Advances in Fractal Compression for Multimedia Applications

John Kominek

Department of Computer Science
University of Waterloo
Waterloo, Ontario, Canada
(jmkominek@jeeves.uwaterloo.ca)

ABSTRACT

Fractal image compression is a promising new technology but is not without problems. Most critically, fast encoding is required for it to find wide use in multimedia applications. This is now within reach: recent methods are five orders of magnitude faster than early attempts. Beginning with the basic ideas and problems, this paper explains how to accelerate fractal image compression.

Keywords: image compression, fast fractal algorithm, r-trees, software video codecs, multimedia

1. INTRODUCTION

The integration of video into the computing environment involves many concerns, including communications hardware and software protocols, storage systems, user interface design, authoring tools, and methods to index and browse vast information resources [29]. Important though they are, it is video compression that resides at the center of this chain. The concerns of video compression are likewise diverse and include: image quality, bandwidth adaptation, playback performance, memory consumption, coding symmetry, size and frame rate scalability, interaction delay, and platform portability [42]. It is a complex task which, as the MPEG standard attests, requires considerable effort to realize.

This paper avoids such complexities by assuming a narrower focus — namely, the potential of fractal image compression for multimedia applications. These include integration of still pictures into on-line documents, and PC-to-PC video communications. Unfortunately, the highly asymmetrical nature of fractal encoding has tended to make the first application unattractive, and the second impossible. Even with dedicated hardware assist, the compression stage is known to be prohibitively slow. (Decompression is sufficiently rapid for real-time operation.) As a consequence, improving the time complexity has attracted considerable research attention. Significant progress has been made since the early days of this field.

Because a general multimedia audience is being assumed, this paper is a hybrid. It is part tutorial, part survey, and part presentation of new material. Beginning with a brief history of the field, Section 2 explains the basic ideas and problems of fractal image compression. Various attempts towards speed improvement are surveyed in Section 3. Section 4 introduces the Fast Fractal Image Compression algorithm, a new approach to breaking the speed problem. For still images, experiments show that at comparable quality levels the FFIC algorithm is 25 to 400 times faster than the current state of the art. Such an improvement brings real-time video applications within the reach of fractal mathematics. The prospects for software-only video codecs are briefly addressed in Section 5. Compared to other methods, fractal compression is relatively immature, and thus some areas requiring further research are outlined in the concluding section, Section 6.

2. BACKGROUND

2.1 Brief Synopsis

Before delving into details, these are some highlights of Fractal Image Compression.

1. It is a promising technology, though still relatively immature.
2. The fractals are Iterated Function Systems (IFSs).
3. It is a block-based, lossy compression method.
4. Fractal image interpolation may prove useful in multimedia applications.
5. Decompression is fast.
6. Compression has traditionally been slow.
7. Two patents have been granted on the technology. More are expected.

2.2 Brief History

The birth of fractal geometry is usually traced to IBM mathematician Benoit B. Mandelbrot and the 1977 publication of his seminal book *The Fractal Geometry of Nature* [35]. The book put forth a powerful thesis: traditional geometry with its straight lines and smooth surfaces does not resemble the geometry of trees and clouds and mountains. Fractal geometry, with its convoluted coastlines and detail *ad infinitum*, does. This insight opened vast possibilities. Computer scientists, for one, found a mathematics capable of generating artificial and yet realistic looking forms.

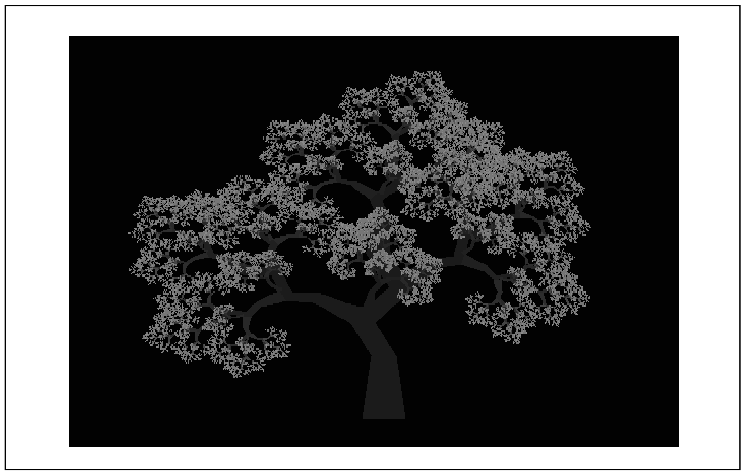


Figure 1. A simple fractal tree.

Shortly after Mandelbrot's work, mathematicians searched for a framework underlying fractal geometry. As John Hutchinson demonstrated in 1981, it is the branch of mathematics known as *Iterated Function Theory* [24]. Later in the decade Michael Barnsley authored *Fractals Everywhere*, another keystone work [6]. The book presents the mathematics of Iterated Functions Systems (IFSs), and develops a result known as the *Collage Theorem*. The Collage Theorem states what conditions an Iterated Function System must satisfy in order to represent an image. The tree-like image in Figure 1 was generated from a two dimensional Iterated Function System.

This presented an intriguing possibility. If, in the forward direction, fractal mathematics is good for generating natural looking images, then, in the reverse direction, could it not serve to compress images? Going from a given image to an Iterated Function System that can generate the original (or at least closely resemble it), is known as the *inverse problem*. In its general form, the inverse problem remains unsolved [46].

According to the testimony in [39] Barnsley's colleague Alan Sloan was the first to see the potential of IFS theory for image compression. Together they applied for (and were later granted) a software patent for the purpose of commercializing technology based upon their work [8]. At that time, however, fractal compression software required excessive human intervention and the method described in the first patent failed to become viable.

In search of something practical, Arnaud Jacquin, one of Barnsley's students, arrived at a modified scheme for representing images using *Partitioned Iterated Function Systems* (PIFSs). In his PhD thesis [26], Jacquin developed the necessary mathematical foundations and implemented the new approach in software, a description of which appears in his landmark 1992 paper "Image Coding Based on a Fractal Theory of Iterated Contractive Image Transformations" [27]. The algorithm was not sophisticated, and was computationally expensive, but it was fully automatic. All contemporary fractal image compression programs are based upon Jacquin's breakthrough.

Fractal compression can therefore be divided into two eras: that defined by the 'classical' approach of Barnsley, and that by the 'contemporary' approach of Jacquin. (Barnsley and Sloan hold a second patent on this later work [9].) The tutorial that follows explains how the contemporary emerged from the classical.

2.3 Classical Approach

An elegant way of introducing the notion of Iterated Functions Systems is by the metaphor of a Multiple Reduction Copying Machine [40]. A MRCM is imagined to be a regular copying machine except that:

1. There are multiple lens arrangements to create multiple overlapping copies of the original.
2. Each lens arrangement reduces the size of the original.
3. The copier operates in a feedback loop, with the output of one stage the input to the next. The initial input may be any image.

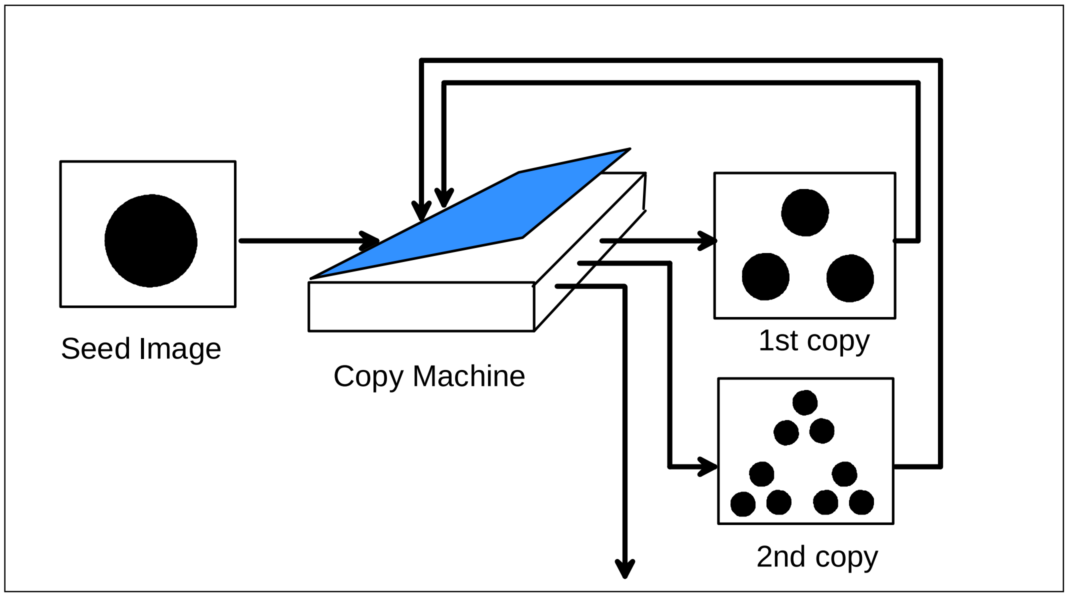


Figure 2. Schematic diagram of a Multiple Reduction Copying Machine.

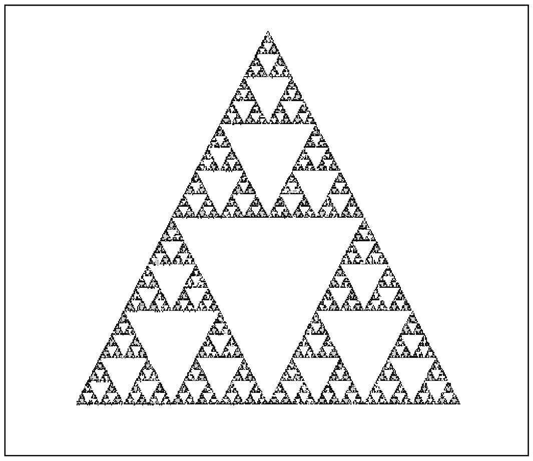


Figure 3. Sierenpinski's Triangle. This image represents the limit of the process depicted in Figure 2.

Figure 2 depicts this process for Sierpinski's Triangle — one of the simplest (and most well known) Iterated Function Systems. It is comprised of three component functions ("lenses"), each of which shrinks the input image by one half and translates it to a new position. This contractive property is crucial, for it guarantees convergence of the iterative process. Because all initial images are "drawn towards" the same final result, it is variously referred to as the *attractor* of the IFS, or the *fixed point* image.

Mathematically, each reducing lens is represented as a contractive affine transformation, w_i , that acts to scale, rotate, shear, and translate a copy of the input image, i.e.

$$w_i = \begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix} \tag{1}$$

so that a point in the initial image (*x*,*y*) will transform to new coordinates (*x*', *y*').

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix} \tag{2}$$

The symbols *a* ... *f* denote the transform coefficients. The three transforms that produce the Sierpinski Triangle of Figure 2 are:

$$w_1 = \begin{bmatrix} \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \end{bmatrix} \qquad w_2 = \begin{bmatrix} \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & 0 \end{bmatrix} \qquad w_3 = \begin{bmatrix} \frac{1}{2} & 0 & \frac{1}{4} \\ 0 & \frac{1}{2} & \frac{1}{2} \end{bmatrix} \tag{3}$$

Here, the first four coefficients (*a* ... *d*) are identical but this need not be the case in general. Just so long as the determinant of each transform is strictly less than one,

$$|ab - cd| < 1.0 \tag{4}$$

then the IFS as a whole, ***W***, will converge to the attractor image $\boldsymbol{I}_{\infty}$ from any initial image $\boldsymbol{I}_o$.

$$I_1 = W(I_o) \tag{5a}$$

$$I_\infty = \lim_{m\to\infty} W^m(I_o) \tag{5b}$$

$$W = \bigcup_{i=i}^{n} w_i \tag{5c}$$

As Figure 2 indicates, and equation (5) implies, each application of ***W*** produces detail at progressively finer levels as the limit is approached. Indeed, the image possess geometric *self-similarity* between different scales. This is why IFSs are said to generate fractal images. The promise of employing fractals for image compression, then, rests on four suppositions.

1. Many natural scenes possess this detail-within-detail structure.
2. Iterated Function Systems can generate fractal images that resemble natural scenes.
3. The corresponding IFS can be represented compactly.
4. The IFS can be reverse-engineered from the original image.

The truth of supposition 1 is what the fractal revolution begun by Mandelbrot is all about. The tree-like form in Figure 1 lends credence to supposition 2. The IFS that generates this tree is only slightly more complex than that of Sierpinski's Triangle: it consists of four transforms, listed in Table 1. Without any special effort devoted to efficient representation, the file that describes the generating IFS is 176 bytes in size, as compared to 264,000 bytes for a traditional pixel array representation. A compression ratio of 1,500 is certainly extreme — thus supposition number 3.

Transform	a	b	c	d	e	f
w_1	0.53	-0.08	0.08	0.53	-0.88	33.44
w_2	-0.31	-0.42	-0.44	0.33	-15.19	19.43
w_3	-0.25	-0.05	-0.07	0.29	1.48	11.73
w_4	0.29	0.54	-0.04	0.29	18.74	9.87

Table 1. The four affine transforms responsible for generating the tree in Figure 1.

