Parallel fractal video coding using triangular partitioning scheme

Shruti U. Gurlhosur

Dept. Computer Science and Technology, Vishwakarma Institute of Technology , Pune, India

***Abstract*— Over the last few decades, lots of research has been going on in video compression. Video compression technique is now mature as is proven by a large number of applications that make use of this technology. But as large space is required for the storage of video, video compression is required. Fractal is a lossy compression technique that has proven to give better compression and quality as compared to traditional compression techniques like DCT, Wavelet, etc. Video is compressed by the fractal compression method by removing the affine redundancy in the frame. This paper proposes an approach which not only removes the redundancy.**

A Fractal based video compression method relies on the concept of self-similarity within a frame and between successive frames in a video. We are adopting the frame-based video compression method, where the first frame is encoded using the intraframe compression technique and subsequent frames are compressed using the inter-frame compression technique. We are going to use triangular fractal partitioning for range and domain generation and to see how triangular partitioning can affect fractal encoding. We can use a parallel approach for the implementation of fractal encoding.

Keywords— **fractal, video coding, triangular partitioning, GPU, compression, parallelism, speedup.**

I. INTRODUCTION

Picture speaks more than words but picture needs more space to get stored. If we talk about 8-bit image then a single pixel needs 8 bits storage, similarly for 32-bit image, single pixel needs 32 bits of storage. One can imagine how much storage is needed to store a single image. Similarly tremendous amount of memory is needed to store videos as video is nothing but sequence of set of images. Lots of wide range applications like Video conferencing, HD-TV, Video telephony etc. uses digital videos. In order to process video in less time and send over the network in available bandwidth, the size of video should be less. Hence video compression techniques are the need of time and many image compression algorithms are evolved in order to achieve these goals. There are many lossy [2, 3] and lossless [4, 5, 6, 7] compression techniques available for compression of images and videos. The basic idea behind lossy compression technique is obtaining high compression ratio by ignoring some less important aspects.

Fractal compression is one of such lossy compression techniques. This method is similar to Vector Quantization method except Codebook is implicit in fractal. The special thing about this method is, it is resolution independent. This technique tries to achieve high compression ratio by removing affine redundancy present in natural images giving good image quality too. The main drawback of this method for not being as popular as other methods like JPG is, it is computationally very intensive method [8]. Its encoding includes over millions of computations for single image which increases encoding time.

II. PROBLEM FORMULATION

Fractal compression works on the principle of self-similarity. The term fractal means broken. This term was first used by Benoit Mandelbrot to represent objects that are self-similar at different scales and have details at every scale. Based on this Michael Barnsley developed Iterated Function System (IFS). Then Michael Barnsley and his coworkers at the Georgia Institute of Technology used IFS in image compression for the first time. According to Barnsley, in IFS only the part of image that is similar with the whole image is required to find, then the whole image can be reformed with that small part of image so lots of compression can be achieved [9]. Later A. Jacquin had the idea of partitioning the image into non-overlapping ranges, and finding local IFS for each range. This transformed the problem into a manageable task, which could be automated. For his doctoral thesis, Jacquin developed the theory of Partitioned Iterated Function Systems (PIFS) [10, 11].

The crux of the Fractal image compression is the construction of Iterated Function System (IFS). IFS is nothing but union of contractive transformations mapping into itself [12]. The transformation W is contractive if following equation satisfies.

$$d(W(P_1), W(P_2)) < d(P_1, P_2) \quad (1)$$

This means that the distance between any two points P_1 and P_2 reduces after applying the transformation W where distance is Euclidean distance given as,

$$d(P_1 , P_2) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad (2)$$

According to contractive mapping theorem, transformation W must be contractive in order to be finite. As IFS is unique for an image, once defining an IFS of an image any initial image will converge to the only one final image i.e. attractor. Hence in fractal compression the goal is to find the IFS for an image.

