Pattern Recognition 45 (2012) 4279–4291

Contents lists available at SciVerse ScienceDirect

Pattern Recognition

journal homepage: www.elsevier.com/locate/pr

Decomposition of binary images—A survey and comparison

Tomáš Suk*, Cyril Höschl IV, Jan Flusser

Institute of Information Theory and Automation of the ASCR, Pod vodárenskou věží 4, 182 08 Praha 8, Czech Republic

ARTICLE INFO

Article history:
Received 10 February 2012
Received in revised form
24 April 2012
Accepted 16 May 2012
Available online 26 May 2012

Keywords:
Binary image decomposition
Distance transform
Quadtree
Bipartite graph
Image compression
Moment computation
Fast convolution

ABSTRACT

We present an overview of the most important methods that decompose an arbitrary binary object into a union of rectangles. We describe a run-length encoding and its generalization, decompositions based on quadtrees, on mathematical morphology, on the distance transform, and a theoretically optimal decomposition based on a maximal matching in bipartite graphs. We compare their performance in image compression, in moment computation and in linear filtering. We show that the choice is always a compromise between the complexity and time/memory consumption. We give advice how to select an appropriate method in particular cases.

1. Introduction

It is intuitively clear and well known that binary images can be represented in a more efficient way than just as a full-sized matrix consisting of zeros and ones. It is also clear that the terms "good representation" and "optimal representation" cannot be generally defined and are always dependent on what we are going to do with the object. Usually there are two basic requirements on the representation—small time/memory consumption and possibility of fast further computation (which is application-dependent). Sometimes we also require fast recovery of the original.

Regardless of the particular purpose, the methods of binary image representation can be divided into two groups referred as *decomposition methods* and *boundary-based methods*. Boundary-based methods employ the property that the boundary of a binary object contains a complete information on the object, all other pixels are redundant. Provided that the boundary consists of much less pixels than the whole object (which applies to "normal" shapes but does not hold true in general), it provides efficient non-redundant representation. Individual boundary-based methods differ from each other by a boundary definition (outer or inner boundary), by a discrete topology used (4-pixel or 8-pixel connectivity) and by the way how the boundary is encoded and stored (chain codes and various piece-wise approximations are mostly used for this purpose).

Decomposition methods try to express the object as a union of simple disjoint subsets called *blocks* or *partitions* which can be effectively stored and consequently used for required processing. Having a binary object B (by a binary object we understand a set of all pixels of a binary image whose values equal one), we decompose it into $K \geq 1$ blocks $B_1, B_2, \ldots, B_K$ such that $B_i \cap B_j = \emptyset$ for any $i \neq j$ and $B = \bigcup_{k=1}^{K} B_k$. Although in a continuous domain we may consider various shapes of the blocks (convex, star-shaped, hexagonal, rectangular, etc., see [1]), all decomposition methods that perform in a discrete domain use only rectilinear rectangular or square blocks because of a native rectangular structure of the discrete image domain. The methods differ from one another namely by the decomposition algorithms.

The power of any decomposition method depends on its ability to decompose the object into a small number of blocks in a reasonable time. Most authors have measured the decomposition quality just by the number of blocks K, while ignoring the complexity of the algorithms (it should be noted that there exist a few other criteria such as the "minimum ink" criterion which minimizes the overall length of the inner boundary but they are out of the scope of this paper). There is a common belief that such decomposition that minimizes K is the optimal one. This criterion is justified by the fact that the complexity of subsequent calculations uses to be $\mathcal{O}(K)$ and compression ratio (if the decomposition is used for compression purposes) also increases as the number of blocks decreases. However, this viewpoint may be misleading. Simple algorithms produce relatively high number of blocks but

* Corresponding author. Tel.: +420 26605 2231; fax: +420 28468 0730.
E-mail addresses: suk@utia.cas.cz (T. Suk), hoschl@utia.cas.cz (C. Höschl IV), flusser@utia.cas.cz (J. Flusser).

0031-3203/$ - see front matter © 2012 Elsevier Ltd. All rights reserved.
http://dx.doi.org/10.1016/j.patcog.2012.05.012

perform fast, while more sophisticated decomposition methods end up with small number of blocks but require more time. Even if the decomposition is performed only once per object in most tasks and can be done off-line, the time needed for decomposing the image is often so long that it substantially influences the efficiency of the whole method.

Image rectangular decomposition has found numerous straightforward applications in image compression methods and formats (RLE, TIFF, BMP and others) and in calculation of image features (mainly moments and moment invariants) used subsequently for object description and recognition [2–9]. Other applications may be found in image spatial filtering and restoration, in integrated circuits design and in other areas. Convolution with a constant rectangular mask can be performed in $\mathcal{O}(1)$ time per pixel if the matrix of partial sums is precomputed. If the mask is constant or piecewise constant but not rectangular, its support can be decomposed into rectangles and the convolution in one pixel can be calculated in $\mathcal{O}(K)$ time as a sum of partial $\mathcal{O}(1)$ convolutions. In VLSI design, the decomposition problem appeared many years ago—the masks are rectilinear polygons (often very complex) which should be decomposed into rectangles in such a way that the pattern generator can effectively generate the mask. The time needed for a mask generation is proportional to the number of rectangles, so it is highly desirable to minimize their number [10,11].

The aim of this paper is to present a survey and a comparison of existing decomposition methods. To ensure an unbiased comparison, all reviewed methods were implemented on the same platform and run on the same computer. We compare their performance in three common tasks—loss-less compression, calculation of image moments and convolution with a binary mask. We show that there is no "generally optimal" decomposition method and explain the pros and cons of individual algorithms.

2. Decomposition methods

In this section, we present a brief survey of the most common decomposition methods. Their performance in real-data experiments is compared in Sections 3–5.

2.1. Decomposition into row segments

Decomposition of an object into rows or columns is the most straightforward and the oldest method. The blocks are continuous row segments for which only the coordinate of the beginning and the length is stored. In image compression, this has been known as run-length encoding (RLE). This principle and its modifications (CCITT, PackBits) are used in several image formats such as TIFF and BMP. In feature calculation, Zakaria et al. [2] used the same representation for fast computation of image moments of convex shapes and called it "Delta-Method" (DM) since the lengths of the row segments were labeled by the Greek letter δ. The method was slightly improved by Dai et al. [3] and generalized for non-convex shapes by Li [4].

The decomposition into rows is very fast but leads to the number of blocks which uses to be (much) higher than the minimal decomposition. A simple but powerful improvement of the Delta-Method was proposed by Spiliotis and Mertzios [5] and adopted by Flusser [6]. This "Generalized Delta-Method" (GDM) employs a rectangular-wise object representation instead of the row-wise one. The adjacent rows are compared and if there are some segments with the same beginning and end, they are unified into a rectangle (see Fig. 1). For each rectangle, the coordinates of its upper-left corner, the length and the width are stored. GDM is only slightly slower than DM while producing

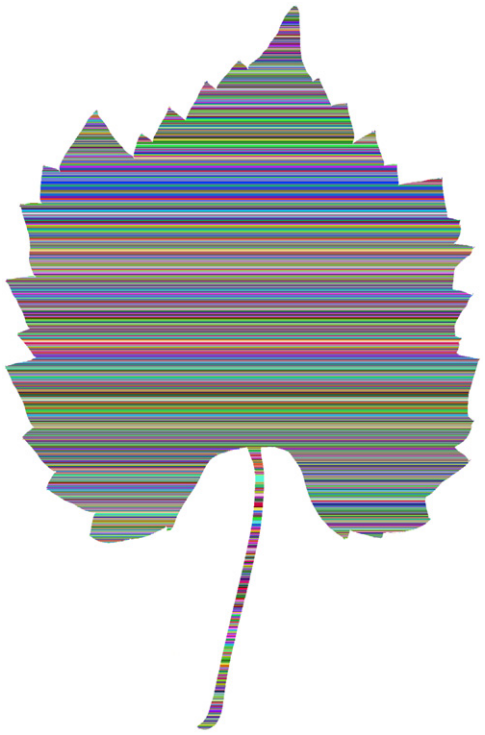

Fig. 1. Decomposition of the leaf image by the Generalized Delta-Method (GDM). The size of the original image is 1196×1828 pixels. The adjacent rows of the same length are unified into blocks (1963 blocks in total).

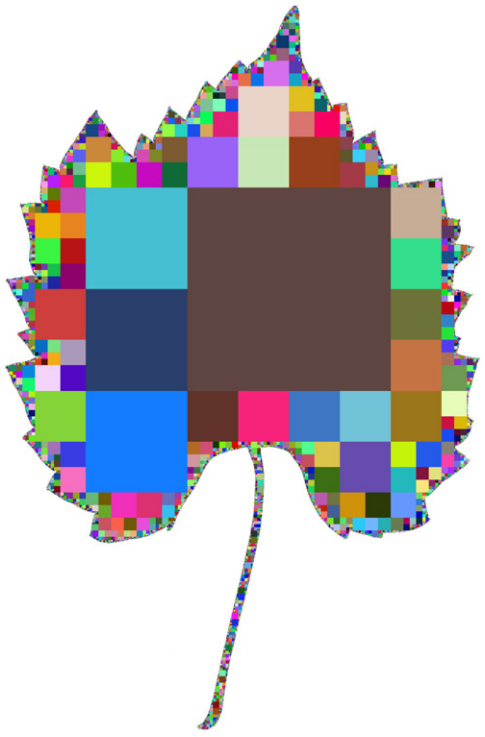

Fig. 2. Decomposition of the leaf image by the quadtree method (8911 blocks in total).

