Object Descriptors Based on a List of Rectangles: Method and Algorithm

Marc Van Droogenbroeck and Sébastien Piérard

INTELSIG Laboratory, Montefiore Institute, University of Liège, Belgium

Abstract. Most morphological operators use a unique structuring element, possibly at different scales, to describe an object. In addition, morphological algorithms are often restricted to 1D structuring elements, combinations of 1D elements, or isotropic structuring elements (like circles), because of the lack of methods directly applicable to arbitrary shaped 2D structuring elements. While these descriptors have proved useful in the past, we propose an alternative that uses the list of maximal rectangles contained in a set X.
In particular, we focus on an opening that preserves large rectangles contained in a set X and on its companion 2D algorithm that builds a list of all the maximal rectangles that fit inside an arbitrary set X. This list is the base of new descriptors that have been used successfully for machine learning tasks related to the analysis of human silhouettes.
For convenience, we provide the C source code and a program of our algorithm at `http://www.ulg.ac.be/telecom/rectangles`.

Keywords: Opening, Granulometry, Algorithm, Rectangle

1 Introduction

Tools for describing the shape of an object are useful for many applications, including classification. In mathematical morphology, the numerous tools include erosions, openings, skeletons, distance functions, etc. In that context, opening and closing operators play an important role, mainly because of their useful property of idempotence which is similar to the notion of ideal filter in linear filtering. If the property and behavior of many openings (like morphological openings [7], area openings [9], openings by reconstruction [6], or attribute openings [4]) are well known to practitioners, their implementation might still be problematic, mainly for multidimensional spaces. For example in the particular case of openings with rectangles, it is common to decompose the structuring element as the dilation of an horizontal line by a vertical line, and to apply the chain rule. While this procedure is extendable to 3D objects, algorithms that rely on a decomposition impose a processing order and become less convenient for 2D granulometries because intermediate results have to be stored.

From a practical point of view, we could classify binary openings in two families: (1) the family of openings that compare a structuring element to the set X to be interpreted (these openings are called morphological openings hereafter), and

(2) the family of attribute openings. Unlike morphological openings, attribute openings preserve the shape of a set X, because they simply test whether or not a connected component satisfies some increasing criterion Γ, called *attribute*. An example of valid attribute consists in preserving a set X if its area is superior to λ or removing it otherwise. This is in fact the surface area opening [9]. More formally, the attribute opening γ_Γ of a connected set X preserves this set if it satisfies the criterion Γ:

$$\gamma_\Gamma(X) = \begin{cases} X, & \text{if } X \text{ satisfies } \Gamma, \\ \emptyset, & \text{otherwise.} \end{cases} \tag{1}$$

Morphological openings affect the shape of an object. Therefore, in order to build a more descriptive shape analysis tool, they are often characterized by a parameter k leading to granulometries (with some specific restrictions on the shape of the structuring element) and granulometric functions or curves that provide a numeric result increasing (or decreasing) with k. Likewise, if attribute openings do not affect the shape of regions that are preserved (because they preserve the entire connected component [4]), they can be parametrized to provide a granulometric curve too. The underlying limitations are that it is hard to build a two-dimensional granulometric function and that all information about the location of an object in the image is lost with a granulometric curve.

In [8], we proposed an algorithm that computes a list of rectangles and derive granulometric curves. The advantages of having a list of rectangles are twofold: (1) it is simple to calculate granulometric curves once the list has been built because the list gathers all the information needed for the granulometry, and (2) it is possible to calculate, for each pixel, some statistics extracted from the list, like the size of the largest rectangles comprising that pixel. Descriptors based on a list of rectangles have proved successful for two classification tasks related to the analysis of human activities; they were used for gait recognition in [3], and for identifying human silhouettes in video scenes both in 2D [2] and in 3D [5].

In the following section, we illustrate our approach with one possible operator computable directly from a list of rectangles and show how to use it to build a granulometric curve. In Section 3, we describe an algorithm that computes the list of rectangles contained in an arbitrarily shaped object X. Section 4 concludes the paper.

