Machine Vision and Applications 9: 193–200 (1997)

Machine Vision and Applications

Contour representation of binary images using run-type direction codes

Takafumi Miyatake[1], **Hitoshi Matsushima**[2], **Masakazu Ejiri**[1]

[1] Central Research Laboratory, Hitachi Ltd., Kokubunji, Tokyo 185, Japan
e-mail: miyatake@crl.hitachi.co.jp
[2] Telecommunications Division, Hitachi Ltd., Totsuka,Yokohama 244, Japan

Abstract. This paper proposes a fast and high-definition contour-tracing algorithm for digitized binary images. Only the coordinates of white-to- black and black-to-white transition points on each scan line are used as data for the contour tracing. The resulting description of the contour is the combined set of ten newly developed run-type direction codes. The code is determined automatically by an automaton while the horizontal coordinates of two transition points in the two adjacent scan lines are compared. The processing speed of this algorithm depends on the number of transition points of an image, and, thus, it has the advantage over conventional algorithms, where the speed usually depends on the number of pixels. A high-definition-type contour description is obtained and complete restoration of the original image is possible.

Key words: Contour tracing – Transition points – Run – Automaton – Image analysis

1 Introduction

Contour lines are the basic and intrinsic information needed to represent shapes of figures in such wide application areas as character recognition, map and drawing recognition, and even 3D object recognition for robotic vision. There have been many research papers on contour-tracing algorithms which generate sets of pixel coordinates on the boundaries of figures in raster-scanned digital binary images. These include the algorithms that can restore the original figures precisely from the obtained coordinate sets (Yokoi et al. 1973; Rosenfeld and Kak 1976; Lee et al. 1982; Suzuki and Abe 1983), as well as the algorithms that efficiently trace the contour from the edge positions on each scan line (Agrawala and Kulkarni 1977; Grant and Reid 1981; Carpson 1984). However, there are few algorithms that satisfy both requirements: the precision and efficiency. Conventional algorithms sometimes require a large buffer image memory as a working area which easily reduces the contour tracing speed especially for high- definition coding of large-scale images such as those used in the field of document understanding and drawing recognition (Matsuzawa et al. 1983; Ejiri et al. 1984; Miyatake et al.1985).

We developed a fast and high-definition contour-tracing algorithm that is applicable to large-scale document and drawing analysis (Miyatake et al. 1987). To attempt to achieve high-speed contour tracing, we used black-to-white and white-to-black image transition points along scan lines, instead of tracking each pixel data along the boundaries. This algorithm completes the contour tracing within a single, raster-mode scan of the image by using only small-scale line buffer memories, thus avoiding two-dimensional random access to a large-scale image buffer memory. To achieve high-definition, newly developed chain codes (run-type direction codes) are used to describe local patterns that constitute the boundaries.

In this paper, we first explain briefly conventional contour-tracing algorithms together with their drawbacks. Then, the new algorithm and the results of its application to the standard images designated by CCITT (International Telegraph and Telephone Consultative Committee) are explained. These images, each having 1728×2287 pixels, require an average processing time of a fraction of a second, and the average speed is more than ten times faster than that of the conventional high-definition-type algorithm, thus proving the effectiveness of the proposed algorithm.

2 Conventional contour tracing algorithm

First, we define some of the expressions needed to explain conventional contour-tracing algorithms. Only binary images are dealt with in this paper. The image is divided into grids to form square pixels, each of unit length 1 in both the horizontal and vertical directions. The black pixels on a figure have a logical value 1, in contrast to the logical value 0 for white background pixels. (The black figure pixels and the white background pixels are also expressed in this paper by symbols B and W, respectively.) Transition points are defined as the points where the pixel values change from 1 to 0 or 0 to 1 when the image is scanned in a raster direction (left to right). The points of transition from the background

Correspondence to: Takafumi Miyatake

to the figure (0 to 1) determine a left edge (of the figure) and the points of transition from the figure to the background (1 to 0) determine a right edge (of the figure). These edges are regarded to be midway between adjacent pixels (i.e., on the boundary of black and white square pixels), and thus the coordinates of the transition points are shifted 0.5 from the pixel centers. A pair consisting of a left edge and a right edge is called "run". Contour points are defined as the black points belonging to a figure whose neighboring pixels involve at least one white pixel. A set of contour points forms a contour line.

Three typical conventional contour-tracing algorithms are illustrated in Fig. 1, where the same example figure is used for ease of comparison. Following are brief descriptions of these algorithms.

A Pixel-center tracing (Yokoi et al. 1973; Suzuki et al. 1983). This algorithm is the most popular one, and traces the contour in a specific manner, e.g., in anti-clockwise direction by looking at the figure to its left, as shown in Fig. 1a. The tracing starts at the first contour point encountered when the image is scanned in a raster mode and ends when the starting contour point is revisited after tracing a closed loop. During the tracing, every contour point once visited is flagged to avoid double tracing. The resulting loop is a contour line, and the raster scanning starts again to find another non-flagged starting point.

