TR-655 April 1978
MCS-76-23763

THE RUN LENGTH MAP: A REPRESENTATION OF COUNTOURS AND REGIONS FOR EFFICIENT SEARCH & LOW LEVEL SEMANTIC ENCODING

Peter Lemkin
Image Processing Unit
Division of Cancer Biology and Diagnosis
National Cancer Institute
National Institute of Health
Bethesda, MD 20014

Computer Science Dept.
University of Maryland
College Park, MD 20742

COMPUTER SCIENCE
TECHNICAL REPORT SERIES

UNIVERSITY OF MARYLAND
COLLEGE PARK, MARYLAND
20742

TR-655 April 1978
MCS-76-23763

THE RUN LENGTH MAP:
A REPRESENTATION OF COUNTOURS AND REGIONS FOR EFFICIENT SEARCH & LOW LEVEL SEMANTIC ENCODING

Peter Lemkin
Image Processing Unit
Division of Cancer Biology and Diagnosis
National Cancer Institute
National Institute of Health
Bethesda, MD 20014

Computer Science Dept.
University of Maryland
College Park, MD 20742

ABSTRACT

The RLM (run length map) algorithm is a boundary enclоding algorithm which produces a representation of the boundary which may be used for determining insidedness and outsidedness of a boundary. The algorithm was developed for use with a boundary follower segmentation algorithm when filling holes. It may be used for testing regions as well. It may also be used in labeling boundaries according to orientation-dependent local boundary semantics taking the global boundary model into account. Analyzing these label sequences produces the semantic labeling run length distribution and (0,90) rotational co-occurrence matrix. It is felt that they may be used to indicate local shape features.

The support of the Division of Mathematical and Computer Sciences, National Science Foundation for the publication of this report is gratefully acknowledged.

1. INTRODUCTION

A region boundary may be represented as an ordered (by row) set of run lengths called a run length map (RLM). This has the advantage of efficient contour encoding and region denotation. Furthermore, the algorithm presented here for mapping a boundary into a run length map produces low level semantics of the parts of the curve. The algorithm for producing the run length map is linear with respect to the length of the boundary. The algorithm for testing whether a point in the plane is within a given region on line y of the plane is of order less than Ry where Ry is the number of runs on line y.

Minsky and Papert [1] proved that it is impossible to determine whether a point in a plane is inside or outside of a closed curve unless one analyzes the entire curve and is able to either revisit various points on the curve or have infinite memory to record information about the curve. Thus an arbitrarly long curve would require either an arbitrarly large memory or an arbitrarily large number of revisitings of points on the curve. Revisiting the curve uses little memory but much time, whereas a large amount of memory allows one to search the representation quickly.

Several schemes have been proposed for mapping the

information about the curve into essentially infinite memory such that efficient search of the memory is possible as well as efficient generation of the representation from the original curve. Freeman [2] reviews several of these methods, emphasizing chain coding schemes.

Davis [3] discusses understanding the shape of an object in terms of recognizing high curvature points on a boundary and then performing a hierarchical angle and side analysis.

Burton [4] represents planar curves and regions as an ordered tree of polygonal approximations organized for efficient binary search. Rutovitz [5] discusses algorithms for and representations of object boundaries by sets of run lengths and some of the properties of such representations.

Rounds [6] discusses an algorithm for mapping boundaries into representations for region extraction. She maps the types of the boundary points into an initially zeroed image. Regions are then extracted by raster scanning the image and recognizing insidedness based on type transitions. Whether a pixel (x',y') is in a region or not requires computing all of the points on line y' from the left edge to x'. Thus the average calculation cost of determining the insidedness property of a random point is of Order (average object

width/2).

Agrawala [7] discusses several run length shape features obtained by a raster sweep through an image keeping a two line memory of the image. This method is similar to the method being proposed in its intent of acquiring shape information from the run length coding of the boundary. The mechanism is different, however, as the algorithm being proposed here uses the ordered set of points along the boundary rather than the points on the boundary detected during a horizontal raster scan. Furthermore, the method proposed here labels boundary points with a richer set of semantic labels which can then be used for further boundary parsing.

Merrill [8] proposed a scheme for sorting an ordered set of boundary points by line and ordering them within a line. The algorithm uses the topological property that a line drawn through a closed object has an even number of boundary crossings. Merrill chooses to represent this property of the curve by ordered lists of the actual boundary points, repeating points where necessary in order to validate the requirement of an even number of points to denote a closed object.

The algorithm proposed here is similar in many respects to Merrill's. Both algorithms operate on a "tightly closed

boundary" (TCB), which is continuous, closed, and contains no loops. Merrill discusses another restriction which has to do with testing lines that are tangent to a horizontal part of the curve. This latter restriction is handled differently in the present algorithm.

A TCB consisting of a list of (x,y) coordinates is partitioned into sets such that each set contains only points which have the same y-coordinate. The associated x-coordinates of each set are ordered in monotonically increasing order. The ordered set is called a y-partition of the TCB. Merrill notes that a region Rk of length b has a TCB, Pk, defined by (1) through (5).

$$TCB = ((x,y)i \text{ in } Rk \text{ for } i=1 \text{ to } b), \tag{1}$$

$$Pk = \{kYmin, kYmax, kY1, kY2, \ldots, kYnk\}, \tag{2}$$

where:

$$kYmin = Yi \mid (Yi \text{ leq } Yj \text{ in } TCB), \tag{3}$$

$$kYmax = Yi \mid (Yi \text{ geq } Yj \text{ in } TCB), \tag{4}$$

and

$$nk = kYmax - kYmin + 1. \qquad (5)$$

The Y-partition of Pk for Yi is given by (6).

$$kYi = (rik,\ ikX1, ikX2, \ldots, ikXrik) \qquad (6)$$

where rik is the number of Yi-coordinates in the TCB of Rk and is always an even number. The ikXj are the points on the TCB having Yi-coordinates. Thus regions inside of the TCB are those x such that an odd-even [ikXj,ikXj+1] exists such that equation (7) holds.

$$ikXj \text{ leq } x \text{ leq } ikXj+1. \qquad (7)$$

Horizontal boundary points are all entered into the data structure, resulting in redundent information being saved.

What is minimally required is to save the the first and last ikXj points on such horizontal boundary segments. This is equivalent to coding the boundary as Y-partitions of boundary run lengths if the object were colored in and the run lengths represented the colored in regions. While Merrill implicitly represents the TCB as run lengths, the present algorithm explicitly represents it as run lengths.

The Y-partitions may be generated with minimum

difficulty using Merrill's algorithm taking his last restriction into account. The Y-partitions generated may then be easily converted to run lengths.

The present algorithm differs in two respects from the TCB generation algorithm of Merrill. The runs are ordered by their occurrence (although this need not be so), and local semantics are produced as a result of the mapping process. In what follows, the algorithm will be refered to as the RLM (Run Length Map) algorithm.

The co-ordinate system to be used in the rest of the paper is called the Logical Co-ordinate System (LCS) and is the convention in the Buffer Memory Monitor System (BMON2) [9] and the Real Time Picture Processor (RTPP) ([10], [11], [12]). For an image of size NxN pixels, the (x,y) co-ordinates at the upper left hand corner are (0,0) and those at the lower right hand corner are (N-1,N-1). The RTPP supports two sizes of images N=256, and N=1024. The former size is used for the examples in this paper. Data for the examples in the paper are taken from previous work by the author [13].

2. LOW LEVEL SEMANTICS OF RLM

Because RLM generation recognizes various local boundary properties in processing the boundary, it is possible to label runs and boundary points with the corresponding local semantics used to generate or modify the runs. Later analysis of these local semantic labels may be used to detect local shape features.

The RLM generator algorithm uses six parallel finite state machine recognizers which look at the boundary in parallel (implementing a non-deterministic FSM recognizer (NDR)). The recognizers are mutually exclusive; only one of them is activated at each point on the boundary. Each NDR actually contains a subset of NDRs. These are listed below in Table 1. The software implementation is of course sequential, but a parallel hardware implementation could be realized. These six recognizers are discussed in more detail later in this paper.

The RLM search algorithm checks whether a point (x,y) is inside of boundary Rk by searching the runs on line y for x inside of any run.

Index	Code	Semantics
1	CGLHSP	explicit split change HAIREP to left split
2	CGRHSP	explicit split change HAIREP to right split
3	CPALFT	complete left adjacent run
4	CPARHT	complete right adjacent run
5	CPNLFT	complete left non-adjacent run
6	CPNRHT	complete right non-adjacent run
7	CMPUNY	complete unary run
8	DUP-PT	duplicate point (in algorithm section [G.2])
9	EXALFT	extend left adjacent run
10	EXARHT	extend right adjacent run
11	EXNLFT	extend left non-adjacent run
12	EXNRHT	extend right non-adjacent run
13	HAIREP	new run hair end point
14	IMPLFT	implied split left CCW split
15	IMPRHT	implied split right CCW split
16	IMPLFH	implied split left hair
17	IMPRHH	implied split right hair
18	INCDPT	explicit split included point
19	MRGLFT	merge run left
20	MRGRHT	merge run right
21	NEWRUN	new run started
22	NULPNT	null point (in algorithm section [G.2])
23	SPCW-L	explicit split CW left
24	SPCW-R	explicit split CW right
25	SPCCWL	explicit split CCW left
26	SPCCWR	explicit split CCW right

Table 1. The following alphabetic list of codes are generated by a NDR upon accepting a boundary point.

2.1 SEMANTIC LABELINGS FOR RUNS

Distinct boundary conditions are detected by each recognizer and marked in the run. These semantic conditions are a result of using the twenty six FSM recognizers listed in Table 1. The set of twenty six semantic labels is called Lsem (where an element of Lsem is SEMpoint) and will be used later in this paper. The seven types of arcs recognized are shown in Figures 1 through 7.

2.1.1 SINGLE POINT WIDTH CRACK DETECTION

Cracks in a boundary are defined as points which go into the inside of the boundary and then come out at the same point. Vertical cracks can be detected as shown in Figure 1 as follows. Left run cracks are detected by (8).

$$(x=En) \text{ and } SEM[En:Xn]=(CPALFT \text{ or } CPNLFT \text{ or } EXALFT \text{ or } EXNLFT). \tag{8}$$

Left run cracks are detected by (9).

$$(x=Xn) \text{ and } SEM[En:Xn]=(CPARHT \text{ or } CPNRHT \text{ or } EXARHT \text{ or } EXNRHT). \tag{9}$$

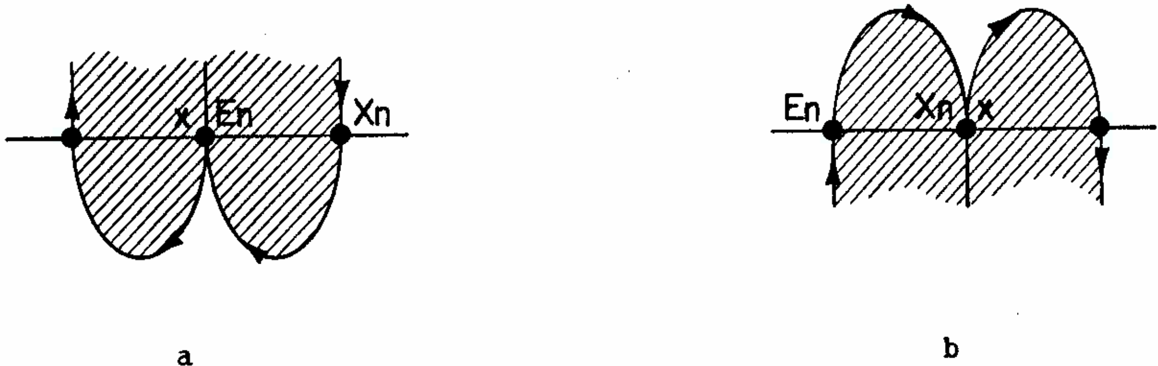


Figure 1. Single point wide crack detection using the fact that a point x occurs at the same end point in the last finished run. a) Left run crack detector, b) right run crack detector.

2.1.2 MULTIPLE LOBE DETECTION

Multiple lobes can be detected if they appear either vertically upward or downward as in Figure 2. For a row y intersecting several lobes several runs are generated. The semantic labels of the runs must be run completions the first

time the run is defined. Thus, the number of lobes corresponds to the number of runs with these labels. Downward lobes correspond to (CPNLFT or CPALFT) semantic labels with equation (10) holding while upward lobes correspond to (CPNRHT or CPARHT) labels with (11).

```
(SEMpoint[E1:X1] = CPNLFT or CPALFT) and          (10)
(SEMpoint[E2:X2] = CPNLFT or CPALFT)

(SEMpoint[E1:X1] = CPNRHT or CPARHT) and          (11)
(SEMpoint[E2:X2] = CPNRHT or CPARHT)
```

The number of downward pointing lobes is the number of runs labeled with CPxLFT. The number of upward pointing lobes is the number of runs labeled with CPxRHT.

