# Accelerating Fractal Image Compression by Multi-Dimensional Nearest Neighbor Search

Dietmar Saupe[1]
Universität Freiburg

**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 a best match for another image portion. Our theory developed here shows that this basic procedure of fractal image compression is equivalent to multi-dimensional nearest neighbor search. 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 from 1.3 up to 11.5 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 increased the computation time. In this way both image quality and compression ratio can be improved at reduced computation time.

## 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.

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

[1]Address: Institut für Informatik, Universität Freiburg, Am Flughafen 17, 79110 Freiburg, Germany. Email: saupe@informatik.uni-freiburg.de. This paper appears in *Proceedings DCC'95 Data Compression Conference,* J. A. Storer, M. Cohn (eds.), IEEE Computer Society Press, March 1995.

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, [2, 4, 5, 10]. For a bibliographic survey of the field of fractal image compression see our paper [13].

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. This paper introduces a new twist for the encoding process and demonstrates its efficiency by a series of empirical studies. What is the current state-of-the-art? 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 contribution of this paper 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 searching. 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.

The remainder of this paper is organized as follows. In the following section we present the definition of the multi-dimensional keys, a theorem that establishes the mathematical foundation (see also [14, 15]), and some practical comments on the implications. Section 3 discusses our implementation of the proposed method and in section 4 we describe our empirical experiments and their results.

## 2 From Classification to Multi-Dimensional Keys

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. In previous schemes differing numbers of classes were used in this way (from 3 or 4 in [9] up to 72 in [5, 6]). Besides classification other complexity reduction methods are: adaptive codebook clustering, archetype classification, Rademacher functions, adaptive search strategies (see [16] for a survey and references).

We consider a point $z \in \mathbf{R}^d$ representing $d$ pixel values in a given range. Let us assume that the domains from the domain pool have been properly filtered and subsampled, thus, yielding a set of $N$ vectors $x^{(1)}, \ldots, x^{(N)} \in \mathbf{R}^d$ (often called codebook blocks). We let $E(x^{(i)}, z)$ denote the smallest possible least squares error of an approximation of the range data $z$ by an affine transformation of the domain data $x^{(i)}$. In terms of a formula, this is $E(x^{(i)}, z) = \min_{a,b \in \mathbf{R}} ||z-(ae+bx^{(i)})||^2$, where $e = \frac{1}{\sqrt{d}}(1, \ldots, 1) \in \mathbf{R}^d$ is just a unit length vector with equal components. Computing the optimal $a$, $b$ and the error $E(x^{(i)}, z)$ is a costly procedure, which we have to perform for all of the domain vectors $x^{(1)}, \ldots, x^{(N)}$ in order to arrive at the minimum error, given by $\min_{1 \le i \le N} E(x^{(i)}, z)$. This staggered minimization problem needs to be solved for many query points $z$ in the encoding process (i.e., for all ranges). We use the notation $d(\cdot, \cdot)$ for the Euclidean distance and $\langle \cdot, \cdot \rangle$ for the inner product in $\mathbf{R}^d$, thus, $||x|| = d(x, 0) = \sqrt{\langle x, x \rangle}$.

**Theorem.** *Let $d \ge 2$, $e = \frac{1}{\sqrt{d}}(1, \ldots, 1) \in \mathbf{R}^d$ and $X = \mathbf{R}^d \backslash \{re \mid r \in \mathbf{R}\}$. Define the normalized projection operator $\phi : X \to X$ and the function $D : X \times X \to [0, \sqrt{2}]$ by*

$$\phi(x) = \frac{x - \langle x, e \rangle\, e}{||x - \langle x, e \rangle\, e||} \quad \textit{and} \quad D(x, z) = \min\left(d(\phi(x), \phi(z)), d(-\phi(x), \phi(z))\right).$$

*For $x, z \in X$ the least squares error $E(x, z) = \min_{a,b \in \mathbf{R}} ||z - (ae + bx)||^2$ is given by*

$$E(x, z) = \langle z, \phi(z) \rangle^2\, g(D(x, z)) \quad \textit{where} \quad g(D) = D^2(1 - D^2/4).$$

