A novel fast and reliable thinning algorithm

D Kalles and D T Morris

D. Kalles and D.T. Morris. "A novel fast and reliable thinning algorithm", *Image and Vision Computing*, Vol. 11, No. 9, pp. 588 - 603, **1993**.

In this paper a new algorithm is proposed for skeletonizing binary digital images. The algorithm does not employ the conventional pixel- or non-pixel-based techniques. Instead, it identifies regions of particular shapes in the image and substitutes apppropriate skeleton patterns for them. Initially, as many horizontal and vertical strips as possible are detected. These correspond to rather straight, long and narrow regions in the original image. Any remaining regions correspond to joints between strips. The strips are then grouped into trapezoidal regions which may be replaced by the appropriate skeleton patterns. The individual skeleton patterns are then merged using the skeletons due to the jointing regions, generating the final skeleton. A compact representation is obtained for post-processing, and the issue of robustness with respect to noise is addressed.

Keywords: image processing, thinning algorithm, skeletonization

The process of thinning is of great importance to many image-processing systems as it allows us to perceive the original image in a simplified way without relaxing our requirement to have access to its characteristics[1].

Thinning algorithms presented so far may be classified as iterative or non-iterative. Iterative algorithms can be based on contour following or raster scan techniques, and either 'peel' the exterior points (pixels) of regions in the image or start from the exterior points, compute a distance value and subsequently establish a range of values indicating which points ought to be preserved. Usually, a 3×3 sliding window is used ($k \times k$, $k > 3$ window algorithms have also been investigated, at a substantially increased cost though) to perform local checks and remove unwanted points. Conditions are established to prevent pixel deletion or postpone decisions to a later iteration.

Kwok[2] has proposed an algorithm that performs very fast thinning and functions well in noisy images (from now on we will classify as noise the various deformations of the contours of image objects, bearing in mind that the boundary between a noisy and a clean image is rather vague). It generates a contour to obtain a primary outline of the object to be thinned. It then iterates through the closed contours, transforming them into smaller ones until they are trivial (simple lines). The concept of point removal is eminent, but is used very efficiently since the algorithm has less points for examination in each iteration (the points retained from the previous iteration).

More recently, Xia[3] proposed a method based on the inspection of a fire-front's successive steps to extinction when it is started on the contour of an object and propagates towards the interior. It is based on the 'intuition' that two parts of the fire front moving separately should meet at skeletal points. It is claimed that the algorithm gives faster results than many existing algorithms. It is obvious, of course, that this method also employs iteration for examining which points are allowed to be present in the final skeleton.

Finally, Jang and Chin[4] came up with a theoretical treatment of thinning algorithms using mathematical morphology. In their work, various operations are defined on the demain of digital binary images. Thinning, then, is defined as a series of applications of operators. However, as some operators are defined to consist of iterative steps, their approach is classified as iterative.

There have been numerous iterative algorithms proposed to tackle the problem of thinning (see References 5–7 for some recent methods and Reference 8 for a comprehensive survey).

However, it would seem natural to devise an algorithm that uses a more abstract way of examining an image to obtain its skeleton. After all, a human does not view an image as pixels when trying to create a skeleton, but rather detects long and narrow strips

Department of Computation, UMIST, PO Box 88, Manchester M60 1QD, UK

Paper received: 28 September 1992; revised paper received: 8 February 1993

0262–8856/93/090588–16

quite easily, then spends some time thinking about skeleton patterns in areas where lines join. Of course, the connectedness criterion required of a thinning algorithm should be satisfied. Furthermore, reasonable noise elimination should be provided, and it should be pointed out that the algorithm's running speed should depend on the complexity of the image rather than it size (where complexity can be thought of as the rarity of continuous, straight lines).

An approach to skeletonization that was free from the concept of the iterative removal of image points has been examined[9]. There, it was assumed that by scanning the image from bottom to top various parts of the skeleton may be built as soon as elementary regions of the image are identified. This worked rapidly and well in most instances, but skeletonized incorrectly those regions which could not be readily identified as vertical or horizontal. It also failed to treat noisy images consistently due to lack of feedback.

The approaches documented[10–11] are descriptive of non-iterative thinning algorithms, though one of them[10] only provides a background for complete thinning and does not provide a skeleton. (See Reference 8 for a list of other non-iterative thinning algorithms.)

This paper is organized in seven sections. The first segmentation which provides a list of vertical strips, and the second one which identifies horizontal strips and strips with non-obvious shapes, are documented in the first two sections. As these segmentations provide thorough knowledge about how the image is partitioned into interacting regions, the third section documents how these regions are substituted by appropriate skeleton parts, and how these skeleton parts are combined towards the final skeleton. As a direct by-product of this process, a planar graph can be constructed which allows the employment of proven algorithms for many post-processing problems. The planar graph is discussed in the fourth section. The last sections present the experiments carried out to validate the algorithm, a discussion about is properties, and the resulting conclusions.

Throughout the paper, we will be using the terms *thinning* and *skeletonization* to describe the same process. Furthermore, we will be using the term *skeleton* to denote the final result of the thinning process (strictly, an image of one-pixel-wide regions).

PRIMARY SEGMENTATION

Definitions

A segment is defined as a set of contiguous black points belonging to the same row, such that the extreme points have only one neighbouring white point on the same row (as employed in the run-length encoding scheme). Two segments S_1 and S_2 are defined to be connected if and only if they belong to adjacent rows and there exist points $p_1 \in S_1$, $p_2 \in S_2$ such that p_1 and p_2 are 8-connected. A block B is defined as a set of one or more connected segments that satisfy the following rules:

1. A segment may belong to only one block.
2. For any segment $S \in B$, there may be only one segment $S_u \in B$, such that S_u belongs to a row above S, and S and S_u are connected. If this is the case, there may not exist any other segment S_{u_a} belonging to a row above S, such that S and S_{u_a} are connected. Additionally, there may be only one segment $S_l \in B$, such that S_l belongs to a row below S and S and S_l are connected. If this is the case, there may not exist any other segment S_{l_a} belonging to a row below S, such that S and S_{l_a} are connected. This rule will be referred to as the simple-path rule.
3. The polygon representing the contour of the figure obtained by inspecting the set of black points must be convex (conventional geometric definition). This rule will be referred to as the convexity rule.

Strict adherence to these definitions allows blocks of very general shapes. A simplifying rule is therefore added to constrain the allowed shapes:

4. A block is confined to a segment or an uninterrupted range of rows of the image and the vertices of the contour polygon may only be on the uppermost or the lowermost segment of the block.

It can be shown that a block may be a single point, a single line, a triangle, a trapezoid or a parallelogram. In addition, a block can be represented by at most four points, these points being the corners of the corresponding figure. This representation is still general and is actually needed for trapezoids and parallelograms only (all other figures are trivial cases).

Two blocks B_u and B_l are defined to be adjacent if and only if there exist segments $S_u \in B_u$ and $S_l \in B_l$, such that S_u and S_l are connected. By the definition of a block, the following conditions hold for any two adjacent blocks:

1. Adjacency is defined in the vertical sense only. This means that if there exist connected blocks B_u and B_l, then as a consequence of the simple-path rule, there is no range of rows common to these blocks. (Refer to *Figure 1* for a clarification of this condition.)
2. Any two adjacent blocks may only touch each other via a pair of connected segments of which the upper segment belongs to the upper block and the lower segment belongs to the lower block. Violation of this condition implies that at least one of the blocks will break both the convexity rule and the simple-path rule.

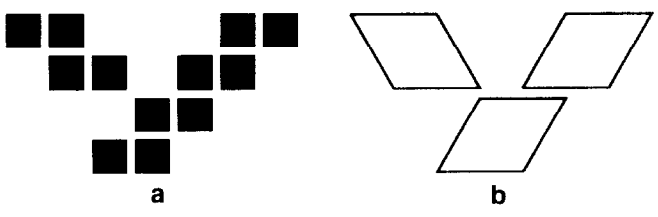


Figure 1 According to the rule the configuration in (a) will deliver the blocks in (b)

The extraction of blocks is achieved by identifying a segment as the lowest one of a candidate block and accumulating further segments while the rules that define legitimate blocks allow it. If for a given segment S there exist no segments on a row below it connected to S, then S is a start segment (the lowest segment of a block). Similarly, if for a given segment S there exist no higher connected segments, then S is a stop segment (the uppermost segment of a block). Start and stop segments are also formed when segment accumulation would violate the simple-path rule.

Block building

A data structure is associated with each block being built. This contains information about the segments making up the block. It allows us to monitor the shape of the block to ensure that the convexity rule is obeyed, and that a realistic segmentation is being made. Connected segments are linked together.

The block building process is carried out by examining pairs of adjacent rows, starting from the lowest row of the image. A segment in the upper row will either be piled above a segment in the lower row for subsequent or immediate block building, or an unpaired segment in the upper row will cause the initiation of a new candidate block. Comparison of the segments is based on examination of their locations on the x-axis (this approach has been also documented elsewhere[9]). The procedure followed when the simple path rule holds is described now (it will be referred to as the piling procedure):

Piling procedure

A candidate block's shape is defined by the segments already piled on it. Depending on how much this shape would change by adding a connected segment to the block, two possibilities arise. If the new segment suggests that the shape will change slightly, then it is added to the pile. If the shape is significantly affected, the candidate block is completed and a new candidate block is defined with the upper segment as its start segment. The connection between the new block and the new candidate block is recorded.

The decision whether to use the new segment in an existing block or a new one is made using the concept of 'acceptable displacemncts'. For a given pile the endpoints of a new segment are estimated by fitting straight lines to the pile's sides, thus determining the best expected new upper segment. If the new segment lies fits in the expected range, then it is piled. If it does not, an error is established between the estimated shape and the shape obtained by adding the new segment to the pile. The error is computed from the areas of the shapes and may occur on either side of the shape (refer to *Figure 2* for an example). If the error exceeds a given threshold, then the shape is broken up, otherwise the new segment is piled. The break-up is achieved by initiating a new candidate block and constructing a new block.

