An Introduction to Fractal Image Compression

Literature Number: BPRA065
Texas Instruments Europe
October 1997

IMPORTANT NOTICE

Texas Instruments (TI) reserves the right to make changes to its products or to discontinue any semiconductor product or service without notice, and advises its customers to obtain the latest version of relevant information to verify, before placing orders, that the information being relied on is current.

TI warrants performance of its semiconductor products and related software to the specifications applicable at the time of sale in accordance with TI's standard warranty. Testing and other quality control techniques are utilized to the extent TI deems necessary to support this warranty. Specific testing of all parameters of each device is not necessarily performed, except those mandated by government requirements.

Certain applications using semiconductor products may involve potential risks of death, personal injury, or severe property or environmental damage ("Critical Applications").

TI SEMICONDUCTOR PRODUCTS ARE NOT DESIGNED, INTENDED, AUTHORIZED, OR WARRANTED TO BE SUITABLE FOR USE IN LIFE-SUPPORT APPLICATIONS, DEVICES OR SYSTEMS OR OTHER CRITICAL APPLICATIONS.

Inclusion of TI products in such applications is understood to be fully at the risk of the customer. Use of TI products in such applications requires the written approval of an appropriate TI officer. Questions concerning potential risk applications should be directed to TI through a local SC sales office.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards should be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance, customer product design, software performance, or infringement of patents or services described herein. Nor does TI warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used.

Contents

List of Figures

An Introduction to Fractal Image Compression

ABSTRACT

This paper gives and introduction on Image Coding based on *Fractals* and develops a simple algorithm to be used as a reference design.

The Fractal Coding System described in this project was developed from the ideas proposed by Arnaud Jacquin in his paper published in IEEE Trans. “Image Coding Based on Fractal Theory of Iterated Contractive Image Transformation”

Two implementations of the proposed Fractal Encoding Technique have been developed. One using an ELF TMS320C31 board from SPOX by Rahmi Hezar (PhD student at Georgia Institute of Technology) and the second using MATLAB DSP Simulator by Alex Candela (Field Application Engineer at Texas Instruments)

1. Introduction

With the advance of the information age the need for mass information storage and fast communication links grows. Storing images in less memory leads to a direct reduction in storage cost and faster data transmissions. These facts justify the efforts, of private companies and universities, on new image compression algorithms.

Images are stored on computers as collections of bits (a bit is a binary unit of information which can answer “yes” or “no” questions) representing pixels or points forming the picture elements. Since the human eye can process large amounts of information (some 8 million bits), many pixels are required to store moderate quality images. These bits provide the “yes” and “no” answers to the 8 million questions that determine the image.

Most data contains some amount of redundancy, which can sometimes be removed for storage and replaced for recovery, but this redundancy does not lead to high compression ratios. An image can be changed in many ways that are either not detectable by the human eye or do not contribute to the degradation of the image.

The standard methods of image compression come in several varieties. The current most popular method relies on eliminating high frequency components of the signal by storing only the low frequency components (Discrete Cosine Transform Algorithm). This method is used on JPEG (still images), MPEG (motion video images), H.261 (Video Telephony on ISDN lines), and H.263 (Video Telephony on PSTN lines) compression algorithms.

Fractal Compression was first promoted by M.Barnsley, who founded a company based on fractal image compression technology but who has not released details of his scheme. The first public scheme was due to E.Jacobs and R.Boss of the Naval Ocean Systems Center in San Diego who used regular partitioning and classification of curve segments in order to compress random fractal curves (such as political boundaries) in

two dimensions [BJ], [JBJ]. A doctoral student of Barnsley's, A. Jacquin, was the first to publish a similar fractal image compression scheme [J].

2. What is Fractal Image Compression?

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

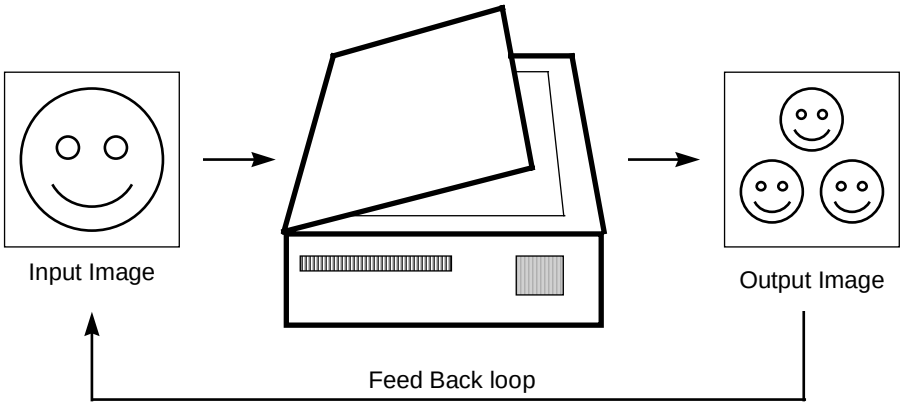

Figure 1: A copy machine that makes three reduced copies of the input image [Y]

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

In practice, choosing transformations of the form

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

is sufficient to generate interesting transformations called *affine transformations* of the plane. Each can skew, stretch, rotate, scale and translate an input image.

