Lean domain pools for fractal image compression[1]

Dietmar Saupe

Universität Freiburg
Institut für Informatik
Am Flughafen 17, 79110 Freiburg, Germany

[1]This paper is from the Conference Proceedings of SPIE Electronic Imaging'96, Science and Technology, Still Image Compression II, San Jose, January 1996, Volume 2669.

Abstract

In fractal image compression an image is partitioned into ranges for each of which a similar subimage, called domain, is selected from a pool of subimages. A typical choice for the domain pool may consist of all square subimages of a particular size. However, only a fraction of this large pool is actually used in the fractal code. This subset can be characterized in two related ways: (1) It contains domains with relatively large intensity variation. (2) The collection of used domains are localized in those image regions with a high degree of structure. Both observations lead us to improvements of fractal image compression. Firstly, we accelerate the encoding process by a priori discarding those domains from the pool which are unlikely to be chosen for the fractal code. This comes at the expense of a slight loss in compression ratio. In our empirical studies (using Fisher's adaptive quadtree method) we have found that a twofold acceleration leads to a drop of only 2 to 3% in the compression ratio while the image quality even improves by 0.1 to 0.2 dB. Secondly, the localization of the domains can be exploited for an improved encoding in effect raising the compression ratio back up without any penalty.

Keywords: image compression, fractals, domain pool, block variances

1 INTRODUCTION

Fractal image compression [1, 4, 7] is capable of yielding competitive rate-distortion curves, however, it suffers from long encoding times. Therefore, large efforts have been undertaken to speed up the encoding process. Most of the proposed techniques attempt to accelerate the searching and are based on some kind of feature vector assigned to ranges and domains. The features can be discrete (leading to classification and clustering methods) or continuous (yielding functional or nearest neighbor methods).[13, 14] When applied, these methods provide greater speed which is traded in for a loss in image fidelity and compression ratio.

A different route to increased speed can be chosen by *less* searching as opposed to *faster* searching. This means that if we can devise a technique to estimate a priori whether a given domain will be used for the fractal code or not, then we can exclude all unlikely domains. In this way greater speed is achieved by restricting the search to a reduced domain pool. Of course, the search in the reduced domain pool may then be supported by a method of the other kind, e.g., by classification. Several possible approaches have been reported in the literature.

- One may argue that those domains that are close to a given range in the image (e.g., domains that overlap the range) are especially well suited as a partner for the given range, thereby localizing the domain pool relative to the range (see, e.g., the work of Monro and Dudbridge [10, 11] and Barthel et al [2]). This is an adaptive domain pool reduction; for each range a different domain pool is constructed.

- Another adaptive variant has been suggested in a functional method by Bedford, Dekking, and Keane [3, 13]. Depending on the range the domain pool is shrunk by excluding a number of domains that do not satisfy a condition which involves certain inner products which are independent of the range and can be calculated for all domains in a preprocessing step. These excluded domains are guaranteed not to be optimal for the given range; thus, no image or compression degradation can occur with this method.

- Complementing these methods one can work with a fixed domain pool, which is initially scanned once in order to discard domains that are unlikely to be of any use.

In principle, this last approach has already been implemented in the early work of Jacquin [7]. He used a classification scheme coming from a study of Ramamurthi and Gersho [12] which classifies domain blocks according to their perceptual geometric features. Three major types of blocks are differentiated: shade blocks, edge blocks, and midrange blocks. In shade blocks the image intensity varies only very little. Since ranges that would be classified as shade blocks can be approximated well by a scaled constant fixed block, it is not necessary to search for a corresponding domain. Thus, in this scheme all domains classified as shade blocks are never used and effectively are excluded from the domain pool. However, in Jacquin's approach the class of shade

blocks cannot be very large. For example, only 11% of all blocks for the 256×256, 6 bit/pixel Lenna have been classified as shade blocks in Jacquin's work [6]. The reason for this is that otherwise too many range blocks would be classified as shade blocks and thus be coded as a constant fixed block yielding poor approximations. Therefore, no variations of shade block definitions have been investigated in these studies.

In this article we consider a parametrized and non-adaptive version of domain pool reduction. Here we allow an adjustable number of domains to be excluded (ranging from 0% to almost 100%) and investigate the effects on computation time, image fidelity and compression ratio.

