Proc. VIIth Digital Image Computing: Techniques and Applications, Sun C., Talbot H., Ourselin S. and Adriaansen T. (Eds.), 10-12 Dec. 2003, Sydney

Diamond, Hexagon, and General Polygonal Shaped Window Smoothing

Changming Sun

CSIRO Mathematical and Information Sciences
Locked Bag 17, North Ryde, NSW 1670, Australia
changming.sun@csiro.au

Abstract. This paper presents fast recursive or moving windows algorithms for calculating local means in a diamond, hexagon and general polygonal shaped windows of an image. The algorithms for diamond shaped window require only seven or eight additions and subtractions per pixel. A number of other shapes of diamond windows such as skewed or parallelogram shaped diamond, long diamond, and lozenged diamond shaped, are also investigated. Similar algorithms are also developed for hexagon shaped windows. The computation for hexagon window only needs eight additions and subtractions for each pixel. Fast algorithms for general polygonal shaped windows are also developed. The computation costs of all these algorithms are independent of the window size. A variety of synthetic and real images have been tested.

1 Introduction

In most of the image analysis and computer vision applications, the local processing windows are square or rectangular shaped. Glasbey and Jones presented fast algorithms for moving average and related filters in regular octagonal windows as approximations to circular windows [4]. Ferrari and Sklansky proposed a two step method for obtaining the mean of an arbitrary shaped window [3]. But their method will not be very efficient for diamond and hexagon shaped windows. Verbeek et al presented min or max filters for low-level image processing [10]. They gave six shapes for the min or max filter, including a full square, a full diamond, a sampled diamond, a discrete approximation of a full circle, the rim and the center, and eight contour points and the center. Soille and Talbot presented a decomposition method of morphological operations for diamond shaped and rotated rectangles [7]. Van Herk also developed a fast algorithm for local min or max filters on rectangular and octagonal kernels [9]. Van Droogenbroeck and Talbot presented a general algorithm that performs basic mathematical morphology operations with any arbitrary shaped structuring element in an efficient way [8].

In some applications such as image processing and stereo matching, the processing window can be diamond or hexagon or general polygon shaped. For example, diamond shaped window could be used when object boundary are roughly in the diagonal direction. The center of the diamond window are closer to the

object boundary than that of the square window without intersecting with the boundary. Baaziz and Dubois used separable diamond shaped filtering for hybrid HDTV image sequence coding [1].

In this paper we will present fast recursive or moving windows algorithms for the calculation of local means using a diamond or hexagon or general polygonal shaped window. The rest of the paper is organized as follows: Section 2 describes three algorithms for local mean calculation in diamond shaped windows. Section 3 presents fast algorithms for other variations of diamond window shapes. Section 4 shows algorithms for hexagon shaped windows. Section 5 gives algorithm for general polygonal shaped windows. Section 6 shows the experimental results obtained using our fast algorithms applied to a variety of images. Section 7 gives concluding remarks.

2 Diamond Shaped Local Sum Calculation

In this section we will propose three algorithms for recursively obtaining the local sums in a diamond shaped window of an image. The size of the window is defined by its radius r. The size, or area, of the diamond shaped window with radius r is then $2r^2 + 2r + 1$.

2.1 Edge-Updating Algorithm

Assuming the local sum of a diamond window has been obtained at a particular position, when we slide the window horizontally to the right by one pixel to find the new sum, we only need to add in the pixel values from the leading edge and subtract out the pixel values from the trailing edge as shown in Fig. 1(b).

Note that the sums of those pixel values on diagonal lines can be obtained recursively using the moving window idea. The computational cost is only two additions and subtractions for each point on a particular line direction irrespective of the window length. Figure 1(b,c) shows the windowing process for the running sum operation on diagonal lines. The shaded triangle as shown in Fig. 1(b) indicates the pixel value to be added and the white triangle indicates the pixel value to be subtracted from the running sum. The length of the running window for the -45 degree line is $r + 1$. Figure 1(c) shows the case for the 45 degree lines. The length of the running window is r, i.e. one pixel less than that of the -45 degree line.

