

Improved Image Compression Using Fractal Block Coding

Reza Hashemian and Satyendra Marivada
Northern Illinois University
Electrical Engineering
reza@ceet.niu.edu

Abstract

Fractal coding of images is shown to be very efficient compression tool in recent years [1, 4]. It is shown that, because of high redundancy present in images a rule-based technique can be used to iteratively reconstruct an image from limited image patterns. In this presentation we offer another efficient technique based fractal block coding [1] for image compression. It is shown that this technique substantially reduces the memory requirement, and speeds up the reconstruction.

1. Introduction

Image coding through fractal geometry has proven to be very effective, and fractal image compression is a promising field that has gained efficiency in both memory consumption and high speed transmission [1-4].

Fractal geometry, being itself an extension of classical geometry, can mathematically describe an image very much like a simple equation can describe a curve or even a circle. Deterministic fractal images, although visually very complex, represent redundant sectors, in the sense that each sector can be approximated by a transformed copy of itself, or part of itself. The fundamental approach to fractal geometry is through Iterated Function System (IFS) [2], which is also a key element in our development. By using IFS techniques and simple deterministic algorithms, images with spatial complexity can be encoded through certain fractal relationships that describe mapping of blocks of images within themselves [1]. More clearly, this fractal image technique finds similar patterns that exist on different scales/orientations, and at different places in an image. Hence, it allows encoding of an image file by systematically analyzing it and saving a much smaller set of instructions that can be used to iteratively reconstruct the entire image from those patterns. In fact, this compression technique can be thought of a method eliminating as much redundancy as possible.

It was first reported by Michael F. Barnsley [2] that an IFS of an image whose attractor approximates the image could be used to reproduce the original image. Jacquin [1] introduced IFS into local IFS, called *fractal block coding*. He presented a new practical image coding technique, where an image is first partitioned into blocks and each

block is match with another block from the same image after going through some specified contraction and transformations. The Jacquin's method is shown to be a lossy coding technique, much similar to a Vector Quantization (VQ). In a VQ method the codebook carries the basic image patterns where the codebook here is part of the original image itself, and it carries not many but only the required patterns.

The encoding and decoding system of Jacquin's method is illustrated in Fig.1. The system is based on the following two characteristics, i) assumption that image redundancy can be efficiently exploited through regional self-transformation, and ii) approximating an image by a fractal image, recorded from the original image, for reconstruction.



Fig.1 - The encoding and decoding system of Jacquin's method

The main drawback with Jacquin's method is the amount of time it takes to compress the image. For example, according to [4] it takes 3min 32s to compress a Lena image (4X4 block) on a PC with PIII (700 Mhz) processor using Jacquin's method and 1min 23s using the algorithm suggested in [4].

Our objective here is to present a Fractal Block Coding Algorithm that will further improve the coding performance both in terms of memory requirement and processing time. The results obtained show that for the same SNR the compression time, for a 256 X 256 Lena image, is reduced to 30 seconds on a PIII (900 Mhz) processor, and the compression ratio is over 3 times better than that of [4], applied for 4 X 4 range blocks.

The remainder of this paper is organized as follows. Fractal Block Coding [1] is discussed in Section 2. An efficient encoding technique very similar to that suggested by Jacquin is presented in Section 3. An alternative and efficient implementation of the encoding procedure is given in Section 4. Also in Section 4 examples of image reconstruction using the Fractal Block is shown. Here also Coding Algorithm and the results are compared with other methods in the literature. Finally, the conclusion is given in Section 5.

2. Fractal Block Coding

In the proposed method the construction of the fractal code for an image is based on finding a close match between individual blocks within the image itself. For this an image is first segmented into a two-dimensional array of non-overlapping $B \times B$ pixel blocks, called *range blocks*. The range blocks are further classified into two types: *edge* and *shade* types. We use the method suggested in [1] to determine whether a block is an edge block or a shade block. Here, a block is classified as an edge block if the magnitude of the gradient crosses a “threshold” value of 30 for multiple of times (for example, B number of times), otherwise the block is a shade. The edge range blocks are then listed in a *range list*, RL for further processing. For the shade blocks we only report the average pixel intensity of the block, For the edge blocks, however, we need to search for other fractal codes to properly encode the block. In effect, shade blocks represent the more uniform and smoother parts of the image and the average intensity of the block is sufficient for encoding. In the case of an edge block not only the average intensity, but also other fractal information describing the pattern is required. These fractal codes are practically obtained through matching between the patterns of the edge blocks and some other block within the image itself. This leads to choosing another type of blocks in the image, called *domain*. Domain blocks are similar to range blocks but oversized (normally $2B \times 2B$). The pattern matching is done by a series of transformation applied on individual domain blocks. In general, this transformation is composed of two transformations on a domain block: Contraction (geometrical) and Massic [1]. If a match is found for an edge range block, after proper transformation, all we need to transmit (or store) is the code for the transformation and the domain position in the image. At the decoding site, this information is enough to reconstruct the image, and then refine it through iterations.

