EVOLUTIONARY FRACTAL IMAGE COMPRESSION

Dietmar Saupe, Matthias Ruhl

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

ABSTRACT

This paper introduces evolutionary computing to fractal image compression. In fractal image compression [1] a partitioning of the image into ranges is required. We propose to use evolutionary computing to find good partitionings. Here ranges are connected sets of small square image blocks. Populations consist of N_p configurations, each of which is a partitioning with a fractal code. In the evolution each configuration produces σ children who inherit their parent partitionings except for two random neighboring ranges which are merged. From the offspring the best ones are selected for the next generation population based on a fitness criterion (collage error). We show that a far better rate-distortion curve can be obtained with this approach as compared to traditional quad-tree partitionings.

1. EVOLUTION OF PARTITIONINGS

Finding the optimal partitioning at a given bit-rate is an unsolved problem in fractal image compression. The space of all partitionings with a given number of ranges is simply too huge. Traditionally, deterministically derived quad-tree [2, 3], rectangular [4], triangular [5, 6], and other polygonal [7] partitionings are used. We follow [8] and define ranges as unions of edge-connected small square image blocks. The type of fractal image encoding chosen is the standard one: 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 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 maximal intensity at every pixel. The coefficient s is clamped to $[-1, 1]$ 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 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 $\overline{o}$.

Initially we subdivide the image to be encoded into *atomic blocks* of the same size, e.g., of size 4 by 4 pixels. With these notations we can now define configurations. A *configuration* 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.

A *population* is a set of N_p configurations which have the same number of ranges in their partitionings. The evolution is started with an initial population of N_p identical configurations 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 $\overline{s}, \overline{o}$.

In each cycle of the evolution an offspring generation of new configurations is produced as follows: For each configuration in the parent population we maintain a list of all neighboring range pairs (ranges are considered neighbors when they share an edge of an atomic block). From this list one range pair is chosen at random. These two ranges are united yielding a new partition with the two old ranges removed and with their union as a new range.

In order to obtain a matching domain block for the new enlarged range we do not afford a search through the full respective domain pool but rather consider only those domains that are given by the lists of the 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 $\overline{s}, \overline{o}$.

This describes how a parent configuration leads to a child configuration with one less range. Repeating this process σ times for each configuration of a population produces a total of σN_p child configurations. Only the best N_p of these are kept and make up the next generation population. The fitness criterion is in accordance with fractal image compression: the N_p surviving configurations are those with the least collage errors $E(D, R)$, summed up over all ranges of the corresponding partition.

From each population the best fractal encoding can be easily extracted for flow control. Overall, the evolutionary process starts with a fractal encoding having a large bit-rate and a small collage error. Each generation has one less range. Thus, the bit-rate decreases, while the collage error increases. The evolution is halted, 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 the parameters in the evolutionary process are the *population size* N_p, the *number* d *of codebook indices* stored for each range, and the *branching factor* σ.

2. CODING OF THE PARTITION

We now describe how the partitions, arising in the evolutionary coding process, are stored efficiently. We have implemented and compared the following four algorithms (see Figure 1).

Method 1. For each atomic block two bits are stored indicating whether it is connected to its right and lower neighbors. The resulting bit-stream is grouped into bytes and compressed using an LZW-algorithm. Despite its simplicity this method compares well with the other algorithms (see Figure 1).

The following algorithms use a *derivative chain code (DCC)* to store the partitions. They proceed by tracking the range boundaries, i.e., the black lines in Figure 3, and storing the necessary movements.

