Rectangular Decomposition of Binary Images*

Tomáš Suk, Cyril Höschl IV, and Jan Flusser

Institute of Information Theory and Automation of the ASCR,
Pod vodárenskou věží 4, 182 08 Praha 8, Czech Republic
{suk,hoschl,flusser}@utia.cas.cz
http://zoi.utia.cas.cz/

Abstract. The contribution deals with the most important methods for decomposition of binary images into union of rectangles. The overview includes run-length encoding and its generalization, decompositions based on quadtrees, on the distance transformation, and a theoretically optimal decomposition based on maximal matching in bipartite graphs. We experimentally test their performance in binary image compression and in convolution calculation and compare their computation times and success rates.

Keywords: Binary image decomposition, generalized delta-method, distance transformation, quadtree, bipartite graph, image compression, fast convolution.

1 Introduction

Binary images can be represented in a more efficient way, than just as a full-sized matrix consisting of zeros and ones. The terms "good representation" and "optimal representation" cannot be generally defined and are always dependent on what we are going to do with the image. The methods of binary image representation can be divided into two groups referred as *decomposition methods* and *boundary-based methods* (which we do not discuss in this paper).

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 [10]), all decomposition methods which perform in a discrete domain use only rectilinear rectangular or square blocks because of a native rectangular structure of the discrete image domain.

* This work has been supported by the grant No. P103/11/1552 of the Czech Science Foundation.

J. Blanc-Talon et al. (Eds.): ACIVS 2012, LNCS 7517, pp. 213–224, 2012.

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, claiming 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 perform fast, while more sophisticated decomposition methods end up with the small number of blocks but require more time. Even if the decomposition is in most tasks performed only once per object 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.

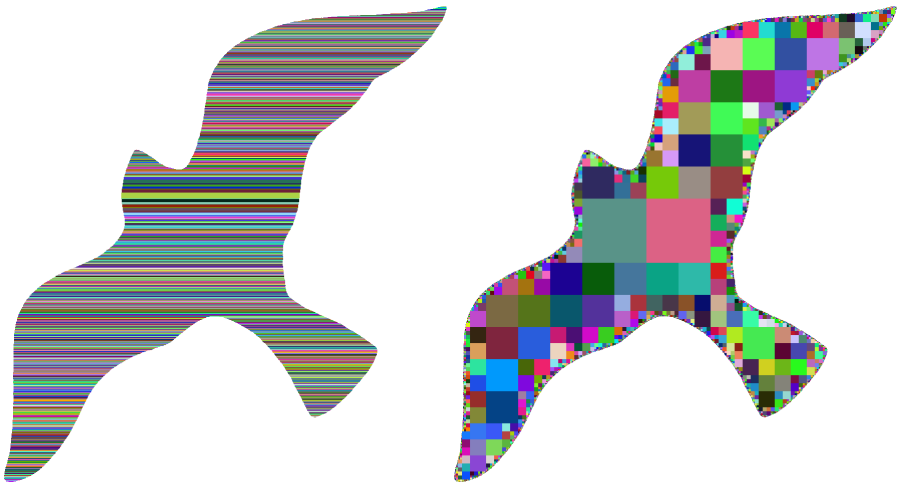
Image rectangular decomposition has found numerous straightforward applications in image compression methods and formats (RLE, TIFF, BMP and others), in calculation of image features (mainly moments and moment invariants) [7] in fast convolution algorithms with binary kernels, and in VLSI design.

The aim of this paper is to review existing decomposition methods, to present one which is new in this context and which is theoretically optimal in terms of the number of blocks, and to compare their performance in loss-less image compression and in fast convolution calculation.

2 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 are 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 [18] used the same representation for fast computation of image moments of convex shapes and called it "Delta-method" (DM). It 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 [17]. 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. 1a). For each rectangle, the coordinates of its upper-left corner, the length and the width are stored. The GDM is only slightly slower than DM while producing (sometimes significantly) less number of blocks. Surprisingly, under our knowledge this method has not been implemented in any commercial image format.



(a) (b)

Fig. 1. Decomposition of the bird image. (a) Generalized Delta-method (GDM – 983 blocks in total), (b) Quadtree decomposition (QTD – 4489 blocks in total).

3 Quadtree Decomposition

Quadtree decomposition (QTD) is a popular hierarchical decomposition scheme used in (but not limited to) image representation and compression [9]. The QTD works with square images of a size of a power of two; if it is not the case, the image can easily be 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. 1b.

The algorithm always yields square blocks that may be advantageous for some purpose but usually leads to a higher number of blocks than necessary. 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.

