Fractal Image Compression via Nearest Neighbor Search[1]

Dietmar Saupe
Institut für Informatik, Universität Freiburg
Am Flughafen 17, 79110 Freiburg, Germany
Email: saupe@informatik.uni-freiburg.de.

January 9, 1996

[1]This paper was held at the NATO Advanced Study Institute *Fractal Image Encoding and Analysis,* in Trondheim, Norway, in July 1995. It also appears in Conf. Proc. NATO ASI *Fractal Image Encoding and Analysis,* Trondheim, July 1995, Y. Fisher (ed.), Springer-Verlag, New York, 1996.

Abstract

In fractal image compression the encoding step is computationally expensive. A large number of sequential searches through a list of domains (portions of the image) are carried out while trying to find best matches for other image portions called ranges. Our theory developed here shows that this basic procedure of fractal image compression is equivalent to multi-dimensional nearest neighbor search in a space of feature vectors. This result is useful for accelerating the encoding procedure in fractal image compression. The traditional sequential search takes linear time whereas the nearest neighbor search can be organized to require only logarithmic time. The fast search has been integrated into an existing state-of-the-art classification method thereby accelerating the searches carried out in the individual domain classes. In this case we record acceleration factors up to about 50 depending on image and domain pool size with negligible or minor degradation in both image quality and compression ratio. Furthermore, as compared to plain classification our method is demonstrated to be able to search through larger portions of the domain pool without increasing the computation time. In this way both image quality and compression ratio can be improved at reduced computation time. We also consider the application of a unitary transformation of the feature vectors which results in a reduction of the dimensionality of the search space. New results from numerical simulations are reported. Also we provide a brief overview of related work and other complexity reduction methods. This paper is an extended version of the article [34].

1 Introduction

With the ever increasing demand for images, sound, video sequences, computer animations and volume visualization, data compression remains a critical issue regarding the cost of data storage and transmission times. While JPEG currently provides the industry standard for still image compression there is ongoing research in alternative methods. Fractal image compression is one of them.

1.1 The search problem in fractal image encoding

Basically, a fractal code consists of three ingredients: a partitioning of the image region into portions R_k, called *ranges,* an equal number of other image regions D_k, called *domains* (which may overlap), and for each domain-range pair two *transformations*, a geometric one, $u_k : D_k \to R_k$, which maps the domain to the range, and an affine transformation, v_k, that adjusts the intensity values in the domain to those in the range. The collection of transformations may act on an arbitrary image, g, producing an output image, Tg, which is like a collage of modified copies of the domains of the image g. The iteration of the image operator T is the decoding step, i.e., it yields a sequence of images which converges to an approximation of the encoded image.

The time consuming part of the encoding step is the search for an appropriate domain for each range. The number of possible domains that theoretically may serve as candidates is prohibitively large. For example, the number of arbitrarily sized square subregions in an image of size n by n pixels is of order $O(n^3)$. Thus, one must impose certain restrictions in the specification of the allowable domains. In a simple implementation one might consider as domains, e.g., only sub-squares of a limited number of sizes and positions. This defines the so-called *domain pool.* Now for each range in the partition of the original image all elements of the domain pool are inspected: for a given range R_k and a domain D the transformations u_k and v_k are constructed such that when the domain image portion is mapped into the range the result $v_k f u_k^{-1}(x)$ for $x \in R_k$ matches the original image f as much as possible. This step (called collage coding) uses the well-known least squares method. From all domains in the pool we select the best one, i.e., the domain D that yields the best least squares approximation of the original image in the range. In other words, fractal image coding consists in approximating the image as a collage of transformed pieces of itself, which can be viewed as a collection of self-similarity properties. The better the collage fits the given image the higher the fidelity of the resulting decoded image.

In this article we cannot explain any further details and variations of fractal image compression. For introductory texts or reviews see, for example, [4, 12, 13, 22]. For a bibliographic survey of the field of fractal image compression see our paper [36] and the updated list of references in [38].

JPEG can be termed *symmetric* in the sense that the encoding and decoding phases require about the same number of operations. On the contrary, fractal image compression allows fast decoding but suffers from long encoding times. In our papers [32, 33] we introduced and discussed a new twist for the encoding process. In [34] we demonstrated its efficiency by a series of empirical studies. In this expository article we review the material in [34] and extend the discussion of the acceleration technique.

During encoding a large pool of image subsets, the domain pool, has to be searched repeatedly many times, which by far dominates all other computations in the encoding process. If the number of domains in the pool is N, then the time spent for each search is *linear* in N, $O(N)$. Previous attempts to reduce the computation times employ *classification schemes* for the domains based on image features such as edges or bright spots. Thus, in each search only domains from a particular class need to be examined. However, this approach reduces only the factor of proportionality in the $O(N)$ complexity.

The main idea of the acceleration is the following. First we show that the fundamental searching for optimal domain-range pairs is equivalent to solving nearest-neighbor problems in a suitable Euclidean space of feature vectors of domains and ranges. The data points are given by the feature vectors (also called multi-dimensional keys) of the domains, while the query point is defined as the feature vector of a given range.

Our application of this reasoning is that we may substitute the sequential search in the domain pool (or in one of its classes) by multi-dimensional nearest neighbor search. There are well known data structures and algorithms for this task which operate in *logarithmic* time, $O(\log N)$, a definite advantage over the $O(N)$ complexity of the sequential search. Our implementation and empirical studies show that these time savings in fact do provide a considerable acceleration of the encoding and, moreover, allow an enlargement of the domain pool yielding improved image fidelity.

1.2 Previous work

The problem of time complexity in fractal image compression was already very clear right from the beginning. The earliest — and some of the latest — implementations use the concept of classification as a tool for complexity reduction. The classification of ranges and domains serves the purpose of reducing the number of domains in the domain pool which need to be considered as a partner for a given range. Just like in 'real life', birds of a feather flock together. For example, if the original image contains an edge running through a range, then domains which contain only 'flat' pieces of the image can be safely discarded when searching for a good match for that range. Jacquin [21, 22] sorts ranges and domains into three classes (shade blocks, edge blocks, and midrange blocks) following a classification, well-known in image processing. The classification of Fisher, Jacobs, and Boss [20, 13] is made with a clever design of a variable number of classes (3–24–72) taking into account not only intensity values but also intensity variance across an image block.[1] Here an ordering of variances in the four sub-quadrants of an image block is used. Although successful this approach is not satisfying in the sense that a notion of neighboring classes is not available. So if the search in one class does not yield a sufficiently strong match for

[1]Since we will use Fisher's implementation as a test bed for our experiments we will describe their approach in some more detail in section 5.

a domain, one cannot extend the search to any neighboring classes. The solution for this problem has been given by Caso, Obrador and Kuo in [9], where the unflexible ordering of variances of an image block has been replaced by a vector of variances. These variance vectors are strongly quantized leading to a collection of classes where each class has a neighborhood of classes which can be searched.

Attempts have also been made to design the set of classes adaptively, i.e. depending upon the target images. Lepsøy and Øien [26] proposed an adaptive codebook clustering algorithm and Boss and Jacobs [8] considered an archetype classification based on a set of training images. A first algorithm based on the self-organizing map (SOM) of Kohonen for codebook clustering was presented by Bogdan in [7]. In [17] the promising SOM approach for clustering has been combined with the nearest neighbor approach presented here.

In addition to the complexity reduction by using these classification schemes Bedford, Dekking, and Keane [6] suggested to use inner products with Rademacher functions to further exclude certains domains from consideration for a partner to a given range. Three multi-resolution approaches with the same goal are presented by Caso, Obrador and Kuo in [9], by Dekking in [10, 11], and by Lin and Venetsanopoulos in [27]. Another approach employing an incremental procedure at the pixel level has been given by Bani-Eqbal in [2, 3]. For a survey of some of these complexity reduction methods see the article [37].

