PARALLEL ANALYSIS OF NON CONVEX SHAPES DIGITIZED ON THE HEXGONAL GRID

Gunilla Borgefors, Swedish Defence Research Establishment*
Gabriella Sanniti di Baja, Istituto di Cibernetica, C.N.R.°

* Box 1165, S-58111 Linköping, Sweden; Gunilla@lin.foa.se
° Via Toiano 6, I-80072 Arco Felice (Napoli), Italy; Gabry@gabry.na.cnr.it

ABSTRACT

Two parallel algorithms for convex polygonal covering of shapes digitized on the hexagonal grid are presented. Both algorithms require operations having a local support of a 6-pixel neighbourhood. The first algorithm computes a polygon with at most six sides, oriented along the six principal directions in the hexagonal grid. The second algorithm computes a nearly convex polygon, which fits the shape closely. The latter algorithm is less sensitive to pattern rotation and is thus more suited to practical applications. The concavity regions are obtained by computing the difference between the covering polygon and the shape itself. A concavity tree can then be built, by iteratively applying the polygonal covering algorithm to the concavity regions themselves. The nodes of the concavity tree can be labelled with parameters, related to geometrical features of the corresponding regions. The concavity tree is a useful tool for automatic visual inspection. It can also be used for identifying shapes.

INTRODUCTION

Shape is a key feature of digital patterns in binary images. It is necessary to describe and classify shapes, both for recognition and quality control applications. Long and thin shapes can be described using skeletons. Some shapes can be decomposed into a number of simple basic shapes. However, for some thick, complex shapes, neither method is suitable. The concavity analysis we present here can be an attractive alternative. The concavity regions are identified by subtracting the shape from a convex polygon covering it. We assume a massively parallel architecture using the hexagonal grid, where each processing element is connected to its six neighbours.

The concavity tree, that can be built from the shape when the concavity regions have been identified, gives a hierarchical description in terms of concavities and meta-concavities (i.e., the concavities of the concavities). The concavity tree can be used as a tool for automatic visual inspection, since it may allow identification of local defects in the silhouettes of industrial parts. It can also be used for identifying shapes in clothing and leather industries, where many irregular shapes have to be sorted before they are joined, [1].

There are many reasons why the hexagonal grid is preferable to the more common square grid:

- A digitized shape is more stable under rotation, as this grid has three principal directions, rather than only two.
- This grid is more suitable for rounded shapes.
- Each pixel has six "equal" neighbours, equal in the sense that the 4-neighbour/8-neighbour connectivity problem does not occur.
- In a parallel architecture each pixel has to be connected to only six neighbours, compared to eight in the square grid case.

The last reason has, in fact, resulted in a number of existing and projected massively parallel image processing architectures using the hexagonal grid, [7].

The main computation in deriving the concavity tree is the construction of the covering polygon. Polygonal covering can be obtained by computing the convex hull, or by filling shape concavities. The latter approach is computationally convenient when a massively parallel architecture is available. It can be implemented by an iterative algorithm which, at each iteration, identifies and ascribes to the shape, pixels from the background located within concavities.

The shape of the computed covering polygon depends on the local support of the performed operations. In [4] we suggested the use of a labelling technique, used together with a propagation process, that allows you to obtain a nearly convex covering polygon using only 3x3 operations. In [5] a hierarchical version of the algorithm is presented, together with a number of suggested concavity properties, that can be easily computed once the concavity regions are identified. These previous algorithms are all constructed for the square image grid.

Here we describe two algorithms for computing the covering polygon for the hexagonal grid. The first algorithm gives a convex covering polygon with at most six sides, aligned along the principal directions of the hexagonal grid. The second algorithm computes a nearly convex polygon, which fits the shape much better. Both algorithms employ operations with the same local support, i.e., the six neighbours of each pixel.

We also compute the concavity tree, and exemplify some concavity region properties that can be used for recognition and analysis.

FILLING CONCAVITIES

Let P be a digital shape digitized on the hexagonal grid. We assume that P consists of one connected component and does not touch the frame of the array. The pixels in P are called "black". The complement of P is denoted $\overline{P}$. The pixels in $\overline{P}$ are called "white".

The smallest neighbourhood in the hexagonal grid is shown in Fig. 1. Each pixel p has six neighbours n_i (i = 1, 6) sharing an edge with it. The set of pixels in the neighbourhood of p is denoted N(p).

Figure 1. The neighbourhood of a pixel p.

The border of the shape, B, is the set of white pixels having at least one black neighbour. For any border pixel p, $\Sigma_b(p)$ denotes the number of black pixels in N(p).

