Faster Fractal Image Coding Using Similarity Search in a KL-transformed Feature Space

Jean Cardinal*

Brussels Free University
Computer Science Department
Campus de la Plaine
Bld. du Triomphe CP212
B-1050 Brussels - BELGIUM
Phone : 00-32-2-6505601
Fax : 00-32-2-6505867
jcardin@ulb.ac.be

Abstract. Fractal coding is an efficient method of image compression but has a major drawback: the very slow compression phase, due to a time-consuming similarity search between image blocks. A general acceleration method based on feature vectors is described, of which we can find many instances in the literature. This general method is then optimized using the well-known Karhunen-Loeve expansion, allowing optimal – in a sense to be defined – dimensionality reduction of the search space. Finally, a simple and fast search algorithm is designed, based on orthogonal range searching and avoiding the "curse of dimensionality" problem of classical best match search methods.

1 Introduction

1.1 Fractal Image Compression

The fractal compression method has been implemented for the first time in 1989 ([12]) and extensively described since then in many different publications (e.g. [9]). In fractal image compression, we try to find a mapping W in the image space so that the fixed point of this mapping exists, is unique, and is as close as possible to the image I we want to encode. W is itself composed of m different block-wise mappings $w_i, i = 1, \ldots, m$. Each mapping w_i maps a so-called *domain block* onto a smaller *range block*. The set R of range blocks forms a partition of the image I, and each range block is uniquely coded by a transformation w_i. There is no restriction concerning the set of domains, but these are usually taken among a set of potential domains called the *domain pool*, and are usually bigger than the ranges. The mappings w_i consist in two parts: a spatial part, translating from the domain position in the image to the range position and scaling the domain size down to the range size, and a massic

* This work is supported by a PhD. scholarship from the National Scientific Research Fund (FNRS)

part, modifying the pixels in the block. The massic part is itself composed of two transformations: a scaling operation, multiplying all the pixel values in the block by a coefficient s_i, and an offset operation, adding a constant coefficient o_i to each pixel value. The encoding process consists in finding for each range the best domain in the domain pool: the one that gives the least mean square error (MSE) when modified by the scaling and offset operations. Of course, the coefficients s_i and o_i are computed so that they minimize the error, using a simple regression formula. The decoding simply consists in iterating the mapping W from any initial image. Having $|s_i| < 1, \forall i = 1, \ldots, m$ ensures the convergence during the decoding (we can however show that a weaker condition such as $|s_i| < 1.2$ is sufficient, see e.g. [9]).

The original algorithm compares each range with every domain. This method is very slow, and the compression of a single image may take hours. The simplest improvement is the use of classification schemes: dividing the domain pool in a number of classes, and comparing only the domains falling in the same class as the range. Classification schemes are numerous (e.g. [12], [9]), but none of them decreases the complexity order of the search. There are anyway some methods that achieve this goal, often with a slight quality loss (see e.g. [1])

The structure of the remainder of this paper is as follows. In section 1.2, we review the idea of feature vectors, and describe some previous work in which this idea is exploited. In sections 1.3 and 1.4, we briefly summarize the properties of the Karhunen-Loeve transform, and show how they can be used in nearest-neighbor searching problems. Section 2 describes the main contribution of this paper: a fast general acceleration technique. In section 2.1, we show how an useful representation of image blocks may be designed using the Karhunen-Loeve transform, and an appropriate search algorithm is described in 2.2. Finally experimental results are presented and commented in section 2.4.

1.2 A Fast General Method Using Feature Vectors

A good method to solve the problem of finding the best domain block for each range has been described in several papers ([17],[22],[13]), with different formulations and features. The general scheme is as follows:

1. find an operator mapping image blocks to *feature vectors*, so that minimizing the distance (e.g. the euclidean distance) between a feature vector corresponding to a range and a feature vector corresponding to a domain is equivalent to minimizing the collage error obtained when choosing the domain to code the range. Let us call S_R and S_D the feature vectors sets corresponding to the ranges and the domains, respectively;
2. for each element of S_R, perform a search in S_D and find the closest domain feature vector. Use this domain to code the range.

