

# A Thinning Algorithm by Contour Generation

Paul C K Kwok

Department of Computer Science  
The University of Calgary  
2500 University Drive NW  
Calgary, Canada T2N 1N4

## Abstract

A fast serial algorithm for thinning binary images is presented. The boundaries of 2-D objects are chain coded and an iterative procedure is set up where the boundary points in the image are visited one after another in every iteration. A boundary point is deleted if it is not a break point or an end point. A section of the new boundary is created and contains all of the break points, end points and points exposed to the background after the previous boundary has been eroded. As the new boundary is generated it is checked for break points. At the end of an iteration, the chain code for the new boundary is in place for processing in the next iteration. The iteration terminates when all the points in the new contour are either break points or end points. The last contour generated becomes the skeleton. For thick objects, the method is 19 times faster than parallel algorithms and 6 times faster than conventional contour tracing thinning algorithms. This method is more efficient than other serial algorithms because it is faster to generate a new contour than to remove the current one and to trace a new one in the next iteration.

## 1. Introduction.

Thinning is an important preprocessing step for many image analysis operations such as finger print recognition [1], optical character recognition [2] and also in biomedical systems [3].

Thinning usually involves the removal of points or layers of outline from a pattern until all the lines or curves are of unit width, or a single pixel wide [4] [5]. The resulting set of lines or curves is called the skeleton of the object. Many algorithms are available. An analogue technique [6] involves the generation of a medial line in which every point is equi-distant from at least two points on the edge of the pattern. In the case of a digital approach, a two-dimensional array of pixels are considered. Constraints are included so that contour pixels of the skeleton either touch or are on the medial line [7-10]. When contour pixels touch the medial line, the skeleton will be two-pixel wide [3][8][9] and a postprocessing step is required to thin the skeleton to unitary width.

Most thinning algorithms are iterative. In an iteration (or pass), the edge points are examined against a set of criteria to decide whether the edge point should be removed or not. Rosenfeld *et al* [11] [12] classified thinning algorithms as being parallel or sequential. With a parallel algorithm, only the result from the previous iteration takes part in the decision to remove a point in the current iteration. This is therefore suitable for processing by parallel hardware such as an array processor. A sequential algorithm, on the other hand, makes use of the result from the previous pass and those results obtained so far in the current pass to process the current pixel. Thus at any point in an

iteration, a number of pixels has already been processed. These results can be made use of immediately to process the next pixel. It is generally reckoned that a sequential algorithm will be faster than a parallel algorithm implemented on a serial computer [11].

As the resolution of both the digitization equipment, frame buffers and displays [13] continues to increase, the time complexity of algorithms that require the examination of all the pixels in a bitmap during every iteration will also increase significantly.

In the next section, the terminologies to be used will be explained and in the two sections to follow, two parallel algorithms and a serial algorithm will be described briefly and later on used as a basis for comparison with a new serial method proposed in this paper.

## 2. The Essential Characteristics of a Skeleton.

Consider a binary image described by a 2-D array of pixels. The object, which forms the foreground  $Q$  of the image is represented by a set of "dark points" while the background corresponds to a set of "white points". For a given pixel  $p$  there are eight neighbors  $n_0, n_1, \dots, n_7$ , with the subscript denoting the direction of the neighbor from  $p$ , with respect to the  $x$ -axis (Figure 1). Thus for  $n_i$ , the direction is  $i \times 45$  degrees.  $n_0, n_2, n_4$  and  $n_6$  are called the D-neighbors, which are accessible from  $p$  by moving in a horizontal or vertical direction. The other neighbors,  $n_1, n_3, n_5$  and  $n_7$  are called the I-neighbors, which are accessible from  $p$  by moving along any of the 45-degree lines. If  $p$  is a dark point and one of its eight neighbors  $n_i$  is also dark,  $p$  and  $n_i$  is said to be 8-connected. On the other hand, if  $p$  is dark and one of its four D-neighbors  $n_{2i}$  is also dark,  $p$  and  $n_{2i}$  is said to be 4-connected.

If  $p_0$  and  $p_m$  are two (dark) points that belong to the same object, there exists a path, describable as a chain of dark points  $p_0, p_1, \dots, p_m$ , and each consecutive pair of pixels,  $p_i$  and  $p_{i+1}$  ( $i = 0, 1, \dots, m-1$ ), are neighbors of each other. If all the eight neighbors are considered,  $p_0$  and  $p_m$  are said to be 8-connected. If the D-neighbors only are considered,  $p_0$  and  $p_m$  are said to be 4-connected.

An object is said to be 8-connected if all pairs of points in the object are 8-connected. An object is said to be 4-connected if all pairs of points in the object are 4-connected.

Given the above framework, the essential characteristics of a skeleton can be summarized as follows:

- (a) Connectivity should be preserved. If the object is connected, the resulting skeleton should also be connected. If the background is connected, the background, as a result of thinning, should also be connected. On the other hand if the background is not connected, the background after thinning should not be connected. In most cases [8], it is required that 8-connectivity should be preserved for the foreground, while 4-connectivity should be preserved for the background. This means that the 2-pixel-wide 45-degree line shown in figure 2a should be further thinned down to a 1-pixel-wide line (figure 2b). A dark point which, if removed, disconnects an object is called a break point. Thus a break point test is incorporated in many thinning algorithms in order to preserve connectivity.
- (b) Excessive erosion should be prevented. The end points of a skeleton should be detected as soon as possible so that the length of a line or curve which represent a true feature of the

object is not shortened excessively.

- (c) It should be immune to small perturbations in the outline of an object. Noise, or small convexities, which do not belong to a skeleton, will very often result in a tail after thinning. The length of these tails should be minimized.

In general, (b) and (c) may be conflicting criteria and constraints will sometimes be added to obtain a compromise between the two.

### 3. Parallel Algorithms.

Naccache and Shinghal [14] reviewed and compared the speed of 14 parallel thinning algorithms [2, 10, 12, 15-19]. These algorithms used the same approach of visiting all the pixels in the bit-map to identify the dark points. The dark points are then classified into edge points and non-edge points. Only the edge points need to be considered. Tests are conducted on their 8-neighbors to determine whether they are break points, end points or non-safe points. The non-safe points are then removed from the pattern at the end of the pass. The break and end points are collectively known as safe points and should not be removed.

All the fourteen algorithms are similar but different in the way they handle break points and end points. The safe-point-thinning-algorithm SPTA was proposed [14] and from experimental results, Naccache and Shinghal concluded that SPTA was the fastest method among the fourteen algorithms tested. An edge point can be further classified as a right, top, left or bottom edge point (or a combination of the four types), depending on which of the four D-neighbors,  $n_0$ ,  $n_2$ ,  $n_4$  or  $n_6$  respectively (or a combination of these) are white points. In SPTA, the safe-point test is conducted by examining a set of 4 windows for a given edge point situation, right, top, left or bottom. A decision tree can be constructed to minimize the number of neighbors that need to be examined. A labelling scheme was used to tag a safe point and a non-safe point so that at the end of a pass, the non-safe points do not have to be eliminated explicitly from the bitmap. This also enabled the reconstruction of the original pattern from the skeleton.

Zhang and Suen [20] proposed a slightly different parallel algorithm. The break point and end point tests consist of (i) an examination of the number of white/dark transitions when the 8-neighbors  $n_0, n_1, \dots, n_7, n_0$ , are traversed in that order, the number should be equal to 1, (ii) a count of the number of dark neighbors so that if a pixel is a candidate for removal, this number should be between 2 and 6 for the Zhang/Suen algorithm and between 3 and 6 for the Lu/Wang algorithm [21], and (iii) two other tests on the D-neighbors to determine the edge conditions.

