Asian Journal of Information Technology 10 (3): 113-118, 2011
ISSN: 1682-3915

Morphological Filtering and Structural Coding Analysis of an Image

M. Sreedevi and P. Jenopaul
Anna University, Tamil Nadu, India

Abstract: This study presents an efficient algorithm that performs basic Mathematical Morphology operations like opening and closing with any arbitrary shaped structuring element. It is shown that this algorithm has a lower complexity and better computing time than all other known methods. The required computing time is independent of the image content and of the number of gray levels used. This also can slightly reduce the computational cost per size or shape. Shape parameters are extracted from objects in order to describe their shape to compare it to the shape of template objects or to partition objects into classes of different shapes. Objects can be viewed from different distances and from different points of view. Thus it is of interest to find shape parameters that are scale and rotation invariant or that are even invariant under affine or perspective projection.

Key words: Opening, closing, mathematical morphology, arbitrary shaped structuring element, decomposition, India

INTRODUCTION

Mathematical Morphology (MM) is a theory devised for the shape analysis of objects and functions. Usual morphological operators are made of two parts: a reference shape called a structuring element or function that is translated and compared to the original function all over the plane and a mechanism that details how to carry out the comparison (Urbach and Wilkinson, 2008). Morphology refers to a broad class of non-linear shape filters. Linear filters can be implemented by direct convolution or in the frequency domain using FFTs. Often, linear filters are fail to solve the problems related to geometrical aspects. So we prefer the non-linear filters. Morphological filters are the most preferable non-linear filter. Like the linear filters the operation is defined by a matrix of elements applied to input image neighborhoods but instead of a sum of products, a minimum or maximum of sums is computed.

Non-linear filters designed to pass or block desired shapes rather than spatial frequencies have been found useful for digital image enhancement. Morphological operators like opening and closing with Structuring Elements (SE) are the most fundamental operators in mathematical morphology and have become common tools for both image filtering and analysis of binary and grayscale images especially since the development of efficient algorithms (Van Vliet and Verwer, 1988; Anelli *et al*., 1998; Gil and Kimmel, 2002). Opening filter, defined as the application of the min filter to the max filter. Closing filter defined as the application of the max filter to the min filter.

MM has become increasingly popular over the last few years. Part of this success is due to the remarkable increase in efficiency of many of the MM algorithms. Many MM operations which used to require expensive dedicated hardware to run in reasonable times can now be performed on standard workstation or even personal computers. Numbers of algorithms were developed to increase the computational efficiency of the Morphological operators. Usually, these efficient algorithms can only be used for binary images (Van Vliet and Verwer, 1988; Anelli *et al*., 1998; Nikopoulos and Pitas, 2000) or they are limited to shapes that can (efficiently) be decomposed into a series of linear SEs (Soille *et al*., 1996; Gil and Kimmel, 2002).

A queue is implemented to store the contours in each iteration for the next iteration. The contours can be passed from one operation to another as well. In the initializing phase of the operation pointers to contour pixels are queued. During the iterations pixels which pop from the end of the queue are processed and the neighbours to be processed in the next iteration are pushed to the front of the queue. After the queue initialization, the operations proceed along different but comparable paths. Either the neighbours of the queue pixels are processed and queued or the pixels on the queue are processed themselves and their neighbours are

Corresponding Author: M. Sreedevi, Anna University, Chennai, Tamil Nadu, India

queued. Most often the first method is used to prevent pixels from being queued twice without having to use labels. Here we process as less pixels as possible and to keep the calculations as simple as possible. Disadvantage of this algorithm is it suitable for only binary images and limited to shapes that can be decomposed into a series of linear structuring elements and requires large queue size.

Van Droogenbroeck and Talbot (1996) proposed an efficient algorithm for computing morphological operations with arbitrary 2-D shapes using a histogram which makes the computing time of their algorithm dependent on the number of gray levels used. Their idea is to compute for one pixel of the image the complete histogram based on the intensities of the pixels round corresponding to elements of the SE. For all succeeding pixels of the image, the histogram is efficiently updated and the position of its minimum intensity changes only if a new minimum value is shifted into the histogram which can be kept track when the histogram is updated or when the current minimum is shifted out of the histogram in which case, the algorithm searches for the first following (brighter) intensity which is now represented in the histogram. But this method depends on number of gray levels used.

All methods based on decomposition of 2-D SEs into linear SEs share the same limitation: many shapes either cannot be decomposed efficiently or they cannot be decomposed at all. In the binary case, efficient algorithms for some 2-D shapes like circles do exist but these cannot efficiently be extended to the grayscale case, for which polygonal approximations (Soille *et al*., 1996; Soille and Talbot, 2001) of circles usually are used instead. For larger circles these approximations tend be either too coarse or too computationally intensive since the number or linear Ses required is proportional to the diameter of the circle.