A word of caution regarding compression ratios is required. Because an IFS generates detail down to the infinitesimal, one can claim "infinite" compression. From a certain perspective this is true. The components of an IFS are mathematical equations operating in $\boldsymbol{R^2}$, and just as with the equation of an ellipse, there are no bounds on precision. However, the legitimate use of "compression ratio" applies in the reverse direction — in beginning with a digitized image and seeking an alternate representation that requires less information. It is here that the classical approach to fractal image compression encounters difficulty.

In a classical IFS, the component transforms establish self-similarity from the image *as a whole* to smaller portions within. But for images that find use in human endeavor, this property is seldom present. A photograph of a landscape, for instance, may contain trees, grass, hills, and sky. Finding self-similarity between the image in whole and all the component parts (i.e. satisfying the Collage Theorem), is a daunting task, made even more problematic with people and buildings present.

A second shortcoming is that equation (2) omits the properties of color and intensity. Early attempts associated the intensity of a pixel with the density of points in the attractor, in analogy to grains of silver on a photographic plate [7]. This can produce interesting images, but is insufficiently flexible. Embedding the IFS in $\boldsymbol{R}^3$ but is a more direct approach, but exacerbates the problem.

Finally, classical Iterated Function Systems have the propensity for leaving “gaps” in the emergent image. The gaps between leaves in a tree may be left as sky, perhaps, but otherwise this is a serious shortcoming. When starting from a digitized photograph, all the pixels must be captured by the IFS.

In sum, the mathematics of Iterated Function Systems provides a powerful tool for *generating* computer graphics, but is, in its original form, deficient for representing existing images. Yet, with a few modifications, fractal-based compression becomes practical.

2.4 Contemporary Approach

The nature of the Partitioned Iterated Function System is illustrated in Figure 4. The basic idea is this: if finding self-similarity between an image in the whole and its parts is unrealistic, then seek self-similarity between *larger parts and smaller parts*. This is accomplished, as the name suggests, by partitioning the original image at different scales. Since images usually take the form of a rectangular array of pixels, partitioning the original image into blocks is a natural choice. Using Jacquin’s notation, the large partitions are called *domain blocks*, and the small partitions are *range blocks*.

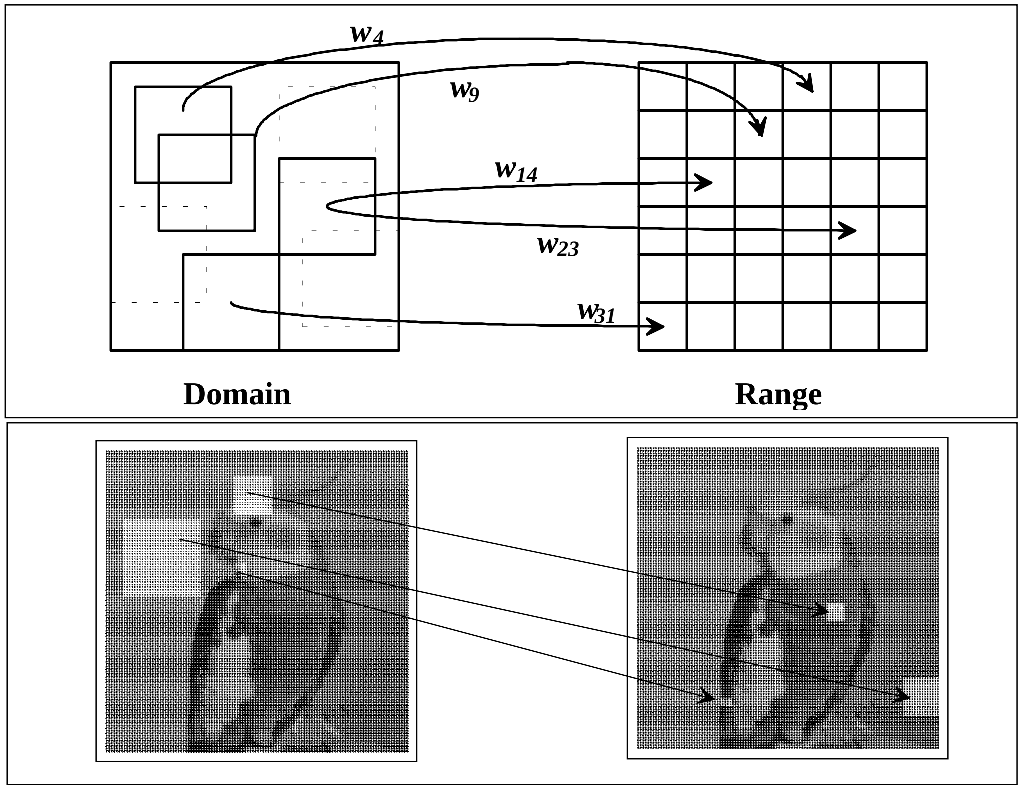


Figure 4. In a Partitioned Iterated Function System, self-similarity is sought between larger portions of the image (domain blocks) and smaller portions (range blocks). These are separated above for clarity. The bottom pair shows similarity at three scales.

Some mappings from domain blocks to range blocks are shown in Figure 4. Both type of blocks derive from the same image, but are separated in the illustration for clarity. The range blocks evenly partition the image so that every pixel is included. The larger domain blocks may overlap, and need not contain every pixel. The goal of the compression process is to find a closely matching domain block for every range block. The set of domain blocks considered in this operation is called the *domain pool*.

2.4.1 Grayscale Extension

In a Partitioned IFS, the intensity value of a pixel, z, is treated as a third spatial dimension. That is, the blocks in Figure 4 are actually cuboids, although the original terminology remains. To achieve convergence the intensity value of a pixel must also be scaled and offset, i.e.

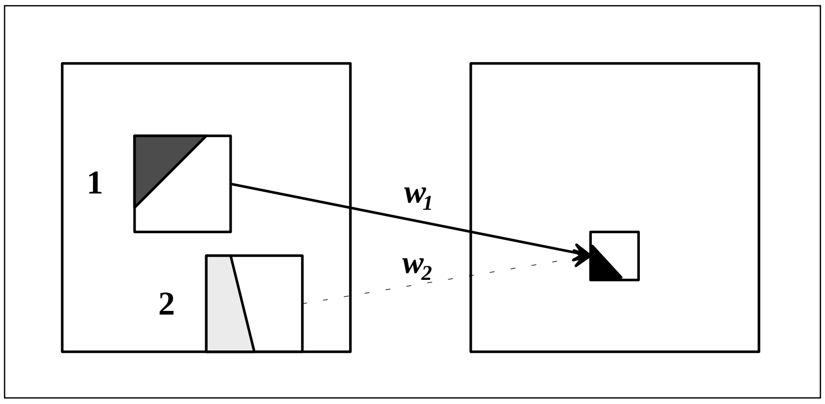
$$z' = s_i z + o_i \tag{6}$$

so that the affine transformation of (2) becomes three dimensional.

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = w_i \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a_i & b_i & 0 \\ c_i & d_i & 0 \\ 0 & 0 & s_i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} e_i \\ f_i \\ o_i \end{bmatrix} \tag{7}$$

The parameter s_i scales the pixel luminance and its effect is like the contrast knob on a television. When s_i is 0 the domain block maps to black, when equal to 1 it remains unchanged; between 0 and 1 the block loses contrast, and above 1 it gains contrast. The parameter o_i introduces an offset to the pixel luminance and is like the brightness knob on a television. Positive values of o_i brighten the block and negative values darken it. With contract and brightness control available, the extended affine transformation can accurately map grayscale domain blocks to grayscale range blocks. Three examples shown in the lower half of Figure 4.

To make the compression process tractable, Jacquin restricted equation (7) so that domain blocks are always square (not rectangles or parallelograms), and always twice the size of range blocks. If the range blocks are, say, 8x8 pixels in size, then the domain blocks are always 16x16. Doing so greatly reduces the size of the domain pool. This is favorable since it shortens the search time, but reconstruction quality suffers as optimal pairings may be excluded from consideration.



Eight symmetry operations
1. rotation by 0°
2. rotation by 90°
3. rotation by 180°
4. rotation by 270°
5. flip about horizontal median
6. flip about vertical median
7. flip about forward diagonal
8. flip about reverse diagonal

Figure 5. A domain block may undergo a symmetry operation before being mapped onto a range block. This increases the size of the domain pool.

One simple and effective way of improving coding quality is by allowing domain blocks to undergo an isometric symmetry operation prior to being transformed. The benefit of such an operation is illustrated in Figure 5, where block 1 is first rotated clockwise by 270° to improve the similarity between it and the range block. If the eight symmetry operations are not allowed, then a less optimal pairing, transformation $\boldsymbol{w_2}$, must be used instead.

Taken together, equation (7) becomes, in the Jacquin approach:

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = w_i \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} \frac{1}{2} & 0 & 0 \\ 0 & \frac{1}{2} & 0 \\ 0 & 0 & s_i \end{bmatrix} \begin{bmatrix} [-1,0,1] & [-1,0,1] & 0 \\ [-1,0,1] & [-1,0,1] & 0 \\ 0 & 0 & 1 \end{bmatrix}_8 \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} e_i \\ f_i \\ o_i \end{bmatrix} \tag{8}$$

The entries [-1,0,1] are a short form indicating that the element may be one, zero, or negative one. The subscript “8” indicates that only the eight symmetry operations (out of 81 possible forms) are considered. If we let m_i denote the symmetry operation applied, then the code for the PIFS representation of an image consists of a sequence of tuples, one per range block.

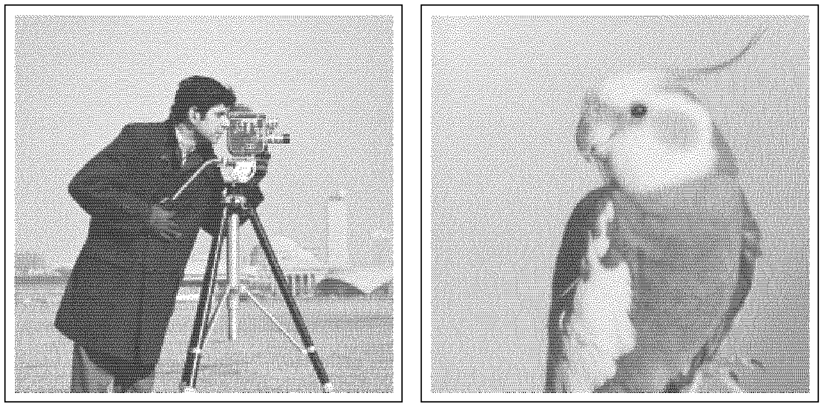
$$w_i = (e_i , f_i , m_i , o_i , s_i) \tag{9}$$

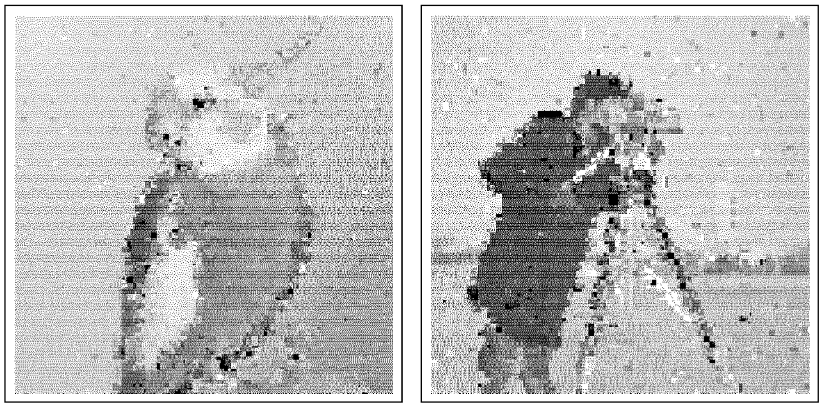
The first two coefficients locate the domain block, the third applies a symmetry operation, and the last two introduce an offset and scaling factor. Because the numbers associated with these coefficients implicitly define a set of affine transformations, a fractal encoded image is sometimes described as being “composed of mathematical equations.” Of itself, this is not new — all computer graphics are described by mathematical equations. What is new is the application to photorealistic images. And because the equations are evaluated by iteration, the decompression process is unique.