For the implementation, the allowed error was 7.5% on any side of the estimated shape and for efficiency reasons we allow a block's shape to be determined by its extreme segments only. It is clear that a block's estimated shape is dynamically changing as new segments are piled on it.

A more compact block representation

A compacted representation may be created by judiciously merging adjacent blocks into larger ones. It must be ensured that the new blocks' configuration is in accord with the rules governing legitimate blocks. A variant of the simple-path rule is used to achieve this:

> Two blocks B_u and B_l such that block B_u lies above block B_l may be merged (actual merging criteria will be presented below) if there does not exist a block B_{u_u} such that blocks B_{u_u} and B_l are adjacent and block B_{u_u} lies above block B_l and there does not exist a block B_{l_l} such that blocks B_{l_l} and B_u are adjacent and block B_{l_l} lies below block B_u.

Two adjacent blocks B_u and B_l that qualify for

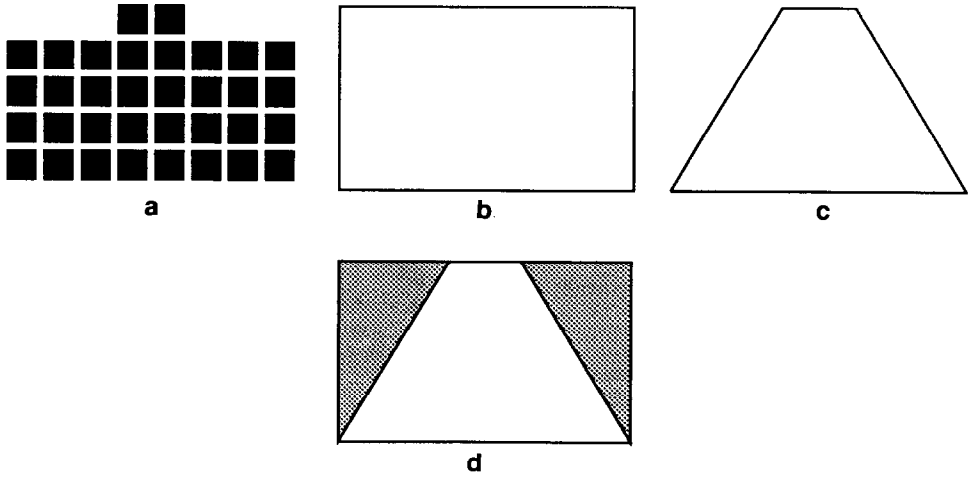


Figure 2 (a) Original image; (b) anticipated shape; (c) shape obtained if the new segment is added; (d) approximation errors (shaded regions)

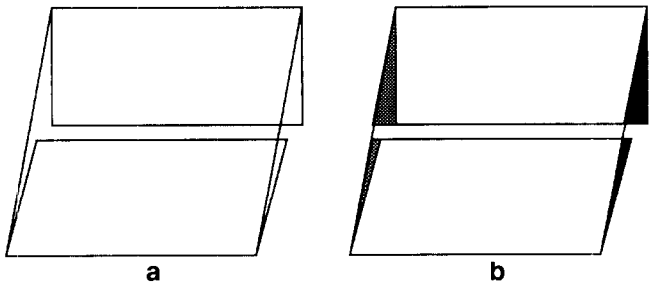


Figure 3 (a) Original blocks with anticipated block after merging; (b) approximation errors due to inclusion of white area (lightly shaded region) and exclusion of actual block area (heavily shaded region)

possible merging may be merged if this does not greatly affect the image representation. If the two blocks are merged into block B_{ul}, define block B_{ul} to inherit the upper corners and adjacencies of block B_u and the lower corners and adjacencies of block B_l. Approximation errors may be introduced due to parts of the original blocks being omitted as well as due to inclusion in the final shape of points not belonging to the original blocks. In the actual implementation the allowed error was 15%. (*Figure 3* illustrates this case.)

The merging process is performed for all pairs of qualifying blocks, ensuring that the same pair is not checked twice. Some pairs of blocks may become eligible for merging once surrounding blocks have been merged as this may reduce the approximation error to an aceptable amount (*Figure 4*).

Another useful post-processing step involves a very relaxed elimination process. A block B may be eliminated if and only if it has one adjacent block B_a (either upper or lower) that is much bigger and has a comparatively large common boundary with B_a. Currently, at least one of B's horizontal bases should have an overlap of at least 80% with one of B_a's horizontal bases. Furthermore, define X_B as B's largest base and height, respectively. Then block B_a is considered 'much bigger' if the conditions $max(B_B, Y_B) < 0.1 \times max(B_{B_a}, Y_{B_a})$ and $min(X_B, Y_B) < 0.1 \times min(X_{B_a}, Y_{B_a})$ hold simultaneously. It should be obvious that the elimination process exists only to remove noise which is easily identified above or below big blocks.

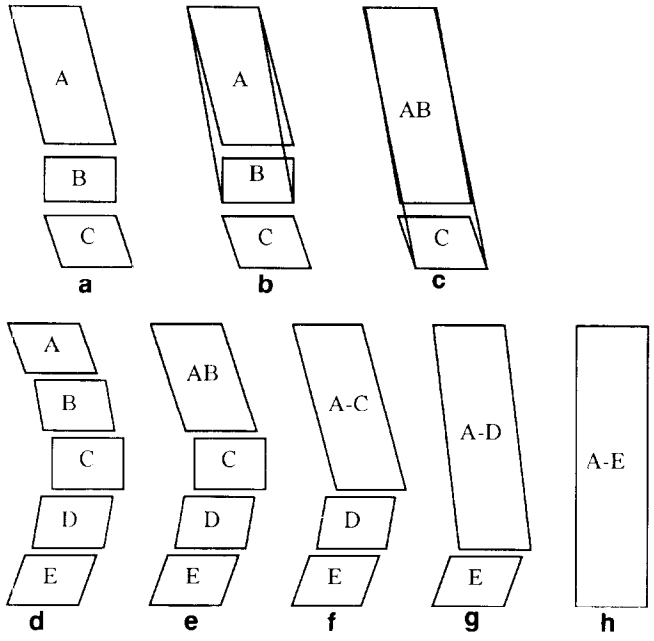


Figure 4 (a–c) What happens if blocks B and C do not qualify for merging but blocks A and B do: (a) original configuration; (b) resulting configuration if pair (B, C) is checked first; (c) resulting configuration if pair (A, B) is checked first and pair (AB, C) is checked next. However, the series of configurations (d–h) is not feasible (it is clear that the result fails to capture the original geometry). For example, at step (f), if the error of merging blocks D and E is sufficiently small, we will have a compound block DE for the next stage, otherwise (due to the change in direction) it is not possible that the candidate pair ($A-C$, D) would have a better score than pair (D, E). It should be obvious that it is up to judiciously selected error thresholds to tune the algorithm's sensitivity to allowed geometric deformations

Block classification

Our main concern now is to be able to identify individual blocks as vertical strips, as straight horizontal strips or as piles of non-vertical strips.

Vertical-shaped blocks represent quite simple lines of the image's skeleton, but care must be taken to their connections with upper and lower adjacent blocks. Similarly, non-vertical blocks that are connected with vertical blocks only or with no blocks at all, are very likely to be simple lines of the skeleton on their own. It is the mutual adjacencies of non-vertical blocks that pose a difficult problem. Big overlapping between such blocks might suggest a common contribution to the skeleton. But, to further complicate the problem, these types of adjacencies have no fixed depth, therefore it is not possible to identify two blocks with a skeleton portion at this stage. (The term adjacency depth may be considered as the number of consecutive adjacent blocks, from upper to lower, or *vice versa*.)

A primary classification of blocks according to their shapes occurs whenever a block is built, regardless of the fact that this building might be a result of the segmentation process or of the merging process. Trapezoids and parallelograms have been selected as the most suitable abstractions for all blocks. According to this convention, a block that looks like a triangle will be classified as a trapezoid, whereas simple points and simple lines will be classified a parallelograms.

As parallelograms will be rather hard to find in non-ideal images, most shapes are likely to be classified as trapezoids. However, parallelograms lend themselves to more efficient handling, therefore an attempt to identify as many parallelograms (or near parallelograms) as possible is made. Some very oblique trapezoids may be classified as parallelograms, but the error due to possible loss of information or unrealistic representation is small.

By inspection, the following rule was formulated for classifying blocks (refer to *Figure 5* for examples):

Detect the leftmost and rightmost corners of block B and compute the smallest possible surrounding rectangle R_B. Obtain D_B, the difference in length

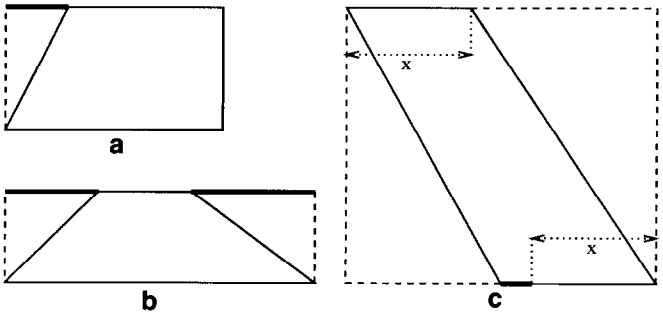


Figure 5 The dashed line shows parts of the blocks' surrounding rectangles. The thick line shows the length of the line segment corresponding to the difference of a block's horizontal bases. (a) Depending on the allowed ratio value, this block could be safely regarded as either trapezoid or parallelogram; (b) clearly, this bock must be considered as trapezoid; (c) the bases' difference is far smaller than the perimeter, therefore this block can be safely regarded as a parallelogram

between the block's horizontal sides and compare it with P_{R_B}, the perimeter of rectangle R_B. If the ratio of P_{R_B} to D_B exceeds a predetermined value (the value used was 10%), then the block is assumed to be a parallelogram rather than a trapezoid.

Once a block has been classified as a trapezoid its orientation can be found by determining if the upper base is the smaller one or not. Trapezoid blocks are classified as non-vertical blocks and are liable for more detailed post-processing.

Simple points and lines have a zero difference between their horizontal bases and thus are classified as parallelograms.