(sometimes significantly) less number of blocks. Surprisingly, under our knowledge this method has not been implemented in any commercial image format.

2.2. Quadtree decomposition

Quadtree decomposition (QTD) is a popular hierarchical decomposition scheme used in several image processing areas including representation and compression [12], spatial transformations [13] and feature calculation [7]. In its basic version, QTD works with square images of a size of a power of two. If this is not the case, the image is zero-padded to the nearest such size. The image is iteratively divided into four quadrants. Homogeneity of each quadrant is checked and if the whole quadrant lies either in the object or in the background it is not further divided. If it contains both object and background pixels, it is divided into quadrants and the process is repeated until all blocks are homogeneous. The decomposition can be efficiently encoded into three-symbol string, where 2 means division, 1 means a part of the object and 0 stands for the background. An example of the quadtree decomposition is in Fig. 2.

The algorithm always yields square blocks which may be advantageous for some purpose but usually leads to a higher number of blocks than necessary. It would be possible to implement a backtracking and to unify the adjacent blocks of the same size into a rectangle but this would increase the complexity. Since the speed is the main advantage of this method, the backtracking is mostly not employed here. A drawback of this decomposition algorithm is that the division scheme is not adapted with respect to the content of the image but it is defined by absolute spatial coordinates. Hence, the decomposition is not translation-invariant and may lead to absurd results when for instance a large single square is uselessly decomposed up to individual pixels. We may use a bintree or other trees producing non-square blocks but it does not overcome this principal weakness.

2.3. Morphological decomposition

In order to better adapt the decomposition to the image content, Sossa-Azuela et al. [8] published a decomposition algorithm based on a *morphological erosion*. The erosion is an operation, where a small structural element (here 3×3 square is used) moves over the image and when the whole element lies in the object, then the central pixel of the window is preserved in the object, otherwise it is assigned to the background. So, each erosion shrinks the object by one-pixel boundary layer. The decomposition works in an iterative manner: it finds the largest square inscribed in the object, removes it and looks for the largest square inscribed in the rest of the object. This outer loop is repeated until the object is completely decomposed. An intermediate object decomposition after two outer loops can be seen in Fig. 3, the final decomposition is in Fig. 4.

Although the original method [8] considers decomposition into squares only, it can be generalized also to rectangles which decreases the number of the resulting blocks. The inner loop serves for finding the center and the size of the largest inscribed square/rectangle. We repeat the erosion until the whole object disappears and count the number of erosions s. Then a $(2s-1) \times (2s-1)$ square can be inscribed into the object and it forms one block of the decomposition. The pixels of the object before the last disappearing erosion are potential centers of the inscribed square. Theoretically, we can choose one of them randomly, but the "corner" pixels provide better odds to more compact rest of the object. If the potential square centers create a line segment, then the corresponding inscribed squares can be unified into a rectangle.

It is possible, especially in the last steps of the method, that several of the identically sized squares (overlapping as well as non-overlapping) can be inscribed into different places of the object. Of course, it is possible to inscribe and remove one of them, repeat the erosions, inscribe and remove another one, etc.

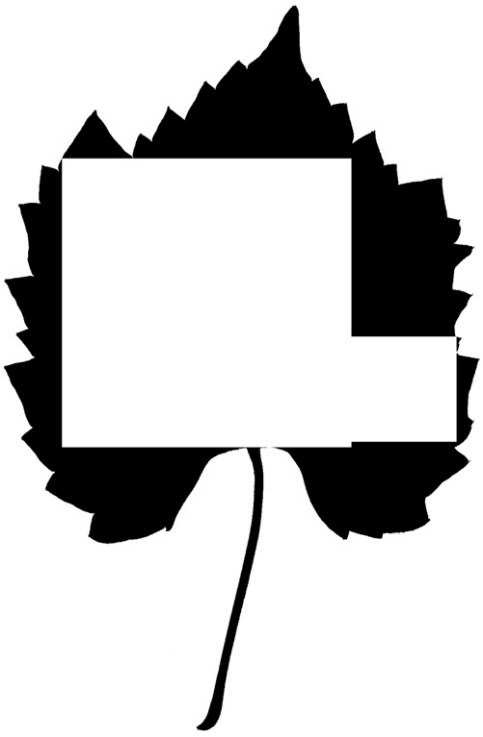

Fig. 3. The leaf image after removing two largest square blocks.

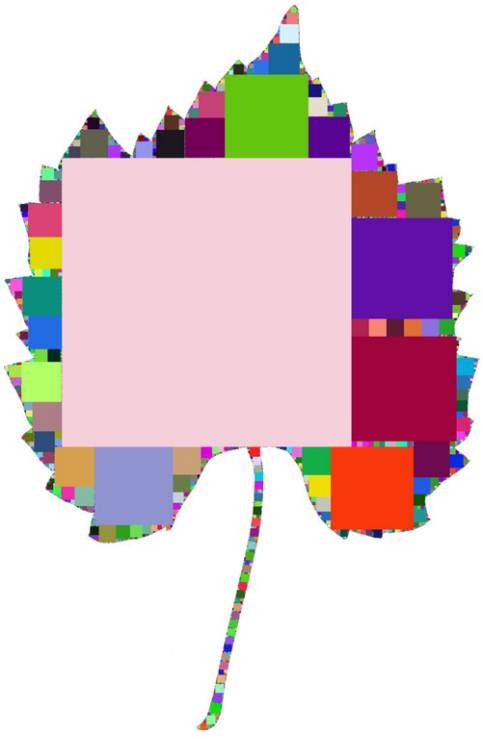

Fig. 4. Morphological decomposition of the leaf image to squares (7489 blocks in total).

A better approach is, after inscribing and removing one of them, to remove the centers of the squares that would overlap this one and search a center of another block of this size without repeating the erosions.

Vizireanu (e.g. [14]) generalized the morphological decomposition for other applications as skeleton computation or image interpolation. The skeleton can be used for image compression,

but its utilization for feature computation is difficult. The interpolation of an image between two frames in a video sequence can be computed even for gray-scale images.

2.4. Distance transform decomposition

If the object is sufficiently compact, i.e. the largest inscribed square is bigger than a certain minimum size, we can speed up the previous algorithm by means of the *distance transform* (DT), which we use for the finding of the centers of the inscribed rectangles [9]. In morphological decomposition, we must repeat the erosions s-times for finding $(2s-1)\times(2s-1)$ inscribed square, while the distance transformation with a suitable metric can be calculated only once. DT of a binary image is an image, where each object pixel shows the distance to the nearest boundary pixel and the background pixels are zero [15].

DT strongly depends on the metric used for the distance measurement. We use a simplified version of the Seaidoun's algorithm [16] for the chessboard metric

$$d(a,b)=\max\{|a_x-b_x|,|a_y-b_y|\}. \quad (1)$$

We successively search the image from the left, right, top and bottom, count distances from the last boundary pixel and calculate the minimum from the four directions. The result is DT, the maximum of the result equals s for the inscribed square $(2s-1)\times(2s-1)$ and the pixels with this maximum value are potential centers of the inscribed squares.

We use an improved version of DT inspired by [17]. If we change only a small part of the original image, then upgrading its close neighborhood is sufficient. In our case, if we remove a rectangle from the image, then the rectangle is zeroed and DT is recomputed in a small frame around it. If the frame in some distance d from the rectangle is not changed, then the rest of DT is left unchanged. In Fig. 5, we can see the visualization of DT of the leaf image.

Similarly to the morphological decomposition, the algorithm consists on iterative repeating the following loop until the object is fully decomposed. In each run, the largest inscribed rectangular block is found. Its potential center(s) are the pixels with the maximum values of DT. If the maximum s is unique, i.e. there is a single pixel with value s only, then a square $(2s-1)\times(2s-1)$ is inscribed. This is however a rare case, often the maximum is not unique and the choice of the block center is ambiguous. We try to keep the blocks as large as possible. Hence, if there is a 2×2 square of the maxima, an even-sized square $2s\times2s$ can be inscribed into the object. If the potential square centers create a line segment (with a single or double-pixel width), then the corresponding squares are unified into one rectangle like in the morphological method. At the end of the loop, the inscribed rectangle is removed from the object and the procedure starts the next loop, which is applied to the rest of the image. The decomposed leaf image is shown in Fig. 6.

Fig. 5. The distance transform of the leaf image. The pixels are labeled by pseudocolors according to the DT values.

