A THINNING ALGORITHM BY CONTOUR GENERATION

A new contour generating serial algorithm is faster and more efficient than conventional contour tracing and parallel algorithms

PAUL C. K. KWOK

Thinning is an important preprocessing step for many image analysis operations such as fingerprint recognition [16] and optical character recognition [5]; it is also used in biomedical systems [9].

Thinning usually involves removing points or layers of outline from a pattern until all the lines or curves are of unit width, or a single pixel wide [8, 22]. The resulting set of lines or curves is called the skeleton of the object. Many algorithms are available. An analog technique [6] generates a medial line in which every point is equidistant 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 [1, 3, 18, 19]. When contour pixels touch the medial line, the skeleton will be two pixels wide [1, 9, 19] 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. [24, 26] classified thinning algorithms as being parallel or sequential. With parallel algorithms, only the result from the previous iteration affects the decision to remove a point in the current iteration, making it suitable for processing by parallel hardware such as an array processor. A sequential algorithm, on the other hand, uses the result from the previous pass plus the results obtained so far in the current pass to process the current pixel. Thus at any point in an iteration, a number of pixels have already been processed. These results can be used immediately to process the next pixel. It is generally believed that a sequential algorithm will be faster than a parallel algorithm *implemented on a serial computer* [24].

As the resolution of digitization equipment, frame buffers, and displays [21] 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 sections, the terminology will be explained, and two parallel algorithms and a serial algorithm will be described briefly and later used as a basis for comparison with a new serial method proposed in this article.

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 $\bar{Q}$ corresponds to a set of "white points." For a given pixel p there are eight neighbors $n_0, n_1, \ldots n_7$, 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 $i \cdot 45°$. 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° 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.

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

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

An object is 8-connected if all pairs of points in the object are 8-connected. An object is 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:

(1) Connectivity should be preserved. If the object is connected, the resulting skeleton should also be connected. If the initial background is connected, the background resulting from 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 [1], 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° line shown in Figure 2a should be further thinned down to a 1-pixel-wide line (Figure 2b). A dark point that disconnects an object if removed is called a break point. Thus a break point test is incorporated into many thinning algorithms to preserve connectivity.

(2) 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 that represents a true feature of the object is not shortened excessively.

(3) The skeleton 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, (2) and (3) may be conflicting criteria and constraints will sometimes be added to obtain a compromise between the two.

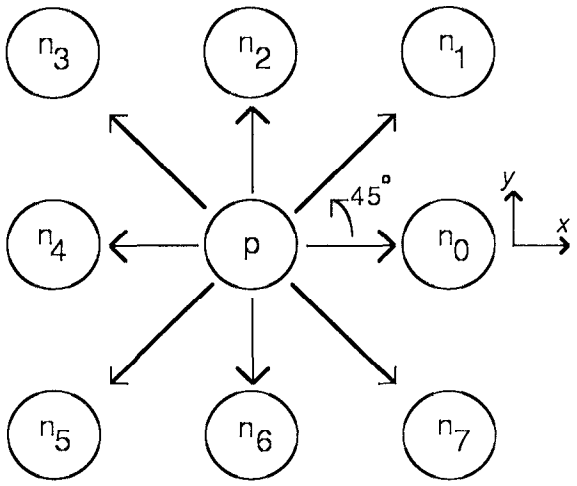

FIGURE 1. The 8-Neighbors of a Pixel p

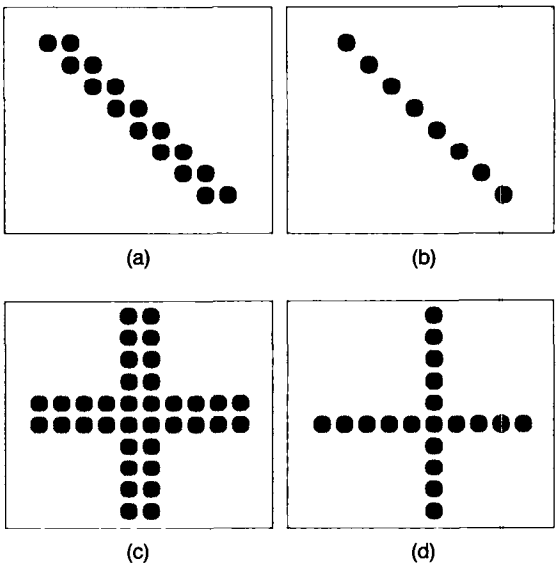

FIGURE 2. 2-pixel Thick Lines (a) A 45° Line (b) Skeleton of a (c) Vertical and Horizontal Lines (d) Skeleton of c

PARALLEL ALGORITHM

Naccache and Shinghal [17] reviewed and compared the speed of 14 parallel thinning algorithms [2, 4, 5, 13, 18, 23, 26, 27]. 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 nonedge points. Only the edge points need to be considered. Tests are conducted on each edge point's eight neighbors to determine whether they are break points, end points, or nonsafe points. The nonsafe 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.

