CHAPTER 3

TIME COMPLEXITY REDUCTION METHODS IN FRACTAL IMAGE COMPRESSION

3.1 Introduction

In the previous chapter a literature survey is made with respect to various issues in fractal image compression. In this chapter, we are going to study various time complexity reduction methods in fractal image compression using genetic algorithm, adaptive domain pool scheme and Huffman coding.

3.2 Fractal Image Compression

Fractal image compression is one of the advanced compression techniques conceived by Barnsley and Sloan and the first fully automated fractal image compression algorithm was published by Jacquin in 1989. Fractal image compression offers interesting features like resolution independence, fast decoding and good image quality at low bit-rates which makes it an interesting candidate for off-line applications, e.g. video-on-demand, photo or video, CD-ROMs, etc. Additionally, basic building blocks of fractal compression technology may be used in many other fields, like e.g. feature extraction, image watermarking and motion compensation in video coding. However, there is a main drawback in fractal image compression. The drawback is that the time required for the encoding process. Therefore, an efficient speed up technique is required to speed up the fractal image compression technique.
3.3 Speed up Techniques

One of the main drawbacks in fractal image compression technique is the time required for the encoding process. Therefore, speed up technique is necessary to decrease the time required in the process. However, many methods were introduced to speed up the fractal image compression techniques. Some of the speed up techniques used in fractal image compression are Fisher scheme, Hurtgen scheme, DCT based technique and Feature vector based technique. However, all these all techniques lack in some circumstances. So there is a need to introduce a new technique to speed up the entire process by overcoming the problems occurring in the existing methods.

3.4 Block Diagram of the Proposed Method Using Genetic Algorithm

The fast search strategies using optimization for fractal image compression block diagram are described in Figure 3.1. Here, the fractal image compression is used to speed up compression time using genetic algorithm. The genetic algorithm used here is to reduce the processing time of the fractal image compression. In the genetic algorithm method, the fitness is calculated for each population. If the fitness satisfies the optimization criterion, the output of best individual is taken; otherwise the process repeats till the fitness satisfies for the determination of new population. Thus, the genetic algorithm process is carried out in fractal image compression to speed up the process.

3.4.1 Genetic Algorithm

The Genetic Algorithm (GA) is an optimization strategy based on the Darwinian evolution process. Genetic algorithms (GAs) are computer programs that mimic the processes of biological evolution in order to solve problems and to model evolutionary systems. Genetic algorithms (GAs) are a promising heuristic approach to find near-optimal solutions in large
search spaces. The objective of the GA search is to find a chromosome that has the optimal fitness value. The selection process is the next step. In this step, each chromosome is eliminated or duplicated (one or more times) based on its relative quality. The population size is typically kept constant. Selection is followed by the crossover step. With some probability, some pairs of chromosomes are selected from the current population and some of their corresponding components are exchanged to form two valid chromosomes, which may or may not already be in the current population. After crossover, each string in the population may be mutated with some probability. The mutation process transforms a chromosome into another valid one that may or may not already be in the current population.

The new population is then evaluated. If the stopping criteria have not been met, the new population goes through another cycle (iteration) of selection, crossover, mutation, and evaluation. These cycles continue until one of the stopping criteria is met. The GA is especially appropriate to obtain the global optimization solution of the complex non-linear problem:

**Standard Genetic Algorithm (SGA):**

Step 1: Generate initial population;

Step 2: while number of generations not exhausted do

Step 3: for i = 1 to Population Size do

Step 4: Randomly select two chromosomes and apply the crossover operator;

Step 5: Randomly select one chromosome and apply mutation operator;

Step 6: End for

Step 7: evaluate all the chromosomes in the population and perform selection;

Step 8: end while
Strp 9: report the best chromosome as the final solution.

Figure 3.1: Block diagram of the proposed method using genetic algorithm
3.5 Proposed Time Complexity Reduction Methods

3.5.1 Genetic Algorithm

Genetic algorithms are procedures based on the principles of natural selection and natural genetics that have proved to be very efficient in searching for approximations to global optima in large and complex spaces in relatively short time.

The basic components of GA are:

1. Representation of problem to be solved
2. Genetic operators (selection, crossover, mutation);
3. Fitness function
4. Initialization procedure

The algorithm for fractal image compression using GA is as follows:

1) As the genetic algorithm takes pairs of strings, we create a random number of strings, depending upon our necessity and also note down their decoded values along with setting a maximum allowable generation number.

2) Using the mapping rule, we next find out the corresponding values of the created strings.

3) The fitness function values are found out using these values.

4) The process of reproduction is carried out on the strings to create the mating pool.

