Speeding Up Fractal Encoding of Images Using a Block Indexing Technique

Riccardo Distasi[1], Michele Nappi[2], Sergio Vitulano[3]

[1] Istituto per la Ricerca sui Sistemi Informatici Paralleli (IRSIP)
Italian National Research Council (CNR)
Via Pietro Castellino, 111, 80135 Napoli, Italy.
Email: ric@irsip.na.cnr.it
[2] Dipartimento di Informatica ed Applicazioni "R. M. Capocelli",
Università di Salerno, 84081 Baronissi (SA), Italy.
Email: micnap@dia.unisa.it
[3] Istituto di Medicina Interna, Università di Cagliari,
via S. Giorgio, 12, 09124 Cagliari, Italy.

Abstract. This paper presents a novel block indexing technique (BIT) to speed up image fractal encoding. The technique assigns feature vectors to image blocks by establishing an analogy between gray level and mass. The experiments show that the BIT preserves bit rate and SNR values very close to exhaustive search, while providing speedups up to over 100.

1 Introduction

Image coding techniques based on iterated function systems (IFS) are very effective as far as image quality is concerned. In the last few years there has been a vast interest on this topic, as shown by the wealth of papers on the subject [4, 11]. The first fully automated technique based on contractive functions is presented in [6].

The basic idea of IFS-based coding is that of exploiting the similarity among differently sized image blocks. Bigger blocks (domains) are mapped onto smaller blocks (ranges) that must tile the original image, so that the Collage theorem ensures the convergence of the reconstruction within precise error limits [1].

The main problem in IFS-based encoding resides in finding the domain that can best be trasformed into a given range. Since a 512×512 image has several thousand ranges and domains, the process—that must be carried out for each (range, domain) couple—is very expensive computationally.

This shortcoming can be partially overcome by several nonexclusive solutions including parallel programming [3, 5], lossless acceleration techniques [9] and feature extraction schemes. The taxonomy of feature extraction schemes described in [12] sorts the schemes into those based on discrete features, including classification and adaptive clustering, and those based on continous features, such as 1D functional methods and feature vectors.

This paper presents a novel block indexing technique (BIT for short) that proved to be very effective in reducing the computing time while preserving bit rate and SNR values very close to exhaustive search. According to the above taxonomy, the BIT falls into the feature vectors category because domains and ranges are assigned multidimensional continous features. In the following section, we will examine the ideas on which the proposed technique is based; the effectiveness and efficiency of the BIT is then assessed by computer simulation in Section 3.

2 The BIT

The primary goal in devising an indexing strategy is that of obtaining necessary conditions for similitudes among blocks. Establishing an analogy between gray values and weight, if two blocks have the same shape, then they have the same weight distribution. Our aim is that of determining a minimal number of features that are sufficiently representative of the weight distribution.

The first such feature that comes to mind is the position of the mass center (MC), since similar weight distributions have their MCs close to each other. The coordinates of the MC for an $N \times N$ block B are given by

$$x_0 = \frac{1}{M} \sum_{\substack{0 \le i < N \\ 0 \le j < N}} iB(i,j), \qquad y_0 = \frac{1}{M} \sum_{\substack{0 \le i < N \\ 0 \le j < N}} jB(i,j), \tag{1}$$

where $M = \sum_{\substack{0 \le i < N \\ 0 \le j < N}} B(i,j)$ is the block's mass. However, MC proximity is just a necessary condition for block similarity: the position of the MC alone does not adequately describe the block's mass distribution. In order to further characterize the mass distribution, we consider the MCs (x_k, y_k) of all the transformed blocks

$$B^{(k)}(i,j) = \begin{cases} B(i,j) & \text{for } k = 0, \\ \left(B^{(k-1)}(i,j) - \mu_{k-1}\right)^2 & \text{for } k \ge 1, \end{cases} \tag{2}$$

where $\mu_k = M_k/N^2$ is the average mass per pixel in block $B^{(k)}$.