Although the fourteen algorithms are similar, they differ in the way they handle break points and end points. The safe-point-thinning algorithm (SPTA) was proposed [17] and from experimental results, Naccache and Shinghal concluded that SPTA was the fastest method. 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 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 nonsafe point so that at the end of a pass, the nonsafe points do not have to be eliminated explicitly from the bitmap. This also makes it possible to reconstruct the original pattern from the skeleton.

Zhang and Suen [29] proposed a slightly different parallel algorithm. The break point and end point tests consist of (1) examining the number of white/dark transitions when the eight neighbors $n_0, n_1, \ldots n_7, n_0$,

are traversed in that order (the number should be equal to 1), (2) counting 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 Lü/Wang algorithm [15], and (3) performing 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 [15] because the result of removing one side of a line is unknown to the other side during the same pass. This is a mutual exclusion problem, well-known in concurrent programming [7], which occurs when several parallel processes share the same memory. In thinning algorithms, when a process examines a certain pixel, it should have exclusive use of that pixel and its eight neighbors. In parallel thinning algorithms implemented on a parallel architecture or simulated on a serial architecture, this 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 four subiterations [23], [26], each responsible for the removal of the top, bottom, left, or right edge points. An alternative is to combine the top and left subiteration or the bottom and right subiteration. This will reduce the number of subiterations to two. However, in many instances, a 2-pixel-wide, 45° line (Figure 2a) may be completely removed for the 2-subiteration case [15]. A slightly different variation of the method combines the left and right subiterations, and the top and bottom subiterations [17]. 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 [14].

The time complexity of a parallel algorithm implemented on a serial computer consists of three components:

(1) 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.

(2) 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.

(3) The number of passes is related to the "thickness" of the object.

The total number of operations in (1) 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 (2) 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 (2) increases sharply when the resolution of the image or the thickness of the objects increases.

SERIAL AND SEQUENTIAL ALGORITHMS

Sequential technique is an alternative to parallel methods. Less memory is required in sequential algorithms [3, 13]. However, as memory cost continues to fall, memory usage is no longer an issue. Similar to parallel algorithms, sequential algorithms also examine every pixel 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 that belong to the outline of an object. Xu and Wang [28] introduced the idea of contour generation where the four types of edge points—east, north, west, and south—are put into buffers. These points are examined sequentially and matched to a 3×3 window. If a point is removable, its D-neighbors in the interior of the object constitute the new contour and are put into the respective buffers. The technique was demonstrated to be superior to many thinning algorithms.

Serial algorithms, and among them, the contour tracing technique was introduced to deal with nearly thinned objects [19] or thick objects [1, 9]. 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 $p_0, p_1, \ldots p_n$, where $p_0 \equiv p_n$ (p_n is redundant and can be eliminated from the chain). Where multiple disjointed objects are involved and break points or 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 [10, 12], which is a sequence of directions $dir_0, dir_1, \ldots dir_{n-1}$, 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 eight 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 heads or starting points p_0 will completely define the bitmap, i.e., the original bitmap can be completely recovered with this information. In fact, thinning algorithms have been related to filling algorithms [20].

In contour tracing, the term *multiple pixel* is defined to describe edge points that satisfy one or more of the following:

(α) It is traversed more than once when tracing the set of contours.
(β) It has no neighbors in the interior of Q.
(γ) 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 [1] presented a variation of the contour tracing thinning algorithm. After obtaining the set of multiple pixels, only the nonmultiple pixels belonging to the contour are removed.

In both of these 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), by adopting the deletability criteria based on the notion of crossing number.

CONTOUR GENERATION USING CHAIN CODES

Evidently the necessity of 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 a parallel algorithm, a pixel is processed on the basis of 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 7e and 8e) will remain after the final iteration.

The solution is therefore to consider 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 in the current contour being visited. The section is checked for break points and this information is available when subsequent pixels in the sequence or those on the next sequence are visited. At the end of the iteration, a new contour will be available for the next iteration without having to remove the old one.

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

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 [18, 25]. An efficient method (see box for details) is used. The resulting set of chains represent the outermost layer of the objects is used here. The chain is counterclockwise for the exterior of an object and clockwise for an interior hole. As one traverses along the contour, the right-hand side is always the background. The chains together with the coordinates of the heads of chains define the bitmap completely. This is a recoding step, after which the original bitmap is no longer needed.

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, 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 a neighbor of 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.

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 are neighbors of at least one 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.

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.

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 α).

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 nonsafe points in that contour. Thus the number of passes required for one contour may differ from another. When the operation completes, the skeleton is formed in S. A chain code describing the skeleton is also available.

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 xy coordinates of p_i are updated from the xy 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 this 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.

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 from 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° into the interior using p_i as the center of rotation. 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° into the interior, using p_{i+1} as the center of rotation. Again this point must be a dark point, otherwise p_i would have been a break point.

