Fast Fractal Image Compression

Vrishab Commuri[1] **Scott Mionis**[2] **Bridget Tan**[1] **Weishan Wang**[1]
Department of Electrical and Computer Engineering[1] School of Computer Science[2]
Carnegie Mellon University
Pittsburgh, PA 15213
{vcommuri, smionis, bridgett, weishanw}@andrew.cmu.edu

Abstract

Fractal Image compression is a technique that attempts to learn a mapping $\mathbf{T}$ that can be iteratively applied to any source signal such that the output will converge on the encoded image, called an *attractor*; this requires capturing recursive structures in images in a computationally efficient way. In the majority of the Fractal Compression literature, the operator $\mathbf{T}$ is taken to be an affine transform, and the encoding process to determine $\mathbf{T}$ is on the order $O(n^2)$. In this report, we present a new technique, based on singular value decomposition, that achieves a computational complexity of $O(n)$ and produces results that are comparable to traditional fractal compression schemes as well as more conventional compression algorithms such as JPEG [2] and JPEG2000 [1].

1 Introduction

1.1 How Does Fractal Compression Work?

Fractal compression is based on the idea that a mapping $\mathbf{T}$ can be found that, given an arbitrary input, will converge to some fixed point called an *attractor*. In the context of image compression, the attractor is the image to be compressed. Once such a $\mathbf{T}$ is found, it can be iterated [1] given an arbitrary input and the result will eventually converge on the original image.

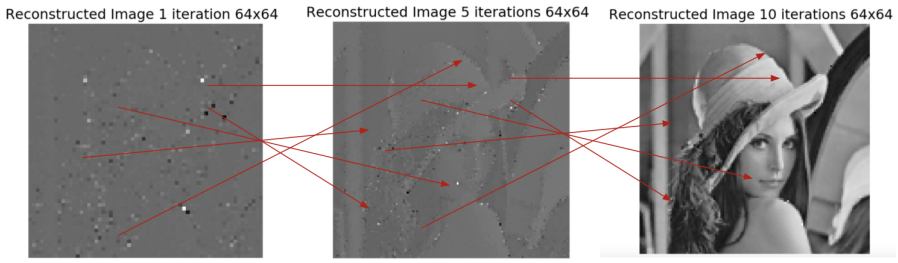

Figure 1: An illustration of fractal convergence through iteration. The leftmost plot presents the output of the mapping $\mathbf{T}$ after it is applied once to a all-zero input. The middle plot presents the result of iteratively applying $\mathbf{T}$ five times, where the input to each iteration is the output of the previous iteration. Notice how the encoded image of Lena is more clear after five iterations. The rightmost plot presents the converged image of Lena, which was recovered after iteratively applying $\mathbf{T}$ ten times.

However, fractal compression comes with some caveats. As we will see below, finding a suitable $\mathbf{T}$ can be computationally expensive, prohibitively so for larger images (informally classified as those with more than 128x128 pixels). Additionally, since $\mathbf{T}$ will be iterated, it must be constrained so that signal values do not increase with each iteration; that is, $\mathbf{T}$ must be *contractive*.

1.2 A Simple Fractal Compression Algorithm

So that the reader can acquire a more complete understanding of fractal compression, which had its zeitgeist in the 1980s and 90s and is quite uncommon today, we present a simple outline of how traditional fractal compression schemes work.

An input image is partitioned into nonoverlapping square blocks. The partitioning is carried out twice; once for the range blocks, which each have a size $M \times M$, and once again for the domain blocks, which each have a size $2M \times 2M$. To encode each range

[1] For example, $\mathbf{T}(\mathbf{T}(x))$ comprises two iterations given the input x.

block we downsample all of the domain blocks so that their dimensions match those of the range blocks, and then we find the domain block that, when scaled by the scalar α and biased by the constant β, is the closest match to the range block. (In this simple case, optimal values of α and β have simple closed-form solutions.)

