EFFICIENT MORPHOLOGICAL SET TRANSFORMATIONS ON LINE DRAWINGS

RAFAEL CARDONER* and FEDERICO THOMAS†

*Institut de Cibernètica (UPC), Diagonal 647, 08028 Barcelona, Spain
E-mail: rcardoner@ic.upc.es

†Institut de Robòtica i Informàtica Industrial (CSIC–UPC)
Gran Capità 2-4, 2 planta, 08034 Barcelona, Spain
E-mail: fthomas@iri.upc.es

Image compression techniques have been recently used not only for reducing storage requirements, but also computational costs when processing images on low cost computers. This approach might be also of interest for processing large engineering drawings, where feature extraction techniques must be intensively applied for their segmentation into regions of interest for subsequent analysis. This paper explores this alternative using a simple run-length compression, leading to excellent results.

Although this approach is not new and can be classified within the decomposition paradigm used since the early stages of line drawing image processing, the developed formalism allows directional morphological set transformations to be performed, on a low cost personal computer, faster than on costly parallel computers for the same, but uncompressed, images. This good performance is proved in two different applications: the generation of homotopic skeletons through thinning processes, and the extraction of linear features through serializing multiangle parallelism operations.

Keywords: Binary morphology, run-length encoding, technical document analysis, feature extraction, homotopy.

1. INTRODUCTION

A variety of approaches has been proposed to solve the problems of interpreting line drawings automatically. Most of them start by identifying linear or curved strokes from the binary data. To achieve this end, some authors require a pixel-based thinning algorithm. Others extract the information by using decomposition techniques. Combinations of the two approaches as well as alternative techniques have also been investigated. See Ref. 3 for recent developments on this area.

Decomposition techniques are usually faster mainly because they form large groups of pixels and then operate on them rather than on individual pixels. Since the early efforts on computer processing of line drawings, grouping techniques based on run-length-encoded representations were used to achieve this end.[6] Nevertheless, as it was already pointed out in Ref. 13, the price to pay for such performance are increased logical complexity, and the length of the program.

It has been generally accepted that the main disadvantage of run-length codes is that they do not give the ordered region boundary points, as in chain coding, and this makes difficult to segment different regions and to extract shape features.

*This work has been partially supported by ENHER (*Empresa Nacional Hidroeléctrica del Ribagorzana*) under several contracts within the frame of the *ALEPH-CORAL project*.

Nevertheless, directional morphological set transformations can be easily computed on this representation relying only on just two simple operations: shifting and intersecting and/or unioning lists of intervals. As a consequence, resulting programs are neither complex nor lengthy.

This article is structured as follows. In Sec. 2, the well-known run-length compression model and its ability to reduce the volume of data in line-like images is revisited. Then, it is shown how basic morphological set transformations can be implemented on run-length-encoded images by simply shifting and intersecting and/or unioning lists of intervals. In Sec. 3, we use the previously described operations to implement two particular applications. Both have been fully implemented and their performance has been compared with that recently reported elsewhere. We conclude in Sec. 4.

2. DIRECTIONAL MORPHOLOGICAL OPERATIONS ON RUN-LENGTH-ENCODED IMAGES

2.1. Run-Length Encoding

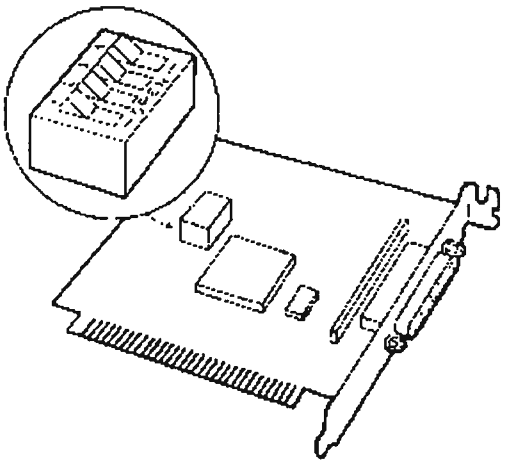
After scanning an image, a thresholding operation is used to obtain a bilevel, or binary, image. Each pixel is classified as belonging to the foreground or the background according to whether its gray level is greater or lower than a suitable threshold. It is crucial for the threshold image to reproduce the original as accurately as possible.