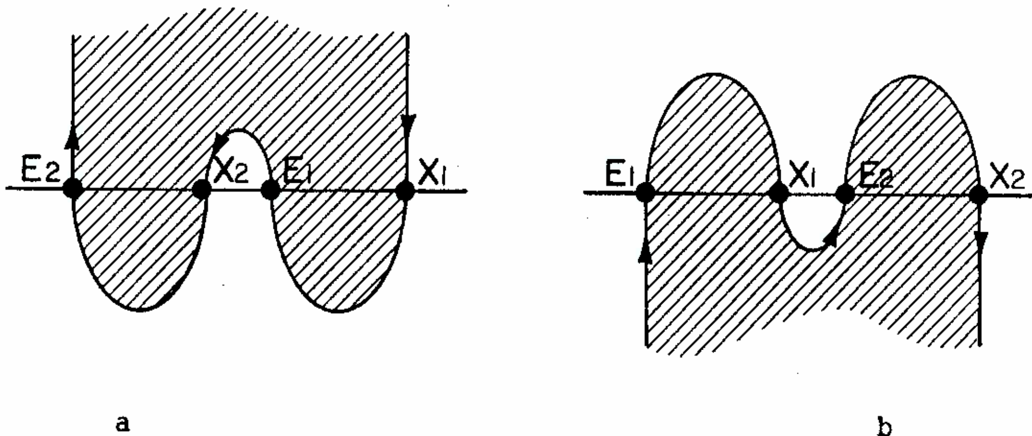


Figure 2. Multiple lobe detection using a count the the number of non-adjacent run completions if there are n runs on a line y. a) Downward lobe detection, b) upward lobe detection.

2.1.3 CONCAVE SURFACE DETECTION

Points on a concave surface of a concave produced by upward or downward lobes can be detected as in Figure 3. The

concave surface starts out as NEWRUNs but is determined to be a concavity when the second surface is analyzed to be implicit splits. Concave upward surfaces are detected with IMPLFT while concave downward surfaces are detected with IMPRHT semantic labels.

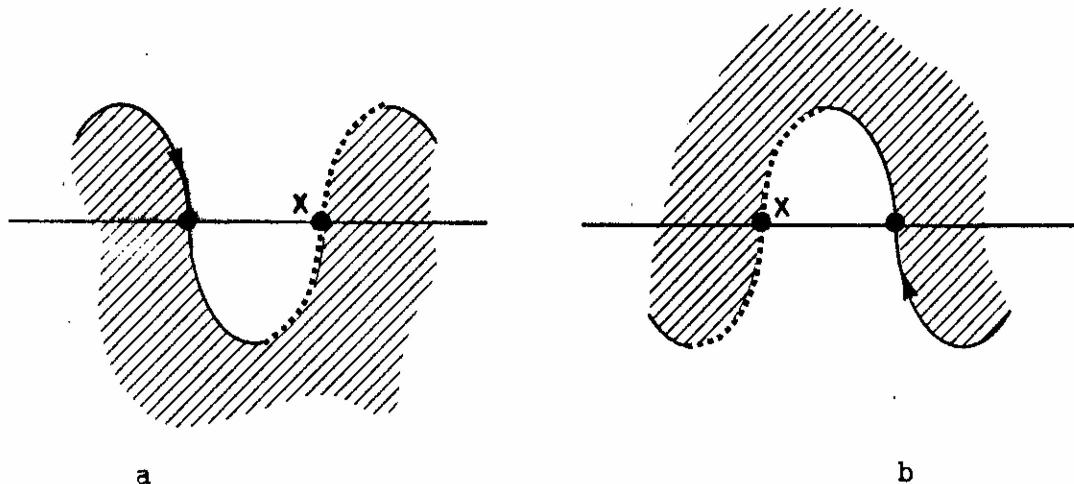


Figure 3. Concave surface detection by the implicit split recognizer. Concavity is detected by the NDR Implicit-Split at x. Note that the run surface opposite x can be inferred to be concave.

2.1.4 CLOCKWISE SPIRAL DETECTION

Spirals are divided into two classes, clockwise (CS) and counterclockwise (CCW). CW spirals are detected by the existence of explicit CW split semantic labels as in Figure 4. Both left and right splits must be present and the level of spiral at a row y is the number of SPCW-L/SPCW-R label pairs found. It is also possible to determine whether the spiral started from the top or bottom if SPCW-R or SPCW-L (respectively) is seen first. The following six labels are found for Figure 4.

Point	SEMpoint
1	NEWRUN
2	CPNRHT
3	SPCW-L
4	SPCW-R
5	MRGRHT
6	MRGLFT

2.1.5 COUNTERCLOCKWISE SPIRAL DETECTION

CCW spirals are detected by the existence of implicit split semantic labels as in Figure 5. Both left and right implicit splits must be present and the level of spiral at a row y is the number of IMPLFT/IMPRHT label pairs found. It is also possible to determine whether the spiral started from the top or bottom if the IMPLFT or IMPRHT (respectively) is seen first. The following six labels are found for Figure 5.

Point	SEMpoint
1	NEWRUN
2	IMPLFT
3	IMPRHT
4	CPNLFT
5	MPGRHT
6	MRGLFT

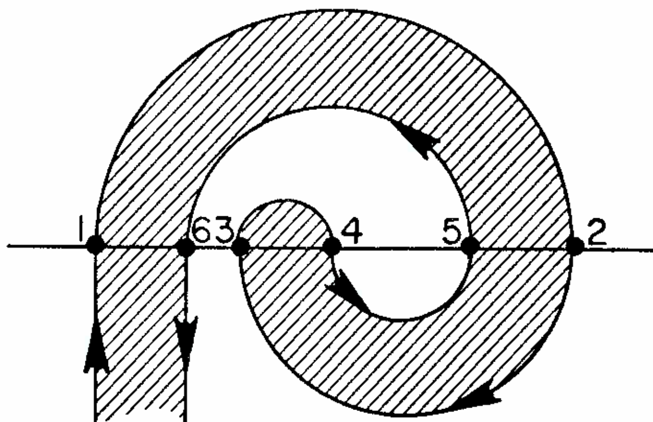


Figure 4. Clockwise spiral detection using pairs of CW split recognizers SPCW-L/R.

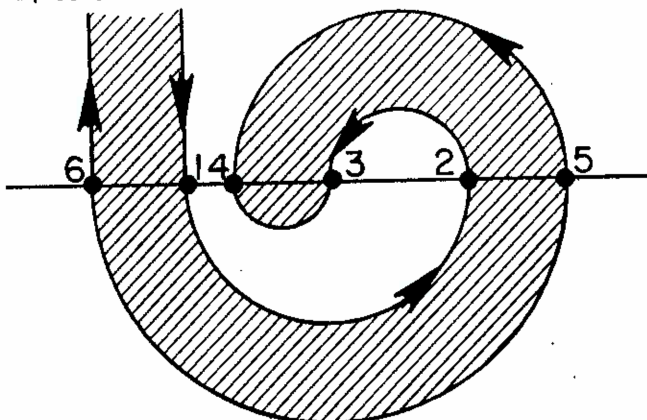


Figure 5. Counterclockwise spiral detection using pairs of CCW split recognizers IMPLFT/RHT.

2.1.6 HORIZONTAL ADJACENT POINT DETECTION

When a run is completed, sometimes additional points are added as in Figure 6. These points are called horizontal adjacent points and are detected by the CPALFT or CPARHT semantic labels. After an adjacent point is recognized, it may be extended further with successive applications of EXALFT or EXARHT respectively.

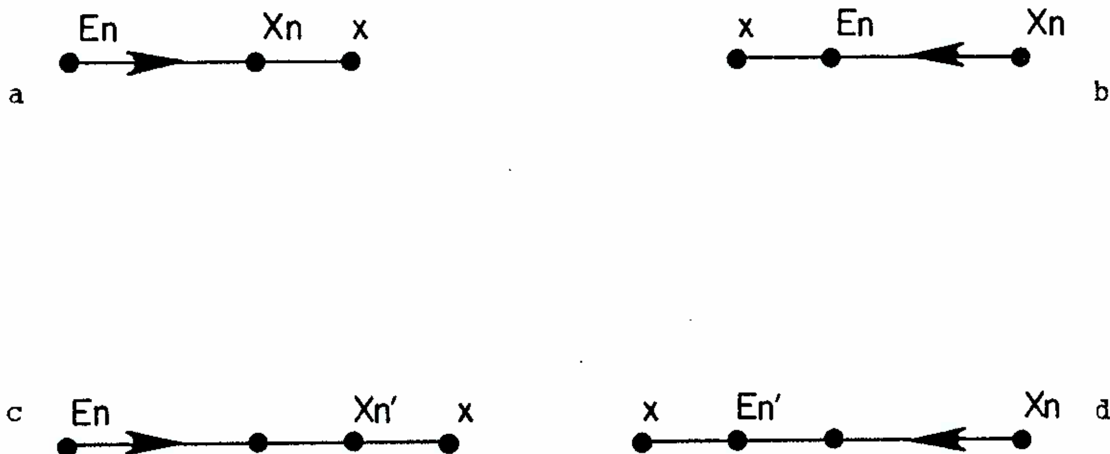


Figure 6. Horizontal adjacent point detection using adjacent run completion and extension recognizers. a) CPARHT, b) CPALFT, c) EXARHT, d) EXALFT.

2.1.7 HAIR ENDPOINT DETECTION

Hairs are single pixel width boundary regions as in Figure 7 Hairs occuring vertically or horizontally may be recognized. Points on vertical hairs are labeled with the unary run semantic label CMPUNY, while points on horizontal runs are labeled with the included point (INCDPT) semantic label. Horizontal hairs pointing to the right (left) have the endpoint labeled with IMPRHH (IMPLFH). Vertical downward pointing hairs are labeled with HAIREP while those pointing upward are labeled CMPUNY during the cleanup pass of the RLM algorithm.

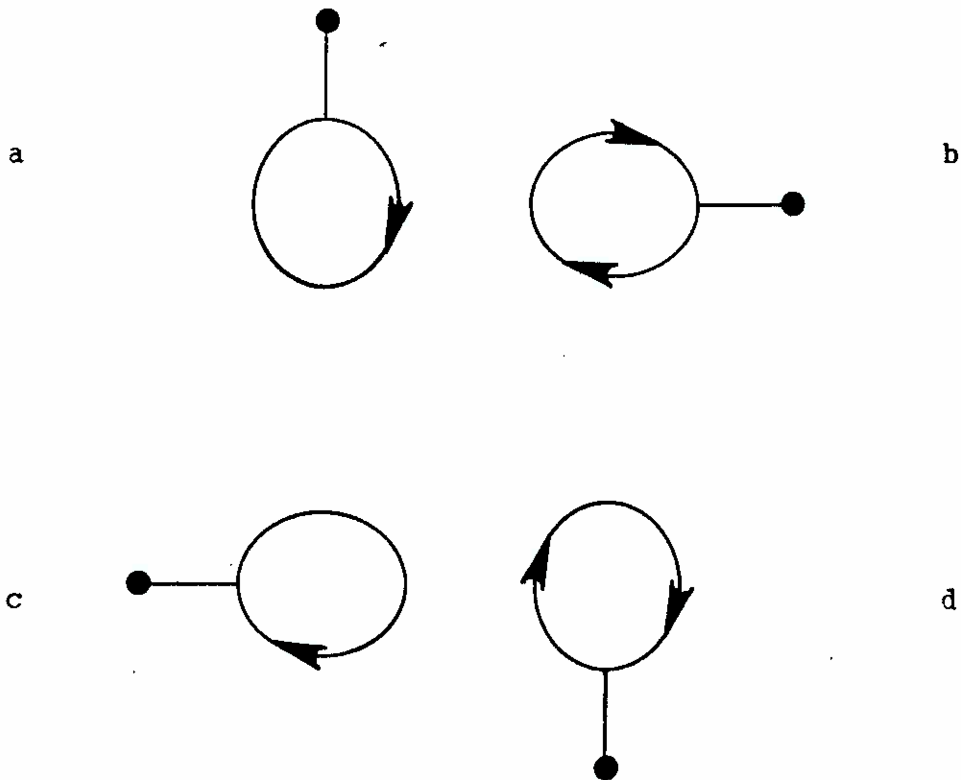


Figure 7. Hair endpoint detection using hair recognizers. a) Unary hair. The endpoint is completed during the cleanup pass and points on the hair are labeled CMPUNY. b) Right hair. The endpoint is labeled IMPRHH and points on the hair are labeled INCDPT. c) Left hair. The endpoint is labeled IMPLFH and points on the hair are labeled INCDPT. d) Downward hair. The endpoint is labeled HAIREP and points on the hair are labeled CMPUNY.

