as very good to excellent.

2. The use of many parameters, allowing knowledgeable users to experiment and achieve the desired compression/quality trade-off.

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. JPEG includes **four 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 (the trade-off is low compression ratio compared to the lossy modes); and (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.

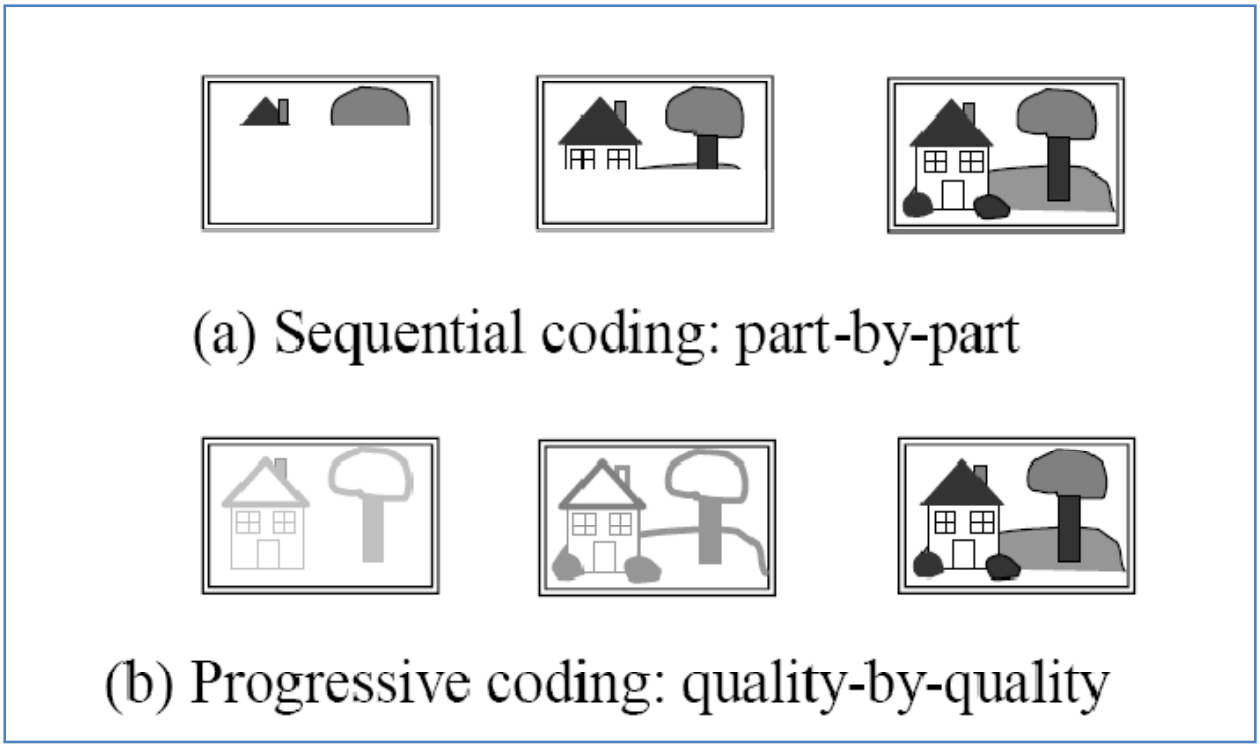


Figure3 Difference between sequential coding and progressive coding

The main JPEG compression steps are:

1. **Color images are transformed from RGB into 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 impairing the overall image quality much. This step is optional but important because the remainder of the algorithm works on each color component separately. Without transforming the color space, none of the three color components will tolerate much loss, leading to worse compression.

2. **Color images are downsampled 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 downsampling is not done for the luminance component. Downsampling is done either at a ratio of 2:1 both horizontally and vertically (the so called 2h2v or 4:1:1 sampling) or at ratios of 2:1 horizontally and 1:1 vertically (2h1v or 4:2:2 sampling). Since this is done on two of the three color components, 2h2v reduces the image to 1/3 + (2/3) × (1/4) = 1/2 its original size, while 2h1v reduces it to 1/3 + (2/3) × (1/2) = 2/3 its original size. Since the luminance component is not touched, there is no noticeable loss of image quality. Grayscale images don't go through this step.

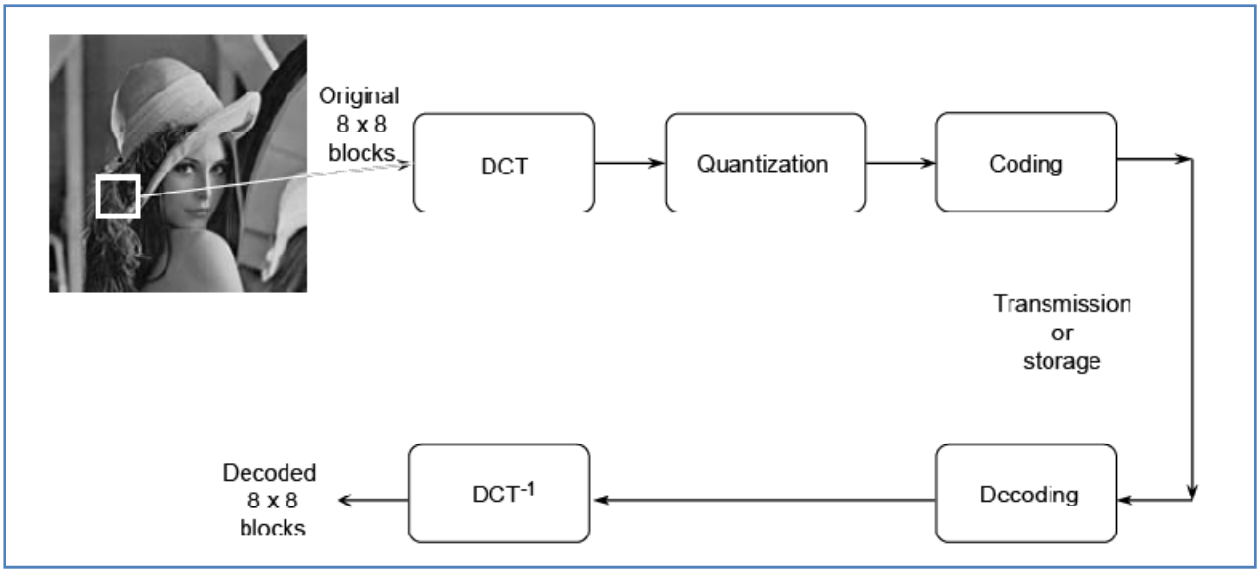


Figure 4: JPEG encoder and decoder

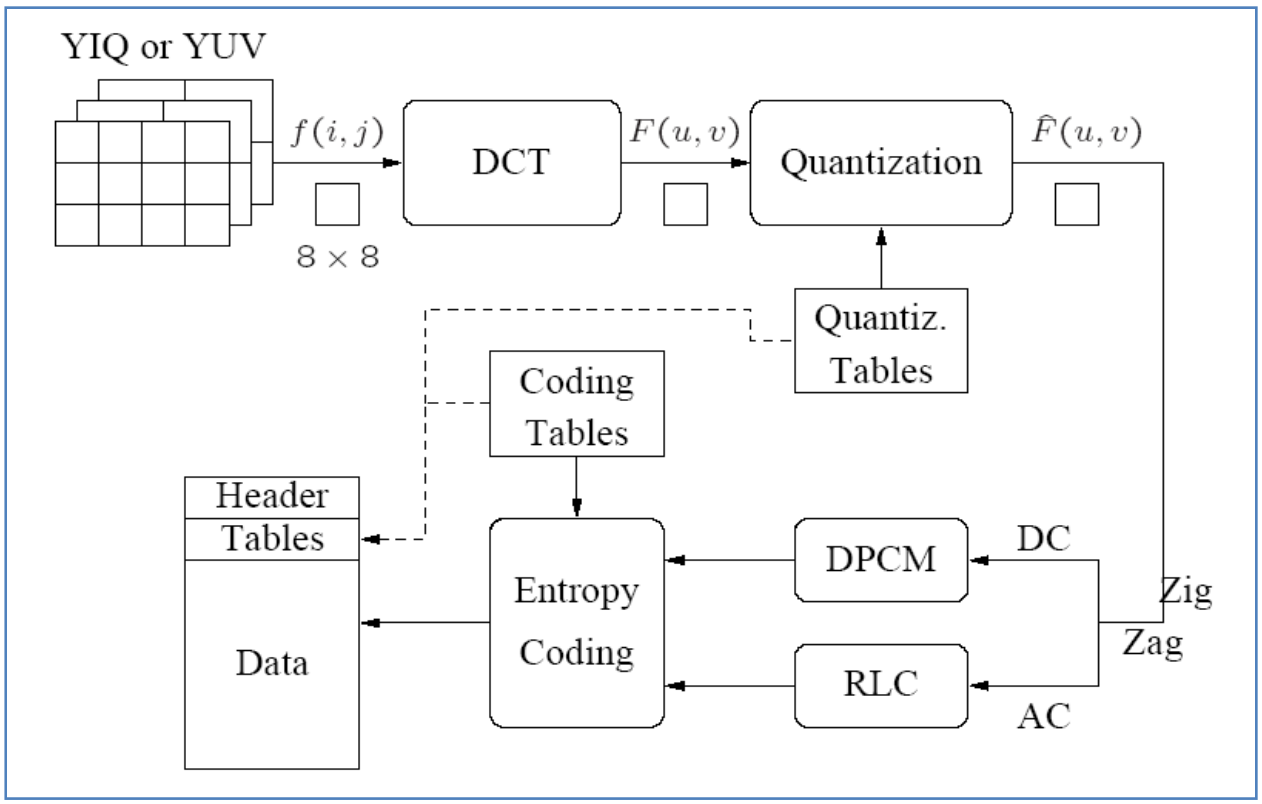


Figure5: JPEG encoder

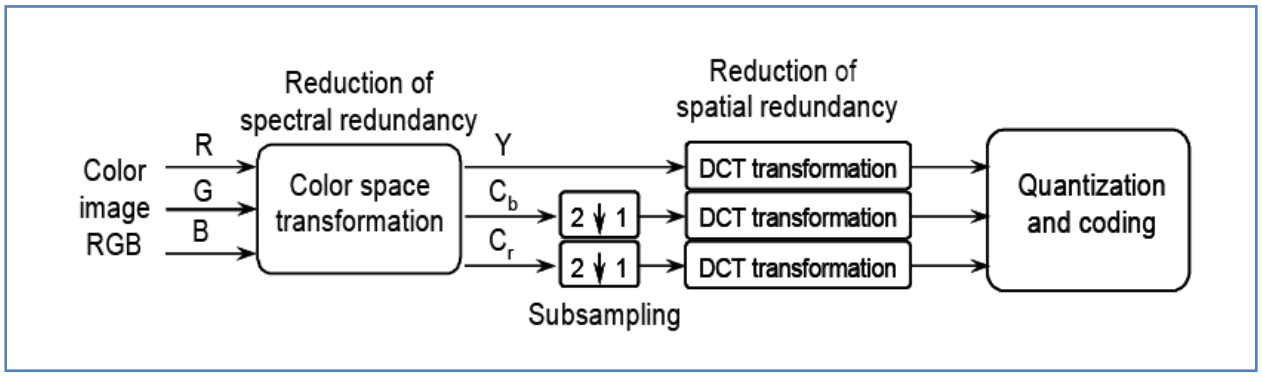


Figure 6: Scheme of the JPEG for RGB images

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. In the noninterleaved mode, the encoder handles all the data units of the first image component, then the data units of the second component, and finally those of the third component. In the interleaved mode the encoder processes the three top-left data units of the three image components, then the three data units to their right, and so on.

