Indian Journal of Fundamental and Applied Life Sciences ISSN: 2231– 6345 (Online)
An Open Access, Online International Journal Available at www.cibtech.org/sp.ed/jls/2015/04/jls.htm
2015 Vol. 5 (S4), pp. 814-818/Zahedi and Dehghani
Research Article

APPLYING A CORRECTIVE SCHEME ON THE DOMAIN POOL TO ACCELERATE AND IMPROVE THE QUALITY OF THE FRACTAL IMAGE COMPRESSION

***Zhaleh Molay Zahedi[1] and Hamid Dehghani[2]**
[1]Department Electronics, Boushehr Islamic Azad University
[2]Malek-Ashtar University of Technology
**Author for Correspondence*

ABSTRACT
Here we offer a corrective domain pool scheme for fractal image compression. This scheme selects the domain pool correctively for each range block, based on the location of that range. Thus, a more effective and smaller domain library is created for each range. This method reduces the computational load and the bits required to store the position of the domain. Results of various experiments on famous images show that this method hasmuch lower distortion rate and computational load.

Keywords: *Fractal Compression, Conventional Domain Pools, Range, Domain Library, Corrective Domain Pool, Reducing the Computational Load*

INTRODUCTION

In (main) fractal image compression, an image is partitioned into a set of non-overlapping blocks of size r × r which are called range blocks and all of possible overlapping blocks of size 2r × 2r, called domain blocks which constitute domain pools. These domains are contracted to accommodate range blocks. This contraction is done by averaging the pixels.

Contracted domains will be used as code book to approximate each range block with a transformation. In addition, this book will be expanded by the inclusion of all rotations and reflections of each domain. Image segmentation, offset and scaling values for each domain range and index with the best match should be saved for image reconstruction. The problem of main method is to calculate domain block seeking with the best matching for each range block. For an image size of $n \times n$, the number of range blocks is $(n/r)^2$ and the number of domain blocks is $(n-2r+1)^2$. Calculation of the best match between a range block and a domain block is $o(r)^2$. If r is constant, the computational complexity is a full seeking of $o(n^4)$.

Because of the huge computational load, encoding phase is time-consuming. Many methods have been proposed to speed up fractal encoding of the image. Some of these methods reduce the number of domains to decrease the computational load according to some rules. Despite the progress made, waste and decreasing in accuracyand integrity are the main disadvantages of this method. Here we present a new method which correctively selects the domain pools for each range block based on the location of the range. This clearly reduces the encoding time in an exciting way and the amount of compression and PSNR of image reconstruction further increase because of a library with more efficient and smaller domain for each range.

Choose a Domain Pool

In conventional methods of selecting domains, sub-squares in an image that upper left corners are located on a network, will be selected as domains. Here network distancing, determines the size of the domain pool (Fisher, 1995) and (Dasgupta *et al.,* 2010). Fisher has described the performance of domain pool and Figure 1 describes this scheme.

This is a 48 × 48 pixel image which the range size is of 4 × 4, the domain size is 8 × 8 and network distancing is 4. Black squares position is upper left corner of the domains. Domain blocks are well distributed over the entire image. In the proposed method, the domains are selected in a concentrated local

area in each range, not the whole image. This local area is called the domain region. The distance between the two adjacent blocks in vertical or horizontal direction is called search phase size. The domain region and the search phase size determine the size of the domains. Figure 2 shows the domain pool selection scheme of the proposed method which the range is 4 × 4 and the domain is 8 × 8, the domain area is a square with sides of length 12 and the search area size is equal to 2. Black squares in domain region are located in the upper left corner of the domain. In this example, the size of the domain pool is equal to 9.

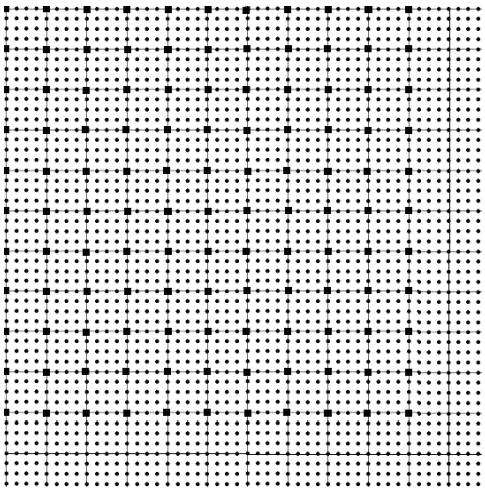

Figure 1: The scope of a common domain pool selection (Fisher, 1995)