While finding affine maps for range block R_i, each domain from domain pool is first scaled to the size of range block. Then this domain block D_i is compared to the range block R_i based on contrast s and brightness o value for all the transformations (four rotations and flip and again four rotations). Then from all these maps, most suitable contractive map (based on minimum error value) is stored for all the range blocks. In case of grayscale images these transformations are of form

$$\omega \begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} a_i & b_i & 0 \\ c_i & d_i & 0 \\ 0 & 0 & s_i \end{bmatrix} \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} e_i \\ f_i \\ o_i \end{bmatrix} \quad (3)$$

Where $s_{i\ and}\ o_i$ are contrast and brightness respectively [13].

A. Partitioning Schemes

The first step while going for fractal image compression is that what type of image partitioning method to go for. Since domain blocks must be transformed to cover range blocks, this decision, together with the choice of block transformation described later, restricts the possible sizes and shapes of the domain blocks. There are many types of image partitioning techniques present most of which being composed of rectangular blocks.

1) Fixed Partitioning

This is the simplest possible way of partitioning. In this image is partitioned into fixed size square blocks. Because of its fixed square blocks it is simple to implement. Domain can be mapped to range in using 8 transformations i.e. four rotations (0^o, 90^o, 180^o, 270^o) and flip and again rotations. But the disadvantage of this type of partitioning is, it doesn't give good results for small complicated regions e.g. eyes in image. The Fig. 1 represents the fixed partitioning scheme [14].

In this partitioning method, image is partitioned only once. Once range pool and domain pool are created then for each range entire domain pool is searched to find minimum RMS value and it is the best match for that range [13]. Hence it is single level partitioning method.

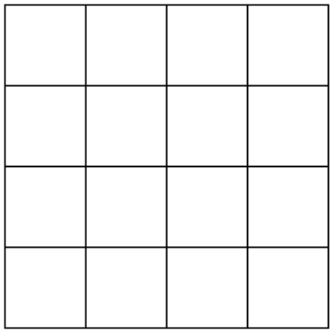

Fig. 1 Fixed Partitioning Scheme

2) Quad-tree Partitioning

This partitioning method is modification to the fixed partitioning scheme which involves multi-level partitioning of ranges. In this method a threshold value is considered for multi-level partitioning. This method also uses minimum and maximum level of partitioning [13].

First image has to be partitioned till minimum level of partitioning then for each range whole domain pool is searched to find RMS values. If the threshold is met for mapping that range then it stored as match for that range else that range is again partitioned into 4 sub ranges and same procedure is repeated till maximum level of partitioning is reached. Hence this result into multi-level partitioning giving a tree like structure in which root represents the original image and leaf nodes represent final ranges. Fig. 2 Represents Quad-tree partitioning scheme.

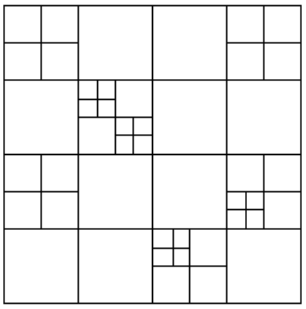

Fig. 2 Quad-tree Partitioning scheme

3) H-V Partitioning

The horizontal-vertical partitioning is another rectangular partitioning method which is similar to quad-tree partitioning scheme and produces tree like structure. Instead of recursively splitting quadrants, however, each image block is split into two by a horizontal or vertical line. Splitting positions may be constructed so that boundaries tend to fall along prominent edges [13]. The Fig. 3 represents H-V partitioning scheme.

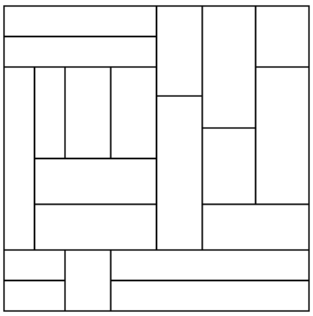

Fig. 3 H-V Partitioning scheme

B. Need of Parallel Computing

The main drawback of fractal is, it includes huge number of computations in encoding part which makes fractal encoding a very time consuming method. The following example will give idea how that how many computations are involved. Suppose we are having 256 x 256 pixel sized grayscale image i.e. all the pixels' intensities vary from 0 to 255 levels. Let fixed partitioning is considered and let non-overlapping ranges are of size 8 x 8 and overlapping domains are twice as that of ranges i.e. of size 16 x 16. So there will be R_1, R_2… R_{1024} ranges and set D will be having 241 x 241 = 58081 number of domains. For encoding range, each range R_i, search is carried out though all D to find the best match, that is, find the part of the image that looks like R_i. There are 8 ways to map one domain to range i.e. rotations 0^o, 90^o, 180^o, 270^o flip and again same 4 rotations. So this way leads to 8 * 58081 = 464648 number of computations to find the best match just for one range and for whole image it requires 1024 * 464648 = 475799552 number of computations. Hence it takes lots of time to encode a single image using fractal. To overcome this problem the fractal encoding stage is needed to be made parallel.

