Fractal Block Based Image Coding Algorithms

Branka Dzerdz

A Thesis
in
The Department
of
Electrical and Computer Engineering

Presented in Partial Fulfillment of the Requirements
for the Degree of Master of Applied Science at
Concordia University
Montréal, Québec, Canada

August 1996

National Library
of Canada

Acquisitions and
Bibliographic Services Branch

395 Wellington Street
Ottawa, Ontario
K1A 0N4

Bibliothèque nationale
du Canada

Direction des acquisitions et
des services bibliographiques

395, rue Wellington
Ottawa (Ontario)
K1A 0N4

Your file Votre référence

Our file Notre référence

ISBN 0-612-18387-4

Canada

CONCORDIA UNIVERSITY
Division of Graduate Studies

This is to certify that the thesis prepared

By: **Branka Dzerdz**

Entitled: **Fractal Block Based Image Coding Algorithms**

and submitted in partial fulfilment of the requirements for the degree of

Master of Applied Science

complies with the regulations of this University and meets the accepted standards with respect to originality and quality.

Signed by the final examining committee:

_____ Dr. O. Schwelb
_____ Dr. M. N. S. Swamy
_____ Dr. E. I. Plotkin
_____ Dr. T. Fancott
_____ Dr. M. O. Ahmad

Approved by _____

_____ 19 ____ _____

Dean of Faculty

ABSTRACT

Fractal Block Based Image Coding Algorithms

Branka Dzerdz

The application of fractal theory to image encoding problems has seen a significant development in the past couple of years. However, there are still numerous questions about practical implementations that make this field interesting for research. This work is an attempt to address some of these problems.

In the first part of this investigation, the effect of different image block partitioning schemes to the performance of the existing fractal algorithms is analyzed. A new adaptive fractal coding algorithm is proposed. The algorithm is based on the partitioning of an image into rectangular blocks. The partitioning is done in an adaptive manner, aimed to increase the probability of finding the similarities between the resulting partitioned image blocks. The partitioned image thus obtained is used to build both the range and the domain libraries. The performance of the proposed algorithm is compared to that of the standard fractal coding schemes. Significant improvements are achieved both in terms of encoding time and the quality of the encoded images.

The thesis also introduces a new, hierarchical interpretation for fractal image coding techniques. Fractal coding is presented as a refinement of details on a fine scale from the information obtained on a coarse scale. This approach can be applied to practically any of the existing fractal block coding schemes. An hierarchical encoding algorithm is developed to illustrate the concept. It has been shown that the encoding time can be improved by first performing a full range-domain search on a coarse image representation only, and then propagating the information thus obtained to finer scales.

Mami i tati

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to my supervisor Dr. M. O. Ahmad, for providing support and guidance that made this work possible. I am grateful for his patience and devotion to his students. I feel privileged for having the opportunity to work with him. I would also like to thank the members of the examination committee, Dr. O. Schwelb, Dr. E. I. Plotkin, Dr. M. N. S. Swamy and Dr. T. Fancott for their most helpful comments and suggestions.

I would also like to express my gratitude to my family, for their indisputable love and support. Pod familijom podrazumevam sve divne, nesebicne ljude koji su ugradili u mene deo sebe i pratili me na mom putu.

This work was supported by a scholarship from the Natural Sciences and Engineering Research Council of Canada and by a Concordia University Graduate Fellowship.

TABLE OF CONTENTS

LIST OF SYMBOLS

$d(\cdot,\cdot)$	Metric (distance) Function
$h_d(\cdot,\cdot)$	Hausdorff metrics
(X,d)	Metric space
inf	Infimum
sup	Supremum
arg	Argument
d_{sup}	Supremum metric
d_{rms}	RMS metric
I^2	Unit square; $I^2 = \{(x.y) \mid 0 \leq x,y \leq 1\}$
w_i	Single contractive transformation
D_i	Domain block for the transformation w_i
R_i	Range block for the transformation w_i
D	Domain library
w^{on}	n-th iterate of transformation w
s	Contractivity factor for a transformation or for a map
s_f	Scaling factor
s_{max}	Maximum allowed value for s_f
o_f	Offset factor
W	Contractive map
f	An image represented as a 2D function
f_{org}	Original image
f_{dec}	Decoded image
g_{max}	Maximum gray-llevel value within an image
x_W, f_W	Fixed-point of the contractive map W
x_0, f_0	Arbitrary initial image
R_v	Vertical size of a range block
R_h	Horizontal size of a range block
h_i	Grey level difference between two consecutive rows
v_j	Grey level difference between two consecutive columns
hd_i	Difference between average grey levels in two areas in a horizontally divided image

vd_j	Difference between average grey levels in two areas in a vertically divided image
B_1	Range block size at a scale on which PIFS code has been obtained
B_q	Size of a range block at q times reduced original scale
f^q	Fixed-point of a PIFS at q times reduced original scale
W^q	Contractive map for a PIFS code at q times reduced original scale
$S(\cdot)$	Scaling function

LIST OF ACRONYMS

DCT	Discrete Cosine Transform
HE	Hierarchical Encoding
HV	Horizontal Vertical
IFS	Iterated Function System
JPEG	Joint Photographic Experts Group
MPEG	Moving Pictures Expert Group
PIFS	Partitioned Iterated Function System
PSNR	Peak Signal to Noise Ratio
QD	Quad-tree
RMS	Root Mean Square

LIST OF FIGURES

Chapter 1

Introduction

1.1 General

Bandwidth compression remains one of the most important issues when considering the problems of data storage and transmission. Ever increasing roles of audio and video signals in our daily lives is the main motivation for the research and development of new and more efficient data compression techniques.

Standardization of the block-coding techniques for image and video data that utilize discrete cosine transformation (DCT) closes a period in which millions of working hours of researchers have been devoted to optimize the performance of these methods. Although they are already accepted as industry standards (JPEG for still images and MPEG for video), there is ongoing research in alternative methods, which are aiming to overcome some of the limitations that are imposed to block-based DCT methods. Image coding motivated by the fractal theory is one of them.

The initial ideas for the fractal image coding were presented in the visionary work of M. Barnsley [3, 4, 2], which was entirely of a theoretical nature. These ideas did not lead to any automated algorithms, they were rather more suitable for interactive computer programs requiring an intelligent human operator. This was the main source of the initial

skepticism for the Barnsley's theory, which was expressed in a famous anecdote about a graduate student assignment. The anecdote goes like this: to solve the difficult problem of finding self-similarities in an image (which is the central point of the fractal-based coding algorithms) one proceeds as follows:

1. Find a quiet room with nothing inside but a workstation.
2. Get a graduate student.
3. Lock the student in.
4. Do not allow him/her to go out until he/she comes up with a solution.

Ironically, it was the work of Barnsley's former graduate student A. Jacquin, that gave some answers regarding the practical application of the fractal coding theory to the real gray-level images [15],[16]. This first algorithm is considered to be *generic*, since it commenced the whole new branch in image coding techniques. Since then numerous improvements and variations of the basic algorithm have been reported, such as in [23, 22, 9, 11] to mention just a few.

The basis of all these algorithms is the so called self-similarity property observed in deterministic fractal objects. Deterministic fractals are objects of huge visual complexity and infinite resolution. Magnification of a part of a fractal object, gives an exact copy of the whole object. In other words, parts of the object are similar to the whole object. Thus, we say that the whole fractal is composed of the scaled copies of itself. However, this highly structured object can be fully described by a single set of equations, or *generating rules*. These rules describe relations between the building blocks that form the fractal object, i.e., they describe the observed regularity in the object. When the generating rules are applied recursively to an arbitrarily chosen *initiator* object, a unique, complex object, an *attractor* is produced. The attractor is an exact copy of the fractal. Since the result (attractor) of this generating process does not depend on the initiator, the set of the generating rules contains sufficient information about the fractal object and can be used as a compact description for it.

The problem of finding the generating rules, is called the *inverse problem*. The objective here is to find similarities in the object and describe them with a minimal number of relations.

Figure 1.1. shows a classical deterministic fractal object called Sierpinski Gasket. This object can be characterized by a set of three basic transformations, each including scaling by a factor 0.5 and a translation, as given below. Iterative process for constructing this object is illustrated in Figure 1.2.

$$w_1 \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

$$w_2 \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0.5 \\ 0 \end{bmatrix}$$

$$w_3 \begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} 0.25 \\ 0.5 \end{bmatrix}.$$

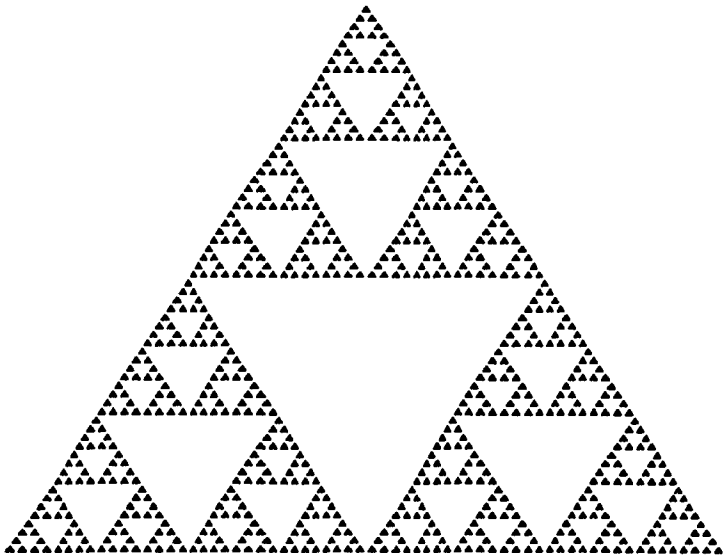


Figure 1.1: Sierpinski Gasket: a classical deterministic fractal object.

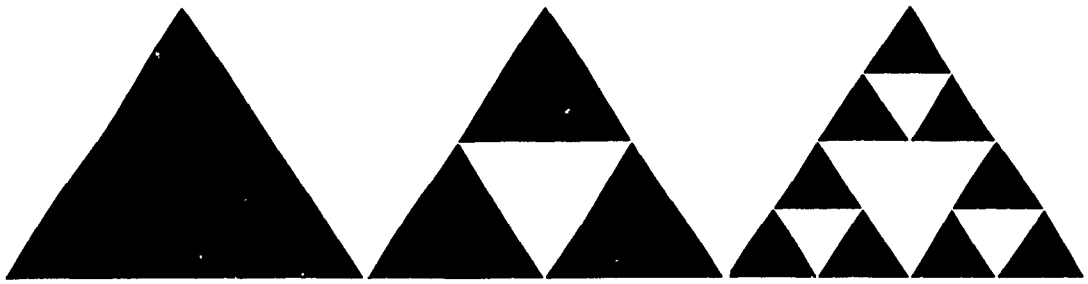


Figure 1.2: Iterative generation of Sierpinski Gasket.

Fractal image coding algorithms assume that real images posses some form of the self-similarity and that they as well can be described by some sets of generating rules, analogically to deterministic fractals. Finding strict self-similarity in real-world images represented by the gray levels is too much to hope for. What is more realistic is to find restricted self-similarities: instead of viewing the image as a composition of the scaled copies of itself, we can try to find similarities between the restricted parts of the image. In this case, the generating rules include the similarity relations between distinct parts of the image. These relations are not restricted to spatial relations only, but also include transformations performed in the gray-level domain. Figure 1.3 shows some similar parts in a real gray-level image. All three marked regions are alike, and can be transformed in a way that enhances the similarity. Transformations would also include spatial scaling (resizing), which could be done by pixel averaging or simply by sub-sampling, and some form of nonlinear modification in the gray-level domain. The transformations obtained would form a part of the fractal code (generating rules) for the image. If our goal is to achieve compression, we have to assure that storing the generating rules for the image will require less memory space than storing the image itself.

Recovering the image from the generating rules, in this case, will be identical to the same process as for deterministic fractals: we start with an arbitrary initiator image, and apply all the transformations to it. We proceed iteratively, and apply the transformations to the resulting image. Every iteration adds up some more detail. We stop when further

refinement in detail becomes too small for a given discrete grid.

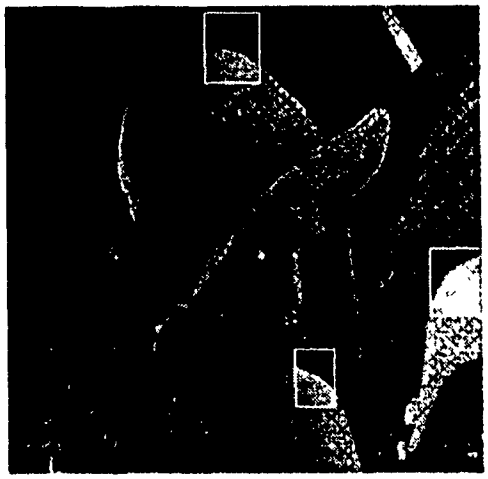


Figure 1.3: Similar regions within an real gray-level image.

The recovery process, i.e., generating the image from the rules, is fairly straight forward. It is the inverse problem, i.e., finding the generating rules, which is challenging. It is easy for a human to detect the similarities between different objects, but this problem is very difficult to be solved by a machine in an automated manner. Furthermore, detecting similarities is not the only task here, it is just one part in the coding algorithm. Thus, we cannot afford complicated and time consuming procedures from the pattern recognition or computer vision techniques. The next section gives some basic mathematical background and notations that will be used throughout this thesis.

1.2 Mathematical Background

1.2.1 Fractals

There is no strict definition of the term fractal. There is rather a set of properties that an object is expected to have in order for it to be considered a fractal. These properties are:

1. The object has details at every scale.

2. The object is (strictly, approximately or statistically) self-similar.

3. There is an algorithmic description of the object.

From the above stated properties of fractals, in image coding applications we concentrate on the first three, with the third property constituting as the main motivation behind using the fractal techniques. In the fractal-based image coding schemes, as it was mentioned in the previous section, we view an image as a fractal object. The main coding problem is to find a simple description of the regularities that exist within the image.

Most of the fractal-based image coding techniques are based on the iterated function systems theory (IFS) and its generalizations. The IFS was introduced by M. Barnsley in [3]. This is a general theory, applicable to sets and was initially applied to binary images viewed as sets of points in a plane. The following section gives a review of the IFS theory, constituting the necessary mathematical background for the fractal-based image coding applications.

1.2.2 Iterated Function Systems

First, we introduce the necessary concepts that form the foundations of the IFS theory.

Let X be a set that is a vector space.

Definition 1.1 *A real valued function $d : X \times X \rightarrow R$ is called a* metric *on X if and only if the following conditions are satisfied:*

(1) $\forall x, y \in X : d(x,y) \geq 0, \qquad d(x,x) = 0.$

(2) $\forall x, y \in X : d(x,y) = d(y,x).$

(3) $\forall x, y, z \in X : d(x,z) \leq d(x,y) + d(y,z).$

(4) $d(x,y) = 0 \Rightarrow x = y.$

The pair (X,d) is called the *metric space.*

Definition 1.2 *A sequence of points* $\{x_n\}$ *in a metric space* (X, d) *is called Cauchy sequence if for any real* $\varepsilon > 0$*, there exist an integer* N *such that:*

$$\forall m, n > N : d(x_m, x_n) < \varepsilon.$$

Definition 1.3 *A sequence of points* $\{x_n\}$ *in a metric space* (X, d) *is said to have a limit* $a, a \in X$*, if for any real* $\varepsilon > 0$*, there exists an integer* N *such that:*

$$\forall n > N : d(x_n, a) < \varepsilon.$$

Definition 1.4 *A metric space* (X, d) *is* complete *if every Cauchy sequence in* X *converges to a limit point in* X*.*

Definition 1.5 *Let* $S \subset X$ *be a subset of a metric space* (X, d)*. A point* $x \in X$ *is called a* limit point *of* S *if there is a sequence* $\{x_n\}$ *of points* $\{x_n\} \in S \setminus \{x\}$ *such that* $\lim_{n\to\infty} x_n = x$*.*

Definition 1.6 *Let* $S \subset X$ *be a subset of a metric space* (X, d)*. The* closure *of* S*, denoted by* $\overline{S}$ *is defined as* $\overline{S} = S \cup \{$ *limit points of* $S \}$*.* S *is closed if it contains all of the limit points, i.e.,* $S = \overline{S}$*.*

Definition 1.7 *Let* $S \subset X$ *be a subset of a metric space* (X, d)*.* S *is* bounded *if there is a point* $a \in X$ *and a number* $T > 0$ *so that*

$$\forall x \in S : d(a, x) < T.$$

Definition 1.8 *A metric space* (X, d) *is* compact *if* X *is closed and bounded.*

Let $\mathcal{H}(X)$ denote the set of all compact subsets of X, i.e., $\mathcal{H}(X) = \{S \subset X | S$ is compact$\}$. For a $A \in \mathcal{H}(X)$, we construct a set of points that are of maximal distance ε from A, as measured by the distance function d : $A_d(\varepsilon) = \{x | d(x, y) \leq \varepsilon, y \in A\}$. We now introduce a new distance function that measures the distance between two sets A, B, which are elements of $\mathcal{H}(X)$ (that is they are compact subsets of the X) as:

$$h_d(A, B) = max\{inf\{\varepsilon | B \subset A_d(\varepsilon)\}, inf\{\varepsilon | A \subset B_d(\varepsilon)\}\}$$

.

The metric $h_d(A, B)$ is called the *Hausdorff distance* between A and B. Index d indicates that h_d depends on the choice of the metric d in the space X.

Theorem 1.1 $(\mathcal{H}(X), h_d)$ *is a metric space.*

The proof of this can be found in [3].

Since $\mathcal{H}(X)$ consists of the compact subsets of X, $(\mathcal{H}(X), h_d)$ is a complete metric space. This is important, since our goal is to describe a given image (subset of the plane) as the limit point of a sequence of images that result from a iterated application of some map defined in the space of images. By having that space $(\mathcal{H}(X), h_d)$ complete, we are guaranteed that the limit point of the sequence of images will also be an image, given that the sequence is a Cauchy sequence. There is another important requirement for the limit point for a sequence of images to exist: the maps that describe images should be contractive.

Definition 1.9 *Let* (X, d) *be a metric space. A map* $w : X \to X$ *is* Lipschitz *with Lipschitz factor* $s \in R, s > 0$ *if*

$$\forall x, y \in X : d(w(x), w(y)) \leq sd(x, y) \quad .$$

If the Lipschitz factor $s < 1$, *then the map* w *is contractive with contractivity factor* s.

All Lipschitz maps are continuous, since if $d(w(x), w(y)) \leq sd(x, y)$ and as x, y get closer $(d(x, y) \to 0)$, then $d(w(x), w(y))$ also becomes small.

Theorem 1.2 The Contractive Mapping Fixed-Point Theorem: *Let* (X, d) *be a complete metric space and* $w : X \to X$ *be a contractive mapping. Then there exists a unique point* $x_w \in X$ *such that for every* $x \in X$

$$x_w = w(x_w) = \lim_{n \to \infty} w^{\circ n}(x)$$

The unique point x_w is called the fixed-point of the mapping w. In the theorem above, $w^{\circ n}(x) = \underbrace{w(w(\ldots w(x)))}_{n \text{times}}$.

Proof: First we prove the existence of the fixed-point. For an $x \in X$ and for $n, m \in N$ such that $n > m$

$$\begin{aligned} d(w^{\circ m}(x), w^{\circ n}(x)) &\leq s d(w^{\circ m-1}(x), w^{\circ n-1}(x)) \\ &\leq s^m d(x, w^{\circ n-m}(x)) \end{aligned} \tag{1.1}$$

Using the triangle inequality for the metric $d(\cdot,\cdot)$, for any $k \in N$

$$\begin{aligned} d(x, w^{\circ k}(x)) &\leq d(x, w^{\circ k-1}(x)) + d(w^{\circ k-1}(x), w^{\circ k}(x)) \\ &\leq d(x, w(x)) + d(w(x), w(w(x))) + \ldots + d(w^{\circ k-1}(x), w^{\circ k}(x)) \\ &\leq (1 + s + s^2 + \ldots, + s^{k-1}) d(x, w(x)) \\ &\leq \frac{1}{1-s} d(x, w(x)) \end{aligned} \tag{1.2}$$

Now, (1.1) becomes

$$d(w^{\circ m}(x), w^{\circ n}(x)) \leq \frac{s^m}{1-s} d(x, w(x)) \tag{1.3}$$

Since $s < 1$, for sufficiently large m, n, the right side of (1.3) can be as small as necessary, i.e., the sequence $x, w(x), w(w(x)), \ldots, w^{\circ n}(x)$ is a Cauchy sequence. Since X is complete, the limit point of that sequence is also in X. Denote this limit point by x_w. Since w is Lipschitz, it is continuous and therefore the following holds:

$$w(x_w) = w(\lim_{n\to\infty} w^{\circ n}(x)) = \lim_{n\to\infty} w^{\circ n+1}(x) = x_w.$$

To prove the uniqueness of the fixed-point, we assume that there are two fixed-points for w, x_{w1} and x_{w2}. Then, $d(w(x_{w1}), w(x_{w2})) = d(x_{w1}, x_{w2})$. On the other hand, $d(w(x_{w1}), w(x_{w2})) \leq s d(x_{w1}, x_{w2})$, $s < 1$. This can hold only for $x_{w1} = x_{w2}$.

From (1.2), as $k \to \infty$, we have

$$d(x, x_w) \leq \frac{1}{1-s} d(x, w(x)) \qquad \square \tag{1.4}$$

Also, the result given by (1.4) is called the *Collage Theorem.*

From the proof of Theorem 1.2, it can be observed that it is not necessary for w to be contractive in order to have a fixed-point. It is sufficient that some iterate of w be contractive to guarantee the existence of the fixed-point. This would lead us to the definition of eventually contractive map and to the generalization of the Theorem 1.2.

Definition 1.10 *Let w be a Lipschitz function. If there is a number n such that $w^{\circ n}$ is contractive, then w is said to be* eventually contractive. *n is called the exponent of eventual contractivity.*

Theorem 1.3 Generalized Contractive Mapping Fixed-Point Theorem: *Let w be eventually contractive with an exponent n. Then, there exists a unique fixed-point $x_w \in X$ such that for any $x \in X$*

$$x_w = w(x_w) = \lim_{k\to\infty} w^{\circ k}(x)$$

and

$$d(x, x_w) \leq \frac{1}{1-s}\frac{1-\sigma^n}{1-\sigma} d(x, w(x)),$$

where s is the contractivity of $w^{\circ n}$ and σ is the Lipschitz factor of w.

The proof of Theorem 1.3 can be found in [13].

Usually a map W that represents an image consists of a collection of simple maps:

$$W(\cdot) = \bigcup_{i=1}^{N} w_i(\cdot) \tag{1.5}$$

where $w_i : R^2 \to R^2, i = 1, \ldots, N$ and $W : \mathcal{H}(R^2) \to \mathcal{H}(R^2)$. For W so defined , the contractivity condition is determined by the following theorem.

Theorem 1.4 *If $w_i : R^2 \to R^2$ is contractive with the contractivity factor s_i for $i = 1, \ldots, N$, then $W = \cup_{i=1}^{N} w_i : \mathcal{H}(R^2) \to \mathcal{H}(R^2)$ is contractive in the Hausdorff metric with the contractivity $s = max_{i=1,\ldots,N}\{s_i\}$.*

Proof: Let $A, B \in \mathcal{H}(R^2)$, and $\varepsilon = h_d(A, B)$. Then, $A \subset B_d(\varepsilon)$ and $B \subset A_d(\varepsilon)$. Let

$$
\begin{aligned}
s &= max_{i=1,\ldots,N}\{s_i\} \\
A_i &= w_i(A) \\
B_i &= w_i(B) \\
A' &= W(A) = \cup_{i=1,\ldots,N} w_i(A) \\
B' &= W(B) = \cup_{i=1,\ldots,N} w_i(B).
\end{aligned}
$$

We want to show that $h_d(W(A), W(B)) \leq s h_d(A, B)$, or in other words $B' \subset A'_d(s\varepsilon)$ and $A' \subset B'_d(s\varepsilon)$. We have

$$B \subset A_d(\varepsilon) \text{ and } w_i \text{ is contractive} \Rightarrow B_i \subset w_i(A_d(\varepsilon)),$$

and

$$w_i(A_d(\varepsilon)) \subset A_{id}(s_i\varepsilon) \subset A_{id}(s\varepsilon) \Rightarrow B_i \subset A_{id}(s\varepsilon).$$

In the same manner, $A_i \subset B_{id}(s\varepsilon)$. Thus,

$$A' = \bigcup_{i=1}^{N} A_i \subset \bigcup_{i=1}^{N} B_{id}(s\varepsilon) = B'_d(s\varepsilon),$$

and

$$B' = \bigcup_{i=1}^{N} B_i \subset \bigcup_{i=1}^{N} A_{id}(s\varepsilon) = A'_d(s\varepsilon). \qquad \square$$

Definition 1.11 *An* iterated function system (IFS) *consists of a complete metric space* (X, d) *and a finite set of contractive mappings* $w_i, i = 1, \ldots, N$ *with respective contractivity factors* s_i. *The contractivity factor for the IFS is given by* $max_{i=1,\ldots,N}\{s_i\}$.