Complexity reduction methods that are somewhat different in character are based on reducing the domain pool rigorously to a small subset of all possible domains. For example, in the work that followed Monro and Dudbridge [28] for each range the codebook block to be used to cover the range is uniquely predetermined to be a specific block that contains the range block. A similar idea has been pursued by Hürtgen and Stiller [19] where the search area for a domain is restricted to a neighborhood of the current range or additionally a few sparsely spaced domains far from the range are taken into account as an option. In [35] we have noticed that one can discard domain blocks with low variance values with a little degradation in fidelity for which we can compensate by an improved storage scheme for the domain addresses. Signes [39] and Kominek [24] pursue similar ideas for domain pool reduction. An adaptive version of spatial search based on optimizing the rate-distortion performance is presented by Barthel in [5].

The remainder of this chapter is organized as follows. In the following section we present mathematical notation and a first simple formula for the least squares error based on projections. In section 3 we provide the definition of the multi-dimensional keys, a theorem that establishes the mathematical foundation, and a corollary giving a necessary and sufficient condition in terms of the multi-dimensional feature vectors for a domain codebook block in order to satisfy a given tolerance criterion for the least squares error. The following sections 4 and 5 outline some practical comments on the implications of the theory and explain our implementation. Section 6 discusses our experiments and their results. Finally, in section 7, we discuss other work as far as it relates to feature vectors and then give a conclusion.

2 A formula for the least squares error based on projections

For the discussion in the paper let us assume that an image is partitioned into non-overlapping square blocks of size $N \times N$ called *range blocks.* This is not a restriction since it will be clear how the principles described carry over to more general partitions.

We consider each range block as a vector R in the linear vector space $\mathbf{R}^n$ where $n = N \times N$. The conversion from a square subimage of side length N to a vector of length $n = N^2$ can be accomplished, e.g., by scanning the block line by line. Working with vectors in place of 2D-arrays simplifies the notation considerably without losing generality.

The *domain pool* is a collection of square blocks which are typically larger than the ranges and taken also from the image, called *domain blocks.* The domain pool is enlarged by including blocks obtained after applying the eight isometrical operators to the domain blocks (i.e., rotations and reflections). Finally, by pixel averaging, the size of these blocks is reduced to the size of a range block. The resulting blocks are called *codebook blocks.*

In the encoding process for a range block a search through the codebook blocks is required. A vector representing a codebook block will be denoted by D. A small set of $p < n$ blocks independent from the image is also considered. We represent them by the vectors $B_1, B_2, \ldots, B_p \in \mathbf{R}^n$, which are chosen so as to form an orthonormal basis of a p-dimensional subspace of $\mathbf{R}^n$. They are known as *the fixed basis blocks.* The encoding problem can then be stated as the least squares problem

$$E(D,R) = \min_{a,b_1,\ldots,b_p \in \mathbf{R}} \|R - (aD + \sum_{k=1}^{p} b_k B_k)\| = \min_{x \in \mathbf{R}^{p+1}} ||R - Ax||, \qquad (1)$$

where A is an n by $p+1$ matrix whose columns are $D, B_1, B_2, \ldots, B_p$ and $x = (a, b_1, \ldots, b_p) \in \mathbf{R}^{p+1}$ is a vector of coefficients.[2] This problem should be solved for all codebook blocks D and the one which gives the smallest error $||R - (aD + \sum_1^p b_k B_k)||$ is selected on condition that the value of the scaling factor a for the codebook block D ensures the convergence of the decoding process (e.g., by requiring $|a| < 1$). This condition on a can be removed when one uses the orthogonalized representation of Øien [30]. A basic result of linear algebra states that if the codebook block D is not in the linear span of the fixed basis blocks $B_1, \ldots, B_p$, then the minimization problem (1) has the unique solution

$$\bar{x} = (A^T A)^{-1} A^T R$$

where the matrix $A^+ = (A^T A)^{-1} A^T$ is also known as the pseudo-inverse of A. Thus, the range block R is approximated by the *collage* block AA^+R where AA^+ is the orthogonal projection matrix onto range(A). Now let P be the orthogonal projection operator which projects $\mathbf{R}^n$ onto the subspace $\mathcal{B}$ spanned by only the fixed basis

[2]In practise usually the root mean square error (rms) is used equivalently in place of $E(D,R)$. This is just $E(D,R)/\sqrt{n}$. Also note, that we have used the Euclidean norm instead of the squared norm. We use the notation $\langle \cdot, \cdot \rangle$ for the common inner product in $\mathbf{R}^n$, thus, $||x|| = \sqrt{\langle x, x \rangle}$.

blocks $B_1, B_2, \ldots, B_p$. Thus, by orthogonality of the fixed basis blocks we have for $R \in \mathbf{R}^n$

$$PR = \sum_{k=1}^{p} b_k B_k = \sum_{k=1}^{p} \langle R, B_k \rangle B_k.$$

Then the range block R has a unique orthogonal decomposition $R = OR + PR$ where the operator $O = I - P$ projects onto the orthogonal complement $\mathcal{B}^{\perp}$. For $Z = (z_1, \ldots, z_n) \in \mathbf{R}^n \backslash \mathcal{B}$, we define the operator

$$\phi(Z) = \frac{OZ}{||OZ||}. \tag{2}$$

Now for a given domain block $D \notin \mathcal{B}$ the collage block $AA^{+}R$ can be given explicitly as

$$AA^{+}R = \langle R, \phi(D) \rangle \phi(D) + \sum_{k=1}^{p} \langle R, B_k \rangle B_k. \tag{3}$$

To get the least squares error we use the orthogonality of $\phi(R), B_1, \ldots, B_p$ to express the range block R as

$$R = \langle R, \phi(R) \rangle \, \phi(R) + \sum_{k=1}^{p} \langle R, B_k \rangle B_k. \tag{4}$$

We insert the result for R in the first part of the collage block $AA^{+}R$ in (3) and after three lines of computations find that

$$\langle R, \phi(D) \rangle \phi(D) = \langle R, \phi(R) \rangle \, \langle \phi(D), \phi(R) \rangle \, \phi(D).$$

Thus, the collage block can be rewritten as

$$AA^{+}R = \langle R, \phi(R) \rangle \, \langle \phi(D), \phi(R) \rangle \, \phi(D) + \sum_{k=1}^{p} \langle R, B_k \rangle B_k. \tag{5}$$

Using (4) and (5) we can now compute the least squares error

$$E(D, R) = ||R - AA^{+}R|| = \sqrt{\langle R - AA^{+}R, R - AA^{+}R \rangle}.$$

The result follows after a few lines of calculations, namely

$$E(D, R) = \langle R, \phi(R) \rangle \, \sqrt{1 - \langle \phi(D), \phi(R) \rangle^2}. \tag{6}$$

Thus, the minimization of the error $E(D, R)$ among domain codebook blocks D can be achieved using an *angle criterion:* The minimum of $E(D, R)$ occurs when the squared inner product $\langle \phi(D), \phi(R) \rangle^2$ is maximal. Since $\langle \phi(D), \phi(R) \rangle^2 = \cos^2 \angle(\phi(D), \phi(R))$ this means minimizing the angle $\angle(\phi(D), \phi(R))$, or, equivalently $\angle(OD, OR)$.

In the next section we will see how the expression (6) for the least squares error can be seen in terms of the distances between $\phi(R)$ and $\pm\phi(D)$.

3 Searching in fractal image compression is nearest neighbor search

We consider a set of N_D codebook blocks $D_1, \ldots, D_{N_D} \in \mathbf{R}^n$ and a range block $R \in \mathbf{R}^n$. As before we let $E(D_i, R)$ denote the smallest possible least squares error of an approximation of the range data R by an affine transformation of the domain data D_i (see (1) and (6)). Then the searching in fractal image compression consists in finding the domain codebook block index k with

$$k = \arg \min_{i=1,\ldots,N_D} E(D_i, R).$$

The following theorem provides the mathematical foundation for our feature vector approach. We generalize the original version of the theorem to the case of several fixed basis blocks. We use the notation for the normalized projection operator ϕ and the linear span $\mathcal{B}$ of the (orthonormalized) fixed basis blocks $B_1, \ldots, B_p$ as before.