4. **The discrete cosine transform 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 crucial step of losing information.

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 irretrievably 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.

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**. 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), and (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.

The JPEG decoder performs the reverse steps. (Thus, JPEG is a symmetric compression method.)

Figure 4 and 5 shows the block diagram of JPEG encoder and decoder. Figure 6 shows JPEG for RGB images.

7.1 Modes of JPEG algorithm:

The **progressive mode** is a JPEG option. In this mode, higher-frequency DCT coefficients are written on the compressed stream in blocks called "scans." Each scan that is read and processed by the decoder results in a sharper image. The idea is to use the first few scans to quickly create a low-quality, blurred preview of the image, and then either input the remaining scans or stop the process and reject the image. The trade-off is that the encoder has to save all the coefficients of all the data units in a memory buffer before they are sent in scans, and also go through all the steps for each scan, slowing down the progressive mode.

In the **hierarchical mode**, the encoder stores the image several times in the output stream, at several resolutions. However, each high-resolution part uses information from the low-resolution parts of the output stream, so the total amount of information is less than that required to store the different resolutions separately. Each hierarchical part may use the progressive mode. The hierarchical mode is useful in cases where a high-resolution image needs to be output in low resolution. Older dot-matrix printers may be a good example of a low-resolution output device still in use.

The **lossless mode** of JPEG calculates a "predicted" value for each pixel, generates the difference between the pixel and its predicted value, and encodes the difference using the same method (i.e., Huffman or arithmetic coding) employed by step 5 above. The predicted value is calculated using values of pixels above and to the left of the current pixel (pixels that have already been input and encoded).

7.2 Why DCT?

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** (the DFT, for example, assumes that the data to be transformed is periodic), and because there are ways to speed it up. DCT has two key advantages: the **decorrelation** of the information by generating coefficients which are almost **independent** of each other and the concentration of this information in a greatly reduced number of coefficients. It **reduces redundancy** while guaranteeing a compact representation.

The JPEG standard calls for applying the DCT not to the entire image but to dataunits (blocks) of 8×8 pixels. 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 for $n = 8$

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. **Mathematically, the DCT is a one-to-one mapping of 64-point vectors from the image domain to the frequency domain.** The IDCT is the reverse mapping. If the DCT and IDCT could be

calculated with infinite precision and if the DCT coefficients were not quantized, the original 64 pixels would be exactly reconstructed.

7.3 **Quantization**

After each 8×8 data unit of DCT coefficients G_{ij} 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. As has already been mentioned, 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 fine-tuned 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 $Q_{ij} = 1+(i + j) \times R$ 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.

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," Three techniques are used by entropy coding to compress the 8 × 8 matrix of integers:

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.

1. The 64 numbers are collected by scanning the matrix in **zigzags**. This produces a string of 64 numbers that starts with some nonzeros and typically ends with many consecutive 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).

7.4 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) 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.

Assume that 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 $64 \times 24/(46 \times 3) \approx 11.13$; **very impressive!**

Each quantized spectral domain is composed of a few non-zero quantized coefficients, and the majority of zero coefficients eliminated in the quantization stage. The positioning of the zeros changes from one block to another. As shown in Figure 7, a zigzag scanning of the block is performed in order to create a vector of coefficients with a lot of zero runlengths. The natural images generally have low frequency characteristics. By beginning the zigzag scanning at the top left (by the low frequency zone), the vector generated will at first contain significant coefficients, and then more and more runlengths of zeros as we move towards the high frequency coefficients. Figure 7gives us an example.

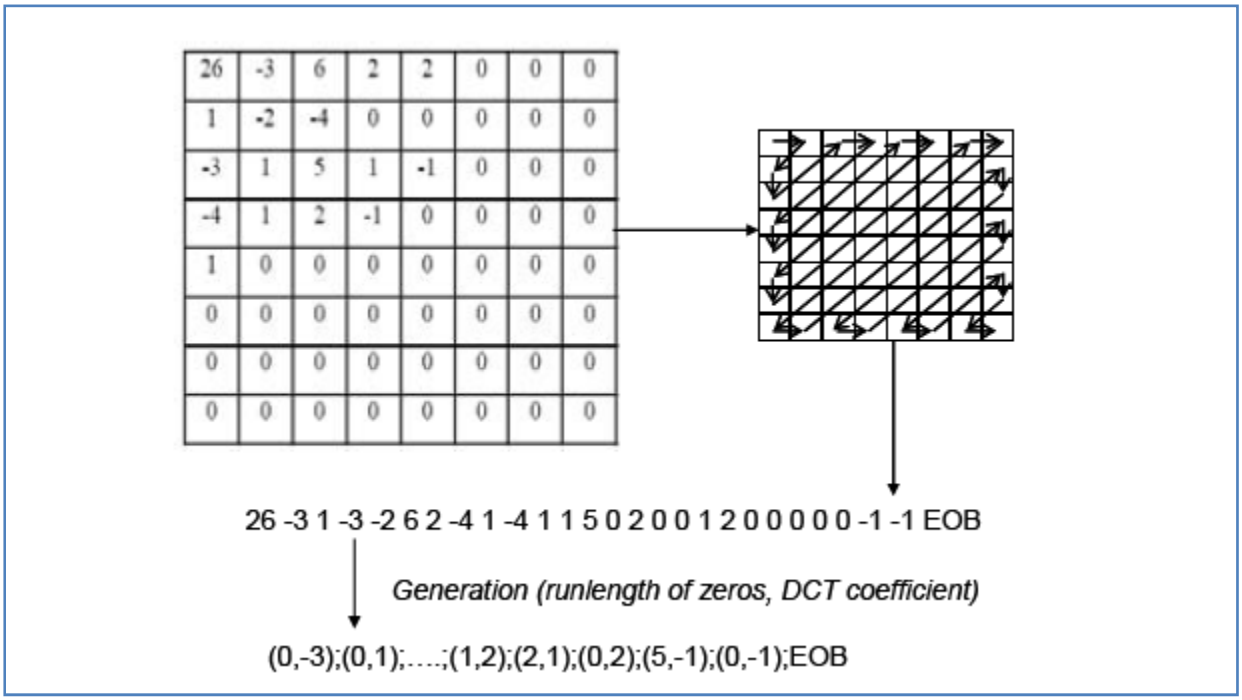


Figure 7. Zigzag scanning of a quantized DCT domain, the resulting coefficient vector, and the generation of pairs (zero runlength, DCT coefficient). EOB stands for "end of block"

Couples of (zero runlengths, DCT coefficient value) are then generated and coded by a set of Huffman coders defined in the JPEG standard. The mean values of the blocks (DC coefficient) are coded separately by a DPCM method. Finally, the ".jpg" file is constructed with the union of the bitstreams associated with the coded blocks.

Why the Zig-Zag Scan:

1. To group low frequency coefficients in top of vector.
2. Maps 8 x 8 to a 1 x 64 vector
3. Zig-Zag scan is more effective

8. JPEG – LS:

JPEG-LS is a new standard for the lossless (or near-lossless) compression of continuous tone images. JPEG-LS examines several of the previously-seen neighbors of the current pixel, uses them as the *context* of the pixel, uses the context to predict the pixel and to select a probability distribution out of several such distributions, and uses that distribution to encode the prediction error with a special Golomb code. There is also a run mode, where the length of a run of identical pixels is encoded. Figure 8 below shows the block diagram of JPEG-LS encoder.

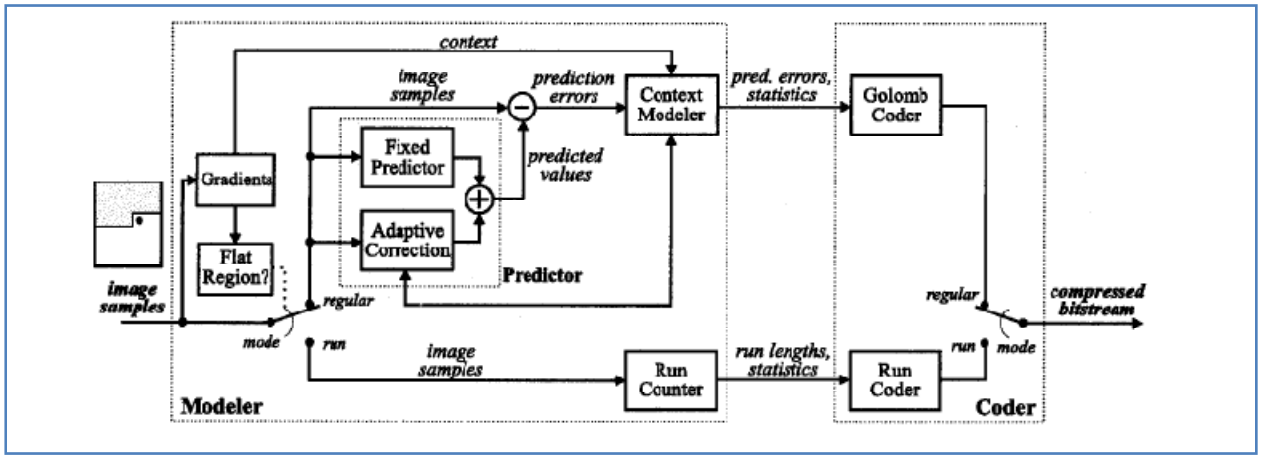


Figure 8: JPEG – LS Block diagram

The context used to predict the current pixel *x* is shown in Figure 9. The encoder examines the context pixels and decides whether to encode the current pixel *x* in the *run mode* or in the

regular mode. If the context suggests that the pixels *y*, *z*,. . . following the current pixel are likely to be identical, the encoder selects the run mode. Otherwise, it selects the regular mode. In the near-lossless mode the decision is slightly different. If the context suggests that the pixels following the current pixel are likely to be almost identical (within the tolerance parameter NEAR), the encoder selects the run mode. Otherwise, it selects the regular mode. The rest of the encoding process depends on the mode selected.