We can now summarize the results of Theorems 1.2 and 1.4. For a given IFS with the contractivity factor s, the map $W : \mathcal{H}(X) \to \mathcal{H}(X)$ defined by

$$W(S) = \bigcup_{i=1}^{N} w_i(S), \qquad S \in \mathcal{H}(X)$$

is a contractive mapping on the complete metric space $(\mathcal{H}(X), h_d)$, with the contractivity factor s, i.e.,

$$h_d(W(A), W(B)) \leq sh_d(A, B), \forall A, B \in \mathcal{H}(X).$$

There exists a unique fixed-point for W, $x_W \in \mathcal{H}(X)$, such that

$$x_W = W(x_W) = \bigcup_{i=1}^{N} w_i(x_W)$$
$$x_W = \lim_{k \to \infty} W^{ok}(S)$$

for any $S \in \mathcal{H}(X)$. This fixed-point is also called the *attractor*.

The uniqueness of the attractor guarantees that it is completely specified by the map W. It is clear that being given a map W, the corresponding attractor can be constructed. The more interesting question, however, is the inverse one. That is, weather it is always possible to find a map W for a given attractor. This question has direct bearing on image coding applications, but a specific answer to it does not yet exist. However, there are two pointers emerging from our discussion, that are useful in image coding applications. The first is that the fixed-point equation

$$x_W = W(x_W) = w_1(x_W) \cup, \ldots, \cup w_N(x_W)$$

says that the attractor is composed of the transformed copies of itself. Thus, our goal in searching for the map W should be that of determining the contractive transformations w_i such that the union of the images $w_i(S)$ yields the initial image S. The second pointer is that the Collage Theorem

$$d(S, x_W) \leq \frac{1}{1-s} d(S, W(S)) \tag{1.6}$$

gives an estimate (unfortunately a very conservative one) for the distance between the initial image, and the actual attractor for the map W determined, that is, the distance between the image we want to code and the one that we can reconstruct from the map W. The estimate is conservative, since the factor $\frac{1}{1-s}$ makes the right side of (1.6) large, even though W has been chosen so that $d(S, W(S))$ is small.

1.2.3 Partitioned Iterated Function Systems

The IFS theory is well suited to the coding of binary images: a binary image can be seen as a collection of transformed copies of itself. However, the theory does not specify either the type of the transformations or the number of the transformed copies of the image that have to be constructed in order to cover the whole image. In the case of real gray-level images, it would be too optimistic to expect that the whole image can be constructed by using the transformed copies of the whole image itself. Even if such a collection of contractive mappings exists for a given image, it would have to include a large number of mappings, each described by a large number of parameters, which would make the whole process practically of little use for the purpose of coding.

To allow encoding of gray-level images, we have to modify the mappings given in Definition 1.11, so that they include the following features.

1. Each contractive transformation w_i includes transforming in the gray-level domain as well.

2. Each w_i be restricted to a part of the original image only.

The second feature is a significant modification as it suggests a partitioning of an image into regions each of which is transformed separately. In this case, the set of mappings $w_i, i = 1, \ldots, N$ is called partitioned iterative function system (PIFS), which is formally defined as follows.

Definition 1.12 *Let X be a complete metric space, and let $D_i \subset X, i = 1, \ldots, N$. A partitioned iterated function system is a collection of contractive mappings $w_i : D_i \rightarrow X$ for $i = 1, \ldots, N$.*

We model an image as a graph of a two dimensional function $f(x, y)$, whose domain is the unit square in the plane[1], and its value is equal to the gray-level at every point (x, y), $(x, y) \in \{(u, v) : 0 \leq u, v \leq 1\} \equiv I^2$ and $f(x, y) \in R$. Figure 1.4 shows the graph of a two dimensional function $f(x, y)$. This graph is generated by plotting the gray-level of a pixel at position (x, y) as the height of the graph at (x, y).

[1] i.e., the image dimensions are normalized to 1.

With this model of an image, the space of all possible images is defined as the graphs of all (measurable) functions over the unit square I^2, with values in R: $F = \{f : I^2 \rightarrow R\}$. We associate with the space F the supremum metric $d_{sup}(f,g) = \sup_{(x,y)\in I^2} |f(x,y) - g(x,y)|$ so that (F, d_{sup}) represents a complete metric space.

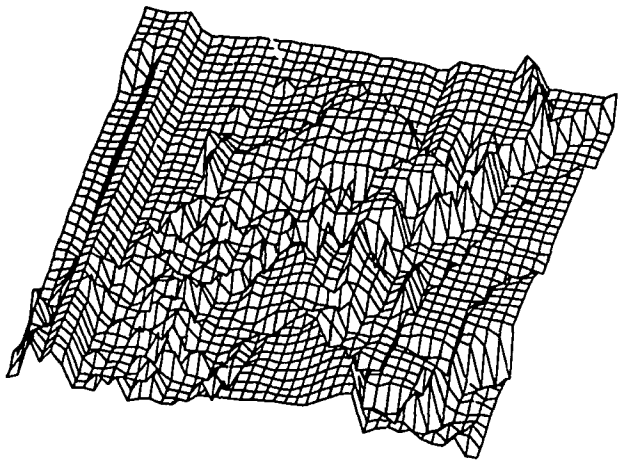


Figure 1.4: A graph generated from a real image.

We define two groups of subsets of I^2: $D_i \subset I^2$, and $R_i \subset I^2$, $i = 1, \ldots, N$. They are called the domains (where the transformation maps from) and the ranges (where the transformation maps to), respectively. Each map w_i is then restricted so that it maps $f \cap (D_i \times R)$ (the portion of image above the D_i) onto the $f \cap (R_i \times R)$ (the portion of image above the R_i). These maps, together with $D_i, i = 1, \ldots, N$, form a PIFS. There is another requirement on the maps w_i's that they tile the region I^2, as defined below.

Definition 1.13 *The maps* $w_i, i = 1, \ldots, N$ *are said to* tile I^2 *if for all* $f \in F$, $\cup_{i=1}^{N} w_i(f) \in F$.

Thus, the requirement of tiling implies that when w_i is applied to the part $f \cap (D_i \times R)$ of the image above D_i, the result must be a graph of a function over R_i, and $\cup_{i=1}^{N} R_i = I^2$. It also implies that the union $\cup_{i=1}^{N} w_i(f)$ yields a graph of a function over I^2 and that R_i's are non-overlapping. The tiling condition is necessary so that we can apply W recursively (since W requires a function over I^2 as an input). Contractivity of W in F then guarantees a unique fixed-point for W in F.

The image model used here, represents an image in three dimensional vector space. However, the supremum metric $d_{sup}(f,g) = \sup_{(x,y)\in I^2} |f(x,y) - g(x,y)|$ depends on the gray-scale value only (the z-coordinate), and not on the spatial coordinates. For this reason we define contractivity in that one dimension only.

Definition 1.14 *If* $w : R^3 \to R^3$ *is a map with* $w(x,y,z_1) = (x',y',z_1')$ *and* $w(x,y,z_2) = (x',y',z_2')$, *with x',y' being independent of* z_1 *or* z_2, *for all* x, y, z_1, z_2, *then* w *is called* z-contractive *if there exists a positive real number* $s < 1$, *such that*

$$|z_1' - z_2'| \leq s|z_1 - z_2|$$

Theorem 1.5 *If* $w_i, i = 1, \ldots, N$ *are z-contractive, then*

$$W = \cup_{i=1}^{N} w_i$$

is contractive in the metric space (F, d_{sup}) *[13].*

Given a collection of z-contractive mappings $w_i, i = 1, \ldots, N$ that tile I^2, we have seen that $W = \cup_{i=1}^{N} w_i$ defines a unique attractor x_W in the space of images. We have also seen that the attractor can be easily reconstructed from W by simply iteratively applying W to any initial image. What is left to be shown is weather a map W can be found for a given image f, such that $f = x_W$. The general solution to this problem does not exist, and it is unrealistic to believe that it can be found. Instead, we accept something more reasonable. We try to find a map W that describes some image f_W which is close enough (in the sense of metric d) to image f. We seek D_i's $i = 1, \ldots, N$ and the corresponding

w_i's such that

$$f \approx W(f) = \bigcup_{i=1}^{N} w_i(f)$$

and we hope that f_W looks similar to f. Our goal is to minimize $d(f, W(f))$ and to have a small s. In that case the Collage Theorem (Eq. 1.4) guarantees that f_W will be very close to f. Unfortunately, the bound for $d(f_W, f)$ given by this theorem is not very good in practice, since the empirical results show that restricting s to be small results in larger error $d(f_W, f)$.

In practice, we first determine $R_i \subset I^2$, $i = 1, \ldots, N$ by partitioning I^2 into non-overlapping regions. Then, for each R_i, we try to find D_i and w_i, $w_i : D_i \times R \to R_i \times R$, such that

$$d(f \cap (R_i \times R), w_i(f))$$

is minimized. The map W is specified by the collection of w_i's. We have to assure that W is at least eventually contractive for it to have the fixed-point. The m-th iterate of the map W, $W^{\circ m}$, on an image region $D_i, i = 1, 2, \ldots, N$, is composed as

$$W^{\circ m}(D_i) = w_{im}(w_{im-1}(\ldots w_{i2}(w_{i1}(D_i))\ldots)) \tag{1.7}$$

The contractivity of the composition of the maps given by (1.7) is determined by the product of contractivity factors for the maps $w_{i1}, w_{i2}, \ldots, w_{im}$. Thus, it is sufficient to have enough of contractive w_{ij}'s $j = 1, 2, \ldots, m$ in (1.7), so that the overall W is contractive while some of the factors w_{ij}'s are allowed to be non-contractive.

1.3 Encoding the Images

We have seen that the major characteristic of the PIFS is the partitioning of the image into regions each of which is transformed separately. In this section we describe a method

for encoding the images that uses the simplest partitioning scheme.

In order to evaluate the contractivity of a map, as well as to determine the distance between two images, we introduce a metric that will be used in the space of images. There are many metrics to chose from, the simplest one being the supremum metric:

$$d_{sup}(f,g) = \sup_{(x,y)\in I^2} |f(x,y) - g(x,y)|,$$

and the most practically used one being the root-mean-square (rms) metric:

$$d_{rms}(f,g) = \sqrt{\int_{I^2} (f(x,y) - g(x,y))^2}$$

The supremum metric is convenient to use for the mathematical analysis, while the rms metric has advantage in practical applications, as it will be seen in the succeeding chapters.

To encode a given image f, we have to find a collection of transformations w_i, $i = 1, \ldots, N$, with the map W defined as $W = \cup_{i=1}^{N} w_i$ and image $f = f_W$. In other words, we want f to be the fixed-point of the map W, as given by

$$f = f_W = W(f) = w_1(f) \cup w_2(f) \cup \ldots \cup w_N(f)$$

This equation also suggests as how to find the map W. We should find a partition of the image f into regions to which the transformations w_i's are applied, and the results are added up to get f. For majority of real images, it is generally not possible to find a partition so that the transformed parts of an image exactly match to some other parts of the same image. What is more realistic, is to find a map W whose fixed-point f_W is close to f, that is, $d_{rms}(f, f_W)$ is small, yielding

$$f \approx f_W = W(f_W) \approx W(f)$$

In the encoding process, we approximate the parts of the image $f \cap (R_i \times R)$ by the result obtained by transforming $f \cap (D_i \times R)$ (see the Figure 1.5)[2] . To find the

[2]For simplicity, we will say that we map block D_i to R_i, although we map the part of the function f that is above D_i (i.e., $f \cap (D_i \times R)$) to the part of the function f above R_i (i.e.,$f \cap (R_i \times R)$)

transformation w_i, we optimize the distance:

$$d(R_i, w_i(D_i)), i = 1, \ldots, N, \tag{1.8}$$

that is, we minimize the distance between R_i and $w_i(D_i)$ for the chosen metric d. Finding the image parts R_i's and corresponding D_i's is the central point in the encoding process.

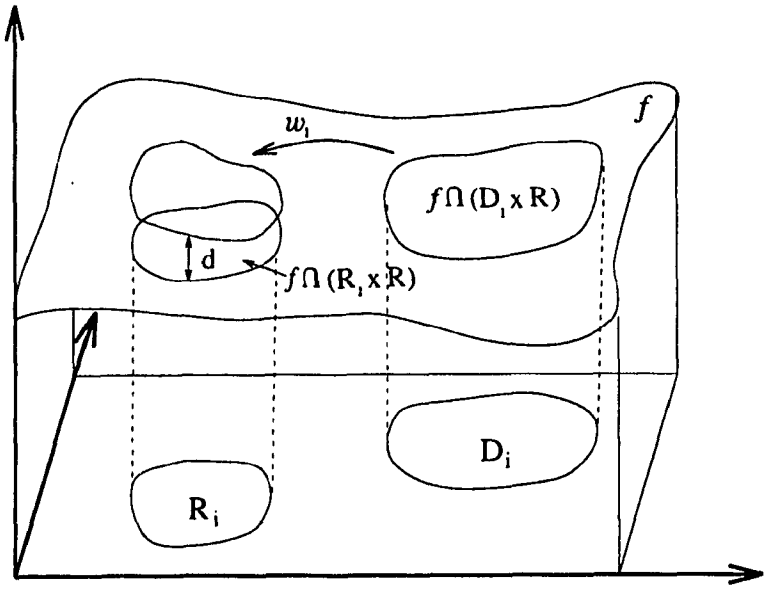


Figure 1.5: The map w_i maps the graph above D_i to the graph above R_i.

We will now present the simplest way to encode an image by using these concepts. We chose d_{rms} for the metric, and assume that $w_i, i = 1, \ldots, N$, are affine transformations (the simplest possible nonlinear transformation). This means that a w_i consists of contrast and brightness adjustment in the gray-level domain, as given by

$$w_i(f(x, y)) = s_i \cdot f(x, y) + o_i$$

Furthermore, w_i's include geometrical scaling by two in both horizontal and vertical directions: surface of the D_i blocks will be four times larger than that of the R_i blocks. For illustration, we take R_i's to be square blocks of size 8×8 and D_i's square blocks of size

16×16. We divide the image into non-overlapping 8×8 blocks, and try to approximate each block by some of the transformed larger blocks. As for the candidates among which we search for the best fit (that is for the corresponding D_i), we construct the set of all possible 16×16 blocks that overlap by one-half of their size. We denote this set of larger blocks by D and call it the *domain pool*.

For each R_i, we search through the whole set D to find a block and the transformation which make the best approximation for R_i. In order to do this, for every R_i, we have to first rescale each of the blocks from D - this requires either sub-sampling, or averaging the groups of four pixels. The next step is to find the coefficients s_i and o_i of the transformation w_i for every pair of blocks consisting of R_i and a larger block from D. After this, we evaluate the closeness of the transformed larger block and R_i in the rms sense. Finally, we memorize the transformation and the position of the larger block from D that gave the best approximation for R_i.

Figure 1.6 illustrates the reconstruction of an image that was encoded by this method. The initial image is one shown at the upper left corner of Figure 1.6. The image obtained after the first iteration (upper right) still shows some textures from the initial image. After the third iteration, no texture from the initial image is visible, and the details have already begun to appear (bottom left). The final image (bottom right) is obtained after 8 iterations. The original image required 65536 bytes to encode. The fractal code for the image required only 3584 bytes[3] yielding a compression ratio of 18.28:1. The quality of the decoded image is good, given the simplicity of the encoding algorithm.

[3]For every block we need 16 bits to encode position of D_i, 5 bits for the s_i and 7 bits for o_i. In total, this is $1024 \times 28 = 28672$ bits which is 3584 bytes

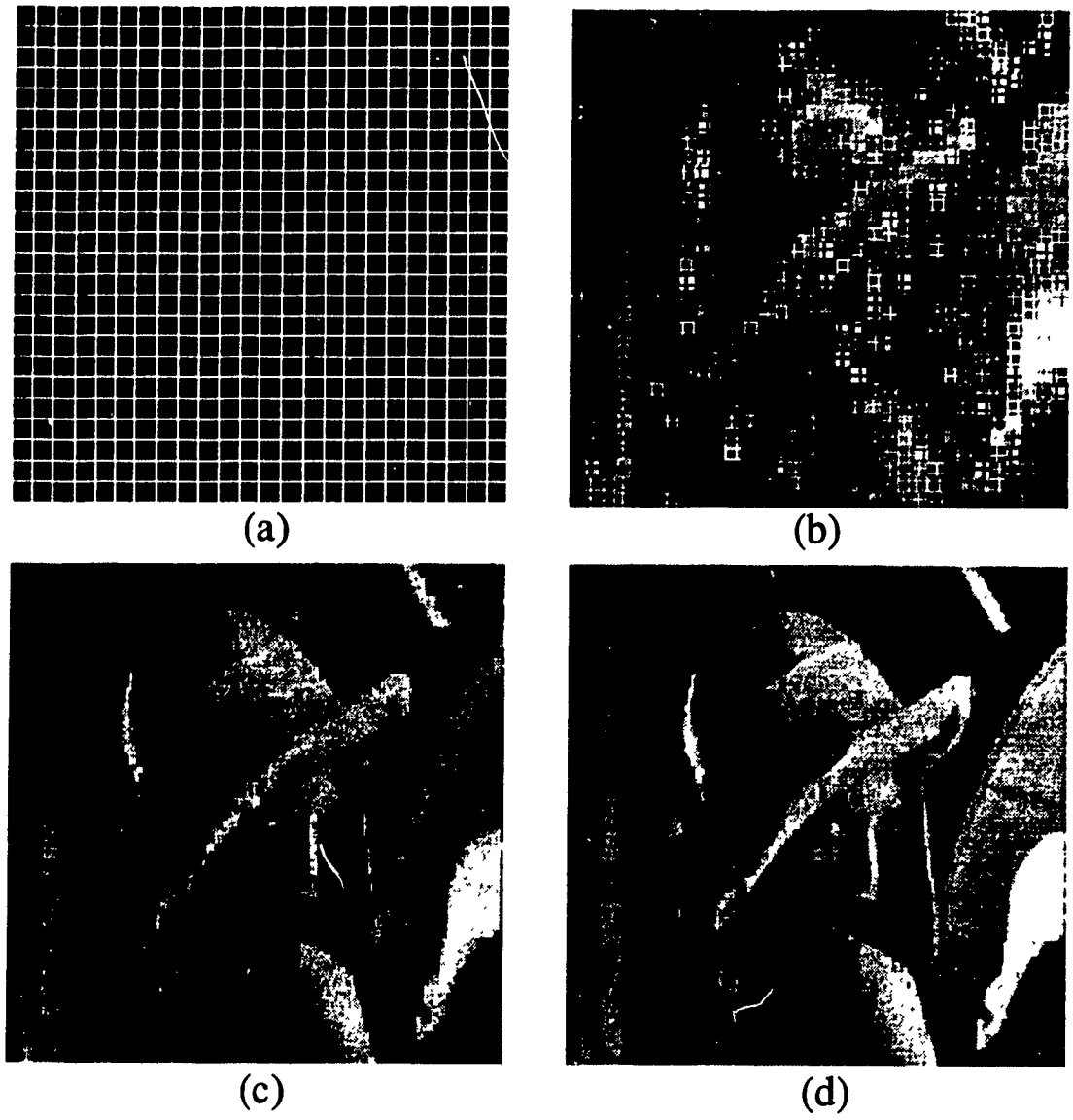


Figure 1.6: Use of the simplest encoding algorithm.
(a) The initial image and images after (b) first, (c) third and (d) eighth iteration of the decoding process.

In the above example, for an image of size 256×256, there are a total 1024 range blocks $R_i, i = 1, \ldots, 1024$ and $241 \times 241 = 58081$ possible larger blocks in D. We have to compare each of the 1024 range blocks to each of the 58081 larger blocks. Each comparison includes resizing the larger block by averaging four pixels, calculating the coefficients of the transformation, and finally, measuring the distance between two blocks. All this is

time consuming. The time performance of this technique represents a large weakness of the algorithm. Nevertheless, this is basically the generic algorithm that was first proposed by Jacquin[4] in [15], and it gives an idea as how the fractal-coding schemes work. It also suggests some possible schemes for further improvement.

1.4 Scope and Organization of the Thesis

One of the ways researches have approached the fractal image coding has been based on choosing particular image partitioning schemes [14, 12, 7]. All these schemes attempt to partition images into range blocks whose sizes vary according to the image activity, thus allowing better distribution of the allocated bit resources. However, none of these schemes has applied any form of image-content analysis in order to determine the position of the domain blocks. This leads to a large number of possible domain blocks in the domain pool, which increases the encoding time and does not necessarily guarantee an improvement in the quality of the coded image quality. In this thesis, a new adaptive fractal image coding algorithm is proposed. The algorithm includes a simple analysis of the image-block contents in image partitioning process. The partitioning information is used to determine both the range and domain blocks in an adaptive manner, which enables a reduction in the domain pool size. It is shown that the proposed algorithm provides a significant improvement in the encoding speed, with the quality of the coded images compared with that obtained by using the existing fractal-based algorithms.

One of the main characteristics of fractal objects is their richness in the detail at every scale. Similarity of the fractal objects' representations on different scales has been one of the initial motivations behind the use of the fractal theory in the image coding applications. However, the relations between the image representations at different scales are not clearly outlined and utilized in any of the existing fractal coding schemes. This thesis introduces a new, hierarchical interpretation of the fractal image coding techniques. The relationship between the fixed-points of a given PIFS, obtained on different scales, is derived. Fractal coding is presented as a refinement of the fine-scale details from the

[4]Jacquin's algorithm introduces some simple classification of the blocks in order to reduce the number of block comparisons.

information obtained on the coarse-scale. Based on this pyramid-like representation of the fixed-points, a hierarchical image coding algorithm is developed.

The thesis is organized as follows. In Chapter 2, two existing fractal coding algorithms are presented. Effect of the type of the image block partitioning on the quality of the fractal-coded images is investigated. Various parameters that are included in the design of a specific coder are considered. In Chapter 3, a new adaptive fractal-based image coding algorithm is proposed. It is shown that a higher level of similarity between the image blocks, resulting from the proposed partitioning, can be achieved by including an analysis of the image-block contents into the partitioning scheme itself. This allows a reduction in the size of the domain pool, and consequently a reduction in the encoding time. Performance of the algorithm is compared to that of the existing algorithms presented in Chapter 2. In Chapter 4, decoding of a fractal-encoded image is looked at on different scales, and relationships between the fixed-points of the same PIFS are obtained. Based on these relationships, a hierarchical image coding algorithm is developed and its performance compared with that of a more "standard" fractal-coding method [14]. Finally, in Chapter 5 the main results of this thesis are summarized and some suggestions for the future continuation of this work are given.

Chapter 2

Partitioning Schemes for Image Encoding

In the previous chapter some theoretical background that supports the idea of using simple maps to describe complex images has been presented. An example on how a fractal-based coding can be done has also been given. The method used in this example is rudimentary to be considered as a serious coding technique. However, the example has pointed out the main issues in the design and implementation of a fractal image coding system. These issues are:

1. The partitioning of an image into nonoverlapping ranges.
2. The choice of a measure of distortion between two images.
3. The type of nonlinear contractive mappings applied to image blocks.
4. The efficient method for quantization and coding of the mapping parameters.
5. The speed of coding.

Each issue brings its own trade-offs that have to be properly addressed and balanced. For example, the partitioning of an image should follow the image content and take advantage of the local similarities within the image. On the other hand, the partitioning algorithm should be as simple as possible, as we need to store the partitioning information and we want it to be compactly describable. If we want the algorithm to apply to any type of

images and not just to a set of specific images, partitioning should be robust and flexible.

In this chapter we will present two existing fractal image coding techniques, the quad-tree (QD) [14] and the HV algorithm [12], that are based on two different partitioning schemes. Both the partitioning schemes result in very simple range and domain sets of image blocks. We will describe the basic algorithms and give analysis of the results these algorithms bring.

2.1 Quad-Tree Partitioning

The encoding algorithm described in the Chapter 1 has used a very simple scheme for partitioning an image into nonoverlapping range blocks. A 256×256 image has been divided into 1024 square blocks of size 8×8. Such a partitioning scheme does not depend on the image content. It uses the same block size for the smooth and the textured areas as well as the areas that contain edges in the image. In this way, the smooth image areas are well approximated, and we can even say that the bit allocation for these areas is abundant. On the other hand, the textured areas and the areas with edges lack good approximation. The way to remedy this situation is to introduce a partitioning scheme that allows use of range blocks of varying sizes. One solution is to use the so called quad-tree partitioning. A description of the quad-tree image block partitioning can be found in [10].

A quad-tree partitioning allows a few different sizes for the range blocks. Generally, the attempt is to cover the smooth image areas by the square blocks as large as possible, while using the smaller blocks to cover the areas of the image with more activity. The benefit of allowing larger blocks for the smooth image areas is that a single transformation is used to describe a large block, thus saving on the final size of the coded image.

Figure 2.1 illustrates the way the quad-tree partitioning is performed. The left part of this figure shows an image divided into 16 squares of four different sizes. First, the image is divided into four squared blocks of size one-half the original image size. This

process continues for the upper left sub-block. The right part of figure 2.1 shows an analogy of the partition with a tree. The root of the tree is the whole image. Each division of a block corresponds to branching of a node into four new branches. Every squared block of the image is a node in the tree. The final block partition is marked by squares.

