Zuerst ersch. in:
Arabian Journal for Science and Engineering / C ; 29 (2004), 2C. - S. 63-83

EDGE-BASED PARTITION CODING FOR FRACTAL IMAGE COMPRESSION

Tilo Ochotta* and Dietmar Saupe†

Department of Computer and Information Science
University of Konstanz, Germany

الخلاصــة

يُقدم هذا البحث طريقة لضغط الصُور باستعمال تـقـنية صورة النمطي الهندسي المتكرّر التي تضمن أحسن أداء مقارنة بطرق صورة النمطي الهندسي المتكرّر التي لا تعتمد على تـشفير التحويلة المُهجّنة. وقد تمّ تحصيل هذا الإنجاز باستعمال نظام العدّ العياري، حيث يتم تـقسيم الصورة إلى أجزاء غير متراكبة المدى التي تـمّ تـقريـبها بوساطة تطبيق أجزاء ذات مجال أوسع حيث تكون شدّات الصورة مماثلة اتّصاليّـاً بشدات مجالات تلك الأجزاء. وتكمُن خصوصية طريقتـنا في التجزئة الحيّـزيّـة الانضباطيّـة التي تعمل بنمط الفصل والدمج ، مع العلم أنه في التجزئات المبنية على نمط الفصل والدمج المستعملة سابقاً في تـشفير الصوّر بتـقـنية صورة النمطي الهندسي المتكرّر، تـشرع عملية الدمج بتجزئة منتظمة متكوّنة من أجزاء مربعة ومتساوية. أما في هذا البحث فسنبني طريقة دمج الأجراء على تجزئة انضباطية رُباعية الفروع. وسيتم أيضاً اقتراح طريقة فعّالة لتـشفير تجزئات الفصل والدمج الرباعيّـة الفروع التي تُعمّم تـشفير المناطق الحافيّـة للخرائط والتي تـشمل التـشكيل ضمن سياق خاص. ويمكن استعمال طريقة التـشفير المقترحة على تطبيقات أخرى تعتمد على تجزئات الصورة.

*Address for correspondence:
University of Konstanz
Computer and Information Science
Box M697
78457 Konstanz, Germany
e-mail: tilo.ochotta@uni-konstanz.de
www: http://www.inf. uni-konstanz.de/~ochotta

†e-mail: saupe@inf.uni-konstanz.de

ABSTRACT

This paper presents an approach for fractal image compression that yields the best performance compared to fractal methods that do not rely on hybrid transform coding. The achievement is obtained using the standard algorithm in which the image is partitioned into non-overlapping range blocks which are approximated by corresponding larger domain blocks with image intensities that are affinely similar to those of the range blocks. The particular feature of our approach is the adaptive spatial partition that proceeds in a split/merge fashion. In previous split/merge partitions for fractal image coding, the merging started from a uniform partition consisting of square blocks of equal size. In this paper, we base the block merging on an adaptive quadtree partitioning. An efficient encoding method for the quadtree-based split/merge partitions is proposed which generalizes the coding of region edge maps and which includes special context modeling. The encoding method may be used for other applications that rely on image partitions.

Key words: Fractal Image Compression, Partition Coding, Region Edge Maps, Image Segmentation.

EDGE-BASED PARTITION CODING FOR FRACTAL IMAGE COMPRESSION

1. INTRODUCTION

Fractal image compression was conceived by Barnsley and Sloan [1, 2], and the first fully automated fractal image compression algorithm was published by Jacquin in 1989 [3, 4]. Given an image, the encoder finds a contractive affine image transformation (fractal transform) T such that the fixed point of T is close to the given image in a suitable metric. The decoding is by iteration of the fractal transform starting from an arbitrary image. Due to the contraction mapping principle, the sequence of iterations converges to the fixed point of T. Jacquin's original scheme showed promising results. Since then, many researchers have improved the original algorithm, outperforming standard JPEG image compression. However, the rate–distortion performance of current state-of-the-art codecs based on wavelet transform coding as, *e.g.*, in JPEG2000 cannot yet be reached, even by the best fractal coders that work in the spatial domain. For surveys, detailed discussions, computer code, and bibliography see [5–7].

The fractal transform in its common form has two essential components, namely first, an image partition and second, for each block of the partition (called range block) an address for a similar image block, called a domain, and a small set of real quantized transform parameters. These parameters determine an affine mapping of the intensity values of the domain block that is shaped like the range block. These mappings are chosen such that the intensities in each range block are approximated by the transformed intensities of the corresponding domain block. The fractal transform T operates on an arbitrary image by mapping domains to ranges as specified in the fractal code. Contractivity of the transform may be provided by a simple condition on one of the parameters of the affine mappings.

Uniform image partitions do not yield satisfying rate–distortion performance for fractal coding. Thus, already the first papers on fractal compression suggested adaptive image partitions. Jacquin used image squares that were subdivided into smaller regions consisting of one to three subsquares of half the size [4]. In [5] adaptive quadtrees were used. These partitions are built with square blocks which are recursively subdivided into four subblocks of equal size until some approximation quality is reached or the block size falls below a given threshold. Other hierarchical image partitions can be based on rectangles [8, 9] and polygons [10]. Delaunay triangulations have also been used [11, 12].

Highly adaptive partitions can be generated by a split/merge scheme [13] in two phases. First, the image is partitioned into successively smaller blocks until a termination criterion is fulfilled. Then neighboring regions may be merged again as long as some coherence criterion is satisfied. In the methods in [14–16], the splitting is simple and results in uniform square blocks. These small squares are merged again to form irregularly shaped range blocks that adapt well to the structure of the underlying image and provide some of the best fractal coders [16]. Uniform and quadtree partitions are easy to encode, comprising just a small fraction of the total rate in a fractal image code. However, more adaptive partitions such as those based on a split/merge process are harder to encode efficiently. There are two main approaches. One may either encode the boundaries of the image regions, or one can encode connectivity of the basic blocks that make up the image regions. Chain coding [17] belongs to the first category, while region edge maps [18] are an example for the second kind.

Our work extends and improves the methods in [16] in two ways.

(1) In 1976 Horowitz and Pavlidis [19] suggested a split/merge scheme for efficient image representation in which the splitting provided a quadtree partition whose blocks are then merged to form adaptive image regions. Such image partitions can also be generated when splitting all the way down to equal sized squares. However, due to the occurrence of larger coherent blocks in the underlying quadtree partition

a more efficient code of the partition may be expected, without significantly reducing the adaptivity of the final partition. Thus, we propose to use this approach in the context of fractal coding. We note that this has already been tried in [20, 21].

(2) Our main contribution is an efficient method to encode the corresponding split/merge partition, *i.e.*, an image partition in which each block is the union of some neighboring square blocks from a given quadtree partition, which must also be encoded efficiently. For this purpose, we generalize the region edge maps proposed in [18], and provide comprehensive context modelling for efficient arithmetic encoding.

The rest of the paper is organized as follows. In Section 2 we briefly review the basics about fractal image coding and give a short survey of previous work in the field of lossless partition coding that is relevant with respect to our approach. In particular, we revisit region edge maps which provide the basis of our new method for quadtree-based region edge maps. In Section 3 we define a compression scheme for quadtree codes and introduce quadtree-based region edge maps and their efficient encoding with context modelling. In Section 4 we apply the proposed scheme to region-based fractal image compression providing efficient partition coding. Experimental results are presented in Section 5.

2. TERMINOLOGY AND RELATED WORK

2.1. Fractal Image Compression

We consider an image F of size $N \times N$ as a function that maps a set of N^2 pixel coordinates (x, y), $x, y = 0, \ldots, N-1$, to the set of grey values $v \in \{0, \ldots, 255\}$. The function F may be represented as a corresponding set of 3-tuples (x, y, v). A *base block* b is an $n \times n$ subset of F, $b = \{(x, y, v) \in F \mid x = x_{min}, \ldots, x_{max}, y = y_{min}, \ldots, y_{max}\}$ where $x_{max} - x_{min} = y_{max} - y_{min} = n - 1$. The base block size n is a fixed parameter of our method and may be chosen as small as $n = 1$ amounting to base blocks being single pixels. For our image partition we need to consider groups of connected base blocks as range blocks for the fractal coder. We call two base blocks b_1, b_2, $b_1 \cap b_2 = \emptyset$ *adjacent*, if and only if there are two pixels $p_1 = (x_1, y_1, v_1) \in b_1$, $p_2 = (x_2, y_2, v_2) \in b_2$ such that $|x_2 - x_1| + |y_2 - y_1| = 1$. We define a *region* as a connected set of base blocks. A set of regions $P = \{r_i \mid i = 0, \ldots, n_R - 1\}$ with $r_i \neq r_j$ and $r_i \cap r_j = \emptyset$ for $i \neq j$ which covers the image F, *i.e.*, $\cup_{r \in P}\, r = F$, is called a *partition* of F.

