Fractal Image Compression Mechanism by Applying Statistical Self-Similarities

Jeet Kumar[1] and Manish Kumar[2]

Department of Computer Application,
Shri Ramswaroop Memorial Group of Professional Colleges,
Lucknow, Uttar Pradesh, India

Abstract

The self-similarities found in the images are of three types. First type is exact self-similarity, second is quasi self-similarity and third is statistical self-similarity. This paper makes use of statistical self-similarity to achieve image compression. Unlike the traditional approaches we do not need two partitions of same image in the form of range blocks and domain blocks; instead a single partition of image is sufficient. This approach divides the code book into two disjoint subsets. One subset contains the blocks in which the pixel values are not much different and satisfying some threshold limit. This subset is the target area for compression. The statistical self-similarity based on the mean value is found among various blocks of this subset. The other subset contains the blocks that do not satisfy the threshold limit and therefore cannot be compressed. The overhead of image partition is halved in this approach; moreover since the target area of compression is separated the procedure is faster than the traditional approaches.

Keywords: *Fractal Image Compression, Exact Self-similarity, Quasi Self-similarity, Statistical Self-similarity, Range Blocks and Domain Blocks.*

1. Introduction

An image can be thought of as a 2-dimensional array of pixels. A single partition of the image is needed in the proposed approach. The given image is partitioned into fix sized square blocks. Now the image is considered as a 2-dimensional array of blocks where each block is itself a 2-dimensional array of pixels.

Traditional approaches for fractal image compression divide the image into range blocks and domain blocks. Two partitions on the same image are required. While proposed approach does not need two partitions instead a single partition serves the purpose. The proposed approach stores the image in the form of two block pools, i.e., 'block pool 0' and 'block pool 1'. 'Block pool 0' contains the blocks for which pixel values are not close to each other and therefore not of interest for this paper. 'Block pool 1' contains all those blocks exhibiting closeness within the block, i.e. the pixel values inside the blocks are not much different. Therefore we focus on the 'block pool 1' for the compression. The mean of all pixel values for each block of 'block pool 1' is calculated. Instead of storing the complete block, only this mean value is stored for each block. If there exist more than one block with same mean value in the 'block pool 1', the mean value is stored once along with all the locations of the block having same mean value. Thus the concept of statistical self-similarity is implemented by storing the same mean value of the block once.

As far as the organization of the paper is concerned, the introduction section introduces the concept and addresses the problems with existing approaches. After this the literature review section carried out the survey of related researches. The next section named as proposed mechanism contains the basic idea in detail along with the illustration and algorithmic implementation of the basic idea. At last conclusion and future scope sections summarize the work done and suggest some extension possibilities in the future.

2. Literature Review

Instead of storing an image bit by bit the idea to store the image in the form of contractive transformation was given by Michael Barnsley in 1988 [1]. Barnsley's graduate student Arnaud Jacquin implemented the first automatic algorithm in software in 1992 [2]. Since then the field of fractal image compression has evolved rapidly. Many ideas have been proposed till date towards the improvement of the image compression with fractal approach but still extensive computation requirement for encoding the image and closeness between domain and range blocks are the major issues. Traditionally an image is partitioned into non-overlapping range blocks and domain blocks (non-overlapping constraint is relaxed in domain blocks). Usually size of the domain blocks is larger than the range blocks to fulfill the contractive requirement. Some research work also advocated domain

blocks of same size as that of range blocks to exploit self-similarity at same scale [3].

In the existing approaches the image is divided into non-overlapping blocks called the range blocks. The size and shape of range blocks may vary to a great extent, but in many approaches square shaped range blocks are preferred. Although various mechanisms have been proposed for image partition, Some approaches like fixed size partition, quad tree partition, horizontal vertical partition and irregular partition fall under right angled partition category while other partition schemes like triangular and polygonal partition can be used [4]. Usually size of the domain blocks is larger than range blocks [5]. But most of the research is focused on the fixed square shaped block of size B×B for range and 2B×2B for domain [6-8]. For each range block, i, every domain block is explored with all possible transformations. Therefore this approach needs complete domain pool searching and applying all the transformations one by one which consumes much time. After this a best matched domain block is selected on the basis of minimum distance [9,10]. At last for each range block the location of the best matched domain block is stored along with the transformation applicable on the particular domain block. Therefore the image is stored as a list of domain block locations and corresponding transformations.