5) The process of crossover and mutation is carried out on the strings with probabilities of 0.8 and 0.05 respectively.

6) After the termination criteria are met with, the value of string with minimum fitness function value is considered as optimum value.
The Mean square error (MSE) is considered. The genetic operators are applied to the string having more fitness value. Here the aim is to compare some of the most significant speed-up techniques such as Classification techniques (namely Fisher scheme and Hurtgen scheme), Genetic algorithm schemes, DCT based techniques and feature vector based techniques.

3.6 Performance Comparison

The three significant speed-up techniques are compared based on some of the performance metrics such as compression ratio, speed-up and PSNR.

The formulas for finding Compression Ratio, Speed Up and PSNR are given below:

\[
\text{Compression Ratio } C_R = \frac{\text{Uncompressed Image Size}}{\text{Compressed Image Size}} \quad (3.1)
\]

\[
\text{Speed up } T_S = \frac{\text{Time taken in exhaustive approach}}{\text{Time taken in a particular scheme}} \quad (3.2)
\]

\[
\text{PSNR} = 10 \log_{10} \left( \frac{R^2}{\text{MSE}} \right) \quad (3.3)
\]

Table 3.1 shows the performance comparison of different speed-up techniques on 8 bit/pixel grayscale Lena image of size 256x256.

In this research work, the performance of the proposed FIC using genetic algorithm is compared with standard techniques like Fisher scheme and Hurtgen scheme.

This work compared three of the most significant speed-up techniques. It is found that Genetic Algorithm (GA) provides better compression ratio of 21.7, as compared with Fisher Method with 19.6 and Hurtgen’s Method with 19.6. Further the Genetic Algorithm technique provides improved speed up of 21.09 and as compared to Fisher Method with 7.5 and Hurtgen’s.
Method with 8.03 and Genetic Algorithm technique has achieved PSNR of 26.16 db as compared to Fisher method with 32.9 db and Hurtgen’s method with 33 db. From the above, it is inferred that the Genetic Algorithm performs better than the other existing method. The performance comparison graph is given in Figure 3.2.

Table 3.1: Performance comparison of significant speed-up techniques

<table>
<thead>
<tr>
<th>Methods</th>
<th>Compression Ratio</th>
<th>Speed-up</th>
<th>PSNR(db)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Fisher’s Method</td>
<td>19.6</td>
<td>7.5</td>
<td>32.9</td>
</tr>
<tr>
<td>Hurtgen’s Method</td>
<td>19.6</td>
<td>8.03</td>
<td>33</td>
</tr>
<tr>
<td>Genetic Algorithm</td>
<td>21.7</td>
<td>21.09</td>
<td>26.16</td>
</tr>
</tbody>
</table>

Figure 3.2: Performance Comparison Graph (Compression Ratio, PSNR, Speed Up)
3.7 Block Diagram of Proposed Method Using Adaptive Domain Pool

The fast search strategies using optimization for fractal image compression using adaptive domain pool scheme are described in Figure 3.3. Here, the fractal image compression is speed up using adaptive domain pool method. The domain pool scheme used here is to reduce the processing time of the fractal image compression.

![Block Diagram](image)

**Figure 3.3:** Block diagram of the proposed method using adaptive domain pool scheme
3.7.1 Domain Pool Scheme

Domain pool selection is the second level of decision. This choice depends on the type of partition scheme used, since domain blocks must be transformed to cover range blocks. The domain pool in fractal encoding is similar to the codebook in vector quantization (VQ), referred to as virtual codebook or domain codebook. Global domain pool was the first and simplest type of domain pool, where a fixed domain block is used for all range blocks of image, or for particular class of range blocks in the image. Global domain pool provides satisfactory experimental results. With more advanced applications of fractal compression many researchers observed in the experiments that the results are much better when spatial distance between range block and respective domain block is less. Then, domain pool is generated by following a spiral search path outward from spatial position of range block. Another way used to generate domain pool is masking of range block. The mask is centered at range block. This is known as local domain pool. A more advance type of domain pool is the synthetic codebook; here the domain pool is extracted from low resolution image approximation rather than images itself. Sometimes a combination of domain block mapping and fixed VQ-codebook is used; it is called as hybrid codebook and it provides much better results.

**Figure 3.4:** Decision making levels in fractal encoding.
3.7.2 Adaptive Domain Pool Scheme

In the proposed method, the domains are selected in a local area centered at each range, not over the whole image. This local area is called domain region. The distance between two nearby domain blocks in the horizontal or vertical direction is called search step size. The domain region and the search step size decide the size of the domains. Figure 3.5 shows the domain pool selecting scheme of the proposed method, where the range is 4*4, the domain is 8*8, and domain region is a square with side 12, search step size equal to 2. The positions of black squares in the domain region are the upper-left corners of the domains. The most time consuming part of the encoding procedure is usually the search for finding the best matching domain blocks. During encoding a large pool of image subsets, called domains, has to be searched repeatedly many times, which by far dominates all other computations in the encoding process.