2.2 ASYMMETRY OF RUN SEMANTICS

Because the run length coding of a boundary is orientation dependent, it causes various conditions to be detected only in particular orientations. For example, a concavity oriented vertically will be detected whereas it will not be detected if it is oriented horizontally. No general solution to this problem exists (as will be seen). A partial

solution to this problem is provided by the following theorem.

D.1 Definition: The total semantic labeling of a boundary, St, is the ordered set of 2-tuples of semantic labelings (S0i,S90i) from Lsem. St is determined by taking the precedence labeling of the RLMs computed for both the original boundary Rk and for Rk rotated 90 degrees. S0i is the semantic label of the original boundary (at 0 degrees) while S90i is the labeling of that point after it is rotated 90 degrees.

The 90 degree rotation in the LCS space from (x,y) to (x',y') is computed by the transformation in (12) and (13).

$$y' = 255 - x, \tag{12}$$

$$x' = y. \tag{13}$$

T.1 Theorem: Given a total semantic labeling of a boundary St, (a) concavities which are fully detected in a given orientation angle eta are fully undetected in a 90 degree rotation of that angle; (b) the size of the concavity detected is a function of the original boundary orientation and might possibly be zero in both the eta and eta+90 degree orientations.

Proof: In Figure 8, the concavity in (a.1) is fully recognized while its 90 degree transformed equivalent is totally unrecognized. As was shown previously, concavities are detected by the occurrence of either IMPLFT or IMPRHT so that (a.1) is fully recognized.

The concavity in (b.1) and (b.2) is incompletely recognized with different parts of the concavity being recognized in the 90 degree transformed image. However, cases exist such as (c) (pointed out by Rosenfeld) where concavities exist in a particular orientation which are not detected in that orientation or in its 90 degree transform.

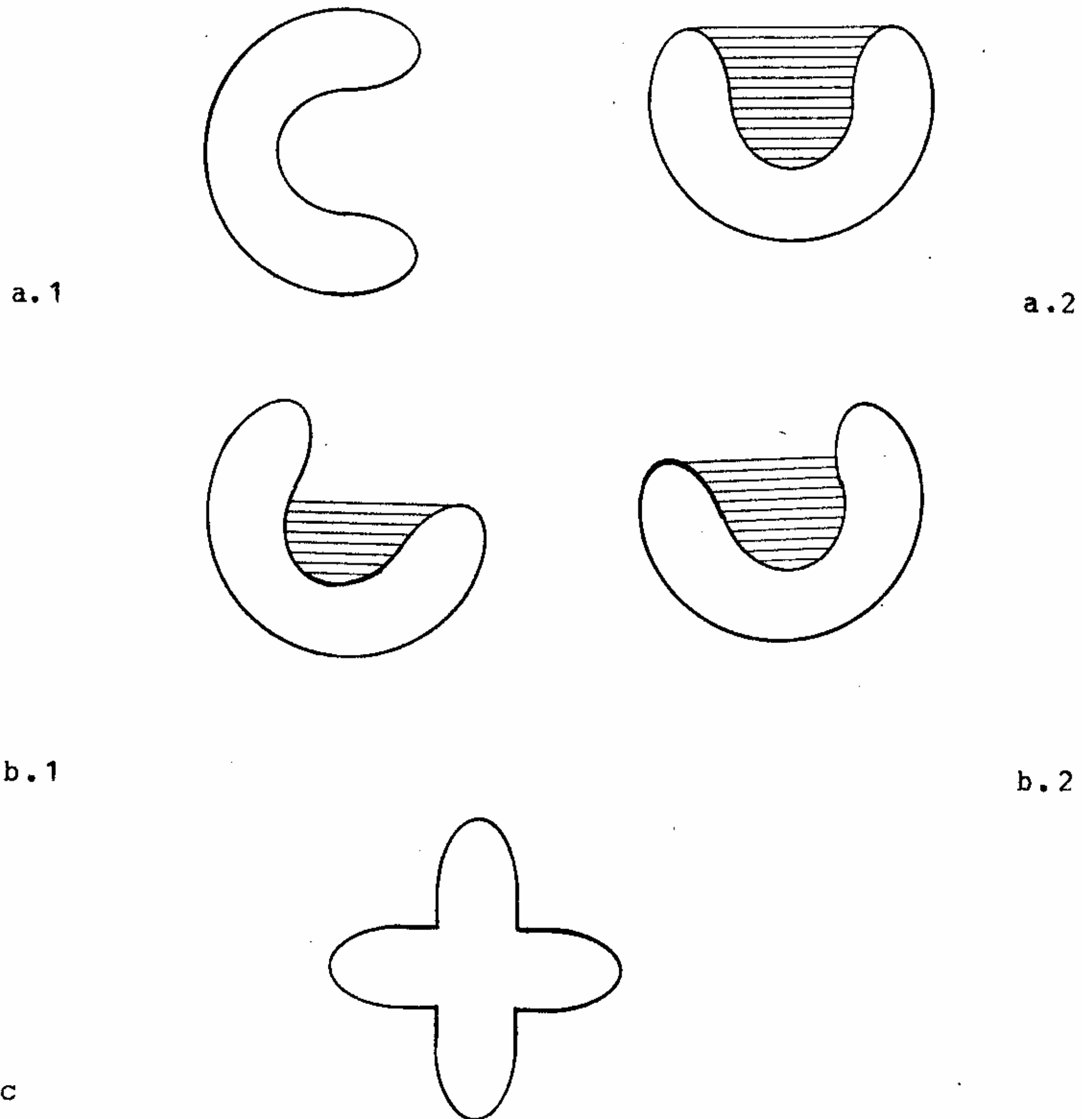


Figure 8. Cases of concavities of "C" shaped object rotated a.1) 0 degrees (no concavity), a.2) 90 degees (maximimal concavity), b.1) -45 degrees (partial concavity), b.2) 45 degrees (partial concavity). c) Example of the case of object with four undetectable concavities which are unrecognized at both 0 and 90 degree rotations.

3. RLM GENERATION ALGORITHM

The RLM generation [G] and search [S] algorithms are presented in an ALGOL-like notation. Given a TCB Rk consisting of a list of ordered (x,y) pairs as in (1), and a maximum number m of lines in the plane, the RLM generation algorithm is as follows:

```
[G.1] "Initialization"
       "Clear  last  x seen on line y. The value has different
        meanings depending on value:
               Value    Semantics
               -----    ---------
               -1       no run in progress
               x > -1   started new run
               x < -1   finished run at x, coded as -(x+1000)"
       lastx[0:m]:=-1;

       "Zero number of runs completed and started"
       nrun[0:m]:=0;

       "Zero  list  of  n  runs  for line y. exdata[y,i] is a
        record with the following fields:
               [Ei:Xi] = run values,
               [Espliti:Xspliti] = split flags,
               [Ediri:Xdiri] = direction flags,
               [Elastxi:Xlastxi] = last x flags."
       exdata[1:n,0:m]:=nil;

       "Clear the previous point"
       oldx:=-1, oldy:=-1;

       "Set minimum enclosing rectangle to a minimum"
       kx1:=ky1:=m+1;
       kx2:=ky2:=-1;


[G.2]  "Process  Rk one  (x,y)  pair at a time. Note that DONE(s)
        labels TCB[i] with the semantic label s and  skips  to
        the end of the 'process  (x,y)i' block."
       For i:=1 Step 1 Until b Do
               Begin "process  (x,y)i"
               x:=NEXTX(TCB[i]);
               y:=NEXTY(TCB[i]);
```

```
"Test for and ignore duplicate points"
If (x=oldx) and (y=oldy)
        Then DONE("DUP-PT");

"Compute minimum enclosing rectangle"
kx1:=Min(kx1,x);
ky1:=Min(ky1,y);
kx2:=Max(kx2,x);
ky2:=Max(ky2,y);

"Apply all of the non-deterministic recognizers
 (NDRs) which will either return failure or
 evaluate a DONE(.) halting all of the NDRs."
{MERGE, IMPLICIT-SPLIT, EXPLICIT-SPLIT
 EXTEND-RUN, COMPLETE-RUN};
NEW-RUN;

"Point was not recognized by any acceptor!"
DONE("NULPNT");
End "process (x,y)i";
```

```
[G.3] "Complete unfinished runs or split runs as unary runs"
      For y:=ky1 step 1 until ky2 Do
              Begin "test if finished line y"
              If (x:=lastx[y]) > 0
                 Then Begin "complete unfinished run as unary"
                      COMPLETE-RUN;
                      lastx[y]:=-1;
                      End "complete unfinished run as unary";
              End "test if finished line y";
```

3.1 RLM SEARCH ALGORITHM

Given a point (x,y), the question of whether it is inside the boundary of Rk is answered by the search algorithm.

```
[S.1] "Get the number of runs for line y."
              n:=nrun[y];

[s.2] "Search runs on line y for x in any run";
      For i:=1 step 1 until n Do If (x in [Ei:Xi]y,i)
                                      Then Return(True);

[s.3] "Search failed."
      Return(False);
```

The RLM may be used in various ways. As was pointed out by Merrill measures such as perimeter and area may be easily computed. Note that the RLM generation algorithm also computes the minimum enclosing rectangle for Rk. In this scheme, perimeter is computed by (14) and area by (15).

```
perimeter = 2(Sum nruns[ y ]) .                              (14)
             y:=ky1:ky2

area = Sum         (Sum (Xi-Ei+1) y,i                        (15)
       y:=ky1:ky2     i:=0:nruns[ y ]
```

The RLM may be used to drive a process over only those points on the plane inside of the curve as in (16).

```
For y:=ky1 Step 1 Until ky2 Do                               (16)
   For i:=0 Step 1 Until nruns[ y ] Do
      For x:=Ei,y Step 1 Until Xi,y Do
                                         Begin
                                         "process";
                                         End;
```

Thus one observes that the process of accessing the RLM is at worst of Order(Nmax) with Nmax being the maximum number of runs in any row (i.e. the maximum of nrun[y]). Furthermore, it will later be shown that (Nmax leq Order(b)).

4. FINITE STATE MACHINE STATE RECOGNIZERS

All of the FSMs are run in parallel for each (x,y) pair of a clockwise traversed boundary. If a FSM succeeds, this is signaled by the DONE predicate which broadcasts this fact and causes other FSMs in progress to abort. Each FSM described in the following text has a figure associated with it showing the specific cases recognized by the FSM.

The following notation is used in describing the NDRs.

[Ei:Xi]y,i	= ith run of line y with entrance x coordinate Ei and exit x coordinate Xi.
MARKRUN(y,n)	= marks the associated semantic marker with the specified run.
NEWRUN(x,y)	= If lastx[y]=-1 then create a new run for the current line y with the value x in lastx[y]. As n=nrun[y], the new run will have run number n+1. It is implemented by running the NEW-RUN FSM but ignoring the DONE(.).
STORERUN[e,x]y,k	= store the run [e:x] in [Ek:Xk]y,k if it exists, otherwise create a new run.
GETRUN[e,x]y,k	= look up the run [Ek:Xk]y,k. If the return exists then return true else return false and the run values [2:1] (an impossible run).
DONE(NDR name)	= label boundary point with the name of the NDR which accepted it and set oldx:=x, oldy:=y and skip to the END "process (x,y)" block.
SPLITP(q)	= predicate which returns true or false depending on whether q is marked split or not. q is either Ei or Xi.

LASTXP(q) = predicate which returns true or false depending on whether q is the last x changed in a run or not. q is either Ei or Xi.

DIR(q) = function which returns "up" or "down" depending on whether q is marked down or up. q is either Ei, Xi or lastx[y]. Note: DIR(x) is equivalent to

```
If last y < y
    Then "down"
    Else "up".
```

LAST(q) = function which returns previous value of q, where q is x or y.

LOOKAHEAD(q) = value of next q value. q is either x or y.

CW = locally clockwise direction.

CCW = locally counter-clockwise direction.

UP = last y > y.

DOWN = last y < y.

RIGHT = lastx[y] < x.

LEFT = lastx[y] > x.

4.1 MERGE CASE NDR

The current point (x,y) is said to cause a merge run if there exists a previously marked split run [Ei:splitflag] or [splitflag:Xi] and the following conditions occur. The conditions for this NDR (and the others which follow) are given in an ALGOL-like notation.

