Line thinning by line following

Orit BARUCH
Koranix Ltd., Koranit, Israel

Received 1 May 1987

Abstract: A line following algorithm is presented as a means to perform line thinning on elongated figures. The method is faster than conventional thinning algorithms and not as sensitive to noise. It can be useful for other applications also.

Key words: Line following, edge following, line thinning, windows, image processing.

1. Introduction

The subject of object thinning has been of repeated interest for the past 3 decades and many algorithms for thinning have been suggested. This is due to two main reasons. The first is the importance of the thinning process in a large number of pattern recognition applications. The second reason, as has been stressed by Davies and Plummet in [1], is that ‘there is still no definitely good method for thinning’. Thinning is usually defined as the successive removal of the outermost layers of a figure (binary shape) until only a connected, unit width skeleton remains. (For example Davies and Plummer in [1] and Piper [5].) This definition includes not only the desired result of thinning (a unit width connected skeleton), but also the description of the process by which thinning is achieved (successive removal of the outermost layers). This binding of the ‘HOW’ with the ‘WHAT’ is wrong. Thinning can be achieved by other processes. Furthermore, this paper claims, that the seemingly obvious Successive layer Removal paradigm (SR) is the cause for substantial disadvantages of current thinning algorithms.

In Section 2, the SR concept is analyzed, to show how it leads to fundamental thinning problems, and a new concept is presented: The concept of line following (LF). In Section 3 a line following algorithm is presented. Section 4 contains a comparison of implementation results of thinning by SR and thinning by LF. Section 5 contains a summary and conclusions.

2. Successive removal versus line following

There are several properties which line thinning algorithms are expected to have in order to produce a skeleton which faithfully reflects the original figure. Before looking into these properties, note that for figures of arbitrary shape the meaning of ‘faithfully reflects’ is application dependent. Therefore, this paper is concerned only with elongated figures, such as characters.

2.1. The problem of locality

Line thinning algorithms are expected, on the one hand, to retain end points of lines or limbs. On the other hand, it is desired that they identify and ignore binary noise. Line ends and noise are both global properties of the picture. From a local point of view, as is the case with the SR concept, these expectations are conflicting. The local decisions lead in some cases to the elimination of whole limbs, and in others to the appearance of false limbs.

Another example of a required property is main-
taining connectedness. Again, this is a global property of the picture. It is obvious that the locality of the SR concept imposes inherent problems, which lead to noise sensitive algorithms and to distortions in the resulting skeleton.

2.2. The problem of processing time

The various thinning algorithms try to cope with the problem of locality by being over-cautious in removing points, and thus requiring many passes to complete the task. This leads us to the second inherent problem of the SR concept: time complexity. The SR concept means: examining each picture element in turn, and if it is set, examining all of its neighbours (at least in a 3 × 3 window) to decide on removal. This process is repeated until the resulting skeleton is achieved. The number of iterations depends on the thickness of the original figures in the picture, and on the specific algorithm. As has been noted above there is a trade off between the number of iterations and the quality of the final result.

2.3. A different approach

How would we, humans, perform line thinning? The most natural way is by following the line with our eyes, and marking the approximate center between the line edges. Seeing the global picture enables us to ignore noise and identify easily line ends. While following, we easily maintain connectedness. Of course, we need only one iteration to accomplish the task. Why not make the computer do the same thing?

The next section introduces a line following algorithm.

3. A line following algorithm (LF)

3.1. Presentation

The algorithm is performed on a binary image matrix. The matrix is raster scanned exactly once. At the end of the scanning, a target matrix contains the resulting skeleton. During the scan, the line follower is called to follow each encountered line. While following, the line follower erases the line in the input matrix and creates the skeleton in the target matrix. When the follower identifies branching of the line, it calls itself recursively to follow each of the branches. The line following is based on two pointers and a dynamic window:

LP – a pointer to follow the left edge of a line.
RP – a pointer to follow the right one.
W – a rectangular window whose sides are parallel to the sides of the matrix.

The line following is done always within the pre-defined current window.

Initially the pointers are set by the scanner to the location of an encountered set pixel, which is assumed to reside on some line figure.

The line follower performs the following procedure:

- First, the window W is set to surround the pointers. The window W is created so that given the locations pointed to by the pointers LP and RP, each of the window sides is set at a distance d from the location which is the nearest to it. The distance of a point to a line is measured as usual by the length (in pixels) of the normal from the point to that line. The value of d is set arbitrarily to 2. The influence of the value d is discussed in the comments in subsection 3.2.
- Second the current window is labeled to enable easy discrimination between the line to be followed, and other segments that may fall within the range of the window.
- Next, the edges of the line are followed, using the pointers LP and RP.

We can divide the line following process into two cases, depending on the relationship between the current window and the current line segment:

(a) In the direction of the following, the line exits the boundary of the current window and thus both edges of the line intersect at some point with the perimeter of the window. This case is illustrated in Figure 1a.

Note that the above observation remains true also when the line splits inside the current window (Figure 1c).

(b) End_of_line case: In the direction of the following, the line ends inside the current window and
thus in the direction of the following, none of its edges intersect with the perimeter of the window. Instead, in this case, the edges of the line meet each other. This situation is illustrated in Figure 1b.

When the second case occurs, the line follower identifies an end of line.