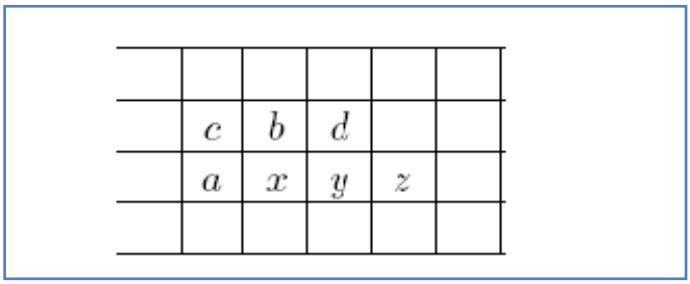


Figure 9: Context for Predicting x.

In the regular mode, the encoder uses the values of context pixels *a*, *b*, and *c* to predict pixel *x*, and subtracts the prediction from *x* to obtain the *prediction error*, denoted by E_{rrval}. This error is then *corrected* by a term that depends on the context (this correction is done to compensate for systematic biases in the prediction), and encoded with a Golomb code. The Golomb coding depends on all four pixels of the context and also on prediction errors that were previously encoded for the same context (this information is stored in arrays *A* and *N).* If near-lossless compression is used, the error is quantized before it is encoded.

In the run mode, the encoder starts at the current pixel *x* and finds the longest run of pixels that are identical to context pixel *a*. The encoder does not extend this run beyond the end of the current image row. Since all the pixels in the run are identical to *a* (and *a* is already known to the decoder) only the length of the run needs be encoded, and this is done with a 32-entry array denoted by *J*. If near-lossless compression is used, the encoder selects a run of pixels that are close to *a* within the tolerance parameter NEAR.

The decoder is not substantially different from the encoder, so JPEG-LS is a nearly symmetric compression method. The compressed stream contains data segments (with the Golomb codes and the encoded run lengths), marker segments (with information needed by the decoder), and markers (some of the reserved markers of JPEG are used). A marker is a byte of all ones followed by a special code, signaling the start of a new segment. If a marker is followed by a byte

whose most significant bit is 0, that byte is the start of a marker segment. Otherwise, that byte starts a data segment.

Advantages of JPEG-LS:

[1] JPEG-LS is capable of lossless compression.

[2] JPEG-LS has very low computational complexity.

JPEG-LS achieve state-of-the-art compression rates at very low computational complexity and memory requirements. These characteristics are what brought to the selection of JPEG-LS, which is based on the LOCO-I algorithm developed at Hewlett-Packard Laboratories, as the new ISO/ITU standard for lossless and near-lossless still image compression.
Ref: "The LOCO-I Lossless Image Compression Algorithm: Principles and Standardization into JPEG-LS", Marcelo J. Weinberger, Gadiel Seroussi, Guillermo Sapiro, IEEE TRANSACTIONS ON IMAGE PROCESSING, VOL. 9, NO. 8, AUGUST 2000.

9. JPEG 2000:

The JPEG 2000 standard for the compression of still images is based on the *Discrete Wavelet Transform* (DWT). This transform decomposes the image using functions called wavelets. The basic idea is to have a more localized (and therefore more precise) analysis of the information (signal, image or 3D objects), which is not possible using cosine functions whose temporal or spatial supports are identical to the data (the same time duration for signals, and the same length of line or column for images).

JPEG-2000 advantages:

JPEG-2000 has the following advantages:

- Better image quality that JPEG at the same file size; or alternatively 25-35% smaller file sizes with the same quality.
- Good image quality at low bit rates (even with compression ratios over 80:1)
- Low complexity option for devices with limited resources.
- Scalable image files -- no decompression needed for reformatting. With JPEG 2000, the image that best matches the target device can be extracted from a single compressed file on a server. Options include:
    1. Image sizes from thumbnail to full size
    2. Grayscale to full 3 channel color
    3. Low quality image to lossless (identical to original image)

- JPEG 2000 is more suitable to web-graphics than baseline JPEG because it supports Alpha-channel (transparency component).
- Region of interest (ROI): one can define some more interesting parts of image, which are coded with more bits than surrounding areas

Following is a list of areas where this new standard is expected to improve on existing methods:

- High compression efficiency. Bitrates of less than 0.25 bpp are expected for highly detailed grayscale images.
- The ability to handle large images, up to 232×232 pixels (the original JPEG can handle images of up to 216×216).
- Progressive image transmission. The proposed standard can decompress an image progressively by SNR, resolution, color component, or region of interest.
- Easy, fast access to various points in the compressed stream.
- The decoder can pan/zoom the image while decompressing only parts of it.
- The decoder can rotate and crop the image while decompressing it.
- Error resilience. Error-correcting codes can be included in the compressed stream, to improve transmission reliability in noisy environments.

9.1 The JPEG 2000 Compression Engine

The JPEG 2000 compression engine (encoder and decoder) is illustrated in block diagram form in Fig. 10.

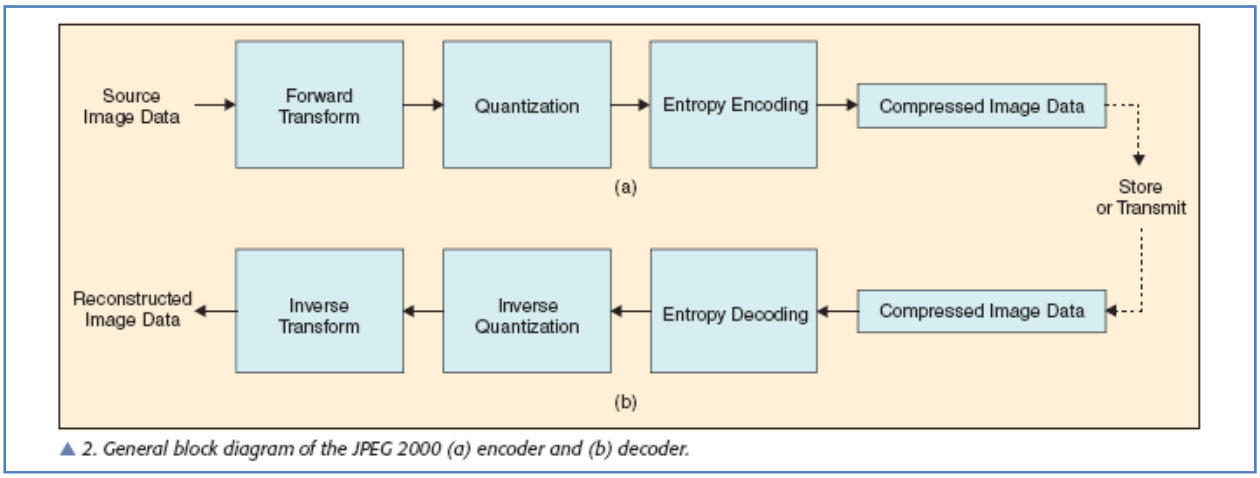


Figure 10: General block diagram of the JPEG 2000 (a) encoder and (b) decoder.

At the encoder, the discrete transform is first applied on the source image data. The transform coefficients are then quantized and entropy coded before forming the output code stream (bit stream). The decoder is the reverse of the encoder. The code stream is first entropy decoded, dequantized, and inverse discrete transformed, thus resulting in the reconstructed image data. Although this general block diagram looks like the one for the conventional JPEG, there are radical differences in all of the processes of each block of the diagram. A quick overview of the whole system is as follows:

- The source image is decomposed into components.
- The image components are (optionally) decomposed into rectangular tiles. The tile-component is the basic unit of the original or reconstructed image.
- A wavelet transform is applied on each tile. The tile is decomposed into different resolution levels.
- The decomposition levels are made up of subbands of coefficients that describe the frequency characteristics of local areas of the tile components, rather than across the entire image component.
- The subbands of coefficients are quantized and collected into rectangular arrays of "code blocks."
- The bit planes of the coefficients in a code block (i.e., the bits of equal significance across the coefficients in a code block) are entropy coded.
- The encoding can be done in such a way that certain regions of interest can be coded at a higher quality than the background.
- Markers are added to the bit stream to allow for error resilience.
- The code stream has a main header at the beginning that describes the original image and the various decomposition and coding styles that are used to locate, extract, decode and reconstruct the image with the desired resolution, fidelity, region of interest or other characteristics.

For the clarity of presentation we have decomposed the whole compression engine into three parts: **the preprocessing, the core processing, and the bit-stream formation part**, although there exist high inter-relation between them. In the preprocessing part the image tiling, the dc-level shifting and the component transformations are included. The core processing part consists of the discrete transform, the quantization and the entropy coding processes. Finally, the concepts of the precincts, code blocks, layers, and packets are included in the bit-stream formation part.

Ref: "The JPEG 2000 Still Image Compression Standard", Athanassios Skodras, Charilaos Christopoulos, and Touradj Ebrahimi, IEEE SIGNAL PROCESSING MAGAZINE, SEPTEMBER 2001, PP. 36-58

10. DPCM:

The DPCM compression method is a member of the family of differential encoding compression methods, which itself is a generalization of the simple concept of relative encoding . It is based on the well-known fact that neighboring pixels in an image (and also adjacent samples in digitized sound) are correlated. Correlated values are generally similar, so their differences are small, resulting in compression.

Differential encoding methods calculate the differences $d_i = a_i - a_{i-1}$ between consecutive data items a_i, and encode the d_i's. The first data item, a_0, is either encoded separately or is written on the compressed stream in raw format. In either case the decoder can decode and generate a_0 in exact form. In principle, any suitable method, lossy or lossless, can be used to encode the differences. In practice, quantization is often used, resulting in lossy compression. The quantity encoded is not the difference d_i but a similar, quantized number that we denote by $\widehat{d_i}$. The difference between d_i and $\widehat{d_i}$ is the *quantization error* q_i. Thus, $\widehat{d_i} = d_i + q_i$.

It turns out that the lossy compression of differences introduces a new problem, namely, the accumulation of errors. This is easy to see when we consider the operation of the decoder. The decoder inputs encoded values of $\widehat{d_i}$, decodes them, and uses them to generate "reconstructed" values $\widehat{a_i}$ (where $\widehat{a_i} = \widehat{a_{i-1}} + \widehat{d_i}$) instead of the original data values *ai*. The decoder starts by reading and decoding a_0. It then inputs $\widehat{d_i} = d_1 + q_1$ and calculates $\widehat{a_1} = a_0 + \widehat{d_i} = a_0 + d_1 + q_1 = a_1 + q_1$. The next step is to input $\widehat{d_2} = d_2 + q_2$ and to calculate $\widehat{a_2} = \widehat{a_1} + \widehat{d_2} = a_1 + q_1 + d_2 + q_2 = a_2 + q_1 + q_2$. The decoded value $\widehat{a_2}$ contains the sum of two quantization errors. In general, the decoded value is,

$$\widehat{a_n} = a_n + \sum_{i=1}^{n} q_i$$

and includes the sum of n quantization errors. Figure 11 summarizes the operations of both encoder and decoder. It shows how the current data item a_i is saved in a storage unit (a delay), to be used for encoding the next item a_{i+1}. The next step in developing a general differential encoding method is to take advantage of the fact that the data items being compressed are correlated.