Method 2. The following procedure is iterated until the range contours are completely described. First, an unvisited point on the range boundary is selected, its position is output and pushed a on stack. As long as the stack is not empty, the following steps are repeated. The stack is popped, and starting at the given position, the boundary is tracked until an already visited position is reached. At every step we output a symbol,

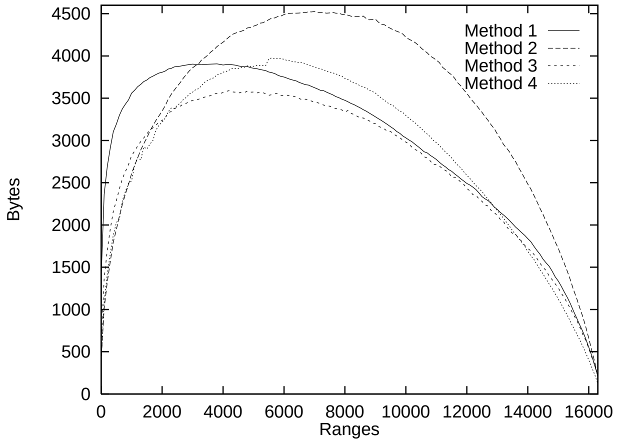

Figure 1: Comparison of partition compression schemes, explained in the text. The underlying partitions stem from 512×512 Lenna.

showing which of the three directions (turn left, continue straight ahead, turn right) we can take next. At contour branching points we take the first branch (in the aforementioned order) and push the endpoints of the other available line segments on the stack for later perusal. Since at every step the contour must continue in at least one of the three directions, $2^3 - 1 = 7$ symbols suffice for the encoding of a step. Finally, the resulting symbol string is arithmetic entropy coded. Note that this method allows for a shorter code than Method 1 if the ranges are large (see Figure 1). This is to be expected since there are no bits used for the interior of these ranges.

A refinement of this algorithm leads to the next two methods.

Method 3. The main deficiency of Method 2 is that some line segments are stored twice (see Figure 2). By outputting at every step the possible movements (one bit each), but omitting the redundant information, we can reduce the code size even further. The bit-stream is grouped into bytes and LZW-compressed.

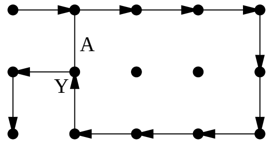

Figure 2: If the range contour is encoded by Method 2, every time the trace closes in on itself, a line segment is stored twice. In the figure segment A is stored twice: first, when position X is traversed and a second time when Y is visited.

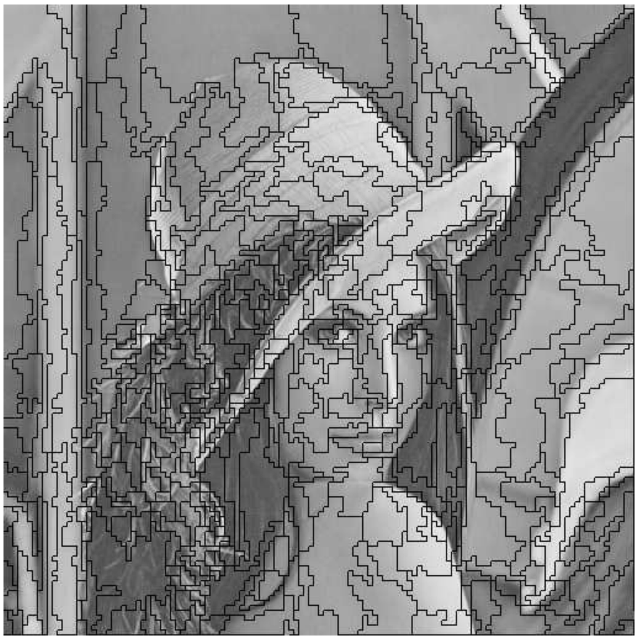

Figure 3: Partitioning of 512 × 512 image by evolution with 500 ranges (compression ratio 70.5). Shape information = 2029 bytes, transformation information = 1691 bytes.

Method 4. Alternatively, instead of using a bit-stream as in Method 3, we can output a symbol at every step of the tracing process. One could use a different symbol set depending on whether there are 1, 2 or 3 unknown directions at a given step. But since the decoder knows at each step which symbol group (1,2,3 bits) to expect, seven symbols suffice as in Method 2. Arithmetic entropy coding the resulting string again yields our final output.