In some parallel algorithms, a 2-pixel wide line will be completely removed because at the beginning of the pass, points on both sides of the line will not break the connectivity of the pattern if they are examined independently. If both sides are examined in parallel using the results from the previous pass, they will be removed simultaneously [21] because the result of removing one side of a line is unknown to the other side during the same pass. This is therefore a mutual exclusion problem, well-known in concurrent programming [22], when several parallel processes are sharing the same memory. In thinning algorithms, when a process examines a certain pixel, it should have exclusive use of that pixel and also its eight neighbors. It should prevent other processes from gaining access to those nine pixels. In parallel thinning algorithms implemented on a parallel architecture or simulated on a serial architecture, it is often not the case. While a pixel A is examined by a process, the process

associated with its neighbor is also examining pixel A, thus violating mutual exclusion.

The solution is to divide a pass into 4 subiterations [12] [18], each responsible for the removal of the top, bottom, left or right edge points. An alternative is to combine the top and left subiteration; bottom and right subiterations. This will reduce the number of subiterations to 2. However, in many instances, a 2-pixel-wide, 45-degree line (figure 2a) may be completely removed for the 2-subiteration case [21]. A slightly different variation of the method combines the left and right subiterations, top and bottom subiterations [14]. In this case, a 2-pixel-wide, horizontal or vertical line will be completely removed. The alternative is to make the edge information of the neighbors available to the current pixel [23].

The time complexity of parallel algorithms consists of three components:

- (i) In every pass and in every subiteration, every pixel in the bitmap has to be examined once to identify the dark pixels. The number of operations is proportional to the area of the bitmap.
- (ii) Every dark pixel has to be examined for edge points. The number of operations is proportional to the area of the objects in every pass.
- (iii) The number of passes is related to the “thickness” of the object.

The total number of operations in (i) is therefore a product of the number of passes, the number of subiterations per pass and the size of the bitmap. The total number of operations in (ii) is a sum of the sizes of the objects in all the subiterations and in all the passes. Even though the size of the objects reduces progressively after each pass, the total number of operations to determine whether a pixel is an edge point or not is still very substantial. The complexity of (ii) increases sharply when the resolution of the image or the thickness of the objects increases.

#### 4. Serial Algorithms.

Sequential technique is an alternative to parallel methods. Less memory is required in sequential algorithms [7] [17]. However, as memory cost continues to fall, memory usage is no longer an issue. Similar to parallel algorithms, a sequential algorithm also examines every pixels in the bitmap to distinguish the foreground from the background. Thus time complexity still depends on the size of the bitmap.

A significant reduction in time complexity can be achieved by examining only those points which belongs to the outline of an object. Serial algorithms, and among them, the contour tracing technique was introduced to deal with nearly thinned objects [9] or thick objects [3][8]. In this case the contour describing the edge of an object is traced in every iteration. The contour is a sequence or chain of edge points  $p_0, p_1, \dots, p_n$ , where  $p_0 \equiv p_n$  ( $p_n$  is redundant and can be eliminated from the chain). Where multiple disjointed objects are involved or where holes exist in an object, a set of chains will evolve. For a pixel  $p_i$ , the two pixels just prior to or following it in the chain, namely  $p_{i-1}$  and  $p_{i+1}$  respectively are called the C-neighbors of  $p_i$  (with  $p_{-1} \equiv p_{n-1}$ ).

The sequence of pixels is usually represented by a chain code [24] [25], which is a sequence of directions  $dir_0, dir_1, \dots, dir_{n-1}$ , pointing to the next point in the sequence. For 8-connected contours,  $dir_i$  is in the range between 0 and 7 inclusive, representing the 8 directions as shown in figure 1. Another variation is to record the change in direction, instead of the absolute angle with respect to the  $x$ -axis. The set of chains, together with the coordinates of the starting points  $p_0$  and a vector describing the direction of the interior, will completely define the bitmap, i.e., the original bitmap can be completely recovered with these information. In fact, thinning algorithms have been related to filling algorithms [26].

In Contour Tracing Thinning algorithms [9], the term “multiple pixel” was defined to describe edge points that satisfies one or more of the following:

- ( $\alpha$ ) It is traversed more than once when tracing the set of contours.
- ( $\beta$ ) It has no neighbors in the interior of Q.
- ( $\gamma$ ) It has at least one D-neighbor which belongs to the contour, but which is not one of its C-neighbors.

After the contour has been traced and the sequence of pixels examined to determine whether it is multiple or not, the contour is removed. Multiple pixels are skeletal pixels and are copied to a bitmap where the skeleton is formed progressively. To ensure connectivity, pixels which are neighbors to skeletal pixels discovered in a previous pass are also identified as skeletal pixels. In the next iteration, the new contour is traced and the operation repeats until all the dark points are removed.

Arcelli [8] presented a variation of the contour tracing thinning algorithm. After obtaining the set of multiple pixels, only the non-multiple pixels belonging to the contour are removed.

In both of these two cases, all the points on a 2-pixel-wide line (figures 2a and 2c) touch the medial line. They are multiple and will therefore precipitate into the skeleton. A postprocessing step is necessary to thin the skeleton down to unit width (figures 2b and 2d).

## 5. Contour Generation.

It is evident that the need to go to multiple subiterations in a parallel algorithm and the possibility of a 2-pixel-wide line in contour tracing can be attributed to the problem of “mutual exclusion”. In the case of a parallel algorithm, the processing of a pixel is on the basis on its previous state so that when pixels are considered in parallel, all the pixels are removed.

When the problem is viewed from another angle, one can see that in both parallel and contour tracing techniques, pixels are removed from the contours without knowledge of what is going to remain in the object. The result is that either all the pixels will have been removed or to prevent this from happening, a thick curve (figures 8d and 9d) is going to remain after the final iteration.

The solution is therefore to take into account of the results obtained so far for processing the current pixel. If a pixel were to be removed, the new contour which will be exposed to the background can be computed. Thus when the current contour is traversed, a section of the new contour is generated for every pixel visited. The section is checked for break points and this information is available when subsequent pixels in the chain or those on the next chains are visited. At the end of the iteration, a new contour will be available for the next iteration without the need to remove the old one explicitly.

At any time, the algorithm will have complete knowledge of what is remained of the object when the current contour is removed. Thinning is completed when there are no non-safe points in any of the new contours. The algorithm, together with an implementation, is described in more detail in the following section.

## 6. Thinning by Contour Generation.

Before all the iterations, the given bitmap is recoded into chain codes. A chain code is generated for every closed contour describing the outlines of the object. This is done only once and contour tracing is not required within an iteration.

Many contour tracing algorithms are available [10] [27]. An efficient method has been adopted and is described in Appendix I. The resulting set of chains represent the outer-most layer of the objects. Also available after the trace are the coordinates of the heads of the chains and a parameter describing the direction of the interior. This parameter, *dir\_int* is equal to +1 for a counterclockwise scan of the exterior of an object or for a clockwise scan of an interior hole of the object. It is equal to -1 for a clockwise scan of the exterior or for a counterclockwise scan of an interior hole of the an object. The chains, the coordinates of the heads of chains and *dir\_int* will define the bitmap completely. This is therefore a recoding step, after which the original bitmap is no longer needed. Other contour following thinning algorithms that do not require knowledge of the direction of the interior will need the original bitmap for subsequent contour tracing. In that case the same algorithm cannot be used as a filling algorithm.

With the chain codes, the outline is plotted on a bitmap *S*. Every pixel visited will have its value incremented. If *S* begins with all the pixels having a value of 0, a pixel visited more than once will have a value greater than or equal to 2 and is therefore a break point (condition  $\alpha$ ). When the operation completes, the skeleton is formed in *S*. A chain code describing the skeleton is also available.

After plotting the first contour on *S*, the algorithm goes through a number of iterations. The iteration terminates for a particular contour when there are no more non-safe points in that contour. Thus the number of passes required for one contour may differ from another.