Theorem 1 [32, 33].
Let $n \geq 2$ and $X = \mathbf{R}^n \backslash \mathcal{B}$. Define the function $\Delta : X \times X \to [0, \sqrt{2}]$ by

$$\Delta(D, R) = \min\left(||\phi(R) + \phi(D)||, ||\phi(R) - \phi(D)||\right).$$

For $D, R \in X$ the least squares error

$$E(D, R) = \min_{a, b_1, \ldots, b_p \in \mathbf{R}} ||R - (aD + \sum_{k=1}^{p} b_k B_k)||$$

is given by

$$E(D, R) = \langle R, \phi(R) \rangle \, g(\Delta(D, R))$$

where

$$g(\Delta) = \Delta \sqrt{1 - \frac{\Delta^2}{4}}.$$

Proof. Since

$$\begin{aligned} ||\phi(R) \pm \phi(D)|| &= \sqrt{\langle \phi(R) \pm \phi(D), \phi(R) \pm \phi(D) \rangle} \\ &= \sqrt{\langle \phi(R), \phi(R) \rangle \pm 2\langle \phi(D), \phi(R) \rangle + \langle \phi(D), \phi(D) \rangle} \\ &= \sqrt{2(1 \pm \langle \phi(D), \phi(R) \rangle)} \end{aligned}$$

we have

$$\Delta(D, R) = \min\left(||\phi(R) \pm \phi(D)||\right) = \sqrt{2(1 - |\langle \phi(D), \phi(R) \rangle|)}.$$

This last equation yields $|\langle \phi(D), \phi(R) \rangle| = 1 - \Delta(D, R)^2/2$ and inserting the square of that in the formula (6) for $E(D, R)$ completes the proof.

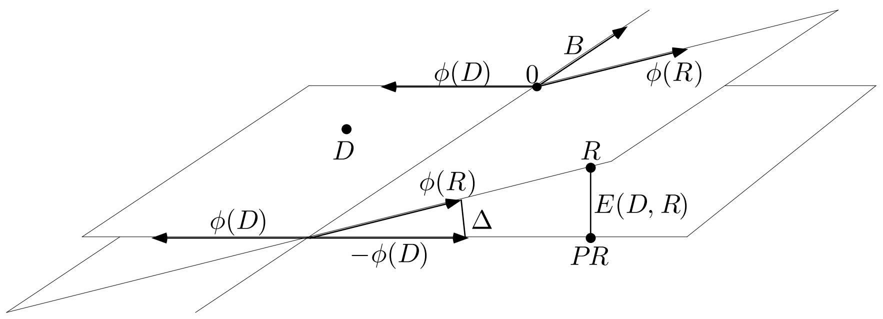

Figure 1: Illustration of the geometry underlying the theorem.

The theorem states that the least squares error $E(D_i, R)$ is proportional to the simple function g of the Euclidean distance Δ between the normalized projections $\phi(D_i)$ and $\phi(R)$ (or $-\phi(D_i)$ and $\phi(R)$). The value of the result is not in terms of a speed-up of the calculation of the least squares error, but of a more fundamental nature. Since $g(\Delta)$ is a monotonically increasing function for $0 \leq \Delta \leq \sqrt{2}$ we conclude that *the minimization of the least squares errors $E(D_i, R)$ for $i = 1, \ldots, N_D$ is equivalent to the minimization of the distance expressions* $\Delta(D_i, R)$. Formally, the sought index k is

$$\begin{aligned} k = \arg \min_{i=1,\ldots,N_D} E(D_i, R) &= \arg \min_{i=1,\ldots,N_D} \langle R, \phi(R)\rangle\, g(\Delta(D_i, R)) \\ &= \arg \min_{i=1,\ldots,N_D} g(\Delta(D_i, R)) \\ &= \arg \min_{i=1,\ldots,N_D} \Delta(D_i, R) \end{aligned}$$

Thus, we may replace the computation and minimization of N_D least squares errors $E(D_i, R)$ by the search for the nearest neighbor of $\phi(R) \in \mathbf{R}^n$ in the set of $2N_D$ vectors $\pm\phi(D_i) \in \mathbf{R}^n$.

For an interpretation we note that for given $D \in \mathbf{R}^n$ all vectors of the form $a\phi(D) + \sum_{k=1}^{p} b_k B_k$ can be exactly represented as a linear combination of D and $B_1, \ldots, B_p$ (with zero least squares error). These vectors form a $(p+1)$-dimensional subspace of $\mathbf{R}^n$, an orthonormal basis of which is given by $\phi(D)$ and $B_1, \ldots, B_p$. For all D in this space $\phi(D)$ is unique up to choice of the sign. Thus, $\phi(D)$ may serve as a 'representative' of this space and will become the multi-dimensional key for searching.

The problem of finding closest neighbors in Euclidean spaces has been thoroughly studied in computer science. For example, a method using kd-trees that runs in expected logarithmic time is presented in [14] together with pseudo code. After a preprocessing step to set up the required kd-tree, which takes $O(N \log N)$ steps, the search for the nearest neighbor of a query point can be completed in expected logarithmic time, $O(\log N)$. However, as the dimension d increases, the performance may suffer. A method that is more efficient in that respect, presented in [1], produces a so-called approximate nearest neighbor. For domain pools that are not large other methods, that are not based on space-partitioning trees, may perform better. For example, the modified equal average nearest neighbor search (ENNS) in [25] seems to be one of the best.

Before we turn to practical issues, we remark, that we can use the result of the Theorem 1 in order to identify all codebook blocks D_i that satisfy a given tolerance criterion $E(R, D_i) \leq \delta$. In other words, solving the equality for Δ in the expression for the error $E(D, R)$ in the theorem yields a necessary and sufficient condition for a codebook block D to fulfill the tolerance criterion.

Corollary 2 (A necessary and sufficient condition)
Let $\delta > 0$ and $n \geq 2$. Let R and D be in $\mathbf{R}^n \backslash \mathcal{B}$ with $\langle R, \phi(R)\rangle \geq \delta$. Then $E(D,R) = \min_{a,b_1,\ldots,b_p \in \mathbf{R}} \|R - (aD + \sum_{k=1}^p b_k B_k)\| \leq \delta$ if and only if

$$\Delta(D,R) \leq \sqrt{2 - 2\sqrt{1 - \frac{\delta^2}{\langle R, \phi(R)\rangle^2}}},$$

where $\Delta(D,R)$ is defined as in Theorem 1.

Proof. From Theorem 1, $E(D,R) = \langle R, \phi(R)\rangle\, g(\Delta(D,R))$ with $g(\Delta) = \Delta\sqrt{1 - \frac{\Delta^2}{4}}$. Thus, for $0 \leq \Delta \leq \sqrt{2}$ we have $E(D,R) \leq \delta$ if and only if $\Delta^4 - 4\Delta^2 + 4\delta^2/\langle R, \phi(R)\rangle^2 \geq 0$. From this the assertion easily follows.

The condition $\langle R, \phi(R)\rangle \geq \delta$ does not impose any restrictions. To see this, observe that in the case of $\langle R, \phi(R)\rangle < \delta$ we already have

$$E(D,R) = \langle R, \phi(R)\rangle \sqrt{1 - \langle \phi(D), \phi(R)\rangle^2} \leq \langle R, \phi(R)\rangle < \delta.$$

for *any* codebook block D. Thus, it suffices to encode R only using the fixed basis blocks, i.e., by $\sum_{k=1}^p b_k B_k$.

4 Practical considerations

We continue with some remarks on generalizations and implications of the theory from the last section.