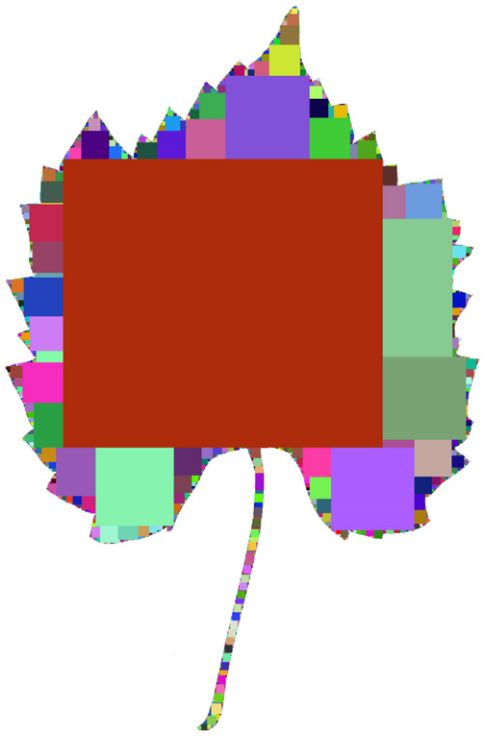

Fig. 6. The distance transform decomposition of the leaf image (2482 blocks in total).

Both morphological and DT decompositions end up with the same set of blocks. However, they are still only sub-optimal even on many simple shapes. This is because a sequence of locally optimal steps that these "greedy" algorithms apply to an image (placing always the largest inscribed rectangle) may not yield an optimal solution. As soon as a block is created, it cannot be removed any more, because the methods do not include any backtracking. These two algorithms differ from each other by computing complexity. Erosion is a relatively complex operation, while the DT can be calculated faster thanks to its simple upgrading. Anyway, both methods perform slower comparing to the previous decomposition algorithms.

2.5. Graph-based decomposition

A large group of decomposition algorithms appeared in 1980s in computational geometry [1]. Surprisingly, they have not received almost any attention from image analysis community. Their formulation was usually much more general than ours. They tried to decompose general polygons into specific polygonal components (convex polygons, star-shape polygons, triangles,

generally oriented rectangles, etc.). A common feature of these methods was that they transformed the decomposition problem to a graph partitioning problem and employed tools known from graph theory. The only subgroup relevant to our purposes is a decomposition of a digital polygon into rectilinear rectangles. An algorithm which was proved to be optimal in terms of the number of blocks was independently proposed in the same form by three different authors [18–20] (in this order) and later discussed by [21,22] and others. The method (denoted here as FER) works for any object even if it contains holes. As will be discussed later, individual versions of the algorithm differ from one another only by the implementation of one step, which may influence the complexity but not the total number of the blocks.

The method performs hierarchically on two levels. On the first level, we detect all "concave" vertices (i.e. those having the inner angle 270°) of the input object and identify pairs of "cogrid" concave vertices (i.e. those having the same horizontal or vertical coordinates). Then we divide the object into subpolygons by constructing chords which connect certain cogrid concave vertices. It is proved in [20] and other papers that the optimal choice of the chord set is such that the chords do not intersect each other and their number is maximum possible.

The problem of optimal selection of the chords is equivalent to the problem of finding the maximal set of independent nodes in a graph, where each node corresponds to a chord and two nodes are connected with an edge, if the two chords have a common point (either a vertex or an intersection). Generally, this problem is NP-complete, but our graph is a bipartite one, since any two horizontal (vertical) edges cannot intersect one another. In a bipartite graph, this task can be efficiently resolved. First, we find a maximal matching, which is a classical problem in graph theory, whose algorithmic solution in a polynomial time has been published in various versions. Some of them are optimized with respect to the number of edges, the others with respect to the number of vertices (see [23–25,21] for some examples of particular algorithms). For binary object decomposition, it is impossible to choose one that would be time-optimal for any object, because the number of vertices and/or edges of the graph depends on the shape of the object. We implemented the algorithm by Edmonds and Karp [25] which is based on the Maximum Network Flow and is linear w.r.t. the number of vertices and quadratic w.r.t. the number of edges.

As soon as the maximal matching has been constructed, the maximal set of independent nodes can be found much faster than the maximal matching itself—roughly speaking, the maximal independent set contains one node of each matching pair plus all isolated nodes plus some other nodes, which are not included in the matching but still independent. As a result, we obtain a set of nodes that is unique in terms of the number but ambiguous in terms of particular nodes. However, this ambiguity does not play any role—although each set leads to different partitioning, the number of partitions is always the same. Hence, at the end of the first level, the object is decomposed into subpolygons, which do not contain any cogrid concave vertices (see Fig. 7).

The second level is very simple. Each subpolygon coming from the first level is further divided. From each its concave vertex, a single chord is constructed such that this chord terminates either on the boundary of the subpolygon or on the chord constructed earlier. This is a sequential process in which each concave vertex is visited only once. We may choose randomly between two possible chords offered in each concave vertex. After that, the subpolygon is divided into rectangles, because rectangle is the only polygon having no concave vertices (see Fig. 8).

The strength of this algorithm is in the fact that it guarantees minimizing the number of decomposing rectangles regardless of the particular choices on the both levels. On the other hand, one may expect slower performance than in the previous algorithms namely because of expensive finding the maximum set of independent graph nodes. Since the optimal partitioning is not unique on both levels, one could require additional constraints, such as

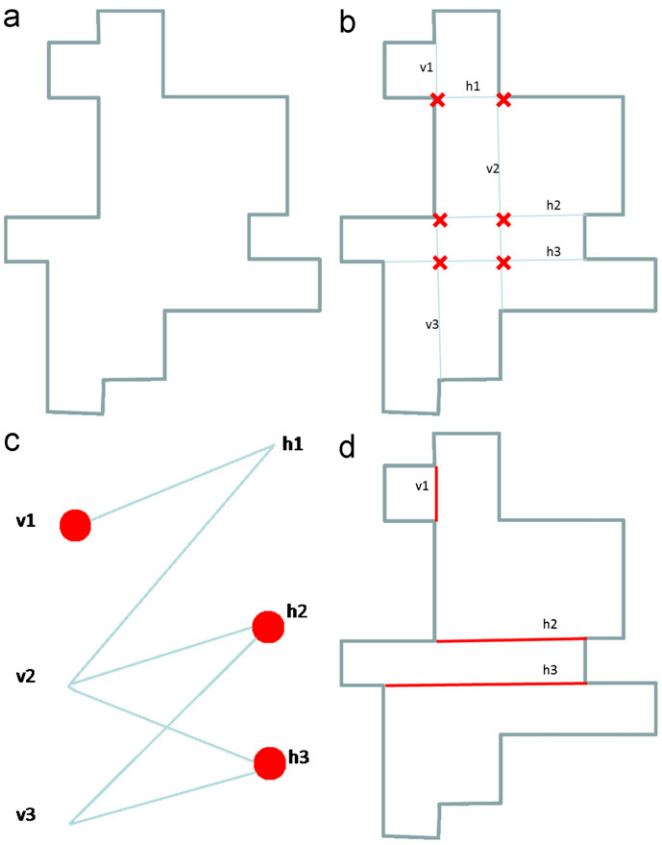

Fig. 7. (b) First-level decomposition of the object (a) by chords connecting the cogrid concave vertices. The crosses indicate the chord intersections. (c) The corresponding bipartite graph with a maximum independent set of three vertices and (d) the object decomposition.

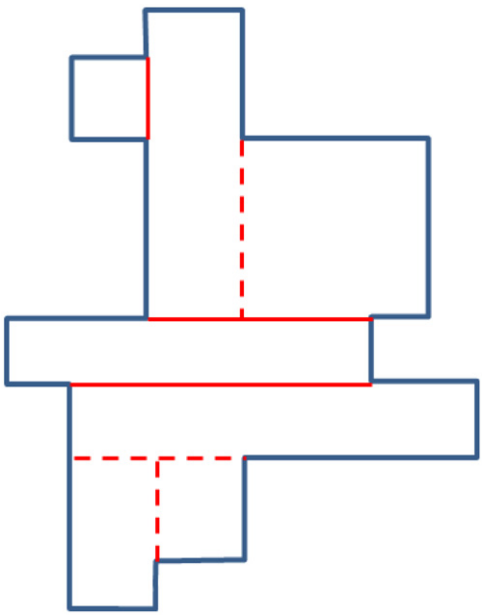

Fig. 8. Second-level decomposition of a subpolygon. From each concave vertex a single chord of arbitrary direction is constructed.

minimum length of inner boundary, but this would lead to further increasing of the complexity.

If the object does not have any cogrid concave vertices or if the maximum independent set of nodes (or, more precisely, at least one of such sets) contains only the nodes corresponding to the horizontal (or vertical) chords, then also all chords in the second level may be constructed in the same direction and, consequently, FER decomposition leads exactly to the same partitioning as GDM applied in that direction (see Fig. 9 for an example). This helps us not only to understand when GDM is strong, but also to interpret the idea behind FER algorithm—it may be viewed as a "local GDM" properly switching between the directions. An example of FER decomposition is in Fig. 10.