Once the image has been binarized, any region in it can be viewed as a sequence of alternating intervals of 0s and 1s. Run-length codes represent these intervals, or runs.[7] A simple run-length code consists of the start address of each interval of 1s, followed by the corresponding end address. Thus, this compression is designed to take advantage of the fact that a pixel value will often repeat, which is particularly true when working with line drawings where compression factors of 20 are usually accomplished when a byte per pixel representation is used.

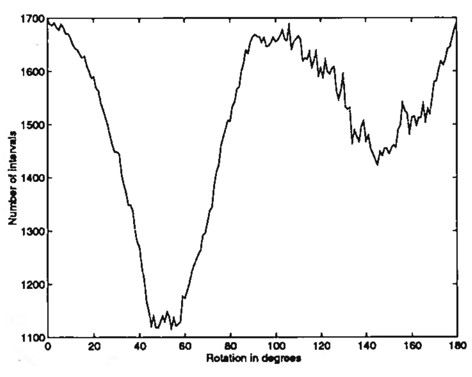
Then, an image can be seen as a list of lines, each line being a possibly empty list of non-overlapping sorted intervals. Run-length encoding is particularly useful because of its simplicity and generality. Actually, interval coding has provided a particularly simple and effective way of speeding a variety of binary image operations. It has been used to speed up operations such as region labeling,[16] boundary extraction,[12] moment invariants calculation,[21] eigenvalue decompositions,[15] some morphologic computations,[9] and skeletons.[14]

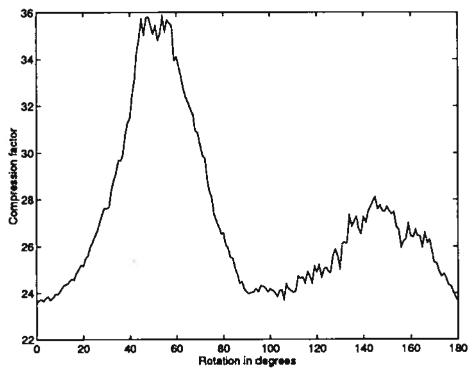
Run-length codes have the advantage that, regardless of the size of an image, memory requirements and processing times depend directly on the complexity of the image. This is of particular interest when processing large engineering drawings.

A run-length code can be obtained in a single raster scan. This fact introduces a privileged direction: the one in which intervals are obtained. This might lead to different computational costs in subsequent operations depending on the orientation of the image (see Fig. 1). To avoid this to some extent, simultaneous vertical and horizontal encoding has been used.[1]



(a)





(b) (c)

Fig. 1. (a) An engineering line drawing; (b) the number of intervals obtained in its run-length representation vs. its orientation in degrees; and (c) the corresponding compression factor.

Other compression techniques for speeding up binary morphological operations can be found in the literature. For example, in Ref. 2 a bitmap codification is proposed. Nevertheless, processing times and memory requirements using this codification still depend heavily on image sizes. This is clearly an important drawback when processing very large drawings. For example, while a 10512×5256 binary image, containing 44831 run-length encoded intervals, requires 270 Kbytes of memory, the same bitmapped image needs 6.9 Mbytes. Moreover, some fundamental operations, such as region labeling and arithmetic computations, cannot be implemented on bitmapped images. In practice, this means that they have to be encoded and decoded depending on the operation to be carried out. As a consequence, medial complexity large images are processed in lower times using run-length encoding. Thus, this compression remains as an important alternative that allows the processing of large drawings.

2.2. Translating Morphological Transformations into Shiftings and Logical Operations

The basis and theory of Mathematical Morphology was introduced as a body of knowledge by Matheron[11] and Serra.[17] It deals with *sets* representing shapes in binary or gray level images. For some purposes, such as object identification, morphological operations are more useful than their common convolution-based counterparts because they process images focusing on shape. Since most objects' features necessary for their identification are directly related with shape, it becomes apparent that these techniques provide a natural approach to deal with the problem of machine vision recognition.

Morphological operations are applied locally using a basic geometrical structure called *structuring element*. The structuring element is defined as a pattern which determines how the operation is performed. The result of applying an operation to a pixel of an image is obtained by positioning the reference of the structuring element on that pixel, performing a sequence of logical operations to the neighbors included in the window, and accumulating results on that pixel.