In practice, there is a limit in terms of storage for the feature vectors of domains and ranges. For example, the keys for ranges of size of 8 by 8 pixels require 64 floating point numbers each. Thus, 32K domains from a domain pool would already fill 8 MB of memory on a typical workstation, while we would like to work with pools of a hundred thousand and more domains. To cope with this difficulty, we settle for a compromise and proceed as follows. We *down-filter* all ranges and domains to some prescribed dimension of moderate size, e.g., $d = 4 \times 4 = 16$. Moreover, each of the d components of a feature vector is *quantized* (8 bits/component suffice). This allows the processing of an increased number of domains and ranges, however, with the implication that the formula of the theorem is no longer exact but only approximate. This, however, is not a severe disadvantage as pointed out in the following remark and as demonstrated in the experiments later on.

The approach of pixel averaging in order to reduce the dimensionality of the domains and ranges (64 and higher is typical) to a more feasible number (here $d = 16$)

may be improved by better concentrating relevant subimage information in the d components. Based on our report [32] Barthel et al [5] have suggested and implemented an alternative reduction of dimension. They have used a two-dimensional discrete cosine transformation (DCT) of the projected codebook blocks $\pm\phi(D_i)$. The distance preserving property of the unitary transform carries over the result of our Theorem to the frequency domain and nearest neighbors of DCT coefficient vectors will yield the smallest least squares errors. In practise one computes the DCT for all domains and ranges. Then, from the resulting coefficients, the DC component is ignored and the next d coefficients are normalized and make up the feature vector.

Because of the downfiltering and the quantization of both the feature vectors and the coefficients $a, b_1, \ldots, b_p$, it can happen that the nearest neighbor in feature vector space is not the codebook block with the minimum least squares error using quantized coefficients. Moreover, it could yield a scaling factor a being too large to be allowed. To take that into consideration, we search the codebook not only for the nearest neighbor of the given query point but also for, say, the next 5 or 10 nearest neighbors (this can still be accomplished in logarithmic time using a priority queue). From this set of neighbors the non-admissible domains are discarded and the remaining domains are compared using the ordinary least squares approach. This also takes care of the problem from the previous remark, namely that the estimate by the theorem is only approximate. While the domain corresponding to the closest point found may not be the optimal one, there are usually near-optimum alternatives among the other candidates.

We make two technical remarks concerning *memory requirements* for the kd-tree. Firstly, it is not necessary to create the tree for the full set of $2N_D$ keys in the domain pool. We need to keep only one multi-dimensional key per domain, e.g., by keeping only the key which has a non-negative first component (multiply key by -1 if necessary). In this set-up a kd-tree of all $2N_D$ vectors has two symmetric main branches (separated by a coordinate hyperplane), thus, it suffices to store only one of them. Secondly, there is some freedom in the choice of the *geometric transformation* that maps a domain onto a range coming from the 8 possible rotations and reflections of a square subimage. This will create a total of 8 entries per domain in the kd-tree, enlarging the size of the tree. However, we can get away without this tree expansion. To see this, just note that we may instead consider the 8 transformations of the *range* and search the original tree for nearest neighbors of each one of them.

The *preprocessing time* to create the data structure for the multi-dimensional search is not a limitation of the method as demonstrated by our experiments.

5 Implementation

Our implementation is based on Fisher's adaptive quadtree algorithm [13]. The image region is subdivided into squares using a quadtree data structure. The leaf nodes of the tree correspond to the ranges while the domain pool for a given range consists of image subsquares of twice the size. The number of domains in the pool can be adjusted by parameters of the method. The quadtree construction is adaptive in the sense that for each node the corresponding domain pool is searched for a matching domain. That node then becomes a leaf node if the search yielded a match satisfying a given tolerance criterion. If the tolerance is not met, the square image region corresponding to the node is broken up into four subsquares which become the child nodes of the given node (see [12, 13] for details). Although the performance of this algorithm in terms of compression and fidelity is not the best possible it serves as a good test bed for our experiments in which we are aiming at evaluating the capabilities of the multi-dimensional nearest neighbor search in comparison to traditional complexity reduction attempts. In that respect Fisher's code is excellent because it contains an advanced classification method.

This classification works as follows. A square range or domain is subdivided into its four quadrants (upper left, upper right, lower left, and lower right). In the quadrants the average pixel intensities A_i and the corresponding variances V_i are computed ($i = 1, \ldots, 4$). It is easy to see that one can always orient (rotate and flip) the range or domain such that the average intensities are ordered in one of the three following canonical orientations:

$$\begin{aligned} &\text{major class 1: } A_1 \geq A_2 \geq A_3 \geq A_4, \\ &\text{major class 2: } A_1 \geq A_2 \geq A_4 \geq A_3, \\ &\text{major class 3: } A_1 \geq A_4 \geq A_2 \geq A_3. \end{aligned}$$

Once the orientation of the range or domain has been fixed accordingly, there are 24 different possible orderings of the variances which define 24 subclasses for each major class. If the scaling factor a is negative then the orderings in the classes must be modified accordingly. Thus, for a given range two subclasses need to be searched in order to accommodate positive and negative scaling factors.

Our implementation of the multi-dimensional nearest neighbor search is based on a code which was kindly provided by Arya and Mount [1]. The critical advances of this algorithm in comparison to the classical kd-tree approach of Friedman et al [14] are twofold. After a space partitioning tree (such as a kd-tree) has been set up for the data a priority list of nodes is maintained during nearest neighbor searching such that nodes corresponding to data volumes that are closest to the query point are searched first. Secondly, the requirement of finding the closest neighbor is relaxed to that of finding an $(1+\epsilon)$-approximate nearest neighbor. This is a data point whose distance to the query point is not larger that $1+\epsilon$ times the distance of the query point to its exact closest neighbor. Arya et al report that even with seemingly large parameters such as $\epsilon = 3$ in practise data points are found which have a 50% chance of being the true nearest neighbor and which are on the average only 1.05 times as far the closest neighbor. On the other hand a speed up of a factor of up to 50 over the exact ($\epsilon = 0$) case can be enjoyed. Using a second priority list we have modified the code so that a given number of approximate nearest neighbors can be returned.

We have joined the above two programs providing the following two options used for our test series:

1. The parameter ϵ and the number M of approximate neighbors requested in each search can be specified. Note that the actual number of neighbors returned for

a given range is twice as large since two searches must be carried out for one range (one for positive and one for negative scaling factors).

2. The dimension of the keys for the ranges and domains can be chosen. We have worked with dimension 4 and 16 corresponding to a downfiltering of an image block to a 2 by 2 or 4 by 4 image block. This fits well with the compression code since the domain and range sizes are of the form 2^k by 2^k, thus, the filtering can conveniently be done by pixel averaging.

We also implemented the two-dimensional discrete cosine transformation (DCT) of the projected codebook blocks $\pm\phi(D_i)$ and ranges $\phi(R)$ in order to search for nearest neighbors in the frequency domain as suggested by Barthel et al. In our study we vary the number of frequency components retained and compare the performance to the direct approach using pixel averaging.

6 Results

In our first series of experiments we make use of the classification and replace the linear search in a class by the approximate nearest neighbor search. Here we cannot yet expect the method to come up with better quality encodings than the plain classification method, since we are searching through the same pool of codebook blocks. Thus, the goal is to show that significant computation time can be saved while providing image fidelity and compression without or with only very minor degradation.

6.1 Choosing parameters for the approximate nearest neighbor search

Before the computations we need to decide which parameters of the fast search to use: $\epsilon \geq 0$, the number M of $(1+\epsilon)$-approximate nearest neighbors returned, and the dimension d of the key space. Figure 2 shows a study for the first two of these parameters. We consider encodings of a 512 by 512 test image (Lenna) with a fixed range size (4×4 pixels), i.e., the adaptive quadtree is trivial having the identical minimal and maximal depths. The classification is active, and only one out of 72 classes is searched using the fast search. The dimension of the key space is $d = 16$, while the parameters ϵ and M are varied. The times reported in this subsection are measured on an SGI Indigo2 running an R4000 processor. The result shows that a lower quality but faster search (high value of ϵ) is beneficial in terms of quality obtained for given computation time. We settle for $\epsilon = 3$ and $M = 5$ (35.21 dB, 16 secs). When comparing this to the performance using the linear search with classification (35.56 dB, 52 secs) we obtain in this case threefold speed in exchange for a 0.3 dB drop in peak-signal-to-noise ratio (PSNR).