C. Parallel Computing

Parallel computing is programming paradigm in which a larger problem is divided into smaller sub-problems and these smaller problems are then solved in parallel fashion using multiple computing cores i.e. Let's say a certain task needs *n* units of time for execution when run on single core and this task can be divided into *n* independent sub-tasks each taking 1 unit of time, then using *n* computing cores this problem can be executed only in 1 unit of time, this is parallel computing.

The main two reasons for long computational times are

- Same code operating on huge amount of data
- Large different independent codes operating on same or different data.

Based on this, the parallel computing approaches can be broadly classified into two classes as Data Parallelism and Code Parallelism. Initially to achieve high performance computing, scientist tried reducing clock cycle [22]. But as clock frequency increased, power consumption also increased leading to heat dissipation and energy was getting wasted in cooling down these processors. To avoid the drawback of heat dissipation due to reducing the clock cycle/increasing frequency, scientist moved to Graphics Processing Unit (GPU) for high performance computing.

Unlike CPU instead of having few powerful cores, GPU has many less powerful cores [22] which are useful in high performance computing GPU is a co-processor which helps CPU to perform more computations in less amount of time. GPUs are used when throughput is more important than latency [15, 22]. Apart from graphics purposes, GPUs are even used for various scientific and engineering applications involving huge computations. Such GPUs are GPGPU (General Purpose GPU). Various GPUs are available in market from NVIDIA, AMD and Intel etc. NVIDIA has majorly four variants of GPUs. Each variant has a specific purpose. These four variants are

- NVIDIA Tesla
- NVIDIA Quadro
- NVIDIA GeForce
- NIVIDIA Tegra

The NVIDIA Tesla series is General purpose GPU and used for research, scientific and engineering applications but are expensive one. Quadro series is used in large organizations and workstations, GeForce series is used for gaming purpose in laptops and desktops while Tegra series is used for mobile graphics applications. To exploit parallel programming paradigm many programming languages such as CUDA, OpenCL, MPI, OpenMP, OpenACC etc. are used.

D. Literature Survey

Many researchers have tried parallelizing the fractal encoding part. Numbers of authors have tried to divide these into different categories in their papers [17, 18].

One of the simplest ways these efforts can be categorized is based on granularity i.e. if the algorithm is *fine grained* or *Coarse Grained*. Some authors provided extreme level fine grained approach [19] in which n^2 processors work on n x n size image, each processor working on a single pixel. Though this method [19] gives good quality but it has the drawback of increased cost due to use of so many processors. On the other side, some authors have given algorithm which works on blocks of pixels [17], where the image is divided into sub-images in the number of available number of processors and each processor applies sequential fractal encoding for its respective sub-image without any communication with the other processors. This is embarrassingly parallel algorithm. And as a processor is working only part of the image, it won't be having all the data about image so quality is degraded in this case.

Some of the researchers tried load balancing technique for parallelizing the encoding part. The simplest load balancing approach was [17, 20] where range pool is equally divided among available number of processors. This kind of approach is suitable where range pool and domain pool are fixed and they are likely to find the best possible match. When all processors need approximately same amount of time for encoding range, then each processor automatically will have same load. But this static load balancing method fails in case of multi-level partitioning as that of Quad-tree partitioning as each range can get subdivided into another 4 squares resulting into unbalancing of load among the processors.

To overcome drawback of static load balancing few researchers tried dynamic load balancing methods using master-slave architecture as extension to static load balancing. Where each slave will process its range, and if the undergo further division then ranges are divided into other slave processors and the original slave processor will become now master processor. This master processor will be responsible for proper execution of its slave processors. Even though dynamism is achieved through this method, but it includes additional communication cost among master and slaves. Plus crashing of communication may lead to infinite state.