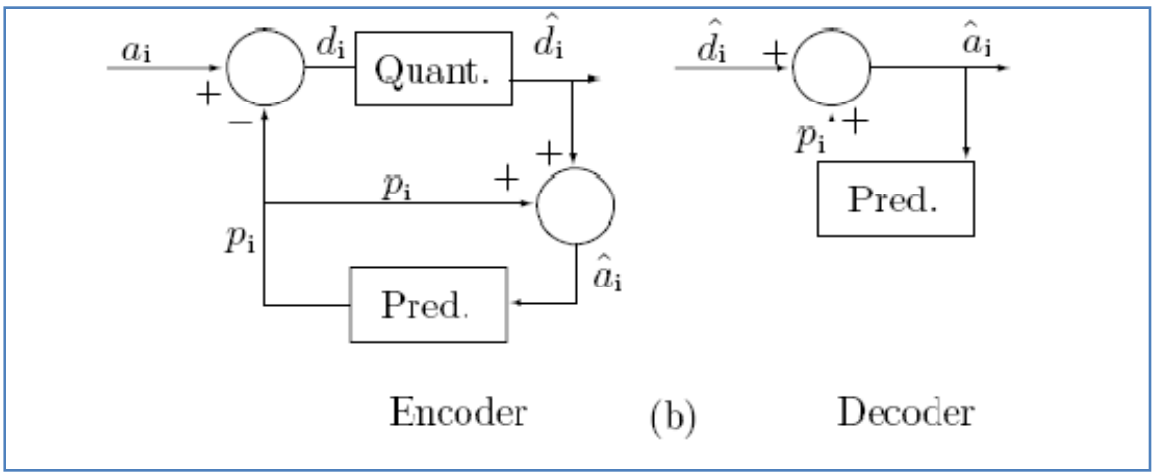


Figure 11: DPCM encoder and decoder

Any method using a predictor is called *differential pulse code modulation*, or DPCM. The simplest predictor is linear. In such a predictor the value of the current pixel a_i is predicted by a weighted sum of N of its previously-seen neighbors (in the case of an image these are the pixels above it or to its left):

$$P_i = \sum_{j=1}^{N} W_j a_{i-j}$$

where w_j are the weights, which still need to be determined. Figure 12 shows a simple example for the case $N = 3$. Let's assume that a pixel X is predicted by its three neighbors A, B, and C according to the simple weighted sum

$$X = 0.35A + 0.3B + 0.35C$$

	B	C
	A	X

The weights used in above equation have been selected more or less arbitrarily and are for illustration purposes only. However, they make sense, because they add up to unity. In order to determine the best weights, we denote by e_i the prediction error for pixel a_i,

$$e_i = a_i - p_i = a_i - \sum_{j=1}^{N} w_j a_{i-j}$$

i=1,2,...,n. and n is the number of pixels to be compressed and we find the set of weights w_j that minimizes the sum

$$E = \sum_{i=1}^{n} e^2 = \sum_{i=1}^{n} \left[a_i - \sum_{j=1}^{N} w_j a_{i-j} \right]^2$$

11. Fractal Image Compression:

Coastlines, mountains and clouds are not easily described by traditional Euclidean geometry. The natural objects may be described and mathematically modeled by Mandelbrot's fractal geometry. This is another reason why image compression using fractal transforms are investigated. The word fractal was first coined by Mandelbrot in 1975.

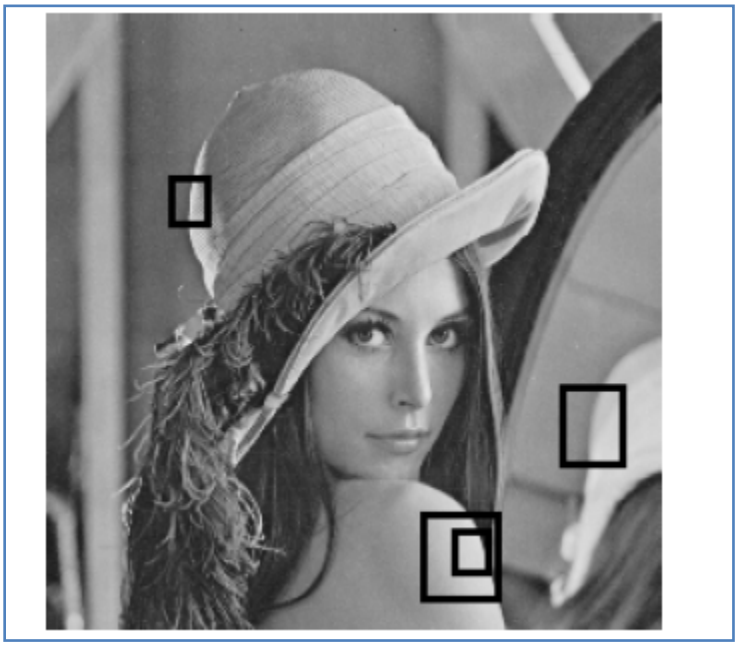
Properties of fractals

1) The defining characteristic of a fractal is that it has a fractional dimension, from which the word fractal is derived.

2) The property of self-similarity or scaling is one of the central concepts of fractal geometry.

11.1 Self-Similarity in Images

A typical image does not contain the type of self-similarity found in fractals. But, it contains a different sort of self-similarity. The figure shows regions of Lenna that are self-similar at different scales. A portion of her shoulder overlaps a smaller region that is almost identical, and a portion of the reflection of the hat in the mirror is similar to a smaller part of her hat.



The difference here is that the entire image is not self-similar, but parts of the image are self-similar with properly transformed parts of itself. Studies suggest that most naturally occurring

images contain this type of self-similarity. It is this restricted redundancy that fractal image compression schemes attempt to eliminate.

What is Fractal Image Compression?

Imagine a special type of photocopying machine that reduces the image to be copied by half and reproduces it three times on the copy (see Figure 1). What happens when we feed the output of this machine back as input? Figure 2 shows several iterations of this process on several input images. We can observe that all the copies seem to converge to the same final image, the one in 2(c). Since the copying machine reduces the input image, any initial image placed on the copying machine will be reduced to a point as we repeatedly run the machine; in fact, it is only the position and the orientation of the copies that determines what the final image looks like.

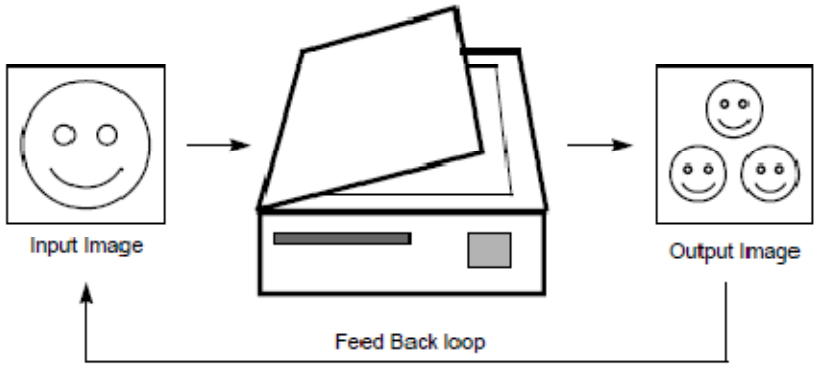


Figure 1: A copy machine that makes three reduced copies of the input image [Y]

The way the input image is transformed determines the final result when running the copy machine in a feedback loop. However we must constrain these transformations, with the limitation that the transformations must be contractive (see contractive box), that is, a given transformation applied to any two points in the input image must bring them closer in the copy. This technical condition is quite logical, since if points in the copy were spread out the final image would have to be of infinite size. Except for this condition the transformation can have any form. In practice, choosing transformations of the form

$$w_i\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_i & b_i \\ c_i & d_i \end{bmatrix}\begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e_i \\ f_i \end{bmatrix} \qquad (1)$$

is sufficient to generate interesting transformations called affine transformations of the plane. Each can skew, stretch, rotate, scale and translate an input image. A common feature of these transformations that run in a loop back mode is that for a given initial image each image is formed

from a transformed (and reduced) copies of itself, and hence it must have detail at every scale. That is, the images are fractals. This method of generating fractals is due to John Hutchinson.

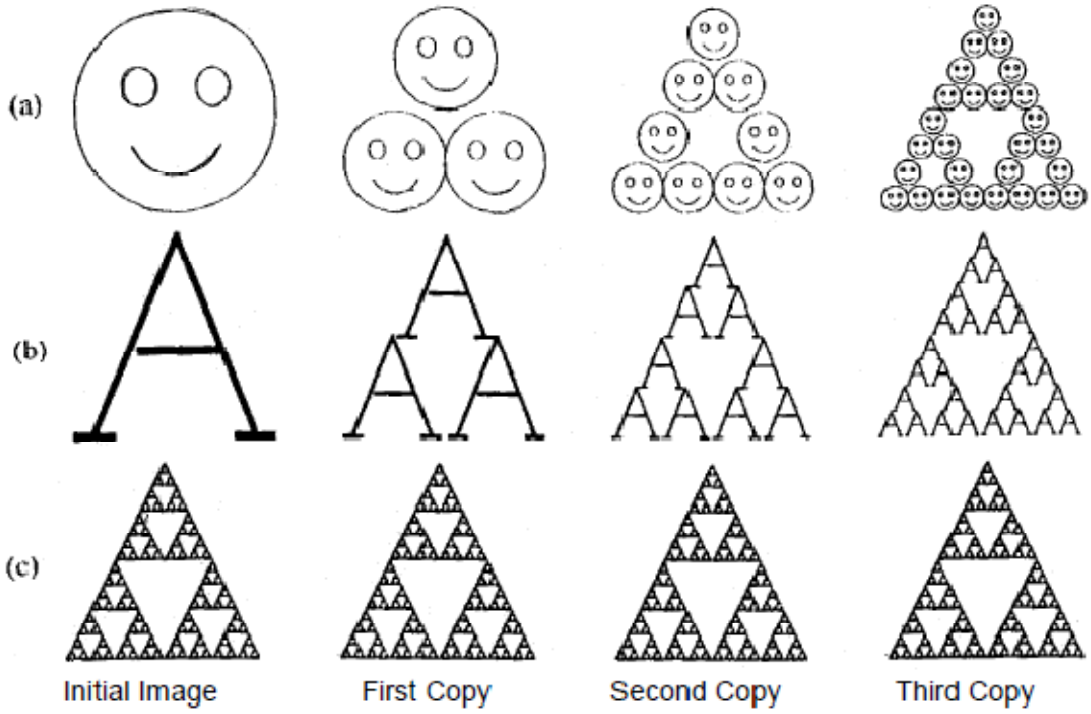


Figure 2: The first three copies generated on the copying machine Figure 1. [Y]

Barnsley suggested that perhaps storing images as collections of transformations could lead to image compression. His argument went as follows: the image in Figure 3 looks complicated yet it is generated from only 4 affine transformations.