B Pixel-corner tracing (Lee et al. 1982). This algorithm regards each pixel as a square, and the corners of the square of each contour point are subject to tracing. Tracing starts at a non-flagged pixel corner, and visits adjacent corners successively, until it reaches the starting corner again. The resulting loop is a contour which only consists of vertical and horizontal vector segments, as shown in Fig. 1b.

C Edge-point tracing of run data (Agrawala and Kulkarni 1977; Grant and Reid 1981; Carpson 1984). This algorithm is applied to the image expressed by the set of run data. Two adjacent horizontal scan lines are usually checked to see if the runs in both scan lines are appropriately connected to each other. The edge points of the two runs are then linked to describe the contour, as shown in Fig. 1c.

Algorithms A and B require a fairly large image memory as a frame buffer to cope with the complicated figures in large-scale drawings. This implies that the speed of contour tracing is essentially slow in these algorithms. When adequate image memory is not available, image data swapping occurs frequently, and the speed is further decreased. Moreover, the algorithm A usually generates an erroneous shape when the image is enlarged, especially at the portion having one-pixel width, as shown in Fig. 2a,b. Also, algorithm C cannot always restore the original shape, and distortion occurs at the concave portion of the figure. These indicate that the precision of the contours in the algorithm A and C is not sufficient to represent the original data exactly. The features of these three typical algorithms are summarized in Table 1. From this table, we can see that each algorithm somehow has drawbacks, either the tracing speed, accuracy of representation, or required buffer memory size.

3 Proposed contour-tracing algorithm

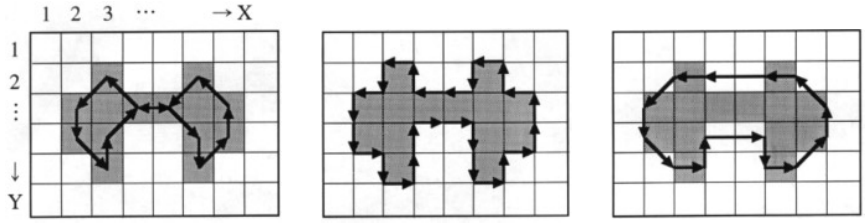
For high-speed and high-definition contour tracing, we propose a new algorithm that needs only a minimal image buffer memory. This algorithm utilizes the coordinate data of transition points at two adjacent scan lines, and converts them to codes called RD codes (Run-type Direction codes). As a result, the contour is represented by a set of chained RD codes. Therefore, in the following sections, the RD codes are explained first, then the method of extracting these RD codes from transition point data is described, and lastly the details of the tracing algorithm are explained together with the data structure for practical implementation.

3.1 Definition of RD codes

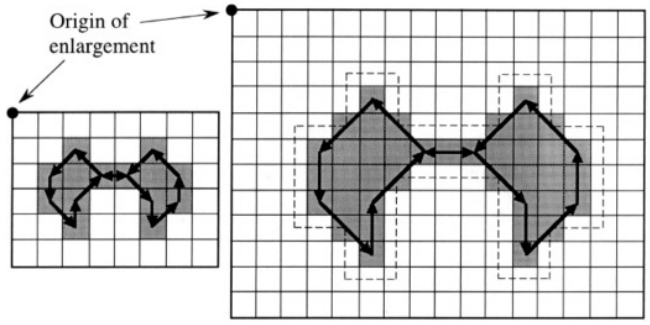
RD codes are a kind of chain code that is newly designed to describe the contour of digital binary images. It was attempted to exactly represent every local pattern that incorporates two adjacent scan lines by an RD code. Easy linking of the RD codes between scan lines was also attempted in designing them. The resulting RD codes consist of ten codes, as illustrated schematically in Fig. 3, each code corresponding to a local black and white pattern between two adjacent scan lines. These ten codes are the complete code set that can represent all local patterns appearing in digital binary images.

In this figure, a solid circle indicates the representative point of each code for specifying contour positions. The arrowheaded edge of each code pattern is called the "head" and the non-arrowheaded edge is called the "tail". The code is thus expressed by a three-part data set (X, Y, CODE), in which X and Y are the position coordinates of the representative point of the code and CODE is one of the code numbers (1) through (10) representing each code. Each code has a variable length in the horizontal direction and fixed height of 0.5 or 1 in the vertical direction. The heads and the tails have only two directions, i.e., upward and downward. In contour tracing, a downward head is always connected to an upward tail and a downward tail to an upward head, of a partner code. Figure 4 represents the result of contour tracing for the example figure shown in Fig. 1, where the contour is described by a serial set of RD codes $\{(1)(3)(2)(4)\ldots\ (10)\}$.

