ISSN 1361 - 6161

THE UNIVERSITY of MANCHESTER

Computer Science
University of Manchester

Fractal Encoding by Classified Domain Trees

Behnam Bani-Eqbal

Department of Computer Science
University of Manchester
Technical Report Series
UMCS-96-1-3

Fractal Encoding by Classified Domain Trees*

Behnam Bani-Eqbal

Department of Computer Science
University of Manchester
Oxford Road, Manchester, UK.
behnam@cs.man.ac.uk

25 January 1996

*Refereed by: Ian Pratt

Abstract

Fractal coding of digital images offers many promising qualities. The encoding process, however, suffers from the long search time of the domain block pool. A standard technique for speeding up the encoding is the feature classification of blocks. In this paper we show that the classes can be arranged in a tree, and searched by a pruning algorithm developed and explained in a previous paper by the author. We show that this hybrid method enhances the search speed by upto 75% over the purely feature based method.

1 Introduction

Recently fractal compression of digital images has attracted much attention. It is based on the mathematical theory of iterated function systems (IFS) developed by Hutchinson [1] and Barnsley [2]. The application of IFS theory to image compression was proposed originally by Jacquin [3] and Barnsley [4]. Since their seminal work, many variants of the IFS method have been reported in the literature [5]. The IFS theory and its applications to image compression together with recent research results are collected in the book edited by Y. Fisher [6].

The basic idea of the fractal image compression is as follows. The image is partitioned into non-overlapping range blocks. For every range block a similar but larger domain block is found. The mapping of the domain block to the range block is recorded. The compressed image consists of all the transformation data. Decoding proceeds as follows. The transformations are applied to an arbitrary initial image, and the process is repeated. Provided the transformations are contracting, the images converge to a stable image, resembling the original picture.

The fractal method offers high compression ratio, good image quality, and resolution independence of the decoded image. Its disadvantage is the long encoding time due to the repeated search of the domain pool. Much effort has been spent in devising ways to overcome this problem. The paper by D. Saupe et. al. [7] presents an excellent summary of the results. One efficient method is block classification. The domains are grouped independently and online into predefined classes. Only the class of a range block is searched for a matching domain.

In a previous paper [8], we presented a tree based scheme for speeding up the search. We structure the domain pool into a tree, and use a pruning algorithm to search for matching blocks. Experimental results demonstrate substantial improvement in the search time.

In this paper we show that the tree method may be combined with the block classification to get a greater speed-up. In block classification the classes are stored without any additional structure, usually as a list, and searched linearly until a best match is found. We should expect an enhancement if we arrange the classes into trees, and use the pruning algorithm for the search. Another way to express this idea is that the domain pool is grown into lots of small trees rather than one big one. The experiments show that a significant speed-up is achieveable provided two conditions are met. First, only reasonable size classes should be grown into a tree. Second, the feature extraction methods for the classification and for the tree construction should be distinct. This latter condition is explained in section 5.

2 Iterated Function Systems

In this section, we explain the basics of the fractal compression scheme. This scheme undelies all further refinements. Our experiments are based on a scheme called quad-tree partitioning, as explained in Fisher's book [6]. The appendix of this book contains a complete implementation of the algorithm in the C language. We have adapted this program to incorporate the tree method.

The image is a digital gray level picture. We cover it by non-overlapping square range blocks of size $n \times n$, where normally n is a power of 2 (and may be variable). The first block is aligned with the bottom left corner of the picture. If the blocks do not reach upto the right or the top edge, then the margin can be covered by rectangular blocks. We assume that the blocks cover the image exactly. Next, for every range block, a similar but larger domain block is found. The domain blocks are usually twice the size of the range, and are located anywhere on the image. The x, y coordinates of the lower left corner of the domain block determine its location. This can be on each pixel. Some authors, however, suggest that the blocks be aligned on a grid size of n or $n/2$. The more domain blocks, the better the decoded image quality, but the longer the compression takes.

The compression process is as follows. A range block R is tested against every block in the domain pool to find a block D and a transformation T such that $T(D)$ is the best match for R. The closeness is measured by a distance metric, usually taken to be the root-mean-square (rms) metric. For blocks $A = (a_i)$ and $B = (b_i)$ $0 \leq i < m$ it is $(\sum(a_i - b_i)^2/m)^{1/2}$.