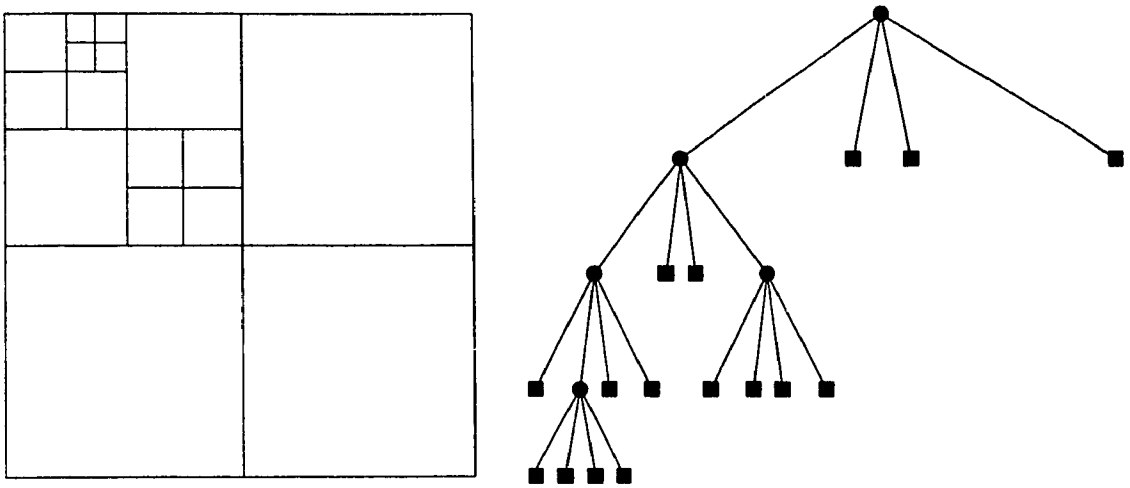
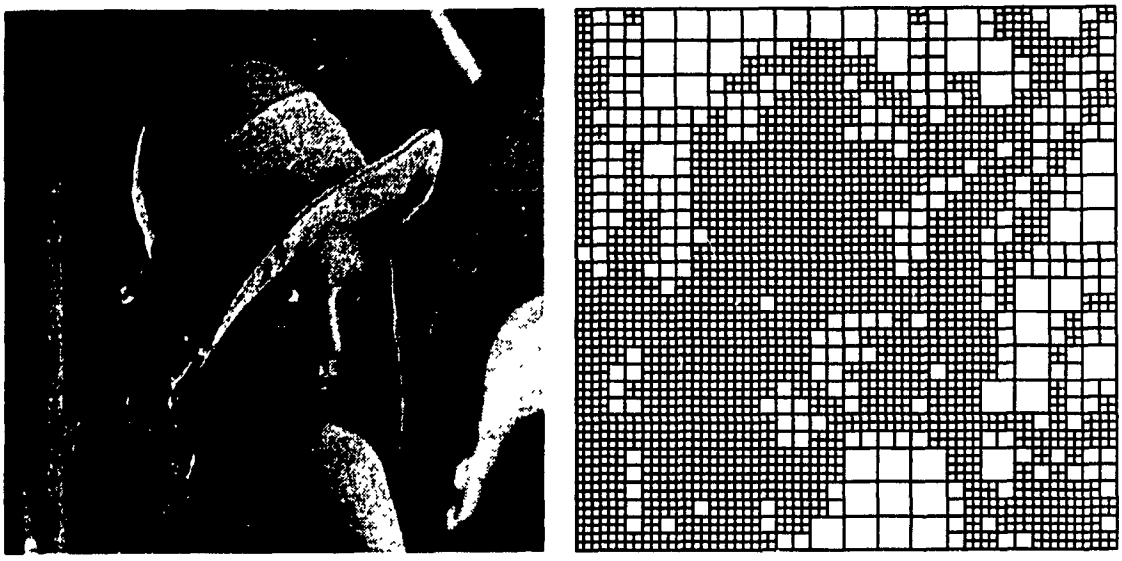


Figure 2.1: Quad-Tree partitioning.
Filled circles denote nodes in the partition tree, the squares denote final block partition.

The decision weather to split a block into four smaller blocks is based on the activity of the block. While in [13] the decision is done on the basis of the block's variance variance being measure of the block's activity, we chose to base the decision on the existence of a good approximation. That is, a block is split only if it is not possible to find a good cover for it, so we try to cover its four sub-blocks instead. This is justified by the fact that the textured areas in the image have high activity, but still can be well covered by some other textured regions of the image, not necessarily having to be divided into smaller blocks. Figure 2.2 shows the final quad-tree partition for a real image.

For the image coding purposes, we need to keep information about the final partition. The quad-tree method makes it simple and efficient to code this information. Usually for an image, we specify in advance the maximum and minimum size for a block in the final partition. Since the partition follows a simple rule - always divide into four squares, the

size of a final block can be coded by simply specifying how many levels down in the partition tree the block is located. The position of the block can be stored implicitly by specifying the ordering of the blocks (for example, from the upper left corner to the lower right corner). Efficient storage of the partitioning information is elaborated in the Section 2.1.4.



(a) (b)

Figure 2.2: QD-tree partitioned image Lenna.
(a) Original image, (b)Segmented image.

2.1.1 Encoding Images with Quad-Tree Partitioning

We have described so far how to obtain the range blocks. The collection of the domain blocks consists of sub-blocks of the image that are twice the range size. Since the sizes of the ranges vary, there is a corresponding set **D** of possible domain blocks for every range size.

The ranges are selected as follows. First we start with the whole image and divide it into four squares. We continue to divide each resulting square into four smaller squares until some predetermined maximum range size allowed (minimum depth in the partition

tree) is reached. Then, a square block in the image thus partitioned is compared with the domain blocks from the corresponding domain library **D**. Comparison consists of the following two steps:

1. average the domains' pixels in groups of four - so that the domain is reduced to the size of the range, and

2. calculate the parameters of an affine transformation for the domain block so that the *rms* distance between the transformed domain and the range is minimized.

If the resulting distance is smaller than some preselected threshold, the range-domain pair is saved together with the transformation parameters. If the *rms* difference is above the threshold, the search continues until a domain is found such that the *rms* distance is within the threshold or the set of domains **D** is completely exhausted. If the set **D** is exhausted, the range block is further split into four smaller blocks and search continues for each of of these blocks. If the minimal allowed range size is reached (the bottom of the tree), no further split can be done, and the best range-domain pair found is saved, regardless of the calculated *rms* distance. Each range-domain pair, together with the transformation parameters, constitutes one map w_i. The collection of all such maps $W = \cup_{i=1}^{N} w_i$ forms the code of the entire image.

An affine transformation applied to a domain block is of the form $s_f \cdot d_{k,l} + o_f$, where s_f is a scaling factor (the contrast adjustment), o_f is an offset term (the average brightness adjustment), and $d_{k,l}$ is a pixel value at the relative position (k,l) in the domain block. For a range-domain pair, s_f and o_f factors are calculated so as to minimize the cost function F given by

$$F = \sum_{k=0}^{M-1} \sum_{l=0}^{M-1} (s_f d_{k,l} + o_f - r_{k,l})^2 \tag{2.1}$$

where $r_{k,l}$ is a pixel value in the range block at the relative position (k,l) and M is the size of the range block. Minimizing F leads to expressions for s_f and o_f for a given range-domain pair, as given by

$$s_f = \begin{cases} 0 & \text{if } M^2 \cdot d2sum - (dsum)^2 = 0 \\ \frac{M^2 \cdot rdsum - rsum \cdot dsum}{M^2 \cdot d2sum - (dsum)^2} & \text{otherwise} \end{cases} \tag{2.2}$$

$$o_f = \frac{rsum - s_f \cdot dsum}{M^2}, \tag{2.3}$$

where

$$\begin{aligned} rsum &= \sum_{k=0}^{M-1}\sum_{l=0}^{M-1} r_{k,l} \\ rdsum &= \sum_{k=0}^{M-1}\sum_{l=0}^{M-1} r_{k,l} \cdot d_{k,l} \\ dsum &= \sum_{k=0}^{M-1}\sum_{l=0}^{M-1} d_{k,l} \\ d2sum &= \sum_{k=0}^{M-1}\sum_{l=0}^{M-1} (d_{k,l})^2. \end{aligned} \tag{2.4}$$

The value of s_f obtained from (2.2) is in addition truncated, so that $|s_f| \leq s_{max}$, where s_{max} is a maximum allowed value for the s_f. The goal is to assure that the final map W is eventually contractive. The parameter s_{max} is optimized experimentally, as it will be shown in the analysis that follows.

The values of the s_f and o_f as calculated from (2.2) and (2.3) have to be quantized for an efficient storage. Since the stored values are those that are available at the decoder, we use the quantized values of s_f and o_f when we calculate the *rms* distance between two blocks:

$$d_{rms} = \sqrt{s_f^2 d2sum + 2s_f o_f dsum + o_f^2 - 2s_f rdsum - 2o_f\ rsum + r2sum} \tag{2.5}$$

with

$$r2sum = \sum_{k=0}^{M-1}\sum_{l=0}^{M-1} r_{k,l}^2$$

The domain library for a given size of a range block, consists of square blocks that are of twice the size of the range block. The spacing between the upper left corners of two adjacent domain blocks is chosen to be one-half of their size, meaning that the domain blocks overlap. Some researchers additionally enrich the domain library by applying the isometrics on the domain blocks, so that every domain block represents actually 8 blocks: four obtained by rotations of the original block taken from the image plus the four obtained by a flip and rotations [14]. However, our experimental results show that these transformations seldom provide any advantage in the image coding. Since they introduce

significant increase in encoding time, with little visible improvement in the quality of the coded images, we have decided not to use them. Thus, a domain library for a given range size consists of all possible squared blocks of twice the range size, whose upper left corners are spaced on a square grid over the image, where the spacing between the points on the grid is equal to the size of the range. There is a technical detail for determining the domains. We have mentioned that, before the parameters of the transformation can be determined, the domain pixels should be averaged in groups of four in order to reduce the domain size to that of the range size. This can be done in an equivalent way by first resizing the whole image by factor of two, and then defining the domain block library as the set of all blocks of size equal to that of the range blocks , with the spacing of one-half the range size between them. This way, the averaging is done only once, instead of doing it for every domain each time a new range block is to be compared to it. Now, the encoding algorithm we have discussed is presented more formally using pseudo codes.

The Quad-Tree Encoding Algorithm

```
Quadtree(x_pos,y_pos,hsize,vsize,depth){
    If (depth<min_depth) {
        Quadtree(x_pos,y_pos,hsize/2,vsize/2,depth+1);
        Quadtree(x_pos+hsize/2,y_pos,hsize/2,vsize/2,depth+1);
        Quadtree(x_pos,y_pos+vsize/2,hsize/2,vsize/2,depth+1);
        Quadtree(x_pos+hsize/2,y_pos+vsize/2,hsize/2,vsize/2,depth+1);
    }
    While(there_are_domains_in_library) {
        rms_error=compare(Range_block,Domain_block);

        If (error ≤ thres_error) {
            write_pair(Range_block,Domain_block,Transformation);
            Return;
        }
```

```
            Else if (error ≤ min_error) {
                min_error=error;
                Best_Domain=Domain_block;
            }
        }

        If (bottom_reached) {
            write_pair(Range_block,Best_Domain,Transformation);
            Return;
        }
        Else {
            Quadtree(x_pos,y_pos,hsize/2,vsize/2,depth+1);
            Quadtree(x_pos+hsize/2,y_pos,hsize/2,vsize/2,depth+1);
            Quadtree(x_pos,y_pos+vsize/2,hsize/2,vsize/2,depth+1);
            Quadtree(x_pos+hsize/2,y_pos+vsize/2,hsize/2,vsize/2,depth+1);
        }
    }
```

2.1.2 Decoding

Decoding of an image from the stored code (map W) consists of iterating the map W on any initial image. The partitioning information from the code is used to determine the positions of the range blocks. For each range $R_i, i = 1, \ldots, N$, the corresponding domain block D_i from the initial image is resized (or equivalently, the domain is taken from the resized initial image), and transformed by $s_{f_i} \cdot D_i + o_{f_i}$. The resulting block is placed at the position of the range R_i. This constitutes one decoding iteration. The decoding process is iterated until no further refinement in the decoded image can be obtained by the current iteration.

2.1.3 Results and Analysis

There are many parameters that can affect the encoding: maximum and minimum range block size, the *rms* error threshold to which we compare the calculated d_{rms} distance for two blocks, choice of the quantization levels for s_f and o_f, maximum allowed scaling factor s_{max} and number of iterations in the decoding process. In this section we try to optimize the choice of some of these parameters and give an analysis of the results obtained from the algorithm described. All the results presented here are obtained using images of size 256×256, with the following parameters fixed (unless otherwise stated):

- Maximum and minimum allowed range block size 32 and 4, respectively.
- Five bits used to quantize the scaling factors and seven for the offset factors.
- $s_{max} = 1.5$

Range-domain distance

The first thing we look at is whether there is some local self-similarity in images. In other words, how close geometrically is a range to the corresponding domain block. Typical results for an image of size 256×256 are presented in Figure 2.3. From this figure, it is clear that we cannot benefit from some "local similarity" to restrict the domain search to an area close to the range, since such local similarity does not exist.

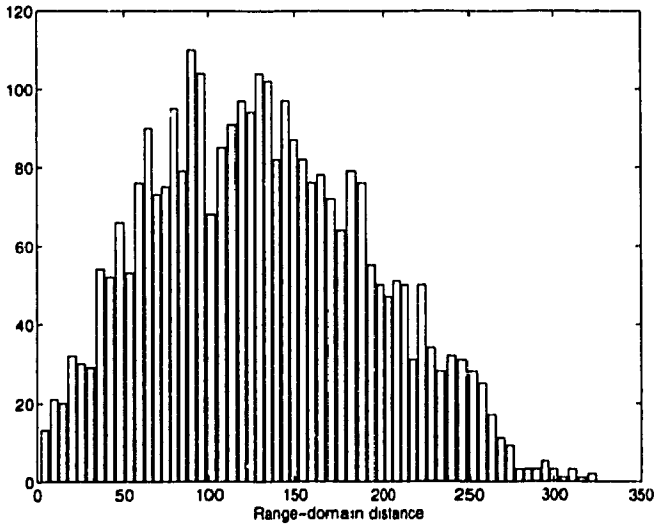


Figure 2.3: Range-domain distance.

Maximum error allowed

The *rms* error threshold parameter at the encoder is used to indirectly influence the quality of the encoded images. Whenever a range block is compared to a domain, the d_{rms} distance (Equation (2.5)) is calculated for the two blocks. The value calculated for d_{rms} is compared to the value specified for the *rms* error threshold, and if the calculated error is bellow the threshold value, the domain block is accepted as a good approximation for the range. Otherwise, the range-domain search is continued. Figure 2.4 shows how the objective quality of a decoded image is affected by varying the *rms* error threshold parameter As a measure of the objective quality, we have used the standard peak signal-to-noise ratio (PSNR)

$$PSNR = 10log_{10}\frac{g_{max}^2}{\sum(fi_{dec} - fi_{org})^2},$$

where g_{max} denotes maximum grey-level in the original image and $\sum(fi_{dec} - fi_{org})^2$ is the squared rms distance between the original and the decoded image.

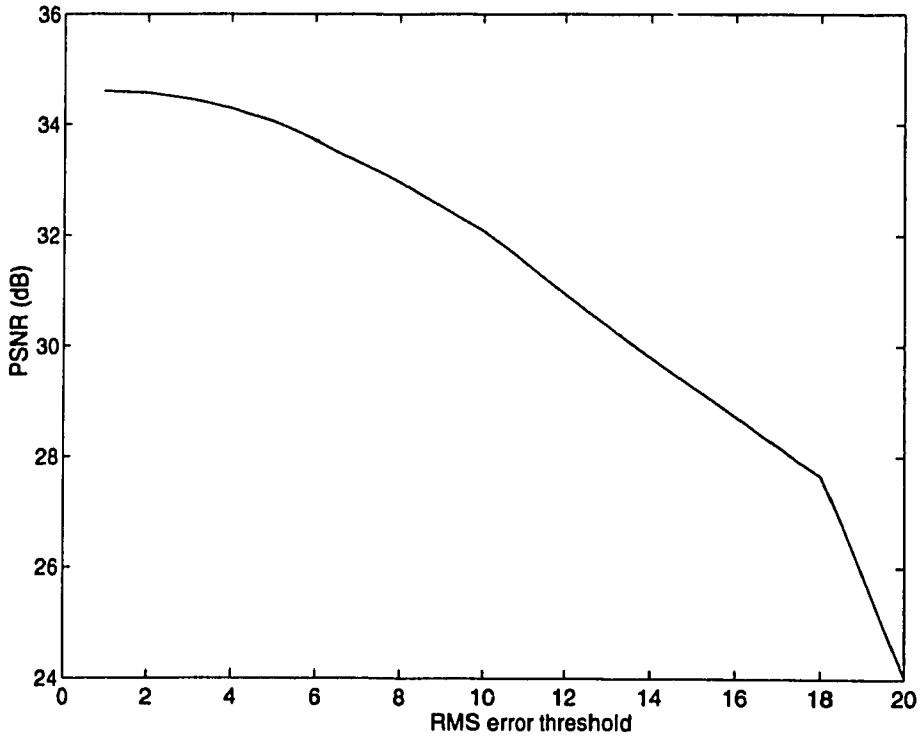


Figure 2.4: PSNR dependence on error threshold.

Figure 2.5 shows the relationship between the error threshold and the compression ratio

achieved. Allowing larger threshold for the error results in fewer ranges to encode, and this higher compression is achieved at the expense of the quality of coded image. On the other hand, for smaller error threshold, we need to encode more ranges, and thus the encoding time increases inversely proportional to decrement of the error threshold parameter.

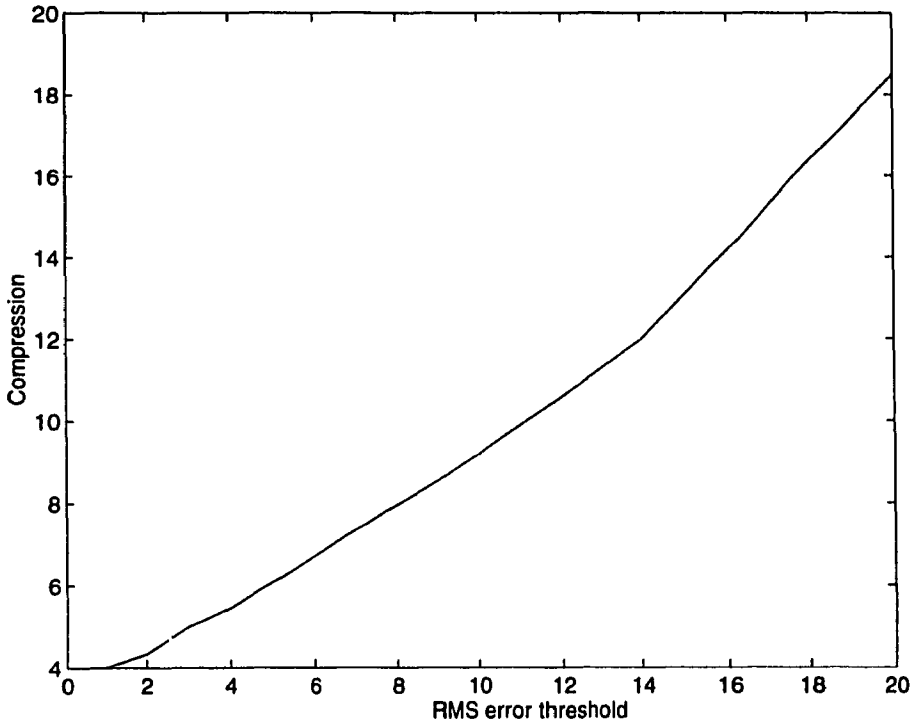


Figure 2.5: Dependence of compression ratio on error threshold.

Effect of Varying s_{max}

We have already mentioned in Section 2.1.1 that in addition to calculating the scaling factors for the domain-range transformations according to (2.2), there is a need to limit the $|s_f|$ factors by some maximum value s_{max}, in order to ensure the eventual contractivity of the map W. Here, we investigate the effect of the different values of this parameter. Figure 2.6 shows that this effect is twofold. Allowing smaller s_{max}, generally results in faster convergence of the iterative decoding process. For $s_{max} = 0.5$ we see that after only 6 iterations, the improvement in the decoded image quality between two

successive iterations is less than 0.05 dB, while for $s_{max} = 10$, 10 iterations are needed to satisfy the same criterion. One can also observe from the same figure that the quality of the encoded images varies for different values of s_{max}. The initial trend is that the PSNR of the encoded images increases with increasing s_{max}. Further increase in s_{max} leads to a slight decrease in the image quality. Finally, a steady state is achieved, when the quality of encoded images does not depend on further increase in s_{max}. The subjective quality of images encoded with $s_{max} = 0.5$ and $s_{max} = 1.5$ can be judged from Figure 2.7. It is interesting to note that even with the s_{max} much larger than 1, the convergence in the decoding is achieved.

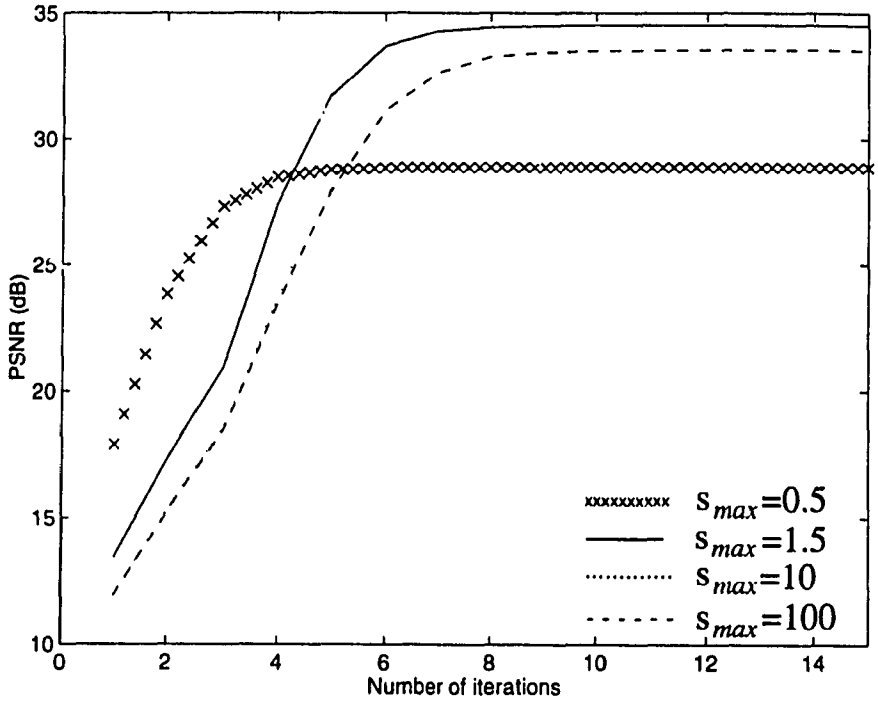


Figure 2.6: Effect of varying s_{max} on decoding speed and PSNR.

(a)

(b)

Figure 2.7: Images encoded (a) with $s_{max} = 0.5$, giving PSNR=28.88 dB, (b) with $s_{max} = 1.5$, giving PSNR=34.54 dB.

Scaling and Offset factors

Figures 2.8 and 2.9 show a typical distribution for the scaling and offset factors. It can be observed from these figures that the distributions for both s_f and o_f show significant structure, so that some kind of adaptive coding of these parameters can yield improvement in the total bit allocation for storing the transformations. In all the experiments, we have used 5 bits for s_f and 7 bits for o_f, as suggested by the analysis carried out in [13] and [17].

Figure 2.10 shows another interesting aspect of having the parameter $s_{max} > 1$. In this figure, the distribution for the scaling factors when $s_{max} = 0.8$ and $s_{max} = 1.5$ is given. It can be seen that $s_{max} = 0.8$ yields an almost uniform distribution for scaling factor, thus no benefit can be obtained by adaptive coding of s_f, unlike the case for the $s_{max} = 1.5$.

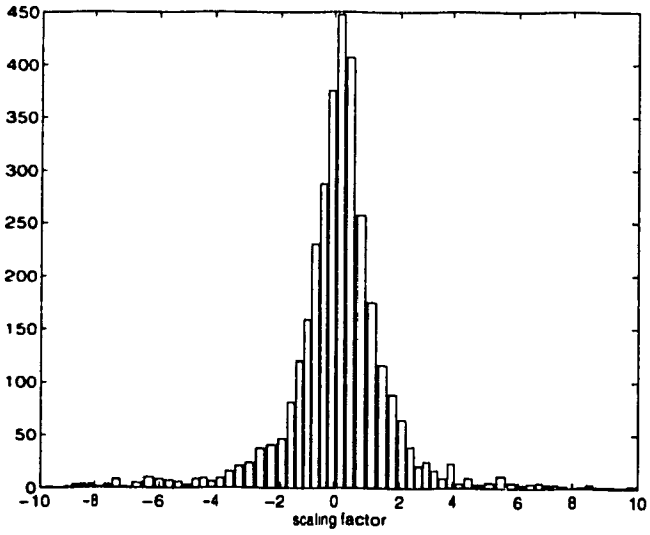


Figure 2.8: Distribution of scaling factor.

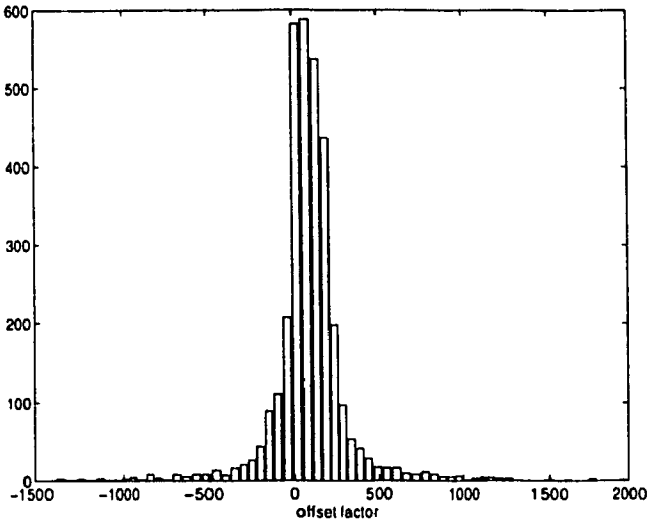


Figure 2.9: Distribution of offset factor.