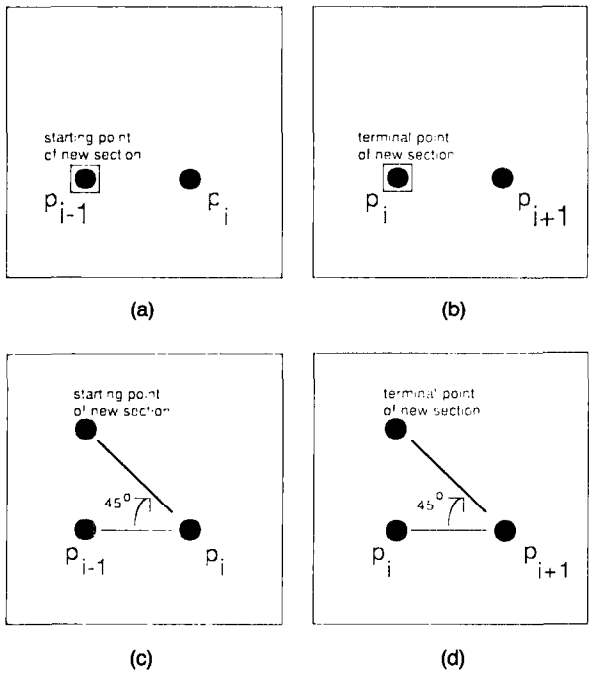

FIGURE 3. Starting and Terminal Points of a Section of the New Boundary (a) Starting Point: p_{i-1} is a Safe Point (b) Starting Point: p_{i-1} is not a Safe Point (c) Terminal Point: p_i is a Safe Point (d) Terminal Point: p_i is not a Safe Point

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 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 toward the final iterations as more and more safe points are uncovered.

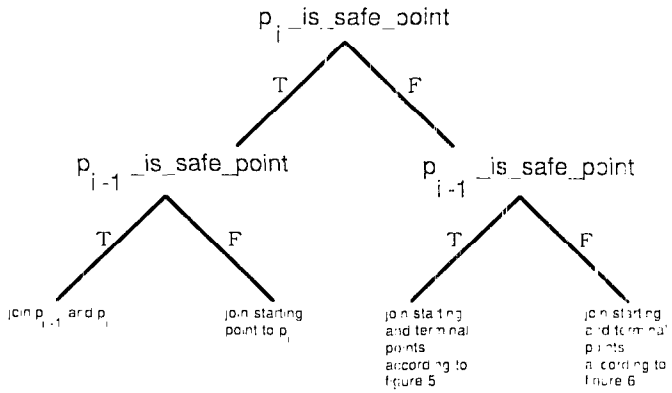

FIGURE 4. The Four Cases for Generating a New Section of a New Chain

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 $(dir_{i-1} + 7)$.

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 begins at the starting point and assumes a counterclockwise path visiting the D-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 will contradict the safe point test just conducted on p_i. The safe point test performed on the bitmap S was up-to-date because it included 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 $(dir_i - dir_{i-1}) = 4$, p_i is an end point and will be flagged as a safe point automatically.

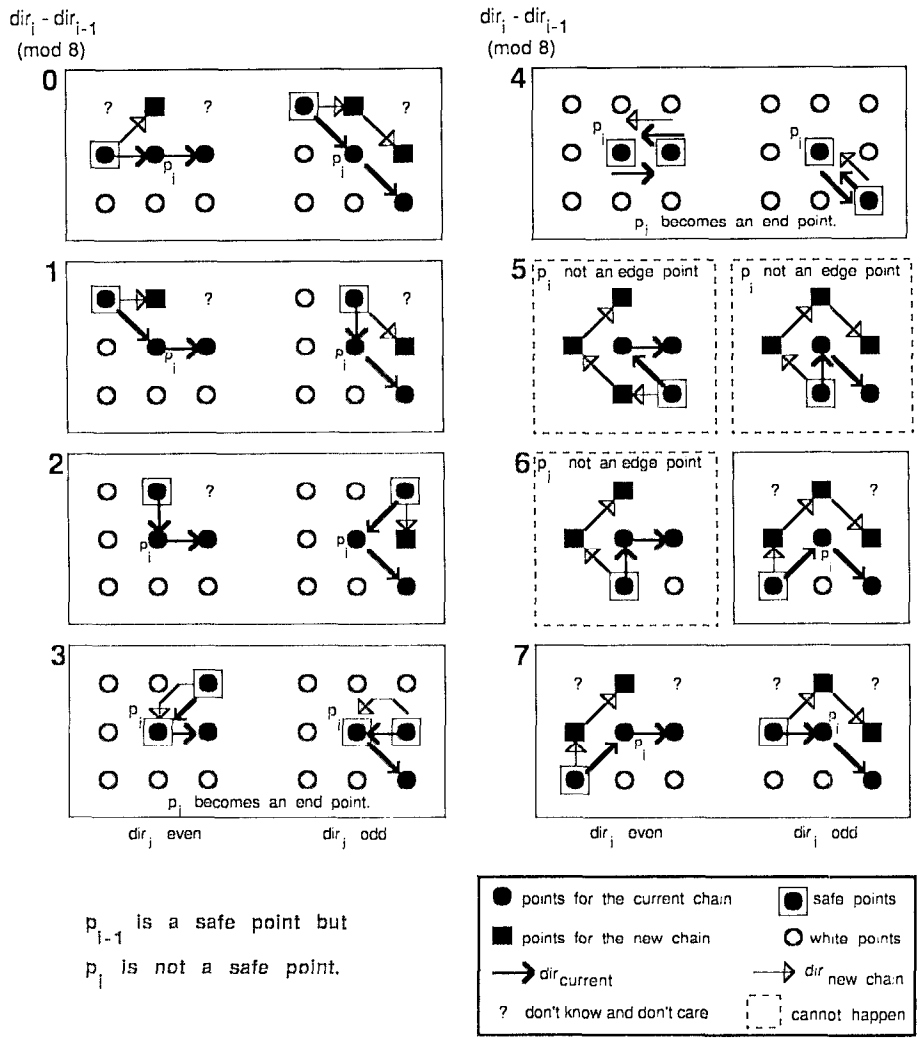

FIGURE 5. Generation of a Section of the New Chain

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° 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 [1, 9]. If desired, a corner finding technique can be used [11] 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., $dir_{i-1} = dir_{i-1} + 1$.

