A Novel Bit-Weight Based Transformation Selection Algorithm for IFS Based Fractal Image Compression

S.Arockiasamy[1], K.Vivekanandan[2], J Subramani[3]
[1]SNR Sons College, Bharathiar University, Coimbatore, India
[2]Bharathar University, Coimbatore, India
[3]Sri Krishna College of Engineering and Technology, Coimbatore, India

Abstract

Recently, it has been discussed in [1][2][3] the possible optimization strategies for enhancing the speed and performance of **IFS** algorithm for fractal image compression based on a duplicate block removal strategy. Invariably, all the existing fractal image compression algorithms suffer in performance mainly because of the lack of an efficient image classification/sorting method. A new method is proposed for further increase in the performance of IFS by implementing a seed block image sorting strategy for efficient search while selecting suitable fractal seed blocks for transformations during the IFS coding. In this method, the bit weight of an image block is derived for the individual bits of the pixel/gray values. Based on the derived weights, the image blocks are addressed and classified to reduce the search area. After selecting one image block based on their 'Bit Weight' the new block matching operation based on a mathematical model proposed. The experiments with different kinds of images have been done using this algorithm. Further Normal IFS and Bit weight based IFS image classification/sorting algorithms have been compared through time and PSNR.

Key Words: IFS, Fractal Image Compression, Quadtree decomposition, Affine Transformation, RMSE.

1. Introduction

The main aspect of fractal based image coding is to find a suitable Domain block and a transformation for a rough type Range block. The Fractal based compression algorithm [7] reduces the images by only storing the image sub spaces (seed images) and the IFS's needed to reproduce the original. There are different kinds of algorithms available based on the selection of image sub-space. The other features used in the IFS's to encode the image are transformation, rotation and scaling. In this paper, a new idea is explored on two different kinds of algorithms based on Block Sizes.

Eventhough variable block size strategies adopted, the existing IFS based algorithms are very weak in terms of speed and performance. Thus the whole problem can be looked upon as a complex search problem. The common global search mechanism consumes a lot of time, since it takes many geometric transformations [8] and image comparison operation. In this paper, the possibilities of applying new image Block classification/sorting algorithm to find the near optimal solution in fast and efficient manner are analyzed.

A typical fractal image compression algorithm is implemented with various stages as follows:

- Image decomposition stage.

- Separation or creation of Domain Blocks and Range Blocks.
- Selection of a Domain Block for the substitution of Range Block.
- Selecting the appropriate geometrical and intensity transformation.

In all these stages, for comparing an image block with one another, simple Root Mean Square Error **(RMSE)**, Norm and Variance are generally used. These techniques may fail in finding an exactly matching Domain Block image for a Range Block **[1]** image in some circumstances. Moreover, one cannot 'sort' the domain blocks in any manner for finding a match in an efficient manner. Further, it is also not possible to filter a group of domain blocks based on the value of variance or norm.

To find a suitable match for a Range block image, compare it with all the domain block images in all possible transformations. Obviously it will take lot of computation time and also the time will increase further, even if little increase in number of domain blocks or range blocks or the both. There are sophisticated image matching or pattern matching techniques based on statistical **[4]** and AI methods such as Principle Component Analysis (PCA), Neural Networks and Fuzzy Logic are available. Basically these techniques consume a lot of time if we adopt them in fractal IFS algorithm. Even though some of these sophisticated approaches **[5]** leads to high level of compression ratio and superior image quality after decompression, they perform very poor in terms of speed.