Algorithms that efficiently perform morphological operators with arbitrary SEs not only are important for those cases where the SE cannot be decomposed. Furthermore, for many applications the benefits of using the fastest specialized algorithm available instead of using one slightly less efficient generic algorithm does not outweigh the costs involved in adapting the methods used SE shape decompositions require some design and programming efforts that can be avoided if a generic algorithm is used (Park and Chin, 1995).

This study presents a new method for performing morphological operators with any 2-D flat structuring element that always outperforms existing algorithms for arbitrary structuring elements, the performance independent of both image content and the number of gray levels used and application of a single operator using many different structuring elements. It works on floating point data also. This algorithm is efficient for any SE and only significantly outperformed when large linear SEs or compositions of linear SEs are used with a dedicated algorithm such as proposed by Gil and Kimmel (2002).

PROPOSED METHOD

Basic principle: The erosion operation is useful for removing small objects. However, it has the disadvantage that all the remaining objects shrink in size. We can avoid this effect by dilating the image after erosion with the same structure element. This combination of operations is called an opening operation:

$$G \circ M = (G; M)\, oM \tag{1}$$

The opening sieves out all objects which at no point completely contain the structure element but avoids a general shrinking of object size. It is also an ideal operation for removing lines with a thickness smaller than the diameter of the structure element. Note also that the object boundaries become smoother. In contrast, the dilation operator enlarges objects and closes small holes and cracks. General enlargement of objects by the size of the structure element can be reversed by a following erosion. This combination of operations is called a closing operation:

$$G \bullet M = (G \circ M); M \tag{2}$$

The area change of objects with different operations may be summarized by the following relations:

$$G; M\ o\ G\ {}^{\circ}M\ \ G\ o\ G \bullet M\ o\ G\ oM$$

Opening and closing are idempotent operations:

$$G \bullet M = (G \bullet M) \bullet M \tag{3}$$

$$G \circ M = (G \circ M) \circ M \tag{4}$$

The algorithm decomposes an arbitrary SE into series of chords i.e., runs of foreground pixels of maximum extent as shown in Fig. 1 for a letter H. Each chord can be considered as a single horizontal linear SE and is represented by a triple containing its y-offset with respect

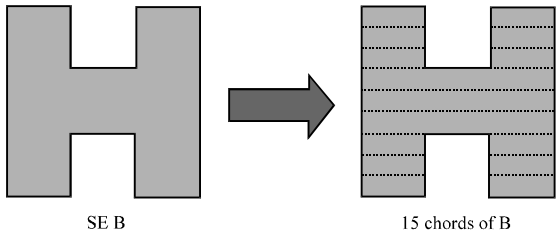

Fig. 1: Decomposition of SE B into chords

to the origin of the SE, its minimal x-position and its length l. For a SE B consisting of N_c chords, we store the number of N_c chords, the set C of N_c chords, the minimum and maximum y-offsets, y_{min} and y_{max}, the minimum and maximum x-values x_{min} and x_{max} and the maximum chord length l occurring in B. Let us now compute the minimum intensity value:

$$v_c^{min}(x,y)=\min_{i=0}^{l(c)-1} f(x+x(c)+i, y+y(i)) \quad (5)$$

Representation of shape

Run-length code: A compact, simple and widely used representation of an image is run length code. The run-length code is produced by the following procedure. An image is scanned line by line. If a line contains a sequence of p equal:

- Original line (hex): 12 12 12 20 20 20 20 25 27 25 20 20 20 20 20 20
- Code (hex): 82 83 2 85
- Original line (hex): 1 1 1 1 1 1 0 0 0 1 1 1 0 0 1 0 0 0 0 0 1 1 1 1 1 1 1 1
- Code (hex) 0 6 3 3 2 1 5 8

pixels, we do not store p times the same value but store the pixel value once and indicate that it occurs p times. In this way, large uniform line segments can be stored very efficiently. For binary images, the code can be especially efficient as only the two pixel values zero and one occur. As a sequence of zeroes is always followed by a sequence of ones, there is no need to store the pixel value. We only need to store the number of times a pixel value occurs. We must be careful at the beginning of a line, however as it may begin with a one or a zero. This problem can be resolved if we assume a line to begin with zero. If a line should start with a sequence of ones, we start the run-length code with a zero to indicate that the line begins with a sequence of zero. Run-length code is suitable for compact storage of images. It has become an