The image F in a range block is approximated by an intensity transformed copy of a larger block, called a domain block. Let us denote by D_F the domain image of size $N/2 \times N/2$ obtained by downsampling the original $N \times N$ image F. Usually, downsampling proceeds by averaging intensity values of pixels in 2×2 blocks. For a given range block r parameters $x_d, y_d \in \{0, \ldots, N/2 - 1\}$ and $s_r, o_r \in \mathbb{R}$ are sought such that

$$v \approx \hat{v} = s_r \cdot D_F((x + x_d) \bmod N/2, (y + y_d) \bmod N/2) + o_r, \quad (x, y, v) \in r. \tag{2.1}$$

The region that is addressed with (x_d, y_d) in the domain image D and the same shape as the range block is the domain block for the range r. The parameters s_r and o_r allow scaling and adjustment of brightness of the grey values of the domain. Scaling factors are chosen with the constraint $|s_r| < 1$. Moreover, scaling factors s_r and offsets o_r are quantized for the purpose of encoding.

We also consider an approximation error for mapping (2.1):

$$E(r) = \sum_{(x,y,v) \in r} (v - \hat{v})^2. \tag{2.2}$$

The approximation error $E(r)$ should be as small as possible. Typically, least squares optimization is used to determine the best scaling factor and offset for a given domain address (x_d, y_d), and a search through a

pool of domains determines the domain which yields the least approximation error. We also consider a global approximation error for the image F which is called *collage error*:

$$E = E(F, P) = \sum_{r \in P} E(r). \tag{2.3}$$

The fractal code must uniquely describe the image partition and for each range block in the partition the domain block used together with the scaling factor and the offset. This code essentially amounts to the definition of an image operator T. Given an image I_k, the transformed image $I_{k+1} = T(I_k)$ is obtained by applying all range block transformations in (2.1), where the domain blocks are taken from the image I_k in place of F. This image operator T is a contraction and, hence, iterating T starting with an arbitrary initial image I_0 converges to a unique image $I^\star$, the fixed point of the fractal transform T. The fixed point $I^\star$ can be constructed by the decoder and serves as the image reconstruction. The reconstruction quality depends on the collage error 2.3, which is due to the so-called collage theorem. For further details, background material, and proofs we refer to the large body of literature, see, *e.g.*, [5, 22, 23].

2.2. **Lossless Partition Coding**

There are two classes for lossless partition coding. The first one is chain coding in which sequences of symbols for directions represent region boundaries [17]. A naive chain coding for image partitions encodes the closed boundary of each region as a starting position and a sequence of steps in the directions North, East, South, and West. This approach is not satisfying. The reason is that boundary segments are encoded twice because an edge is shared by two neighboring regions except at the image boundary. Furthermore, the encoding of the start positions can become expensive in bits even when the absolute address for a region is encoded relative to another one. Several optimizations have been introduced, *e.g.*, in [24], where three symbols are used to represent a chain (go straight, left, or right). In [25] the partition is not coded as a set of closed region boundaries but as set of line sequences. Each sequence describes a number of connected lines for which a relative starting point, the length of each line, and the relative directions of the line segments are encoded. Each region edge is visited only once and long straight lines can be encoded by a small number of symbols. Other, more sophisticated, methods for chain coding (in the context of fractal image coding) are in our previous paper [26].

The second class of methods to represent region shapes is related to encoding binary images. A fundamental method for compressing black-and-white images has been proposed in [27]. The binary intensities are encoded individually in line-by-line order using an arithmetic coder [28] with context modelling. In [18], this approach was extended to encode grid-based partitions in which the connectivity of a base block is encoded with one out of four symbols each. For a given base block it suffices to indicate whether its Northern and Western borders are part of a region boundary, which requires two bits per base block (see Figure 1(a)). The resulting 4-nary image is called a region edge map.

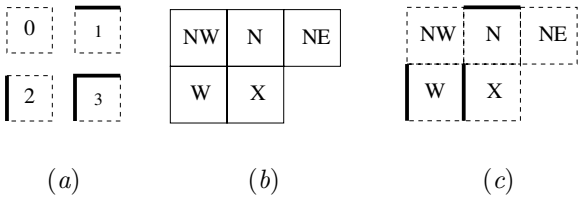


Figure 1. *Region edge maps with context modeling:* (a) *a block is represented by a symbol from* $\{0, 1, 2, 3\}$ *which encodes the presence resp. absence of region edges at the Northern and Western boundary of the block;* (b) *the symbols of four preceding blocks (West, North West, North, and North East) are used to define a context;* (c) *the symbol combination* $N = 1$, $W = 2$, $X = 2$ *cannot occur in a partition.*

Table 1. Impossible Symbol Combinations.

W	NE	N	NW	X	W	NE	N	NW	X
0	*	0	*	1	2	*	0	*	1
0	*	0	*	2	2	*	0	*	2
0	*	1	*	1	2	*	1	*	1
0	*	1	*	2	2	*	1	*	2
0	*	2	*	0	2	*	2	*	0
0	*	3	*	0	2	*	3	*	0
1	*	0	*	0	3	*	0	*	0
1	*	1	*	0	3	*	1	*	0

The asterisks correspond to any symbol $\{0, \ldots, 3\}$.
All in all 768 out of $4^5 = 1024$ combinations may occur.

In [18] the symbols of a region edge map are encoded line-by-line with an adaptive arithmetic coder and context modelling similar to [27]. The context modelling (Figure 1(b)) uses four previously encoded/decoded symbols (symbols for blocks in West, North West, North and North East direction). Due to elementary geometric facts some symbol combinations cannot occur (Figure 1(c), Table 1). Taking this into account, the following formula is proposed to update the probability p for a symbol z, that may appear in context c:

$$p(z|c) = \frac{\lambda \cdot count(z|c) + 1}{\lambda \cdot count(c) + |A_c|} \,, \tag{2.4}$$

where $count(z|c)$ is the number of symbols z encoded in context c, $count(c)$ is the total number of occurrences of context c, and $|A_c| \leq 4$ is the number of symbols that are possible in context c. Probabilities $p(z|c)$ for symbols z that are not possible in context c as indicated in Table 1 are set to zero. Following the experimental results in [18], the weight factor λ should be set to 4 or 5 for best compression results. In our empirical studies we used $\lambda = 4$.

3. PARTITION CODING METHOD

3.1. Quadtree Decomposition

Our partition method is based on a split/merge approach as introduced by Horowitz and Pavlidis in [19]. In our case, the splitting phase consists of a quadtree segmentation. Tate's edge-based coding scheme [18] uses a uniform grid of equally sized square base blocks. In this section, we extend the coding to partitions in which regions are unions of square base blocks from a quadtree segmentation.

A quadtree is a hierarchical data structure [29] amounting to an ordered tree with inner nodes of degree four. In image processing applications a node corresponds to a rectangular (or square) image area, which is recursively split into four equal size subrectangles (or subsquares) corresponding to the four children of the node. A split rule evaluated for a given node, respectively the corresponding rectangle (square), determines termination of the recursion, in which case the node ends up as a leaf node of the quadtree. The blocks belonging to the leaf nodes of the tree comprise the quadtree partition. By means of the split criterion, the quadtree may adapt to

image content. For example, large blocks in low frequency areas and small blocks in high frequency areas can be generated, respectively. Pseudo code for the procedure is given in Algorithm 1.

After quadtree segmentation, an iterative merging phase follows where in each iteration two adjacent blocks are merged, which may be further merged with other blocks in a succeeding iteration. An application specific merge criterion, evaluated for each pair of adjacent blocks to be merged, determines two blocks that are merged in an iteration step. Pseudo code for the procedure is given in Algorithm 2.

During the merging process it is possible that a group of four blocks belonging to four child nodes of the same parent node in the quadtree are merged into one region. In such a case it is possible to simplify the quadtree by pruning it at the parent node. We call the resulting quadtree a *fitted quadtree* and expect to be able to encode the fitted quadtree as well as the quadtree-based regions of the final image partition with a smaller rate.

Algorithm 1 split(n)

Input: quadtree leaf node n with corresponding image block $r(n)$.
Output: quadtree root node n with corresponding quadtree partition of $r(n)$.
- split criterion is given by boolean function $SC(\cdot)$.

if $SC(r(n))$ **then**
  expand the node n using four child nodes $n_i, i = 0, ..., 3$
  for i=0,...,3 **do**
    - n_i = split(n_i)

  end for
end if
return node n.