So to avoid these additional communications cost in coarse grained parallelism, data partitioning method is used where even while working in distributed environment either each processor will have sufficient memory to store domain pool or it doesn't. When each processor has sufficient memory available with it, hence can map range without communicating to any other processor. The second case doesn't stand in recent years due to technology has scaled sufficiently well [12].

III. PROBLEM SOLUTION

The traditional fractal compression algorithms apply rectangular partitioning approaches like fixed partitioning, quad-tree partitioning and H-V partitioning etc. These partitioning techniques have drawback of rigid 90^o rotations of domain while mapping to range. The rectangular geometry of range and domain needs 8 transformations (0^o, 90^o, 180^o, 270^o, flip and same 4 rotations) while mapping leading to more number of computations.

This paper proposes a modern partitioning technique called triangular partitioning technique in which image is

partitioned triangularly. The triangles considered in this paper are isosceles right angled triangle in order to cover whole image without leaving any gray area in between. Fig. 4 represents triangular partitioning used in this paper. As range and domain are isosceles right angled triangles, only 2 transformations (0^o and then flip 270^o rotation) are needed to map similarly oriented domain-range (1&7, 2&5, 4&5, etc.) and 2 more transformations (180^o rotation and flip then 90^o rotation) needed to map differently oriented domain-range (2&7, 4&7, 3&6 etc).

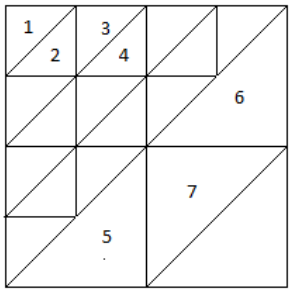

Fig. 4 Triangular Partitioning

While encoding video, frames undergo triangular fractal partitioning. The following algorithms explain details of fractal video coding.

Algorithm: Encoding (V)
Input:
V: Video
Output:
E: Encoded File
Variables
s: Contrast
o: brightness
Device: GPGPU
Frame number: Current frame number
Pool number: number of the frame to which current frame is similar to
RMS: Root mean square

1. Extract all frame from Video
2. For each Frame
   2.1. Save current frame
   2.2. Partition frame triangularly
   2.3. Create range pool
   2.4. Create domain pool
   2.5. Save frame number and pool number
   2.6. Copy domain pool to global memory of device
   2.7. Copy range pool to device
   2.8. For each range launch a thread on GPU and for each thread
      i. Search domain pool
      ii. Calculate s, o for each domain and its transforms
      iii. Calculate RMS for each domain and its transforms
      iv. Find domain transform with minimum RMS
      v. Store mapping for each thread
   2.9. Copy mapping values from device to host
   2.10. Update encode file
3. If current frame is last frame then go to 5
4. Else
   4.1. Get Next frame
   4.2. If PSNR of current frame with last saved frame is less than threshold PSNR
      i. Go to step 2
   4.3 Else
      ii. Save the current frame number.
      iii. Save Pool number as last saved frame number.
      iv. Update encode file
      v. Go to step 3
5. Save Encode File and exit.

Algorithm: Decoding (E)
Input:
E: Encoded File
Output:
V: Video
Variables
s: Contrast
o: brightness
Frame count: number of frames in video
Frame number: current frame number
Pool number: number of the frame to which current frame is similar to
RMS: Root mean square

1. Open Encode file
2. Read frame count
3. For each frame till current frame number reaches frame count
   3.1. Read frame number and pool number.
   3.2. If frame number and pool number. are same
      3.2.1. Calculate number of ranges
      3.2.2. Create blank image or take any initial image
      3.2.3. Read domain number, s, o and transformation number.
      3.2.4. Multiply corresponding domain pixel values of image by s and add o to it.
      3.2.5. Apply transformation to domain
      3.2.6. If current range was last
         3.2.6.1. If image converges
            3.2.6.1.1. Dump image as temporary frame image

3.2.6.1.2. Go to step 3.4
3.2.6.2. Else go to step 3.2.3
3.2.7. Else go to step 3.2.1
3.3. Else
3.3.1. Copy last decoded image again
3.4. Is frame number. is equal to frame count, go to step 4
3.5. Else go to step 3.1
4. Integrate all decoded temporary images to video.
5. Exit

The experiments for various videos have been carried out. The results of those experiments are shown below in charts. For comparison all videos have been tested for 20 frame size. Various results are taken out for different range size, sequential methods and parallel method, for various partitioning techniques and for intra frame and inter frame processing.