We can formulate the above mathematically, defining operators to perform each of the steps in the encoding process. This notation will be useful for developing a new encoding scheme, which we will cover later in this report. We denote the entire above process as the mapping $\mathbf{T}$, and we denote the input image x. This yields the following encoding equation:

$$\mathbf{T}x = \sum_{i=1}^{N} \alpha_i \mathbf{P}_i \mathbf{D} \mathbf{F}_i x + \sum_{i=1}^{N} \beta_i \mathbf{P}_i \mathbf{1} \tag{1}$$

Where $\mathbf{F}_i$ fetches the closest-match domain block for the ith range block. $\mathbf{D}$ performs a downsampling operation on the fetched block. $\mathbf{P}_i$ places the downsampled block at the location of the ith range block and zeros out all other pixels. And α_i scales the placed block accordingly. The right hand term produces a bias of the ith block by the constant β_i. In that expression, $\mathbf{1}$ simply denotes a vector of ones.

From the above, we can readily identify the encoding performance bottleneck as the $\mathbf{F}_i$ operator. Finding the optimal domain block is an $O(N^2)$ operation, since for each range block we must examine each domain block. Each of the remaining operations can be performed in constant or linear time.

Additionally, we note that the convergence properties of equation 1 have been well-studied, and in most cases the algorithm produces a contractive mapping. If the mapping is not contractive, then it can be made contractive by thresholding the value of α at a tradeoff to reproduction quality.

2 Related Work

Fractal Image Compression (FIC) was first proposed in 1988 by Michael F. Barnsley[7]. The algorithm was then implemented as an iterated function system (IFS) similar to that described above; images are divided into patches, downsampled, and then coefficients for upsampling the patch are found to minimize error. This technique was explored theoretically, but the first fully automated algorithm for fractal image compression was not proposed until 1992 by Jacquin[8] using the "Partial Iterative Function System," or PIFS[6]. This algorithm certainly improved the viability of FIC, but several problems remained, foremost being the disproportionately large compression time. In terms of common ground for creating new algorithms, most modern algorithms are based on the baseline algorithm we derived above, which is best described in the definitive text on the subject by Yuval Fischer[10]. Recent surveys of the technology have successfully identified bottlenecks and areas of improvement for the technique, but no research has advanced the algorithm enough to make it a viable substitute for more poplar standards [5][3].

3 Fast Fractal Image Compression

In our work, we replace the performance bottleneck of the fetch operator $\mathbf{F}_i$ by devising a different method to enable us to efficiently compute a representation for each range block.

Repeating the segmentation scheme described in the baseline implementation, we then collect all the domain blocks $\mathbf{d}_i$ together in a matrix and downsample each block with operator $\mathbf{D}$. For the sake of simplicity, we use a vectorized notation for indexing range and domain blocks.

$$\mathbf{B} = \left[\mathbf{D}\mathbf{d}_1 \; \mathbf{D}\mathbf{d}_2 \; ... \; \mathbf{D}\mathbf{d}_M\right]$$

Now, we can perform singular value decomposition on $\mathbf{B}$.

$$\mathbf{B} = \mathbf{U}\mathbf{S}\mathbf{V}^H$$

Where the left singular vectors $\mathbf{U}$ represent an orthonormal set of bases for the downsampled domain blocks of the image. Thus, we can simply represent each range block $\mathbf{r}_i$ as a weighted combination of bases in $\mathbf{U}$ instead of applying the fetching operation.

$$\mathbf{r}_i = \mathbf{U}\mathbf{w}_i$$

And since $\mathbf{U}$ is orthonormal, each range block can be encoded by the weights

$$\mathbf{w}_i = \mathbf{U}^T\mathbf{r}_i$$

and compression can be achieved by retaining a subset of the elements of the w_i vectors. Now we can define a new operator $\mathbf{T}_S$:

$$\mathbf{T}_S x = \sum_{i=1}^{N} \mathbf{P}_i S(\mathbf{D}x)\mathbf{w}_i$$

Above, $\mathbf{D}$ is the downsampling operator, as before. The operator S computes the left singular vectors, given by $\mathbf{U}$, of the collection of domain vectors. The left singular vectors are then right multiplied by the weight vector, $\mathbf{w}_i$, corresponding to the ith range block. The operator $\mathbf{P}_i$ simply places the result at the location of the ith range block as before.

