Fast FPGA Implementation of YUV-based Fractal Image Compression

Thai Nam Son
Television Graphic Center,
Vietnam National Television
Hanoi, Vietnam
e-mail: namson@vtv.gov.vn

Thang Manh Hoang, Nguyen Tien Dzung
Hanoi University of Science of Technology
Hanoi, Vietnam
e-mails: {thang, ntdzung}@ieee.org

Nguyen Hoai Giang
Faculty of Electronics,Telecommunication and Information Technology,
Hanoi Open University
e-mail: giangnh@hou.edu.vn

Abstract – **Fractal Image Compression (FIC) method provides a color image compression solution with an extremely high compression ratio and an relatively good PSNR. In this paper, we have developed an efficient approach for a fractal image compression applied to a color image, which utilizes a fractal coding on RGB to YUV color transformation at 4:1:1 sampling mode. The experimental results performed by Fisher's method for a color image have verified the possibility to increase the compression ratio of FIC for color image while retaining an acceptable PSNR. It's purposed to design the low-bit-rate video encoding system by fractal coder/decoder of a color image.**

Keywords- *FIC, Fractal, Color Image Compression.*

I. INTRODUCTION

Fractal image compression (FIC) is based on Partitioned Iterated Function System (PIFS) which uses the self-similarity feature of image to compress images [1]. In fractal image compression, an image is partitioned into a set of non-overlapping blocks called ranges. Another set of larger blocks called domains is used to identyfy the best region in each range, which is most similar to it [2].

In all, the fractal coding is always applied to grey level images. The most straight forward method to encode a color image by gray-level fractal image coding algorithm is to split the RGB color image into three channels, red, green and blue, and compress them separately by treating each color component as a single gray-scale image, the so called three-component Seperated Fractal Coding (SFC)[9].

In place of going for three independent planes, in this paper, a one plane image from the three planes of RGB color image have composed using trichromatic coefficients. This one plane image is then compressed by proposed modified Fractal coding on Spiral Architecture, which minimizes the the number of domain blocks from 343 to 10 using local search to optimize the encoding time[9].

Others way, the color data of seperatly three channels, red, green and blue (RGB components) are transformed to YUV components, to take the advantage of the existing spectral correlation to gain more compression. The test attained trade-off results is PSNR of 33.3 dB with compression ration of 9.72 and encoding time of 128.06 sec [8]. The experimental results of imlementing FIC for color image using variable size of range block method that show the maximum compression ratio of 16.4 and the maximum PSNR of 25.9 dB[6]. The experimental results of FIC for color image using vector quantization method have the maximum compression ratio of 48.87 and the PSNR of 32.12 dB[10].

In our previous work [7], we have implemented FIC on Xilinx Virtex 5 (XUPV5-LX110T) FPGA board to test the possibility of successfully coding and decoding without memory overflow using 64×64 color images at the clock rate of 100MHz. The test attained trade-off results is PSNR of 34.68 dB with compression ratio of 8.86.

The drawback of our previous work is not only limit the size of encoding image (64x64) cause of FPGA board's memory but also the compression ratio is not extremely high. In this paper, we purpose to optimize the compession ratio by transforming the RGB components to YUV components to take the advantages of the floating point for the high dynamic range imaging and adjusting the RMS and the interation of FIC to increase the compression ratio to the higher level while retaining an acceptable PSNR. The FIC implementation for color images on Linux PC with C code. The experimental results showed that FIC algorithms can be optimized for color image compression and also applied to fractal low-bit-rate video compression with the extremely compression ratio.

The paper is organized in four sections. The introduction will be first presented. The summary of theory of Fractal Color Image Compression (FCIC) will be then described following the introduction. The next section presents the proposed FCIC implementation scheme on Linux PC with C code. Finally, the last section discusses about experimental results and performance evaluation followed by the conclusion of the paper.

II. THEORY OF FRACTAL COLOR IMAGE COMPRESSION

A. *Fractal Image Compression Algorithm*

In fractal image compression, an image is partitioned into a set R of n non-overlappig square range blocks. Another set D of $2n \times 2n$ larger square domain block is subsampled by pixel averaging to have the same size as the ranges.