**Proof.** The least squares approximation of $z$ by a vector of the form $ae+bx$ (resp. $ae+b\phi(x)$) is given by the projection

$$\text{Proj}(z) = \langle z, e \rangle\, e + \langle z, \phi(x) \rangle\, \phi(x) = \langle z, e \rangle\, e + \langle z, \phi(z) \rangle\, \langle \phi(x), \phi(z) \rangle\, \phi(x),$$

where the last equation is derived from $z = \langle z, e \rangle\, e + \langle z, \phi(z) \rangle\, \phi(z)$ (see figure 1). The least squares error in this approximation is calculated as

$$E(x, z) = ||z - \text{Proj}(z)||^2 = \langle z, \phi(z) \rangle^2\, (1 - \langle \phi(x), \phi(z) \rangle^2).$$

Since $d(\pm\phi(x), \phi(z)) = \sqrt{2(1 \mp \langle \phi(x), \phi(z) \rangle)}$ we have $D(x, z) = \sqrt{2(1 - |\langle \phi(x), \phi(z) \rangle|)}$. Solving for $|\langle \phi(x), \phi(z) \rangle|$ and inserting the square of the result in the formula for $E(x, z)$ completes the proof.

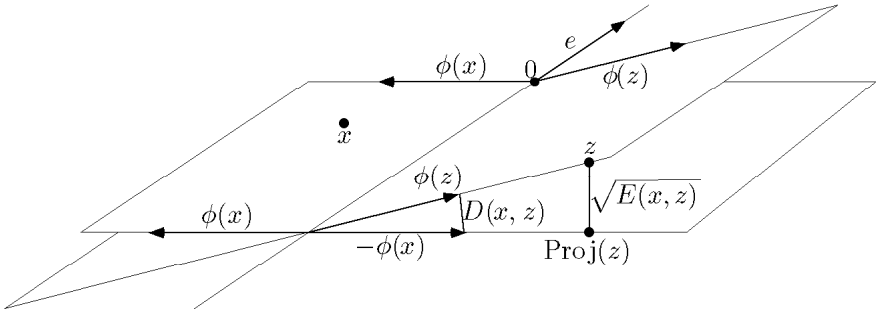


Figure 1: Illustration of the geometry underlying the theorem.

For an interpretation we note that for given $x \in \mathbf{R}^d$ all vectors of the form $ae + b\phi(x)$ can be exactly represented as a linear combination of $x$ and $e$ (with zero least squares error). These vectors form a two-dimensional subspace of $\mathbf{R}^d$, an orthonormal basis of which is given by $\phi(x)$ and $e$. For all $x$ in this space $\phi(x)$ is unique up to choice of the sign. Thus, $\phi(x)$ may serve as a 'representative' of this space and will become the multi-dimensional key for searching.

The theorem states that the least squares error $E(x, z)$ is proportional to the simple function $g$ of the Euclidean distance $D$ between the projections $\phi(x)$ and $\phi(z)$ (or $-\phi(x)$ and $\phi(z)$). 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 $g(D)$ is a monotonically increasing function for $0 \leq D \leq \sqrt{2}$ we conclude that *the minimization of the least squares errors* $E(x^{(i)}, z)$ *for* $i = 1, \ldots, N$ *is equivalent to the minimization of the distance expressions* $D(x^{(i)}, z)$*!* Thus, we may replace the computation and minimization of $N$ least squares errors $E(x^{(i)}, z)$ by the search for the nearest neighbor of $\phi(z) \in \mathbf{R}^d$ in the set of $2N$ vectors $\pm\phi(x^{(i)}) \in \mathbf{R}^d$.

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 [7] 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.

We conclude this section with some remarks on generalizations and implications.

1. 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 can be *quantized* (8 bits/number 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.