Algorithm 2 merge(P,n_R)

Input: quadtree partition P, target region number $n_R < |P|$.
Output: partition P' with $|P'| = n_R$ regions.
- Set $P' = \emptyset$.

for all $q \in P$ **do**
  - initialize q as region r
  - insert r into P'

end for
while $|P'| > n_R$ **do**
  - find the region pair (r_i, r_j), r_i, r_j neighbors in P' minimizing an error criterion
  - insert $r_i \cup r_j$ into P'
  - remove r_i and r_j from P'

end while
return P'.

3.2. **Quadtree Coding**

In this subsection we describe the encoding of the quadtree partition. The following subsection covers the encoding of the image partition derived from merging quadtree blocks. The (ordered) quadtree (Figure 2(a),(b)) is traversed in depth-first order processing each node by outputting a binary symbol 1 for an inner node and 0 for a leaf node. Since we also encode the smallest and largest depth of a leaf node as side information, we may delete the symbols 1 for inner nodes above the minimal leaf depth and the symbols 0 for leaf nodes at the maximal depth.

Context-based arithmetic coding is crucial for obtaining compact codes for quadtrees. For example, context modelling can exploit the fact that situations given by one large quadtree block adjacent to many small quadtree

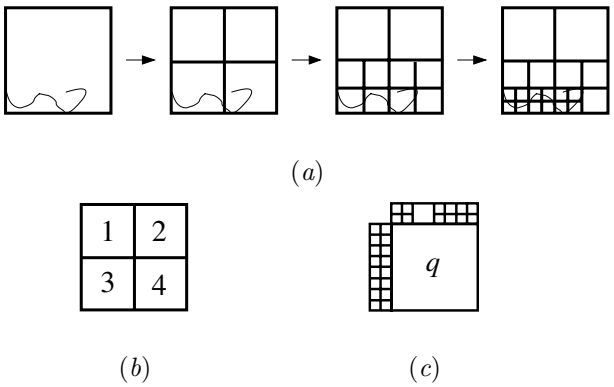


Figure 2. *Quadtree decomposition after the top down scheme:* (*a*) *stepwise split up of the blocks;* (*b*) *encoding order;* (*c*) *situation with low probability of occurrence: one block q has many neighbor blocks.*

blocks (Figure 2(*c*)) seldomly occur. For a given block q we consider the number of neighboring leaves in North and West (z_N and z_W respectively). These numbers are known also at the decoder due to the depth-first traversal of the quadtree. These counts are bounded, $0 \leq z_N,\ z_W \leq z_{max}$, where z_{max} is determined by the side information (minimal and maximal depth of leaf nodes). For example, $z_{max} = 8$ for a 4–32-quadtree, *i.e.*, a quadtree segmentation in which the minimal and maximal block sizes are 4×4 and 32×32, respectively.

We propose the following three heuristic schemes to build up a context for the given block q. In the last subsection we empirically compare the performance of these three context models for quadtree partitions of varying complexity for test images.

Full Context: Two symbols are defined to be in the same context class if and only if the corresponding tuples (z_N, z_W) of neighbor leaf numbers are equal. If we number the context classes by $c = 0, ..., c_{max}-1$, we may write $c = (z_{max}+1)z_N + z_W$ and $c_{max} = (z_{max}+1)^2$.

Sum Context: Two symbols are in the same context class if and only if the corresponding combined neighbor leaf numbers, *i.e.*, $z_N + z_W$, are equal. Thus, $c = z_N + z_W$ and $c_{max} = 2 \cdot (z_{max}+1)$.

Reduced Context: Four context classes are defined by checking whether there are several neighboring leaf nodes in the North and in the West. More precisely,

$$c = \begin{cases} 0 & \text{for} \quad z_N \leq 1 \quad \text{and} \quad z_W \leq 1 \\ 1 & \text{for} \quad z_N \leq 1 \quad \text{and} \quad z_W \geq 2 \\ 2 & \text{for} \quad z_N \geq 2 \quad \text{and} \quad z_W \leq 1 \\ 3 & \text{for} \quad z_N \geq 2 \quad \text{and} \quad z_W \geq 2 \end{cases}$$

and $c_{max} = 4$.

3.3. **Quadtree-based Region Edge Maps**

In this subsection we generalize the region edge map (REM) encoding scheme described in Subsection 2.2 to the case where the blocks used in the merging phase are from a quadtree image partition.

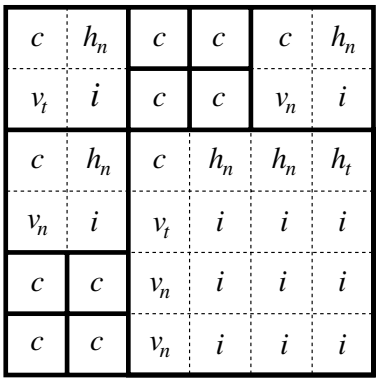


Figure 3. *Types of atomic blocks in quadtree-based region edge maps: corner blocks* (c)*; interior blocks* (i)*; trivial horizontal boundary blocks* (h_t)*; nontrivial horizontal boundary block* (h_n)*; trivial vertical boundary blocks* (v_t)*; and nontrivial vertical boundary blocks* (v_n).

We assume that the encoding of the split/merge partition occurs after encoding the quadtree partition as given in the last Subsection 3.2. Thus, we may use the quadtree partition in the context modelling. We first note that the split/merge partition based on merging blocks of the quadtree partition can easily be encoded using the previous method for REM encoding when we define the region edge map in terms of a uniform image partition consisting of *atomic blocks* of the size that is given by the smallest blocks that occur in the quadtree partition. In other words, we imagine expanding the quadtree so that it becomes a full quadtree (all leaves have the same depth) and define a symbol $z \in \{0, 1, 2, 3\}$ for each leaf block depending on the presence of a region boundary along its Western and Northern edges as before (Figure 1(a)). We again encode the resulting symbol sequence line by line from top to bottom and from left to right. Let us call this symbol sequence the quadtree-based region edge map (QBREM). This sequence typically has many more symbols than there are blocks in the initial quadtree partition. However, since the quadtree is available at the coder and the decoder we may use it in order to design the context modelling to take advantage of this knowledge. For example, in this way the decoder is able to completely determine many symbols from their context alone without any additional information.

With the context modelling for region edge maps of Subsection 2.2 the coder maintains a table of up to four probabilities for each of 256 contexts. With the quadtree in the background we can differentiate between six types of atomic blocks (Figure 3), for each of which we propose to maintain a separate probability table. Some of these tables are trivial, however, and need not be explicitly implemented in a computer program. In the following paragraphs we describe the six cases and the facts that determine the structure of the corresponding probability tables.

Case 1 (corner block). The neighboring atomic block in the West and the block in the North do *not* belong to the quadtree block that contains the given *corner block*, see the blocks marked by "c" in Figure 3. In this case the quadtree partition does not lead to a simplification of the probability table. Thus, for corner blocks the same context model is used as in usual REM coding (Subsection 2.2).

Case 2 (interior block). The neighboring atomic blocks in the West and in the North both belong to the quadtree block that contains the given *interior block*, see the blocks marked by "i" in Figure 3. For such an interior block it is clear that neither the Northern nor the Western edge is a region edge. Thus, the symbol for any interior atomic block in a QBREM must be equal to 0. Formally, we may set the probability for symbol 0 equal to 1 for all of the 256 contexts, while all other probabilities are zero. Encoding the symbol 0 for an interior block does not require any rate.

Case 3 (horizontal boundary block). The neighboring atomic block in the West belongs to the quadtree block that contains the given *horizontal boundary block*, while the neighboring atomic block in the North

does not, see the blocks marked by "h_t" and "h_n" in Figure 3. In this case we further differentiate between a trivial and a nontrivial subcase.

Case 3*a* (trivial horizontal boundary block). The neighboring atomic blocks in the North and in the North–West both belong to the same quadtree block, see the blocks marked by "h_t" in Figure 3. This case is trivial because the symbol for the atomic block is already determined by that of the preceding block in the West. The atomic block has a region edge at its Northern edge if and only if the same holds for the neighboring block in the West. In terms of the probability table, it can be seen that the probability for symbol $X = 1$ must be set to 1 if $W = 1$ or $W = 3$, and the probability for symbol $X = 0$ must be set to 1 if $W = 0$ or $W = 2$. As for interior blocks, these trivial horizontal boundary blocks do not require any rate for encoding.