At any point in an iteration, a section of the new contour is to be generated to correspond to the pixel  $p_i$  under consideration. Two direction vectors,  $dir_{i-1}$  and  $dir_i$  are maintained. Arithmetic involving *dir* is understood to be of modulo-8. The result always lies between 0 and 7 inclusive. The x-y coordinates of  $p_i$  is updated from the x-y coordinates of  $p_{i-1}$  using  $dir_{i-1}$ . The coordinates are represented by a pointer to a 1-D array describing the bitmap. A look-up table is used that gives the offsets needed for each of the 8 directions. A flag  $p_{i-1\_is\_safe\_point}$  is kept to show whether the

previous pixel is a safe point or not. The current pixel  $p_i$  is checked for safe point by examining its value in the bitmap S. With the above set of information, a unique section of the new contour can be generated. The new contour will include all the safe points uncovered so far and some dark points which are neighbors of the current outline.

## 7. The starting and terminal points for the section.

Sections of the new boundary are created as pixels in the current chain are visited serially. One section is generated for each successive pixel in the chain and it always joins to the section generated as a result of processing the last pixel.

Different methods can be used to define the starting and ending points of the section of the new contour. The scheme shown in figure 3 is adopted. If the previous point  $p_{i-1}$  is a safe point, the previous section of the new chain terminates at  $p_{i-1}$ . If it is not, the previous section terminates at a common neighbor of  $p_{i-1}$  and  $p_i$ , found by rotating the line joining  $p_{i-1}$  and  $p_i$  by 45 degrees into the interior using  $p_i$  as the centre of rotation. The direction of rotation is clockwise if  $dir\_int = +1$  and counterclockwise if  $dir\_int = -1$ . This point must be a dark point, otherwise  $p_{i-1}$  would have been a break point. The terminal point for the previous section is the starting point of the new section.

If the current point  $p_i$  is a safe point, the new section will terminate at  $p_i$ . If it is not, it will terminate at a common neighbor of  $p_i$  and  $p_{i+1}$ , found by rotating the line joining  $p_i$  and  $p_{i+1}$  by 45 degrees into the interior, using  $p_{i+1}$  as the centre of rotation. Again this point must be a dark point, otherwise  $p_i$  would have been a break point.

In the implementation, no rotation is necessary. The terminal point of the previous section is evaluated on a case-by-case basis as shown in figures 5 and 6.

## 8. Generating a section of the new contour

There are four different cases that need to be considered. They are shown in figure 4. If both  $p_i$  and  $p_{i-1}$  are both safe points, the new section consists of one vector, the direction being  $dir_{i-1}$ . This vector links  $p_{i-1}$  to  $p_i$ . This case will be predominant towards the final iterations as more and more safe points are uncovered.

If  $p_i$  is a safe point while  $p_{i-1}$  is not, the new section consists of one vector which links the starting point to  $p_i$ . The direction of this vector is  $(dir_{i-1} - dir\_int)$ .

If  $p_i$  is not a safe point, the difference  $(dir_i - dir_{i-1})$  is computed and the new section is generated on a case by case basis as shown in figures 5 and 6. Figure 5 is for the case where  $p_{i-1}$  is a safe point: the new section begins at  $p_{i-1}$ . Figure 6 is for the case where  $p_{i-1}$  is not a safe point: the new section begins at the starting point as determined in the last section. It can be seen that the starting and terminal points are neighbors of  $p_i$ . If one starts from the starting point and rotates by  $-45 \times dir\_int$  degree increments about  $p_i$ , visiting the D and I neighbors of  $p_i$  until one reaches the terminal pixel, all the points visited will be dark, otherwise  $p_i$  will be a safe point and contradicts with the safe point test just conducted on  $p_i$ . The safe point test performed on the bitmap S was up-to-date

because it includes the results of processing of the previous pixels in the same chain, and of all the pixels belonging to the previous chains. Thus connectivity is guaranteed even for objects with holes in it, in which case, two different chains describe the outer and inner edges, e.g. as in the letter "O" of the alphabet. The set of pixels just visited are pixels of the section to be generated and hence belong to the new contour.

For figure 5, where  $p_{i-1}$  is a break point, the new section will consist of 0 to 3 elements. In the case where  $(dir_i - dir_{i-1}) \equiv 4$ ,  $p_i$  is an end point and will be flagged as a safe point automatically.

In Figure 6, where  $p_{i-1}$  is not a break point, the new section will consist of 0 to 2 elements. In the case where  $(dir_i - dir_{i-1}) \equiv 3$  or 5, the starting point of the new section is  $p_{i+1}$  and it is a break point.  $p_i$  therefore also belongs to the new section and it is an end point. Again  $p_i$  is flagged as a safe point automatically. The new section consists of one element. The value of this element in the chain is  $dir_i + 4$ , i.e., in the opposite direction of  $dir_i$ .

When  $(dir_i - dir_{i-1}) \equiv 2$  or 6 and  $dir_i$  is even, the present contour is going through a convexity. This may be a 90-degree corner or it may be noise. In some cases, it is desirable to generate a tail while in other cases, a tail should not be generated [3] [8]. If desired, a corner finding technique can be used [28] and special constraints can be put in to enable the generation or suppression of a tail. Such a technique is not considered in the present implementation. To suppress a tail,  $p_i$  is deleted from the current chain. No new section is generated. The direction vector  $dir_{i-1}$  is recomputed as if  $p_{i-1}$  is linked to  $p_{i+1}$ , i.e.,  $dir_{i-1} = dir_{i-1} + dir_{int}$ .

In general, a convexity results in fewer elements being created for the new section than for the case of a concavity. Figures 5 and 6 also show the cases which do not occur (enclosed in dotted lines) either because an 8-connected contour will not be chain-coded in that way, or that it contradicts with the safe point tests.

Where a new section is generated, the corresponding pixels, including the terminal point but excluding the starting point, will have their values in bitmap S incremented. If the value is less than 2 (condition  $\alpha$ ), a flag corresponding to that chain will be set to indicate that a non-safe point has been created in the new chain. This flag was initially reset at the beginning of the iteration. This flag is to be tested before the beginning of the next iteration and if it is reset, the iteration for that chain terminates.

After  $p_i$  has been considered,  $dir_{i-1}$  are updated in preparation for the next element  $p_{i+1}$  in the chain.

## 9. The Head and Tail of a Chain.

As far as  $p_0$ , the head of a chain is concerned, one has to make the assumption that the previous element processed was  $p_{n-1}$ , the tail of the chain. Serial processing of a closed contour means that  $p_{n-1}$  will not be processed before  $p_0$  and the information that is available for  $p_0$  at the beginning of the chain may have been altered at the end of the chain when  $p_{n-1}$  is processed. For instance, at the beginning of the chain,  $p_{n-1}$  was not a safe point but it has since then becomes a safe point during the pass. It is necessary to append an extra vector to the new chain at the end of the pass.



The following strategy is used to deal with the head and tail of a chain. At the beginning of a pass, if  $p_0$  is a safe point, the new chain starts at  $p_0$ . If not, a test is made on  $p_{n-1}$ . If  $p_{n-1}$  is a safe point, the new chain starts at  $p_{n-1}$ . If both  $p_{n-1}$  and  $p_0$  are non-safe points, the new chain starts at the usual starting position for the first section, i.e. a common neighbor of  $p_{n-1}$  and  $p_0$  found by rotation as described in section 7. In this case a flag  $p_{n-1\_was\_not\_safe\_point}$  is set. If the starting pixel has been visited once before, i.e., it is on the boundary, its value in S is incremented so that it becomes a safe point.

At the end of the chain, when  $p_{n-1}$  is processed, if  $p_0$  is a safe point, an extra vector is added to link the last point of the chain to  $p_0$ .

If  $p_0$  is not a safe point;  $p_{n-1}$  is a safe point but the  $p_{n-1\_was\_not\_safe\_point}$  flag has been set, it means that  $p_{n-1}$  has since then become a safe point and an extra vector is needed to link up  $p_{n-1}$  and the start of the chain. The value of the head of the new chain is incremented in the bitmap S and tested for safe point.

## 10. Implementation.

The algorithm was coded in the C language and implemented on an Apollo DN3000 workstation. The details are found in Appendix II. Although the coding may be complex and low level, structured programming practices can still be applied and the resulting module is very efficient. The other algorithms, SPTA, Zhang/Suen, Contour tracing are also coded in the same manner so that the comparison described in the following section can be made upon a fair and equal basis.

