

Guojun Lu "Fractal-Based Image and Video Compression"
The Transform and Data Compression Handbook
Ed. K. R. Rao et al.
Boca Raton, CRC Press LLC, 2001

Chapter 7

Fractal-Based Image and Video Compression

Guojun Lu

Monash University

7.1 Introduction

This chapter describes image and video compression techniques based on affine transforms or iterated function systems (IFS) [2, 4]. These techniques are fundamentally different from techniques based on other transforms, such as discrete cosine transform (DCT). In DCT-based techniques, image data are transformed from the spatial domain to the frequency domain where quantization and entropy coding are carried out to achieve data compression. In IFS-based techniques, we exploit the fact that part of an image is similar to another part of the image after certain affine transforms called IFS. Data compression is achieved by determining these transforms and storing parameters representing them.

The chapter is organized as follows. Section 7.2, describes the basic properties of fractals and the basic principle of fractal-based image compression. Section 7.3 describes concepts of contractive affine transforms, iterated function systems, and the fractal image generation process. Section 7.4 discusses how to find IFS directly from images. Section 7.5 describes how to compress images based on a library of known IFS.

Techniques introduced in Sections 7.4 and 7.5 can achieve very high compression ratios. However, it is difficult to find IFS in a natural image. To solve this problem, image coding methods using partitioned IFS (PIFS) have been developed. The difference between IFS and PIFS is that IFS consists of affine transforms that map an entire image to parts of the image while PIFS consists of transforms that map parts of an image to other parts of the image. In these methods, an image to be coded is divided into nonoverlapping blocks. For each block, a transformation is found that converts part of the image into a block similar to this block. The combination of all transforms found for each block is called PIFS. PIFS corresponds to a unique image. The compression performance depends on the contents of the image and how

the image is partitioned. There are many types of partitions: fixed size partitioning, quadtree partitioning, horizontal-vertical (HV) partitioning, and triangular partitioning. We discuss PIFS-based coding using fixed size partition and quadtree partition in Sections 7.6 and 7.7, respectively.

One property of fractals is scalability: fractals have fine detail in any scale. Section 7.8 explores how we can use this property to reduce the required compression time and improve performance.

It is well known that there is much redundancy among neighboring frames of a video sequence. It is very likely that part of a current frame is the transformation of a part in the previous frame. Based on this observation, we can apply quadtree partition techniques to video sequence coding, which is discussed in Section 7.9.

There are other image compression techniques based on various properties of fractals. Section 7.10 briefly describes two techniques based on fractal dimension and fractal approximation, respectively. Section 7.11 concludes the chapter.

7.2 Basic Properties of Fractals and Image Compression

The word fractal was coined by Mandelbrot from the Latin word *fractus*, meaning broken, to describe objects that were too irregular to fit into traditional geometrical settings [1]. Several definitions have been proposed. Mandelbrot defined a fractal to be a set with Hausdorff dimension strictly greater than its Euclidean dimension, i.e., a set for which the only consistent description of its metric properties requires a “dimension” value larger than our standard, intuitive definition of the set’s “dimension.” A fractal has a fractional dimension; thus some people say we get the word *fractal* from *fractional dimension*. According to Barnsley [2], a fractal is a geometric form whose irregular details recur at different scales and angles which can be described by affine or fractal transforms. Various other definitions have been proposed, but they are not complete in that they exclude a number of sets that clearly ought to be regarded as fractals.

Falconer [3] proposed that it is best to regard a fractal as a set that has properties such as those listed below, rather than to look for a precise definition which will almost certainly exclude some interesting cases. Typical properties of a fractal are

- It has a fine structure, i.e., details on arbitrarily small scales.
- It is too irregular to be described in traditional geometrical language, both locally and globally.
- It usually has some form of self-similarity, perhaps approximate or statistical.
- Its fractal dimension (Hausdorff dimension) is usually higher than its Euclidean dimension.

- In most cases of interest, a fractal is defined in a very simple way, perhaps recursively.

Fig. 7.1 shows the construction of one of the common fractals called the Sierpinski gasket or triangle [3]. It is constructed by repeatedly replacing an equilateral triangle with three equilateral triangles of half the height. This construction process can be represented by a set of fractal transforms (see Section 7.4). Fractal transforms have been used to generate complicated fractal images. Fractal image compression is the inverse of fractal image generation; instead of generating an image from a given formula, fractal image compression searches for sets of fractals in a digitized image which describe and represent the entire image. Once the appropriate sets of fractals are determined, they are represented by very compact fractal transform formulae. These formulae are the rules for reproducing the various sets of fractals which, in turn, regenerate the entire image. Because fractal transform formulae require a very small amount of data to be represented and stored, fractal compression can result in very high compression ratios.

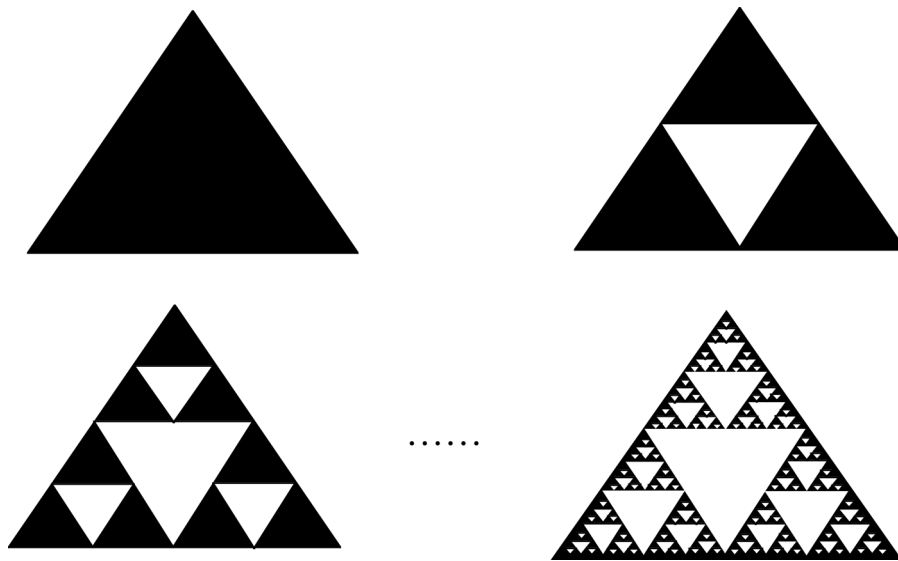


FIGURE 7.1
Construction of the Sierpinski triangle or gasket.

7.3 Contractive Affine Transforms, Iterated Function Systems, and Image Generation

Since fractal image compression based on iterated function systems is the inverse of image generation, in this section, we see how fractals are generated from an iterated function system. We introduce contractive affine transforms and iterated function systems.

Contractive Affine Transforms

A two-dimensional affine transform W maps points in the Euclidean plane into new points in the Euclidean plane, according to the formula

$$W \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix}. \quad (7.1)$$

It consists of a linear transform, represented by the 2×2 matrix with entries a , b , c , and d , followed by a shift or translation, represented by the vector with entries e and f . An example for an affine transform is

$$W \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}.$$

Applying the above transform to all points in the triangle F of [Fig. 7.2\(a\)](#) results in a smaller triangle in [Fig. 7.2\(b\)](#). Notice that the cross and the star in the transformed triangle $W(F)$ are closer than in F . We say that the transform is contractive if it always moves pairs of points closer together. Formally, a transform W is said to be contractive if for any two points P_1 and P_2 , the distance

$$d(W(P_1), W(P_2)) < s d(P_1, P_2) \quad (7.2)$$

where $s \in (0, 1)$, and is called the contractive factor.

Contractive affine transforms have the property that when they are repeatedly applied, they converge to a point which remains fixed upon further iterations. For example, applying the transform

$$W \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

repetitively to any initial point (x_0, y_0) will yield the sequence of points $(x_0/2, y_0/2)$, $(x_0/4, y_0/4)$, \dots , which can be seen to converge to the point $(0, 0)$, in the limit.

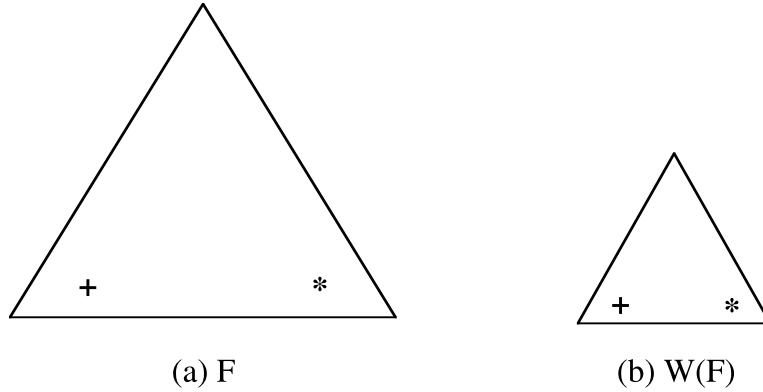


FIGURE 7.2
Effect of applying a contractive affine transform to a shape.