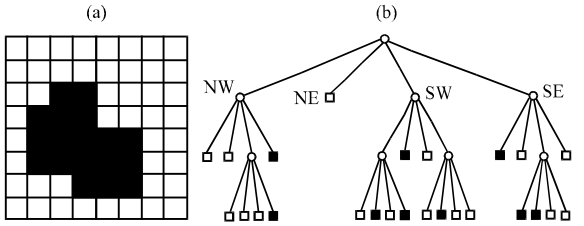

Fig. 2: Representation of a binary image by a region quadtree: (a) successive subdivision of the binary image into quadrants; (b) the corresponding region quadtree

integral part of several standard image formats, for example, the TGA or the TIFF file formats. Run-length code is less useful for direct processing of images however because it is not object oriented. As a result, run-length encoding is more useful for compact image storage. Not all types of images can be successfully compressed with this scheme. Digitized gray-scale images, for example always contain some noise so that the probability for sufficiently long sequences of pixels with the same gray value is very low. However, high data reduction factors can be achieved with binary images and many types of computer-generated gray-scale and color images.

Quad trees: The run-length codes are a line-oriented representation of binary images. Thus they encode one-dimensional rather than two-dimensional data. The two-dimensional structure is actually not considered at all. In contrast, a quad tree is based on the principle of recursive decomposition of space as shown in Fig. 2 for a binary image.

First, the whole image is decomposed into four equal-sized quadrants. If one of the quadrants does not contain a uniform region i.e., the quadrant is not included entirely in the object or background, it is again subdivided into four sub quadrants. The decomposition stops if only uniform quadrants are encountered or if the quadrants finally contain only one pixel. The recursive decomposition can be represented in a data structure known in computer science as tree. At the top level of the tree, the root, the decomposition starts. The root that we return to a higher level in the tree only after the visited branch is completely encoded down to the lowest lev corresponds to the entire binary image. It is connected via four edges to four child nodes which represent from left to right the NW, NE, SW and SE quadrants. If a quadrant needs no further subdivision, it is represented by a

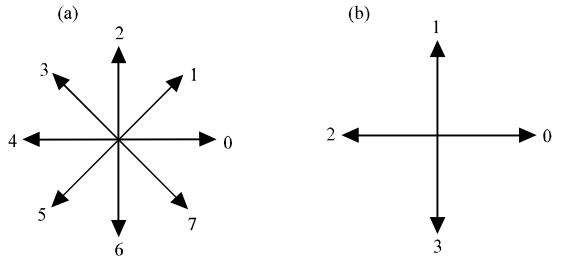

Fig. 3: Direction coding in (a) an 8-neighborhood and (b) a 4-neighborhood

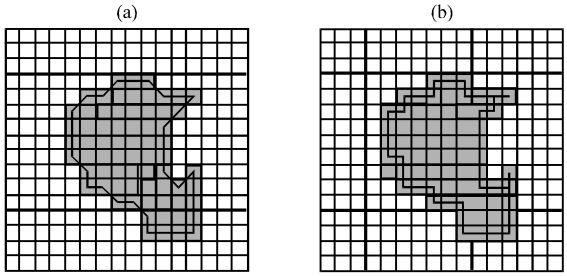

Fig. 4: Boundary representation with the chain code (a) 8- neighborhood; (b) 4-neighborhood

terminal node or a leaf node in the tree. It is called black when the quadrant belongs to an object and white otherwise indicated by a filled and open square, respectively. Non leaf nodes require further subdivision and are said to be gray and shown as open circles. Quad trees can be encoded for example by a depth-first traversal of the tree starting at the root. It is only required to store the type of the node with the symbols b (black), w (white) and g (gray). We start the code with the value of the root node. Then we list the values of the child nodes from left to right. Each time we encounter a gray node, we continue encoding at one level lower in the tree. This rule is applied, recursively. This means that we return to a higher level in the tree only after the visited branch is completely encoded down to the lowest level. This is why this encoding is named depth-first. The example quad tree shown in Fig. 2b results in the code becomes more readable if we include a left parenthesis each time we descend one level in the tree and a right parenthesis.

