3 Image Compression

Multimedia - Department of Computing, Imperial College
Professor GZ Yang ! **http://www.doc.ic.ac.uk/~gzy**

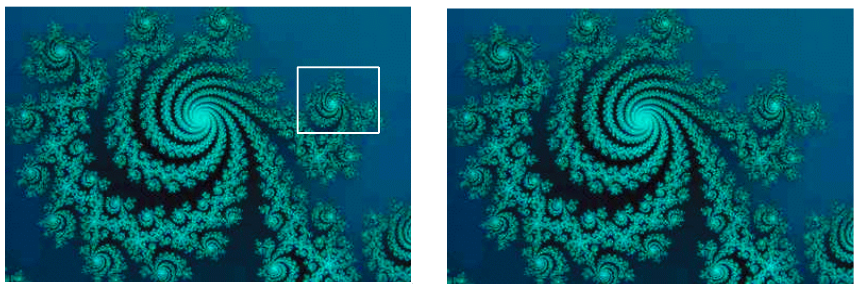
Why Image Compression

Research in compression techniques has stemmed from the ever-increasing need for efficient data transmission, storage and utilisation of hardware resources. Uncompressed graphics, audio and video data require considerable storage capacity and transmission bandwidth. Despite rapid progresses in mass-storage density, processor speeds, and digital communication system performance, demand for data storage capacity and data-transmission bandwidth continues to outstrip the capabilities of avail-able technologies. The recent growth of data intensive digital audio, image, and video (multimedia) based applications, have not only sustained the need for more efficient ways to encode signals and images but have made compression of such signals central to signal-storage and digital communication technology. Here are some examples of the space required for storing different kinds of multimedia information:

Multimedia Data	Size/Duration	Bits/Pixel or Bits/Sample	Uncompressed Size
A Page of text	11'' x 8.5''	Varying resolution	16 – 32 Kbits
Telephone quality speech	1 sec	8 bps	64 Kbits
Grayscale image	512 x 512	8 bpp	2.1 Mbits
Color image	512 x 512	24 bpp	6.29 Mbits
Medical image	2048 x 1680	12 bpp	100 Mbits
Full-motion video	640 x 480 , 10 sec	24 bpp	2.21 Gbits

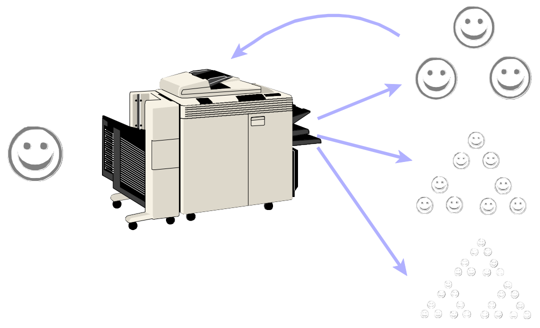
Image compression methods are typically divided into two categories, lossless and lossy. Compression techniques belonging to the first category have the main characteristic that the image involved can be perfectly reconstructed from the compressed file, thus having no information loss. On the other hand lossy compression methods rely on the fact that some of the data can be discarded with almost no detectable loss in image quality by the human visual system. When research into image compression began in the late 1970s, most compression concentrated on using conventional lossless techniques, meaning that the reconstructed image after compression is numerically identical to the original image on a pixel-by-pixel basis. However, such types of compression techniques, which included statistical and dictionary methods of compression, did not tend to perform well on photographic, or *continuous tone* images. The primary problem with statistical techniques is due to the fact that pixels in photographic images tend to be well spread out over their entire range. Hence, if the colours in an image are plotted as a histogram based on frequency, the histogram is not as “spiky” as it would be required for statistical compression to be effective. Each pixel code has approximately the same chance of appearing as any other, negating any opportunity for exploiting entropy differences.

By the late 1980s, extensive research pushed the development of lossy compression algorithms that take advantage of known limitations of the human eye. Such algorithms play on the idea that slight modifications and loss of information during the compression/decompression process often do not affect the quality of the image as perceived by the human user. One such technique is to exploit the self similarity nature of image patterns based on fractal geometry for image compression.