An important consequence is that an operation with a structuring element composed by a single pixel next to its reference can be carried out shifting the image one pixel in the opposite direction and applying the corresponding logical operation between the original image and the shifted one.

Since structuring elements can be decomposed in our case into minimal templates including the reference and just one more pixel, it is possible to perform morphological operations by applying a set of ordered shiftings and logical operations. While shiftings depend on the shape of the structuring element, the nature of the logical operations involved is given by the desired morphological transformation. Actually, erosion is associated with the AND logical operation while dilation comes together with the OR operation.

As an example, if the 3×3 structuring element appearing in Fig. 2 is used for eroding an image, say Img, the result, say Erod, will be the same as applying the

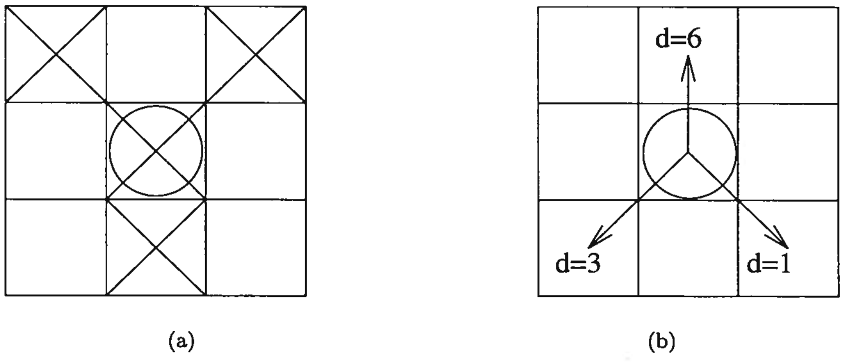


Fig. 2. (a) Structuring element used to define a certain morphological transformation, and (b) how it can be translated into a set of shiftings and logical operations.

following operations on Img

$$\text{Erod} = \text{Img} \cap \text{Img}_6 \cap \text{Img}_1 \cap \text{Img}_3\,, \tag{1}$$

where Img_d stands for the image Img shifted in the d direction. This direction is defined by 0.7 from the right direction turning clockwise.

If the same structuring element would be used for dilating Img, the result, say Dil, could be simply obtained as

$$\text{Dil} = \text{Img} \cup \text{Img}_6 \cup \text{Img}_1 \cup \text{Img}_3\,. \tag{2}$$

In Mathematical Morphology, operations usually appear in dual pairs. Duality allows to perform an operation by applying its dual to the complement of the original image. In this sense, erosion and dilation can be linked by duality as follows

$$\overline{\mathbf{Erode}(Img)} = \mathbf{Dilate}(\overline{Img})\,, \tag{3}$$

where $\overline{Img}$ stands for the complement of the image Img, that is, its background.

Thus, dilation and erosion transformations are related in that what one does to the image foreground the other does to the image background. This property allows us to perform the erosion in Eq. (1) by the following dual dilation that uses only unions instead of intersections

$$Erod = \overline{\mathbf{Dilate}(\overline{Img})} = \overline{\overline{Img} \cup \overline{Img_6} \cup \overline{Img_1} \cup \overline{Img_3}}\,. \tag{4}$$

If duality is applied to (2), we get

$$Dil = \overline{\mathbf{Erode}(\overline{Img})} = \overline{\overline{Img} \cap \overline{Img_6} \cap \overline{Img_1} \cap \overline{Img_3}}\,. \tag{5}$$

These simple considerations on duality are proved of great relevance when implementing efficient computer programs. Next, we will concentrate ourselves in the efficient implementation of shiftings and logical operations on run-length-encoded images.

2.3. Intersecting and Unioning Run-Length-Encoded Images

The brute-force algorithm to process a compressed image would consist of decompressing the image, modifying each pixel value, and compressing the resulting image. Obviously this should be avoided.