A common feature of these transformations that run in a loop back mode is that for a given initial image each image is formed from a transformed (and reduced) copies of itself, and hence it must have detail at every scale. That is, the images are *fractals*. This method of generating fractals is due to John Hutchinson [H], and more information about the various ways of generating such fractals can be found in books by Barnsley [B] and Peitgen, Saupe, and Jurgens [P1, P2].

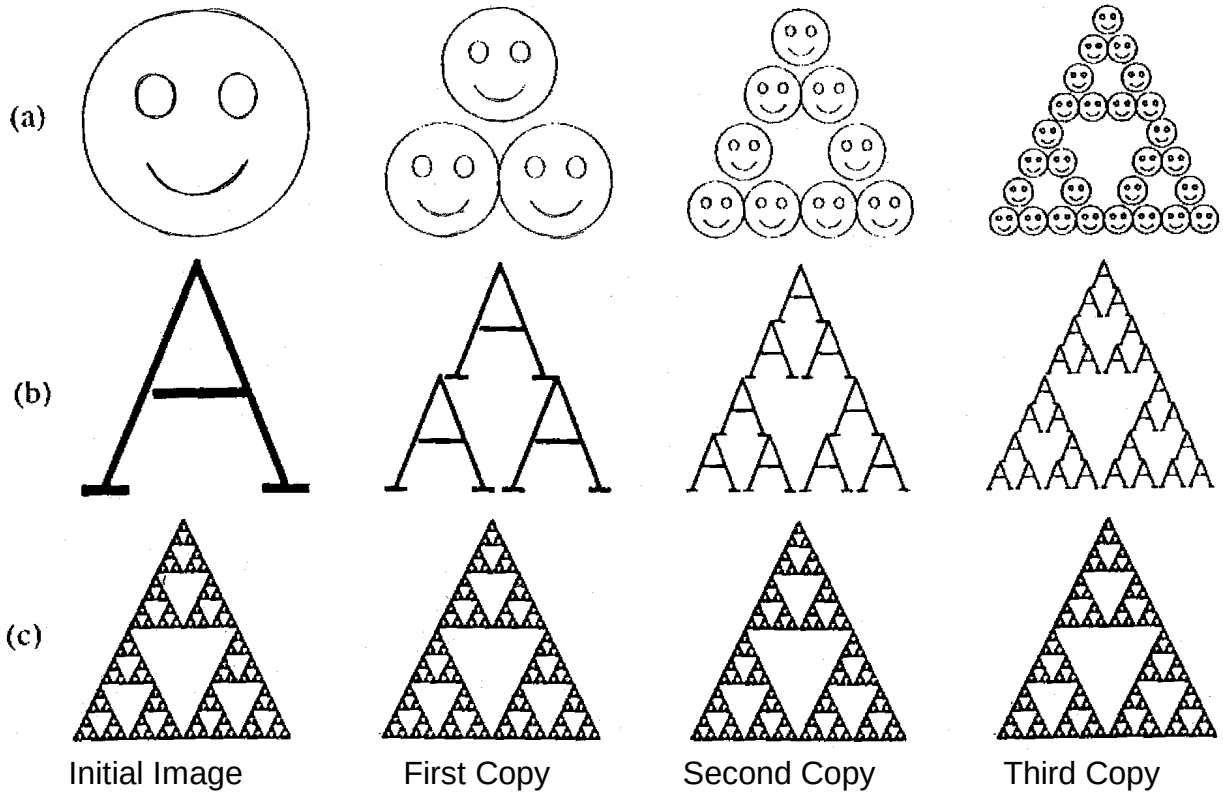

Figure 2: The first three copies generated on the copying machine Figure 1. [Y]

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

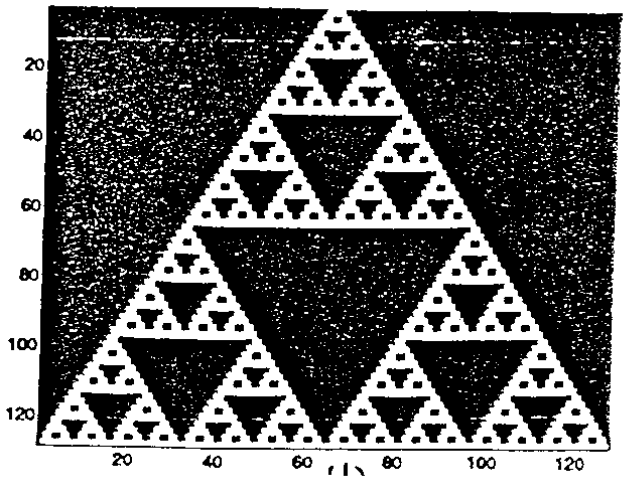

Figure 3: Fractal Fern

Each transformation w_i is defined by 6 numbers, a_i, b_i, c_i, d_i, e_i, and f_i, see eq(1), which do not require much memory to store on a computer (4 transformations x 6 numbers / transformations x 32 bits /number = 768 bits). Storing the image as a collection of pixels, however required much more memory (at least 65,536 bits for the resolution shown in Figure 2). So if we wish to store a picture of a fern, then we can do it by storing the numbers that define the affine transformations and simply generate the fern whenever we want to see it. Now suppose that we were given any arbitrary image, say a face. If a small number of affine transformations could generate that face, then it too could be stored compactly. The trick is finding those numbers.