Self similarity in images

Fractal and Iterated Function Systems

The birth of fractal geometry is usually traced to IBM mathematician Benoit B. Mandelbrot and the 1977 publication of his seminal book *The Fractal Geometry of Nature* [35]. It stresses the fact that traditional geometry with its straight lines and smooth surfaces does not resemble the geometry of trees and clouds and mountains. Fractal geometry, with its convoluted coastlines and detail *ad infinitum*, does. This insight opened vast possibilities, allowing computer scientists to generate artificial yet realistic looking forms. Shortly after Mandelbrot's work, mathematicians searched for a framework underlying fractal geometry. As John Hutchinson demonstrated in 1981, it is the branch of mathematics known as *Iterated Function Theory* . Later in the decade Michael Barnsley authored *Fractals Everywhere,* another milestone work. The book presents the mathematics of Iterated Functions Systems (IFSs), and develops a result known as the *Collage Theorem*. The Collage Theorem states what conditions an Iterated Function System must satisfy in order to represent an image. This presented an intriguing possibility. If, in the forward direction, fractal mathematics is good for generating natural looking images, then, in the reverse direction, could it not serve to compress images? Going from a given image to an Iterated Function System that can generate the original (or at least closely resemble it), is known as the *inverse problem*. In its general form, the inverse problem remains unsolved. In search of something practical, Arnaud Jacquin, one of Barnsley's students, arrived at a modified scheme for representing images using Partitioned Iterated Function Systems (PIFSs). In his PhD thesis, Jacquin developed the necessary mathematical foundations and implemented the new approach in software, a description of which appears in his landmark 1992 paper "Image Coding Based on a Fractal Theory of Iterated Contractive Image Transformations". The algorithm was not sophisticated, and was computationally expensive, but it was fully automatic. All contemporary fractal image compression programs are based upon Jacquin's approach.



An elegant way of introducing the notion of Iterated Functions Systems is by the metaphor of a Multiple Reduction Copying Machine (MRCM). An MRCM is imagined to be a regular copying machine except that:

- There are multiple lens arrangements to create multiple overlapping copies of the original.
- Each lens arrangement reduces the size of the original.
- The copier operates in a feedback loop, with the output of one stage the input to the next. The initial input may be any image.

The above figure depicts this process for Sierpinski's Triangle, one of the simplest (and most well known) IFS. It is comprised of three component functions ("lenses"), each of which shrinks the input image by one half and translates it to a new position. This contractive property is crucial, for it guarantees convergence of the iterative process. Because all initial images are "drawn towards" the same final result, it is variously referred to as the attractor of the IFS, or the fixed point image.

Mathematically, one can represent each reproduction lens as a contractive affine transformation which rotates, scales, shears, and translates the original copy to a target location, i.e.,

$$w_i = \begin{bmatrix} a & b \\ c & d \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix}$$

which maps a given point in the original image (x, y) to a new coordinates (x',y'), where

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} a & b \\ c & d \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} e \\ f \end{bmatrix}$$

For the Sierpinski Triangle shown above, the three transformations used can therefore be represented as

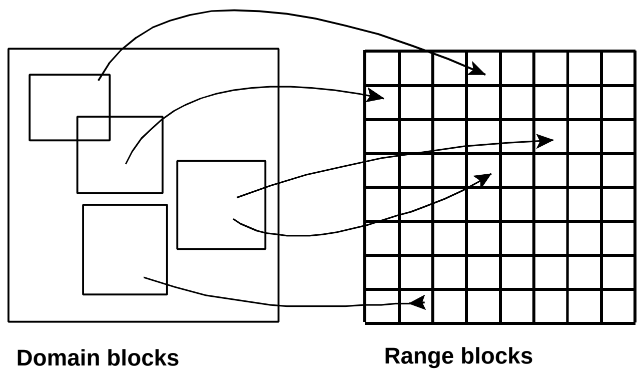
$$w_1 = \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \end{bmatrix} \quad w_2 = \begin{bmatrix} 0.5 & 0 & 0.5 \\ 0 & 0.5 & 0 \end{bmatrix} \quad w_3 = \begin{bmatrix} 0.5 & 0 & 0.25 \\ 0 & 0.5 & 0.5 \end{bmatrix}$$

It can be proven that if the determinant of each transform is strictly less than one, i.e., |ad-cb|<1, then the IFS as a whole will converge to the attractor image from **any** initial image.