Note that, since the ranges and the domains have not the same dimensionality, each block should be subsampled to the same size. Furthermore, the contractivity condition must be checked, so more than one domain should be tested in order to avoid scaling coefficients that are greater than one.

Using this method, the traditional linear search is replaced by a *nearest neighbor*, or *best match* search. This problem may be solved in $\mathcal{O}(\log |S_D|)$ average time for each query using k-d trees ([5], [3]), with a $\mathcal{O}(|S_D|.\log|S_D|)$ time preprocessing step. The overall complexity is then $\mathcal{O}((|S_D|+|S_R|).\log|S_D|)$, which is a quite good improvement compared to the original $\mathcal{O}(|S_D|.|S_R|)$.

Different approaches have been explored, using different operators to compute the feature vectors, and different structures to perform the search. In [22] and [2] the feature vectors used are the normalized DCT coefficients of the blocks, in [13], the structure used for the search is a R-tree, and the feature vectors used are the same as in [17]. We review the different methods in the following paragraphs.

Normalization The simplest way of finding good feature vectors might be the one described in [17], and also used in [13]. Suppose that x is a vector obtained by an ordering of the pixels values of a block (e.g. in scan-line order), and k the number of pixels in the block. The so-called *normalized projection operator* is defined as follows

$$\phi(x) = \frac{M.x}{\| M.x \|} \tag{1}$$

where

$$M_{ij} = \begin{cases} \frac{k-1}{k} & \text{if } i = j \\ -\frac{1}{k} & \text{if } i \neq j \end{cases} \tag{2}$$

Actually, M is a mean-removing matrix: the mean value of the vector's components is substracted to each component, making it invariant to any translation whose direction is parallel to the vector $(1, 1, \ldots, 1)$ (the offset addition in the block coding transformation). In [18], this operator has been generalized to the case where a set of p orthogonal fixed basis blocks was used: the matrix M is then replaced by the operator projecting the vectors on the orthogonal complement of the subspace spanned by the basis blocks. Finally, the normalization operation makes the feature vector invariant to any scaling operation on the block. These vectors have two main properties:

1. the sum of their components is zero (due to the form of the matrix in 2),
2. the sum of their squared components is 1 (due to the normalization in 1).

We can then prove that minimizing the collage error is equivalent to minimizing the value $\min(d(\phi(x), \phi(y)), d(-\phi(x), \phi(y)))$ (where x is the domain and y the range). The geometrical interpretation of this computation has been extensively discussed in [18] and [21]. In the latter, the same kind of operator is used to eliminate useless domain blocks using an angular distance criterion.

Frequency Domain Features. In [4], [22] and [2], feature vectors are computed from the DCT representation of the block, as follows:

1. compute the DCT transform of the block,
2. remove the DC component,
3. normalize the other coefficients.

The DC coefficient of the transformed block is the mean intensity of the block, so that removing it is equivalent to removing the mean of the block, as performed by M in the image space. It has the interesting side effect of decrementing the number of dimensions. The main advantage in this representation is that the energy packing property of the DCT allows the search algorithm to ignore the last components. This implies, however, a time-consuming frequency transform. Also note that reducing the number of dimensions by ignoring the high frequency components is a lossy speedup: the quality is worse than for a search taking all the components into account.

Extensive experiments on the use of this feature vector are described in [18], showing the obtained improvements.

1.3 The Discrete Karhunen-Loeve Transform

The *discrete Karhunen-Loeve transform* (KLT) is also known as *Principal Component Analysis*, *Hotelling transform* or *eigenvector transform*[1]. It has many properties that are used in the image processing field as well as in multivariate analysis.

Let S be a set of vectors in a k-dimensional euclidean space. We define the *covariance matrix* of this set in the following way:

$$C_S = \frac{1}{|S|} \sum_{x \in S} (x - m_S).(x - m_S)^T, \tag{3}$$

where $m_S = \frac{1}{|S|} \sum_{x \in S} x$ is called the *mean vector*. C_S is a $k \times k$ matrix where each element c_{ij} is the sample covariance of the i^{th} and j^{th} components of the vectors in S, and the elements c_{ii} on the diagonal are the variances of the i^{th} component of each vector.