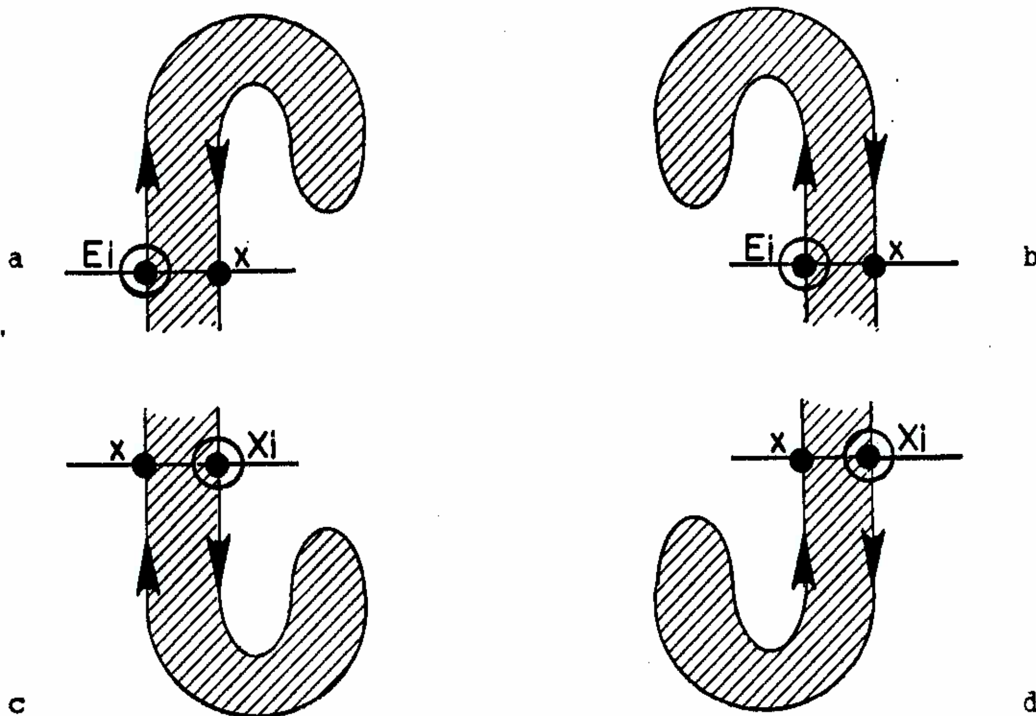


Figure 9. Merge cases fcr splits produced by both implicit and explicit splits. Right merge MRGRHT (Ei is a split) applies to both a) CW (explicit split) and b) CCW (implicit split) spirals. Left merge MRGLFT (Xi is a split) applies to both c) CW (implicit split) and d) CCW (explicit split) spiral.

Figure 9 shows the two cases which can cause a merge Note that the left and right splits are the same for both CCW and CW produced splits. The n runs (nrun[y]) on line y are searched backwards as runs occurring later in the run list are more likely to be candidates for merger (as a result of recent splits) than earlier runs.

```
If not((nrun[y] > 0) and (LAST(y) neq y))
   Then Return; "Don't try to merge."

For i:=nrun[y] Step -1 Until 1 Do
   If not (x in [Ei:Xi]y,i)
     Then Begin "test merges"
```

4.1.1 RIGHT MERGE: MRGRHT

```
If (GETRUN[Ei:Xi]y,i and SPLITP(Xi))
        and DIR(Ei)="up" and (LOOKAHEAD(y) leq y)
        and (LAST(y) < y) and (x geq Ei)
        Then Begin "right merge"
             If (x-1=Ei) and (LOOKAHEAD(Y) leq y)
                Then STORERUN([Ei:x+splitflag+lastxflag]y,i)
                Else STORERUN([Ei:x+lastxflag]y,i);
             MARKRUN(y,i):="outside right merge";
             lastx[y]:=-1;
             DONE("MRGRHT");
             End "right merge"
             Else
```

4.1.2 LEFT MERGE: MRGLFT

```
If SPLITP(Ei)) and DIR(Xi)="down" and (LOOKAHEAD(y) geq y)
        and (LAST(y) > y) and (x leq Xi)
        Then Begin "left merge"
             If (x+1=Xi) and (LOOKAHEAD(Y) geq y)
                Then STORERUN([x+lastxflag+splitflag:Xi]y,i)
                Else STORERUN([x+lastxflag:Xi]y,i);
             MARKRUN(y,i):="outside left merge";
             lastx[y]:=-1;
             DONE("MRGLFT");
             End "left merge";
        End "test merges";
```

4.2 IMPLICIT-SPLIT CASE NDR

The current point (x,y) is said to cause an implicit split when the following conditions are true. Figure 10 illustrates the left and right implicit splits. An implicit split marks the lastx[y] point as an implicit split if a run from lastx[y] to x is not valid. The implicit split handles the case where the first point of a line y will later form a merge. The cases in Figures 10.a and 10.c are those where the lastx[y] was labeled NEWRUN, while Figures 10.b and 10.d show the cases where a short line segment caused by lastx[y] was labeled a run.

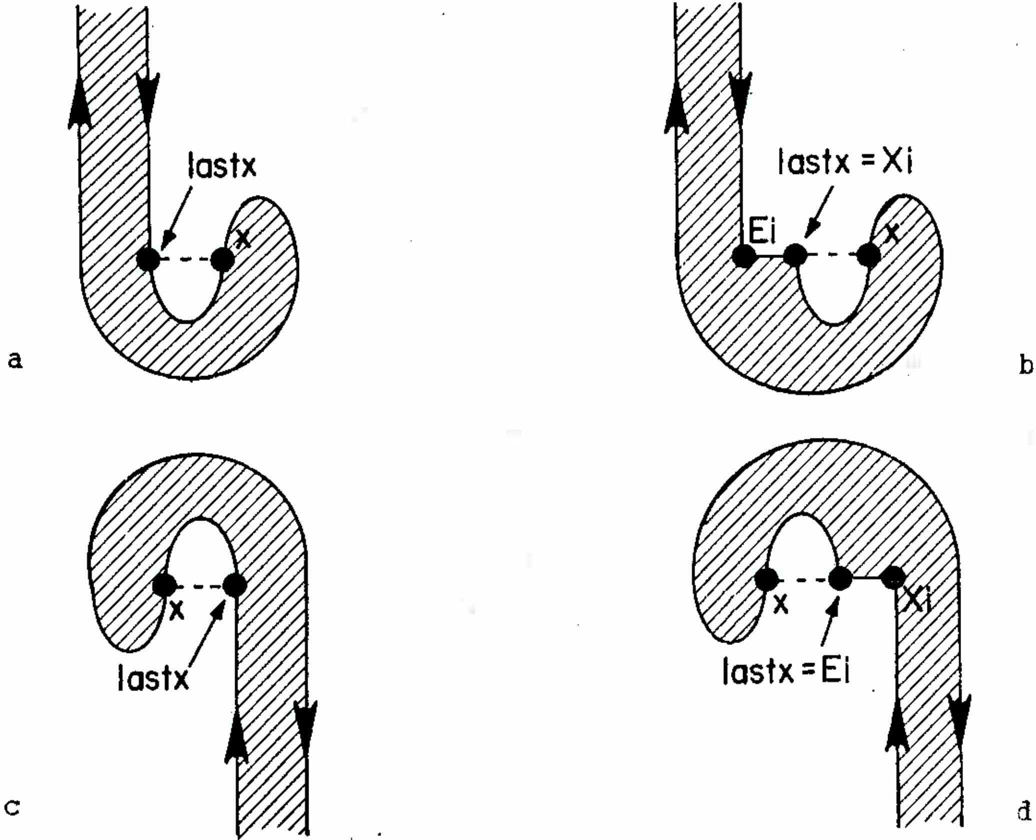


Figure 10. Implicit split cases. a) and b) Implicit left CCW split IMPIFT, c) and d) Implicit right CCW split IMPRHT.

```
If ((n:=nrun[y]) = 0) or ((lx:=lastx[y]) = -1)
        Then Return "Don't test implicit splits."
        Else Begin "test for implicit split"
             If (lx < -1)
                Then lx:=-(lx+1000);
             If (lx = x)
                Then Return; "ignore unary run"
```

4.2.1 IMPLICIT LEFT CCW SPLIT: IMPLFT

```
If (lx < x) and DIR(x)="up" and DIR(lx)="down"
  Then ndrname:="IMPLFT", GOTO [found-imp-split]
   Else
```

4.2.2 IMPLICIT RIGHT CCW SPLIT: IMPRHT

```
If (lx > x) and DIR(x)="down" and DIR(lx)="up"
   Then ndrname:="IMPRHT", Goto [found-imp-split]
   Else Return;

[found-imp-split]:
        Begin "found-imp-split"
        svlastx:=lastx[y];
        Ei:=Xi:=lx;
        If (n = 1)
                Then If (lastx[y] > -1)
                        Then Goto [save-run] "save unary run"
                        Else Begin "get and test first run"
                             GETRUN[E1:X1]y,1;
                             If DIR(E1) = DIR(X1)
                                Then Goto [save-run]
                                Else Goto [new-run];
                             End "get and test first run";
        If (n > 1)
                Then Begin "look for split of opposite dir."
                     GETRUN[En-1:Xn-1]y,n-1;
                     If (SPLITP(En-1) and (ndrname="IMPRHT")) or
                        (SPLITP(Xn-1) and (ndrname="IMPLFT"))
                        Then If (lastx[y] < -1)
                                Then GETRUN[En:Xn]y,n,
                                     Goto [save-run]
                                Else En:=Xn:=lastx[y],
                                     Goto [save-run];
                     If lastx[y] < -1
                        Then Begin "finish 'NEWRUN' as a SPLIT"
                        Comment: a split run must be created
                        since the implied split now being
                        created would generate two or more
                        NEWRUNs otherwise;
                             n:=nrun[y];
                             En:=Xn:=lx;
                             Goto [save-run];
                             End "finish 'NEWRUN' as a SPLIT";
```

```
                    Goto [new-run];
                    End "look for split of opposite dir.";
        End "found-imp-split"

[save-run]:
        Begin "save-run"
        n:=nrun[y];
        If (ndrname="IMPLFT")
                Then Begin "set left run"
                    DIR(Ei):="down"; DIR(lx):="up";
                    STORERUN([Ei+splitflag:lx+lastxflag]y,n;
                    MARKRUN(y,n):="implicit left run";
                    End "set left run";
        If (ndrname="IMPRHT")
                Then Begin "set right run"
                    DIR(Ei):="up"; DIR(lx):="down";
                    STORERUN([lx+lastxflag:Xi+splitflag]y,n;
                    MARKRUN(y,n):="implicit right run";
                    End "set right run";
        End "save-run";
```

4.2.3 IMPLICIT LEFT AND RIGHT HAIRS: IMPLFH IMPRHH

```
[new run]:
        Begin "new run"
        lastx[y]:=-1;
        "Enter it as a new run. It may reappear as HAIREP!"
        NEWRUN(x);
        If lastx[y] = -1
           Then Begin "implied hair"
                lastx[y]:=svlastx;
                If ndrname = "IMPLFT"
                        Then DONE("IMPLFH")
                        Else DONE(IMPRHH");
                End "implied hair";

        "Not a hair, return the previous decision."
        DONE(ndrname);
        End "new run";
        End "test for implicit split";
```

4.3 EXPLICIT-SPLIT CASE NDR

The current point (x,y) is said to cause an explicit split of a run [En:Xn]y,n when the following conditions are true. Figure 11 shows the four cases. The explicit split takes place when x is inside an existing run and enters the run region in such a way that the existing run is required to be broken into a smaller sub-run and an extremum point. The old extremum point is marked as "split" so that it may be merged into another run later on.

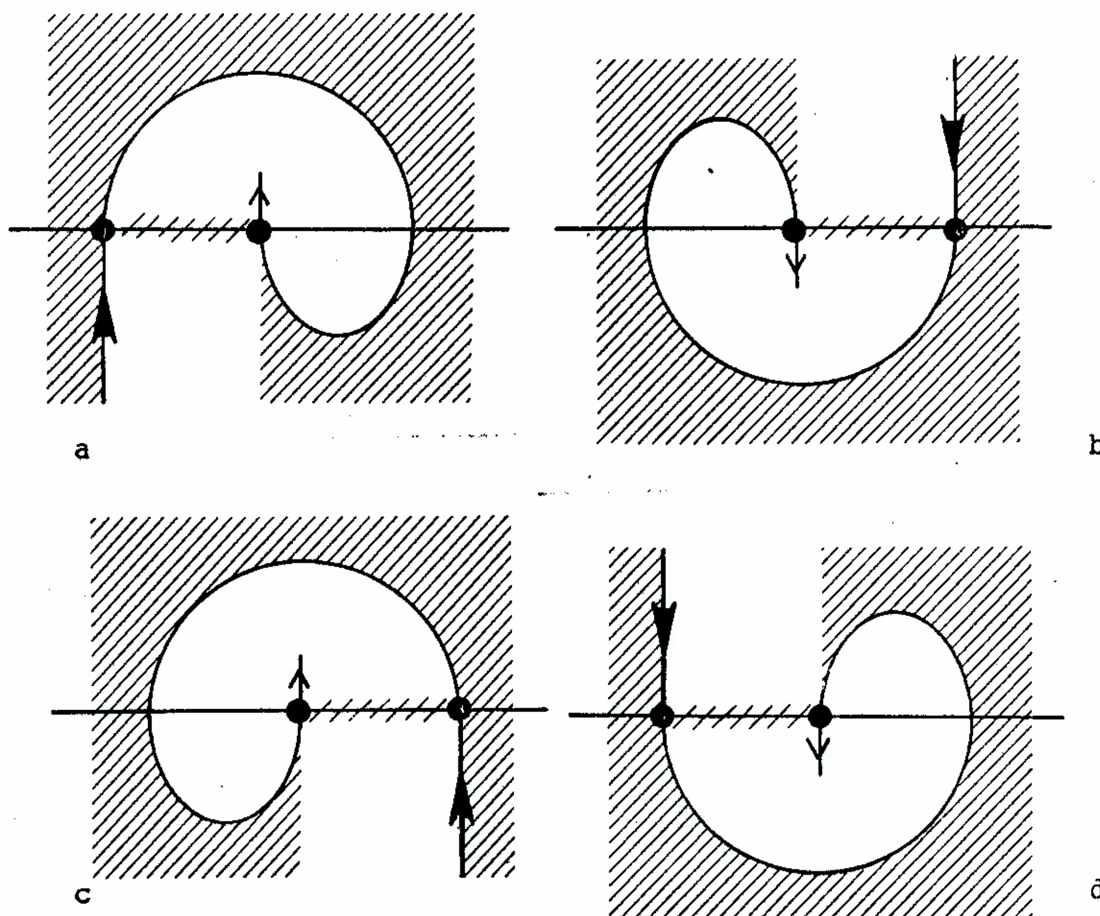


Figure 11. Explicit split cases. a) Explicit left CW split SPCW-L, b) explicit right CW split SPCW-R, c) explicit right CCW split SPCCWR, d) explicit CCW split SPCCWL.