In Fig. 2 the border pixels of a shape are labelled with their corresponding Σ_b. It can be seen that pixels along a straight line segment, oriented along any of the six principal directions in the hexagonal grid, have $\Sigma_b=2$ (bottom right). Thus, the pixels bordering protrusions have $\Sigma_b<2$ and pixels placed within concavities have $\Sigma_b>2$.

To build the covering polygon, all the concavities of P have to be filled in. This can be done by iteratively changing the colour of all border pixels placed in concavities from white to black. These pixels are identified by their Σ_b value.

The first suggested algorithm for filling the concavities is the following:

Hexagonal covering

1. Compute Σ_b for all white pixels.
2. Change the colour of all pixels with $\Sigma_b \geq 3$.
3. Repeat from 1 until no changes occur.

Using this rule results in a covering polygon that has at most six sides, i.e., is a hexagon. Any pixel p of the outer border of the polygon is either located along a straight line segment aligned in any of the six principal directions ($\Sigma_b(p)=2$), or is placed at the meeting point between two such straight line segments ($\Sigma_b(p)=1$). An example showing the performance of this algorithm is illustrated in Fig. 3. The hexagonal grid is simulated by shifting the pixels in every even row by half a pixel.

Due to the small number of directions permitted for the sides of the covering polygon, it can not fit the shape closely. Furthermore, the covering polygon may drastically change if the shape is rotated. Thus, a different number of differently structured and shaped concavity regions would be computed for the same shape, depending on its orientation. For shape analysis, this may be a significant problem.

To get a better approximation of the convex hull of the shape, we must derive curvature information from a local support larger than a 6-pixel neighbourhood. It is obvious that pixels with $\Sigma_b \geq 4$ should always be changed and that pixels with $\Sigma_b \leq 2$ should never be changed. However, border pixels with $\Sigma_b=3$ can be located both along straight lines (not oriented along any of the principal directions) and in concavities, see right edge and bottom dent in Fig. 2, respectively. Thus, the pixels with $\Sigma_b=3$ should be changed only in some configurations.

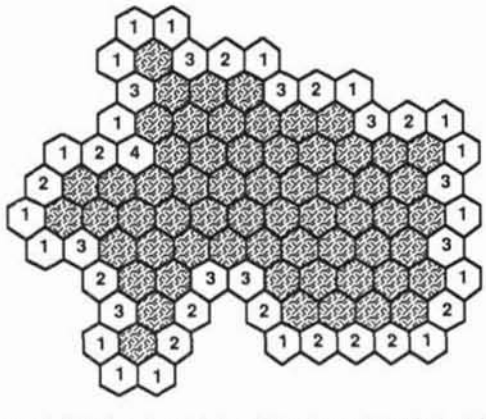

Figure 2. The border pixels of the shape labelled with the number of their black neighbours.

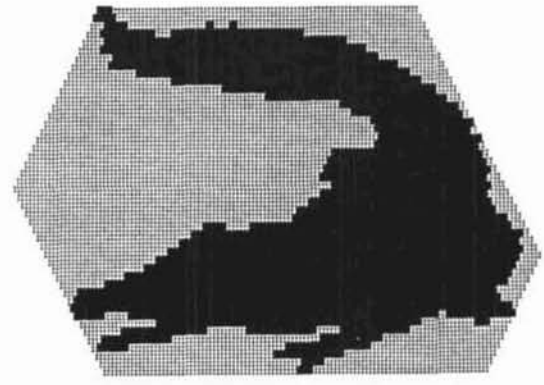

Figure 3. A shape, black, and its covering polygon, grey, as computed by the hexagonal covering algorithm.

The problem of computing the convex hull can be formulated as follows: any direction of the sides of the covering polygon must be allowed. If we allow sides with "runs" of two pixels, see the right edge of Fig. 2, we must not change pixels with $\Sigma_b(p)=3$ having two neighbours valued 1. As the neighbours of p get their values from *their* neighbours, checking the neighbours of p is equal to deriving curvature information from a neighbourhood with radius two, e.i., with 18 pixels.

If we also allow sides with runs of three pixels, see the upper right edge of Fig. 2, we need information from an even larger neighbourhood. The pixels with $\Sigma_b(p)=3$ in the straight line is located between a 1 and a 2. But this is also true for the two pixels with $\Sigma_b(p)=3$ in the lower left corner in Fig. 2 and these pixels are clearly located within a concavity. The difference is that in the latter case the neighbour n with $\Sigma_b(n)=2$ has, in its turn, a neighbour q with $\Sigma_b(q) \geq 2$. This is not true in the first case. Thus, if we also check neighbour of neighbours, we derive information from pixels up to three pixels from p along the outer border, i.e., we use a 36-pixel neighbourhood.