C_S is a real, symmetric matrix, so that it is always possible to find an orthonormal basis of eigenvectors. Let us call A the $k \times k$ matrix of eigenvectors, where the i^{th} column of A is the eigenvector corresponding to the i^{th} greatest eigenvalue of C_S. The K-L transform of a vector $x \in S$ is

$$y = A.(x - m_S) \ . \tag{4}$$

The mean vector of the set $S' = \{y \mid y = A.(x - m_S), x \in S\}$ is zero, and the vector set S' is oriented in such a way that the variance is maximized on the first coordinate axes, and minimized on the last ones. The variances on each coordinate axis are in fact the positive eigenvalues $\lambda_1, \lambda_2, \ldots, \lambda_k$, where $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_k$. The first axis is called the *principal component*. The KLT

[1] Actually, the Hotelling transform is the discrete version of the Karhunen-Loeve expansion.

is optimal in the sense that the mean square error induced by the suppression of the last vector components, those having the least variance, is minimal (see below).

1.4 The Use of the KLT in Best Match Searching

The KLT has been used since a long time in best match searching algorithms (see e.g. [14]). Some vector quantization algorithms project the set S on its principal component and use this projection to quickly eliminate best match candidates ([15],[7]). It has also been shown that applying the KLT on a vector set before constructing a multidimensional indexing structure improves the speed of the search (see e.g. [20]). Some multidimensional tree structures based on a hierarchical space subdivision also use the principal component to partition the vector set ([19]). Actually, the KLT is very efficient in de-skewing a distribution, i.e. removing the correlation between vector components, so it is useful in strongly correlated data sets.

Before continuing, we will note some facts about distance measurement in a KL-transformed vector space. As A denotes an orthonormal basis of eigenvectors, the KLT is a distance-preserving transformation. If we denote $d(x, y)$ the euclidean distance between $x \in S$ and $y \in S$, we have:

$$d(x, y) = d(x', y') \tag{5}$$

where $x' = A.(x - m_S) \in S'$ and $y' = A.(y - m_S) \in S'$. So, if we define the set S'' as the projections of the vectors in S' on the first b coordinate axes, we can say:

$$d(x, y) \geq d(x'', y'') \tag{6}$$

where x'' and y'' are the vectors corresponding to x and y in the set S''. This property is useful in the search methods using a projection on the principal component: if the distance between a projection of a vector x on this axis and the projection of a query vector is already greater than the distance to the current best match, then x does not deserve any more consideration. The principal component should be used as projection axis, since it maximizes the number of eliminated candidates. To control the amount of variance projected on the first b axes, we will use the same notation than in [7], and introduce the following *preservation ratio*:

$$\frac{\sum_{i=1}^{b} \lambda_i}{\sum_{i=1}^{k} \lambda_i} \tag{7}$$

This definition is generalized to any type of transform by replacing the eigenvalues by the variances on each axis. Another definition of the optimality of the KLT is that it maximizes the preservation ratio, i.e. there is no other orthonormal basis for which the preservation ratio is greater. When the data set

is strongly correlated, we can choose b so that the preservation ratio is not below a predefined threshold. Using this threshold, we can bound the difference $d(x,y) - d(x'',y'')$ on the average, and perform a search in the b-dimensional space.

2 Using the KLT in the Fractal Compression Best Match Search

2.1 Applying the KLT to the Feature Vectors

The computation of KL-transformed feature vectors may be carried out in four steps:

1. compute the mean-removed normalized feature vectors $\phi(x)$,
2. center the distribution, so that the mean vector is zero,
3. compute A,
4. multiply each feature vector by A.

The distribution of interest here is S_D, but one could imagine to use S_R as well. We will introduce the notations $S'_D = \{y' \mid y' = A.(y - m_{S_D}), y \in S_D\}$, and $S'_R = \{y' \mid y' = A.(y - m_{S_R}), y \in S_R\}$.

