Naval Ocean Systems Center
San Diego, California 92152

Technical Report 1315
September 1989

Fractal-Based Image Compression

R. D. Boss
E. W. Jacobs

NAVAL OCEAN SYSTEMS CENTER

San Diego, California 92152–5000

E. G. SCHWEIZER, CAPT, USN
Commander

R. M. HILLYER
Technical Director

ADMINISTRATIVE INFORMATION

This work was performed by the Research Branch, Code 633, Naval Ocean Systems Center, for the Office of Chief of Naval Research, Independent Exploratory Division, Arlington, VA.

Released by
J. C. Hicks, Head
Research Branch

Under authority of
R. H. Moore, Head
ASW Technology
Division

JG

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION UNCLASSIFIED	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) NOSC Technical Report 1315	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Ocean Systems Center 6b. OFFICE SYMBOL (if applicable) Code 633	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS (City, State and ZIP Code) San Diego, CA 92152-5000	7b. ADDRESS (City, State and ZIP Code)
8a. NAME OF FUNDING/SPONSORING ORGANIZATION Office of Chief of Naval Research (IED) 8b. OFFICE SYMBOL (if applicable) OCNR-20T	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (City, State and ZIP Code) Arlington, VA 22217-5000	10. SOURCE OF FUNDING NUMBERS PROGRAM ELEMENT NO.: 0602936N PROJECT NO.: RV36 TASK NO.: ZE88 AGENCY ACCESSION NO.: DNIC000037

11. TITLE (include Security Classification)

FRACTAL-BASED IMAGE COMPRESSION

12. PERSONAL AUTHOR(S)

R. D. Boss, E. W. Jacobs

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (Year, Month, Day)	15. PAGE COUNT
Interim	FROM TO	September 1989	35

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS (Continue on reverse if necessary and identify by block number)
FIELD	GROUP	SUB-GROUP	iterated function systems (IFS)

19. ABSTRACT (Continue on reverse if necessary and identify by block number)

A short review of the theory of iterated function systems (IFS), a thorough explanation of their implementation, and an example using computer code useful in developing encoded images are presented. An example of an encoded map, with a brief discussion of data compression and error analysis, is presented. Details of an extension of IFS codes which allows for mixing of images, thereby resulting in a system with substantially increased power, are given. A simple scheme for automatic generation of IFS codes is given, followed by a discussion of improvements which may lead to a more generally useful data compression system.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION
☐ UNCLASSIFIED/UNLIMITED ☒ SAME AS RPT ☐ DTIC USERS	UNCLASSIFIED
22a. NAME OF RESPONSIBLE PERSON E. W. Jacobs	22b. TELEPHONE (include Area Code) (619) 553-1614 22c. OFFICE SYMBOL Code 633

DD FORM 1473, 84 JAN

83 APR EDITION MAY BE USED UNTIL EXHAUSTED
ALL OTHER EDITIONS ARE OBSOLETE

SUMMARY

A short review of the theory of iterated function systems (IFS), a thorough explanation of their implementation, and an example using computer code useful in developing encoded images are presented. An example of an encoded map, with a brief discussion of data compression and error analysis, is presented. Details of an extension of IFS codes which allows for mixing of images, thereby resulting in a system with substantially increased power, are given. A simple scheme for automatic generation of IFS codes is given, followed by a discussion of improvements which may lead to a more generally useful data compression system.

CONTENTS

FIGURES

TABLE

1. INTRODUCTION

The need for data compression is not new. With humble beginnings such as the use of acronyms and abbreviations in spoken and written word, the methods for data compression became more advanced as the need for information grew. The Morse code, developed because of the need for faster telegraphy, was an early example of a data compression technique. Largely because of the growing role of computer technology in today's society, the need for digital data compression (for both storage and communication) is becoming more critical.

In this report, the specific topic of compression of digital images will be addressed. If one is to consider that just one eight-bit grey scale, 1024- by 1024-pixel image requires a megabyte of storage (or for communication purposes, an appropriate bandwidth and time interval to transfer a megabyte of information), it is easy to understand that, even with the increasing availability of high volume, relatively inexpensive storage medium, there is a great need for developing methods for compression of digital images.

There are several conventional techniques which can be used to compress digital images. The most common of these can be classified as redundancy-reduction techniques. The simplest example of redundancy-reduction is run-length coding. There are variations of the run-length coding method, but, in short, the method can be described as follows. The value of the first pixel in the image is stored, along with the number of consecutive subsequent pixels that have the same value as the first pixel. The value of the next different pixel along with its repeat number is stored, and the process is continued for the entire image. Run-length coding is particularly attractive because encoding and decoding are simple and fast; it is a reversible method (i.e., there is no information lost in the encoding/decoding process) and for cases where large sections of the image are the same color, run-length coding can result in significant data compression.

While the methods discussed above may find new applications, the methods themselves are well known. Recently, a new approach to image compression has been developing, an approach which evolved out of the study of fractal objects and nonlinear chaotic systems. With the work of Mandelbrot (1977), the concept of representing images (specifically natural images) using non-Euclidian shapes, which he termed *fractals*, was introduced. Mandelbrot describes techniques such as fractional Brownian functions, with which it is possible to easily generate images of coastlines and mountains which have startling detail and are qualitatively quite similar to real images. The advantage of these techniques is that they use compact algorithms for generating fractals, objects of infinite detail, and, thereby, capture the detail and nature of complex real images without requiring retention of a large amount of information. The deficiency of these techniques, and the problem that is relevant to the image compression issue, is they are not useful in accurately encoding and, subsequently, regenerating a specific image. The work of Barnsley (1988) uses fractal techniques to answer this problem. The cornerstone of Barnsley's work is the Collage theorem, which is used as a guide in determining the necessary codes (which collectively form an iterated function system (IFS)) for regenerating a specific image. The report begins with a review of the Collage theorem and IFS, and an example of a practical application. In following sections, an extension of IFS is presented and approaches to the problem of automated encoding are discussed.

2. ITERATED FUNCTION SYSTEMS

2.1 Background

An iterated function system is defined as

$$W = \bigcup_{i=1}^{m} w_i, \tag{1}$$

where each w_i is an affine transform. An affine transform is the result of a set of rotations, skewings, scalings, and translations. The effect of such a transform, w_i on any point (x_n, y_n) is described by the set of equations

$$\begin{aligned} x_{n+1} &= a_i\ x_n + b_i\ y_n + e_i \\ y_{n+1} &= c_i\ x_n + d_i\ y_n + f_i \end{aligned} \tag{2}$$

If the transforms are restricted such that they are contractive (i.e., only transforms which make points become closer together, or equally spaced, after the transform), there is a guarantee that any iteration on the set of transforms must result in a bounded set of points. (In practice, in special circumstances, a w_i which does not adhere to the contractivity requirement may be included in the set W. These circumstances will not be considered in this report.)

The set of points generated by iteration (called the attractor of the set) can be adjusted to become any image. The Collage theorem (Barnsley, 1988, and Barnsley, Ervin, Hardin, and Lancaster, 1986) states that for any image, A, there exists a set of transforms, W, where

$$A \approx W(A) \tag{3}$$

where the error, measured using the Hausdorff metric (the maximum of all minimum distances from each point in either image to the other image) may be made as small as desired. Indeed, the Collage theorem predicts the maximum possible error for the attractor (versus the desired image). Consequently, to produce a given image, it is only necessary that the union of the subimages (tilings) produced by the individual transforms be sufficiently close to the desired image. Thus, the Collage theorem guarantees that it is possible to produce a set of transforms to create a customized attractor which will match any desired image. For a thorough description of the Collage theorem and the mathematics upon which it is based see Barnsley (1988).

2.2 Mechanics of Iterated Function Systems

The Collage theorem is the basis of finding a set of affine transforms from which, by means of an iterative execution of the transforms (be it in a random or deterministic way) a desired image can be reproduced. Barnsley and Sloan (1988) have written an article which explains how to perform the encoding of an image into a set of affine transforms and the subsequent decoding of the transforms to reproduce the image. For clarity, an explanation of the process and some examples will be given here.

The process of reproducing an image using IFS can be broken down into two steps. The first step is encoding the image into the proper set of affine transforms by using the Collage theorem as a guide, and the second step is decoding the transforms into the reproduction of the image. A simple example which illustrates the concept is the encoding and decoding of the Sierpinski triangle. A Sierpinski triangle is shown in figure 1a.* Because of its self-similarity, the encoding of the Sierpinski triangle is trivial. One can see by inspection that the Sierpinski triangle can be exactly covered by three reduced copies of itself. The location of these copies are shown in figure 1b. With the knowledge of three coordinates on the original image, $(x_n , y_n)^1$, $(x_n , y_n)^2$, $(x_n , y_n)^3$ (for example, the coordinates of the three corners of the Sierpinski triangle), and the coordinates of the corresponding three transformed points, $(x_{n+1}, y_{n+1})^1$, $(x_{n+1}, y_{n+1})^2$, $(x_{n+1}, y_{n+1})^3$ (for example, the three corners of copy 1 in figure 1b), equation 2 results in two independent sets of three equations and three unknowns, which, in general, can be solved for the six coefficients of the affine transform. The six coefficients of each of the three affine transforms which define the encoded data for the Sierpinski triangle are given in table 1.

Table 1. Encoded data for the Sierpinski triangle.

a	b	c	d	e	f
0.5	0.0	0.0	0.5	-175.0	-175.0
0.5	0.0	0.0	0.5	175.0	-175.0
0.5	0.0	0.0	0.5	0.0	175.0

The Sierpinski triangle is now represented by the 18 numbers given in table 1, so storage or transmittal of the image amounts to storage or transmittal of these numbers.

Given the encoded image, a decoding process must be performed to reproduce the image. In the following paragraphs, first the mechanics of generating the decoded image will be discussed, followed by a brief discussion considering iterated function systems in terms of more familiar systems.

*Figures are placed at the end of the text.