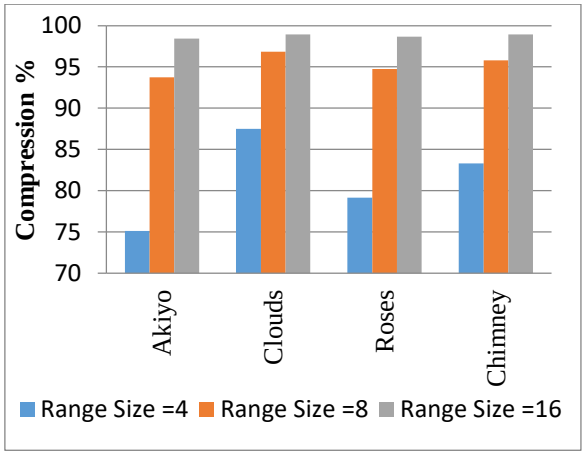

Fig. 5 effect of range size on compression

It seen from above figure 5 that range size and compression are directly proportional to each other i.e. as range size increases, compression percentage also increases. Even if the compression for video depends on how many frames of video got encoded, the size of range plays important role in deciding the compression. Irrespective of how many frames undergo encoding, small range size leads to lower compression of the video.

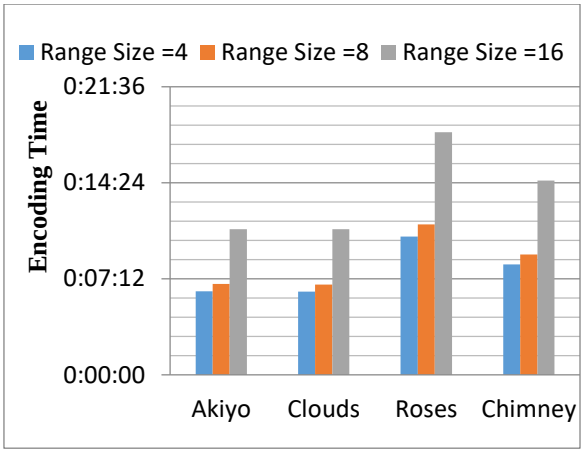

Fig. 6 Effect of range size on encoding time

It is seen from the figure 6 that range size is directly proportional to encoding time. This is because, in case of small range size, the same amount of data to be computed is partitioned among more number of parallel processing threads which reduces the strictly sequential time each thread is going to required. Whereas, when range size is large, same amount of data is partitioned among few parallel processing threads leading to increase in strictly sequential time required for each thread. Hence increase in range size increases the encoding time.

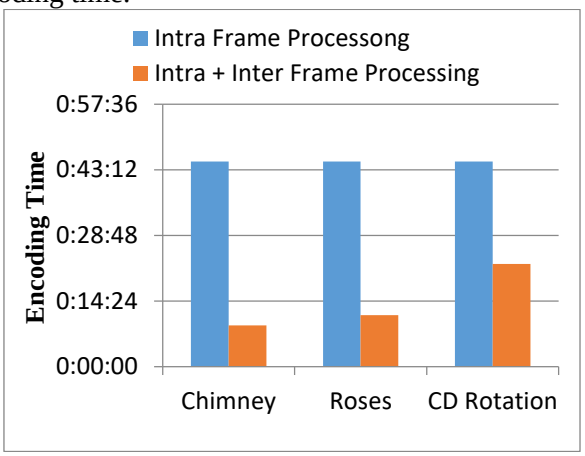

Fig. 7 Encoding time difference between intra and inter frame processing

It seen that only intra-frame video compression leads to higher encoding time than Inter + Intra frame video compression as in intra frame video compression irrespective of how much similar adjacent frames are, every frame undergoes encoding, leading to more number of computations. But in case of Inter frame compression if adjacent frames are similar then those are not encoded i.e. only few frames are encoded hence it takes lesser time.