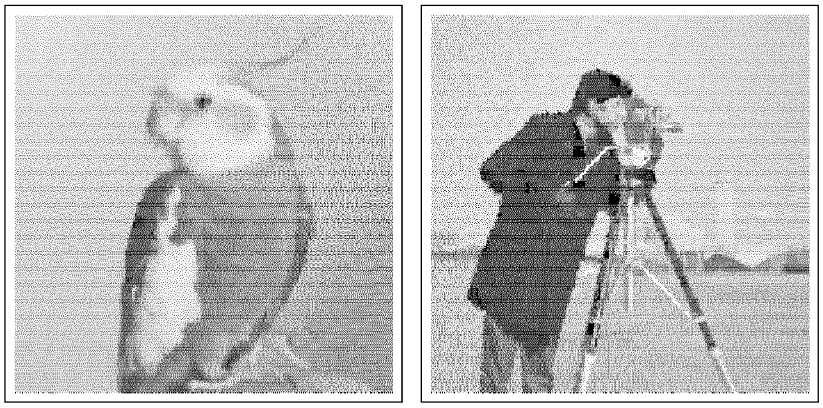
2.4.2 Decompression Process

The decompression process usually begins by setting the computer’s image buffer to a uniform mid-gray value. This is used as the seed image. During one iteration, the pixels of each range block in the transform list are evaluated. The result is used as the input for the second stage of iteration, as per Figure 2. After just two iterations the original image is recognizable, and after four the process will usually have converged (when eight bit precision is used per pixel). In the original description of [9] two alternating image buffers are used, one for the current iteration (containing the destination range blocks) and one for the previous iteration (containing the source domain blocks), but this is not necessary. One image buffer can be used for both.

Figure 6 illustrates the decompression process for two fractal encoded grayscale images — “Bird” and “Cameraman.” Each is 256x256 pixels in size, eight bits deep. All range blocks are 8x8; therefore, the corresponding PIFS is comprised of 1024 component affine transformations. To emphasize the significance of equation (5) — that the IFS describes an attractor and that the process converges regardless of the starting point — Cameraman is used as the seed image for Bird, and Bird is used as the seed image for Cameraman.


(a) left: seed image for Bird
(b) right: seed image for Cameraman


(c) 2 iterations of Bird IFS.
(d) 2 iterations of Cameraman IFS
Note: the defects result from starting with the ‘wrong’ image. They are seldom present when starting with an even mid-gray background.


(e) 4 iterations of Bird IFS.
(f) 4 iterations of Cameraman IFS.

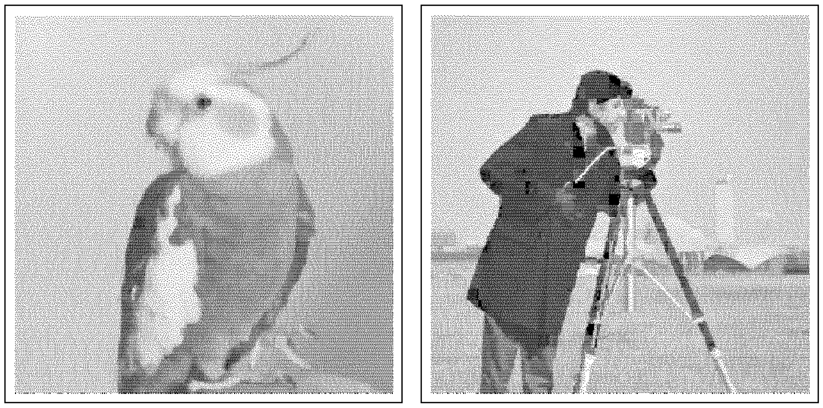

(g) 6 iterations of Bird IFS.
(h) 6 iterations of Cameraman IFS.

Figure 6. Decompression of Bird and Cameraman PIFSs. Each is partitioned into 8x8 range blocks.

Although the choice of seed image does not affect the outcome, it can affect how quickly the decompression process converges. One could instead begin with an all-black seed image, or an all-white one, but usually mid-gray is preferable. A successful way of increasing decompression speed, as first described by Beaumont in [5], is to begin with a low resolution version of the original. This is accomplished by modifying equation (6) so that o_i describes the mean value of a range block, rather than the relative offset from the corresponding domain block. The improvement afforded by this modification is show in Figure 7.

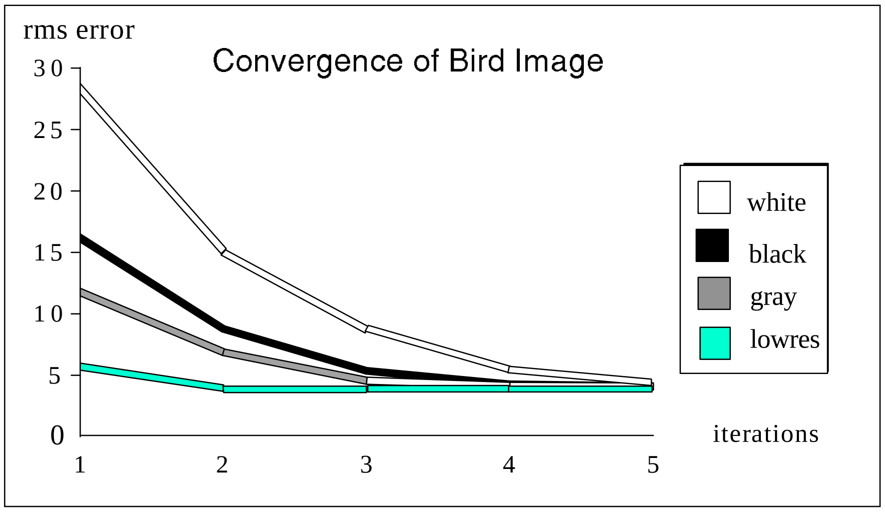


Figure 7. Convergence of the Bird image from different seed images. They are, from top to bottom: all white, all black, all mid-grey, and a low resolution version derived from the transform offset values.

2.4.3 Partitioning Extensions

The fractal images in Figure 6 have a compression ratio of about 16:1. This is derived by using the following bit allocations:

e_i	8 bits — 256 horizontal positions
f_i	8 bits — 256 vertical positions
m_i	3 bits — 8 symmetry operations
s_i	5 bits — sufficient from empirical tests
o_i	6 bits — sufficient from empirical tests
w_i	32 bits x 1024 transforms = 4096 bytes

This is a respectable degree of compression, but is not outstanding for the level of quality achieved. Numerous refinements can be made towards improvement. The most significant involves modifying the range block partitioning. Notice that for both the Bird and Cameraman images, there are large portions of smoothly varying background. These can be adequately represented by larger range blocks, perhaps 32x32 pixels in size. Moreover, in areas of high contrast and active detail, 8x8 blocks are insufficiently accurate; in these locations 4x4 range blocks are appropriate. Consequently, current fractal compression programs do not use a regular grid partitioning as indicated in Figure 4, but a hierarchical partitioning. Employing such a technique provides a good tradeoff between compression and quality.

The simplest, and most commonly used hierarchical partitioning is the *quadtree*. In a two-level quadtree structure, one might begin with a regular partitioning of 8x8 blocks. For each range block, the domain pool is searched for the best match. If the accuracy of the match falls within a certain tolerance, it is accepted. If not, the range block is subdivided, and a search is initiated for each sub-block. The partitioning scheme used determines the method of subdivision. Several possibilities are illustrated in Figure 8. Blue areas indicate where the original range-domain pairing is adequate, and yellow areas indicate regions where new pairings are required.

In schemes that employ *full quadtree* partitioning, a range block is either left un-split (top row), or split into four quadrants (rightmost entry in fourth row). In *partial quadtree* partitioning, one may use any of the sixteen possibilities for splitting a range block. In practice, only twelve are used, since if three quadrants need refinement, refining the forth quadrant will improve the image quality without increasing the total number of transforms. The refined quadrants may themselves be refined, creating deeper quadtree structures. Two and four-level structures are most commonly used, with the smallest blocks being 4x4 pixels in size.

An alternative method investigated by Fisher is HV (horizontal-vertical) partitioning, where a range block may be split into two rectangular pieces [15]. Others have attempted Delaunay triangulation, although it does not appear more successful than simpler schemes [12]. Figure 9 shows a full quadtree and a partial quadtree partitioning of the Bird image.

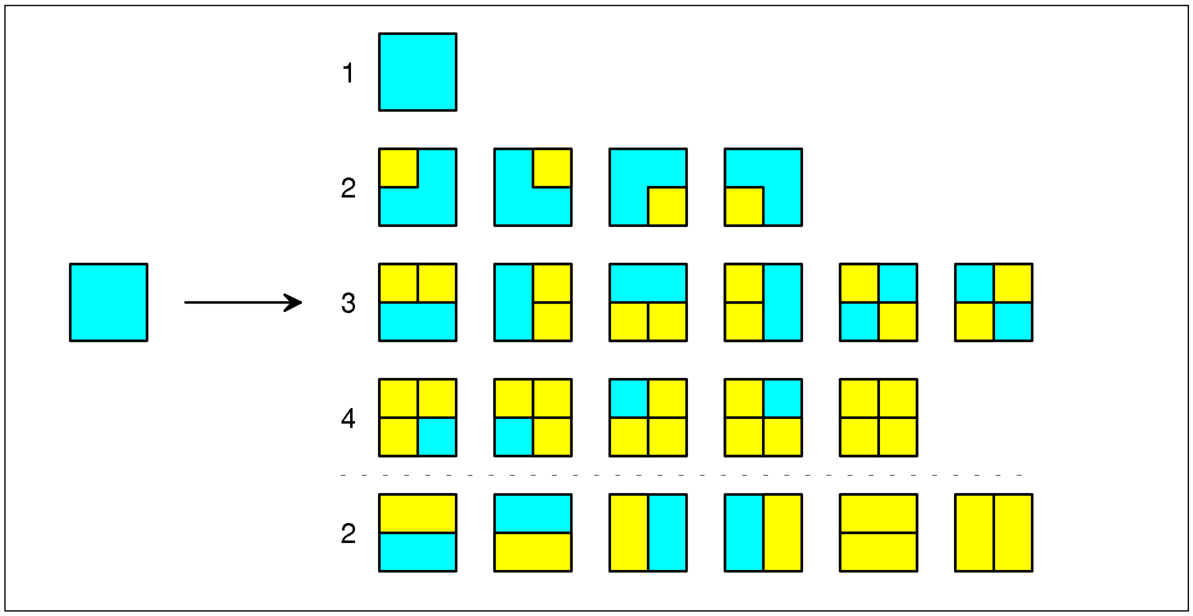


Figure 8. Possible ways of subdividing a range block. Yellow indicates areas where refinements is sought. The number of transformations required to represent a block is listed beside each row. The squares above the dotted line indicate possible quadtree partitioning; those below the dotted line are possibilities for HV partitioning.

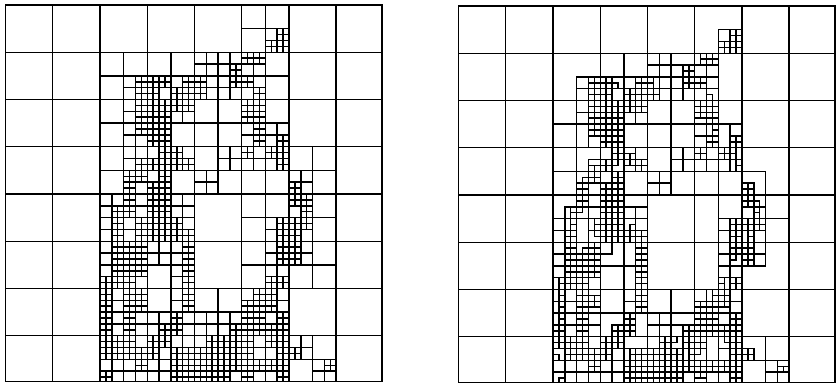


Figure 9. Full quadtree (left) and partial quadtree (right) partitioning of Bird.

2.4.4. Compression Process

The rapid convergence of the bottom curve in Figure 7 argues in favor of employing a low resolution version of the original. This is convenient because the offset parameters of (9) can be quickly computed as range block averages. Still, two crucial issues remain: how to determine the scaling parameters for the best range-domain pairing; and what is meant by the "best" match. In the original algorithm of Jacquin, the goal is to minimize the Haussdorff distance (i.e. greatest pixel-to-pixel difference) between a specific range block and a candidate domain block. To do so, a small set of scale values {0.45, 0.60, 0.80, 0.97} are tested in sequence, and the one that produces the smallest Haussdorff distance is retained.

If, instead, the mean square error measure is used, the optimal scaling parameter can be determined algebraically. First, assume that the domain block $\boldsymbol{D}_{xy}$ has been reduced to the size of the range block $\boldsymbol{R}_{xy}$ (by averaging 2x2 pixel cells), and that they have been adjusted to a zero-mean intensity level. Then, the mean square error between the blocks is

$$Error = \frac{1}{n^2} \sum_{x=1}^{n} \sum_{y=1}^{n} (s_i D_{xy} - R_{xy})^2 \tag{10}$$

By setting the derivative to zero,

$$\frac{\partial Error}{\partial s_i} = \frac{2}{n^2} \sum_{x=1}^{n} \sum_{y=1}^{n} (s_i D_{xy} - R_{xy}) D_{xy} = 0 \tag{11}$$

we have the desired result.

$$s_i = \frac{\Sigma\Sigma R_{xy} D_{xy}}{\Sigma\Sigma D_{xy}^2} \tag{12}$$

In other words, the optimal scaling factor between a range block and a domain block is their inner product divided by the domain block sum-of-squares. This value is calculated for all candidate domain blocks, under all eight symmetry operations, in search of the smallest error.

