# Fractal Image Compression Using a Motorola DSP96002

R.Bareket, Y.Or-Chen , N.Peleg,
G.Spiro , O.Tzifronin
*Signal & Image Proc. Lab., EE Dept., Technion IIT, Haifa, Israel*

Abstract
This Paper Describes an image compression application based on approximating an original image by a fractal one. It is running on a DSP board, utilizing the Motorola DSP96002.

According to the algorithm we shall descibe, the original image is divided into non-overlapping blocks, and the approximation is done blockwise. For each block we examine blocks in its vicinity, and find a larger block and a transformation that shrinks the large block and maps it onto the original block. The process is speed up by classifying the blocks into one of 3 categories: shade, midrange and edge. Finally, we store the position of the large block and the transformation, thus storing the fractal qualities of the image. For every block a fractal transformation is found. When this transformation is performed iteratively on any initial image, the result converges to the fractal approximation of the image.

The system includes a PC-386 host (running DOS) and a PC-Board, based on the Motorola DSP96002 with 128KB RAM, running at 12MHz. The DSP's major features are its highly parallel assembly language and multiple bus structure. We also use the fact that the DSP is a dual port processor, so it can execute codes using an external memory bank, connected only to it, while communicating with its host computer through another external memory bank (Connected to both).
The DSP does all the calculations, while the host does the control and displays output (e.g. elapsed time, iteration no., decoded images etc.)

## Introduction to Fractal Image Compression

A theoretical background for this work can be found in [1] and [2]. According to [2], one can describe a curve characterized by a set of points (nodes), by a set of iterative functions which act upon the nodes.
For every two following nodes (range) one finds 2 other nodes (domain) and a transformation that maps the domain nodes onto the range nodes.
The transformation should be contractive, so that the domain nodes can not be the range nodes themselves, because this results in an identity transformation which is not contractive.

In this way, one finds relations between parts of the curve, so by forcing a set of nodes to obey this relations, one recreates an approximation of the original curve. Only the transformation parameters are saved, and this is obviously less memory consuming. To reconstruct the curve one lets the transformations to act iteratively on any set of initial nodes.

In order to implement this idea when compressing an image, it must be generalized. An image, instead of a curve, will be divided into sub-blocks, instead of nodes. For each range block a domain block and a transformation are found. The compressed image will in fact be represented by the set of parameters characterizing the transformation necessary to create the range blocks out of the domain blocks. Reconstructing the image is very similar to reconstructing the curve: we let the transformation act on any initial image, to create a new imagewhich is closer to the original image (the encoded image). Applying this procedure iteratively, where the recent image serves as the domain for creating a new one, yields a series of images which converge to the fractal approximation of the original one.

Formally, we consider the digital images space $(\mu_0,d)$, where $d$ is a distortion measure. We refer to the original images as $\mu_{orig}$ and search for a contractive transformation $\tau$, which is defined from the digital images space to itself, and that $\mu_{orig}$ is approximately a fixed point (in this space) under it. We say that $\mu_{orig}$ is self-transformable (invariant) under $\tau$.
The requirements on $\tau$ are:
1. There exists some $0<s<1$ , such that for every $\mu$ and $\tau$ , the contraction condition holds:
$d[\tau(\mu),\tau(\nu)] \le s \cdot d[\mu,\nu]$
2. The distortion $d[\nu(\mu_{orig}),\mu_{orig}]$ is very small (the invariance condition).
3. $\tau$ is less complex than the original image (can be described by less information)
The number s is said to be the contractivity of $\tau$.

When $\tau$ acts n times iteratively, on any initial image $\mu_0$ , we have:

$$d[\mu_{orig},\tau^n(\mu_0)) \le \frac{1}{1-s} \cdot d(\mu_{ori},\tau(\mu_{or})) + s^n \cdot d(\mu_{orig},\mu_0)$$

The contractivity number s should be smaller then one, so that when n grows higher the second term vanishes, and the upper limit for the distortion between the reconstructed image and the original one is of the size of the distortion between $\mu_{orig}$ and $\tau(\mu_{or})$, which is very small. We see then that the initial image is not important, as the second term, which involves it , vanishes. After a sufficient number of iterations, the fractal approximation of the image will evolve out of any initial image. The transformation $\tau$ is called the "fractal code" of $\mu_{orig}$ .