Let us suppose two run-length-encoded images to be intersected. l_1 and l_2 represent the lists of intervals associated with the same line in the first and second images respectively. Then, the list representing the intersection of both lists of intervals can be computed as follows:

```
**algorithm intersect;**
**input** $l_1 =< [a_0, b_0], \ldots, [a_n, b_n] >$ **and** $l_2 =< [c_0, d_0], \ldots, [c_m, d_m] >$;
$i \leftarrow 0; j \leftarrow 0;$
**repeat**
```

```
    if $b_i < c_j$ then $i \leftarrow i+1$;     /* no overlap, skip an interval in $l_1$ */
      else if $d_j < a_i$ then $j \leftarrow j+1$;     /* no overlap, skip an interval in $l_2$ */
             else    /* overlap, intersect current intervals */
               $l_3 \leftarrow l_3 \circ [\max(a_i, c_j), \min(b_i, d_j)]$;
               if $b_i < d_j$ then $i \leftarrow i+1$;
               else $j \leftarrow j+1$;
             endelse;
    endif;
until $i > n$ or $j > m$;
output $l_3$;
```

where $\circ$ denotes concatenation. The generated list, l_3, can be computed in linear time in the total input size $| l_1 | + | l_2 |$.

The algorithm for the union of two lines is similar, and also runs in linear time. Note that both intersection and union operations produce a list l_3 with at most $| l_1 | + | l_2 |$ intervals.

The evaluation of shifted images is avoided so that displacements of an image are taken into account, introducing the proper indices, directly when intersected or unioned with another image. In other words, a directional erosion or dilation takes the same time as an intersection or union operation.

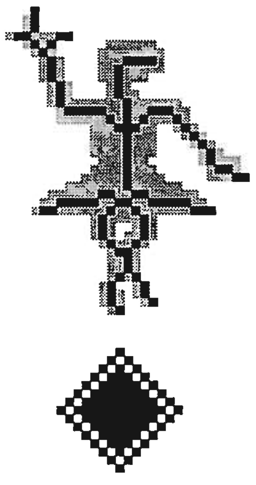


Fig. 3. Skeleton obtained using the proposed algorithm. In this case, it is clear that reconstructability, connectedness, and thinness are mutually incompatible requirements.

3. APPLICATIONS

3.1. Homotopic Skeletons

In most drawing interpretation systems, the raster-to-vector conversion of a line structure is achieved by thinning the image, thus reducing thick lines to linear sequences of pixels, and then finding an approximating polygonal line. Thinning is, in general, a very time consuming process. This is why some algorithms try to generate, directly from the original line-like image, a connectivity graph consisting of nodes and edges that formalize the intuitive notions of *lines* and *crossing points between lines*. This is the approach taken in Refs. 1 or 5 relying at some points on heuristic procedures. In what follows, it is shown how thinning procedures can be easily implemented on run-length-encoded images leading to low computational overhead executions.

Many thinning algorithms, or modifications of existing ones, have been proposed in recent years. More than 300 published papers on this subject reflect this proliferation (see Refs. 10 or 19).

The common required properties of a skeleton are:

- *Reconstructability.* A skeleton must contain sufficient information which can be used to reconstruct the original binary image.
- *Rotation-invariance.* Because of robustness reasons, a skeleton for a given pattern must be independent from its orientation.
- *Connectedness.* A skeleton must be homotopic to the region it corresponds to. In other words, it must preserve 8-neighbor connectedness for foreground and 4-neighbor connectedness for background.
- *Thinness.* A skeleton must be single-pixel in width. This is also a common requirement to ease subsequent processes, such as vectorization.

Unfortunately, as recognized in Ref. 17, in the discrete plane these requirements become mutually incompatible, so that practical skeleton methods are invariably a compromise between them. In this sense, thinness may be incompatible with preservation of connectedness, or with reconstructability.