This operation yields a transformation from the space of pixels to a $2n$-dimensional space of features. The value $n = 3$ is a good choice since higher values of k add more overhead than useful information. As a result, we obtain a feature vector

$$(x_0, y_0, x_1, y_1, x_2, y_2)$$

that has an attractively low dimension.

The blocks in the space of features must now be organized to allow the application of a spatial access method (SAM). Among the possible choices for a SAM, there are tree-based methods (k-d-trees and R*-trees are the most efficient [13, 2]), linear methods, which can be based on quadtrees [8] or space-filling curves [7], and, finally, grid-based methods such as the cell technique [8]. The relatively small dimension of feature space suggested that a grid-based SAM was appropriate. In particular, we used a variation of the n-dimensional cell method where the cells are unevenly spaced: as shown in Fig. 1, each (x_k, y_k)-plane is divided into "planecells." The central stripes are thinner as to minimize collisions, since the features are more likely to fall near the center.

0	1	2
3	4	5
6	7	8

Fig. 1. Subdivision of the (x_k, y_k)-plane in 9 planecells. The Cartesian product of n planecells on different planes yields a cell.

The search strategy is as follows: each range becomes the query point $\mathbf{r}$ for the domain that will encode it. The search is limited to a hypersphere $\mathcal{H}_\delta(\mathbf{r})$ of radius δ centered in $\mathbf{r}$, comprising all the points $\mathbf{x}$ such that $d(\mathbf{x}, \mathbf{r}) \leq \delta$, where the distance d between two feature points $\mathbf{v}$ and $\mathbf{w}$ is defined by

$$d\big((v_0, \ldots, v_{2n-1}), (w_0, \ldots, w_{2n-1})\big) = \max_{0 \leq i < 2n} |v_i - w_i|. \tag{3}$$

All the cells that have a nonempty intersection with $\mathcal{H}_\delta(\mathbf{r})$ are then searched for the optimum match.

The radius δ obviously influences the search time: when $\delta = \infty$ we have ordinary exhaustive search, which is very costly but guarantees the best possible quality for a fixed bit rate; when $\delta = 0$, we have a method that, according to [12], falls in the "classification" category, as only one cell is searched for each range; for $0 < \delta < \infty$, we have a true "feature vectors" technique.

The BIT has the advantage of being easily combined with other acceleration methods, such as the lossless technique described in [9] and [10], that can provide an additional speedup factor.

3 Experimental Results

A wide variety of experimental tests has been performed on standard 8-bit 512×512 test images, such as `lena`, `peppers` and `boats`. The experiments, focused on finding out how efficiently the BIT can locate a domain to encode each range with good fidelity, consisted in applying the BIT to adaptive 3-level quadtree-based encoding with blocks of size 4×4, 8×8 and 16×16.

The number of block-to-block comparisons is the main measure of the BIT's effectiveness as an acceleration technique, while the quality of the resulting encoding can be assessed by the peak signal-to-noise ratio (PSNR) between the original image I and the reconstructed image $\hat{I}$, which in dB is given by

$$\text{PSNR} = 10 \log_{10} \frac{S_1 S_2 \cdot 255^2}{\sum_{\substack{0 \leq i < S_1 \\ 0 \leq j < S_2}} \big(I(i,j) - \hat{I}(i,j)\big)^2} \tag{4}$$

for images of size $S_1 \times S_2$. The bit rates express the length of the encoded image in bits per pixel, while the times were taken on an IBM RS/6000 320. The results are summarized by Tables 1–3 and Figs. 2–5.

In the tables, the notation $\mathcal{H}_\delta$ indicates that for each range $\mathbf{r}$ to be encoded we only search inside $\mathcal{H}_\delta(\mathbf{r})$, while the notation $\mathcal{C}_\delta$ denotes a search in its complement, i.e., the set of all domains $\mathbf{d} \notin \mathcal{H}_\delta(\mathbf{r})$.