Iterated Function Systems

An iterated function system (IFS) is a collection of contractive affine transforms. The following is an IFS consisting of four transforms.

$$W_1 \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.16 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix}; \quad (7.3)$$

$$W_2 \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0.2 & 0.26 \\ 0.23 & 0.22 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0.2 \end{bmatrix}; \quad (7.4)$$

$$W_3 \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} -0.15 & 0.28 \\ 0.26 & 0.24 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0.2 \end{bmatrix}; \quad (7.5)$$

$$W_4 \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0.85 & 0.04 \\ 0.04 & 0.85 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0.2 \end{bmatrix}. \quad (7.6)$$

Actually, this is the IFS of a fern leaf. Later we see how this IFS generates a fern image.

A fundamental theorem of fractal geometry is that each IFS, that is, each set of contractive transforms, defines a unique fractal image. It is called the attractor of the IFS. The attractor of an IFS is unique; for each IFS there is only one attractor. This is the contractive mapping fixed-point theorem. The attractor of an IFS has the following property: if the IFS is made up of N affine transforms which are denoted by W_1, W_2, \dots, W_N , then the corresponding attractor A obeys

$$A = W_1(A) \cup W_2(A) \cup \dots \cup W_N(A).$$

This says that the attractor of the IFS is the same as the union of the transforms of the attractor. Now let us see how to generate the attractor of an IFS using the chaos game algorithm [2].

The Chaos Game Algorithm

Suppose an IFS contains N affine transforms W_1, W_2, \dots, W_N . Let these transforms have associated probabilities p_1, p_2, \dots, p_N , respectively. They obey

$$p_1 + p_2 + \dots + p_N = 1 \quad \text{and} \quad p_i > 0 \quad \text{for} \quad i = 1, 2, \dots, N.$$

They are the probabilities with which each transform is to be selected and applied in the chaos game algorithm.

Here is how the chaos game is played: choose any point (x_0, y_0) in a Euclidean plane. Select one transform in the IFS according to its probability and apply it to point (x_0, y_0) to get a new point (x_1, y_1) . Select another transform according to its probability and apply it to point (x_1, y_1) to get a new point (x_2, y_2) . Repeat this process to obtain a long sequence of points:

$$(x_0, y_0), (x_1, y_1), (x_2, y_2), (x_3, y_3), \dots$$

A basic result of the IFS theory is that this sequence of points will converge, with 100% probability, to the attractor of the IFS.

The following is pseudocode showing how the chaos game algorithm is applied in general:

- (i) Let $x = 0; y = 0$.
- (ii) Choose k to be one of the numbers $1, 2, \dots, N$, with probability p_k .
- (iii) Apply transform W_k to point (x, y) to obtain a new point $(x_{\text{new}}, y_{\text{new}})$.
- (iv) Let $x = x_{\text{new}}; y = y_{\text{new}}$.
- (v) Plot (x, y) .
- (vi) Return to step (ii) and repeat until a preset number of iterations is reached.

Using the above algorithm and the four transforms in Eqs. (7.3)–(7.6), we can generate a fern leaf as shown in [Fig. 7.3](#).

Transforms (7.3)–(7.6) can be represented compactly by just 24 parameters, although the fern generated from them is quite complicated and requires large amounts of data if stored in a bit-mapped format. So, if we can do the reverse of image generation and find the IFS of a given image, we can achieve a very high compression ratio. The next section describes how to find the IFS for a given image.

7.4 Image Compression Directly Based on the IFS Theory

The direct method of image compression using IFS is based on the collage theorem [2, 6]. Loosely speaking, it states that if we can find IFS W for an image B so that



FIGURE 7.3
The fern image generated from the transforms (7.3)–(7.6).

B and $W(B)$ are very similar, then the attractor of W will also be very similar to image B . Thus, we can store W instead of image B to achieve a very high compression ratio. Hence, to compress an image is to find the IFS of that image.

To find an IFS for an image, based on the collage theorem and the property of IFS attractors, we split the whole image into nonoverlapping segments whose union covers the entire image. If each segment is a transformed copy of the entire image or is very close to it, the combination of these transforms is the IFS of the original image. In other words, to encode an image into IFS is to find a set of contractive affine transforms, W_1, W_2, \dots, W_N , so that the original image B is the union of the N subimages:

$$B = W_1(B) \cup W_2(B) \cup \dots \cup W_N(B) .$$

We use an example to show how to find an IFS for an image. As shown in [Fig. 7.4](#), the Sierpinski triangle is the union of three small triangles: the top, bottom left, and bottom right triangles. Each small triangle is a copy of the transformed original Sierpinski triangle. If we can find these transforms, the IFS of the Sierpinski triangle is their combination.

Let us find the transform for the top triangle first. We know that the general format of the affine transform is Eq. (7.1). To determine the transform, we have to find six variables: a, b, c, d, e , and f . The first thing to do is to find corresponding points in the original Sierpinski triangle and the top triangle. Since it is clear, from [Fig. 7.4\(a\)](#), that point (x_1, y_1) is transformed to (x'_1, y'_1) , (x_2, y_2) to (x'_2, y'_2) and

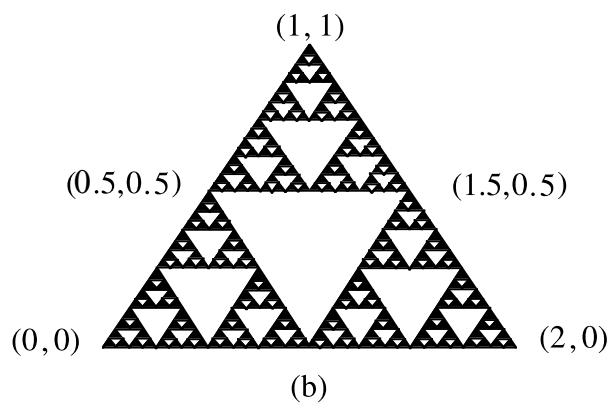
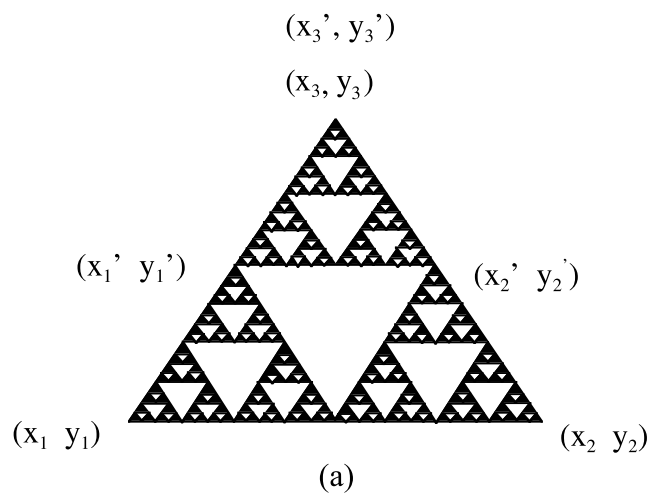


FIGURE 7.4
An example to find IFS for a given image.

(x_3, y_3) to (x'_3, y'_3) , we have the following six equations:

$$\begin{aligned} ax_1 + by_1 + e &= x'_1 \\ ax_2 + by_2 + e &= x'_2 \\ ax_3 + by_3 + e &= x'_3 \\ cx_1 + dy_1 + f &= y'_1 \\ cx_2 + dy_2 + f &= y'_2 \\ cx_3 + dy_3 + f &= y'_3 \end{aligned}$$

Using the coordinates in Fig. 7.4(b), we can solve the above equations to get $a = 0.5$, $b = 0$, $c = 0$, $d = 0.5$, $e = 0.5$, and $f = 0.5$.

In a similar way, we can find six variables for each of the transforms for the bottom left and bottom right triangles, respectively. Combining these transforms we have the IFS of the Sierpinski triangle as follows:

$$\begin{aligned} W_1 \begin{bmatrix} x \\ y \end{bmatrix} &= \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0.5 \\ 0.5 \end{bmatrix} \\ W_2 \begin{bmatrix} x \\ y \end{bmatrix} &= \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \end{bmatrix} \\ W_3 \begin{bmatrix} x \\ y \end{bmatrix} &= \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 1.0 \\ 0 \end{bmatrix}. \end{aligned}$$

Readers can verify the obtained IFS by applying the chaos game algorithm (with $p_i = 1/3$) using this IFS to see whether the Sierpinski triangle is obtained. After obtaining the IFS of the Sierpinski triangle, we can store the 18 parameters of three transforms instead of the bit-mapped data to achieve significant compression.