We have studied the performance regarding the dimension of the key space using images of various sizes and dimensions $d = 4$ and 16. We find that with the smaller dimension $d = 4$ we gain a little processing time, however, we lose up to 2 dB in image fidelity without providing a gain in compression (see also figure 3). Thus, in the following we downfilter domains and ranges to 4 by 4 pixels.

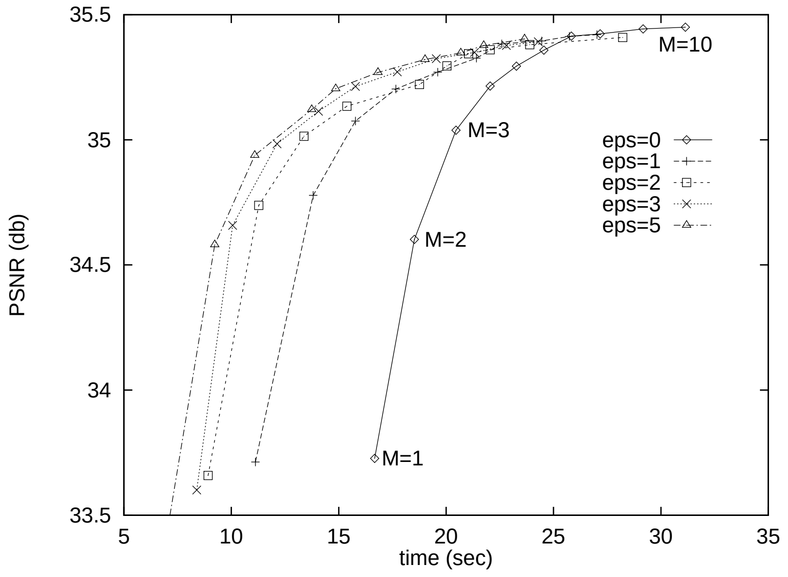

Figure 2: For the parameters $\epsilon = 0, 1, 2, 3, 5$ and with $M = 1, ..., 10$ neighbors returned per search we show the PSNR versus computation time. The maximal attainable PSNR is 35.56 dB (obtained in 52 secs with linear search using classification).

6.2 Fast nearest neighbor search with classification

We have used a few popular test images of differing resolutions: 256 by 256 Collie, 512 by 512 Baboon and Lenna, and 1024 by 1024 Composite consisting of the following four subimages of resolution 512 by 512: Kiel Harbor, Peppers, Baboon, Lenna. Also we vary the domain pool size by factors of 4. This way the scaling behavior of the pure classification method can be compared to that of the method using nearest neighbor search.

The parameters used for the adaptive quadtree method are the default parameters [13], except for the tolerance (set at the value 4) and the minimal block size in the range partition, which is set to 4×4 pixels in all cases. Quadrupling the domain pool size is obtained by halving the domain spacing in both image x- and y-directions. The data in table 1 summarize the findings.[3] The results are as follows:

- The compression ratios obtained with the fast search versus the linear search differ on the average by less than 0.03. Thus, for all practical purposes the fast search does not significantly worsen the compression ratios.
- As expected a slight loss in image quality is traded in for higher speed in the encoding with the fast search (see the last two columns for the loss in PSNR versus the speed up factor).
- It is interesting to note that although different images may require much different encoding times (compare Lenna with Baboon) the speed up factor scales

[3]The timings in the table are improved w.r.t. the data in [34], due to a technical improvement in our code. The times reported in this subsection and the following are measured on an SGI Indy running an R4600 processor.

		linear search			fast search				
image size	pool size	comp. ratio	PSNR dB	time h:m:s	comp. ratio	PSNR dB	time h:m:s	change dB	speed up
Collie 256	1	5.81	34.55	4	5.81	34.52	3	−0.03	1.3
	4	5.58	35.50	13	5.57	35.42	5	−0.08	2.6
	16	5.37	36.19	52	5.35	36.02	10	−0.17	5.2
Lenna 512	1	8.48	34.87	34	8.45	34.57	13	−0.30	2.6
	4	8.45	35.69	2:01	8.39	35.40	20	−0.29	6.1
	16	8.43	36.21	7:40	8.31	35.82	40	−0.39	11.5
Baboon 512	1	4.75	25.29	56	4.75	25.19	20	−0.10	2.8
	4	4.44	26.39	3:26	4.43	26.13	30	−0.26	6.9
	16	4.17	27.13	14:17	4.16	26.69	56	−0.44	15.3
Comp. 1024	1	6.34	30.89	14:02	6.31	30.55	1:31	−0.34	9.3
	4	6.11	31.75	54:01	6.05	31.32	2:15	−0.43	24.0
	16	5.89	32.43	3:22:46	5.80	31.77	4:16	−0.66	47.5

Table 1: Performance of the fast nearest neighbor search when implemented within the classification scheme of Fisher.

nicely with image size and domain pool size. When the domain pool size is quadrupled or the image size is doubled (which also results in a quadrupled domain pool size) the speed up factor roughly doubles. This is reflected in the computation times. While quadrupled domain pool size requires also fourfold computation time with the linear search, only 1.5 to 2 times as much time is needed with the fast search.

The slight loss in image quality can be further reduced by enlarging the number M of neighbors computed in each search or by reducing ϵ. For example, with $M = 10$ and $\epsilon = 3$ we obtained (for pool size 1 and tolerance 8) PSNR degradations of only 0.02, 0.10, 0.06, 0.20 dB with speed up factors of 0.9, 1.7, 1.6, 5.2 for the four images respectively.

6.3 Fast nearest neighbor search without loss in fidelity

It is also possible to accelerate the encoding of fractal image compression without having to pay the price of a slightly degraded image quality. The idea is to include more classes in the search tree instead of searching through only one class. In a first try we ignore the classification altogether and thus search the entire domain pool for each range. In fact, since all 8 orientations of all domains in all of the 72 subclasses belong to the domain pool, we estimate that about 576 times as many domains are considered in each search as compared to the method with classification (where one out of 72 classes is searched). Thus, the searching covers a lot more domains and may result in a better matching domain. Although this global attack of the problem is clearly overkill, it produces the desired qualities in some of the 12 cases corresponding

		fast search				
image size	pool size	comp. ratio	PSNR dB	time m:s	change dB	speed up
Collie 256	1	6.18	35.62	6	+1.10	0.7
	4	5.88	36.15	9	+0.73	1.4
	16	5.62	36.52	17	+0.50	3.1
Lenna 512	1	9.02	35.59	23	+1.02	1.5
	4	8.86	36.02	35	+0.62	3.5
	16	8.59	36.27	1:10	+0.47	6.6
Baboon 512	1	4.76	26.34	41	+1.15	1.4
	4	4.45	26.96	1:07	+0.83	3.1
	16	4.18	27.43	2:00	+0.75	7.1
Comp. 1024	1	6.48	31.41	3:04	+0.86	4.6
	4	6.17	31.89	4:49	+0.57	11.2
	16	5.89	32.20	8:56	+0.43	22.7

Table 2: Performance of the fast nearest neighbor search when implemented within the major classes of the classification scheme of Fisher.

to the table above. In all cases both image fidelity and compression ratio improve (by 0.88 dB PSNR and 0.30 ratio on the average). However, due to the vast search a significant speed up occurs only for the larger domain pools and images. A more sensible approach would be to enlarge the search by a smaller factor. For example, Fisher [13, page 69] notes that the 24 class search gives almost the same quality results as the 72 class search. Thus, we may expect similar improvements in PSNR and compression ratio but with notably reduced computation times when searching only through one of the major classes for each range.