2 Towards a Family Opening

2.1 Reminder

Consider the discrete space $\mathbb{Z}^2$. Given a set $X \subseteq \mathbb{Z}^2$ and a vector $b \in \mathbb{Z}^2$, the translate X_b is defined as $X_b = \{x + b | \, x \in X\}$. Let us take two subsets X and B of $\mathbb{Z}^2$. The dilation and erosion are respectively defined as

$$X \oplus B = \bigcup_{b \in B} X_b = \bigcup_{x \in X} B_x = \{x + b | \, x \in X, \, b \in B\}, \tag{2}$$

$$X \ominus B = \bigcap_{b \in B} X_{-b} = \{p \in \mathbb{Z}^2 | \, B_p \subseteq X\}. \tag{3}$$

Dilation and erosion are not inverse operators. If X is eroded by B and then dilated by B, one may end up with a smaller set than the original set X. This set, denoted by $X \circ B$, is called the *opening* of X by B and defined by $X \circ B = (X \ominus B) \oplus B$. One useful property of openings is that a morphological opening is the union of all the translate B_p included in X, that is

$$X \circ B = \bigcup \{B_p \, | \, B_p \subseteq X\} \, . \tag{4}$$

2.2 Definition of a Family Opening

We define a parametric opening operator that encompasses all the rectangles whose cardinality is larger or equal to k. In the following, we limit possible rectangles to rectangles whose sides are parallel to the $x - y$ system coordinates; in other words, all rectangles can be expressed as $nH \oplus mV$, where H and V are horizontal and vertical segments respectively. We could also consider other directions but the algorithm described in Section 3 then would have to be adapted.

The *family opening* of a set X by a family of rectangles whose cardinality or area is larger or equal to k, denoted $\gamma_k(X)$ hereafter, is defined by

$$\gamma_k(X) = \bigcup \{R \, | \, \#(R) \geq k \text{ and } R \subseteq X\} \, , \tag{5}$$

where $\#(R)$ denotes the cardinality of a rectangle R. This operator is the union of openings by all the rectangles that meet the size constraint, $\#(R) \geq k$, therefore it is an opening , but it is not an area opening as not only the area but also the shape is constrained. Note that we could use a different criterion to select rectangles from the list. In [3] for example, the shape descriptor is based on the histograms of all the rectangle widths and heights. A subset of all maximal rectangles was successively used to discriminate human shapes from objects shapes in [2].

A granulometric curve is easily derived from $\gamma_k(X)$. Granulometric curves can be obtained by taking the cardinality of the reconstructed area with all the rectangles that have a cardinality larger than a given threshold, which is the area of $\gamma_k(X)$. Fig. 1 draws these granulometric curves for some simple binary shapes. To ease the interpretation, we have only displayed the values for area threshold when they change the cardinality of the reconstructed area (we have removed all the plateaus); it is also possible to interpolate the values to smooth the curves. It is interesting to note for example that the overall shape of the granulometric curves of a diamond and a circle are similar, but that gaps are larger for a circle. Classification tools and machine learning techniques are adequate to interpret the information provided by such granulometric curves, for example to differentiate between several shapes.