Figures 5 and 6 also show the cases that do not occur (enclosed in dotted lines) either because an 8-connected contour will not be chain-coded in that way or because it contradicts the safe point tests.

From Figures 5 and 6, a table of chain codes can be formed (Table I) and can be used for the subsequent implementation. In general, a convexity results in fewer elements being created for the new section than for the case of a concavity.

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 (condition α). Table I also shows the direction of these points with respect to p_i.

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

THE HEAD AND TAIL OF A CHAIN

As far as p_0, the head of a chain is concerned, one must assume that the previous element processed was p_{n-1} (the tail of the chain). Serial processing of a closed con-

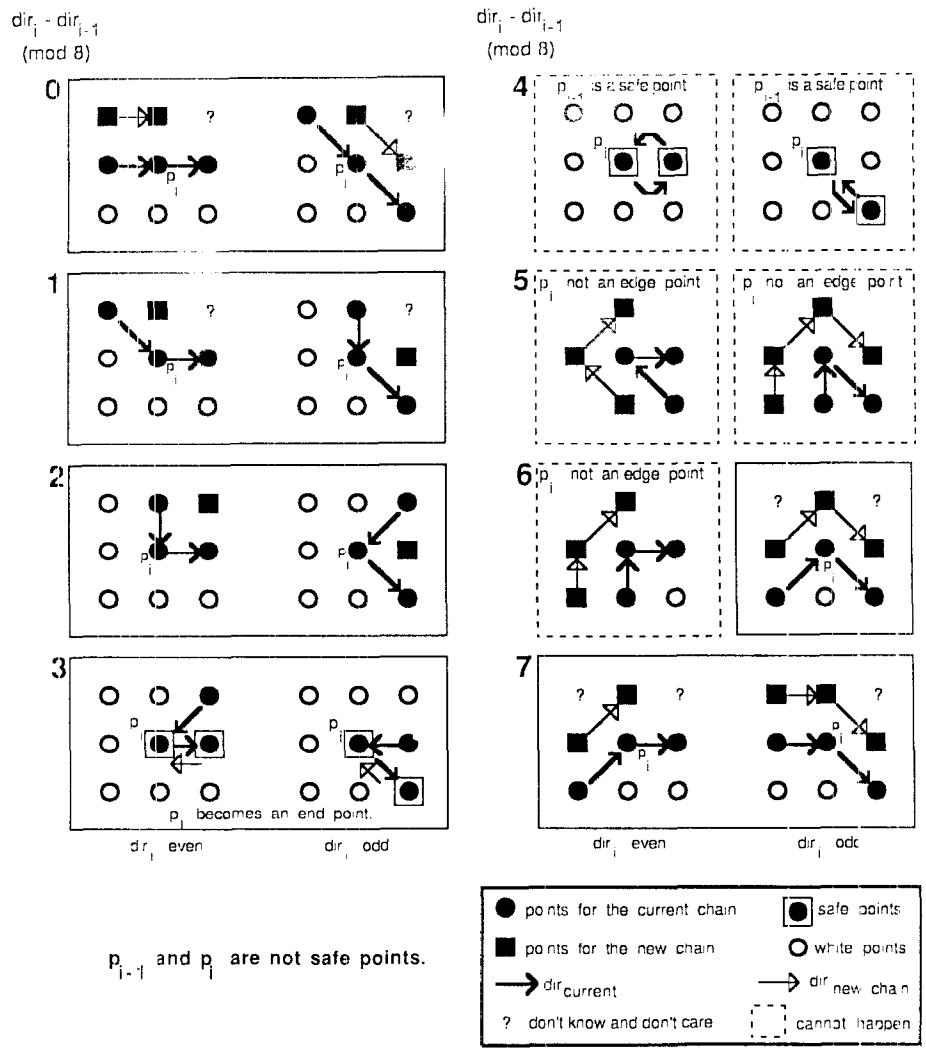

FIGURE 6. Generation of a Section of the New Chain

tour 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 by the time p_{n-1} is processed the end of the chain. For instance, at the beginning of the chain, p_{n-1} was not a safe point, but it becomes a safe point during the pass. It may be necessary to append an extra vector to the new chain to complete a closed contour.

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 nonsafe points, the new chain starts at the usual starting position for the first section,[1] i.e., a common neighbor of p_{n-1} and p_0 found by rotation as described earlier. 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, meaning that p_{n-1} has since become a safe point and an extra vector is needed to link up p_{n-1} and the start of the chain. In one case it is necessary to alter the position of the head of the new chain. The value of the head of the new chain is incremented in the bitmap S.