## 11. Comparisons with other algorithms.

SPTA[14], Zhang/Suen[20] and the Contour tracing described by Pavlidis [9] are implemented in C and optimized so that they are as efficient as they can be. A labelling technique similar to the one used in SPTA has been used in the Zhang/Suen algorithm so that at the end of a pass, removal of pixels are not needed. SPTA has been modified slightly so that connectivity is preserved. The test images are 128x128 patterns, two of which are shown in figures 7 and 8. For the Chinese character in figure 7a, the computation times are as follows, Contour Generation: 0.517 sec (including recoding into chain codes); SPTA: 5.03 seconds (excluding salt-and-pepper noise removal time); Zhang/Suen: 5.73 seconds; and Pavlidis Contour Tracing: 2.02 seconds (excluding postprocessing). In this case, Contour Generation is 4 times faster than Contour Tracing; and 10 times faster than the two parallel algorithm.

For a thicker object, such as the bold Old-English B shown in figure 8a, the computation times are as follows, Contour Generation: 0.717 seconds (including recoding into chain codes); SPTA: 13.6 seconds (excluding salt-and-pepper noise removal time); Zhang/Suen: 14.4 seconds; and Pavlidis Contour Tracing: 4.37 seconds (excluding postprocessing). In this case, Contour Generation is 6 times faster than Contour Tracing; and 19 times faster than the two parallel algorithm.

Serial algorithms are faster than parallel algorithm implemented on a serial computer [11]. According to the data given by Arcelli [8] a serial algorithm is about 4 four times faster than a parallel algorithm. The reason is that in the case of a parallel algorithm, the time complexity is related to the size of the bitmap and the thickness of the objects; whereas in the case of a serial algorithm, the time complexity is related to the total size of all the objects in the bitmap. The difference in speed is more significant for thick objects because of the increase in complexity in the case of parallel algorithms for determining whether a dark point is an edge point or not.

Contour generation is considerably faster than contour tracing because in the former case, the chain codes are only extracted once from the original bitmap whereas in the latter case, a contour tracing is required in every pass.

In Contour Generation, the complexity of the initial contour tracing is related to the size of the bitmap. In each iteration, the outer layer of pixels of the object is processed. Processing time is proportional to the total length of the contours. The overall computation time is therefore proportional to the sum of the lengths of the contours in each pass. This sum is approximately equal to the size of the object because the non-safe points which make up the majority of the dark points will have been traversed exactly once during the entire process.

The efficiency of this method can be attributed to the following:

- (i) All the points visited are edge points.
- (ii) The only examination needed to perform a safe point test is on the value of the pixel in the bitmap  $S$ , i.e., the pixel is a non-safe point if its value in  $S$  is less than 2.
- (iii) Removal of a non-safe point is not necessary, but a new section of contour is generated. The break point information and the associated direction vectors of the current and last pixels in the chain, together with  $dir\_int$ , are all that is necessary and sufficient to identify the darkness of relevant neighbors of a pixel, to enable the generation of a new contour.

Contour generation is immune to noise in the image. For instance, an isolated pixel attached to a straight line (figure 9a) will either have no effect on the skeleton or will be precipitated as a 1-pixel long spike in the skeleton, depending on the order in which pixels on the contour are traversed. In the case of algorithms which are sensitive to noise, a long tail will be generated. Thus preprocessing is not needed in the case of Contour Generation to remove salt-and-pepper noise.

The order in which each of the contours is considered; the starting point and the direction of traversal of a contour will determine the final shape of the skeleton. Minor variations may be resulted if a different starting point and/or a different direction of traversal are chosen. If the head of a contour is at the middle of a straight line, even number of pixels thick, the skeleton will consist of two connected straight lines, with one of them displaced by one pixel from the other. The contour tracing algorithm used here ensures that a chain always begins at a corner position and the problem is avoided.

Connectivity and unit-thickness skeleton is guaranteed in Contour Generation because of the nature of algorithm, i.e., every point generated as a section of the new contour is tested for safe-point; and the iteration terminates when every point generated is a safe-point. Thus the last contours generated are just sufficient to ensure connectivity so that a skeleton of unit thickness is generated. No post-processing is needed.

Contour Generation can also be implemented on an MIMD type environment. The different chains of an object can be assigned to the individual processor or the chains can be subdivided into several subchains, each of which is assigned to a processor. The processors share the same bitmaps where the safe-point information can be updated and communicated between the different processes. When a pixel is processed, mutual exclusion must be enforced on itself and its neighbors. In this way, connectivity and unit-thickness skeleton is ensured.

## 12. Conclusion.

The problem of connectivity and thick skeleton experienced by many thinning algorithms can be attributed to the fact that when the outer layer of pixels of an object is removed, the structure of the resulting object is unknown as far as the current iteration is concerned. In the case of a parallel algorithm, the solution has been to divide a pass into several subiterations or to obtain edge information of the neighbors of a pixel. In the case of a serial algorithm, constraints were introduced to ensure connectivity so that in some cases, a doubly thick skeleton is produced.

In the case of the proposed Contour Generation method the result of processing the current pixel is made available to the subsequent pixels. This is in essence the enforcement of mutual exclusion. Since new contours are generated instead of old ones being removed, the algorithm can always keep track of the status of the new shape of the object after every pass. Therefore connectivity is ensured automatically.

By using a more relaxed safe point test, which is condition ( $\alpha$ ) of the multiple point test, the algorithm will not stop when two sections of a contour is neighboring each other, as in the case of a 2-pixel wide line. The final pass will retain the section that is visited last.

Although the coding of the algorithm is complex, Contour generation is faster than other methods because contour tracing is only performed once. During the iterations, only edge points are visited. When pictures are digitized to a higher and higher resolutions, made possible by the falling cost of memory and the increase in precision of digitizing equipment, time complexity of thinning algorithms should be made independent upon the product of the size of bitmap and number of passes. Contour Generation represents a significant reduction in time complexity.

## 13. Acknowledgement

This work was supported by the Natural Sciences and Engineering Research Council of Canada.

## Appendix I. An efficient contour tracing method.

The bitmap is scanned and for every scan line, the runs of dark points are extracted and positions of the edge points are noted and compared with the previous scan line. Chains may be appended to; new chains may be opened; or two chains may be merged depending on the situation. For a certain run in the current scan line, if none of the pixels is neighboring to a pixel of a run in the previous scan line, the run is disconnected with any opened chains. In this case, two new chains are opened. A direction *dir\_int* is attached to each new chain. *dir\_int* is equal to +1 for a counterclockwise scan of the exterior of an object or for a clockwise scan of an interior hole of the object. It is equal to -1 for a clockwise scan of the exterior or for a counterclockwise scan of an interior hole of the an object. This vector can always be determined uniquely and automatically, and is used to determine the direction of the interior of the object with respect to the chain.

If two or more runs in the current scan line are connected to the same run in the previous scan line, new chains are opened.

If there is at least one point which is a neighbor to a pixel in a run in the previous scan line, the two runs are connected and chain codes can be added to join the two runs. 8-connectivity is considered here. Finally for a certain run in the previous scan line, if none of the pixels is neighboring to at least a pixel of a run in the current scan line, the corresponding chains merge. If a run in the current scan line is connected to two or more runs in the previous scan line, one or more pairs of chains are merged.

After all the scan lines have been visited, the chains are joined together and the final number of chains is equal to the number of closed contours in the bitmap. The procedure returns a set of pointers for the heads of the chains and the interior direction *dir\_int* is determined for each of the final chains.

Incidentally, salt-and-pepper noise can also be removed without increasing the computation time. This is not required in the contour generation method because it is immune to this type of noise.

## Appendix II. Implementation details.

In the Apollo-DN3000 workstation, the screen can be mapped directly onto the user's address space so that the building up of the skeleton can be viewed while the individual bits are plotted on the bitmap S. The rate of accumulation of the skeleton is an indication of the speed of the thinning algorithm.