In order to find $\tau$, we rely on the invariance of the original image under it. If we find a transformation that maps the original image approximately onto itself, then that transformation is the desired one. Since it is very difficult to find a transformation that acts on the whole image, we

divide the image into many sub-blocks and split $\tau$ into many block-transformations. This allows us to use a small set of relatively simple transformations. The image is divided into square non-overlapping blocks. For each block in the image (range) we find a block from a domain pool and a transformation from a transformations set. This transformation should, when acting on the domain block, result in a block which is similar to the original block.

The transformation consists of two parts: a geometric transformation and a massic transformation. The geometric transformation $g$ shrinks a domain block whose size is four times the size of the range block. The massic transformation changes the gray levels of the pixels in the block and arranges them, i.e. rotating the block, inverting the block with reference to some axis, and so on. For every range block we find a domain block $D_i$ and a transformation $\tau_i$ (Figure 1), such that $\tau_i[g(D_i)]$ is the best approximation of the original range block.

The domain pool is created from the original image dividing it into large partially overlapping square blocks (the size of 4 range blocks).

The domain pool is related to the image, and this will lead to "fractal linkage" between different parts of the image. We point out that this particular domain pool is not necessary when decoding the image and is therefore not saved, yielding a very short code.

The transformation $\tau$ is the union of all the block transformations, noting the domain and range blocks position in the image and the parameters of the massic transformation. The decoding is done block by block, just like the encoding.

## The Algorithm

The original image is divided into square non-overlapping 8x8 blocks, Creating a 2-dimensional array of blocks. A finer division or a two-level division can also be implemented [1] (and was performed in simulation [6]). This array of blocks will be referred to as R-space. smaller blocks allow finer results but demand moreprocessing time and result in less compression, whereas larger blocks allow faster encoding and more compression, but yield poor results [6].

The domain pool is generated by dividing the image into square partially overlapping 16x16 blocks, creating another 2-dimensional array of blocks. The blocks are created by sliding a window of 16x16 pixels over the original image with $\delta h$ and $\delta v$ as horizontal and vertical shifts, respectively. The smaller $\delta h$ and $\delta v$ are, the bigger the domain pool is and the longer the encoding process is. If $\delta h$ and $\delta v$ are big, there are less domain blocks which are close to the range block, and this proves to result in a bad reconstruction [6]. Accordingly, we used $\delta h = 2$ and $\delta v = 2$. This array of blocks will be referred to as D-space.

As mentioned before, we let geometric transformation and a massic transformation act on the domain block, and compare the resulting block with the range block. To speed up this process, the blocks are classified, both in the range and the domain. The classification, based on [3], is to one of three categories: 'shade', 'midrange' and 'edge'. A 'shade' block is a smooth block with no significant gradient (in the sense of gray levels), an 'edge' block is a block in which there is a significant change in gray levels along a curve, usually at the margins of an object in the image, and a 'midrange' block is a block in which there is a slight gradient of the gray levels, but not a definite edge (Fig. 1). The classification takes place during the pre-processing of the image, in which every range block and every domain block is analyzed. This classification enables us to compare blocks of the same category only, and this is all the comparisons we need, since the set of transformations that we use does not change the classification of the blocks. This means that we check only relevant blocks, and save time.

As mentioned before, the transformations consists of a geometric part and a massic part. The geometric transformation, $\theta$, shrinks a 16x16 (domain) block into an 8x8 (range) block, by averaging four pixels into one. The massic transformation, $\phi_i$, belongs to a set of transformations which perform global changes on a block. These transformations can be split up into two groups: transformations that change the gray level of the pixels and transformations that change the position of the pixels in the block. The first group consists of three transformations: creating a block with a given gray level ($g$- absorption), changing the current gray level by a given value ($\delta g$-translation) and changing the current gray level by multiplying it with a given number ($\alpha$-scaling). These transformations act on all the pixels in the block, and make sure the gray level of every pixel is within the range of 0-255.

The second group consists of eight isometric transformations. They perform inversions, with respect to the X axis, the Y axis or the diagonals, and rotations of 90°, 180° and 270°. All the transformations, from both groups, have contractivity one, except the scaling transformation, which has contractivity $\alpha^2$. These transformations enable us to derive a set of blocks out of one block, thus enriching the domain pool.