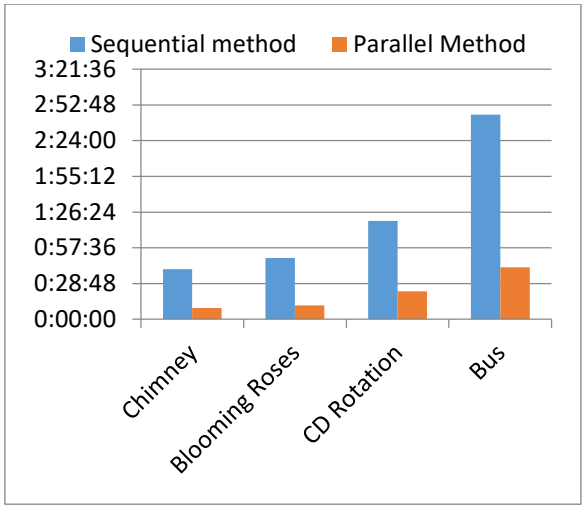

Fig. 8 encoding time difference between sequential and parallel processing

As in sequential processing, all the computations are carried out by a single processor, so it takes more time as in Fig 8. Whereas in case of parallel video processing, multiple threads are carrying out the computations which reduces the encoding time.

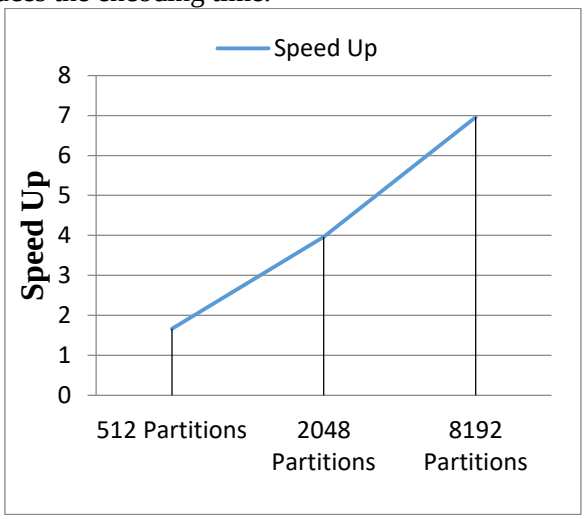

Fig. 9 Speedup Vs Number of partitions

Figure 9 represents the chart for Speed up Vs number of partitions. It shows that with the increase in number of partitions, speed up also increases as increasing the number of partitions reduces the amount of computations to be done by single thread. Speed up depends on computational time, I/O time (between CPU and GPU memory) and synchronization time. Out of which synchronization time is very small (less than milliseconds) so can be ignored. The encoding time of fractal depends on following factors

- I/O time – Time required for transferring range pool and domain pool from CPU to GPU memory and copying map results back from GPU to CPU.
- Computational time – Time required to find the best match for each range based from whole domain pool.
- Synchronization time – Time spent in waiting for other threads to finish execution.

Increasing number of partitions reduces the computational time hence speed up increases. If number of partition goes on increasing, beyond one limit speed up stops increasing. This happens due to I/O overhead. As range size goes decreasing (i.e. number. of partitions increasing), domain number goes increasing (4 times than the previous higher domain size) which increases the I/O time to copy whole domain pool from CPU to GPU memory as well as it also increases the strictly sequential time of each thread (finding map for one range whole domain pool is to be searched). Hence due to these overheads after one limit speed up stops increasing.

IV. CONCLUSION

Fractal video coding is very efficient technique giving high compression and good PSNR for natural videos removing affine redundancies. Fractal gives better results for the videos with greater affine redundancy. Traditional fractal coding approaches needed more encoding time for processing video and even has large storage requirements than the proposed algorithm over GPU. Use of triangular partitioning scheme increases the quality of video by capturing the small areas of frame. The proposed algorithm also removes redundancy between adjacent frames by comparing the adjacent frames for similarity based on PSNR value. This also reduces the computational time for video processing as well as storage is also reduced giving higher compression than traditional methods. Use of GPU for parallelizing mapping computations reduces the encoding time to almost to one fourth of the traditional way sequential encoding time. This method launches as many threads as that of number of ranges in a frame leading to effective use of GPU cores. This way speed up is achieved in fractal encoding by the proposed method. Experiments show that algorithm achieves high compression with less amount of time with good PSNR.