The problem is that, even if the original image blocks form a strongly correlated vector set, the normalized feature vectors may lose this property, and the preservation ratio for a fixed b is quite different in the two sets. The principal component of the non-normalized 4×4 blocks in the Lena image preserves more than 85% of the variance, but less than 55% for the feature vectors. Another important remark is that all the normalized feature vectors belong to the unit sphere, so the variances may not be as large as for the non-normalized vectors.

Another approach would be to compute the feature vectors from a KLT-based representation. We would like these KLT-based feature vectors to be invariant to the transformations used in the iterated system. This approach is more difficult than the one used for the DCT, since the KLT is a data-dependent transform. The DCT approach is also simpler when dealing with the projection part of the operator: we simply ignore the DC coefficient of the transformed vector. The equivalent operation in a KL-transformed space is much more complicated. This is why we think that the KLT should be applied in the last stage of the process, on the normalized projected feature vectors $\phi(x)$. More efficient energy packing and dimensionality reduction are expected compared to the DCT, since the KLT performs an optimal dimensionality reduction on every distribution.

Experimental results showing the preservation ratios confirm these hypothesis. Figure 1 shows the results obtained on 16K 4×4 image blocks taken from the Lena image.

A few remarks might be useful on the way the transformation may be computed efficiently. First, note that the centering step of the KLT is facultative in our application: the variances will not be affected by the position of the mean

vector, so this step may be skipped, and 4 may be replaced by

$$y = A.x \tag{8}$$

Another remark is about the compared computation time of the DCT and the KLT. The computation of A has a $\mathcal{O}(k^2.|S_D|+k^3)$ time complexity, if we suppose that the matrix is computed from the S_D distribution. This is a severe drawback compared to the DCT, where no such computation is needed. A good approximation of A may however be obtained by subsampling the data set on which it is computed. The KL-transformation of a vector also takes $\mathcal{O}(k^2)$ operations, compared to $\mathcal{O}(k.\log k)$ for the DCT. Experimental tests carried on on 512×512 images show however that this additional computation time is not as important as the one needed for the computation of A.

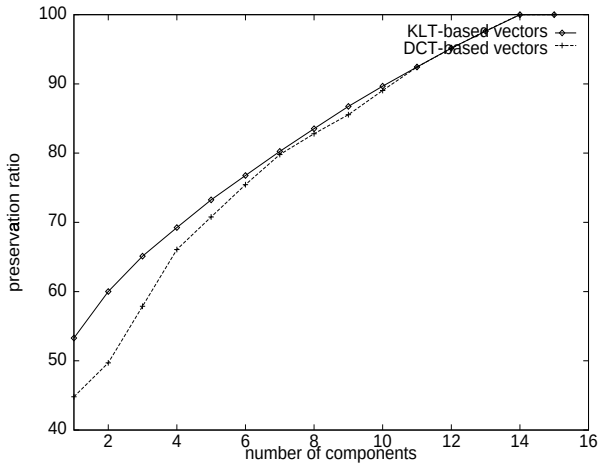

Fig. 1. Preservation ratios (in percent) for energy-packing feature vectors.

2.2 An Appropriate Search Algorithm

Range[2] searching consists in reporting all vectors that are contained in a query volume. For the search of the domain in the Karhunen-Loeve transformed feature space, we use a modified orthogonal range search algorithm. Orthogonal range searching in a k-dimensional space consists in searching only the subspace defined by the products of k 1-dimensional intervals, on each of the k coordinate axes. So the search is restricted in a hyper-volume bounded by two $(k-1)$-dimensional hyper-planes on each axis. It is the equivalent of a spherical range search when

[2] The word "range" is to be understand in its original meaning, it does not refer to range blocks.

the L_∞ metric is used. Range searching and nearest neighbor searching are two closely related problems, which often use the same kind of structures: quadtrees ([8]), k-d trees ([5]), R-trees ([11]).

