Fractal Image Compression using Locality Sensitive Hashing

Ashutosh Sharma
Department of Computer Science and Engineering,
National Institute of Technology Calicut, India

V K Govindan
Department of Computer Science and Engineering,
National Institute of Technology Calicut, India

Abstract-- **In Fractal Image Compression the major computational cost goes in comparison of Domain and Range blocks. To reduce the complexity, different researchers attempted to reduce the number of comparisons which ultimately lead to lower the image quality. This paper proposes a low complexity algorithm which is based on locality sensitive hashing. This fast algorithm finds correct Domain block for a Range block with a very high probability, without compromising with image quality. The idea is to preprocess the image and create an adjacency list data structure using hash values generated by Locality Sensitive Hashing when applied on all Domain Blocks. Locality Sensitive Hashing generates similar hash values for similar Blocks. All the Domain Blocks with similar Hash values are kept in same bucket. During Comparison a Range Block is compared with only those blocks which are in the bucket whose hash value is similar to the current Range Block. Complexity of this algorithm is approximately O(n) where n is number of pixels.**

Keyword – Fractal, Range block, Domain block, Locality Sensitive Hashing.

I. INTRODUCTION

To Compress an Image using Fractal Image Compression, Initially divide it in logical non-overlapping blocks of 8x8 pixels called Range Blocks. Then logically divide the image into logical overlapping blocks of 16x16 pixels. Now for each Range block we have to search a Domain block which is structurally similar to it. To find the most accurate and nearest block for a Range block we Down sample the Domain Block to bring it to the size of Range Block and then apply some transformations on Domain Block such as Rotation, Flip etc. For each transformation find the Least Square Distance between Range block and Transformed domain Block. If this distance happens to be less than a threshold value then the affine transformation for this domain block is stored into a compressed file.

Affine Transformations are computed using formula

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} a_i & b_i & 0 \\ c_i & d_i & 0 \\ 0 & 0 & s_i \end{bmatrix} . \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} e_i \\ f_i \\ o_i \end{bmatrix}$$

Where, (x', y', z') represents the transformed point; s_i represents the scaling of contrast; o_i represents the brightness offset (or translation of pixel value). e_i and f_i represent the spatial position translations, a_i and d_i represent the spatial scaling; b_i and c_i represent rotation; and z=f(x, y), the pixel value at (x, y). This Transformation has to be contractive in all three directions x, y, z.

For each Range and Domain Block pair we can compute contrast and brightness using least square regression which has the least root mean square (rms) difference.

The square Euclidean's method used for this purpose is

$$E = \sum_{i=1}^{n} [(s.d_i + o) - r_i]^2$$

Where r_i is the pixel intensity from range Block R_i and d_i is the pixel intensity from domain block D_i, s and o are contrast and brightness respectively. To find minimum value of E we take partial derivative with respect to s and o.

$$\frac{\partial E}{\partial s} = 0, \frac{\partial E}{\partial o} = 0$$

Using the above equation, the value of s and o can be calculated as:

$$s = \frac{[n^2.(\sum_{i=1}^{n} a_i b_i) - (\sum_{i=1}^{n} a_i)(\sum_{i=1}^{n} b_i)]}{[n^2 \sum_{i=1}^{n} a_i^2 - (\sum_{i=1}^{n} a_i)^2]}$$

And

$$o = \frac{[\sum_{i=1}^{n} b_i - s.\sum_{i=1}^{n} a_i]}{n^2}$$

Each transformation requires 8 bit in each direction to locate position of D_i, 7 bits for o_i and 5 bits for s_i and 3 bits to determine Rotation or flip, making it total of 31 bits for a 256x256 image.

II. LITERATURE SURVEY

The Idea of Fractal image compression was first introduced by Michael F. Barnsley in 1988 [1]. His approach was further improved by his PhD student Arnard Jacquin who believed an image can be constructed by smaller part of itself instead of some other image. To implement this he partitioned the image and search for similar blocks in it. This approach uses partitioned iterated function system (PIFS) in which we have a collection of affine transformations which when apply on any set gives back our original image. Since this approach, many approaches have been proposed to enhance and improve fractal image compression.

Abdelrahman Selim et al. [2] proposed another method in which they used spiral architecture instead of square blocks. This approach improved the compression ratio but degraded the image quality. Jianji Wang and Nanning Zheng [3] proposed a way using Pearson's Correlation Coefficient. They used the concept that affine similarity between two blocks in FIC is equal to the absolute value of Pearson's Correlation Coefficient, using this they classified domain blocks. And for each range block, they searched only a domain set whose difference with the Range block was minimum. T.K. Truong at al. [4] proposed a method of spatial correlation in which properties of a block is used to reduce search space. The new method proposed in this paper also reduces search space but does not compromise with image quality. [5] and [6] gives two different ways to search nearest neighbour using Locality sensitive hashing. In [7], Kulis at al. tried to search an image in scalable database of images. In [8], Rajaraman et al. gave complete details of Locality Sensitive hashing with min-hash signature and Jaccard Distance.