The results are given in table 2 and as expected; instead of losing fidelity we gain up to over 1 dB in PSNR, while the execution times are still less than for the classification method. As compared to the fast search with classification the run times are only doubled although the number of codebook blocks considered in each search is increased by a factor of 24. For example, when we consider the image Lenna with the domain pool such that domains are unions of ranges (i.e., pool size 4), the method accelerates fractal image compression using the already fast classification scheme by a factor of 3.5 and in addition improves fidelity by 0.62 dB. In this case even the compression ratio is also improved by 0.41.

6.4 On the use of orthonormally transformed feature space

In the following experiment we investigate the alternative representation of the feature vectors $\pm\phi(D)$ in an orthonormally transformed space. We use the two-dimensional discrete cosine transformation for this purpose. As mentioned further above, from the DCT coefficients of D, the DC component is deleted and the next d coefficients are normalized and make up the feature vector. We fix the sizes of the ranges to 8×8

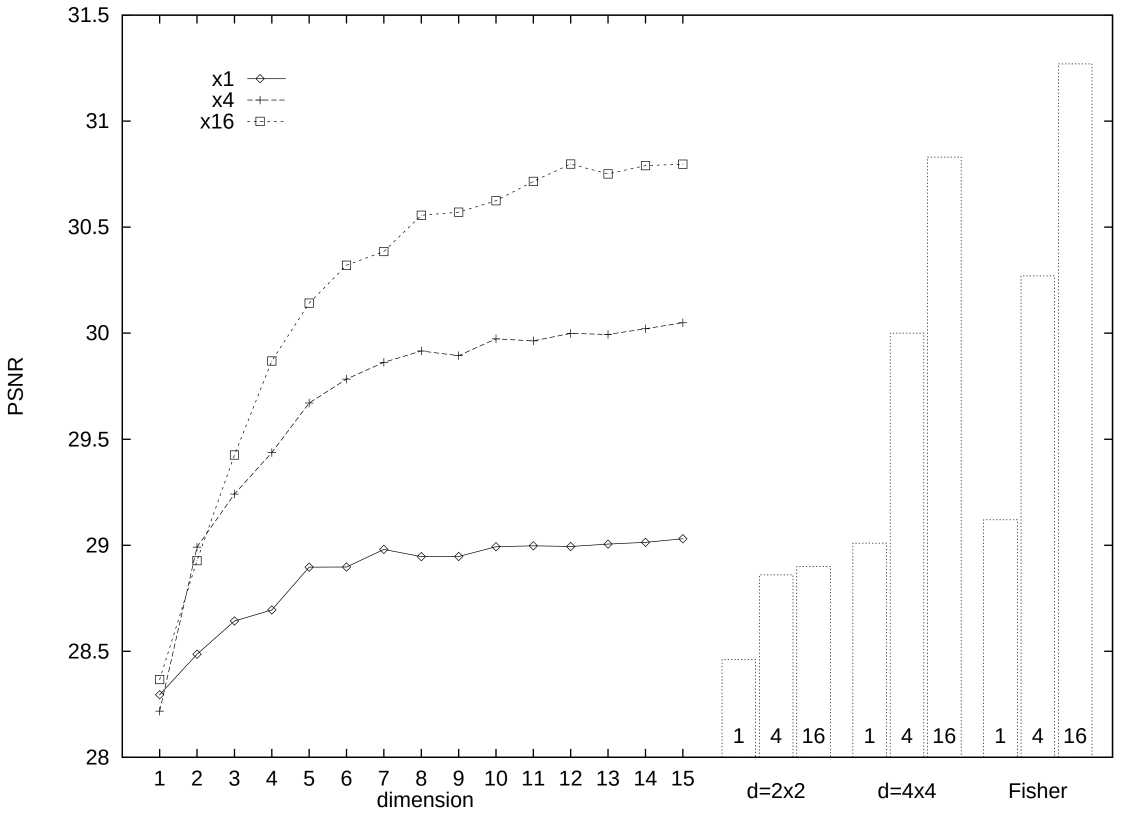

Figure 3: Using DCT coefficients for feature vectors. Results are shown for three domain pool sizes (1, 4, 16). On the left the fidelity in dB when using transformed feature vectors of dimension 1 to 15. The three groups of bars on the right correspond to the use of untransformed feature vectors (pixel averaging down to 2×2 or 4×4 pixels) and to the full search in each class as in the unaltered program of Fisher.

pixels and encode the 512×512 image Lenna several times using the classification as in section 6.2 and with the following variations:

- Straight search in the classes. This gives an upper bound for the quality of all encodings in this study.
- Using the fast nearest neighbor search with feature vectors computed from pixel averaging down to size 2×2, i.e., $d = 4$.
- Using the fast nearest neighbor search with feature vectors computed from pixel averaging down to size 4×4, i.e., $d = 16$.
- Using the fast nearest neighbor search with feature vectors computed from a variable number of DCT coefficients, i.e., $d = 1, 2, \ldots, 15$.

We record the corresponding PSNR and repeat the experiment with fourfold and sixteenfold expanded domain pools. Figure 3 displays the results. For an interpretation of the graph we note:

- We already know that the achievable quality with pixel averaging and $d = 2 \times 2$ is poor. However, when we take 4-dimensional feature vectors from the DCT coefficients, the results are much better (0.2, 0.6, 1.0 dB for domain pool size 1x, 4x, 16x). This confirms the idea of exploiting the energy compaction in the first components of the transform.
- The fidelity obtained with pixel averaging and $d = 4 \times 4$ can be achieved with about 10 DCT coefficients. Thus, we can reduce the dimension of the feature space by 6 without penalty. This is a useful insight as the nearest neighbor search is more efficient in lower dimensions. However, the time for the additional DCT must also be taken into account.[4]

range 1				range 2				range 3			
rms unquantized		distance unquantized		rms unquantized		distance quantized		rms unquantized		distance quantized	
rank	rms	rank	dist	rank	rms	rank	dist	rank	rms	rank	dist
1	1.467	1	0.333	1	0.881	1	7470	1	1.596	1	16567
2	1.533	2	0.349	2	1.008	2	9949	2	1.741	2	20450
3	1.625	3	0.370	3	1.128	5	13078	3	1.744	3	20719
4	1.646	4	0.375	4	1.128	4	13035	4	1.751	4	20926
5	1.684	5	0.384	5	1.132	3	13010	5	1.754	5	21020
6	1.687	6	0.385	6	1.171	6	14126	6	1.797	6	22352
7	1.716	7	0.392	7	1.185	7	14577	7	1.824	7	23184
8	1.720	8	0.393	8	1.188	8	14661	8	1.832	8	23556
9	1.724	9	0.394	9	1.201	9	14870	9	1.851	9	24265
10	1.734	10	0.396	10	1.205	10	15098	10	1.858	11	24416

Table 3: Comparison of ranking w.r.t. rms and distance of the top 10 domains, with unquantized (range 1) and quantized (ranges 2 and 3) feature vectors. The quantized distance is given in corresponding units of 1/16384.

6.5 Quantization issues

In the last experiment we study the effects of quantization. Quantization occurs at two places, namely for the feature vectors used in the nearest neighbor search, and then also for the coefficients $a, b_1, \ldots, b_p$. Thus, the nearest neighbor in feature vector space is not necessarily corresponding to the codebook block with the minimum least squares error using quantized coefficients.

We have compared all 4×4 range blocks R with all 8×8 domain blocks, which were first downfiltered codebook blocks D of size 4×4. For a given range the codebook blocks can be ordered in one of four ways, namely according to

[4]Wohlberg and de Jager [41] report that the DCT representation reveals an improvement in computation time by a factor of 1.5 to 2.0 but give only few details about the parameters of their experiment.