We outline here a general algorithm which does not use any tree structure. It is extensively described and analyzed in [16], although the same paradigm has already been exploited in some earlier algorithms ([10], [14]). The steps are shown in algorithm 1. The notation $x = (x_1, \ldots, x_k)$ is used for the vector components. This algorithm returns the set of vectors contained in the hypercube of side $2.\epsilon$ centered on the query point Q. This set may contain many vectors, and in that case an additional exhaustive search should be made. If the result is the empty set, then another search should be performed with a greater ϵ. In [16], an efficient structure is designed to handle these operations. First, the

Algorithm 1 The general range search algorithm

S is the searched vector set
Q is the query vector
the side of the search box is $2.\epsilon$
vector set V, T
$V \leftarrow \{x | x \in S, x_1 \in [Q_1 - \epsilon, Q_1 + \epsilon]\}$
for $i = 2$ to k **do**
  $T \leftarrow \{x | x \in S, x_i \in [Q_i - \epsilon, Q_i + \epsilon]\}$
  $V \leftarrow V \cap T$
end for
return V

vectors are sorted along each coordinate axis: k sorted lists are obtained. A mapping is maintained from the elements of each sorted lists to each vector of the original set, so that one can retrieve the index of a vector from its position in the sorted lists in constant time. This indirection is called *backward mapping*. The opposite mapping is also recorded, and is called *forward mapping*. To find the set $\{x | x \in S, x_i \in [Q_i - \epsilon, Q_i + \epsilon]\}$, two binary searches are performed, one for each limit, in the i^{th} ordered list. Then we compute the intersection between the set of vectors lying in the interval and V, the set of vectors left in the previous step. The forward mapping allows the algorithm to use only integer comparisons for computing this intersection.

In order to suit the needs of the application, and to exploit as much as possible the properties of the transformed feature vectors, this algorithm is slightly modified. The modified version is shown in algorithm 2. In this algorithm, the trimming of the list stops as soon as the current set contains less than a predefined number r of vectors. The returned vectors are not always contained in the $2.\epsilon$-sided hypercube, but the algorithm guarantees that at least r vectors are returned (excepted in the case where the first set $\{x | x \in S, x_1 \in [Q_1 - \epsilon, Q_1 + \epsilon]\}$ contains less than r elements, which can be made exceptional by correctly tuning the value of ϵ).

Algorithm 2 The modified range search algorithm

S is the searched vector set
Q is the query vector
the side of the search box is $2.\epsilon$
vector set V, V', T
$V \leftarrow \{x | x \in S, x_1 \in [Q_1 - \epsilon, Q_1 + \epsilon]\}$
$i \leftarrow 2$
while $|V| > r$ **and** $i \leq k$ **do**
  $T \leftarrow \{x | x \in S, x_i \in [Q_i - \epsilon, Q_i + \epsilon]\}$
  $V' \leftarrow V$
  $V \leftarrow V \cap T$
  $i \leftarrow i + 1$
end while
if $|V| \geq r$ **then**
  return V
else
  return V'
end if

The preprocessing step has time complexity $\mathcal{O}(k.|S|.\log|S|)$ if we use the heapsort algorithm. The cost of establishing the $Q_i \pm \epsilon$ bounds on each axis is $\mathcal{O}(k.\log|S|)$. The cost of the trimming of the lists for a single query depends on the distribution of the vectors. It has been shown in [16] that the time complexity of the original algorithm for the case where the vectors are uniformly distributed in the unit hypercube is $\mathcal{O}(|S|.(\epsilon + \frac{1}{1-\epsilon}))$ for small ϵ.

It can seem strange to use an algorithm for which the complexity grows linearly with the number of domains instead of a tree-based one, logarithmic in the same variable. Two reasons motivate this choice:

1. the cost of a search in a multidimensional tree is logarithmic in the number of elements in the tree, but roughly exponential in the number of dimensions of the searched space: this is the well-known "curse of dimensionality" problem, which often makes some tree search methods useless in high dimensions. One can easily understand that by remarking that in a $k-$dimensional rectangular cell grid, a single cell has always $3^k - 1$ neighbors (two cells are neighbors if they have at least one common point), which is exponential in k.
2. The cost of the range search algorithm also depends on ϵ, which is quite small.

As a consequence, the constant factors involved in the two complexities are of very different orders, and make the range search method attractive.