Two methods of generating the decoded image will be described, first the method of random iteration, and then deterministic iteration. Using the Sierpinski triangle as an example, the method of random iteration proceeds as follows. An initial point (x_1 , y_1) is chosen. The position of this point is not important. By means of a random number generator, one of the transforms in table 1 is picked at random, and using equation 2, the point (x_2 , y_2) is generated. Again a transform from table 1 is picked at random, and equation 2 is applied to obtain (x_3 , y_3). This process is repeated and, after several iterations, the point (x_n , y_n) at each iteration is printed. The first several points must be skipped because, much like a dissipative dynamic system, a transient may exist before the system comes to a stable state. Therefore, since the initial point may not have been in the image, it may take several steps before the iteration falls into the image. In figure 2, the Sierpinski triangle is shown at two stages of the iteration.

Transforms for the Sierpinski triangle in table 1 have equal contraction and no overlap of the reduced images. As a result, the attractor in figure 2 possesses a uniform intensity over its entire area as it is generated. In cases where the contraction of the transforms is not equal, if the decoded image is to maintain a uniform intensity as it is generated, the probability of choosing a given transform must be inversely proportional to the amount the transforms contract areas. This does not require storage of extra information because the probability of executing a transform

$$p_i = \frac{| a_i \, d_i - b_i \, c_i |}{\sum_{j=1}^{m} | a_j \, d_j - b_j \, c_j |} \tag{4}$$

can be calculated from the the coefficients of the set of affine transforms.

The second method for decoding an image is a deterministic iteration. An example using the Sierpinski triangle is illustrated in figure 3. Starting with an arbitrary point, or set of points (in figure 3a the initial points were chosen to be a square with an x inside), every transform in table 1 is executed on every point in the initial set. This first iteration generates the set of points in figure 3b. The process is then repeated on every point in figure 3b. Figure 3c shows the attractor after 6 iterations, and figure 3d shows the attractor after 11 iterations. The advantage of the deterministic method is that it can take less time to generate the completed image. The drawback is that the image does not, in general, have a uniform intensity as it is generated.

Using the process outlined above, it is seen that, similar to a nonlinear dissipative system that possesses a strange attracting set, an attracting set (which in fact is a reproduction of the original image) can be generated from the proper set of affine transforms. The term "strange attractor" is usually used in the context of a chaotic dynamic system. Because there are no rules dictating the time evolution of the generation of the decoded image (since there are two totally different ways of decoding, there cannot be a "time" ordering), the system is fundamentally different from a dynamic system. The term fractal, emphasizing the fact that the decoded image is a geometric construct, might be a more appropriate label for the decoded image. There are two reasons why the term fractal is not completely appropriate.

First, the decoded image is truly an attractor, in that one starts with a point, or set points chosen at random (inside or outside the image), and the system is attracted to the set of points which makes up the image. This contrasts with a fractal (for example, the triadic Koch island), which is typically purely a geometric construct and not an attractor in the sense that the rules of construction do not take a randomly chosen initial state and dictate the evolution of this state towards a stable state. Secondly, the images generated by an IFS do not necessarily have fractal character. In the example above, a Sierpinski triangle, certainly a fractal object, was generated. By adding a fourth transform, one that covers up the unlabeled center triangle in figure 1b, the entire triangle becomes covered by small copies of itself. The resulting decoded image therefore will be a solid triangle, which is not a fractal object. The decoded image is clearly different than what is typically thought of as a strange attractor, or a fractal, and is probably best described as a geometrical attractor.

Another example of a more natural image which is easily generated using IFS codes is the fern leaves in figure 4. The fern can be generated with just four transforms, one that transforms the fern into all but the bottom of the stem and the bottom right and left branches (i.e., a small contraction and translation towards the top of the fern), one that transforms the entire fern into the bottom of the stem, and one each for the bottom right and left branches. The fern leaf will be discussed in more detail later in this report.

2.3 Generating General IFS Codes

The Sierpinski triangle and the fern, because of their inherent self-similarity, are particularly simple images to reproduce using the IFS technique. Writing a computer code to aid in encoding more general images is straightforward, and the authors have written such a code to demonstrate the use of IFS for data compression.* This code was used to encode and, subsequently, decode a digitized map of Point Loma (a peninsula at the opening of San Diego Bay on which the Naval Ocean Systems Center is located). This example is presented with the dual purpose of illustrating that IFS codes can be used for general images (as opposed to contrived images like the Sierpinski triangle and fern) and that practical applications for this method are feasible.

The 960- by 428-pixel digitized map of Point Loma is shown in figure 5a. The image was generated from 637 (x,y) coordinates along the outline of the map. The points were then connected with straight lines, and the interior was filled in. Listed in appendix A are the 48 transforms (i.e., the encoded data) generated by use of the encoding program. When decoded, these transforms result in the image shown in figure 5b. Note that the encoding program did not generate the transforms automatically. The program simply supplied an interactive graphics interface to aid the user in choosing a set of transforms (by complying with the Collage theorem) which generates the desired attractor. The formidable problem of fully automated encoding will be discussed later in this report.

*The IFS encoding software was written in Pascal for an Apollo workstation and is available from the authors upon request.

The end objective in encoding images is data compression. Claims of large compression ratios are commonly made for various compression techniques, often with little explanation of how the compression ratios were computed. One may arrive at vastly different compression ratios depending on how much information is considered to be in the original image. Two different compression ratios for the encoded Pt. Loma will be given here, one a very conservative estimate, and the other a very liberal estimate. If one considers the original image to consist of only the 637 (x,y) pairs, which were used to generate the digitized original image (figure 5a), a fair estimate of the image compression would be 4 to 1. Note that is a very conservative estimate because, considered in this way, the original image is already highly compressed. If one considers the original image as the entire 960- by 428-pixel map (i.e., 410,880 bits), a fair estimate of the compression would be 120 to 1 (allowing 12 bits for each coefficient of the transforms). Most other ways of estimating the compression ratio for this example would result in a number between the two compression ratios stated here.

Achieving high compression ratios is of great importance, but only if the reproduced image is a good representation of the original image; therefore, some estimation of the error in the reproduced image is necessary. The absolute difference of the original image (figure 5a) and the decoded image (figure 5b) is shown in figure 5c. The root mean square (rms) error for a black and white image is given by

$$e_{rms} = \sqrt{\frac{p_e}{p}} \tag{5}$$

where p_e is the number of lit pixels in the absolute difference of the images (figure 5c) and p is the total number of pixels in the image. The rms error for the decoded Pt. Loma image is 0.076. While this is a meaningful number, a good value of the rms error does not necessarily indicate a good image reproduction because the rms error does not consider how far away the points in error are from the desired image. A measure of how far away two images are is given by the Hausdorff distance. The Hausdorff distance is defined as

$$h_d = \max \left\{ \max_{ij} [\min_{kl} (\text{distance } A_{ij} \text{ to } B_{kl})], \max_{ij} [\min_{kl} (\text{distance } B_{ij} \text{ to } A_{kl})] \right\} \tag{6}$$

where A_{mn} is a lit pixel in image A, and B_{mn} is a lit pixel in image B. The Hausdorff distance for the decoded Pt. Loma image is 5.83 pixels or, on the true scale of Pt. Loma (roughly 3.5 miles long), about 130 feet.

2.4 Limitations of IFS Technique

The examples above demonstrate some of the capabilities of IFS codes, but IFS codes do have limitations to their usefulness. When encoding an image using IFS codes, one is restricted to covering the original image with copies of itself. The Collage theorem guarantees that this can be done, but, quite often, it cannot be done with a reasonable number of transforms. As an example, the Mercedes Benz symbol generated using an IFS code is illustrated in figure 6. The image took 42 transforms

to generate and, clearly, is a rather poor reproduction of the Mercedes Benz symbol. Because of the particular geometry of the Mercedes Benz symbol, it is quite hard to cover with copies of itself. A solution to this problem will be given in the next section.

There are two other major limitations to IFS codes, as outlined in the preceding paragraphs. The first problem is the question of automation. For some applications, where encoding of a limited number of images is required (for instance, data compression of maps), an automated system is not necessarily required. But for more general applications, automated encoding would be essential. The question of automation will be addressed in section 4. The second problem is to accurately reproduce images with grey scale (for instance, a photograph). In the examples given above, only high-contrast images were considered. There are simple ways of introducing grey scale into a decoded image (e.g., by adjusting the probabilities p_i), but to do so in such a way as to duplicate the grey scale of a desired image would be extremely difficult without a prohibitive number of transforms.

3. RECURRENT ITERATED FUNCTION SYSTEMS

3.1 Introduction

An extension to the IFS technique has been used by Barnsley and Jacquin (1988) to encode the outline of a cloud. This extension uses a *part* of the image to cover another part of the image. Unlike IFS, where any transform may be applied at any iteration, i.e., the probability of applying any given transform is constant, in this technique the probability of applying a transform is dependent upon which transform has been last applied. In fact, after a given transform has been applied (resulting in a point being generated in a specific area of the attractor, in this case the cloud) only a few transforms may be allowed, with the remaining transforms having zero probability of being applied. This is tantamount to giving the system a memory of its operation. This is done, in practice, by replacing the probability vector of standard IFS (the p_i's) by a matrix. Furthermore, since the Collage theorem has been proven for such a recurrent system (Barnsley, Elton, and Harding, 1989), it is known that covering an "image" with parts of itself will result in a system which will produce the "image" as the attractor of the system.

This concept of tiling an image with copies of parts of itself is a significant step upward in complexity. If several separate images, which are in different regions of space (either on a single plane or separate planes) are considered as one total image, the individual "sub"-images can be used to tile each other (and themselves), resulting in a recurrent iterated function system (RIFS). A RIFS technique which allows for the mixing of images and the subsequent production of a greater variety of images is described in the following sections.

Although the RIFS technique represents a major improvement over the IFS technique, RIFS does not address the important limitations of grey scale and automation raised at the end of the last section.

3.2 The Mechanics of RIFS

Since standard IFS techniques require that an image be tiled by copies of itself, the method can have difficulty in generating certain images. For example, while it is easy to produce the fractal ferns shown in figures 4a and 4b, it is not possible to produce the ferns in figures 7a and 7b with fewer than an infinite number of transforms. This is because of the complete self-similarity possessed by the ferns of figures 4a and 4b, while the ferns of figures 7a and 7b do not possess such complete self-similarity.