Contractive Transformations

A transformation w is said to be contractive if for any two points $P1$, $P2$, the distance

$$d(w(P1),w(P2)) < sd(P1,P2)$$

for some $s < 1$, where d = distance. This formula says the application of a contractive map always brings points closer together (by some factor less than 1).

The Contractive Mapping Fixed Point Theorem

This theorem says something that is intuitively obvious: if a transformation is contractive then when applied repeatedly starting with any initial point, we converge to a unique fixed point.

If X is a complete metric space and $W: X \rightarrow X$ is contractive, then W has a unique fixed point $|W|$.

This simple looking theorem tells us how we can expect a collection of transformations to define an image.

3. Why the name “Fractal”

The image compression scheme describe later can be said to be fractal in several senses. The scheme will encode an image as a collection of transforms that are very similar to the copy machine metaphor. Just as the fern has detail at every scale, so does the image reconstructed from the transforms. The decoded image has no natural size, it can be decoded at any size. The extra detail needed for decoding at larger sizes is generated automatically by the encoding transforms. One may wonder if this detail is “real”; we could decode an image of a person increasing the size with each iteration, and eventually see skin cells or perhaps atoms. The answer is, of course, no. The detail is not at all related to the actual detail present when the image was digitized; it is just the product of the encoding transforms which originally only encoded the large-scale features. However, in some cases the detail is realistic at low magnifications, and this can be useful in Security and Medical Imaging applications. Figure 4 shows a detail from a fractal encoding of “Lena” along with a magnification of the original.

4. How much Compression can Fractal achieve?

The compression ratio for the fractal scheme is hard to measure since the image can be decoded at any scale. For example, the decoded image in Figure 3 is a portion of a 5.7 to 1 compression of the whole Lena image. It is decoded at 4 times it’s original size, so the full decoded image contains 16 times as many pixels and hence this compression ratio is 91.2 to 1. This many seem like cheating, but since the 4-times-later image has detail at every scale, it really is not.

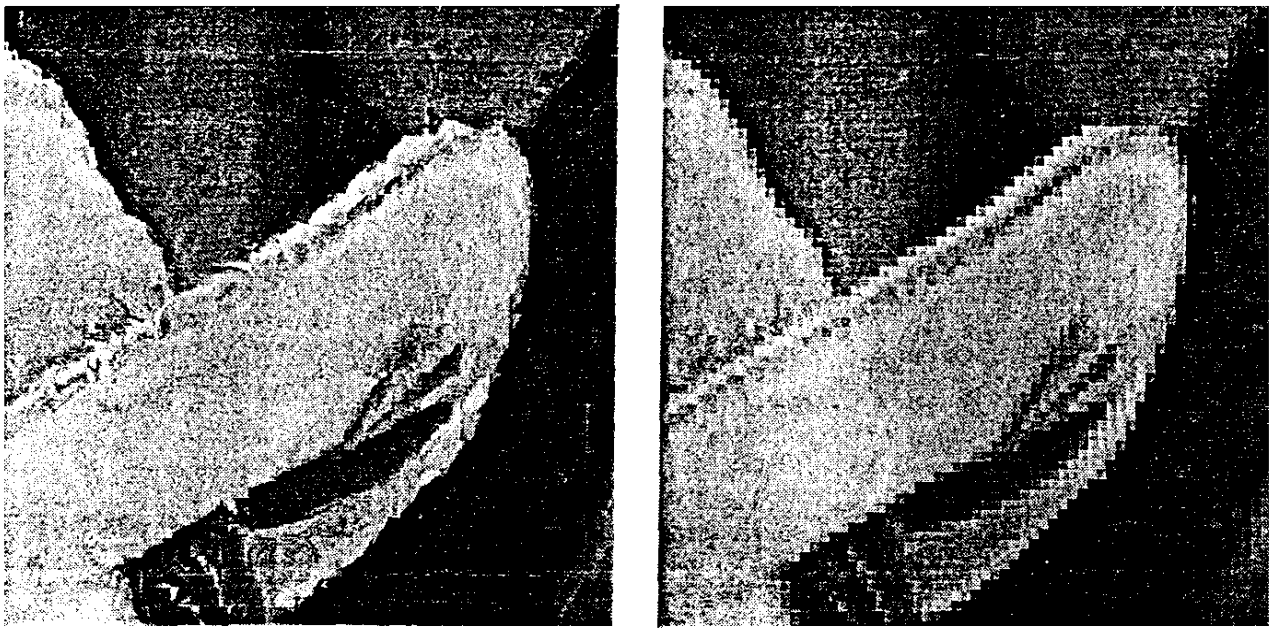

Figure 4: Portion of Lena's hat decoded at 4 times its encoding size 4(a), and the original image enlarged 4 times 4(b), showing pixelization [Y]

5. Encoding Images

The previous theorems tells us that transformation W will have a unique fixed point in the space of all images. That is, whatever image (or set) we start with, we can repeatedly apply W to it and we will converge to a fixed image.