Case 3*b* (nontrivial horizontal boundary block). The neighboring atomic blocks in the North and in the North–West belong to different quadtree blocks, see the block marked by "h_n" in Figure 3. For these blocks, we can exploit only the fact that the neighbor in the West belongs to the same quadtree block and, thus, the Western edge of a *nontrivial horizontal boundary block* cannot be a part of a region boundary. For the probability table this implies that in addition to the simplification due to the impossible cases listed in Table 1 we have that the probabilities for the symbols $X = 2$ and $X = 3$ must both be zero for all context cases. For many of the contexts this lead to trivial cases. For example, suppose $W = 0$ and $N = 0$. Then $X = 1$ and $X = 2$ are ruled out, see Table 1. Since also $X = 3$ cannot occur for nontrivial horizontal boundary blocks, we conclude that only $X = 0$ is possible.

Case 4 (vertical boundary block). The neighboring atomic block in the North belongs to the quadtree block that contains the given *vertical boundary block*, while the neighboring atomic block in the West does not; see the blocks marked by "v_n" and "v_t" in Figure 3. Naturally, this case is similar to that of a horizontal boundary block discussed above. To complete the list of cases, we just state the trivial and the nontrivial subcases below omitting the details which are identical to Cases 3*a* and 3*b* except for an appropriate and straightforward adaptation.

Case 4*a* (trivial vertical boundary block).

Case 4*b* (nontrivial vertical boundary block).

Consider the case of nontrivial horizontal and vertical boundary blocks from Cases 3*b* and 4*b* above. We may ask which of the 256 contexts for either case leads to a nontrivial list of probabilities for the symbols $X = 0, 1$ (Case 3*b*) or $X = 0, 2$ (Case 4*b*). Taking into account Table 1 we deduce that the only contexts in which the symbol X is not already uniquely determined is given by $W = 1$ or $W = 2$ and $N = 2$ or $N = 3$. This holds for nontrivial horizontal as well as vertical boundary blocks. Together with $NW, NE = 0, 1, 2, 3$ we obtain 64 contexts in which $X \in \{0, 1\}$ for horizontal boundary blocks and $X \in \{0, 2\}$ for vertical boundary blocks. For simplicity and in order to avoid context dilution due to an insufficient number of occurrences of nontrivial boundary blocks we have joined the probability tables in our implementation for the horizontal and vertical cases, listing probabilities only for $X = 0$ and $X \neq 0$.

Special cases should also be considered for blocks at the Western and Northern image border, which is known to form region boundaries throughout. We omit these details. Alternatively, one may allow range blocks to wrap around image boundaries [16].

The proposed context modelling above yields larger probability tables than for the case of region edge maps without quadtree structure. For small images or for coarse quadtree partitions smaller context models may be preferable in order to avoid the context dilution problem. Therefore we also consider reduced context models in which some of the tables are merged. In a first step we may ignore the dependence on the symbol for the North East neighbor. Then we may drop the neighbors in the North West, North, and finally also the West. Table 2 shows how the number of non-redundant probabilities in the tables is reduced accordingly. The resulting

different context models are called QBREM{num}, where num = 0,..., 4 is the number of neighboring blocks used in the context model. In the same way, we define REM{0,1,2,3,4}, in which case only one table of probabilities is needed, whose size is the same as that for Case 1 of QBREM.

Table 2. (QB)REM with Different Context Models.

Type	Neighbor blocks used in context	Probs. Case 1	Probs. Case $3b/4b$
(QB)REM0	–	4	2
(QB)REM1	West	16	4
(QB)REM2	West, North	64	8
(QB)REM3	West, North West, North	192	32
(QB)REM4	West, North West, North, North East	768	128

The right hand columns list the number of (nontrivial) probabilities that need to be maintained in the tables of the context model.

3.4. **Results for Partition Coding**

We used our fractal image coder to produce a sequence of quadtree-based image partitions with varying numbers of regions. Here we consider only the performance of the encoding of the octrees alone and the partitions including the octrees. In Figure 4 we show the rates (cost in bits) for QBREM{2,3,4} compared to REM*. Here by REM* we mean that each of REM{0,1,2,3,4} has been tried and the context template with the lowest costs was taken at each point. In Figure 4(a) the encoding of the quadtree alone is studied. For the complete quadtree (with 32 384 leaves), the costs vanish. Regarding the different proposed context models, we observed that the sum context is least efficient, whereas the sum context and the reduced context have a better and comparable performance. Thus, in all further experiments we use the reduced context for encoding quadtrees. Figures 4(b–f) show the costs of the entire partition coding based on 4–32-quadtrees with various numbers of blocks. Generally, the partition coding with quadtree-based region edge maps is superior to that using plain region edge maps, which in turn is better than the chain coding given by [21]. The advantage of QBREM is especially large when the partition is based on quadtrees with relatively few blocks.

If the number of quadtree blocks is very large compared to the number of regions formed by merging (see Figure 4(b) and (e)), then coding the REMs is more economical than coding QBREMs. However, this drawback can be eliminated by using the quadtree fitting procedure explained in Subsection 3.1, see Figure 4(e).

4. **FRACTAL IMAGE CODING**

In this section we present implementation details of our fractal coding method. In particular we explain the parameters that must be chosen in the method.

4.1. **Domain Pool and Coefficients**

As usual [4], we consider domain blocks that are twice as large as range blocks. In [21, 30], it has been shown that domains with small collage error may be found near the range block. We make use of this observation by defining a square array of L^2 domains which is centered over the corresponding range. Two neighboring domains are offset horizontally or vertically by a fixed step size d (Figure 5). Domain blocks may wrap around image