There are two ways of setting up the bitmaps. The array can be declared as a character array in which 8 bits assigned to every pixel. The second choice is to have one bit per pixel where 8 pixels are packed into a character. In implementing SPTA, an integer equal to the number of the pass is used to flag a safe-point. This is useful for reconstructing the original pattern. Consequently the former approach has been adopted. This resulted in the use of more memory but the access to a pixel is considerably faster than in the packed format.

In contour generation, since the chain code, *dir\_int* and the coordinates of the head of the chain define the bitmap completely, the memory space allocated for the original bitmap can be reused for the bitmap S without the need for clearing it. At the beginning, the dark points in the bitmap have values equal to 0. When a contour is created, each pixel generated will have its value incremented. When its value is greater than or equal to 2, it becomes a safe-point (Condition  $\alpha$ ). Its value will be tested for safe points. This operation of incrementing and test is analogous to a *test-and-set* operation in Concurrent Programming.

The use of pointers in C speeds up the location of a pixel considerably. This results in a higher data transfer rate to and from the bitmap than is possible with a 2-D matrix. The ability to manipulate bits means that modulo-8 addition and subtraction can be done using the normal integer addition/subtraction with the result ANDed to 7.

In order to compare the different algorithm on a fair basis, every algorithm is coded using the same strategy regarding the set up of the bitmap and of the pointers for the neighbors of a given pixel. Furthermore, all the implementations are optimized so that they run faster than those implicated by their respective publications.

## References

- [1] Moayer, B. and Fu, K.S. A tree system approach for fingerprint pattern recognition. *IEEE Trans. Comput. C-25*, (1976), 262-275.
- [2] Beun, M. A flexible method for automatic reading of hand-written numerals. *Phillips Technical Review* 31, 4 (1973), 89-101.
- [3] Dill, A.R. and Levine, M.D. Multiple resolution skeletons. *IEEE Trans Pattern Analysis and Machine Intelligence PAMI-9*, 4 (1987), 495-504.
- [4] Davies, E.R. and Plummer, A.P.N. Thinning algorithm, a critique and a new methodology. *Pattern Recognition* 14, 1 (1981), 53 - 63.
- [5] Pfaltz, J.L. and Rosenfeld, A. Computer representation of planar regions by their skeletons. *Commun. ACM* 10, 2 (February 1967), 119-125.
- [6] Blum, H. A transformation for extracting new descriptors of shape. *Symposium on Models for the Perception of Speech and Visual Form*, MIT Press, Cambridge, Mass., (1964).
- [7] Arcelli, C. and Sanniti di Baja, G. On the sequential approach to medial line transformation. *IEEE Trans. Systems, Man and Cybernetics SMC-8*, (1978), 139-144.
- [8] Arcelli, C. Pattern thinning by contour tracing. *Computer Graphics and Image Processing* 17, (1981), 130-144.
- [9] Pavlidis, T. A thinning algorithm for discrete binary images. *Computer Graphics and Image Processing* 13 (1980), 142-157.
- [10] Pavlidis, T. *Algorithms for graphics and image processing*, Rockville MD : Computer Science Press, 1982..
- [11] Rosenfeld, A. and Pfaltz, J.L. Sequential operations in digital picture processing. *Journal ACM* 13, 4 (October 1966), 471 - 494.

- [12] Stefanelli, R. and Rosenfeld, A. Some parallel thinning algorithms for digital pictures. *Journal of ACM* 18, 2 (April 1971), 255 - 264.
- [13] Perry, T.S. and Wallich, P. Computer displays — new choices, new trade-offs and From lab to lap. *IEEE Spectrum* 22, 7 (July 1985), 52-59.
- [14] Naccache, N.J. and Shinghal, R. SPTA : a proposed algorithm for thinning binary patterns. *IEEE Trans. SMC-14*, 3 (1984), 409 - 418.
- [15] Arcelli, C. A condition for digital points removal. *Signal Processing*, 1, (1979), 283 - 285.
- [16] Bel-Lan, A. and Montoto, L. A thinning transform for digital images. *Signal Processing*, 3, (1981), 37 - 47.
- [17] Hilditch, C.J. Linear skeletons from square cupboards, *Machine Intelligence*, 4, (1969), 403 - 420.
- [18] Rosenfeld, A. A characterization of parallel thinning algorithms. *Inform. Contr.* 29, (1975), 286 - 291.
- [19] Tamura, H. A comparison of line thinning algorithms from digital geometry viewpoint. *Proceedings of 4th International Conf. on Pattern Recognition*, Kyoto, Japan, (1978), 715 - 719.
- [20] Zhang, T.Y. and Suen, C.Y. A fast parallel algorithm for thinning digital patterns. *Communications of ACM* 27, 3 (1984), 236- 239.
- [21] Lu, H.E. and Wang, P.S.P. A comment on “A fast parallel algorithm for thinning digital patterns”. *Communications ACM* 29, 3 (1986), 239-242.
- [22] Brinch Hansen, P. Concurrent programming concepts. *ACM Computing Surveys* 5, 4 (December 1973), 223-245.
- [23] Holt, C.M. et al. An improved parallel thinning algorithm. *Commun. ACM* 30, 2 (February 1987), 156-160.
- [24] Freeman, H. On the encoding of arbitrary geometric configurations. *IEEE Trans. Electronic Computers EC-10*, (1961), 260 - 268.
- [25] Freeman, H. and Garder, L. A pictorial jigsaw puzzles, the computer solution of a problem in pattern recognition. *IEEE Trans. Electronic Computers EC-13*, (1964), 118 - 127.
- [26] Pavlidis, T. Filling algorithms for raster graphics. *Computer Graphics and Image Processing* 10, (1979), 126-141.
- [27] Sobel, I. Neighbourhood coding of binary images for fast contour following and general binary array processing. *Computer Graphics and Image Processing* 8, (1978), 127 - 135.
- [28] Freeman, H. and Davis, L. A corner-finding algorithm for chain-coded curves. *IEEE Trans Computers* 26, 3 (1977), 297 -303.

*Computing Review Categories.* 1.5.2 [Pattern Recognition]: Design Methodology - *pattern analysis*; 1.5.4 [Pattern Recognition]: Applications - *computer vision*.

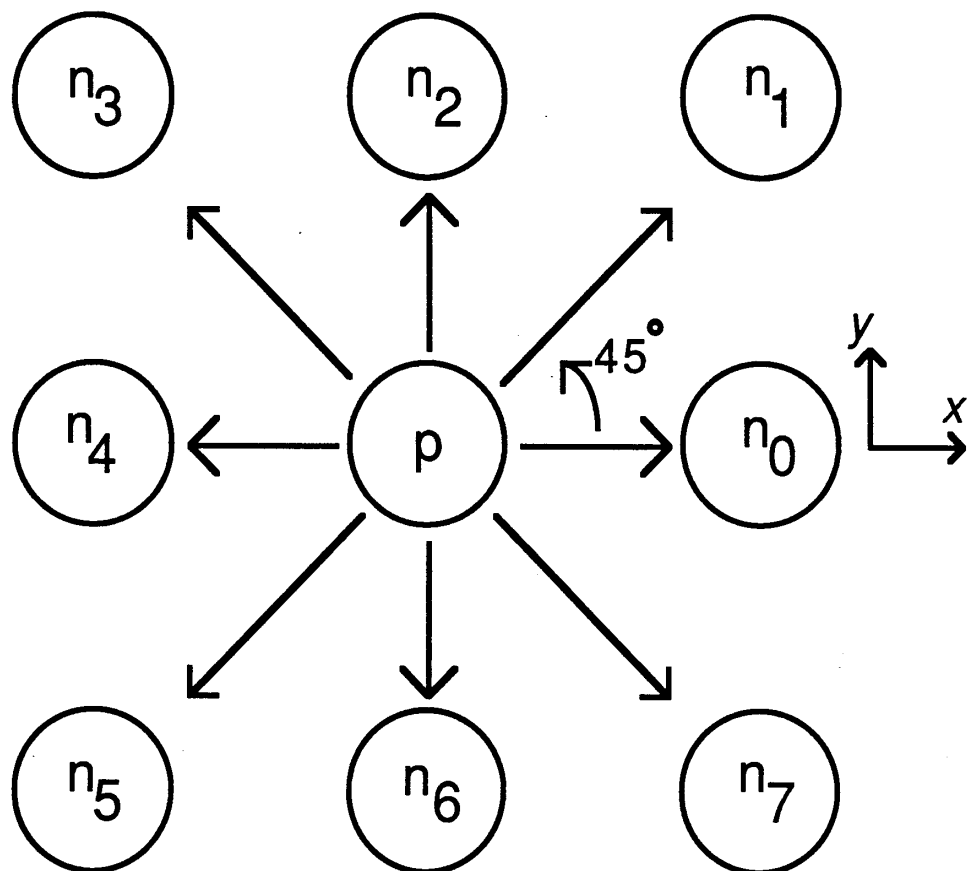
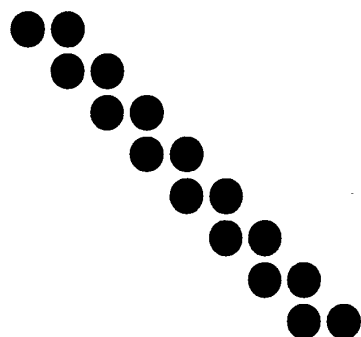
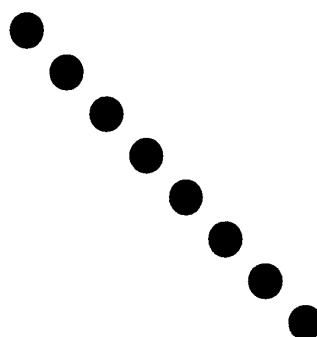