2. Reducing the search space.

In a constant block based algorithm **[6]**, the blocks in the original image are mapped with the blocks in another reduced version of the same image as shown in **Figure 2.1.**

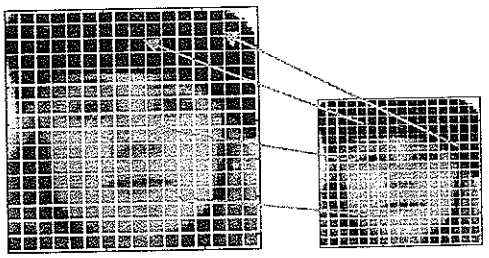


Figure. 2.1. Constant block matching scenario

Here the search space is more for doing the block matching operation. In the proposed algorithm the search space is reduced before the intensive search, which involves a lot of transformation. For reducing the search space, duplicate blocks, are removed from the search space. For finding the similar blocks, a simple block matching **[8]** technique based on **RMSE** value is used. During this process, no geometric or intensity transformation is used. The duplicate block removal phase of this algorithm consumes very low time compared to the overall time for finding **IFS** using all the possible transformations. The following figure **2.2** illustrates the search space reduction scenario.

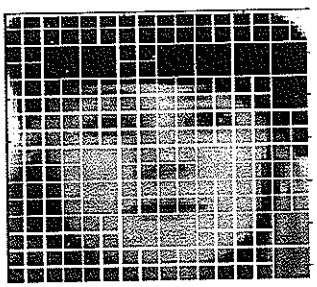


Figure 2.2 After Decomposition

Figure 2.2 shows the original image after quadtree decomposition **[13]**. **Figure 2.3** shows the areas, which are going to be used as seed blocks. In this

implementation, consider the blocks in **figure 2.2** as Domain Blocks and the removed Blocks as Range Blocks. More size reduction can be achieved by dividing the Domain Blocks by considering it as an image and repeating the same procedure in multiple levels recursively.

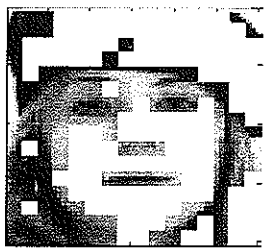


figure 2.3 After Duplicate Block removal

3. Block Classification and Matching Technique

There are some statistical block matching techniques, which are commonly used in fractal image compression algorithms **[9]**. In this paper, some of the standard techniques and their properties, which make them not suitable for image block comparison, are discussed.

Variance

Variance is a measure used to determine, how for the data points differ from the mean. Formula for variance is:

$$S^2 = \frac{1}{N}\sum_{i=1}^{N}(X_i - \overline{X})^2$$

For the block matching operations the variance cannot be simply used and care must be taken while using variance for several block comparisons.

For example, Variance can be used to compare more than one method of block comparisons of the same image. i.e. the minimum variance indicates best matching, under the assumption that the different methods are used for same image comparisons.

In statistical approach, while using variance as a measure to compare several methods, it is assumed that the expected values are one and the same for all these methods. The variance should not be used to match two different images, because of each image having different mean values. For example, let us consider the images represented through following matrices,

$$A = \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

$$B = \begin{pmatrix} 11 & 12 & 13 \\ 14 & 15 & 16 \\ 17 & 18 & 19 \end{pmatrix}$$

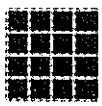
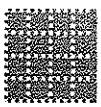
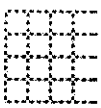
$$C = \begin{pmatrix} 101 & 102 & 103 \\ 104 & 105 & 106 \\ 107 & 108 & 109 \end{pmatrix}$$

All these matrices will give the same variance.

i.e., Variance (A) = 0; Variance(B) = 0; Variance(C) = 0;

The Problem in hand

Consider the following smooth or uniform gray images of dimension 4 x 4 (8 bits per pixel). The figures logically show the magnified view of the actual 4 x 4 Image pixels

(A)  (B)  (C) 