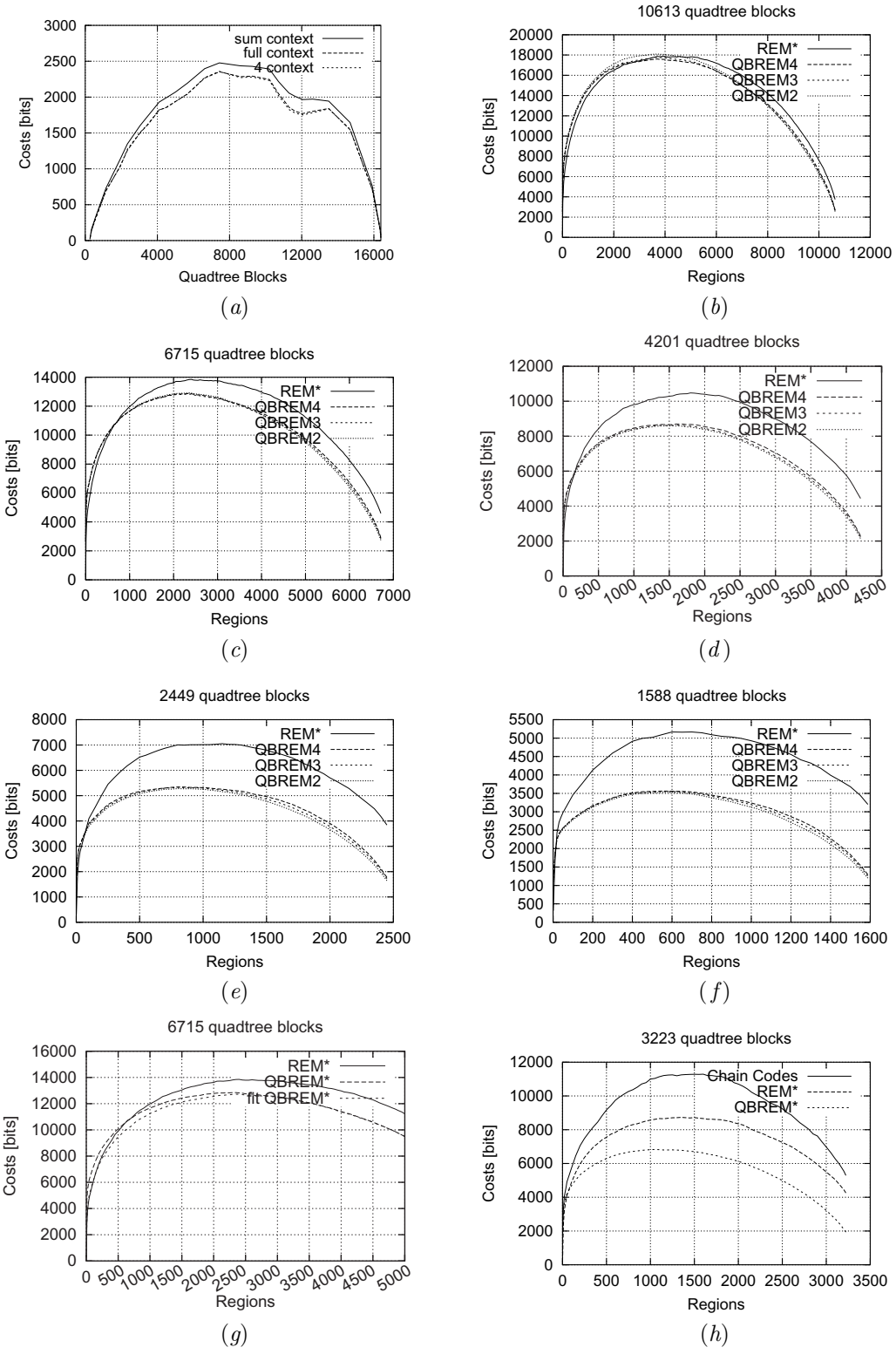


Figure 4. *Coding performance for quadtree-based region edge maps (QBREM) compared to plain region edge maps (REM) for partitions of the* 512×512*-Lenna image based on split/merge using* 4–32*-quadtrees:* (*a*) *compares the costs of different context models for quadtree coding;* (*b*)–(*f*) *compare the performance of encoding QBREMs based on quadtrees of differing complexities;* (*g*) *shows the benefits of additional quadtree fitting; and* (*h*) *compares encoding methods for QBREMs and REMs with the chain coding method in* [21].

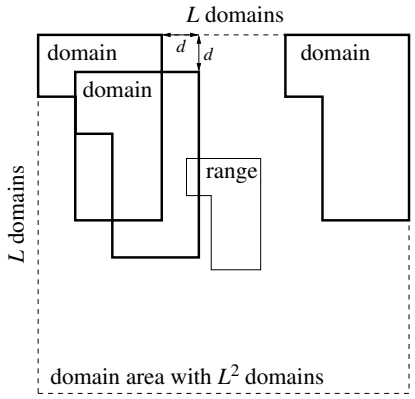


Figure 5. The domain pool for a given range is formed by domains that are twice as large as the range. L^2 domains are arranged around the range block with horizontal and vertical offsets of d pixels.

boundaries. The domain coordinates (x_d, y_d) in (2.1) can be encoded by an integer index $i_d = 0, \ldots, L^2 - 1$, since the decoder can reconstruct the domain pool from a given range block.

The transform parameters s, o are quantized with the method in [5], using representations of n_s and n_o bits, respectively. The quantized parameters and the domain address index i_d are encoded using adaptive arithmetic coding [28].

4.2. **Quadtree Partitioning in the Splitting Phase**

The quadtree partition proceeds along Algorithm 1. We restrict the block sizes in the partition to a minimum and maximum size, q^2_{min} and q^2_{max}, respectively (see Subsection 3.2). The split criterion is given by a threshold T on the mean square collage error $E(r)/|r|$ of a block r corresponding to a node of the tree. The node is expanded and the block is subdivided into four subblocks, if $E(r)/|r| > T$. As in [21, 31], we use an adaptive threshold T, which is multiplied by a factor k for blocks of the next deeper quadtree level. For the blocks of maximum size q^2_{max}, an initial threshold $T = T_1$ is used.

4.3. **Region Growing in the Merging Phase**

After the splitting phase, the quadtree partition P consists of a number of range blocks that depends on the image and on the parameters q^2_{min}, q^2_{max}, T_1, and k. In the subsequent merging phase an iteration takes place (Algorithm 2). In each iteration step neighboring pairs (r_i, r_j) of blocks are managed in a priority list structure L_P. From this list of candidate pairs an optimal pair (r_i, r_j) is merged to form a new larger block $r_i \cup r_j$ until the desired number of blocks, n_R, is reached. The pair that is actually merged is defined by minimizing the increase in collage error due to the merging, *i.e.*, by the minimal value of

$$\Delta E(r_i, r_j) = E(r_i \cup r_j) - E(r_i) - E(r_j).$$

After a merging step the number of range blocks in the partition P has decreased by 1. Then the list L_P of pairs of adjacents blocks and their associated collage error increases, ΔE, requires an update for the next iteration. Pairs involving r_i or r_j must be removed and new pairs with $r_i \cup r_j$ must be inserted. This procedure is computationally complex. Moreover, many collage errors are computed in vain, namely for those range block pairs that will never be merged.

To overcome this problem, a lazy evaluation strategy was proposed in [16]. The update of the priority list is not performed. Instead in each iteration the top pair (r_i, r_j) on the list L_P is processed as follows. If it is still a valid pair, *i.e.*, if both blocks r_i and r_j are still members of the current partition P, merging takes place and the iteration step is complete. If one of the two blocks, either r_i or r_j, is obsolete due to a previous merger, then the collage error for the enlarged block must be recomputed and the block pair is moved to the corresponding position in the priority list. If both of the blocks, r_i and r_j, have already been merged in previous steps, a new collage error needs to be computed only when the larger blocks that r_i and r_j belong to are disjoint (and thus form a new legal candidate pair for a merger).

This approach yields savings of more than 50% of the computation time compared to the complete updating. The procedure delivers the same result as the one using complete updating in each iteration step if $\Delta E(a \cup b, c) \geq \Delta E(b, c)$ holds for any given neighboring regions a, b, and c. Unfortunately, this condition is not guaranteed. However, in practice, for fractal image coding discussed here, only at very high compression ratios there is a small loss of about 0.05 dB in PSNR quality of the reconstructed image with respect to the merging method using a full update in each iteration step. For details see [32].

4.4. **Partition Coding**

For the encoding of the quadtree-based split/merge partition P we use our quadtree-based region edge maps as proposed in Section 3. We also try the different context models from Table 2 and choose the model that works best.

5. **RESULTS AND DISCUSSION**

In this section we present results for the proposed fractal image coder using quadtree-based region edge maps. For the experiments we chose the 512×512 images Lenna, Boat, Peppers, and Barbara from the Waterloo BragZone [33]. We used the following settings for the parameters of the fractal coder introduced in the last section.

Minimum and maximum allowed block size for the quadtree blocks ((q_{min}, q_{max}), see Subsection 3.2). For each bit rate the setting from $(4, 32)$, $(4, 64)$, $(5, 40)$, $(5, 80)$, $(6, 24)$, $(6, 48)$, $(7, 28)$, $(7, 56)$, $(8, 32)$ that led to the lowest distortion was selected.

Threshold and threshold factor for quadtree segmentation (T_1, k, see Subsection 4.2). Experiments showed that a value of $k = 2$ leads to good results in most cases. However, the differences in PSNR of the reconstructed images were very small compared to other values for k. The best value for the threshold $T_1 \in \{0, 5, \ldots, 100\}$ was determined experimentally for each bit rate.

Number of rows/columns of the domain arrays and step length (L, d, Subsection 4.1). Values of $L = 256$ and $d = 2$ spread the domains uniformly over the entire image and led to best rate–distortion performance.

Number of bits for quantization of fractal parameters (n_s and n_o, Subsection 4.1). We set $n_s = 4$ and $n_o = 6$. These values led to best rate–distortion performance at compression ratios larger than 30.

Number of regions in the split/merge partition (n_R, Subsection 4.3). The parameter varies and controls the bit rate and the distortion.