The rest of this paper is organized as follows: Section 3 provides the basic idea, principles and various parameters involved in the proposed method. Section 4 provides performance comparison of the proposed approach with other methods, and finally, Section 5 draws the conclusions.

III. PROPOSED APPROACH

In this paper two different techniques are combined one, Fractal Image Compression and another, Locality Sensitive Hashing. Locality sensitive hashing is used in cloud databases to find out duplicate or similar document. The best thing with locality sensitive hashing is that it gives similar hash value for similar documents. To find out similar documents, a database of hash values of all the documents is created. And whenever we need to find out all similar documents to a particular document, say A, then hash values of A and the hash values of all documents from hash database are matched; if they are found similar then these documents are similar.

A. *Basic Idea*

In our approach, we apply Locality Sensitive Hashing on each domain block and all transformations (4 rotation and 2 Flips) of that domain block and for each transformation, save a node which contains coordinates and transformation applied on that domain block in different buckets such that transformation with similar hash values goes in same bucket. Create enough buckets to reduce intra bucket search but do not create too many bucket as they may increase the inter bucket search time. We can use adjacency list as a data structure.

After creating this data structure, for each Range block calculate hash value and search a domain block structurally similar to it, only in the bucket whose hash value is similar to the calculated hash value of Range block and then save the affine transformation in a compressed file.

B. *Locality Sensitive Hashing*

Our approach in Locality Sensitive Hashing is to hash Domain blocks several times so that similar domain blocks will hash to same bucket. To achieve this goal take a family of hash functions called *F*. Create another Family of hash function *H* of functions *h* which itself is a collection of many hash functions from *F*. For a domain block *d*, $hi(d)=[h_1(d), h_2(d), \ldots h_k(d)]$, where $h_1, h_2, \ldots h_k$ are k randomly chosen functions from *F*, a set of hash tables *L* are created for each randomly chosen *h* .In order to classify different domain blocks to different buckets, hash all pixel values from domain block d_i into each of the hash table *L*.

Now while comparing a range block *r*, apply all *L* hash functions *h*. It gives all domain blocks which were hashed to the same bucket. Apply the transformation stored in the node and find root mean square difference, continue the process until we reached a difference which is below some threshold value *t*. After finding the acceptable match save the location of domain block, transformation applied and other parameters. Complexity of creating the hash tables is O(*nkLt*) where n is the number of pixels in a domain block which is constant, k and *L* are also constant in our case, so the complexity of creating hash table depends only on the time taken by hash function. Thus, the complexity of the algorithm is O(t). For details on Locality sensitive hashing read [8].

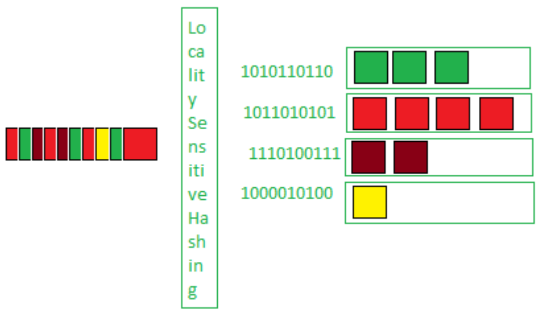

Figure1: Classification of Domain blocks using Locality Sensitive Hashing

C. *Probability of similar Domain blocks in same bucket*

For the successful completion of our algorithm the probability of finding similar block, if exists, must be high. Let P be the probability for a range block that a similar domain block exists. P^k is the probability of matching all hash values in a hash table. $1 - P^k$ is the probability that no hash function gives same value for a hash table. $(1 - P^k)^L$ gives probability that none of the hash value matches for any of the L hash tables. $1 - (1 - P^k)^L$ is the probability that at least one of the hash value matches in any one Table. For example, let the probability P = 0.2 means for a range block there is very less probability that it matches with some domain block. Let k = 9 and L = 3, then Value of $1 - (1 - 0.2^9)^3$ is 0.0000015 which is very less means if two similar blocks do not exists then there is hardly any possibility that they will match.

Similarly, if probability P = 0.9, then value of $1 - (1 - 0.9^9)^3$ is 0.78; means, if similar blocks exists then with a very high probability collision will occur.

D. *Data structure used during pre-processing*

Data structure used during pre-processing is the adjacency list. Front node stores the hash value of the bucket, and the structure of internal node will contain two things first location of the domain block (x and y coordinate) and then the transformation applied (3bits for 8 transformations).