After obtaining the running sum on the two diagonal line directions, the updating process for the local sum of a diamond shaped window can be carried out. There will be four additions and subtractions for updating the diamond window sum based on the running sums of the diagonal pixels. Because there are two additions and subtractions when carrying out each of the four diagonal lines, the number of operations for diamond shaped window will be eight additions and subtractions. We call this algorithm for obtaining the local window sum the Edge-Updating (EU) algorithm.

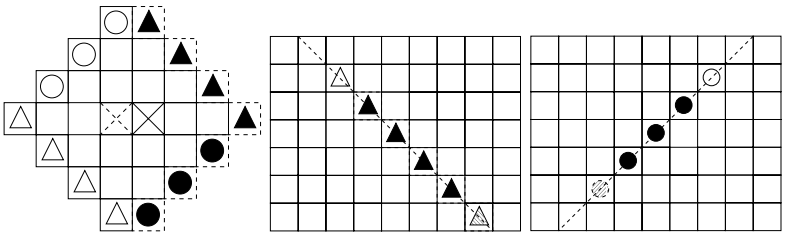


Fig. 1. (a) Diamond shaped window updating process using 45 and -45 degree lines. The dark disk and triangles are the image pixels coming into the diamond window and the hollow circle and triangles are the pixels leaving the window as the diamond window moves one pixel to the right. (b) The recursive updating process for diagonal lines. Only two additions and subtractions are needed for each window position. Along the -45 degree line, where the length of window along this line is $r + 1$; (c) The 45 degree line, where the length of window along this line is r.

2.2 Two-Grids Algorithm

As can be seen from Fig. 2(a), the pixels in the diamond window consist of the $(r + 1) \times (r + 1)$ black dots and the $r \times r$ white dots (or circles) interleaved together. The centers of these two different sized grids coincide with each other. We can therefore obtain the pixel sums for the diamond window by adding together the black pixels sums and the white pixels sums, i.e. adding together the pixel values on the two grids. We name this algorithm the Two-Grids (TG) algorithm.

For obtaining the sums of the white dots, we can adopt the separable principle by adding the pixels first along the 45 degree lines, and then along the -45 degree lines. Each pass only requires two additions and subtractions for each position. Similarly, for obtaining the sums of the black dots, we can also adopt the separable principle. Each pass only requires two additions and subtractions. Although the black and the white grids are interleaved, we need to obtain the local operations at each pixel position for both the black and the white pixels. The first pass for obtaining the black pixels sums along the 45 degree lines can be combined with the first pass for obtaining the white pixels sums of the same direction. That is, after obtaining the local pixel sums for the white pixels, the local sums for black pixels have already been obtained. The local window length for the black pixels $(r + 1)$ is only one pixel longer than that of the white pixels (r). Therefore there is no extra cost for obtaining the black pixel sums along the 45 degree lines except an assignment operation to store the needed values.

Therefore we have four passes for this algorithm. The first pass is the combination pass along the 45 degree line which includes the updating of the white pixels sums (two additions and subtraction for each pixel) and obtaining the black pixels sums from the white sums (no additional operation is needed except storing the needed sum). The second pass is along the -45 degree lines for the white pixels (two additions and subtractions for each pixel). The third pass is

along the -45 degree lines for the black pixels (two additions and subtraction for each pixel). Finally we need to add together the white pixel sums and the black pixel sums within the diamond window (one addition for each pixel). So the total addition and subtractions for each pixel on the image is seven for obtaining the local sum within a diamond shaped window.

2.3 Grid-and-Edges Algorithm

Another approach to obtain the sum in a diamond window is by obtaining the grid sum and adding in two edge sums as illustrated as in Fig. 2(b). Hence we name it the Grid-and-Edges (GnE) algorithm. The diamond window comprises of the grid with the black dots, the grid with the white dots which has one pixel offset upward with respect to the black grid, the black triangles, and the white triangles. The grid size used in this case is equal to r^2 for both the black and the white grids. For calculating each of the grid sums, one only needs to carry out the running sums using the separable principle along the 45 degree and the -45 degree lines. The running sum for each pixel along the -45 degree line for the black triangles with window length r has been obtained during the first pass for the grid sum calculation. One need another pass to obtain the running sum along the 45 degree lines with window length $r + 1$ for the white triangles. Therefore, the total computation complexity will be nine additions and subtractions per pixel on the image: 4 for obtaining the grid sums, 1 for adding the sum of the offset grid, 1 for adding the sum of the black triangles, 2 for obtaining the running sums along the 45 degree line, and 1 for adding the sum of the white triangles.