As can be seen, even for small radii, the BIT's results are very close to those obtained by exhaustive search in terms of both PSNR and bit rate. This is due to the ability of this technique in locating the blocks that are more similar to the target range: the '$\mathcal{C}_\delta$' columns show clearly that when the encoding engine is forced to choose outside of the neighborhood region $\mathcal{H}_\delta$, the performance is worse than exhaustive search under all aspects, even computing time, because the best domains cannot be utilized since they are in $\mathcal{H}_\delta$. It is interesting to note how the much bigger domain pool $\mathcal{C}_\delta$ provides no significant improvement over $\mathcal{H}_\delta$ in terms of encoding quality.

From the point of view of computing time, the slight variation in PSNR and bit rate between $\mathcal{H}_\delta$-search, $\mathcal{C}_\delta$-search and exhaustive search translates into a dramatic improvement—$\mathcal{H}_\delta$-search achieves a speedup of two orders of magnitude. The execution time of the whole encoding process increases nearly linearly with the number of comparisons.

The graph in Fig. 2 shows how the number of comparisons in $\mathcal{H}_\delta$ relates to the attained PSNR. The values are averaged over the 3 test images and are obtained for

Table 1. Results of the tests on image lena.

δ	Comparisons $\times 10^6$		Bit rate (bpp)		PSNR (dB)		Time (sec)	
	$\mathcal{H}_\delta$	$\mathcal{C}_\delta$	$\mathcal{H}_\delta$	$\mathcal{C}_\delta$	$\mathcal{H}_\delta$	$\mathcal{C}_\delta$	$\mathcal{H}_\delta$	$\mathcal{C}_\delta$
0	5.27	577.40	0.411	0.409	32.896	33.339	190	19874
1	11.03	621.20	0.401	0.442	33.066	33.071	391	21508
2	16.37	631.00	0.394	0.452	33.180	33.062	614	22726
3	21.64	631.11	0.391	0.454	33.243	32.954	843	23415
4	27.28	631.30	0.387	0.460	33.321	32.886	1112	24901
∞	540.79	—	0.382	—	33.419	—	21220	—

Table 2. Results of the tests on image peppers.

δ	Comparisons $\times 10^6$		Bit rate (bpp)		PSNR (dB)		Time (sec)	
	$\mathcal{H}_\delta$	$\mathcal{C}_\delta$	$\mathcal{H}_\delta$	$\mathcal{C}_\delta$	$\mathcal{H}_\delta$	$\mathcal{C}_\delta$	$\mathcal{H}_\delta$	$\mathcal{C}_\delta$
0	6.81	741.14	0.532	0.514	31.712	31.991	246	25933
1	13.05	765.65	0.513	0.534	31.844	31.820	449	26379
2	20.20	768.26	0.506	0.541	31.939	31.758	712	27528
3	28.14	770.29	0.502	0.548	31.973	31.734	1021	28752
4	35.55	776.63	0.500	0.558	31.984	31.672	1350	29750
∞	704.13	—	0.487	—	32.060	—	26760	—

δ varying between 0 and 6 (the values of δ are shown beside the data points in the plot). The plotted PSNR values are bounded by the maximum value obtained by exhaustive search. This value, which would correspond to $\delta = \infty$, is plotted as a horizontal asymptote.

Figure 3 illustrates the speedups obtained for the same values of δ. In this case, too, the results are averaged over the 3 test images.

As a last remark about the encoding quality, Fig. 4 shows the bit rates obtained by $\mathcal{H}_\delta$-search and $\mathcal{C}_\delta$-search as the radius δ varies. As can be seen, $\mathcal{H}_\delta$-search has decreasing bit rates, while the opposite holds for $\mathcal{C}_\delta$-search. This behavior can be explained by the fact that, as the hypersphere $\mathcal{H}_\delta(\mathbf{r})$ grows bigger, it is more likely to find good similarities between the range $\mathbf{r}$ and a candidate domain in $\mathcal{H}_\delta(\mathbf{r})$. As a consequence, it is less likely that we have to descend the quadtree by subdividing $\mathbf{r}$.

Finally, Fig. 5 shows an example of the BIT in action, depicting both the query range $\mathbf{r}$ and the domains found by the search. The picture also shows how multidimensional features help in thinning out the number of candidate domains, illustrating the additional domains that would be found for smaller values of the hemidimension n.