To denote the image mathematically, it is represented as

$$A = ((A_{ij})) \ , \ B = ((B_{ij})) \text{ and } C = ((C_{ij}))$$

We can represent them as a 4 x 4 matrix with its gray levels in equivalent numerical values as:

$$A = ((a_{ij})) \ , \ B = ((b_{ij})) \text{ and } C = ((c_{ij}))$$

$$A = \begin{pmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{pmatrix}$$

$$B = \begin{pmatrix} 100 & 100 & 100 & 100 \\ 100 & 100 & 100 & 100 \\ 100 & 100 & 100 & 100 \\ 100 & 100 & 100 & 100 \end{pmatrix}$$

$$C = \begin{pmatrix} 255 & 255 & 255 & 255 \\ 255 & 255 & 255 & 255 \\ 255 & 255 & 255 & 255 \\ 255 & 255 & 255 & 255 \end{pmatrix}$$

Even though the three images are obviously dissimilar with respect to one another, the individual value of variance of them will be equal to zero.

i.e., Variance(A) = 0; Variance(B) = 0; Variance(C) = 0;

It is to be noted that, the three images are not the same eventhough the variances are the same and are equal to zero. Hence the concept of variance cannot be taken as a useful measure to compare the two images. Suppose, if only one pixel of C changes, then it will change the value of variance significantly.

$$C^* = \begin{pmatrix} 0 & 255 & 255 & 255 \\ 255 & 255 & 255 & 255 \\ 255 & 255 & 255 & 255 \\ 255 & 255 & 255 & 255 \end{pmatrix}$$

Now the Variance(C*) = 3810.05859, Similarly, if we change A(1,1) as 255, and the variance of A* = 3810.05859. Now the images C and C* are not having the same variance. However, we know that the images C and C* are one and the same except the C_{11} *and* C^*_{11}.

The variation of C_{11} *and* C^*_{11} may be due to unavoidable circumstances, like dirt or dust etc. Similarly one can conclude for the case of A and A*. So directly comparing two images based on their individual variance will not be appropriate in these cases.

However, in a case of an image, which is made up of well-distributed pixel values, can be compared with another image, which is visually similar to the previous image. In such cases, the equality in the value of individual variances will reflect the similarity of the images.

That is, the value of variance or norm of an individual image will be meaningful, if and only if it is used with the variance or norm of another image as in the case of RMSE. So sorting or grouping the image in an orderly fashion based on individual variance or norm will not help during searching the images for finding a match [9].

4. The solution Strategy

As discussed earlier, the statistical block matching techniques fail in certain circumstances. So in this section we discuss a novel technique for image block classification **[10]** and matching suitable for fractal image compression.

4.1 Calculating the Bit Weight of an Image

Consider the following 3 x 3 image

$$D = \begin{pmatrix} 234 & 235 & 255 \\ 243 & 242 & 255 \\ 255 & 234 & 255 \end{pmatrix}$$

We can represent the same image in binary as:

$$D = \begin{pmatrix} 11101010 & 11101011 & 11111111 \\ 11110011 & 11110010 & 11111111 \\ 11111111 & 11101010 & 11111111 \end{pmatrix}$$

It is mathematically represented as: $D = ((d_{ij}))_2$, where base '2' is used to represent the binary values. Combine the columns one after another to form single column as follows

$$X_9 = \begin{pmatrix} 11101010 \\ 11110011 \\ 11111111 \\ 11101011 \\ 11110010 \\ 11101010 \\ 11111111 \\ 11111111 \\ 11111111 \end{pmatrix}$$

Which can be obtained as $X((i-1)n + j) = d_{ij}$

Where i, j = 1,2,3...n. (n=3)

The sum of individual columns will be:

$X_{sum} = (99967496)$

The Decimal sum of ones and zeros in a individual column will not be higher than 9 since we are dealing with a 3 x 3 matrix of binary numbers. Since all the bits from LSB to MSB signifies its weight with respect to its bit position, the individual decimal numbers of the resultant number made by the sum of all individual binary bits will also posses the weight with respect to the position of the individual position of the decimal number.

If we consider the bit weights of the individual blocks as a feature and group the blocks accordingly, then finding a matching block during IFS coding can be done in comparatively very small time.