The proposed approaches to obtain the skeleton of a given binary image can be classified either into those that peel off the boundary of an object in a sequence of raster, or parallel operations, or into those based on a previous distance transformation. The following algorithm, which falls fully inside the former approach, satisfies the requirements given above in the following order of priority: connectedness, thinness and reconstructability. Whenever possible, these three requirements are satisfied.

```
algorithm Skeleton_A;
/* A = Accumulator; G= Gap; E= Eroded ; R= Residual; S= Skeleton */
input: X;
output: S;
$S \leftarrow \emptyset$;
$A \leftarrow \emptyset$;
$S \leftarrow X$;
do;
    $X \leftarrow S$;
    for($d \leftarrow 0$; $d < 8$; $d \leftarrow d + 2$)
        $G \leftarrow S \cap \overline{S_d} \cap [(S_{d+1} \cap \overline{S_{d+2}}) \cup (S_{d-1} \cap \overline{S_{d-2}})]$;
        $E \leftarrow S \cap S_d)$;
        $R \leftarrow S \cap \overline{(E \cup E_{d+4})}$;
        $A \leftarrow G \cup A \cup R$;
        $S \leftarrow A \cup E$;
    **endfor**;
**while** $(X \neq S)$;
**end.**
```

This algorithm, introduced in Ref. 4, is perhaps the simplest proposed thinning algorithm that provides a connected single-pixel in width (whenever possible, see Fig. 3), well-centered homotopic skeleton that allows shapes to be nearly reconstructed.

The motivation behind this thinning algorithm is seen as follows. First those pixels whose deletion by a directional erosion might destroy the connectedness are retained and classified as *gaps*, then the region is effectively eroded and the corresponding *residual* computed. Gaps and residuals are retained in the image since it can be shown they are part of the skeleton. This process is repeated until no more progress is made. As a consequence, the same pixels are classified as gaps or residuals many times. In order to take advantage of the run-length codification, when those parts of the image which are classified as gaps or residuals are no longer required by the thinning process, they are removed from the image and accumulated as part of the final skeleton. This strategy leads to the following algorithm:

```
**algorithm Skeleton_B;**
**input:** X;
**output:** S;
$i \leftarrow 0$;
$S \leftarrow \emptyset$;
$L \leftarrow \emptyset$; /* increment of skeleton */
**do**
    $I \leftarrow \emptyset$;
    **for**$(d \leftarrow 0;\ j < 8;\ d \leftarrow d + 2)$
        $G \leftarrow X \cap \overline{X_d} \cap \left[\left(X_{d+1} \cap \overline{X_{d+2}}\right) \cup \left(X_{d-1} \cap \overline{X_{d-2}}\right)\right]$;
        $E \leftarrow X \cap X_d$;
        $R \leftarrow X \cap \overline{(E \cup E_{d+4})}$;
        $I \leftarrow I \cup R \cup G$; /* increment of skeleton */
        $X \leftarrow E \cup I$; /* updated image */
    **endfor**;
    $X \leftarrow X \cap \overline{L}$; /* removal of previous skeleton increment*/
    $S \leftarrow S + (L)^i$; /* updated skeleton */
    $L \leftarrow I \cap \overline{L}$; /* updated last skeleton increment */
    $i \leftarrow i + 1$;
**until** $X == \emptyset$;
**end,**
```

where $(I)^i$ denotes the operation that sets image I to level i.

This algorithm introduces three main advantages over the previous one. First, the iterations continue until an image becomes empty, which is faster that detecting idempotence. Second, the thinning effort is concentrated on the thick regions. Third, a grey-level skeleton is generated, the value of each skeleton pixel being its chess-board distance to the boundary.

The obtained skeletal legs are identical for both algorithms but differences sometimes arise at some skeletal joints. This is a consequence of our deletion-retention strategy: pixels in joints appear as gaps or residuals many times, then they are removed while they have to be retained to get exactly the same results.

Figure 4 shows the result of executing these two algorithms on an image of dimension 1060 $\times$ 1560 (the circle in it has a diameter of 230 pixels). 83 iterations of the outer loop were necessary to reach idempotence. This required 2'01" minutes for the first algorithm and just 22 seconds of CPU time running on a PC Pentium/90Mhz with 8 Mbytes of RAM for the second one. The time required to convert a bitmap into a run-length representation (typically milliseconds) can be neglected in front of these timings.