For each $R_i \in R$, this compression method searches through all of D to find a $D_i \in D$ most looks like the range R_i. It also find the best contrast and brightness setting s_i and o_i for the transformation w_i of the mapping from D_i into R_i:

$$w_i \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a_i & b_i \\ c_i & d_i \\ 0 & 0 \end{bmatrix} \tag{1}$$

where s_i controls the contrast and o_i controls the brightness of the transformation, z is the gray level of a pixel at position (x, y) [1].

Therefore, for each $D_i \in D$, s_i and o_i using least square regression are calculated and the D_i with the least rms difference is picked [1]. A set of all w_i called W, is the transformation of the encoding image. The image f that satisfies $f = W(f)$ is the fixed point of W. If W is contractive, f is unique and an approximation of the original image, therefore the decoding process is based on this property.

Since the size of R is very large, the number of comparison is very large, too. Hence, the way to partition image is very important in Fractal Image Compression to reduce this size but still keep the quality of decoding image. There are many ways to partition image to cover the image well. In this paper, we use quadtree partition. In a quadtree partition, a square larger than the minimum size in the image is broken up into four equal-sized sub-squares when its entropy is greater than the entropy threshold or there is no domain to satisfy the rms error tolerance [1].

After all ranges are covered, we do not store all the coefficients in (1). The contrast coefficient s_i and the brightness coefficients o_i are quantized and stored in a fixed number of bits. In this paper, we use 4 bits to store s_i and 7 bits to store o_i. Instead of storing the other coefficients, we store the positions of R_i, both the positions and size of D_i and the orientation involving the rotation and flip information [1].

The decoded image is created by iterating W from an initial image. For each (R_i, D_i) unpacked from the compressed file, domain D_i is sub-sampled by averaging each non-overlapping 2×2 square sub-block. Then, each pixel value in subsampled domain is multiplied by s_i, added to o_i, and placed in the location in the corresponding range R_i determined by the orientation. This process is repeated until the decoding image is fixed (i.e. the fixed point f is approximated).

B. *Fractal Color Image Compression Algorithm*

The process of using Fractal Coding compression algorithm are described as in Figure 1.

Preprocessing Image: Module RGB2YUV implements conversion of RGB color space into YUV color space according to the following equation:

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0,114 \\ -0.1687 & -0.3313 & 0,5 \\ 0.5 & -0.4187 & -0,0813 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix} \tag{2}$$

Sub Sample: Components Y, U, V is sampled by Module subsample according to the 4:1:1 sampling mode. After that, This data is stored in buffer to complete fractal coding.

Buffering: Buffer is used in order to store the components of image including: Y, U, V. These components are utilized in fractal coding.

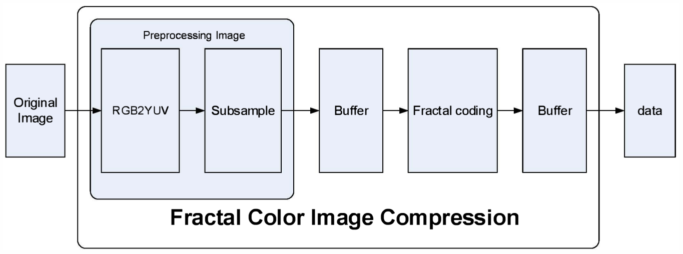


Fig. 1. Fractal Color Image Compression Process

Fractal Coding: This module performs a compression of each component (Y, U, V) using Fractal algorithm. The coding normally use for Grey scale image. The component of image (Y, U, V) are considered as grey scale image, and implement compression each of this component.

First of all, the component Y is encoded by fractal coding, then U and finally V is encoded. In decoding process, order of process is similar to in encoder.

III. Implementation Of FCIC

A. *Processing color images:*

Before the fractal image encoding, a transformation of the color image from RGB to YUV space is implemented. The Y component represents the brightness portion of the image that represent the U and V share together the color information. The illustration of a pixel from the RGB space to the YUV space can be made by a simple matrix multiplication (2). The integer color values are converted to floating-point values and transformation was carried out using floating-point values. The transformed values are rounded and converted back to integer values. The image in YUV space is represented by three unsigned 8-bit values per pixel, the U and V component by the addition of 128 to be adjusted accordingly. The advantage of the representation of the YUV space lies in the separation of the luminance and the color information. Because the human eyes can not detect color changes to the extent that changes in brightness, it is possible the image information without compromising clearly visible in the ratio 4:1:1 to reduce. This means that 2 to 4 brightness information and color information must be stored. This data is already reduced by half. Therefore, the presented algorithm for fractal image coding uses the reduced representation in the ratio 4:1:1 YUV space. From this point, the three components Y, U and V are separated and processed independently of each other, as separate images. The brightness portion of the image (Y) is directly encoded fractal, while the two color components (U and V) are still reduced in the ratio 4:1 respectively before the fractal encoding. This reduction is produced by averaging four pixels.