ACKNOWLEDGMENT

I am immensely grateful to Prof. Dr. M. V. Kulkarni, Vishwakarma University, Pune, India for sharing his pearls of wisdom with me during the course of this research.

REFERENCES

[1]. Proceedings of World Academy of Science: Engineering & Technology; 2007, Vol. 20, p370 “comparison of compression ability using DCT and fractal technique on different imaging modalities – Poolbal, Sumathi, Ravindran.G”, April, 2007.

[2]. D. A. Huffman, “A method for the Construction of Minimum-Redundancy Codes”, Proceedings of the Institute of Radio Engineers, 40(9):1098-1101, 1952.

[3]. J. Ziv, A. Lempel, “Universal Algorithm for Sequential Data Compression”, IEEE Transactions on Information Theory, 23(3): 337-343, 1977.

[4]. Video codec for audio visual services of Px64 Kbits/s. CCITT Recommendation H.261, Aug.1990.
[5]. Digital Compression and Coding of Continuous-Tone Still Images Part 1, Requirements and guidelines. ISO/IEC JTCI Committee Draft 10918-1.
[6]. Digital Compression and Coding of Continuous-Tone Still Images Part 2, Compliance testing. ISO/IECJTCI Committee Draft 10918-2.
[7]. M. L. Liou, “Overview of the Px64 Kbits/s Video Coding Standard”, Communication of the ACM, Vol.34, No.4, 1991.
[8]. Y. Fisher," Fractal Image Compression: Theory and Application". New York: Springer-Verlag New York, Inc., 1995.
[9]. M. Barnsley, L. Hurd, “Fractal Image Compression”, AK Peters, Wellesley, 1993.
[10]. A.Jacquin, “Image coding based on a fractal theory of iterated contractive Markov operators”, Construction of fractal codes for digital images- Part II, Report Math. 91389-017. Georgia Institute of Technology, 1989.
[11]. A. Jacquin, “Fractal image coding: A review”, Proceedings of the IEEE 81, 10 (1993) 1451-1465.
[12]. M. L. Liou, P.K.Jimack, “A survey of parallel algorithms for fractal image compression”, Journal of Algorithms & Computational Technology, Pages 171-186, ISSN 1748-3018, July 2009.
[13]. Y.Fisher, “Fractal Image Compression: Theory and Application”. New York: Springer-Verlag, New York, Inc. 1995.
[14]. B. Wohlberg, G. Jager, “A review of the fractal image coding literature”, IEEE transactions on image processing, vol. 8, No. 12, December 1999.
[15]. CUDA C Best Practices Guide, NVIDIA Corporation
[16]. CUDA C Programming Guide, NVIDIA Corporation.
[17]. P. Palazzari, M. Coli, and G.Lulli. “Massively parallel processing approach to fractal image compression with near-optimal coefficient quantization”. Journal of Systems Architecture 45. 765-779.1999
[18]. A. Uhl, and J. Hammerle. “Fractal Image Compression on MIMD architectures I: Basic Algorithms”. The First International Conference on Visual Information Systems. 1996.
[19]. X. Min, T. Hanson, and A. Merigot. “A massively parallel implementation of fractal image compression”. IEEE International Conference on Image Processing. 1994.
[20]. D. J. Jackson, and G. S. Tinney. “Performance analysis of distributed implementation of a fractal image compression algorithm”. Concurrency-Practice and Experience 8 (5): 357-386. 1996.
[21]. Nvidia’s next generation CUDA compute architecture Fermi, NVIDIA Corporation.
[22]. Sunpyo Hong, Hyesoon Kim, “An analytical model for a GPU architecture with memory-level and thread-level parallelism awareness”, ISCA '09 Proceedings of the 36th annual international symposium on Computer architecture, Pages 152-163, ISBN: 978-1-60558-526-0.