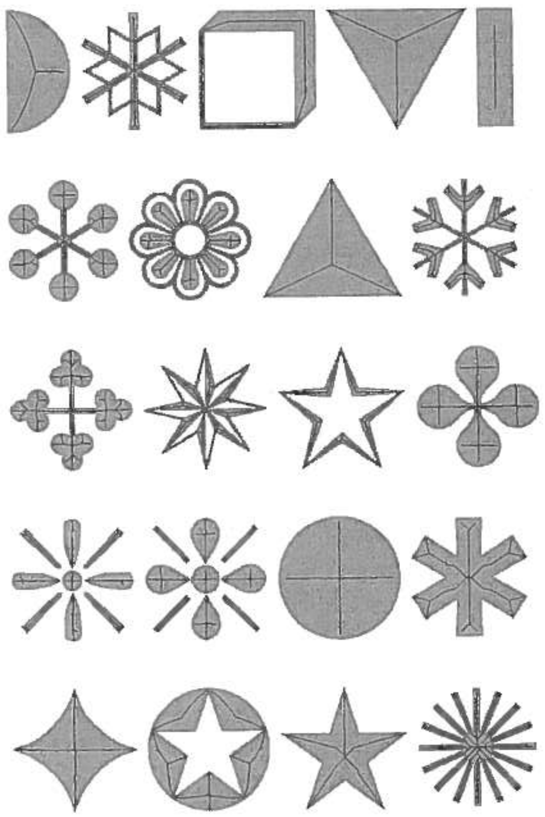


Fig. 4. Scanned image of 1060 $\times$ 1560 pixels used to exemplify the performance of the proposed skeletonization algorithm.

Times are far lower when the second algorithm is applied on line-like images; for example, the skeleton of Fig. 5(a) is obtained in two seconds on a Pentium-90 (Fig. 5(b)).

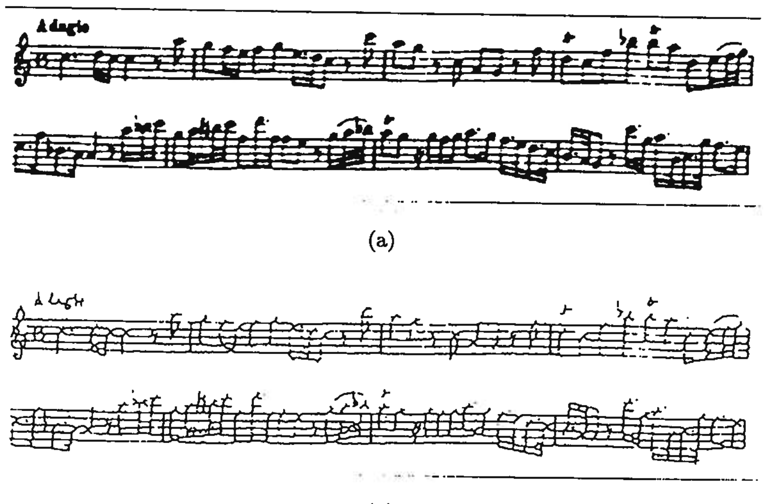


Fig. 5. (a) A 1024 $\times$ 256 image; and (b) its skeleton.

3.2. Implementation of MAP Operations

The MAP (Multi Angled Parallelism) operation, as defined in Ref. 20, is an extension of the conventional mathematical morphology developed for identification and segmentation of different kinds of features in topographic maps.

Basic applications of mathematical morphology usually perform non-directional erosion and dilation operations directly on the image. The MAP Operation Method extends these operations defining transformations based on directional erosion and dilation on a directional representation of the original image. Images are decomposed in 8 directional feature planes containing separate information on each direction and then processed. As a result, separation between point-wise features and linear features is possible.

The concept underlying linear feature extraction in MAP is *extracting the core of linear features and expanding it further.* MAP operation method separately extracts the core of linear features in each direction by performing parallel computations on the eight different feature planes using image processing dedicated hardware. Once the directional linear features are extracted on each plane, results are merged to obtain the linear features as they are in the original image.

Actually, all operations implemented in MAP can be directly translated into shiftings and logical operations. Thus, their implementation on run-length-encoded images is also direct. This has been done in our laboratory on a PC 486/50Mhz with no image processing specific hardware. As expected, feature segmentation results are identical to those obtained using the parallel approach but, in our case, time processing becomes surprisingly lower compared to those obtained at the Electrotechnical Laboratory (ETL) and reported in Ref. 20.