If a parallelogram's height is more than twice its width, the block is given a vertical status. Conversely, if the width is more than twice its height, the block is horizontal. Should neither of these options be selected, the block is assigned an undetermined-shape status.

A block's name is defined as the combination of features describing its shape (i.e. the block's type and orientation).

The process of changing the names of existing blocks should ensure that if a block's name is changed, subsequent restoration of the original name is prevented. Furthermore, if a block changes its name, the adjacent blocks are liable to change their names as well.

Defining which blocks are allowed to have their names changed is governed by the following rules:

1. Vertically oriented blocks' names are permanent. Vertical blocks cannot be influenced by any adjacencies.
2. To avoid arbitrary renaming, only blocks with one adjacent block may be renamed. For example, a block with a single upper connection is liable to have its name changed due to that connection.
3. Trapezoids may only have their name changed due to connections occurring at their larger base.

If a block is liable to have its name changed due to any adjacencies, then it is added to a queue set up for this purpose. When a block is taken from the queue it is examined to see if it has already been checked for a name change. If this is the case it is discarded and the next block in the queue is processed. If the block has not been checked it must be ensured that checking is allowed according to the above rules. Note that the queue is initialised with blocks liable for a name change but, it may be expanded by adding any non-checked block. If a name change is allowed, then an adjacent vertical block able to enforce the name change must be found. Such a block will have a significant common boundary with the original block. If these conditions hold then the original block will be renamed as a vertical block. Note that this change may be effected by any of the original block's single vertically adjacent blocks, either upper or lower. The block must be flagged as having been renamed and its adjacent blocks in the opposite direction from where the name change was effected must be checked for possible name changes.

Blocks of undetermined-status are processed in a separate pass through the blocks' list (note that new name changes may be required for some non-vertical blocks if one of their adjacent blocks has been changed from undetermined-status to vertical).

SECONDARY SEGMENTATION

Requirement for a secondary segmentation – definitions

It would initially appear that detecting the overlapping sides of adjacent non-vertical blocks was a comparatively simple problem to solve: it is required to determine whether a given block's corners lie within the range of columns of its adjacent non-vertical block B and within the range of columns of one of block B's adjacent non-vertical blocks. However, not all overlaps in the resulting configuration may be so simple (*Figure 6*).

The solution devised is to record the columns corresponding to corners of existing blocks. Neighbouring recorded columns are then examined and the strips between them are inspected to detect which original blocks they pass through. By assuming a population of n blocks, a naive algorithm to detect which blocks span a specific column would require $O(n)$ steps. If this check is to be carried out for all of a block's columns and for all blocks, it follows that in order to obtain a list of the intersections, a computational complexity of

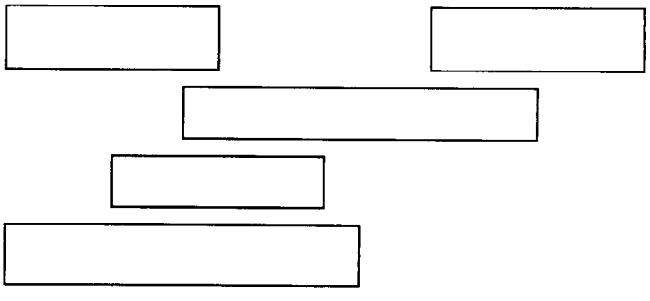


Figure 6 A continuous overlap is defined as a set of overlaps over a set of adjacent blocks

$O(n^2)$ should be anticipated. This computational load can be reduced to $O(n \log n)$ by using a variation of a data structure called the interval tree[12]. To set up and use the structure it is essential that there exists the knowledge about the segments involved and the columns that questions will be asked about. To determine whether a column is included within a block's range of columns the block's leftmost and rightmost corners only need be known, however, it is also desirable to ask questions about the columns of the block's other corners.

For each non-vertical block B, the following steps are taken:

1. The block's adjacent blocks are examined. If they are all vertical they are discarded along with block B and the process restarts with a new block.
2. The overlap between the shared sides of block B and an adjacent block B_a is examined. If the overlap is at least two points, B_a is marked as elected, otherwise it is discarded B is marked as checked. Note that, even if some of B's adjacent blocks are selected they have not been checked; they may later be used to mark other blocks for selection. Selecting a block consists of recording its corners for the interval tree and adding its leftmost and rightmost corners to a list of segments, each segment associated with the corresponding block.

Overlaps with a length of less than two points are recorded but the corresponding blocks are not marked as selected, instead the blocks are kept separately for later processing. The blocks may appear in the interval tree if they are clearly overlapped by other blocks.

The interval tree that has been built will be used for the plane-sweep process that follows and generates the new configuration. In this configuration the basic element will be referred to as the sweep-block.

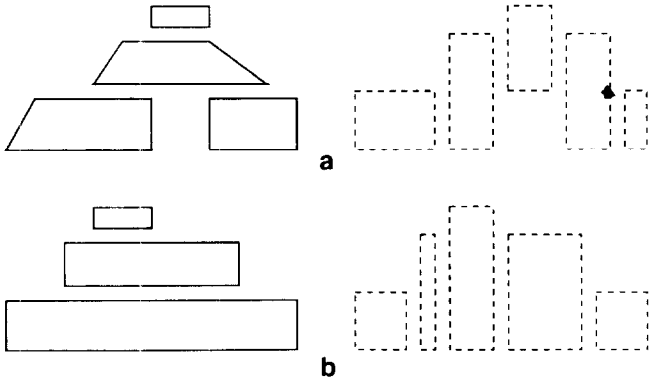


Figure 7 Figures in dashed lines correspond to the new configuration obtained by the original configurations in the figures in continuous lines

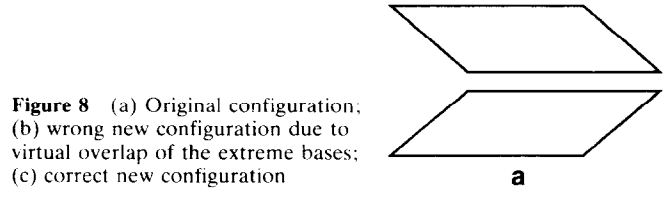


Figure 8 (a) Original configuration; (b) wrong new configuration due to virtual overlap of the extreme bases; (c) correct new configuration

A sweep-block is a rectangle divided into horizontal strips. Each strip corresponds to a block. In order for a block to be associated with a strip, it must larger horizontally and smaller vertically than the sweep-block (*Figure 7*). The sweep-block is a way of defining partial overlaps: adjacencies between sweep-blocks suggest successive partial overlaps, these being part of major inter-block overlaps. Whilst this is generally satisfactory, care must be taken of virtual overlaps which should be detected and treated separately (*Figure 8*).

Sweep-block building

The blocks intersected (cut) by a specific column (the term sweep-point will be used to indicate such a column) must now be identified. They are placed in a list and sorted with respect to their vertical position in the image. At this point, the only evidence for a possible sweep-block is that the sweep-point is included in its range. As the inclusion rule must hold for all blocks associated with that candidate sweep-block, the objective becomes to detect consecutive lists of consecutive blocks, each list suggesting a candidate sweep-block (*Figure 9*).

The algorithm to build these lists is quite simple. Each of the blocks is checked to see which edges are cut by the sweep-point. If both are cut then block may be added to the current list. If only the upper edge is cut then the current list should be terminated after adding the current block and a new list should be initiated. Similarly if only the lower edge is cut, then the current list is terminated and a new list is initiated with the current block as its first one.

The purpose of comparing these lists is to identify adjacent vertical strips (or a strip) which, when piled together, will make up a sweep-block. It would be expected that when two lists have some blocks in common, the common blocks could be associated with the candidate sweep-block; then the range of rows of

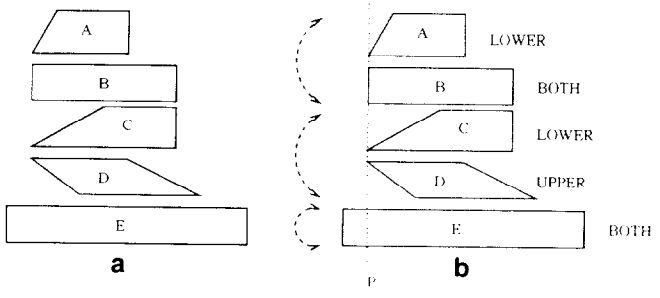
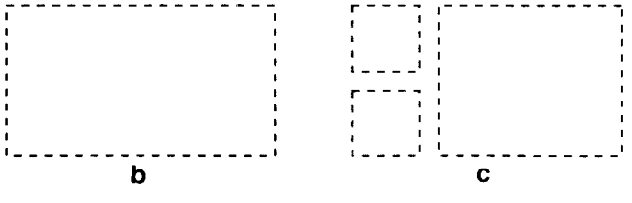


Figure 9 (a) Original configuration; (b) the text on the right of each block shows the way the block is cut by the vertical (dotted) line at sweep-point P, whereas the dashed arcs represent the lists of blocks