The geometric transformation is not adaptable. It has no parameters and simply shrinks a 16x16 block to a 8x8 block. The massic transformations are adaptable and all three has one parameter each. The isomtries have no parameters, but since the final massic transformation uses only

one (if any) isometry, one can consider the isometries as a transformation with one parameter, determining the isometry type used. The massic transformations, therefore, may vary with parameters. Finding the correct parameters is the major part of the encoding process.

Encoding a range block involves finding the best $(D_i, \phi_i)$ pair, so that the distortion between the reconstructed block (after letting $\theta$ and $\phi_i$ act on a domain block $D_i$) and the original range block is minimal. The product space $D\phi$ is called a global pool, or a virtual codebook. The product space is vast. For a 256x256 image, with 8x8 range blocks and 16x16 domain blocks, and $\delta h = \delta v = 2$, the size of D-space is 14641 elements. If we consider the massic transformation too, we arrive to a conclusion that $D\phi$ is several times bigger than D. This gives us an enormous codebook, and this is why the encoding process is time consuming.

The size of the D-space is the bottleneck of the encoding process. In order to make the encoding faster, one should use a smaller domain pool. This should be done cautiously, since arbitrary reduction of the domain pool can cause several range blocks to be poorly approximated. This will cause a poor approximation of $\tau$ and will reflect badly on the distortion between the original image and the reconstructed one. Our solution is based on the assumption that for a given range block, the best domain block will be near the range block, seemingly because close blocks are more alike than distant blocks. We therefore work with a dynamic domain pool, which changes from one range block to another. The dynamic domain pool consists of domain blocks which are positioned around the range block (in the image). We used a sliding window of 48x48 pixels that was centered on the range block, and found that bigger windows did not improve the fidelity of the reconstructed image, but required moreprocessing time. We also used small horizontal and vertical shifts ($\delta h$ and $\delta v$). In this way we get a sufficiently large domain pool which consists of domain blocks that are 'physically" close to the range blocks.

The System

The sytem on which the algorithm is executed consists of two major parts (Fig.2):
1. an Intel-386sx PC/AT (IBM compatible),25MHz with 512KB RAM base memory, 2MB extended memory, S-VGA adaptor and display monitor, to be referred as host computer (HC).
2. A DSP board (DSPB), based on the Motorola DSP96002, with two external memory banks (connected to ports A and B of the DSP). The processor is operated with a 12MHz clock.

The two memory banks are of 128KB each, one is common to the HC (located just above the regular 512K upto 640K) and to the DSP (port A) and the other belongs only to the DSP. The Host Interface of the DSP is mapped as a peripheral device in the PC address space, and occupies 16 following addresses, enabling the DSP operation under the HC control and direct data transfer between the HC and the DSPB.
The DSP96002 is a floating point processor, working according to IEEE-754 standard single precision (32-bit) and single extended precision (44-bit) arithmetic. Detailed architecture representation of the DSP96002 is beyond the scope of this paper [8].
Its major features are:
- 6 MIPS (with 12MHz clock)
- 18 MFlops (with 12MHz clock)
- Single-cycle 32x32 bit parallel multiplier
- High parallel instrucrtion set with unique addressing modes
- Hardware nested DO-loops
- Two independent on-chip 512 word data RAM
- Two independent on-chip 1024 word data ROM
- Off-chip up to $2*2^{32}$ word of data memory
- On-chip 1024 word program RAM
- On-chip 1024 word program RAM
- On-chip 64 word bootstrap ROM
- Off-chip up to $2*2^{32}$ word of program memory
- Two identical external memory expansion ports

Sytstem Performance

The algorithm was implemented on the above system. The system performance will be described comparing to a simulation running on an Intel PC-486DX with 8MB RAM memory, at 33MHz.The simulation s/w was written in Borland Turbo-C++. Table 1 shows the ratio between the two systems:

| | PC-486(33MHz) : DSP96k(12MHz) |
|---|---|
| Compression: | |
| Pre-Processing | 1.74:1 |
| Encoding | 4.91:1 |
| Decompression: | |
| Decoding (20 Iterations) | 4.74:1 |
| The same + Displaying | 2.0:1 |

Table 1: Timing ratio between systems

The pre-processing stage consists of image partition into blocks and classification of the blocks into 'shade', 'midrange' and 'edge'. The classification occupies about 94% of the pre-processing time. The preprocessing time is rigid and does not depend on the image. Using other pre-processing methods may alter the execution time duration of this part, e.g. images containing fine textures will demand longer pre-processing time whereas smooth images will demand shorter time.