4 Decomposition to the Largest Inscribed Blocks

This group of methods can be described as "the largest first". They search the largest block that can be inscribed into the object, remove it and repeat searching

in the rest of the object until the entire object disappears and the decomposition is complete. They differ from each other by the way, how to search the largest block and also by the type of blocks (squares or rectangles, even or odd sizes).

4.1 Morphological Decomposition

Sossa et al [16] published a decomposition algorithm based on a *morphological erosion*. The erosion is an operation, where a small structural element (here a 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. They 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. So, they have two nested loops: the inner loop over the erosions for searching one block and the outer loop over the blocks for whole decomposition. If we consider that erosion is a relatively time consuming operation, it suggests this method performs much slower comparing to the previous decomposition algorithms. Although the original method [16] considers decomposition into squares only, it can be generalized also to rectangles which decreases the number of the resulting blocks.

4.2 Distance Transformation Decomposition

We proposed to speed up the previous algorithm by means of the *distance transformation* (DT), which we use for finding of the centers of the inscribed rectangles. 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. The 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 [1].

DT strongly depends on the metric used for the distance measurement. Seaidoun [15] proposed an algorithm for the Euclidean metric, we used a simplified version for the chessboard metric

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

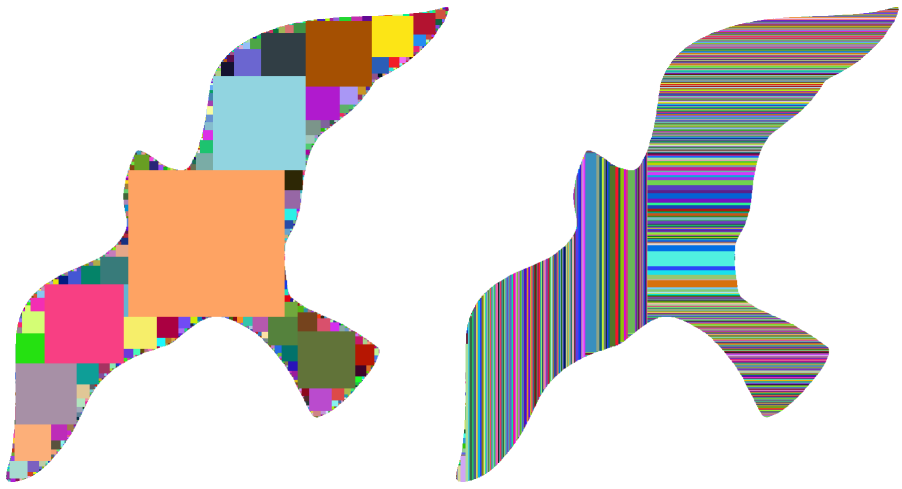
The metric is closely related to the form of the blocks, the chessboard metric leads to the decomposition to rectilinear squares, the Euclidean metric to circular blocks and a city block metric to the squares rotated by 45°.

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 [14]. 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.

The potential center(s) of the largest inscribed blocks are the pixels with the maximum values of DT. If the maximum s is unique, then a square $(2s-1) \times (2s-1)$ is inscribed. However, 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 $2s \times 2s$ 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. 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 bird image is shown in Fig. 2a.

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 which 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, see the experiments.



(a) (b)

Fig. 2. (a) Distance transformation decomposition (DTD) of the bird image (1306 blocks in total), (b) GBD decomposition (923 blocks in total)

5 Graph-Based Decomposition

A large group of decomposition algorithms appeared in 80's in computational geometry [10]. 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 known tools from graph theory. The only subgroup relevant to our purposes is a decomposition of a digital polygon into rectilinear rectangles. An algorithm that was proved to be optimal in terms of the number of blocks was independently proposed in the same form by three different authors [12], [13], [6] (in this order) and later discussed by [8], [4] and others. The method (denoted here as graph-based decomposition – GBD) works for any object even if it contains holes.

The vertices of a binary object can be divided into two groups, those having the inner angle 90° (we can call them "convex") and those having the inner angle 270° (we can call them "concave"). The method performs hierarchically on two levels. On the first level, we detect all concave vertices 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 [6] 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). Since any two horizontal (vertical) edges cannot intersect one another, our graph is a bipartite one which implies a solution in a polynomial time. First, we find a maximal matching (classical problem in graph theory which looks for labeling the maximum number of edges such that each node is incident to at most one labeled edge). It is a classical problem in graph theory, whose algorithmic solution in polynomial time has been published in various versions (probably the most popular approach is by Maximum Network Flow [3] which we also used in our implementation).