It should be noted that although an IFS has a unique attractor, many IFS can be found for a given image. For example, the Sierpinski triangle can be thought of as the union of nine small triangles, and then the determined IFS will consist of nine transforms.

In the above example, the attractor of the IFS perfectly covers the original image, which can be reconstructed from the IFS exactly. For an arbitrary image, it may be impossible or difficult to find an IFS whose attractor perfectly covers the original image. We should then find the “collage” as close to the original image as possible. According to the collage theorem [2, 6], the attractor of the IFS determined by the collage will be close to the original image.

For images with gray-scale or color, the same principle can be used to find the IFS, but obviously it would be much more difficult and time consuming.

7.5 Image Compression Based on IFS Library

For natural images, it is difficult to find the IFS directly as discussed in the previous section. But it is possible that images consist of a number of small objects whose IFS are known. For this kind of image we can proceed as follows. Using an image segmentation technique, we segment the image into small objects. An object can be a fern, leaf, cloud, fence post, or more complex collection of pixels.

We then look up these objects in a library of fractals. The library does not contain literal fractals; that would require large amounts of storage. Instead, the library contains compact sets of IFS codes that will reproduce the corresponding fractals. The library is searched to find fractals that approximate each segmented object. The corresponding IFS codes of these fractals are stored instead of the original image to

achieve high compression. This image compression approach using the IFS library is very similar to vector quantization in which each image block is represented with the index of the codeword that is most similar to a particular image block [15].

IFS codes in the library are obtained using the direct approach as discussed in the previous section. Since the same object can be contracted, rotated, and translated, it is not practical for the library to contain the same object in many different scales and different angles. Instead, fractals in the library are transformed to match the objects in the image. This matching process is similar to the matching process used in PIFS coding described in the next section.

The main problem with the library searching method is how to automatically and accurately segment an image into meaningful objects. There is no single effective method to do this.

7.6 Image Compression Based on Partitioned IFS

The direct and library-based approaches can compress images with a very high compression ratio. However, it is very difficult to find IFS automatically in natural images. To solve this problem, an alternative method has been developed [5, 6]. A natural image, such as a face, does not contain the type of self-similarity that can be found in the fractals. The image does not appear to contain affine transformations themselves. However images do, in fact, contain a different sort of similarity. Part of the image is similar to another part of the image. The distinction from fractal self-similarity is that rather than forming the image from copies of its whole self (under appropriate affine transformation), here the image is formed from copies of properly transformed parts of itself. Experimental results suggest that most images that one would expect to see can be compressed by taking advantage of this type of self-similarity [6].

Based on the above observation, fractal-based block coding, or PIFS-based coding, was developed [5]–[8]. The PIFS-based approach is as follows. To encode an image f , we divide it into range blocks $R_1, R_2, \dots, R_i \dots R_N$, such that

$$f = R_1 \cup R_2 \cup \dots \cup R_N$$

and

$$R_i \cap R_j = 0 \quad \text{when} \quad i \neq j.$$

That is, the range blocks cover the whole image and do not overlap.

The image is also divided into overlapping domain blocks $D_1, D_2, \dots, D_j, \dots, D_M$. For each range block R_i , we find a contractive transform W_i and a domain block D_j in the image, so that

$$R_i \approx W_i(D_j).$$

The combination of $W_1, W_2, \dots, W_i, \dots, W_N$ is called PIFS W . If W is simpler than the original image, we can encode f into W and achieve certain compression. When

decoding, according to the contractive mapping fixed point theorem, so long as W is contractive, application of W to an arbitrary image repeatedly will result in a fixed image. When $W(f)$ is close to f , the fixed image will be close to the original image f .

The three main issues involved in the design and implementation of a fractal block-coding system based on the above idea are (i) how the image is partitioned, (ii) the choice of a distortion measure between two images, and (iii) types of contractive affine transformations to be used. We now discuss these issues.

7.6.1 Image Partitions

The simplest partition of an image is fixed size partitioning; an image is divided into nonoverlapping square range blocks of fixed size ($B \times B$ pixels). For each range block, the entire image is searched for a square domain block which when suitably transformed is similar to the range block. The domain block is larger than the range block. Typical size is $2B \times 2B$ pixels, and they overlap every B pixels in both x -direction and y -direction.

The selection of sizes for range blocks is a compromise between compression ratio and reconstructed image quality. It is easy to find good matching domain blocks for small range blocks (4×4 and below), leading to high decoded image quality. But the achievable compression ratios will not be very high. On the other hand, it is generally harder to find good matching domain blocks for large range blocks (8×8 and above). But they allow a good exploitation of the redundancy in smooth image areas, leading to high compression ratios.

Fixed-size partitioning is the simplest. It ignores the contents of the image. To take advantage of image contents, other partitioning techniques, such as quadtree partitioning, horizontal-vertical (H-V) partitioning, have been proposed [6, 9]. In Section 7.7, we discuss the quadtree partitioning technique.

7.6.2 Distortion Measure

We use a distortion measure to determine the closeness of two image blocks: the smaller the distortion measure the more alike the two image blocks. There are many kinds of distortion measures. The common one used is the root-mean-square (RMS) distortion. For two square image blocks u and v of size $B \times B$ pixels, it is defined as

$$d(u, v) = \sqrt{\sum_{i,j} (u(i, j) - v(i, j))^2}$$

where summation is for $i = 0$ to $B - 1$ and $j = 0$ to $B - 1$.

In the search and mapping process, the RMS distortion is used to determine the closeness between a range block and a transformed domain block. The domain block causing the least distortion after certain transformation is deemed as a matching domain block to the range block.

We use the peak signal-to-noise ratio (PSNR) to measure the decoded image quality relative to the original image. Note that PSNR is not an exact measure of picture quality. It is used here for comparison purposes only.

7.6.3 A Class of Discrete Image Transformations

For each R_i , we must find D_j and W_i . Since we want to compress images with gray levels or color, we have to extend the basic form of affine transform to include pixel depth. We define the pixel depth at position (x, y) as $z = f(x, y)$. The extended affine transform becomes

$$W_i \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a_i & b_i & 0 \\ c_i & d_i & 0 \\ 0 & 0 & s_i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} e_i \\ f_i \\ o_i \end{bmatrix}.$$

For ease of reference, we simplify this as

$$V_i \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_i & b_i \\ c_i & d_i \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e_i \\ f_i \end{bmatrix}.$$

V_i determines how the partitioned domain blocks of an original image are mapped to range blocks, while s_i and o_i determine the contrast scaling and brightness shift of the transform, respectively. The operation of W_i to transform D_j to R_i can be decomposed into the following stages: geometric contraction of D_j , contrast scaling, brightness shift, and rotation and flip operations [5, 7]. If all pixels in R_i have the same (or similar) pixel values, these operations are not required. We just need to store a single pixel value for this range block. This operation is called absorption.

Geometric Contraction

The domain blocks must be spatially contracted to the size of the range block. In the simple case where the domain block is twice the size of the range block, the pixel values of the contracted domain block are the average values of four neighbouring pixels in the domain block.

Contrast Scaling

A contrast scaling factor s_i must be found to make the contrast among pixels in the domain block match the contrast among pixels in the range block. s_i is defined as the greatest brightness difference among pixels in the range block divided by the greatest brightness difference among pixels in the domain block.

Brightness Shift

The brightness shift o_i must be found to make the brightness of the domain block and the range block the same. o_i is defined as the difference between average pixel value in the range block and average pixel value in the domain block.

Rotation and Flip Operations

The rotation and flip operations do not modify pixel values; they simply shuffle pixels within a block, in a deterministic way — we call them isometries. There are many isometries. The following eight are commonly used [5, 7]:

- identity (no rotation or flip operation),
- orthogonal reflection about mid-vertical axis of block,
- orthogonal reflection about mid-horizontal axis of block,
- orthogonal reflection about first diagonal of block,
- orthogonal reflection about second diagonal of block,
- rotation around center of block, through $+90^\circ$,
- rotation around center of block, through $+180^\circ$,
- rotation around center of block, through -90° .

In effect, these operations are able to generate, from a single block, a whole family of geometrically related transformed blocks, which provides a pool in which matching blocks will be sought during the encoding. More complex transformations can be used. But more bits will be required to identify each transformation.

7.6.4 Encoding and Decoding Procedures

An image is divided into nonoverlapping range blocks. For each range block R_i , we first determine whether it is an absorption block. If it is not, a domain block D_j is sought which matches the range block best after certain transformation. The search starts at the domain block closest to the range block and extends in a spiral fashion until a satisfactory match is found.