We will see that there is no need for keeping domains with low intensity variance in the pool. Thus, we propose to eliminate a fraction $1 - \alpha$, $\alpha \in (0, 1]$ of the domain pool consisting of the domains with least variance. In this way we remove the mostly useless domains from the pool achieving a lean and more productive domain pool. Using the adaptive quadtree method of Fisher [4, Appendix A] we will show the following:

1. The computation time scales linearly with α.

2. Even for low values of α, e.g., $\alpha = 0.15$, there is no degradation in image quality. On the contrary, the fidelity improves slightly.

3. For medium values of α, e.g., $\alpha = 0.50$, the compression ratio suffers a little, decreasing by about 2%.

The fractal code for an image essentially consists of the partitioning of the image into ranges and the data for one affine transformation per range. These data are given by an offset o (typically 7 bits), a scaling factor s (5 bits), a domain D_k from the domain pool and the code for an isometry (3 bits). The intensity values in the coded range are then taken from the scaled, transformed copy of the domain plus the added offset. The domains from the pool are indexed and referenced by that index. Let us discuss the simple example of a grey scale image of resolution 512×512 with a domain pool of non-overlapping domains of size 8×8. Thus, there are $64^2 = 4096 = 2^{12}$ domains in this pool altogether and the storage of one domain index costs 12 bits. It turns out that only a certain fraction, e.g., say 1000, of these domains are used. We propose to make use of this observation in the following way. We use a standard "white block skipping" quadtree storage scheme to identify the 1000 domains used out of the total of 4096. With the quadtree on hand we can now code indices of domains in the range from 1 to 1000, costing only 10 bits each. Thus, we save 2 bits per transformation. If M is the total number of ranges we have an overall file size reduction if the code for the quadtree does not exceed $2M$ bits. This approach will become beneficial in combination with lean domain pools since the collection of domains used will have even more structure yielding a smaller code for the domain quadtree.

Parallel to this work researchers are currently investigating other methods for domain pool reduction for fractal image compression. Kominek [8, 9] and Signes [15] propose to remove domain blocks from the domain pool if they can be covered well by

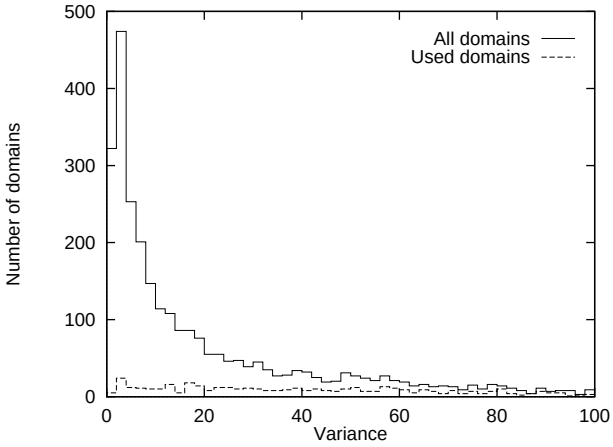

Figure 1: Histogram of variances in the domain pool of domain blocks of size 8×8 versus that for domains actually used in an adaptive quadtree fractal code of Lenna.

other blocks still in the pool. Also high variance domain blocks are generally favored over low variance blocks; however, no analysis of the time/performance tradeoff is attempted for this approach.

The remainder of this article is organized as follows. In Section 2 we present the details and results of the domain pool reduction by eliminating domains with low intensity variation. In Section 3 the optimized domain storage scheme is presented with results.

2 ACCELERATION BY LEAN DOMAIN POOLS

In a first experiment we check our hypothesis that there is no need for keeping domains with low intensity variance in the pool. We carry out a fractal encoding of a test image using the adaptive quadtree method and record a histogram of intensity variances of blocks of size 8×8 from the domain pool and also the corresponding histogram for the variances of those domains actually used in the code (see Figure 1). The result is very clear. There is a very large subset of domains in the pool with small variances while there is no such trend in the histogram for the blocks used. Thus, we may indeed expect that discarding a large fraction of low variance blocks will effect only a few ranges. For these ranges a suboptimal domain with a larger variance may be found. If, however, there is no longer a domain available in the pool which admits a collage error within the prescribed tolerance, then the range needs to be subdivided into four smaller ranges.