The standard IFS ferns are produced by four transforms described at the end of section 2.2. Such a set of transforms results in the complete self-similarity of these ferns. To be quite explicit, since the third and fourth transforms copy the fern to secondary branches directly opposed from one another and each leaf is a copy of the entire fern, each leaf has tertiary branches in opposing positions. Because the tertiary branches are produced by copies of the secondary branches (when the fern is copied to make the secondary branches) the quaternary branches are also opposed, and, in fact, at every level the branchings are opposed (thus, the fern is an all opposed fern, or $\overline{\mathrm{O}}$). The transforms which produce the fern in Figure 4b are identical except that the copies which make the bottommost leaves are in nonopposed (alternate) positions, resulting in all branchings being in alternate positions (an $\overline{\mathrm{A}}$ fern).

By contrast, the ferns in figures 7a and 7b show elements of both of the $\overline{\mathrm{O}}$ and $\overline{\mathrm{A}}$ ferns. In fact, the fern in figure 7a has secondary branches opposed, tertiary branches alternate, quarternary branches opposed, etc. (denoted as $\overline{\mathrm{OA}}$), while the fern in figure 7b has the opposite orientations ($\overline{\mathrm{AO}}$). Some reflection will convince the reader that while these $\overline{\mathrm{OA}}$ and $\overline{\mathrm{AO}}$ ferns cannot be covered with copies of themselves (that is less than an infinite number of copies), they can be covered by a mixture of copies of themselves (a collapse to a stick to cover the bottommost part of the stem, and a shrink and shift to cover all but the bottommost leaf on both sides of the stem) and *copies of each other* (the two bottommost leaves).

As an example, to illustrate the process of constructing a RIFS, the Mercedes-Benz symbol will be used. It is seen that the symbol can easily be constructed from a ring and a few triangles. Hence, the RIFS will operate on three images: a ring, a triangle, and the Mercedes-Benz symbol, which are shown in figure 8. Transforms are constructed such that each image is completely covered by copies of itself and/or copies of the other images. For example, the triangle is covered by copies of itself (shown as transforms w_{22} in figure 8), the ring is covered by copies of itself (shown as transforms w_{33}), and the Mercedes-Benz symbol is covered by four copies of the triangle (shown as transforms w_{21}) and one copy of the ring (shown as transform w_{31}). Furthermore, it is necessary that the images not only be completely covered, but it is necessary that some path exist from any image to any image (hence, transforms w_{12} and w_{13}).

This technique of RIFS is particularly powerful as it opens up the possibility of the use of a library of images as the tiling images. In so doing, the subimages could then be represented by the library number of the tiling image, rather than the full IFS-like codes needed to generate the image.

The storage of the encoded information is done by first entering the number of planes, n, (subimages) used in the RIFS. Next an $n \times n$ matrix is recorded where the jth entry in the ith row, m_{ij}, is the number of transforms which operate on plane i which result in a point on plane j (this matrix is the connection or plane hopping matrix). Finally the transforms themselves are stored. They are stored in the order m_{11}, m_{12}, ..., m_{1n}, m_{21}, m_{22}, ..., m_{nn}. In figure 9 the regenerated *perfect* image of the Mercedes-Benz symbol is shown. Note that not only does the RIFS technique create a better Mercedes-Benz symbol than IFS does, it also does so using less data (the IFS image is generated from 42 transforms, or 252 numbers, while the RIFS uses 12 transforms on three planes, or 72 numbers). Given the amount of data used to generate the image by RIFS versus that needed to pass a 600- by 600-bit map, a rough compression ratio of about 400:1 is obtained.

As in IFS, it is necessary to give each transform a probability, and it is desired that such a probability not be stored, but rather computed automatically. In order to automatically generate the probabilities for the various transforms the area is again used; however, unlike IFS it is not possible to directly use the areas. In IFS the areas are used to insure that the points are generated uniformly throughout the image, i.e., the probabilities are computed from the area covered by the transform. It is, therefore, desirable to use the same idea, i.e., the true probability of applying any given transform, which generates a point on plane j, should be proportional to any other transform which generates a point on that plane, where the ratio of the true probabilities should be equal to the ratio of the areas covered by the transforms.

For a given plane hopping, plane i to plane j, there exists a set of transforms W_{ij} which is

$$W_{ij} = \bigcup_{k=1}^{m_{ij}} w_k(i,j) \tag{7}$$

where $w_k(i,j)$ is the kth transform from plane i to plane j. Clearly, if the individual probabilities $p(w_k(i,j))$ are given by

$$p(w_k(i,j)) = P(i,j) * \frac{| a_k(i,j) * d_k(i,j) - b_k(i,j) * c_k(i,j)|}{\sum_{k=1}^{m_{ij}} | a_k(i,j) * d_k(i,j) - b_k(i,j) * c_k(i,j) |} \tag{8}$$

then $P(i,j)$ is the member of the hopping probability matrix for hopping from i to j, where the effective area for a matrix member, $A(i,j)$ is given by the simple sum

$$A(i,j) = \sum_{k=1}^{m_{ij}} | a_k(i,j) * d_k(i,j) - b_k(i,j) * c_k(i,j) | \tag{9}$$

The additional requirements on the probably matrix members are

$$\sum_{j=1}^{n} P(i,j) = 1. \tag{10}$$

However, the actual probability of implementing a given plane hopping element is not $P(i,j)$ but rather the product $\mathbf{P}(i) * P(i,j)$, where $\mathbf{P}(i)$ is the probability of being on plane i. This product is the true probability of applying any one element; thus, to fulfill the requirement concerning the ratio of the areas it is required that

$$\frac{\mathbf{P}(i) * P(i,j)}{\mathbf{P}(n) * P(n,j)} = \frac{A(i,j)}{A(n,j)} \quad . \tag{11}$$

Furthermore, the actual probability of being on any plane is given by the expression

$$\mathbf{P}(j) = \sum_{i=1}^{n} \mathbf{P}(i) * P(i,j). \tag{12}$$

Finally, normalization requires that

$$\sum_{i=1}^{n} \mathbf{P}(i) = 1. \tag{13}$$

These equations represent $n^2 + n$ equations in $n^2 + n$ unknowns and can be easily solved.

3.3 An Example From Nature

The tree shown in figure 10a is a typical IFS-fractal tree. It possesses complete self-similarity (i.e., each branch divides into sub-branches in precisely the same way as all the others), and it has no true width (i.e., the trunk is a line). Both of these properties are "failures" in that real trees do not possess these traits.

In order to improve the tree image, first the branching pattern can be altered in a method analogous to that used in the construction of the $\overline{AO}$ and $\overline{OA}$ ferns above, namely that the tree can have a set of secondary branchings which are

different from the primary branchings by mixing two (or more) patterns. The result of such a process is the tree shown in figure 10b. Here the tree has four primary branchings, each of which is a copy of an unseen tree which has three primary branchings, each of which is a copy of the tree shown.

To further improve the tree, the trunk can be given width. This is accomplished by replacing the transforms which form the trunk by another entire plane, on which a trunk with width is generated. The transforms to make a trapezoidal trunk have been added to the tree of figure 10b, resulting in the tree in figure 10c.

4. AUTOMATION OF IFS ENCODING

4.1 Introduction

The preceding sections have described techniques which can result in the storage of data/images with substantial compression; however, the key step of encoding the images must be done interactively, with a person performing the pattern recognition and data reduction. While this does not preclude the application of these techniques to various problems, it does restrict the variety of problems to which they may be applied (as examples, compression of satellite images prior to radioing them to earth precludes having a person in the loop, but using an IFS technique to compress the map of the world is a sufficiently finite task, with no temporal restrictions, that having a person/people perform the encoding would not necessarily be restrictive). Nonetheless, for the described fractal compression techniques to be widely applicable, it will be necessary to be able to automate the encoding step, that is, to have a computer do the encoding of the image.

The following sections describe one method to produce such an automation. It is a primitive first step, but nonetheless shows a proof of concept and indicates that (with improvement) an automated method for the construction of IFS transforms is possible.

4.2 The Quad-Tree Method of Automation

A method to automatically produce IFS transforms to cover an arbitrary image has been developed. It uses a quad-tree to portion the image into "lit" and "unlit" regions. This is accomplished by dividing the total image space (some $n \times m$ pixel array) into four pieces or quadrants (hence "quad").

Each quadrant is randomly searched for lit and unlit pixels. The numbers of each are then compared to the total number of pixels and a decision is made to (1) make the entire region lit, (2) keep the entire region unlit, or (3) further divide the region into a subset of quadrants (another branch down the "tree"). Each branch is searched to whatever depth is required (to maximum depth), thus resulting in a pattern of lit and unlit regions. The lit regions are then covered by IFS transforms.

The covering is done by compressing the entire lit pattern into continuous strings of lit regions. Continuity checking is done first along rows (generating a

series of horizontal coverings), then in columns (generating vertical coverings). Each horizontal (and vertical) covering covers a strip as wide as the smallest branching would produce (that is, if the image may be divided 10 times then the smallest width is $1/2^{10}$ of the total size).

At each stage the decision to further subdivide is made by comparing the number of lit and unlit pixels found in a random search to a predetermined number. The number used in the comparison is determined from a worst-case scenario. Assume that the region being considered is an $n \times n$ region, which at most could be subdivided m more times, then the smallest box would have $n^2/2^{2m}$ pixels. If exactly half of one box is lit and the entire rest of the region is unlit, the probability of randomly choosing a lit pixel, p_{on}, is given by

$$p_{on} = \frac{1}{2^{(2m+1)}} \quad , \tag{14}$$

and the complementary probability, p_{off}, is $1-p_{on}$. If a total of j points is chosen, the probability of finding exactly i lit pixels, P_{ij}, is given by the binomial distribution

$$P_{ij} = \frac{j!}{i!(j-i)!} p_{on}^{i} p_{off}^{(j-i)} \quad . \tag{15}$$

Then define k_{min} to be the minimum integer which satisfies the condition

$$\sum_{i=0}^{k_{min}} P_{ij} \leq 0.1 \quad . \tag{16}$$

Thus, for j points if less than k_{min} points are found to be lit the region is not subdivided and is left unlit (or vice versa for the number of unlit pixels). This ensures that any region which is left as a "block" has less than a 10-percent probability of including one smallest box which is incorrectly colored. Note that the two parameters which determine k_{min}, namely the number of chosen points j and the number of possible subdivisions m, are both known in advance, with m depending on the desired resolution and j being chosen to compromise between speed of computation (j small) and a reasonable k_{min} (j large).