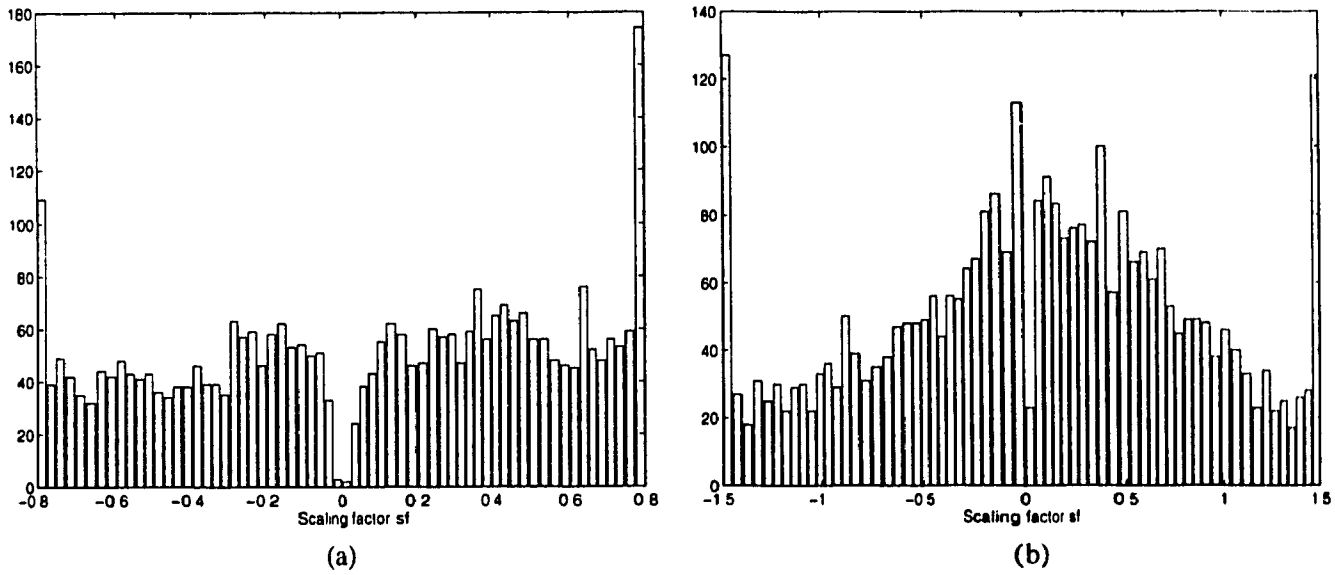


Figure 2.10: Distribution of scaling factor with (a) $s_{max} = 0.8$, (b) $s_{max} = 1.5$.

2.1.4 Efficient Storage

A quad-tree based scheme particularly allows efficient storage of the range partitioning information. If we agree in advance about the general rule for ordering the ranges in the final code, then due to the simple square shape of the range blocks, the range partition can be reconstructed from the information about the range sizes only. For example, if the ordering is from the upper left corner of the image to the lower right corner (as shown in the Figure 2.11 (a)), we can reconstruct the partitioning for the configuration presented in Figure 2.11 (b) from the sequence of range sizes, $\{r_{max}, r_2, r_2, r_2, r_{min}, r_{min}, r_{min}, r_{min}, r_{max}, r_{max}\}$. Thus, to code the position of a range, we need to store the information about its size only. If four sizes are allowed (for example, 32, 16, 8 and 4 for a 256 × 256 image), this information can be coded by two bits only.

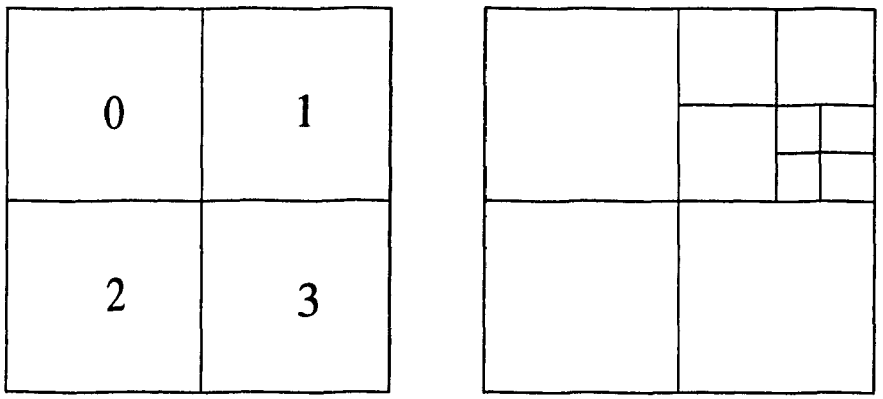


Figure 2.11: Ordering for range partition.

The position of a domain block is coded by indexing all possible domains for the given range size, since the potential domains are known in advance. For the smallest range of size 4 × 4, there are a total of 249 × 249 possible domains, thus, this information can be coded with 16 bits. The transformation parameters s_f and o_f are quantized and coded with 5 bits and 7 bits, respectively. Thus, for one range-domain pair, we need to store a total of 2+16+12=30 bits in the most conservative case.[1]

[1]In the case $s_f = 0$, we don't need the information about the domain, but only about the offset factor

In the following pages we provide examples of the QD coded images. We present two types of images, so that the performance of the algorithm can be evaluated with different input image characteristics. We can say that, in general, the algorithm gives images of satisfactory quality - no visible artifacts - for the compression ratios bellow 10. Above this compression ratio, blockiness in large smooth areas (such as sky in Figure 2.13(d)) becomes too visible, and non-horizontal or non-vertical edges (such as girl's shoulder in Figure 2.12(d)) lose smoothness. In Figure 2.12(d), it can be observed that even some of the horizontal or vertical edges are not perfectly reconstructed (upper right corner). This effect is mainly due to the range block boundaries not coinciding with the real edges in the image.

2.1.5 Speeding up the Encoding

Searching for the range-domain match is computationally very intensive. This was recognized even from the early works in fractal coding techniques. One of the ways to speed up the encoding is to classify range and domain blocks into several classes and then compare blocks from the same class only. Many classification schemes are possible. In [16] three classes have been used and blocks are classified as *flat, edge and texture*, while in [13] 72 classes have been proposed based on the energy content of the blocks.

Before encoding, all the domains in the domain library are classified. During the encoding, a range block is classified following the same criterion as for the domains and only the domains from the same class are compared with the range. This significantly reduces the number of range-domain comparisons, which is the most time consuming part of the algorithm.

We outline here one possible classification scheme. To classify a block, we divide it into four quadrants, and then calculate the average pixel value for each quadrant.

---

o_f. Thus, on the average, we require less than 30 bits/range

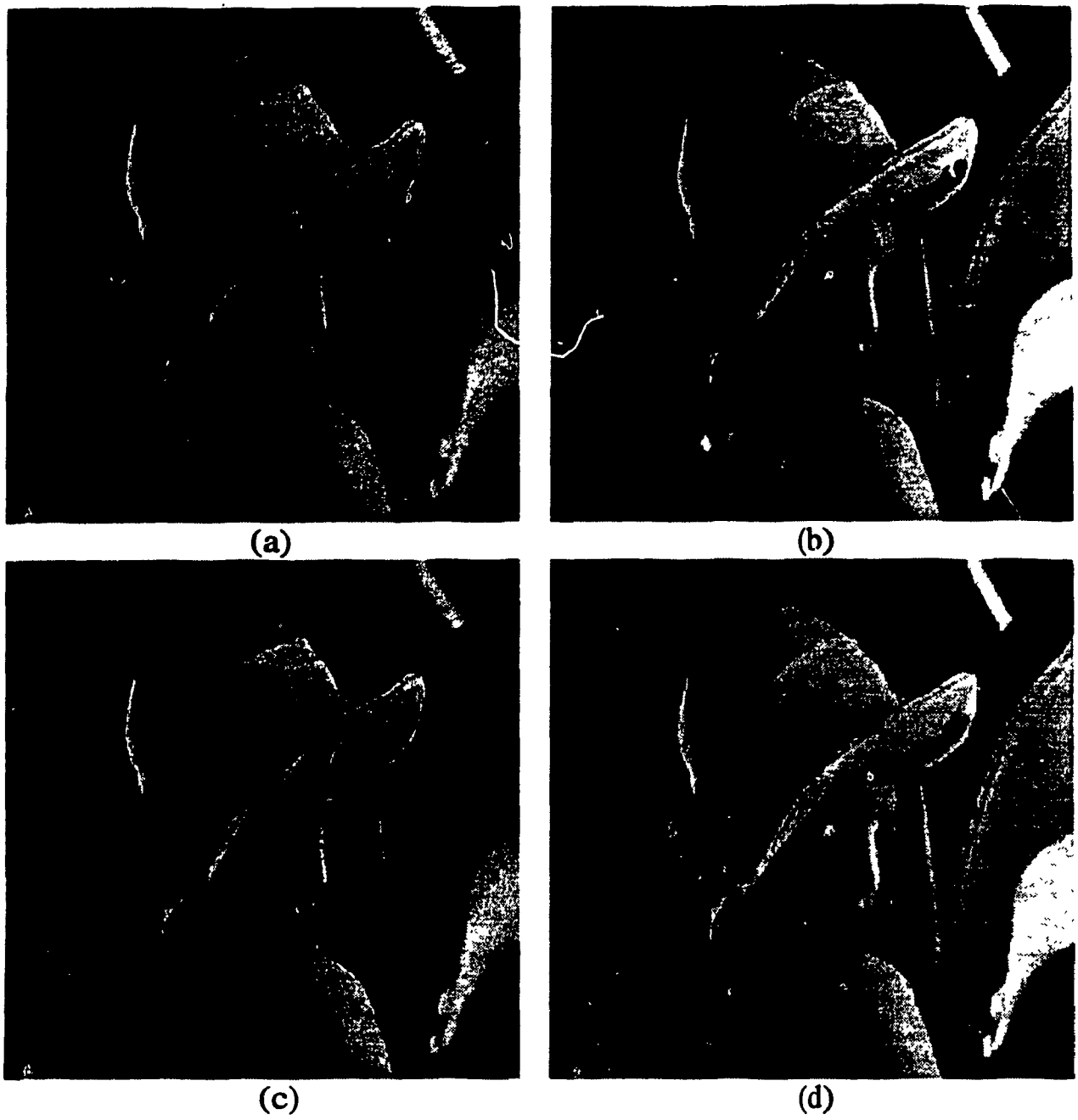


Figure 2.12: The QD encoded Lena image.

(a) Compression ratio 4.33, PSNR=34.57 dB.

(b) Compression ratio 6.44, PSNR=33.29 dB.

(c) Compression ratio 9.72, PSNR=32.50 dB.

(d) Compression ratio 14.49, PSNR=29.85 dB.

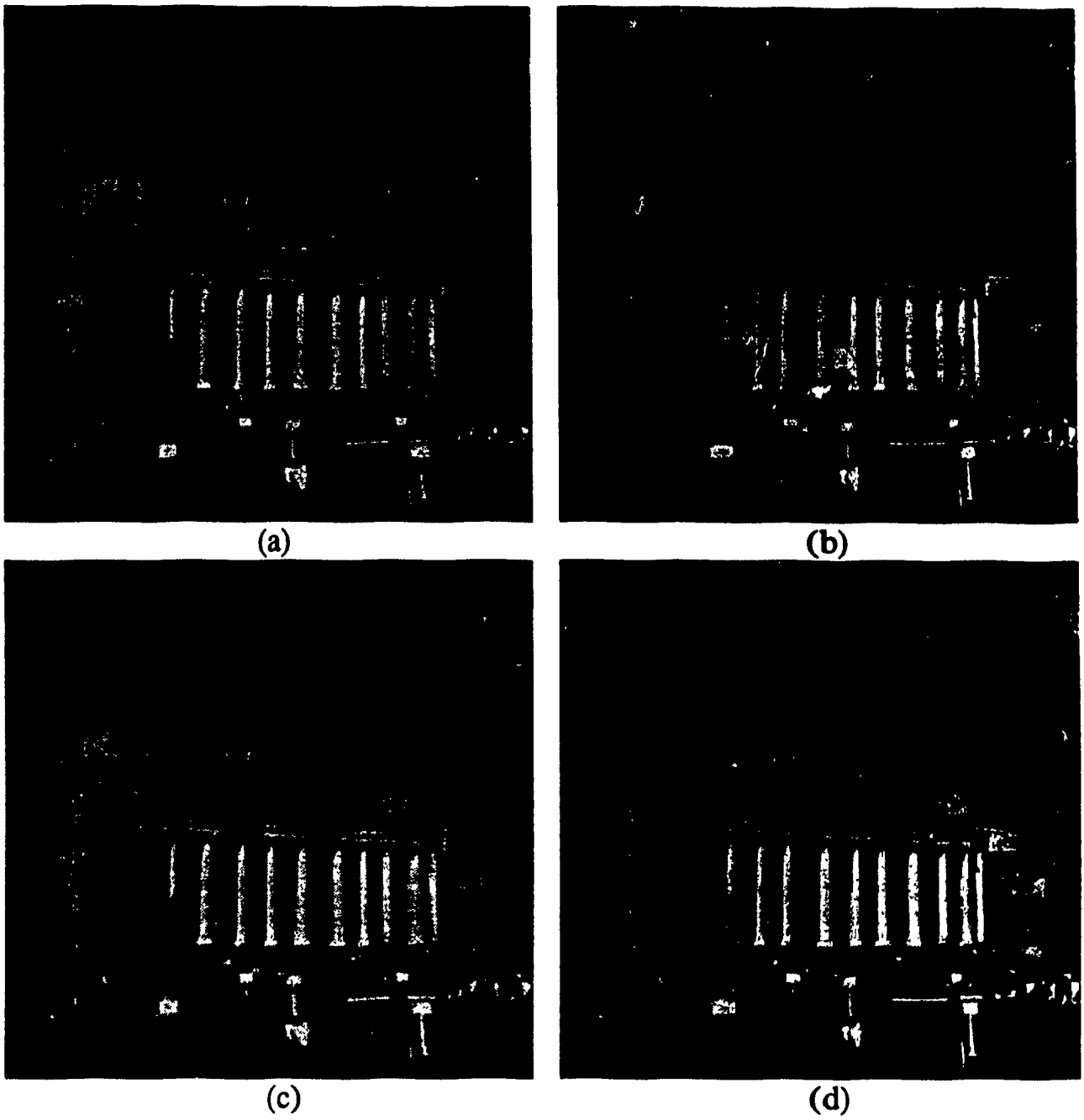


Figure 2.13: The QD encoded Columbia image.

(a) Compression ratio 5.60, PSNR=34.30 dB.

(b) Compression ratio 7.10, PSNR=32.12 dB.

(c) Compression ratio 10.12, PSNR=29.40 dB.

(d) Compression ratio 14.52, PSNR=28.65 dB.

This gives us four numbers, A_1, A_2, A_3 and A_4. We classify blocks according to the ordering of these four numbers, so that all the blocks for which $A_1 < A_2 < A_3 < A_4$ form one class, the blocks with $A_1 < A_3 < A_2 < A_4$ constitute the second class, and so on. There are 4! possible classes defined this way. The criterion for classification works because of the type of the transformation used. Since we use the affine transformations, the ordering of the average intensities for quadrants does not change after applying the transformation to a domain block, so that the blocks from a good range-domain pair belong to the same class. This type of classification can significantly improve the time performance of the algorithm, with quality of coded images preserved.

2.2 HV Partitioning for Image Coding

The main advantage of the quad-tree (QD) based image partitioning over the partition into equal size square blocks has been its ability to generally follow the activities of the image content. However, once the decision for dividing a block into four smaller blocks is made, the division is always performed the same way, without taking into account the actual content of the block. All blocks are divided into four quadrants. The partitioning scheme that we present in this section is another step towards achieving a greater flexibility in image partitioning.

The partitioning scheme is called the HV (abbreviated for Horizontal-Vertical) scheme. The partitioning is always carried on in either a horizontal or a vertical direction. A block division is not done at an apriori known position as in the QD case, but it is rather determined based on the block content. The goal is to divide a block along the strongest horizontal or vertical edge present within the block. This results in a final partitioning in which most of the blocks do not contain edges (at least horizontal and vertical). Such blocks are easier to cover.

2.2.1 HV Partitioning Scheme

The main motivation for choosing the HV partitioning scheme over the quad-trees is that of having a greater adaptivity of the range partition to the image content. The goal is not

only to generally capture the image activity, but also to adapt to the information content of the image. It is well known that human visual system is very sensitive to faithful preservation of edges in the image representation. Dividing the blocks at positions known in advance, makes it difficult to approximate well all the edges, as they may occur at any position within the block. Dividing the blocks along the strongest edges, as it is done in the HV partitioning, results in the blocks that do not contain edges, which increases probability of finding good approximations for them.

The partition of an image is done recursively in horizontal or vertical direction to form two new rectangular sub-images, generally of different sizes. The orientation (horizontal or vertical) and the position of the partition for every image block tend to follow the contents of the block: partitioning is attempted along the strongest horizontal or vertical edge present in the block. If a block of size $R_v \times R_h$ contains the pixel values $r_{i,j}$'s, then for every row i, $i = 0, 1, \ldots, R_v - 1$, and every column j, $j = 0, 1, \ldots, R_h - 1$, the following parameters are calculated

$$\begin{aligned} h_i &= \frac{min(i, R_v - i)}{R_h}\left(\sum_{j=0}^{R_h-1} r_{i,j} - \sum_{j=0}^{1} r_{i+1,j}\right) \\ v_j &= \frac{min(j, R_h - j)}{R_v}\left(\sum_{i=0}^{R_v-1} r_{i,j} - \sum_{i=0}^{1} r_{i,j+1}\right), \end{aligned} \tag{2.6}$$

These parameters measure the difference (in gray-level domain) between two adjacent rows (columns), thus detecting a strong variation in the grey-levels between them (edges). Partitioning is done at the location of maximum value: if one of the h_i's (v_j's) is found to be maximum among all the h_i's and v_j's, partitioning is done in horizontal (vertical) direction at this row (column) position. We keep the information about the cut position as the horizontal (vertical) direction and the offset from the left (top) side of the block. The purpose of the factors $\frac{min(i,R_v-i)}{R_h}$ and $\frac{min(j,R_h-j)}{R_v}$ in (2.6) is to prevent the occurrence of the blocks that are large in one direction but small in the other.

Compared to the QD method in which the positions of the block division positions

are known in advance, in the case of the HV partitioning, there is an additional computational load introduced for determining the position for the division for every block. However, this is not so critical, since it can be argued (and the results show) that the HV partition generally produces a smaller number of range blocks for the same image, than the QD method does. Since the most time consuming part in a block-based coding algorithm is the range-domain matching, computing the direction and the position for division is compensated by having to make less range-domain comparisons. The smaller number of range blocks comes from the way the division of a block is done: in the case a block cannot be covered, it is divided into two blocks only, instead of four as in the QD case. There is a tree representation for the HV partitioning scheme, similar to that described for the QD case. The only difference is that in the HV case, the nodes split into two branches instead of four. An image partitioned using the HV scheme is shown in Figure 2.14.

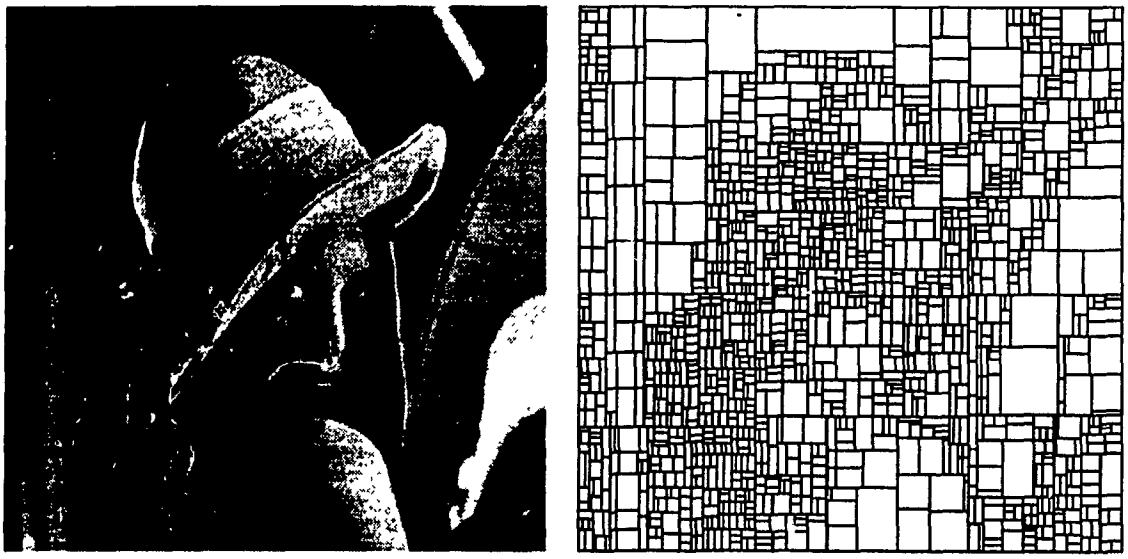


Figure 2.14: HV partitioned image Lenna.
(a) Original image. (b)Segmented image.

2.2.2 Encoding the Images

Encoding images with the HV partitioning follows the same basic concept as the one in encoding with the quad-trees. To obtain the range partition of an image, we start with the whole image, and divide it into two rectangular sub-images according to calculated parameters using (2.6). We first divide an image into rectangles until some predetermined minimum depth is reached. Then, the node blocks are compared with the domains from the corresponding domain library (these are all possible rectangular blocks that are of twice the size of the range block). For each range-domain pair, the transform parameters are found, blocks compared, and the error (distance) between two blocks calculated. If the resulting difference is smaller than some preselected threshold, the range-domain pair is saved together with the transformation parameters. Otherwise, the node block is further divided either vertically or horizontally, according to the parameters given by (2.6). The search then continues for both the resulting rectangular blocks. If the minimal allowed range size is reached (the bottom of the tree), no further split can be done, and the best range-domain pair found is saved. Each range-domain pair together with the transformation parameters constitutes one map w_i. The collection of all such maps $W = \cup_{i=1}^{N} w_i$ represents the coding for the image. Now, we present the HV Algorithm more formally in the form of pseudo codes.

The HV Algorithm

```
HV(x_pos,y_pos,hsize,vsize,depth){
   If (depth<min_depth) {
      get_offset_and_direction(x_pos,y_pos,hsize, vsize);

      If  (direction == HORIZONTAL)
         HV(x_position,y_position+offset,h_size, y_size-offset,dept+1);
      Else
         HV(x_position+offset,y_position,x_size-offset, y_size,depth+1);
   }
   While(there_are_domains) {
      error=compare(Range_block,Domain_block);
```

```
        If (error ≤ thres_error) {
            write_pair(Range_block,Domain_block,Transformation);
            Return;
        }
        Else if (error ≤ minerror) {
            minerror=error;
            best_Domain=Domain_block;
        }
    }
    If (bottom_reached)
        write_pair(Range_block,best_Domain,Transformation);
        Return;
    }
    Else {
        get_offset_and_direction(x_pos,y_pos,hsize, vsize);
        If (direction == HORIZONTAL)
            HV(x_position,y_position+offset,h_size, y_size-offset,dept+1);
        Else
            HV(x_position+offset,y_position,x_size-offset, y_size,depth+1);
    }
}
```

The transformation parameters (the scaling and offset factors) are calculated in a way similar to the QD case. The only difference is that we have to take into account two different dimensions for the rectangular blocks. For a range block with vertical and horizontal sizes R_v and R_h, respectively, (2.2) and (2.3) can be rewritten as

$$s_f = \begin{cases} 0 & \text{if } R_h R_v \cdot d2sum - (dsum)^2 = 0 \\ \frac{R_h R_v \cdot rdsum - rsum \cdot dsum}{R_h R_v \cdot d2sum - (dsum)^2} & \text{otherwise} \end{cases} \tag{2.7}$$

$$o_f = \frac{rsum - s_f \cdot dsum}{R_h R_v} \tag{2.8}$$

with $rsum, dsum, rdsum$ and $d2sum$ changed appropriately as well.

The domain library is built from the blocks that are twice the size of the range blocks, that is from the blocks of size $2 \cdot R_v \times 2 \cdot R_h$. The domain blocks overlap, and they are chosen so that the positions of upper left corners of the domain blocks form the grid with spacing R_v (R_h) in the vertical (horizontal) direction.

Large number of different sizes for the range blocks has the advantage of allowing more flexible partitioning, but it has a disadvantage as well. Since the dimensions R_v and R_h of a block are not known in advance, it is not possible to know apriori the positions of the domain blocks as well. Thus, it is not possible to classify all the domain blocks before comparisons are made. The result is that the encoding cannot be speeded up as much by domain classification as it was possible in the QD case.

2.2.3 Results and Analysis

Most of the results in this section are obtained with the similar parameter setting as that in the Section 2.1.3:

- Maximum and minimum allowed range block size 32 and 4, respectively.
- Five bits used to quantize the scaling factors and seven for the offset.
- $s_{max} = 1.5$

Maximum error allowed

Figure 2.15 shows the dependence of PSNR objective quality measure on the *rms* error threshold for the HV and QD partitioning cases. For the high quality images (low compression ratios), the two algorithms have approximately the same performance. However, for higher compression ratios, the HV algorithm outperforms the algorithm

based on QD the partitioning. Dependance of the compression ratio on the RMS error, for the HV and QD algorithms is shown in Figure 2.16. It can be observed that the HV algorithm gives slightly higher compression ratio in the range in which the objective image quality is approximately the same. For the RMS threshold over 15, the QD algorithm gives higher compression, but at the expense of worse image quality.

