ADAPTIVE PARTITIONINGS FOR FRACTAL IMAGE COMPRESSION

Matthias Ruhl, Hannes Hartenstein, Dietmar Saupe

Universität Freiburg, Institut für Informatik, Am Flughafen 17, 79110 Freiburg, Germany
E-mail: ruhl,hartenst,saupe@informatik.uni-freiburg.de

ABSTRACT

In fractal image compression a partitioning of the image into ranges is required. In our previous work [1] we have proposed to find good partitionings by means of a split-and-merge process guided by evolutionary computing. In this approach ranges are connected sets of small square image blocks. Far better rate-distortion curves can be obtained as compared to traditional quadtree partitionings, however, at the expense of an increase of computing time. In this paper we show how conventional acceleration techniques and a deterministic version of the evolution reduce the time-complexity of the method without degrading the encoding quality. Furthermore, we report on techniques to improve the rate-distortion performance and evaluate the results visually.

1. INTRODUCTION

In fractal image compression the image to be coded is partitioned into blocks called ranges. Each range is approximated by another part of the image called domain. Finding a partitioning that minimizes the approximation error while not exceeding a given bit-rate is a hard problem in fractal image compression. Traditionally, hierarchical schemes like quadtree, rectangular, triangular, and other polygonal partitionings have been used.

Here, we consider adaptive partitionings in which ranges are unions of edge-connected small square image blocks, called atomic blocks (cf. figure 2). This class of partitionings was introduced to fractal coding by [2]. In [1], we have applied techniques motivated by evolutionary programming to find good partitionings of this kind. Significant gains in terms of rate-distortion performance over quadtree based encodings and over the results in [2] were reported.

This paper extends and continues the discussion of [1]. In particular we replace the evolutionary algorithm by a significantly faster deterministic version, as was already proposed in the conclusion of [1]. This leads to a significant speedup. Additionally, we vary the size of the atomic blocks, leading to an improved rate-distortion curve.

Recently published papers that are related to the subject of adaptive partitionings in fractal coding are [3, 4]. In [3] partitionings are Delaunay triangulations which are subjected to an adaptive merging process. In [4] a region-merging approach is also used, but the initial partitioning is a quadtree segmentation. This has some advantages regarding the encoding of the partition information.

2. BASICS OF FRACTAL IMAGE COMPRESSION

This paragraph reviews the standard type of fractal image encoding and introduces some basic notions used in this work. For a range block R we consider a pool of domain blocks twice the linear size. The domain blocks are shrunken by pixel averaging to match the range block size. This pool of codebook blocks is enlarged by including all 8 isometric versions (rotations and flips) of a block. This gives a pool of codebook blocks $D_1, \ldots, D_{N_D}$. For a range R and codebook block D we let

$$(s, o) = \arg \min_{s,o \in \mathbf{R}} \|R - (sD + o\mathbf{1})\|^2$$

where $\mathbf{1}$ is the flat block with intensity 1 at every pixel. The parameters s and o are called scaling factor and offset, respectively. The coefficient s is clamped to $[-s_{max}, s_{max}]$ with $0 < s_{max} < 1$ to ensure convergence in the decoding and then both s and o are uniformly quantized yielding $\overline{s}$ and $\overline{o}$. The collage error for range R and codebook block D is

$$E(D, R) = \|R - (\overline{s}D + \overline{o}\mathbf{1})\|^2.$$

Sorting the codebook blocks D_k with respect to increasing collage error $E(D_k, R)$ yields indices $k_1, \ldots, k_{N_D}$. The fractal code for range R consists of the optimal index k_1 and the corresponding quantized scaling and offset parameters $\overline{s}$ and $\overline{o}$.

3. OUTLINE OF THE PROPOSED ALGORITHM

Our proposed method starts with the partitioning where all ranges are single atomic blocks, e.g., 4×4 pixel blocks. Then, we iteratively merge neighboring ranges to yield a sequence of partitionings with a decreasing number of ranges.