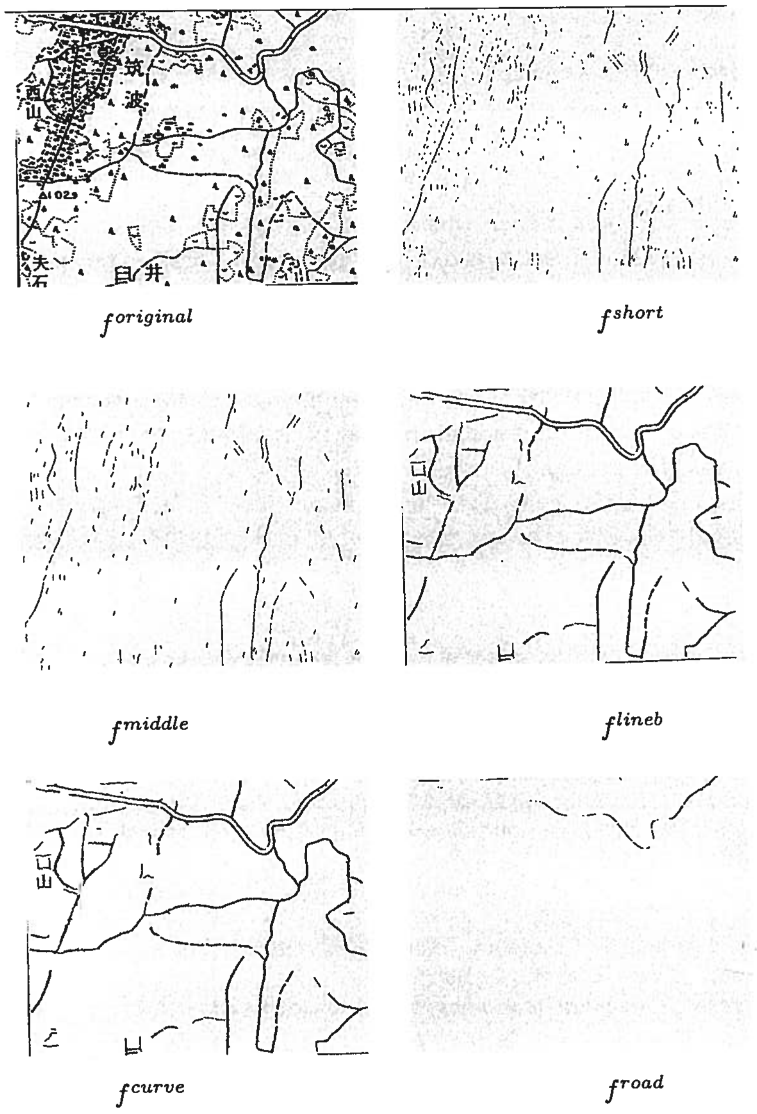


Fig. 6. A 640 × 512 cartographic drawing ($f^{original}$) and five different features extracted from it (f^{short}, f^{middle}, f^{lineb}, f^{curve} and f^{road}).

The following table contains a comparison between times required for the extraction of six different features (see Fig. 6), for the same 640 × 512 image, using our method and their counterparts in the original analysis carried out at the ETL.

Feature (see Fig. 6)	Serial implementation on run-length-encoded images	Parallel implementation as reported in Ref. 20
f^{short}	10"	–
f^{middle}	16"	1'33"
f^{longb}	21"	3'48"
f^{lineb}	22"	–
f^{curve}	44"	7'58"
f^{road}	58"	–

4. CONCLUSIONS

We have described how directional morphological operations can be applied directly on run-length-encoded images. This is of particular interest for processing line drawing images. Since this simple codification reduces the volume of data significantly in these kinds of images, typically by factors of 20, performing the operations on the compressed data leads to important speedups. It has been shown that the extraction of linear features and homotopic skeletons can benefit from this approach. Moreover, it is still possible to parallelize the resulting algorithms, at least using eight processors, one for each directional plane. This is an interesting point for future research.