4.3 Some Examples and Criticisms

A program was written to process an image consisting of a circle of radius 1 centered at (0,0) in a box of size $-2 < x < 2$, $-2 < y < 2$. The total box was divided a maximum of six times. The output of the program was 64 affine transforms. The image generated by these transforms is shown in figure 11a. Likewise, the program was used to analyze the image in figure 11b, with the resulting regenerated image shown in figure 11c.

Both images suffer from errors in that they have large holes in the image. This is due to an incomplete coverage This incompleteness is due to having used

nonoverlapping rectangular boxes into which contracted tilings were placed (i.e., there was no possibility for any two horizontal or vertical coverings to overlap). Further, each image shows raggedness on the boundary due to the coverings filling the rectangular boxes, rather than touching the true boundary. These two problems are, in fact, rather simple to solve by using a more careful selection of the positioning and scaling of the coverings.

A more serious, and at this point unsolved, problem is that this method does not, in fact, use the true power of the IFS technique, because it does not allow for the rotation or skewing of the image. Consequently, it is not possible to take full advantage of the IFS technique's ability to make the distorted copies (tilings) truly cover the original image. This version of automation will have (in general) a rather poor compression ratio unless a very low resolution is used, in which case the accuracy of the reproduction will be poor.

5. DISCUSSION

In the second section of this report, the concept of IFS codes was reviewed and the map of Pt. Loma was given as an example of its possible practical application. An idea of the limits of the class of images for which IFS codes can be used with good data compression and low error was given by means of the Mercedes-Benz symbol. In section 3 of this report, an extension of IFS codes, a recurrent IFS type system, was introduced. The increased power of the RIFS-type system comes about by means of an extension of the probability vector (used in the IFS system) to a probability matrix where certain elements in the matrix are allowed to be zero. In the treatment of section 3, the zeroes in the probability matrix are manifested by means of the connection matrix. Because the connection matrix puts a restriction on which transform can be executed, the resulting image loses its total self-similar properties (i.e., it is no longer made up of contracted copies of itself), yielding a system where an image can be generated from contracted copies of selected sections of itself. The result is a method that can be used on a much larger class of images than IFS codes, with vastly improved compression and reduced error.

In section 4, the problem of automated encoding and a simple example illustrating the start of a possible solution to this problem were presented. The applications of a system requiring interactive encoding are limited, whereas a wide array of applications awaits an automated system. For this reason, the importance of finding a solution to the automation problem cannot be underestimated. The quad-tree automation using an IFS code illustrated in section 4 shows that automation is a possibility. This example clearly fails to retain good data compression and low error. Retaining good data compression and low error with a reasonable computation time is the problem which must be overcome in developing an automated encoding system.

At the end of section 4, some simple improvements which would reduce the error without necessitating an increase in the amount of data were given. There are certainly other avenues which might lead to even larger improvements in automation techniques. For instance, an extension of the quad-tree technique using an IFS code described in section 4, to one that uses a RIFS type method similar to that discussed in section 3 could improve the accuracy of the image with, at worst, minimal cost in

compression. As a simple example, rectangles (instead of copies of the image) could be used to fill the image, thus resolving the problem of the grid-like quality of figures 11a and 11c at the expense of only four additional transforms (three to make the rectangle and one to copy the image from its plane to the rectangle). Use of a RIFS-type method could also improve the compression ratio and error by allowing for the use of a variety of images for the coverings.

The quad-tree method is just one possible approach to the encoding automation problem. Other techniques might also be employed for the automation process. Because of the stability of the attracting set, small changes in the set of affine transforms $W(A)$ used to generate the attractor A will result in small changes in A. The Collage theorem guarantees that as the Collage improves, the Hausdorff distance between the attractor and target image improves. Therefore, a simulated annealing process on the affine transforms may be a possible method of improving a given set of transforms to a predetermined image resolution criteria. Libeskind-Hadas and Maragos (1987) have investigated the method of skeletonization of images as a means of selecting affine transforms that compose the Collage. This method is of particular interest because the skeletonization process is well suited for automation. Another possible approach to the encoding automation problem is the use of neural network processing as a means of creating the Collage. Whether one of these methods, or a combination of these methods, will result in an automated system that can substantially compress and accurately reproduce images is the subject of future work.

So far in this report, only high-contrast black and white (binary) images have been considered. The issue of grey scale images is of sufficient importance that a brief discussion is warranted. In section 2, the method of introducing grey scale into the image by adjusting each p_i, thereby controlling the rate of development of the attractor in the region of each transform, was discussed. Although this method is good in that it requires little additional information storage, the requirement it puts on selection of transforms for reproducing the grey scale and shape of a target image is far too restrictive to be practical. Therefore, another approach is needed. Possibly the key to the grey scale problem is the realization that if a general technique for compressing high-contrast images is achieved, breaking up a grey scale image into a high-contrast image for each bit of the grey scale and compressing each one of these images separately will result in a compressed grey scale image. During the decoding process, a high-contrast image for each level of the grey scale would be generated from which the complete grey scale image could be reconstructed. For most grey scale images, encoding the image represented by the high-order bits of the grey scale should be relatively easy. The problem with this technique would be encoding the images represented by the lower order bits. The low-order bit images may often look like noise and would be difficult to encode using these techniques.

The problems involved in developing an automated image compression system with the ability to handle grey scale images using the fractal techniques described in this report are significant, but the benefits of such a system if it could be developed are also significant. If more motivation is needed, it has been reported (Science, 1989) that a system for automated compression of 256- by 256-pixel black and white images has been developed. The techniques of using iterated function systems for data compression are new and powerful, and the potential benefits of systems which use them make further investigation of these problems worthwhile.

6. REFERENCES

[illegible] *Wavelets*, Academic Press, Inc., San Diego.

[illegible] and D. P. Hardin. 1989. Constr. Approx. 5, 3.

[illegible] 1986. *Proc. Natl. Acad. Sci.* [illegible]

[illegible] 1988. *SPIE* 1001, "Visual Comm. and Image [illegible]

[illegible] 1988. *Byte* 13, 215.

[illegible] Chicago. 1987 *SPIE* 845, "Visual Comm. and Image [illegible]

[illegible] *Fractals, Form, Chance, and Dimension*, W.H. Freeman and Company, San Francisco.

[illegible] "Image Compression by Computer." 243, 1288.

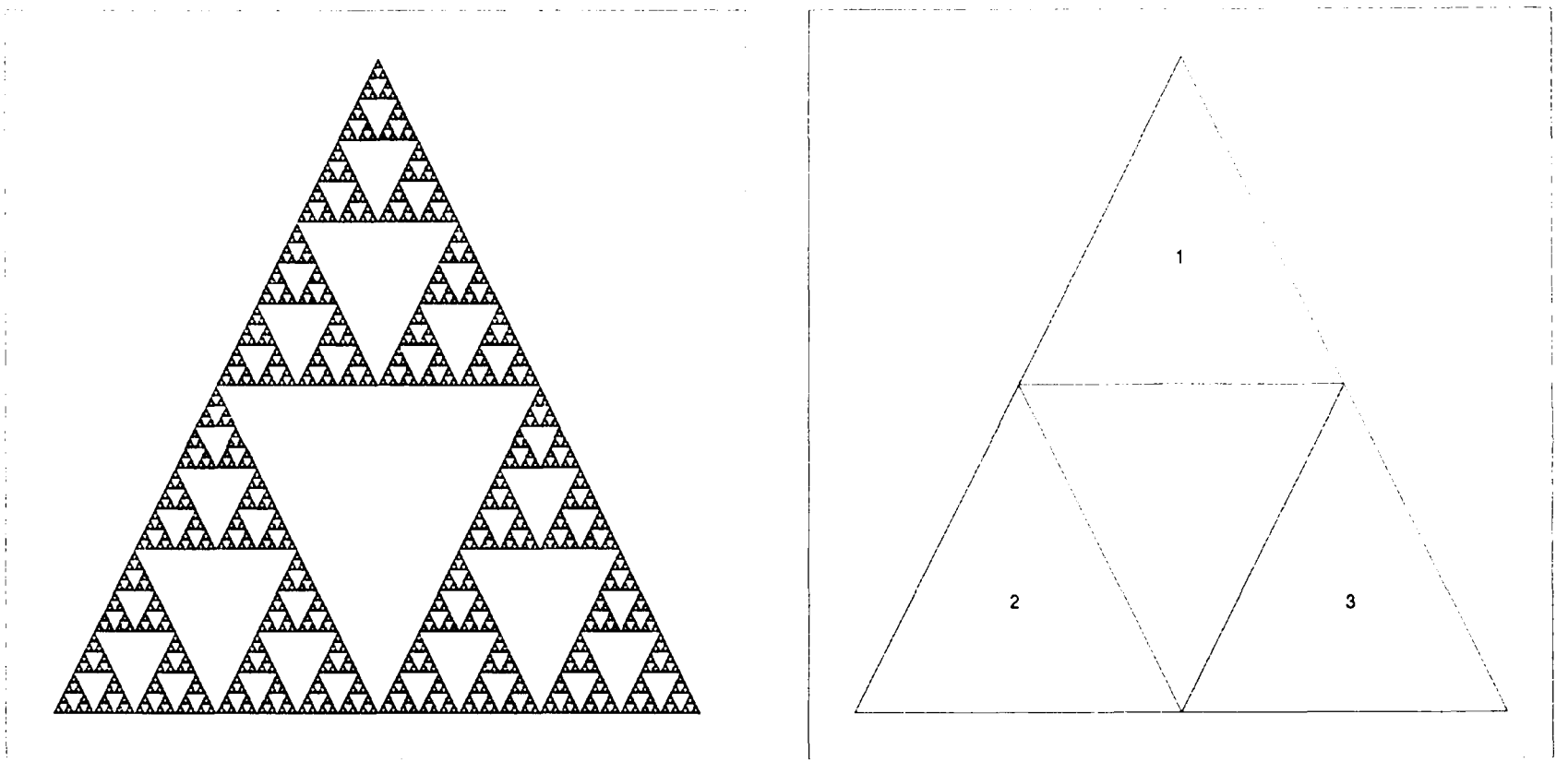


Figure 1. (a) A Sierpinski triangle, and (b) the location of three reduced copies resulting in complete coverage.