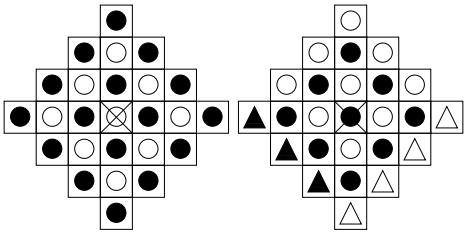


Fig. 2. (a) Diamond composed of two grids: the black grid with $(r+1)^2$ pixels, and the white grid with r^2 pixels. (b) Diamond sum by grid and edges. The sum of the pixels in the diamond window equals the sums of the black and white grids, black triangles, and white triangles.

3 Variations of Diamond Shaped Window

In this section we will describe some variants of the standard diamond window and propose algorithms for obtaining the local sum or mean for each of these windows.

3.1 Skewed Diamond or Parallelogram Shape

Figure 3(a,b) shows two diamond shaped windows which may be taken as the skewed rectangular windows. These shapes are actually parallelograms with one pair of edges aligned with image rows or columns. Two of the parameters are the lengths of the window edges, $k1$ and $k2$. The other two parameters, dx and dy, define the skewness of the window shape, where dx and dy are integer numbers which have no common divisors. The line representation of these two sides can be carry out using Bresenham lines [2] or periodic lines [5]. An example of a digital line segment with $dx = 2$ and $dy = -1$ is shown in Fig. 4(a). Figure 4(b) shows a skewed diamond window by putting together five of the line segments shown in Fig. 4(a).

We adopt the separable principle for obtaining the pixel sums within the skewed diamond window. The first pass is for obtaining the running sums along the digital lines represented by dx and dy. The second pass will carry out the running sums horizontally (for shapes similar to Fig. 3(a)) or vertically (for shapes similar to Fig. 3(b)) based on the values obtained from the previous pass. The algorithm only needs four addition and subtractions per pixel in the image. General parallelogram shapes where none of the sides is aligned with the image row or column can also be used. In this case, another pair of parameters describing the slope of the sides is needed.

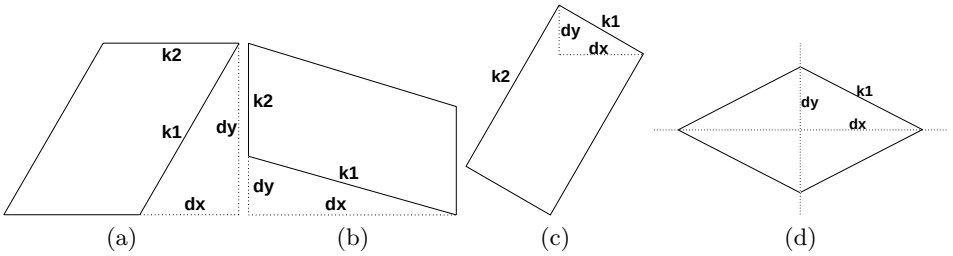


Fig. 3. (a) Window skewed in the horizontal direction. (b) Window skewed in the vertical direction. (c) Elongated diamond shapes. (d) Lozenged diamond shapes.

3.2 Long Diamond, Lozenge, Diamond with Angle

The local window can also be a rotated rectangle, or a long diamond shape as shown in Fig. 3(c). We need four parameters to define this window: $dx, dy, k1$ and $k2$. dx and dy gives the slope for the line with length $k1$. The lines in the orthogonal direction will have slope given by dy and -dx, and have length $k2$. We will then use the separable principle by carrying out the running sum along one direction first and then along the other direction. The computation cost will be four additions and subtractions per pixel in the image.