In these algorithms, the elimination of candidates begins on the first axis, and ends on the k^{th} (or before in the modified version), in that order. This order is justified by the energy packing property of the vectors. We can easily show that the average number of candidates left after the b^{th} iteration (including the

search on the first axis) is:

$$|S|.\prod_{i=1}^{b} P_i \tag{9}$$

where

$$P_i = P\{x_i \in [Q_i - \epsilon, Q_i + \epsilon] \,|Q\} \tag{10}$$

is the probability that x_i lies in the interesting interval on the i^{th} axis, for a given Q and any $x \in S$. As the coordinate axes are sorted according to the variances of the corresponding components of the vectors, the algorithm ends up more rapidly with a small subset of the elements than for any other order, and the most important components are taken into account first.

To give an estimation of the average number of iterations of algorithm 2, we will place ourselves in a simplistic situation where the two following assertions hold:

1. the vector components $\{x_i | i = 1, \ldots, k,\ x \in S\}$ are independent, uniform random variables lying in the range $[-l_i/2, +l_i/2]$,
2. $\forall\, 1 < i \leq k$ we have $l_i = \alpha.l_{i-1}$, where $0 < \alpha < 1$.

The second assertion is an attempt to model the energy packing due to the KLT. It is fairly easy to show that the average number of iterations is[3]

$$\hat{b} = \frac{\log \frac{P_1}{\sqrt{\alpha}} + \sqrt{\log(P_1.\sqrt{\alpha})^2 - 2.\log\alpha.\log c}}{\log \alpha}\ , \tag{11}$$

where $c = r/|S|$ is the desired ratio of points left at the end of the search. In the case where $\alpha = 1$, i.e. if we search in a hypercube, the solution is

$$\hat{b} = \frac{\log c}{\log P_1} - 1\ . \tag{12}$$

For a precise description of the developments, one can refer to [6].

2.3 Implementation

In order to show the improvements obtained, four different methods have been implemented:

1. a simple k-d tree search using the mean-removed normalized feature vectors,
2. a k-d tree search using the KL-transformed feature vectors,
3. a modified range search (algorithm 2) using the KL-transformed feature vectors,
4. a modified range search using the DCT-based feature vectors.

[3] the basis in which the logarithms are taken has no importance.

Only the first method finds the exact nearest neighbor in the feature vector space. The two others perform an approximated search: for the k-d tree using the KL-transformed vectors, the least significant components are ignored, and for the modified range search, the trimming of the lists should stop before the k^{th} axis, and ignore as well the remaining components.

Three different range block sizes have been used: 4×4, 8×8 and 16×16. These sizes are the ones usually used in quadtree partitioning schemes, and they have been tested separately. The domain blocks are aligned on a 4×4 square lattice, and they are two times bigger than the range blocks. No square isometry is used in the transformations. All the blocks, regardless of the original size, are first down-sized to size 4×4, thus fixing k =16, and the different normalizations and transform operations are performed in $\mathbb{R}^{16}$. The KL basis – as described in section 2.1 – is computed from the domain blocks distribution. Also note that in the three methods, for the sake of simplicity, we only use the vectors corresponding to $\phi(x)$, and forget the opposite key $-\phi(x)$.

The transformation coefficients are quantized, and the collage error is computed using the quantized coefficients. 5 and 7 bits are used for s_i and o_i, respectively. We allow s_i to vary between -1.2 and +1.2.

For the k-d tree methods, an optimized algorithm has been used, based on the original algorithm from Bentley ([5]), and implementing the *distance refinement* of Arya ([3]), which avoids a lot of floating point operations during the search. The search is not approximate, and the r nearest neighbors are sought. In our tests, we have found $r = 10$ to be a good value for the application. This value is also the maximum number of elements contained in each bucket of the tree. For the case where the KL-transformed vectors are used, only the first b components are used: as we have seen, the 90% preservation ratio is obtained with b =12 on the Lena image, which is the reason why we have used this value for our tests.

We have not tested the scheme using a k-d tree with DC-transformed vectors, because we can see on figure 1 that the preservation ratio for b =12 is the same as with the KLT, so the two methods should perform comparably. One can refer to [18] for this experiment.