2. For a given range not all domains from the pool are *admissible*. There are restrictions on the resulting number $b$, the contraction factor of the affine transformation (e.g., $|b| < 1$ in some implementations). This is necessary in order to ensure convergence of the iteration in the image decoding. Also one imposes bounds on the size of the domain (for a given range a suitable domain should generally be larger than the range, but not too large). To take that into consideration, we search the domain pool 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 solved 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 (see section 4).
3. We make two remarks concerning *memory requirements* for the kd-tree. Firstly, it is not necessary to create the tree for the full set of $2N$ keys in the domain pool. The full kd-tree of all $2N$ vectors has two symmetric main branches (separated by a coordinate hyperplane). Thus, it suffices to store only one of them, which then needs to be searched twice. Secondly, there is some freedom in the choice of the *geometric transformation* that maps a domain onto a range. A square domain, e.g., may undergo any of the 8 transformations of its dihedral group (rotations by multiples of 90 degrees and flips). 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.
4. 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 (see section 4).
5. The method is described here for the common case of fractal image compression in which range data is approximated by a scaled copy of domain data plus a constant ($z \approx ae + bx^{(i)}$). There are generalizations of this approach allowing so called higher order fixed basis blocks. In this case a range block is approximated by a linear combination of domain data ($x^{(i)}$), the constant block ($e$) and discretized quadratic and possibly cubic blocks (see, e.g., [8, 11, 12]). Using a proper orthogonalization procedure it is possible to first deal with all fixed basis blocks and then to treat the remaining domain block similar to the common case discussed above. See [16] for details.

# 3 Implementation

Our implementation is based on Fisher's adaptive quadtree algorithm [5]. 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 sub-

squares which become the child nodes of the given node (see [4, 5] 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 ways:

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

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 scale factor $b$ 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 scale 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 [7] 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 $\epsilon$ and the number 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 scale 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 a range or domain 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.

# 4 Results

In this 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 domains. 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.

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, 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 $\epsilon$ and $M$ are varied. All times reported in this section are measured on an SGI Indigo2 running an R4000 processor. The result shows that a lower quality but faster search (high value of $\epsilon$) is beneficial in terms of quality obtained for given computation time. We settle for $\epsilon = 3$ and $M = 5$ (35.21 dB, 15.8 secs). When comparing this to the performance using the linear search with classification (35.56 dB, 51.7 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 only about 10–20% processing time, however, we loose up to 2 dB in image fidelity without providing a gain in compression. Thus, in the following we downfilter domains and ranges to 4 by 4 pixels.

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

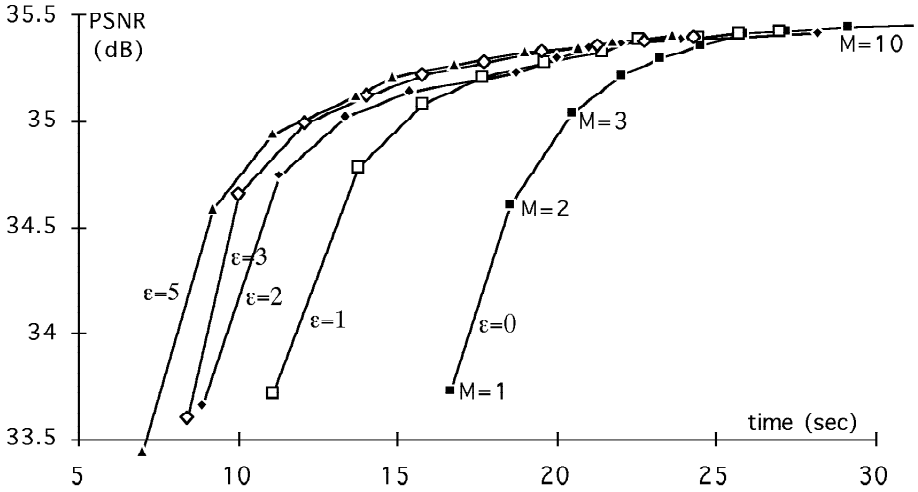


Figure 2: For the parameters $\epsilon = 0, 1, 2, 3, 5$ and with $M = 1, ..., 10$ neighbors returned per search we show the PSNR versus computation time. The maximal attainable PSNR is 35.56 dB (obtained in 51.7 secs with full linear search).