This measure is used to minimize the search space when Domain and Range blocks are compared for matching operation, i.e. for all the range blocks numerical sum is calculated and the blocks are sorted and stored. In the sorted order the range of values alone are taken for the block matching operation with numerical sum of Domain blocks instead of comparing with whole set of Range blocks.

4.2 Calculating the Bit- Weight of a Image Block For Transformation Selection

Consider the following 4 x 4 image

$$I = \begin{pmatrix} 234 & 235 & 255 & 255 \\ 243 & 242 & 255 & 255 \\ 255 & 234 & 255 & 255 \\ 255 & 255 & 255 & 134 \end{pmatrix}$$

The above image can be represented in binary form as:

$$I = \begin{pmatrix} 11101010 & 11101011 & 11111111 & 11111111 \\ 11110011 & 11110010 & 11111111 & 11111111 \\ 11111111 & 11101010 & 11111111 & 11111111 \\ 11111111 & 11111111 & 11111111 & 11101010 \end{pmatrix}$$

The Average Weight of columns and the rows are (four or three ones are considered as 1)

$ColSum = (11111011 \quad 11101010 \quad 11111111 \quad 11111111) / 28.3$

$= (9 \quad 8 \quad 9 \quad 9)$

$RowSum = (11111111 \quad 11111111 \quad 11110010 \quad 11101011) / 28.3$

$= (9 \quad 9 \quad 9 \quad 8)$

The individual binary values are devided by decimal 28.3 to get a digit <= 9

The sum of individual columns will be:

$$TheVerticalBit - Weight = (9899)$$

$$TheHorizontalBit - Weight = (9998)$$

The resultant digit of a indivudul colum or Row will not be higher than 9 . Now based on the column and row bit weights, one can decide the orientation of the blocks. The following figures illustrates the process of finding the suitable transformation. The choice of a particular transform in a given application **[14]** depends on the amount of error that can be tolerated nd the computational resource available. Compression is ackieved during the quantization of the transformed coefficients.

The transformation selection algorithm is a idea implemented through numerical value calculated for various image blocks. This process is done once for each block and the comparision operation **[15]** take place later. The transformation selection algorithm is successfully implemented and the test results also shown here. The results achieved in this algorithm is comparable and considered to be a candidate for the future compression algorithm using fractal techniques. Once the transformation selection **[16]** is over the remaining operation is similar to the existing IFS algorithm only. This process is given below in the form of diagrams.

Finding Suitable Transformation of a Domain Block for a Range Block.

Here, A1, A2 are Vertical Bit-Weights of the images (Range Block & Domain Block) and B1, B2 are Horizontal Bit-Weights of the images. If we consider the first column of square block as one of the range blocks and the next column square blocks as the domain blocks, then the right hand side rectangular boxes shows the suitable transformation for the domain block to match the range block.The above description is indicated in **figure 4.1**

4.3 The Transformation Selection Algorithm

Consider the Vertical and Horizontal bit-weights A1, B1 & A2, B2 of two image blocks I1 & I2. By considering all the above said facts, the following algorithm can be used to find the suitable transformations.

{If A1=A2 then
If (B1=B2) then
Perform No Transformation
Else
Perform Horizontal Flip
}else