To ensure convergence of the decompression process it is common practice to force all component transforms to be contractive, that is, to restrict $|s_i| < 1.0$. This is not strictly necessary. In fact, releasing this constraint has been shown to improve image quality in some cases [15]. A minority of scale factors may well be expansive — just so long as the contractive transforms dominate, the overall process will converge [33].

With the ability to determine the optimal scaling factor between a pair of range and domain blocks, the last major issue is finding the best pairing. It is this realm of *search strategies* where fractal image compression programs diverge most dramatically in their approach. Before covering this topic in Section 3, it is worthwhile to examine the quality characteristic of fractal compression in some detail.

2.5. Quality Characteristics

The main source of interest in fractal compression is probably its ability to achieve very high compression ratios while still maintaining reasonable image quality [31]. Unlike the JPEG international standard [47], which undergoes sharp degradation after a certain critical point, fractal compressed images degrade more gradually. This is illustrated in Figure 10. Typically, JPEG introduces smaller errors at low compression ratios and, visually, exhibits a sharper preservation of fine details. Fractal compression has a propensity for blurring fine textures (e.g. the fur on a mammal's body) with a "flattened-out" effect. This effect can be quite subtle, however, and at higher compression ratios the relative superiority is apparent. The location of the crossover point in rate-distortion curves varies, but is usually between 10:1 and 30:1 on photorealistic grayscale images.

Figure 11 compares JPEG to fractal image compression for the Bird image. Samples are presented at 0.4 bits per pixel (20:1) and 0.2 bits per pixel (40:1), along with enhanced error images.

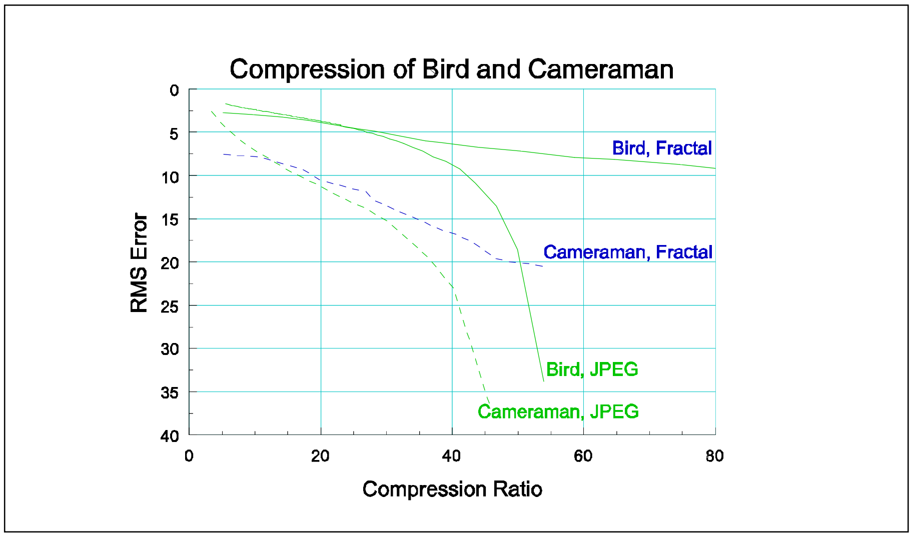
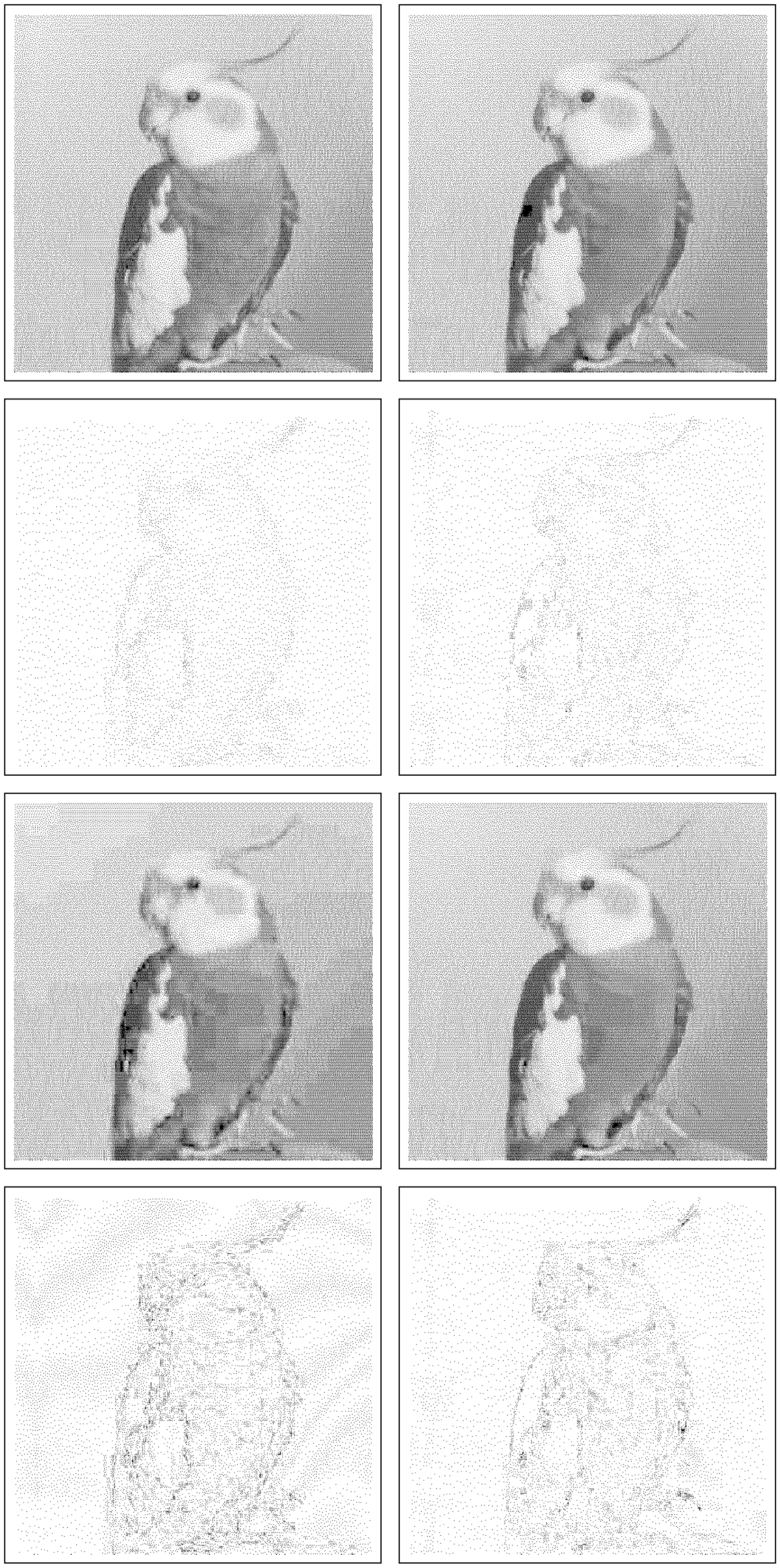


Figure 10. A comparison of rate-distortion curves for the Bird and Cameraman images.



(a) left: JPEG compression at 20:1.
(b) right: fractal compression at 20:1.

(c) JPEG error image at 20:1,
rms error = 3.93
(d) fractal error image at 20:1,
rms error = 4.26

(e) JPEG compression at 40:1
(f) fractal compression at 40:1

(g) JPEG error image at 40:1,
rms error = 8.52
(h) fractal error image at 40:1,
rms error = 6.36

Figure 11. Comparison of JPEG and fractal compression of Bird image.

Image Category	Low Compression	High Compression
text and line art	poor	poor
computer graphics	poor to good	poor to fair
photorealistic images	good	very good

Table 2. Qualitative rating of fractal compression for different image categories.

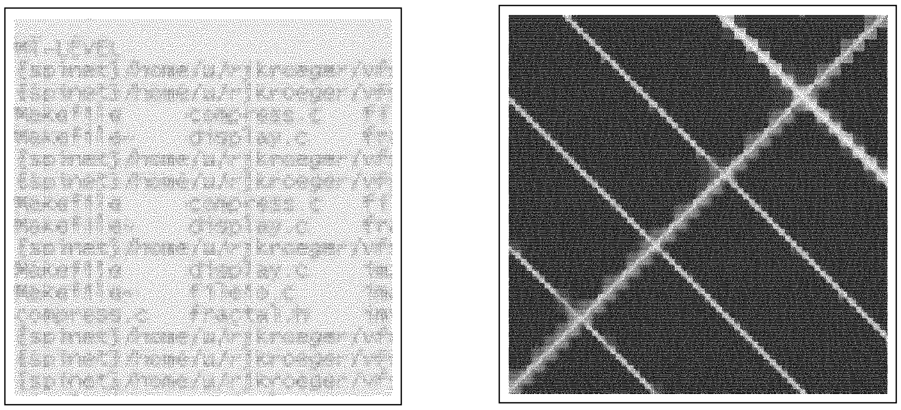


Figure 12. Examples where fractal image compression performs poorly. The text sample on the left is seriously blurred; the diagonal lines on the right are also blurred, with a discernible blockiness. In both situations the compression program could not adequately find regions of self-similarity.

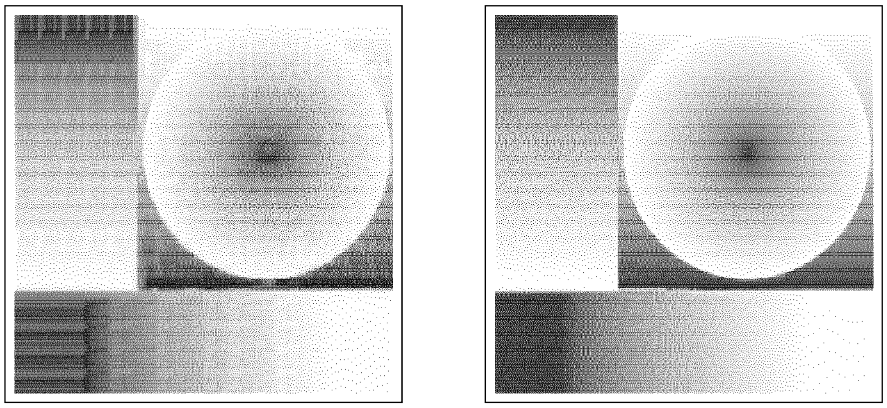


Figure 13. Example of a computer graphic that causes difficulty for fractal image compression. The image on the left is the result of applying the Jacquin algorithm. Only when the algorithm is modified to account for even slopes, right, can an it render an image that closely resembles the original. The compression ratio for both is approximately 40:1.

With the understanding that fractal image compression offers some unique strengths, it also possesses some weaknesses. These are not well publicized, but can significantly affect the success of an application. Briefly, fractal compression is poorly adapted to content not derived from "natural scenes." Most significantly, text and line art are seldom well preserved, as the images in Figure 12 show. This is particularly true for narrow lines that run diagonal to the grid partitioning.

Computer generated graphics with characteristically smooth gradients, such as that in Figure 13, may also not be well preserved. The defective left image of Figure 13 was generated using the process described in subsection 2.4.4. To properly treat smooth gradients, the position of a pixel within a range block must be considered. In a modification introduced by Monro [36], equation (6) becomes:

$$z' = [s_x x + s_y y + s_z z + o]_i \tag{13}$$

Thus, by employing a position-dependent affine transform [17], each range block is approximated by a bilinear patch, rather than just an average value. The improvement afforded by this modification is apparent in the right image of Figure 13. However, two extra parameters are required for each transform and the rate-distortion curve of a specific image may or may not be improved. In addition, deriving these parameters increases the encoding time.

2.6. Fractal Interpolation

Since a Partitioned IFS, like a normal IFS, is comprised of a set of contractive affine transforms, detail is created at progressively finer scales with each iteration. In practice, the limited resolution of a display monitor (or other output device) makes it unnecessary to proceed beyond a few stages. Still, this does not restrict the output image to be the same size of the original: it can be larger or smaller, provided that the aspect ratio remains unchanged. This is a useful property for multimedia applications. When, for instance, a library of images are available for browsing, it is worthwhile to present a palette of thumbnail versions. In the other direction, displaying an enlarged version may be necessary — electronic documentation containing embedded pictures is one likely circumstance. What is desirable, is that when the picture is enlarged, it does not suffer from pixelation artifacts. Fractal images are largely immune from pixelation problems.

The ability to expand a fractal encoded image without introducing artificial blocking is sometimes referred to as "resolution enhancement." This is an inappropriate term. Self-similar detail may be *generated* at a higher resolution, but it has not been retained from the original. If one expanded the Cameraman image, for instance, the grassy field will still maintain its characteristic texture, but at no stage will individual blades appear. What the method does offer is a sophisticated form of interpolation. As such, the proper comparison is not with pixel replication, but with other intelligent approaches. These include bilinear interpolation, cubic spline interpolation, and statistical methods based on a Bayesian approach [44].

It is worth mentioning that fractal interpolation can be applied to an image that has not been previously encoded. The procedure begins by constructing an accurate PIFS (high compression is unnecessary), and then using it to expand the original, before discarding the list of affine transforms. In other words, fractal compression is the means to an end, rather than the end itself.