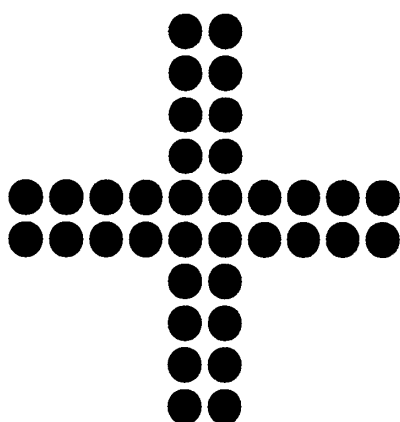
Figure 1 The 8-neighbours of a pixel  $p$



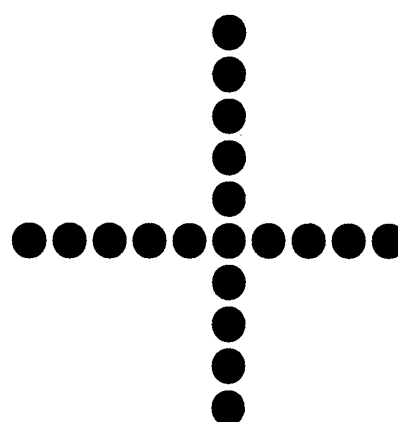
(2a) a  $45^\circ$  line



(2b) skeleton of 2a



(2c)  
vertical and  
horizontal lines



(2d)  
skeleton of 2c

Figure 2 2-pixel thick lines



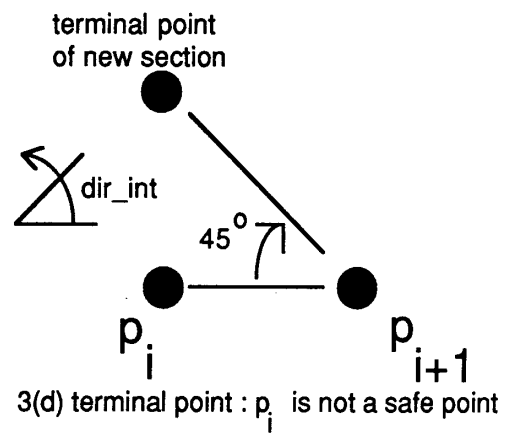
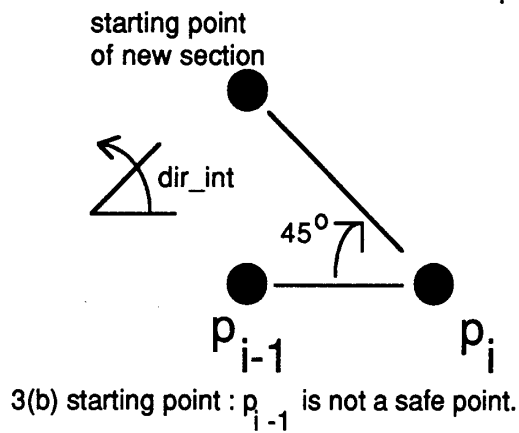
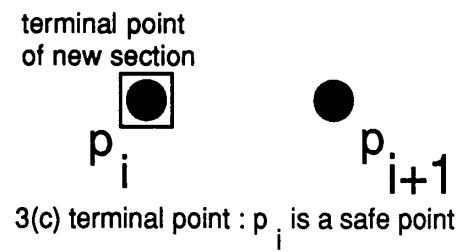
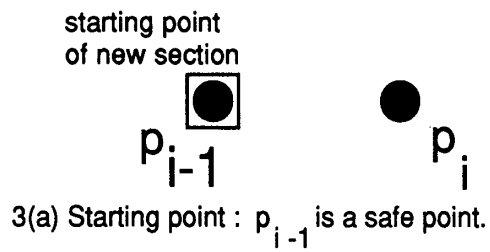


Figure 3 Starting and terminal points of a section of the new boundary

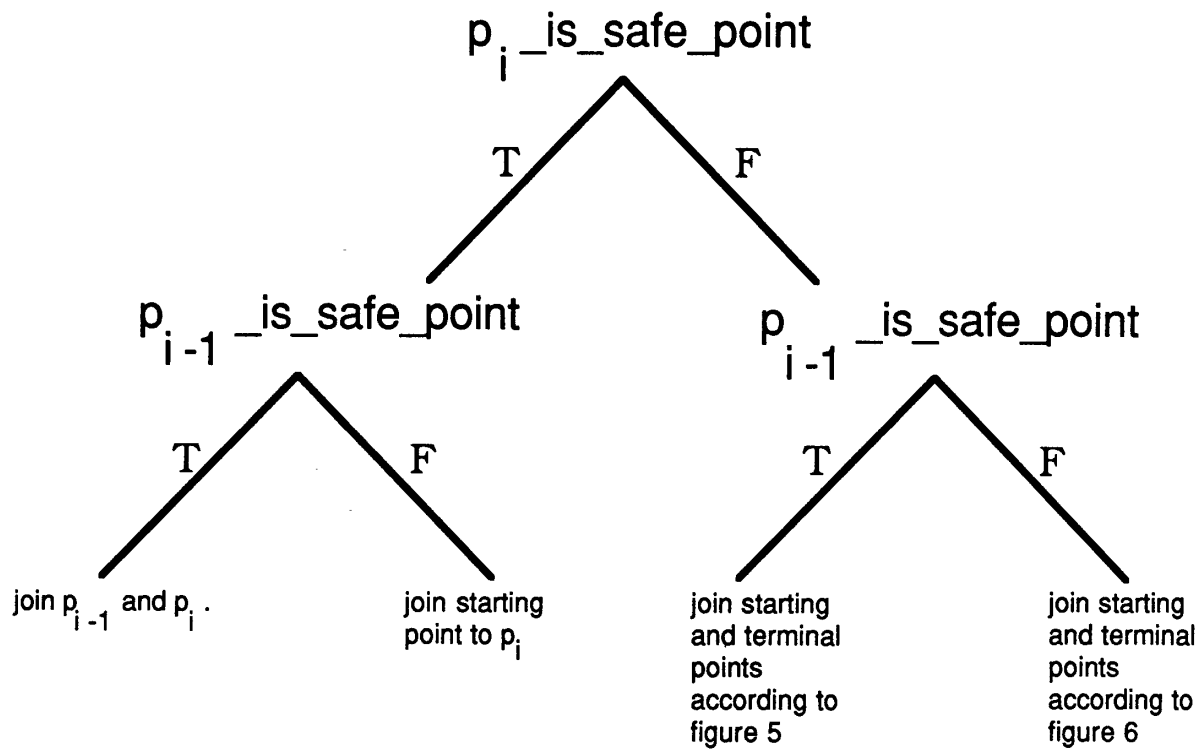



Figure 4 The four cases for generating a new section of a new chain

$p_{i-1}$  is a safe point but  $p_i$  is not a safe point.