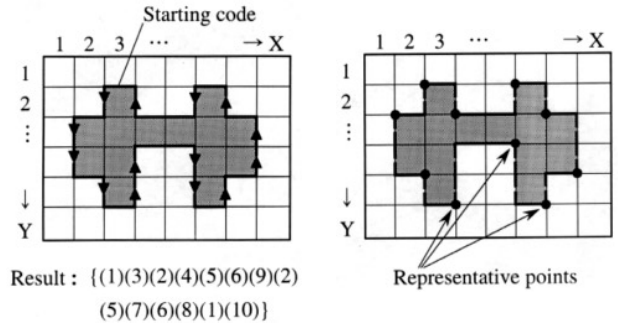
This serial set of RD codes is easily converted to a series of coordinates which is one of the ordinary forms of contour representation. In this case, however, the simple series of coordinates formed only by the combination of representative positions of the codes lacks intermediate position coordinates. Therefore, these intermediate coordinates must be added to complete the series by using the coordinates of two consecutive codes. For example, in the RD code set $\{(1)(3)(2)(4)\ldots\ (10)\}$ in Fig. 4a, the representative position of (1) is (2.5, 1.5) and that of (3) is (1.5, 2.5). Thus, the intermediate coordinates are (2.5, 2.5), and the final series of coordinates becomes $\{(2.5, 1.5)(2.5, 2.5)(1.5, 2.5)\ldots\}$. In the case of a long vertical contour portion, however, the codes (2) or (6) usually appear consecutively. In this case, the intermediate points derived from these consecutive codes are redundant. Therefore, in actual implementation, the representative points of the codes (2) and (6) are eliminated



1a 1b 1c



2a 2b

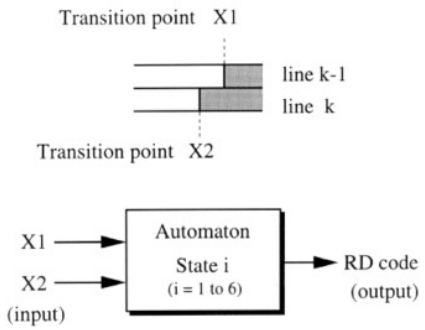


4a

4b

CODE	Local pattern	CODE	Local pattern
(1)		(6)	
(2)		(7)	
(3)		(8)	
(4)		(9)	
(5)		(10)	

3



5

Fig. 1a–c. Contour representation in conventional algorithms: **a** pixel-center tracing; **b** pixel-corner tracing; **c** edge-point tracing of run data

Fig. 2a,b. Contour enlargement and its restored image in the case of pixel-center-tracing algorithm: **a** original image and its contour; **b** enlarged (x 2) contour and its image restored by interpolating pixels on oblique vector segments (Broken lines indicate the direct enlargement of original image for comparison)

Fig. 3. RD codes to describe the local binary patterns

Fig. 4a,b. Result of the contour tracing using RD codes: **a** RD code chain obtained; **b** registered positions

Fig. 5. An automaton for RD code extraction

in advance when these codes are registered, as shown in Fig. 4b (and in Fig. 8b afterwards). The fractions involved in each coordinate shown above come from the fact that the pixel-centered coordinate system is adopted. In actual contour representation, these fractions can simply be ignored. The contour represented by the RD codes is equivalent to the one obtained by the conventional pixel corner-tracing algorithm, and thus the high-definition representation of figures can be realized by this algorithm.

3.2 Extraction of RD codes

The RD codes can be derived automatically while the image transition data are being scanned in a raster mode, i.e., right within a line and then downward from a line to the line below. The X coordinates of the two transition points each derived from the two different line buffers (i.e., the buffer for the previous scan line $k-1$ and the buffer for the current scan line k), are denoted as X1 and X2, respectively. After deriving the X1 and X2 from these line buffers, their

Table 1. Features of conventional algorithms

	Conventional algorithms		
	Pixel-center tracing	Pixel-corner tracing	Edge tracing of run-data
Tracing speed	Slow	Slow	Fast
Accuracy of restoration	Perfect	Perfect	Imperfect (at concave figure portion)
Accuracy of enlargement	Imperfect (at one-pixel width portion)	Perfect	Imperfect (at concave figure portion)
Required memory capacity	Large (frame buffer)	Large (frame buffer)	Small (line buffers)
Applications	For small-scale images where slower tracing is allowed	For small-scale images but high definition needed. (e.g., character font design)	For large-scale images where lower definition tracing is allowed. (e.g., industrial use)

Table 2. Aspects of state transition (in case of the example figure in Fig. 4, while line 3 is scanned)

State transition	1	→	3	→	4	→	5	→	4	→	5	→	1
X1	2.5		2.5		3.5		5.5		6.5		∞		∞
X2	1.5		7.5		7.5		7.5		7.5		7.5		∞
RD code output				(3)				(10)				(8)	

magnitudes are compared using an automaton, as shown in Fig. 5.