There are cases where the current point might be an explicit split. Lookahead at the next point is required to resolve this. Figure 12 shows the resolution of the ambiguous explicit split cases.

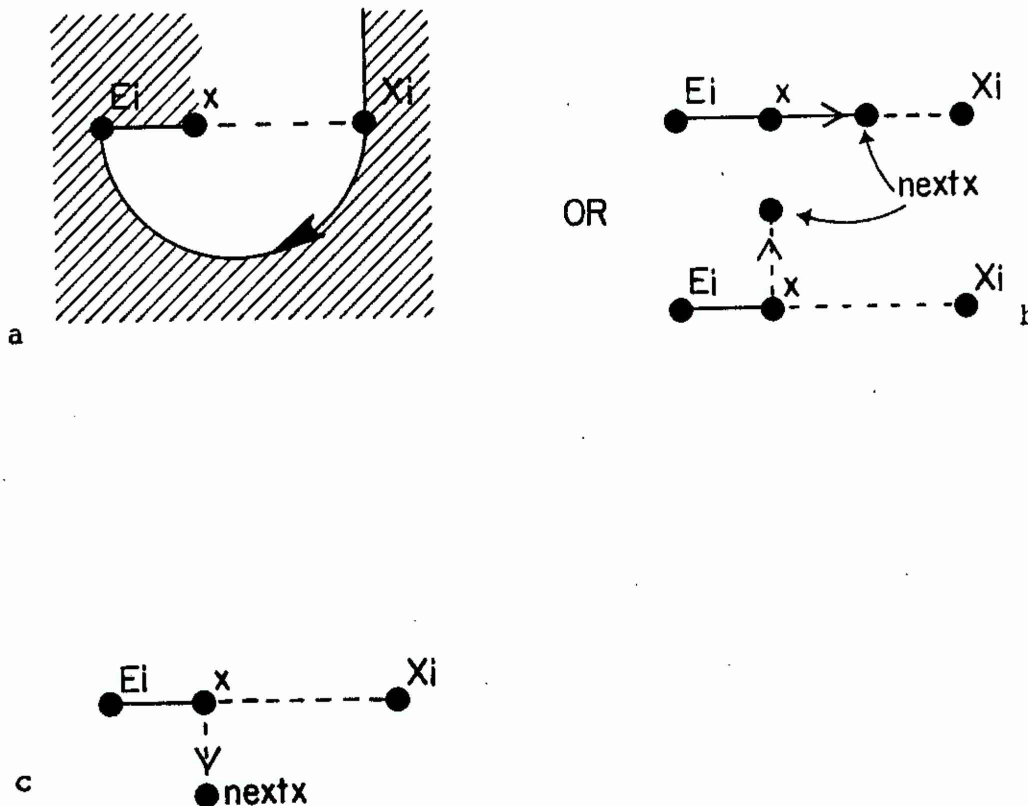


Figure 12. Resolution of the ambiguous states of the explicit split cases. a) Ambiguous state at x in explicit split. b) Two possible cases of (a) which are included points (INCDPT) and are ignored. c) Another case of (a) which is a split. The other three ambiguous cases (orientations) are resolved similarly.

```
If (n:=nrun[y]) = 0
   Then Return; "Don't test for explicit splits."

For i:=1 Step 1 Until n Do
   If (GETRUN[Ei:Xi]y,i) and (x in [Ei:Xi]y,i) and (Ei leq Xi)
     Then Begin "Explicit split test"
```

4.3.1 TEST FOR INCLUDED POINT

```
If ((LAST(y) = y = LOOKAHEAD(y)) or (Ei = Xi))
   Then Goto [included-point]
   Else If not (SPLITP(Xi) or SPLITP(Ei))
                 Then
```

4.3.2 EXPLICIT CW LEFT SPLIT: SPCW-L

```
If (LAST(y) geq y > LOOKAHEAD(y))
        and LASTXP(Xi) and DIR(Xi) = "down"
        Then Begin "CW left split"
             STORERUN([Ei+lastxflag:Ei+splitflag]y,i);
             MARKRUN(y,i):="outside left split CW spiral";
             DIR(x):="up"; lastx[y]:=-1; NEWRUN(x,y);
             STORERUN([x+lastxflag:Xi]y,n+1);
             MARKRUN(y,n+1):="inside left split CW spiral";
             DONE("SPCW-L");
             End "CW left split"
        Else
```

4.3.3 EXPLICIT CW RIGHT SPLIT: SPCW-R

```
If (LAST(y) leq y < LOOKAHEAD(y))
        and LASTXP(Ei) and DIR(Ei) = "up"
        Then Begin "CW right split"
             STORERUN([Xi+splitflag:Xi+lastxflag]y,i);
             MARKRUN(y,i):="outside right split CW spiral";
             DIR(x):="down"; lastx[y]:=-1; NEWRUN(Ei,n+1);
             STORERUN([Ei:x+lastxflag]y,n+1);
             MARKRUN(y,n+1):="inside right split CW spiral";
             DONE("SPCW-R");
             End "CW right split"
        Else
```

4.3.4 EXPLICIT CCW LEFT SPLIT: SPCCWR

```
If (LAST(y) leq y < LOOKAHEAD(y))
        and LASTXP(Xi) and DIR(Xi) = "up"
        Then Begin "CCW left split"
             STORERUN([Ei+splitflag:Ei+lastxflag]y,i);
             MARKRUN(y,i):="outside left split CCW spiral";
             DIR(x):="down"; lastx[y]:=-1; NEWRUN(x,y);
             STORERUN([x+lastxflag:Xi]y,n+1);
             MARKRUN(y,n+1):="inside left split CCW spiral";
             DONE("SPCCWR");
             End "CCW left split"
        Else
```

4.3.5 EXPLICIT CCW RIGHT SPLIT: SPCCWL

```
If (LAST(y) geq y > LOOKAHEAD(y))
        and LASTXP(Ei) and DIR(Ei) = "down"
        Then Begin "right split"
             STORERUN([Xi+lastxflag:Xi+splitflag]y,i);
```

```
          MARKRUN(y,i):="outside right split CCW spiral";
          DIR(x):="up"; lastx[y]:=-1; NEWRUN(Ei,n+1);
          STORERUN([Ei:x+lastxflag]y,n+1);
          MARKRUN(y,n+1):="inside right split CCW spiral";
          DONE("SPCCWL");
          End "right split"
     Else
```

4.3.6 DO INCLUDED ADJACENT POINT: CGRHSP CGLHSP INCDPT

```
[included-point]:
        Begin "included adjacent point"
        If (Ei = Xi)
           Then Begin "Change 'HAIREP' to split"
                lastx[y]:=x;
                DIR(lastx[y]):=DIR(x);
                If DIR(x)="up"
                   Then Begin "change right"
                        STORERUN([x+splitflag:x+lastxflag]y,i);
                        DONE("CGRHSP");
                        End "change right"
                   Else Begin "change left"
                        STORERUN([x+lastxflag:x+splitflag]y,i);
                        DONE("CGLHSP");
                        End "change left";
                End "Change 'HAIREP' to split"
           Else DONE("INCDPT");
        End "included adjacent point";
      End "Explicit split test";
```

4.4 EXTEND-RUN CASE NDR

The current point (x,y) is said to cause a run extension of the last run [En:Xn]y,n of line y when the following conditions hold. Figure 13 shows the non-adjacent point run extension cases and Figure 14 shows the adjacent point run extension cases.

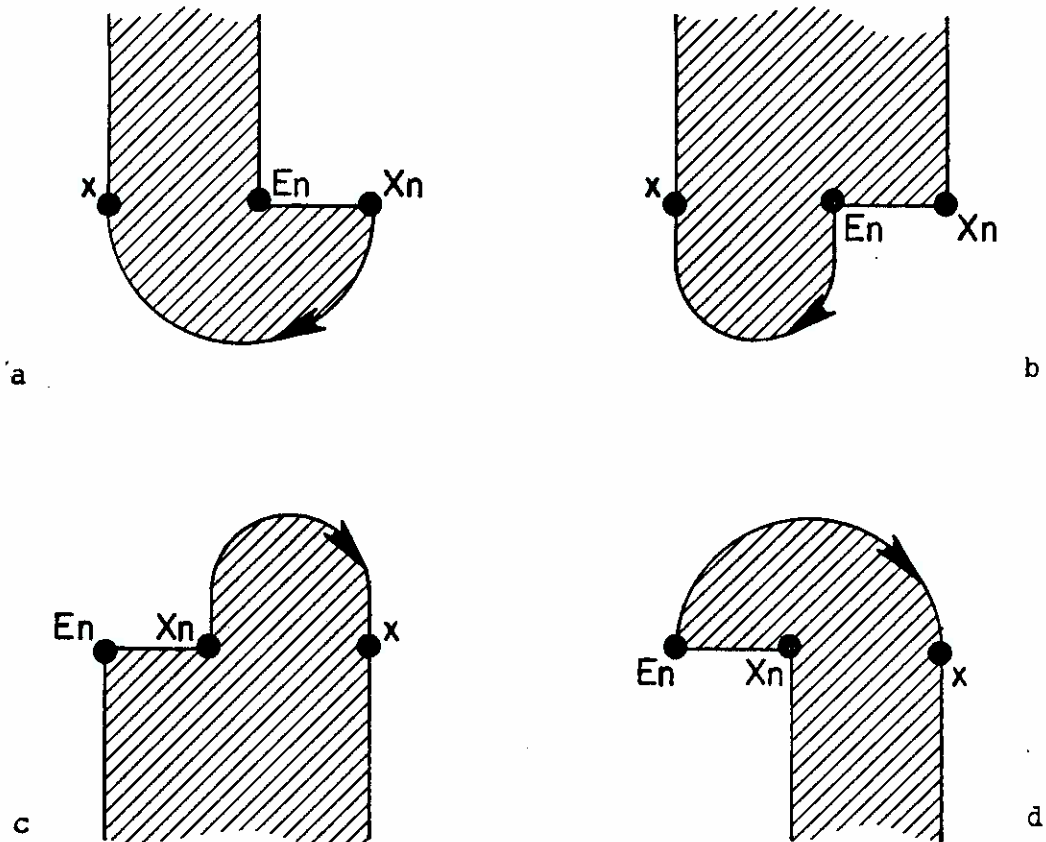


Figure 13. Non-adjacent run extension cases. a) and b) Left non-adjacent run extensions EXNLFT. c) and d) Right non-adjacent run extensions EXNRHT.