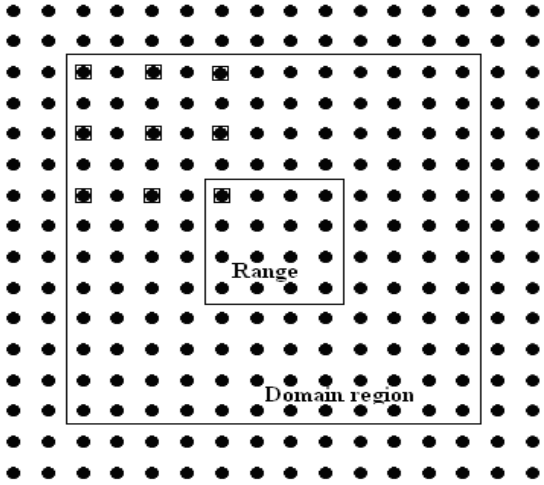

Figure 2: The scope of a domain pool selection in proposed method

The Steps of the Algorithm

The 512 × 512 pepper image is selected as the test image and following domain pools are compared in the experiments.

D_1 Domains are selected as sub-squares of the image which its upper left corner is located on a network and network has distances equal to 64, 32, 16, 8 and 4. For a 512 × 512 image, 4, 6, 8, 10 and 12 bits are required respectively to store the domain index. Consequently, the number of operations to find the best match for each range is 16, 64, 256, 1024 and 4096, respectively.

D_2 Domains are selected as sub-squares from each domain range region. The size of the search phase is set on 1. The number of domain- range comparisons are 64, 256, 1024 and 4096, respectively and they are set for each range by changing the domain.

D_3 Domains are selected as sub-squares from each domain range region. The size of the search is set on 2. The number of domain- range comparisons for each range is set on 64, 256, 1024, 4096, respectively by changing the domain region. In experiments, 512 × 512 pepper image is divided in non-overlapping blocks of size 4 × 4. Domain pool D_1, domain pool D_2 and domain pool D_3 will be used respectively.

Experiments

In the experiments that were done, 512 × 512 pepper image were divided in 4 × 4 non-overlapping range blocks. Domain pool D_1, domain pool D_2 and domain pool D_3 were used according to what was described in step 3.

Table (1), (2) and (3) show PSNR reconstructed images, the number of matching operations for each range and the bits required to store the domain index for various domain pool schemes. The use of the domain pools D_2 and D_3 compared with domain pool D_1 improve PSNR about 3 dB. Chart (1) shows PSNR versus required bits for the 512 × 512 pepper image for various domain pool selection schemes.

Chart (2) shows computational complexity versus PSNR for 512 × 512 pepper image for various domain pool selection schemes.

Figure 3 shows reconstructed image of D_1 (PSNR =28/7) which bits equal to 4 and the number of operations for a range is equal to 16.

Figure 4 shows the reconstructed image of D_2which here bits equal to 4 and the number of operations for a range is equal to 16 (PSNR =31/5414).
Figure 5 shows the reconstructed image of D_3which here bits equal to 4 and the number of operations for a range is equal to 16 (PSNR =31/8007).
Scaling and offset values and domain matching index for each range block to store the scaling and 7 bits for offset will be used. About 11 bits are required to store the scaling and offset for each range.

Table 1: PSNR versus the number of operations and bits for D_1

The number of bits	The number of operations	PSNR (dB)
4	16	28/7019
6	64	31/1142
8	256	32/3269
10	1024	33/9178
12	4096	35/1994

Table (2): PSNR versus the number of operations and bits for D_2

The number of bits	The number of operations	PSNR (dB)
4	16	31/5414
6	64	32/2570
8	256	33/4192
10	1024	34/4737
12	4096	35/3878

Table 3: PSNR versus the number of operations and bits for D_3

The number of bits	The number of operations	PSNR (dB)
4	16	31/8007
6	64	32/9107
8	256	33/8138
10	1024	34/6674
12	4096	35/5566

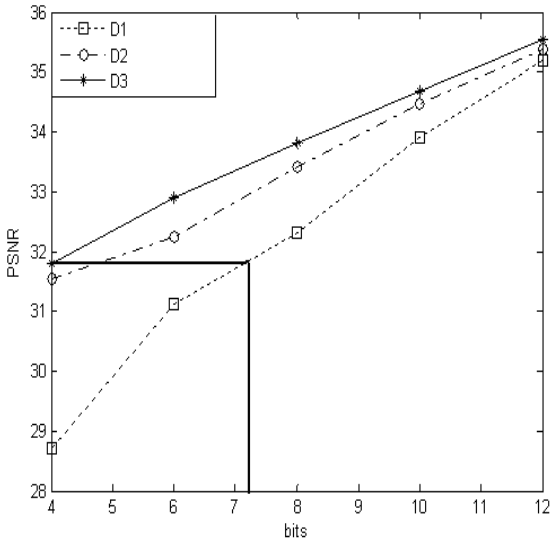

Chart 1: PSNR versus the bits required for the 512 × 512 pepper image for various domain pool schemes