3. SEARCH STRATEGIES

As already explained, the *domain pool* is the set of candidate domain blocks considered for pairing with range blocks. The compression stage then amounts, in essence, to a search for optimal pairings. The surest way is to take a brute force approach — by exhaustively examining the entire domain pool for each range block. The drawback is that such a search is O(n^2) and becomes impractical as the size of the source image grows. For good reason, then, reducing the computational demands has attracted the bulk of research to date [16,28].

Two avenues are open to attack: 1. lowering the computational complexity, and 2. decreasing the constant coefficient of performance. Some previous attempts at this task will be surveyed next, beginning with a closer look at brute force methods.

3.1 Heavy Brute Force

The fractal images shown in Figure 6 are based on a uniform 8x8 range block partitioning. Since the original is 256x256 pixel in size, 1024 pairings need to be established. If the domain blocks are restricted to twice the size of range blocks, the domain pool contains 8x(256-16+1)^2 = 464,648 elements (recall that eight the eight symmetry operations are allowed). In total, 464,648 x 1024 = 475,799,552 possible pairings require testing.

As is more typically the case, the images shown in Figure 11 employ a four level quadtree structure, with range blocks {4, 8, 16, 32} pixels wide. Although the number of comparisons increases as blocks become smaller, the total number of operations is mostly unchanged. This is because the inner product term of equation (12)

$$\sum_{x=1}^{rbs}\sum_{y=1}^{rbs} R_{xy}D_{xy}, \quad rbs = \text{range block size} \tag{14}$$

resides in the innermost loop of a fractal compression program. If we let

$$\begin{aligned} l &= \log_2(\text{range block size}) = 2,3,4,5 \\ m &= \log_2(\text{image width}) \\ n &= \log_2(\text{image height}) \end{aligned} \tag{15}$$

then the number of blocks in an image will be,

$$\begin{aligned} &\text{range blocks} = 2^{m-l}2^{n-l} = 2^{m+n-2l} \\ &\text{domain blocks} \cong 2^{m+n} \end{aligned} \tag{16}$$

and the number of multiply/accumulate operations required per comparison, according to (14), is $rbs^2 = 2^{2l}$. Thus, in a four level brute force search, a total of

$$\sum_{l=2}^{5} 8\times(2^{2l}2^{m+n-2l}2^{m+n}) = 2^{2+3}2^{2m+2n} = 2^{2(m+n)+5} \tag{17}$$

multiply/accumulate operations are necessary. This underestimates the full situation somewhat because it discounts the computation of the scale factors and the consequent error estimates, as well as numerous load/store/compare operations, but it is correct to within a factor of two.

Therefore, for 256x256 grayscale images, a brute force algorithm requires about 2^{37} = 128 Gflops (floating point operations). On a vector supercomputer, such as a Cray YMP-16, the encoding time might approach 10 seconds. But for workstations and PCs built around a single scalar microprocessor (with a much narrower memory-to-processor interface), the encoding may take upwards of a day. Still, having a modern supercomputer at one's disposal is less than a full solution: a 1024x1024 full color image demands about 100 Teraflops. It will be some years before Teraflop machines become available, let alone migrate to the desktop.

3.2 Light Brute Force

One way to reduce the search time is by restricting the location of domain blocks. For example, the top left corner of a domain block may be constrained to even pixel locations. In Figures 11b and 11e, all block positions are integral multiples of four. Doing so cuts the domain pool by a factor of sixteen and brings the search time within reasonable bounds. This approach is called a *light* brute force search, in contrast to the *heavy* brute force search of the previous subsection.

Because not every domain block is under consideration, the optimal pairing for a given range block may be overlooked. Consequently, image quality suffers. This does not imply that the rate-distortion curve will degrade, however, because fewer bits are required to specify domain block locations. Such trade-off situations are endemic to fractal image compression. It is difficult to know in advance what particular bit allocation scheme will produce superior results.

3.3 Restricted Area Search

Figure 4 shows three domain-range pairing for the Bird image, all obtained from a light brute force search. Consider the largest block pair, the one mapping a 64x64 section of background just left of the Bird's head to the 32x32 square in the bottom right corner. These are paired because the smooth gradient of the domain block most closely matches that of the range block. But notice that the area in the bottom right corner is also similar to the range block. Though it does not offer the best pairing available, a domain block taken from this area will still provide a good match.

Therefore, a reasonable way of reducing the encoding time complexity is to restrict the search to nearby areas. For example, the source image may be sectioned into four quadrants. For a range block in the bottom left quadrant, say, only domain blocks in that same quadrant are searched. As a result, the search time is reduced by a factor of four over a brute force search. In general, the time complexity is $O(nq)$, where n is the number of pixels in an image, and q is the number of pixels per quadrant.

It is important to realize that a restricted area search depends on an image possessing locality of similar form. This cannot be guaranteed. But it is sufficiently common that it may be exploited to good advantage.

3.4 Local Spiral Search

Taking the idea further, Beaumont suggested an outward spiral search originating from the current range block position [4], as illustrated in Figure 14. But instead of examining all candidate domain blocks for the best match, the search halts as soon as a sufficiently good match is found. In many cases the search time is dramatically reduced, albeit with some loss of image fidelity.

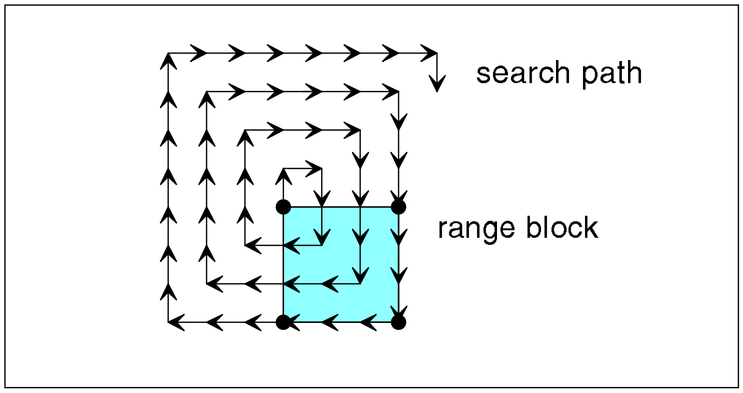


Figure 14. Local spiral search for a matching domain block from the current range block position.

3.5 Look Same Place

In the extreme, a local spiral search can be halted after examining the first domain block. That is, one looks in the same place as the range block, and nowhere else. (Blocks along the image boundary require special treatment.) If maximum speed is required, the eight symmetry operations are additionally ignored. As may be expected, image quality suffers dramatically. Research by Monro shows that only by adopting the bilinear approximation of (13) does quality become acceptable [36]. High fidelity may never be possible in this approach, but by being O(n) it does establish a lower bound on computation.

3.6 Categorized Search

Taking a different approach in [14,25] Fisher, Jacobs and Boss argue that the bulk of computation can be eliminated by categorizing blocks prior to comparison. If a range block contains a strong edge, for instance, searching for a good match among the multitude of smoothly varying domain blocks is wasted effort.

In their approach, each domain block is inserted into one of 72 categories. To do so, a block is first divided into four quadrants and oriented into “canonical position,” as shown in Figure 15. Once divided into these three major classes, the quadrants of each square are ordered from highest variance to lowest, for 4! = 24 possibilities within each class. A range block is also categorized in this manner. When seeking a matching domain block, only the corresponding category is searched. Because the entries are in canonical orientation, it is not necessary to test the eight symmetry operations.

A quantitative comparison of categorized search and local spiral search does exist in the primary literature, but it is believed that these two approaches represent the current state of the art in fractal image compression.

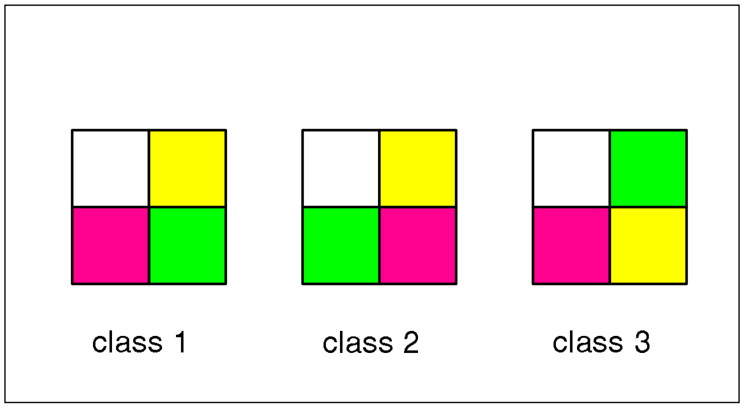


Figure 15. The three canonical orientations of a range or domain block. A block not in one of these forms can be so oriented by applying one of the eight isometric operations. The colors indicate a relative ranking of mean values of each quadrant.

4. FAST FRACTAL ALGORITHM

Fractal compression is unusual in that image quality improves with increased processing devoted to the encoding phase. It is also true that most computation can be averted by an appropriately selective algorithm. The algorithms outlined in Section 3 are progressive attempts in this direction. It should be apparent, however, that the task does not easily lend itself to an optimal solution and that all attempts so far are heuristically derived.

This section introduces the Fast Fractal Image Compression algorithm (FFIC), a novel approach to the speed problem (first presented in [30]). With possible multimedia applications in mind, the goal of this work has been to approximate the high speed of the Look Same Place algorithm (Section 3.5), while still sustaining acceptable image quality. Although not an optimal solution, experiments confirm that the FFIC algorithm is a significant advance. At comparable quality levels, it can be 25 to 400 times faster than the current state of the art.

4.1 Algorithm

To achieve computational efficiency the Fast Fractal algorithm employs two key techniques:

1. aggressive filtering of domain blocks by pixel variation, and
2. multi-dimensional indexing of the domain pool using the r-tree data structure.

These choices are motivated by the following considerations. First, fractal compression relies on an image possessing block-wise self-similarity. In some cases it is not present (see Figure 12). But when it does exist, self-similarity is duplicated throughout the images, e.g. in the background sky and grassy field of Cameraman. Therefore, if a few representative examples are selected, the bulk of repetitive domain-to-range block comparisons do not need to be made.

The second key observations is that, mathematically, a block of pixels compose a single entity. Namely, a *position vector* in an abstract *position space*, where each distinct point represents a different block. The location of a vector is derived from a scanline ordering of pixels within a block, i.e. $\mathbf{z} = (z_1, z_2, ..., z_n)$. When a distance metric is applied to this space the relative position of two vectors determines their closeness. Viewed this way, the search strategies of Section 3 are different ways of identifying position vectors in close proximity. The ideal search pattern, therefore, would directly address the location of a given range block (in this abstract position space), then select the closest neighbor. Practical implementation of this idea requires a multi-dimensional data structure capable of storing and indexing position vectors. The r-tree is one such data structure.

To investigate their merit, these ideas have been applied to the Jacquin approach of equation (8), where the offset parameter represents the mean value of range blocks. As an additional constraint, a four-level full quadtree partitioning is used in all tests.

4.1.1 Domain Pool Filtering

When using four-level partitioning, domain blocks are restricted to being {8, 16, 32, 64} pixels wide. This effectively divides the image into four disjoint domain pools, as illustrated in Figure 16. If blocks are selected at positions that are integral multiples of four, (similar to a light brute force search), each pool may contain the following number of entries.

$$\left(\frac{ImageWidth - DomainBlockSize}{4} - 1\right) \times \left(\frac{ImageHeight - DomainBlockSize}{4} - 1\right) \tag{18}$$

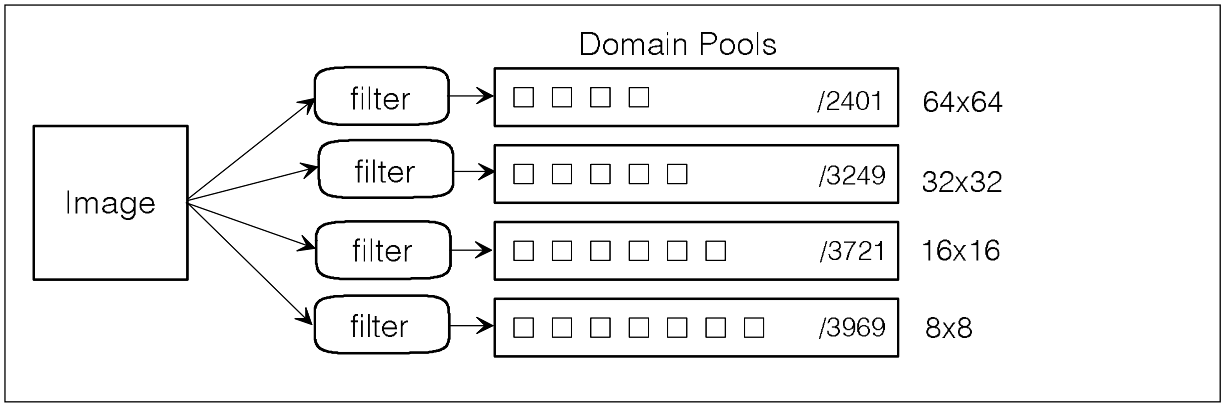


Figure 16. A four level partitioning structure results in four disjoint domain pools. For the test images used, the maximum number of domain blocks per pool is shown in the rectangles. A filter applied to the input streams restricts the total number of entries.