The encoding process is mainly a search for the best blockwise transformation. It is an image dependent algorithm, and therefore the encoding time may vary from one image to another. The easiest block to encode are the 'shade' blocks

(negligable time) and the most difficult are the 'edge' blocks (twice the time of 'midrange' blocks). For example, the time needed for the compression of the well known image 'Lena' is (using the DSPB) 9'50" (2'10" for the pre-processing and 7'40" for the encoding), while compressing the same image on the PC-486 took 41'20" (3'50"for the pre-processing and 37'3" for the encoding).
A detailed analysis of the encoding process shows [6] that (as expected) the bottleneck is the search for domain blocks. The search is conducted over a small portion of the D-space by checking domain blocks which are in the neighbourhood of the range block. A 48x48 pixels window, centered about the range block, is used to determine the range block's neighbourhood. The domain window contains 289 domain blocks and this part of the process occupies about 50% of the encoding time (about 20% is dedicated for the analysis of 'edge' blocks).

The decompression is an iterative decoding process and is much faster then the compression time (about 30" on the DSPB and 2'27" with the PC-486).

A detailed analysis of every stage of the compression /Decompression processes is available [6].

## Conclusions

We are sure that a more dedicated plan of the board (with more memory, on-board cache and 2 DSP's) will achieve much better results. It is important to mention that we ran the DSP at 12MHz because the board used is a prototype version. The DSP can run now at 40MHz and this is expected to improve the results significantly (A board at this speed is being built now).


## Acknowledgment

The work described has been undertaken at the Signal and Image Processing Laboratory, Faculty of Electrical Engineering, Thechnion IIT, as part of final project for 1st degree.
The author wishes tohank Mr. Z.Beharav for his helpful discussions and Prof. D.Malah for his valuable notes and support.


## References:


[1] A.E.Jacquin, "Image coding based on a fractal theory of iterated contractive image transformations", IEEE Trans. on image proc. January 1992.
[2] M.F.Barnsley, A.E.Jacquin, "Application of Recurrent Iterated FunctionSytsem to Images" SPIE Visual Comm. & Image Proc.,
[3] B.Ramamurthi, A.gersho, "Classified Vector Quantization of Images", IEEE Trans. on Comm., COM-34,No.11, 1986
[4] M.Barnsley "Fractals Everywhere", Academic Press Inc, San-Diego, 1988 (Ch.3,6)
[5] R.C.Gonzales, P.Wintz, "Digital Image Processing", Addison-Wesley,2nd Ed., 1987
[6] O.Zifrony, R.Barequet, "Fractal Image Compression" (Parts A & B), Final Project, Sig. and Image Proc. Lab, EE Faculty Technion , IIT
[7] Zvi Rozenshein, "Boost PC's Floating-Point Speed with an Add-On DSP Coprocessor Board" Electronic Design, Jan.10, 1991.
[8]DSP96002 Users Manual, Motorola, 1992.


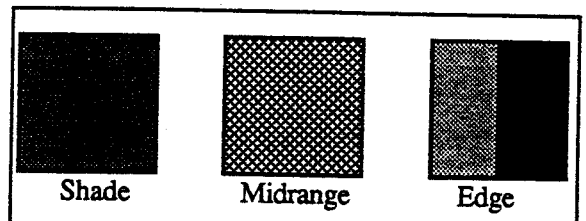

Figure 1: Types of Blocks

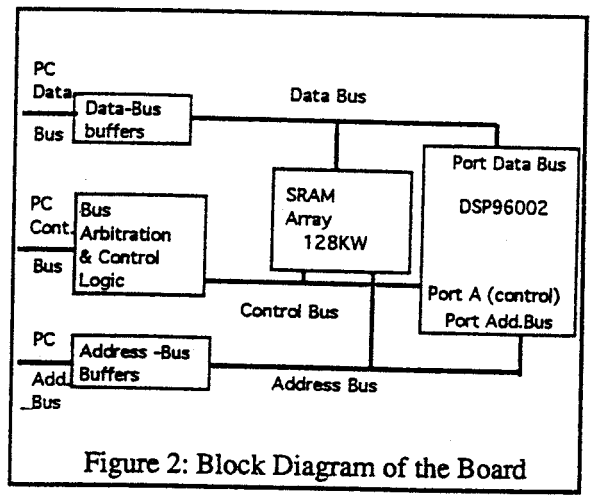

Figure 2: Block Diagram of the Board