For each nonabsorption range block, we must store information on the position of the corresponding domain block, the contrast scaling factor, the brightness shift, and the rotation and flip operations. Table 7.1 shows the number of bits required to encode a range block, assuming that the image size is $I \times I$ pixels and the range block size is $R \times R$ pixels.

During decoding, starting from an arbitrary image of the same size as the original image, each range block is computed from the corresponding domain block using the encoding information. Computing all the range blocks once is one iteration. After several iterations, the reconstructed image will be very close to the original image. The PSNR is used to determine whether further iterations can improve the quality of the reconstructed image. If there is no improvement in PSNR, the iterative process ends. According to the collage theorem, how close the reconstructed image can be to the original image is determined by the accuracy of the mapping from domain blocks to range blocks at the encoding stage.

Table 7.1 Number of Bits Required for Coding a Fixed-Size Range Block

Type of Coding	Parameters	Number of bits
Absorption	Identifier	4
	Absorption factor	8
Isometric	Identifier	4
	Scaling factor	3
	Shifting factor	8
	Domain block coordinates	$2 \log_2(I/R)$

7.6.5 Experimental Results

Table 7.2 shows experimental results obtained on the image “Lena” (512×512 pixels) and image “Lena” (256×256 pixels) with 8 bits per pixel (bpp), using different range block sizes [9]. It can be seen that decoded image quality is determined by the range block size used. The smaller the range block size, the easier it is to find a closer matching domain block, thus the higher the decoded image quality. Fig. 7.5 shows the original and compressed images “Lena” of 512×512 pixels with a compression ratio of 19 to 1.

Table 7.2 Experimental Results on Image “Lena” Using Fixed Size Range Blocks

Image size	Range block size	Compression ratio	PSNR (dB)
512×512	32×32	361.4	22.41
512×512	16×16	84.3	25.73
512×512	8×8	19.0	29.65
512×512	4×4	4.4	34.98
256×256	16×16	88.6	23.08
256×256	8×8	20.5	27.24
256×256	4×4	4.8	32.99

7.7 Image Coding Using Quadtree Partitioned IFS (QPIFS)

The weakness of the fixed-size partition is that the image is partitioned without considering the image contents. There are regions of the image that are difficult to cover well using fixed size range blocks. Similarly, there are regions that could be covered well with larger size range blocks, thus reducing the total number of maps needed (and increasing the compression ratio). This observation leads to the use of the quadtree partitioning technique [6, 9]. In a quadtree partition, range blocks as large as possible are used to code the image. When no good matching domain

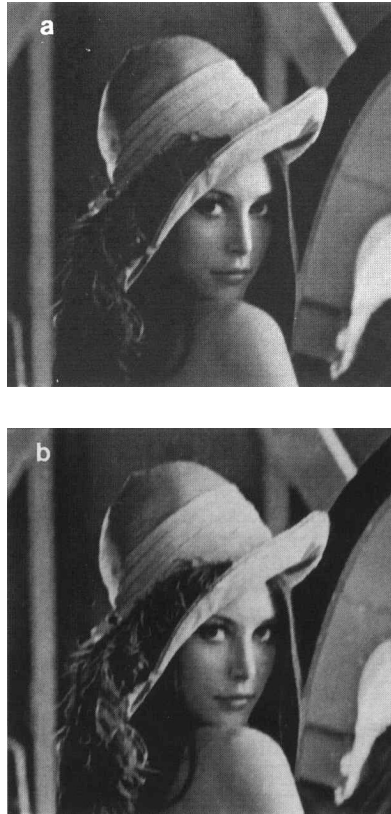


FIGURE 7.5

(a) The original image “Lena” of 512×512 pixels, 8 bpp. (b) Compressed image using fixed size range block of 8×8 pixels, at 0.42 bpp with PSNR of 29.65 dB. Reproduced by Special Permission of *Playboy* magazine. Copyright ©1972, 2000 by Playboy.

block (of one size bigger than the range block) can be found for a range block in the image, it is divided into four equally sized child range blocks as shown in [Fig. 7.6](#). These four children are named according to their direction in the parent range block – Northwest (NW), Northeast (NE), Southwest (SW), and Southeast (SE). This process repeats, starting from the whole image and continuing until the range blocks are small enough to be matched within some specified RMS tolerance. Small range blocks can be matched better than large ones because contiguous pixels in an image tend to be highly correlated. Therefore, one important issue in quadtree partitioning is to select a suitable RMS tolerance used in the matching process for range blocks of different sizes.

In fixed-size partitioning, the range block size is stored only once for the entire image, and the range block positions are implied if coded information of each range

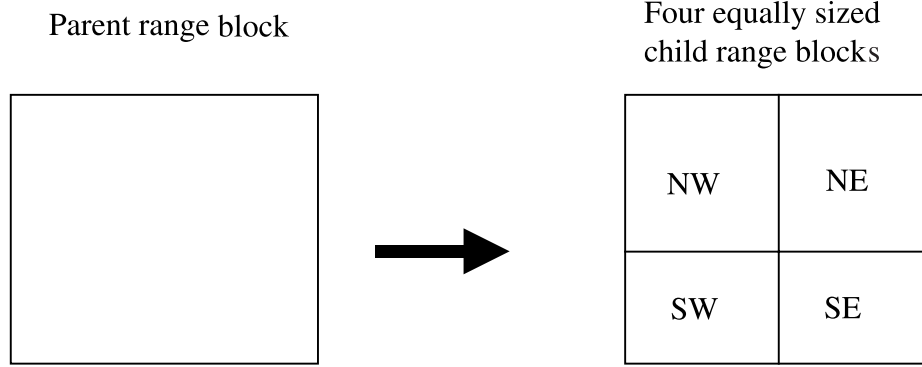


FIGURE 7.6
Partitioning of a parent range block.

block is stored sequentially. In quadtree partitioning, range block size varies. Information regarding range block size and position must be stored somehow. If this information is stored straightforwardly, we need 3 bits to store range block size and 18 bits for position for each block, assuming image size is 512×512 pixels. This total 21-bit overhead would likely offset the possible compression improvement gained by taking advantage of the image contents. So it is important to design a scheme to store this information compactly. In the next two subsections, we discuss RMS tolerance threshold selection and a compact storage scheme.

7.7.1 RMS Tolerance Selection

One of the main criteria for partitioning a parent range block into four equally sized child range blocks lies in the RMS tolerance value. Although a large tolerance value will lead to a high compression ratio, the quality of the decoded image will be low. Conversely, a small tolerance value will ensure that the decoded image is of high quality but the compression ratio will be compromised. Thus the selection of a suitable tolerance threshold plays a major role in the achievable compression ratio and decoded image quality.

PSNR is used to determine the decoded image quality. Consider a range block of size $R \times R$ pixels and pixel values range from 0 to 255,

$$PSNR = 10 \log_{10}[(255 \times 255 \times R \times R)/(d(f, g) \times d(f, g))]$$

where $d(f, g)$ is the RMS of the pixel difference between the range block and the transformation of the corresponding matching domain block.

Reorganizing the above equation, we have

$$d(f, g) = \frac{255R}{10^{PSNR/20}}.$$

This function determines the maximum allowable RMS difference used in the matching process given the size of the range block and required PSNR. This method has the advantage that the user can control the quality of the decoded image by choosing the required PSNR. The encoder, in turn, will calculate the maximum allowable distortion for the range blocks of different sizes.

7.7.2 A Compact Storage Scheme

Our aim is to use as few bits as possible to store the information about the range block sizes and positions, so that high compression ratios can be achieved.

Fig. 7.7 shows a simplified quadtree. The root node corresponds to the entire image of size $2^n \times 2^n$ pixels. We call this layer level n . The root has four children, which are on level $n - 1$. The next level is called level $n - 2$, and so on.

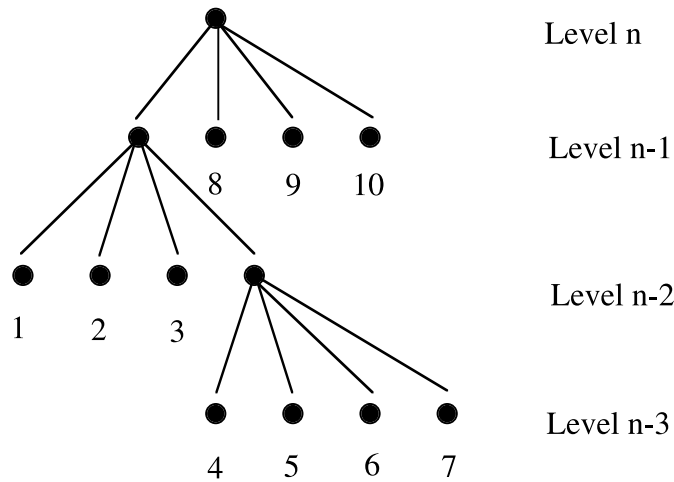


FIGURE 7.7
A simple quadtree.