The encoding of the three components Y, U and V runs completely analogous. Therefore, the algorithm for fractal image encoding, must only handle images of any size, which can be represented by a function f (x, y). Thus, gray level images are encoded in a single pass, while color images require 3 passes. If this chapter is spoken by the coding of an image in the further course, either a halftone image or a component of a color image is intended.

B. *Distribution of the image*

To partition the image, a simple HV-partitioning was chosen. HV-partitioning is more flexible than the square, wherein not only square, but rectangles are permitted. It can thus be edited without problems any image sizes. The decomposition of the ranges can either be done on a fixed schedule or context-sensitively depend on the image content. An advantage of the HV-partitioning is also that the range be reduced by a decomposition is not as strong as in the square partitioning. The output of range squares with size 16 x 16 is elected. Should not completely be represented by an image 16 x 16 range, the remaining areas are filled by rectangles.

Through this sharing scheme arising from squares by 2 squares splits again, this is due to the higher number of squares with pictures is desirable. The management of the range of an image stack oriented happens. Using two stacks can also be potentially required by subdivisions generate a unique order of rank. This is important because the position of a range is not stored due to the saving of space, but is implicitly clear from the memory order of rank. It follows the algorithm of Figure 2. The edited Ranges can be saved directly from the stack down. This result in a sequence without further sorting through the range position can be easily determined again during loading.

C. *Finding the Domains*

The search domain for a range of the search area is limited to the close vicinity of the range. To enable a compact storage format, the relative position of the domain to range is stored in a byte. Normally, therefore, is for the domain a distance of ± 7 points from the upper left corner of the range. However, the range should be near the edge of the picture, the possible domain range is scrolled to always have 225 positions (± 7 points to the upper left corner) are available. In order to find the right domain that allowed pictures to be checked for all possible for the range domains. Since only rectangular or square Ranges occur through the HV used simple methods to check every 4 or 8 illustrations.

D. *Format a memory range*

To obtain the highest possible degree of compression of the fractal coded image, expresses a compact storage format was developed, with which can each range with the corresponding figure in store only 3.5 or 4 bytes. As mentioned above, must be stored by each range, only the relative position of a domain, the size, and the parameters s and replication style o. The absolute position of the range can be determined from the memory of the sequence.

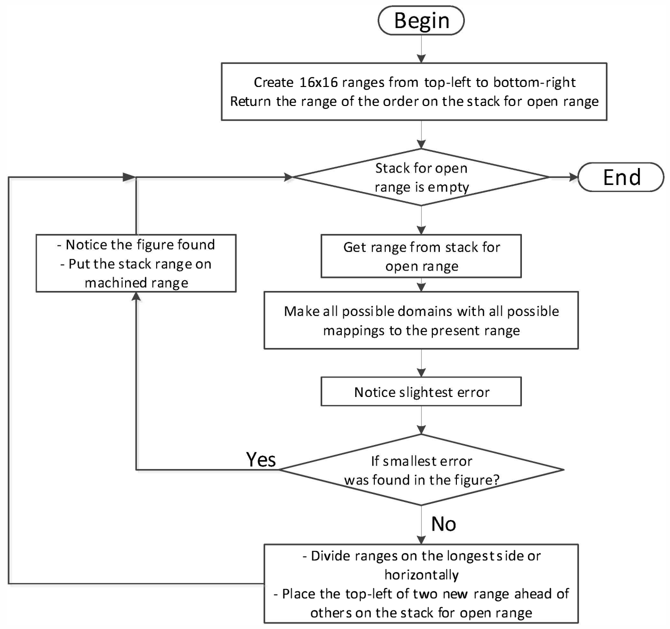

Begin
Create 16x16 ranges from top-left to bottom-right
Return the range of the order on the stack for open range
Stack for open range is empty
End
- Notice the figure found
- Put the stack range on machined range
Get range from stack for open range
Make all possible domains with all possible mappings to the present range
Notice slightest error
Yes
If smallest error was found in the figure?
No
- Divide ranges on the longest side or horizontally
- Place the top-left of two new range ahead of others on the stack for open range