| | | linear search | | | fast search | | | | |
|---|---|---|---|---|---|---|---|---|---|
| image size | pool size | comp. ratio | PSNR dB | time h:m:s | comp. ratio | PSNR dB | time h:m:s | drop dB | speed up |
| Collie | 1 | 5.81 | 34.55 | 4 | 5.81 | 34.52 | 3 | 0.03 | 1.3 |
| 256 | 4 | 5.58 | 35.50 | 13 | 5.57 | 34.42 | 5 | 0.08 | 2.6 |
| | 16 | 5.37 | 36.19 | 52 | 5.35 | 36.02 | 11 | 0.17 | 4.7 |
| Lenna | 1 | 8.48 | 34.87 | 34 | 8.45 | 34.57 | 13 | 0.30 | 2.6 |
| 512 | 4 | 8.45 | 35.69 | 2:01 | 8.39 | 35.40 | 24 | 0.29 | 5.0 |
| | 16 | 8.43 | 36.21 | 7:40 | 8.31 | 35.82 | 57 | 0.39 | 8.1 |
| Baboon | 1 | 4.75 | 25.29 | 56 | 4.75 | 25.19 | 21 | 0.10 | 2.7 |
| 512 | 4 | 4.44 | 26.39 | 3:26 | 4.43 | 26.13 | 37 | 0.26 | 5.6 |
| | 16 | 4.17 | 27.13 | 14:17 | 4.16 | 26.69 | 1:30 | 0.44 | 9.5 |
| Comp. | 1 | 6.34 | 30.89 | 14:02 | 6.31 | 30.55 | 1:51 | 0.34 | 7.6 |
| 1024 | 4 | 6.11 | 31.75 | 54:01 | 6.05 | 31.32 | 4:03 | 0.43 | 13.3 |
| | 16 | 5.89 | 32.43 | 3:22:46 | 5.80 | 31.77 | 17:39 | 0.66 | 11.5 |

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 [5], except for the tolerance (set at the value 4). Quadrupling the domain pool size is obtained by halving the domain spacing in both image $x$- and $y$-directions. The data of the above table summarize the findings. 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 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 (except for the last line in the table). This is reflected in the computation times. While quadrupled domain pool size requires also fourfold computation time with the linear search, only 2 to 3 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 $\epsilon$. 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, 4.3 for the four images respectively.

It is also possible to accelerate the encoding of fractal image compression without having to pay the price of a slightly worsened 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 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 speed up occurs only for the larger domain pools and images. For example, we get acceleration factors of 1.3 and 3.8 for the Lenna and Composite images at domain pool size 16. A more sensible approach would be to enlarge the search by a smaller factor. For example, Fisher [5, 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.

Our 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 [14] Barthel et al [3] have suggested and implemented an alternative reduction of dimension. They have used a two-dimensional discrete cosine transformation (DCT) of the projected codebook blocks $\pm\phi(D_i)$. Properties of the DCT transform carry 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 zero AC component is ignored and the next $d$ coefficients make up the feature vector.

# 5 Summary

We have introduced a new theory linking the domain-range comparison fundamental to fractal image compression to 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 key 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, 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] Barnsley, M., Hurd, L., *Fractal Image Compression*, AK Peters, Wellesley, 1993.

[3] 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.

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

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

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

[7] 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.

[8] Gharavi-Alkhansari, M., Huang, T., *A fractal-based image block-coding algorithm,* Proc. ICASSP 5 (1993) 345–348.

[9] 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.

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

[11] Monro, D. M., Woolley, S. J., *Fractal image compression without searching,* Proc. ICASSP, 1994.

[12] Øien, G. E., Lepsøy, S., Ramstad, T., *An inner product space approach to image coding by contractive transformations,* Proc. ICASSP (1991) 2773–2776.

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

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

[15] Saupe, D., *From classification to multi-dimensional keys*, in [5], p. 302–305.

[16] 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.