In practice, we generate all possible $2B \times 2B$ domain blocks where each domain covers four neighboring range blocks, and any two neighboring domain blocks overlap by two range blocks. Next the domain blocks are geometrically contracted (Γ) by a factor of two, given in Eq. (1).

$$\Gamma(\mu)_{i,j} = (\mu_{2i,2j} + \mu_{2i+1,2j} + \mu_{2i,2j+1} + \mu_{2i+1,2j+1})/4 \quad (1)$$

Where, μ is domain block of size $2B \times 2B$, and Γ is the contraction transformation, for $i, j \in \{0, 1, \dots, B-1\}$.

This geometrically contraction is required to match the reduced domain with the $B \times B$ range blocks. To save time and space we only record the domain blocks that pass the “edge” test, as specified for the range blocks.

We then list the edge domain blocks in a *domain list*, DL . A process of fractal matching between a range block and a domain block start by applying Massic transformation T to each domain block and then comparing with the designated range block. In Massic transformation we need to get three parameters for each range block. The first parameter is isometry (*iso*), representing geometrical orientation. The second and third parameters are dynamic range scaling factor α , and luminescent shift β . Now, we only need to transmit the contraction transformation (average intensity, *iso* α and β) and the domain position information for decoding. At the decoding site, using the contraction transformations and the position of the domain blocks accomplishes the reconstruction of the image. The image is then refined through iterations.

To find the *iso* each (edge) range block is divided into four $B/2 \times B/2$ sub-blocks and the average intensity of pixels in each sub-block is calculated. We quantized the average intensity into 8 levels, as shown in Table 1, where each sub-block can get any of the levels. As a result, a total of $8^4 = 4096$ categories are possible, where a block with L_1, L_2, L_3 , and L_4 levels of intensity for its four sub-blocks falls into C category, given by Eq. 2.

Table 1

Levels	0	1	2	3	4	5	6	7
Intensity	0	32	64	96	128	160	192	224
	-	-	-	-	-	-	-	-
	31	63	95	127	159	191	223	255

$$C = 512*L_1 + 64*L_2 + 8*L_3 + L_4 \quad (2)$$

Next, to simplify the matching procedure the range blocks, in RL , are further grouped into different categories, calculated through Eq. 2. Matching categories of edge ranges with those of the edge domains will specifies *iso* for each edge range. We still need to match the dynamic range and the average intensity.

Dynamic range in a block is defined as

$$dr = \text{highest pixel value} - \text{lowest pixel value} + 1,$$

and the dynamic range scaling factor is defined as $\alpha = dr_r / dr_d$, where dr_r and dr_d are the dynamic ranges of the range block and the domain block, respectively. To further simplify the operation α is also quantized to the nearest value of 0.5, 0.6, 0.7, 0.8, 0.9, or 1.0. To match the average intensity we define a luminance shift β as $\beta = avg_r - \alpha(avg_d)$, where avg_r and avg_d are the average pixel intensity for the range and domain blocks, respectively. In the next section we show how the fractal encoding is performed.

3. Encoding of Range Blocks

The encoding process starts by going through the list of the range blocks RL , one block at a time. First, for each range block, γ_i , the corresponding category C_i is calculated (Eq. (2)). The categories are then listed in CL , and all ranges with identical categories are grouped and placed in, say CL_k , entry of CL .

We are now ready to match between edge range and domain blocks. For each range block γ_i , belonging to say CL_k group, we need to find a domain block μ_j such that the error ε , given in Eq. 3, is minimized.

$$\varepsilon = (T(\Gamma(\mu_j)) - \gamma_i) \quad (3)$$

Where, $T(\Gamma(\mu_j)) = iso(\alpha(\Gamma(\mu_j)) + \beta)$.

The search for the minimum error, and overall encoding procedure is described in the following algorithm.

Algorithm:

1. Start with the first edge range blocks located in CL_0 group. Search for a possible domain blocks from DL with both T and Γ transformation applied until one is found with minimum error ε_{\min} (Eq. (3)). The domain block (with coordinates x and y), iso , α and β are then recorded for encoding the range block.
2. Similar procedure continues for every range block in CL_0 group, except that iso does not need to be searched, because all range blocks in a group CL_i share one value of iso .
3. Now, move into the next group, CL_1 , and apply steps 1 and 2 to the range blocks in that group. Continue moving to other groups and repeat steps 1 and 2 until the list is exhausted, and all edge range blocks are encoded.
4. Finally, terminate the encoding process by recording the average intensity, avg , for the remaining range blocks, that are shade blocks.

However, one problem still remains to be addressed. How can we be certain that there is always at least one edge domain block that matches with an edge range block? This specifically might happen for small size images where the edge domain categories are fewer than can be matched with each and every edge range block. Part of the problem can be dealt with by resizing the blocks of changing the threshold value and increasing the iterations, but in general the problem may persist for limited size images creating limited number of edge domain categories.