Figure 6 shows the state transition of the automaton. The automaton is designed so as to be equivalent to 8-neighbor-type pixel tracing where oblique direction tracing is allowed. In this figure,

$$\begin{bmatrix} \mathrm{B} \\ \mathrm{B} \end{bmatrix}, \begin{bmatrix} \mathrm{W} \\ \mathrm{B} \end{bmatrix}, \begin{bmatrix} \mathrm{B} \\ \mathrm{W} \end{bmatrix}, \text{ and } \begin{bmatrix} \mathrm{W} \\ \mathrm{W} \end{bmatrix}$$

indicate four possible combinations of the background portion (W) and the figure portion (B) of two adjacent scan lines, and the upper row and lower row correspond to the previous scan line and current scan line, respectively. When scanning at the background portion (W), the next transition (from W to B) is expected to occur at the left edge of the figure. Therefore, more precisely speaking, these combinations represent two transition points to be compared, where B indicates a left edge and W indicates a right edge. There are six states, 1 through 6, in this automaton, and at the starting (and thus the finishing) period of each horizontal line scan, the automaton returns to state 1. In each state, there are three types of transition available, depending upon the result of the comparison between X1 and X2, and a specific RD code is designated to be output at a certain transition (as indicated as a parenthesized numeral in Fig. 6). In each state transition, a new transition point is read out for the next comparison, from either of the two line buffers, depending upon the result of the previous comparison. The readout rules are: if X1 is smaller than X2, then a new X1 is read out from the line buffer for the previous scan line $(k-1)$; if X2 is smaller than X1, then a new X2 is read out from the line buffer for the current scan line (k); and if X1 is equal to X2, both the X1 and X2 are read out from each line buffer.

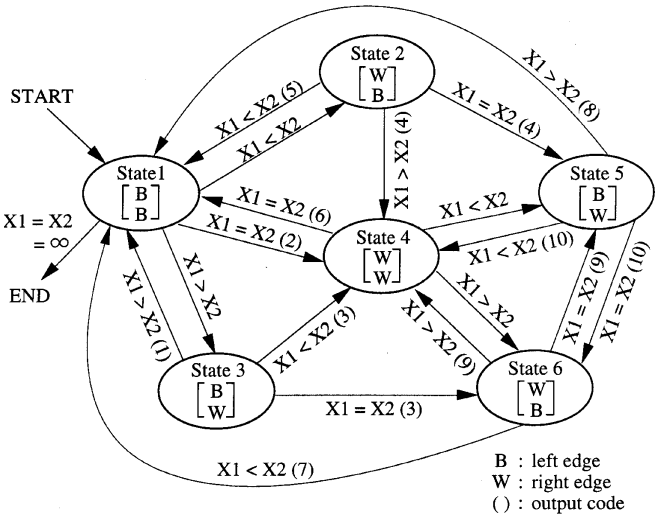
Based on this automaton, the condition for extracting RD code (1), for example, is as follows: $X1 > X2$ holds at state 1, the state changes to 3 after renewing X2, and again $X1 > X2$ holds at state 3. This physically means that the upper portion of this run is recognized as the background covering the run from its left edge through its right edge. The relations between the state transition and the coordinate renewal can be explained by examining the third scan line (Y = 3) in Fig. 4. The data of the previous scan line (Y = 2) are $\{(2.5, 3.5, 5.5, 6.5, \infty\}$, and the data of the current scan line (Y = 3) are $\{1.5, 7.5, \infty\}$ where ∞ means both a large number and a terminate symbol of the scan line data. First, X1 and X2 are set to 2.5 and 1.5, respectively. Then, the state transition occurs as shown in Table 2, based on the automaton in Fig. 6. The processing for the third scan line is completed when both the X1 and X2 become ∞. Thus, the output RD codes during the third scan line are (3), (10), and (8).

Table 3 summarizes the concrete operations that must be executed for each state, based on the state transition diagram in Fig. 6. By judging the condition $X1 : X2$, where the symbol ":" indicates one of the comparison symbols ">", "<", or "=", the following processes are invoked for execution:

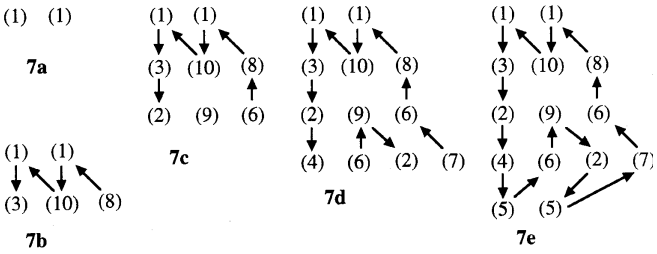
- preparation of a new X coordinate for the registration of representative point,
- registration of extracted RD code,
- renewal of a state register representing present state to the next state number, and
- readout of the next X1 and/or X2 coordinates for subsequent processing.