1. root mean square error, which (for 4×4 image blocks) is 0.25 times the minimum least squares error $E(R, D)$, without quantization of the coefficients

2. root mean square error, but with quantization of the coefficients (uniform quantization with 5 bits for the scaling coefficient a and 7 bits for the offset b_1, only the constant fixed basis block is used)

3. distance measure $\Delta(\phi(D), \phi(R))$, treating the components of the feature vectors as unquantized real numbers

4. distance measure $\Delta(\phi(D), \phi(R))$, but with quantization of the components of the feature vectors (uniform quantization with 8 bits/component)

When we compare the ordering of the codebook blocks with respect to criteria 1 and 3 we expect a one-to-one correspondence, because that is just the statement of the theorem in section 3. This is exactly what happens as the table 3 (first 4 columns) confirms.

To investigate the losses due to the 8 bit quantization of the feature vector components we similarly compare the rankings of rms versus distance based on quantized vectors (table 3, columns 5 to 12) for two typical ranges. The table shows that there seems to be almost no negative quantization effect. The rankings are much the same. Thus, our quantization to one byte per feature vector component is appropriate.

rms (quantized)		1	2	3	4	5	6	7	8	9	10
distance	range 4	2	7	9	11	6	10	18	22	8	1
(unquantized)	range 5	27	9	48	39	46	63	56	66	84	41
	range 6	1	3	2	9	6	8	11	14	4	22

Table 4: Comparison of ranking w.r.t. rms after quantization of the coefficients and distance (unquantized) of the top 10 domains.

In the last study in this series we examine the quantization effect that occurs with the finite resolution of the collage coefficients. In table 4 we display the rankings for three ranges using the (unquantized) distance in feature space relative to that given by the rms after coefficient quantization. Here we see that the quantization of the coefficients does have a significant effect. In some cases (range 5) we lose the best domain when we restrict the search to the ten best ones according to distance. However, most cases are similar to range 6, which is not quite so bad. This shows, that one cannot expect to achieve an overall lossless result when applying the nearest neighbor search in feature space with the restriction on the best 10 codebook blocks (compare also figure 2).

7 Other work related to feature vectors

In this section we briefly discuss other work as it relates to our feature vector approach. The material can be grouped into three parts: the straight usage of the projected and normalized codebook blocks as in this paper, the attempt of using invariant moments of ranges and domains, and tree structured methods.

7.1 Straight feature vectors

A forerunner of feature vectors as used in this paper has been presented by Hürtgen and Stiller [19]. As in the classification of Fisher, Jacobs, and Boss an image block is partitioned into its four quadrants and their mean intensities are computed. Then a vector consisting of four bits is constructed as follows: the i-th bit is 1 if the mean of the i-th quadrant is above the overall mean, and 0 otherwise. Thus, in the terminology of this paper, this is our feature vector after downsampling to size $d = 2 \times 2$ and quantizing to 1 bit per component. Due to these strict limitations a nearest neighbor search is not practical, rather, these vectors serve as a means for classification into 16 classes. Then a range is compared only with codebook blocks from the same class. We remark, that in order to accommodate negative scaling factors a a different class ought to be searched, a fact, that had been overlooked in [19].

Kominek in [23, 24] follows the straight feature vector approach with the difference that another type of data structure for the multi-dimensional feature vectors (r-trees) is adopted. However, the method previously presented in [32, 33] is not completely realized: the necessary pairing of positive/negative feature vectors to support negative scaling factors is ignored (as in [19]) and the search for a nearest neighbor in the r-tree is unnecessarily suboptimal (only a single bucket is inspected).

7.2 Invariant moments

In [29], Novak assigns a 4-dimensional feature vector to each block. The components of the feature vector are certain moment invariants defined from the grey level distribution within the block. A useful property of these moment invariants is that they are invariant with respect to the geometric transformation, i.e., one feature vector suffices for each domain. The isometric versions of that domain block then have the same moment vector. However, the moments are not invariant w.r.t. the affine transformation regarding the luminance. To cope with this problem Novak proposed a normalization procedure. There are several problems with this approach: Again, the negative intensity blocks are omitted from consideration causing the loss of some of the possible fidelity. The values of the invariant moments range over several orders of magnitude, thus, a logarithmic rescaling becomes necessary before nearest neighbor search becomes feasible. And then, most importantly, the method is intuitive in the sense that no supporting theory is given to the goal that closeness in the feature space ensures good approximations in the least squares sense. The fact is, that such a theory cannot exist. Novak worked with triangular partitioning, and Frigaard continued the work in [15] using a quadtree partitioning. However, Frigaard does not normalize feature vectors with respect to mean and variance in order to make the moments invariant relative to the affine luminance transformation. He reports that normalizing would in fact degrade the overall quality of an encoding of an image, which apparently documents the weakness of the method.

Götting, Ibenthal, and Grigat [16] and Popescu and Yan [31] also pursue complexity reduction using invariant moments of different types.

7.3 Tree structured methods

Besides the dimensional reduction and the variance based classification mentioned above Caso, Obrador and Kuo propose a tree structured search in [9]. The pool of codebook blocks is recursively organized in a binary tree. Initially two (parent) blocks are chosen randomly from the pool. Then all blocks are sorted into one of two bins depending on by which of the two parent blocks the given block can be covered best in the least squares sense. This results in a partitioning of the entire pool into two subsets. The procedure is recursively repeated for each one of them until a prescribed bucket size is reached. Given a range one can then compare this block with the blocks at the nodes of the binary tree until a bucket is encountered at which point all of the codebook blocks in it are checked. This does not necessarily yield the globally best match. However, the best one (or a good approximate solution) can be obtained by extending the search to some nearby buckets. A numerical test based on the angle criterion is given for that purpose. The procedure is related to the nearest neighbor approach since the least squares criterion (minimize $E(D,R)$) is equivalent to the distance criterion (minimize $\Delta(\phi(D),\phi(R))$. Thus, the underlying binary tree can be considered to be randomized version of the kd-tree structure we have used here.

Van der Walle [40] worked on a wavelet representation of fractal image compression, where similarly to ordinary fractal image compression, range vectors (corresponding to subtrees of the tree of wavelet coefficients) have to be matched with domain vectors (also corresponding to nodes of the wavelet tree), which may be scaled by an arbitrary scaling factor. For each node a feature vector is generated based on angles between the coefficient vectors and axes in the wavelet coefficient space. These vectors are then sorted into a multi-dimensional space-partitioning data structure within which the fast search is organized. In terms of distances of feature vectors $\pm\phi(D)$ the interpretation is as follows: We define a small set of anchor points in feature space (e.g., at the positions of the main principal components of the set of all feature vectors). For each (projected and normalized) codebook block as well as for each range block we compute the distances Δ to the anchor points. Then a point in feature space that is close to a given range feature vector must necessarily have distances to the anchor points that are near those of the range. To facilitate the search for such codebook blocks, the blocks can be organized in a tree structure. The core of this method for finding nearest neighbors is reminiscent of the annulus testing known in vector quantization [18].

Of all the methods discussed in this section only the tree structured ones correctly consider positive and negative feature vector corresponding to positive and negative scaling coefficients in the collage.

8 Conclusion

We have reviewed the theory which links the domain-range comparison fundamental to fractal image compression with nearest neighbor search. This result leads to a new technique at the core of the encoding process. It consists of the fast approximate nearest neighbor search in a multi-dimensional feature space and can easily be integrated into existing implementations of classification methods. The approach reduces the time complexity of the encoding step thereby creating faster fractal image compression. The speed up can be adjusted so that it comes with only minor degradation in image quality and compression ratio or with improvements in both fidelity and compression.

Acknowledgements. The author thanks Klaus Bayer, Kai Uwe Barthel, Amitava Datta, Raouf Hamzaoui, Thomas Ottmann, and Sven Schuierer for fruitful discussions, Sunil Arya and Dave Mount for the fast nearest neighbor search code and Yuval Fisher for the adaptive quadtree code. Matthias Ruhl expertly adapted and merged the two programs for our studies.