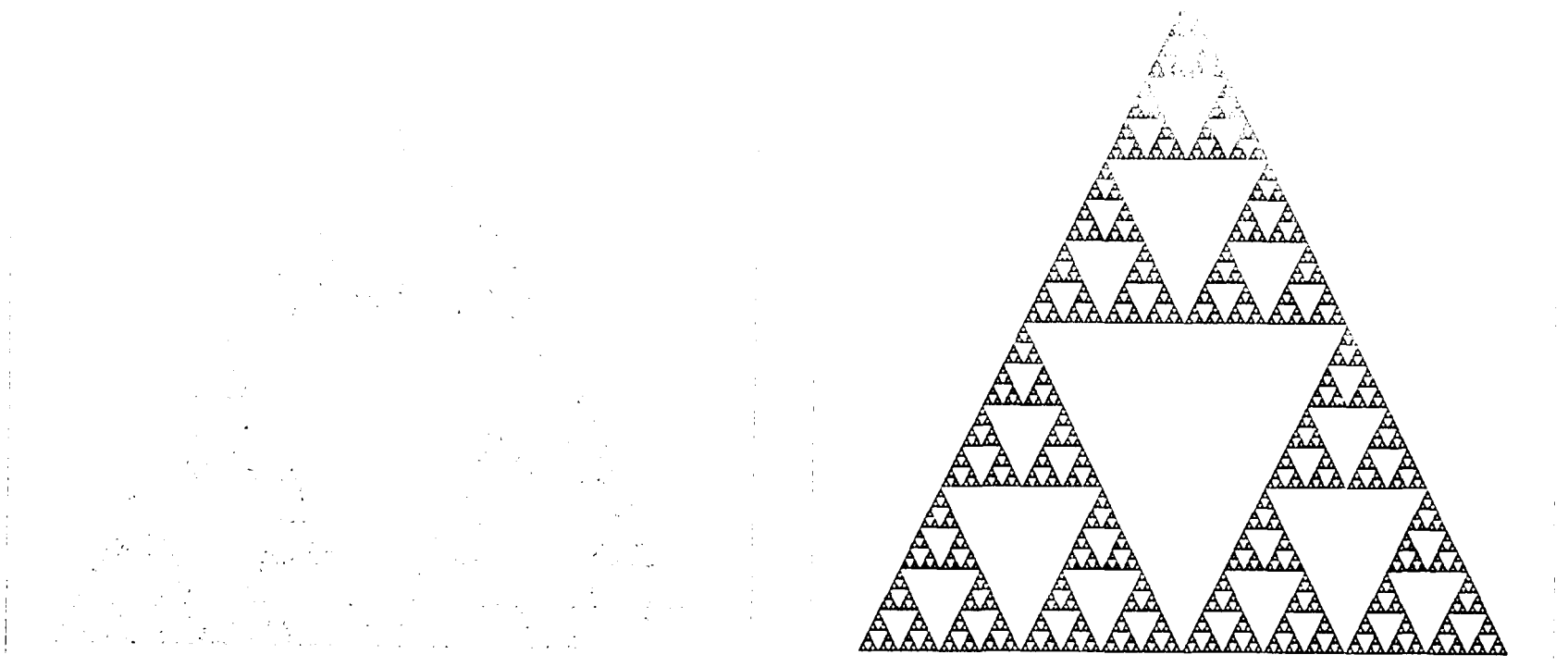


Figure 2. The Sierpinski triangle at two different stages of iteration.

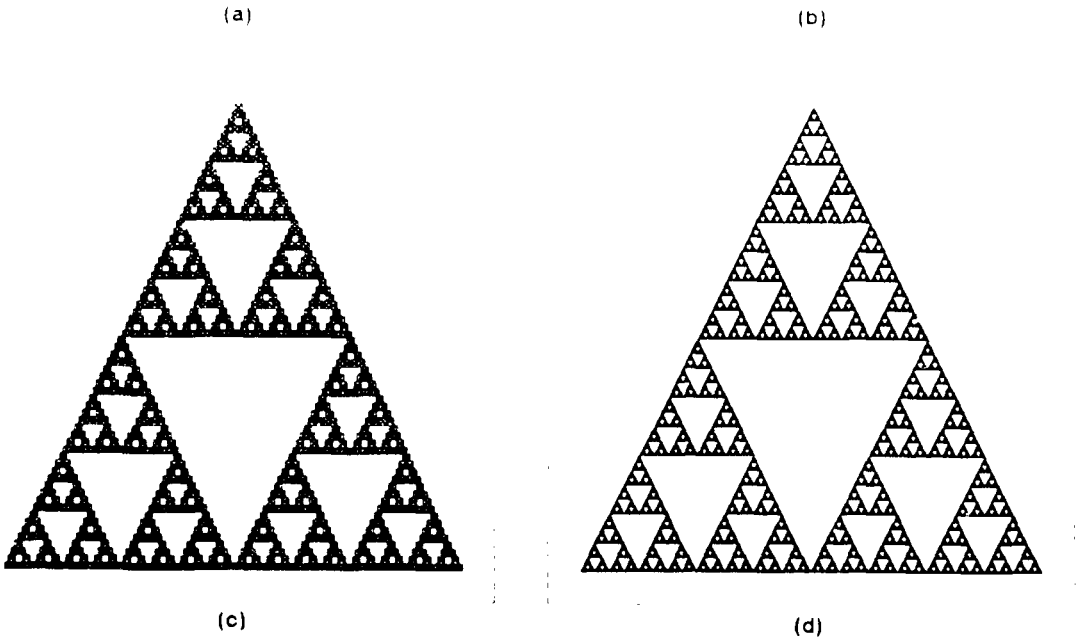


Figure 3. (a) Initial points of deterministic decoding,(b) the attractor after the first iteration, (c) the attractor after six iterations, and (d) the attractor after 11 iterations.

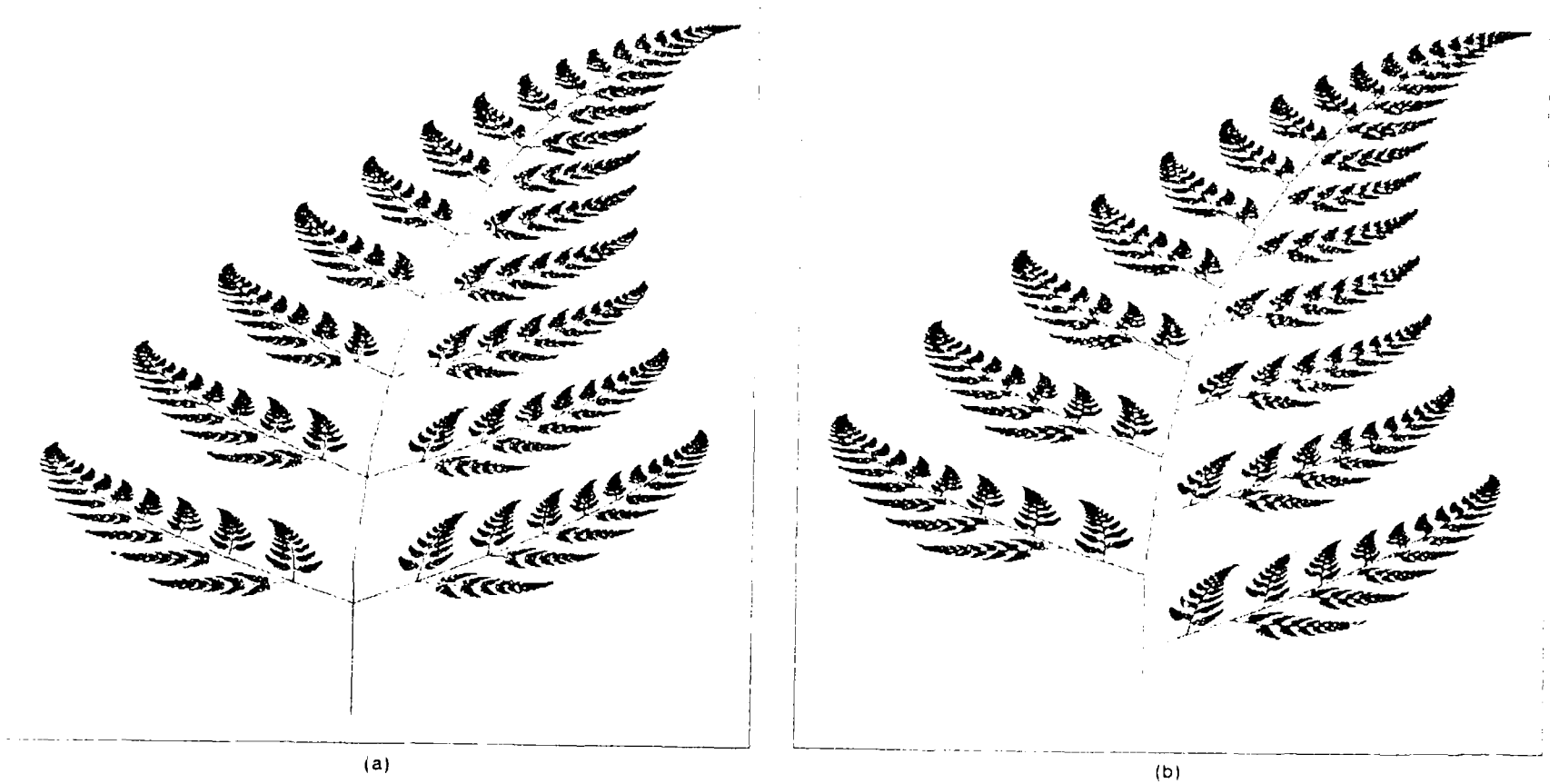


Figure 4. (a) $\overline{O}$ fern leaf and (b) $\overline{A}$ fern leaf generated with IFS codes.