COMPARISONS WITH OTHER ALGORITHMS

SPTA [17], Zhang/Suen [29], Xu/Wang's CGT [28], and the contour tracing described by Pavlidis [19] 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, pixels do

[1] There are two special cases: if $(dir_{n-1} - dir_{n-2} = 2)$ and (dir_{n-1}) is even, p_{n-1} is deleted from the chain. If $(dir_{n-1} - dir_{n-2} = 3)$ and (dir_{n-1}) is even, the value of p_{n-1} is incremented to become a safe point.

TABLE I. Chain Code Elements and the Generated Contour Points

$(dir_i - dir_{i-1})$ & 7	p_{i-1} is a safe point.				p_{i-1} is not a safe point.			
	$(dir_i \& 1) == 0$	$(dir_i \& 1) != 0$	$(dir_i \& 1) == 0$	$(dir_i \& 1) != 0$	$(dir_i \& 1) == 0$	$(dir_i \& 1) != 0$	$(dir_i \& 1) == 0$	$(dir_i \& 1) != 0$
0	$dir_i + 1$	$dir_i + 1$ dir_i	$dir_i + 2$	$dir_i + 3$ $dir_i + 1$	dir_i	dir_i	$dir_i + 2$	$dir_i + 1$
1	dir_i	dir_i	$dir_i + 2$	$dir_i + 1$	none	dir_{i-1}		$dir_i + 1$
2	none	$dir_i + 7$		$dir_i + 1$	none	none	delete p_i	
3, 4	dir_{i-1}		p_i itself		$dir_i + 4$		p_i itself	
6	$dir_i + 3$ dir_{i-1} dir_i		$dir_i + 5$ $dir_i + 3$ $dir_i + 1$		dir_{i-1} dir_i		$dir_i + 3$ $dir_i + 1$	
7	$dir_i + 2$ dir_{i-1}	$dir_i + 2$ dir_i	$dir_i + 4$ $dir_i + 2$	$dir_i + 3$ $dir_i + 1$	dir_{i-1}	dir_{i-1} dir_i	$dir_i + 2$	$dir_i + 3$ $dir_i + 1$
	chain code elements		contour point direction w.r.t. p_i		chain code elements		contour point direction w.r.t. p_i	

not need to be removed. Xu/Wang's CGT algorithm was implemented using the SPTA windows. The test images are 128 × 128 patterns, two of which are shown in Figures 7 and 8. For the Chinese character shown in Figure 7a, the computation times are as follows: contour generation using chain codes: 0.517 seconds (including recoding into chain codes); SPTA: 5.03 seconds (excluding salt-and-pepper noise removal time); Zhang/Suen: 5.73 seconds; Xu/Wang's CGT: 0.783 seconds and Pavlidis contour tracing: 2.02 seconds (excluding post-processing). In this case, contour generation using chain codes is 50% faster than CGT, 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 using chain codes: 0.717 seconds (including recoding into chain codes); SPTA:

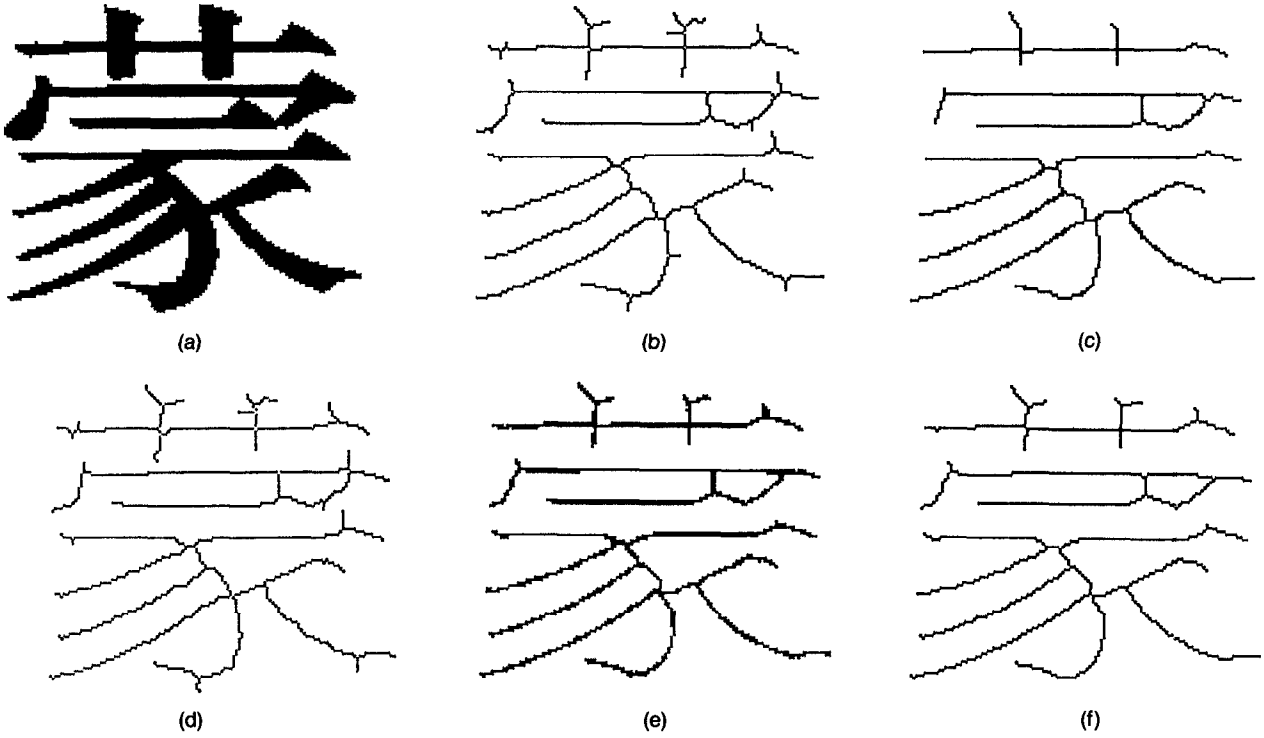

FIGURE 7. A 128 × 128 Chinese Character (a) The Original Character Consists of Three Disjoint Objects, One with a Hole (b) SPTA (c) Zhang/Suen Algorithm (d) Xu/Wang's CGT (e) Pavlidis's Contour Tracing (f) Contour Generation with Chain Codes

Implementation Details

The algorithm was coded in the C language and implemented on an Apollo DN3000 workstation. 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, Xu/Wang's CGT, and Pavlidis's contour tracing are also coded in the same manner so that the comparison described in the article can be made upon a fair and equal basis.

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 eight bits are assigned to every pixel. The second choice is to have one bit per pixel where eight 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 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 α). 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 *AND*ed to 7.

To compare the algorithms fairly, every algorithm is coded using the same strategy regarding the setup 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.

13.6 seconds (excluding salt-and-pepper noise removal time); Zhang/Suen: 14.4 seconds; Xu/Wang's CGT: 1.28 seconds; and Pavlidis contour tracing: 4.37 seconds (excluding postprocessing). In this case, contour generation using chain codes is 80 percent faster than CGT; 6 times faster than contour tracing; and 19 times faster than the two parallel algorithm.

Serial algorithms are faster than a parallel algorithm implemented on a serial computer [24]. According to the data given by Arcelli [1] 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 the chain codes are extracted only once from the original bitmap whereas 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 nonsafe points which make up the majority of the dark points will have been traversed exactly once during the entire process.

Contour generation using chain codes is faster than CGT because in CGT, examination of a 3×3 window of neighbors is time consuming.

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

(1) All the points visited are edge points.
(2) 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 can be removed if its value is less than 2.
(3) Removal of a pixel 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 are all that is necessary 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 that are sensitive to noise, a long tail will be generated. Thus preprocessing is not needed with contour generation to remove salt-and-pepper noise.

The order in which each of the contours is considered and the starting point and the direction of traversal of a contour will determine the final shape of the skeleton. Minor variations may result 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, and is an even number of pixels thick, the skeleton will consist of two connected straight lines, one 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 of the skeleton is