The transformation T is composed from a contracting map followed by a geometric map followed by a massic map. The contracting map shrinks the domain block to half the size, replacing 2×2 pixel areas by their average. The geometric map is one of the eight flips (or symmetries) of the square. We consider the blocks independently of the image. Some authors consider them in the image plane, so their geometric map involves a translation. For the sake of simplicity, we call the flipped contracted block a domain block. It will be clear from the context which kind of domain block is meant.

The massic map changes the contrast and brightness by a scale factor a_0 and an offset a_1. The domain block (d_{ij}) is mapped to $(a_0 d_{ij} + a_1)$. The map T is contracting if $a_0 < 1$. Jacobs et al. [9], however, show that it is not necessary to enforce this condition for all the maps. The offset is constant as in the original work of Jacquin [3], but researchers have used more general forms.

The massic map is chosen to minimise the distance from the (contracted flipped) domain block D to the range block $R = (r_{ij})$. This amounts to minimising the positive definite quadratic form $F = \sum(a_0 d_{ij} + a_1 - r_{ij})^2$. Thus a_i are the solutions of the equations:

$$\partial F / \partial a_i = 0$$

Let us call F_{min} the minimum F obtained from the solutions. The compression algorithm runs as follows. Given a range block, the minimum difference F_{min} with every domain block is calculated. The transformation parameters with the smallest F_{min} are stored as the IFS code. This process is repeated for all the range blocks.

```
For every range block,
    For every domain block,
        Shrink and flip in 8 ways.
        Determine the best a0, a1, and Fmin.
    Choose the block giving the smallest Fmin,
    Store the transformation.
```

Most of the the encoding time is spent in the repeated search of the domain pool.

3 Block Classification

By far the slowest part of the encoding process is the repeated search of the domain pool. Accordingly researchers have investigated many schemes to overcome the shortcoming [7]. One method is to classify the blocks according to their content. A range block is matched against only the domains in the same class. The key question is how to assign a class number to a block. The class number must be insensitive to scale and offset adjustments to the pixels, and also to small perturbations of their values.

One method was devised by Jacobs, Fisher and Boss [9]. It is also explained in Fisher's book [6]. This book includes a C implementation of the algorithm. The idea is to distinguish a block by the order of the average brightness of its four quadrants. More specifically, a block is divided into four quadrants, and their average brightness are computed. The blocks are classified by the relative position of their quadrants numbered in order of brightness. The relative ordering is independent of offset and scale change, provided the scale is positive. Thus if a domain is a good match for a range, then they both possess the same relative ordering. There are 24 permutations possible, resulting in 24 classes.

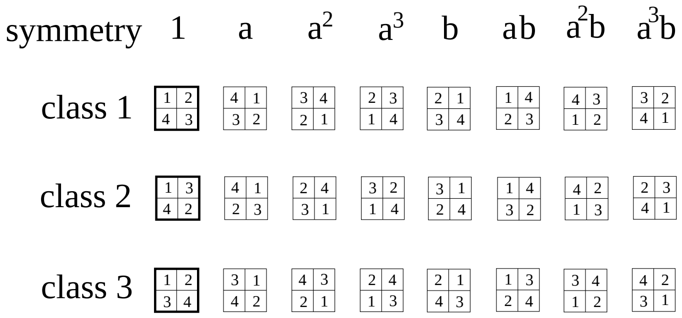

Figure 1: The 24 orientations of the quadrants fall into 3 classes under the symmetry operations.

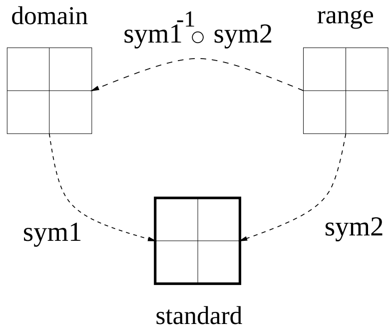

Figure 2: If $sym1$ and $sym2$ bring the domain and the range to the standard position, then $sym1^{-1} \circ sym2$ transforms the range to the domain.

A range may be flipped by a symmetry before matching against a domain. There are 8 symmetries (4 rotations and 4 reflections). So in effect a range is tested against 8 of the 24 classes. We combine the classes which may be permuted by the symmetries, and obtain three parent classes, as shown in the figure 1. The numbers in the blocks denote the relative ordering of the quadrants, so 1 has the biggest average, etc.