Suppose we are given an image f that we wish to encode. This means we want to find a collection of transformations w_1 , w_2 , ...,w_N and want f to be the fixed point of the map W (see fixed Point Theorem). In other words, we want to partition f into pieces to which we apply the transformations w_i , and get back the original image f.

A typical image of a face, does not contain the type of self-similarity like the fern in Figure 3. The image does contain other type of self-similarity. Figure 5 shows regions of Lena identical, and a portion of the reflection of the hat in the mirror is similar to the original. These distinctions form the kind of self-similarity shown in Figure 3; rather than having the image be formed by whole copies of the original (under appropriate affine transformations), here the image will be formed by copies of properly transformed parts of the original. These transformed parts do not fit together, in general, to form an exact copy of the original image, and so we must allow some error in our representation of an image as a set of transformations.

Figure 5: Self similar portions of Lena

6. Proposed Algorithm

6.1 Encoding

The following example suggests how the Fractal Encoding can be done. Suppose that we are dealing with a 128 x 128 image in which each pixel can be one of 256 levels of gray. We called this picture Range Image. We then reduce by averaging (down sampling and lowpass-filtering) the original image to 64 x 64. We called this new image Domain Image.

We then partitioned both images into blocks 4 x 4 pixels (see Figure 6)

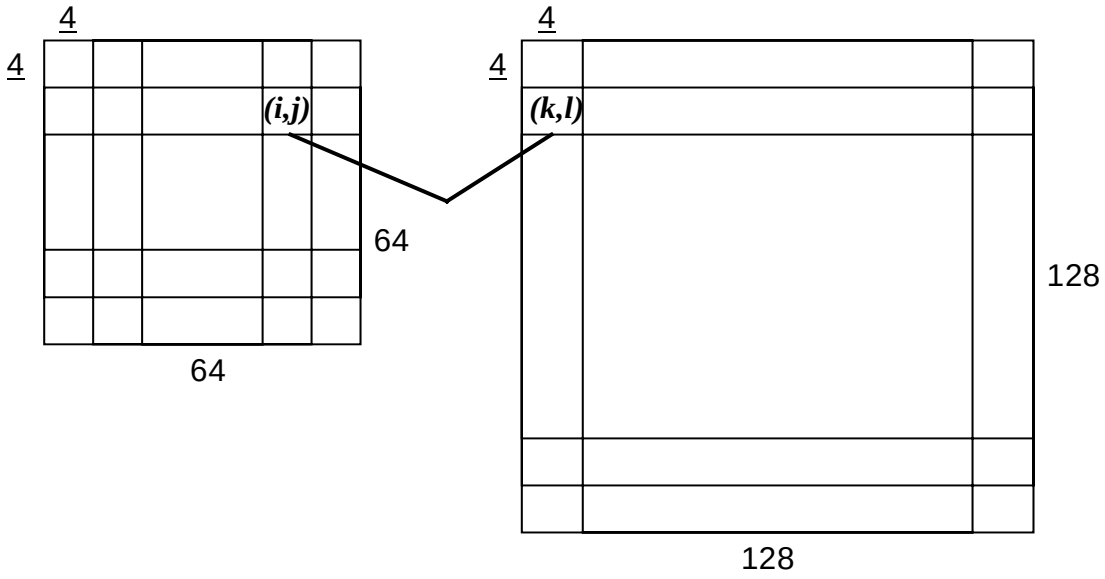

Figure 6: Partition of Range and Domain

We performed the following affine transformation to each block:

$$(D_{i,j}) = \alpha D_{i,j} + t_o \qquad (2)$$

where $\alpha = [0,1]$, $\alpha \in \Re$ and $t_o \in [-255, 255]$, $t_o \in Z$.

In this case we are trying to find linear transformations of our Domain Block to arrive to the best approximation of a given Range Block. Each Domain Block is transformed and then compared to each Range Block $R_{k,l}$. The exact transformation on each domain block, i.e. the determination of α and t_o is found minimizing

$$\min \sum_{m,n} (R_{k,l})_{m,n} - (\Gamma(D_{i,j}))_{m,n} \qquad (3)$$

with respect to α *and* t_o

$$\alpha = \frac{N_s^2 \sum_{m,n} (D_{i,j})_{m,n} (R_{k,l})_{m,n} - (\sum_{m,n} (D_{i,j})_{m,n})(\sum_{m,n} (R_{k,l})_{m,n})}{N_s^2 \sum_{m,n} ((D_{i,j})_{m,n})^2 - (\sum_{m,n} (D_{i,j})_{m,n})^2} \qquad (4)$$

$$t_o = \frac{(\sum_{m,n} (D_{i,j})_{m,n})^2 - \sum_{m,n} (R_{k,l})_{m,n})^2}{N_s^2 \sum_{m,n} ((D_{i,j})_{m,n})^2 - (\sum_{m,n} (D_{i,j})_{m,n})^2} \qquad (5)$$

where m, n, N_s = 2 or 4 (size of blocks)