References

[1] Arya, S., Mount, D. M., Netanyahu, N. S., Silverman, R., Wu, A., *An optimal algorithm for approximate nearest neighbor searching,* Proc. 5th Annual ACM-SIAM Symposium on Discrete Algorithms (1994) 573–582.

[2] Bani-Eqbal, B., *Speeding up fractal image compression,* in: *Proceedings from IS&T/SPIE 1995 Symposium on Electronic Imaging: Science & Technology,* Vol. 2418: Still-Image Compression, 1995.

[3] Bani-Eqbal, B., *Combining tree and feature classification in fractal encoding of images,* submitted for publication, 1995.

[4] Barnsley, M., Hurd, L., *Fractal Image Compression*, AK Peters, Wellesley, 1993.

[5] Barthel, K. U., Schüttemeyer, J., Voyé, T., Noll, P., *A new image coding technique unifying fractal and transform coding*, IEEE Conf. on Image Processing, Texas (1994) 112–116.

[6] Bedford, T. J., Dekking, F. M., Keane, M. S., *Fractal image coding techniques and contraction operators,* Nieuw Arch. Wisk. (4), 10, no. 3 (1992) 185–218.

[7] Bogdan, A, Meadows, H. E., *Kohonen neural network for image coding based on iteration transformation theory,* SPIE, Vol. 1766-39, 1992.

[8] Boss, R. D., Jacobs, E. W., *Archetype classification in an iterated transformation image compression algorithm,* in [13], p. 79–90.

[9] Caso, G., Obrador, P., Kuo, C.-C. J., *Fast methods for fractal image encoding,* Proc. SPIE Visual Communication and Image Processing '95, Vol. 2501 (1995) 583–594.

[10] Dekking, M., *An inequality for pairs of martingales and its application to fractal image coding,* Technical Report 95-10 of the Faculty of Technical Mathematics and Informatics, Delft University of Technology, 1995, also to appear in the Journal of Applied Probability, 1996.

[11] Dekking, M., *Fractal image coding: some mathematical remarks on its limits and its prospects,* in: Conf. Proc. NATO ASI *Fractal Image Encoding and Analysis,* Trondheim, July 1995, Y. Fisher (ed.), to appear in Springer-Verlag, New York, 1995.

[12] Fisher, Y., *A discussion of fractal image compression,* in: H.-O. Peitgen, H. Jürgens, D. Saupe, *Chaos and Fractals,* Springer-Verlag, New York, 1992.

[13] Fisher, Y., *Fractal Encoding — Theory and Applications to Digital Images,* Springer-Verlag, New York, 1994.

[14] Friedman, J. H., Bentley, J. L., Finkel, R. A., *An algorithm for finding best matches in logarithmic expected time,* ACM Trans. Math. Software 3,3 (1977) 209–226.

[15] Frigaard, C., *Fast fractal 2D/3D image compression,* Manuscript, Institute for Electronic Systems, Aalborg University, 1995.

[16] Götting, D., Ibenthal, A., Grigat, R., *Fractal image coding and magnification using invariant features,* NATO ASI Conf. *Fractal Image Encoding and Analysis,* Trondheim, July 1995, to appear in a special issue of Fractals, 1995.

[17] Hamzaoui, R., *Codebook clustering by self-organizing maps for fractal image compression,* NATO ASI Conf. *Fractal Image Encoding and Analysis,* Trondheim, July 1995, to appear in a special issue of Fractals, 1996.

[18] Huang, C. M., Bi, Q., Stiles, G. S., Harris, R. W., *Fast full search equivalent encoding algorithms for image compression using vector quantization,* IEEE Trans. Image Processing 1,3 (1992) 413–416.

[19] Hürtgen, B., Stiller, C., *Fast hierarchical codebook search for fractal coding of still images,* EOS/SPIE Visual Communications and PACS for Medical Applications '93, Berlin (1993) 397–408.

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

[21] Jacquin, A., *Image coding based on a fractal theory of iterated contractive Markov operators, Part II: Construction of fractal codes for digital images,* Report Math. 91389-017, Georgia Institute of Technology, 1989.

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

[23] Kominek, J., *Algorithm for fast fractal image compression,* Proceedings from IS&T/SPIE 1995 Symposium on Electronic Imaging: Science & Technology Vol. 2419 Digital Video Compression: Algorithms and Technologies (1995) 296–305.

[24] Kominek, J., *Advances in fractal compression in multimedia applications,* submitted to Multimedia Systems Journal.

[25] Lee, C.-H., Chen, L. H, *Fast closest codeword search algorithm for vector quantization,* IEE Proc.-Vis. Image Signal Process. 141, 3 (1994) 143–148.

[26] Lepsøy, S., Øien, G. E., *Fast attractor image encoding by adaptive codebook clustering,* in [13], p. 177–198.

[27] Lin, H., Venetsanopoulos, A. N., *A pyramid algorithm for fast fractal image compression,* Proceedings of 1995 IEEE Intern. Conf. on Image Processing (ICIP), Washington, to appear Oct. 1995.

[28] Monro, D. M., Dudbridge, F., *Fractal approximation of image blocks,* Proc. ICASSP 3 (1992) 485–488.

[29] Novak, M., *Attractor coding of images,* Licentiate Dissertation, Dept. of Electrical Engineering, Linköping University, May 1993.

[30] Øien, G. E., *L_2 Optimal Attractor Image Coding with Fast Decoder Convergence*, PhD Thesis, The Norwegian Institute of Technology, Trondheim, Norway, April 1993.

[31] Popescu, D. C., Yan, H., *MR image compression using iterated function systems,* Magnetic Resonance Imaging 11 (1993) 727–732.

[32] Saupe, D., *Breaking the time-complexity of fractal image compression*, Technical Report 53, May 1994, Institut für Informatik, Universität Freiburg.

[33] Saupe, D., *From classification to multi-dimensional keys*, in [13], p. 302–305.

[34] Saupe, D., *Accelerating fractal image compression by multi-dimensional nearest neighbor search,* in: Proceedings Data Compression Conference, March 28–30, 1995 • Snowbird, Utah, J. A. Storer, M. Cohn (eds.), IEEE Computer Society Press, 1995.

[35] Saupe, D., *Lean domain pools for fractal image compression,* Proceedings IS&T/SPIE 1996 Symposium on Electronic Imaging: Science & Technology – Still Image Compression II, Vol. 2669, Jan. 1996.

[36] Saupe, D., Hamzaoui, R., *A review of the fractal image compression literature,* ACM Computer Graphics 28,4 (Nov. 1994) 268–276.

[37] Saupe, D., Hamzaoui, R., *Complexity reduction methods for fractal image compression,* in: I.M.A. Conf. Proc. on *Image Processing; Mathematical Methods and Applications,* Sept. 1994, J. M. Blackledge (ed.), to appear with Oxford University Press, 1995.

[38] Saupe, D., Hamzaoui, R., *A bibliography for fractal image compression,* in: Conf. Proc. NATO ASI *Fractal Image Encoding and Analysis,* Trondheim, July 1995, Y. Fisher (ed.), to appear in Springer-Verlag, New York, 1996.

[39] Signes, J., *Geometrical interpretation of fractal image coding,* NATO ASI Conf. *Fractal Image Encoding and Analysis,* Trondheim, July 1995, to appear in a special issue of Fractals, 1995.

[40] van der Walle, A., *Merging fractal image compression and wavelet transform methods,* NATO ASI Conf. *Fractal Image Encoding and Analysis,* Trondheim, July 1995, to appear in a special issue of Fractals, 1996.

[41] Wohlberg, B. E., de Jager, G., *Fast image domain fractal compression by DCT domain block matching,* Electronic Letters 31 (1995) 869–870.