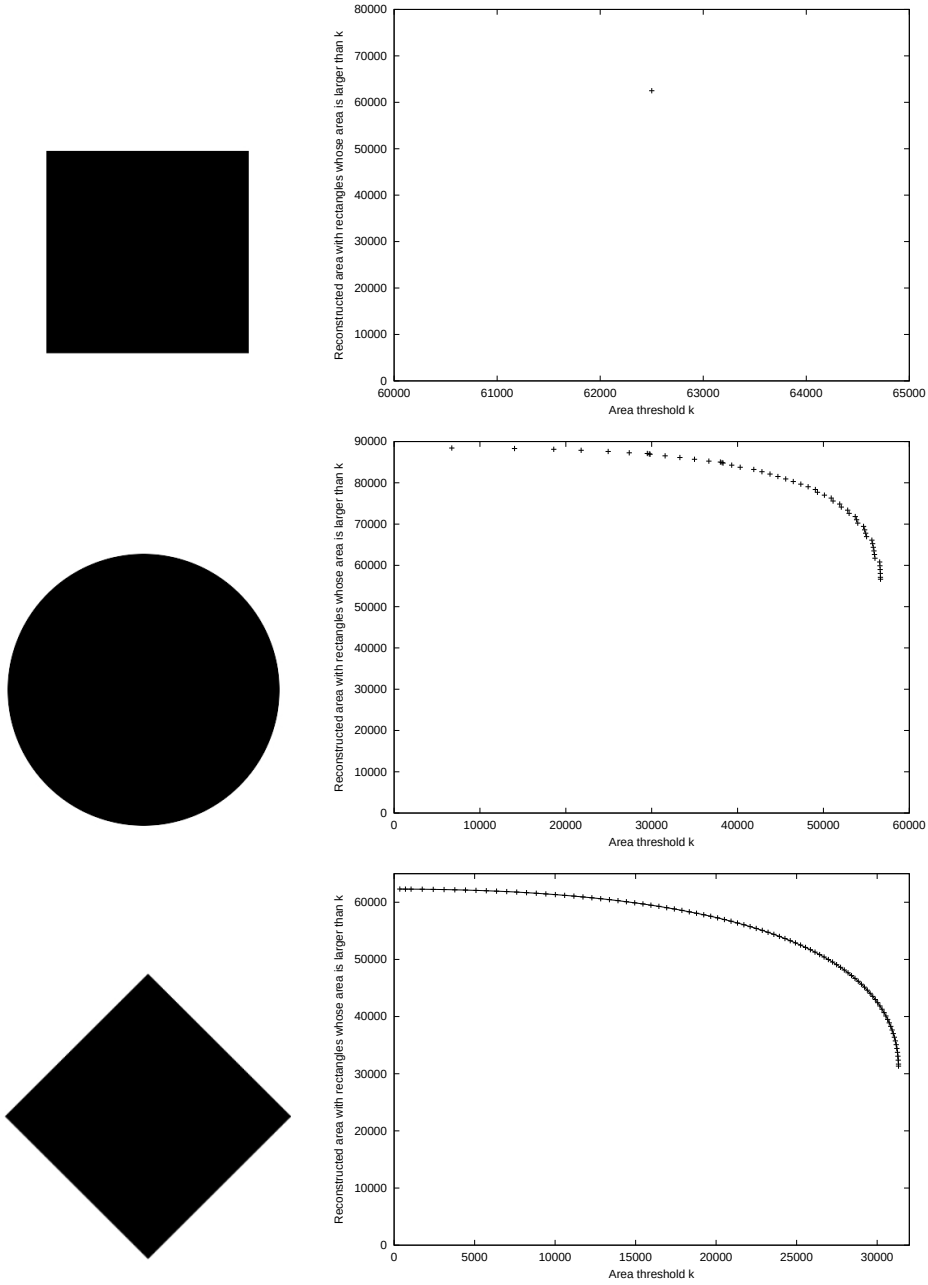

Fig. 1. Simple binary objects and their corresponding granulometric curves (obtained by taking the area of the family opening $\gamma_k(X)$ with respect to k).

3 Description of an Algorithm that Builds a List of all Maximal Rectangles Contained in a Binary Set X

While Vincent [10] proposed an efficient algorithm for computing 1D granulometries, only a few algorithms are applicable to 2D spaces. Several authors proposed variants that rely on the chain rule, which states that $X \ominus (H \oplus V) = (X \ominus H) \ominus V$ and that $X \oplus (H \oplus V) = (X \oplus H) \oplus V$, to deal with rectangles, but we consider that these algorithms are 1D in nature. For example, some decomposition properties were used by Bagdanov and Worring in [1] to derive rectangular granulometries and interpret the similarity of document images. In [8], we proposed a complex algorithm that computes two intermediate images containing all the necessary information for granulometries by rectangles. While this algorithm is efficient, the new algorithm described hereafter is much simpler and fast enough for real-time applications.

3.1 The Principles

There are different ways to characterize rectangles and to build a list of rectangles that fit inside an object X. In the following, we concentrate on maximal rectangles. By definition, if R is a maximal rectangle, then there is no R' such that $R \subset R' \subseteq X$. Note that if we consider all the possible rectangles, including the singleton (a rectangle degenerate to a single pixel), then for each location $x \in X$, there is at least one maximal rectangle that contains x. Of course, there might be more such rectangles.