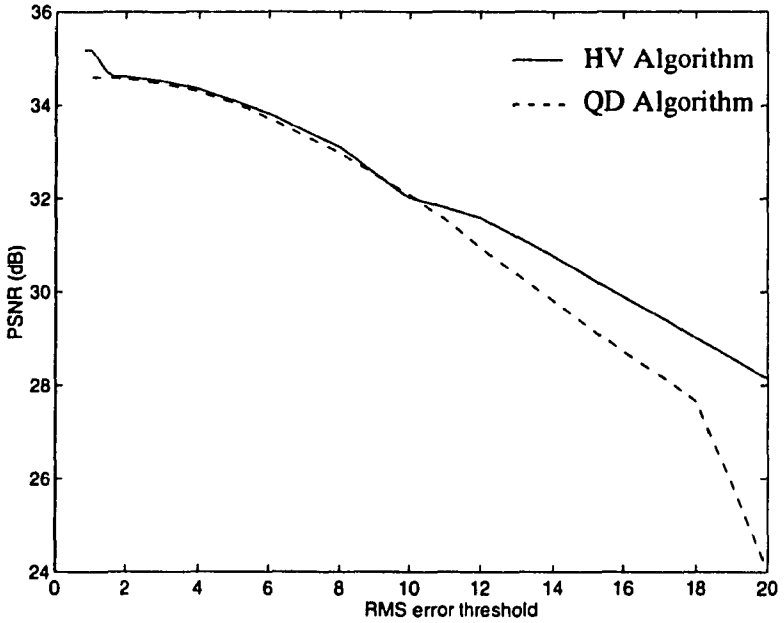


Figure 2.15: PSNR dependence on RMS error threshold.

Efficient Storage

An image is coded as information about the range partition and corresponding transformations that map domain blocks to a range blocks. Since the size and shape of range blocks vary, and consequently, their positions are not known in advance, storing the information about the ranges' dimensions only would not be sufficient to reconstruct the partitioning, as it was the case for the QD scheme. The information about the partitioning tree has to be stored instead, which means that starting from the image as whole, we have to store the direction (horizontal or vertical) and the offset from the top (in the case of a horizontal cut) or the offset from the left (in the case of a vertical cut). Some of the range blocks in the final partition are larger than the minimum size allowed, so we need to store information about terminating further branching in the tree.

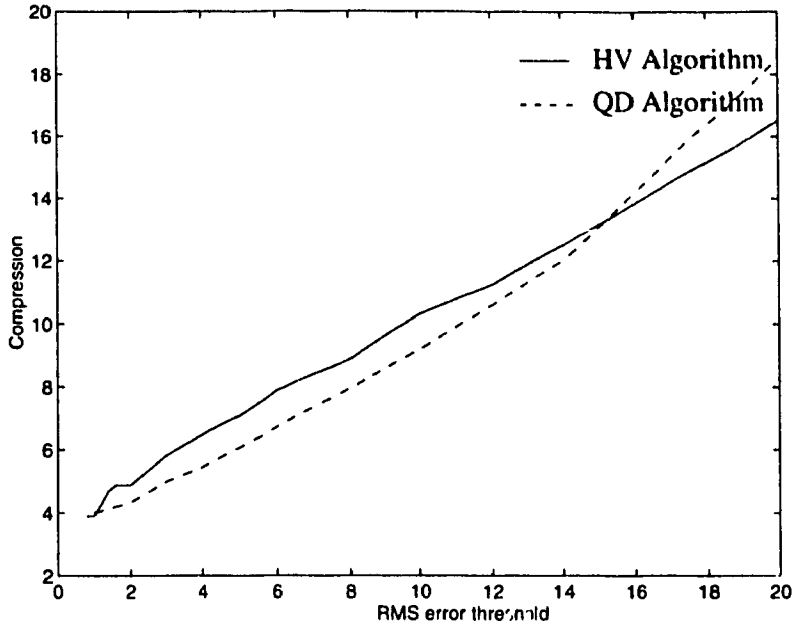
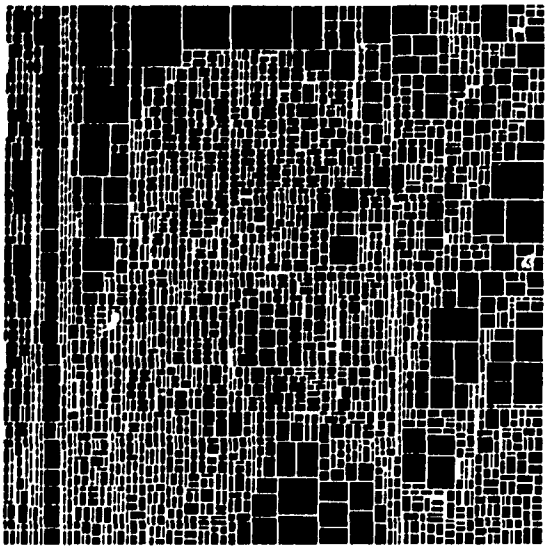


Figure 2.16: Compression ratio dependence on RMS error threshold.

We need one bit to encode the direction, seven bits to encode the offset (for 256×256 images) and two bits for the information about the terminating further branching. This information is needed for every branching node in the partitioning tree. However, if a division of a block results in two blocks that are of the order of the minimum size allowed, we only need to indicate that there is not any further branching. Positions of the range blocks are implicitly defined by this partitioning information. Figure 2.17 shows the statistics on the sizes of the resulting range blocks, for a partitioned image shown on the left. It can be seen that there are a lot more small blocks then the large ones[2] in the final partition. An conservative estimation is that we need to store six bits per range to encode the partitioning information.

[2]By small blocks we mean those that are of sizes of the order determined by the minimum size allowed.



Block type	Number of range blocks
Small blocks	1765
Larger blocks	930
Total	2695

Figure 2.17: Statistics on range block sizes.

The position of a domain block is coded as an absolute position of the upper left corner of the domain block. For the image size 256×256, 16 bits are needed to code the domain position. The transformation parameters s_f and o_f are quantized and coded with 5 bits and 7 bits, respectively. For one range-domain pair, a total of 6+16+12=34 bits are needed.

In Figures 2.18 and 2.19 we provide examples of the images coded by the HV Algorithm. They are the same example images as those used to illustrate the QD algorithm, so that the two algorithms can be compared. Performance of the two algorithms is fairly similar in the range of compression ratios less then 10. Comparing Figures 2.12(d) and 2.13(d) to Figures 2.18(d) and 2.19(d), respectively, it can be seen that the HV algorithm performs better than the QD in the case of the high compression ratios. The blockiness is less visible, and the edges are smoother in the case of the HV algorithm. All horizontal and vertical edges are very well preserved, even for very high compression ratios.

2.3 Summary

In this chapter, we have presented two algorithms that are considered to be the state of the art of the fractal applications to image coding. It has been shown how the image block partitioning scheme, that is part of the encoding process, affects the performance of the coder. The partitioning schemes better adapted to the image content, result in superior quality of the coded images. However, the higher degree of adaptivity in the image partitioning schemes comes at the expense of the increased encoding time. The scheme that applies the HV partitioning achieves higher quality encoded images than those obtained by using the scheme based on the QD partitioning. However, the encoding time of the HV algorithm is increased, compared to the QD case. Moreover, the possibility to improve the encoding time, by classifying the blocks in the domain library, is lost in the case of the HV algorithm, due to a large variation in the sizes of the resulting domain blocks. In the following chapter, we propose a new algorithm, that provides this increased flexibility of the HV partitioning without increasing the encoding time.

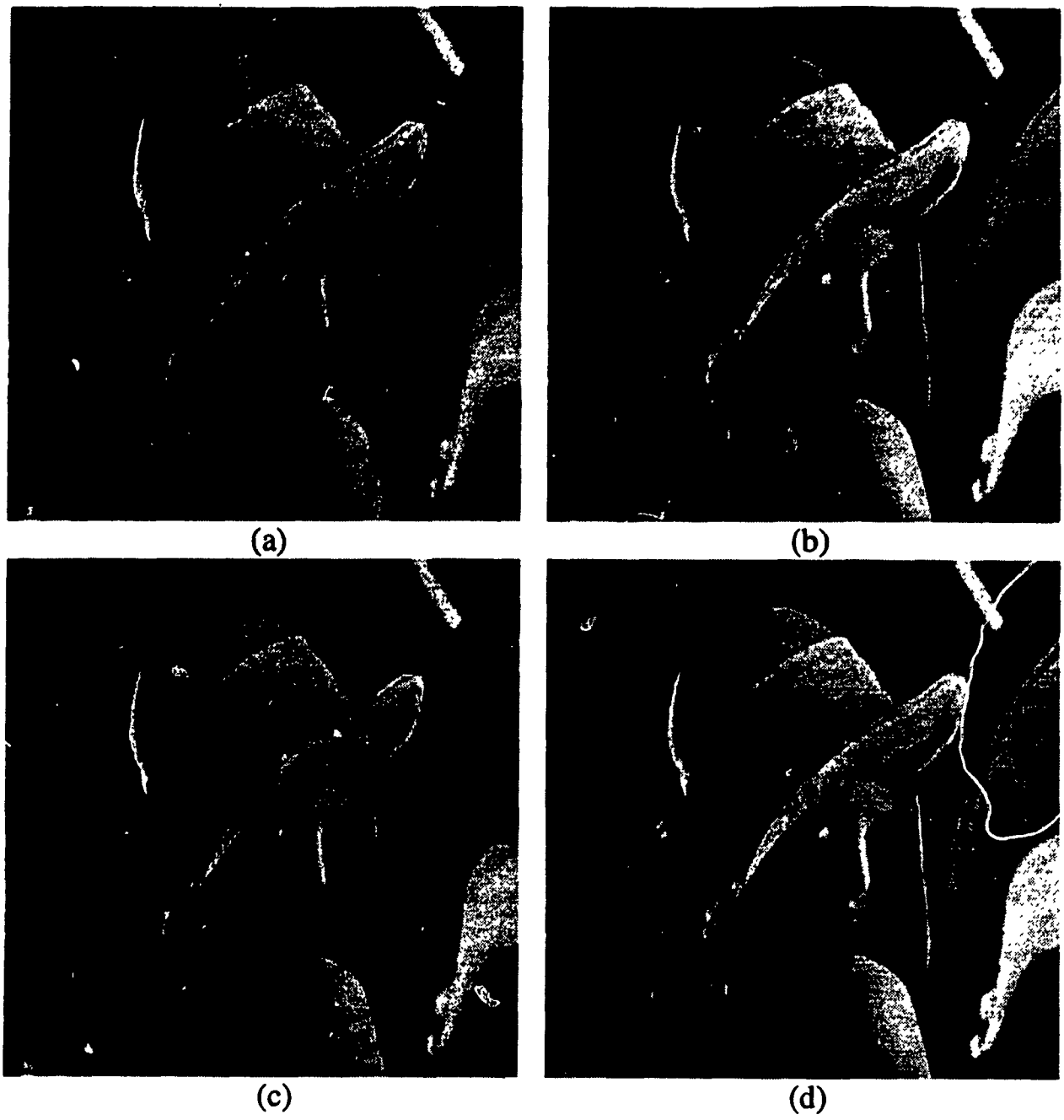


Figure 2.18: The HV encoded Lena image.

(a) Compression ratio 6.5, .PSNR=34.34 dB.

(b) Compression ratio 8.3, PSNR=33.81 dB.

(c) Compression ratio 10.2, PSNR=32.83 dB.

(d) Compression ratio 14.1, PSNR=31.57 dB.

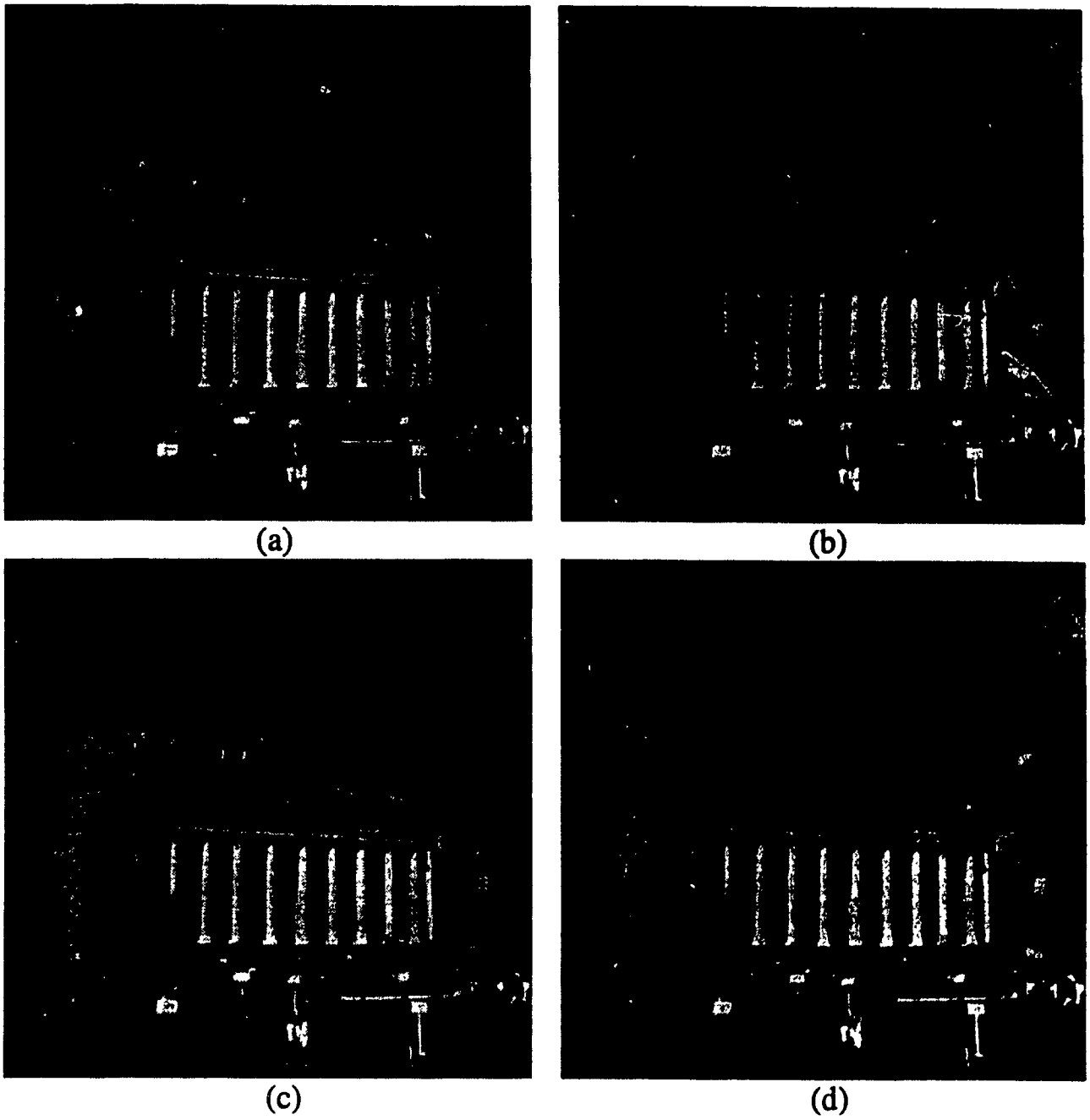


Figure 2.19: The HV encoded Columbia image.

(a) Compression ratio 5.82, PSNR=34.43 dB.
(b) Compression ratio 7.48, PSNR=33.56 dB.
(c) Compression ratio 9.78, PSNR=32.85 dB.
(d) Compression ratio 12.7, PSNR=30.86 dB.

Chapter 3

An Adaptive HV Algorithm

3.1 Introduction

The basic fractal block image coding algorithm with the HV range block partitioning scheme, as described in the previous chapter, has shown an improved performance compared to the algorithm with the QD scheme. In the HV scheme, the range block distribution adapts to the image content to a greater extent than it does in the QD case. However, there is no change in the way the domain blocks are chosen. While there is an image analysis included in the process of determining the range blocks, no such analysis is used to determine which potential domains could be good candidates for covering a particular range. Once a range block is fixed, the domain library for it gets determined as well. For a given range block, in both HV and QD cases, the domain block library consists of all possible image blocks of twice the range size, regularly placed over the image area, so that their upper left corners coincide with a grid with the horizontal and vertical spacing equal to the size corresponding to the range block. Choosing the domain blocks in such a deterministic fashion results in a large possible set of domains (domain library) and does not necessarily guarantee an improvement in the quality of the coded image. A large domain library for a given range implies a large number of range-domain comparisons, thus requiring large encoding time.

Encoding speed is one of the most important concerns in the design of a coder, and

it may be one of the main weaknesses of the fractal block-based image coding algorithms. However, there are ways to improve the performance of these algorithms. One of them has been presented for the QD algorithm in the Section 2.1.5. In the QD scheme, all possible domain block positions are known prior to determining the ranges, which makes it fairly simple and effective to classify the domain blocks in certain number of classes, thus reducing the encoding time by a significant factor. With such a classification scheme included in the fractal block-based image coder, the time performance of the fractal image coding algorithm approaches to that of the standard image coding techniques.

In the HV case, the number of allowed range block sizes is large and not known in advance. Consequently, there is a large number of different domain libraries, each corresponding to a range size. In this case, any possible benefit derived from a domain library classification scheme is lost, since the classification information can be applied for one particular range size only and cannot be used for other range blocks.

While one of the solutions for speeding up the encoding might be simply reducing the domain library size, this may have very negative effect on the coded image quality. If the domain library size is reduced arbitrarily (for example, by increasing the distance between two consecutive potential domain blocks), there might not be enough good domain block candidates, and thus a poor approximation of the coded image may result. The effect of reducing the domain block library by increasing the distances between the adjacent domain candidates is shown in Figure 3.1. In this chapter, an adaptive fractal block based image coding algorithm which tends to overcome the problem of the encoding speed of the HV scheme is proposed [8].

3.2 The Proposed Adaptive HV Algorithm

The proposed adaptive algorithm uses an HV partitioning for an image, taking advantage of its flexibility. Unlike the basic HV algorithm, in which the partitioning information is used to determine the range blocks only, in this algorithm the partitioning information is used in the encoding process for determining both the range and the domain blocks

of an image. Neither the ranges nor the domains are determined in advance. Instead, the image is first fully partitioned into rectangular blocks, as shown in Figure 3.2. Both the ranges and domains are selected from this partitioned image. Performing the full image partitioning first, and choosing the domain blocks from the partitioning tree, gives an advantage of knowing the exact positions of the domain blocks prior to any range-domain comparison. This makes it possible to include a block classification scheme into the encoding algorithm itself, thus allowing improvement of the time performance of the proposed algorithm.

Figure 3.1: The effect of reducing the domain library size in an arbitrary fashion.

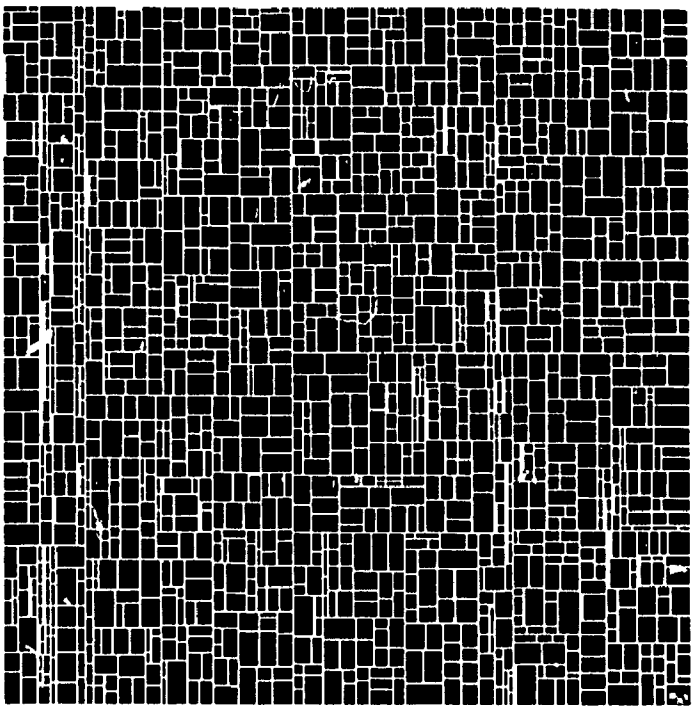


Figure 3.2: Fully partitioned image.

The range blocks are chosen as follows. Starting at some level in the partitioning tree, a rectangular block (potential range) is compared to other rectangular blocks from the partitioning tree that cover an area that is at least two times larger than that of the potential range (see Figure 3.3).

We search among all those larger blocks (potential domains), for the one that best approximates the potential range. If a sufficiently good match for a range is found , the range is marked as covered, and all its descendants in the tree are discarded from the set of possible ranges. But they are still kept as candidates for domains to be compared with other possible ranges. If all larger blocks in the partitioning tree are exhausted and no match for the potential range is found, the algorithm proceeds to the next lower level in the tree and repeats the matching process for a new range candidate. If the bottom of the tree is reached and no satisfactory match is found, the best match found is accepted.

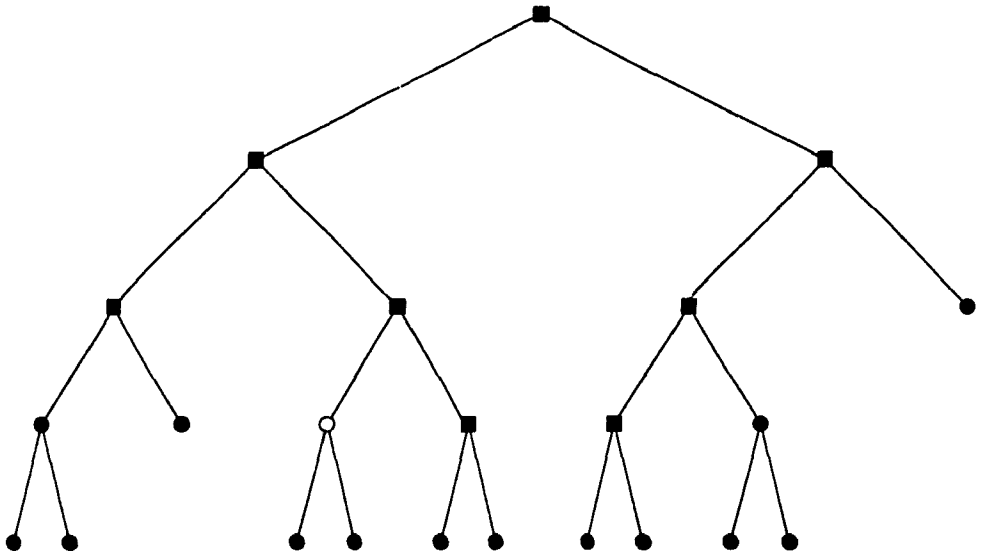


Figure 3.3: Image partitioning tree.
The empty node in the tree represents a range block candidate that is currently being compared with all the blocks (filled square blocks on the figure) from the partitioning tree that are of at least twice its size.

There is a modification in the partitioning scheme used in this algorithm over that in the basic HV scheme. The modification is in determining the position of the division point and the orientation of the partition. The partitioning is done along the strongest horizontal or vertical edge, according to (2.6), if a horizontal or vertical edge is detected in the block. However, in the case when there is no such an edge, the partitioning is done in such a way as to result in two blocks, where the edge runs diagonally in one of them (see Figure 3.4). This additional detail tends to make the blocks from different levels in the partitioning tree to be more similar, so that better matches between them are possible.

Since we do not know in advance if a block does or does not have a horizontal or vertical edge, we calculate, at the same time, the parameters of (2.6) as well as the eventual partitioning position in the case of a diagonal edge.

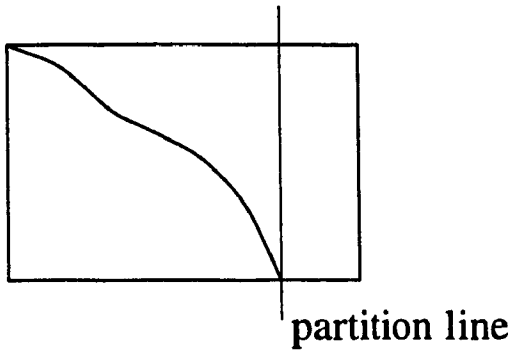


Figure 3.4: Partitioning of a block with no horizontal or vertical edge.

The position for the partition in the presence of a diagonal edge, as shown in Figure 3.4, is calculated on the basis of variations of the average pixel values over an image block. If a block of size R_vxR_h contains the pixel values $r_{i,j}$'s then for every row i, $i = 0, 1, \ldots, R_h - 1$, and every column j, $j = 0, 1, \ldots, R_v - 1$, the following parameters are calculated:

$$\begin{aligned} hd_i &= \left| \frac{\sum_{k=0}^{i} \sum_{l=0}^{R_h-1} r_{k,l}}{(i+1) \cdot R_h} - \frac{\sum_{k=i+1}^{R_v-1} \sum_{l=0}^{R_h-1} r_{k,l}}{(R_v - i - 1) \cdot R_h} \right| \\ vd_j &= \left| \frac{\sum_{k=0}^{R_v-1} \sum_{l=0}^{j} r_{k,l}}{(j+1) \cdot R_v} - \frac{\sum_{k=0}^{R_v-1} \sum_{l=j+1}^{R_h-1} r_{k,l}}{(R_h - j - 1) \cdot R_v} \right|. \end{aligned} \tag{3.1}$$

If a horizontal or vertical edge is found, i.e., parameters h_i and v_j from (2.6) are above some threshold value, we proceed as in the original HV algorithm. Otherwise, we assume that the block is of the type shown in Figure 3.4. In that case, we determine the overall maximum of the parameters hd_i's and vd_j's from (3.1). If the overall maximum found corresponds to one of the parameters hd_i's, the partitioning is done in the horizontal direction at a position equal to i that gives the maximum hd_i; otherwise, the partitioning is done in the vertical direction at a position that is equal to j that gives the maximum vd_j.