In the main study of this paper we scan each domain pool (i.e., the pools for block sizes 8×8, 16×16, 32×32, and 64×64) and keep only a fraction α, $\alpha \in (0, 1]$, of them in the pool, namely those domains that have the largest variances. For differing choices of the parameter α we compute the fractal code and record the computation time used, the peak-signal-to-noise ratio (PSNR), and the compression ratio (see the four left columns in Tables 1 and 2). The results are as follows:

	Results for 512 × 512 Lenna				
	CPU	PSNR	Compression		
α	time		comp.	new	bytes
	sec	dB	ratio	ratio	saved
1.00	15.2	32.73	14.88	14.84	−39
0.90	14.0	32.71	14.86	14.84	−34
0.80	12.6	32.75	14.85	14.83	−21
0.70	11.3	32.76	14.83	14.82	−7
0.60	10.1	32.80	14.75	14.77	24
0.50	8.7	32.87	14.57	14.62	60
0.40	7.4	32.90	14.45	14.56	137
0.30	6.0	32.93	14.19	14.88	856
0.20	4.6	32.88	13.49	14.23	1009
0.15	3.9	32.78	13.10	13.89	1135
0.10	3.1	32.53	12.64	13.98	1982
0.08	2.8	32.40	12.36	13.69	2070
0.06	2.4	32.03	12.21	14.13	2921
0.04	2.1	31.80	11.86	13.79	3101
0.02	1.7	31.03	11.39	13.86	4103

Table 1: Performance of the adaptive quadtree method with lean domain pools. The parameter α indicates the fraction of domains which are kept in the pool. The time is measured on an Indy R4600SC of Silicon Graphics in seconds. The compression ratio is in the fourth column. When applying the optimized coding procedure for the domains, we obtain the ratios of the fifth column with the difference in file size measured in bytes indicated in the last column.

1. **Time.** Regarding the computation times there seems to be an overhead of about 1 to 2 seconds. The remaining *time scales linearly* with the parameter α. This is as expected since the major computational effort in the encoding lies in the linear search through the domain pool.

2. **Fidelity.** The quality of the encoding in terms of fidelity measured by the *PSNR increases* by 0.1 to 0.2 dB when lowering α (except for the Baboon image). This is caused by the fact that some larger ranges can be covered well for $\alpha = 1.0$ by some domains which are removed from the pool at smaller values of α. As a consequence some of these ranges are subdivided and their quadrants can be covered better by smaller domains than the large range previously. This mechanism works for values of α down to about 0.15.

3. **Compression.** The range splitting mentioned above also increases the number of ranges, thus, causes the compression rate to decrease slightly. For example, this drop is 1 to 2% at $\alpha = 0.5$ and 2 to 9% at $\alpha = 0.2$. It is remarkable that only relatively little loss in overall quality of the encoding is encountered for speed up factors of 10 and higher.

α	CPU time sec	PSNR dB	Compression comp. ratio	new ratio	bytes saved
	Results for 512 × 512 Peppers				
1.00	16.6	32.43	15.20	15.06	-161
0.50	10.0	32.49	14.93	14.95	17
0.20	5.1	32.55	14.00	14.73	921
0.10	3.3	32.35	13.22	14.56	1828
0.05	2.4	32.11	12.31	14.22	2859
	Results for 512 × 512 Baboon				
1.00	33.2	25.15	5.68	5.59	-727
0.50	17.3	25.13	5.61	5.75	1148
0.20	8.0	24.81	5.55	5.93	3028
0.10	4.7	24.37	5.52	6.16	4915
0.05	3.3	23.87	5.50	6.42	6766
	Results for 512 × 512 Boats				
1.00	25.6	32.03	10.11	10.18	185
0.50	14.6	32.07	10.06	10.19	335
0.20	7.3	31.89	9.92	10.51	1476
0.10	4.4	31.53	9.82	10.85	2531
0.05	2.8	30.95	9.79	11.29	3562
	Results for 512 × 512 F16				
1.00	21.3	32.86	12.55	12.60	75
0.50	12.6	32.97	12.42	12.55	211
0.20	6.2	32.94	12.05	12.75	1186
0.10	3.8	32.63	11.98	13.23	2079
0.05	2.4	32.13	11.82	13.65	2967

Table 2: Results as in Table 1 for a few more test images.