Figure 14. Adjacent run extension cases. a) Adjacent right run extension EXARHT. b) Adjacent left run extension EXALFT.

```
GETRUN[En:Xn]y,n;
If ((n:=nrun[y])=0) or (En > Xn)
   Then Return; "Don't test run extensions"
```

4.4.1 LEFT ADJACENT RUN EXTENSION: EXALFT

```
If (En-1=x)
   Then Begin "left adjacent run extension"
        DIR(x):=DIR(En); STORERUN([x+lastxflag:Xn]y,n);
        lastx[y]:=-(x+1000);
        MARKRUN(y,n):="left adjacent run extension";
        DONE("EXALFT");
        End "left adjacent run extension"
        Else
```

4.4.2 RIGHT ADJACENT RUN EXTENSION: EXARHT

```
If (Xn+1=x)
   Then Begin "right adjacent run extension"
        DIR(x):=DIR(Xn); STORERUN([En:x+lastxflag]y,n);
        lastx[y]:=-(x+1000);
        MARKRUN(y,n):="right adjacent run extension";
        DONE("EXARHT");
        End "right adjacent run extension"
        Else If not (SPLITP(En) or SPLITP(Xn))
                Then
```

4.4.3 LEFT NON-ADJACENT RUN EXTENSION: EXNLFT

```
If (LAST(y) > y) and (lastx[y] < -1) and (x < En)
  Then Begin "left non-adjacent run extension"
        DIR(x):=DIR(En); STORERUN([x+lastxflag:Xn]y,n);
        lastx[y]:=-(x+1000);
```

```
        MARKRUN(y,n):="left non-adjacent run extension";
        DONE("EXNLFT");
        End "left non-adjacent run extension"
        Else
```

4.4.4 RIGHT NON-ADJACENT RUN EXTENSION: EXNRHT

```
If (LAST(y) < y) and (lastx[y] < -1) and (x > Xn)
   Then Begin "right non-adjacent run extension"
        DIR(x):=DIR(Xn); STORERUN([En:x+lastxflag]y,n);
        lastx[y]:=-(x+1000);
        MARKRUN(y,n):="right non-adjacent run extension";
        DONE("EXNRHT");
        End "right non-adjacent run extension";
```

4.5 RUN COMPLETION NDR

The current point (x,y) is said to cause a run completion for line y when the following conditions are true. There are three types: adjacent run, unary run, and non-adjacent run completions. The algorithm tests to see where a run can be legally constructed from lastx[y] and x. Figure 15 shows the three major types of run completions with 15a-b being adjacent runs, 15c the unary run case, and 15d-e the non-adjacent run completion cases.

```
If (n:=nrun[y] = 0) or (lastx[y] < 0)
        Then Return; "Do not test run completion."

"Save old last x direction"
oldlxdir:=DIR(lastx[y]);
```

4.5.1 RIGHT NON-AJD. OR ADJ. RUN COMPLETION: CPARHT CPNRHT

```
If (lastx[y] < x)
  Then If (LAST(y) < y)
        Then Begin "right non-adj run completion"
             DIR(x):="down"; DIR(lastx[y]):="up";
             STORERUN([lastx[y]:x+lastxflag]y,n);
             lastx[y]:=-(x+1000); DIR(lastx[y]):="down";
             DONE("CPNRHT");
             End "right non-adj run completion"
        Else If (LAST(y) = y)
                Then Begin "right adj run completion"
                     DIR(x):=DIR(lastx[y]);
                     STORERUN([lastx[y]:x+lastxflag]y,n);
                     lastx[y]:=-(x+1000);
                     DIR(lastx[y]):=oldlxdir;
                     DONE("CPNRHT");
                     End "right adj run completion"
   Else
```

4.5.2 UNARY RUN COMPLETION: CMPUNY

```
If (lastx[y] = x)
   Then Begin "unary completion"
        En:=Xn:=x;
        If LOOKAHEAD(y) < y
```

```
                Then DIR(En):="up", DIR(Xn):="down"
                Else DIR(En):="down", DIR(Xn):="up";
        STORERUN([En:Xn]y,n);
        lastx[y]:=-(x+1000); DIR(lastx[y]):=oldlxdir;
        DONE("CMPUNY");
        End "unary completion";
    Else
```

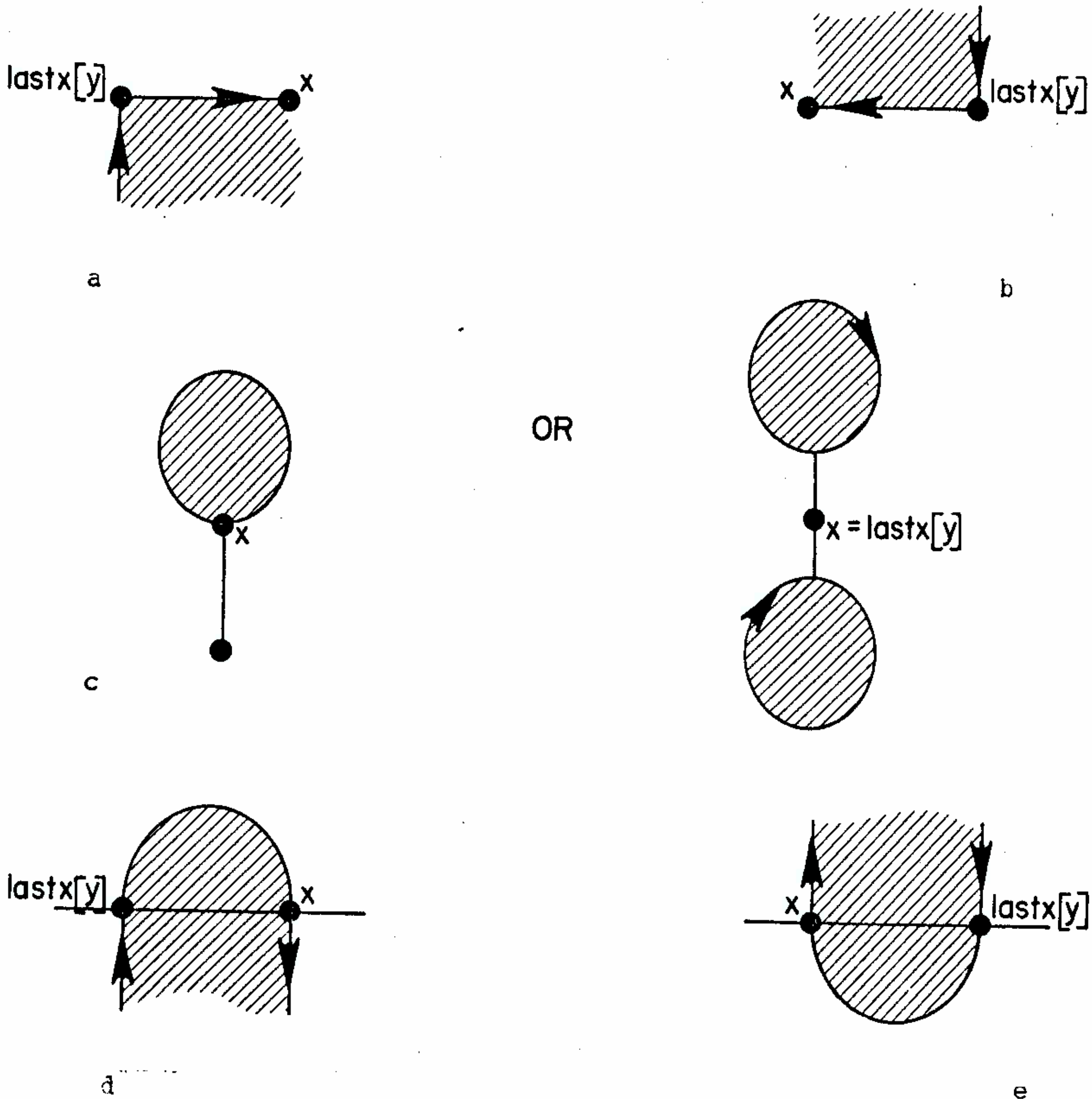


Figure 15. Run completion cases. a) Adjacent right run completion CPARHT, b) adjacent left run completion CPALFT, c) unary run completion CMPUNY, d) non-adjacent right run completion CPNRHT, e) non-adjacent left run completion CPNLFT.

4.5.3 LEFT NON-AJC. OR ADJ. RUN COMPLETION: CPALFT CPNLFT

```
If (lastx[y] < x)
  Then If (LAST(y) > y)
        Then Begin "left non-adj run completion"
             DIR(x):="up"; DIR(lastx[y]):="down";
             STORERUN([x+lastxflag:lastx[y]]y,n);
             lastx[y]:=-(x+1000); DIR(lastx[y]):="up";
             DONE("CPALFT");
             End "left non-adj run completion"
        Else If (LAST(y) = y)
                Then Begin "left adj run completion"
                     DIR(x):=DIR(lastx[y]);
                     STORERUN([x+lastxflag:lastx[y]]y,n);
                     lastx[y]:=-(x+1000);
                     DIR(lastx[y]):=oldlxdir;
                     DONE("CPNLFT");
                     End "left adj run completion";
```

4.6 NEW-RUN CASE NDR

The current point (x,y) is said to cause a new run n+1 for line y which currently has n runs when the following conditions are true. The new run is simply a marking of the start of a potential run. It sets lastx[y] so that it may be tested on the next point by other NDRs. Figure 16a shows the new run case. The hair endpoint case shown in Figure 16b is found by recognizing that the point backups on the next point.

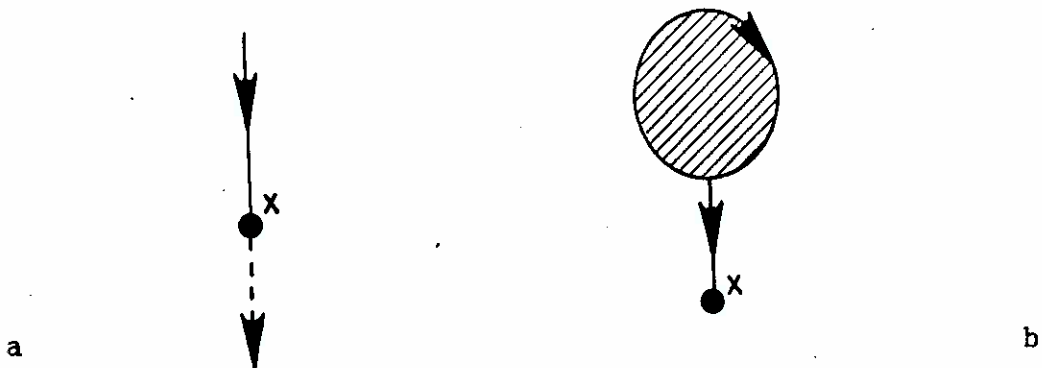


Figure 16. New run cases. a) Normal new run NEWRUN, b) new run which is a hair end point HAIREP determined by the condition (lasty = lookahead[y] neq y).

```
If (lastx[y] = -1)
   Then Begin "new run"
        lastx[y]:=x; (n:=nrun[y]:=nrun[y]+1);
        DIR(lastx[y]):=DIR(x);
```

4.6.1 HAIR END POINT: HAIREP

```
        "Check for hair point"
        If (LAST(y) = LOOKAHEAD(y) neq y)
           Then Begin "hair point"
                DIR(lastx[y]):=(If LOOKAHEAD(y) < y
                                     Then "up
                                     Else "down");
                STORERUN([lastx[y]:lastx[y]]y,n);
```

```
                DONE("HAIREP");
                End "hair point"
4.6.2 NEW RUN: NEWRUN
        Else DONE("NEWRUN");
        End "new run";
```

5. CHARACTERISTICS OF THE ALGORITHM

Various characteristics of the RLM algorithm are discussed including its completeness and the complexity of the RLM generation and search algorithms.

5.1 COMPLETENESS OF THE RLM ALGORITHM

T.2 Theorem: For each point in Rk, there exists a semantic label from Lsem such that the corresponding NDR accepted it.

Proof: The proof is not presented here. It can be shown by listing all cases of boundary transitions and then showing that each case is handled by a NDR.

5.2 COMPLEXITY MEASURE OF RLM GENERATION

A complexity measure may be computed to determine the worst case computation for generating a RLM from a TCB Rk of length b. The result is comparable to other methods in computational effort required to represent the TCB in a easily searched structure.

T.3 Theorem: The computation of the RLM of a TCB Rk of length b is Order(b).

Proof: The RLM at each step is computed with a finite state transition function. At worst the function must look up a previous run on a given line y. If the runs are stored in an infinite array "exdata[n,y]" then this is a single step. If it must search a linked list (as is the case in the current implementation) then the computation is Order(b*E[r]) where E[r] is the expected number of runs/line. If, for a given y, [Ei:Xi]y,i is searched using a binary search, then the number of computations becomes Order(b*log(E[r])).

5.3 COMPLEXITY MEASURE OF SEARCHING THE RLM

A complexity measure may be computed to determine worst case computations for searching a RLM to determine whether a point (x,y) is inside the TCB represented by the RLM.