If the line follower identifies an intersection of the line with the perimeter of the window, it examines whether the line branches inside the window. The intersection point on the left edge is marked by LLP and the intersection point on the right edge by RRP. The perimeter of the window is then traced between LLP and RRP. If there is no branching, then all pixels on the trace will be set (see Figure 1a). Therefore if a white pixel is encountered the follower determines a branch. In this case, when going on the perimeter from LLP to RRP the last pixel, which is set is on the right edge of the leftmost branch. It is marked as LRP. The first pixel set after the gap of one or more contiguous white pixels is on the leftmost edge of the right branch(es). It is marked as RLP (see Figure 1c).

Now, repeatedly:

(a) The pointers LP and RP are set to the left and right intersection points (respectively) of the leftmost (or only) branch which has not yet been followed, and then the line following process is performed for that branch.

Then:

(b) The right branch, if any, is tested for branching.

This is done until an end of line is encountered on all branches. Thus, all branches of the line which evolve from the current window $W$, are followed from leftmost to rightmost successively.

Before following the last (or only) branch, the portion of the line, which resides exclusively inside $W$, is erased.

The skeleton is defined as the unit width line connecting the centers of successive windows which are built during the line-following process.

Appendix A contains the line following algorithm in a Pascal-like notation.

3.2. Comments

The window $W$ is defined always in such a way
that it surrounds the current location of the tracing pointers, thus surrounds the intersection points of the followed line with the previous window. Therefore, it is guaranteed that the window embraces the line and grows or shrinks according to the width of the line at the location of the window. It is also guaranteed that the center of successive windows advances in the direction of the line following. The process can be viewed as a rubber-ring which moves along the line and while doing so changes its form dynamically.

Figure 2 illustrates several successive windows created by the line following process. It shows how the windows are adjusted to fit the line and how this enables the process to be insensitive to local noise.

Except for the followed line, the windows are determined by the parameter $d$ which is a controllable factor. Thus the value of $d$ influences the speed and accuracy of the algorithm. The bigger $d$ is the bigger the windows are and the faster the algorithm is. But since the skeleton is defined to be the line connecting centers of successive windows, bigger windows may lead to skeletons which are not on the medial axis of the line. Yet, even for the small value $d = 2$, the algorithm proved to be very fast (as can be seen from the examples).

The line following algorithm was introduced here as a tool for thinning line shaped figures, but thinning is only one of the possible outputs of line following. In general, while following many global properties can be identified. Here are some examples:

- Line following can be used as a pre-processor for identifying noise.
- Line following can be used for recognizing the global shape of the followed figure. It can be used to identify irregularities in the shape, and the location of extreme directional changes.

4. Comparative implementational results

The LF algorithm was implemented in Pascal on an IBM-AT. A typical thinning algorithm based on the SR concept, was implemented too in the same language and on the same machine. It is an algorithm by Ben-Yosef and Feigin [2] which works on a $5 \times 5$ window. The algorithms were used to perform thinning on images of elongated figures such as characters and fingerprints. The results are compared in Table 1 below.

<table>
<thead>
<tr>
<th>reference figure</th>
<th>matrix size</th>
<th>SR processing time</th>
<th>LF processing time</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Fig. 3</td>
<td>$268 \times 128$</td>
<td>31 sec</td>
<td>5 sec</td>
</tr>
<tr>
<td>2. Fig. 4</td>
<td>$208 \times 128$</td>
<td>38 sec</td>
<td>7 sec</td>
</tr>
<tr>
<td>3. Fig. 5</td>
<td>$512 \times 128$</td>
<td>92 sec</td>
<td>5 sec</td>
</tr>
</tbody>
</table>
Table 1 contains the size of the processed matrix, the processing time and a reference to the figures which show the output results. The results show that the LF algorithm is much faster than the conventional SR algorithm. The improvement in computation time is especially significant for images of 'thick' objects (Figure 5). When working on a smooth picture of fine quality the resulting skeletons from the SR and LF algorithms are almost identical (Figure 3), but when the picture is noisy, there is a significant improvement in the quality of the skeleton in the case of the LF algorithm (Figure 4).

5. Summary and conclusions

This paper analyzes current thinning algorithms and shows that they are based on a concept (successive removal) which is the cause for fundamental thinning problems. These problems are summarized under two main categories: the problem of locality - which is the inability or difficulty to recognize global properties of the processed picture, and the problem of the large processing time.

Then the paper presents a new concept - line thinning by line following, and a line following al-
algorithm. The new algorithm solves some of the problems of the conventional thinning algorithms. For binary images it produces skeletons which are more accurate in the sense of not being influenced by noise, and it is much faster than the conventional thinning algorithms.

In addition the line following algorithm possesses the generally useful feature of recognizing global information and can therefore be adapted by many applications other than line thinning.

Acknowledgement

I would like to thank Prof. M.H. Loew for his useful suggestions to the presentation of this paper.

Appendix A

The line following algorithm in a Pascal-like notation

Procedure FOLLOW (LP, RP: pointer to pixel);
(* LP, RP: pointers to Left (Right) edge resp. *)
VAR exit: boolean;
    W: window record;
exit := false;
repeat
    BUILD_WINDOW(LP,RP,W); (* define current window to surround LP and RP *)
    LABEL_WINDOW(W); (* label the segments inside window *)
if LP and RP intersect with perimeter of window W then
    begin
        while line branches inside W do
        begin (* LRP points to right edge of leftmost branch, RLP points to left edge of right branch(es) *)
            FOLLOW(LP,LRP); (* call follower recursively to follow leftmost branch *)
            LP := RLP;
        end;
    (* last/only branch *)
    ERASE_LINE(W); (* clear the line exclusively in window *)
    FOLLOW(LP,RP); (* follow last branch *)
    end
    else (* end_of_line encountered *)
        exit := true (* terminate *)
    until exit;
end;

References