Fig. 2. Algorithm for fractal image encoding

- Storage of the size range

A range may have any size between 1x1 and 16x16. Therefore, there would be actually a whole byte to be needed to store the size. But as the output square range as 16x16 will possibly be generated only and in the periphery of the image, may appear crooked output variables, the range size can be accommodated in the normal case in half a byte. By using simple HV-partitioning scheme that always divides horizontally to range on the longer side, or may consist of ranges of size 16x16 by division only ranges of sizes 16×8, 8×8, 8×4, 4×4, 4x2, 2x2, 2x1 and 1x1 occur. These 9 different sizes can be accommodated by simple numbering easily in half a byte. Ranges which have arisen from a different output of the 16x16, are stored in all the bytes. It is each a half bytes are available, allowing all sorts of sizes between 1x1 and 16x16 for the width and height of the range.

From other output sizes as 16 x 16 might arise over the division range in one of the 9 standard sizes. However, in order not to be stored in addition to whether the present range has standard size or not, the size of each rank, which arose from a different starting size of 16x16, always stored in a whole byte. This simplifies the loading process considerably, since before reading it is already known that the size of the next range in a half or a whole byte is coded.

- Storage of the position domain

As mentioned in Section III.C, the relative position of the domain to the range by restricting the search area can save on 225 possible positions in 1 byte. Because of the special cases in the border area, the offset of each domain to the upper left corner of the search square is stored. Since the search square having an edge length of 15 points, a half byte is sufficient for each dimension. The absolute position of the domain can be so determined as follows: First the search square is determined from the range position; By adding the offset of the domain to the upper left corner of the search square, we obtain the absolute position of the domain.

- *Storage of the imaging parameters*

Parameter s of the image is a floating point number in the interval [-1.2, 1.2]. It is possible to accommodate this value without much loss of quality in 5 bits. Rounding the output to 2 decimal places, multiply by 100 and divide by 8, can achieve this.

- *Storage of Replication style*

Since there are only 4, and for rectangular ranges in square only 8 ways to map the domain to the range, the picture style can be accommodated in 3 bits. It makes sense to the imaging parameters s and accommodate the picture style in a byte.

E. Storage format of an entire image

- *Header*

To distinguish the different types of images and storing the image size of each file, an identical head is placed at the top. The first contains a "magic number" that the type of image stored in 2 bytes. Below the image width and height is stored in 2 bytes each. Since these values are considered in the present form as a normal integer, the maximum encodable image size is therefore 32767x32767. This should be sufficient for all cases occurring in practice. At the end of the head is followed by information that is the average aberrations V of all the range indicates. This error is weighted by the size range, so that a rough evaluation of the image quality is made possible.

- *Order of range*

By judicious choice of the encoding algorithm already created a suitable storage order of range. The rank can be written directly from the stack range for machined down. In particular, no expensive sort operations are required, so the save operation falls in the fractal encoding of image time negligible.

- *Bit shift*

Since the range has a length of 3.5 or 4 bytes, is in consecutive range possibly a shift of 0.5 bytes necessary because in a file only whole bytes can be stored. The optimal use of all data written to the file bytes is achieved by a ring buffer.

This ring buffer has space for two ranges of the full length, which are 2 * 4 = 8 bytes. The ranges are not directly written to the file, but always sent to the ring buffer. The ring buffer pushes each rank together as tightly as possible, and stores all full bytes. At the end of each image is always an "empty" range with 4 bytes in length or 4.5 as the end marker so that all bytes are padded and can be saved.

- *Image Types*

Between gray-scale images and color images are hardly any differences in storage. The color image is in contrast to the gray-scale image of three independently processed sub-images (Y, U and V), which are simply written one after the other in the same file. All images receive a mark out analogously, so that always the same algorithm can be used. The smaller size of the U-and V-component can be easily calculated from the size of the Y component.