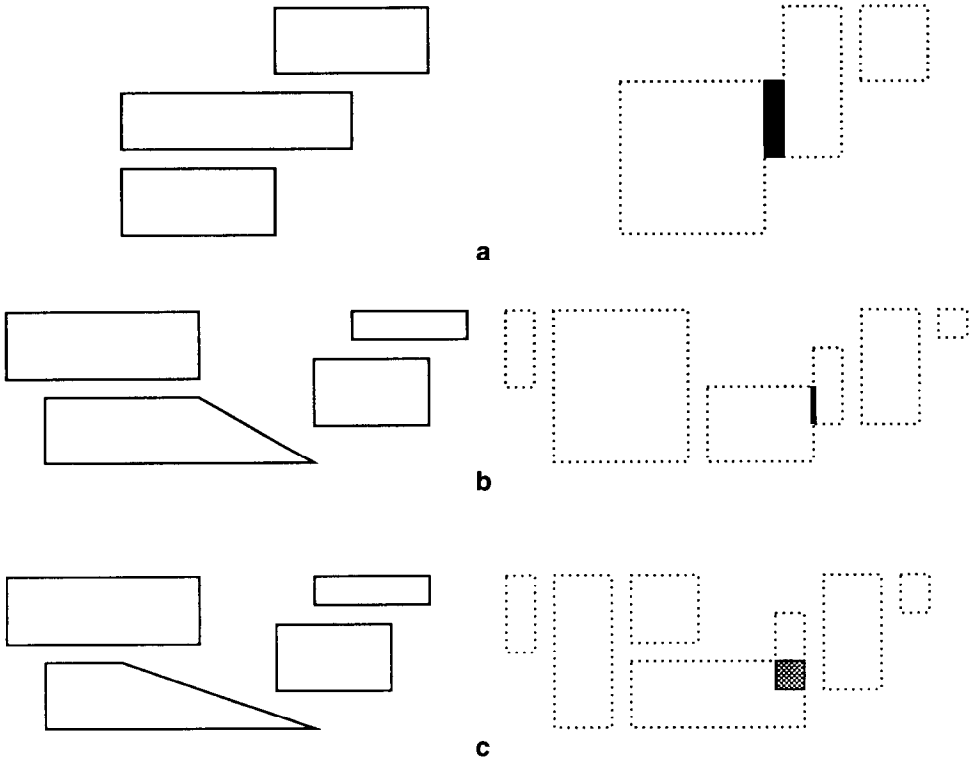


Figure 10 The blocks' configuration is shown on left-side figures whereas the sweep-blocks' configuration appears on right-side figures. (a) Correct adjacency (shown by the thick line); (b) incorrect adjacency (shown by the thick line); (c) incorrect adjacency (shaded region shows the overlap of sweep-blocks)

this sweep-block would be the range of rows of the common blocks. However, as there may be more than one vertical strip making up the final sweep-block a method similar to the first segmentation will be employed, but now block-lists of successive strips will be compared. Sweep-blocks with a *width/height* ratio of more than 1.25 are considered horizontal whereas the rest are considered as vertical.

As the sweep-blocks are virtual structures there may be some overlaps between their corresponding rectangles without an actual adjacency between these sweep-blocks. This point is illustrated in *Figure 10*.

Adjacency between vertical sweep-blocks therefore requires sophisticated treatment. For each vertical sweep-block SB_{v1} it is interesting to know if there exists any vertical adjacent sweep-block SB_{v2} such that all blocks associated with SB_{v1} are also associated with SB_{v2}. Should this be the case, it may be said SB_{v1} injects SB_{v2}. Injections can occur on both sides of a vertical sweep-block, but a sweep-block may inject only one sweep-block to each side. Injections are defined for vertical sweep-blocks only: horizontal sweep-blocks do not present any risk as they correspond to clearly defined horizontal parts of the image.

Let us assume that a sweep-block SB_1 is injecting sweep-block SB_2 which in turn is injecting sweep-block SB_3. It would be very useful to represent the fact that there exists an artificial injection directed from SB_1 to SB_3. Actually, the most important relevant information concerns the first sweep-block artificially injected by SB_1 that does not itself perform any injections. Such a sweep-block is called a major sweep-block and plays the role of clustering 'minor' vertical sweep-blocks.

As a major sweep-block indicates a cluster centre for other vertical sweep-blocks, a move up a level of abstraction is performed by establishing how the major sweep-blocks are related to each other. The objective is to detect the regions of the image where many line joints occur or where a joint has a very complicated shape.

Two major sweep blocks are virtually adjacent if they are connected to each other and no other major sweep blocks by a path of vertical sweep blocks. There are several types of virtual adjacencies. Assume that there exist major sweep-blocks SB_1 and SB_2 such that sweep-block SB_1 lies to the left of sweep-block SB_2:

1. A virtual adjacency can be an actual adjacency (*Figure 11a*).
2. If there exists a sweep-block SB_3 such that its left injection points to sweep-block SB_1 and its right injection points to sweep-block SB_2, then there exists a virtual adjacency between the two major sweep-blocks (*Figure 11b*). The descriptor of the virtual adjacency is a pointer to sweep-block SB_3 (such a sweep-block will be referred to as a common feeding sweep-block).
3. If there exists a sweep-block SB_3 such that its left injection points to sweep-block SB_1 and it is adjacent to sweep-block SB_2, then there exists a virtual adjacency between the two major sweep-blocks that is described by the pair (SB_3, SB_2) (*Figure 11c*).
4. If there exists a sweep-block SB_3 such that its right injection points to sweep-block SB_2 and it is adjacent to sweep-block SB_1, then there exists a virtual adjacency between the two major sweep-blocks that is described by the pair (SB_1, SB_3) (*Figure 11d*).
5. If there exists sweep-blocks SB_3 and SB_4 such that

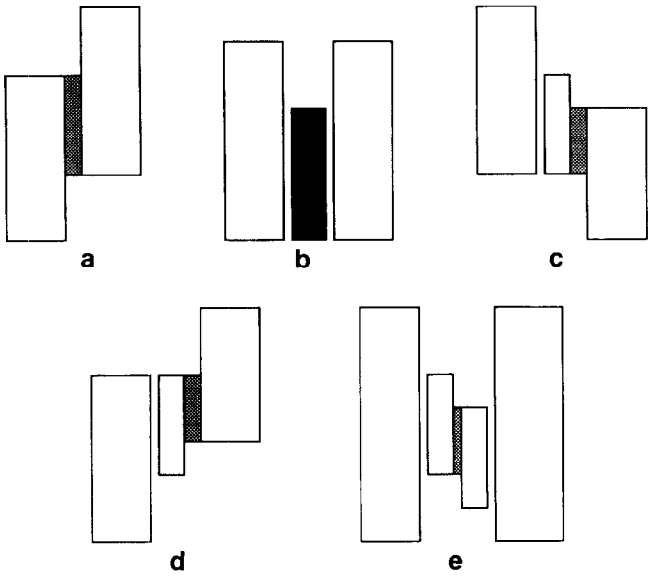


Figure 11 Types of virtual adjacencies between major sweep-blocks. Common feeding sweep-blocks are denoted by heavily shaded regions, whereas lightly shaded regions denote pairs of adjacent sweep-blocks

the left injection of sweep-block SB_3 and the right injection of sweep-block SB_4 point to sweep-blocks SB_1 and SB_2 respectively, then the pair (SB_3, SB_4) describes the virtual adjacency between the two major sweep-blocks (*Figure 11e*).

By observing successive virtual adjacencies, vertical shapes may be detected suggesting that the major sweep-blocks involved in these adjacencies are actually sharing the part of the image's skeleton corresponding to that region (*Figure 12*). If successive virtual adjacencies could be combined, the conclusion would be the detection of the primary skeleton patterns for such regions. Thus, the concept of the adjacency path must be introduced to formalize the above intuitive assertion.

An adjacency path is defined as a list of successive overlapping virtual adjacencies; its representation also includes references to connecting adjacency paths. The orientation of the extreme virtual adjacencies of an adjacency path indicates whether a vertical or horizontal skeleton pattern is appropriate for this region. Building of the adjacency paths is carried out by inspecting the virtual adjacencies of every major sweep-block, detecting their overlaps and thus deciding how each of them will affect the existing configuration.

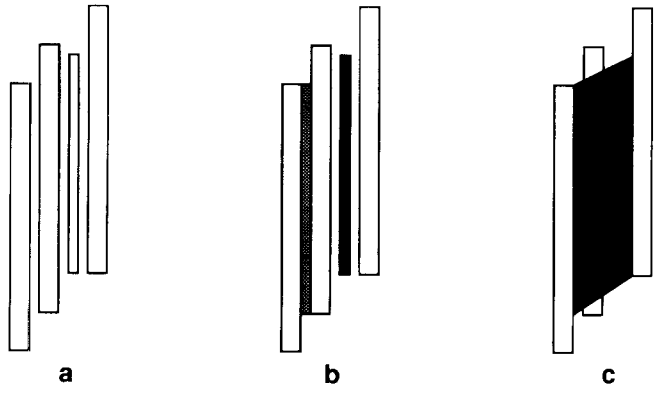


Figure 12 (a) Original sweep-blocks' configuration; (b) virtual adjacencies detected (descriptors are of both types); (c) suggested vertical shape in the middle

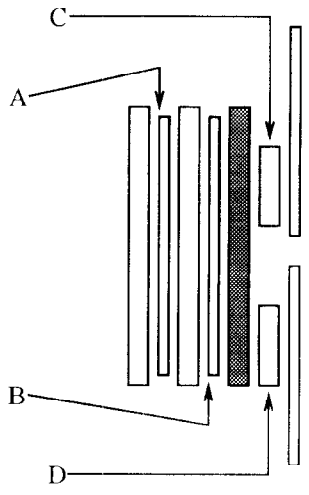


Figure 13 There are four virtual adjacencies described by the common feeding sweep-blocks A, B, C, D. B is the left virtual adjacency of the current (shaded) major sweep-block. This means that it cannot be on the same adjacency path with any of the virtual adjacencies C, D but may be on the same adjacency path with virtual adjacency A

Two rules are used as guidelines for building legitimate adjacency paths:

1. The shape obtained by inspecting the virtual adjacencies of a path must be convex or nearly convex.
2. A virtual adjacency belonging to an adjacency path may not overlap more than one virtual adjacency associated with the current major sweep-block unless it is the adjacency path's extreme virtual adjacency (refer to *Figure 13* for an illustration of this point).

Every major sweep-block has two lists of virtual adjacencies associated with it, one for each side. Virtual adjacencies in one side do not overlap virtual adjacencies in the same side. Two pointers are used, one for each list and decisions are always made based on the current pair. If for a given pair of virtual adjacencies both rules are obeyed, the two virtual adjacencies are incorporated in the same adjacency path, otherwise they are allocated to different paths and record the connection between these adjacency paths. The leftmost virtual adjacency may either be appended to an existing adjacency path or initiate a new one whereas the rightmost virtual adjacency will initiate a new adjacency path at any time.

After having built all the adjacency paths the skeleton pattern suggested by each of them is examined. If it is horizontal (*width/height* ratio of at least 4/3), it is ignored and all the virtual adjacencies belonging to the adjacency path are marked as horizontal. If the suggested skeleton pattern is vertical it is computed and associated with all the virtual adjacencies belonging to the adjacency path.