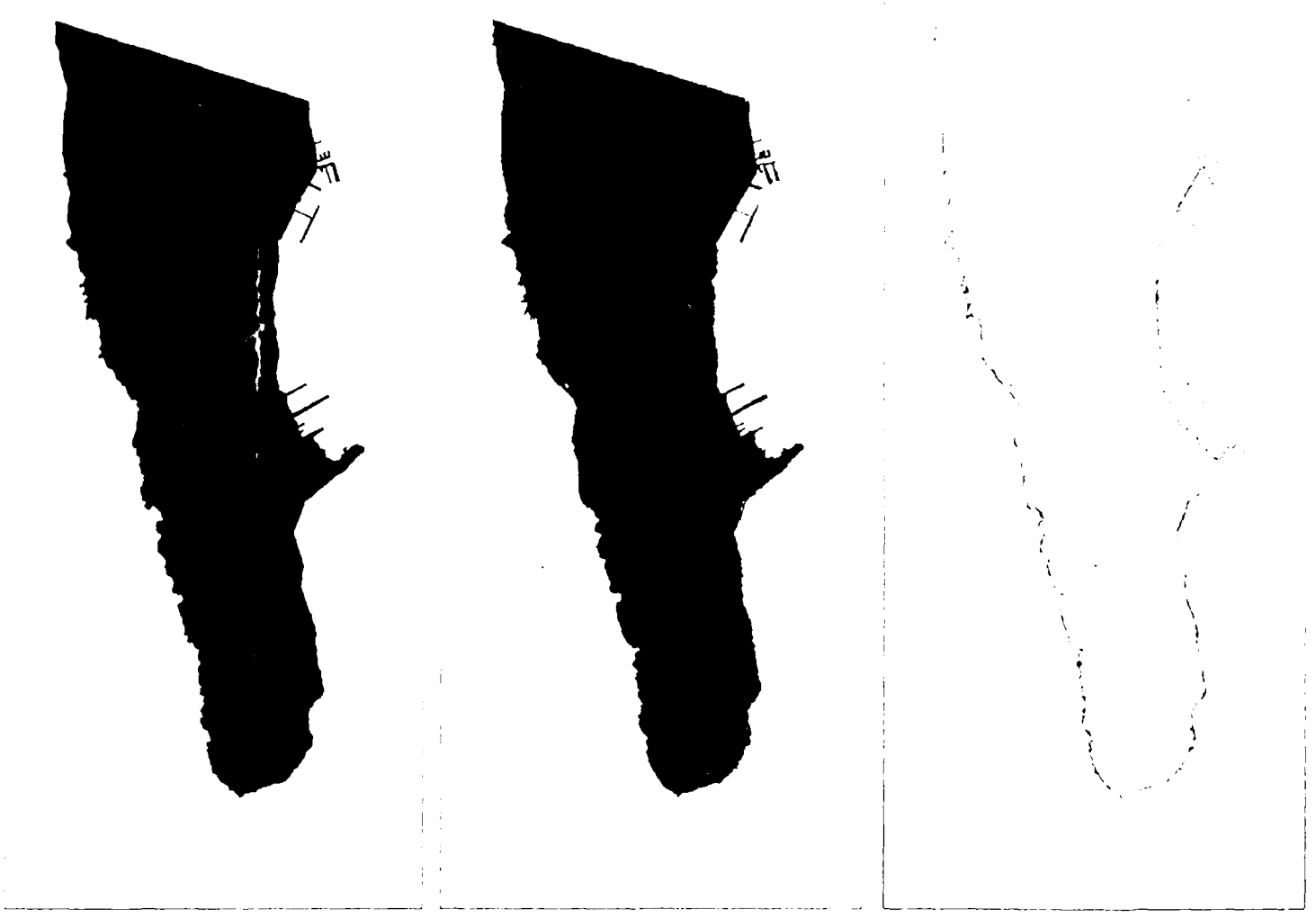
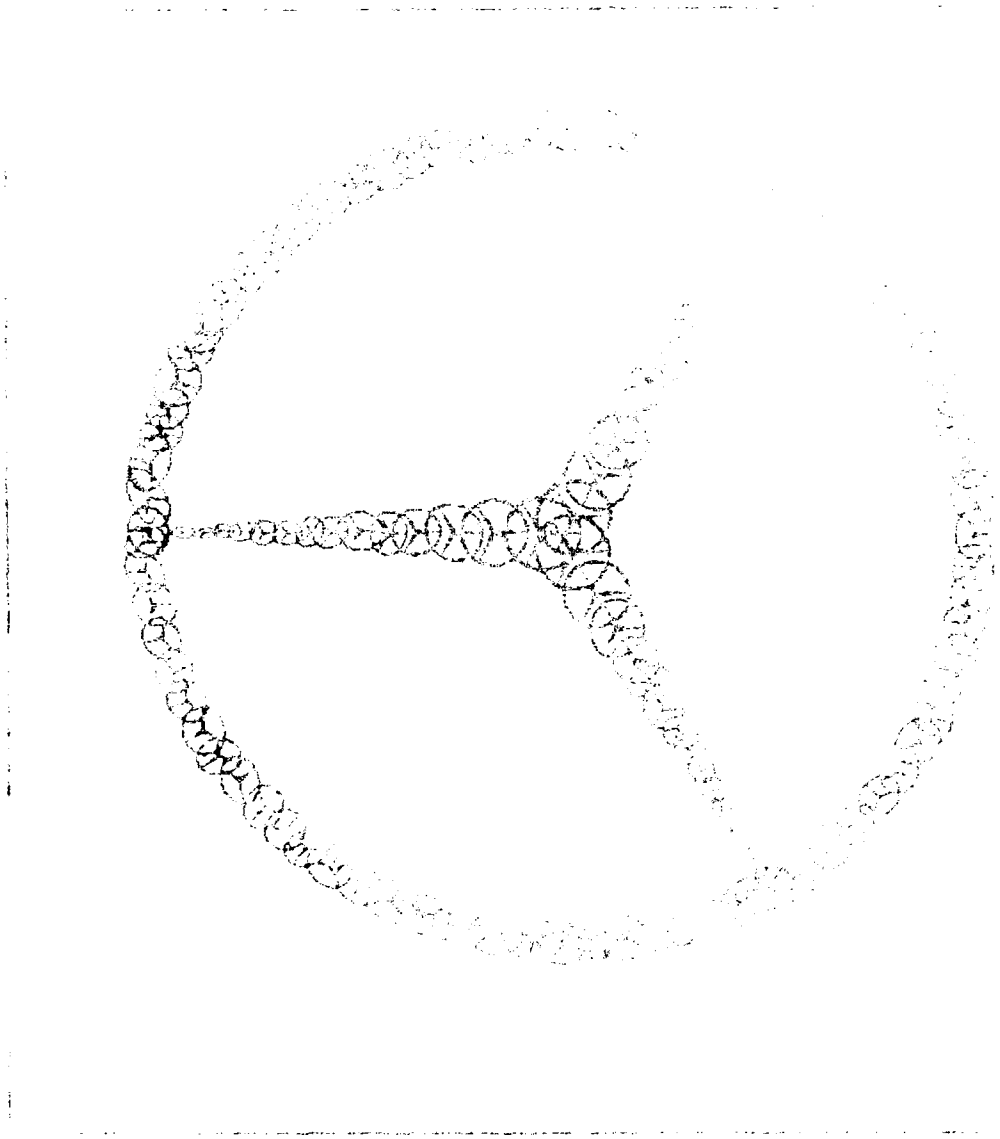


Figure 5. (a) The 960- by 428-pixel digitized map of Point Loma, (b) image of decoded transforms, and (c) absolute difference of original image and decoded image.



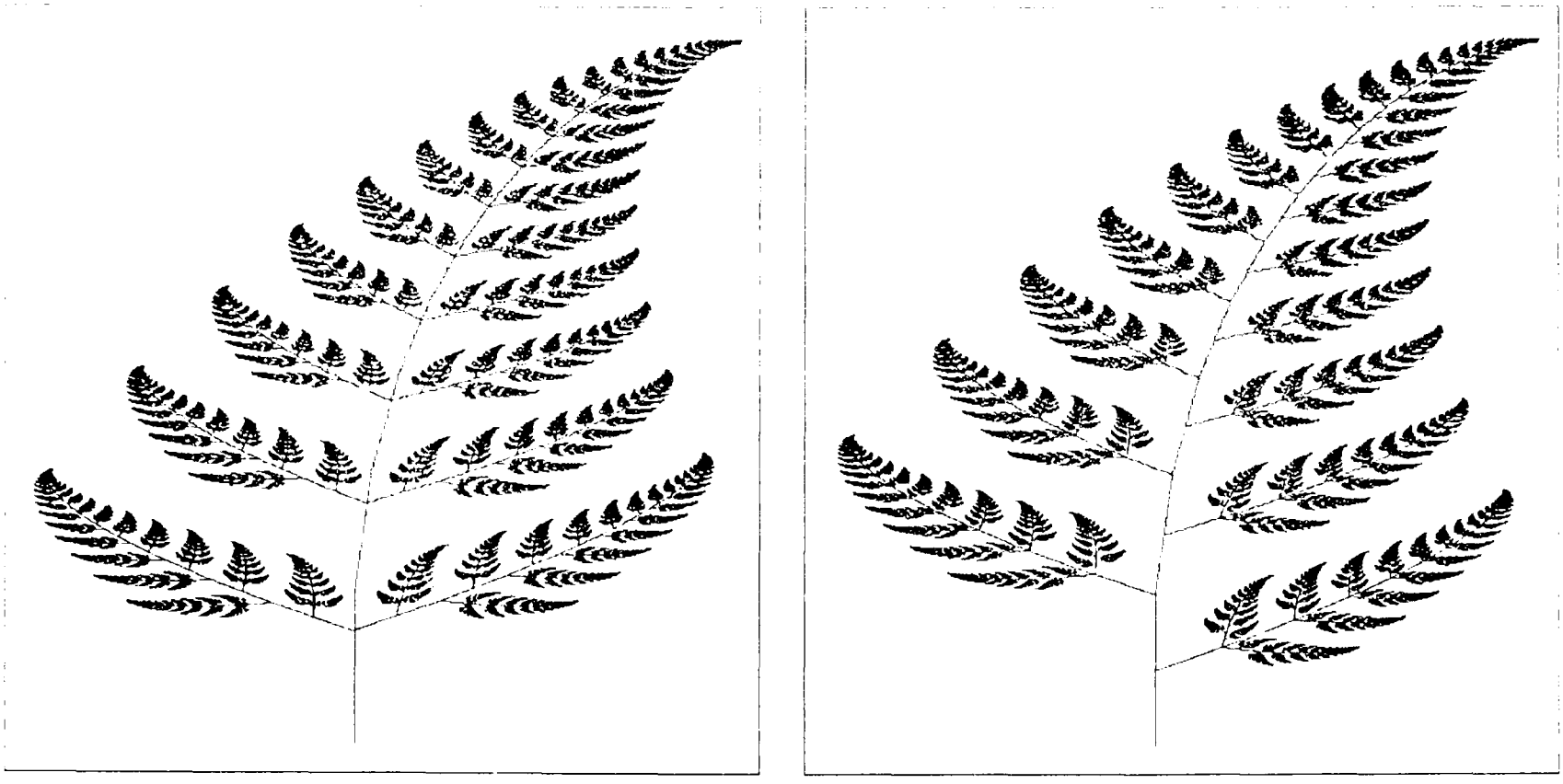


Figure 7. (a) $\overline{\mathrm{OA}}$ fern and (b) $\overline{\mathrm{AO}}$ fern generated with RIFS codes.