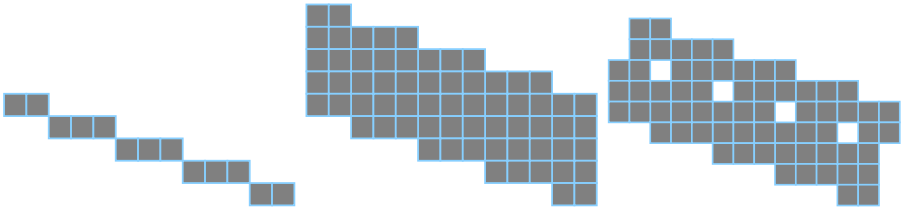


Fig. 4. (a) Digital line segment with $dx = 2$ and $dy = -1$. (b) Skewed window by stacking five line segments shown in (a) together. (c) Long window containing gap pixels. Digital lines along the long edge of the window have the same slope as in Fig 4(a). The slope on the orthogonal direction is defined by $dx = 1$ and $dy = 2$.

Another variant of the diamond window is called lozenge shaped, or rhombus shaped window, as shown in Fig. 3(d). Three parameters can define the window shape: dx, dy and $k1$. dx and dy gives the slope for the line on the top right side of the window. The slope for the line on the top left side of the window will be given by -dx and dy. Each side of the window has the same length $k1$. We can use the separable principle to obtain the window sums. The standard diamond can be rotated by a certain angle.

Depending on the parameters used for the diamond window, there will be situations when the window does not include all the points in the window as shown in Fig. 4(c). For those pixels not included in the computation, we call them the gap pixels. When not all the points in the window are included, the mean calculation will be approximate.

4 Hexagon Shaped Windows

A different window shape is the hexagon as illustrated in Fig. 5(a,b). We first need to obtain the running sums along the two line directions which are not horizontal. The exact slope for the two lines in Fig. 5(a) are $\tan(\pi/3)$ and $\tan(-\pi/3)$. But in a digital implementation, we need to use two integers dx and dy to represent the line slopes. The dx and dy values as illustrated in Fig. 5(c) can be calculated from the number of pixels of the top edge of the hexagon. If the number of pixels on the top edge is l (this number needs to be odd so that the shape of the hexagon can be symmetric), dx can take the integer value of $l/2$. dy can be calculated by using the integer part of $\sqrt{3}l/2$.

The window updating procedure will be similar to the process we used for the Edge-Updating algorithm described in Section 2.1. The window length for the line direction with brick texture is dy, and the window length for the line direction with gray shade is $dy - 1$. We need four additions and subtractions for updating the running sums for the non-horizontal edges. The cost for obtaining each of the running sum for the non horizontal edge has two additions and subtractions. Therefore we have eight (2 times 4) additions and subtractions for

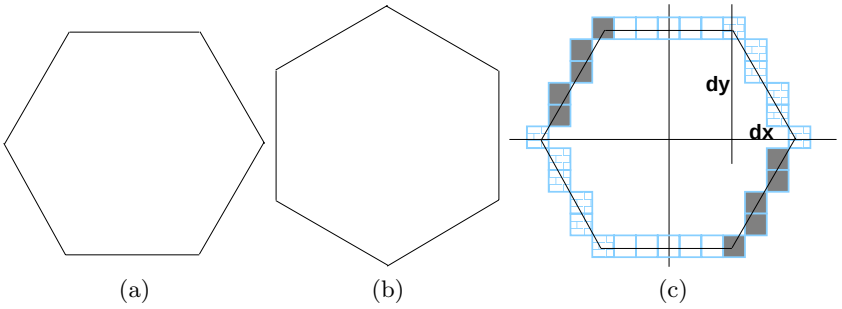


Fig. 5. (a) Hexagon shaped window 1. (b) Hexagon shaped window 2. (c) Hexagon shaped window updating.

each window position in the image. Similar process can be used to obtain the local sum for the hexagon shape shown in Fig. 5(b). In this case, we can move the hexagon window from top to bottom of the image rather than from left to right, and the slopes of the non-vertical edges are $\tan(\pi/6)$ and $\tan(-\pi/6)$.

Table 1 summarizes the number of addition and subtraction operations for each of the algorithms described in Sections 2-4. All the algorithms shown in the table have constant computation cost. One more division operation is needed to obtain the mean from the local sum if needed.