A first subtle difference with the algorithm proposed in [8] is that we do not impose that each rectangle contains at least one point not included in any other rectangle of the list (this would lead to a minimal cover by rectangles). A second difference is the way to find the rectangles and the association of a rectangle with a *reference point*.

Before we define the notion of reference point, let us first remark that each maximal rectangle touches the upper, lower, left, and right borders; otherwise the rectangle would not be maximal. Our objective is to associate a unique reference point for each rectangle, but we accept that rectangles share a common reference point. By convention, we choose the reference point of a maximal rectangle to be the lowest point of the rectangle that touches the left border of the object.

Our algorithm is built around the concept of reference point. But, because we are not able to localize reference point in advance in an image, we introduce another notion, which is that of *candidate*. Candidates of a set X are elements of X that might be reference points of a maximal rectangle included in X. By definition, candidates are located on the left border of the object. This does not mean that all these points are reference points, but at least they are candidates.

The steps of our algorithm are then:

1. determine candidates, that is possible reference points,
2. search all the rectangles that could be associated to a candidate,

3. associate the maximal rectangle to a reference point if such a rectangle exists, and add the rectangle to the list of maximal rectangles.

A candidate is first selected during a scanning process of the image. A specific rule applies to the detection of candidates. If a maximal rectangle contains two candidates in the same column, the highest of the two candidates has to be ignored because that candidate is not a reference point according to our definition. Consequently, the downwards extension of a maximal rectangle rectangle starting from a reference point is bounded by the next to left contour point located beneath the reference point; this downwards vertical extension value is denoted maxS hereafter (see Fig. 2 for a graphical illustration of maxS). This specific rule eases the search for maximal rectangles.

The steps of our algorithm are illustrated on Fig. 2. The candidate is represented by a dark disk. We have put a horizontal line to denote the downwards limit maxS. The respective drawings are described hereafter.

1. Suppose that we have detected a candidate located at (col, row). We consider a first rectangle whose width is 1 and height is maximal. As the rectangle is extendable to the right with the same height, this rectangle is discarded.
2. The first rectangle is then extended to the right to reach a width of 2. This rectangle is maximal because it does not extend to the right, but it crosses the lower limit as defined by maxS. We have to ignore this rectangle too because the candidate is not the reference point for this rectangle. Depending on the scanning order, this rectangle is discovered earlier (upwards column scanning) or later (downwards column scanning).
3. We reduce the height of the rectangle and extend it to the right. This rectangle is maximal and its downwards extension is inferior or equal to maxS. Therefore the candidate is indeed a reference point and we must add this rectangle to the list.
4. The height of the third rectangle is reduced and the rectangle is extended to the right. This rectangle is not maximal, therefore, it is also discarded.
5. Again, we extend the rectangle to the right. The resulting rectangle is maximal and does not cross the lower limit. It is therefore added to the list of maximal rectangles.
6. Finally, we reduce the height of the fifth rectangle and extend it to the right. Since this sixth rectangle touches the right border of the object on the row row, it is maximal. Furthermore, it does not cross the lower limit. It is thus also added to the list.

Note that we now have three maximal rectangles associated to the same reference point located at (col, row). Therefore, the number of reference points is not an upper bound of the number of maximal rectangles.

3.2 Implementation

Listing 1 provides a C implementation of our algorithm. Note that 50 lines of C code suffice to compute the list! The function “listRectangles” computes the list