Figure 6 summarizes the results of our experiments. In part (a) we show that the choice of the minimal and maximal block sizes in the quadtree partition that provides the base blocks for the merging phase is crucial for the rate–distortion performance of the fractal codec. Especially for low compression ratios (below 40), we observed that a small minimal block size of 3×3 pixels provided much better results than larger sizes. For high

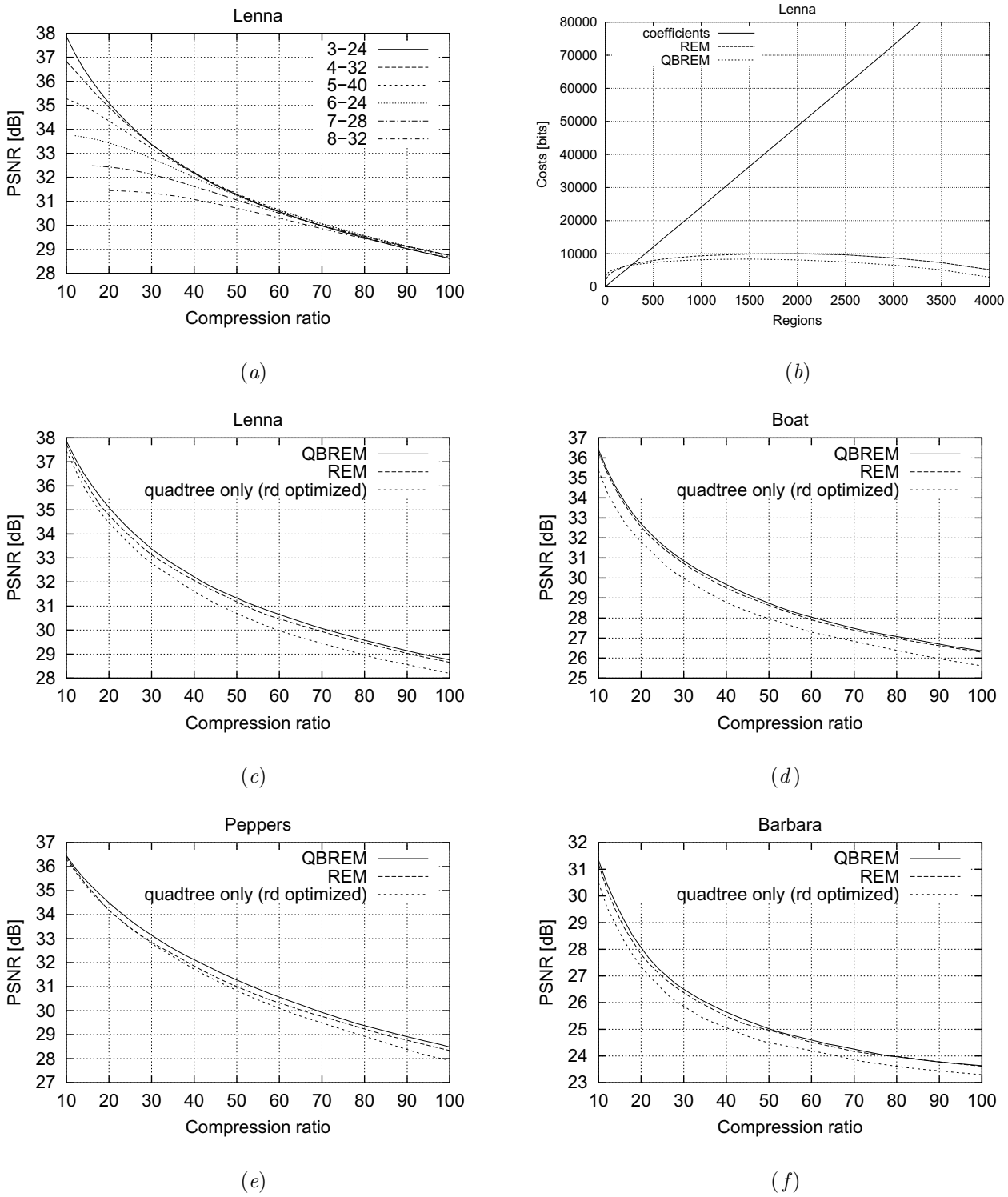


Figure 6. (*a*) *Comparison of different base block sizes for region merging (Lenna);* (*b*) *costs for fractal coefficients and domain addresses compared to partition coding costs for REM and QBREM (Lenna);* (*c–f*) *rate–distortion performance for four test images: quadtree-based split/merge approach versus merging of blocks from a uniform partition and fractal encoding with rate–distortion optimized quadtree (without block merging).*

Table 3. Coding Results for 512 × 512-Images Lenna, Boat, Peppers, and Barbara.

image	parameters			partition costs [bits]		coefficient	compression	PSNR
	q_{min}, q_{max}	T_1	n_R	total	quadtree	costs [bits]	ratio	[dB]
Lenna	4, 32	7	2500	12727	2606	60735	28.55	33.52
	5, 80	7	1140	7346	1703	27690	59.85	30.62
	6, 24	45	760	5076	1206	18392	89.36	29.13
Boat	4, 32	5	2250	15686	2626	54632	29.82	30.85
	4, 64	25	1115	7798	2169	27081	60.13	28.01
	7, 28	15	720	5774	696	17495	90.13	26.68
Peppers	3, 24	15	2305	13382	3673	56639	29.95	33.14
	4, 32	30	1160	6673	1891	28215	60.11	30.58
	7, 56	25	775	4444	1148	18706	90.60	28.89
Barbara	3, 24	15	3525	21490	3793	82951	20.08	28.04
	4, 32	5	2250	16586	2939	53030	30.13	26.43
	4, 32	50	1120	9722	2277	26531	57.85	24.58

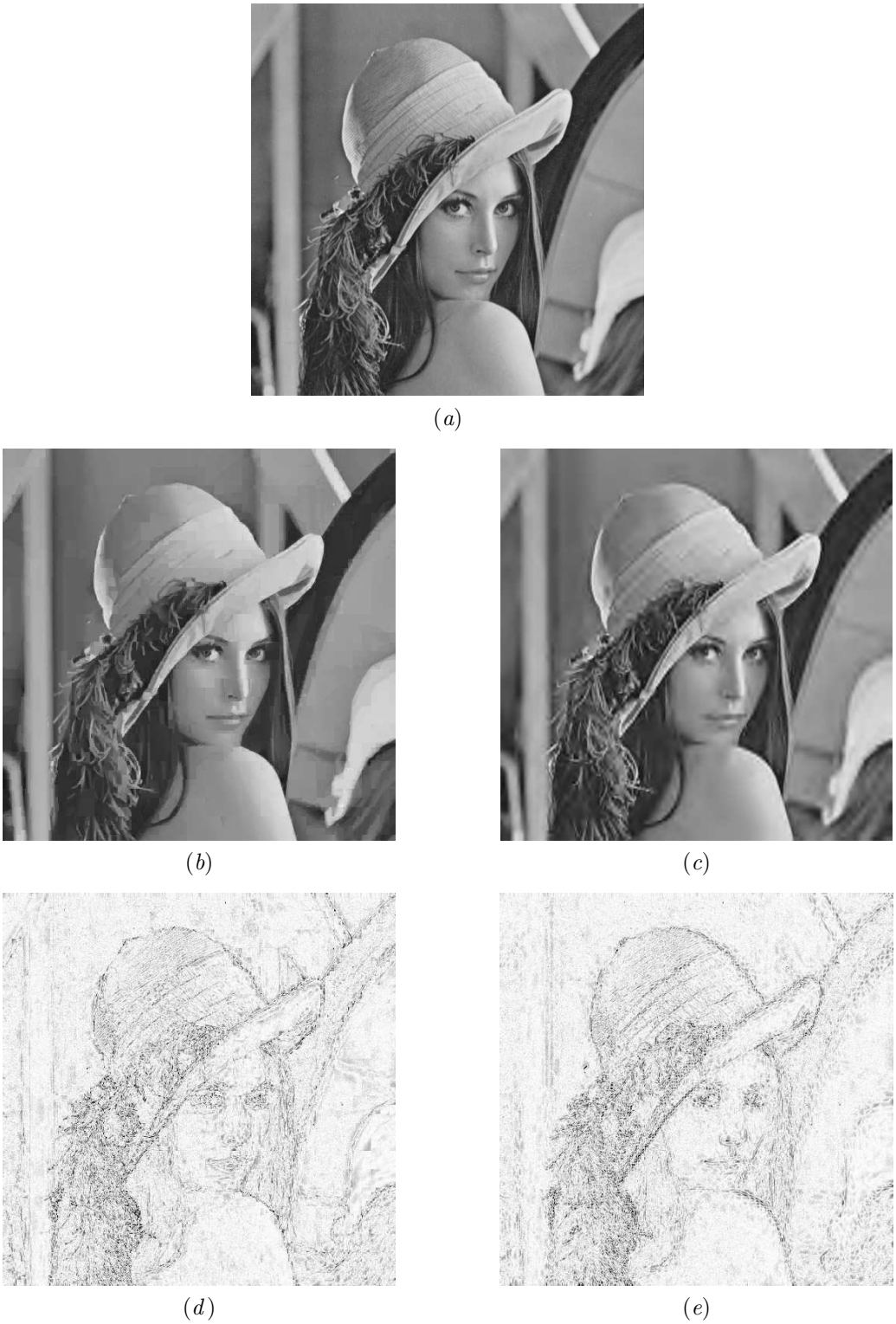
$k = 2,\ \ L = 256,\ \ d = 2,\ \ n_s = 4,\ \ n_o = 6.$

compression ratios (above 80) larger blocks sizes gave better PSNR performance, however, only by a very small amount.