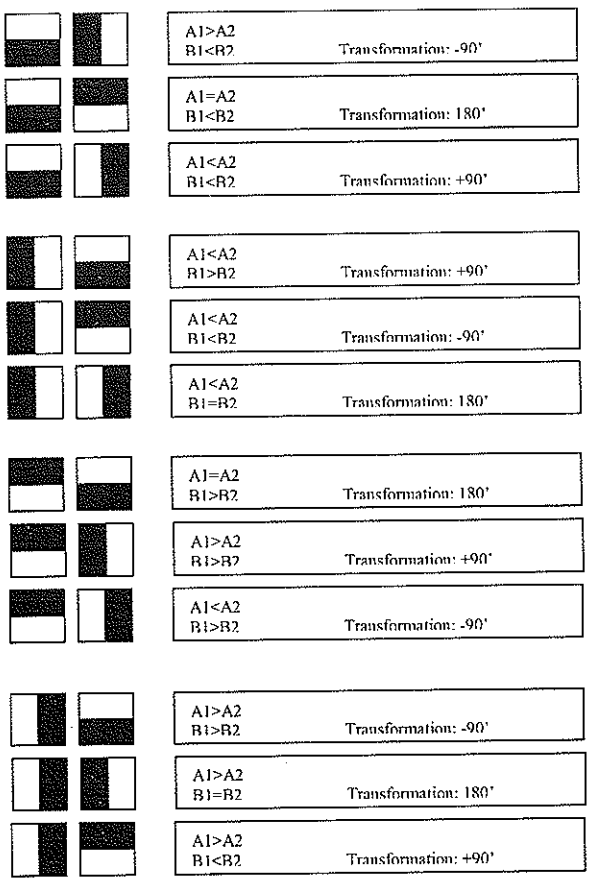


Figure 4.1 Transformation Selection Method

{If (B1=B2) then

Perform Vertical Flip

Else{

If (A1 # A2 and B1 # B2) then

Diagonal Flip Right Top to Left Bottom

or

Diagonal Flip Left Top to Right Bottom}}

5. The proposed mathematical model for the Transformation Selection Operation

Let 'A' be the original image, suppose that the image 'B' is obtained by reading the image 'A' upside down or some other rotations/directions.

For example,

$$\text{let A} = \begin{bmatrix} A_{11} & A_{12} \\ A_{21} & A_{22} \end{bmatrix} \text{ and } \text{B} = \begin{bmatrix} A_{21} & A_{11} \\ A_{22} & A_{12} \end{bmatrix}.$$

It is to be noted that the image 'B' is obtained from 'A' by rotating 90° in clockwise. On the other hand the image 'A' is obtained from 'B', by rotating 90° in anticlockwise. In such situations the bitwise comparisons discussed in [3] is not applicable. In case anyone uses the above method, which will lead to a wrong conclusion that the two images are not one and the same. However, it is known that both the images 'A' and 'B' are the same, but represented with different directions. To overcome such difficulties, an improved version of the algorithm given in [3] is proposed here. The proposed method can be used for comparing the images, which are arranged in any order.

For example, $A = \begin{bmatrix} A_{11} & A_{12} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{31} & A_{32} & A_{33} \end{bmatrix}$ and

$$B = \begin{bmatrix} A_{12} & A_{11} & A_{13} \\ A_{21} & A_{22} & A_{23} \\ A_{33} & A_{31} & A_{32} \end{bmatrix}$$

The steps involved in the proposed method are as follows:

Step 1: Consider the image $A = \sum_{i=1}^{n} {}^{+} \sum_{j=1}^{n} {}^{+} A_{ij}$

Here + is used to denote the summation of the pixels of the image A.

Let $A = ((a_{ij}))$, where a_{ij} is the numerical values of the pixel A_{ij} of the image A.

Step 2: Convert the matrix $A = ((a_{ij}))$ into a column vector $X = (X_i)$ using the formula

$X((i-1)n + j) = a_{ij}$ for all i and j.

Step 3: Arrange the vector X in increasing order and denote it by X_o.

i.e. $X_o(1) \le X_o(2) \le \ldots\ldots \le X_o(n^2)$

where $X_o(i)$ is the i^{th} element of X_o.

Step 4: Consider the image $B = \sum_{i=1}^{n} {}^{+} \sum_{j=1}^{n} {}^{+} B_{ij}$.

Here + is used to denote the summation of the pixels of the image B.

Let $B = ((b_{ij}))$, where b_{ij} is the numerical values of the pixel B_{ij} of the image B

Step 5: Convert the matrix $B = ((b_{ij}))$ into a column vector $Y = (Y_i)$ using the formula

$Y((i-1)n + j) = b_{ij}$ for all i and j.