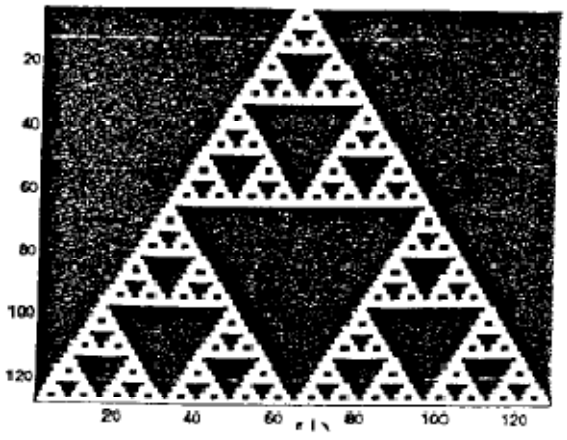


Figure 3: Fractal Fern

12. Video Compression:

Video compression is based on two principles. The first is the **spatial redundancy** that exists in each frame. The second is 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. In the video compression literature, 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*).

With this in mind it is easy to imagine a situation where the encoder encodes frame 2 based on both frames 1 and 3, and writes the frames on the compressed stream in the order 1, 3, 2. The decoder reads them in this order, decodes frames 1 and 3 in parallel, outputs frame 1, then decodes frame 2 based on frames 1 and 3. Naturally, the frames should be clearly tagged (or time stamped). A frame that is encoded based on both past and future frames is labeled ***B* (for *bidirectional*).**

Predicting a frame based on its successor makes sense in cases where the movement of an object in the picture gradually uncovers a background area. Such an area may be only partly known in the current frame but may be better known in the next frame. Thus, the next frame is a natural candidate for predicting this area in the current frame.

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 12a shows a sequence of such frames in the order in which they are generated by the encoder (and

input by the decoder). Figure 12b 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.

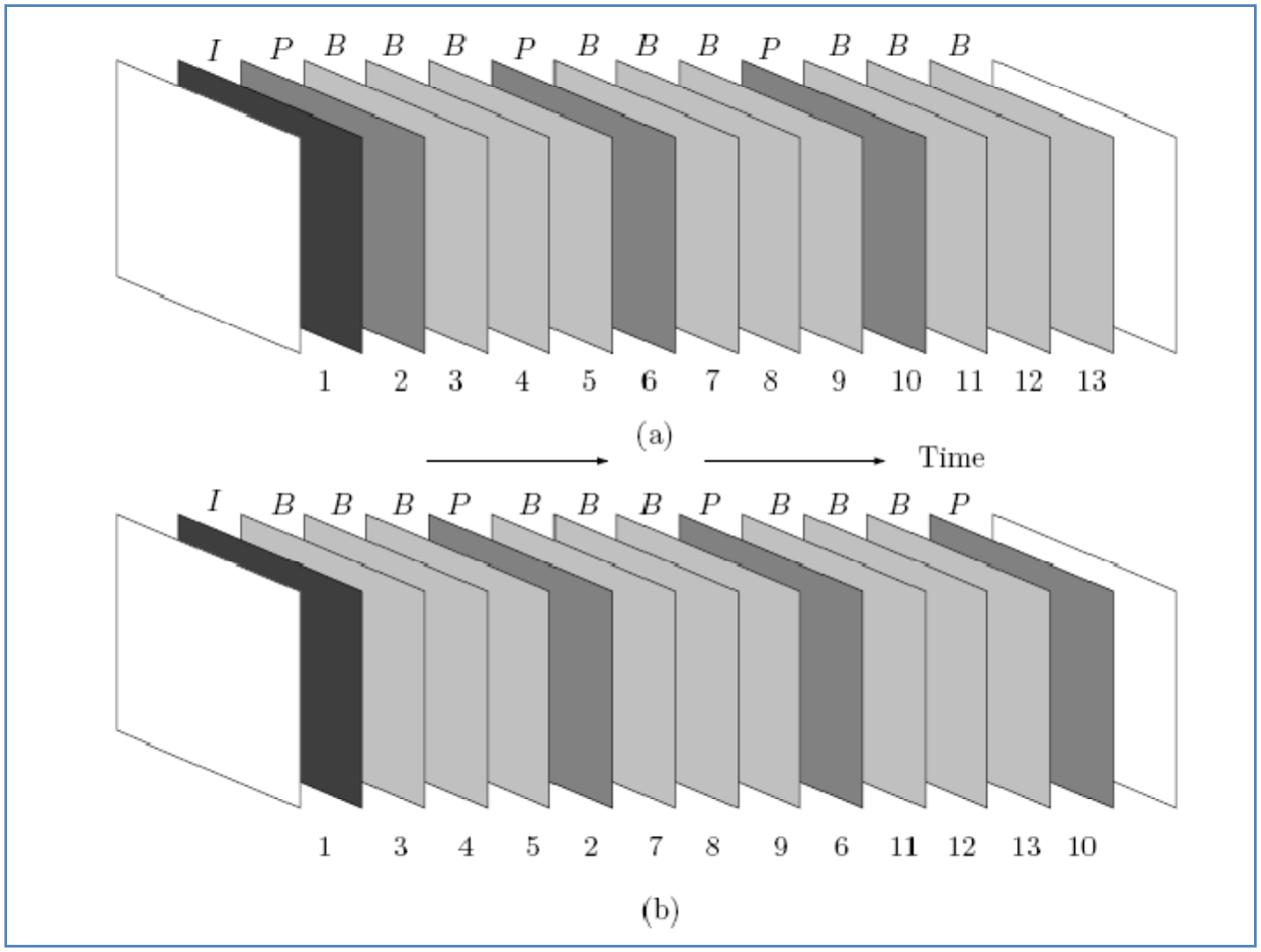


Figure 12: (a) Coding Order. (b) Display Order.

We start with a few intuitive video compression methods.

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.

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, 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.

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. The advantage is that the block coordinates are small numbers (smaller than a pixel's coordinates), and these coordinates have to be written just once for the entire block. On the downside, the values of all the pixels in the block, even those that haven't changed, have to be written on the output. However, since these values are expressed as differences, they are small numbers. Consequently, this method is sensitive to the block size.

12.1 Motion Compensation

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 equalize 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 13 a,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).

Motion compensation is effective if ***objects are just translated, not scaled or rotated***. Drastic changes in illumination from frame to frame also reduce the effectiveness of this method. In general, ***motion compensation is lossy***. The following paragraphs discuss the main aspects of motion compensation in detail. Figure 14 shows the flow of information through the motion compensation process.

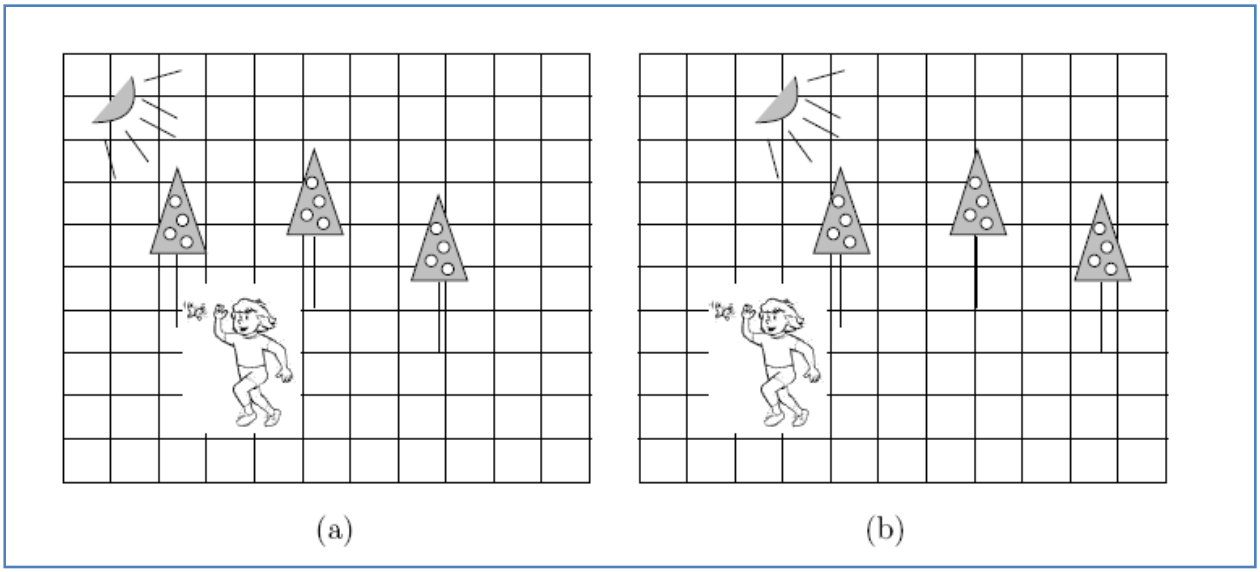


Figure 13: Motion Compensation.

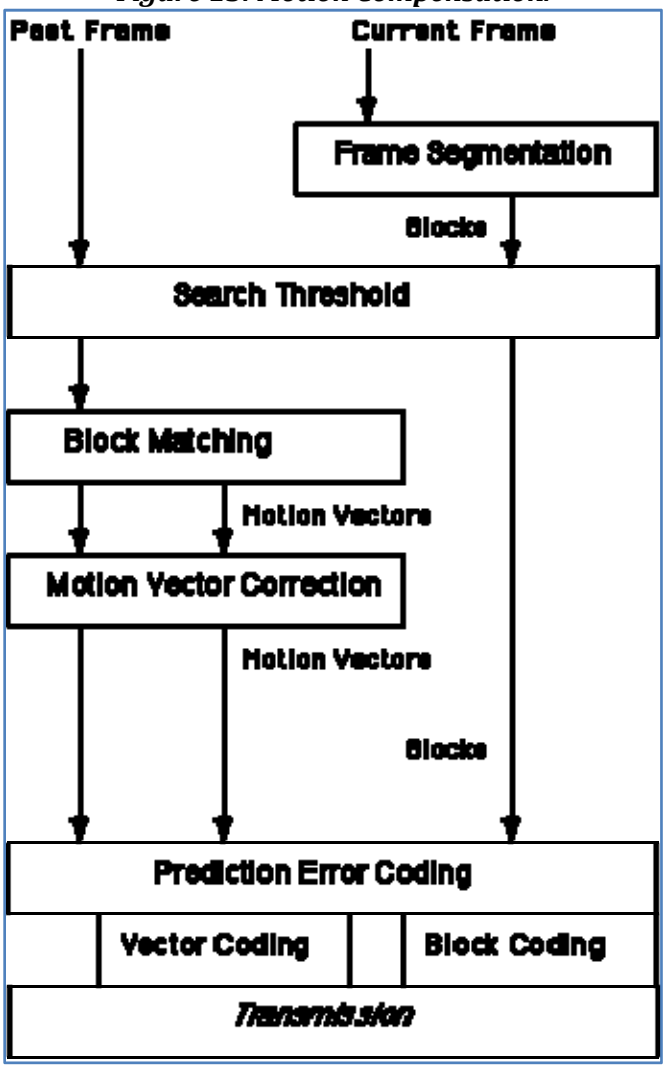


Figure 14: Flow of information in motion compensation process

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 15) and will consist of $(2dx+1)(2dy+1)$ distinct, overlapping $b \times b$ squares. The number of candidate blocks in this area is therefore proportional to $dx \cdot dy$.