In order to minimize the additional computational load introduced due to the calculation of the partitioning position according to the two criteria, we need to optimize the computational complexity for evaluating the parameters given by (2.6) and (3.1). The

best way to achieve this is to use only one block scan, when calculating both h_i and hd_i (v_j and vd_j), instead of using two separate ones. To calculate an h_i from (2.6), we need to calculate the difference between the sums of the pixel values over the two consecutive rows. To determine an hd_i from (3.1), we need to calculate the average of the pixel values in the portions of the block above and below row i. We illustrate the the method for calculating the parameters for the possible horizontal partition. For the vertical partition, similar solution is used.

At the beginning, we define the following variables:

1. *sum_up* - variable used to store the sum of the pixel values in the portion of the block above row i. It is initially set to 0.
2. *sum_down* - variable used to store the sum of the pixel values in the portion of the block below row i. It is initially set to the sum of all pixel values in the block.
3. *sum_i* - variable used to store the sum of the pixel values in row i. Initially set to 0.
4. *sum_*$(i+1)$ - variable used to store the sum of the pixel values in the row $i+1$ of the block. Initially set to the sum of all pixel values in the first row of the block.

We scan over the block rows ($i = 1, \ldots, R_{v-1}$), and for every i we do the following:

1. Store *sum_*$(i+1)$ into the *sum_i*
2. Calculate the sum of all the pixels in row i and store it into *sum_*$(i+1)$
3. Add *sum_i* to the *sum_up* variable
4. Subtract *sum_*$(i+1)$ from the *sum_down* variable
5. Use *sum_i* and *sum_*$(i+1)$ to determine h_i from 2.6:

$$h_i = \frac{min(i, R_v - i)}{R_h} \cdot |sum\_i - sum\_(i+1)|$$

6. Use *sum_up* (*sum_down*) variable to determine the average value of the pixel values over the portion of the block above (below) the row i

$$hd_i = |\frac{sum\_up}{R_h \cdot i} - \frac{sum\_down}{R_h \cdot (R_v - i)}|$$

The range and domain blocks are determined from the same partitioned image (partitioning tree). This results in a large number of different range-domain size ratios. For the QD and the basic HV algorithms, this was not the case. The domains were always chosen to be of exactly twice the range block size. Comparing the blocks with wide range of size ratios is an important problem that must be considered in designing the coding algorithm. In our study, experiments with different solutions have been made. Resizing (by pixel averaging) of the domain blocks to the size of a range block is the most straightforward solution, but it has been found tc be very time consuming. However, a much simpler solution also gives very good results. In this solution, we allow certain number of size ratios only. That is, the possible domain blocks are resized by some predefined factors only, even though these may not be the exact ratios of the range-domain blocks. We have chosen four different ratios: $(H/k_1, V/k_2)$, $k_1, k_2 = 2, 4$. Thus, H/2, for example, denotes a resizing by a factor of two in horizontal direction.

We now present the proposed Adaptive HV Algorithm using pseudo codes.

Adaptive HV Algorithm

1. Make full partition

```
partition(Block,x_position,y_position,x_size,y_size) {
    If (Block_size > threshold ) {
        get_offset_and_direction(Block);
        If (direction == HORIZONTAL)
            partition(Block,x_position,y_position+offset,x_size,y_size-offset);
        Else
            partition(Block,x_position+offset,y_position,x_size-offset,y_size);
    }
    Return;
}
```

2. Parse the tree and do comparison:

```
While(there_are_uncovered_blocks) {
    get_Range_block();
    get_Domain_pool();

    While(there_are_domains) {
        error=compare(Range_block,Domain_block);
        If (error ≤ thres_error) {
            write_pair(Range_block,Domain_block,Transformation);
            Return;
        }

        Else if (error ≤ minerror) {
            minerror=error;

            best_Domain=Domain_block;
        }
    }

    If (bottom_reached) {
        write_pair(Range_block,best_Domain,Transformation);
        Return;
    }
}
```

3.3 Efficient Storage

A fractal code for the image should contain information necessary to reconstruct the partitioning tree. Since the same partitioning tree is used to determine both the positions of the domain and the range blocks, information about the partitioning tree is sufficient to determine the positions of both the range and the domain block. This information is

stored in a way similar to one described in the Section 2.1.4. What is different is that we know that partitioning will be done until the block size reaches minimum size allowed, so we do not need to code information about possible termination of branching. We store the direction (horizontal or vertical) and the offset from the top (in the case of a horizontal cut) or the offset from the left (in the case of a vertical cut). We need one bit to encode the direction, and seven bits to encode the offset (assuming 256×256 images). The partitioning information is stored per node in the tree and, on average, we need to store eight bits per range block to determine its position. Once the partitioning tree is known, a domain block position can be stored as an index in the list of all possible blocks in the partitioning tree. To encode a domain block, we need 12 bits.

The transformation parameters s_f and o_f are quantized and coded with 5 bits and 7 bits, respectively. In addition, the domain-range size ratio used has to be coded as well, which for the proposed scheme requires 2 bits. In order to store the information about one range-domain pair, a total of 34 bits are needed.

3.4 Results and Analysis

Before presenting the results, it should be pointed out that the proposed algorithm has not been trained for specific types of the images. A total of ten different images, other than those used for testing the performance, were used for this purpose. For a fair comparison of the performance of the proposed algorithm, with those of the QD and HV algorithm , the same set of images were presented to all the three algorithms. Although the results presented in this section concern only two images, a large set of different images were used for testing, and the results were found to be approximately the same as those given in this section.

Figure 3.5 compares performance of the proposed adaptive HV algorithm to that of the original HV and the QD algorithms. In the range of low compression ratios (less than 5), QD and HV algorithms achieve slightly better quality of encoded images for the same compression ratio. For the compression ratios above 8, the adaptive algorithm

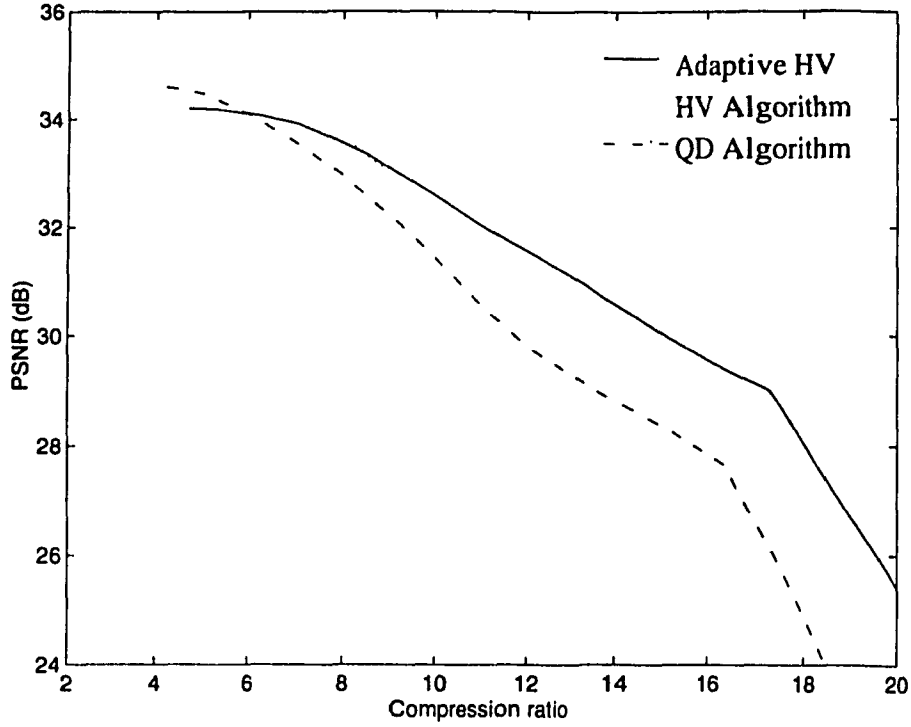


Figure 3.5: PSNR vs. compression ratio.

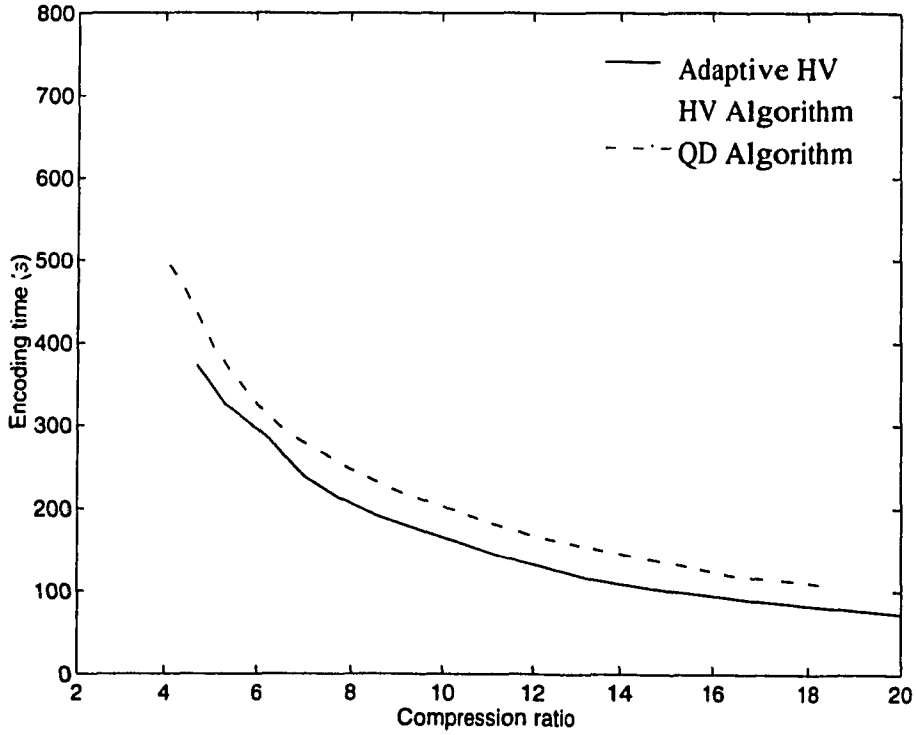


Figure 3.6: Encoding time vs. compression ratio.

outperforms both the original HV and QD scheme. Figure 3.6 shows the comparison of performance of the three algorithms as measured by the encoding times. The encoding time is given in terms of the execution times of the algorithms on the Silicon Graphics Indigo machine. Adaptive HV algorithm has performed slightly better than the QD, and significantly better than the original HV scheme. Better time performance of the Adaptive HV Algorithm is achieved due to the reduced number of the domain blocks in the domain library. Reduction in the number of range-domain comparisons compensates for the additional encoding time in determining the position of the block partition according to the two criteria, (namely, one for the blocks with the horizontal/vertical edges and another for the blocks with the diagonal edges), and in the resizing operations with different ratios. The examples of encoded images can be seen in Figures 3.7 and 3.8. It can be observed that the reduction in the domain library size, had little effect on the quality of the encoded images. There is no visible difference between the images presented in Figures 3.7 and 3.8 and those presented in Figures 2.18 and 2.19, respectively.

It would be interesting to note the quality of the images encoded using the proposed adaptive HV algorithm at very high compression ratios. From the results already presented we see that in this range of compression ratio the adaptive HV algorithm results in the encoded images of higher quality than that of the images encoded by using the QD or the original HV algorithm. From Figure 3.9 can be observed that an image encoded with very high compression ratio using the proposed algorithm, still retains enough information about the image for it to be useful for some applications, in comparison with the images encoded using the other two algorithms.

As mentioned before, there is another advantage of the Adaptive HV Algorithm over the original HV scheme. Before any domain-range comparison is done, the image is fully partitioned and the domains and ranges are selected from that partition. Thus, all the domains are known in advance, and a classification scheme similar to the one explained in the Section 2.1.5 can be applied, yielding a further improvement in the encoding time.

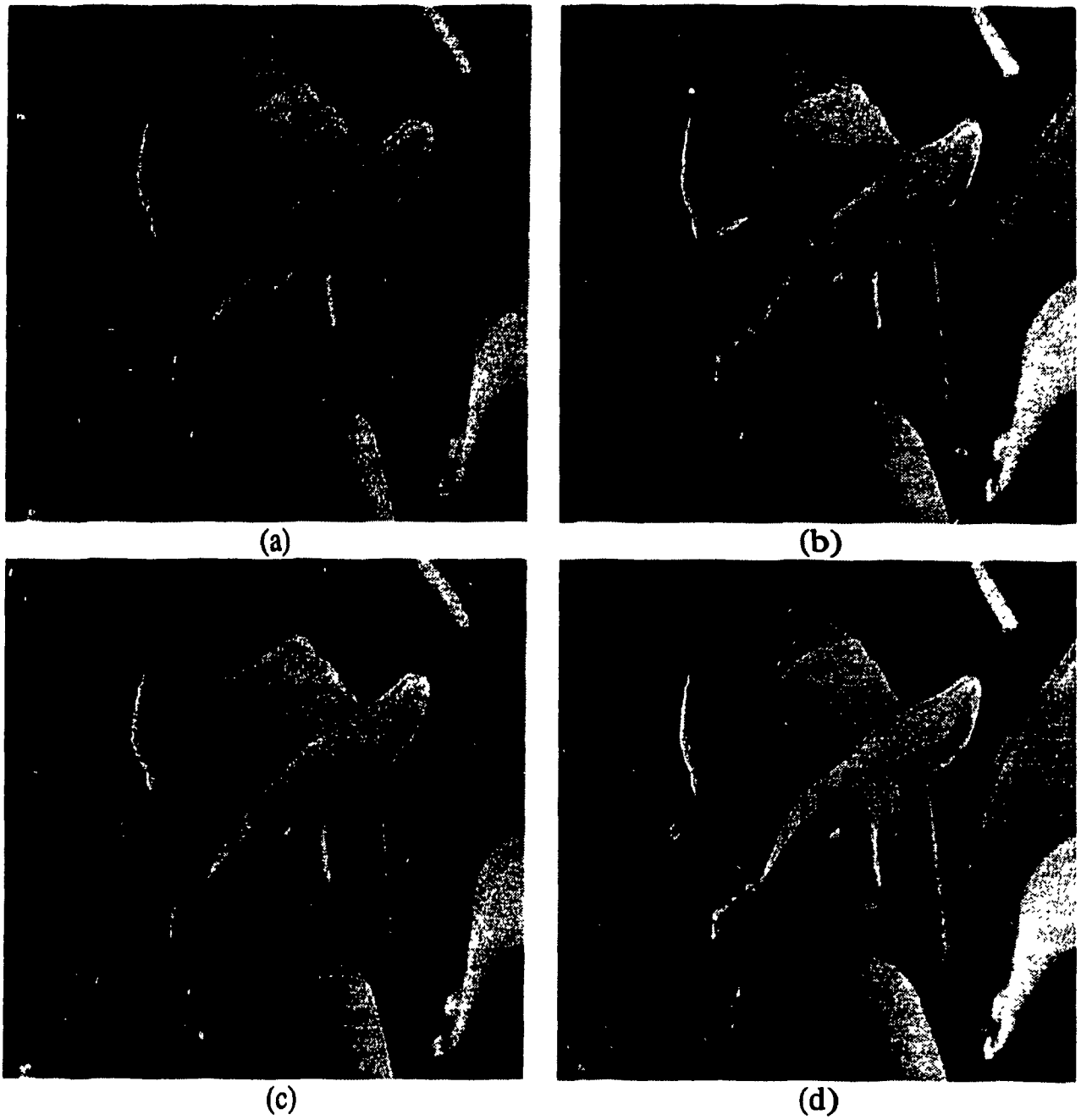


Figure 3.7: The Adaptive HV encoded Lenna image.

(a) Compression ratio 6.30, PSNR=33.56 dB.

(b) Compression ratio 8.74, PSNR=32.91 dB.

(c) Compression ratio 9.83, PSNR=32.66 dB.

(d) Compression ratio 14.35, PSNR=31.18 dB.

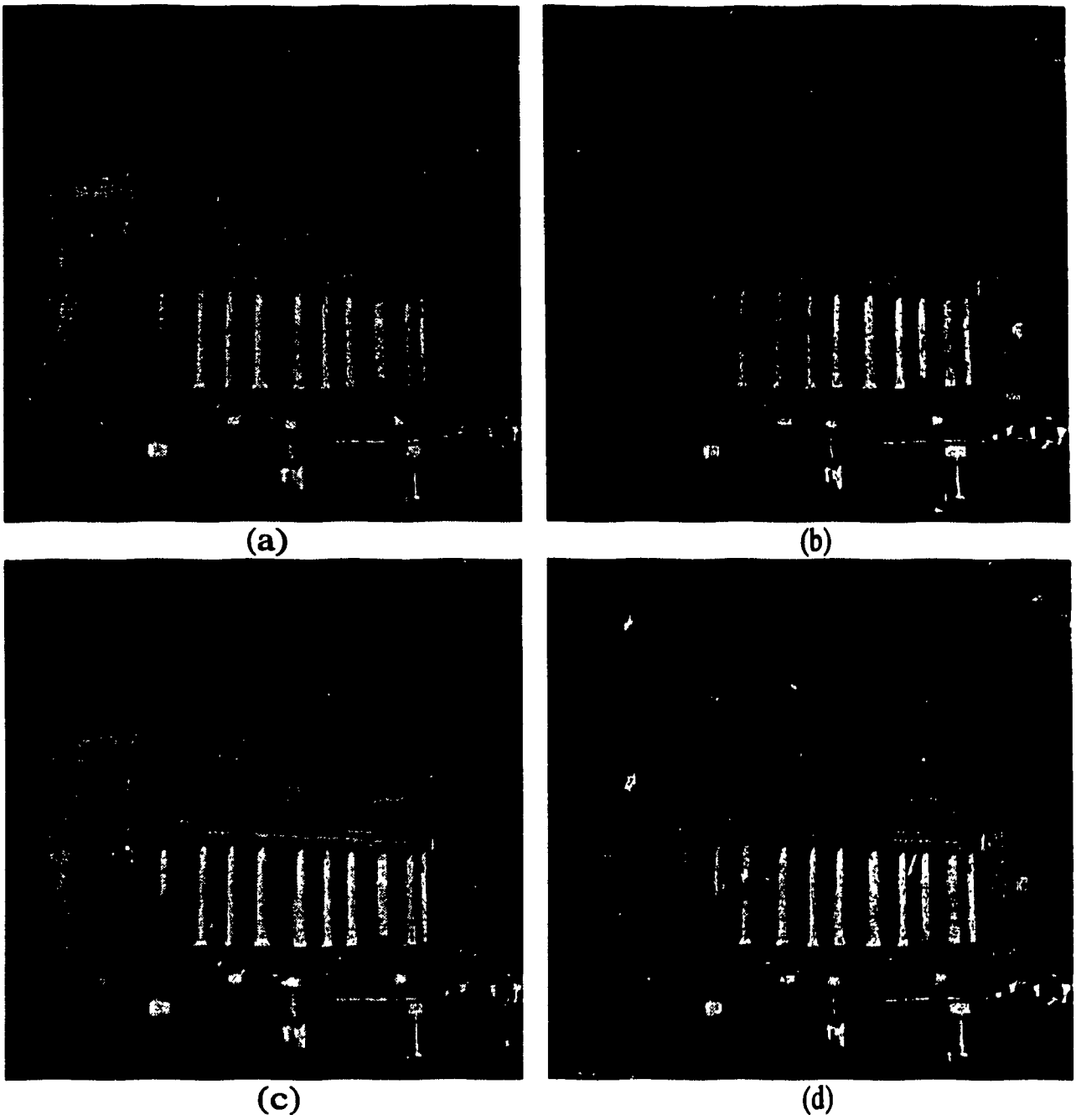


Figure 3.8: The Adaptive HV encoded Columbia Image.

(a) Compression ratio 6.80, PSNR=33.75 dB.

(b) Compression ratio 8.50, PSNR=33.60 dB.

(c) Compression ratio 10.2, PSNR=32.98 dB.

(d) Compression ratio 15.04, PSNR=29.0 dB.

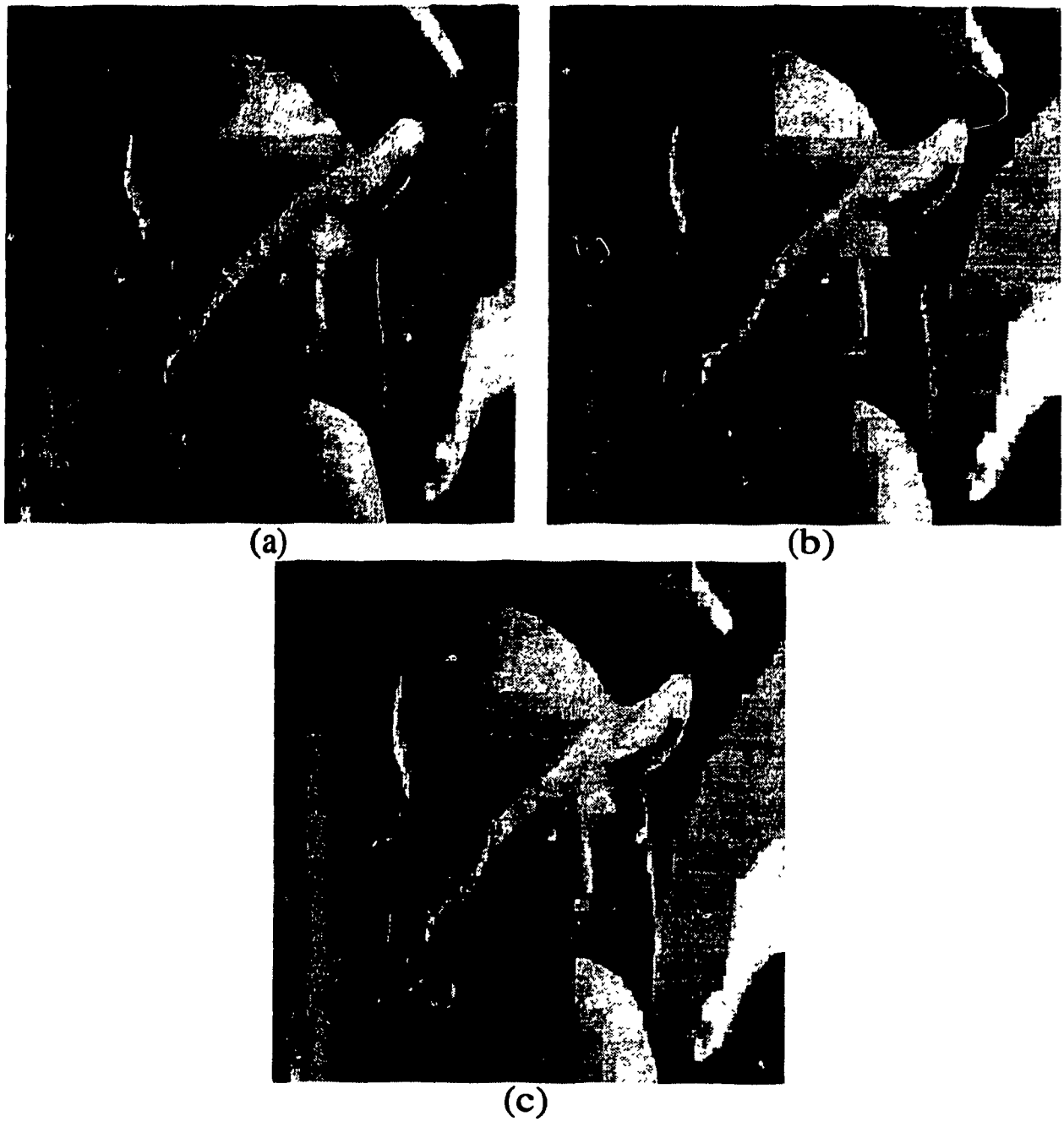


Figure 3.9: Images encoded at very high compression ratio.

(a) QD encoded - compression ratio 28, PSNR=25.6 dB.
(b) HV encoded - Compression ratio 30, PSNR=21.4 dB.
(b) AD encoded - Compression ratio 30, PSNR=24.71 dB.

3.5 Summary

In this chapter, we have proposed an adaptive HV algorithm that tends to utilize the strength of the HV partitioning scheme for fractal block-based image coding, while overcoming its weaknesses. The original algorithm with the HV block partitioning scheme, achieves higher quality coded images, compared to the images obtained using the QD technique, due to a better adaptivity of the range partition to the image content. However, higher quality of the coded images is achieved at the expense of increased coding time. The proposed algorithm has managed to balance these two aspects with a unique approach. The time performance concern has been addressed by reducing the size of the domain library and by performing a full image partition prior to range-domain comparison. The quality of the encoded images has been preserved to that of the original HV scheme by introducing an additional analysis on the block contents in the partitioning algorithm. This has ensured a greater similarity between the partitioned image blocks, and thus, has provided good range-domain candidates within the Partitioned image.