Because there are likely to be redundant entries, each pool can be restricted in number (e.g. to one hundred entries per pool) without seriously degrading the available variety. One possibility to perform a *k*-means clustering algorithm common to vector quantization methods, but the computational costs of doing so can be high [38]. For the sake of speed, filtering is based on the variance of pixel intensities within a block,

$$variance = \frac{1}{n}\sum_{i=1}^{n}(z_i - \bar{z})^2, \quad \bar{z} = \frac{1}{n}\sum_{i=1}^{n} z_i, \quad n = \text{number of pixels in domain block} \tag{19}$$

according to the following scheme:

- 64x64 blocks — all extracted from bottom 10% (smooth areas)
- 32x32 blocks — half from top 10% (edges), half from 20-50% range (moderate texture)
- 16x16 blocks — half from top 10% (edges), half from 50-80% range (moderate texture)
- 8x8 blocks — all extracted from top 10% (edges).

This heuristic is grounded on the observation that large blocks can successfully cover smooth areas of an image, while small blocks are necessary to cover sharp edges. The next step is to convert these selected blocks into position vectors.

4.1.2 Domain Block Preparation

The purpose populating an abstract position space is to facilitate the pairing of range blocks to domain blocks. But in a fractal codec they are related under affine transformations. This poses a complication because it is necessary to first normalize the blocks into standard form. The preparation step has four parts.

1. Domain blocks contain four times as many pixels as range blocks, and must be downsampled (by averaging 2x2 pixel cells) for a legitimate comparison. For example, 8x8 blocks become 4x4. Otherwise, domain blocks would reside in 64 dimensional space, and range blocks in 16 dimensional space.
2. A domain block may undergo one of the eight symmetry operations, an action that alters its location in position space. Therefore, all blocks need to be oriented in canonical form, as described in Section 3.6.
3. In order to compensate for the effect of the offset parameter, o_i, all blocks must be normalized to the same average value.
4. In order to compensate for the effect of the scale parameter, s_i, all blocks must be normalized to the same variance level (provided that it is larger than zero).

By performing these four steps, neighboring vectors in position space represent pixels blocks that are similar in form, as related by the affine transformation of equation (8). Brute force methods can be used to identify neighboring position vectors, but the r-tree data structure accelerates the process.

4.1.3 R-Tree Facilitated Search

The r-tree (and its close cousin the r*-tree) is a data structure capable of efficiently indexing a multi-dimensional data space. The r-tree is not well known but can be considered an extension of the more familiar b-tree. Full accounts are found in [10,21].

To contrast their operation, consider the task of indexing words from a dictionary. Using a b-tree, words are arranged in lexicographic order and accessed via a binary search. The binary tree is *balanced* (hence the name) so that each branch partitions the list into two nearly equal parts. The b-tree is useful for indexing a one-dimensional data set.

When using an r-tree, words are considered *n*-dimensional vectors, based upon the first *n* letters. A useful value of *n* is sixteen since few words, if any, are identical past sixteen letters. Each location in this 16-dimensional position space represents a unique arrangement of letters — there are $256^{16} = 2^{128}$ possibilities — a small percentage of which are valid words. Since the data space is not one-dimensional it cannot be accessed using a binary search. Instead, position vectors are organized into a nested set of bounding *rectangles* (hence the name) with words of close spelling clustered together.

Figure 17 illustrates the operation in two dimensions. Ten two-letter vectors are arranged in a two-dimensional space, as shown in the left graph. These are organized into separate rectangles, with ***R2*** and ***R3*** nested inside ***R1***. Because, in this example, each rectangle can hold a maximum of five nodes, the addition of vector ***im*** cases ***R3*** to be split in two. It is desirable that rectangles do not overlap, although it can happen. The r-tree and r*-tree differ primarily in their splitting strategy: minimizing the total volume (area) contained within the bounding rectangles is the strategy adopted by r-tree algorithms.

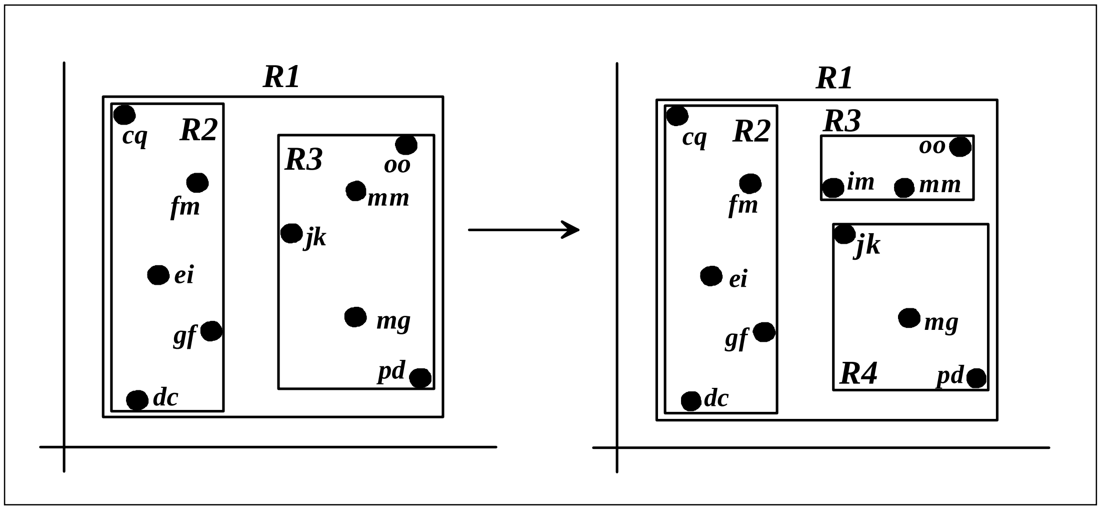


Figure 17. One organization of two-dimensional position vectors into an r-tree data structure. The first letter of each label corresponds the x-axis, and the second letter to the y-axis. The splitting of rectangle ***R3*** is caused by the addition of vector ***im***.

The application to fractal compression is similar. A 16-dimensional r-tree is associated with each domain pool in Figure 16. Domain blocks are reduced to 4x4 pixels in size and normalized as described in the preceding subsection. These are converted to position vectors by sequencing the pixels in scanline order. Instead of the ASCII letter value denoting the position along an axis, it is the grayscale value of a pixel. Thus, the group of four r-trees index all the candidate domain blocks. Range blocks are not inserted into the structure, but are used as search keys.

Continuing with the current example, the search process is illustrated in Figure 18. The X marks the location of a particular range block in the corresponding position space, after having been normalized. The circles represent domain blocks. In finding the nearest domain block the search algorithm narrows in from ***R1*** to ***R4***, tests every element in ***R***4, and returns ***jk*** as the closest match. The average search time is $\propto n\log_m(n)$ where n is the number of domain blocks inserted and m limits the number of elements contained in any one rectangle.

From the standpoint of theory, the root mean square error is the distance metric of choice (as dictated by equation (11)), but mean absolute error is a viable alternative. And because it can be calculated in sixteen addition operations (as opposed to sixteen multipy/addition operations for root mean square error), it was employed in the current implementation.

Notice that the vector ***mm*** in the separate bounding rectangle ***R3*** is an equally good match, but is missed by the search. Unfortunately, locating the optimal pairing is not guaranteed. But unlike other algorithms that examine only those domain blocks *physically* close to a particular range block, the FFIC algorithm examines those that are *structurally* close. This allows the image quality to remain reasonably high, while reducing the search time dramatically.

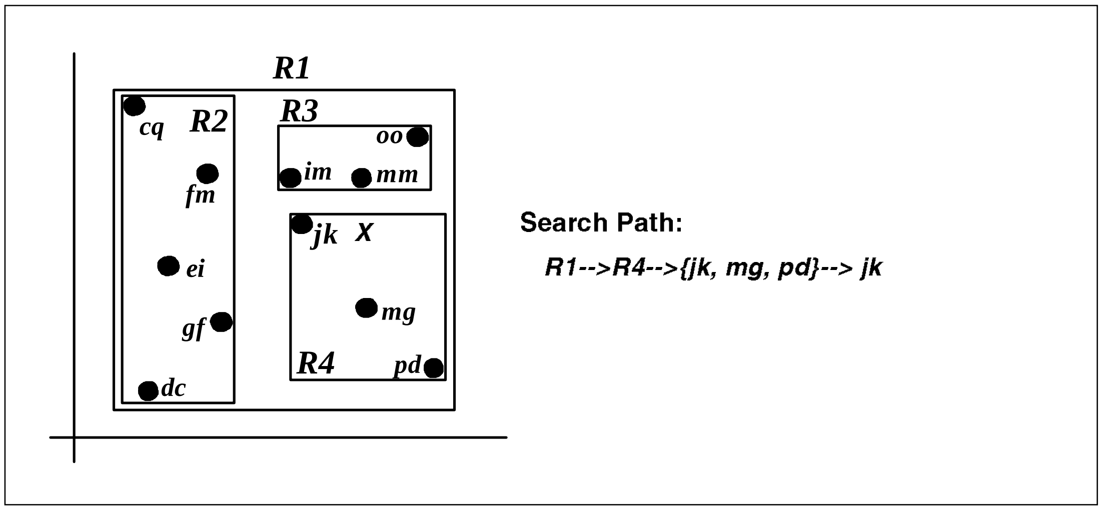


Figure 18. The r-tree facilitated search process. The X marks the location of the current range block and the circles mark candidate domain blocks. The point labeled ***jk*** is returned as the closest match.

4.2 Similar Methods

Although the FFIC algorithm of Section 4.1 is new, somewhat similar ideas have recently been reported by other researchers. In a purely theoretical paper, Saupe considers the one-dimensional case of functions (instead of images), and proves that if pixel blocks are treated as position vectors, then the search for best pairings can be performed in log-linear time [43]. Although not mathematically proven, the encoding of one-dimensional functions carries over readily to images, and the author suggests using *k*-d trees as an indexing mechanism. It would be interesting to compare the efficiency of *k*-d trees to r-trees.

As reported in a recent letter to *Electronic Imaging*, Fryer *et al*. may have implemented Saupe's idea. The letter is terse, saying only that their approach "applies indexing theory to reduce the number of tests to a small number per tile." For one 640x400 pixel grayscale image, they cite compression times of 31.9 to 38.1 seconds on a SPARC II workstation [20].

Finally, Frigaard *et al*. introduce a two-dimensional *feature space* for improving search speed. A domain block is positioned in this space according to pixel variance (equivalently, standard deviation), and by the number of dominant gray levels in the block. To determine this latter quantity, a histogram of pixel values is formed for each domain block. The number of histogram entries above a certain threshold (e.g. 10%) gives the number of dominant gray levels. The authors do not specify an indexing mechanism, but do present data showing a 250 second compression time on a SPARC workstation for the 512x480 grayscale version the "Lena" test image [19].

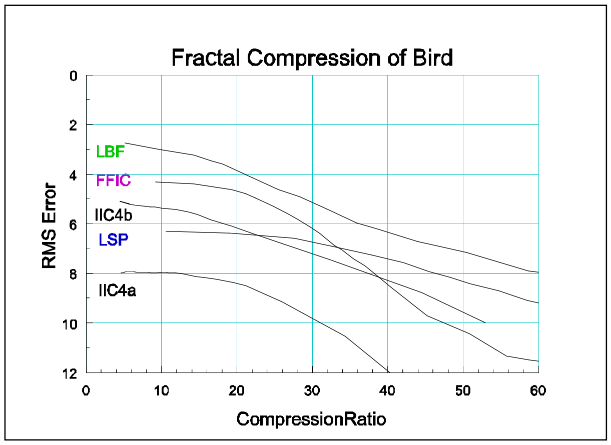
4.3 FFIC Results

When devising a new compression algorithm, numerous and often contradictory goals challenge the researcher. Four such goals are: high compression ratios, high image quality, low compression times, and near-linear scalability. Seldom can one be improved without adversely affecting another. The main objective of this work has been to minimize compression times while attempting to retain the other three goals.

To evaluate the success of the FFIC, it is compared to four other algorithms. At the extremes they are: Look Same Place, to establish a lower bound on computation time, and Light Brute Force, to establish an upper bound on image quality. As a "stress-test," a commercially available fractal compression program — referred to here as IIC4[1] — is taken to represent the current state of the art. This program was run at two different settings: one optimized for speed (at the expense of quality) and one optimized for quality (at the expense of speed). All tests were run on a 486DX2-66 personal computer, with 32 MB main memory. Measurements of the commercial program necessitated hand timing; other timings are based on a system clock with single millisecond accuracy. Timing is halted when all range-domain block pairings are identified. Times do not include writing the compressed file to disk.