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 lead 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 (when considered individually) do not contain any cogrid concave vertices (see Fig. 3d).

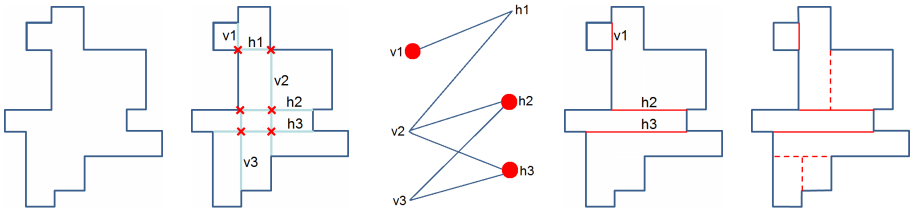


Fig. 3. (a) The object to be decomposed. (b) All possible chords connecting the cogrid concave vertices. The crosses indicate the chord intersections. (c) The corresponding bipartite graph with a maximum independent set of three nodes. (d) The first-level decomposition of the object. (e) The second-level decomposition which adds the decomposition of subpolygons. From each concave vertex a single chord of arbitrary direction is constructed.

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. Between two possible chords offered in each concave vertex we may choose randomly. After that, the subpolygon is divided into rectangles because rectangle is the only polygon having no concave vertices (see Fig. 3e).

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.

If the object does not have any cogrid concave vertices or if one of the maximum independent sets of nodes 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, the GBD decomposition leads exactly to the same partitioning as the GDM applied in that direction. It corresponds with the surprisingly good results of the GDM. An example of the GBD decomposition is in Fig. 2b.

6 Experimental Comparison of 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 [2] which is a database of 795 scanned and binarized leaves of trees and shrubs of vegetation growing in the Czech Republic (see

Fig. 4)[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.

Fig. 4. Examples of the LEAF database (acer platanoides, aesculus hippocastanum, betula pendula, fagus sylvatica, hedera helix, quercus robur, salix alba, ulmus glabra)

First, we monitor and compare two parameters: the number of blocks and the corresponding decomposition time for the GDM, QTD, DTD and GBD 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 GBD, as was expected. This is of course on the expense of the time but surprisingly the time is lower than that of DTD and only five times higher than the time of QTD. The winner of this test is the GDM method yielding only a slightly worse number of blocks than GBD but in the by far shortest time. On the other hand, QTD produce the highest block number and the DTD is the slowest.

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

Method	Number of blocks	Number of blocks [%]	Time [s]
GDM	419489	112	1.3
QTD	1913275	511	7.2
DTD	545528	146	50.3
GBD	374149	100	37.5

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 DTD 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 GBD 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

[1] The database exists in two versions – original (the leaves with petioles, high resolution) and simplified (the leaves without petioles, twice downsampled). Tab. 1 refers to the simplified version, Tab. 2 to the original one.

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 source codes of commercial compressions, it would be impossible to ensure an objective time comparison.

Table 2. Compression ratios on the LEAF database

Format	Method	size [byte]	ACR [%]
TIFF	no compression	172886448	100.00
TIFF	PackBits	13282998	7.68
TIFF	RLE	8297340	4.80
GIF	LZW	6914637	4.00
BLK	DTD	5173371	2.99
PNG	deflate	5066019	2.93
BLK	GDM	4723166	2.73
BLK	GBD	4603942	2.66
BLK	QTD	3012532	1.74

Surprisingly the best performance of QTD (comparing to the previous experiment) was achieved namely because the three-symbol encoding in the BLK format is very efficient. The ACR of GBD and GDM is slightly worse, but still very good; the GBD method is better because it guarantees the minimum number of blocks. The good result of DTD was achieved by efficient encoding of one-pixel blocks, but in the longest time. The only commercial format with 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: the 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. The GBD and DTD 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. GBD provides minimum number of blocks and slightly better ACR and time of compression.

7 Experimental Comparison of Computing 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 transformation (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 precalculations of row-wise and consequently column-wise sums of the image. Let us denote $S(X)$ the row-wise and column-wise sum of the original image from (0,0) to pixel X, then the convolution with a rectangular mask $ABCD$ is given as $S(D) - S(C) - S(B) + S(A)$.

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 [11]. The trick is to insert a special occluder within the aperture of the camera lens to create a coded aperture. The CA output must be deconvolved, which incorporates (if an iterative deconvolution method such as Richardson-Lucy or similar is applied) repeating 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 that seemingly takes us back to the calculation from the definition or via FFT. However, if we decompose the mask into rectangles $B_1, \cdots, B_K$, 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 $K << mn$ 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 a downsampled version of the large one (see Fig. 5a). The kernels were decomposed by the GBD method into 10 rectangles (see Fig. 5b).