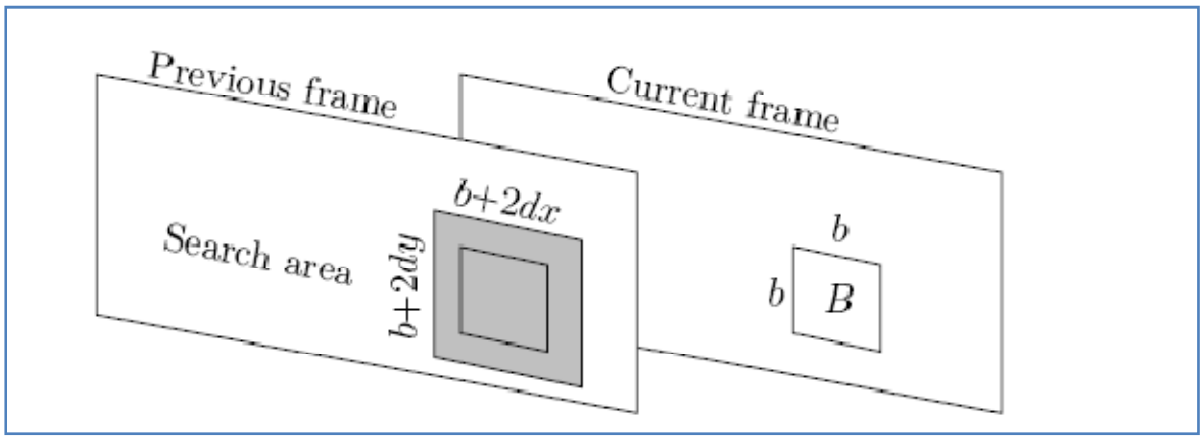


Figure 15: Search Area.

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*:

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

This involves b^2 subtractions and absolute value operations, b^2 additions, and one division. This measure is calculated for each of the $(2dx+1)(2dy+1)$ distinct, overlapping $b \times b$ candidate blocks, and the smallest distortion (say, for block C_k) is examined. If it is smaller than the search threshold, then C_k is selected as the match for B. Otherwise, there is no match for B, and B has to be encoded without motion compensation.

Suboptimal Search Methods: These methods search some, instead of all, the candidate blocks in the $(b+2dx)(b+2dy)$ area. They speed up the search for a matching block, at the expense of compression efficiency.

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. Regardless of how the matching was determined, the motion vector may be wrong because of noise, local minima in the frame, or because the matching algorithm is not perfect. It is possible to apply smoothing techniques to the motion vectors after they have been calculated, in an attempt to improve the matching. Spatial correlations in the image suggest that the motion vectors should also be correlated. If certain vectors are found to violate this, they can be corrected.

This step is costly and may even backfire. A video presentation may involve slow, smooth motion of most objects, but also swift, jerky motion of some small objects. Correcting motion vectors may interfere with the motion vectors of such objects and cause distortions in the compressed frames.

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 crucial; it must also be lossless. Two properties of motion vectors help in encoding them: (1) They are ***correlated*** and (2) their ***distribution is nonuniform***. As we scan the frame block by block, adjacent blocks normally have motion vectors that don't differ by much; they are correlated. The vectors also don't point in all directions. There are often one or two preferred directions in which all or most motion vectors point; the vectors are nonuniformly distributed.

No single method has proved ideal for encoding the motion vectors. Arithmetic coding, adaptive Huffman coding, and various prefix codes have been tried, and all seem to perform well. Here are two different methods that may perform better:

1. Predict a motion vector based on its predecessors in the same row and its predecessors in the same column of the current frame. Calculate the difference between the prediction and the actual

vector, and Huffman encode it. This algorithm is important. It is used in MPEG and other compression methods.

2. Group the motion vectors in blocks. If all the vectors in a block are identical, the block is encoded by encoding this vector. Other blocks are encoded as in 1 above. Each encoded block starts with a code identifying its type.

Coding the Prediction Error: Motion compensation is lossy, since a block *B* is normally matched to a somewhat different block *C*. Compression can be improved by coding the difference between the current uncompressed and compressed frames on a block by block basis and only for blocks that differ much. This is usually done by transform coding. The difference is written on the output, following each frame, and is used by the decoder to improve the frame after it has been decoded.

14 MPEG

The name MPEG is an acronym for 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-1*** is intended for intermediate data rates, on the order of 1***.5 Mbit/sec MPEG-2*** is intended for high data rates of at least ***10 Mbit/ sec***. ***MPEG-3*** was intended for ***HDTV c***ompression but was found to be redundant and was merged with MPEG-2. ***MPEG-4*** is intended for very low data rates of less than ***64 Kbit/sec.*** A third international body, the ITU-T, has been involved in the design of both MPEG-2 and MPEG-4.

The formal name of MPEG-1 is the international standard for moving picture video compression, IS11172-2. Like other standards developed by the ITU and ISO, the document describing MPEG-1 has ***normative and informative*** sections. A normative section is part of the standard specification. It is intended for implementers, is written in a precise language, and should be strictly adhered to when implementing the standard on actual computer platforms. An informative section, on the other hand, illustrates concepts discussed elsewhere, explains the reasons that led to certain choices and decisions, and contains background material. An example of a normative section is the various tables of variable codes used in MPEG. An example of an informative section is the algorithm used by MPEG to estimate motion and match blocks. MPEG does not require any particular algorithm, and an MPEG encoder can use any method to match blocks.

The importance of a widely accepted standard for video compression is apparent from the fact that many manufacturers (of computer games, DVD movies, digital television, and digital recorders, among others) implemented MPEG-1 and started using it even before it was finally

approved by the MPEG committee. This also was one reason why MPEG-1 had to be frozen at an early stage and MPEG-2 had to be developed to accommodate video applications with high data rates.

To understand the meaning of the words ***"intermediate data rate***" we consider a typical example of video with a resolution of 360×288, a depth of 24 bits per pixel, and a refresh rate of 24 frames per second. The image part of this video requires 360×288×24×24 = 59,719,680 bits/s. For the audio part, we assume two sound tracks (stereo sound), each sampled at 44 kHz with 16-bit samples. The data rate is 2× 44,000×16 = 1,408,000 bits/s. The total is about 61.1 Mbit/s and this is supposed to be compressed by MPEG-1 to an intermediate data rate of about 1.5 Mbit/s (the size of the sound track alone), a compression factor of more than 40! Another aspect is the decoding speed. An MPEG-compressed movie may end up being stored on a CD-ROM or DVD and has to be decoded and played in real time.

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 eye is sensitive to small spatial variations of luminance, but is less sensitive to similar changes in chrominance. As a result, MPEG-1 samples the chrominance components at half the resolution of the luminance component. The term ***intra*** is used, but ***inter*** and ***nonintra*** are used interchangeably.

The input to an MPEG encoder is called the ***source data***, and the output of an MPEG decoder is the ***reconstructed data***. The source data is organized in packs (Figure 16b), where each pack starts with a start code (32 bits) followed by a header, ends with a 32-bit end code, and contains a number of packets in between. A packet contains compressed data, either audio or video. The size of a packet is determined by the MPEG encoder according to the requirements of the storage or transmission medium, which is why a packet is not necessarily a complete video picture. It can be any part of a video picture or any part of the audio.

The MPEG decoder has three main parts, called *layers*, to decode the ***audio, the video, and the system data***. The system layer reads and interprets the various codes and headers in the source data, and routes the packets to either the audio or the video layers (Figure 16a) to be buffered and later decoded. Each of these two layers consists of several decoders that work simultaneously.

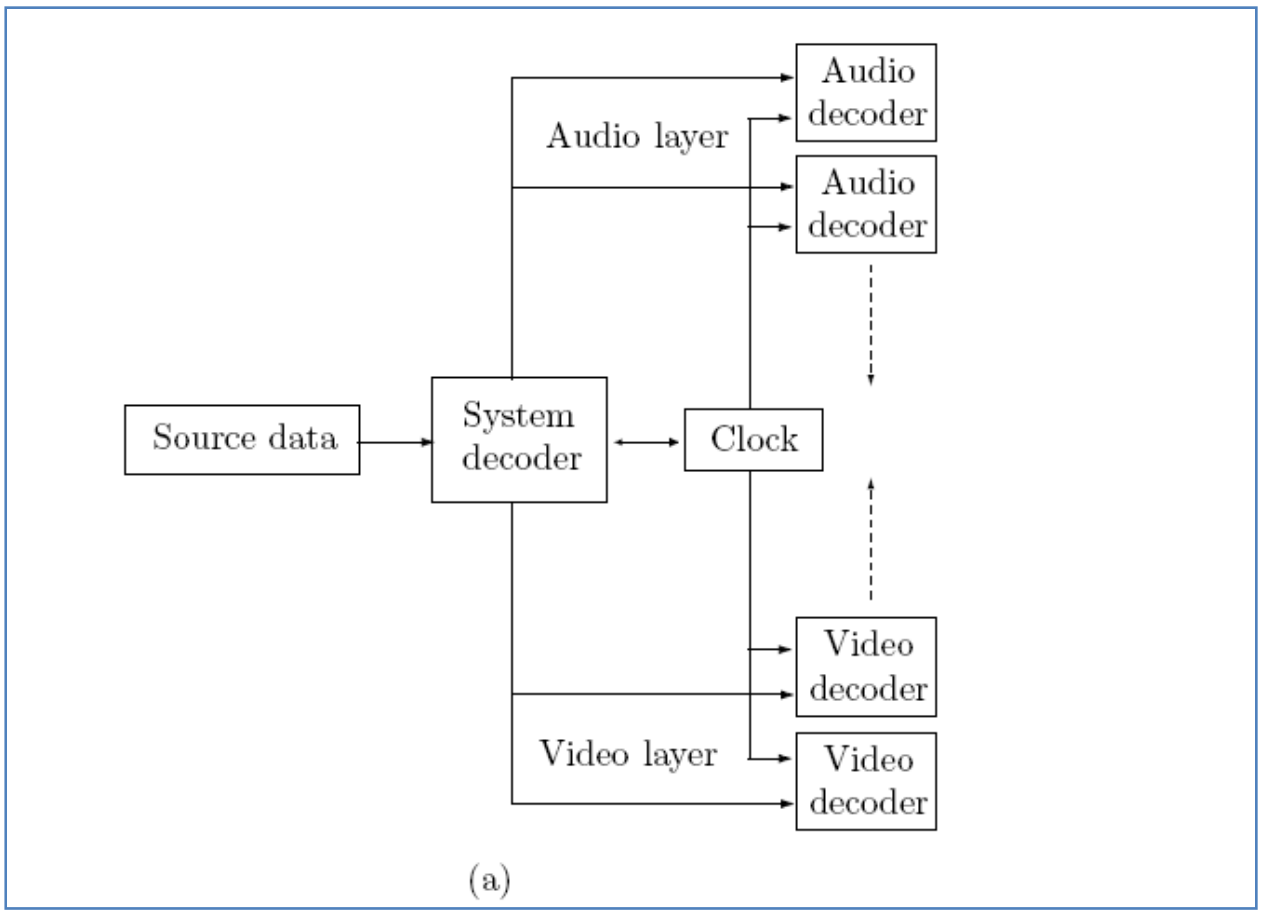


Figure 16: (a) MPEG Decoder Organization. (b) Source Format.