The quadtree shows that the size and coordinates of four children can be calculated given the size and coordinates of the parent range block and the direction of children in the parent block. In other words, the range block size and coordinates are stored implicitly in the quadtree. The following recursive functions can be used for the decoder to calculate the sizes and coordinates of range blocks in a $2^n \times 2^n$ image given the level number and direction in the partition.

$$\begin{aligned}
 R(L) &= 2^n & \text{if } L &= n \\
 R(L) &= R(L + 1)/2 & \text{otherwise}
 \end{aligned}$$

where $R(L)$ is the size of a range block at level L of the quadtree,

$$\begin{aligned}
x(L, D) &= 0, y(L, D) = 0 \\
&\quad \text{if } L = n \\
x(L, D) &= x(L + 1, PD), y(L, D) = y(L + 1, PD) + R(L) \\
&\quad \text{if } L \neq n \text{ and } D = NW \\
x(L, D) &= x(L + 1, PD) + R(L), y(L, D) = y(L + 1, PD) + R(L) \\
&\quad \text{if } L \neq n \text{ and } D = NE \\
x(L, D) &= x(L + 1, PD), y(L, D) = y(L + 1, PD) \\
&\quad \text{if } L \neq n \text{ and } D = SW \\
x(L, D) &= x(L + 1, PD) + R(L), y(L, D) = y(L + 1, PD) \\
&\quad \text{if } L \neq n \text{ and } D = SE
\end{aligned}$$

where $x(L, D)$ and $y(L, D)$ are the x and y coordinates of a range block at level L , D is the direction of the range block with respect to its parent range block, and PD is the direction of the parent block with respect to the grandparent range block.

From the above discussion, it is clear that if we know the quadtree level number of each block and store the compressed information of blocks in a certain order, the decoder will be able to work out the block sizes and positions. Block information is stored in depth-first order. Take quadtree in Fig. 7.7 as an example; we store range block (node) information in the order as indicated by the number under each node. For each range block, we can simply store the level number followed by the information presented in Table 7.1.

A close look reveals that it may be redundant to store the level number for each block because there is a good chance that several consecutive blocks share the same level number. To eliminate this redundancy, we reserve a bit (called a *run bit*) for each block to indicate whether this block is at the same level as the previous block. If the block is at the same level as the previous block, this bit is set to 1 and no level number is stored. Otherwise, this bit is set to 0 and the level number is stored following it.

Experimental results show that for an image of $2^n \times 2^n$ pixels, the useful maximum range block size is $2^{n-2} \times 2^{n-2}$ pixels; the useful maximum range block size is one sixteenth of the original image size [9]. The minimum block size is 4×4 pixels. So the total number of quadtree levels is $n - 3$. The number of bits required to store this level information is $\log_2(n - 3)$ rounded to the next integer.

To summarize, the compressed file contains a list of compressed block information stored in depth-first order. For each block, we store a run bit, followed by transformation information (if run bit is set) or by level information and transformation information (if the run bit is off). The format of transformation information is the same as that used for the fixed-size partitioning method discussed in the previous section (Table 7.1).

7.7.3 Experimental Results

Tables 7.3 and 7.4 show the results of experiments carried out on the image “Lena” of 512×512 and image “Lena” of 256×256 pixels, respectively [9]. The maximum range block size is set to 128×128 pixels while the minimum is set to 4×4 pixels. It can be seen that more large range blocks are used when the required decoded image quality is low, and few large blocks are used when the required decoded image quality is high. This is because when the required decoded image quality increases, the distortion thresholds used for various range blocks decrease. As a result, large range blocks are not able to encode the region to within the acceptable threshold, and they are broken down further by quadtree partition. Fig. 7.8 shows the compressed image of 0.44 bpp using the quadtree partitioning method with PSNR of 30.30 dB.



FIGURE 7.8

Compressed image of 512×512 pixels using QPIFS at 0.44 bpp with PSNR of 30.3 dB. Reproduced by Special Permission of *Playboy* magazine. Copyright ©1972, 2000 by Playboy.

Table 7.3 Test Results on “Lena” 512×512 Using QPIFS

Number of range Blocks						Compression	Decoded image quality
128×128	64×64	32×32	16×16	8×8	4×4	Ratio	(PSNR in dB)
0	23	90	196	345	220	84.8	23.56
0	11	107	223	558	920	39.5	25.90
0	6	89	295	676	1728	25.4	28.27
0	4	63	325	866	2664	18.0	30.30
0	2	49	287	1025	4044	13.0	32.17
0	0	42	215	1113	5804	9.8	33.37
0	0	13	188	1111	8100	7.4	34.27

Table 7.4 Test Results on “Lena” 256×256 Using QPIFS

Number of range Blocks					Compression Ratio	Decoded image quality (PSNR in dB)
64×64	32×32	16×16	8×8	4×4		
0	25	90	196	272	33.0	23.82
0	11	109	255	628	18.9	26.50
0	7	88	320	960	13.7	28.76
0	4	72	353	1276	11.0	30.39
0	3	52	349	1676	9.0	31.81
0	1	48	283	2132	7.6	32.42
0	0	37	237	2556	6.6	32.73

By comparing the results obtained using fixed-size partitioning (Table 7.2) and those of quadtree partitioning (Tables 7.3 and 7.4), we can make the following observations:

1. When the required decoded image quality is low (below 25 dB), the fixed size partitioning provides a better compression ratio. For example, compressing image “Lena” of 256×256 pixels using range block size 16×16 pixels, a compression ratio of 88.6 is achieved, with a corresponding decoded image quality of 23.08 dB. In comparison, the same image compressed using quadtree partitioning gives a compression ratio of 33.0, although the quality is slightly higher at 23.82 dB. This is because the overhead information of range block size and position (on average about 2 bits for each block) is required for quadtree partitioning. Also, at low required decoded image quality, the image can be encoded using larger range blocks. In other words, the use of range blocks of 16×16 pixels in fixed-size partitioning is sufficient to generate a decoded image quality of 23 dB. This fact is evidenced in the quadtree partitioning method, in which approximately 75% of the image is encoded by range blocks of size 16×16 pixels or larger. Although only 25% of the image is encoded by smaller range blocks, the number of smaller range blocks used (total of 468, the sum of range blocks of 8×8 and 4×4) is much higher than the number of blocks of 16×16 or larger (total of 115, the sum of range blocks of 32×32 and 16×16). This is why the compression ratio decreases so much, although the decoded quality is increased by 0.8 dB due to usage of these smaller range blocks.
2. When the required decoded image quality is high, 30 dB and above, the quadtree partitioning method gives a better compression ratio and quality compared to the fixed-size partitioning. For example, we achieve a compression ratio of 4.8 with decoded image quality of 32.99 dB when image “Lena” of 256×256 pixels is compressed with fixed-size range blocks of 4×4 pixels. In comparison, the quadtree method is able to achieve a compression ratio of 6.6 with similar decoded image quality of 32.73 dB. This demonstrates the strength of the

quadtree partitioning method as it is able to encode certain regions with large range blocks, thus reducing the total number of range blocks to 2830. In fixed size partitioning, a total of 4096 range blocks of 4×4 pixels are used.

3. The results also show the flexibility of the quadtree partitioning method. The user is able to select the required decoded image quality. The encoder is then able to calculate the respective distortion tolerances for range blocks of different sizes. As a result, the method is able to use a combination of range blocks of different sizes to reproduce the decoded image with required quality. Fixed size partitioning does not have this flexibility. The achievable decoded image quality is determined by the range block size used in the encoding process. For example, in Table 7.2, when 4×4 range blocks are used, the resulting decoded image quality is 34.98 dB. When 8×8 range blocks are used, the resulting decoded image quality is 29.65 dB. As it is difficult to use range block sizes between 4×4 and 8×8 , the fixed-size partitioning will not be able to produce a decoded image quality between 34.98 dB and 29.65 dB with the highest possible compression ratio.
4. The larger the image size, the higher the achievable compression ratio because larger images normally contain more spatial redundancy.

7.8 Image Coding by Exploiting Scalability of Fractals

Fractals are scalable in the sense that they have fine details at any scale. Based on this property, many researchers [4, 6, 7] have indicated that a fractal-encoded image can be decoded to any size. But this is true only to a certain extent with natural image compression. The original image is not a fractal, but the reconstructed image from PIFS is a fractal. Although we can display fractals at any size without losing fine details, it will not be very close to the original image if we enlarge it too many times.