The traditional fractal image compression method described in the previous paragraph is lossy. In fact most of the fractal image compression methods are lossy. Only very few methods are lossless, for example the method given by Korakot Prachumrak et. al. that makes extensive usage of simultaneous equations is lossless [11]. Proposed method gives a simplified algorithm with simple graphical transformations and lesser matching overhead.

3. Proposed Mechanism

3.1 The Basic Idea

The input image of size N×N is considered as a 2-dimensional array of pixels as shown below:

(1,1) , (1,2) , (1,3) , ……, (1,N)
(2,1) , (2,2) , (2,3) , ……, (2,N)
(3,1) , (3,2) , (3,3) , ……, (3,N)
. .
. .
. .
(N,1) , (N,2) , (N,3) , ……., (N,N)

where each location (i,j); 1≤i,j≤N represents a pixel.

The input image is then partitioned into fix square sized blocks of size M×M. Now the image is considered as a 2-dimensional array of size N/M×N/M as shown below:

(1,1) , (1,2) , (1,3) , ……, (1,N/M)
(2,1) , (2,2) , (2,3) , ……, (2,N/M)
(3,1) , (3,2) , (3,3) , ……, (3,N/M)
. .
. .
. .
(N/M,1) , (N/M,2) , (N/M,3) , ……..., (N/M,N/M)

where each location (i,j); 1≤i,j≤N/M represents a block of size M×M.

The key concept is to search such blocks of size M×M in which the values of each pixel is not much different. We now place the mean of all pixel values of that block. We will use a variable T for threshold, i.e., maximum permissible difference between each pair of pixel in that block.

The proposed mechanism stores the compressed image in two disjoint block pools, i.e., block pool 0 and block pool 1. At the time of decompression the image is reconstructed from both block pools. Block pool 0 stores such blocks of the original image where the difference between any pair of pixel is more than the threshold therefore these blocks cannot be compressed. Block pool 1 stores such blocks of the original image where the difference between every pair of pixel is not more than the threshold. Therefore for these blocks of block pool 1 the mean value of all pixels for each block is obtained and stored. Now each block of block pool 1 is represented by these mean values. If the image is partitioned into the blocks of size M×M, we need to store only a single value for each block instead of storing M*M values.

3.2 Illustrative Example

Take an image of size 256×256 (N×N) and blocks of size 8×8 (M×M). If the image of size 256×256 is divided into blocks of size 8×8, we have total (256*256)/(8*8) = 1024 blocks (N^2/M^2). Suppose we have found 800 blocks out of 1024 satisfying the criteria for block pool 1. Now we need to store only 800 values for these 800 blocks instead of storing 8*8*800 = 51200 values. Moreover if the set of these 800 values have redundant entries, we need not to store these values more than once. Therefore we are trying to find statistical self-similar blocks among these 800 blocks of block pool 1.

3.3 The Novel Procedure

The compression procedure takes an N×N image, the block size M and threshold value T as input. The procedure returns ‘block pool 0’ and ‘block pool 1’ along with their sizes. The decompression procedure takes ‘block pool 0’ and ‘block pool 1’ along with their sizes as

input and returns the final reconstructed image. The process described in the previous section can be summarized into following two procedures given in the next two subsections.