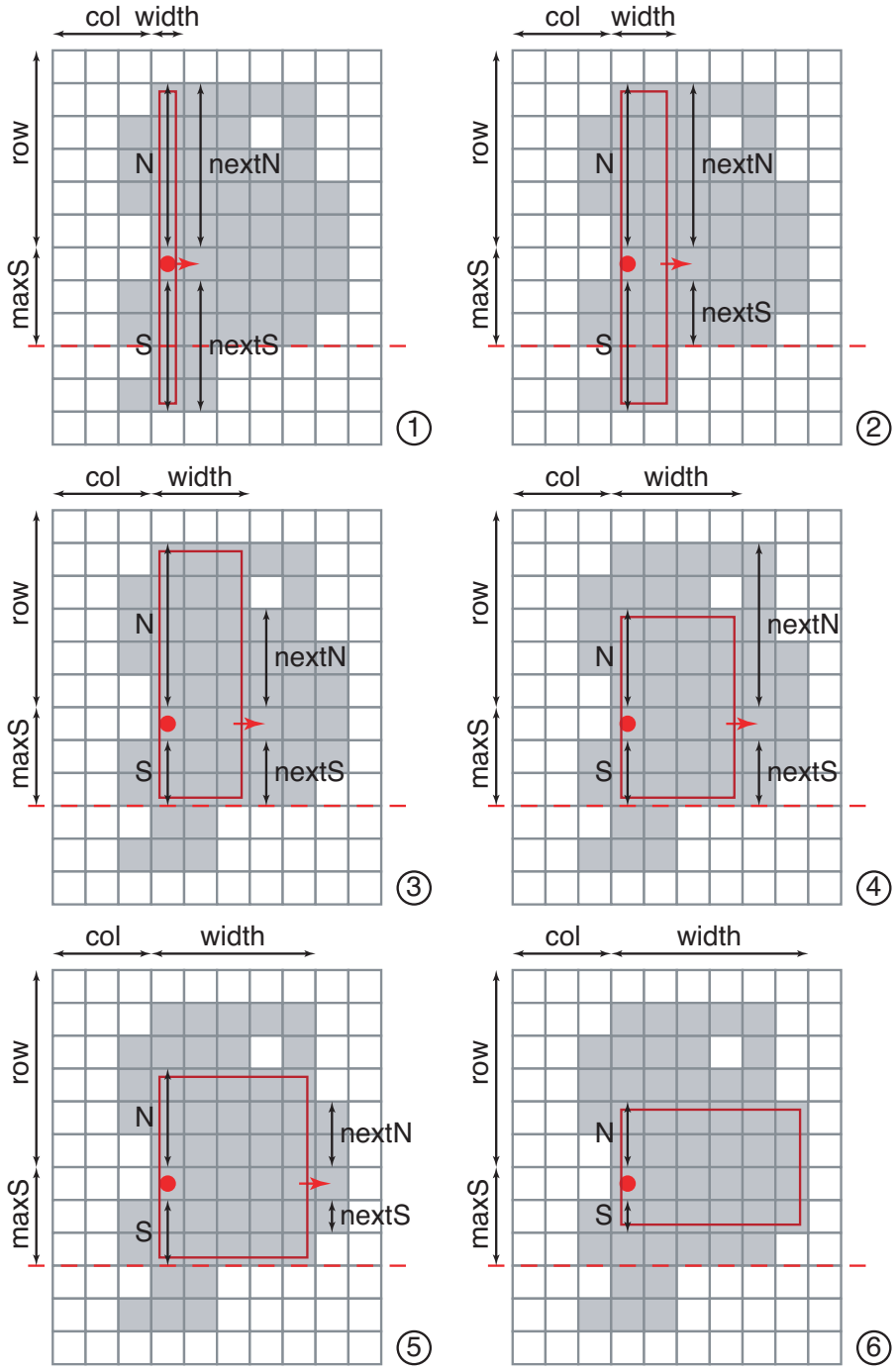

Fig. 2. Steps to detect all the rectangles associated to a candidate (denoted by a disk on the drawings).