Step 6: Arrange the vector Yin increasing order and denote it by Y_o.

i.e. $Y_o(1) \le Y_o(2) \le \ldots\ldots \le Y_o(n^2)$

where $Y_o(i)$ is the i^{th} element of Y_o.

Step 7: Compare Xo and Yo. If for every i=1,2,..., n^2

$X_o(i) = Y_o(i)$, conclude that the two images A and B are the same.

Step 8: If $X_o(i) \neq Y_o(i)$ for atleast one 'i', conclude that

the two images are not exactly equal. For further conclusion, one may use the 2nd strategy given in [3], which will give the closeness of the two images.

5.1 Implementation of the Algorithm

1. Input a 256 gray level image 'I' and cropped as a square image, which will be suitable for quadtree decomposition.

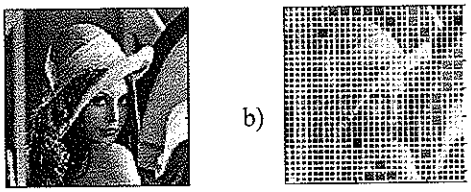


Figure 5.1 Original and decomposed Image

2. The given image into a number of non-overlapping blocks of various sizes based on its features and details using quadtree decomposition.
3. Since the seed blocks of sizes smaller than a particular size is going to be used for coding the image, remove all the blocks greater than that size and visually similar blocks in smaller size groups. This will remove all the visually similar blocks from all size groups by leaving only one seed block.

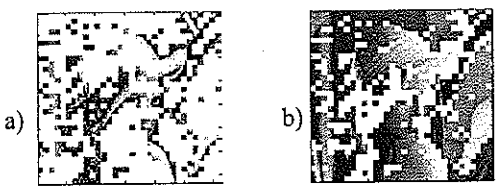


Figure 5.2 Range and Domain blocks

4. Now consider the image as 'D_n' Domain Blocks (seeds) of various sizes and 'R_m' Range blocks of various sizes.
5. Calculate the Bit-Weight of the Domain Blocks (seeds).
6. Select a Range Block and calculate its Bit-Weight.
7. Select a group of Domain Block Which has a Bit-Weights approximately equal the Bit-Weight of the Range Block.
8. Find the matching block based on the calculated bit weight.
9. Calculate the column bit weight and row bit weight.
10. Find the suitable transformation based on column and row bit weights.

    a) Write out the local IFS code.
11. Repeat all the steps from step-6 for each and every range blocks.

a) Normal IFS **b) Bit-weight Method**

Figure 5.3 Image after decompression

6. Experimental Results

The algorithm has been implemented in MATLAB and performed the experiments on a normal 1.7 GHz. Intel Celeran desktop computer, which has 128 MB of RAM. The comparative results for Lena and Nature image is shown in Table 6.1. After the quadtree decomposition, the minimum size of the block was 4 x 4 in size. But, for calculating bit weight of the blocks, only the pixels of 3 x 3 from the original 4 x 4 were used. Because of this, some of the blocks were wrongly coded while decompressing the image.

TABLE 6.1

Time and PSNR value for Lena and Nature image

Range Blocks	Domain Blocks	Normal IFS (Secs.)		Bit-weight Method (Secs.)	
		IFS Time	PSNR	IFS Time	PSNR
943	310	131.6	27.59	80.44	28.91
631	133	44.04	27.83	40.00	30.45

For the simplicity of design, in normal method as well as the proposed Bit-Weight Based method, all the Geometric transformations and intensity transformations **[17]** are not taken into account. In comparing the results of both Normal Approach and Bit-weight based approach, it is found that the proposed method has some interesting qualities.