When we try and match a range against a domain, the best symmetry to apply is naturally the one which harmonises their ordering. This can be computed immediately if we know the ordering which brings both blocks into a fixed ordering, say the ordering of the first block in the class. This idea is illustrated in figure 2. If the domain and the range are brought to the standard orientation by $sym1$ and $sym2$. Then the symmetry $sym1^{-1} \circ sym2$ arranges the range to the orientation of the domain.

In figure 1 the first block in each row is chosen as the standrad orientation for the class. The symmetry operations in the first row transform the blocks to the standard order. The symmetries of the square are generated by a and b which are the rotation by 90 degrees anti-clockwise and the reflection through the vertical midline.

A negative scale multiplier rearranges the relative ordering of the quadrants. So we also compute the class and the symmetry when the sign of the quadrant averages are changed. It turns out that the class a block belongs to does not change, but the symmetry operation is different. In the search process, a range block is compared against two classes, one for positive scale and one for negative scale.

The code presented in Fisher's book [6] introduces two refinements. First the classes are further split into 24 sub-clsses each by noting the 24 possible orderings of the quadrant variances. Second it uses variable size range blocks, in a method aptly called quad-tree partitioning. If a range block cannot be matched to a tolerance set by the user, then it is partitoned into four quadrants, and they are tried recursively for a match. The starting size of the range blocks and the depth of the recursion are set by the user. The idea is that the use of larger range blocks results in a higher compression ratio.

4 Tree search

In a previous paper [8], the author presented a novel method of speeding up the encoding process. This is based on arranging the domains in a tree. The reader should refer to the paper for a complete discussion of the method. For completeness we explain the main idea. For a domain D and a range R, thought of as points in Euclidean space, we define the conic distance:-

$$d(D,R) = min_s \sum (sd_i - r_i)^2 = |R|^2 - (D \circ R)^2/|D|^2$$

In the above, $|R|$ and $|D \circ R$ are the standard norm and the inner product functions in the euclidean space. The $d(D, R)$ is the distance of R to a line through D. If the blocks are normalised by subtracting their average brightness, then the offset adjustment is zero. The best match to R is then the nearest domain D in this conic sense.

Now suppose we know that the best match should lie within a distance e. To find it, we must search the cone defined by the equation $d(D, R) \leq e$. The main result of [8] is an algorithm to enumerate blocks lying in a cone. The core of the algorithm depends on the following incremental distance evaluation.

Lemma 1 *Let D, R be two blocks. Choose a postion and let the pixels there be d, r. Let the remaining pixels define the sub-blocks D', R'. Put $e' = d(D', R')$. Let $e \neq |R'|^2$ be such that $e' \leq e \leq |R|^2$. Define*

$$c = r\frac{D' \circ R'}{|R'|^2 - e}, \quad l = \frac{|D'|\sqrt{(|R|^2 - e)(e - e')}}{||R'|^2 - e|}$$

Then $d(D, R) \leq e$ if and only if d satisfies:-

1. *If $e < |R'|^2$, then $|d - c| \leq l$.*
2. *If $e > |R'|^2$, then $|d - c| \geq l$.*

This Lemma enables us to enumerate the blocks one step at a time. Let us say, recursively, that we have enumerated the sub-blocks D' within the distance e of R'. We compute the admissible range of values of d. Then we can enumerate the bigger blocks provided we can reach from a sub-block to them. This requires a tree arrangement of the blocks, where the nodes are the pixel values. For fractal encoding, we set e to a sub-multiple of $|R|^2$. Further details of the data structure and the search algorithm may be found in the quoted paper.

5 Hybrid Method

The main idea of this paper is that we can improve both the tree method and the classification method by a combination of both. The idea simply is to build several trees. Thus we classify the domain pool into classes and build them into distinct trees. Since the tree pruning is faster than linear search, so we should expect to gain on the encoding speed.

There is one problem. As noted in [8], building the tree on the full set of block coefficients results in a widely branching stem, which thins out rapidly to the leaves. This is both wasteful and inefficient. The solution to this problem is to concentrate the block information into fewer coefficients, and use them to build the tree.

In [8] we achieve this with quadrant averages, utilising the following lemma:-

Lemma 2 *Let the pixels of the blocks D and R be partitoned into equally-sized sets of size k. Let $\overline{D}, \overline{R}$ be obtained by averaging the partitions. Then $d(\overline{D}, \overline{R}) \leq d(D, R)/k$.*

However, the quardrant average method does not work with the classification scheme. The reason is they are both based on quadrant averages. Thus the blocks in a class have already similar averages (at least they make the same order) and so the search algorithm is less likely to prune the tree. We need distinct ways of extracting the block information.