During this process we not only have to maintain a partitioning, but also an associated fractal code. After each merger we derive a new fractal code by a local modification of the previous one. This modification avoids an exhaustive search for an optimal domain for the newly merged range. The information needed for this update is contained in the following datastructure, called *configuration*, which consists of

- a partitioning, i.e., a set of mutually disjoint range blocks which cover the entire image; each range block consists of an edge-connected set of atomic blocks,
- for each range block of the partitioning:
  - a list of d codebook indices $k_1, \ldots, k_d$,
  - the optimal quantized coefficients $\overline{s}, \overline{o}$ corresponding to codebook index k_1.

The algorithm is started with the initial configuration given by

- the uniform partitioning obtained by subdividing the image into atomic blocks,
- for each range (atomic block): optimal codebook indices $k_1, \ldots, k_d$, and coefficients $\overline{s}, \overline{o}$.

After the initialization phase, we begin to merge neighboring range pairs. In order to obtain a matching domain block for the union of two ranges we consider only those domains that are given by the lists of domains inherited from the parent ranges. Of course these domains have to be extended appropriately to match the larger size of the new range. We thus obtain $2d$ codebook blocks from which we keep only the better half yielding a new set of d domain indices for the new range along with the corresponding quantized coefficients $\overline{s}, \overline{o}$.

The strategy to select the two ranges that are merged at each step is a greedy one. We simply take the two ranges whose merger results in the least increase of collage error.

Overall, the process starts with a fractal encoding having a large bit rate and a small collage error. Each new partitioning has one less range. Thus, the bit-rate decreases, while the collage error increases. The process halts, when a given tolerance threshold for the collage error is exceeded, or when the desired bit-rate is achieved.

Besides collage error threshold and final bit rate a parameter of the algorithm is the number d of codebook indices stored for each range.

4. EFFICIENT IMPLEMENTATION

The outlined algorithm can be implemented very efficiently. In fact, it can be sped up by some traditional techniques from fractal coding, yielding a state-of-the-art program that is considerably faster than fractal coders which achieve a similar rate-distortion performance. Still, not all approaches that improve the performance of quadtree based fractal coders work equally well with our algorithm.

In the initialization phase a fractal encoding of the image is sought for which all ranges are atomic blocks. This phase is computationally expensive. Each range block is compared with the same set of domain blocks. Therefore, we can use acceleration techniques that involve heavy preprocessing such as the nearest neighbor search for feature vectors described in, e.g., [5]. The basic idea is to remove the mean and to normalize the variance of ranges and domains. Minimizing the collage error then is equivalent to minimizing Euclidean distance of the resulting feature vectors (and their negative versions). Fast nearest neighbor search techniques based on binary space partitions (kd-trees) are used to accelerate the search and large gains in speed are obtained.

During the merging process we maintain a priority queue (e.g., an heap) containing entries for all neighboring range pairs. The pairs are sorted by the increase in collage error resulting from the merger of the two ranges. Thus, we can easily extract the range pair whose merger gives the least increase in overall collage error. The maintainance of the priority queue is an important factor in the speed of the algorithm. The crucial operation is the update of the heap after merging two ranges. We can avoid recomputing the collage error of the involved range pairs (i.e., the range pairs where one of the ranges is the newly merged range), since the error can only become *larger* due to a merger. We delay recomputing the error until such an outdated pair is extracted from the top of the heap, in which case we compute the correct collage error and reinsert it into the heap. By this we can avoid unnecessary computations while being sure that we always extract the best range pair at each step.

5. EXTENSIONS

Our method can easily be extended to encode color images. We have obtained good results by first transforming the image into YUV-color space and then coding the Y-component as a grey scale image. For the so obtained partitioning we additionally encode the mean of the U- and V-components for every range.

For quadtree based fractal coders a significant improvement of the image quality can usually be achieved by postprocessing the image to reduce blocking artifacts at the borders of the ranges (cf. figure 3). For our approach things turn out not to be that simple. The irregularity of the ranges leads to a less structured distribution of the error. Thus, averaging across range borders tends to deteriorate rather than improve image quality in our setting.

Furthermore, entropy coding of the offset parameters is reported to ameliorate the rate-distortion performance for fractal coders. While this generally only gives a marginal gain, for our scheme the improvements were negligible.