Fractal Image Compression

Fractal image compression is based on the observation that many natural scenes possess a detail-within-detail structure and IFS can generate fractal images that resemble natural scenes. The IFS can be reverse-engineered from the original image such that the corresponding IFS can be represented compactly for the original image. One unique feature of IFS based image compression is that we only need to store the transformation found within the image, and the decompression process is to apply these transformations to any given pattern so as to restore the original data.

The nature of how the Partitioned IFS is used for image compression is illustrated in the following figure. The basic idea is this: if finding self-similarity between an image in the whole and its parts is unrealistic, then seek self-similarity between larger parts and smaller parts. This is accomplished, as the name suggests, by partitioning the original image at different scales. Since images usually take the form of a rectangular



array of pixels, partitioning the original image into blocks is a natural choice. Using Jacquin's notation, the large partitions are called domain blocks, and the small partitions are range blocks. The range blocks evenly partition the image so that every pixel is included. The larger domain blocks may overlap, and need not contain every pixel. The goal of the compression process is to find a closely matching domain block for every range block. The set of domain blocks considered in this operation is called the *domain pool*.

When applied to gray scale images, the intensity value of a pixel, *z*, is treated as a third spatial dimension. That is, the blocks in the above figure are actually cuboids, although the original terminology remains. To achieve convergence the intensity value of a pixel must also be scaled and offset, i.e.

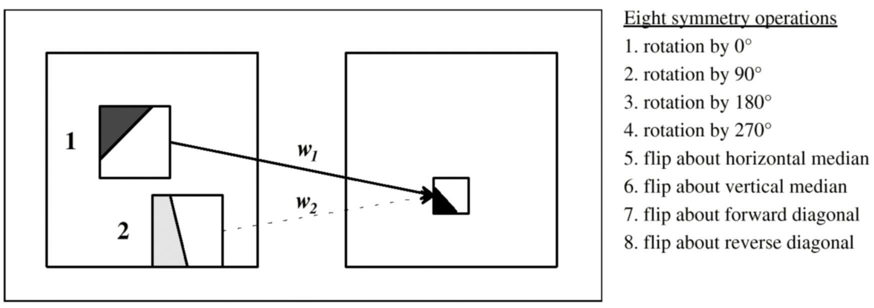
$$z' = s_i z + o_i$$

so that the affine transformation mentioned earlier becomes

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

The parameter s_i scales the pixel luminance and its effect is like the contrast knob on a television. When s_i is 0 the domain block maps to black, when equal to 1 it remains unchanged; between 0 and 1 the block loses contrast, and above 1 it gains contrast. The parameter o_i introduces an offset to the pixel luminance and is like the brightness knob on a television. Positive values of o_i brighten the block and negative values darken it. With contract and brightness control available, the extended affine transformation can accurately map grayscale domain blocks to grayscale range blocks. The above expression indicates that in order to compress a gray scale image, we have to find 8 variables for the transformation equation of each range block. If each coefficient is partitioned as 100 steps, the search space is of the size of 10^{16}, which is computationally very expensive.

To make the compression process tractable, Jacquin restricted equation so that domain blocks are always square (not rectangles or parallelograms), and always twice the size of range blocks. If the range blocks are, say, 8x8 pixels in size, then the



Eight symmetry operations
1. rotation by 0°
2. rotation by 90°
3. rotation by 180°
4. rotation by 270°
5. flip about horizontal median
6. flip about vertical median
7. flip about forward diagonal
8. flip about reverse diagonal

domain blocks are always 16x16. By doing so, it greatly reduces the size of the domain pool. This is favourable since it shortens the search time, but reconstruction quality suffers as optimal pairings may be excluded from consideration. One simple and effective way of improving coding quality is by allowing domain blocks to undergo an isometric symmetry operation prior to being transformed. The benefit of such an operation is illustrated in the following figure, where block 1 is first rotated clockwise by 270° to improve the similarity between it and the range block. If the eight symmetry operations are not allowed, then a less optimal pairing, transformation ***w_2***, must be used instead. With the above simplifications, the new transformation for the gray scale extension of the PIFS becomes

$$\begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} = \begin{bmatrix} 0.5 & 0 & 0 \\ 0 & 0.5 & 0 \\ 0 & 0 & s_i \end{bmatrix} M_i \begin{bmatrix} x \\ y \\ z \end{bmatrix} + \begin{bmatrix} e_i \\ f_i \\ o_i \end{bmatrix}$$

where M is a 3x3 matrix representing one of the eight symmetry operations. So the image compression process involves the identification of (e,f,M, o,s) for each range block, where the first two coefficients locate the domain block, the third applies a symmetry operation, and the last two introduce an offset and scaling factor. Because the numbers associated with these coefficients implicitly define a set of affine transformations, a fractal encoded image is sometimes described as being "composed of mathematical equations."