The operator $\mathbf{T}_S$, much like the simpler operator $\mathbf{T}$ that we started with, can be iterated to converge on the encoded image. The critical difference being that $\mathbf{T}_S$ is far more computationally efficient than $\mathbf{T}$. We will examine the performance and convergence characteristics of $\mathbf{T}_S$ in more detail below.

3.1 Performance

Because S performs singular value decomposition on all of the domain vectors from the input, the value of S can be precomputed before any range blocks are processed. So for each iteration of $\mathbf{T}_S$, S is a one-time computation with complexity $O(\min(M^2L, L^2M))$ where $L \times M$ is the dimension of $\mathbf{B}$. At first glance, it may seem like we have made scant performance gains by using SVD, but it is important to remember that L is the number of pixels in the downsampled domain block. Since the block size is fixed, the size of L does not change as the size of the image is increased (though, of course, M does). Therefore the complexity of SVD for our method is given by $O(M)$. In general, it is not necessary to perform SVD on the entire $\mathbf{B}$ matrix; it suffices to perform SVD on a randomly-sampled subset of the columns of $\mathbf{B}$. This will yield a good approximation to the full SVD, while curtailing the computational costs.

Because the output of S is an orthonormal matrix, it follows that the computation of the weights $\mathbf{w}_i$ is a simple matrix multiplication for each range block, and since each individual range block is constant in size (irrespective of number of blocks), we can treat this operation as $O(1)$. Thus, the computation is linear in the number of range blocks, or $O(N)$.

The remaining operators can all be computed in constant time as well, meaning that the overall algorithmic performance of our method is $O(M + N)$, or linear with respect to the number of range and domain blocks in the image.

3.2 Convergence

In order to determine whether the operator $\mathbf{T}_S$ converges to the attractor image, we must determine whether $\mathbf{T}_S$ is contractive. An operator being contractive means that, for each iteration, the output of the operator is strictly closer to the attractor than its input. This implies that, after some number of iterations, the output will converge on the encoded image. Unfortunately, it is a known result that arbitrary operators, such as the $\mathbf{T}_S$ we have defined as well as the majority of operators used in fractal compression, are not generally contractive.

Since contractivity is a sufficient condition to converge to the encoded image – according to the Contractive Mapping Fixed-Point Theorem and the Collage Theorem [10] (Theorem 2.3) – we impose a simple thresholding constraint on our compression algorithm to ensure that the mapping never produces an output that diverges from the encoded image. Such constraints are commonplace in various fractal compression schemes.

To determine where in our pipeline to impose the constraint, we must determine which of the above operations results in a scaling of the input. Scalings that are greater than 1 must be thresholded in order to prevent divergence. Recall the simplified expression for coding each range block:

$$\mathbf{r}_i = \mathbf{U}\mathbf{w}_i$$

We observe from the above equations that

$$\begin{aligned}
\mathbf{B} &= \mathbf{U}\mathbf{S}\mathbf{V}^H \\
&= \mathbf{U}\mathbf{R} \\
\implies \mathbf{U} &= \mathbf{B}\mathbf{R}^{-1} && \text{(rearranging)} \\
\implies \mathbf{r}_i &= \mathbf{B}\mathbf{R}^{-1}\mathbf{w}_i && \text{(substitution)}
\end{aligned}$$

Since $\mathbf{B}$ was composed from the domain blocks, it follows that $\mathbf{R}^{-1}\mathbf{w}_i$ is a transform from the domain $\mathbf{B}$ to the range block $\mathbf{r}_i$. Therefore, any scaling that might violate contractivity can be observed through the elements of the vector $\mathbf{R}^{-1}\mathbf{w}_i$. Simply ensuring that the elements of $\mathbf{R}^{-1}\mathbf{w}_i$ are less than 1 is sufficient to ensure that we have a contractive mapping, and given how the weights are derived, that the algorithm converges to the target atttractor.