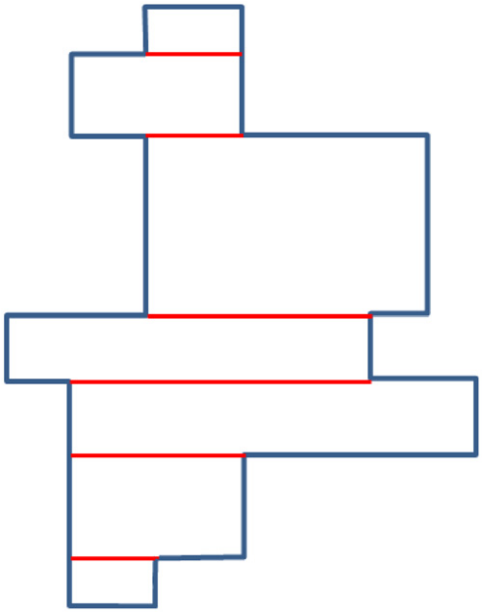

Fig. 9. An example where both FER and GDM yield the same decompositions.

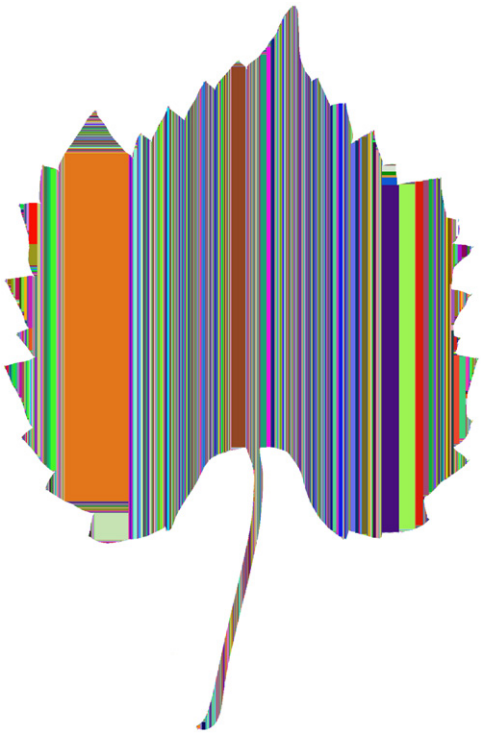

Fig. 10. FER decomposition of the leaf image (1748 blocks in total).

3. Experimental comparison—image compression

In this section, we compare the performance of the above decomposition methods in binary image compression. The experiments were performed on the publicly available LEAF database [26], which is a database of 795 scanned and binarized leaves of trees and shrubs of vegetation growing in the Czech Republic (see Fig. 11).[1] All methods were implemented in C++ language and run on a PC with Intel Core 2 Duo, 2.8 GHz CPU and Windows 7 Professional.

3.1. Efficiency of the decomposition

First we monitor and compare two parameters: the number of blocks and the corresponding decomposition time for GDM, QTD, DT and FER methods, respectively. The data presented in Table 1 are cumulative for all objects in the database. The time was always measured just of the decomposition itself, no input/output operations were included. The minimum number of blocks was achieved by FER, as one expects from the theory. This is of course on the expense of the time, but surprisingly the time is lower than that of DT and only five times higher than the time of QTD. The winner of this test is GDM method yielding only a slightly worse number of blocks than FER but in the by far shortest time. On the other hand, QTD produces the highest block number and the DT is the slowest.

It is interesting that the differences among these decomposition methods are observable even on very small and simple images. For example, the 8×8 object in Fig. 12a was decomposed by (DM, GDM, QTD, DT, and FER) into (10, 5, 23, 6, and 4) rectangular blocks, respectively.

3.2. Compression ratio

Although the decomposition itself makes a substantial compression, we further increase the compression ratios of all methods by a proper block ordering. We propose a new file format denoted as BLK. It uses three types of compression: the blocks from DT are grouped according their size, this allows to store the size only once per each group and then to store just upper left corner of each block. The long narrow blocks from GDM and FER are sorted by the coordinates in the "narrow" direction and sizes and coordinate differences in this direction are encoded in the decreased number of bits. QTD uses its traditional three-symbol encoding. The label of the actual decomposition method is stored in the file header along with other auxiliary parameters.

We compared compression ratios of BLK format (with all these four decomposition methods) to commercial formats. As we already explained, it does not reflect only the number of blocks, since other factors playing role there. We calculated average compression ratios (ACR) over the LEAF database. ACR is a ratio of the size of all files in the database in the specific format and the size without any compression. The results are in Table 2. In this experiment, it is not meaningful to measure the time, because it inherently includes I/O operations. Since we do not have an access to the source codes of commercial compressions, it would be impossible to ensure an objective time comparison.

[1] The database exists in two versions—original (the leaves with petioles, high resolution) and simplified (the leaves without petioles, downsampled by a factor 2). Table 1 refers to the simplified version, Table 2 to the original one.

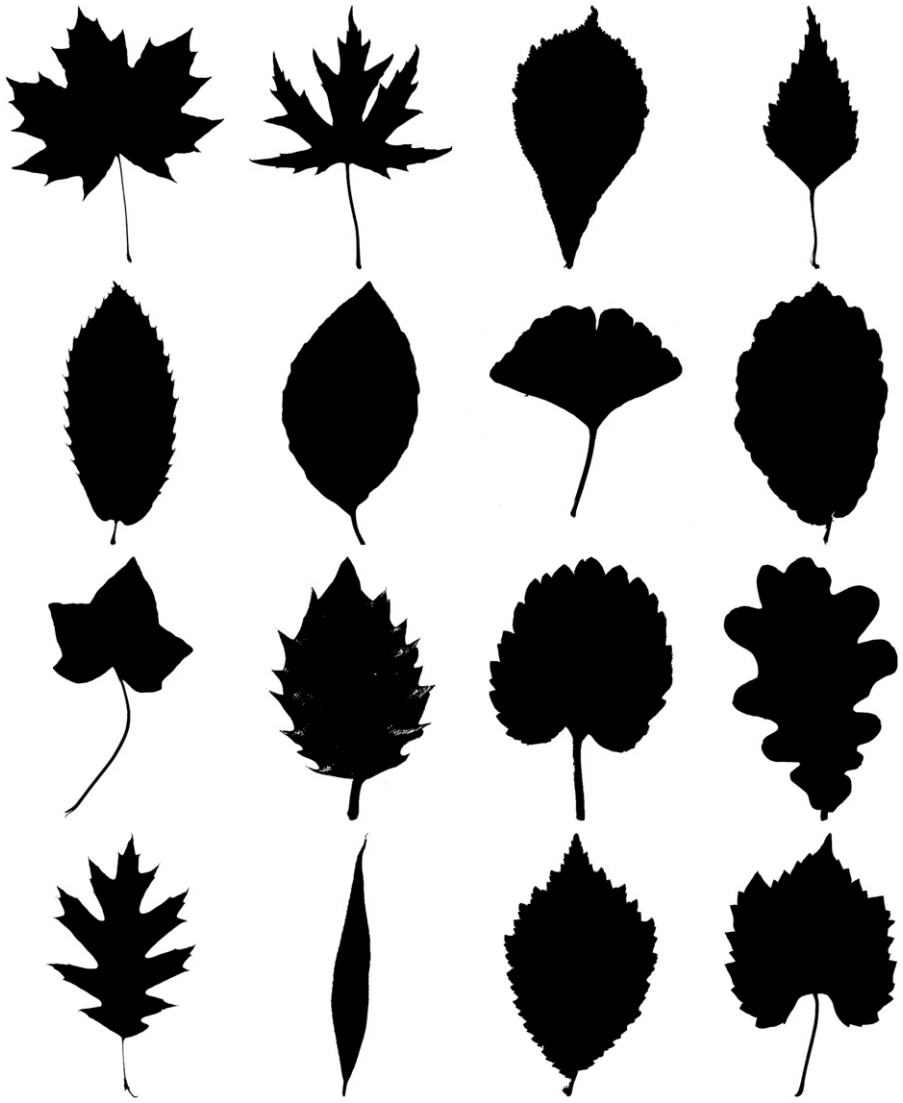

Fig. 11. Examples of the LEAF database (*Acer platanoides*, *Acer saccharinum*, *Aesculus hippocastanum*, *Betula pendula*, *Castanea sativa*, *Fagus sylvatica*, *Ginkgo biloba*, *Hamamelis virginiana*, *Hedera helix*, *Ilex aquifolium*, *Morus alba*, *Quercus robur*, *Quercus rubra*, *Salix alba*, *Ulmus glabra*, and *Vitis riparia*).

Table 1
The number of blocks and decomposition time achieved on the LEAF database.