3.4 The Compression Procedure

```
Procedure compression (N×N Image I, Block size M,
Threshold T)
begin
        partition the image into blocks of M×M, (M<N);
        set the status of each block as 'Zero' initially;
        counter0=N^2/M^2;
        counter1=0;
        counter2=0;

        for each block Bi with status 'Zero' (i=1 to
        counter0)
        begin
        if the difference between each pair of elements of
        block Bi≤T
        begin
        calculate the mean of all the elements of the block;
        set the status of block as 'One';
        counter1=counter1+1;
        end;
        else
        begin
        add the block to 'block pool 0' along with the
        respective location;
        set the status of each block as 'Two';
        counter2=counter2+1;
        end;
        i=i+1;
        end;

        for each block Bi with status 'One' (i=1 to
        Counter1)
        begin
        set the status of block Bi as 'Two';
        add the mean value of the block along with the
        coordinate of the upper left corner of the block to
        'block pool 1';
        for each block Bj with status 'One' (j=i+1 to
        counter1)
        begin
        if (mean(Bi)==mean(Bj))
        begin
        set the status of block Bj as 'Two';
        if the mean value of Bj is not found in the
        'domain pool 1', then add the mean value of the
        block Bj along with the coordinate of the upper
        left corner of the block to 'block pool 1'
        else
        find the value same as that of mean(Bj) in the
        'block pool 1', and attach the coordinate of the
        upper left corner of the block Bj with that value;
        end;
        j=j+1;
        end;                    //End of inner for loop
        i=i+1;
        end;                    //End of outer for loop
end.                            //End of Procedure compression
```

3.5 The Decompression Procedure

```
Procedure decompression ('Block Pool 0', counter2, Block
Pool 1', counter1)
begin
        for each block Bi of 'Block Pool 0' (i=1 to
        Counter2)
        begin
        place the block Bi from 'Block Pool 0' to the
        appropriate position stored with Bi in the
        reconstructed final image;
        end;
        for each mean value(rounded to integer) for block
        Bi of 'Block Pool 1'(i=1 to Counter1)
        begin
        replicate the mean value to form a block Bi of
        size M×M.
        place the block Bi to the appropriate position(s)
        (stored with mean value of Bi in the 'block pool1')
        in the reconstructed final image;
        end;
end.
```

4. Conclusions

If we increase the block size, the number of blocks for 'block pool 1' will be lesser therefore reducing the target area for compression, but a bigger sized block will be represented by a single mean value therefore achieving more compression per block of 'block pool 1'.
If we increase the threshold limit, T, the number of blocks for 'block pool 1' will be more but the quality of reconstructed image is poorer.

5. Future Scope

The work can be generalized to any arbitrary chosen block size and threshold value as required by different types of images. Moreover the exact self-similarity can be applied to 'block pool 0' to achieve further compression. A

different partition mechanism of the image like quad tree etc. can be used with this approach.

References

[1] Michael Barnsley, "Fractals Everywhere", Academic Press, Inc., 1988.

[2] Arnaud E. Jacquin, "Image Coding Based on a Fractal Theory of Iterated Contractive Image Transformations", IEEE Transactions on Image Processing, Vol.1, No.1, January 1992.

[3] Yao Zhao and Baozong Yuan," A Novel Scheme for Fractal Image Coding", Institute of Information Science Northern Jiaotong University, Beijing 100044, P.R.China, May 2001.

[4] Brendt Wohlberg and Gerhard de Jager, "A Review of the Fractal Image Coding Literature", Member, IEEE, December 1999.

[5] Gaoping Li, "Fast Fractal Image Encoding Based on the Extreme Difference Feature of Normalized Block", College of Computer Science & Technology, Southwest University for Nationalities, Chengdu, China, 2009.

[6] Jinshu Han, "Fast Fractal Image Encoding Based on Local Variances and Genetic Algorithm", Dezhou University, China, 2009.

[7] Ying Zhao, Jing Hu, Dongxiang Chi and Ming Li, "A Novel Fractal Image Coding based on Basis Block Dictionary", School of Electronics and Information, Shanghai dian ji University, Shanghai, China, 2009.

[8] Yang Liu and Jin-guang Sun, "Face Recognition Method Based on FLPP", Liaoning Techinical University, Huludao Liaoning, China, 2010.

[9] D. Loganathanff, J. Amudha and K.M. Mehata", Classification and Feature Vector Techniques to Improve Fractal Image Coding", Electrical and Electronics Engineering, Amrita Institute of Technology and Science, Coimbatore, INDIA, 2003.

[10] Shen Furao and Osamv Hasegawa,"An Effective Fractal Image Coding Method Without Search", Japan, 2004.

[11] K. Prachumrak, A. Hiramatsu, T. Fuchida and H. Nakamura, "Lossless Fractal Image Coding ", Croatia, 2003.