Image coding using PIFS is time consuming. The complexity of computation is $O(n^4)$, assuming the image size is $n \times n$ and fixed-size partition is used. Thus it takes much longer to compress a larger image. This problem leads to the development of a scheme to reduce required compression time by using the scalability of fractals [9].

The basic idea is as follows. Given an image to compress, we reduce the original image size. Then we find the PIFS for this reduced image. During decoding, we use the scalability property to display the decoded image in its original size. As long as the decoded image has acceptable quality compared to the original one, we have achieved image compression using much less time and possibly with a higher compression ratio because a smaller image is coded. The important issues are how to reduce the original image size and how to decode the PIFS found on the reduced image to the original size.

7.8.1 Image Spatial Sub-Sampling

To reduce the original image, we use basic spatial subsampling techniques. Given an original image of $2^n \times 2^n$. If we want to reduce it to $2^{n-m} \times 2^{n-m}$, we represent each $2^m \times 2^m$ square block using one pixel. The value of the pixel is the average of these $2^m \times 2^m$ pixel values. For example, if we want to reduce the image size to one quarter of the original, we use one pixel to represent each square block of four pixels.

7.8.2 Decoding to a Larger Image

Decoding a compressed image of size $I \times I$ pixels represented by PIFS to $2^m I \times 2^m I$ pixels, where m is a nonnegative integer, involves decoding all the range blocks to size $2^m R \times 2^m R$ pixels instead of $R \times R$. The same decoding algorithm described in Section 7.6 is used, but special consideration must be taken to change block sizes and coordinates as shown below.

For a range block of size $R \times R$ with lower left corner at (x, y) that is encoded using absorption, to decode it to size $2^m R \times 2^m R$, the decoder will simply replace the block of $2^m R \times 2^m R$ pixels defined by lower left corner at $(2^m x, 2^m y)$ with the absorption factor.

To decode a range block with lower left corner at (x, y) that has been encoded by a contractive transformation, the decoder must be able to extract the matching domain block from the enlarged image. Thus the coordinates of the domain block (x_d, y_d) indicated in the compressed file must be changed to $(2^m x_d, 2^m y_d)$. A block of $2^m 2R \times 2^m 2R$ pixels with lower left corner at $(2^m x_d, 2^m y_d)$ is obtained from enlarged image. The appropriate contractive affine transformation is then applied to the block, to obtain the $2^m R \times 2^m R$ range block.

7.8.3 Experimental Results

Given the image “Lena” of 512×512 pixels, we first reduce it to 256×256 pixels. We find QPIFS for the reduced images. During decoding, we decode the QPIFS to image size of 512×512 pixels using the scalability property. Table 7.5 shows the experimental results obtained when images are compressed at 256×256 and then decoded to 512×512 using fractal scalability. Note that, in the table, both compression ratio and decoded image quality are calculated relative to the original image of 512×512 pixels. Table 7.6 shows results obtained when images are compressed at 512×512 pixels and decoded at the same size. Note that the results shown were obtained on a slow processor (SGI Indigo with a 50 MHz IP20 processor). With a faster processor, the time shown should be much shorter. Fig. 7.9 shows a reconstructed image using fractal scalability.

From the table we can make the following observations:

1. At similar decoded image qualities, coding time required to compress images of 256×256 is much less compared with compressing 512×512 directly without reducing its size first. Although not shown in the table, the decompression times required for decoding are similar for all cases.



FIGURE 7.9

Image “Lena” of 512×512 pixels reconstructed from QPIFS found in images of 256×256 pixels using fractal scaling, at 0.22 bpp with PSNR of 28.77 dB. Reproduced by Special Permission of *Playboy* magazine. Copyright ©1972, 2000 by Playboy.

Table 7.5 Test Results Using Combination of Subsampling and Scaling Technique

QPIFS coded at	Compressing time (s)	Decoded to	Compression ratio	Decoded image quality (PSNR in dB)
256×256	777	512×512	131.5	23.10
256×256	1392	512×512	75.4	25.26
256×256	1957	512×512	54.7	26.91
256×256	2444	512×512	44.0	27.96
256×256	3006	512×512	35.9	28.77
256×256	9538	512×512	30.0	29.08
256×256	11008	512×512	26.4	29.24

2. At similar decoded image qualities, the compression ratio achieved using scaling is much higher than that using direct encoding and decoding at the same size. For example, we achieved a compression ratio of 35.9 with decoded image quality of 28.77 dB using the scaling property, compared with a compression ratio of 25.4 with image quality of 28.27 dB achieved with direct encoding and decoding.
3. Since our purpose is to obtain a decoded image at the original image size, in the case of 512×512 pixels, the decoded image quality is calculated relative to the original image of 512×512 pixels. There is a quality degradation when decoding QPIFS found in the image 256×256 pixels to an image of 512×512 pixels. For example, in the last row of Table 7.5, the QPIFS found

Table 7.6 Test Results Using QPIFS Directly on a 512×512 “Lena” Image

Compressed at	Compressing time (s)	Compression ratio	Decoded image quality (PSNR in dB)
512×512	4645	84.8	23.56
512×512	10235	39.5	25.90
512×512	16075	25.4	28.27
512×512	22789	18.0	30.30
512×512	31797	13.0	32.17
512×512	42639	9.8	33.37
512×512	56415	7.4	234.27

for the image of 256×256 pixels should be able to generate an image of 256×256 pixels at 32.73 dB (relative to the uncompressed image of 256×256 pixels). However, when it is decoded to 512×512 pixels, the PSNR drops to 29.24 (relative to the original image of 512×512 pixels). This proves that scalability of fractals can be used only to a certain extent in natural image compression. Enlarging too often will result in unacceptable decoded image quality.

7.9 Video Sequence Compression using Quadtree PIFS

There are high temporal correlations among images in a video sequence. To achieve high video compression performance, these temporal redundancies must be removed or reduced. In this section, we describe a technique to remove temporal redundancy using quadtree partitioning [9].

The basic idea of still image compression using PIFS is to find similarities among parts of an image. We can borrow this idea to compress video sequences. However, instead of finding similarities among parts of an image, we find similarities among neighboring images in a video sequence. In this case, quadtree partitioning will be very efficient because there are large areas that are unchanged between consecutive frames; thus many large range blocks can be used. Based on whether or not there is a change in a block of pixels relative to the corresponding block in the previous image and the type of changes (translation or rotation, etc.), we identify different types of blocks in the image and code them differently to achieve better compression performance.

7.9.1 Definitions of Types of Range Blocks

If a block in the current frame is identical or very similar to the corresponding block in the previous frame, we call this block a *type one block*. If a block in the current frame is a translation of a block in the previous frame, we call it a *type two block*.

If a block in the current frame is an affine transformation of a block in the previous frame, we call it a *type three block*.

Type One Range Blocks

In a typical video sequence, a large portion of the current frame is similar to the previous frame except for areas where moving objects are located. We can use type one blocks to code this unchanged portion. Since the only information to be stored is the position of this block, a high compression ratio can be achieved. To identify a type one range block, a block in the current frame is compared with the corresponding range block in the previous frame. If the error between the two range blocks is below a preset threshold value, the block is deemed as a type one range block.

Type Two Range Blocks

For the area containing moving objects, if the movement of objects can be traced from one frame to the next, a high compression ratio can be achieved. This observation leads to the implementation of type two range blocks. Each type two range block in the current frame is produced by a translation of a same size block in the previous frame. A range block is identified as a type two block if the distortion between itself and a block obtained from the search region defined in the previous frame is smaller than a preset threshold.

The search region is defined based on the observation that objects generally do not move too far from one frame to the next. Thus the search region is a small area in the previous frame instead of the whole image, saving bits required to represent the x and y offsets between the range block and the matching block in the previous frame. A search range of M is used. The search region is defined as in [Fig. 7.10](#). The size of the search region is $(2M + R) \times (2M + R)$, assuming the range block size is $R \times R$. When the range block is near the boundaries of the image, the search region is constrained by the boundaries. During the search process, the matching block will be searched exhaustively in the previous frame by changing the block position one pixel at a time within the search region. After the exhaustive search process, the block giving the minimum error between itself and the range block to be coded is deemed as a type two matching block if the distortion is below the preset distortion threshold.

Type Three Range Blocks

For the area that cannot be encoded using type one or type two blocks, type three blocks are used. Type three range blocks are encoded in a similar way as the range block in the still image as discussed in Sections 7.6 and 7.7. The only difference is that the matching domain blocks are searched in the previous frame. This is based on the observation that parts of the current frame may possibly be parts in the previous frame after rotation, intensity change, etc. Thus, it would be easier to find matching blocks in the previous frame than in the current frame.

