A Fast Skeleton Algorithm
on Block Represented Binary Images

Iraklis M. Spiliotis and Basil G. Mertzios
Automatic Control Systems Laboratory
Department of Electrical and Computer Engineering
Democritus University of Thrace
Xanthi 67100
HELLAS
Fax: +30-541-26473
e-mail: spiliot@demokritos.cc.duth.gr
mertzios@demokritos.cc.duth.gr

Abstract
This paper describes a binary image representation scheme, which is called Image Block Representation and presents a new skeletonization algorithm, which is fast implemented on block represented binary images. The main purpose of the Image Block Representation is to provide an efficient binary image representation that permits the execution of operations on image areas instead of image points. The skeletonization algorithm operates in four subiterations: each subiteration deletes the north, the south, the west and the east boundary points, respectively. Due to the substitution of the boundary points by the block’s boundary points the relevant operations are performed fast, while preserving the end points and the object connectivity.

Keywords: Image Block Representation, Skeletonization, Thinning

I. INTRODUCTION
The most common image representation format is the two-dimensional (2-D) array. However, many research efforts for deriving alternative image representations have been motivated by the need of fast processing of huge amount of data. Such image representation approaches aim to provide machine perception of images in pieces larger than a pixel and are separated in two categories: boundary based methods and region based methods and include quadtree representations [1], chain code representations [2], contour control point models [3], autoregressive models [4], run length encoding [5],[6] and interval coding representation [7]. A region based method, which is called image block representation has been presented [8]-[12].

This paper presents a skeletonization algorithm, which is characterized by low computational cost and it is suitable for fast processing rates, due to the substitution of image pixels from blocks. The algorithm operates in four subiterations: each subiteration deletes the north, the south, the west and the east boundary points, respectively. Due to the substitution of the boundary points by the block’s boundary points the relevant operations are performed fast.

II. IMAGE BLOCK REPRESENTATION
A bilevel digital image is represented by a binary 2-D array. Without loss of generality, we suppose that the object pixels are assigned to level 1 and the background pixels to level 0. Due to this kind of representation, there are rectangular areas of object value 1, in each image. These rectangular areas, which are called blocks, have their edges parallel to the image axes and contain an integer number of image pixels. At the extreme case, one pixel is the minimum rectangular area of the image.

Consider a set that contains as members all the nonoverlapping blocks of a specific binary image, in such a way that no other block can be extracted from the image (or equivalently each pixel with object level belongs to only one block). This set represents the image without loss of information. It is always feasible to represent a binary image with a set of all the nonoverlapping blocks with object level. We call this representation of the binary image, Image Block Representation (IBR).
The block representation concept leads to a simple and fast algorithm, which requires just one pass of the image and simple bookkeeping process. In fact, considering a \( N \times N \) binary image \( f(x,y) \), \( x=0,1, \ldots, N_1-1 \), \( y=0,1, \ldots, N_2-1 \), the block extraction process requires a pass from each line \( y \) of the image. In this pass all object level intervals are extracted and compared with the previous extracted blocks.

A block represented binary image \( f(x,y) \) is comprised of a set of nonoverlapping blocks that completely cover the image areas with object level and it is denoted as:

\[
 f(x,y) = \{ b_i : i = 0,1, \ldots, k-1 \} 
\]

where \( k \) is the number of the blocks. Each block is described by the coordinates of two corner points, i.e.:

\[
 b_i = (x_{1,b_i}, x_{2,b_i}, y_{1,b_i}, y_{2,b_i}) 
\]

where for simplicity it is assumed that: \( x_{1,b_i} \leq x_{2,b_i} \) and \( y_{1,b_i} \leq y_{2,b_i} \). In Fig. 1, the blocks that represent an image of the character d are illustrated.

Figure 1. Image of the character d and the blocks.

In many operations it is important to have information concerning not only the location of the blocks but also information concerning the neighbor and connected blocks. The following definition provides a template for block connectivity:

**Definition**

Two blocks are defined as connected, if their projections on both the \( x \) or \( y \) axis are overlapped or there are neighbors.

The information concerning block connectivity requires a suitable data structure for storage. Therefore, each block \( b_i \) is represented as the ordering:

\[
 b_i = (x_{1,b_i}, x_{2,b_i}, y_{1,b_i}, y_{2,b_i}, n_c_i, c_i) 
\]

where \( x_{1,b_i}, x_{2,b_i} \) are the coordinates of the \( i \)-th block according to the horizontal axis, \( y_{1,b_i}, y_{2,b_i} \) are the coordinates of the block according to the vertical axis, \( n_c_i \) is the number of the connected blocks and \( c_i \) is a list with the indexes of these connected blocks.

The image block representation is reduced to the run length encoding [5],[6] of binary images at the extreme cases, where each block is comprised of pixels belonging in only one row of the image. Such a case is that of a chessboard image, where the transitions from white to black have 1 pixel length and the number of the blocks is \( N^2/2 \), i.e. it is exactly equal to the object level run lengths. However, in most practical situations, the image block representation is superior to the run length encoding, since the number of the blocks is significantly smaller than the number of the run lengths. In Fig. 2 four test images are illustrated, while in Table 1 the number of the pixels with object level, the number of the rows with object pixels, the number of the blocks extracted from these images (using the Algorithm 1) and the required storage space for both the 2-D represented and the block represented images are shown. It can be seen that the number of the blocks generated by the Algorithm 1, is significantly less than the number of the rows with black pixels. In the worst case of the island Mikonos image (of Fig. 2 (b)), where the number of the rows with object pixels is 249 and the number of the blocks is 232, it should be noted that the number of the gulfs and peninsulas of the island is significantly large and therefore the number of the blocks is respectively large.