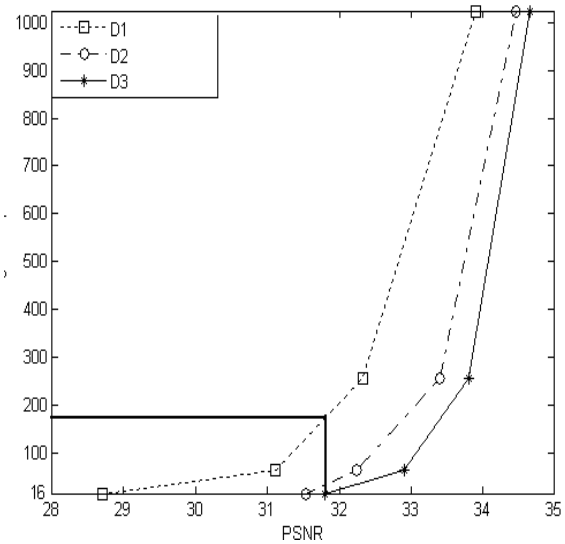

Chart 2: The number of operations for each rangeversus PSNR for the 512 × 512 pepper image for various domain pool schemes

Figure 3: Reconstructed image by domain pool D_1 (PSNR= 28/7019 dB, bits= 4, computation= 16)

Figure 4: Reconstructed image by domain pool D_2 (PSNR= 31/5414 dB, bits= 4, computation= 16)

Figure 5: Reconstructed image by domain pool D_3 (PSNR= 31/8007 dB, bits= 4, computation= 16)

Chart 1 shows that at least 7 bits are necessary for $D_1$to store the domain index but only 4 bits are needed for $D_3$to improve the same PSNR (31/8 dB). So full bits for each block are reduced from 18 to 15.It means about 3 bits will be stored for each range block for the proposed scheme and the compression rate will be increased to 20%.

Chart 2 shows that about 160 comparisons is necessary for D_1, but we need only 16 comparisonsfor $D_3$to achieve the same PSNR (PSNR=31/8 dB). Therefore, the computational load will be reduced to 90%.

CONCLUSION

Here a corrective scheme for the domain pool was offered. Results of experiments on the famous images show that the proposed method will lead to a better performance in terms of distortion rate and much less computational load is needed, which consequently increases the speed of compression.

REFERENCES

Barnsley MF (1993). *Fractal Image Compression* (A. K. Peters, Ltd., Wellesly, MA).

Dipankar Dasgupta, German Hernandez and Fernando Niño (2010). An Evolutionary Algorithm for Fractal Coding of Binary Images. *IEEE Transactions on Evolutionary Computation* **4**(2).

Fisher Y (1995). *Fractal Image Compression: Theory and Applications* (Springer-Verlag, New York, USA).

Hamzaoui R, Saupe D and Hiller M (2001). Distortion minimization with fast local search for fractal image compression. *Journal of Visual Communication and Image Representation* (12) 450–468.

Kin-Wah Ching Eugene and Ghim-Hwee Oug (2006). A Two pass Improved Encoding Scheme for Fractal Image Compression. *Proceedings of the International conference on Computer Graphics, Imaging and Visualization, IEEE* 214.

Lai CM, Lam KM and Siu WC (2009). Improved Searching Scheme for Fractal Image Coding. *Electronics Letters* **38**(25) 153-54.

Lee CK and Lee WK (1998). Fast fractal image block coding based on local variances. *IEEE Transaction on Image Processing* **7**(6) 888-891.

Mitra SK, Murthy CA and Kundu MK (2008). Technique for fractal image compression using genetic algorithm. *IEEE Transaction on Image Processing* **7** 586–592.

Polvere M and Nappi M (2000). Speed-Up in Fractal Image Coding: Comparison of Methods. *IEEE Transactions on Image Processing* **9**(6) 1002-1009.

Tong C and Pi M (2010). Fast Fractal Image compression using Adaptive Search. *IEEE Transactions on Image Processing* **10**(9) 1269- 1277.

Yong Ho Moon, Hyung Soon Kim and Jae Ho Kim (2000). A fast fractal decoding algorithm based on the selection of an initial image. *IEEE Transaction on Image Processing* **9**(5) 941-945.