For the modified range search method, the number of elements needed has been fixed to r =20, and we use ϵ =0.3. These values have been found empirically. The number of iterations in the trimming loop of algorithm 2 using these parameters is roughly varying between 6 and 13.

The memory requirements for the KLT-based methods are the same as for the simpler feature vectors, since only the KL-transformed representations are saved.

2.4 Results

The three methods have been tested on 7 grey-scale images of size 512×512: Lena, Zelda, Peppers, Baboon, Goldhill, Barbara and Boat. For each block size, the overall collage error (MSE) and the average compression times in seconds are given. The time needed for the preprocessing steps (computation of the KLT basis and construction of the search structures) is also reported. The collage

error is used instead of the decoded error for the sake of both simplicity and generality. The compression ratio is constant for a given block size and a coding of the transformations. In our simple quantization scheme, a transformation is coded in 26 bits, and the compression ratios for the three block sizes are respectively 1:5, 1:20 and 1:100.

Average time and error results are presented in tables 1, 2 and 3. The timings have been made on a Pentium-based machine. We can make the following conclusions:

1. the dimensionality reduction using the KLT does not introduce noticeable error,
2. this dimensionality reduction speeds up the search up to a factor of 2,
3. the modified range search method is faster than the other two, excepted for the 16×16 blocks,
4. significant MSE gains are observed for the 16×16 case,
5. the preprocessing steps in the modified range search are not prohibitive, and largely compensated by the speed gain during the search (at least for the 4×4 and 8×8 blocks),
6. the performances of the KLT-based feature vectors and the DCT-based ones are very close to each other, with a slight advantage for the DCT, mainly due to the fast preprocessing step.

Note that the block dimension used does not change the value of k: the search is always performed in $\mathbb{R}^{16}$. The main difference is that for a larger block size, the total number of ranges is decreasing, and the encoding is faster.

We can explain the results obtained for the bigger blocks by the fact that the distribution of the feature vectors is not the same for all the sizes: bigger blocks seem to give feature vectors with less variance. The choice of ϵ in the modified range search method should have been different for each block size, according to the distribution of the feature vectors.

We can imagine to use the new search method in a quadtree partitioning scheme. In that case, one transform is necessary for each quadtree level. The memory requirements are however the same as for a static partitioning corresponding to the last quadtree level.

Table 1. Results for 4×4 blocks

Encoder	Preprocessing time (s.)	Total time (s.)	MSE
k-d tree – no transform	5.32	935.92	37.50
k-d tree – with KLT and b =12	13.18	417.93	37.83
modified range search – with KLT	15.32	153.79	38.49
modified range search – with DCT	8.61	153.36	38.49

Table 2. Results for 8×8 blocks

Encoder	Preprocessing time (s.)	Total time (s.)	MSE
k-d tree – no transform	6.32	186.79	137.09
k-d tree – with KLT and b =12	13.89	92.21	140.63
modified range search – with KLT	15.75	73.79	137.37
modified range search – with DCT	9.32	68.21	137.54

Table 3. Results for 16×16 blocks

Encoder	Preprocessing time (s.)	Total time (s.)	MSE
k-d tree – no transform	9.89	45.93	265.07
k-d tree – with KLT and b =12	17.32	33.93	267.85
modified range search – with KLT	18.89	51.36	257.12
modified range search – with DCT	12.32	44.92	257.04

3 Conclusion

In this paper, we have reviewed the general method of fractal compression using feature vectors, shown the improvements that could be obtained using a decorrelating rigid transform, and designed a simple algorithm using these features, faster than other methods of the same flavor. An optimized implementation of the technique, integrating an efficient block partition, could lead to competitive coder performances. Other experiments using more specific feature vectors, or wavelet domain transformations, could be fruitful too.

4 Acknowledgements

The author thanks Dietmar Saupe, Brendt Wohlberg, and Giuseppe Lauria for useful discussions and pieces of advice on this subject, and Efstathios Hadjidemetriou and Sameer Nene for providing the code of the algorithm described in [16]. The remarks of the referees also greatly helped to improve the quality of this text.