**III. FAST SKELETONIZATION ALGORITHM**

The algorithm uses the criteria specified by the well-known skeletonisation algorithm of Zhang and Suen [13]. The proposed algorithm is implemented iteratively; each iteration is divided into four subiterations. In the first, second, third and fourth subiteration, the north, west, south and east pixels of the object, are removed respectively. The main advantage of the proposed algorithm is that operates in blocks and therefore permits the deletion of areas of pixels, instead of a single pixel.
Figure 2. A set of test images. (a). Image of the island Corfu of 512x512 pixels. (b) Image of the island Mikonos of 512x512 pixels. (c) Image of the island Santorini of 512x512 pixels. (d) Aircraft image of 512x697 pixels.

Table 1. The number of the pixels with object level, the number of the rows with object pixels, the number of the blocks, the required storage for the 2-D images and the required storage for the block represented images for the set of the test images of Figure 2.

<table>
<thead>
<tr>
<th>Image</th>
<th>Pixels with object level</th>
<th>Rows with object pixels</th>
<th>Number of blocks</th>
<th>Storage for the 2-D image</th>
<th>Storage for blocks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corfu</td>
<td>41605</td>
<td>411</td>
<td>250</td>
<td>32768 bytes</td>
<td>2000 bytes</td>
</tr>
<tr>
<td>Mikonos</td>
<td>47368</td>
<td>249</td>
<td>232</td>
<td>32768 bytes</td>
<td>1856 bytes</td>
</tr>
<tr>
<td>Santorini</td>
<td>63203</td>
<td>474</td>
<td>257</td>
<td>32768 bytes</td>
<td>2056 bytes</td>
</tr>
<tr>
<td>Aircraft</td>
<td>118831</td>
<td>494</td>
<td>397</td>
<td>44608 bytes</td>
<td>3176 bytes</td>
</tr>
</tbody>
</table>

A. North subiteration
At first the neighboring blocks of the considered block are determined, sorted and placed in two lists; one for the upper neighbors and one for the lower neighbors. Different procedures are applied for those blocks that have unity width than those with greater width.

A.1. Blocks with unity width
For the blocks with unity width, the blocks that remain after the subiteration are described as:

\[ R = N \cup S \cup A \]  

where \( N \) are the areas that have a north neighbor, \( S \) are those areas that have a south neighbor and \( A \) are those areas that have transitions from 01 patterns different from 1. If \( b \) is the considered block and \( a_i, i=1,...,k \) are the north neighbor blocks of \( b \), then

\[ N=(\max(b_{i1},a_{i1}), \min(b_{i2},a_{i2})), i=1,...,k. \]  

\[ S \] is computed in a similar manner, and \( \bar{S} \) is computed using logic operations in blocks, as described in [11], [12].

The algorithm counts the transitions for the two extreme pixels of the considered block. For the middle pixels of the block, the algorithm recognizes the areas \( A \) as those that have at least one north and at least one south neighbor pixel.

A.2. Blocks with width greater than 1
For the blocks with width greater than 1, the algorithm determines the remaining areas as those that have a north neighbor block. Care should be taken only for the two extreme pixels, the upper left and the upper right of the block, where all the criteria of Zhang and Suen are applied.

The procedure is quite similar for the south subiteration.

B. East subiteration
At first, the neighboring blocks of the considered block are determined, sorted and placed in two lists; one for the upper neighbors and one for the lower neighbors.

B.1. Blocks with unity width
In the case of a block with unity width, all the criteria of Zhang and Suen [13], are taken into account.

B.2. Blocks with width greater than 1
In the case of a block with width greater than 1, the algorithm deletes all the middle points of the left side of the block. For the two extreme pixels, the upper left and the lower left of the block, the criteria of Zhang and Suen are also applied.

The procedure is quite similar for the west subiteration.

The algorithm is fast implemented, since it operates on image areas instead on single pixels. Table 2 demonstrates the required computational times for the execution of the skeletonization for the images of Fig. 2, using the Zhang and Suen algorithm and the proposed algorithm that operates in blocks.
Table 2. The required computational times in seconds, for the execution of the skeletonization operation using the Zhang and Suen algorithm and the proposed algorithm. The third column is the reduction factor.

<table>
<thead>
<tr>
<th>Image</th>
<th>Zhang - Suen</th>
<th>Blocks</th>
<th>Reduction</th>
</tr>
</thead>
<tbody>
<tr>
<td>Corfu</td>
<td>12.1</td>
<td>2.9</td>
<td>4.1</td>
</tr>
<tr>
<td>Mikonos</td>
<td>14.6</td>
<td>2.8</td>
<td>5.2</td>
</tr>
<tr>
<td>Santorini</td>
<td>12.6</td>
<td>3.1</td>
<td>4.1</td>
</tr>
<tr>
<td>aircraft</td>
<td>18.4</td>
<td>4.7</td>
<td>3.9</td>
</tr>
</tbody>
</table>

CONCLUSIONS

In the recent years other of fast image processing and analysis algorithms that operate on block represented binary images have been presented in the literature: specifically the real-time computation of the statistical moments (both software [10] and hardware algorithms [14]), the fast implementation of image shift, image scale, image rotation, determination of the minimum and of the maximum distance from a point to an object, perimeter measurement, area measurement, logic operations, connectivity checking, object detection and edge extraction [12].

The proposed skeletonization algorithm may be considered as a parallel skeletonization algorithm that operates in a serial machine. This conclusion arises from the fact that the algorithm decides which are the pixels that should form a new block, or equivalently deletes simultaneously a number of boundary pixels for which the conditions of removing are valid.

REFERENCES