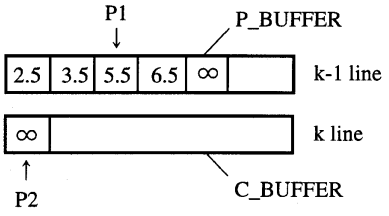
At the registration of the extracted RD code, the linking operation is also executed between the extracted code and previously found codes. With the exception of RD codes (5) and (10), codes have to wait for linking until the pairing RD code appears during the processing of the next scan line. This linking operation is a non-search process, and straightforward linking is executed using pointers that indicate the top and bottom addresses of a waiting list.

Figure 7 illustrates the aspects of the linking process for each scan line using the example figure shown in Fig. 4. Figure 7a is the result after scan line 2 (Y = 2) and two RD codes (1) are extracted. Figure 7b is the result after scan

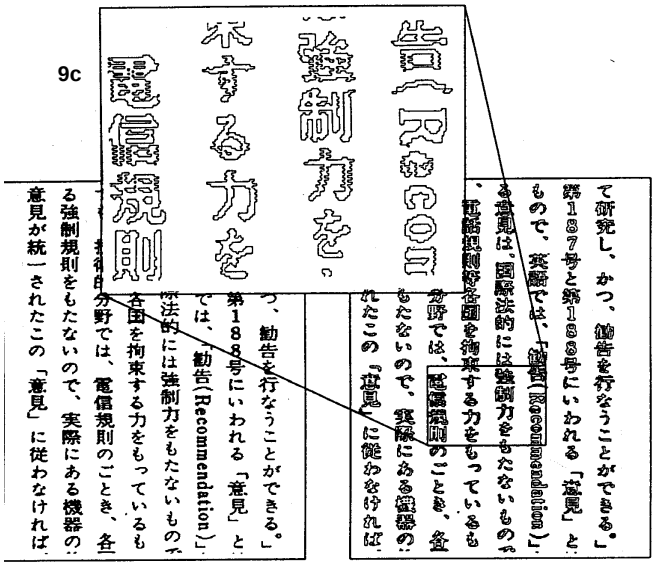


6





8a



9a 9b 9c

RD_LIST

Address	X	Y	CODE	LINK	W_LINK	
1	2.5	1.5	(1)	3	2	
2	5.5	1.5	(1)	4	3	
3	1.5	2.5	(3)	6	5	
4	3.5	2.5	(10)	1	-	
5	6.5	2.5	(8)	2	6	
6	-	-	(2)	9	7	
7	5.5	3.5	(9)	11	8	
8	-	-	(6)	5	9	
9	2.5	4.5	(4)		10	←P4
10	-	-	(6)	7	11	
11	-	-	(2)		12	
12	7.5	4.5	(7)	8		←P5
13						
14						

P3→ 12

8b

Fig. 6. The state transition of the automaton used for RD code extraction

Fig. 7a–e. RD code links after each line is scanned (for the example figure shown in Fig. 4): **a** scan line 2; **b** scan line 3; **c** scan line 4; **d** scan line 5; **e** scan line 6

Fig. 8a,b. Data structures for contour tracing: **a** data structure of line buffers; **b** data structure of output table RD_LIST (in the case of the example figure shown in Fig. 4, after line 5 is scanned)

Fig. 9a–c. Contour tracing by the proposed method for standard image #7 (Japanese document): **a** original image (approximately 1/16 area of A4 size is shown); **b** contour image for the original image; **c** details enlarged

line 3 (Y = 3) and the new RD codes (3), (10), and (8) are extracted and are linked with RD codes (1) extracted during the processing of the previous scan line. After scan line 6 (Y = 6), all RD codes are linked as shown in Fig. 7e, and thus, the representation of the contour is completed.

As can be understood from the above, the proposed algorithm does not actually trace the contour points in the same way as conventional pixel center or pixel corner-tracing algorithms. Therefore, this algorithm would be better described as a contour representation algorithm. In this paper, however, the words “contour tracing” are also used to describe this algorithm according to convention.

3.3 Data structure

The contour-tracing algorithm based on RD codes requires two types of data memory: line buffers, P_BUFFER and C_BUFFER, for holding the horizontal coordinates of the transition points of the previous and current scan lines, and an output table, RD_LIST, for the registration of extracted

Table 3. Interpretation of the automaton based on the positional relations between transition points

Condition		Execution			
State	X1:X2	Coordinate renewal	RD code extraction	New state	Readout line of next coordinate
	$<$	-	-	2	Previous
1	$=$	-	(2)	4	Both
	$>$	X←X2	-	3	Current
	$<$	X←X1	(5)	1	Previous
2	$=$	X←X1	(4)	5	Both
	$>$	X←X2	(4)	4	Current
	$<$	-	(3)	4	Previous
3	$=$	-	(3)	6	Both
	$>$	-	(1)	1	Current
	$<$	X←X1	-	5	Previous
4	$=$	-	(6)	1	Both
	$>$	-	-	6	Current
	$<$	-	(10)	4	Previous
5	$=$	-	(10)	6	Both
	$>$	-	(8)	1	Current
	$<$	X←X1	(7)	1	Previous
6	$=$	X←X1	(9)	5	Both
	$>$	X←X2	(9)	4	Current

RD codes and link data. Figure 8 illustrates the data structure in these memories.