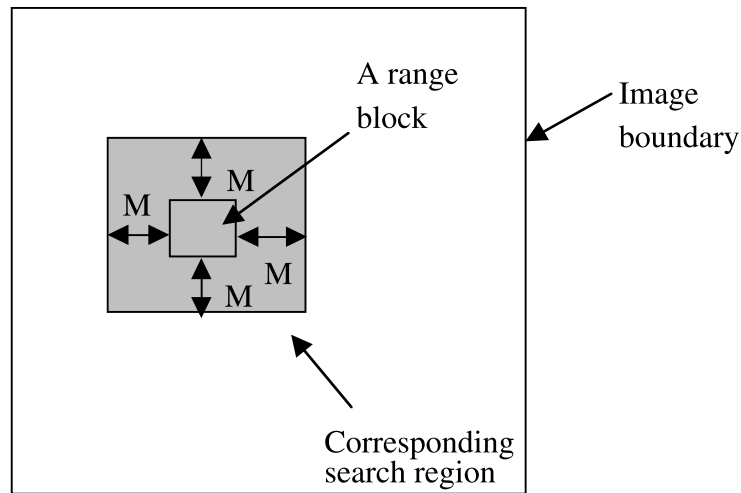


FIGURE 7.10
Definition of the search region.

Distortion Tolerance

The distortion tolerance for all three types of blocks is determined by the same function as described in Section 7.7.

7.9.2 Encoding and Decoding Processes

We treat the first image in a video sequence as a still image (intraframe) and code it using quadtree PIFS as described in Section 7.7. There are two ways to code the subsequent frames. One way is to code them relative to the previous original images. The advantage of this method is that the similarity between the current frame and the previous original image will be high, resulting in the use of more type one and type two blocks, thus achieving higher compression ratios. The disadvantage is that the decoder does not have the original images. It has to decode images based on the previous decoded image. Since we are using a lossy coding method, the decoded image is different from the original image. If we decode images based on previous decoded image, the differences will accumulate and eventually the decoded image quality will no longer be acceptable unless we take remedial measures. The other way to code the images is to code them relative to the previous decoded image. The advantage and disadvantage are the opposite of the first method. In the following, we describe the implementation of the first method. Measures are taken to prevent error accumulation to an unacceptable level.

To solve the error accumulation problem, we divide a video sequence into layers (Fig. 7.11). The highest layer is the video sequence itself. It is divided into a number of fixed length subsequences. The first image in each subsequence is intraframe

coded. That is, it is treated as a still image and coded using quadtree PIFS. Each subsequence is further divided into a number of fixed-length groups of images or frames. The first image in each group of images is coded relative to the first image in the previous group of images. By doing so, the rate of quality degradation is reduced. The encoding process for all images in a group of images except the first is the same; they are coded relative to the immediately previous images. Therefore, except for the first image in a subsequence, all images are interframe coded.

In the following discussion, we use the general term *reference image* to describe the image based on which the current image is encoded. The reference image is the previous image if the current image is not the first in a group of images. Otherwise, the reference is the first image in the previous group of images.

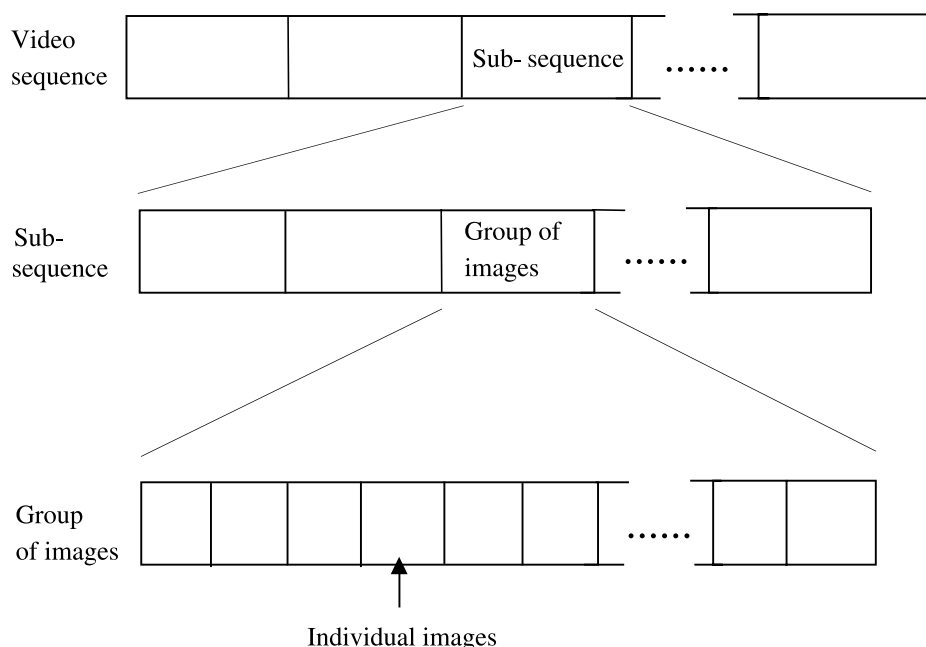


FIGURE 7.11
The hierarchy of a video sequence.

The reference image is used to search for type one and type two range blocks, and for matching domain blocks to encode type three range blocks. The quadtree partitioning method with extension to take care of the use of type one and type two range blocks is utilized to encode each image except the first image in a subsequence.

An image to be interframe coded is divided into 16 equal square range blocks. Each range block to be encoded is first tested to determine whether or not it can be coded using type one block relative to the reference image. It is encoded using type one block if the distortion is below the preset threshold. Otherwise, it is tested to determine if it can be encoded using type two block. If the current range block cannot

be encoded using type one or type two blocks, a matching domain block is searched from the reference image. If the difference between the range block and a domain block, after it has undergone an appropriate transformation, is smaller than a preset threshold, the range block is encoded using type three block. Otherwise, the range block must be partitioned into four smaller range blocks using quadtree partitioning. For each smaller range block, the above encoding process is repeated until all range blocks are encoded.

During decoding, the quadtree used in the encoding process is rebuilt as described in Section 7.7. The range blocks encoded using type one or type two blocks are reconstructed from the reference image. The areas encoded using type three blocks are initialized with the contents at the corresponding location in the reference image and are reconstructed by applying stored PIFS. By initializing these areas with the contents of the reference image, very few iterations are required to obtain the converging image. Experiments show that in most cases only one iteration of applying the PIFS is required to achieve convergence to the final image.

7.9.3 Storage Requirements

Table 7.7 shows the storage requirements for different types of blocks. In an actual implementation, additional bits are required to store range block sizes and coordinates as described in Section 7.7. By using quadtree partitioning, we expect that unchanged areas will be coded using larger range blocks, leading to higher compression ratios.

Table 7.7 Storage Requirements for Different Types of Range Blocks

Type of encoding	Parameters	Number of bits
Type one	Identifier	4
Type two	Identifier	4
	Coordinates	$2 \log_2(2M)$
Type three with absorption	Identifier	4
	Absorption	8
Type three with isometric	Identifier	4
	Scaling factor	3
	Shifting factor	8
	Domain coordinates	$2 \log_2(I/R)$

7.9.4 Experimental Results

In the reported experiments, the size of a subsequence used is 100 images and the size of a group of images is 10. These numbers are chosen to achieve optimal compromise between compression ratios and effects of accumulated error.

The “salesman” video sequence of 90 frames is compressed using the algorithm described above. The compression ratio achieved is 57.9 with an average decoded

image quality of 29.17 dB. Using software-only decoding, a decoding rate of 3 frames per second with image size of 256×256 pixels was achieved on an SGI Indigo workstation with 50 MHz IP20 processor. With optimization of the code and faster processors, real-time video decoding is possible.

7.9.5 Discussion

The interframe coding method discussed in this section is similar, to a certain extent, to the motion estimation and compensation techniques used in the MPEG standard [14]. The QPIFS-based method has two advantages. First, it uses quadtree partitioning instead of fixed-block size used in MPEG. This leads to better compression performance because nonmoving areas can be coded using larger blocks. Second, the QPIFS-based method not only estimates and compensates for translation but also considers object rotation, brightness changes, etc. by using affine transformations.

7.10 Other Fractal-Based Image Compression Techniques

The techniques discussed so far are all based on iterated function systems. In this section, we briefly describe two techniques that are not based on IFS but make use of other properties of fractals.

7.10.1 Segmentation-Based Coding Using Fractal Dimension

In most cases, images are meant to be viewed by the human eye. The human visual system (HVS) is not perfect, and it is less sensitive to certain frequencies than to others. The less sensitive components can be coded coarsely without much perceived quality loss. Therefore, if a coding system can take advantage of the properties of HVS, a high compression ratio can be achieved.