From Figure 1 it becomes clear that for all ranges either Method 3 or Method 4 yields the shortest code. In our implementation we therefore compute both codes and store the shorter one, spending one additional bit to tell the decoder about our choice.

3. RESULTS

In a first test we apply the evolutionary algorithm and the quad-tree method [9] to the 512 × 512 test image Lenna (Figures 3 and 4). The procedure is initialized with a fractal code using atomic blocks of size 4 × 4 pixels obtained by full search of a codebook of size 64 × 64 × 8. With the quad-tree method we use codebooks of the same size at each level of the quad-tree and full search (no classification). The performances of both methods are comparable at low compression ratios as expected. However, at high compression ratios we observe large gains with our method up to several dB of PSNR.

Parameters			Times in secs	
N_p	σ	d	Initialization	Evolution
5	5	5	1305	129
10	10	10	1305	458
20	20	20	1305	2473

Table 1: Computation times of evolution to 500 ranges.

PSNR Results for 256 × 256 Lenna (dB)									
	$N_p = 5$			$N_p = 10$			$N_p = 20$		
σ	d=5	10	20	d=5	10	20	d=5	10	20
5	27.6	27.7	27.7	27.7	27.8	27.8	27.7	27.8	27.8
10	27.9	28.0	27.9	27.9	27.9	28.1	28.0	28.0	28.0
20	28.1	28.0	28.1	28.0	28.1	28.1	28.0	28.1	28.2

Table 2: Performance of the evolutionary method with different parameter settings for the population size N_p, the branching factor σ and the adaptive domain pool size d. For all 27 runs the bit-rates of the fractal codes are the same, based on partitions with 500 ranges.

In a second experiment we test the performance of the evolutionary algorithm as a function of its parameters N_p, σ, and d. Tables 1 and 2 list the results for the 256 × 256 image Lenna. The most critical parameter is the branching factor σ. With the large value of $\sigma = 20$ we get close to the best results even when the other two parameters are small. Table 1 shows some of the corresponding computation times of our experimental non-optimized compression program (running on an R4600SC 133 MHz processor of SGI). A large part of the time is spent finding the initial fractal code by full search.

Finally, the image quality may be improved by more searching in the final stage. Based on the partitions generated by the evolutionary encoder, we computed the optimum range-domain pairings. Interestingly, the improvement over the original code is rather small (see Figure 5).

4. CONCLUSION

In this paper we have demonstrated the potential of evolutionary computing to solve one of the most important problems in fractal image compression: optimizing the range partitioning. This work motivates many more investigations in this direction of research:

- The initialization takes up a large fraction of the total encoding time (see Table 1). The effect of faster (suboptimal) initialization schemes should therefore be examined.
- The evolution can be accelerated by replacing error based decisions by something simpler, e.g., by using

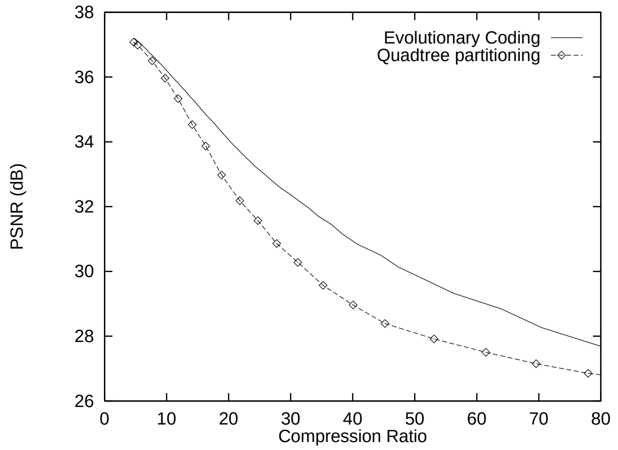

Figure 4: Rate distortion curves for the evolutionary method versus the quad-tree method (512×512 Lenna). Parameters of our method are $N_p = d = 10$ and $\sigma = 20$.