3 DECREASED BITRATE BY EXPLOITING SPATIAL DOMAIN ENTROPY

Figure 2 shows the domains of size 8 × 8 that are used in the fractal code of Lenna (from the first row in Table 1). As expected the indicated domains are located mostly along edges and in regions of high contrast of the image. These black squares can be interpreted as a bitmap (of resolution 64×64 in this case) and the goal of the procedure outlined in the introduction is to store this bitmap efficiently. Then the number of bits required to identify a particular used domain is reduced. If the structure of the bitmap is strong then these savings are greater than the overhead necessary for coding the bitmap and an overall reduced file size for the code can be achieved.

We use the "white block skipping" (wbs) quadtree storage scheme described in

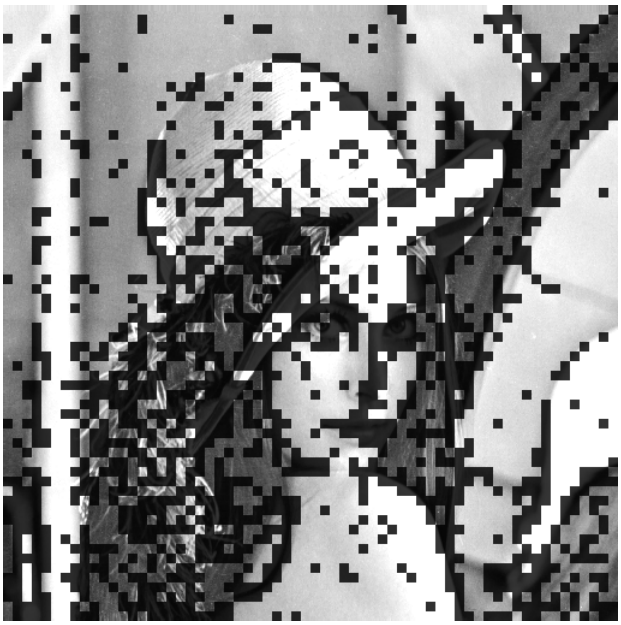

Figure 2: Domains of size 8×8 that are used for a fractal code of 512×512 Lenna are shown in black.

Gonzales and Woods[5, page 354]. For a bitmap of size $2^k \times 2^k$ we proceed recursively starting with the block given by the entire bitmap. A solid white block is coded as 0, all other blocks are coded with a prefix 1 and followed by the four codes of their four subquadrants, which are generated in the same way until a subblock of size 1 is reached which is coded as 0 (white) or 1 (black). For an example, see Figure 3.

The last two columns in Table 1 report the results of this procedure for the Lenna test image. For the case without domain pool decimation ($\alpha = 1.00$) there are no savings. The costs for storing the wbs quadtree outweigh the savings from shorter domain codes. However, as we decrease the value of α below 0.7 we are obtaining some gain in compression. An especially notable result is obtained for $\alpha = 0.30$. The new enhanced storage scheme completely makes up for the loss of compression which occurred due to the domain pool decimation. Thus, in effect, when comparing with the original method (no domain pool decimation, no enhanced domain storage, line 1 in Table 1) we arrive at a fractal encoding with exactly the same compression ratio of 14.88, an improved PSNR (by 0.2 dB) and a computation time reduced from 15.2 seconds down to only 6.0 seconds!

We have carried out the same experiment with domain pools enlarged by factors of 4 and 16. In these cases the break-even point of $\alpha = 0.70$ is reduced to 0.25 and 0.15 respectively. Thus, the method seems to be applicable in situations where either the domain pool is not very large or when extremely fast encodings are desired (e.g., by choosing $\alpha \leq 0.05$) and quality can be compromised. In a production implementation one could carry out the wbs quadtree coding, check whether it yields any savings or not, and then use the better storage scheme.

Also we note that with strongly decimated domain pools we have that domains

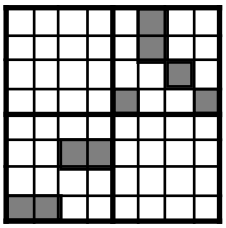

Figure 3: Example for the white block skip coding of a bitmap of size 8×8. The code is

$$\begin{array}{ll} \mathbf{1} & [\mathbf{1}\,(0)\,(\mathbf{1}\,1001)\,(\mathbf{1}\,0010)\,(\mathbf{1}\,0101)] \\ & [0] \\ & [\mathbf{1}\,(\mathbf{1}\,0011)\,(0)\,(\mathbf{1}\,0011)\,(0)] \\ & [0] \end{array}$$

obtained by recursively processing subblocks counterclockwise each time starting from the corresponding upper right quadrant. For better readability we have written prefix codes in bold face, and bracketed the codes for the main- and sub-quadrants.

are used more often than once in the fractal code. Thus, standard entropy coding (Huffman or arithmetic coding) of the domain indices will yield additional savings in storage. We have not yet implemented this option.