![Adaptive domain pool selecting scheme](image)

**Figure 3.5:** Adaptive domain pool selecting scheme

*Fractal encoder and decoder*

In general method, the rotation and the reflection of each domain blocks are stored in a separate book called codebook. This helps in order to reconstruct the image. In this method, the
scaling and offset values and the match-domain’s index are stored for each range block in order to reconstruct the image.

### 3.7.3 Experimental Results

Lena image of size 512*512 is considered for the fractal image compression. The reconstructed images D1, D2 and D3 are shown in Figure 3.6, 3.7 and 3.8 respectively.

**D1:** The domains are selected as the sub squares of the image whose upper-left corners are positioned on a lattice, and the lattice has spacing equal to 64, 32, 16, 8, and 4 respectively. For the 512*512 image, it is 4, 6, 8, 10, 12 bits respectively to be required to store the domain index. Accordingly, the numbers of operations to find the best match domain are 16, 64, 256, 1024, and 4096 respectively for each range.

![Figure 3.6: Reconstructed image D1](image)

(PSNR: 28.60 dB; bits: 4; Encoding time: 16 sec)

**D2:** The domains are selected as the sub squares of each range’s domain region. Search step sizes are set as 1. The number of domain-range comparison is set as 64, 256, 1024, and 4096 respectively for each range by changing the domain region.
Figure 3.7: Reconstructed image D2

PSNR: 31.68 dB; bits: 4; Encoding time: 16 sec)

**D3:** The domains are selected as the sub squares of each range’s domain region. Search step sizes are set as 2. The number of domain-range comparison is set as 64, 256, 1024, and 4096 respectively for each range by changing the domain region.

Figure 3.8: Reconstructed image D3

(PSNR: 31.97 dB; bits: 4; Encoding time: 16 sec)

The PSNR of the reconstructed image, the number of matching operations for each range, and the bits needed to store the domain index for various domain pool selection schemes. About 3dB is improved for using domain pools D2 and D3 compared with using domain pool D1. The comparison between the PSNR versus bits shows that at least 7 bits are needed for D1 to store the
domain index, but only 4 bits for D3 in order to gain the same PSNR (31.97dB). So, the total bits needed for each range block are reduced from 18 to 15. About 3 bits are saved for each range block for the proposed scheme. The compression rate increases by 20%. In the same way, the comparison between the PSNR versus operations shows that about 160 comparisons are needed for D1, but only 16 for D3 in order to gain the same PSNR (31.97dB). So, the computational load decreases by 90%.

3.8 Block Diagram of the Proposed Method Using Huffman Coding

The fast search strategies using optimization for fractal image compression using Huffman coding are described in Figure 3.9. Here the fractal image compression is speed up using Huffman coding method. The Huffman coding scheme used here is to reduce the processing time of the fractal image compression.

3.8.1 Huffman Coding

Huffman coding is a data compression technique that encodes data (the data string) by creating a code string which represents a fractional value on the number line between 0 and 1. The coding algorithm is symbol wise recursive; i.e., it operates upon and encodes (decodes) one data symbol per iteration or recursion. On each recursion, the algorithm successively partitions an interval of the number line between 0 and 1, and retains one of the partitions as the new interval. Thus, the algorithm successively deals with smaller intervals, and the code string, viewed as a magnitude, lies in each of the nested intervals. The data string is recovered by using magnitude comparisons on the code string to recreate how the encoder must have successively partitioned and retained each nested subinterval. Arithmetic coding differs considerably from the more familiar compression coding techniques, such as prefix (Huffman) codes.
3.8.2 Huffman Coding Algorithm

The technique was developed by David Huffman

The codes generated using this technique are called Huffman codes

![Block diagram of proposed method using huffman coding](image)

**Figure 3.9:** Block diagram of proposed method using huffman coding

These codes are

1. Prefix Codes
2. optimum for a given model

Based on two observations regarding optimum prefix codes
1. In an optimum code, symbols that occur more frequently (have a higher probability of occurrence) will have shorter code words than symbols that occur less frequently.