We note that in our experiments, interestingly, convergence can be achieved in all but the most pathological of cases without the imposition of any constraints at all.

4 Datasets

We used the University of Waterloo's Greyscale Set 1 dataset [11]. This dataset contains 12 PGM images comprised of natural images, synthetic images, and a montage of smaller photos. It also contains standard image processing images like Lena. Each of these is a 256 x 256 grayscale image, but we have obtained 512x512 and larger versions of standard images like Lena and Mandrill to test our algorithm on larger examples as well. In addition to the Waterloo dataset, we have utilized the Labeled Faces in the Wild (LFW) dataset from the University of Amherst [12] for some 64x64 greyscale facial images.

It is important to note that we did not train any form of model on this data, since our algorithm is not data-driven. Hence, the dataset was primarily used for evaluation. Our algorithm input is an image from our dataset, and the output is a compressed representation that yields an accurate reconstruction of the original image when decompressed. Our dataset required no pre-processing or post-processing.

5 Results

We evaluated our algorithm on a number of images from the Waterloo dataset using Peak Signal to Noise Ratio (PSNR) and Compression Ratio as our evaluation metrics. These metrics specifically were chosen since they are the most common metrics used in the domain of image compression algorithms. Of additional interest is algorithm runtime, which we examine in the next section.

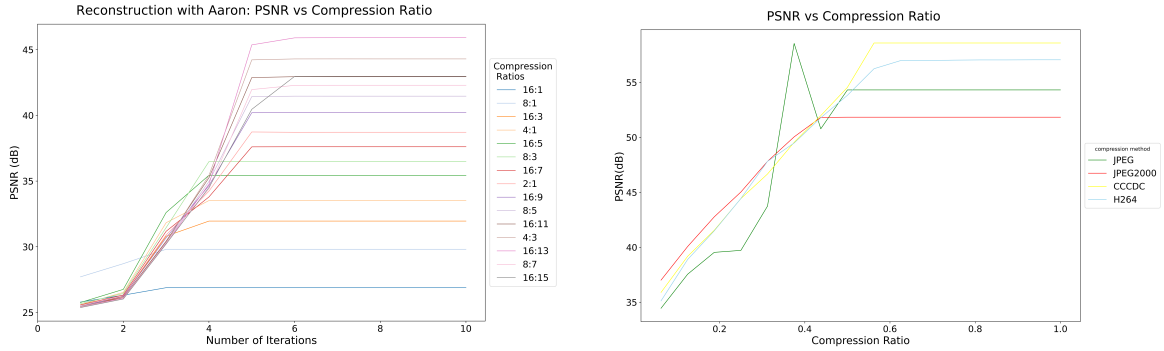

Figure 2: PSNR vs Compression Ratio for our fractal approach (left) versus Traditional (DCT and Wavelet-based) approaches (right). In both experiments, the image encoded was a 512x512 greyscale image of Lena. For the reconstruction, a 64x64 image of Aaron Eckhart's face from the LFW dataset was used as input. Notice that in the left plot, higher PSNRs are achieved for lower compression ratios. We note that our method (left) achieves comparable results to established methods such as JPEG and H.264 (right).

Above are plots showing the PSNR values for the Fast Fractal algorithm (left, plotted over iteration count) and various Traditional algorithms (right) plotted against compression ratio.

We can see that for very aggressive compression ratios, most existing approaches outperform our algorithm. For JPEG (which takes the DCT of image segments) this is expected because the algorithm compresses blocks (or "tiles") individually, meaning that information is only discarded independently in each block, and that artefacts are isolated to the block from which they originate. This reasoning can also extend to JPEG 2000. For our algorithm, we take the SVD bases from the entire image and use this dictionary to encode each range block. We believe that this approach is more fragile because even though both JPEG and Fast Fractal Compression are throwing away the same amount of data, JPEG fits each tile to a fewer number of bases than we do, and so each of their basis vectors is more expressive.