Chapter 4

Hierarchical Interpretation of Fractal Coding

As mentioned in the Chapter 1, fractals are objects that contain detail at every scale and a high degree of self-similarity at different resolutions. Self-similarity is understood in the sense that the whole fractal object is composed of scaled copies of itself. This means that a magnified portion of the fractal looks identical to the whole object, as viewed at the original scale.

These fascinating observations, among the others, were the initial motivation for investigating the use of fractal concepts in image coding applications. However, in the fractal coding schemes that we have presented in this thesis, relations between image representations at different scales have not been clearly outlined, even though they have been implicitly implemented.

All fractal coders give representatior of an image in terms of a map. This map usually consists of a set of nonlinear transformations. Each transformation is represented by a set of parameters and implicitly defined position of the transtormation domain and range. However, the map does not contain information about the scale at which it was initially obtained. This information has to be added in the form of a header in the coded

file. At the decoder, the image size is read from the header, and the mapping parameters are used with an initial image of this size. Experiments have been carried out to omit the header information and to use some different size of initial image [3]. The set of transformations obtained are applied to this initial image. This process results in an image which is the fixed-point for the map in the space of images of the new size. In other words, the same set of transformations applied to two initial images of different sizes would result in two representations of the original image. This particular fact has often been a source of initial misinterpretation of the performance of fractal coders. Spectacular results of the order of 200:1 compression ratios have been claimed, based on the reasoning that if a code obtained on original image of size $N \times N$, is decoded as a $2N \times 2N$ image, the compression ratio would be four times the initial one, since the decoded image would contain four times more pixels [11].

In this chapter, we present a hierarchical interpretation of fractal image coding. First, we give a short review of fractal image coding/decoding from which the question about decoding at different scales naturally arises. Relationship between image representations at different scales is given. Fractal-based coding algorithms are interpreted as a refinement of fine-scale details from coarse-scale information. Based on these results, a practical implementation of a hierarchical coder is proposed.

4.1 Image Encoding/Decoding Review

The goal of a fractal coder is to find a representation of a given image f_0 in the form of a contractive mapping W, such that the fixed-point of W is as close as possible to f_0. A metric function d is defined on the space of all images and it measures the closeness of two images. In the discrete case, an image is represented as a two-dimensional function, where value of the function at every point on the discreet grid corresponds to the grey value of the image at that point.

At the encoder side, we seek a mapping W such that the following requirements are fulfilled:

1. W maps the space of images to itself:

$$W : I^2 \times R \rightarrow I^2 \times R$$
$$v \in I^2 \times R \Rightarrow u = W(v) \in I^2 \times R$$

2. W is a contractive transformation:

$$\exists s \in [0,1) \quad | \quad \forall u, v \in I^2 \times R, \; d(W(u), W(v)) \leq s \cdot d(u,v)$$

These two requirements define the set of all allowed mappings $W \in \mathcal{W}$.

3. W is chosen from $\mathcal{W}$ in such a way as to minimize the distance between f_W, the fixed-point of W, and original image f_0:

$$W = argmin_{V \in \mathcal{W}} d(f_V, f_0)$$

Finding such a mapping W is a very complex problem, since the set $\mathcal{W}$ is extremely large. A suboptimal solution is to limit the number of possible mappings in $\mathcal{W}$ by restricting the type of mapping W. In our work, we have restricted each allowed mapping W to be a set of the transformations w_i, $i = 1, 2, \ldots, N_T^2$ where each w_i is restricted only to a region D_{m_i} of the image. The number of possible mappings is further reduced so that every transformation w_i is of the form

$$w_i : D_{m_i} \rightarrow R_i = w_i(D_{m_i}) = s_{f_i} \cdot S(D_{m_i}) + o_{f_i}. \tag{4.1}$$

where

D_{m_i} - is the m_i-th domain block in a list of all specified possible blocks in f_0, the subscript m_i emphasizing the fact that w_i maps the domain D_{m_i} to R_i.

R_i - denotes a range block. There are N_T^2 range blocks in the image, and they form a nonoverlapping partition of image f_0.

$S(\cdot)$ - denotes spatial contraction, i.e., function that resizes a domain block to the size of a range.

s_{f_i}, o_{f_i} - are scaling and offset factors, respectively.

In the analysis contained in this chapter, we consider a domain to be a square portion of the image f_0 of size M. In addition, the range blocks are considered to be square blocks of size $B = M/2$. We also assume the distance between the adjacent domains to be $\frac{B}{2}$. Such a choice for the range and the domain blocks simplifies the analysis, but should not be taken as a general restriction.

It is clear that the three parameters, s_{f_i}, o_{f_i}, m_i, define entirely one transformation w_i. These parameters should be such that as to minimize $d(f_W, f_0)$, where f_0 is the original image and f_W is the fixed-point of the mapping W sought. By the Collage Theorem, we have

$$d(f_W, f_0) \leq \frac{1}{1-s} d(f_0, W(f_0)). \tag{4.2}$$

Instead of minimizing the $d(f_W, f_0)$, we actually try to minimize the upper bound for it - the right side of the (4.2) - by minimizing $d(f_0, W(f_0))$. Though this method does not necessarily minimize $d(f_W, f_0)$, it is the most practical way of coding, known presently [14]. The minimization of $d(f_0, W(f_0))$ consists of finding W so that $f_0 \approx W(f_0)$. Thus, f_0 is approximately the fixed-point of W. Since W uniquely defines its fixed-point, storing W defines a lossy code for f_0.

The decoding process simply involves finding the fixed-point f_W for the coded contractive mapping W. This is done, as mentioned in the Section 1.1, by iterating W on any initial image until the fixed-point is reached. The only requirement for the initial image is that it has to be of the same size as the size of the original f_0.

There is, however, an interesting observation here. Starting with an initial image of size equal to that of the original image, and iteratively constructing its range blocks of size B yields an approximation of the original image. But if we reduce the size of initial image by a factor of 2, that is, we use the size of range blocks equal to $B/2$ (and accordingly reduce the size of the domain blocks to $M/2$), and keep the same values for the scaling and offset factors, we create a new mapping $W^{\frac{1}{2}}$. The mapping $W^{\frac{1}{2}}$ is also contractive in the space of all images of one-half of the original image size. Decoding process for $W^{\frac{1}{2}}$

leads to the fixed-point $f_W^{\frac{1}{2}}$. The conclusion we reach is that the given PIFS code can be decoded in different image spaces, yielding a different fixed-point in each space. We will see that these different fixed-points are indeed representations of the original image at different scales.

Throughout this chapter, we will use the notation W^q and f^q to denote, respectively, the mapping and the fixed-point which result from the PIFS code when using the range block size $B = qB_1$, where B_1 is the size of range blocks used in calculating the original PIFS code that resulted in W^1, and the corresponding fixed-point f^1. The following section gives relations between these different fixed-points and their interpretations.

4.2 Hierarchical Interpretation

We have seen that the coding information (the set of stored triplets (s_f, o_f, m_i) for every range block) is scale independent. It is the size of the range blocks B that determines the scale of the decoded image. In this section, we derive a relationship between the fixed-point of a given PIFS, obtained by decoding an image at the original scale, and the fixed-points of the same PIFS, obtained by decoding the image at a scale reduced (increased) by a factor of two. Similar analysis for the one-dimensional case is given in [1].

4.2.1 Scale down

Given a PIFS code, the fixed-points f^1 and $f^{\frac{1}{2}}$ of the maps W^1 and $W^{\frac{1}{2}}$, respectively, are uniquely defined. If we have already reconstructed f^1, then $f^{\frac{1}{2}}$ can be calculated according to the following theorem.

Theorem 4.1 *For a given PIFS code, a function g, defined as*

$$g(k,l) = \frac{1}{4}\{f^1(2k,2l) + f^1(2k,2l-1) + f^1(2k-1,2l) + f^1(2k-1,2l-1)\} \qquad (4.3)$$

for $k,l = 1,2,\ldots,N_T\frac{B_1}{2}$, is the fixed-point for the $W^{\frac{1}{2}}$, i.e., $g(k,l) = f^{\frac{1}{2}}(k,l)$.

Proof:

To prove this theorem, we need to show that $g(k,l)$ given by the (4.3) satisfies the equation for the fixed-point of $W^{\frac{1}{2}}$. The fixed-point of the $W^{\frac{1}{2}}$, is characterized by the

$$f^{\frac{1}{2}}_{W} = W^{\frac{1}{2}}(f^{\frac{1}{2}}_{W}). \tag{4.4}$$

First, we introduce a mapping of indices to simplify the notations. We express k and l in $f^{\frac{1}{2}}(k,l)$ as

$$\begin{aligned} k &= (s-1)\tfrac{B_1}{2} + m, \quad s = 1,2,\ldots,N_T; \quad m = 1,2,\ldots,\tfrac{B_1}{2} \\ l &= (t-1)\tfrac{B_1}{2} + n, \quad t = 1,2,\ldots,N_T; \quad n = 1,2,\ldots,\tfrac{B_1}{2} \end{aligned} \tag{4.5}$$

which emphasizes that the element $f^{\frac{1}{2}}(k,l)$ is actually the element (m,n) in the (s,t) range block of $f^{\frac{1}{2}}$. This is illustrated in Figure 4.1.

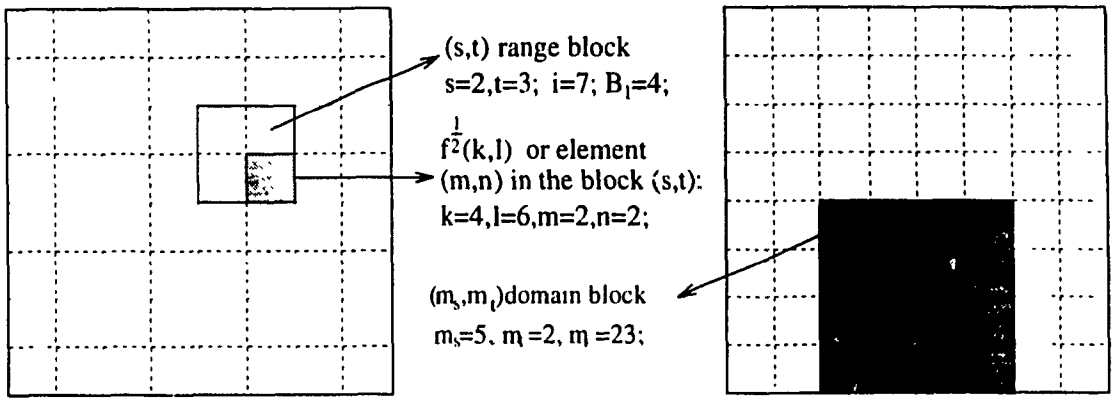


Figure 4.1: Index mapping.

The mapping W is composed of N_T^2 transformations w_i, $i = 1,2,\ldots,N_T^2$, each acting only in a limited region D_{m_i} of the image. Therefore,

$$W^{\frac{1}{2}}(g(k,l)) = s_{f_i} S(D^{\frac{1}{2}}_{m_i})(m,n) + o_{f_i} \tag{4.6}$$

where $S(D^{\frac{1}{2}}_{m_i})(m,n)$ denotes the element (m,n) in the resized block $D^{\frac{1}{2}}_{m_i}$ of the image $g(k,l)$. In order to derive $S(D^{\frac{1}{2}}_{m_i})(m,n)$, four pixels in g are averaged, as given bellow

$$S(D^{\frac{1}{2}}_{m_i})(m,n) = \frac{1}{4} \cdot [g((m_s-1)\,\frac{B_1}{2} + 2m, (m_t-1)\frac{B_1}{2} + 2n)$$

$$+g((m_s-1)\frac{B_1}{2}+2m,(m_t-1)\ \frac{B_1}{2}+2n-1)$$
$$+g((m_s-1)\frac{B_1}{2}+2m\ -1,(m_t-1)\ \frac{B_1}{2}+2n)$$
$$+g((m_s-1)\frac{B_1}{2}+2m-1,(m_t-1)\ \frac{B_1}{2}+2n-1)] \qquad (4.7)$$

where m_s and m_t denote the positions of the upper left corner of the domain block D_{m_i}, i.e.,

$$m_i = (m_s-1)\cdot N_T + m_t.$$

Now, by substituting (4.7) into (4.6), regrouping the resulting terms and expressing each of the four terms using the corresponding elements from f^1, as suggested by (4.3), we obtain

$$\begin{aligned} W^{\frac{1}{2}}(g(k,l)) \;=\; & \frac{1}{4}\cdot[f^1((s-1)B_1+2m,(t-1)B_1+2n)+ \\ & f^1((s-1)B_1+2m,(t-1)B_1+2n-1)+ \\ & f^1((s-1)B_1+2m-1,(t-1)B_1+2n)+ \\ & f^1((s-1)B_1+2m-1,(t-1)B_1+2n-1)]. \end{aligned} \qquad (4.8)$$

By substituting back m and n from (4.5) into (4.8), we have

$$W^{\frac{1}{2}}(g(k,l)) = \frac{1}{4}\,[f^1(2k,2l)+f^1(2k,2l-1)+f^1(2k-1,2l)+f^1(2k-1,2l-1)]. \qquad (4.9)$$

The right side of (4.3) and (4.9) are the same, so we conclude

$$W^{\frac{1}{2}}(g(k,l)) = g(k,l) \qquad (4.10)$$

Thus, we have shown that $g(k,l)$ defined by (4.3) indeed satisfies the equation of the fixed-point of the mapping $W^{\frac{1}{2}}$. Since the fixed-point is unique, $g(k,l) = f^{\frac{1}{2}}$, i.e., we can write

$$f^{\frac{1}{2}}(k,l) = \frac{1}{4}\{f^1(2k,2l)+f^1(2k,2l-1)+f^1(2k-1,1l)+f^1(2k-1,2l-1)\} \qquad (4.11)$$

□

In the derivation of (4.11), we have used the same "trick" as in the practical implementations of our fractal algorithms. An intuitive understanding of the process can be

reached by observing that the domain blocks for f^1, after contraction, are actually the blocks contained in $f^{\frac{1}{2}}$. Thus, instead of resizing every potential domain block, we resize the whole image just once before we start the encoding. When selecting possible domains, we simply choose them from the resized version of the original image instead of choosing them from the original image itself and then resizing them.

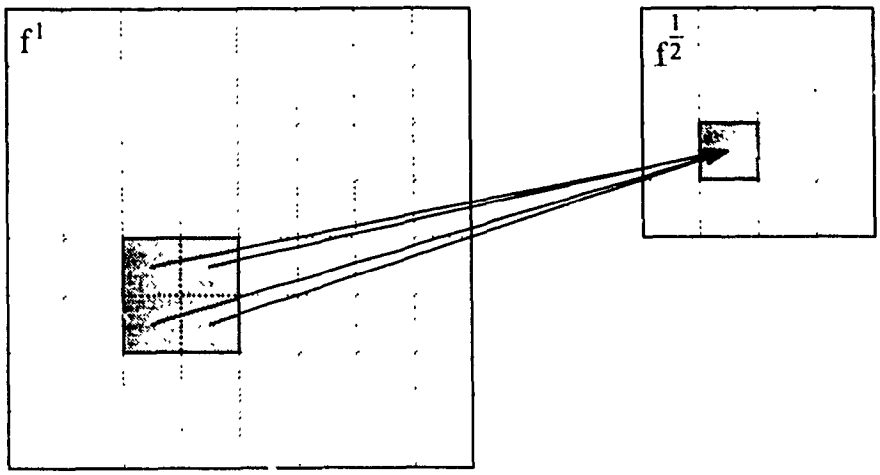


Figure 4.2: Calculating elements of $f^{\frac{1}{2}}$ from f^1.

The interpretation of (4.11) is simple: in order to compute each element of $f^{\frac{1}{2}}$ from a given f^1, we average those four adjacent elements from f^1 that cover the same proportional area in f^1 as does a single element from $f^{\frac{1}{2}}$. This is illustrated in Figure 4.2.

4.2.2 Scale up

The relationship between already calculated fixed-point of $W^{\frac{1}{2}}$, $f^{\frac{1}{2}}$, for a given PIFS, and the corresponding fixed-point of W^1, f^1, is governed by the following theorem.

Theorem 4.2 *For a given PIFS code, a function g defined by*

$$g((s-1)B_1+k,(t-1)B_1+l) = s_{f_i}\cdot f^{\frac{1}{2}}((m_s-1)\frac{B_1}{2}+k,(m_t-1)\frac{B_1}{2}+l) \; + o_{f_i} \quad (4.12)$$

$$s = 1,2,\ldots,N_T;\;\; t = 1,2,\ldots,N_T;$$
$$k = 1,2,\ldots,B_1;\;\; l = 1,2,\ldots,B_1;$$
$$i = (s-1)N_T + t; \quad m_i = (m_s-1)N_T + m_t$$

is the fixed-point of W^1, *that is,* $g(k,l) = f^1(k,l)$.

Proof:

The right side of (4.12) can be rewritten by using (4.11), as

$$\begin{aligned} s_{f_t} f^{\frac{1}{2}}((m_s-1)\tfrac{B_1}{2}+k,(m_t-1)\ \tfrac{B_1}{2}+l)+o_{f_t} &= s_{f_t}\tfrac{1}{4}\{f^1\ ((m_s-1)\tfrac{B_1}{2}+2k,(m_t-1)\tfrac{B_1}{2}+2l) \\ &+ f^1((m_s-1)\tfrac{B_1}{2}+2k,(m_t-1)\tfrac{B_1}{2}+2l-1) \\ &+ f^1((m_s-1)\tfrac{B_1}{2}+2k-1,(m_t-1)\tfrac{B_1}{2}+2l) \\ &+ f^1((m_s-1)\tfrac{B_1}{2}+2k-1,(m_t-1)\tfrac{B_1}{2}+2l-1)\} \end{aligned} \tag{4.13}$$

Equation (4.13) can be rewritten by using the mapping W^1 applied to f^1, giving

$$s_{f_t} f^{\frac{1}{2}}((m_s-1)\tfrac{B_1}{2}+k,(m_t-1)^{\frac{B_1}{2}}2+l)+o_{f_t} = W^1(f^1((s-1)B_1+k,(t-1)B_1+l)). \tag{4.14}$$

By definition, f^1 is the fixed-point of the W^1. Thus,

$$W^1(f^1((s-1)B_1+k,(t-1)B_1+l)) = f^1((s-1)B_1+k,(t-1)B_1+l) \tag{4.15}$$

Finally, from (4.12), (4.14) and (4.15), we have

$$g((s-1)B_1+k,(t-1)B_1+l) = f^1((s-1)B_1+k,(t-1)B_1+l). \tag{4.16}$$

Thus, we conclude that

$$f^1((s-1)B_1+k,(t-1)B_1+l) = s_{f_t}\cdot f^{\frac{1}{2}}((m_s-1)\frac{B_1}{2}+k,(m_t-1)\frac{B_1}{2}+l)\ +o_{f_t}. \tag{4.17}$$

□

From the above theorem, in order to calculate the elements of a given range in f^1, one takes the elements from the corresponding domain block at the scale of $f^{\frac{1}{2}}$ and applies the corresponding transformation w_t to them. This is similar to calculating W^1 itself, by applying the "trick" mentioned above. Figure 4.3 illustrates the process described by (4.17).

Equations (4.11) and (4.17) establish relationships between the pair f^1 and $f^{\frac{1}{2}}$. The same relation is carried over to the pairs $f^{\frac{1}{2}}$ and $f^{\frac{1}{4}}$, $f^{\frac{1}{4}}$ and $f^{\frac{1}{8}}$, and so on. The collection

of the fixed-points for a given PIFS can be described in terms of a hierarchical structure of a pyramid of the fixed-points. The fixed-point $f^{\frac{1}{2^p}}$ comprises the p-th level of the pyramid and it is of size $\frac{N}{2^p} \times \frac{N}{2^p}$. The level on the coarsest scale is called the top level.

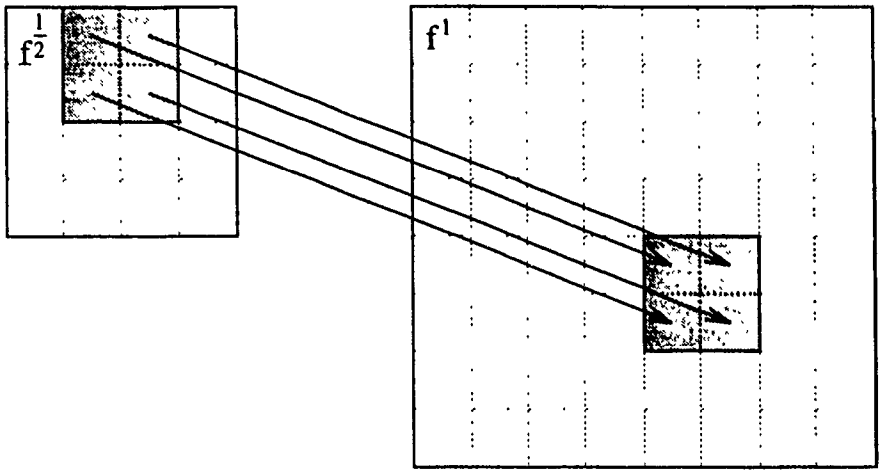


Figure 4.3: Calculating elements of f^1 from $f^{\frac{1}{2}}$.

Generalized versions of (4.11) and (4.17) for the fixed-points at levels p and $p+1$ are, respectively, given by

$$f^{\frac{1}{2^{p+1}}}(k,l) = \frac{1}{4}(f^{\frac{1}{2^p}}(2k,2l) + f^{\frac{1}{2^p}}(2k,2l-1) + f^{\frac{1}{2^p}}(2k-1,2l) + f^{\frac{1}{2^p}}(2k-1,2l-1))$$
$$k = 1,2,\ldots,\tfrac{N_T B_1}{2^{p+1}}, \quad l = 1,2,\ldots,\tfrac{N_T B_1}{2^{p+1}} \tag{4.18}$$

$$f^{\frac{1}{2^p}}((s-1)B_{\frac{1}{2^p}}+k,(t-1)B_{\frac{1}{2^p}}+l) = s_{f_i} \cdot f^{\frac{1}{2^{p+1}}}((m_s-1)B_{\frac{1}{2^{p+1}}}+k,\ (m_t-1)B_{\frac{1}{2^{p+1}}}+l) + o_{f_i}$$
$$s = 1,2,\ldots,N_T, \quad t = 1,2,\ldots,N_T$$
$$k = 1,2,\ldots B_{\frac{1}{2^p}}, \quad l = 1,2,\ldots B_{\frac{1}{2^p}} \tag{4.19}$$

In order to apply PIFS code at the top level, every range block at that level has to be represented by at least one element. This implies that there is a limited number of

levels in the pyramid of PIFS fixed-points. This number is $l = \log_2 B_1 + 1$.

One straight-forward application of the above hierarchical interpretation would be decoding images from the stored PIFS code. In the previous chapters, we have done decoding by iteratively applying a set of transformations to any initial image of size equal to that of the original image. However fast the decoding might be, compared to the encoding process, it includes iterative application of a large number of operations. This number of operations is proportional to the image size. In hierarchical decoding method, we begin by computing the top level (the coarsest representation of the image). This is done in the same manner as before, by iteratively repeating the transformations from the PIFS code. However, this time the image size is much smaller, and the range block size is equal to 1. This leads to a much shorter time until the fixed-point is reached. Then, we follow (4.19) to advance to a higher scale. The process of advancing to a higher scale is repeated until the desired image size is reached. Computational savings are more significant when a large number of iterations is required or the initial size of the range block, used when computing the original PIFS, is large.

Similar approach can be used for increasing the resolution of an image. Namely, (4.19) can be viewed as a means for function interpolation. While linear interpolation tends to smoothen the image, PIFS code interpolation preserves the so called fractal dimension, which ensures the richness of details even at high resolutions. This feature is most evident when dealing with the textures. The linear interpolation typically results in a blurred texture, while the PIFS code interpolation preserves the appearance of the texture.

Both the hierarchical decoding and the PIFS code interpolation assume that the PIFS code is already given. In the following section we intend to use the hierarchical interpretation for the encoding purposes. Our objective is to decrease the encoding time by coding an image at a lower scale first, and then using the coding information thus obtained as a starting point for the encoding at the original image scale.

4.3 Hierarchical Encoding

Motivated by the ideas governing the relations given by (4.11) and (4.17), we propose in this section a hierarchical encoding (HE) algorithm. Our goal is to increase the encoding speed by propagating the relations between the image blocks (for the range-domain pairs determined) from coarser scales to finer scales.

In this algorithm an original image is represented by a set of its versions at several scales. We encode each of these images starting with the largest scale (smallest image). The transformations obtained at a scale (range-domain pairs) are used to limit the domain search for the corresponding ranges at the subsequent finer scales. Our assumption is that if two image areas are found to be similar at one image scale, then their corresponding areas at a finer scale would also be similar. We successively advance to the finer scales until the original image size is reached. In this algorithm, a full range-domain search is done only at the largest scale (the smallest image) while at other scales the search is limited to certain regions only. The benefit of this approach is that block sizes are small at the largest scale, and therefore, the calculation time for each transformation in this and succeeding scales is reduced.

To encode the image at the coarsest resolution, we use a quad-tree method, similar to the one described in Chapter 2. Three scale levels are used: the original image I_1 and the two scaled versions of the original image, $I_{\frac{1}{2}}$ and $I_{\frac{1}{4}}$, scaled by factors of two and four, respectively.