Each transformed domain block $\Gamma(D_{i,j})$ is compared to each range block $R_{k,l}$ in order to find the closest domain block to each range block. This comparison is performed using the following distortion measure.

$$d_{l_2}(\Gamma(D_{i,j}), R_{k,l}) = \sum_{m,n} ((\Gamma(D_{i,j}) - (R_{k,l})_{m,n})^2 \qquad (6)$$

Each distortion is stored and the minimum is chosen. The transformed domain block which is found to be the best approximation for the current range block is assigned to that range block, i.e. the coordinates of the domain block along with its α and t_o are saved into the file describing the transformation. This is what is called the *Fractal Code Book.*

$$\Gamma(D_{i,j})_{best} \Rightarrow R_{k,l} \qquad (7)$$

6.2 Decoding

The reconstruction process of the original image consists on the applications of the transformations describe in the fractal code book iteratively to some initial image $\Omega_{init,}$ until the encoded image is retrieved back. The transformation over the whole initial image can be described as follows:

$$\begin{aligned} \Omega 1 &= \eta(\Omega_{init}) \\ \Omega 2 &= \eta(\Omega_1) \\ \Omega 3 &= \eta(\Omega_2) \\ ..... &= ..... \\ \Omega n &= \eta(\Omega_{n-1}) \end{aligned} \qquad (8)$$

η can be expressed as two distinct transformations:

$$\eta = \Gamma(\Omega)\Psi(\Omega) \qquad (9)$$

$\Gamma(\Omega)$ represents the down sampling and lowpass filtering of an image Ω to create a domain image e.g. reducing a 128x128 image to a 64x64 image as we describe previously. $\Psi(\Omega)$ represents the ensemble of the transformations defined by our mappings from the domain blocks in the domain image to the range blocks in the range image as recorded in the fractal. Ωn will converge to a good approximation of $\Omega orig$ in less than 7 iterations.

6.3 Results

We decoded Lena (128x128) using the set-up described in Figure 6. This is performed using the 2x2, and 4x4 block size and several different reference images (see appendix). Here is a summary of the results for the first example:

	Method I	Method II
Block Size	2x2	4x4
No Iterations	6	6
Time to encode	10mn	55s
Time to decode	45s	28s
Size of the code book	16384 bytes	6144 bytes
SNR	27dB	21dB
Peak Error	115	95
Bit Rate	1 byte/pixel	0.375 byte/pixel
Reference Image	square	square

* Peak Error: Pixel difference between original and decoded image.
* PC used: 386/25MHz, 4Mbyte RAM.

7. Conclusion

The results presented above were obtained using the MATLAB Software Simulator. A great improvement on the encoding/decoding time can be achieved with the use of real DSP hardware. Source code, for MATLAB and C31 SPOX Board can be obtained by contacting the author. Encoding/Decoding results for the SPOX Board are not included in this paper.

A weakness of the proposed reference design is the use of fixed size blocks for the range and domain images. There are regions in images that are more difficult to code

than others (Ex. Lena's eyes). Therefore, there should be a mechanism to adapt the block size $(R_{k,l}, D_{i,j})$ depending on the error introduced when coding the block.

I believe the most important feature of Fractal Decoding that I discovered on this project is the high image quality when Zooming IN/OUT on the decoded picture (See Figure 3). This type of compression can be applied in Medical Imaging, where doctors need to focus on image details, and in Surveillance Systems, when trying to get a clear picture of the intruder or the cause of the alarm. This is a clear advantage over the Discrete Cosine Transform Algorithms such as that used in JPEG or MPEG.

References

[B] Barnsley M., Fractals Everywhere. Academic Press. San Diego, 1989

[BJ] R.D. Boss, E.W. Jacobs, “Fractals-Based Image Compression,” NOSC Technical Report 1315, Sept. 1898. Naval Ocean Systems Center, San Diego CA 92152-5000.

[FJB1] Y. Fisher, E.W. Jacobs, and R.D Boss, “Fractal Image Compression Using Iterated Transformes,” NOSC Technical Report, Naval Ocean Systems Center, San Diego CA 92152-5000.

[H] John E. Hutchinson, Fractals and Self Similarity. Indiana University Mathematics Journal, Vol. 35, No.5. 1981.

[J] Jacquin A., Fractal Theory of Iterated Markov Operators with Applications to Digital Image Coding, Doctoral Thesis, Georgia Institute of Technology, 1989.

[Y] Y. Fisher, Fractal Image Compression, Siggraph 92 course Notes.