It is relatively easy to efficiently derive information from a 36-pixel neighbourhood. This is the reason why we chose this level of information propagation along the border. Thus, we will fill all concavities that can be identified using a 36-pixel neighbourhood. To simulate a 36-pixel neighbourhood by means of the 6-pixel neighbourhood N(p), each iteration of the concavity filling algorithm is split into three sub-iterations, denoted *Labelling*, *Propagation*, and *Filling*.

During the Labelling sub-iteration, the white pixels are labelled with the corresponding value of Σ_b. Labelling allows the simulation of an 18-pixel neighbourhood, as the value of a neighbour carries information from its neighbours.

Any border pixel labelled 3 that has *either* one neighbour labelled (at least) 3 *or* two neighbours labelled 2 should be changed to black. A pixel labelled 3 that has two neighbours labelled 1 remains white. The two rules above handle all possible configurations in which a pixel p with $\Sigma_b(p)=3$ can occur, except the case where p has one neighbour labelled 1 and one neighbour labelled 2. To resolve this case, the 36-pixel neighbourhood is needed. To be exact, we need to know the labels of the neighbour of p with $\Sigma_b(n)=2$. The Propagation sub-iteration will provide this information. Let $\Sigma_w(p)$ denote the *sum* of the

labels of the white neighbours of a border pixel p. Every white pixel p with $\Sigma_b(p)=2$ for which $\Sigma_w(p)\geq 5$, is marked in some suitable way (e.g. by multiplication with -1).

The Filling sub-iteration changes the colour of the appropriate border pixels.

The enhanced algorithm for filling the concavities becomes:

Polygonal covering algorithm

Labelling:

Label all white pixels with their Σ_b values.

Propagation:

Mark all pixels with $\Sigma_b = 2$ for which $\Sigma_w \geq 5$.

Filling:

All white pixels with $\Sigma_b \geq 4$ are changed to black.

All white pixels with $\Sigma_b = 3$ with

at least one pixel labelled ≥ 3 in N(p), or
two pixels labelled 2 in N(p), or
one marked pixel in N(p)
are changed to black.

Repeat from the Labelling step until no changes occur.

An example of the performance of this algorithm is given in Fig. 4, where the covering polygon of the same shape as in Fig. 3 has been computed. The covering polygon is now much closer to the convex hull of the shape. The increased number of permitted sides (with respect to the six sides of the first algorithm), makes this polygonal covering less sensitive to shape orientation.

The covering polygon computed by this algorithm is, unfortunately, not quite convex. "Weak" concavities can occur. The smallest angle that will not be filled occurs at the vertex between two allowed straight lines. The worst case occurs when a side consisting of runs of length two meets a side consisting of runs of length three. The label configuration becomes ...1-3-1-3-1-**3**-2-1-3-2-1..., where the meeting point is printed in boldface. This central pixel will not change colour. The angle at this point is $[180 - (\arctan(\sqrt{3}/2) - 30)]^\circ = 169^\circ\ 6'$. It should be noted that deriving information from an even larger neighbourhood will not improve this result, as the two directions meeting at this angle will still be allowed. The result may even become worse, as even more directions would be allowed.

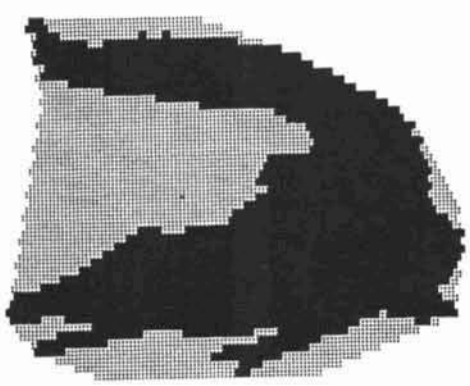

Figure 4. A shape, black, and its covering polygon, grey, as computed by the polygonal covering algorithm.

ANALYSING CONCAVITIES

The shape of the covering polygon itself can be used to describe the global shape of the shape. However, to achieve a more detailed description, the regions of the polygon, whose pixels do not belong to the shape, should be analysed.

First, the set difference between the polygon and the shape is computed, to detect the concavity regions. These regions can be labelled, using a parallel connected component labelling algorithm the concavity regions can be labelled differently. Thus, a number of different features can be computed for any one of the regions.

The *area* is computed as the number of pixels in the region.

The *perimeter* is computed as the number of pixels having a neighbour outside the region. The perimeter is composed of two sections, denoted the *external* perimeter and the *internal* perimeter, respectively. The external perimeter consists of the pixels having neighbours in the complement of the polygon. The internal perimeter is defined as the remaining pixels in the perimeter.