Table 1. The computation costs (numbers of additions and subtractions) for different algorithms. EU: Edge-Updating, TG: Two-Grids, GnE: Grid-and-Edges, S/L/L/A: the window shapes for skew, long, lozenge and diamond with angle, Hex: hexagon window.

	Diamond			**Other**	
Radius	**EU**	**TG**	**GnE**	**S/L/L/A**	**Hex**
$r = n$	8	7	9	4	8

5 General Polygonal Shaped Windows

Similar algorithms using edge updating can also be developed for general polygonal window shapes such as triangle, trapezoid, pentagon, heptagon, decagon.

We will now derive a general expression for the computational complexity for general polygonal shaped windows. Let n be the number of sides of a polygon, m be the number of sides which are aligned with the window's moving direction, p be the number of pair of polygon sides which are parallel to each other and with the same length and not aligned with the windows moving direction. For each edge of the polygon window, there are two additions and subtractions for obtaining the running sums. Because we do not need to calculate the running sums for the m sides which are aligned with the windows moving direction.

We then have $2(n-m)$ additions and subtractions for just obtaining running sums for edges. We also need to add in or subtract out these running sums to obtain the sums for the whole window. Then we have $3(n-m)$ additions and subtractions for a general polygonal window. If there are p pairs of edges which are in the same direction and having the same length (as for the opposite edges in the shape of hexagon or octagon windows), then redundant calculation of $2p$ for the running sums can be eliminated. The computation complexity, C, is:

$$C = 3(n-m) - 2p \tag{1}$$

The computational complexity is related to the number of sides, but is independent of the size of the polygonal windows. This formula holds for all the regular even sided polygons with opposite sides parallel to each other. Take the hexagon shape for example, the number of sides, n, is 6. The number of sides that are aligned with the windows moving direction, m, is 2. The number of pairs of parallel sides/edges having the same length, p, is 2. Therefore based on (1) the computation complexity is eight additions and subtractions for a hexagon shaped window.

For polygon windows with a set of parallel and equal length edges, if this set contains more than two edges, further computation redundancy exists and can be eliminated. A more general formula than that expressed in (1) for C is:

$$C = 3(n-m) - \sum_{i=0}^{p} 2(n_i - 1) \tag{2}$$

where $n_i > 1$ is the number of parallel edges with equal length within a set, and p is the number of such sets. If all the n_i's equal 2, Eq. (2) becomes Eq. (1).

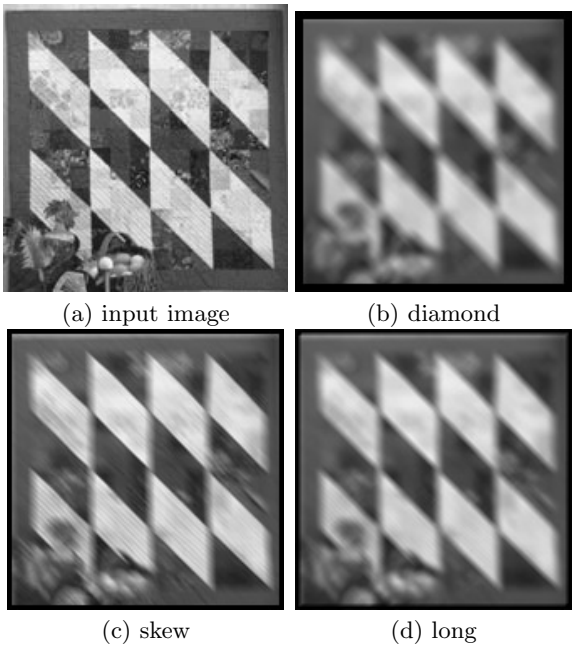
6 Experimental Results

This section shows some of the test results obtained using our new algorithms described in this paper. A variety of images have been tested, including synthetic images, and different types of real images. Figure 6 shows some obtained results using different algorithms developed in this paper. Figure 6(a) is the input image. Figure 6(b), (c), and (d) give the results obtained by using exact diamond, skew diamond, long diamond for local mean calculation. Figure 6(c) is essentially a linear averaging along the 45 degree line.