Figure 1: 512x512 Lenna compressed using our method, compression ratio = 1:69.5, PSNR 28.3 db

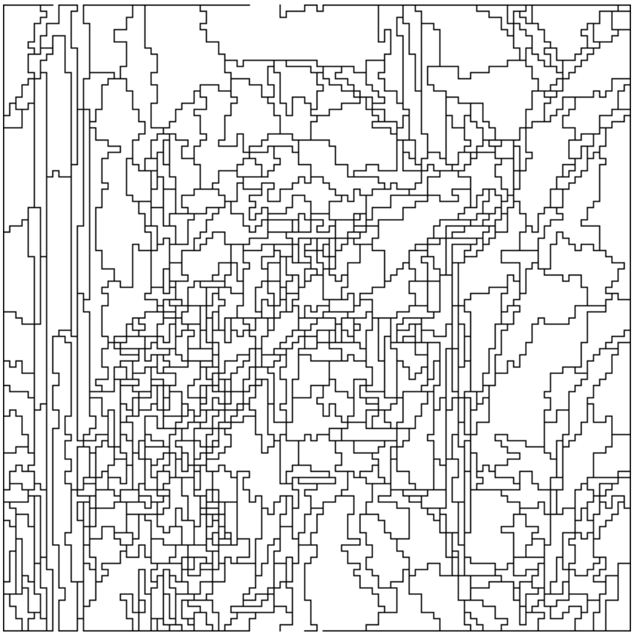

Figure 2: Partitioning pertaining to figure 1, 600 ranges

Figure 3: 512x512 Lenna compressed using Fisher's quadtree coder, compression ratio = 1:28.1, PSNR 28.8 db

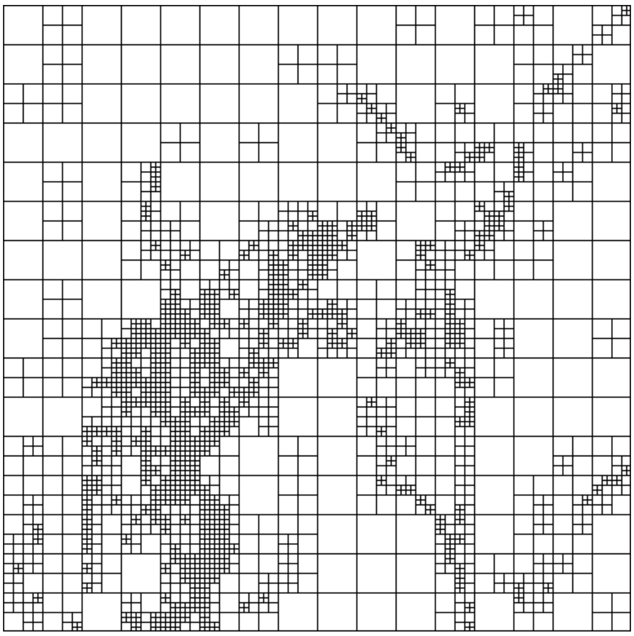

Figure 4: Quadtree partitioning for figure 3, 2812 ranges

6. RESULTS

In our experiments, the number d of codebook indices associated with a range was set to 10. The partitioning is stored as described in [1].

Figure 5 shows the rate-distortion curves for our method using atomic block sizes 3×3, 4×4, 5×5 and 8×8. Varying the atomic block sizes depending on desired compression ratio improves the performance of our method.

In figure 6 we give a comparison between our method, Fisher's HV coder [6] and quadtree coder [7]. In this plot, we have adapted the atomic block size for our method de-

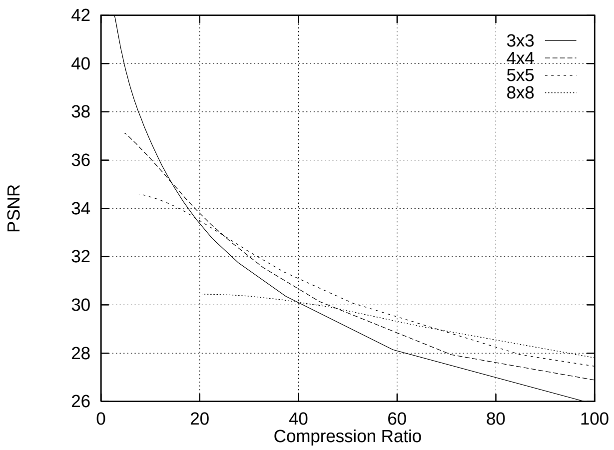

Figure 5: Results for encoding 512×512 Lenna using our scheme and various atomic block sizes