BUILDING AND COMBINING THE SKELETON PARTS

Definitions

For representation purposes, the skeleton of the image is a collection of elementary line segments. A line is defined to be a group of contiguous line segments.

A line will be either vertical or horizontal. By definition, in a horizontal line, two points may not share the same column. The leftmost point is considered as the starting point of the line. The line may only pass rightwards through eight-connected points (compare a horizontal line to the graph obtained by any function $f: \mathbb{R} \rightarrow \mathbb{R}$). The converse is true for vertical lines.

A simple representation of a line could be an array of points ordered with respect to their key attribute (for a vertical line the key attribute is the row of a point). This representation would make it easy to locate a point but quite hard to add a new one in the middle of the line. Specifically, should the line contain n points an average $O(n)$ time would be required to add a new point to the list.

A better representation of the line is provided by using a data structure called the AVL-tree (for an extensive treatment of AVL-tree refer to Reference 13). The AVL-tree is a variation of the binary balanced tree and will allow us to perform the addition of a new point in the line structure in $O \log n)$ time. The two endpoints of the line are also recorded in the line structure (actually, the nodes whose key values are the key attributes of the endpoints are recorded). The AVL-tree structure can be set up in $O(n)$ time. Other modifications to a line are also facilitated by the AVL-tree.

The simplest modification to a line is deletion, in part or totally. Deleting the whole line does not mean that all points included in the line will definitely be absent from the final skeleton: the endpoints may be part of another line. Points within the bulk of a line may not be deleted since this would break the line, but the line may be shortened from either endpoint.

To deal with deletion the definition of a point is enhanced with an attribute showing whether the point is definitely to be deleted, preserved or is indifferent. Only endpoints may be deleted, all points may be preserved.

All points are initialized as indifferent, that is they are not definitely to be preserved or deleted. Shortening a line may be effected by marking one of its endpoints for deletion. Since the points contributing to a joint between lines must, in most cases, be preserved, the points between a deletable, non-preservable endpoint and the first joint may be deleted. This means that occasionally deletable endpoints will also be preserved.

Building skeleton parts for vertical sweep-blocks

Building the skeleton corresponding to the vertical sweep-blocks is a process based on the individual examination of each major sweep-block. By working around a major sweep-block all the vertical sweep-blocks that inject it may be considered. Having recorded all the virtual adjacencies and their respective adjacency paths decisions about the skeleton in this region of the image may be made. The objective is to be able to compute a basic skeleton for a major sweep-block and then examine how its virtual adjacencies and all injecting sweep-blocks will affect it.

All adjacency paths that are to be considered must be recorded. There are two separate lists for each type of path, corresponding to the two vertical sides of the major sweep-block. The lists are combined and segmented into zones, ranges of rows influenced by one of the major sweep block's virtual adjacencies. There are four different types of zones that might occur in any major sweep-block: no virtual adjacencies, virtual adjacencies to the left, right or both sides of the zone. By definition, two adjacent zones cannot be of the same type.

The major sweep block's initial skeleton is a vertical line. It is modified by the results of examining all the blocks associated with the major sweep-block some of which are also associated with vertical sweep-blocks injecting the major sweep-block. For each associated block the range of columns over which vertical sweep-blocks injecting the current major sweep-block is detected. A simple interpolation technique is used to compute the points of the basic skeleton that correspond to each block. Two problems arise with this approach.

Firstly, when two neighbouring zones suggest distinct adjacency paths. It would seem that within these zones no part of the skeleton should be based on the basic skeleton, since the adjacency paths clearly represent a skeleton for the region. However, as this might affect the connectedness of the skeletonized image, a suitable point of the basic skeleton is selected and inserted between the corresponding endpoints of the adjacency paths' skeleton lines.

The second problem arises when a zone is described by two adjacency paths. The same solution can be adopted since the method of generating adjacency paths guarantees that one is present in the zone above the present zone and the other in the zone below. *Figure 14* illustrates these points.

At this point a major sweep-block has been divided into zones containing either portions of the sweep block's skeleton or portions of the sweep block skeleton modified by adjacency paths. For example, if a vertical adjacency path is big enough to suppress the corresponding major sweep-block, the correspnding endpoints of the major sweep-blocks covered by a virtual adjacency belonging to that adjacency path may have to be marked as deletable (*Figure 15*).

A further problem occurs if the possibility of vertical sweep-blocks having a long path to the major sweep-block they are injecting is not taken into account. In this case, the orientation of the path could be horizontal and a new, horizontal line joining and distorting one of the major sweep-block's skeleton parts should be introduced. An algorithm to detect all clusters of

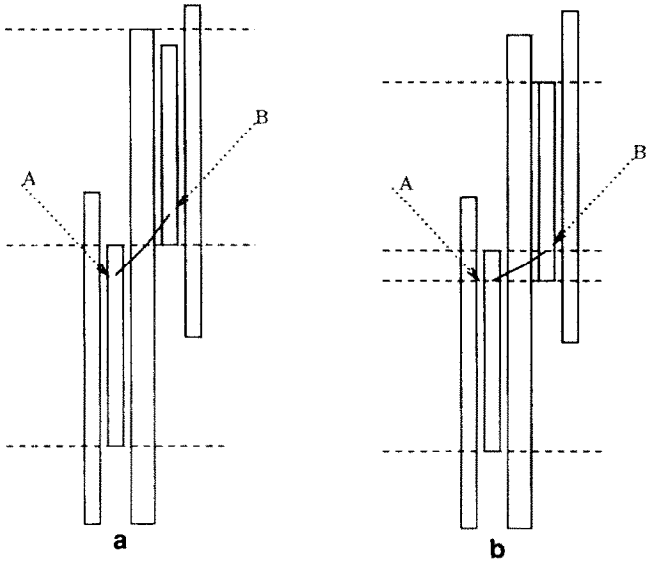


Figure 14 The continuous line is the skeleton part built beween the endpoints of the adjacency paths' lines. (a) Non-overlapping zones; (b) over-lapping zones

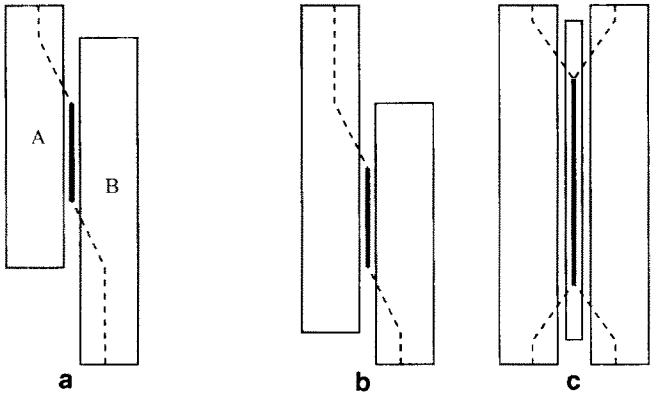


Figure 15 The thick line corresponds to the line created out of the vertical adjacency path, and the dashed lines correspond to skeleton parts of the major sweep-blocks. (a) It seems that the virtual adjacency suggests that the upper part of the skeleton parts of major sweep-block A could be safely discarded because it is rather horizontal. In contrast, the lower skeleton part of major sweep-block B is indispensable; (b) the opposite situation might also arise; (c) the only line actually qualified to stay is the adjacency path's skeleton line

vertical sweep-blocks injecting a major sweep-block, and rules to represent as many distortions as possible have been derived.

The procedure describing how clusters are built shall now be presented with reference to the left side of a major sweep-block SB. The techniques presented are identical for the processing of clusters on the right side of the sweep-block.

Of the various vertical sweep-blocks whose right injections point to sweep-block SB we can select the leftmost ones (these will not be injected by other sweep-blocks, of course). The clustering procedure is now initiated on each of the detected leftmost blocks.

Clustering procedure

Starting from the current vertical sweep-block step rightward between adjacent vertical sweep-blocks until sweep-block SB or a vertical sweep-block injected by more than one vertical sweep-block is met (note that this rule is actually another variation of the simple-path rule). If the major sweep-block SB has been reached, it is discarded; otherwise the sweep-block that has been reached is placed in a queue for later processing.

Throughout the walk the overall shape suggested by the successive sweep-blocks of the cluster is monitored. If at any point the suggested shape becomes horizontal, the cluster's type will be horizontal. Sweep-blocks suggesting that the cluster's shape is vertical and not horizontal may follow, but this does not matter since the objective is to detect potential horizontal shapes. If the cluster is completed without being labelled as horizontal, it is named either vertical or undetermined.

A cluster C_1 is defined as injecting cluster C_2 if and only if the rightmost sweep-block of cluster C_1 is adjacent to the leftmost sweep-block of cluster C_2 (this definition holds for the left side of the major sweep-block; a similar one may be made for the right side).

When the clustering procedure has been completed for all the leftmost sweep-blocks, sweep-blocks are removed from the queue and used as initial sweep-blocks of new clusters.

Recalling that all vertical sweep-blocks have been accounted for in some part of the major sweep-block's basic skeleton, this means that some of the vertical clusters may be discarded. Specifically, all leftmost vertical clusters are to be disposed. However, removal of a cluster may imply the removal of other clusters. For example, suppose that there exist vertical clusters V_1, V_2 and V_3 such that cluster V_3 is injected by clusters V_1 and V_2 only. If clusters V_1 and V_2 are removed then cluster V_3 must be disposed of too. Note that a cluster of horizontal or undetermined-shape entitles all vertical clusters to its right to stay in the configuration.

Having removed useless clusters a higher level of clustering can be achieved by examining the remaining clusters to see if they may be grouped. For example, consider a configuration with vertical cluster V_1 and horizontal clusters H_1 and H_2 such that cluster H_2 in injected by clusters V_1 and H_1 only. If cluster Vd21 is deleted, then clusters H_1 and H_2 could be merged to yield a new horizontal cluster (the new cluster's rightmost sweep-block confirming a horizontal shape would be cluster H_2's rightmost sweep-block). To perform the merging the clustering procedure is used with the modification that the building elements are now the clusters themselves.