14.1 MPEG 1

MPEG uses *I*, *P*, and *B* pictures. 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. A region may be decoded from some preceding pictures, from some following pictures, from both types, or from none. Similarly, a region in a *P* picture may use several preceding pictures for its decoding, or use none at all, in which case it is decoded using MPEG's intra methods.

The basic building block of an MPEG picture is the ***macroblock*** (Figure 17a). 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, 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.

A picture in MPEG is organized in slices, where each slice is a contiguous 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 contiguous macroblocks to have the same grayscale. Figure 17b 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.

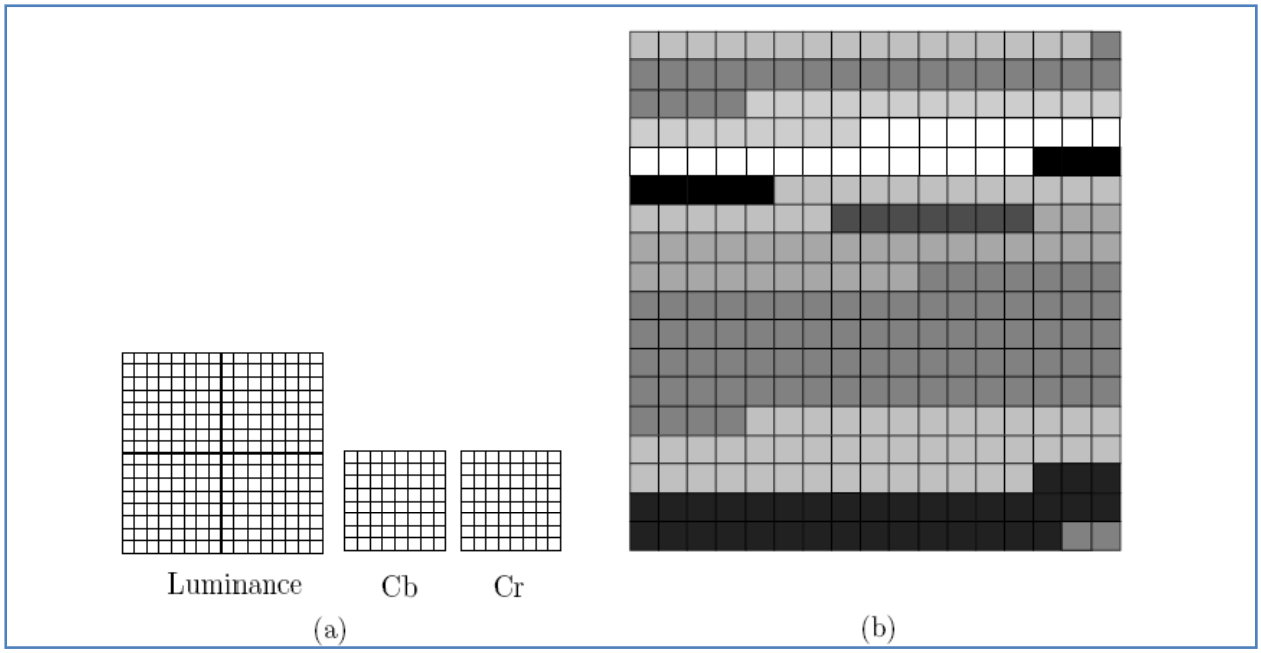


Figure 17: (a) A Macroblock. (b) A Possible Slice Structure.

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.

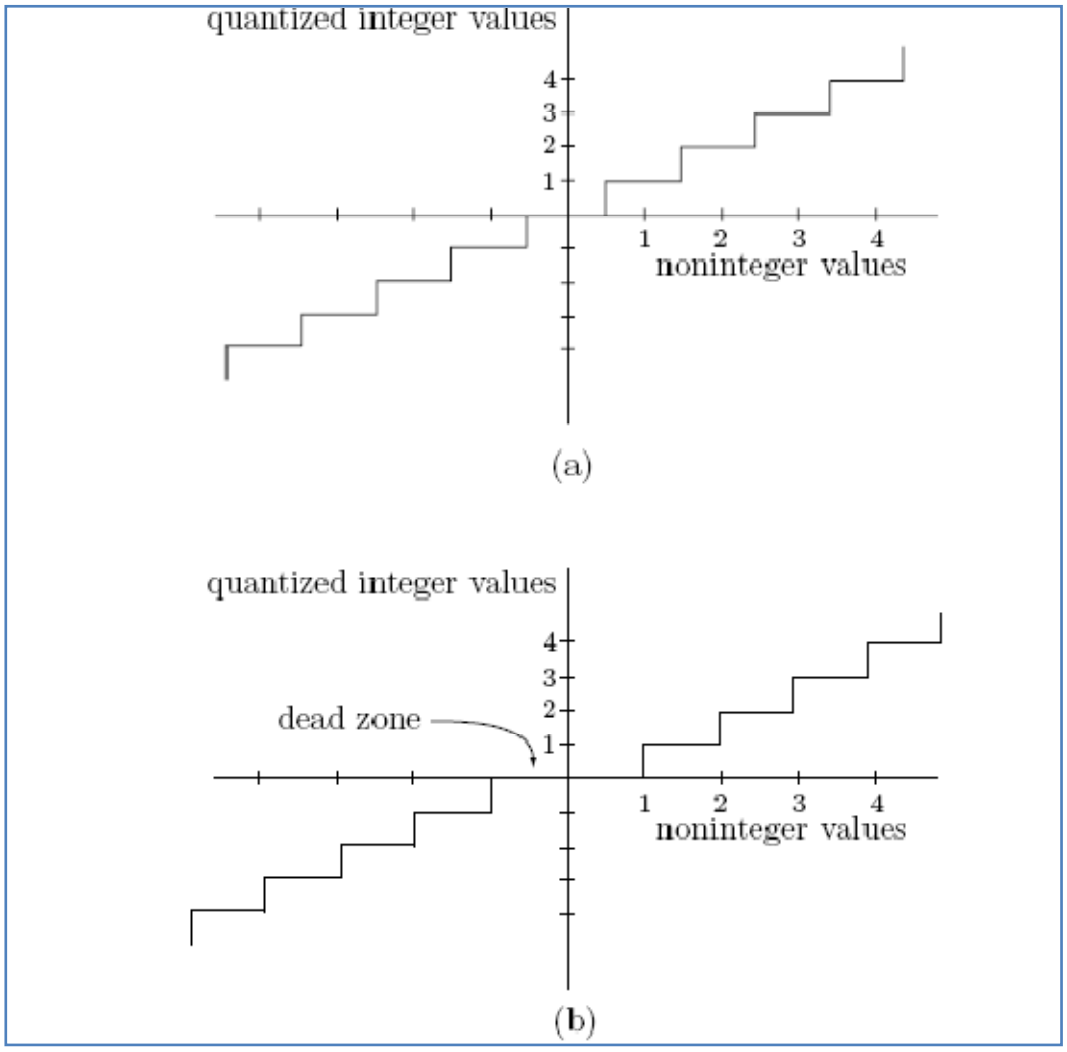


Figure 18: Rounding of Quantized DCT Coefficients.(a) For Intra Coding. (b) For Nonintra Coding.

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. The two summations of the two-dimensional

DCT, can at most multiply a sample by $64 = 2^6$ and may therefore result in an 8+6 = 14-bit integer. In those summations, a sample is multiplied by cosine functions, which may result in a negative number. The result of the double sum is therefore a 15-bit signed integer. This integer is then multiplied by the factor $C_iC_j/4$ which is at least 1/8, thereby reducing the result to a 12-bit signed integer. This 12-bit integer is then quantized by dividing it by a quantization coefficient (QC) taken from a quantization table. The result is, in general, a noninteger and has to be rounded. It is in quantization and rounding that information is irretrievably lost. MPEG specifies default quantization tables, but custom tables can also be used. 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 18a,b shows the results graphically. Notice the wide interval around zero in nonintra coding. This is the so-called *dead zone*.

The quantization and rounding steps are complex and involve more operations than just dividing a DCT coefficient by a quantization coefficient. They depend on a scale factor called ***quantizer_scale***, an MPEG parameter that is an integer in the interval [1, 31]. The results of the quantization, and hence the compression performance, are sensitive to the value of quantizer_scale. The encoder can change this value from time to time and has to insert a special code in the compressed stream to indicate this. The precise way to compute the IDCT is not specified by MPEG. This can lead to distortions in cases where a picture is encoded by one implementation and decoded by another, where the IDCT is done differently. In a chain of inter pictures, where each picture is decoded by means of its neighbors, this can lead to accumulation of errors, a phenomenon known as *IDCT mismatch*. This is why MPEG requires periodic intra coding of every part of the picture. This *forced updating* has to be done at least once for every 132 *P* pictures in the sequence. In practice, forced updating is rare, since *I* pictures are fairly common, occurring every 10 to 15 pictures.

The quantized numbers ***QDCT are Huffman coded***, using the nonadaptive Huffman method and Huffman code tables that were computed by gathering statistics from many training image sequences. The particular code table being used depends on the type of picture being encoded. To avoid the zero probability problem, all the entries in the code tables were initialized to 1 before any statistics were collected. Decorrelating the original pels by computing the DCT (or, in the case of inter coding, by calculating pel differences) is part of the statistical model of MPEG. The other part is the creation of a symbol set that takes advantage of the properties of Huffman coding.

The Huffman method becomes inefficient when the data contains symbols with large probabilities. If the probability of a symbol is 0.5, it should ideally be assigned a 1-bit code. If the

probability is higher, the symbol should be assigned a shorter code, but the Huffman codes are integers and hence cannot be shorter than one bit. To avoid symbols with high probability, MPEG uses an alphabet where several old symbols (i.e., several pel differences or quantized DCT coefficients) are combined to form one new symbol. An example is run lengths of zeros. After quantizing the 64 DCT coefficients of a block, many of the resulting numbers are zeros. The probability of a zero is therefore high and can easily exceed 0.5. The solution is to deal with runs of consecutive zeros. Each run becomes a new symbol and is assigned a Huffman code. This method creates a large number of new symbols, and many Huffman codes are needed as a result. Compression efficiency, however, is improved.