Table 3. Results of the tests on image boats.

δ	Comparisons $\times 10^6$		Bit rate (bpp)		PSNR (dB)		Time (sec)	
	$\mathcal{H}_\delta$	$\mathcal{C}_\delta$	$\mathcal{H}_\delta$	$\mathcal{C}_\delta$	$\mathcal{H}_\delta$	$\mathcal{C}_\delta$	$\mathcal{H}_\delta$	$\mathcal{C}_\delta$
0	6.74	950.18	0.687	0.647	32.100	32.537	237	32933
1	15.38	979.68	0.658	0.671	32.301	32.263	511	33478
2	25.03	985.74	0.643	0.682	32.438	32.232	866	34664
3	34.03	984.25	0.639	0.687	32.513	32.221	1224	35357
4	42.12	987.11	0.628	0.692	32.586	32.206	1582	38172
∞	911.15	—	0.618	—	32.661	—	35798	—

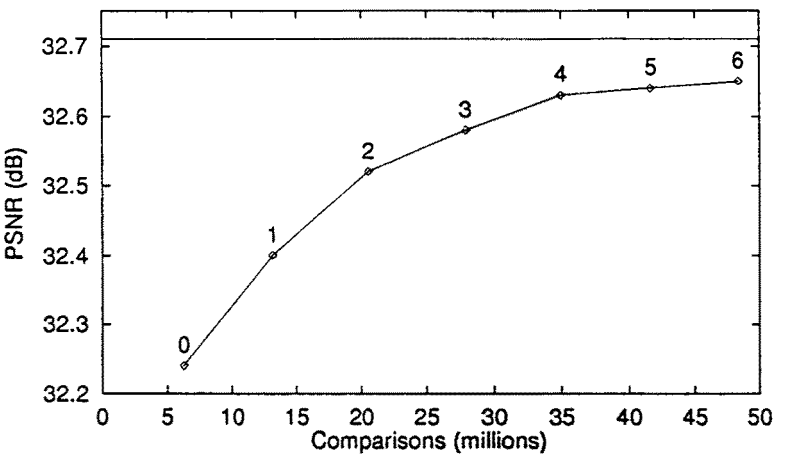

Fig. 2. PSNR as a function of the number of block-to-block comparisons. The horizontal asymptote corresponds to $718.69 \cdot 10^6$ comparisons. The results are averaged over the three test images.

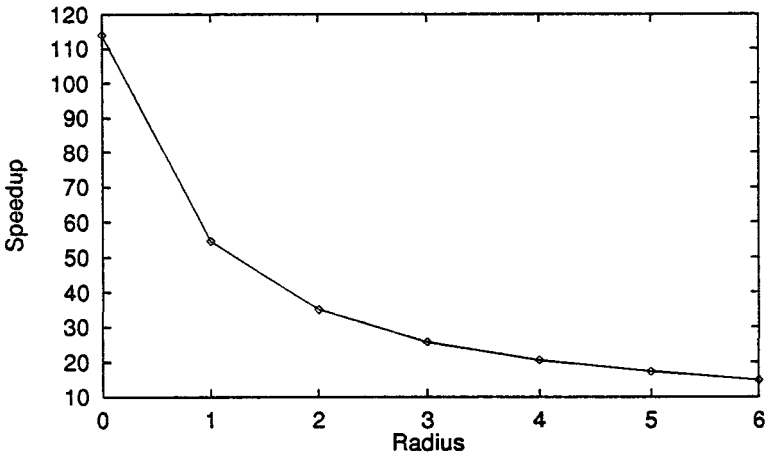

Fig. 3. The speedups obtained as a function of the hypersphere radius δ. The results are averaged over the three test images.

4 Conclusion

This paper introduced a new block indexing technique for fast fractal encoding (BIT) that is meant to reduce the search space. The BIT characterizes a block's gray-level distribution by exploiting the concept of center of mass.

The indexing technique is very attractive: computer simulation has shown that the BIT provides a speedup that reaches two orders of magnitude, while preserving the SNR and bit rate statistics nearly intact.