F. Loading the IFS

If a fractal-coded image decoding again be loaded into memory, has the absolute position of each range be restored from the given order of rank, and the image size. Since the division of the image in range on a fixed schedule was performed, a recursive loading algorithm can be used, which is very similar to the encoding algorithm:

First, the image size of the header data is read. Now the picture is by rank, the maximum size of 16 x 16 is partitioned. These ranges are, however, not generated, but loaded using a recursive method of loading from the file. It is tested whether the loaded range represents the desired range. This can be done using the stored size range. Should this be the case, the range can be generated with the image data in place and continue with the next range. Is read from the file range to be small, however, the original range was divided at this point into smaller ranges. By sharing the same scheme as in the coding and recursive function call, the desired range position is obtained if and only if for the first time by the division produces a range of data read from the file size. Now, the next range can be read from the file. This range fits into the chosen layout, since the same allocation scheme was used in the encoding.

G. Decoding

An iteration in the decoding consists of the unique design of the figure from the respective domain for all ranges. Since a picture is always displayed in another, the order of range is irrelevant for this purpose. To maintain the existing order on the stack anyway, is always carried out in the present implementation, an even number of iterations. First, the ranges are successively taken from the output stack, run each figure and set the rank to an intermediate stack. During the next iteration of this process is reversed, so that the range is placed back in the old order to the output stack. For the very first iteration, a small optimization has been introduced:

$$w\left(\begin{pmatrix} x \\ y \\ z \end{pmatrix}\right) = \begin{bmatrix} a & b & 0 \\ c & d & 0 \\ 0 & 0 & d \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} e \\ f \\ o \end{bmatrix} \tag{3}$$

Since the output file is empty, it must at a picture of the kind only the addition to be performed because the multiplication by zero (empty output image) is zero again anyway. It arises after the first iteration of an image in which all rank with the corresponding parameters o the figure are filled. By directly filling the range of the corresponding o parameters instead of carrying out the pictures during the first iteration, sometimes is saved. The number of iterations depends on, among other things, by the image. 12 iterations are usually sufficient, but some images have convergence problems in the amount of color, so that further iterations are needed to remove all of the color error.

After the last iteration large ranges can be smoothed in brightness in the image. Since the smoothing only for large rank is useful to an 8x8 range was chosen as the minimum size for the smoothing.

H. *Magnification of Fractal encoded images*

There is an interesting feature of fractal encoded images in the ability to create enlargements without coarse grid of pixels. The generation of enlargement is very simple: if the positions and sizes of the ranges and domains entsprew the overall image will be scaled accordingly, the fractal image can be decoded normally in the new resolution. An enlargement by a factor of 4, for example, thus requiring only multiplication of the x-and y-position as well as the width and height of all ranges and all 4 domains. The resulting magnification is amazingly good and surpasses in quality often even the normally enlarged original image.

IV. Experimental Results

In order to evaluate the performance of the proposed implementation, a 24-bit Lena's image with size of 512×512 has been used in the experiments. Fractal coding schemes are performed at sampling modes of 4:1:1 respectively, and the full searching mode have been set up to determine the optimized compression ratio. The results of PSNR and compression ratios for Lena's image are shown in Table I for different interation modes.

TABLE I. FCIC's Results of Lena's Image

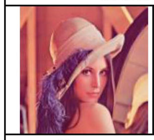
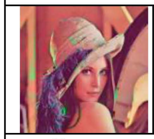
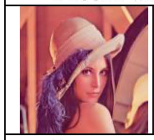
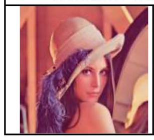
JPEG Encoding		Fractal Encoding	
	En.Time: 0.005s Size: 1707 bytes C.Ratio: 7.23 PSNR: 29.01		En.Time: 0.200s Size: 1526 bytes C.Ratio: 8.09
Decode with 2 Iteration		Decode with 3 Iteration	
	De.Time: 0.062s PSNR: 22.0249		De.Time:0.077s PSNR: 28.5723
Decode with 10 Iteration		Decode with 20 Iteration	
	De.Time:0.183s PSNR: 31.7797		De.Time:0.320s PSNR: 31.7981
Decode with 30 Iteration		Decode with 40 Iteration	
	De.Time: 0.465s PSNR: 31.7986		De.Time: 0.614s PSNR: 31.7986