For less aggressive compression ratios, we achieve extremely comparable results. This is encouraging, especially in the middle ranges, because it shows the viability of our technique. While we certainly hoped to outperform existing techniques, we believe there are additional benefits that are shown from our results. Specifically, as seen by the oscillating JPEG data, we believe our approach may be easier to reason about deterministically, since we reliably see a monotonic increase in PSNR as compression ratio becomes less severe.

Also of note is the convergence rate of our algorithm across compression ratios. As expected, our algorithm converges faster with much lower compression ratios, since our peak PSNR is much lower and we hit it sooner. It is also worth noting that the convergence rate (PSNR/number of iterations) is fairly constant across different compression ratios.

6 Discussion and Analysis

As seen above, we achieve comparable PSNR and compression results to existing techniques such as JPEG and JPEG 2000. This not only illustrates the viability of fractal techniques, but strongly indicates that the field is worth exploring further since it has since been neglected in favor of more modern techniques.

Table 1: 128x128 Image Benchmarks

	Time to Encode 4x4 Blocks (sec)	Time to Decode 4x4 Blocks (sec)	Time to Encode 2x2 Blocks (sec)	Time to Decode 2x2 Blocks (sec)
Standard	132.30	1.09	2001.30	4.53
Standard + GPU	27.59	0.98	437.80	3.96
Fast	0.01	0.11	0.08	0.58
Fast + GPU	0.89	0.12	0.77	0.47

Encoding and decoding speed comparison between a standard fractal image compression algorithm (described in section 1.2) and our fast implementation adapted from existing parallelization methods [4]. Both implementations encoded the same 128x128 image during encoding, and both utilized the same 64x64 image during decoding. The number of iterations for decoding was fixed to 10 in both cases. Both algorithms were benchmarked on the same machine, a Dell XPS 15 9560 with an Nvidia GeForce GTX 1050 GPU. Notice that our fast implementation considerably outperforms the standard implementation, even with GPU acceleration. We note that the GPU acceleration does not generally improve performance for small images, unlike in the case of larger images.

Table 2: 512x512 Image Benchmarks (Fast Implementation Only)

	Time to Encode 4x4 Blocks (sec)	Time to Decode 4x4 Blocks (sec)	Time to Encode 2x2 Blocks (sec)	Time to Decode 2x2 Blocks (sec)
Fast	0.068	5.89	5.80	48.85
Fast + GPU	1.40	4.30	3.78	24.06

Encoding and decoding speed evaluation for fast fractal compression. The standard fractal compression algorithm was not evaluated since it would not complete within a reasonable timeframe (less than 10 hours). The encoded image was 512x512 in dimension, and a 64x64 image was used for decoding. The algorithm was benchmarked on the same machine as in Table 2, a Dell XPS 15 9560 with an Nvidia GeForce GTX 1050 GPU. Notice that the fast implementation performs quite well, even for this larger image size. Also note that GPU acceleration has drastically improved the encoding and decoding time in the 2x2 block case.

In addition to producing comparable results, our other goal was to make the runtime of the algorithm viable. Our algorithm is significantly faster than prior fractal compression methods even when they utilize GPU acceleration.

Important to note is the impact of the starting bases on convergence speed. In the plot above, we show the convergence rates of Lena across compression ratios, starting from an image of a brick wall. Noticeably, it takes more iterations to achieve a reasonable result if the starting bases are sufficiently different from the target attractor.