First, a list $\mathcal{L}$ of the triplets: range, domain and transformation between the two is formed: $\mathcal{L}= \{(r_i^{(\frac{1}{4})}, d_i^{(\frac{1}{4})}, w_i^{(\frac{1}{4})}), i = 1,2,\ldots,N_{\frac{1}{4}}\}$. The superscript $(\frac{1}{4})$ denotes that these triplets are obtained by coding the image $I_{\frac{1}{4}}$. The next step is to encode the image $I_{\frac{1}{2}}$. A range-domain pair for $I_{\frac{1}{2}}$ is obtained from the list $\mathcal{L}$. It is chosen in such a way that it covers in $I_{\frac{1}{2}}$ the same proportional area as does the pair from the list in $I_{\frac{1}{4}}$. If the match between the range and the domain in $I_{\frac{1}{2}}$ is not satisfactory, the position of the domain

is adjusted. This is done by searching a limited area that surrounds the original domain position. If this limited search does not result in a good match, then a full search for the domain is performed. If, however, this step also does not give a satisfactory result, the range block is split into four sub-ranges, number of transformations is enlarged by four and for each sub-range a full search for the domain is performed. The coding of $I_{\frac{1}{2}}$ will result in a possibly enlarged new set of triplets $\mathcal{L} = \{(r_i^{(\frac{1}{2})}, d_i^{(\frac{1}{2})}, w_i^{(\frac{1}{2})}), i = 1, 2, \ldots, N_{\frac{1}{4}}\}$. Now, the same process is repeated to encode the original image,I_1, by using the information from the code list $\mathcal{L}$. The proposed algorithm is now formally presented using the pseudo codes.

The HE Algorithm

image_1/2=resize(image_1);
image_1/4=resize(image_1/2);
Quadtree_encode(image_1/4);
Create initial list of range-domain pairs with the corresponding transformations: $\mathcal{L} = \{(r_i, d_i, w_i) \mid i = 1, 2, \ldots, N\}$
tmp_image=image_1/2;

start **For** i=1 **to** N {
Form r_i and d_i blocks in tmp_image that correspond to the r_i and d_i from the list (formed at previous scale);
Compare $w_i(d_i)$ to r_i;
If error $\leq$ Threshold
Keep the triplet (r_i, d_i, w_i) on the list $\mathcal{L}$;
Else {
Search for a better match, look for a new domain in the restricted area around d_i; Find the best d_i;

Compare $w_i(d_i)$ to r_i;

If error $\leq$ Threshold

Save the new triplet (r_i, d_i, w_i) in the list $\mathcal{L}$;

Else {

Split the range r_i into four sub-ranges;

$N = N + 4$;

Search for the best domain for each sub-range

(full search);

Add four new triplets to the list $\mathcal{L}$;

}

}

}

If tmp_image≠image_1 {

tmp_image=image_1;

go to **start**;

}

End;

4.4 Results and Analysis

The training and testing environment for the HE algorithm introduced in this chapter, has been the same as that discussed in section 3.4.

The HE Algorithm includes a large number of parameters that influence the coding results. Apart from the standard parameters that were investigated for the QD and HV schemes (s_{max}, maximum and minimum block sizes, allowed block sizes, error thresholds)

there is a number of parameters inherent to the HE Algorithm. One such parameter to determine is the number of levels up in the pyramid of image representations one ought to go from the original image, i.e. what should be the size of the smallest image that has to be encoded with a full-search QD algorithm. In the implementation of the the algorithm described in the previous section, we have chosen to go two levels up, i.e., to use a full-search QD algorithm on an image of size four times smaller than that of the original image. This choice has been simply empirically based and it has been used to illustrate a practical implementation of the algorithm.

An important characteristic of the HE encoding is the effect of specifying the limited area, surrounding the original domain position, in which the domain search is performed. In our work, we have experimented with different domain search areas. Allowing larger search area results in a slightly smaller number of range blocks. This is due to the fact that larger domain pool yields larger number of good range-domain matches at a coarser scale, so that there is no need to split the block. However, the improvement in the time performance is not significant since longer search is required. We have chosen the limited search area as in Figure 4.4. The domain search area is specified by a square frame of width equal to the size of the domain. The spacing between two adjacent domain candidates within the search area is equal to one-quarter of the domain size both in the horizontal and vertical directions. Thus, for every original domain position, 81 possible domain candidates are compared to the range block.

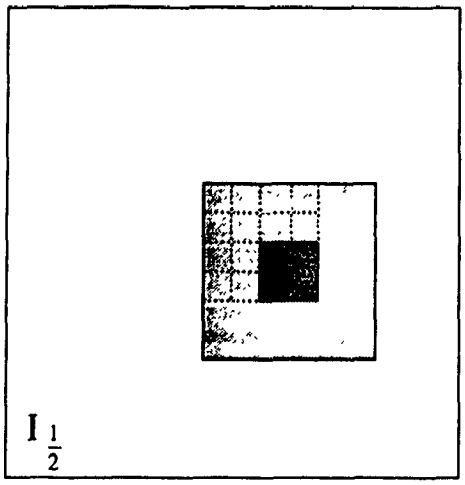


Figure 4.4: Limited domain-search area.
Grid of the domain search area indicates the positions of the upper left corners of all the domain candidate blocks corresponding to the original domain.

Figure 4.5 gives the performance of the HE Algorithm in comparison with that of the QD Algorithm Chapter (2). It can be seen that the HE Algorithm has a better PSNR by more than 1 dB over the whole range of compression ratios. In Figure 4.6, the encoding time for the HE Algorithm is compared to that of the QD Algorithm for the same range of compression ratios. It can be observed that the HE Algorithm has a better time performance for the entire range of compression ratios.

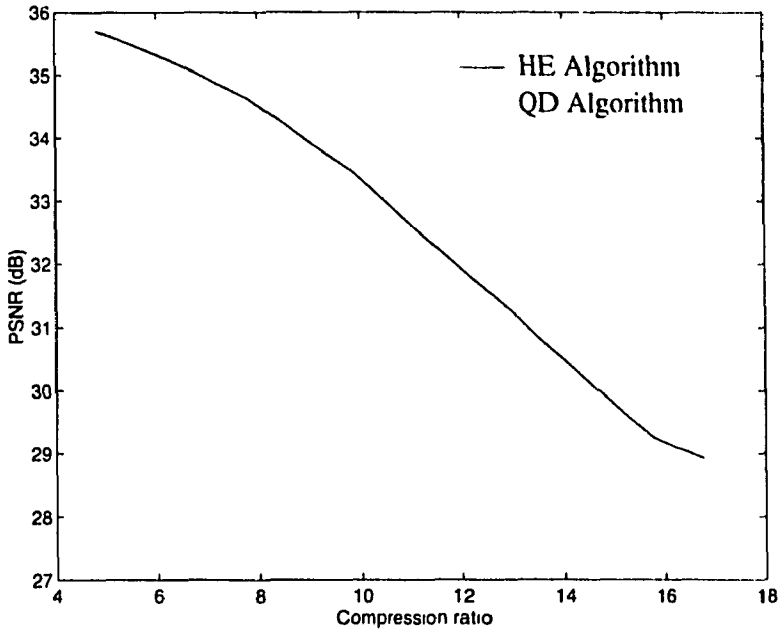


Figure 4.5: Compression ratio vs. PSNR.

In Figures 4.7 and 4.8, we provide examples of images encoded by the HE Algorithm. Since our implementation is based on the quad-tree partitioning on the coarsest scale, these images should be compared with those presented in Figures 2.12 and 2.13, respectively. It can be seen that the images encoded by the HE Algorithm not only have higher PSNR, but their subjective quality is also improved compared to the images encoded by the QD Algorithm.

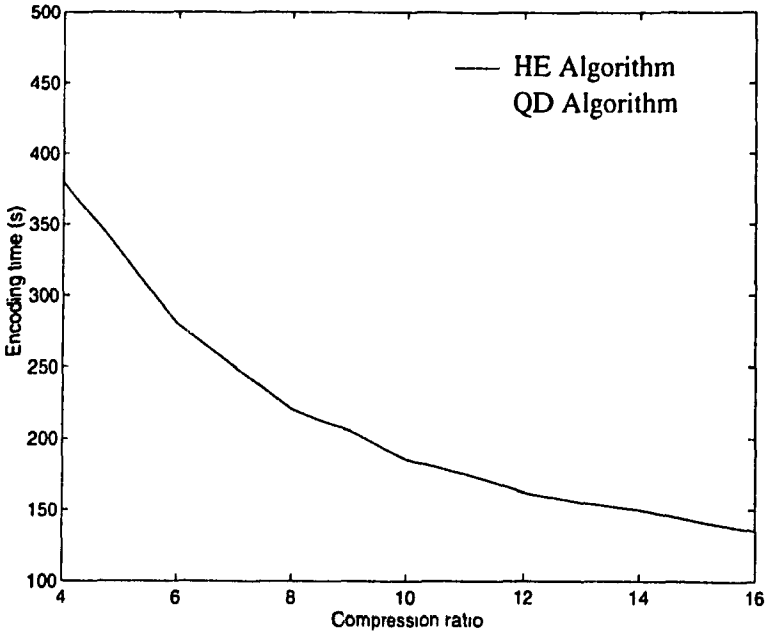


Figure 4.6: Compression ratio vs. the encoding time.

4.5 Summary

In this chapter, we have introduced a hierarchical interpretation of fractal coding algorithms. The strength of the proposed approach is that it can be applied to practically any of the existing fractal-based coding algorithms. In this chapter, we have developed an application based on the quad-tree image block partitioning scheme. It has been shown that the encoding time can be improved by performing a full range-domain search on a coarse image representation only, and by propagating the information thus obtained to the finer scales. The quality of the HE encoded images is also improved compared to that of the standard QD Algorithm. The proposed algorithm, although based on a very simple concept, has been found to provide very encouraging results.

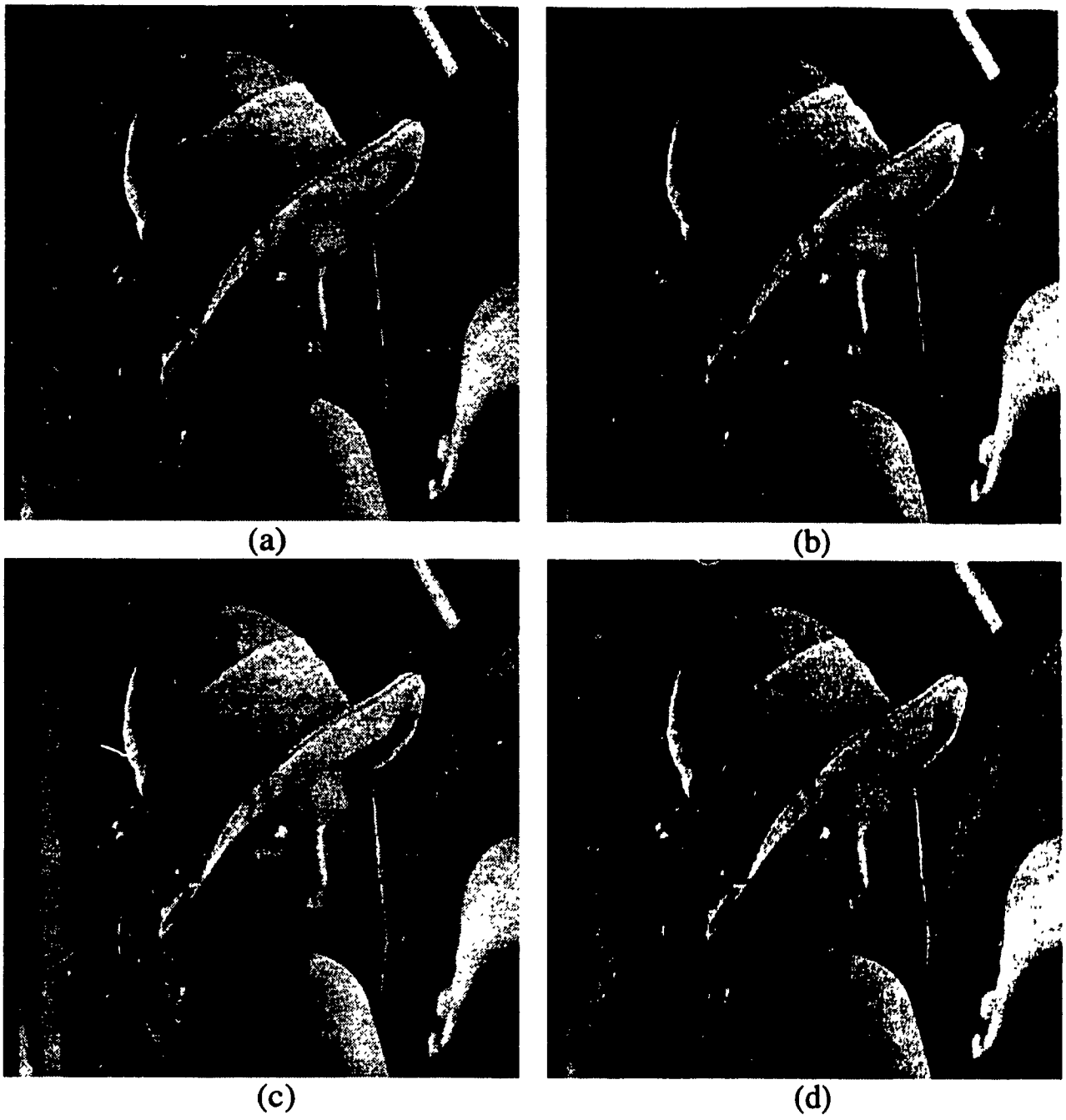


Figure 4.7: The HE encoded Lenna image.

(a) Compression ratio 6.16, PSNR=35.28 dB.

(b) Compression ratio 7.85, PSNR=34.61 dB.

(c) Compression ratio 9.83, PSNR=33.48 dB.

(d) Compression ratio 14.5, PSNR=30.57 dB.

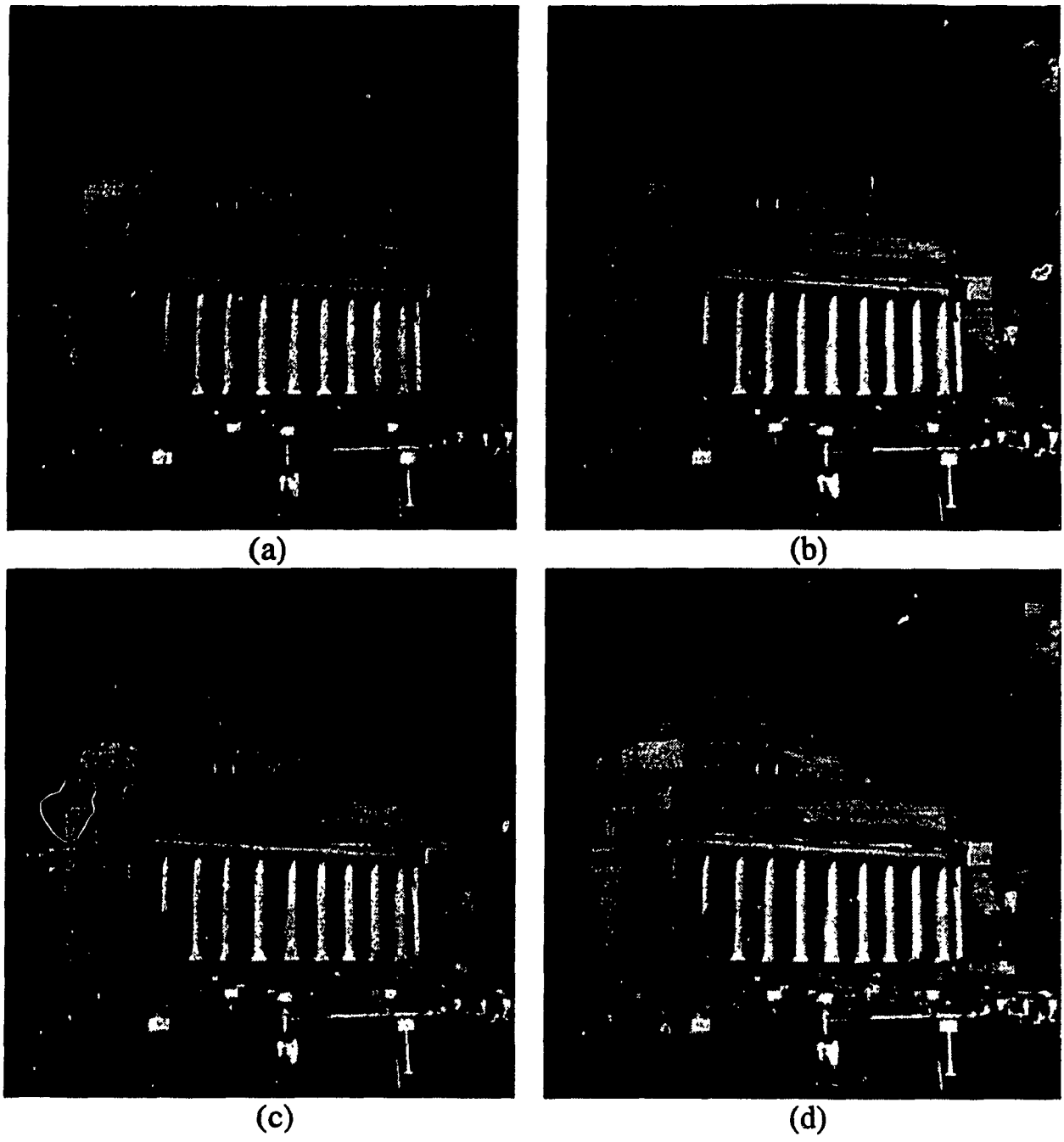


Figure 4.8: The HE encoded Columbia image.

(a) Compression ratio 6.73, PSNR=34.32 dB.

(b) Compression ratio 8.51, PSNR=34.10 dB.

(c) Compression ratio 10.17, PSNR=33.12 dB.

(d) Compression ratio 14.90, PSNR=29.80 dB.

Chapter 5

Conclusions and Suggestions for Future Investigation

5.1 Concluding Remarks

This thesis has been concerned with the development of an adaptive approach to the problem of finding similarity relations between the image blocks, and of a hierarchical, multi-scale fractal-based image coding scheme.

In the first part of the contribution, it has been shown that the partitioning schemes that are more adapted to the image content, result in better quality coded images. However, the higher degree of adaptivity in the image partitioning schemes comes at the expense of an increased encoding time. This has been illustrated by the analysis of the two existing fractal block-based image coding algorithms, namely the Quad-Tree and the HV algorithms. The original algorithm with the HV block partitioning scheme, achieves higher quality coded images, compared to the images obtained using the QD technique, due to a better adaptivity of the range partition to the image content in the former. However, the encoding time is increased for the HV algorithm. In the proposed adaptive HV scheme, a balance is achieved between preserving the image quality and maintaining a reasonable encoding time. It has been observed that a selective reduction of the domain

library size may be beneficial to improving the encoding time performance. The proposed adaptive HV algorithm introduces a mechanism of simple block contents analysis into the partitioning scheme itself, which results in a higher degree of similarity between the blocks at different levels in the partitioning tree, thus providing a good choice of domain blocks within the image partition itself. Consequently, the size of the domain library is reduced in a manner adapted to the image content. It has been shown that the proposed algorithm gives encoded images of quality comparable to that of the existing state of the art HV fractal encoding scheme, with the encoding times significantly reduced. Moreover, the proposed algorithm allows application of simple classification schemes to the domain library, thus making further decrease in the encoding times possible.

In the second part of the contribution, a multi-scale fractal coding interpretation has been introduced. The relationships between the fixed-points of a given PIFS, obtained on different scales, have been derived and applied to the image coding problem. The strength of this approach is that it can be applied to practically any of the existing fractal-based coding algorithms. A practical implementation of this hierarchical approach has been proposed. It has been shown that the encoding time can be improved by performing a full range-domain search on a coarse image representation only, and by propagating the information thus obtained to the finer scales. The results obtained by an implementation of the proposed approach to a quad-tree based image coding scheme, has shown a significant improvement in both the coded image quality and the encoding time. Although the proposed approach is based on a very simple concept, it has provided some very encouraging results.

5.2 Scope for Future Work

One of the possible directions for future work on the topic of block-based fractal image coding would be that of implementing some of the partitioning schemes that include orientations for the block boundaries other than the horizontal or vertical. The implementation may be based on a triangular or a polygonal image partitioning similar to that described in [29]. It can be expected that introduction of new orientations for the block boundaries would result in increased compression ratios, since there would be less range

blocks required to well approximate the non-horizontal or non-vertical edges. However, a solution would have to be found for the increase in the encoding time, as more complicated partitioning schemes could be expected to be more time consuming.

Multi-scale fractal coding opens new challenges that can also be investigated in possible future work. Experiments with using different types of filters for building the pyramidal representation for an image can be performed. Filters that have better edge preserving properties are expected to give images of even higher fidelity than that of the images obtained by the scheme proposed in this work.

References

[1] Z. Barahav, D. Malah, and E. Kernin. Hierarchical interpretation of fractal image coding and its application to fast decoding. In *International Conference on Digital Signal Processing*, Cyprus, July 1993.

[2] Michael F. Barnsley. Fractal modelling of real world images. In Heinz-Otto Peitgen and Dietmar Saupe, editors, *The Science of Fractal Images*, chapter 5, pages 219–239. Springer-Verlag, 1988.

[3] Michael F. Barnsley. *Fractals Everywhere.* Academic Press, San Diego, 1988.

[4] Michael F. Barnsley, Arnaud Jacquin, Francois Malassenet, Laurie Reuter, and Alan D. Sloan. Harnessing chaos for image synthesis. *Computer Graphics*, 22(4):131–140, August 1988.

[5] Geoff Davis. Adaptive self-quantization of wavelet subtrees: A wavelet-based theory of fractal image comression. In *SPIE Conference on Mathematical Imaging: Wavelet Applications in Signal and Image Processing*, July 1995.

[6] Geoff Davis. A wavelet-based analysis of fractal image compression. *IEEE Transactions on Image Processing*, 1996. submitted.

[7] F. Davoine and J-M. Chassery. Adaptive delaunay triangulation for attractor image coding. In *12th International Conference on Pattern Recognition*, October 1994.

[8] Branka Dzerdz and M.O Ahmad. An adaptive fractal-based algorithm for image compression. In *Proceedings of the Canadien Conference in Electrical and Computer Engineering*, September 1995.

[9] R. D. Boss E. W. Jacobs, Y. Fisher. Image compression: A study of the iterated transform method. *Signal Processing*, 29:251–263, 1992.

[10] Paul Michael Farrelle. *Recursive block coding for image data compression.* Springer-Verlag, New York, 1990.

[11] Yuval Fisher. A discussion of fractal image compression. In Heinz-Otto Peitgen, Hartmut Jürgens, and Dietmar Saupe, editors, *Chaos and Fractals: New Frontiers of Science*, chapter Appendix A, pages 903–919. Springer-Verlag, New York, 1992.

[12] Yuval Fisher. Fractal encoding with hv partitions. In Yuval Fisher, editor, *Fractal Image Compression: Theory and Application*, chapter 6, pages 119–136. Springer-Verlag, 1995.

[13] Yuval Fisher, editor. *Fractal Image Compression: Theory and Application.* Springer-Verlag, New York, 1995.

[14] Yuval Fisher. Fractal image compression with quadtrees. In Yuval Fisher, editor, *Fractal Image Compression: Theory and Application*, chapter 3, pages 55 77. Springer-Verlag, 1995.

[15] Arnaud Jacquin. A novel fractal block-coding technique for digital images. In *Proceedings of ICASSP*, volume 4, pages 2225–2228, 1990.

[16] Arnaud E. Jacquin. Image coding based on a fractal theory of iterated contractive image transformations. *IEEE Transactions on Image Processing*, 1(1):18-30, January 1992.

[17] Arnaud E. Jacquin. Fractal image coding: A review. *Proceedings of the IEEE*, 81(10):1451–1465, October 1993.

[18] Michael S. Lazar. *Applications of multiresolution analysis in multidimensional signal processing.* PhD thesis, The University of Calgary, 1994.

[19] S.G. Mallat. Multifrequency channel decompositions of images and wavelet models. *IEEE Transaction on Accoustic, Speech, and Signal Processing*, 37(12):2091–2110, December 1987.

[20] S.G. Mallat. A theory for multiresolution signal decomposition: The wavelet representation. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 11(7):674–693, July 1989.

[21] D.M. Monro and F. Dudbridge. Fractal approximation of image blocks. In *IEEE Proceedings of ICASSP*, volume III, pages 485–488, 1992.

[22] D.M. Monro and F. Dudbridge. Fractal block coding of images. *Electronics Letters*, 28(11):1053–1055, May 1992.

[23] Geir E. Øien, Skjalg Lepsøy, and Tor A. Ramstad. An inner product space approach to image coding by contractive transformations. In *International Conference on Accoustics, Speech and Signal Processing*, pages 2773–2776, 1991.

[24] H.-O. Peitgen, D. Saupe, and H. Jurgens. *Fractals for the Classroom.* Springer-Verlag, New York, 1991.

[25] Heinz-Otto Peitgen, Hartmut Jürgens, and Dietmar Saupe. Encoding images by simple transformations. In *Chaos and Fractals: New Frontiers of Science*, chapter 5, pages 229–296. Springer-Verlag, New York, 1992.

[26] Heinz-Otto Peitgen and Dietmar Saupe, editors. *The Science of Fractal Images.* Springer-Verlag, New York, 1988.

[27] B. Ramamurthi and A. Gersho. Classified vector quantization of images. *IEEE Transaction on Communications*, 34(11):1105–1115, November 1986.

[28] Dietmar Saupe. Breaking the time complexity of fractal image compression. Technical report, Institut für Informatik, Universität Freiburg, 1994.

[29] Xiaolin Wu and Yonggang Fang. A segmentation-based predictive multiresolution image coder. *IEEE Transactions on Image Processing*, 4(1):34–47, January 1995.