For a given image, if we use the following quantization for each of the five parameters,

- e_i 8 bits - 256 horizontal positions
- f_i 8 bits - 256 horizontal positions
- M_i 3 bits - 8 horizontal positions
- s_i 5 bits - sufficient from empirical tests
- o_i 6 bits - sufficient from empirical tests

we only need 4 bytes for representing each w_i. If the 256x256 image is divided into 8x8 range blocks, there are 1024 altogether, therefore the compressed image only requires 1024 x 32 bits = 4096 bytes (if uncompressed, it requires 64 Kbytes).

Matching Domain and Range Blocks

Before the actual compression can take place, we need to determine the scaling parameters for the best range-domain pairing. In the original algorithm of Jacquin, the goal is to minimize the Haussdorff distance (i.e. greatest pixel-to-pixel difference) between a specific range block and a candidate domain block. To do so, a small set of scale values {0.45, 0.60, 0.80, 0.97} are tested in sequence, and the one that produces the smallest Haussdorff distance is retained.

If, instead, the mean square error measure is used, the optimal scaling parameter can be determined algebraically. First, assume that the domain block $\boldsymbol{D_{xy}}$ has been reduced to the size of the range block $\boldsymbol{R_{xy}}$(by averaging 2x2 pixel cells), and **that they have been adjusted to a zero-mean intensity level**. Then, the mean square error between the blocks is

$$e = \frac{1}{n^2} \sum_{x,y\in\Omega} \left(s_i D_{xy} - R_{xy} \right)^2$$

By minimising e, one can solve for s_i. This can be achieved by taking the derivative of e with respect to s_i as zero, i.e.,

$$\frac{de}{ds_i} = \frac{2}{n^2} \sum_{x,y\in\Omega} \left(s_i D_{xy} - R_{xy} \right) D_{xy} = 0$$

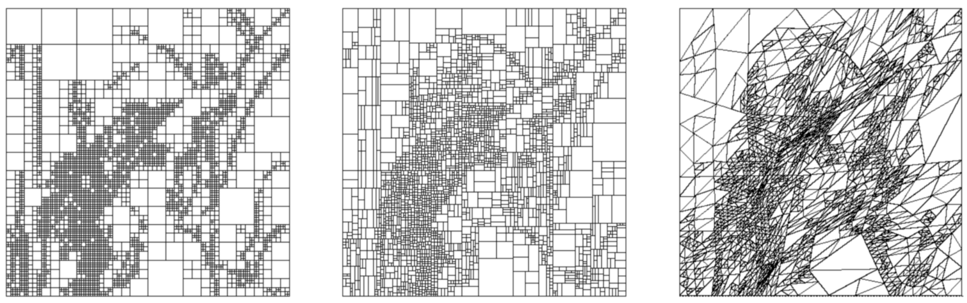
therefore,

$$s_i = \frac{\sum_{x,y\in\Omega} R_{xy} D_{xy}}{\sum_{x,y\in\Omega} D_{xy}^2}$$

In other words, the optimal scaling factor between a range block and a domain block is their inner product divided by the domain block sum-of-squares. This value is calculated for all candidate domain blocks, under all eight symmetry operations, in search of the smallest error. To ensure convergence of the decompression process it is common practice to force all component transforms to be contractive, that is, to restrict $|s_i| < 1.0$. This is not strictly necessary. In fact, releasing this constraint has been shown to improve image quality in some cases.

Search Strategy and Image Partitioning

With fractal image compression, two other important issues needs to be considered. One is the search strategy used for finding range-domain pairs, and the other is related to how to effectively partition the range blocks.