Appendix - Decoding Results

2x2 decoding using square

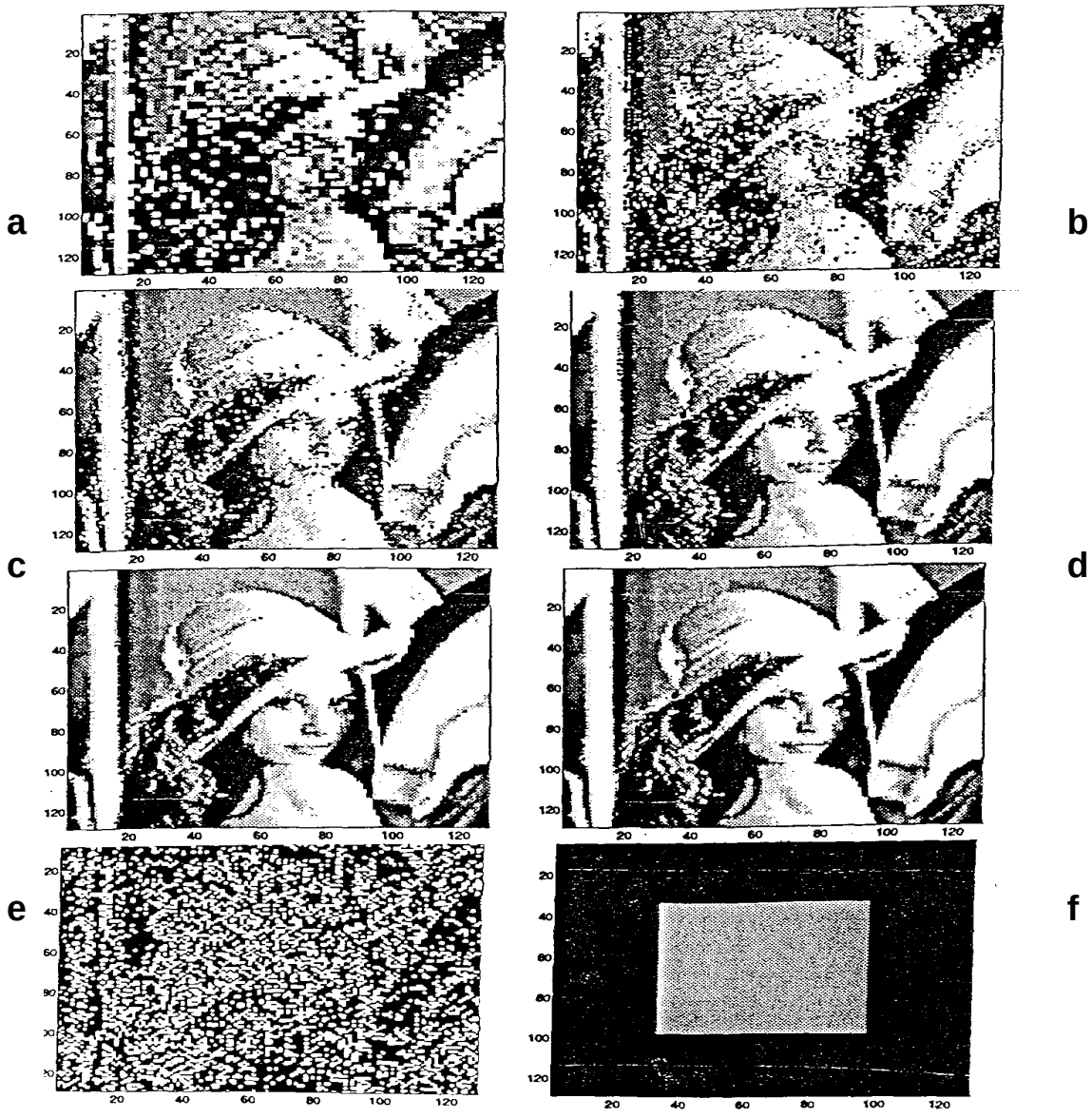

Figure 7: "a-f" first 6 decoding iterations with 2x2 decoding using square

4x4 decoding using square