Method	# blocks	# blocks (%)	Time (s)
GDM	419,489	112	1.3
QTD	1,913,275	511	7.2
DT	545,528	146	50.3
FER	374,149	100	37.5

The best performance of QTD was achieved namely because its three-symbol encoding in the BLK format is very efficient. This yields good compression even if the number of blocks is high. The ACR of FER and GDM is slightly worse, but still very good; FER method is better because it guarantees the minimum number of blocks. The result of DT is still good comparing to TIFF and was achieved by efficient encoding of one-pixel blocks. However, this method is very time-consuming. The only commercial format providing a comparable ACR is PNG with a deflate method, others provide worse compression ratios than all BLK methods.

Based on these two measurements a general recommendation can be as follows: QTD method yields good compression ratio because of efficient implementation in BLK format, but we should keep in mind that the number of blocks was four times higher and the decomposition time even six times higher than GDM offering a very good compromise between compression ratio and decomposition speed. FER and DT methods are suitable mainly for applications, where the compression is performed only once (usually off-line) and thus the time of the compression is not critical. FER provides minimum number of blocks and slightly better ACR and time of compression.

The results of the previous experiments are statistically significant and can be generalized. However, for specific shapes the results may be different. The same experiment was carried out on special images “Transmitter” and “Labyrinth” to illustrate the behavior of the algorithms in extreme cases. Transmitter is decomposed into very high number of blocks (7207 in an optimal case), while Labyrinth represents an opposite case with only 37 blocks (see Figs. 13 and 14 for their decompositions). The compression ratio of various methods is summarized in Table 3. While PNG deflate, DT, GDM and FER methods efficiently employs the rectangular structure of the Labyrinth, the other methods (namely QTD) are not able to do so.

3.3. Backward composition

The compression methods would be useless, if we would not be able to restore the original shape if requested. In many

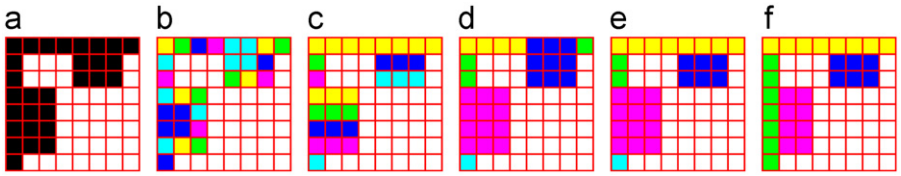

Fig. 12. Decomposition of a simple object: (a) original object—29 pixels, (b) QTD—23 blocks, (c) DM—10 blocks, (d) DT—6 blocks, (e) GDM—5 blocks, and (f) FER—4 blocks.

Table 2
Compression ratios on the LEAF database.

Format	Method	Size (byte)	ACR (%)
TIFF	No compression	172,886,448	100.00
TIFF	PackBits	13,282,998	7.68
TIFF	RLE	8,297,340	4.80
GIF	LZW	6,914,637	4.00
BLK	DT	5,173,371	2.99
PNG	Deflate	5,066,019	2.93
BLK	GDM	4,723,166	2.73
BLK	FER	4,603,942	2.66
BLK	QTD	3,012,532	1.74

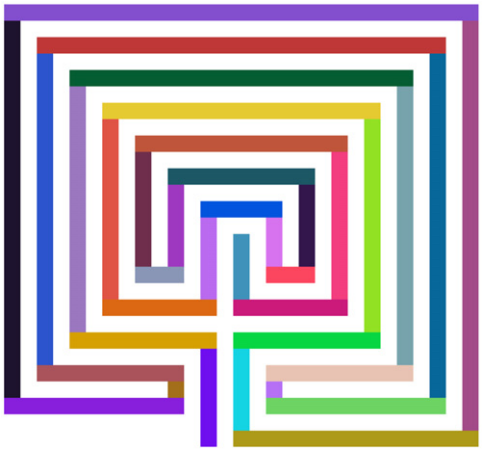

Fig. 13. Decomposition of the Labyrinth. DT, GDM, and FER produce the same number of 37 blocks.

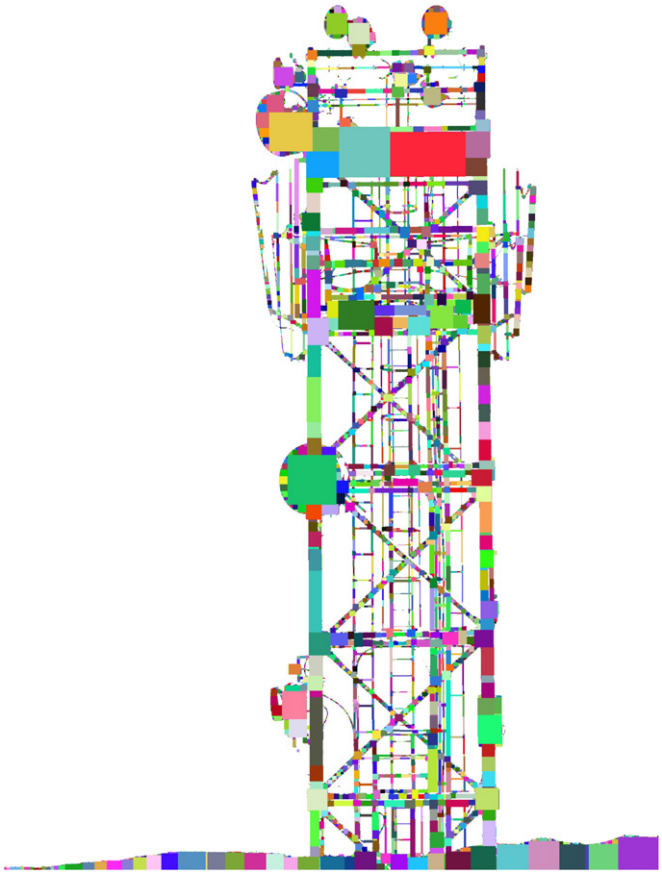

Fig. 14. Decomposition of the Transmitter.

applications the compressed images are stored in widely-accessed databases. While decomposition and compression is performed mostly off-line only once, composition (reconstruction) of the objects is required repeatedly and (almost) real time. Hence, three basic constraints on the reconstruction algorithm are no information loss, high speed and low memory consumption.

A naive reconstruction consists just of drawing all the rectangles to an allocated pixel array. This is accurate, simple to implement and relatively fast, but such method consumes a large amount of memory in an inefficient way. A much better way is to avoid drawing of the whole object and reconstruct the object boundary only. Since each binary object is fully determined by its boundary, this is still a lossless reconstruction. Such an algorithm handles computer memory more efficiently, because it does not require an array of the size of the original image.

We introduce an algorithm that requires only $\mathcal{O}(m+n)$ memory for an $m \times n$ image. Its time complexity varies with the shape of the image, i.e. with the number of the blocks K in the compressed version, but it performs very fast for "standard" objects. The algorithm works with images in BLK format regardless of the particular decomposition method used.

Table 3
Compression ratios of Transmitter and Labyrinth.

Format	Method	Transmitter (%)	Labyrinth (%)
TIFF	No comp.	100.00	100.00
TIFF	PackBits	24.01	49.65
TIFF	RLE	11.95	23.49
GIF	LZW	13.66	16.04
PNG	Deflate	6.29	1.43
BLK	DT	8.18	0.54
BLK	GDM	7.35	0.54
BLK	QTD	9.28	17.40
BLK	FER	6.66	0.54

The idea of the algorithm is to search the compressed BLK file and to maintain lists of border lines that we update whenever new rectangle is inserted. We start with the first rectangle of the compressed representation, so in the first step, the object boundary is formed by border lines of the first rectangle only. When processing a new rectangle, we need to update the current outline by inserting or removing lines or its parts. If the new

rectangle is adjacent to the current boundary, we insert only those parts of its border that are not adjacent to the current boundary. Moreover, in the current boundary, those lines or its parts that are adjacent to the new rectangle are removed (see Fig. 15).

To find adjacent border lines quickly, we maintain two arrays of lists, one for every horizontal coordinate and the other for every vertical one, see Fig. 16. Each element of the array contains pointer to the list of lines in the corresponding row or column. The borders of the new rectangle are inserted into the four corresponding lists and then the adjacent lists are searched.

First, we tested the correctness of the algorithm. We restored all leaves from BLK format to the standard matrix representation by the algorithm described above and compared them with the originals by matrix XOR. The cumulative error over the whole database was zero, which illustrates a perfect reconstruction. Also the reconstruction of complex objects is error free (see Fig. 17 for a reconstruction of a part of the Transmitter).

In the next experiment, we compared time complexity and memory consumption of the proposed decompression method with a "traditional" restoration, i.e. with putting all rectangles into an array and consequent boundary detection by convolution with $(-1,1)$ kernel. The results achieved on the LEAF dataset are in Table 4. The column "Lower 10%" means 10% of objects with the lowest number of blocks; analogously the column "Upper 10%". "Overall" means a mean value over a complete database. One can see that we achieved huge savings in memory consumption and that also certain speed-up of the reconstruction can be observed. Clearly, both these factors are more significant for objects with a low number of blocks.