Figure 19 compares image quality as a function of compression ratio for the 256x256 Bird and Cameraman test images. As expected, the brute force algorithm produces the most favorable curve. Interestingly, the least favorable curve is not Look Same Place, but the high speed setting of IIC4. Because it is not necessary to store the domain block positions in LSP (they are implicitly known), the loss in quality is partly compensated by reduced storage requirements.

[1] Short for *Images Incorporated*, version 4.0, one of a family of products available from Iterated Systems Incorporated, Norcross, GA. For the sake of reproducibility, the setting used were as follows. A) Speed over quality: HiRes DLL, extended precision, setting of Fair, pool size of 256x256, archive bit off. B) Quality over speed: HiRes DLL, extended precision, setting of Best, pool size of 512x400, archive bit on.



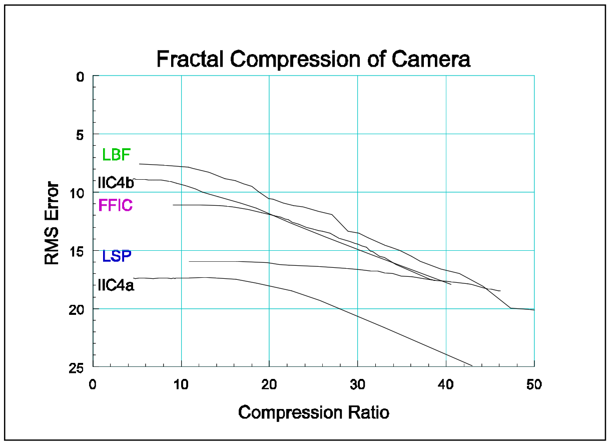
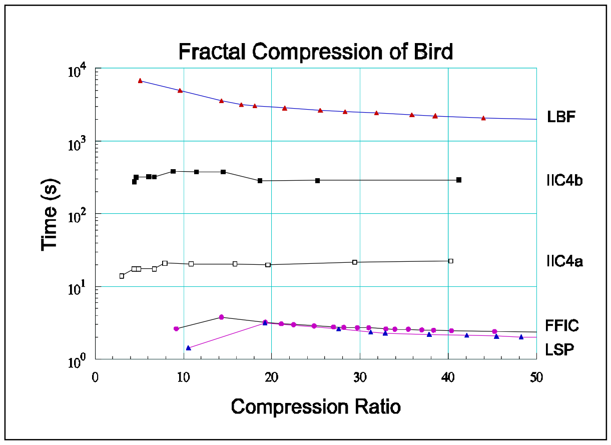


Figure 19. Rate-distortion curves for five different fractal compression algorithms.
LBF: Light Brute Force, domain block positions are integral multiples of four.
FFIC: Fast Fractal Image Compression algorithm.
LSP: Look Same Place algorithm.
IIC4: Images Incorporated, version 4.0: a) set for fastest speed, b) set for best quality.



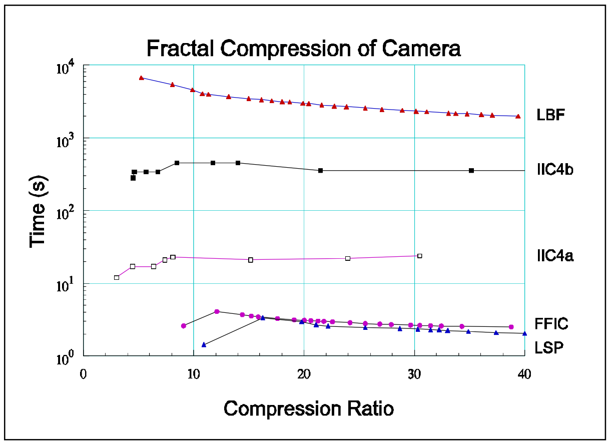


Figure 20. Compression times as a function of compression ratio. Note that the time scale is logarithmic.

	128x128			256x256			512x512		
	IIC4b	FFIC	speedup	IIC4b	FFIC	speedup	IIC4b	FFIC	speedup
Airplane	45.5	1.34	34.0	340	3.38	100.6	1120	9.53	117.5
Fox	76.0	1.34	56.7	878	3.49	251.6	4840	11.20	432.2
Goldhill	52.0	1.32	39.4	367	3.35	109.6	1028	10.32	99.6
Lena	37.5	1.34	28.0	263	3.43	76.7	1335	9.48	140.8
Peppers	37.0	1.43	25.9	270	3.52	76.7	1379	10.32	133.6

Table 3. Effect of image size on compression, and relative speedup of FFIC.
All times are in seconds. The third columns are defined by speedup = $\frac{\text{IIC4b}}{\text{FFIC}}$.

Judging by the curves in Figure 19, the rate-distortion characteristics of FFIC are comparable to IIC4 when set to produce the highest quality compression (IIC4b curve). On the Bird image, the root mean square error is lower for FFIC up to 40:1. On the Cameraman image, the error is somewhat greater up to 20:1; beyond that point the curves nearly overlap. For a visual comparison, some representative samples are shown in Figures 21-22. These close-ups are extracted from images compressed at 30:1, a level where coding defects become apparent.

Figure 20 compares encoding times as a function of compression ratio. The Look Same Place requires the least computation, achieving times in the 2-3 second range. FFIC follows closely behind with times in the 2-4 second range. Both algorithms employ a four-level full quadtree partitioning. When evaluating range-domain block pairing, the LSP algorithm tests all eight symmetry operations explicitly. The FFIC algorithm does not do so, relying instead on the normalization procedure described in Section 4.2.2. Taking much longer to complete, the Light Brute Force algorithm produces times in excess of an hour.

(Aside: The left end of the time-compression curves can exhibit a sharp downward turn. This occurs when the error tolerance controlling the partitioning process is set to zero. The corresponding quadtree thus degenerates to an even partitioning of 4x4 range blocks. Anticipating this, some algorithms proceed directly to this stage.)

The compression times of FFIC and IIC4 are also compared in Table 3. Considering that the quality achieved by these two algorithms is competitive, the comparison is rather dramatic. The relative speedup ranges from 25:1 (128x128 version of Peppers) to over 400:1 (512x512 version of Fox). This proves that proper characterization and indexing of pixel blocks can eliminate the bulk of unnecessary computation. Perhaps most impressive, the FFIC algorithm exhibits a log-linear time complexity with a *sub-normal* constant coefficient. This requires some explanation.

Compression times for the Fast Fractal algorithm can be divided into three stages. In the *setup* stage, candidate domain blocks are extracted from the input image with useful information (e.g. variance values) calculated in advance. In the *insertion* stage, the domain blocks are filtered according to variance values and inserted into the respective r-tree structures. In the *search* stage, the r-trees are used to find a matching domain block for each range block. Representative times are contained in Table 4. The setup stage is precisely linear with image size, as expected. But due to filtering of the domain pools, the insertion and search times increase by *less* than a factor of four as the image size quadruples.

The net result is this. Using the FFIC algorithm, fractal compression has become a nearly time-symmetric process. This opens opportunities for multimedia applications.

Lena	128x18	256x256	512x512
Setup	0.11	0.43	1.72
Insertion	0.63	1.24	3.44
Search	0.60	1.76	4.32
Total (seconds)	1.34	3.43	9.48

Table 4. Breakdown of FFIC compression times into three stages.
The test image is Lena at three different resolutions.

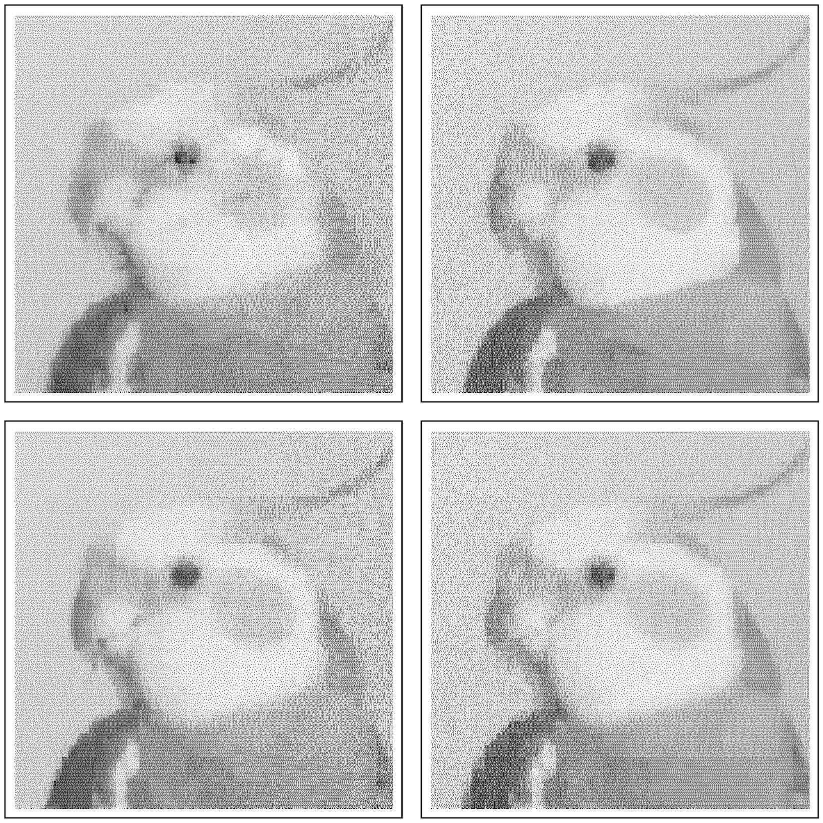


Figure 21. Detail of Bird at 30:1.
(a) left: IIC4a, rmse = 10.53.
(b) right: IIC4b, rmse = 7.47.

(c) left: FFIC, rmse = 6.12.
(d) right: LSP, rmse = 6.81.

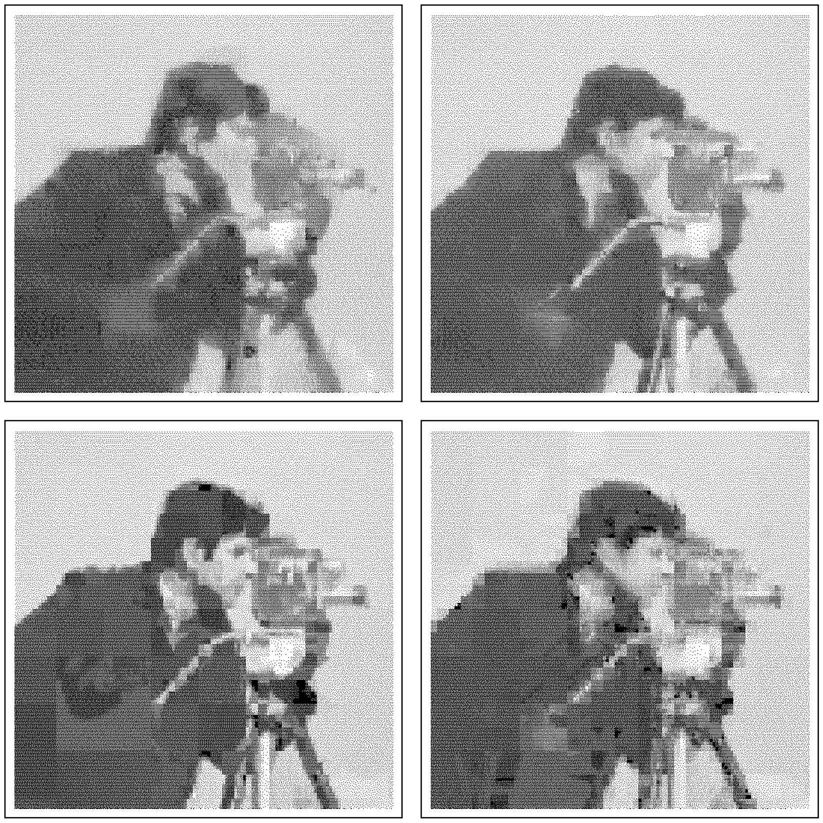


Figure 22. Detail of Camerman at 30:1.
(a) left: IIC4a, rmse = 21.61.
(b) right: IIC4b, rmse = 15.33.

(c) left: FFIC, rmse = 14.45.
(d) right: LSP, rmse = 16.67.

5. MULTIMEDIA APPLICATIONS

Multimedia is distinguished by the integration of sound, images, and video into the computing environment. Potential applications include electronic documents, video sequences on CD-ROM, and live PC-to-PC video communications. Two qualities of fractal compression make it worth investigating. First, the potential for high quality at low bitrates. And second, the flexibility offered by fractal interpolation.

It is too early tell whether fractal compression will gain widespread use in multimedia applications, or secure niche markets, or be rejected entirely in favor of something else. As a guide to the problem area, the following subsection offers a quick survey of recent work.

5.1 Existing Work

Fractal compression has already been applied to the domain of electronic newspapers and on-line image databases (e.g. for use in real-estate catalogs) [3]. Typically, a central server controls the image repository and is responsible for compression and archiving. Decompression/viewing software resides on distributed client machines. However, user's of existing programs note that the lengthy compression times are a hindrance [45].