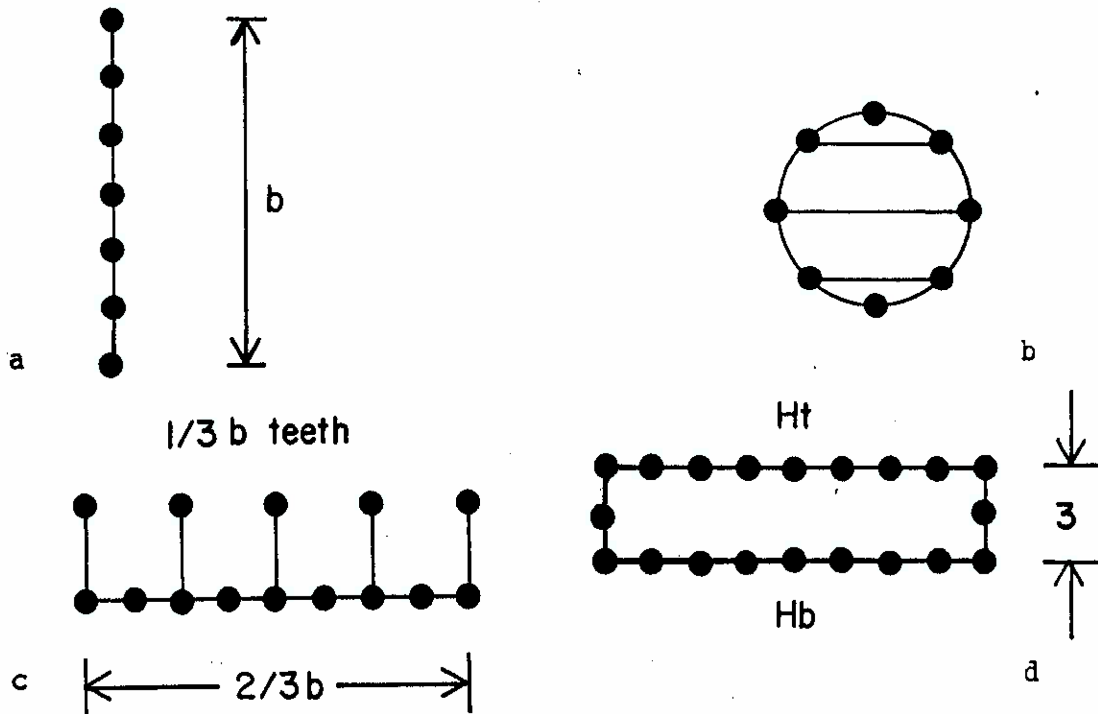


Figure 17. Extreme cases of run complexity. a) Vertical line of b points with b runs, b) simple convex object of length b points and (b-2)/2 runs, c) a horizontal comb of boundary length b with vertical teeth of length 1 having (2/3)b points on the line and (1/3)b teeth and (1/3)b runs, d) rectangular boundary with minor width of 3 and major length (b-2)/2 having 3 runs for any length b.

T.4 Theorem: Given TCB Rk of length b points, the number of runs N required to represent Rk is bounded by [3 leq N leq b]. For a simple convex object, it is bounded by [3 leq N leq ((b/2)+1)]. The worst case object with respect to the maximum number of runs/line is a comb of length (2/3)b and (1/3)b teeth which has (1/3)b unary

runs. Thus the expected number of runs E[r] leq (1/3)b.

Proof: There are two cases (a) Rk is an unclosed vertical arc and (b) Rk is a closed arc. These cases are illustrated in Figure 17.

(a) All runs are of length 1 since the entrance and exit points for any run are the same x point for a line y on Rk. Therefore for |Rk|=b, there are b runs as in Figure 17a.

(b) For a closed convex arc, there are at least two points on Rk which have runs of length 1: the top and bottom point. Thus there are (b-2)/2 runs of length greater than 1 together with the two end point runs, or a total of ((b/2)+1) runs as in Figure 17b.

The closed object with the maximum number of runs/boundary length is a comb with comb body thickness 1, tooth length 1 and tooth spacing 2 with the teeth positioned vertically (shown in Figure 17. c). Thus as we travel around the comb there are (1/3)b teeth corresponding to (1/3)b runs plus the run (extension) of the base.

The case where n points (n>2) map into a run occurs on horizontal runs (see Merrill's last restriction). Let there be two horizontal runs: the number of points in the top, Ht, and the bottom, Hb. Thus there are ((b-Ht-Hb)/2)+2 runs. Since Ht=Hb, this is ((b-2*Ht)/2)+2.

The object with the least number of runs is a 3-line rectangle as in Figure 17d. The three runs are: Ht, 1, Hb. Thus convex objects are bounded by [3 leq N leq ((b/2)+1)] runs. Arbitrary closed objects are bounded by [3 leq N leq b].

T.5 Theorem: Given a TCB of length b, the RLM uses storage leq 3b.

Proof: From Theorm T.2, the maximum number of runs required to represent the TCB Rk is b. Let each run require one unit of storage to store [Ei:Xi]y,i. Each line requires a run counter "nrun[y]" and the value of the last x seen "lastx[y]".

T.6 Theorem: Testing whether a point (x,y) is inside a TCB region requires Order(E[r]/2) leq Order(b/6) operations.

Proof: A point (x,y) has equal probability of appearing in one run as in another. Thus E[search length]=E[r]/2.

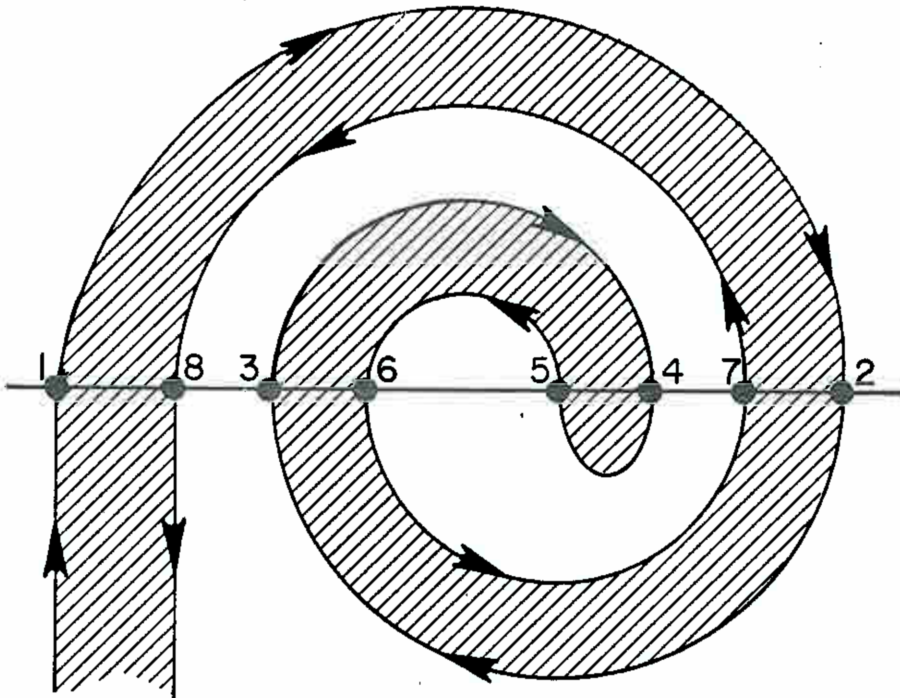
Since the maximum number of runs for a boundary of length b is b/3, we have Order(E[r])/2 leq Order(b/6). In practice, E[r] is very small (about 1 to 4).

6. EXAMPLES USING THE RLM

The following two examples illustrate the RLM algorithm. The first shows the ability to keep track of the level inside of a spiral as well as the ability to differentiate between clockwise (CW) and counterclockwise (CCW) spirals. The second example illustrates the analysis of the statistics of the semantic labeling produced by the RLM generation process.

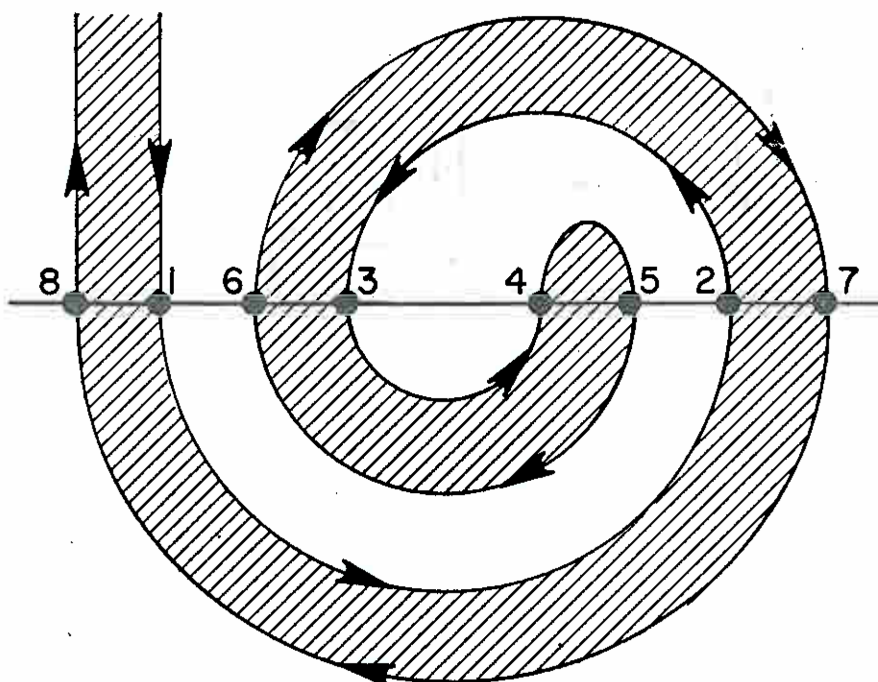
6.1 SPIRALS

The spiral is a fairly complicated case from the point of view of determining connectedness. The number of runs required to represent it grows very rapidly. It has two distinct forms: clockwise (CW) and counterclockwise (CCW). These two forms make use of most of the NDRs of the algorithm. Figure 18 shows an example of a line of semantic labels through a CW spiral. Figure 19 shows an example of a line of semantic labels through a CCW spiral. Figure 20 shows an example of a line of semantic labels through a hand drawn double spiral boundary spiral.



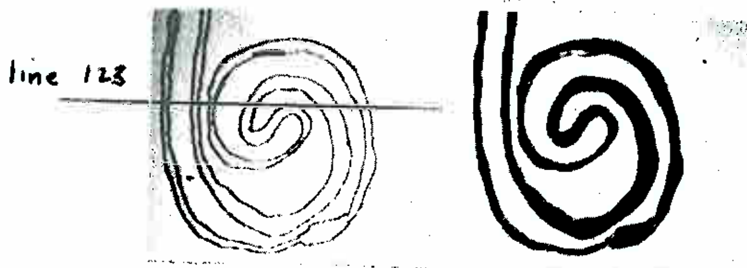
Point	Runs	SEMpoint
1	1U	NEWRUN
2	1U---------------------------------------2D	CPNRHT
3	1U 3U---------------------------------2D	SPCW-L
4	1U 3U--------------------4D 2D	SPCW-R
5	1U 3U 5U-------4D 2D	SPCW-L
6	1U 3U---6D 5U-------4D 2D	MRGRHT
7	1U 3U---6D 5U-------4D 7U---2D	MRGLFT
8	1U---8D 3U---6D 5U-------4D 7U---2D	MRGRHT

Figure 18. Example of clockwise spiral SEMpoint labeling for line y which intersects all edges of the spiral. The runs for line y are given at each step by showing runs with "---" joining the runs. Each step is followed by a U (points up) or a D (points down).



Point					SEMpoint
1	1D				NEWRUN
2	1D			2U	IMPRHT
3	1D	3D		2U	IMPLFT
4	1D	3D	4U	2U	IMPRHT
5	1D	3D	4U---5U	2U	CPNRHT
6	1D	6U---3D	4U---5U	2U	MRGLFT
7	1D	6U---3D	4U---5U	2U---7D	MRGRHT
8	8U---1D	6U---3D	4U---5U	2U---7D	MRGLFT

Figure 19. Example of counterclockwise spiral SEMpoint labeling for line y which intersects all edges of the spiral. The runs for line y are given at each step by showing runs with "---" joining the runs. Each step is followed by a U (points up) or a D (points down).



a-b

c

Figure 20. Example of RLM generation by computer of a) hand drawn double spiral boundary, (b) the inside of the boundary filled using the generated RLM, c) the SEMpoint labeling for line y=123.

6.2 CO-OCCURRENCE OF LABELING SEMANTICS AS A SHAPE DESCRIPTION

The total semantic labeling {Sti} for a boundary may be described by a semantic label co-occurrence matrix consisting of a 26x26 point array (given the possible 26 semantic labels). Analysis of the co-occurrence matrix may be used to compute several global shape features. For example, the percentage of implicit splits, IMP---, indicates the degree of concavity of the surface. The percentage of explicit splits, SPCCW- or SPCW--, indicates the amount of counterclockwise or clockwise spiral on the surface. The percentage of hairs indicates the noisiness of the boundary.