TABLE II. FCIC's Results of Parrot's Image

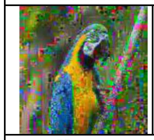
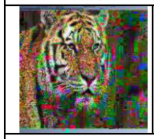
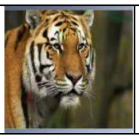
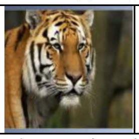
JPEG Encoding		Fractal Encoding	
	En.Time: 0.018s Size: 25331 bytes C.Ratio: 31.05 PSNR: 36.6919		En.Time: 4.425s Size: 18291 bytes C.Ratio: 43.00
Decode with 1 Iteration		Decode with 3 Iteration	
	De.Time: 0.038s PSNR: can not estimated		De.Time: 0.069s PSNR: 32.3455
Decode with 5 Iteration		Decode with 10 Iteration	
	De.Time: 0.103s PSNR: 33.7592		De.Time: 0.176s PSNR: 33.7861
Decode with 20 Iteration		Decode with 25 Iteration	
	De.Time: 0.304s PSNR: 33.7861		De.Time: 0.416s PSNR: 33.7861

TABLE III. FCIC's Results of One-tiger's Image

JPEG Encoding		Fractal Encoding	
	En.Time: 0.048s Size: 49,124 bytes C.Ratio: 16.01 PSNR: 33.763		En.Time: 7.389s Size: 45,802 bytes C.Ratio: 17.17
Decode with 1 Iteration		Decode with 2 Iteration	
	De.Time: 0.044s PSNR: can not estimated		De.Time: 0.058s PSNR: 23.4595
Decode with 3 Iteration		Decode with 5 Iteration	
	De.Time: 0.073s PSNR: 29.7412		De.Time: 0.101s PSNR: 30.7461

The experimental results have implied the trade-off between decoding time and PSNR should be taken into account for different compression scenarios. In order to get more experimental results, some more 24bit color images with size of 512x512 are implemented. The Table II show the experimental result for Parrot's image, Table III for One-tiger's image and Table IV for Two-tiger's image.

Since JPEG is well-known compression standard, for performance evaluation, the quantum table of JPEG method has been utilized and set to 1:1024 for maximal compression ratio. As illiustrated in Table I, II, II and IV, the performance comparison in terms of PSNR and compression ratios of the two methods in different interation modes have shown that the fractal coding schemes could out-performed similar PSNR and the higher compression ratio, even better in Parrot's image at 4:1:1 sampling mode.

TABLE IV. FCIC'S RESULTS OF TWO-TIGER'S IMAGE

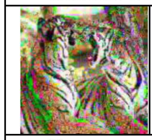
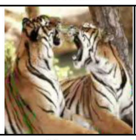
JPEG ENCODING	FRACTAL ENCODING
En.Time: 0.022s Size: 55,392 bytes C.Ratio: 14.20 PSNR: 33.827	En.Time: 8.338s Size: 47,492 bytes C.Ratio: 16.56
DECODE WITH 1 ITERATION	DECODE WITH 2 ITERATION
 De.Time: 0.049s PSNR: can not estimated	 De.Time: 0.062s PSNR: 22.5742
DECODE WITH 3 ITERATION	DECODE WITH 5 ITERATION
De.Time: 0.079s PSNR: 28.8267	De.Time: 0.103s PSNR: 30.3776

TABLE V. FCIC'S RESULTS BY ADJUSTING RMS

JPEG ENCODING	FRACTAL ENCODING WITH 10 INTERATIONS (E=10; R=8)
C.Ratio: 31.05 PSNR: 36.6919	C.Ratio: 43.00 PSNR: 33.7861
FRACTAL ENCODING WITH 10 INTERATIONS (E=20; R=19)	FRACTAL ENCODING WITH 10 INTERATIONS (E=25; R=22)
C.Ratio: 60.13 PSNR: 32.3635	C.Ratio: 64.55 PSNR: 32.0520
FRACTAL ENCODING WITH 10 INTERATIONS (E=30; R=29)	FRACTAL ENCODING WITH 10 INTERATIONS (E=49; R=48)
C.Ratio: 69.95 PSNR: 31.6728	C.Ratio: 82.82 PSNR: 30.7350

Table I, II, III, and IV demonstrate the corresponding decompressed images using FIC and JPEG compression methods in diffrence interation modes. It is obvious that the quality of decompressed image by Fractal method is somewhat superious over that by JPEG.