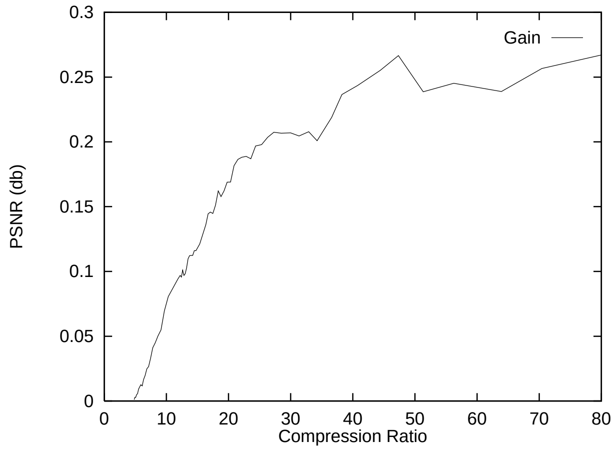

Figure 5: PSNR gain if full search fractal encoding is applied to the partitions generated by the evolutionary process (parameters as in Figure 4).

range block variances. That is, we seek to minimize $\sum_i \sum_{j=1}^{size(i)} (x_{ij} - (\frac{1}{size(i)} \cdot \sum_{k=1}^{size(i)} x_{ik}))^2$, where the x_{ij} are the pixels belonging to range i. Since here it is not necessary to do the time-consuming range-domain comparisons, this scheme is much faster than the error based method. The range-domain pairing needs only be computed at the end of the evolution. Sophisticated algorithms exist for this purpose [10].

- The error-based fitness criterion can be replaced by a fitness criterion taking into account the size of the partition code. In this way we introduce *entropy-constrained partitions.*
- One may even consider a set of *competitive strategies* to select the range pair to be joined in an offspring configuration. Different strategies may prove to be appropriate at different stages of the evolution.
- The size of the atomic blocks can be set to other values than 4×4 pixels.
- Entropy coding of the parameters for the transformations of the fractal encoding may further improve the bit-rates.
- Finally, instead of *probabilistically* selecting at each step of the encoding process a range pair to merge, one can *deterministically* compute the optimal range pair, i.e., the range pair whose merger yields the lowest increase of the overall error. In fact, a first implementation indicates, that this algorithm may outperform the nondeterministic one in terms of speed without decreasing the resulting image quality.

Several of these extensions will be investigated in a forthcoming paper.

5. REFERENCES

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

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

[3] Jacobs, E. W., Fisher, Y., Boss, R. D., *Image compression: A study of the iterated transform method,* Signal Processing 29 (1992) 251–263.

[4] Fisher, Y., Menlove, S., *Fractal encoding with HV partitions,* in [9].

[5] Davoine, F., Antonini, M., Chassery, J.-M., Barlaud, M., *Fractal image compression based on Delaunay triangulation and vector quantization,* IEEE Trans. Image Proc. 5,2 (1996) 338–346.

[6] Novak, M., *Attractor coding of images,* Licentiate Dissertation, Dept. of Electrical Engineering, Linköping University, May 1993.

[7] Reusens, E., *Partitioning complexity issue for iterated function systems based image coding,* in: *Proceedings of the VIIth European Signal Processing Conference EUSIPCO'94,* Edinburgh, Sept. 1994.

[8] Thomas, L., Deravi, F., *Region-based fractal image Compression using heuristic search,* IEEE Trans. Image Proc. 4,6 (1995) 832–838.

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

[10] Saupe, D., Hartenstein, H., *Lossless acceleration of fractal image compression by fast convolution,* Proc. 1996 Intern. Conf. Image Proc. (ICIP), Lausanne, Sept. 1996.