An undetermined-shape cluster C is picked and a path of clusters up to the major sweep-block constructed. If on this path there exist any other undetermined-shape or horizontal clusters, then cluster C is named horizontal, otherwise the range of rows of its rightmost sweep-block is recorded. In a similar manner, the range of rows of the final legitimate sweep-block of a horizontal cluster is recorded, if no other undetermined-shape or horizontal cluster may be

detected on the way from the originating cluster to the major sweep-block.

The result of this procedure is a list of segments (ranges of rows). Each segment is associated with a cluster and corresponds to a side of the final major sweep-block of the cluster. All segments associated with undetermined-shape clusters on both sides of a major sweep block are now examined. If one of these segments has a significant overlap with any segment associated with a cluster on the other side of the major sweep-block, then it is named horizontal; otherwise it is named vertical.

Some skeleton patterns may have to be built and attached to the skeleton part derived from the major sweep-block for the remaining clusters. Building of the skeleton line for individual clusters starts with clusters near to the major sweep-block and advances towards the exterior. For all clusters, if an additional line is to be built, all vertical sweep-blocks in the cluster will be made to point to that line. Losing the trace of the skeleton by discarding a cluster (a vertical one, for example) does not present a problem as reference may be made to the major sweep-block's skeleton parts.

If the cluster is horizontal then a basic horizontal line is built based on the participating sweep-blocks' central points. If the cluster injects another cluster then the injected cluster is checked whether it has been associated with a horizontal line. If this is the case the two lines are joined, otherwise the original line is extended to join the skeleton of the major sweep-block.

If the cluster is vertical and adjacent to a major sweep-block it is discarded, otherwise the cluster it injects is examined. That cluster is discarded if it has no line associated with it, otherwise all the current cluster's sweep-blocks are made to point to that line. No new line is built.

The deleted vertical clusters are now bound to the lines associated with the clusters they injected, if any such lines exist. This will not require the building of new lines.

After all the information supplied by the injecting blocks has been exploited, the virtual adjacencies belonging to horizontal adjacency paths are examined. These must be associated with horizontal lines pointing to each major sweep-block involved in the virtual adjacency. They are processed as each major sweep-block is examined and are treated as horizontal clusters. The line originating from a virtual adjacency has the ajdacency's central point as an endpoint and is attached to other skeleton lines by the rules governing horizontal clusters' lines. Selection of this endpoint ensures that the other major sweep-block involved in the virtual adjacency provides a line to the same point thus preserving connectedness.

A skeleton pattern containing some unwanted protrusions may be computed (*Figure 16*), since horizontal lines joined to the basic skeleton may retain some small line segments. Clusters generating horizontal lines will have only one sweep-block (*SB*) nearest to the major sweep-block. How this sweep-block overlaps the major sweep-block is examined. In general, some range of rows at the top and bottom of the major sweep-block will not be overlapped by

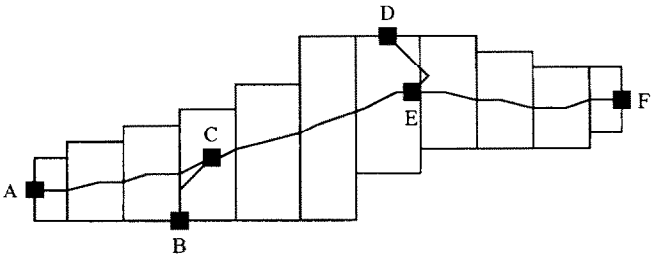


Figure 16 Segments [*B*, *C*] and [*D*, *E*] should be deleted for a realistic skeleton

sweep-block *SB*. If the shape corresponding to the top range is rather horizontal the protrusion that emerges is eliminated. The bottom range is treated similarly. Horizontal lines generated by virtual adjacencies belonging to horizontal adjacency paths are treated in the same way.

Line shortenings are not performed at the time they are established because more skeleton building steps follow. Lines that could have been eliminated are recorded: once the whole skeleton is built they are examined to determine whether they should still be deleted. Elimination of a protrusion may consist of shortening or deleting lines. Lines correpsonding to vertical adjacency paths may not be skipped, only their endpoint is deletable. Lines due to the basic skeleton may also be shortened in this way but they may also be deleted entirely. Line-specific processing takes place both in the line-building process and the graph process.

Building skeleton parts for horizontal sweep-blocks

A horizontal sweep-block may be associated with one or more blocks. By examining the shape suggested by those blocks within the range of columns of the horizontal sweep-block, a basic skeleton is obtained for the horizontal sweep-block. This is, of course, a horizontal line which may be shortened from either end. Shortening will occur only when the horizontal sweep-block has sweep-blocks adjacent to it in the shortening direction. The original horizontal line is then shortened to provide space for lines built to reflect such adjacencies (*Figure 17*).

The line is then registered as a valid line, associated with the sweep-block and lines for the horizontal sweep-block's adjacencies are established. The processing of adjacencies to the left side of the sweep-block shall now be discussed; the rules governing processing of the right-side adjacencies are similar.

Each of the horizontal sweep-block's adjacent

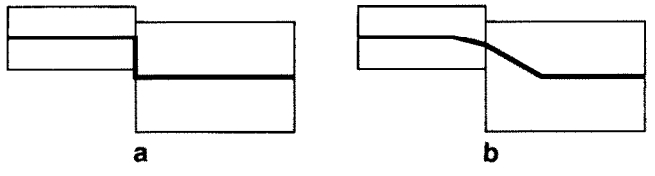


Figure 17 (a) If the original lines are kept unchanged we might end up with a sharp-shaped skeleton; (b) extra space guarantees a more realistic result

sweep-blocks is examined in turn (let *SB* be the examined adjacent sweep-block).

If *SB* is of horizontal type then an additional line is drawn from the left endpoint of the original line to the central point of the actual adjacency of the two sweep-blocks. Thus, connectedness is guaranteed when sweep-block *SB* is processed (since *SB* will contribute a line from its skeleton's endpoint to this central point).

If, however, *SB* is of vertical type, two alternatives are presented depending on how *SB* was treated in the pevious skeleton-building phase. If it was associated with a line then the two lines are joined through the central point of the actual adjacency between the two sweep-blocks. If it was not associated with a line then it may either be a major vertical sweep-block or a vertical sweep-block belonging to a cluster (note that it cannot be a common feeding sweep-block). In either case the horizontal line will have to pass through the central point of the actual adjacency and be attached to one of the major sweep-block's skeleton parts (if sweep-block *SB* is not a major sweep-block its injection to the left may be inspected and the corresponding major sweep-block traced). Extra care is required when the horizontal sweep-block's line is attached to a vertical line because this may require shortening or even elimination of some lines of the corresponding major sweep block's skeleton.

Building skeleton parts for non-vertical blocks

The next step is to process all the non-vertical blocks which have not been subjected to the sweeping process. Each of these blocks will be associated with a new horizontal line which will follow the outline of the whole block. This line is not connected to any existing lines of the skeleton. However, the current block may be included in pairs which record the blocks with sufficiently small overlap to allow them to skip the sweeping process. If this is the case the line associated with the block may need shortening.

The special pairs of blocks are now examined. Separate lines must be created to reflect the connections between them. Three possible combinations must be considered:

1. Neither block has participated in the sweeping process. In this case the central point of the blocks' adjacency is located and joined to the lines associated with the blocks.
2. One block has participated in the sweeping process: the central point of the blocks' adjacency is extended to join the line associated with the non swept block. The block that has been subjected to sweeping is associated with some sweep-bocks which may overlap the column of the adjacency's central point. In this case, there exists a point on a sweep-block's associated line that will be nominated as the block's connecting point.
3. Both blocks have participated in the sweeping process. In this case the previous procedure is applied to both of them.

If one or both of the blocks has been associated with sweep-blocks the lines that have been marked for shortening or removal may have to be reconsidered. This will be discussed with respect to the case of protrusions on the left side of an upper block of a special pair. The discussion is obviously applicable to all other cases.

Once the connection between a special pair of blocks is made, two specific columns can be recorded. The first is called the connect column and is the column of the main connecting point. The second is the join column. Joining the two blocks has been achieved using the lines associated with one of each block's associated sweep-blocks, the point on the line where the joint occurs defines the join column (there is a common connect column but a separate join column for the blocks).

The sweep-blocks associated with the block having their leftmost columns to the left of the connect column are determined. If any of them is horizontal and its rightmost column is not to the right of the connect column, the search is abandoned because a definitely horizontal region has been found. A problem occurs when there is only one such sweep-block and it is of horizontal type. If this sweep-block has more than one adjacent sweep-block to its left, it is left unchanged due to the connectedness criterion. If it does not have any adjacent blocks and non horizontal regions can be detected to the left of the connect column, the associated line is shortened.

If there exists only one adjacent sweep-block and it is horizontal, any line changes may be ignored. However, if it is vertical all connections to all parts of its skeleton must be examined to ensure that if the sweep-block is removed there will be no connectedness-related problems. If all these checks are successful both the vertical and the horizontal sweep-blocks are marked as being deletable. The case where a horizontal sweep-block is the last in the list of sweep-blocks leading to the connect column is treated similarly and may result in a similar pair of sweep-blocks marked for later inspection.

Finally, if there are no horizontal sweep-blocks in the list, the horizontal clusters are examined for the presence of vertical sweep-blocks. If there is only one horizontal cluster in the region and it contains a vertical sweep-block, the cluster's line may require shortening. This is checked.

The marking scheme employed will now be discussed in more detail. During the process of building lines for the major sweep-blocks, major sweep-blocks and their two upper or lower skeleton parts had been marked. In the present process, lists of two sweep-blocks have been marked, larger lists will be marked later. The list building scheme informs us where a line change is about to occur (on the left or the right side of the list) and provides the capability to navigate through the skeleton. Given a horizontal sweep-block its leftmost and rightmost (top and bottom) line connections can be recorded, so presented with a list of sweep-blocks their associated lines and the lines built for connections between sweep-blocks may be traced. This capability is very important because access to all configurations

using a single representation is provided when the whole skeleton is examined for possible changes. It follows that switching from the sweep-blocks' representation to the lines' reprsentation and translating the adjacencies detected between sweep-blocks into connections between their associated lines is a simple task.