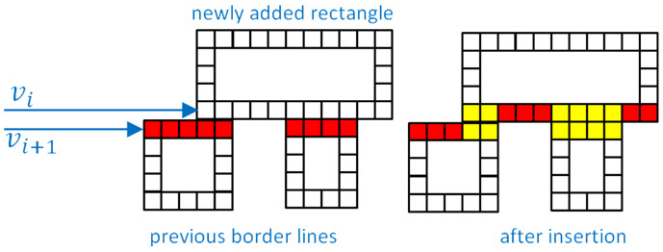

Fig. 15. Boundary list updating. Symbol v_i is the list of lines in the ith row.

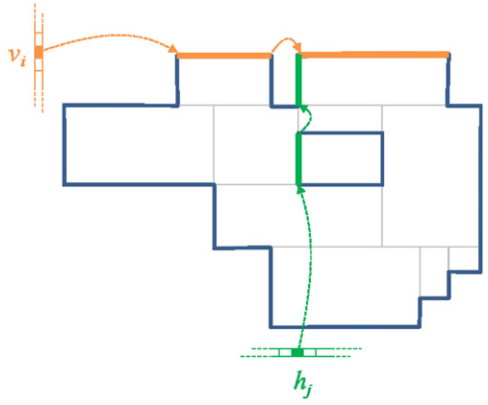

Fig. 16. Horizontal and vertical block boundaries. Symbols v_i and h_j are the list of lines in the ith row and jth column.

Fig. 17. Detail of the reconstruction of the Transmitter.

Table 4
Comparison of the proposed algorithm and the traditional reconstruction.

Criterion	Lower 10%	Upper 10%	Overall
Time (ms/image)			
Proposed	1.3	2.3	1.6
Traditional	4.0	4.0	4.0
Memory consumption (bytes/image)			
Proposed	4955	41,165	17,589
Traditional	1,048,576	1,048,576	1,048,576
Ratio (%)	0.5	3.9	1.7

4. Experimental comparison—moment calculation

Moments are scalar quantities that have been used to characterize an image and to capture its significant features. From the mathematical point of view, moments are "projections" of an image function onto a polynomial basis. Functions of moments, insensitive to certain group of transforms, are called *moment invariants* (see [27] for a survey). Moment invariants have become one of the most important and most frequently used tools for object description and recognition. Hence, efficient algorithms of their computation are of a high importance and have attracted much attention (see [27, Chapter 7] for an overview).

Geometric moment of a continuous image $f(x,y)$ is defined as

$$m_{pq}^{(f)} = \int_{-\infty}^{\infty}\int_{-\infty}^{\infty} x^p y^q f(x,y)\,\mathrm{d}x\,\mathrm{d}y, \tag{2}$$

where $p+q$ is the order of the moment. If the image $f(x,y)$ is a discrete one of the size $M\times N$, then we can estimate its moment as[2]

$$\overline{m}_{pq}^{(f)} = \sum_{i=0}^{M-1}\sum_{j=0}^{N-1} i^p j^q f_{ij}. \tag{3}$$

For a binary object, f is just its characteristic function.

Object decomposition can be employed in moment calculation in the following way. If we decompose an object B into disjoint blocks $B_1,B_2,\ldots,B_K$ such that $B=\bigcup_{k=1}^{K} B_k$, then, thanks to the linearity of moments,

$$m_{pq}^{(B)} = \sum_{k=1}^{K} m_{pq}^{(B_k)}. \tag{4}$$

Since we can calculate the moment of each rectangular block in $\mathcal{O}(1)$ time (either by symbolic integration of the kernel function or, in a discrete domain, by summation rules), the overall complexity of $m_{pq}^{(B)}$ is $\mathcal{O}(K)$. If $K \ll MN$ the speed-up may be significant. As we already have seen, simple decomposition algorithms produce relatively high number of blocks but perform fast, while more sophisticated decomposition methods end up with small number of blocks but require more time. The

[2] Another way is using higher-order approximation of the integral and/or exact integration of $x^p y^q$ over rectangular regions [6].

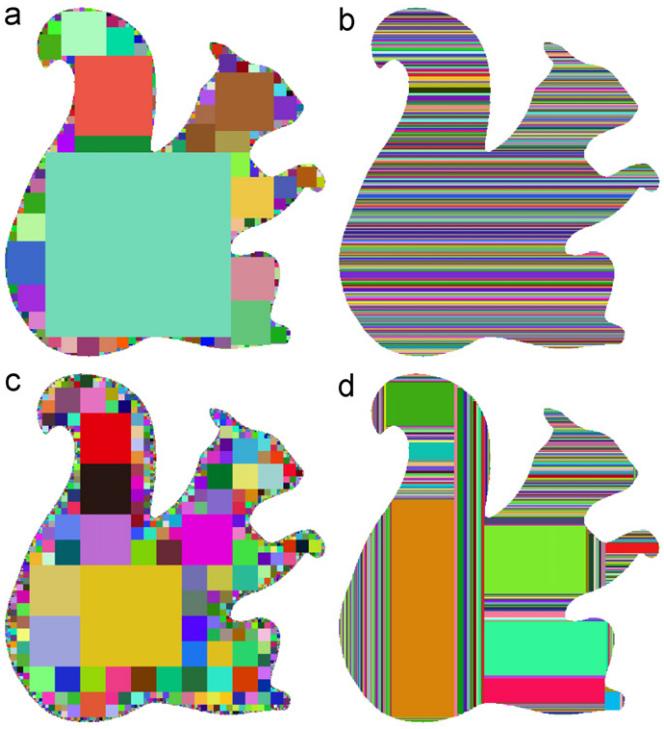

Fig. 18. Decomposition of the squirrel image: (a) DT—697 blocks, (b) GDM—497 blocks, (c) QTD—2513 blocks and (d) FER—476 blocks.

complexity of the decomposition must be always considered as a part of the whole algorithm. Even if the decomposition is performed only once and can be used for the calculation of all moments, the time needed for decomposing the image is often so long that it substantially influences the efficiency of the whole method.

We decomposed a 465×465 squirrel silhouette by means of DT, GDM, QTD and FER decompositions (see Fig. 18a–d) which lead to 697, 497, 2513, and 476 rectangular blocks and took 32, 1, 4, and 19 ms, respectively. Then we calculated the moments of the image up to the order 464. To prevent floating point overflow for high orders, we used discrete Chebyshev polynomials $\tau_p(x)$ instead of x^p in the kernel functions. The particular choice of the polynomial basis does influence neither the decomposition algorithms themselves nor their mutual comparison. The moments of the individual rectangles were calculated using summation rules of the kernel functions independently of the size of the rectangle.

The computation times are on the graph in Fig. 19, where the horizontal axis shows the number of moments calculated. The initial time $t(0)$ shows the expense of the decomposition. It can be better seen in more detailed Fig. 20.

The best results were achieved for GDM because of its fast initial decomposition time and only slightly sub-optimal number of blocks. The theoretically optimal FER method requires so much initial time that it produces best results only if we calculate about 45,000 and more moments, which is not realistic. QTD is the second fastest at the beginning, but the time grows quickly, as the number of moments increases such that it becomes the worst one, if 3000 and more moments are calculated. DT requires the longest initialization and then its time complexity grows faster than that of FER and GDM. It gave the worst results among the four tested methods in this experiment. All decomposition methods outperformed the calculation from the definition even for very low number of moments.

The above results can be generalized for most "reasonable simple" shapes. However, it is easy to find a counterexample.

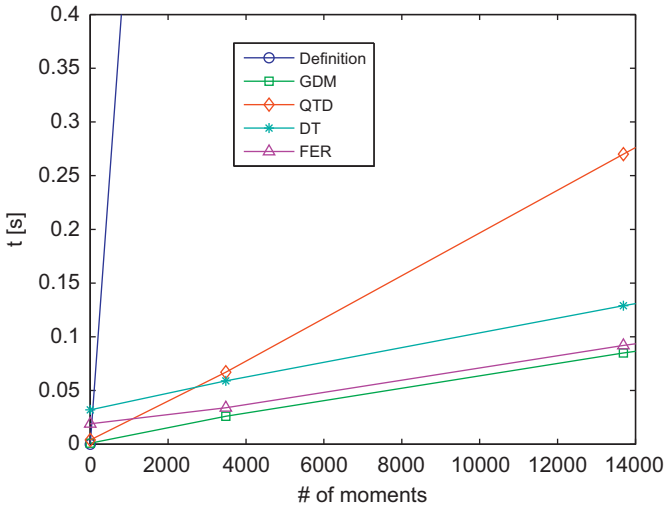

Fig. 19. The time complexity of the moments of the squirrel image.