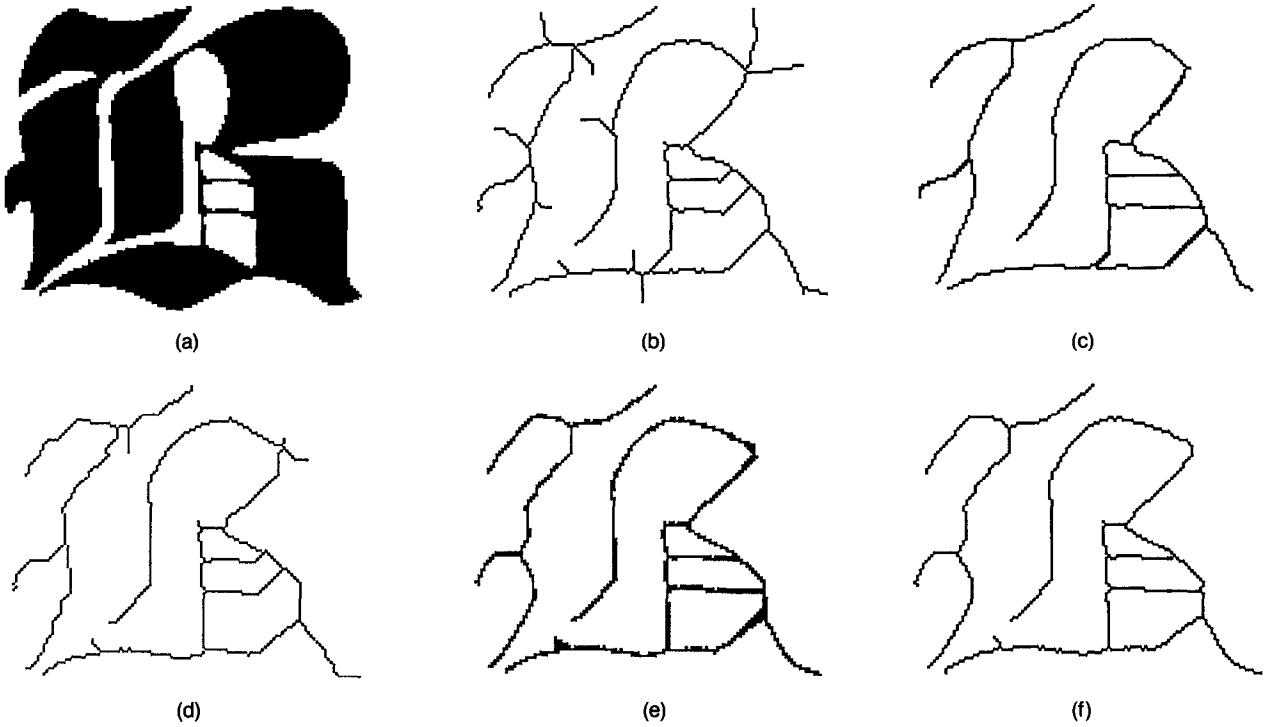

FIGURE 8. A 128 × 128 Bold Old English 形 (a) The Original Character Consists of Two Disjoint Objects, One with Three Holes (b) SPTA (c) Zhang/Suen Algorithm (d) Xu/Wang's CGT (e) Pavlidis's Contour Tracing (f) Contour Generation with Chain Codes

guaranteed in contour generation because of the nature of the algorithm, i.e., every point generated as a section of the new contour is tested 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 in 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

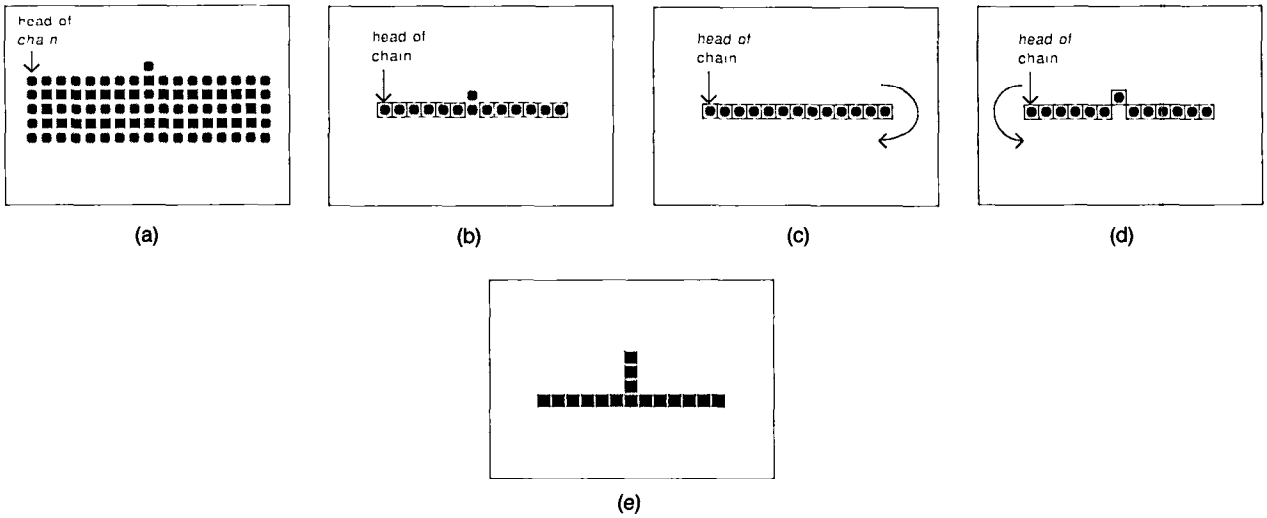

FIGURE 9. Lines with noise (a) A 5-pixel Wide Line with Noise. The Square Boxes Represent the Chain Generated at the End of the First Pass (b) The Chain Generated after the Second Pass. All but Two of the Pixels have become Safe Points (c) Final Skeleton with a Clockwise Chain (d) Final Skeleton with a Counterclockwise Chain (e) Skeleton Obtained from SPTA without Noise Removal

different processes. When a pixel is processed, mutual exclusion must be enforced on it and its neighbors to ensure connectivity and a unit-thickness skeleton.

CONCLUSION

Connectivity problems and thick skeletons experienced by many thinning algorithms often result when the outer layer of pixels of an object is removed and the structure of the resulting object is unknown as far as the current iteration is concerned. In the case of parallel algorithms, the solution has been to divide a pass into several subiterations or to obtain information about the neighbors of neighbors of a pixel. In the case of serial algorithms, constraints were introduced to ensure connectivity so that in some instances, 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, thereby ensuring connectivity.

By using a more relaxed safe point test, which is condition α of the multiple point test, the algorithm will not stop when two sections of a contour are neighbors, 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 performed only once during the iterations, and only edge points are visited. When pictures are digitized to a higher and higher resolution (made possible by the falling cost of memory and the increasing precision of digitization equipment) time complexity of thinning algorithms should be made independent of the product of the size of bitmap and number of passes. Contour generation represents a significant reduction in time complexity.