We observe that the conic distance is a sum of squares. Thus it is preserved under an orthonormal transform of the image space. Formally:-

Lemma 3 *Let $\alpha : R^m \to R^m$ be an orthonormal transform of the image block space, then $d(D, R) = d(D^\alpha, R^\alpha)$.*

Thus we may use a subset of the transformed coefficients to build the tree. Examples of orthonormal transforms are the wavelet transform and the DCT.

1. Wavelet Transform. We use the Haar wavelets, which in 1D is generated by translated dilations of the mother Haar wavelet. More formally, the haar wavelets at level j are $\psi_i^j(x) = \psi(2^j x - i)$ where

$$\psi(x) = \begin{cases} 1 & \text{for } 0 \leq x < 1/2 \\ -1 & \text{for } 1/2 \leq x < 1 \\ 0 & \text{otherwise} \end{cases}$$

   To transform a 2D image block we use the 2D haar basis which is simply the (tensor) product of two 1D bases. If we view the image array as a piecewise constant function supported on the unit interval, then it can be written as a linear sum of the basis elements. The coefficients form the Haar transform. For more detail refer to [10].

2. Discrete Cosine Transform (DCT). This is a well studied transform in image processing applications [11]. It is the underlying coder of the JPEG image compression standard. The DCT coefficients of an $n \times n$ block $A = (a_{ij})$ are calculated from the formula:-

$$DCT_{uv} = \frac{2}{n} C(u)C(v) \sum_{i,j=0}^{n-1} \cos\frac{(2i+1)u\pi}{2n} \cos\frac{(2j+1)v\pi}{2n} a_{ij}$$

   where $C(i) = \begin{cases} 1/\sqrt{2} & \text{for } i = 0 \\ 1 & \text{otherwise.} \end{cases}$

We experimented with trees of different depths, however a depth of 4 gives the best time reduction. So we use the first four of the transformed coefficients (but not the term representing the average). As in [8], the tree is searched with an error set to $e = |R'|^2/\beta$ where β is a fixed constant, and R' is the subset of the transformed coefficients. Experiments show that a range of β values from 4 - 10 give good results.

6 Experimental Results

In this section we evaluate the performance of our coder. As the base coder, we use Fisher's quad-tree fractal coder as presented in his book [6], called here the QF algorithm. Our coder is based on Fisher's code but it includes the tree search algorithm. We used both the Haar transform and the DCT for tree data. It turns out that the Haar transform does not produce significant improvement in the run-time, so we focus on the DCT. We used trees of depth 4, built from the first 4 DCT coefficients. This coder is called the hybrid fractal (HF) algorithm. All the experiments were performed on a SUN sparcstation 10 model 30 with 64 MB RAM. The test images are the classic ones from the USC database. The HF shows real gain for the bigger images of size 512×512, so all test images are of this size.

Fisher's code classifies the domains, then it recursively partitions the image for range blocks. Typically it encodes an image of 512×512 by range blocks of size 32×32 through to 4×4, corresponding to domains of twice the size. The domain blocks of each size are classified at the beginning. The classes are arranged into linked lists. If a range block cannot be matched within a tolerance, then it is quad-treed down. Table 1 shows the number of domains in each class for Lena.

Table 2 shows the search time for each domain size for various images. It is clear that the encoding time is spent mainly in the search of the 8×8 domains. The class lists of size 16 or above are short and are searched few times. Thus arranging them into a tree does not pay off. So the hybrid coder grows only the size 8 classes into trees. Table 2 also shows the tree search times. It is clear that the tree algorithm saves massively there because it returns a much reduced number of domains for comparsions. Table 3 shows this average number per each tree.

Finally, Table 4 compares the running times of our hybrid fractal coder against the Fisher coder for various images, and the PSNR of the decoded images. The hybrid algorithm shows a speed-up of 75% with a drop in PSNR of 1dB on Lena, while the Mandrill image shows an improvement of 83% with a PSNR drop of 1.4dB. The Mandrill gives a better improvement because most of the range blocks are partitioned down to a size of 4×4. So the tree algorithm does more of the work.

7 Conclusions

We have presented a technique for speeding up the fractal encoding based on a combination of the tree method and the traditional block classification. We have shown experimentally that it can lead upto 80% speed-up gain against the class-based search method. We used the tree algorithm on the smallest domain size because they are more numerous. In future, we plan to investigate ways of speeding up the search of the bigger domain classes.