Pointers P1 and P2 indicate readout positions of the transition point data X1 and X2 in the P_BUFFER and C_BUFFER, respectively. When the processing of the next scan line starts, the contents of C_BUFFER must, in general, be shifted to P_BUFFER and then the C_BUFFER must be loaded with the coordinate data of the next scan line. However, in the actual implementation, the roles of P_BUFFER and C_BUFFER are alternately switched in every scan line, thus eliminating the operation of shifting the contents from one buffer to the other.

The RD_LIST is a two-dimensional table consisting of five columns in total. The first three are for RD codes, where the set data (X, Y, CODE) are stacked in the order they are extracted. The LINK column stores a link pointer that points to the address where the pairing RD code is stored. The W_LINK column is used for a working memory, where a temporarily linked pointer for the waiting RD code is stored. The contents of W_LINK are thus substantially the code address to be processed after the present code address.

To manipulate this RD_LIST, three additional pointers P3, P4, and P5 are used. Pointer P3 addresses the point of entry for the newly detected RD code, and is increased every time a new RD code is found. Pointers P4 and P5 respectively indicate the top and bottom addresses of the RD codes which are waiting for linking. When the extracted RD code is the one that can be linked upward, the corresponding code address is directly stored in the LINK pointed by P4, and the pointer is advanced to the next waiting address. Thus, the linking operation is straightforward without any search-type processing. All linking processes complete when the pointer P4 coincides with P5. Figure 8b shows the intermediate result for the example figure shown in Fig. 4, when the scanning of line 5 is just finished. Vacant spaces in the RD_LIST will be filled during the subsequent scanning of line 6.

3.4 Conversion to coordinate series

After this algorithm is processed, conversion of the RD code set to the actual coordinate series is sometimes necessary. This conversion can be easily executed by referring to the RD_LIST between the top address 1 and the bottom address indicated by pointer P3. First, RD code (1) or (9) is found from the CODE column. By making this the starting point, the LINK column is referred to and the address indicated by the content of the LINK column is then visited for finding the next RD code. This process is repeated until the address returns to the starting point. Every X and Y coordinate thus visited is sequentially output while interpolating the intermediate position coordinates. As mentioned in Sect. 3.1, the interpolation rule is to generate a new intermediate coordinate (X, Y) by combining the X coordinate of the previous RD code and the Y coordinate from the current RD code. The deletion of the successive vertical codes (2) and (6) is not necessary in this conversion process, as the registration of coordinates to the X and Y columns of the RD_LIST is omitted when these codes are found. To avoid a double visit, the RD code once visited is flagged in this conversion process. The resulting contour is the outer contour of a figure if the starting code is (1), and is the inner contour of a figure if the starting code is (9).

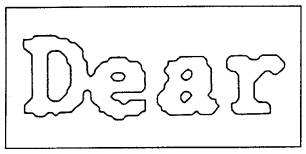
As mentioned above, the new contour-tracing algorithm executes the extraction, registration, and linking of RD codes automatically in accordance with the input of transition point coordinates. Execution does not need any random-access search process, only accessing data of two adjacent scan lines sequentially according to the increasing order of the scan line numbers. Thus, the algorithm is a one-pass type, and finishes the contour representation at the same time as the input of the coordinates of all the scan lines is completed.

4 Experimental verification

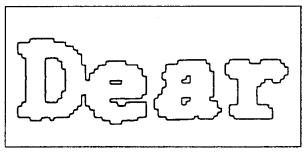
4.1 Method of experiments

Experimental verification was performed on a general-purpose computer (Hitachi, M680H). Each algorithm was programmed using the FORTRAN language. Eight standard CCITT images (#1 through #8) were used for the experiments. The size of these images, 1728×2287 pixels, was obtained by scanning the original A4-size standard images with the 8 dots/mm resolution. As the main purpose of this study was to realize the high-speed and high-definition contour-tracing algorithm, the conventional and most popular high-definition algorithm, that is, the pixel-center-tracing algorithm (Yokoi et al. 1973) was used for the comparison. This algorithm has the same precision as the proposed algorithm in the sense that both can restore the original image perfectly from the obtained contour data.