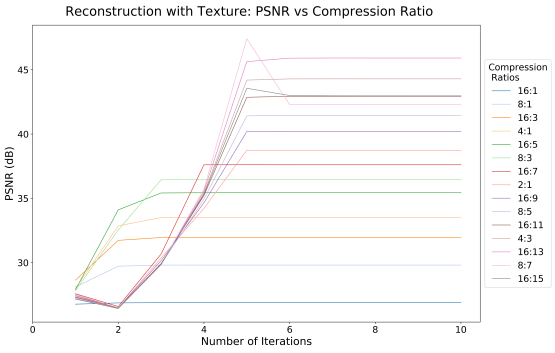

Figure 3: Reconstructing Lena with Brick Wall texture

Fundamental limitations of our approach are that the algorithm only works best if you have bases relatively related to the image you are trying to decompress. While this is not a problem if you have prior knowledge, it means that decompressing a generic image may yield suboptimal results. To solve this problem, we intend to try DCT bases that are data-agnostic, which would make the algorithm easier to reason about since starting bases would no longer matter. (This method would still differ from JPEG, since we would retain our iterative framework.)

Additional limitations are outlined in the results section, foremost of which is our underperformance when applied with aggressive compression ratios. We believe this to be relatively inherent to the algorithm, as stated, and thus we look to more creative ways of solving this problem such as using better bases or compressing our weight vectors thoughtfully (regularization, or drawing concepts from information theory) versus just thresholding the components.

The fact that fractal compression works as well as it does speaks volumes about the inherent, repetitive structure of image data. In the spatial domain (i.e. baseline approach), we can classify a complex image such as Lena with a single, simple transform. Our success with this algorithm also provides insight into the power IFS systems in general, and hopefully sheds light on a hitherto niche technique

To view the existing public codebase please visit: https://github.com/Scottamattic/18797_FastFractalCompression_Group7_Repository

References

[1] J. Zhu, "Image compression using wavelets and JPEG2000: a tutorial," in Electronics Communication Engineering Journal, vol. 14, no. 3, pp. 112-121, June 2002, doi: 10.1049/ecej:20020303.

[2] G. K. Wallace, "The JPEG still picture compression standard," in IEEE Transactions on Consumer Electronics, vol. 38, no. 1, pp. xviii-xxxiv, Feb. 1992, doi: 10.1109/30.125072.

[3] Joshi M., Agarwal A.K., Gupta B. (2019) Fractal Image Compression and Its Techniques: A Review. In: Ray K., Sharma T., Rawat S., Saini R., Bandyopadhyay A. (eds) Soft Computing: Theories and Applications. Advances in Intelligent Systems and Computing, vol 742. Springer, Singapore. https://doi.org/10.1007/978-981-13-0589-4_22

[4] Dan L., Peter K J., (2007) A Survey of Parallel Algorithms for Fractal Image Compression in Journal of Algorithms and Computation Technology, doi: 10.1260/174830107781389021

[5] Gauri Joshi, (2013) Fractal Image Compression and its Application in Image Processing in Internation Journal of Science and Research, issn: 2319-7064

[6] Zhou Wang and Ying Lin Yu "Partial iterated function system-based fractal image coding", Proc. SPIE 2751, Hybrid Image and Signal Processing V, (7 June 1996); https://doi.org/10.1117/12.242020

[7] M.F. Barnsley, Fractals Everywhere, New York: Acacemic, 1988.

[8]Arnaud Jacquin; “Image Coding Based on a Fractal Theory of Iterated Contractive Image Transformations,” IEEE Transactions on Image Processing, vol. 1, no. 1, January 1992, pp. 18-30.

[9] Kominek, J. Advances in fractal compression for multimedia applications. Multimedia Systems 5, 255–270 (1997). https://doi.org/10.1007/s005300050059

[10] Y. Fisher, Fractal Image Compression: Theory and Application. New York: Springer, 1995.

[11] University of Waterloo, "Image Repository", 2009. [Online]. Available: http://links.uwaterloo.ca/Repository.html.

[12] Gary B. Huang, Marwan Mattar, Honglak Lee, and Erik Learned-Miller. Learning to Align from Scratch. Advances in Neural Information Processing Systems (NIPS), 2012.