Part (*b*) of Figure 6 displays how much of the bit rate was spent on the adaptive split/merge partition in comparison to the rest of the code consisting of domain addresses and coefficients. It also demonstrates the advantage of our coding scheme exploiting the underlying quadtree structure, as opposed to encoding plain region edge maps. The difference was as large as about 2000 bits for medium compression ratios.

Parts (*c*–*f*) of Figure 6 shows complete rate–distortion curves of our fractal coder with quadtree-based region edge maps, with plain region edge maps, and with a quadtree partition without subsequent block merging. Each of the plots is for one of the four test images. The plain quadtree coder is superior to that of Fisher [5]. We implemented a rate–distortion optimization that is guaranteed to select the subtree of the complete quadtree that yields minimum collage error using optimization methods given in [34, 35]. Moreover, for the quadtree encoding we used arithmetic coding with context modelling as proposed in this paper. Our results indicate that fractal coders that use an adaptive split/merge image partition may outperform optimized fractal quadtree coders by about 0.5 to 1 dB in PSNR. Using a quadtree as a base partition in the split/merge partitions provided gains of about 0.2 dB in PSNR. Table 3 lists detailed numerical coding results obtained with the proposed method.

In Figures 7 and 8 we present reconstructions of encoded Lenna and Peppers images at compression ratios about 70. Parts (*a*) show the original image for reference, parts (*b*) show the reconstructed images (detailed parameters are given in the captions). Parts (*c*) show images decoded from JPEG2000 encoded originals.



(*a*)

(*b*)

(*c*)

(*d*)

(*e*)

Figure 7. *Coding example for* 512×512*-image Lenna with parameters* $q_{min} = 6$, $q_{max} = 48$, $T_1 = 15$, $k = 2$, $L = 256$, $d = 2$, $n_s = 4$, $n_o = 6$, *and* $n_R = 990$. (*a*) *Original image;* (*b*) *decoded image with proposed fractal coder,* PSNR = 30.00 dB, *compression ratio* = 70.29*;* (*c*) *decoded image from encoding with* JPEG2000 *(kakadu encoder),* PSNR = 30.40 dB, *compression ratio* = 71.02*;* (*d*) *difference between fractal reconstruction and original;* (*e*) *difference between* JPEG2000 *reconstruction and original.*

(*a*)

(*b*)

(*c*)

(*d*)

(*e*)

Figure 8. *Coding example for* 512×512*-Peppers with parameters* $q_{min} = 4$, $q_{max} = 32$, $T_1 = 40$, $k = 2$, $L = 256$, $d = 2$, $n_s = 4$, $n_o = 6$, *and* $n_R = 990$. (*a*) *Original image;* (*b*) *decoded image with proposed fractal image coder,* PSNR = 29.91 dB, *compression ratio* = 70.30*;* (*c*) *decoded image from encoding with* JPEG2000 *(kakadu encoder),* PSNR = 30.24 dB, *compression ratio* = 69.46*;* (*d*) *difference between fractal reconstruction and original;* (*e*) *difference between* JPEG2000 *reconstruction and original.*

For the JPEG2000 encodings we used the kakadu encoder [36]. Parts (d) and (e) show the inverted difference images between decoded and original images. The difference images were obtained from the absolute differences between grey values of corresponding pixels, scaled by a factor of 5 and clamped at 255. The rate distortion performance of our coder is comparable to that of the JPEG2000 coder with an advantage of JPEG2000 of about 0.3 to 0.5 dB PSNR. The decoded images and the difference images, however, show much more noticeable differences in terms of typical artifacts. While the images obtained from JPEG2000 codes look defocussed especially around edges, here the fractally decoded images clearly have sharper edges. This impression is confirmed by the difference images which show broader edge artifacts for the wavelet coded images of JPEG2000. On the other hand there are blocking artifacts in smooth image regions of the fractally decoded images. We remark that there are postprocessing techniques that may improve the sharpness of the JPEG2000 reconstructions [37] and reduce the blocking artifacts of fractally encoded images [38, 21].

Finally, in Figure 9 we display quadtree partitions for two test images that provided the base blocks for the merging phase. The resulting adaptive split/merge partitions are also shown.

The survey in [6] showed a comparison of the performance of published pure fractal coding methods that are working in the spatial domain ([6, Figure 6]). This comparison includes methods based on various types of adaptive image partitions, such as quadtrees [5], HV-partitions [5], Delaunay-triangulations [11, 12], and irregular partitions [14, 15, 20]. The rate–distortion performance of our coder is clearly superior to all of these.

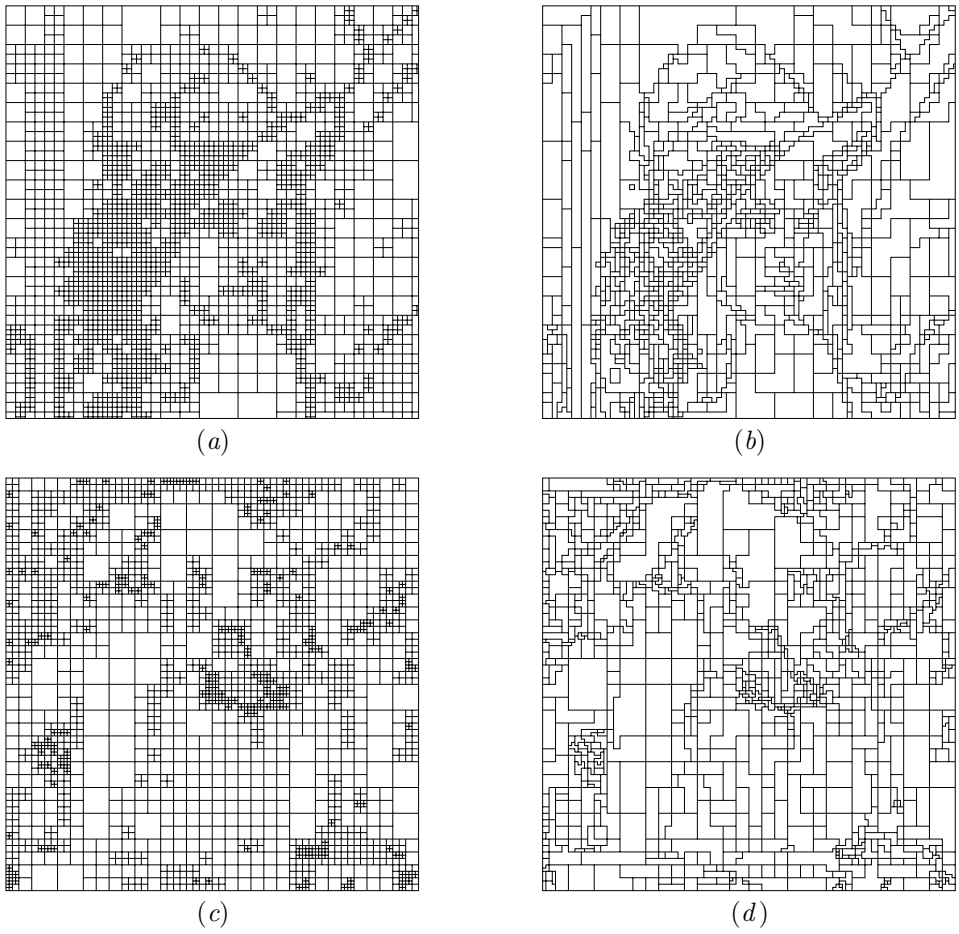


Figure 9. *Partitions for the encodings in Figures* 7 *and* 8. (a) *Lenna image, quadtree partition with* 1345 *blocks before merging;* (b) *partition with* 990 *regions after merging quadtree blocks;* (c) *Peppers, quadtree partition with* 1698 *blocks before merging;* (d) *partition with* 990 *regions after merging quadtree blocks.*

Fractal image coding with quadtree-based split/merge partitions was first proposed in [20, 21], where the chain coding method of [25] was adapted. Our reimplementation of the chain coder showed that our QBREM coding algorithm compressed partitions with only half the rate, see Figure 4(h).

6. CONCLUSIONS

We have proposed a fractal image compression method based on split/merge image partitions. In the splitting phase an image adaptive quadtree partition is generated. Neighboring blocks are iteratively merged, using an appropriate error criterion. We provided an algorithm for lossless arithmetic compression of the resulting image partition with suitable context modelling that exploits the structure of the underlying quadtree. Experimental results show that our method can save up to 30% of the total rate for the partition in comparison to encoding with region edge maps based on uniform base partitions. When applied to fractal image coding we obtained rate–distortion performances that exceed the best known results for pure, non-hybrid fractal image coders. Compared to state of the art wavelet coders in industry, given by the JPEG2000 standard, our fractal compression method is comparable in terms of rate–distortion performance up to less than 0.5 dB PSNR. Subjectively, fractally reconstructed images have sharper edges and more contrast than JPEG2000 reconstructions but suffer from mild blocking artifacts.