One such technique is segmentation-based image coding [10]. Images are segmented into homogeneous regions with similar features, and each region is coded using different techniques based on their visual importance. However, there are limitations in the traditional segmentation-based coding. The main limitation is due to the fact that the image is normally segmented into regions of constant intensity. In complicated texture areas, a trade-off must be made. Good representation of texture requires many small segments. In order to get low bit rates or high compression ratios, however, the number of segments should be small. This problem can be solved by segmenting images into texturally homogeneous regions with respect to the degree of roughness perceived by HVS.

A characteristic of a fractal is the fractal dimension that provides a good measure of perceptual roughness of texture, with increasing values in fractal dimension representing perceptually rougher texture. Techniques to calculate the fractal dimension

can be found in Falconer [3] and Peleg, Naor, Hartley, and Avnir [11]. After the fractal dimension is calculated, an image is segmented into several texture classes according to the fractal dimension. After image segmentation, an efficient image coding technique is chosen to encode each texture class according to visual importance. It was reported that a compression ratio of 40 was achieved for gray scale images with good quality reconstructed images [10].

7.10.2 Yardstick Coding

This method is based on fractal geometry to measure the length of a curve using a yardstick of fixed-length (see Fig. 7.12). We want to measure the curve drawn in thin dotted lines using a yardstick. The thick lines are covered by the yardstick travelling along the curve. It is obvious that the shorter the yardstick, the closer the measurement result will be to the true length of the curve, and the better the curve will be covered by the yardstick travelling along the curve.

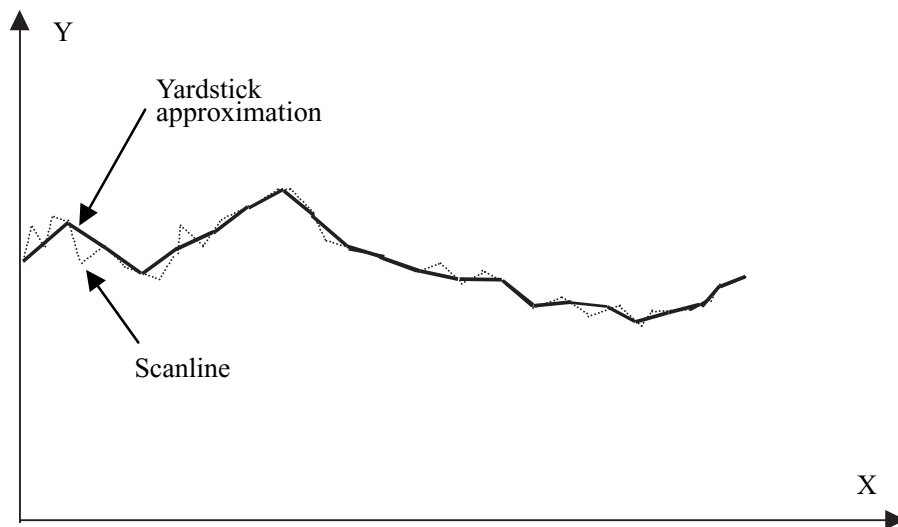


FIGURE 7.12
Coding (approximation) of a scan line.

Walach and Karnin [12] proposed the so called “yardstick travelling” mechanism for coding each line of pixels in an image. A scan line of pixels can be thought of as a curve: the x -coordinate is the pixel number along the line, and the y -coordinate is the intensity of pixels. The curve can be approximated with a set of straight lines covered by the running yardstick. These straight lines can be represented by points coinciding with the ends of the yardstick when it is running along the curve. Since the number of end points will be smaller than the pixel numbers when the length of the yardstick is appropriately chosen (normally between 8 and 24 pixels), compression can be achieved by storing these points instead of pixels [12, 13].

The yardstick method is similar to the traditional subsampling and interpolation method, both exploiting the correlation among neighbouring pixels.

7.11 Conclusions

This chapter has described a number of image and video compression techniques based on fractal properties. Techniques based on IFS and PIFS are most promising. The compression performance of PIFS-based techniques is similar to that of DCT-based techniques.

The PIFS-based compression technique has an advantage in that the compression achievable at a given signal to noise ratio scales with image size: higher compression ratios can be achieved with larger images. This shows the potential of this technique in applications involving large images. IFS-based fractal image coding is highly asymmetric in that significantly more processing is required for encoding than for decoding. It is highly suitable for information dissemination applications where images are encoded once and decoded many times. At the decoding site, no sophisticated hardware is needed to achieve high decoding speed.

IFS-based image/video compression techniques are potentially suitable for interactive multimedia applications where indexing, retrieving, and browsing of images are required. When an image is encoded into an IFS, one image or object is just one IFS. It is easy to index and search based on IFS.

Pointers to Further Reading and Available Software

This chapter has introduced basic concepts and techniques of fractal-based image and video compression. Many papers and much software are available online. Fisher maintains a Web site (<http://inls.ucsd.edu/y/Fractals/>) where one can find papers, books, software, and other resources. The University of Waterloo has an active research group working on fractal compression. Its home page (<http://links.uwaterloo.ca/>) has links to many papers and software. Institut fuer Informatik of Universitaet Freiburg, Germany, has an FTP site (<ftp://ftp.informatik.uni-freiburg.de/documents/papers/fractal/>) that contains many papers. One can find the latest products and developments in fractal compression and applications from the home page of Iterated Incorporated (<http://www.iterated.com/>).

References

- [1] Mandelbrot, B.B., *The Fractal Geometry of Nature*, Freeman, San Francisco, 1982.

- [2] Barnsley, M.F., *Fractals Everywhere*, Academic Press, Boston, 1988.
- [3] Falconer, K., *Fractal Geometry—Mathematical Foundations and Applications*, John Wiley & Sons, New York, 1990.
- [4] Barnsley, M.F. and Sloan, A.D., A better way to compress images, *Byte*, 215–223, 1988.
- [5] Jacquin, A.E., *A Fractal Theory of Iterated Markov Operators with Applications to Digital Image Coding*, Ph.D. thesis, Georgia Institute of Technology, August 1989.
- [6] Fisher, Y., Fractal image compression, SIGGRAPH'92, course notes.
- [7] Jacquin, A.E., A novel fractal block-coding technique for digital images, ICASSP'90, 2225–2228, Albuquerque, NM, 1990.
- [8] Jacquin, A.E., Fractal image coding based on a theory of iterated contractive image transformations, *SPIE*, vol. 1360, *Visual Communications and Image Processing*, 227–239, 1990.
- [9] Lu, G. and Yew, T.L., Applications of partitioned iterated function systems in image and video compression, *J. Visual Communication and Image Representation*, 7(2), 144–154, 1996.
- [10] Jang, J. and Rajala, S.A., Segmentation-based image coding using fractals and the human visual system, ICASSP'90, 1957–1960, Albuquerque, NM, 1990.
- [11] Peleg, S., Naor, J., Hartley, R., and Avnir, D., Multiple resolution texture analysis and classification, *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 6(4), 518–523, 1984.
- [12] Walach, E. and Karnin, E., A fractal based approach to image compression, ICASSP'86, 529–532.
- [13] Zhang, N. and Yan, H., Hybrid image compression method based on fractal geometry, *Electronics Letters*, 27(5), 406–408, 1991.
- [14] MPEG home page, <http://drogo.cselt.stet.it/mpeg/>.
- [15] Gersho, A. and Gray, R.M., *Vector Quantization and Signal Compression*, Kluwer Academic Publishers, Boston, 1992.
- [16] Forte, B. and Vrscay, E.R., Theory of generalized fractal transforms, in *Fractal Image Encoding and Analysis*, Fisher, Y., Ed., Springer Verlag, Heidelberg, 1998.
- [17] Mendivil, F. and Vrscay, E.R., Correspondence between fractal-wavelet transforms and iterated function systems with gray-level maps, in *Fractals in Engineering: From Theory to Industrial Applications*, Levy Vehel, J., Lutton, E., and Tricot, C., Eds., Springer Verlag, London, 1997.

- [18] Vrscay, E.R., A generalized class of fractal-wavelet transforms for image representation and compression, *Canadian J. of Electrical and Computer Engineering*, 23(1–2), 69–83, 1998. (Special issue on Visual Computing and Communications.)
- [19] Saupe, D. and Vrscay, E.R., Can one break the “collage barrier” in fractal image coding?, *Fractals in Engineering Conference*, June 14–15, 1999, Delft University, The Netherlands.
- [20] Zhao, Y. and Yuan, B., A new affine transformation: its theory and application to image coding, *IEEE Transactions on Circuits and Systems for Video Technology*, 8(3), 1998.