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 [5]), 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.

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).

(a) (b)

Fig. 5. (a) The binary convolution kernel used in the experiment. (b) Its 10 blocks of GBD decomposition.

However, the mask decomposition was so fast that the corresponding time was not measurable.

As expected, the slowest calculation is from the definition in the image domain – 26 seconds for the small and 411 seconds for the large kernel, respectively. The speed of the calculation via FFT is independent of the kernel size – 4.3 seconds in both cases. Also the time of the decomposition method actually does not depend on the mask size. It took only 0.96 seconds including precomputing of the partial sums (0.1 s) and the mask decomposition (negligible time).

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 – we expect masks much smaller than the images) 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 blocks. If the number of blocks is lower, the decomposition-based convolution is the best choice.

8 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 two frequent tasks – image compression 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 GBD 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 (DTD). They may find applications in specific tasks only. The decomposition methods can be used for speed-up of various other computations, e.g. computing features for pattern recognition as moments etc.

References

1. Borgefors, G.: Distance transformations in digital images. Computer Vision, Graphics, and Image Processing 34(3), 344–371 (1986)
2. Department of Image Processing: Tree leaf database, http://zoi.utia.cas.cz/tree_leaves
3. Edmonds, J., Karp, R.M.: Theoretical improvements in algorithmic efficiency for network flow problems. Journal of the Association for Computing Machinery 19(2), 248–264 (1972)
4. Eppstein, D.: Graph-Theoretic Solutions to Computational Geometry Problems. In: Paul, C., Habib, M. (eds.) WG 2009. LNCS, vol. 5911, pp. 1–16. Springer, Heidelberg (2010)
5. Fernandes, A.: FFTPACK translated to pure iso c/c++, http://www.fernandes.org/txp/article/4fftpack-translated-to-pure-iso-cc
6. Ferrari, L., Sankar, P.V., Sklansky, J.: Minimal rectangular partitions of digitized blobs. Computer Vision, Graphics, and Image Processing 28(1), 58–71 (1984)
7. Flusser, J., Suk, T., Zitová, B.: Moments and Moment Invariants in Pattern Recognition. Wiley, Chichester (2009)
8. Imai, H., Asano, T.: Efficient algorithms for geometric graph search problems. SIAM Journal on Computing 15(2), 478–494 (1986)
9. Kawaguchi, E., Endo, T.: On a method of binary-picture representation and its application to data compression. IEEE Transactions on Pattern Analysis and Machine Intelligence 2(1), 27–35 (1980)
10. Keil, J.M.: Polygon decomposition. In: Handbook of Computational Geometry, pp. 491–518. Elsevier (2000)
11. Levin, A., Fergus, R., Durand, F., Freeman, W.T.: Image and depth from a conventional camera with a coded aperture. In: Special Interest Group on Computer Graphics and Interactive Techniques Conference, SIGGRAPH 2007. ACM, New York (2007)
12. Lipski Jr., W., Lodi, E., Luccio, F., Mugnai, C., Pagli, L.: On two-dimensional data organization II. In: Fundamenta Informaticae. Series IV, vol. II, pp. 245–260 (1979)
13. Ohtsuki, T.: Minimum dissection of rectilinear regions. In: Proceedings of the IEEE International Conference on Circuits and Systems, ISCAS 1982, pp. 1210–1213. IEEE (1982)
14. Schouten, T.E., van den Broek, E.L.: Incremental distance transforms (IDT). In: 20th International Conference on Pattern Recognition, ICPR 2010, pp. 237–240. IEEE Computer Society (August 2010)
15. Seaidoun, M.: A Fast Exact Euclidean Distance Transform with Application to Computer Vision and Digital Image Processing. Ph.D. thesis, Northeastern University, Boston, USA (September 1993) advisor John Gauch
16. Sossa-Azuela, J.H., Yáñez-Márquez, C., Díaz de León Santiago, J.L.: Computing geometric moments using morphological erosions. Pattern Recognition 34(2), 271–276 (2001)
17. Spiliotis, I.M., Mertzios, B.G.: Real-time computation of two-dimensional moments on binary images using image block representation. IEEE Transactions on Image Processing 7(11), 1609–1615 (1998)
18. Zakaria, M.F., Vroomen, L.J., Zsombor-Murray, P., van Kessel, J.M.: Fast algorithm for the computation of moment invariants. Pattern Recognition 20(6), 639–643 (1987)