However, the code is unique without the parentheses. A quad tree is a compact representation of a binary image if it contains many leaf nodes at high levels. However, in the worst case, for example a regular checkerboard pattern, all leaf nodes are at the lowest level. The quad tree then contains as many leaf nodes as pixels and thus requires many more bytes of storage space than the direct representation of the binary image as a matrix. The region quad tree discussed here is only one of the many possibilities for recursive spatial decomposition is shown in Fig. 3. Three-dimensional binary images can be recursively decomposed in a similar way. The 3-D image is subdivided into eight equally sized octants. The resulting data structure is called a region octree. Quad trees and octrees have gained significant importance in geographic information systems and computer graphics. Quad trees are a more suitable encoding technique for images than the line-oriented run-length code. But they are less suitable for image analysis. It is rather difficult to perform shape analysis directly on quad trees. Without going into further details, this can be seen from the simple fact that an object shifted by one pixel in any direction results in a completely different quad tree. Region quad trees share their most important disadvantage with run-length code: the technique is global image decomposition and not one that represents objects extracted from images in a compact way.

Chain code: In contrast to run-length code and quad trees, chain code is an object related data structure for representing the boundary of a binary object effectively on a discrete grid. Instead of storing, the positions of all the boundary pixels, we select a starting pixel and store only its coordinate. If we use an algorithm that scans the image line by line, this will be the uppermost left pixel of the object. Then we follow the boundary in a clockwise direction. In a 4-neighborhood there are 4 and in an 8- neighborhood 8 possible directions to go which we can encode with a 3 or 2 bit code is shown in Fig. 4a and b. The boundary and its code extracted in this way for a 4-neighborhood and 8-neighborhood. The chain code shows a number of obvious advantages over the matrix representation of a binary object: First, the chain code is a compact representation of a binary object. Let us assume a disk-like object with a diameter of R pixels. In a direct matrix representation we need to store the bounding box of the object i.e., about R2 pixels which are stored in R2 bits. The bounding rectangle is the smallest rectangle enclosing the object. If we use 8-connected boundary, the disk has about πR boundary points. The chain code of the πR points can be stored in about 3πR bits. For objects with a diameter >10, the chain code is a more compact representation. Second, the chain code is a translation invariant representation of a binary object. This property makes it easier to compare objects. However, the chain code is neither rotation nor scale invariant. This is a significant disadvantage for object recognition, although the chain code can still be used to extract rotation invariant parameters such as the area of the object. Third, the chain code is a complete representation of an object or curve. Therefore, we can at least in principle compute

any shape feature from the chain code. we can compute a number of shape parameters including the perimeter and area more efficiently using the chain-code representation than in the matrix representation of the binary image. The limitation here is of course that the chain code is a digital curve on a discrete grid and as such describes the boundary of the object only within the precision of the discrete grid.

Performing the structural analysis: The total computation time required for efficiently if C would be a list of vertical chords. If the SE has the shape of a letter H of width and height 49 and with its legs 1 pixel performing an erosion using the proposed algorithm depends strongly on the number of chords.

Since some SEs have considerably fewer vertical run-lengths than horizontal ones, erosions using those can be computed more thick is shown in Fig. 5 that 2×48+1 = 97, chords whereas using vertical run lengths a set of the same SE is represented using only 2+47 = 49 chords. This algorithm shows no change in computing time between different images of the same size.

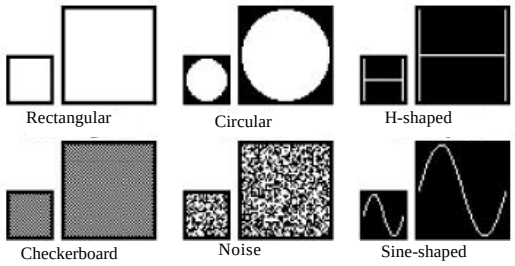

Fig. 5: SEs used in experiments; shapes shown are 23 and 49 pixels wide. Black borders were added to these illustrations here for clarity; these borders were not used in the experiments

EXPERIMENTS

When an image is eroded with multiple SEs this approach provides a way to compute these less time than would be required when each of them would be computed separately. It is clear that the method proposed is always better than the existing methods. It shows that the required computing time for width l = k using the naive (Direct), the Van Droogenbroeck-Talbot (DT8 and DT16) and the proposed algorithm on a natural image is shown in Fig. 6. This approach provides less computing time than would be required each of them computed separately.