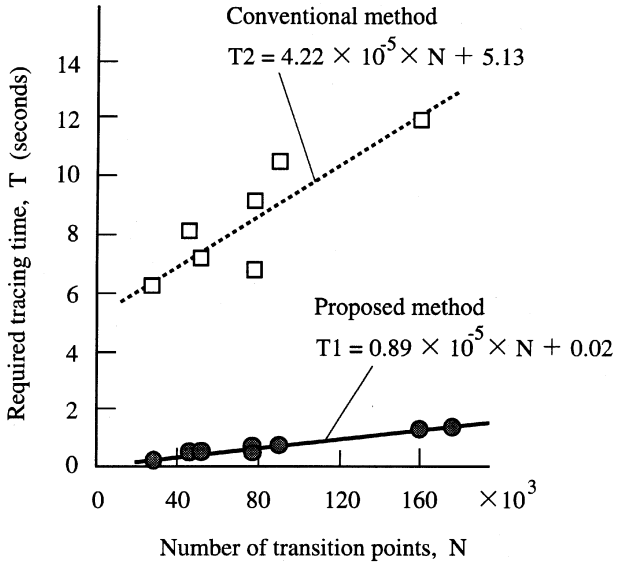
Figure 9 shows, as an example, a part of the original CCITT image #7 and its contour derived by the proposed algorithm. The enlarged contours of a small portion selected



10a



10b



11

Fig. 10a,b. Examples of contour tracing (enlarged): **a** conventional method; **b** proposed method

Fig. 11. Comparison of contour tracing time

from CCITT image #1 is shown in Fig. 10 to compare the results obtained by means of the conventional and the proposed algorithms. Here, the contours are expressed as sets of vector segments by connecting each coordinate in succession. The differences in appearance are due only to the fact that the conventional algorithm is based on the pixel centers and the proposed algorithm is based, equivalently, on the pixel corners. There is no intrinsic difference in precision between the two algorithms, as both can restore the original image perfectly. The only exception is that the conventional pixel-center-tracing algorithm may generate an erroneous shape, as shown already in Fig. 2, especially at the one-pixel width portion of the figure, when it is enlarged. The proposed algorithm, however, never generates such erroneous figures, and the result is exactly the same as that obtained by the pixel-corner-tracing algorithm (Lee et al. 1982).

4.2 Experimental results

Execution time of the conventional pixel-center-tracing algorithm and the proposed algorithm for each standard image is compared in Table 4. The proposed algorithm is 8 to 25 times faster than the conventional algorithm, the average time required for these A4-size document images being 0.78 s. The smallest improvement in speed using the proposed algorithm was observed in the case of image #7 (a document written in Japanese) and the greatest improvement in speed was observed in the case of image #2 (a circuit diagram). This is because the circuit diagram includes long horizontal lines and thus the number of transition points is small.

Table 4. Comparison of performance

CCITT standard images (1728×2287 pixels)	Number of transition points, N	Required tracing time, T (s) T(s)		Ratio (T2/T1)
		Proposed algorithm (T1)	Conventional algorithm (T2)	
# 1 Business letter	45,806	0.44	8.11	18.4
# 2 Circuit diagram	25,750	0.25	6.22	25.0
# 3 Sales order table	79,168	0.71	9.13	12.9
# 4 French document	175,830	1.63	12.55*	7.7*
# 5 Technical paper	90,950	0.80	10.53	13.2
# 6 Technical graph	51,530	0.47	7.23	15.4
# 7 Japanese document	160,686	1.46	11.92	8.2
# 8 Handwritten memo	79,814	0.46	6.81	14.8
Average	89,030	0.78	8.56	15.4

*Estimated values (Not examined because of the memory capacity limitation)

In general, the time required for contour tracing depends not only on the image size, but also on the complexity of the figures involved in the image. Therefore, by using the number of transition points as a measure of complexity, the time increase due to complexity is examined. Fig. 11 graphically illustrates the experimental results of contour-tracing time in Table 4 by taking the number of transition points in the horizontal axis. For both the conventional algorithm and the proposed algorithm, the contour-tracing time increases linearly as the number of transition points increases.

Though the time increase ratio per transition point only differs 4.7 times (= 4.22/0.89), the total speed of the proposed algorithm is 15 times better on average than the conventional algorithm. This is because the number-independent term (5.13 s) for the conventional algorithm is much higher than that of the proposed algorithm (0.02 s). However, it is estimated that the speed of the proposed algorithm is similar to or slightly higher than that of another conventional algorithm, the edge-point-tracing algorithm of run-data, shown in Fig. 1c.

Experiments using the recent RISC-type workstation (Silicon Graphics, ONYX) and C-language programs were also performed and a similar tendency was observed. The average processing time of the proposed algorithm was 0.29 s, instead of 0.78 s in the previous case.

5 Conclusion

A high-definition-type fast algorithm for representing contours of binary images is described in this paper. The algorithm is based on the image transition points of each scan line, and a unique set of chain codes consisting of 10 RD codes is proposed for expressing all types of local figure patterns existing in binary images.