7. Conclusion and Further Research

This paper deals with two different types of IFS based fractal image compression algorithm using duplicate block removal strategy to reduce the overall search time in Fractal Image Compression **[11]**. An implementation of these ideas has successfully done and tested on MATLAB **[12]**. The attained results are significant and comparable. While comparing with most of the other constant block size or variable block size based implementations, this algorithm achieved very good performance in terms of speed and performance. Further exploration can be made on the ideas outlined in this work to improve the performance of existing fractal image compression algorithms. Other areas that can be explored in future to improve both image quality and processing speed includes trying different error calculations **[16]** between child and parent or Domain and Range blocks other than the Root Mean Square Error estimation. And, more attention can be given for finding suitable transformation without doing all the possible transformations during finding the **IFS.**

References

[1] S. Arockiasamy, K. Vivekanandan, " A new Duplicate Block Removal strategy for Reducing the Search Space to Enhance the Performance of IFS Fractal Image Compression Algorithms, " National Conference on Networking and Multi Agent System, Gobi Arts College, Gobichettipalayam, September 2004.

[2] S. Arockiasamy, K. Vivekanandan, " A proposal for reducing the Search Space by new duplicate Block Removal Strategy to Enhance the Performance of IFS Fractal Image Compression Algorithms,"ACCST Quarterly Research Journal to Arts, Commerce and Computer Science and Technology, May 2005.

[3] S.Arockiasamy, K.Vivekanandan " A Novel Seed Block Image Sorting and Classification Strategy for fast and efficient Block Matching in Fractal Image Compression," International Conference on NUMBER THEORY AND FOURIER TECHNIQUES, SASTRA Deemed University, Kumbakonam, India, December 2004.

[4] Mark nelson, Jean-Loup Gaily, "The Data compression Book," BPB Publications, Second Edition, 1996.

[5] Khalid sayood, " Introduction to Data compression," Harcourt India Pvt., Ltd., Second edition, 1996, 2000 by Academic press.

[6] Barnsley M., "Fractals Everywhere," Academic press, San Diego, 1989.

[7] Yuxuan Ruan and Toh Gua Nge, " Fractal Image Compression," School of Electrical and Electronic Engineering, Nanyang Technological University, Singapore.

[8] Jean Cardinal " Fast Fractal Compression of Grey scale Images, " IEEE Trans. Image Processing, Vol. 10. No. 1, January 2001.

[9] Hsuan T.Chang and Chung J. Kuo, Member, IEEE, "Iteration-Free Fractal Image Coding Based on Efficient Domain Pool Design," IEEE Trans. Image Processing, Vol.9, No.3, March 2000.

[10] Mario Polvere and Michele Nappi, " Speed-Up In Fractal Image Coding: Comparison of Methods," IEEE Trans. on Image Processing, Vol. 9, No.6, June 2000.

[11] Cheung-Ming Lai, Kin-Man Lam, member IEEE, and Wan-Chi Siu, Senior Member, IEEE, " A Fast Fractal Coding based on Kick-Out and Zero Contrast Conditions," IEEE Trans. Image Processing, Vol.2, No.4, April 2002.

[12] Rafael C. Gonzalez, Richard.E.woods and Steven L. Eddins, "Digital Image Processing Using MATLAB, " Pearson Education, Reprint 2004.

[13] B. E. Wohlberg and G. De. Jager, "A Review of Fractal Image coding literature," IEEE Trans. Image Processing, 8 (12): 1716 - 1729, Dec. 1999.

[14] E. W. Jacobs, Y. Fisher, and R. D. Boss, " Image Compression: A study of the Iterated Transform Method, " IEEE Trans. Image Processing, 29(3): 251-263, 1993.

[15] Rafael C. Gonzalez and Richard E. Woods, " Digital Image Processing, " Prentice Hall of India, Tenth Indian reprint 2005.

[16] Wagdy H. Mahoud, Tennessee Technological University, USA, David Jeff Jackson, The University of Alabama, Tuscaloosa, USA, "Hybrid Image Partitioning Algorithms For Fast Fractal Image Compression," IJCA, Vol. 11, No.1, March 2004.

[17] Y. Fisher, Ed., "Fractal Image Compression: Theory and Application," Berlin, Germany: Springer-Verlag, 1992.