The *depth* of the region can be computed using a distance transform, that computes the distance between a feature set and all other pixels. Distance transforms for the hexagonal grid can be found in [2]. The simplest one is the *honeycomb* distance, where the distance between two pixels is equal to the number of steps in a path between them (cf. the city block and chessboard distances in the square grid). An efficient parallel algorithm for computing the distance transform is found in [3]. If the distance between the complement of the covering polygon and the pixels in the concavity region is computed, then the highest distance value in the region will be equal to its depth. Note that we must use a *constrained* distance transform, [6], where the original shape acts as a barrier.

A concavity region is depicted in Fig. 5. The honeycomb distance from the background has been computed. For this concavity we have: Area = 36, Perimeter = 29, External perimeter = 4, Internal perimeter = 25, and Depth = 10. These values give an easily computed characterization of this concavity, that can be used both for recognition and quality control.

Some concavity regions are characterized by an area equal to or only slightly different from the external section of the perimeter or, equivalently, by a very small depth. These regions can, in most cases, be disregarded. They can be interpreted as very shallow concavity regions filled in to remove contour noise, and they could possibly disappear if the shape is rotated.

Significant regions have the property that the internal section of the perimeter exceeds the external section and that they have some depth. Such concavities may be non-convex, and hence *structured*. A more sophisticated anal-

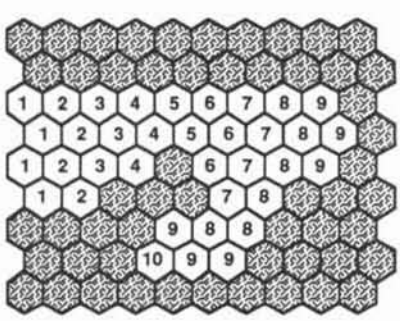

Figure 5. A concavity region with distance values from the background (at left).

ysis of this type of regions can be performed. The polygons covering the structures concavity regions themselves can be computed, using the same process as that applied to the original shape. In this way, for each concavity region, meta-concavities are identified (if they exist). Note that each concavity region has to be individually analysed. Otherwise, merging between pattern components separated by narrow channels could occur. The concavities and their meta-concavities can be linked, to create a meta-concavity tree, see [5]. This meta-concavity tree contains much information on the original shape.

CONCLUSION

The algorithms discussed in this paper can be used to compute an almost convex covering polygon for any non-convex shape digitized on the hexagonal grid. In the algorithm that yields the most closely fitting covering polygon, curvature information is derived from a 36-pixel neighbourhood, even though only a 6-pixel neighbourhood is used at each step. The concavity regions of the shape can be identified and subsequently analysed. The fact that the covering polygon fits the shape closely makes the computed concavities very stable under shape rotation. A number of easily computed concavity features are suggested. A concavity tree can also be built, by running the algorithm on the extracted concavities themselves. The algorithms presented are intended for massively parallel architectures.

Acknowledgment

This work has been partially supported by the "Progetto Finalizzato 'Sistemi Informatici e Calcolo Parallelo'" of the National Research Council of Italy.

REFERENCES

[1] B.G. Batchelor, "Using concavity trees for shape description," *Computers and Digital Techniques*, **2**, pp. 157-167, 1978.

[2] G. Borgefors, "Distance transformations on hexagonal grids," *Pattern Recognition Letters*, **9**, pp. 97-105, 1989.

[3] G. Borgefors, T. Hartmann, and S.L. Tanimoto, "Parallel distance transforms on pyramid machines: theory and implementation," *Signal Processing*, **21**, pp. 61-86, 1990.

[4] G. Borgefors and G. Sanniti di Baja, "Filling and analysing concavities of digital patterns parallelwise," in *Visual Form Analysis and Recognition*, C. Arcelli, L.P. Cordella, and G. Sanniti di Baja Eds., Plenum, New York, pp. 57-66, 1992.

[5] G. Borgefors and G. Sanniti di Baja, "Methods for hierarchical analysis of concavities," *Proc. 11th Int. Conf. Pattern Recognition*, The Hague, The Netherlands, **III**, pp. 171-175, 1992.

[6] J. Piper and E. Granum, "Computing distance transforms in convex and non-convex domains," *Pattern Recognition*, **20**, pp. 599-615, 1987.

[7] S.R. Sternberg, "An overview of image algebra and related architectures," *Integrated Technology for Parallel Image Processing*, S. Levialdi, Ed., Academic Press, London, pp. 79-100, 1985.