References

1. E. Amram and J. Blanc-Talon (Oct. 1997): "Quick search algorithm for fractal image compression". *Proc. ICIP-97 IEEE International Conference on Image Processing*, Santa Barbara, California.
2. O. C. Au, M. L. Liou and L. K. Ma (Oct. 1997): "Fast fractal encoding in frequency domain". *Proc. ICIP-97 IEEE International Conference on Image Processing*, Santa Barbara, California.
3. S. Arya and D. Mount (1993): "Algorithms for fast vector quantization". *Proc. Data Compression Conference*, J. A. Storer and M. Cohn eds., Snowbird, Utah, IEEE Computer Society Press, 381-390.

4. K. U. Barthel (1995): *Festbilcodierung bei niedrigen bitraten unter verwendung fraktaler methoden im orts und frequenzbereich.* PhD. Thesis, Technische Universität Berlin.
5. J. L. Bentley, R. A. Finkel and J. H. Friedman (1977): "An algorithm for finding best matches in logarithmic expected time". *ACM Trans. Math. Software*, 3(3): 209-226.
6. J. Cardinal (1998): "Fractal compression using the discrete Karhunen-Loeve transform". Brussels Free University, Internal Report, `http://homepages.ulb.ac.be/~jcardin`.
7. C.-C. Chang, D.-C. Lin and T.-S. Chen (1997): "An improved VQ codebook search algorithm using principal component analysis". *Journal of Visual Communications and Image Representation*, Vol.8, No.1, March, pp. 27-37.
8. R. Finkel and J. L. Bentley (1974): "Quadtrees: a data structure for retrieval of composite keys". *Acta Inf.*, 4, 1, 1-9.
9. Y. Fisher (1994): *Fractal image compression - Theory and application.* Springer-Verlag, New York.
10. J. H. Friedman, F. Baskett and L. J. Shustek (Oct. 1975): "An algorithm for finding nearest neighbors". *IEEE Trans. Comput.*, vol. C-24, pp. 1000-1006.
11. A. Guttman (1984): "R-trees: a dynamic index structure for spatial searching". *Proceedings of the ACM SIGMOD International Conference on the Management of Data*, pp. 47–57.
12. A. Jacquin (August 1989): *A Fractal Theory of Iterated Markov Operators with Applications to Digital Image Coding.* PhD. thesis, Georgia Institute of Technology.
13. J. Kominek (1995): "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.
14. R. Lee (1976): "Application of principal component analysis to multikey searching". *IEEE Transactions on Software Engineering*, SE 2(3): 185–193.
15. C. H. Lee and L. H. Chen (1995): "High-speed closest codeword search algorithms for vector quantization", *Signal Processing* 43, pp. 323–331.
16. S. A. Nene and S. K. Nayar (Sept. 1997): "A simple algorithm for nearest neighbor search in high dimensions". *IEEE Trans. on Pattern Analysis and Machine Intelligence*, vol. 19, No 9, pp. 989–1003.
17. D. Saupe (1998): "Accelerating fractal image compression by multi-dimensional nearest-neighbour search". *Proceedings DCC'95 Data Compression Conference*, J.A. Storer and M.Cohn (eds.), IEEE Comp. Soc. Press.
18. D. Saupe (1995): "Fractal image compression via nearest neighbor search". *Conf. Proc. NATO ASI Fractal Image Encoding and Analysis*, Trondheim, July 1995, Y. Fisher (ed.), Springer-Verlag, New York.
19. R. F. Sproull (1991): "Refinements to nearest-neighbor searching in k-dimensional trees". *Algorithmica*, 6, p. 579–589.
20. D. A. White and R. Jain (1997): "Algorithms and strategies for similarity retrieval". Visual Computing Laboratory, University of California, Internal Report. `http://vision.ucsd.edu/papers`.
21. B. E. Wohlberg and G. de Jager (1994): "On the reduction of fractal image compression encoding time". *1994 IEEE South African Symposium on Communications and Signal Processing (COMSIG'94)*, pp. 158–161.
22. B. E. Wohlberg and G. de Jager (1995): "Fast image domain fractal compression by DCT domain block matching". *Electronic Letters*, 31, 869–870.