Algorithm 1 An algorithm that detects all the maximal rectangles contained in an object X represented by a binary image named "object [w][h]".

```
1  void listRectangles ( int w , int h , bool object [ w ][ h ] ) {
2
3    int dN [ w ][ h ] ;
4    for ( int col = 0 ; col < w ; ++ col ) {
5      dN [ col ][ 0 ] = object [ col ][ 0 ] ? 0 : -1 ;
6    }
7    for ( int row = 1 ; row < h ; ++ row ) {
8      for ( int col = 0 ; col < w ; ++ col ) {
9        if ( ! object [ col ][ row ] ) dN [ col ][ row ] = -1 ;
10       else dN [ col ][ row ] = dN [ col ][ row - 1 ] + 1 ;
11     }
12   }
13
14   int dS [ w ][ h ] ;
15   for ( int col = 0 ; col < w ; ++ col ) {
16     dS [ col ][ h - 1 ] = object [ col ][ h - 1 ] ? 0 : -1 ;
17   }
18   for ( int row = h - 2 ; row >= 0 ; -- row ) {
19     for ( int col = 0 ; col < w ; ++ col ) {
20       if ( ! object [ col ][ row ] ) dS [ col ][ row ] = -1 ;
21       else dS [ col ][ row ] = dS [ col ][ row + 1 ] + 1 ;
22     }
23   }
24
25   for ( int col = w - 1 ; col >= 0 ; -- col ) {
26     int maxS = h ;
27     for ( int row = h - 1 ; row >= 0 ; -- row ) {
28       ++ maxS ;
29       if ( object [ col ][ row ]
30             && ( col == 0 || ! object [ col - 1 ][ row ] ) ) {
31         int N = dN [ col ][ row ] ;
32         int S = dS [ col ][ row ] ;
33         int width = 1 ;
34         while ( col + width < w && object [ col + width ][ row ] ) {
35           int nextN = dN [ col + width ][ row ] ;
36           int nextS = dS [ col + width ][ row ] ;
37           if ( ( nextN < N ) | ( nextS < S ) ) {
38             if ( S < maxS ) add ( col , row - N , width , N + S + 1 ) ;
39             if ( nextN < N ) N = nextN ;
40             if ( nextS < S ) S = nextS ;
41           }
42           ++ width ;
43         }
44         if ( S < maxS ) add ( col , row - N , width , N + S + 1 ) ;
45         maxS = 0 ;
46       }
47     }
48   }
49
50 }
```

of rectangles for an object represented by a 2D array of booleans named "object". This array has w columns and h rows. For each discovered maximal rectangle, the function calls a callback function named "add" with four arguments: the top left corner coordinates, the width, and the height. Note that, in this code, we have arbitrarily chosen the column-major order.
Some parts of the C code are commented hereafter.

- Lines 3 to 12. To optimize the search, we first compute the distance that separates a pixel located at (col , row) to the upper border of the object. The result is stored in the dN [col][row] data structure; the dN notation stands for *Distance to the North*.
- Lines 14 to 23. Likewise, distances between (col , row) and the downwards border is computed and stored in a specific data structure dS [col][row]. About half of the code is devoted to these simple search operations!
- Lines 25 and 27. This is the main loop on all the pixels and, inside the loop, the current location is (col , row).
  Ideally, only the border should be examined because reference points have to belong to the border, but this supposes that the border of the object is known prior to the scanning process. Note that the image is scanned upwards (line 27). This is an indirect consequence of our definition for reference points, because this order eases the determination of maxS, as explained hereafter.
- Lines 29 to 30. We detect if a pixel is a candidate. Only pixels located on the left border are candidates.
- Lines 31 to 44. All the maximal rectangles comprising (col , row) are considered. The upper left corner of such a maximal rectangle, its width, and its height are respectively (col , row − N), width, and N + S + 1.
- Lines 38 and 44. A rectangle is added to the list only if (col , row) is a reference point. An efficient way to determine if a candidate is a reference point uses the instructions of lines 26, 28, and 45. The scanning order helps us to determine the distance between (col , row) and a reference point located downwards in the same column, that is (col , row + maxS). This distance is stored in a variable called maxS, which is initialized as if a previous virtual reference point was located outside the image (therefore maxS is taken such that it is equal to h which is a value larger than the possible values, at line 26). Indeed, (col , row + h) is located outside the image. Once a candidate has been dealt with and before the algorithm moves to the next row index upwards, maxS is set to 0 (line 45) which avoids redundant rectangles in the list.

In the next two subsections, we discuss two issues related to the algorithm: the maximal size of the rectangle list, in order to bound the memory needed to store the list, and the complexity of the algorithm.

3.3 On the Number of Maximal Rectangles

We have seen that the number of reference points is not directly useful to determine a bound for the number of maximal rectangles contained in a binary set

X. However, for each maximal rectangle, the location (col + width − 1 , row) plays a particular role. It is located on the same row as the reference point and it indicates the right edge of the rectangle. By construction, it is impossible for two maximal rectangles to share this point. This observation allows us to derive an important upper bound for the number of rectangles on an object.

Property 1. The number of maximal rectangles contained in an object X is bounded by the cardinality of X.