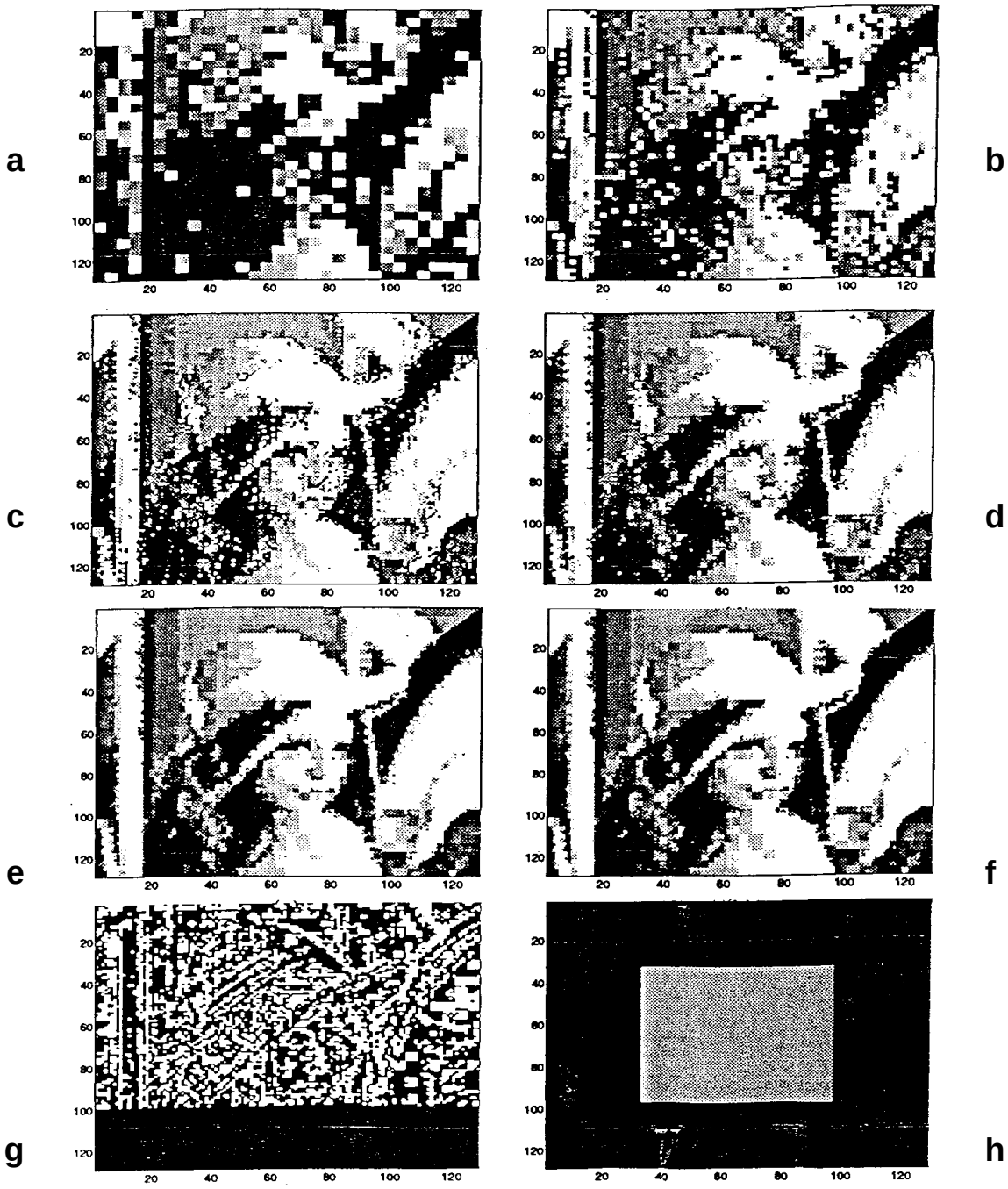

Figure 8: "a-f" first 6 decoding iterations with 4x4 decoding using square

4x4 decoding using fractal fern

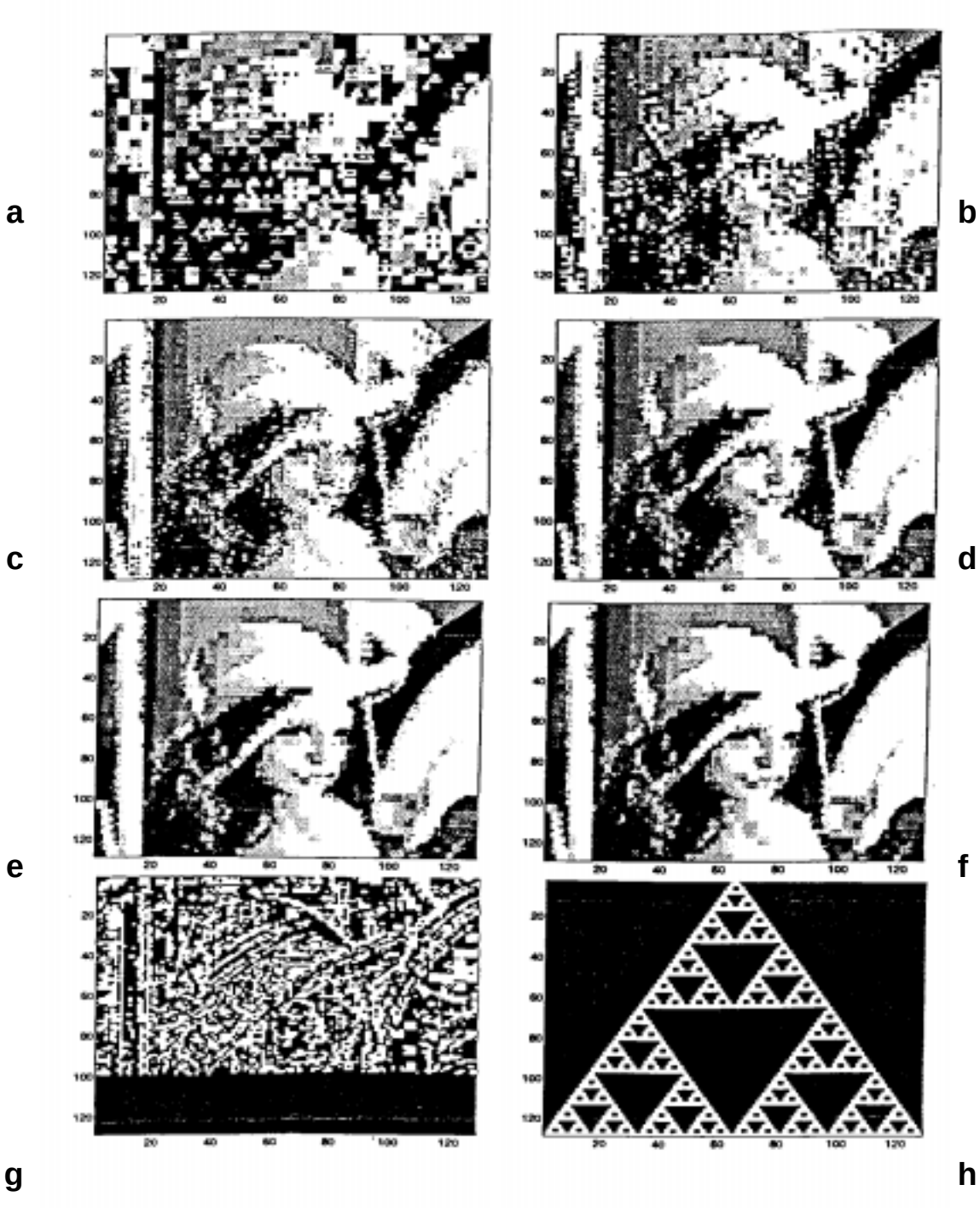