The speed of the algorithms were tested on a 1.7GHz Pentium 4 PC running Linux. The typical running time for the fast algorithms on a 1024×1024 image is in the order of one second. Table 2 gives some of the typical running times of different algorithms on different size of windows. The second column gives the running times for the local mean calculation in a diamond window using direct implementation. The running time is in accordance with the computation complexity of $2r^2+2r$, apart from that of the very small radius size due to implementation overheads. The last three columns clearly show that the computation costs of the EU, TG and the GnE algorithms are invariant to window sizes. The

slight reduction of computation time with the increase of window size is due to the boundary effect. Although the computation costs for the EU algorithm is one less than that of the TG algorithm as shown in Table 1, the memory storage and the number of runs going through the image are different. Therefore we can see the different running times for the EU and TG algorithms as shown in third and fourth columns of Table 2.

To achieve translation-invariant operation for general discrete lines, periodic lines can be used rather than Bresenham lines [6, 5, 4, 7].



(a) input image (b) diamond

(c) skew (d) long

Fig. 6. The mean filtering results using different shapes of windows. (a) input image; (b) using diamond window using the Edge-Updating algorithm ($r = 5$); (c) result using skew in x shaped ($dx = 1, dy = 1, k1 = 7, k2 = 1$); (d) result using long diamond ($dx = 1, dy = 1, k1 = 7, k2 = 3$).

7 Conclusions

We have developed several fast recursive algorithms for local sums calculation using diamond shaped window. The algorithms require seven or eight or nine additions and subtractions for each pixel on the image. The algorithms are also extended for a variety of other shaped diamond windows, such as long diamond, skewed diamond, and lozenge shapes. Similar algorithms are also developed for hexagon shaped windows. The computation for hexagon window only needs eight

Table 2. Running times of different algorithms with different window sizes (in seconds). Image size is 1024×1024.

Window size (r)	Drct	EU	TG	GnE
1	0.147 s	0.483 s	1.039 s	0.640 s
4	0.396 s	0.475 s	1.032 s	0.635 s
7	0.781 s	0.468 s	1.008 s	0.635 s
10	1.381 s	0.467 s	1.007 s	0.631 s
13	2.133 s	0.467 s	1.005 s	0.629 s

additions and subtractions for each pixel. Fast algorithms for general polygonal shaped windows operation are also developed. The computation complexity of all these algorithms are independent of the window size. Apart from image smoothing operation, the algorithms could also be used for stereo matching applications when different window shapes are needed.

Acknowledgement

The author thanks Dr Hugues Talbot and Dr Michael Buckley of CSIRO and the anonymous referees for their valuable comments and suggestions.

References

1. N. Baaziz and E. Dubois. Separable diamond-shaped filtering for hybrid HDTV image sequence coding. In *8th Workshop on Image and Multidimensional Signal Processing*, pages 162–163, Cannes, France, September 1993.
2. J. E. Bresenham. Algorithm for computer control of digital plotter. *IBM System Journal*, 4(1):25–30, 1965.
3. L.A. Ferrari and J. Sklansky. A fast recursive algorithm for binary-valued two dimensional filters. *Computer Vision, Graphics, and Image Processing*, 26(3):292–302, 1984.
4. C. Glasbey and R. Jones. Fast computation of moving average and related filters in octagonal windows. *Pattern Recognition Letters*, 18(6):555–565, June 1997.
5. R. Jones and P. Soille. Periodic lines: Definition, cascades, and application to granulometries. *Pattern Recognition Letters*, 17(10):1057–1063, September 1996.
6. P. Soille, E. Breen, and R. Jones. Recursive implementation of erosions and dilations along discrete lines at arbitrary angles. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 18(5):562–567, May 1996.
7. P. Soille and H. Talbot. Directional morphological filtering. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 23(11):1313–1329, November 2001.
8. Marca van Droogenbroeck and Hugues Talbot. Fast computation of morphological operations with arbitrary structuring elements. *Pattern Recognition Letters*, 17(14):1451–1460, 1996.
9. M. van Herk. A fast algorithm for local minimum and maximum filters on rectangular and octagonal kernels. *Pattern Recognition Letters*, 13(7):517–521, July 1992.
10. P. W. Verbeek, H. A. Vrooman, and L. J. van Vliet. Low-level image processing by max-min filters. *Signal Processing*, 15(3):249–258, October 1988.