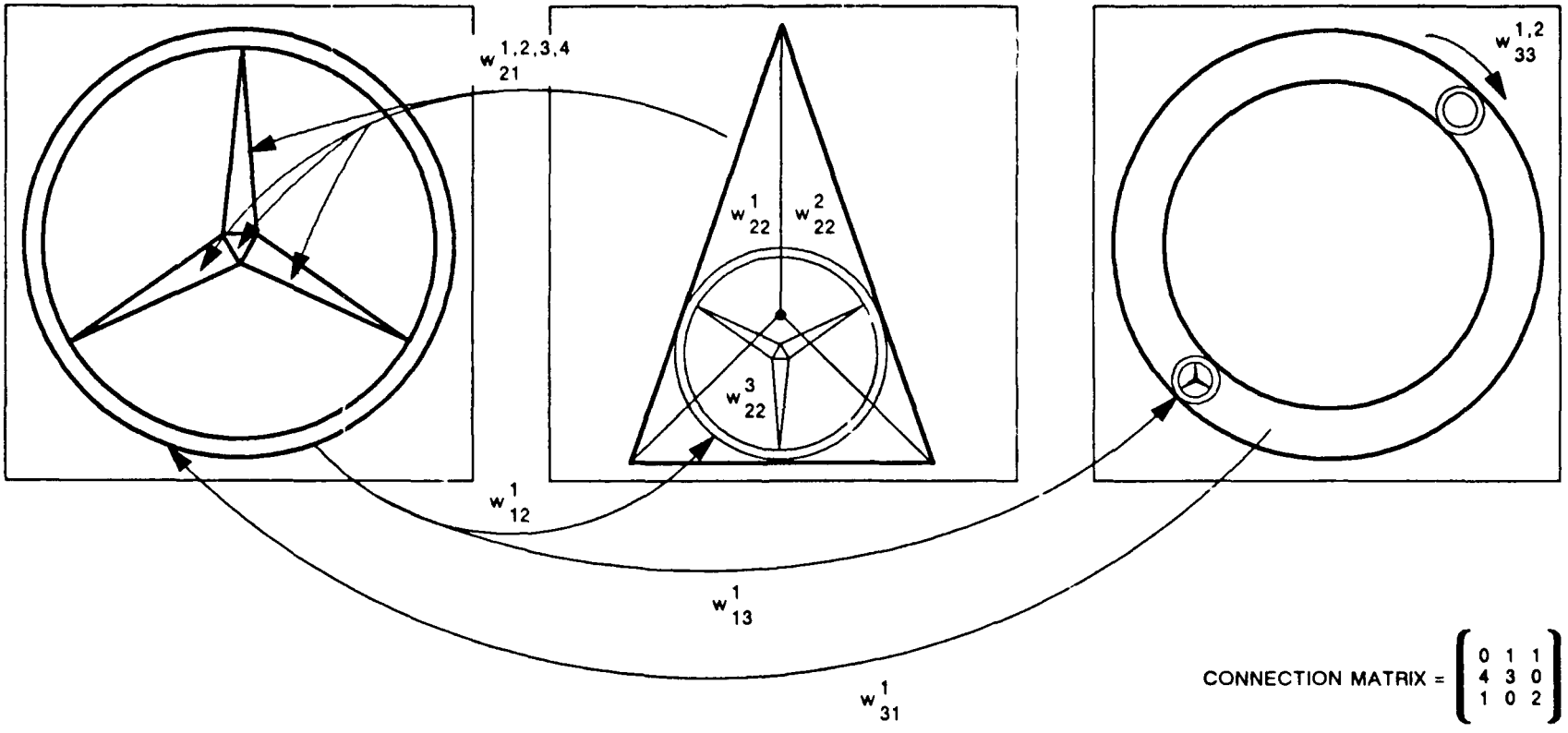


Figure 8. Construction of the Mercedes-Benz symbol using RIFS.

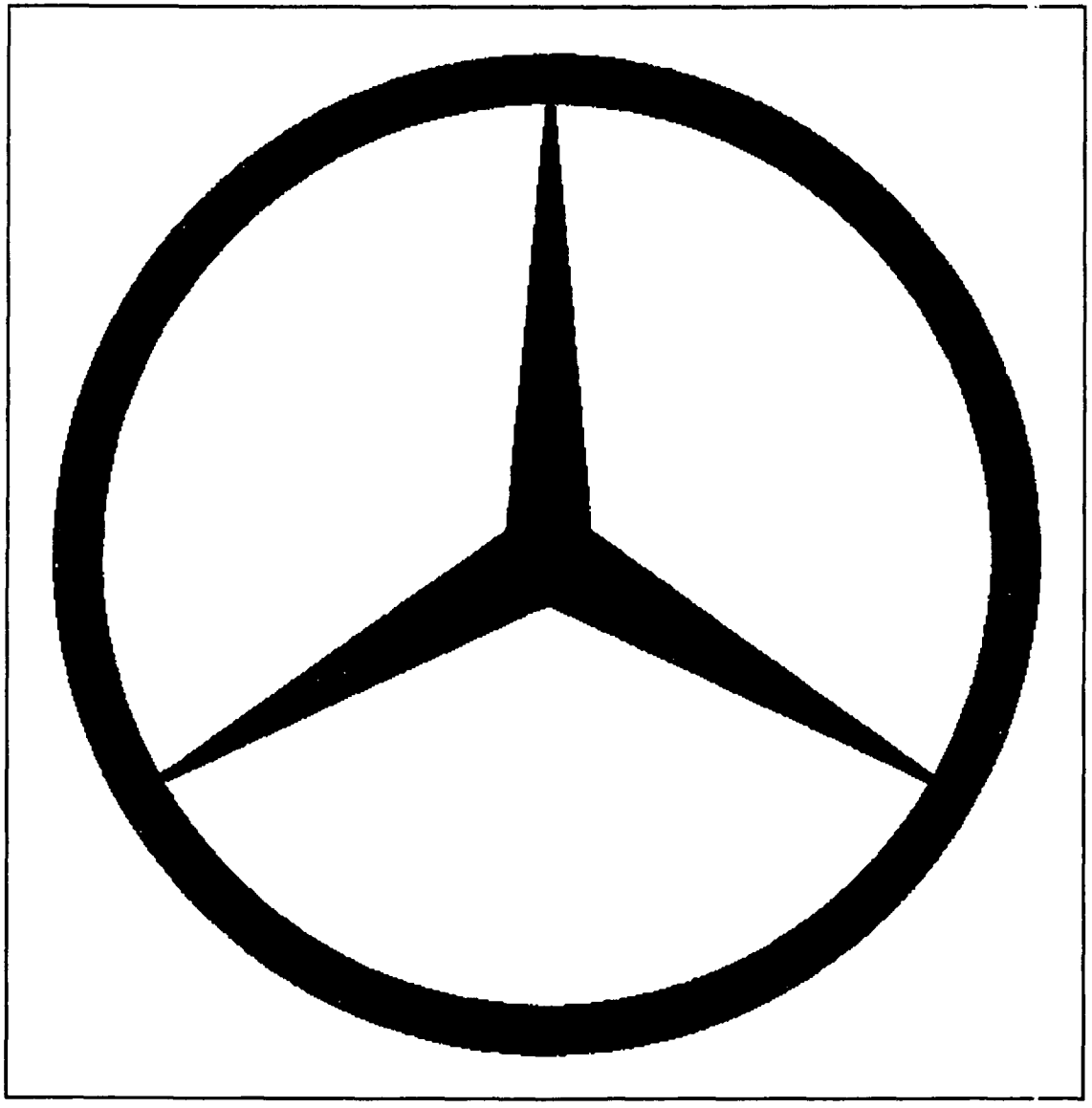


Figure 9. The regenerated perfect image of the Mercedes-Benz symbol using RIFS.