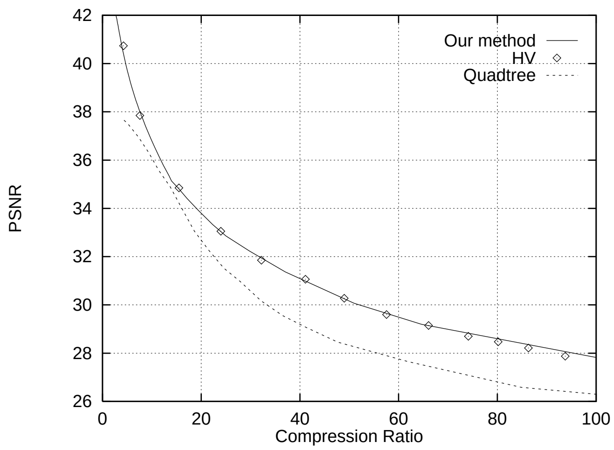

Figure 6: Comparison between our method and Fisher's HV and Quadtree-based schemes.

pending on the compression rate. The HV-data is taken from [6], the quadtree results were obtained by running Fisher's program with parameters *-f -D 2 -d 4*. For compression ratios larger than 1:30 we obtain a 2 dB improvement over the quadtree coder. The results in terms of PSNR are comparable to Fisher's HV coder.

Figure 1 shows a decompressed image of Lenna with corresponding partition in figure 2. Note that we allow that the ranges can wrap around image borders. In figures 3 and 4 an encoding of Lenna using Fisher's quadtree scheme at a slightly *better* PSNR is shown. Our method uses less than half the bit-rate of Fisher's code and also is visually more pleasing. This is mainly due to the less regular blocking structure.

Table 1 compares the computing times with the evolutionary scheme proposed in [1], showing a speedup factor of over 100. The times were measured on an SGI O2 with a R10000 processor running at 150 MHz. In each case, 512×512 Lenna was compressed to 1000 ranges using an atomic block size of 4×4. The evolutionary method gives an insignificantly better PSNR ($<$ 0.3 dB).

These experiments show that our algorithm belongs to the best fractal coders regarding image fidelity, but cuts down the encoding time to a fraction of the running time of fractal coders that achieve comparable image quality.

A program implementing this algorithm is available at http://www.informatik.uni-freiburg.de/frap.

	Init	Merge	Total
New	0:02:00	0:01:23	0:03:23
Evo	5:09:27	1:25:17	6:34:44

Table 1: Computing times (hrs:min:sec) for the proposed (New) and the evolutionary (Evo) [1] methods. Given are the times for the initialization phase, the merging phase, and the total running times.

7. REFERENCES

[1] Saupe, D., Ruhl, M., *Evolutionary fractal image compression,* Proceedings IEEE ICIP, Lausanne, 1996, pp. 129–132.

[2] Thomas, L., Deravi, F., *Region-based fractal image compression using heuristic search,* IEEE Transactions on Image Processing 4,6, 1995, pp. 832–838.

[3] Davoine, F., Robert, G., J.-M. Chassery, *How to improve pixel-based fractal image coding with adaptive partitions,* Proceedings Fractals in Engineering, Véhel, J.-L., Lutton, E., Tricot, C. (eds.), Springer Verlag, London, 1997, pp. 292–306.

[4] Chang, Y.-C., Shyu, B.-K., Wang, J.-S., *Region-based fractal image compression with quadtree segmentation,* Proceedings ICASSP, Munich, 1997, pp. 3425–3428.

[5] Saupe, D., *Accelerating fractal image compression by multi-dimensional nearest neighbor search,* Proceedings IEEE Data Compression Conference, Storer, J., Cohn, M. (eds.), Snowbird, Utah, 1995, pp. 222–231.

[6] Fisher, Y., Menlove, S., *Fractal encoding with HV partitions,* in [7], pp. 119–136.

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