REFERENCES

1. Arcelli, C. Pattern thinning by contour tracing. *Computer Graphics and Image Processing 17*, 2 (Oct. 1981), 130–144.
2. Arcelli, C. A condition for digital points removal. *Signal Processing 1*, 4 (1979), 283–285.
3. 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.
4. Bel-Lan, A., and Montoto, L. A thinning transform for digital images. *Signal Processing 3*, (1981), 37–47.
5. Beun, M. A flexible method for automatic reading of hand-written numerals. *Philips Technical Review 33*, 4 (1973), 89–101.
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. Brinch Hansen, P. Concurrent programming concepts. *ACM Comp. Surv. 5*, 4 (Dec. 1973), 223–245.
8. Davies, E.R., and Plummer, A.P.N. Thinning algorithm, a critique and a new methodology. *Pattern Recognition 14*, 1 (1981), 53–63.
9. Dill, A.R., and Levine, M.D. Multiple resolution skeletons. *IEEE Trans. Pattern Analysis and Machine Intelligence PAMI-9*, 4 (July 1987), 495–504.
10. Freeman, H. On the encoding of arbitrary geometric configurations. *IEEE Trans. Electronic Computers EC-10*, (June 1961), 260–268.
11. Freeman, H., and Davis, L. A corner-finding algorithm for chain-coded curves. *IEEE Trans. Computers C-26*, 3 (Mar. 1977), 297–303.
12. Freeman, H., and Garder, L. Apictorial jigsaw puzzles, the computer solution of a problem in pattern recognition. *IEEE Trans. Electronic Computers EC-13*, (Apr. 1964), 118–127.
13. Hilditch, C.J. Linear skeletons from square cupboards. In *Machine Intelligence IV*, B. Mertzer and D. Michie, Eds. University Press, Edinburgh, 1969, 403–420.
14. Holt, C.M. et al. An improved parallel thinning algorithm. *Commun. ACM 30*, 2 (Feb. 1987), 156–160.
15. Lü, H.E., and Wang, P.S.P. A comment on "A fast parallel algorithm for thinning digital patterns". *Commun. ACM 29*, 3 (Mar. 1986), 239–242.
16. Moayer, B., and Fu, K.S. A tree system approach for fingerprint pattern recognition. *IEEE Trans. Comput. C-25*, 3 (Mar. 1976), 262–275.
17. Naccache, N.J., and Shinghal, R. SPTA: A proposed algorithm for thinning binary patterns. *IEEE Trans. Systems, Man and Cybernetics SMC-14*, 3 (May 1984), 409–418.
18. Pavlidis, T. *Algorithms for graphics and image processing*. Computer Science Press, Rockville, Md: 1982.
19. Pavlidis, T. A thinning algorithm for discrete binary images. *Computer Graphics and Image Processing 13*, (1980), 142–157.
20. Pavlidis, T. Filling algorithms for raster graphics. *Computer Graphics and Image Processing 10*, (1979), 126–141.
21. 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.
22. Pfaltz, J.L., and Rosenfeld, A. Computer representation of planar regions by their skeletons. *Commun. ACM 10*, 2 (February 1967), 119–125.
23. Rosenfeld, A. A characterization of parallel thinning algorithms. *Inform. Contr. 29*, 3 (Nov. 1975), 286–291.
24. Rosenfeld, A., and Pfaltz, J.L. Sequential operations in digital picture processing. *J. ACM 13*, 4 (October 1966), 471–494.
25. 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.
26. Stefanelli, R., and Rosenfeld, A. Some parallel thinning algorithms for digital pictures. *J. ACM 18*, 2 (April 1971), 255–264.
27. Tamura, H. A comparison of line thinning algorithms from digital geometry viewpoint. In *Proceedings of 4th International Conference on Pattern Recognition*, Kyoto, Japan, (1978), 715–719.
28. Xu, W., and Wang, C. CGT: A fast thinning algorithm implemented on a sequential computer. *IEEE Trans. Systems, Man and Cybernetics SMC-17*, 5 (September 1987), 847–851.
29. Zhang, T.Y., and Suen, C.Y. A fast parallel algorithm for thinning digital patterns. *Commun. ACM 27*, 3 (Mar. 1984), 236–239.

CR Categories and Subject Descriptors: I.5.2 [**Pattern Recognition**]: Design Methodology—*pattern analysis*; I.5.4 [**Pattern Recognition**]: Applications—*computer vision*

General Terms: Algorithms, Design, Theory

Additional Key Words and Phrases: Serial algorithms, skeletonization, thinning of digital patterns

ABOUT THE AUTHOR:

PAUL KWOK graduated from the University of Essex with a B.Sc. in 1976 and obtained a Ph.D. in electrical engineering at the University of Cambridge in 1979. He joined Monotype International in Cambridge in 1979, and worked on ideographic laser typesetting systems. Between 1981 and 1986 he was on the faculties at the Chinese University of Hong Kong and the University of Hong Kong. Since 1986, he has been an assistant professor in the Department of Computer Science, the University of Calgary, Canada. His research interest is in the areas of image processing and ideographic computing. He designed and implemented the first bilingual videotex system in Hong Kong. Author's present address: Paul C.K. Kwok, Department of Computer Science, The University of Calgary, 2500 University Drive NW, Calgary, Canada T2N 1N4.