  $\text{dir\_int} = +1$

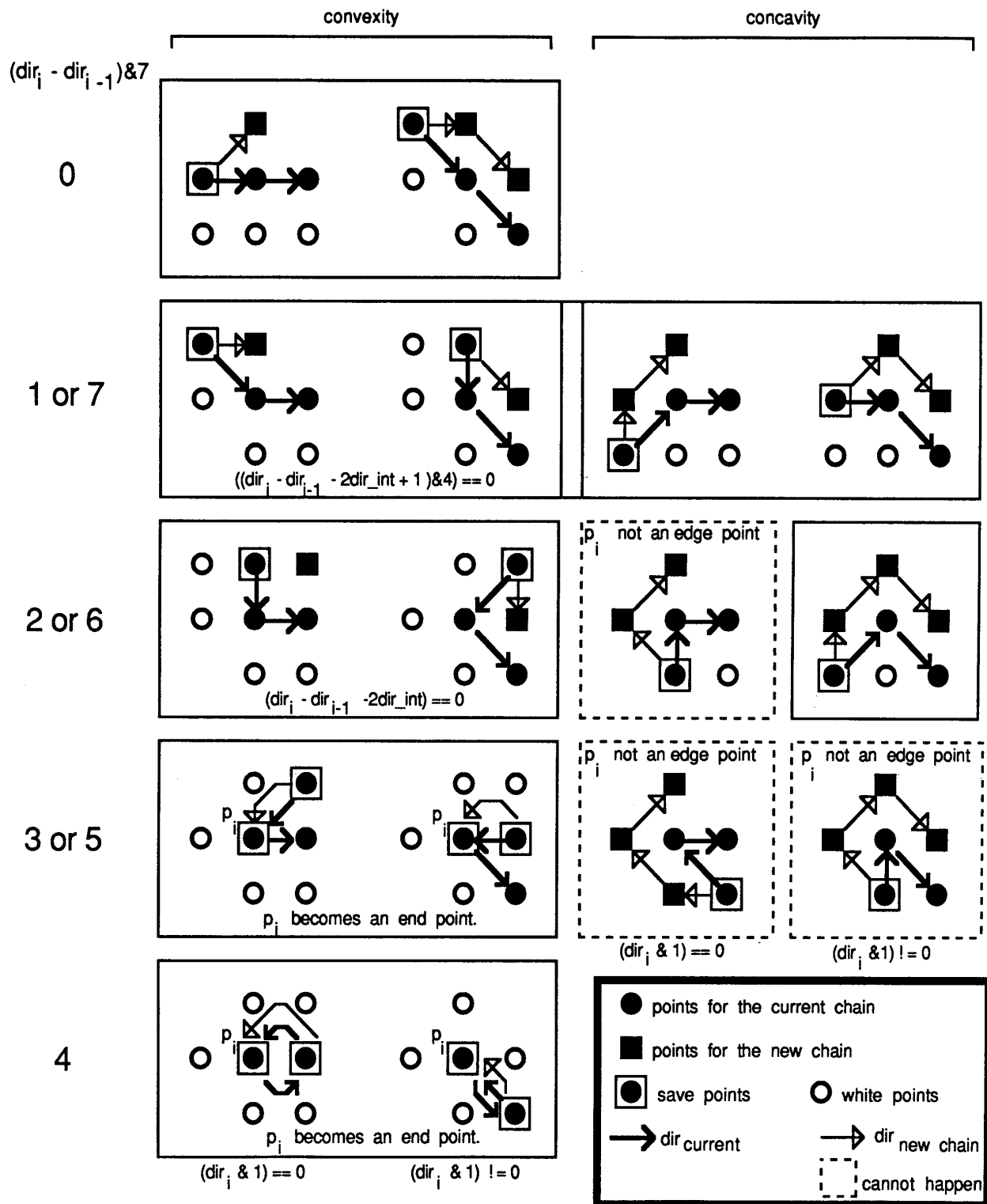


Figure 5 Generation of a section of the new chain, shown here with  $\text{dir\_int} = +1$

$p_{i-1}$  and  $p_i$  are not safe points.

$\text{dir\_int} = +1$

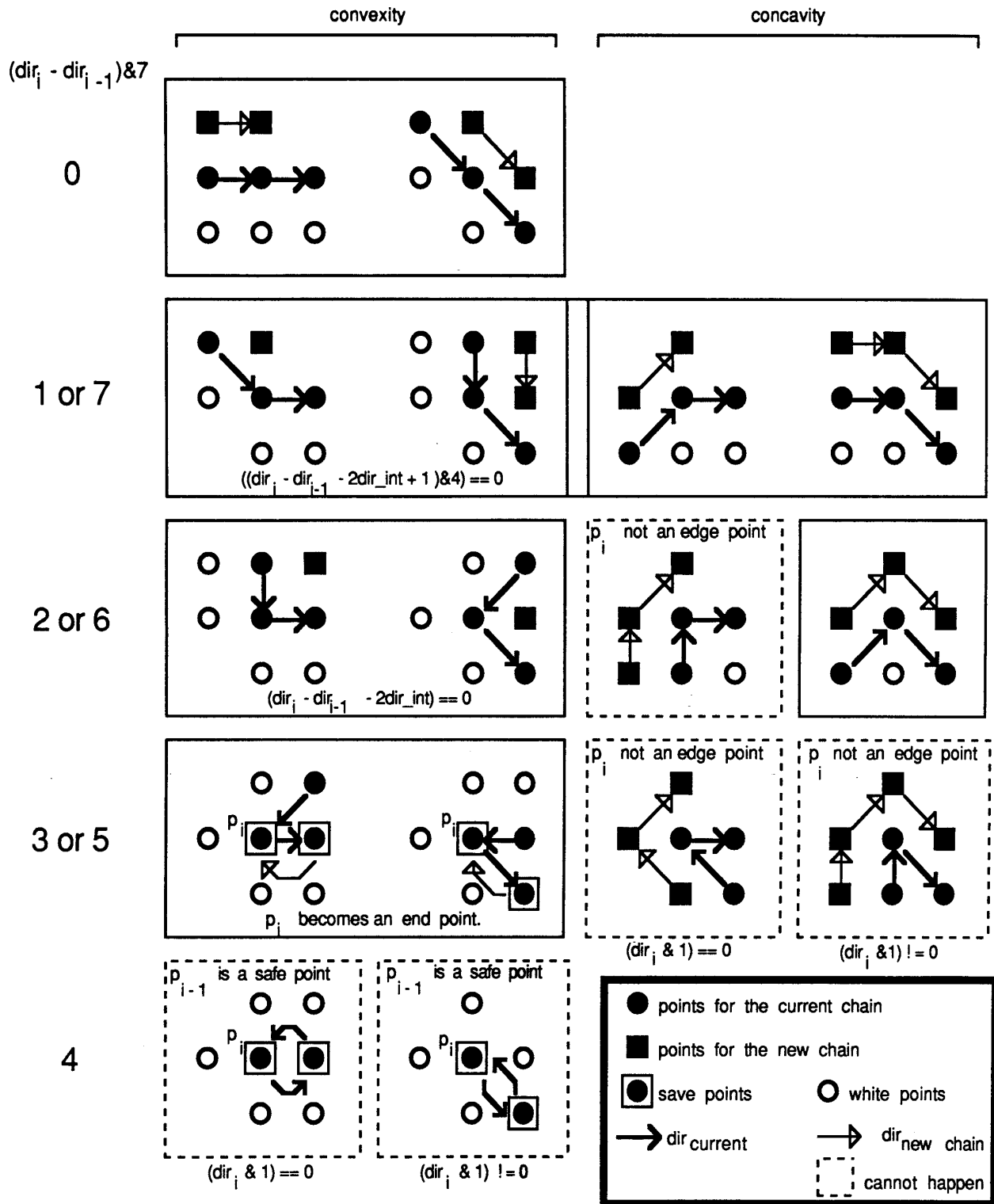
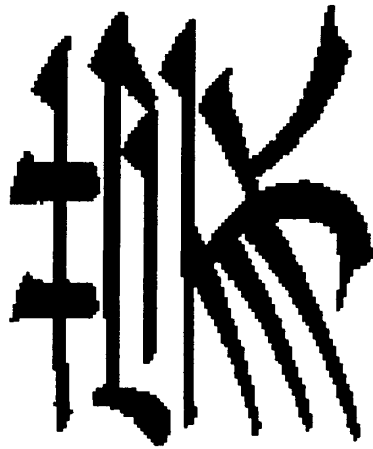
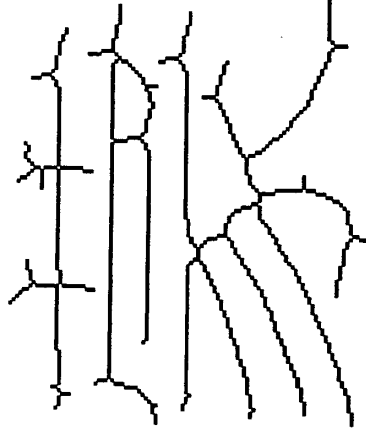