The computational cost for indexing itself is rather low, especially when compared with state-of-the-art techniques.

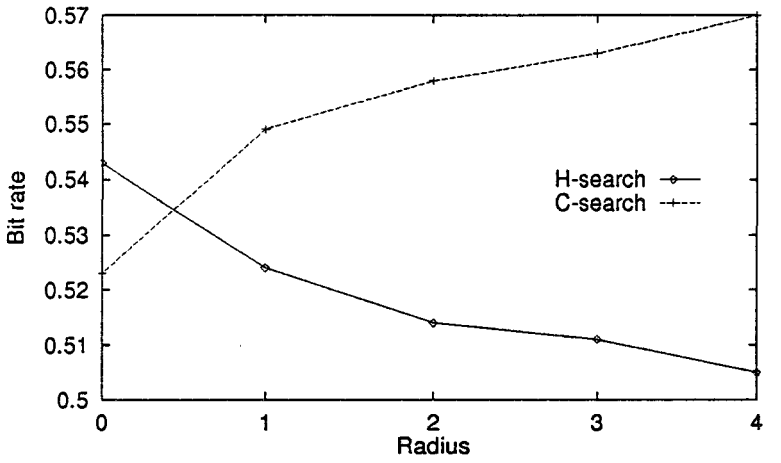

Fig. 4. Bit rates obtained by $\mathcal{H}_\delta$-search and $\mathcal{C}_\delta$-search as a function of the radius δ. The results are averaged over the three test images.

References

1. M. Barsnley. *Fractals everywhere.* Academic Press, New York, 1988.
2. N. Beckmann, H. P. Kriegel, R. Schneider, B. Seeger. The R*-tree: an efficient and robust access method for points and rectangles. *Proc. ACM SIGMOD*, p. 322–331, May 1990.
3. G. Della Vecchia, R. Distasi, M. Nappi, D. Vitulano. A parallel implementation of image coding using linear prediction and iterated function systems. *Lecture Notes in Computer Science*, vol. 1124, pp. 147–150, Springer–Verlag, 1996.
4. Y. Fisher, ed. *Fractal Image Compression: Theory and Applications to Digital Images.* Springer–Verlag, 1994.
5. J. Hämmerle, A. Uhl. Parallel algorithms for fractal image coding on MIMD architectures. *Proc. Int. Conf. on Visual Information Systems*, pp. 182–191, Melbourne, Feb. 1996.
6. A. E. Jacquin. Image coding based on a fractal theory of iterated contractive image transformations. *IEEE Trans. Image Proc.*, vol. 1, pp. 18–30, Jan. 1992.
7. H. V. Jagadish. Linear clustering of objects with multiple attributes. *Proc. ACM SIGMOD*, p. 332–342, Atlantic City, May 1990.
8. H. Samet. *The Design and Analysis of Spatial Data Structures.* Addison–Wesley, 1989.
9. D. Saupe. A new view of fractal image compression as convolution transform coding. *IEEE Signal Proc. Letters* 3, 1996.
10. D. Saupe, H. Hartenstein. Lossless acceleration of fractal image compression by fast convolution. *Proc. IEEE Int. Conf. on Image Proc. (ICIP'96).* Lausanne, Sep. 1996.
11. D. Saupe, R. H. Hamzaoui. A guided tour of the fractal image compression literature. *ACM SIGGRAPH 94 course notes.* Available by ftp from `fidji.informatik.uni-freiburg.de` under `/papers/fractal/Guide.ps.Z`.
12. D. Saupe, R. H. Hamzaoui. Complexity reduction methods for fractal image compression. *Proc. 1994 Conf. on Image Processing: Mathematical Methods and Applications*, J. M. Blackledge ed., Oxford University Press, 1995.
13. J. D. Ullman. *Principles of Database and Knowledge Based Systems.* Computer Science Press, Rockville, MA, 1988.