Building skeleton parts for vertical blocks

The basic skeleton of a vertical block is easily built and reflects the block's outline. It may be shortened from the top or bottom to provide extra space for connections between the block and adjacent blocks. For each of the adjacent blocks some type of connecting line is built. If the adjacent block is vertical, the blocks will be joined through their corresponding endpoints. If it is a block that has not been subjected to sweeping the joining point will be selected from the points of the non-vertical block's associated line (sometimes the joining point will have to be created and inserted between two existing line points). In this case care must be taken since the non-vertical block's associated line may have to be shortened in future. If the adjacent block has been subjected to sweeping sequences of already built lines may have to be destroyed because the impact of the vertical block on the final skeleton could not be anticipated. Spurious protrusions in the skeleton must be detected, these may consist of several horizontal lines. Their detection and removal is achieved in the same manner as in the special pairs: a list of sweep-blocks whose associated lines need further consideration is built. Building of the list is terminated when the join column is met or when further building would destroy the connectedness of the skeleton. If the list building generates only one horizontal sweep-block the list is ignored and the sweep-block's associated line is shortened (*Figure 18* illustrates this point).

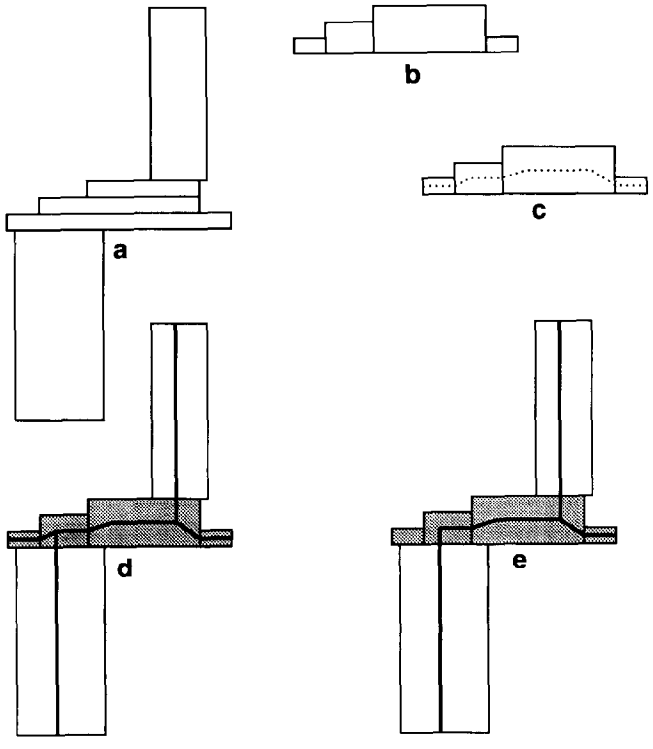


Figure 18 In (d) and (e) we are mixing the blocks' and sweep-blocks' representations to give a better insight in how the skeleton is built. (a) Blocks' configuration; (b) sweep-blocks' configuration; (c) skeleton before the vertical blocks have been processed; (d) skeleton after vertical blocks have been processed without protrusion checking; (e) same as before but with complete checking

BUILDING THE PLANAR GRAPH

The final stage of the skeletonization process consists of examining lines which should be removed and building the planar graph which will represent the image.

Checking protrusions is a procedure with two steps:

1. Check the protrusions in a major sweep-block that have been marked as having deletable lines.
2. Check the protrusions in the areas where a list of marked sweep-blocks was built.

Assume that the upper skeleton parts of a major sweep-block are exmained first. There exist corresponding vertical lines $L_1$and L_2 such that L_1 has been derived from the basic skeleton and L_2 is a vertical adjacency path's line (L_1 will lie above L_2). Such a structure would have been processed by marking line L_1 as invalid and by marking the topmost point of line L_2 as deletable. However, suppose that there exists a vertical block connected with L_1. If this is the case then the proposed deletion, if performed, may break the connectedness criterion.

To overcome this difficulty the uppermost and lowermost lines connected to any of a major sweep-block's skeleton parts were recorded during the skeleton building stages. The topmost respective joining point is examined, if it lies below the topmost point of L_2 then the deletion is effected and since no other line is connected to L_1 connectedness is preserved. If the joining point lies above the topmost point of L_2 only the topmost point of L_1 as marked as deletable.

A similar technique is employed to process a list of sweep-blocks. Throughout the previous stages the leftmost and rightmost points of a sweep-block that have been selected as joining points (for connection lines with vertical blocks and special pairs) have been recorded. As the list is traversed lines are deleted provided they are not critical to connectedness and are not associated with sweep-blocks that have been connected with skeleton lines of later stages. (Suppose that when processing vertical block V_1 a list was built in which sweep-block SB was included. If, when vertical block V_2 was later processed it was connected with the line associated with sweep-block SB, that line must not be deleted even though its respective sweep-block was included in the list.)

The final graph building process is straightforward. The endpoints of every valid line are examined. If an endpoint is found to be deletable and non-preservable the line is tracked until a preservable point is found. The shortened line is then transformed into a series of nodes (points) and edges (reflecting successive line points). The nodes represent branching points with

reference to changes of direction in the line segments that make up the skeleton.

EXPERIMENTAL RESULTS

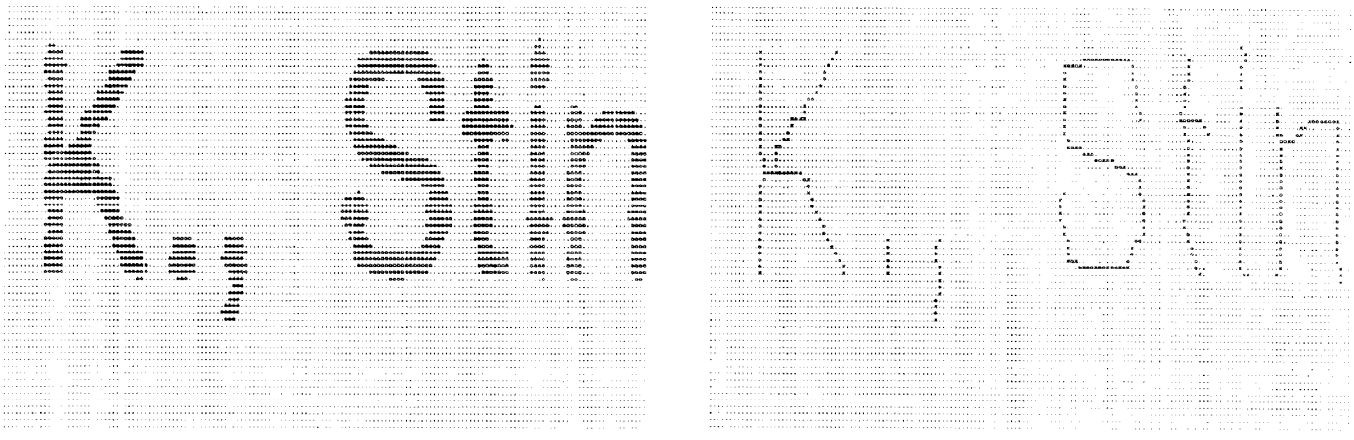
The algorithm described was implemented in C using Apollo DN4000 workstations running Unix. A version of the classical thinning algorithm[14] has also been implemented to serve as basis of comparison (its ease of implementation should make it a universal thinning comparison benchmark, as far as speed in concerned).

The proposed algorithm was subject to an extensive optimization to eliminate function calls. However, no optimization was carried out as far as structure or programming tricks were concerned. The classical algorithm hardly needs any hand-crafted optimization at all. Both algorithm implementations were compiled using the available optimization options of the compiler.

The test images shown in *Figures 19–22* were used to validate the algorithm. Each input sample was 64×160

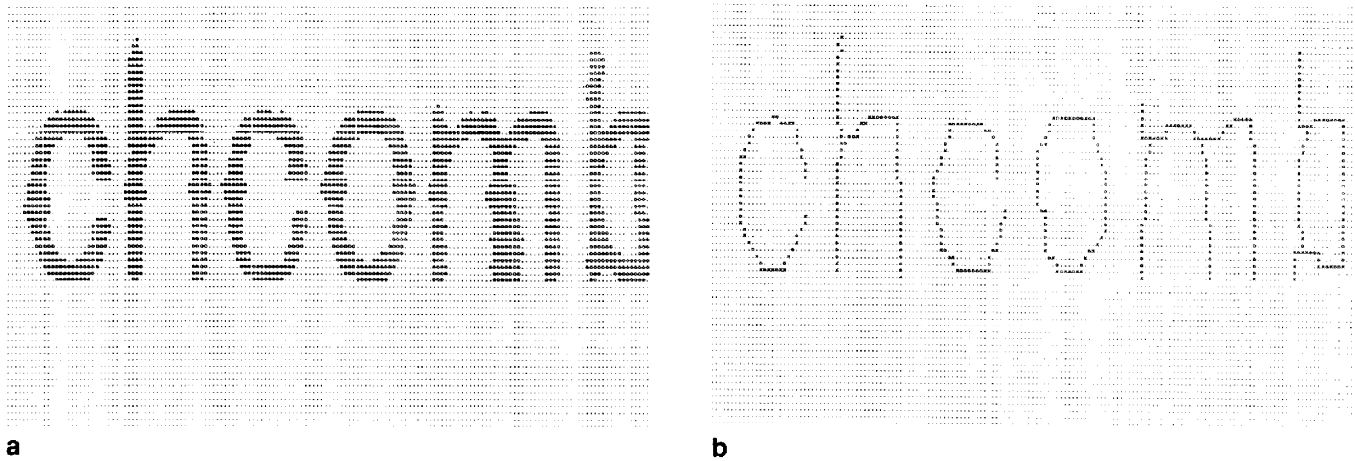
a b

Figure 19 (a) input image; (b) thinned image (**x** denote graph nodes)