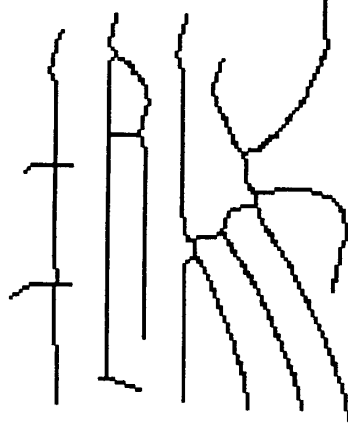
Figure 6 Generation of a section of the new chain, shown here with  $\text{dir\_int} = +1$



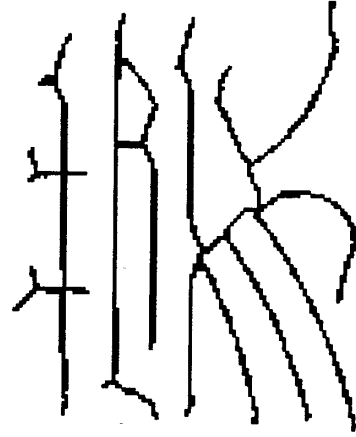
(7a). The original character consist of 3 disjoint objects, one with a hole



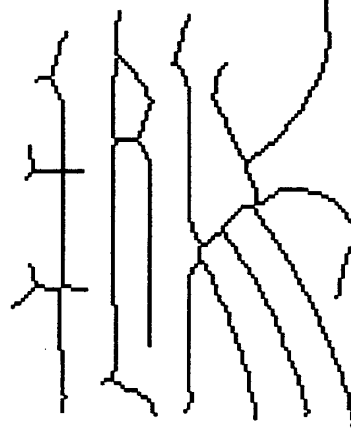
(7b) SPTA algorithm, cpu time: 5.03 sec, without preprocessing



(7c) Zhang/Suen algorithm, cpu time: 5.73 sec



(7d) Thinning by Contour Tracing, cpu time: 2.02 sec, without preprocessing



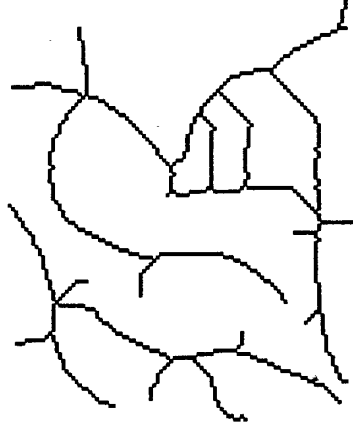
(7e) Thinning by Contour Generation, cpu time: 0.517 sec

Figure 7 A 128x128

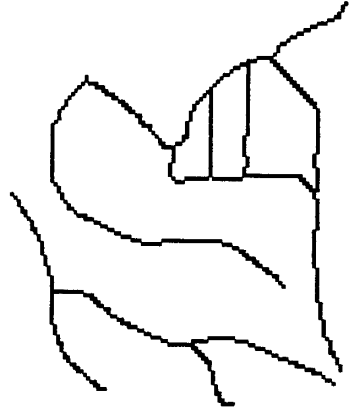
Chinese character



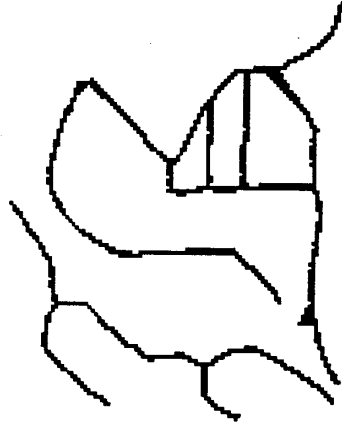
(8a) The original character consist of 2 disjoint objects, one with 3 holes.



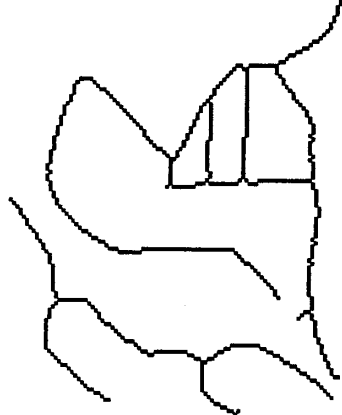
(8b) SPTA algorithm, cpu time: 13.6 sec, without preprocessing



(8c) Zhang/Suen algorithm, cpu time: 14.4 sec

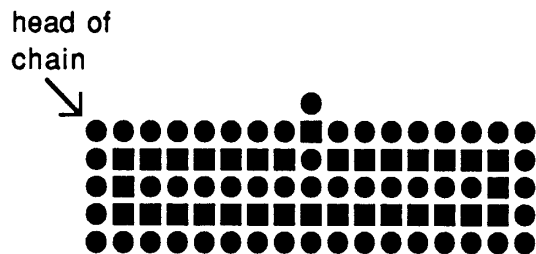


(8d) Thinning by Contour Tracing, cpu time: 4.37 sec, without postprocessing

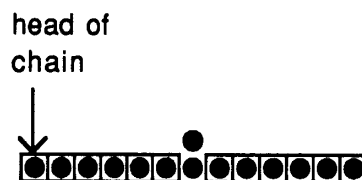


(8e) Thinning by Contour Generation, cpu time: 0.717 sec

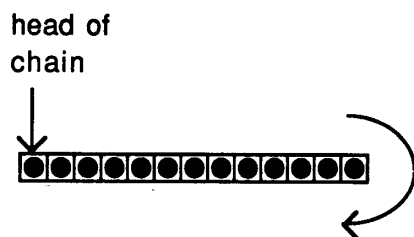
Figure 8 A 128x128, bold Old English B.



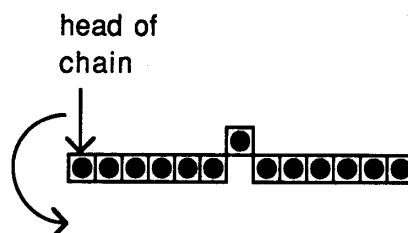
9(a) A 5-pixel wide line with noise.  
The square boxes represent the chain generated at the end of the first pass.



9(b) The chain generated after the second pass. All but two of the pixels have become save points.



9(c) Final skeleton with a clockwise chain.



9(d) Final skeleton with a counterclockwise chain



9(e) Skeleton obtained from SPTA  
without noise removal

Figure 9 Lines with noise