References

[1] J. E. Hutchinson, 'Fractals and Self-Similarity', *Indiana Univ Math Journal, Vol 35, pp 713 - 747, 1981.*

[2] M. F. Barnsley, *'Fractals Everywhere'*, Academic Press, San Diego, CA, 1988.

[3] A. E. Jacquin, 'A fractal theory of iterated Markov operations, with applications to digital image coding', *Ph.D. Thesis, Department of Mathematics, Georgia Institute of Technology, 1989.*

[4] M. Barnsley, L. Hurd, *'Fractal Image Compression'*, Jones and Bartlett, 1992.

[5] D. Saupe, R. Hamzaoui, 'A review of the fractal image compression literature,' *ACM Computer Graphics 28, 4 (1994) pp 268 - 276.*

[6] Y. Fisher, Editor, *Fractal Image Compression: Theory and Applications to Digital Images*, Springer Verlag, 1994.

[7] D. Saupe, R. Hamzaoui, 'Complexity Reduction methods for fractal image compression', in: IMA Conf. Proc. on Image Processing; Mathematical Methods and Applications, Oxford University Press, 1995.

[8] B. Bani-Eqbal, 'Enhancing the Speed of Fractal Image Compression', *Optical Engineering, Vol 34, No 6, pp 1705-1710, June 1995.*

[9] E. W. Jacobs, Y. Fisher, R. D. Boss 'Image Compression: A Study of the Iterated Transform Method' *Signal Processing, Vol 29, 1992, pp 251 - 263.*

[10] E.J. Stollnitz, T.D. DeRose, D.H. Salesin, 'Wavelets for Computer Graphics: A Primer', Department of Computer Science and Engineering, University of Washington, Seattle, Technical Report 94-09-11.

[11] N. Ahmed, T. Natarajan and K. R. Rao, 'Discrete Cosine Transform', IEEE Trans Computer, Vol C-23, pp 90-93, Jan 1974.

size	C1	2	1	1	5	2	1	1	1	2	2	1	1
		2	1	1	2	1	1	1	1	1	2	1	1
32	C2	2	2	1	1	1	1	2	1	1	1	1	1
		2	2	1	1	2	2	1	1	1	1	1	2
	C3	2	8	1	2	1	1	1	2	1	1	1	1
		1	2	1	3	1	1	1	2	1	2	3	4
size	C1	10	7	13	8	4	5	7	1	2	9	6	8
		2	1	5	4	1	8	2	3	5	9	4	7
16	C2	28	19	11	14	6	16	20	15	11	16	5	15
		16	10	8	5	29	29	2	7	6	8	14	32
	C3	26	33	10	15	9	8	19	43	8	19	9	11
		10	28	15	21	33	29	18	21	20	36	60	50
size	C1	38	35	35	24	34	28	14	11	28	24	23	16
		23	21	24	24	22	22	20	19	26	36	30	35
8	C2	116	86	46	38	40	57	63	53	39	49	34	56
		47	23	43	44	53	107	29	34	38	51	63	94
	C3	99	123	54	72	47	44	113	142	46	65	59	70
		53	109	67	71	99	108	73	76	80	126	210	175

Table 1: The number of domains of each size and class.

image	size 64	32	16	8 QF	8 HF
lena	2	3	14	34	13
mandrill	1	7	13	90	31
peppers	3	4	17	38	15

Table 2: The total search time per domain size. The longest time is spent searching 8×8 ranges.

C1	3	6	3	2	2	3	2	4	4	4	2	3
	3	5	2	4	2	2	6	4	5	3	4	6
C2	15	13	4	2	4	4	17	12	3	5	7	7
	4	5	5	7	37	41	2	3	7	7	28	24
C3	21	14	7	7	5	4	25	21	8	4	8	8
	6	11	10	6	34	32	8	13	8	18	34	41

Table 3: The average number of domains per class returned by the tree method.

image	algorithm	ratio	time	PSNR
lena	QF	14.9	55	32.3
lena	HF $\beta = 10$		37	31.4
lena	HF $\beta = 20$		36	31.1
mandrill	QF	5.7	119	25.2
mandrill	HF $\beta = 4$		65	23.8
peppers	QF	14.9	66	32.2
peppers	HF $\beta = 10$		42	31.1
combined	QF	6.5	127	26.4
combined	HF $\beta = 10$		71	24.8

Table 4: Comparison of the Hybrid and the Quad-Tree methods. The combined picture is joined from four standard 256×256 images.