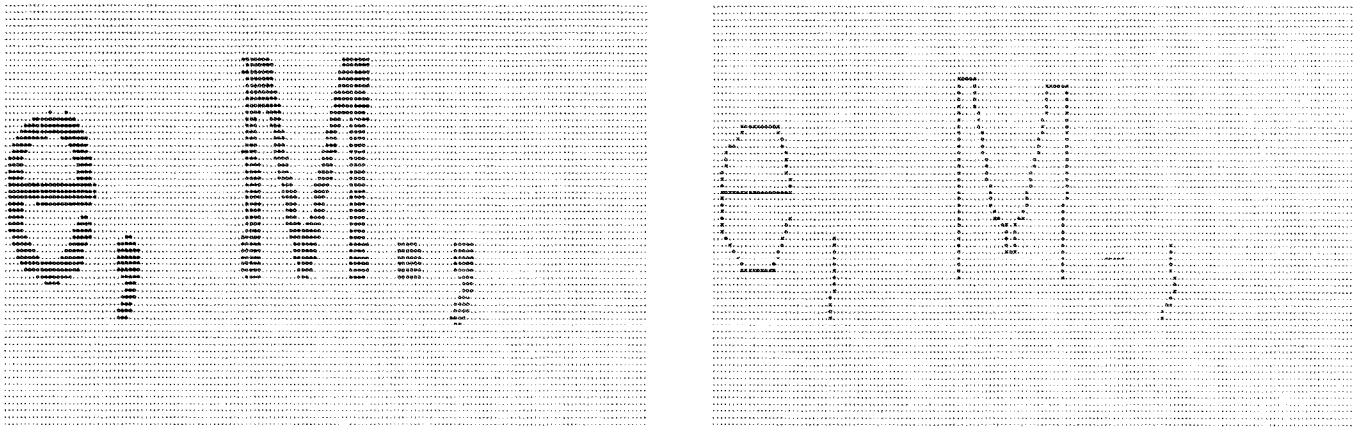
a b

Figure 20 Input image; (b) thinned image (**x** denote graph nodes)



a b

Figure 21 (a) Input image; (b) thinned image (**x** denote graph nodes)



a b

Figure 22 (a) Input image; (b) thinned image (x denote graph nodes)

Table 1 Comparison of thinning algorithms

Algorithms	*Figure 19*	*Figure 20*	*Figure 21*	*Figure 22*
Proposed	0.2	0.2	0.25	0.15
Classical	0.5	0.5	0.6	0.5

pixels. The algorithms were timed using the *getrusage* system function which measures only CPU time dedicated to a process. Because the algorithms produced timing results with quite some variation, each one was executed 20 times on each image to estimate its speed. *Table 1* summarizes the results. The values actually obtained were distributed around the average reported values in a normal manner (time is measured in seconds).

It should be noted that the proposed algorithm requires significant data structures to support the various computational steps and it also requires significant programming effort. In these terms it is outperformed by the classical algorithm which is straightforward in implementation and hardly demanding in terms of space resources.

As far as the quality of the results is concerned it should be noted that the proposed algorithm does not produce smooth skeletons in general. However, it should also be pointed out that a curve is very seldom processed in a straightforward manner in any application, but it is rather subjected to a procedure to detect its dominant points. This procedure is quite time-consuming, and the proposed algorithm performs this task to a considerable extent. Note that the reported results concern the graph-building process of the proposed algorithm as well, whereas for the classical algorithm this step has not been included.

As far as speed is concerned, it is obvious that the proposed algorithm consistently outperforms the classical one, even with the graph-building step included. It remains to be seen if the current speed improvement ratio of about 1 to 3 can be further improved by extensive hand-crafted optimization.

Table 2 suggests that the algorithm's performance is affected by the complexity of the image with respect to the counts of objects detected and processed.

Table 2 Comparison of configurations for proposed algorithm

Parameters	*Figure 19*	*Figure 20*	*Figure 21*	*Figure 22*
Blocks				
Originally built	76	70	93	47
After merging	55	52	66	26
Selected for sweep	23	32	40	12
Sweep blocks				
Total	30	52	63	16
Minor vertical	14	32	32	6
Major vertical	10	14	18	6
Graph nodes	137	130	177	72

TOPOLOGICAL AND GEOMETRICAL PROPERTIES

The algorithm, as all non-pixel-based algorithms[8] does not deliver an image that possesses any information about pattern width. This limits its ability to represent images of varying thickness (in terms of reconstruction abilities) although it can be applied to them successfully. However, there exist problem domains (feature extraction in OCR is a major example) where this ability is of no importance. Nevertheless, it is reasonable to anticipate that an improved version would be able to accommodate such information into the description of the graph's nodes with low overhead.

As far as noise elimination is concerned, we feel that there are two ways of tackling the problem: the first way (clearly identified in the reviewed iterative algorithms) is to employ some conservatism in the process and expect that noise pixels will be treated one by one. In this approach, a lot of individual pixel tests are involved which are time-consuming. However, in the proposed approach we try to obtain more global information regarding the image (that is the goal of the segmentations) and then apply it to the various regions with some local adaptation (the goal of the individual skeleton parts). The algorithm's adaptive nature is best shown by the way that lines are shortened when there is evidence of spurious protrusions. Finally, we feel that the ability to process relatively noise-free images fast

would be an inherent advantage of any thinning approach.

The issue of connectedness will be discussed for each algorithm stage separately. The objective is to demonstrate that each phase delivers a sound configuration (as far as conectedness is concerned) to the subsequent phases.

For the primary segmentation the objective is to be able to navigate through the resulting blocks' configuration in a way representing the original topology. However, with respect to connectedness, we need only examine the start and stop segments of any block.

Let us consider a block's start segment. If there exist one or more segments below it, then each of these segments will belong to a separate (already formed) block. Thus, the pair of touching segments establishes connections between blocks. Stop segments are handled similarly. Moreover, the merging and elimination processes do not affect the connectedness properties of the representation.

In the secondary segmentation, only non-vertical blocks are allowed to participate, provided they can demonstrate a substantial (at least two pixels wide) overlap with other non-vertical blocks. If two non-vertical blocks share such an overlap then it is guaranteed that there will be at least one sweep-block which will accommodate them both. This is because there exists a vertical strip at least two pixels wide which contains successive parts of these blocks. By extending the notion of start and stop segments to start and stop block-lists of successive strips and applying the connectedness discussion of the primary segmentation it can be guaranteed that adjacency between sweep-blocks conforms to the connectedness property of the primary segmentation.

By building a basic backbone for every major sweep-block a continuous line is established to reflect the fact that successive adjacent blocks (connected) contribute to the skeleton. On the occasion that virtual adjacency paths are processed, some parts of the backbone may be deleted, but this can only happen after new lines are introduced to safeguard connectedness.

Horizontal sweep-blocks are processed only after the primary skeleton configuration has been delivered (all major sweep-blocks have had their local skeleton patterns computed), so that they can establish to which already built skeleton parts they have to be connected. Actually, whenever an adjacency with a vertical sweep-block is processed there exists a pointer to the local skeleton part and connectedness is preserved. Adjacencies between horizontal sweep-blocks are trivial as all that is needed is the 'negotiation' of a common connecting point. Similar procedures apply to blocks (either vertical or non-vertical) which have not participated in the decision process.

Summarizing, the key concept is that a series of configurations is built which reflect increasing detail about the adjacency and the interaction of the image's regions. At the last stage, all the available detail is used to process the 'difficult' regions and provide a draft of the local skeleton parts. As we backtrack to the more abstract configurations, less detail is required to build skeleton parts but all the underlying (already built) skeleton parts are there to be connected to and thus preserve connectedness.

CONCLUSIONS

A novel approach to thinning has been presented. The proposed algorithm (though complex) skeletonizes digital binary images and delivers the input's representation in graph form to be used for post-processing applications. Speed of execution issues have also been addressed, and have suggested that the algorithm can provide considerable time savings. The quality of the results obtained is suitable for pattern recognition purposes.

REFERENCES

1 Levine, M. D *Vision in Man and Machine*. McGraw-Hill. New York (1985)

2 Kwok, P C K 'Thinning algorithm by contour generation', *Commun. ACM*, Vol 31 No 11 (November 1988) pp 1314–1324

3 Xia, Y 'Skeletonization via the realization of the fire front's propagation and extinction in digital binary shapes', *IEEE Trans. PAMI*, Vol 11 No 10 (October 1989) pp 1076–1086

4 Jang, B K and Chin, R T 'Analysis of thinning algorithms using mathematical morphology', *IEEE Trans. PAMI*, Vol 12 No 6 (June 1990) pp 541–551

5 Arcelli, C and Sanniti di Baja, G 'A Width-independent fast thinning algorithm', *IEEE Trans. PAMI*, Vol 7 No 4 (July 1985) pp 463–474

6 Arcelli, C and Sanniti di Baja, G 'A one-pass two-operation process to detect the skeletal pixels on the 4-distance transform', *IEEE Trans. PAMI*, Vol 11 No 4 (April 1989) pp 411–414

7 Wang, P S and Zhang, Y Y 'Fast and flexible thinning algorithm', *IEEE Trans. Comput.*, Vol 38 No 5 (May 1989) pp 741–745

8 Lam, L, Lee, S-W and Suen, C Y 'Thinning methodologies – a comprehensive survey', *IEEE Trans. PAMI*, Vol 14 No 9 (September 1992) pp 869–885

9 Dilalou, A, Kalles, D and Xanos, D *Optical Character Recognition*, Diploma Thesis (in Greek), Department of Computer Engineering and Informatics, University of Patras, Greece (1991)

10 Pavlidis, T 'A vectorizer and feature extractor for document recognition', *Comput. Vision, Graph. & Image Process.*, Vol 35 (1986) pp 111–127

11 Martinez-Perez, M P, Jimenez, J and Navalon, J L 'A thinning algorithm based on contours', *Comput. Vision, Graph. & Image Process.*, Vol 39 (1987) pp 186–201

12 Samet, H *The Design and Analysis of Spatial Data Structures*, Addison-Wesley, Reading, MA (1990)

13 Knuth, D *The Art of Computer Programming: Vol 3 – Sorting and Searching*, Addison-Wesley, Reading, MA (1973)

14 Pavlidis, T *Algorithms for Graphics and Image Processing*, Springer-Verlag, Berlin (1982)