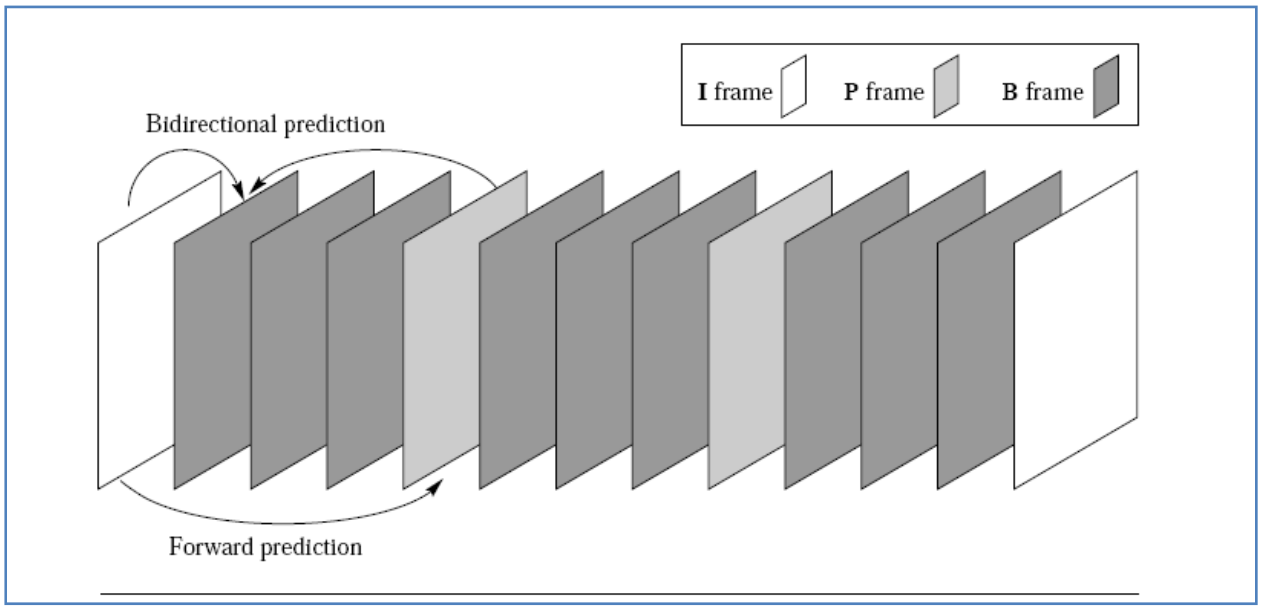


Figure 19: A possible arrangement for a group of pictures.

The different frames are organized together in a *group of pictures* (GOP). A GOP is the smallest random access unit in the video sequence. The GOP structure is set up as a trade-off between the high compression efficiency of motion-compensated coding and the fast picture acquisition capability of periodic intra-only processing. As might be expected, a GOP has to contain at least one **I** frame. Furthermore, the first **I** frame in a GOP is either the first frame of the GOP, or is preceded by **B** frames that use motion-compensated prediction only from this **I** frame. A possible GOP is shown in Figure 19.

14.2 MPEG – 2:

While MPEG-1 was specifically proposed for digital storage media, the idea behind MPEG-2

was to provide a generic, application-independent standard. To this end, MPEG-2 takes a "tool kit" approach, providing a number of subsets, each containing different options from the set of all possible options contained in the standard. For a particular application, the user can select from a set of ***profiles and levels***. The profiles define the algorithms to be used, while the levels define the constraints on the parameters. There are five profiles:

1. ***simple***
2. ***main***
3. ***snr-scalable*** **(signal-to-noise ratio)**
4. ***spatially scalable***
5. ***high*.**

There is an ordering of the profiles; each higher profile is capable of decoding video encoded using all profiles up to and including that profile. For example, a decoder designed for profile *snr-scalable* could decode video that was encoded using profiles *simple, main*, and *snr-scalable*. The *simple* profile eschews the use of **B** frames. Recall that the **B** frames require the most computation to generate (forward and backward prediction), require memory to store the coded frames needed for prediction, and increase the coding delay because of the need to wait for "future" frames for both generation and reconstruction.

Therefore, removal of the **B** frames makes the requirements simpler. The *main* profile is very much the algorithm we have discussed in the previous section. The *snr-scalable, spatially scalable*, and *high* profiles may use more than one bitstream to encode the video. The base bitstream is a lower-rate encoding of the video sequence. This bitstream could be decoded by itself to provide a reconstruction of the video sequence. The other bitstream is used to enhance the quality of the reconstruction. This layered approach is useful when transmitting video over a network, where some connections may only permit a lower rate. The base bitstream can be provided to these connections while providing the base and enhancement layers for a higher-quality reproduction over the links that can accommodate the higher bit rate.

The levels are ***low, main, high 1440*, and *high***. The *low* level corresponds to a frame size of 352×240, the *main* level corresponds to a frame size of 720×480, the *high 1440* level corresponds to a frame size of 1440×1152, and the *high* level corresponds to a frame size of 1920×1080. All levels are defined for a frame rate of 30 frames per second. There are many possible combinations of profiles and levels, not all of which are allowed in the MPEG-2 standard. A particular profile level combination is denoted by *XX@YY* where *XX* is the two-letter abbreviation for the profile and *YY* is the two-letter abbreviation for the level.

Because MPEG-2 has been designed to handle interlaced video, there are field, based alternatives to the **I**, **P** and **B** frames. The **P** and **B** frames can be replaced by two **P** fields or two **B** fields. The **I** frame can be replaced by two **I** fields or an **I** field and a **P** field where the **P** field is obtained by predicting the bottom field by the top field. Because an 8×8 field block actually covers twice the spatial distance in the vertical direction as an 8 frame block, the zigzag scanning is adjusted to adapt to this imbalance.

The most important addition from the point of view of compression in MPEG-2 is the addition of several new motion-compensated prediction modes: ***the field prediction*** and the ***dual prime prediction modes***. MPEG-1 did not allow interlaced video. Therefore, there was no need for motion compensation algorithms based on fields. In the **P** frames, field predictions are obtained using one of the two most recently decoded fields. When the first field in a frame is being encoded, the prediction is based on the two fields from the previous frame. However, when the second field is being encoded, the prediction is based on the second field from the previous frame and the first field from the current frame. Information about which field is to be used for prediction is transmitted to the receiver. The field predictions are performed in a manner analogous to the motion-compensated prediction described earlier.

In addition to the regular frame and field prediction, MPEG-2 also contains two additional modes of prediction. One is the ***16×8 motion compensation***. In this mode, two predictions are generated for each macroblock, one for the top half and one for the bottom half. The other is called the ***dual prime motion compensation***. In this technique, two predictions are formed for each field from the two recent fields. These predictions are averaged to obtain the final prediction.

14.3 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. 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. In July 1994, this goal was significantly altered in response to developments in audiovisual technologies. 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.

At present (mid-2006), the MPEG-4 standard is designated the ISO/IEC 14496 standard, and its formal description, which is available from [ISO 03], consists of 17 parts plus new

amendments. 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 triumph 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 tremendous commercial success.

The MPEG-4 project deliver reasonable video data in only a few thousand bits per second. Such compression is important for video telephones, video conferences or for receiving video in a small, handheld device, especially in a mobile environment, such as a moving car. 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. 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.

Another 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. During its important July 1994 meeting, the MPEG-4 committee decided to revise its original goal and also started thinking of future developments in the audiovisual field and of features that should be included in MPEG-4 to meet them. They came up with eight points that they considered important functionalities for MPEG-4.

1. **Content-based multimedia access tools**. The MPEG-4 standard should provide tools for accessing and organizing audiovisual data. Such tools may include indexing, linking, querying, browsing, delivering files, and deleting them.

2. **Content-based manipulation and bitstream editing**. A syntax and a coding scheme should be part of MPEG-4. The idea is to enable users to manipulate and edit compressed files (bitstreams)

without fully decompressing them. A user should be able to select an object and modify it in the compressed file without decompressing the entire file.

3. **Hybrid natural and synthetic data coding**. A natural scene is normally produced by a video camera. A synthetic scene consists of text and graphics. MPEG-4 recognizes the need for tools to compress natural and synthetic scenes and mix them interactively.

4. **Improved temporal random access**. Users may often want to access part of the compressed file, so the MPEG-4 standard should include tags to make it easy to quickly reach any point in the file. This may be important when the file is stored in a central location and the user is trying to manipulate it remotely, over a slow communications channel.

5. **Improved coding efficiency**. This feature simply means improved compression. Imagine a case where audiovisual data has to be transmitted over a low-bandwidth channel (such as a telephone line) and stored in a low-capacity device such as a smartcard. This is possible only if the data is well compressed, and high compression rates (or equivalently, low bitrates) normally involve a trade-off in the form of smaller image size, reduced resolution (pixels per inch), and lower quality.

6. **Coding of multiple concurrent data streams**. It seems that future audiovisual applications will allow the user not just to watch and listen but also to interact with the image. As a result, the MPEG-4 compressed stream can include several views of the same scene, enabling the user to select any of them to watch and to change views at will. The point is that the different views may be similar, so any redundancy should be eliminated by means of efficient compression that takes into account identical patterns in the various views. The same is true for the audio part (the soundtracks).

7. **Robustness in error-prone environments**. MPEG-4 must provide error orrecting codes for cases where audiovisual data is transmitted through a noisy channel. This is especially important for low-bitrate streams, where even the smallest error may be noticeable and may propagate and affect large parts of the audiovisual presentation.

8. **Content-based scalability**. The compressed stream may include audiovisual data in fine resolution and high quality, but any MPEG-4 decoder should be able to decode it at low resolution and low quality. This feature is useful in cases where the data is decoded and displayed on a small, low-resolution screen, or in cases where the user is in a hurry and prefers to see a rough image rather than wait for a full decoding. An MPEG-4 author faced with an application has to identify the requirements of the application and select the right tools. It is now clear that compression is a central requirement in MPEG-4, but not the only requirement, as it was for MPEG-1 and MPEG-2.

In general, audiovisual content goes through three stages: production, delivery, and consumption. Each of these stages is summarized below for the traditional approach and for the MPEG-4 approach.

Production. Traditionally, audiovisual data consists of two-dimensional scenes; it is produced with a camera and microphones and consists of natural objects. All the mixing of objects (composition of the image) is done during production. The MPEG-4 approach is to allow for both two-dimensional and three-dimensional objects and for natural and synthetic scenes. The composition of objects is explicitly specified by the producers during production by means of a special language. This allows later editing.

Delivery. The traditional approach is to transmit audiovisual data on a few networks, such as local-area networks and satellite transmissions. The MPEG-4 approach is to let practically any data network carry audiovisual data. Protocols exist to transmit audiovisual data over any type of network.

Consumption. Traditionally, a viewer can only watch video and listen to the accompanying audio. Everything is precomposed. The MPEG-4 approach is to allow the user as much freedom of composition as possible. The user should be able to interact with the audiovisual data, watch only parts of it, interactively modify the size, quality, and resolution of the parts being watched, and be as active in the consumption stage as possible.

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. Our short description of MPEG-4 concludes with a list of the main tools specified by the MPEG-4 standard.