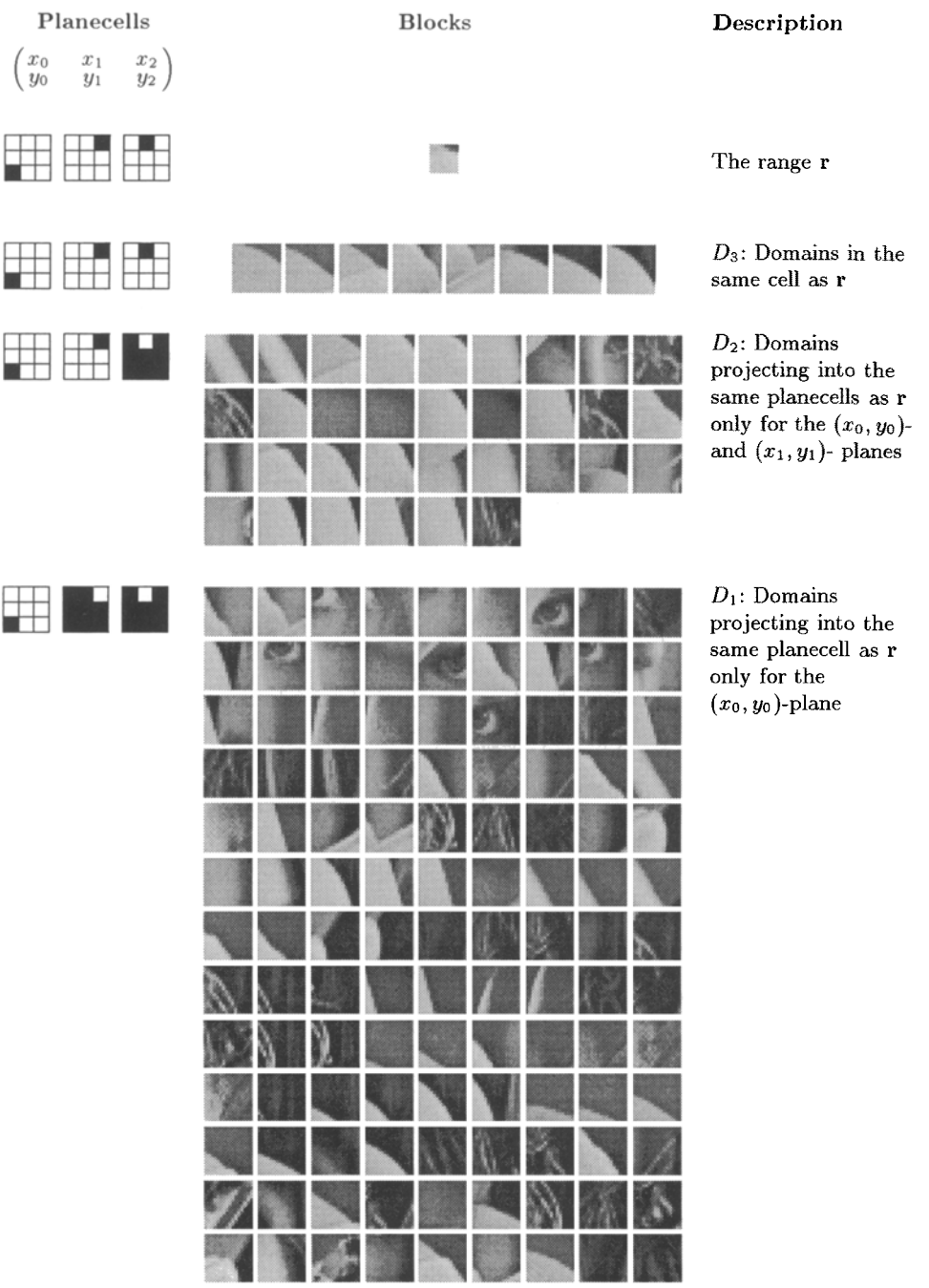

Fig. 5. A sample query performed with radius $\delta = 0$ and hemidimension $n = 3$. Given $\mathbf{r}$ as a query point, the search returns the set D_3 containing the 8 domains in the same cell as $\mathbf{r}$. For $n = 2$ and $n = 1$, the search would return respectively $D_3 \cup D_2$ and $D_3 \cup D_2 \cup D_1$.