Our coding method for quadtree-based region edge maps can also be used in connection with other image coding systems that rely on adaptive partitions. For example, similar partitions have been used with dynamic coding [39], or one may modify the segmentation-based image coding method in [40] to make use of quadtree-based split/merge partitions.

ACKNOWLEDGMENT

This work was supported by Grant Sa449/8-2 of the Deutsche Forschungsgemeinschaft.

REFERENCES

[1] M.F. Barnsley and A.D. Sloan, "Chaotic Compression", *Computer Graphics World*, 1987, pp. 107–108.

[2] M.F. Barnsley and A.D. Sloan, "A Better Way to Compress Images", *Byte*, 1988, pp. 215–223.

[3] A.E. Jacquin, "A Fractal Theory of Iterated Markov Operators with Application to Digital Image Coding", *Ph.D. Dissertation, Georgia Institute of Technology*, 1989.

[4] A.E. Jacquin, "Image Coding Based on Fractal Theory of Iterated Contractive Image Transformations", *IEEE Transactions on Image Processing*, **1**(**1**) (1992), pp. 18–30.

[5] Y. Fisher, *Fractal Image Compression — Theory and Application*. New York: Springer Verlag, 1995.

[6] B. Wohlberg and G. de Jager, "A Review of the Fractal Image Coding Literature", *IEEE Transactions on Image Processing*, **8**(**12**) (1999), pp. 1716–1729.

[7] D. Saupe, R. Hamzaoui, and A. Zerbst, "The Paper Collection on Fractal Image Compression", http://www.inf.uni-konstanz.de/cgip/fractal/papers.html

[8] Y. Fisher and S. Menlove, "Fractal Encoding with HV Partitions", in *Fractal Image Compression — Theory and Application*. New York: Springer-Verlag, 1995, pp. 119-126.

[9] D. Saupe, M. Ruhl, R. Hamzaoui, L. Grandi, and D. Marini, "Optimal Hierarchical Partitions for Fractal Image Compression", *Proceedings IEEE International Conference on Image Processing*, 1998, pp. 737–741.

[10] E. Reusens, "Partitioning Complexity Issue for Iterated Function Systems Based Image Coding", *Proceedings VIIth European Signal Processing Conference,* **I** (1994), pp. 171–174.

[11] F. Davoine, J.-M. Chassery, "Adaptive Delaunay Triangulation for Attractor Image Coding", *Proceedings 12th International Conference on Pattern Recognition*, 1994, pp. 801–803.

[12] F. Davoine, J. Svensson, and J.-M. Chassery, "A Mixed Triangular and Quadrilateral Partition for Fractal Image Coding", *Proceedings IEEE International Conference on Image Processing*, 1995.

[13] R.C. Gonzales and R.E. Woods, *Digital Image Processing*. Reading, Massachusetts: Addison-Wesley, 1993.

[14] L. Thomas and F. Deravi, "Region-Based Fractal Image Compression Using Heuristic Search", *IEEE Transactions on Image Processing*, **4**(**6**) (1995), pp. 832–838.

[15] M. Ruhl, H. Hartenstein, and D. Saupe, "Adaptive Partitionings for Fractal Image Compression", *Proceedings IEEE International Conference on Image Processing*, 1997.

[16] H. Hartenstein, M. Ruhl, and D. Saupe, "Region-Based Fractal Image Compression", *IEEE Transactions on Image Processing*, **9**(**7**) (2000), pp. 1171–1184.

[17] H. Freeman, "On the Encoding of Arbitrary Geometric Configurations", *IRE Transactions on Electronic Computers*, **EC-10** (1961), pp. 260–268.

[18] S.R. Tate, "Lossless Compression of Region Edge Maps", *Technical Report DUKE–TR–1992–09, Duke University, Durham*, 1992.

[19] S.L. Horowitz and T. Pavlidis, "Picture Segmentation by a Tree Traversal", *Journal of the ACM*, **23**(**2**) (1976), pp. 368–388.

[20] Y.C. Chang, B.K. Shyu, and J.S. Wang, "Region-Based Fractal Image Compression with Quadtree Segmentation", *Proceedings IEEE International Conference on Acoustics, Speech and Signal Processing*, **4** (1997), pp. 3125–3128.

[21] Y.C. Chang, B.K. Shyu, and J.S. Wang, "Region-Based Fractal Compression for Still Image", *Proceedings 8-th Internations Conference in Central Europe on Computer Graphics, Visualization and Interactive Digital Media*, 2000.

[22] M. Barnsley, *Fractals Everywhere*. San Diego: Academic Press, 1988.

[23] M. Barnsley and L. Hurd, *Fractal Image Compression*. Wellesley, Mass.: AK Peters Ltd, 1993.

[24] M. Eden and M. Kocher, "On the Performance of a Contour Coding Algorithm in the Context of Image Coding. Part I: Contour Segment Coding", *Proceedings IEEE Internations Conference on Acoustics, Speech and Signal Processing*, **8** (1985), pp. 381–386.

[25] T. Ebrahimi, "A New Technique for Motion Field Segmentation and Coding for Very Low Bitrate Video Coding Applications", *Proceedings IEEE International Conference on Image Processing*, **2** (1994), pp. 433–437.

[26] D. Saupe and M. Ruhl, "Evolutionary Fractal Image Compression", *Proceedings IEEE International Conference on Image Processing*, 1996.

[27] G.G. Langdon and J. Rissanen, "Compression of Black–White Images with Arithmetic Coding", *IEEE Transactions on Communications*, **29**(**6**) (1981), pp. 858–867.

[28] I.H. Witten, R.M. Neal, and J.G. Cleary, "Arithmetic Coding for Data Compression", *Communications of the ACM*, **30**(**6**) (1987), pp. 520–540.

[29] H. Samet, "The Quadtree and Related Hierarchical Data Structure", *ACM Computing Surveys*, **16**(**2**) (1984), pp. 187–260.

[30] K.U. Barthel, T. Voyé, and P. Woll, "Improved Fractal Image Coding", *Proceedings International Picture Coding Symposium*, 1993.

[31] E. Shusterman and M. Feder, "Image Compression via Improved Quadtree Decomposition Algorithms", *IEEE Transactions on Image Processing*, **3**(**2**) (1994), pp. 207–215.

[32] T. Ochotta, "Regionenbasierte Partitionierung bei fraktaler Bildkompression mit Quadtrees", *Diplomarbeit, Institut für Informatik, Universität Leipzig, Germany*, 2002.

[33] J. Kominek, *Waterloo BragZone*. University of Waterloo, Canada, http://www.uwaterloo.ca/bragzone.base.html.

[34] G. Sullivan and R. Baker, "Efficient Quadtree Coding of Images and Video", *IEEE Transactions on Image Processing*, **3**(**3**) (1994), pp. 327–331.

[35] R. Hamzaoui and D. Saupe, "Combining Fractal Image Compression and Vector Quantization", *IEEE Transactions on Image Processing*, **9**(**2**) (2000), pp. 197–208.

[36] D. Taubman, "The Sporty New JPEG Model", *SPIE's oemagazine*, 2002, pp. 38–40.
(Kakadu Software http://www.kakadusoftware.com)

[37] G. Fan and W.K. Cham, "Post-Processing of Low Bit-Rate Wavelet-Based Image Coding Using Multiscale Edge Characterization", *IEEE Transactions on Circuits and Systems for Video Technology*, **11**(**12**) (2001), pp. 1263–1272.

[38] K.G. Nguyen and D. Saupe, "Adaptive Postprocessing for Fractal Image Compression", *Proceedings IEEE International Conference on Image Processing*, 2000.

[39] E. Reusens, T. Ebrahimi, C. Le Buhan, R. Castagno, V. Vaerman, L. Piron, C. de Solà Fàbregas, S. Bhattacharjee, F. Bossen, and M. Kunt, "Dynamic Approach to Visual Data Compression", *IEEE Transactions on Circuits and Systems for Video Technology*, **7**(**1**) (1997), pp. 197–211.

[40] X. Wu, "Image Coding by Adaptive Tree-Structured Segmentation", *IEEE Transactions on Information Theory*, **38**(**6**) (1992), pp. 1755–1767.