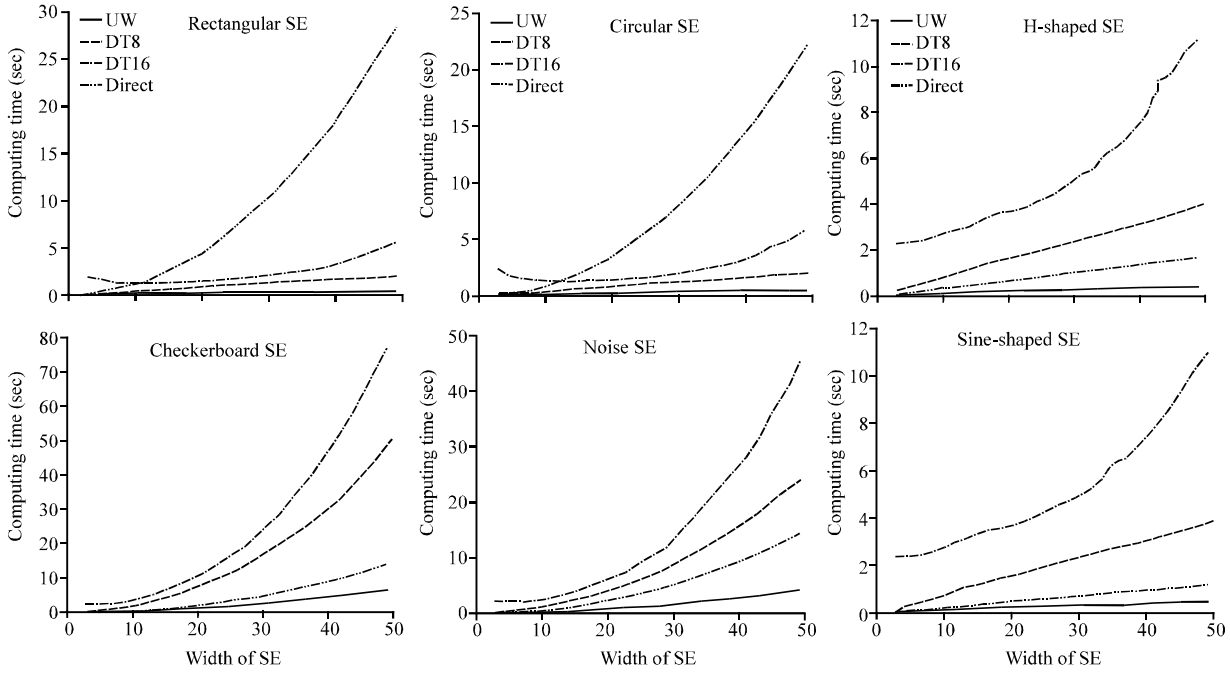

Fig. 6: Required computing time for width l = k using the naive (Direct), the Van Droogenbroeck-Talbot (DT8 and DT16) and the proposed algorithm on a natural image

CONCLUSION

A new method for computing morphological operations with arbitrary 2-D flat structuring elements was proposed. It has a computational complexity that is independent of the number of gray levels in the image. The proposed method has a clear computational performance advantage over existing methods when SEs are used that cannot be easily decomposed into linear structuring elements.

Remarkably, this method is even faster for erosions with shorter linear SEs than dedicated algorithms. A further, minor improvement is achieved when multiple SEs are used with a single operator. However, this method always outperforms the existing method especially when images with higher bit depth as common in applications such as medical imaging and astronomy are used.

REFERENCES

Anelli, G., A. Broggi and G. Destri, 1998. Decomposition of arbitrarily shaped binary morphological structuring elements using genetic algorithms. IEEE Trans. Pattern Anal. Mach. Intell., 20: 217-224.

Gil, J.Y. and R. Kimmel, 2002. Efficient dilation, erosion, opening and closing algorithms. IEEE Trans. Pattern Anal. Mach. Intell., 24: 1606-1617.

Nikopoulos, N. and I. Pitas, 2000. A fast implementation of 3-D binary morphological transformations. IEEE Trans. Image Process., 9: 283-286.

Park, H. and R.T. Chin, 1995. Decomposition of arbitrarily shaped morphological structuring elements. IEEE Trans. Pattern Anal. Mach. Intell., 17: 2-15.

Soille, P. and H. Talbot, 2001. Directional morphological filtering. IEEE Trans. Pattern Anal. Mach. Intell., 23: 1313-1329.

Soille, P., E.J. Breen and R. Jones, 1996. Recursive implementation of erosions and dilations along discrete lines at arbitrary angles. IEEE Trans. Pattern Anal. Mach. Intell., 18: 562-567.

Urbach, E.R. and M.H.F. Wilkinson, 2008. Efficient 2-D grayscale morphological transformations with arbitrary flat structuring elements. IEEE Trans. Image Process., 17: 1-8.

Van Droogenbroeck, M. and H. Talbot, 1996. Fast computation of morphological operations with arbitrary structuring elements. Pattern Recognition Lett., 17: 1451-1460.

Van Vliet, L.J. and B.J.H. Verwer, 1988. A contour processing method for fast binary neighbourhood operations. Pattern Recognition Lett., 7: 27-36.