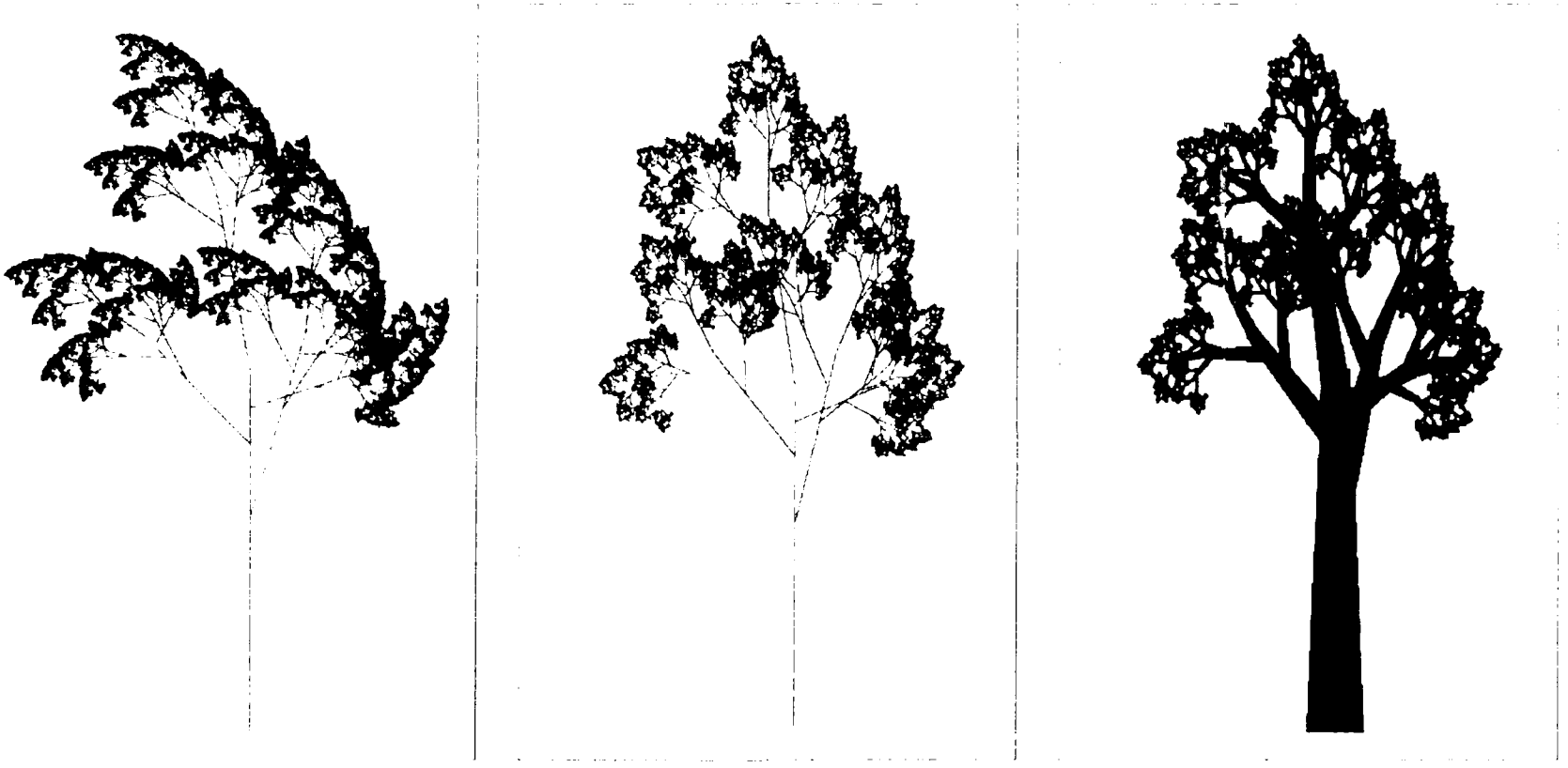
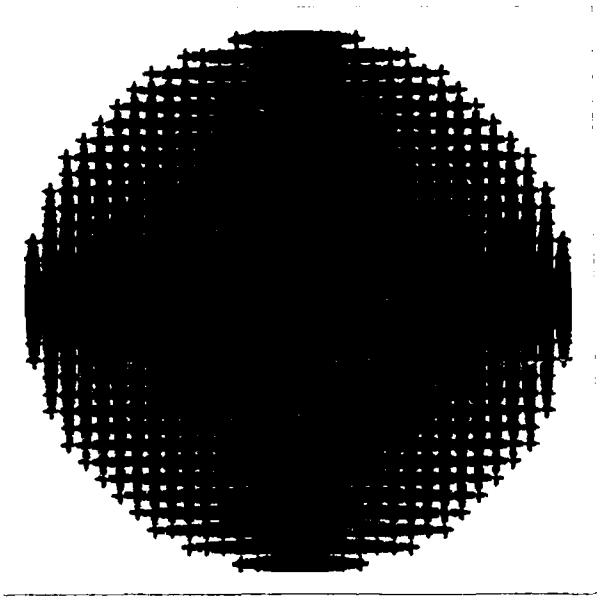
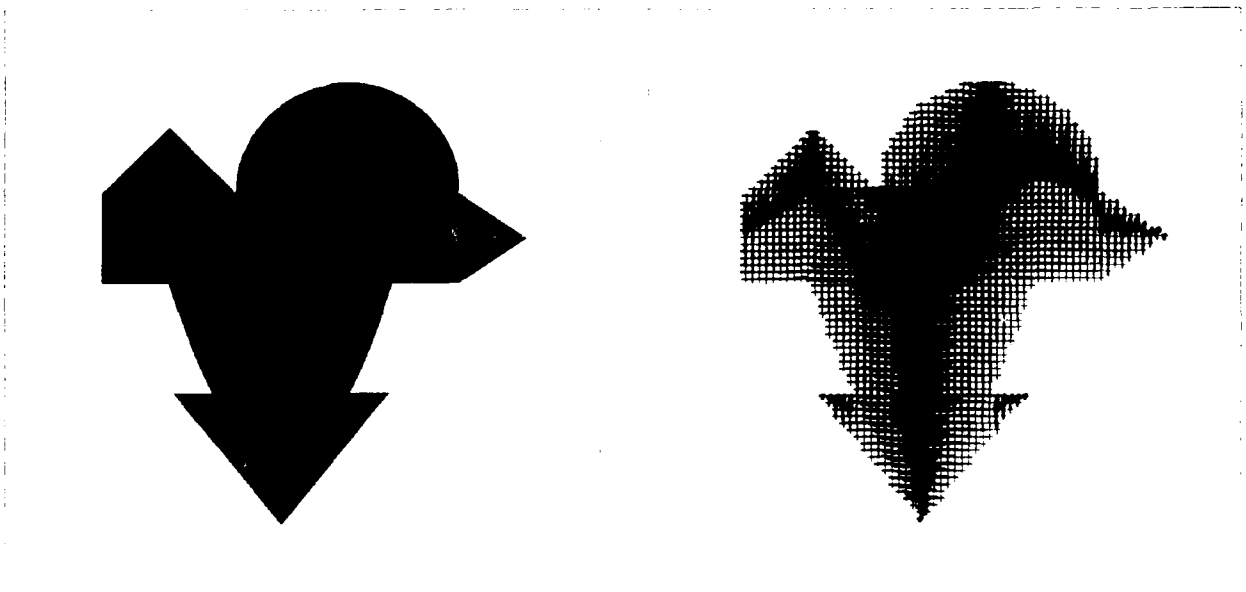


Figure 10. (a) A typical IFS-fractal tree, (b) an RIFS tree image, and (c) an RIFS tree image incorporating a trapezoidal trunk.



(a)



(b) (c)

Figure 11. (a) Decoded transforms of an automatically encoded circle and (c) decoded transforms of automatically encoded image shown in (b).

APPENDIX A. THE SET OF AFFINE TRANSFORMS FOR THE ATTRACTOR OF PT. LOMA.

a	b	c	d	e	f
0.195	-0.162	0.068	-0.198	-143.8	-73.36
0.122	0.005	0.054	-0.119	-218.6	-92.16
0.058	-0.006	-0.110	0.042	11.23	-76.87
0.029	-0.001	-0.061	0.023	33.39	-71.74
0.001	0.038	0.006	-0.087	-5.623	-100.2
0.004	0.010	-0.002	-0.023	-3.284	-89.42
0.007	-0.006	-0.018	0.006	2.225	-86.50
-0.002	0.028	-0.006	0.071	215.2	-90.24
-0.017	-0.002	-0.044	0.026	245.3	-93.96
0.007	-0.010	-0.034	0.015	263.6	-111.4
0.005	0.006	-0.024	0.008	270.1	-110.3
-0.053	0.025	0.024	0.000	204.9	-97.99
-0.025	0.008	-0.009	0.014	256.0	-123.8
-0.023	-0.001	-0.011	0.015	253.5	-115.8
-0.009	-0.001	-0.006	0.010	258.7	-108.0
0.009	0.001	0.002	-0.005	272.2	-108.2
0.009	0.001	0.003	-0.012	273.3	-112.2
0.010	-0.003	0.003	-0.009	274.5	-117.0
-0.026	-0.063	0.050	-0.029	-30.18	-135.4
0.063	-0.063	0.035	0.042	-29.46	-107.0
0.109	0.065	-0.016	0.309	353.8	118.8
-0.062	-0.942	-0.115	-0.050	218.6	109.0
0.105	0.465	-0.103	0.229	120.3	80.56
0.225	-0.066	0.007	-0.230	39.43	57.9
0.007	-0.485	-0.095	-0.031	-71.31	38.4
0.131	0.364	-0.099	0.463	-253.4	-21.57
0.092	-0.423	-0.121	-0.200	-187.9	-22.40
0.078	0.023	-0.042	0.244	-136.1	27.82
0.169	0.047	0.063	-0.240	138.7	-42.43
-0.209	-0.103	-0.016	0.115	11.61	-57.54
0.079	0.060	-0.349	0.111	284.9	27.74
-0.070	-0.110	0.175	-0.214	-63.21	-76.82
-0.131	0.166	0.052	0.128	-283.1	-80.03
0.002	-0.073	0.070	-0.029	-353.6	-46.73
0.099	0.010	-0.033	0.207	192.6	126.0
0.059	0.057	0.018	-0.076	136.9	121.6
-0.158	0.079	-0.034	-0.115	123.0	111.5
0.137	0.147	-0.203	0.141	-48.22	-12.87
0.488	-0.112	-0.091	0.174	-192.9	-55.98
0.122	0.185	-0.356	0.151	235.6	25.71
-0.142	0.738	-0.208	-0.334	193.6	34.34
0.143	-0.353	-0.048	-0.269	-314.4	-37.06
-0.044	-0.092	0.070	-0.127	-322.1	-76.05
-0.148	0.094	-0.036	-0.058	337.0	142.2
-0.443	-0.240	-0.229	0.523	197.7	64.80
0.472	0.111	0.173	-0.549	-133.1	-38.88
0.006	0.002	-0.024	0.003	285.2	-100.9
0.304	-0.351	0.047	-0.837	261.7	23.20