4 CONCLUSION

We have introduced the concept of lean domain pools in which a fraction $1 - \alpha$ of low intensity variance domains are discarded from the domain pool. This reduces the time complexity of the encoding by a factor of roughly α. With this procedure, implemented in an adaptive quadtree fractal encoder, the tradeoff between increased speed and quality in terms of fidelity and compression has been investigated. For a twofold speedup we gain 0.1 to 0.2 dB PSNR and loose up to 3% of the compression rate. For up to fourfold speedup we typically still gain 0.1 to 0.2 dB PSNR and loose up to 10% of the compression rate. Also we have introduced a new way for the specification of the domains used for the fractal code which improves efficiency when the collection of domains used show a high degree of structure which often is the case when lean domain pools are used. Our technique is simple and can easily be incorporated into existing fractal coding programs, even in combination with other acceleration methods. In summary, the lean domain pools introduced in this work cause only negligible or no loss with $\alpha = 0.5$ thereby halfing the encoding time. Moreover, with smaller values of α, the method also has a strong potential in applications where extremely fast encodings are desired and some quality can be compromised.

Acknowledgements. The author appreciates the invaluable contribution of Matthias Ruhl, who organized the computer programs and ran the experiments.

References

[1] Barnsley, M., Hurd, L., *Fractal Image Compression*, AK Peters, Wellesley, 1993.

[2] Barthel, K. U., Schüttemeyer, J., Voyé, T., Noll, P., *A new image coding technique unifying fractal and transform coding*, IEEE Int. Conf. on Image Processing, Texas (1994) 112–116.

[3] Bedford, T., Dekking, F. M., Keane, M. S., *Fractal image coding techniques and contraction operators,* Nieuw Arch. Wisk. (4) 10,3 (1992) 185–218.

[4] Fisher, Y., *Fractal Image Compression — Theory and Application,* Springer-Verlag, New York, 1994.

[5] Gonzales, R. C., Woods, R. E., *Digital Image Processing,* Addison-Wesley, Reading, 1992.

[6] Jacquin, A. E., *Image coding based on a fractal theory of iterated contractive Markov operators, Part II: Construction of fractal codes for digital images*, Technical Report Math. 91389-17, Georgia Institute of Technology, 1989.

[7] Jacquin, A. E., *Image coding based on a fractal theory of iterated contractive image transformations,* IEEE Trans. Image Processing 1 (1992) 18–30.

[8] Kominek, J., *Advances in fractal compression in multimedia applications,* submitted for publication.

[9] Kominek, J., *Codebook reduction in fractal image compression,* Proceedings IS&T/SPIE 1996 Symposium on Electronic Imaging: Science & Technology – Still Image Compression II, Vol. 2669, Jan. 1996.

[10] Monro, D. M., *A hybrid fractal transform,* Proc. ICASSP 5 (1993) 169-172.

[11] Monro, D. M., Dudbridge, F., *Fractal approximation of image blocks,* Proc. ICASSP 3 (1992) 485–488.

[12] Ramamurthi, B., Gersho, A., *Classified vector quantization of images,* IEEE Trans. Commun., COM-34, 1986.

[13] Saupe, D., Hamzaoui, R., *Complexity reduction methods for fractal image compression,* in: I.M.A. Conf. Proc. on *Image Processing; Mathematical Methods and Applications,* Sept. 1994, J. M. Blackledge (ed.), to appear with Oxford University Press, 1995.

[14] Saupe, D., *Fractal image compression via nearest neighbor search,* in: Conf. Proc. NATO ASI *Fractal Image Encoding and Analysis,* Trondheim, July 1995, Y. Fisher (ed.), to appear in Springer-Verlag, New York, 1996.

[15] Signes, J., *Geometrical interpretation of fractal image coding,* NATO ASI Conf. *Fractal Image Encoding and Analysis,* Trondheim, July 1995, to appear in a special issue of Fractals, 1996.