In extending fractal techniques from stills to moving pictures, different approaches may be taken. The simplest is to encode every frame individually. In MPEG parlance, every frame is an intraframe. Unfortunately, the very low bitrates demanded by video applications cannot usually be met by this approach. A second option is treat time as an extra spatial dimension. A video sequence is then compressed (and decompressed) in whole, as if it were volumetric data. This demands huge amounts of memory to serve as a frame store. In addition, the results of early investigations into volumetric compression are not encouraging, both with regards to image quality and computational requirements [11]. Beaumont tried to reduce the memory requirements by breaking a video sequence into "slabs" of twelve frames each [5]. Range blocks were used to tile each frame in 4x4 sections, and, unique to this attempt, 12x12 domain blocks were aligned along the *temporal* axis. Unfortunately, the compressed sequences exhibited annoying flicker effects along moving edges.

Taking a different direction, Ali employs fractal techniques to encoded each line of video separately as one-dimensional data [1]. The hope is that by drastically reducing the frame store size (each line is treated in sequence), hardware-based video coders can be inexpensively manufactured. This strategy may not succeed, however, because experimental results show significant blurring in the scanline direction [2].

The most viable approach is to code frames sequentially, with conditional block replenishment. In other words, range blocks of the n*th* frame are matched to domain blocks of the n-1*th* frame. Only the *k* blocks with the highest interframe error are transmitted, where *k* is bounded by the channel capacity [13,18,23,34,41]. Since the entire image is not repeatedly refreshed in whole, the output stream does not have a definable frame rate. In most implementations the decoding can be performed in real time, but the encoding stage is computationally expensive. One system has been developed into a commercial product [22] but as yet no quantitative analysis is available in the literature.

In a noteworthy development, Monro and Nicholls report an optimized system that achieves real time playback and encoding on 486DX-33 PCs [37]. They report encoding times of 250 μs per pixel, and decoding times of 200 μs per pixel. To achieve such high speed they apply the Look Same Place algorithm to the bilinear fractal transform, i.e. equation (13). Good voice/image synchronization is reported, but the system suffers dramatic distortions during abrupt scene changes.

5.2 Prospects for FFIC-Accelerated Video

The Fast Fractal algorithm has yet to be applied to video sequences. To assess this possibility, a “back of the envelope” calculation can be made [32]. Suppose that the video frame is 640x480 pixels in size. Fractal compression permits this to be downsampled to 320x240 at the sending end, and fractal interpolated up at the receiving end without introducing gross distortions. For the sake of argument, let the range blocks be 8x8 pixels in size, as is the case with MPEG, so that there are 1200 range blocks per frame. The times presented in Table 3 correspond to about 600 range-domain pairings per second on a 486DX2-66, or 20 per 1/30*th* of a second. Typical teleconferencing video, however, requires approximately 60-100 block replenishments per input frame in order to maintain good fidelity. So as it stands, the current implementation is too slow by a factor of three to five.

This shortfall is overcome, obviously, on computers with greater processing power. A 100 MHz Pentium should prove adequate to the task. More significantly, the FFIC timing results are not minimal. Great effort has been devoted to refining the *algorithm*, but comparatively little has gone towards optimizing the algorithm’s *critical code*. With suitable attention to detail, FFIC may potentially serve as the basis for real time PC-to-PC video communications.

6. CONCLUSION

Since Jacquin’s breakthrough in fractal compression, the principle goal has been to simultaneously maximum speed and quality. While still an open challenge, the Fast Fractal Image Compression algorithm is a significant advance in this direction.

Some aspects are worthy of further investigations. Alternative partitioning structures, especially HV partitioning, need to be compared to the full quadtree decomposition used in this work. Second, the domain pool filters may be refined, or based on some other quantity than block variances. Third, there may be merit in designing a hybrid algorithm by combining FFIC with a Local Spiral Search. And fourth, the extension to bilinear fractal transforms certainly seems worthwhile.

All of these issues involve uncertain trade-offs between speed and quality and algorithmic complexity. Such is the nature of the field.

ACKNOWLEDGMENTS

Figure 1 and Figure 3 are from Dick Oliver’ *Fractal Vision* software program. Figures 12 and 13b are from Iterated System’ *Images Incorporated*, version 4.0. Figure 13a is from *Images Incorporated*, version 3.1. All other images are from the author’s work unless otherwise noted.

REFERENCES

1. Maaruf Ali, Costas Papadopoulos, Trevor Clarkson; “The Use of Fractal Theory in a Video Compression System,” *IEEE Data Compression Conference ‘92*, James Storer (editor), pp. 259-268, 1992.
2. Maaruf Ali, Trevor Clarkson; “Using Linear Fractal Interpolation Functions to Compress Video Images,” *Fractals — An Interdisciplinary Journal On The Complex Geometry of Nature*, vol. 2, no. 3, pp. 417-421, 1994.
3. Nicholas Baron; “Fractal Compression Goes On-Line,” *BYTE*, p. 40, September 1993.
4. J. Mark Beaumont; “Advances in Block Based Fractal Coding of Still Pictures,” *Proceedings of IEEE Colloquium: The Application of Fractal Techniques in Image Processing*, pp. 3.1-3.6, 1990.
5. J. Mark Beaumont; “Image Data Compression Using Fractal Techniques,” *British Telecom Technology Journal*, vol. 9, no. 4, pp. 93-109, 1992.
6. Michael Barnsley; *Fractals Everywhere*, Academic Press, 1988.
7. Michael Barnsley, Arnaud Jacquin, Francois Malassenet, Laurie Reuter, Alan Sloan; “Harnessing Chaos for Image Synthesis,” *SIGGRAPH ‘88 Computer Graphics Conference Proceedings*, vol. 22, no. 4, pp. 131-140, 1988.
8. Michael Barnsley, Alan Sloan; “Methods and Apparatus for Image Compression by Iterated Function Systems,” U.S. patent no. 4,941,193, July 10, 1990.
9. Michael Barnsley, Alan Sloan; “Method and Apparatus for Processing Digital Data,” U.S. patent no. 5,065,447, November 12, 1991.
10. Norbert Beckmann, Hans-Peter Kriegel, Ralf Schneider, Bernhard Seeger; “The R*-tree: An Efficient and Robust Access Method for Points and Rectangles,” *Proceedings of the ACM SIGMOD Conference on Management of Data*, vol. 19, no. 2, pp. 322-331, 1990.
11. Wayne Cochran, John Hart, Patrick Flynn; “Fractal Volume Compression,” Internal Report, Washington State University, School of EECS, pp. 1-27, 1994.
12. Franck Davoine, J. Chassery; “Adaptive Delaunay Triangulation for Attractor Image Coding,”*12th International Conference on Pattern Recognition*, October 1994.
13. Sergio de Faria, Mohamnmad Ghanbari; “Variable Block Size Fractal Video Coding with Spatial Transform Motion Compensation,” *PCS ‘94 — Proceedings of the International Picture Coding Symposium*, pp. 437-439, 1994.
14. Yuval Fisher, Bill Jacobs, Roger Boss; “Iterated Transform Image Compression,” NOSC Technical Report 1408, Naval Ocean Systems Center, San Diego, CA, pp. 1-27, April 1991.
15. Yuval Fisher, Bill Jacobs, Roger Boss; “Fractal Image Compression Using Iterated Transforms,” in *Image and Text Compression*, edited by James Storer, Kluwer Academic Publishers, pp. 35-61, 1992.
16. Yuval Fisher, editor; *Fractal Image Compression — Theory and Application to Digital Images*, Springer-Verlag, 1994.
17. Bruno Forte, Edward Vrscay; “Solving the Inverse Problem for Function and Image Approximation Using Iterated Function systems,” to appear, 1995.
18. R. Franich, R. Lagendijk, J. Biemond; “Fractal Picture Sequence Docing: Looking for the Effective Search,” *PCS ‘94 — Proceedings of the International Picture Coding Symposium*, pp. 433-436, 1994.

19. Carsten Frigaard, Jess Gade, Thomas Hemmingsen, Torben Sand; "Image Compression Based on a Fractal Theory," internal report, Aalborg University, Denmark, pp. 1-10, 1994.
20. R. Fryer, D. McGregor, P. Cockshott, P. Murray; "Approaches to Real-Time Fractal Image Compression," *Electronic Imaging*, vol. 4, no. 2, pp. 1+, July 1994.
21. A. Guttman; "R-trees: A Dynamic Index Structure for Spatial Searching," *Proceedings of ACM SIGMOD Conference on Management of Data,* pp. 47-57, 1984.
22. Lyman Hurd, M. Gustavus, Michael Barnsley; "Fractal Video Compression," *Proceedings of the Thirty-seventh IEEE Computer Society International Conference (COMPCOM),* vol. 37, pp. 41-42, 1992.
23. B. Hurtgen, P. Buttgen; "Fractal Approach to Low-Rate Video Coding," *SPIE Visual Communications and Image Processing,* vol. 2094, 1993.
24. John Hutchinson; "Fractals and Self-Similarity," *Indiana University Mathematics Journal*, vol. 30, no. 5, pp. 713-747, 1981.
25. Bill Jacobs, Roger Boss, Yuval Fisher; "Image Compression: A Study of the Iterated Transform Method," *Signal Processing,* vol. 29, pp. 251-263, 1992.
26. Arnaud Jacquin; "A Fractal Theory of Iterated Markov Operators with Applications to Digital Image Coding," Ph.D. Thesis, Georgia Institute of Technology, Atlanta, GA, 1989.
27. Arnaud Jacquin; "Image Coding Based on a Fractal Theory of Iterated Contractive Image Transformations," *IEEE Transactions on Image Processing,* vol. 1, no. 1, January 1992, pp. 18-30.
28. Arnaud Jacquin; "Fractal Image Coding: A Review," *Proceedings of the IEEE*, vol. 81, no. 10, pp. 1451-1465, 1993.
29. Rick Kazman, John Kominek; "Information Organization in Multimedia Resources," *Conference Proceedings of SIGDOC '93*, pp. 149-162, 1993.
30. John Kominek; "Understanding Fractal Image Compression," University of Waterloo internal report, pp. 1-52, 1993.
31. John Kominek; "Still Image Compression: An Issue of Quality," University of Waterloo Technical Report, 1994.
32. John Kominek; "Algorithm for Fast Fractal Image Compression," *Conference Proceedings of Digital Video Compression: Algorithms and Technologies*, to appear, February 1995.
33. John Kominek; "Convergence of Fractal Encoded Images," *Data Compression Conference '95*, to appear, 1995.
34. Haibo Li, Mirek Novak, Robert Forchheimer; "Fractal-Based Image Sequence Compression Scheme," *Optical Engineering*, vol. 32, no. 7, pp. 1588-1595, 1993.
35. Benoit Mandelbrot; *The Fractal Geometry of Nature*, W.H. Freeman & Co., second edition, 1983.
36. Donald Monro, David Wilson, Jeremy Nicholls; "High Speed Image Coding with the Bath Fractal Transform," *IEEE International Symposium on Multimedia Technologies*, April 1993.
37. Donald Monro, Jeremy Nicholls; "Real Time Fractal Video for Personal Communications," *Fractals — An Interdisciplinary Journal On The Complex Geometry of Nature*, vol. 2, no. 3, pp. 391-394, 1994.
38. Nasser Nasrabadi, Robert King; "Image Coding Using Vector Quantization: A Review," *IEEE Transactions on Communications*, vol. 36, no. 8, pp. 957-971, 1988.

39. Dick Oliver; *Fractal Vision: Put Fractals to Work for You*, Sams Publishing, pp. 293-297, 1992.

40. Heinz-Otto Peitgen, Martmat Jurgens, Dietmar Saupe; "Encoding Images by Simple Transformations," *SIGGRAPH '90 Course Notes, Fractals: Analysis and Modeling*, vol. 15, chapter 11, pp. 1-21, 1990.

41. E. Reussens; "Sequence Coding Based on the Fractal Theory of Iterated Transformation Systems," *SPIE Visual Communications and Image Processing*, vol. 2094, 1993.

42. Arturo Rodriguez; "Video Codecs in Software Multimedia Environments," *Proceedings of the International Picture Coding Symposium, PCS '94*, pp. 58-61, 1994.

43. Dietmar Saupe; "Breaking the Time Complexity of Fractal Image Compression," Internal Report, Institut fur Informatik, University of Freiburg, 1994. (A condensed version appears in [16].)

44. Richard Schultz, Rovert Stevenson; "A Bayesian Approach to Image Expansion for Improved Definition," *IEEE Transactions on Image Processing*, vol. 3, no. 3, pp. 233-242, 1994

45. Jim Thompson; "Fracterm — A Vision of the Future for the On-Line World," *Boardwatch*, pp. 27-33, July 1993.

46. Edward Vrscay; "Iterated Function Systems: Theory, Applications and the Inverse Problem," in *Fractal Geometry and Analysis*, Jacques Belair, Serge Dubuc (editors), pp. 405-468, 1991.

47. Gregory Wallace; "The JPEG Still Picture Compression Standard," *Communications of the ACM*, vol. 34, no. 4, April 1991, pp. 30-44.