Figure 9: "a-f" first 6 decoding iterations with 4x4 decoding using fractal fern

4x4 decoding using tools

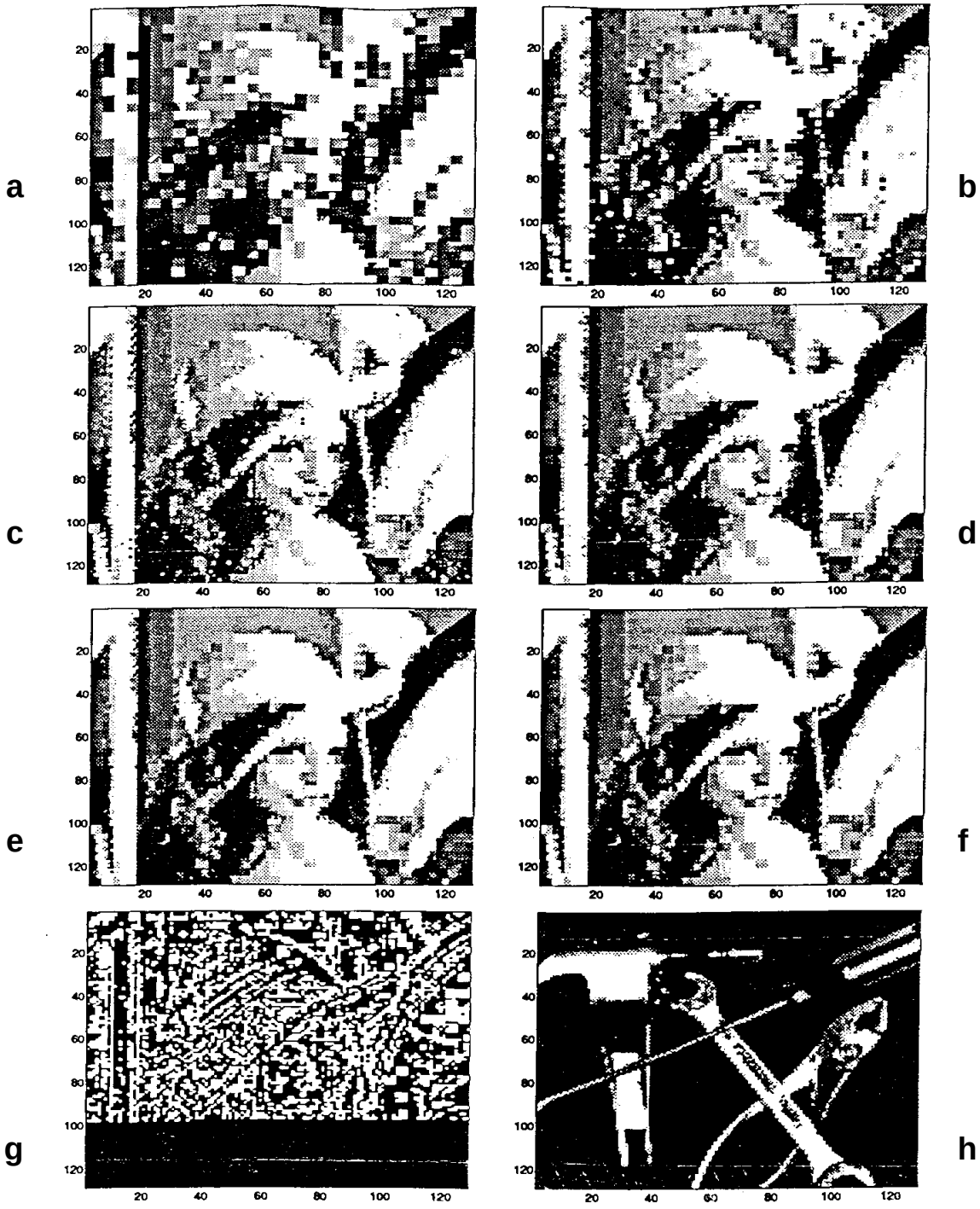

Figure 10: "a-f" first 6 decoding iterations with 4x4 decoding using tools