REFERENCES

1. L. Boatto *et al.*, "An interpretation system for land register maps", *IEEE Comput.*, July 1992, pp. 25–33.
2. R. van der Boomgaard and R. van Balen, "Methods for fast morphological image transforms using bitmapped binary images", *CVGIP: Graphical Models and Image Process.* **54**, 3 (1992) 252–258.
3. H. Bunke, P. S. P. Wang and H. Baird, eds., *Int. J. Pattern Recognition and Artificial Intelligence, Special Issue on Document Image Analysis*, World Scientific, October 1994.
4. R. Cardoner and F. Thomas, "Residuals + directional gaps = skeletons", *Patt. Recogn. Lett.* **18**, 4 (1997) 343–353.
5. A. J. Filipski and R. Flandrena, "Automated conversion of engineering drawings to CAD form", *Proc. IEEE* **80**, 7 (1992).
6. H. Freeman, "Computer processing of line-drawing images", *Comput. Surveys* **6**, 1 (1974) 57–97.
7. A. K. Jain, *Fundamentals of Digital Image Processing*, Prentice-Hall, Englewood Cliffs, New Jersey, 1989, pp. 377–394.
8. B.-K. Jang and R. T. Chin, "Analysis of thinning algorithms using mathematical morphology", *IEEE Trans. Patt. Analy. and Mach. Intell.* **12**, 6 (1990) 541–551.
9. L. Ji, J. Piper and J.-Y. Tang, "Erosion and dilation of binary images by arbitrary structuring elements using interval coding", *Patt. Recogn. Lett.* **9** (1989) 201–209.
10. L. Lau, S.-W. Lee and C. Y. Sueu, "Thinning methodologies. A comprehensive survey", *IEEE Trans. Patt. Analy. and Mach. Intell.* **14**, 9 (1992) 869–885.
11. G. Matheron, *Random Sets and Integral Geometry*, John Wiley, New York, 1975.

12. T. Pavlidis, *Algorithms for Graphics and Image Processing*, Springer-Verlag, Berlin, 1982.
13. T. Pavlidis, "A vectorizer and feature extractor for document recognition", *Comp. Vision, Graphics, and Image Process.* **35** (1986) 111–127.
14. J. Piper, "Interval skeletons", *IEEE Proc. 11th IAPR Int. Conf. on Patt. Recogn.* **III** (1992) 468–471.
15. J. B. Roseborough and H. Murase, "Partial eigenvalue decomposition for large image sets using run-length encoding", *Patt. Recogn.* **28**, 3 (1995) 421–430.
16. A. Rosenfeld and A. C. Kak, *Digital Picture Processing*, Academic Press, New York, 1982.
17. J. Serra, *Image Analysis and Mathematical Morphology*, Academic Press, London, 1982.
18. B. C. Smith and L. A. Rowe, "Algorithms for manipulating compressed images", *IEEE Comput. Graphics and Appl.*, September 1993, pp. 34–42.
19. C. Y. Suen and P. S. P. Wang, eds., *Thinning Methodologies for Pattern Recognition, Special Issue Int. J. Pattern Recognition and Artificial Intelligence* **7**, 5 (October 1993).
20. H. Yamada, K. Yamamoto, T. Saito and K. Hosokawa, "Directional mathematical morphology and reformalized Hough transformation for the analysis of topographic maps", *IEEE Trans. Patt. Analy. and Mach. Intell.* **15**, 4 (1993) 380–387.
21. M. F. Zakaria, L. J. Vroomen, P. J. A. Zsombor-Murray and J. M. H. M. van Kessel, "Fast algorithm for the computation of moment invariants", *Patt. Recogn.* **20** (1987) 631–643.

Rafael Cardoner received the Technical Telecommunications Engineering degree in 1985 from *La Salle Telecommunications School*, Barcelona, Spain. He then joined the *Institutde Cibernètica* where he is the head of Laboratory of Electronics.

Photograph of R. Cardoner is not available.

Federico Thomas received the Telecommunications Engineering degree (1984) and the PhD degree in Computer Science (1988), both from the Polytechnical University of Catalonia, Barcelona, Spain.

From 1985 to 1996, he was a member of the Robotics group of the *Institut de Cibernètica* in Barcelona. In 1996 he joined the *Institut de Robòtica i Informàtica Industrial*, also in Barcelona, where he conducts research on kinematics and certain aspects of image processing.

He holds a position of Research Scholar at the Spanish High Council for Scientific Research (CSIC). He is also an Associate Professor at the Polytechnical University of Catalonia.