4. Algorithm Implementation

Conventionally integers are assigned to represent five parameters (x , y , iso , α and β) for each edge range block, and one parameter (avg) for a shade

block. Thus, in case of a black and white image we need ten bytes to encode an edge block and two bytes to encode the intensity of a shade block [4]. However, due to different range of values for different fractal parameters we can definitely customize the data sizes so that the bytes will reduce without losing the accurate information. In our proposed implementation we have exactly applied this scheme and have reduces the data sizes very substantially.

As we discussed earlier, iso takes only 8 values, and α takes only six values $\{0.5, 0.6, 0.7, 0.8, 0.9, 1\}$, so we can record both parameters in 6 bits. Parameter β can take any value from -255 to 255, and hence 9 bits are needed to represent it. Finally, with two bytes assigned to the corresponding domain coordinates, x and y , (for 256X256 images) a total of 4 bytes are all needed to record an edge block. For a shade block one byte is sufficient to record the average intensity (0 to 255). Therefore, in comparison with [4] we get a ratio of 5/12 reductions in data size, and hence reduction in hardware for a pair of edge and shade blocks. In addition, reduction in the encoding data size not only reduces the memory requirement but it also speeds up the transmission as well as the decoding process. Alternatively, we can interpret this reduction in the encoding data, particularly for the edge blocks, as a sign for adopting a more accurate scheme. For instance, with only four bytes needed to encode an edge block we may want to reduce the “threshold” value in the classification of edge blocks, and increase the number of edge blocks in the encoding procedure. As a result, less number of iterations may be necessary and a better, and overall, a more accurate decoding performance may result.




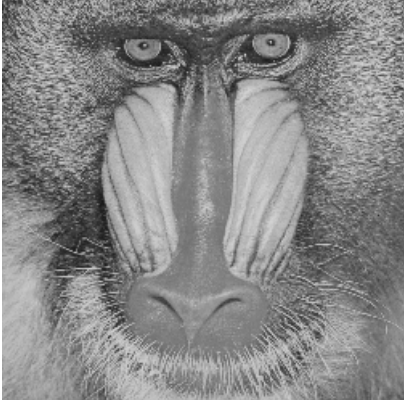
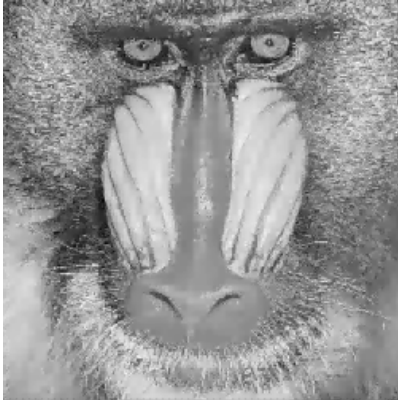
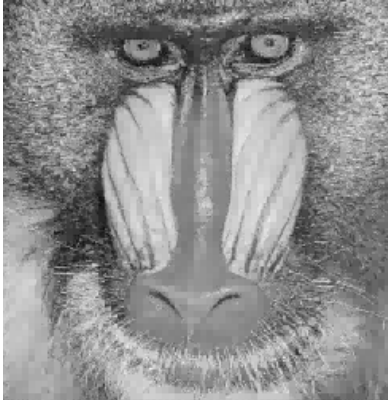
Figures 2 and 3 represent sample images processed using fractal block coding technique with 4 X 4 blocks. The figures compare the overall compression ratios and the decoding time obtained in our proposed method compared with a simulation using the method discussed in [4], as programmed by the authors.

5. Conclusion

It is shown that by adopting a new implementation procedure for fractal encoding of images one can receive a considerable saving in memory requirement without compromising on the image quality at the decoding side. Due to selecting correct sizes for data, and hardware implementation a reduction ratio close to 58% in memory consumption is achieved. In addition, because of smaller volume of data for decoding the reconstruction procedure also becomes faster in the receiving site. The results show that compression rations close to 1:30 with SNR of 45 dB are easily achieved

6. References

- [1]. A.E. Jacquin, "Image coding based on fractal theory of iterated contractive image transformations", IEEE Trans, on Image Proc, vol.1, January 1992.
- [2]. Michael F. Barnsley, *Fractals Everywhere*, New York Academic, second edition.
- [3]. B. Mandelbrot, *The Fractal Geometry of Nature*, San Fransisco, CA, Freeman, 1982.
- [4]. A.Kapoor, K.Arora, A.Jain, and G.P.Kapoor "Stochastic Image Compression Using Fractals", International conference on Information Technology: Coding and Computing (ITCC 2003) Las Vegas, Nevada, Pg. 574-579, 28-30 April 2003.

ORIGINAL IMAGE	Proposed method	Method discussed in [4]
		
Memory in bytes : 266014 Processing time: -- Compression ratio: -- SNR: --	9153 (in binary) Enc.21 sec (Dec. 0.59 sec) 1:9.6 29.83 dB	35280 23 sec. 1:7.54 23.75 dB
		
Memory in bytes : 266014 Processing time: -- Compression ratio: -- PSNR: --	13341 (in binary) Enc. 59 sec. (Dec. 0.81 sec) 1:7.8 23.16 dB	35280 63 sec. 1:5.43 22.42 dB

Figs. 2 and 3 – Fractal compression results of the test images.