2. In an optimum code, the two symbols that occur least frequently will have the same length.

Huffman coding is a lossless data compression technique. Huffman coding is based on the frequency of occurrence of a data item i.e. pixel in images. Huffman coding is a popular method for compressing data with variable-length codes. Given a set of data symbols (an alphabet) and their frequencies of occurrence (or, equivalently, their probabilities), the method constructs a set of variable-length code words with the shortest average length and assigns them to the symbols. Huffman coding serves as the basis for several applications implemented on popular platforms. Some programs use just the Huffman method, while others use it as one step in a multistep compression process. It generally produces better codes, and like the Shannon–Fano method, it produces the best variable-length codes when the probabilities of the symbols are negative powers of 2. The main difference between the two methods is that Shannon–Fano constructs its codes from top to bottom (and the bits of each codeword are constructed from left to right), while Huffman constructs a code tree from the bottom up (and the bits of each codeword are constructed from right to left).

The code-length determination is almost unique (except for some special cases) in every technique of Huffman coding. However, it is the coding representation that differs from one technique to another. And it is, in fact, this feature that affects both space requirement and the time for search.
3.8.3 Huffman Coding Compression and Decompression

A. Compression

The Huffman coding technique works by creating a binary tree of nodes. These can be stored in a regular array, the size of which depends on the number of symbols. A node can be either a leaf node or an internal node. Initially, all nodes are leaf nodes, which contain the symbol itself, the weight (frequency of appearance) of the symbol and optionally, a link to a parent node which makes it easy to read the code (in reverse) starting from a leaf node.

![Huffman Tree Diagram](image)

**Figure 3.10:** Huffman codes for equal probability

Internal nodes contain symbol weight, links to two child nodes and the optional link to a parent node. As a common convention, bit '0' represents following the left child and bit '1' represents following the right child. A finished tree has leaf nodes and internal nodes. A Huffman tree that omits unused symbols produces the most optimal code lengths. The process essentially begins with the leaf nodes containing the probabilities of the symbol they represent, and then a new node whose children are the 2 nodes with smallest probability is created, such that the new node's
probability is equal to the sum of the children's probability. With the previous 2 nodes merged into one node, and with the new node being now considered, the procedure is repeated until only one node remains, the Huffman tree.

**B. Decompression**

Generally speaking, the process of decompression is simply a matter of translating the stream of prefix codes to individual byte values; usually by traversing the Huffman tree node by node as each bit is read from the input stream (reaching a leaf node necessarily terminates the search for that particular byte value). Before this can take place, however, the Huffman tree must be somehow reconstructed.

In the simplest case, where character frequencies are fairly predictable, the tree can be pre-constructed (and even statistically adjusted on each compression cycle) and thus reused every time, at the expense of at least some measure of compression efficiency. Otherwise, the information to reconstruct the tree must be sent a priori. A naive approach might be to pretend the frequency count of each character to the compression stream. Unfortunately, the overhead in such a case could amount to several kilobytes, so this method has little practical use. If the data are compressed using canonical encoding, the compression model can be precisely reconstructed with just bits of information. Another method is to simply pretend the Huffman tree, bit by bit, to the output stream.
3.8.4 Screen shots of the Huffman Coding

Figure 3.11 shows the screen shot of decoded image using Huffman coding, and Figure 3.12 shows the graph for Huffman coding.

1. Apply DWT (discrete wave transform) and decode the image by decoding technique

2. Then apply histogram function.

3. Apply Huffman coding

4. Finally validate and get PSNR, Decode time, Encode time value.
Figure 3.12: Graph for Huffman coding

From the Table 3.2 it is found that Huffman coding gives the better performance in terms of PSNR 55.67 (db) and Encoding Time of 5.43 (sec), whereas adaptive domain pool gives highest comparison ratio of 42.85.

Table 3.2: Comparison of PSNR, CR, and Encoding Time

<table>
<thead>
<tr>
<th>Performance Methods</th>
<th>PSNR (db)</th>
<th>Compression Ratio</th>
<th>Encoding Time (sec)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Adaptive domain pool scheme</td>
<td>31.97</td>
<td>42.85</td>
<td>16</td>
</tr>
<tr>
<td>Huffman Coding</td>
<td>55.67</td>
<td>5.61</td>
<td>5.43</td>
</tr>
</tbody>
</table>

Figure 3.13 shows the performance comparison of the two techniques, adaptive domain pool scheme and Huffman Coding in terms of PSNR, Compression Ratio and Speed Up. From the inference, Huffman performs better than the Adaptive domain pool scheme.
3.9 Summary

In this chapter, various time complexity reduction methods like genetic algorithm, Fisher method, Hurtgen method, and also adaptive domain pool scheme and Huffman coding techniques are discussed and comparative merits and demerits of each method are studied. It is found that the Genetic Algorithm performs better than the other methods like Fisher’s method and Hurtgen’s Method. Huffman gives better performance when compared to Adaptive domain pool scheme.