The development of an effective search strategy is important in that by using a 8x8 range block partitioning for a 256x256 image, 1024 pairings need to be established.

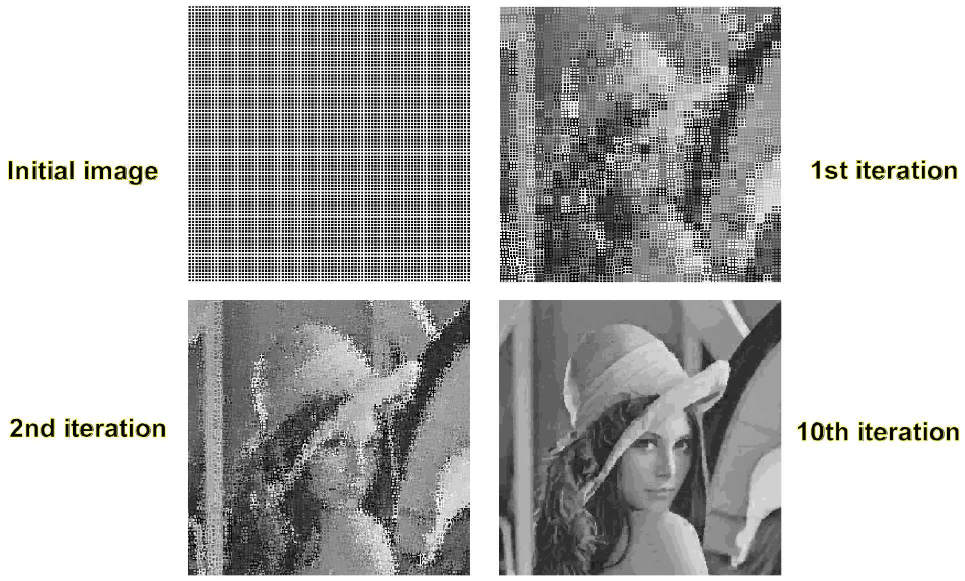
Even with Jacquin's simplification, the domain pool contains 8x(256-16+1)^2 = 464,648 elements (recall that eight symmetry operations are allowed). In total, 464,648 x 1024 = 475,799,552 possible pairings are required for testing, which requires 128 Gflops (floating point operations), and it takes 10 seconds on a Cray YMP-16 supercomputer ! (You can workout how long it will take for a 1024x1024 image). To address this problem, the following search strategies have been developed over the years.

- Heavy brute force - an exhaustive search method, don't expect the compression algorithm will work on a desktop computer
- Light brute force - look at x2, or x4 pixel locations during matching
- Restricted area search - restrict the search only to nearby areas
- Local Spiral Search - dramatically recudes the search time

The development of an efficient search strategy is only part of the story. In fact, the performance of the system is closely related to how one partitions the image. In general, the smaller the number of range blocks, the fewer the number of parings needs to be identified. The above figure shows three different partitioning schemes where the number of range blocks varies from 5008, 2910, to 2954.

Decompression

The decompression process usually begins by setting the computer's image buffer to a uniform mid-gray value. This is used as the seed image. During one iteration, the pixels of each range block in the transform list are evaluated. The result is used as the input for the second stage of iteration. The following figure shows that after just two iterations, the original image is recognizable, and after four the process will usually have converged (when eight bit precision is used per pixel). Due to the very nature of the IFS, the choice of the seed image is not important, they will all ultimately converge to the attractor image. Although the choice of seed image does not affect the outcome, it can affect how quickly the decompression process converges. One could instead begin with an all-black seed image, or an all-white one, but usually mid-gray is preferable. A successful way of increasing decompression speed, as first described by Beaumont is to begin with a low resolution version of the original. This is accomplished by modifying the PIFS equation so that o_i describes the mean value of a range block, rather than the relative offset from the corresponding domain block.



Limitations and Conclusions

In summary, fractal image compression is a promising technology, although it is still relatively immature. The technique is block based, lossy compression method. In general, decompression is very fast, but the compression process can be very slow. Compared to other parallel techniques, the following table summarised the pros and cons of each approach:

Image Category	Low Compression	High Compression
text and line art	poor	poor
computer graphics	poor - good	poor - fair
photo-realistic images	good	very good

Source

J Kominek, Advances in Fractal Compression for Multimedia Applications.