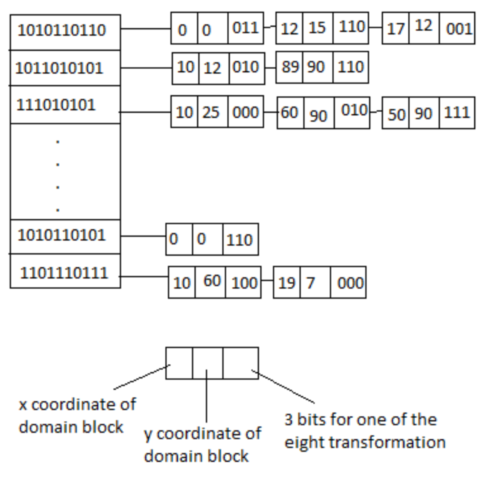

Figure 2: Data structure used for pre-processing

E. *Proposed Algorithms*

Two Algorithms are proposed: first is Pre-processing which creates the data structure to speedup the compression process. After pre-processing, Encoding algorithm saves the affine transformation parameters in the compressed file.

→ Algorithm Pre-processing

Input: Image file.
Output: Adjacency list with similar items in each bucket.

```
Logically divide the image into overlapping Domain blocks of 16 x 16 pixels
For each Domain Block
    Down sample it to 8 x 8 pixels
    Apply all transformations (Rotate 0°, Rotate 90°, Rotate 180°, Rotate 270° with 2 Flips)
        Calculate hash on each transformation
        Save a node (which contains location of domain block and transformation) in a bucket whose hash value is similar to the block otherwise create another bucket and save in that.
.
```

→ Algorithm Encoding Image

Input: Adjacency list from pre-processing algorithm
Output: Compressed image file

```
For each Range block
    Calculate hash value and search a bucket which has similar hash.
    While difference between node and range block is not less than a threshold t
        Jump to Domain block
        Apply transformation stored in the node
        Check rms(Range block, Domain block with transformation) < t
            Save location of domain block, transformation parameter ($s_i$, $o_i$, rotation, flip)
```

IV. PERFORMANCE COMPARISON AND DISCUSSION

Let us consider a square image with $N \times N$ where N is the number of rows or columns in the image, so, the total number of pixels in the image is $P = N^2$. Let the size of Range block is 8×8 and size of Domain Block is 16×16. Therefore, total number of Range block is $N^2/64$ and total number of domain block is $(N-16+1)^2$, this is a very big number. In the worst case, for each range block we have to compare $(N-16+1)^2$ blocks which is ineffective. The above algorithm reduces these numbers of comparisons to a larger extent. According to the above algorithm for a range block, only those domain blocks whose probability of matching is very good is searched. This reduces the time complexity and brings it closer to the number of pixels, i.e. N^2, the time complexity of calculating hash is less than $O(N^2)$. In this algorithm, we should consider the space complexity during preprocessing which is $O(8(N-16+1))$ where 8(transformations) is constant, so the space complexity is $O(N-16+1)$,i.e., the number of Domain blocks.

V. CONCLUSION

This paper proposes a simple method for compressing an image using fractal image compression. Initially hash of all domain blocks are pre-computed and saved as an adjacency list data structure in memory. This will help in reducing the number of comparisons to a larger extent. Time complexity is linear to the number of pixels which is much better than algorithms proposed in [1] and [2]. Future work includes finding a simple hash function which will compute faster than the series of hash functions used in this paper. It is also interesting to investigating techniques to reduce memory requirement during preprocessing.

REFERENCES

[1] Michael F. Barnsley and Alan D. Sloan. 1988. A better way to compress images. *BYTE* 13, 1 (January 1988), 215-223.

[2] Selim, Abdelrahman, Moawad I. Dessouky, Mohiy M. Hadhoud, and Fathi E. Abd El-Samie. "Spiral Fractal Image Compression." *Digital Image Processing*5, no. 12 (2013): 515.

[3] Wang, Jianji, and Nanning Zheng. "A Novel Fractal Image Compression Scheme with Block Classification and Sorting Based on Pearson's Correlation Coefficient." (2013): 1-1.

[4] Truong, T. K., C. M. Kung, J. H. Jeng, and M. L. Hsieh. "Fast fractal image compression using spatial correlation." *Chaos, Solitons& Fractals* 22, no. 5 (2004): 1071-1076.

[5] Slaney, Malcolm, and Michael Casey."Locality-sensitive hashing for finding nearest neighbors [lecture notes]." *Signal Processing Magazine, IEEE* 25, no. 2 (2008): 128-131.

[6] Datar, Mayur, Nicole Immorlica, PiotrIndyk, and Vahab S. Mirrokni. "Locality-sensitive hashing scheme based on p-stable distributions." In *Proceedings of the twentieth annual symposium on Computational geometry*, pp. 253-262.ACM, 2004.

[7] Kulis, Brian, and Kristen Grauman. "Kernelized locality-sensitive hashing for scalable image search."In *Computer Vision, 2009 IEEE 12th International Conference on*, pp. 2130-2137.IEEE, 2009.

[8] Rajaraman, Anand, and Jeffrey David Ullman. *Mining of massive datasets*, pp. 72-94.Cambridge University Press, 2012.