The experimental results of Parrot's image in Table II show the extremely higher compression ratio of FIC than JPEG but the PSNR of FIC is not really better, eventhough adjusting interation have been performed. The PSNR is not increased anymore when the interation reach to 10. Following FIC algorithm, the acting RMS is used to find what domain is matched to range. The acting RMS is defined by the following equation:

$$RMS = \left(\left(\left(\frac{srt\_size\_X * srt\_size\_Y}{pRange.W * pRange.H} \right) - 1 \right) * r \right) + e \qquad (4)$$

RMS: the acting RMS; srt_size_X and srt_size_Y: the initial size of range; pRange: the pointer point to acting range; H and W: Height and Width of range; e: RMS and r: increase of RMS.

Purposing to improve the compression ratio of FIC for Parrot's image, the acting RMS adjusting have been performed with 10 interations by adjusting the e and r parameters. The Table V show the experimental results.

At the Table V, when the compression ratio is increased very fast, the PSNR is decreased slower. The experimental results in Table V demonstrate the posibility of increasing the compression ratio when sacrificing the PSNR at acceptable level for human vision.

V. CONCLUSION

In this paper, an implementation of fractal color image has been realized in Linux PC. The analysis of the experimental results shows that the compression ratio and PSNR of fractal coding is quite similar to JPEG coding, however the image's quality is acceptable while the compression ratio are superior over JPEG coding. The successful implementation of FCIC is expected to exploit the advantages of FCIC such as high compression ratio, faster decoding and so forth. With an increasing demand for high compression ratio in coder/decoder techniques, the proposed approach may be further developed to apply for fractal coding for low-bit-rate video coding.

REFERENCES

[1] Y. Fisher, "Fractal Image Compppression - Theory and Application", NewYork: Springer-Verlag, 1995.

[2] Mario Polvere and Michele Nappi, "Speed-Up In Fractal Image Coding: Comparison of Methods", *IEEE Transactions on Image Processing*, Vol. 9, No. 6, pp. 1002-1009, 2000.

[3] A. Selim, M. M. Hadhoud, M. I. Dessouky and F. E. Abd El-Samie, ERTU, Egypt, "A Simplified Fractal Image Compression Algorithm", *International Conference on Computer Engineering & Systems 2008* (ICCES 2008), pp. 53 – 58, 25-27 Nov. 2008.

[4] B. Hurtgen and C. Stiller, "Fast hierarchical codebook search for fractal coding of still images," *EOS/SPIE Visual Communications PACS Medical Applications '93*, Berlin, Germany, 1993.

[5] D. Saupe, "Accelerating Fractal image compression by multi-dimensional nearest neighbor search," *Data Compression Conference* (DCC'95), pp. 222-231, Mar. 1995.

[6] A R Nadira Banu Kamal, P.Priyanga, "Iteration Free Fractal Color Image Compression Using Vector Quantization", *International Journal of Advanced Research in Computer and Communication Engineering*, Vol. 3, Issue 1, pp. 5154-5163, January 2014.

[7] Thai Nam Son, Nguyen Tien Dzung, Hoang Manh Thang, Tran V Long, "Efficient implementation of a fractal color image compression on FPGA", *International Conference of Soft Computing and Pattern Recognition* (SoCPaR), Hanoi, Vietnam, 15-18 December, pp. 190-195, 2013.

[8] Eman A. Al-Hilo, Loay E. George, "Study of Fractal Color Image Compression Using YUV Components," *2012 IEEE 36th Annual Computer Software and Applications Conference* (COMPSAC 2012), pp. 596-601, 2012.

[9] Nileshsingh V. Thakur, G.H. Raisoni, Dr. O. G. Kakde, Visvesvaraya, "Color Image Compression with Modified Fractal Coding on Spiral Architecture," *Journal of Multimedia*, Vol. 2, No.4, pp. 55-66, August 2007.

[10] Veenadevi.S.V & A.G.Ananth, "Fractal Image Compression of Satellite Color Imageries Using Variable Size of Range Block", *International Journal of Image Processing (IJIP)*, Volume (8): Issue 1, pp.1-8, 2014.