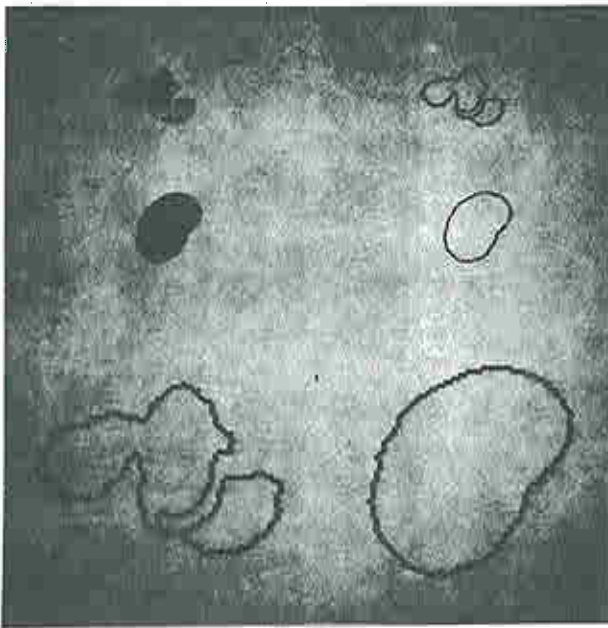
Two bone marrow image nuclear boundaries were obtained with the BMON2 system (from a Wright's stained marrow smear at 800X). These are shown in Figure 21. The top nucleus is from a mature neutrophil while the bottom is from a large lymphocyte. The SEMpoint co-occurrence matrices for the two boundaries are summarized in Table 2a and 2b.

The neutrophil boundary has many more implicit splits and spirals than the large lymphocyte, corresponding to a more convoluted boundary.

6.3 RUN LENGTH DISTRIBUTIONS OF SEMpoint LABEL SEQUENCES

Instead of analyzing the point co-occurrence of

semantic labels, the run length distributions of labels or label groups along the boundary might be analyzed to get information as to the size of specific features. Table 3 (3a for the neutrophil and 3b for the lymphocyte) shows the distributions for the two objects for the 0 degree SEMpoint labeling. Note that the first boundary has several implied splits of a reasonable size. These are the concavities on the boundary. That is, the concavities are explicitly labeled! Notice also a few smaller spiral edges on the first boundary which correspond to a few local spiral regions. Both boundaries have few hair labels which means that they are probably thresholded at about the correct threshold.



a-d

Figure 21. Nuclei of neutrophil cell (top) and large lymphocyte cell (bottom): a) gray value images, b) traces of the segmented boundaries, c) 3:1 zoom of the neutrophil, d) 3:1 zoom of the large lymphocyte.

Semantic label (0 degree only) run length distributions

NDRlength	1				5					10					15					20					25	
1 [CGLHSP]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2 [CGRHSP]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3 [CPALFT]	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4 [CPARHT]	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5 [CPALFT]	9	3	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6 [CPARHT]	3	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7 [CMPUNY]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8 [DUP-PT]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9 [FXALFT]	6	2	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10 [EXARHT]	6	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11 [EXNLFT]	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12 [EXNRHT]	2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13 [HAIREP]	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14 [IMPLFT]	3	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15 [IMPRHT]	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16 [IMPLFH]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17 [IMPRHH]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18 [INCDPT]	4	4	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19 [MRGLFT]	2	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20 [MRGRHT]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21 [NEWRUN]	8	0	4	0	2	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
22 [NULPNT]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23 [SPCW-L]	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24 [SPCW-R]	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22 [NULPNT]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25 [SPCCWL]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26 [SPCCWR]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 3a. Run length distribution matrices for the SEMpoint labeling of the neutrophil boundary.

Semantic label (0 degree only) run length distributions

NDR	length	1				5					10					15					20					25	
1	[CGLHSP]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	[CGRHSP]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	[CPALFT]	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	[CPARHT]	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	[CPALFT]	3	2	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
6	[CPARHT]	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	[CMPUNY]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	[DUP-PT]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	[EXALFT]	1	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	[EXARHT]	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	[EXNLFT]	11	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	[EXNRHT]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	[HAIREP]	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	[IMPLFT]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	[IMPRHT]	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	[IMPLFH]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	[IMPRHH]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	[INCDPT]	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	[MRGLFT]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	[MRGRHT]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	[NEWRUN]	9	1	2	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
22	[NULPNT]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	[SPCW-L]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	[SPCW-R]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	[NULPNT]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	[SPCCWL]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	[SPCCWR]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 3b. Run length distribution matrices for the SEMpoint labeling of the lymphocyte boundary.

Semantic label (0 degree only) run length distributions

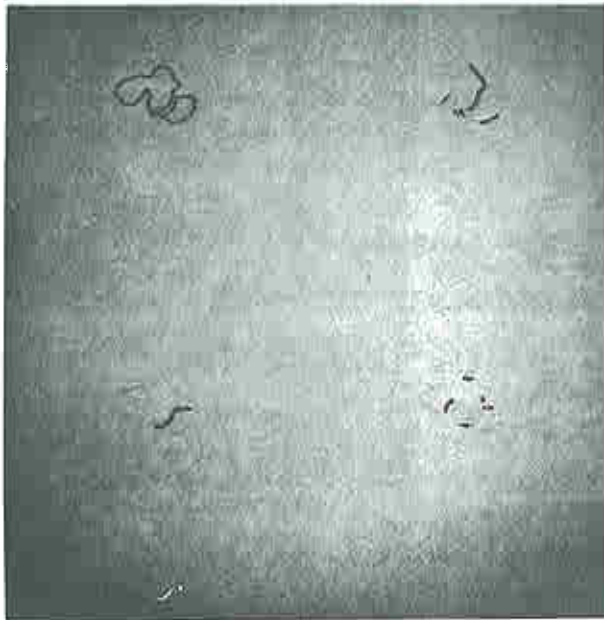
NDR	length	1				5					10					15					20					25	
1	[CGLHSP]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	[CGRHSP]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	[CPALFT]	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	[CPARHT]	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	[CPALFT]	9	3	0	1	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
6	[CPARHT]	3	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	[CMPUNY]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	[DUP-PT]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	[EXALFT]	6	2	1	0	1	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	[EXARHT]	6	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	[EXNLFT]	11	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	[EXNRHT]	2	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	[HAIREP]	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	[IMPLFT]	3	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	[IMPRHT]	1	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	[IMPLFH]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	[IMPRHH]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	[INCDPT]	4	4	0	1	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	[MRGLFT]	2	1	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	[MRGRHT]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	[NEWRUN]	8	0	4	0	2	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0
22	[NULPNT]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	[SPCW-L]	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	[SPCW-R]	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	[NULPNT]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	[SPCCWL]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	[SPCCWR]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 3a. Run length distribution matrices for the SEMpoint labeling of the neutrophil boundary.

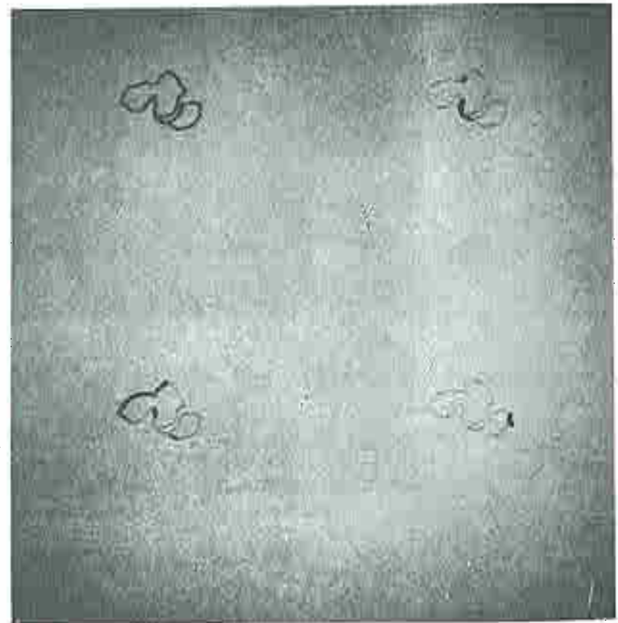
Semantic label (0 degree only) run length distributions

NDR	length	1				5					10					15					20					25	
1	[CGLHSP]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
2	[CGRHSP]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
3	[CPALFT]	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
4	[CPARHT]	5	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
5	[CPALFT]	3	2	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
6	[CPARHT]	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	[CMPUNY]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	[DUP-PT]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
9	[EXALFT]	1	4	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
10	[EXARHT]	1	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	[EXNLFT]	11	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	[EXNRHT]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	[HAIREP]	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
14	[IMPLFT]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
15	[IMPRHT]	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
16	[IMPLFH]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
17	[IMPRHH]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	[INCDPT]	6	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	[MRGLFT]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	[MRGRHT]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	[NEWRUN]	9	1	2	0	0	0	0	1	0	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0
22	[NULPNT]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	[SPCW-L]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
24	[SPCW-R]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	[NULPNT]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
25	[SPCCWL]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
26	[SPCCWR]	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Table 3b. Run length distribution matrices for the SEMpoint labeling of the lymphocyte boundary.



a-d



e-h

Figure 22. Boundaries of semantic labels. The SEMpoint at each point are used to darken those boundary points meeting a specific label at 0 degree rotation. a) all boundary points, b) NEWRUN, c) IMPLFT, d) IMPRHT, e) all boundary points, f) MRGLFT, g) CPNLFT, h) CPNRHT.

7. DATA STRUCTURES - LINKED LISTS

A linked-list representation is used for the runs. This was done for several reasons. The most important is that it is possible for the number of runs on a line y to be very large. If the runs were stored as arrays (ENT[n,y], XIT[n,y]) then the maximum storage required to handle the largest expected run would be quite large.

Each line y is stored as a separate circular doubly linked list pointed to by exptr[y]. A run in the list is a single node consisting of five fields (for run (y,i)):

backward pointer for run i to run (i-1) mod n,
forward pointer for run i to run (i+1) mod n,
last local semantic label on run i,
Ei (entrance x value for run),
Xi (exit x value for run i).

The second reason for using linked list is that the boundaries themselves are stored as linked lists which facilitates editing them as well as performing other operations as discussed in [14].

8. DISCUSSION

The RLM generation algorithm, in addition to producing an efficient boundary representation for searching regions, also provides local shape information in the form of run and boundary point semantic labels. Further experiments are in progress to investigate object shape information using these labelings.

ACKNOWLEDGEMENTS

The author would like to thank Lewis Lipkin for his enthusiastic support and discussions on the RLM algorithm. In addition thanks are due to George Carman of Corvallis Oregon, and Mort Schultz, Bruce Shapiro (as well as Lewis Lipkin) of the Image Processing Unit for their contributions in the design and construction of the RTPP system on which this algorithm was developed. Much thanks are also due to Jo Abbott for the illustrations. Parts of this paper were taken from work done in the Ph.D. dissertation of the author [13].

REFERENCES

1. Minsky M, Papert S: Perceptrons: An Introduction to Computational Geometry. MIT Press, Cambridge, Mass, 1969.

2. Freeman H: Computer Processing of Line Drawing Images. Comp Surveys, Vol 6, March 1974, 57:97.

3. Davis L S: Understanding Shape: Angles and Sides. IEEE Trans Comp, Vol C-26, 1977, 236:242.

4. Burton W: Representation of Many-Sided Polygons and Polygonal lines for Rapid Processing. CACM Vol 20, 1977, 166:176.

5. Rutovitz D: Data Structures for Operations on Digital Images. In "Pictorial Pattern Recognition", G. C. Cheng etal. (eds), Thompson Book Co, Wash DC, 1968, 115:117.

6. Rounds E: Figure Construction from its Contours. USCIPI Report 720, Sept, 1976, 55:62.

7. Agrawala A K, Kulkarni A V: A Sequential Approach to the Extraction of Shape Features. Comp Graph Image Proc, Vol 6, 1977, 538:557.

8. Merrill R D:Representation of Continuous Regions for Efficient Computer Search. CACM, Vol 16, 1973, 69:82.

9. Lemkin P:Buffer Memory Monitor System for Interactive Image Processing. NCI/IP Technical Report #21b, NTIS PB278789

(listings PB278790), April, 1978.

10. Lemkin P, Carman G, Lipkin L, Shapiro B, Schultz M, Kaiser P:A Real Time Picture Processor for Use in Biological Cell Identification - I System Design. J Hist Cyto, Vol 22, 1974, 725:731.

11. Carman G, Lemkin P, Lipkin L, Shapiro B, Schultz M, Kaiser P:A Real Time Picture Processor for Use in Biological Cell Identification - II Hardware Implementation. J Hist Cyto, Vol 22, 1974, 732:740.

12. Lemkin P, Carman G, Lipkin L, Shapiro B, Schultz M:Real Time Picture Processor - Description and Specification. NCI/IP Technical Report #7a, NTIS PB269600/AS, June, 1977.

13. Lemkin P: Bone Marrow Smear Image Analysis. Ph.D. Disseratation, Univ. Md, College Park, Md., 1978.

14. Zahn C T: Data Structures for Pattern Recognition Algorithms: A Case Study. Proc Conf Comp Graph, Pat Rec, and Data Struct, Beverly Hills, Ca, May, 1975, 191:195.