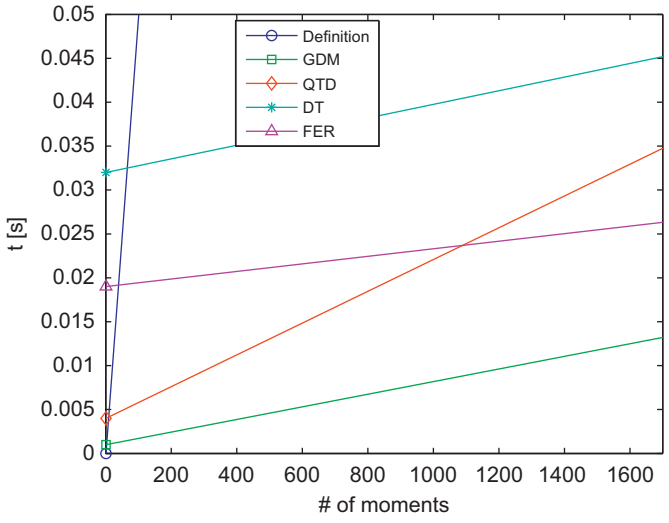

Fig. 20. A detail of Fig. 19 showing the complexity of lower-order moments.

If we repeat this experiment for a chessboard image, the results change dramatically, see Fig. 21. The chessboard image is the worst possible case, because it cannot be decomposed efficiently. The decomposition is just a waste of time and a direct calculation from the definition exhibits the best performance.

5. Experimental comparison—convolution

In this experiment, we show that the decomposition can be a powerful tool for speeding-up convolution filtering of an image with a binary kernel. While traditional "by definition" discrete convolution of an $M \times N$ image with an arbitrary $m \times n$ mask requires $\mathcal{O}(mnMN)$ operations and convolution via fast Fourier transform (FFT) $\mathcal{O}(MN \log MN)$ operations, convolution with a constant-valued rectangle takes only $\mathcal{O}(MN)$ operations regardless of the mask size. There are several ways how to implement such algorithm. The best one is probably to employ the precalculations of row-wise and consequently column-wise sums of the image. The convolution in a certain position is then calculated just from four values, which corresponds to the mask corners (see Fig. 22).

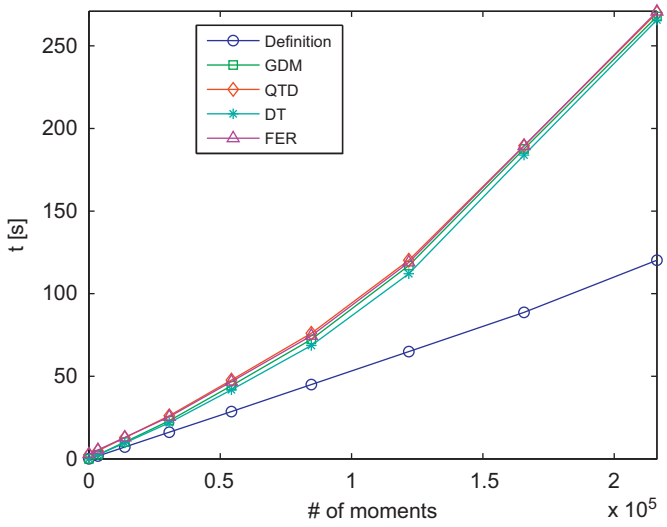

Fig. 21. The time complexity of the moments of the chessboard image.

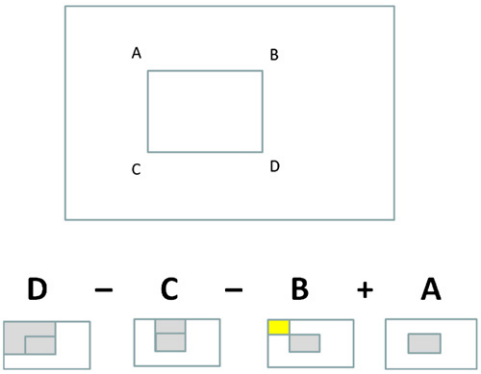

Fig. 22. Convolution of an image with a constant rectangular mask ABCD. The large rectangle contains in point X row-wise and column-wise sum $S(X)$ of the original image from (0, 0) to X. Then the convolution in the depicted position is given as $S(D)-S(C)-S(B)+S(A)$.

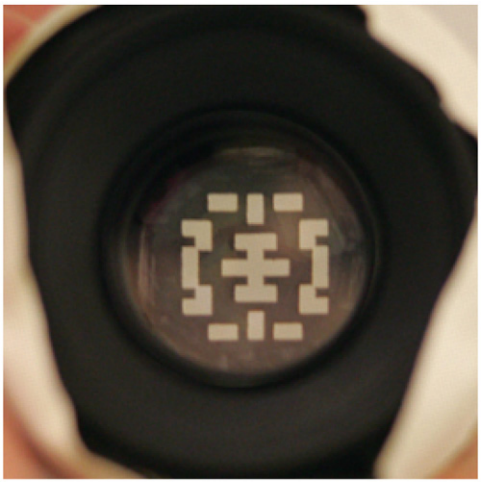

Fig. 23. An example of the coded aperture mask (courtesy of [29]).

Now, let us imagine a binary kernel, where the non-zero values form a more complex set than a single rectangle. Such a situation typically appears for instance in *coded aperture* (CA) imaging. CA is a method of recovering the depth map of the scene from a single image [28]. The trick is to insert a special occluder within the aperture of the camera lens to create a coded aperture (see Fig. 23 for an example). The CA output must be deconvolved, which incorporates (if an iterative deconvolution method such as Richardson–Lucy or similar is applied) repeating the calculations of convolution of an estimated image with the aperture mask. If the mask has a full or close-to-full rank, we cannot effectively use any factorization, which seemingly takes us back to the calculation from the definition or via FFT. However, if we decompose the mask into rectangles $B_1,\ldots,B_K$, we can, thanks to the linearity of convolution, calculate the convolutions with each B_j separately by the method described above and then just sum up the results. (This method can be used even if the kernel is not binary and thus the blocks have different values but that is a rare case in practice.) In that way, we achieve the overall complexity of $\mathcal{O}(KMN)$. In this experiment, we demonstrate that for $K \ll mn$ the speed up is really huge comparing to the direct calculation from the definition and still significant comparing to the FFT.

We filtered a 3456×2592 image with two binary kernels of the same shape but different size—the small one of (35×38) and the large one of (141×152) pixels. The small kernel is scaled version of the large one. The kernels were decomposed by the FER method into 10 rectangles (see Fig. 24 for the kernel and its decomposition). We tested three methods—direct convolution in the image domain from the definition, convolution via FFT in the frequency domain (we used popular public-domain FFTPack software [30,31]), and fast convolution using kernel decomposition as described above. In the last case, the matrix of the partial sums of the image was precomputed and then the (cyclic) convolution was calculated as explained in Fig. 22. We wanted to measure the time of each individual step separately, because in practice, either the mask decomposition or the precomputing of partial sums uses to be done only once and hence its complexity is negligible (in batch processing either the mask or the image stays the same while the other factor varies). However, the mask decomposition was so fast that the corresponding time was not measurable.

The smoothed image is always the same regardless of the method used. The time comparison is summarized in Table 5. As expected, the slowest calculation is from the definition in the image domain. Even for the small mask, it is many times slower than the methods via FFT and via mask decomposition. Note that the times for both FFT and decomposition methods actually do not depend on the mask size, which is clear from the theory. Although FFTPack is a very powerful implementation, the decomposition method was still able to perform four times faster. Precomputing of the partial sums took only 0.1 s of the total time.

Fig. 24. The convolution kernel used in the experiment (left) and its FER decomposition into 10 blocks (right).

Table 5
Time comparison of various convolution methods (the time in s).

Mask size	Definition	FFT	Decomposition
35×38	26	4.3	0.96
141×152	411	4.3	0.96

This experiment illustrates that the convolution with a binary mask can be implemented by means of mask decomposition in a very efficient way. Two main factors influence whether or not the convolution via decomposition is faster than via FFT—the image size (not the mask size!) and the number of blocks of the mask. In our implementation and hardware and for 8–10 Mpix images the threshold value is about 40. If the number of blocks is lower, the decomposition-based convolution is the best choice.

6. Conclusion

We presented an overview of methods which decompose an arbitrary binary object into rectilinear rectangles, starting from very simple one up to the optimal graph-based decomposition. We tested their performance in three frequent tasks—image compression, moment computation and linear filtering. We showed that there is no "generally best" method; the choice must reflect our requirements and is always a compromise between complexity on one hand and time and memory consumption on the other hand. The weights given to these two factors are user-defined parameters. This paper should help the users to select proper decomposition method according to their preferences. In our opinion, GDM is the most appropriate in common situations, while FER is recommended, if we want to achieve as few blocks as possible on the expense of higher complexity. The other two tested methods either produce too many blocks (QTD) or perform slowly (DT). They may find applications in specific tasks only.

An interesting extension in the future could be a usage of overlapping blocks (we speak about *covering* instead of partitioning). This may significantly decrease the block number, however, on the expense of the NP-hard complexity.