Consequently, the maximal memory footprint to store the list of rectangles is bounded by storage size of one rectangle multiplied by the cardinality of an object X, which in some cases might be as large as the image.

3.4 Run-Time Complexity Analysis

Let $w \times h$ be the size of the image, and A the cardinality (that is the area expressed in pixels for discrete sets) of the object X. The overall run-time complexity of our algorithm is $\mathcal{O}(wh)$. Here are the details:

- The processing cost for adding rectangles to the list (which is expressed in the code by calling the function "add") is $\mathcal{O}(A)$. This is derived from Property 1.
- Precomputing dN and dS for each point inside X has a complexity of $\mathcal{O}(wh)$.
- Likewise, detecting all candidates also takes $\mathcal{O}(wh)$.
- For each candidate and each value of width to be considered, deciding if the rectangle is maximal plus computing the location and the size of this rectangle is achieved in constant time. Therefore, if dE is the distance between the candidate and the right border of X, the complexity of these operations is given by $\mathcal{O}(dE)$ for each candidate. As a conclusion, once the candidates are known, computing the list of rectangles only takes $\mathcal{O}(A)$, where A is the cardinality of X.

The combination of the complexity of all these steps leads to a complexity of our algorithm given as $\mathcal{O}(wh)$. In practice and in our algorithm, the bottleneck originates from the computation of dN and dS, and the detection of all candidates.

4 Conclusions

This paper presents methods that deal with rectangles to characterize the shape of objects. These methods, that rely on the list of all the maximal rectangles included into an arbitrarily shaped object X, offer an interesting alternative to 1D or isotropic descriptors. In addition, we propose an efficient algorithm that computes this list. For convenience, we provide the C source code and a program of our algorithm at `http://www.ulg.ac.be/telecom/rectangles`.

From this list, it is possible to derive granulometric curves and many other statistics. Some of these statistics were used successfully for gait recognition and

for the detection of human silhouettes. New opportunities that originates from the possibility to determine local statistics (that is, statistics for each pixel) are open for future works.

Acknowledgments. S. Piérard has a grant funded by the FRIA (`http://www.frs-fnrs.be/`).

References

1. Bagdanov, A., Worring, M.: Granulometric analysis of document images. In: IEEE International Conference on Pattern Recognition (ICPR). pp. 478–481. Québec, Canada (August 2002)
2. Barnich, O., Jodogne, S., Van Droogenbroeck, M.: Robust analysis of silhouettes by morphological size distributions. In: Advanced Concepts for Intelligent Vision Systems (ACIVS 2006). Lecture Notes on Computer Science, vol. 4179, pp. 734–745. Springer (September 2006)
3. Barnich, O., Van Droogenbroeck, M.: Frontal-view gait recognition by intra- and inter-frame rectangle size distribution. Pattern Recognition Letters 30(10), 893–901 (July 2009)
4. Breen, E., Jones, R.: Attribute openings, thinnings, and granulometries. Computer Vision and Image Understanding 64(3), 377–389 (November 1996)
5. Piérard, S., Lejeune, A., Van Droogenbroeck, M.: 3D information is valuable for the detection of humans in video streams. In: Proceedings of 3D Stereo MEDIA. Liège, Belgium (December 2010)
6. Salembier, P., Serra, J.: Flat zones filtering, connected operators, and filters by reconstruction. IEEE Transactions on Image Processing 4(8), 1153–1160 (August 1995)
7. Soille, P.: Morphological image analysis: principles and applications. Springer-Verlag, Berlin Heidelberg (1999)
8. Van Droogenbroeck, M.: Algorithms for openings of binary and label images with rectangular structuring elements. In: Talbot, H., Beare, R. (eds.) Mathematical morphology, pp. 197–207. CSIRO Publishing, Sydney, Australia (April 2002)
9. Vincent, L.: Morphological area openings and closings for greyscale images. In: NATO Shape in Picture Workshop. pp. 197–208. Springer-Verlag, Driebergen, The Netherlands (September 1992)
10. Vincent, L.: Grayscale area openings and closings, their efficient implementation and applications. In: Mathematical Morphology and its Applications to Signal Processing. pp. 22–27. Barcelona, Spain (May 1993)