The RD codes are extracted by examining the relations between the two adjacent scan lines, using the 6-state automaton with three transition rules in each state. To link the RD codes, straightforward pointer-based techniques are used, thus eliminating the complex search-type processing

usually used in conventional algorithms. The proposed algorithm is applied to the actual A4-size standard document images and the effectiveness of the algorithm is verified. The features of this algorithm are summarized as follows.

- *High-speed capability*; utilizes image transition points in each scan line instead of original pixel data, thus reducing the access workload.
- *High-definition capability*; restores original images perfectly with no distortion, even in the case of enlargement.
- *Small memory capacity*; requires only two line buffers for coordinates of image transition points, thus eliminating the need for image buffers.
- *Concise algorithm*; achieves one-pass-type processing, based on the state transition of an automaton.

Thus, it is easy to use the algorithm for the processing of large-scale binary images such as office documents, engineering drawings, and topographical maps. Furthermore, as the algorithm mainly handles coordinate data, it is suitable for general-purpose computers, which usually have a high capability in numerical processing. It is also feasible that, with the recent progress in computer technology, even small-scale personal computers will be able to execute the algorithm effectively.

References

1. Agrawala AK, Kulkarni AV (1977) A sequential approach to the extraction of shape features. Comput Graphics Image Process 6:538-557
2. Carpson DW (1984) An improved algorithm for the sequential extraction of boundaries from a raster scan. Comput Graphics Image Process 28:109-125
3. Ejiri M, Kakumoto S, Miyatake T, Shimada S, Matsushima H (1984) Automatic recognition of design drawings and maps. In Proc. 7th Int. Conf. on Pattern Recognition. Montreal, pp.1296-1305
4. Grant G, Reid AF (1981) An efficient algorithm for the boundary tracing and feature extraction. Comput Graphics Image Process 17:225-237
5. Lee H, Yokoi S, Toriwaki J, Fukumura A (1982) Border following and reconstruction of binary pictures using grid point representation. Trans. IEICE J65-D 10:1203-1210 (In Japanese)
6. Matsuzawa S, Miyatake T, Matsushima H (1983) Hatched area extraction algorithm for topographic map recognition. In: Proc. IEICE Information and Systems Division Conference 57, IEICE, Tokyo (In Japanese)
7. Miyatake T, Matsushima H, Ejiri M (1985) Extraction of roads from topographical maps using a parallel line extraction algorithm. Trans. IEICE J68-D 2:153-160 (In Japanese)
8. Miyatake T, Matsushima H, Ejiri M (1987) A fast algorithm for contour tracing using run-type direction code. In: Proc. IEICE Annual Conference 1554, IEICE, Tokyo (In Japanese)
9. Rosenfeld A, Kak AC (1976) Digital picture processing. Academic Press, New York, pp. 341-347
10. Suzuki S, Abe K (1983) Border following algorithms for analyzing the topological structure of digitized binary images. Technical Report of IEICE, PRL83-2, pp.9-16 (In Japanese)
11. Yokoi S, Toriwaki J, Fukumura A (1973) An analysis of topological properties of digitized binary pictures using local features. Trans. IEICE J56-D 11:662-669 (In Japanese)

Note: IEICE stands for the Institute of Electronics, Information and Communication Engineers of Japan, Tokyo

Takafumi Miyatake graduated from Tadotsu Technical High School, Kagawa, Japan, in 1971, and from Hitachi Technical College, Kanagawa, Japan, in 1973. From 1981 to 1982 he was a Visiting Researcher at Kyoto University, Kyoto, Japan. Since 1971 he has been with the Central Research Laboratory of Hitachi Ltd. and is presently a Senior Engineer at its Multimedia Research Department. He has been engaged in R&D activities in the fields of 3D object recognition, picture processing, and drawing recognition, and his recent interest is being extended to real-time video analysis for multimedia systems. He received his Doctoral Degree in Infomation Engineering from the University of Tokyo in 1996.

Hitoshi Matsushima received his BE degree and ME degree in Electrical Engineering from Kyoto University, Kyoto, Japan, in 1969 and 1971, respectively. He joined the Central Research Laboratory of Hitachi Ltd. in 1971 and engaged in the R&D in the field of image processing architecture, drawing recognition algorithms, and their application to geographic information systems. He moved to Telecommunications Division of Hitachi Ltd. in 1991 and is presently a Chief Engineer at its Development Center. His recent interest includes digital picture phone, computer telephony and related multimedia systems.

Masakazu Ejiri received his BE degree in Mechanical Engineering and his Ph.D. degree in Electrical Engineering, both from Osaka University, Japan, in 1959 and 1967, respectively. Since 1959, he has been with the Central Research Laboratory of Hitachi Ltd., and is presently a Senior Chief Research Scientist, Corporate Technology, both at the Central Research Laboratory and the Mechanical Engineering Research Laboratory. His research areas are control engineering, pattern recognition, robotics, machine vision , and artificial intelligence. He is a Fellow of the IEEE and a Fellow of the IAPR. He serves as an editorial board member for Machine Vision and Applications.