Although we have been talking just about binary objects in the paper, all methods can be theoretically used for graylevel and color images as well. Graylevel image can be expressed as a union of disjoint binary images, which can be obtained either as intensity slices [32] or bit planes [33]. However, these "images" use to be highly fragmented (especially low bit planes resemble a "random chessboard") and decomposition methods do not perform well. Our experiments indicate that for graylevel/color images the decomposition algorithms of this kind have only little practical importance.

Acknowledgment

This work has been supported by the Grant no. P103/11/1552 of the Czech Science Foundation.

References

[1] J.M. Keil, Polygon decomposition, in: Handbook of Computational Geometry, Elsevier, 2000, pp. 491–518.
[2] M.F. Zakaria, L.J. Vroomen, P. Zsombor-Murray, J.M. van Kessel, Fast algorithm for the computation of moment invariants, Pattern Recognition 20 (6) (1987) 639–643.
[3] M. Dai, P. Baylou, M. Najim, An efficient algorithm for computation of shape moments from run-length codes or chain codes, Pattern Recognition 25 (10) (1992) 1119–1128.
[4] B.C. Li, A new computation of geometric moments, Pattern Recognition 26 (1) (1993) 109–113.
[5] I.M. Spiliotis, B.G. Mertzios, Real-time computation of two-dimensional moments on binary images using image block representation, IEEE Transactions on Image Processing 7 (11) (1998) 1609–1615.
[6] J. Flusser, Refined moment calculation using image block representation, IEEE Transactions on Image Processing 9 (11) (2000) 1977–1978.
[7] C.-H. Wu, S.-J. Horng, P.-Z. Lee, A new computation of shape moments via quadtree decomposition, Pattern Recognition 34 (7) (2001) 1319–1330.
[8] J.H. Sossa-Azuela, C. Yáñez-Márquez, J.L. Díaz de León Santiago, Computing geometric moments using morphological erosions, Pattern Recognition 34 (2) (2001) 271–276.
[9] T. Suk, J. Flusser, Refined morphological methods of moment computation, in: 20th International Conference on Pattern Recognition ICPR'10, IEEE Computer Society, 2010, pp. 966–970.
[10] W. Liou, J. Tan, R. Lee, Minimum partitioning simple rectilinear polygons in O(n log log n)-time, in: Proceeding of the 5th Annual ACM Symposium on Computational Geometry SoCG'89, ACM, New York, NY, USA, 1989, pp. 344–353.
[11] K.D. Gourley, D.M. Green, A polygon-to-rectangle conversion algorithm, IEEE Computer Graphics and Applications 3 (1) (1983) 31–36.
[12] E. Kawaguchi, T. Endo, On a method of binary-picture representation and its application to data compression, IEEE Transactions on Pattern Analysis and Machine Intelligence 2 (1) (1980) 27–35.
[13] J. Flusser, An adaptive method for image registration, Pattern Recognition 25 (1) (1992) 45–54.
[14] D.N. Vizireanu, Generalizations of binary morphological shape decomposition, Journal of Electronic Imaging 16 (1) (2007) 013002-1-6.
[15] G. Borgefors, Distance transformations in digital images, Computer Vision, Graphics, and Image Processing 34 (3) (1986) 344–371.
[16] M. Seaidoun, A Fast Exact Euclidean Distance Transform with Application to Computer Vision and Digital Image Processing, Ph.D. Thesis, Northeastern University, Boston, USA, Advisor John Gauch, September, 1993.
[17] T.E. Schouten, E.L. van den Broek, Incremental distance transforms (IDT), in: 20th International Conference on Pattern Recognition ICPR'10, IEEE Computer Society, 2010, pp. 237–240.
[18] W. Lipski Jr., E. Lodi, F. Luccio, C. Mugnai, L. Pagli, On two-dimensional data organization II, in: Fundamenta Informaticae, Annales Societatis Mathematicae Polonae, Series IV, vol. II, 1979, pp. 245–260.
[19] T. Ohtsuki, Minimum dissection of rectilinear regions, in: Proceedings of the IEEE International Conference on Circuits and Systems ISCAS'82, IEEE, 1982, pp. 1210–1213.
[20] L. Ferrari, P.V. Sankar, J. Sklansky, Minimal rectangular partitions of digitized blobs, Computer Vision, Graphics, and Image Processing 28 (1) (1984) 58–71.
[21] H. Imai, T. Asano, Efficient algorithms for geometric graph search problems, SIAM Journal on Computing 15 (2) (1986) 478–494.
[22] D. Eppstein, Graph-theoretic solutions to computational geometry problems, in: 35th International Workshop on Graph-Theoretic Concepts in Computer Science WG'09, Lecture Notes in Computer Science, vol. 5911, Springer, 2009, pp. 1–16.
[23] A.V. Goldberg, S. Rao, Beyond the flow decomposition barrier, Journal of the Association for Computing Machinery 45 (5) (1998) 783–797.
[24] A.V. Goldberg, R.E. Tarjan, A new approach to the maximum-flow problem, Journal of the Association for Computing Machinery 35 (4) (1988) 921–940.
[25] J. Edmonds, R.M. Karp, Theoretical improvements in algorithmic efficiency for network flow problems, Journal of the Association for Computing Machinery 19 (2) (1972) 248–264.
[26] Department of Image Processing, Tree LEAF database, URL ⟨http://zoi.utia.cas.cz/tree_leaves⟩.
[27] J. Flusser, T. Suk, B. Zitová, Moments and Moment Invariants in Pattern Recognition, Wiley, Chichester, 2009.
[28] A. Levin, R. Fergus, F. Durand, W.T. Freeman, Image and depth from a conventional camera with a coded aperture, in: Special Interest Group on Computer Graphics and Interactive Techniques Conference SIGGRAPH'07, ACM, New York, NY, USA, 2007.
[29] A. Levin, R. Fergus, F. Durand, B. Freeman, Image and depth from a conventional camera with a coded aperture, URL ⟨http://groups.csail.mit.edu/graphics/CodedAperture⟩.
[30] P.N. Swarztrauber, FFTPACK, URL ⟨http://www.netlib.org/fftpack/⟩.
[31] A. Fernandes, FFTPACK translated to pure iso C/C++, URL ⟨http://www.fernandes.org/txp/article/4/fftpack-translated-to-pure-iso-cc⟩.
[32] G.A. Papakostas, E.G. Karakasis, D.E. Koulouriotis, Efficient and accurate computation of geometric moments on gray-scale images, Pattern Recognition 41 (6) (2008) 1895–1904.
[33] I.M. Spiliotis, Y.S. Boutalis, Parameterized real-time moment computation on gray images using block techniques, Journal of Real-Time Image Processing (2011) 81–91.

Tomáš Suk received the MSc degree in electrical engineering from the Czech Technical University, Faculty of Electrical Engineering, Prague, 1987. The CSc degree (corresponds to PhD) in computer science from the Czechoslovak Academy of Sciences, Prague, 1992. From 1991 he is a researcher with the Institute of Information Theory and Automation, Academy of Sciences of the Czech Republic, Prague, a member of Department of Image Processing. In 2002 he received the Otto Wichterle's premium of the Academy of Sciences of the Czech Republic for young scientists.

His research interests include digital image processing, pattern recognition, image filtering, invariant features, moment and point invariants, geometric transformations of images. Applications in botany, remote sensing, astronomy, medicine and computer vision. He has published 18 papers in international journals and more than 50 conference papers, mostly from international conferences. He is a coauthor of the monograph "Moments and Moment Invariants in Pattern Recognition" (Wiley, 2009).

Cyril Höschl IV received the MSc degree in computer science from the Charles University, Faculty of Mathematics and Physics, Prague, 2010. Now he is a PhD student in the Institute of Information Theory and Automation, Academy of Sciences of the Czech Republic, Prague. His supervisor is Jan Flusser.

His research interests include digital image processing. Among this area he is author of algorithms and software for visualization of relations known as Sociomapping.

Jan Flusser received the MSc degree in mathematical engineering from the Czech Technical University, Prague, Czech Republic in 1985, the PhD degree in computer science from the Czechoslovak Academy of Sciences in 1990, and the DSc degree in technical cybernetics in 2001. Since 1985 he has been with the Institute of Information Theory and Automation, Academy of Sciences of the Czech Republic, Prague. In 1995–2007, he was holding the position of a head of Department of Image Processing. Currently (since 2007) he is a director of the Institute. He is a full professor of computer science at the Czech Technical University and at the Charles University, Prague, Czech Republic, where he gives undergraduate and graduate courses on digital image processing, pattern recognition, and moment invariants and wavelets.

His research interest covers moments and moment invariants, image registration, image fusion, multichannel blind deconvolution, and super-resolution imaging. He has authored and coauthored more than 150 research publications in these areas, including the monograph "Moments and Moment Invariants in Pattern Recognition" (Wiley, 2009), tutorials and invited/keynote talks at major international conferences.

He is a senior member of the IEEE.