The Creation Of Topological Outlines From Binary Images

Dr D. G. Elliman & P. J. Connor
Drawing Recognition Group
University of Nottingham

Abstract

This paper presents a fast and efficient method for producing outlines from binary images, in a single pass. The method can be applied directly to runlength encoded data, and its' speed is independent of the length of image runlengths.

It is demonstrated how outline code may be created using only four x pixel values in the memory at a time. The simultaneous creation of the topology information for any picture is also presented.

The approach is faster than previously described masking techniques, and may generate outline loops as Freeman code, coordinate pairs or in the novel form of "orthogonal vector code". By testing for directional patterns, the outlines are automatically ordered anticlockwise for external loops and clockwise for internal loops.

Cet exposé présente une méthode rapide et efficace de produire des contours à partir d'images binaires au cours d'une lecture simple. Cette méthode peut s'appliquer directement aux données encodées comme runlength, et sa vitesse est indépendante de la longeur des runlengths des images.

Il est démontré comment créer un code de contour employant seulement quatre valeurs de pixel à la fois dans la mémoire. La création simultanée de l'information topologique pour toute image est aussi présentée.

Ce procédé est plus rapide que les méthodes de correspondance de masques décrites précédemment, et peut générer bes boucles de contour comme le code Freeman, les paires de coordonnées cartésiennes ou dans la forme originale de "code de vecteur orthogonal". En essayant de découvrir des schémas directionnels, les contours sont automatiquement ordonnés dans le sens inverse des aiguilles d'une montre pour les boucles extérieures et dans le sens des aiguilles d'une montre pour les boucles internes.

Keywords: Outline, Topology, Vector Code, Runlength.

1. Introduction

The uses for extracted boundaries in picture processing has been well established, and many papers have been written on the creation of outlines from runlength and pixel images.[1-3] By finding the outline loops and their topology, the information contained in an image can be greatly compacted, or processed using recognition techniques.[2,4]

Of the papers written on outline creation from digitised images, most methods presented use a 3x3 or 2x2 pixel mask over the image to find the positions where black and white pixels are adjacent. As the image size or resolution is increased, so the time taken, on conventional architectures, for the masking operation increases dramatically. Kim et al[5] use the first and last x values of runlengths to narrow the search for outlines, but require two intermediate storage tables for the image to be held in memory, and a subsequent pass of those tables to find the outline. Cederberg[3] suggests a one-pass method which creates "RC-code", held in a graph structure containing the information as to how the chains of RC-code were joined. His method avoids the need to store the whole pixel or runlength image in memory, by creating loops dynamically from three lines of the image. He uses a masking technique to find "max" and "min" points, and many masks need to be applied to every 2x2 pixel area on the image to find the correct outline. As many as four masks may be successfully matched on each 2x2 area, and the implementation of the method is likely to be slow. The RC-code also requires subsequent re-ordering to create Freeman code or vector output.

Capson[6] presents a one-pass algorithm, which creates circular linked lists of coordinate pairs, in anticlockwise and clockwise sense for objects and holes respectively. There is not a clear definition of an output point, and the algorithm only gives a close vector approximation to the holes within an object.

As part of the work of the drawing recognition group we require to create outline descriptions which have anticlockwise external loops and clockwise internal loops. For character/symbol recognition it is also important to have a complete description of the object/hole relationships (topology). The method presented here is designed to find this information quickly and efficiently. It is similar to that of Capson, finding the outline information in one pass of the lines in the image, and an improved algorithm to create the outlines is given in an appendix. Like Capson's method the object/hole inclusion relationships are maintained, in addition the similar relationship between objects-within-holes in objects is also found. It does not utilise any masking techniques, and uses only the x values for the first and last pixel of one or two black runlength on two consecutive lines of the image to be held in memory at any time.

The definitions used to describe the outlines are given in the section 2. The vector chain and loop topology creation are described in section 3. The algorithm presented can create outlines of the commonly accepted types such as the x,y coordinates of the edge pixels, or chain code such as Freeman code. An alternative outline description, "orthogonal" vector code, is given in section 4. Implementation and an example are given in section 5.

2. Definitions.

2.1 Connectivity of Edge Pixels

The image can be any combination of black and white runlengths on consecutive image lines. The connectivity definition used in the algorithm is that of 8 connected black pixels. However it is very simple to adapt the tests to use a 4 or 6 connected definition. A full description of connectivity is given by Rosenfeld[7]

2.2 Outline Loops

For the purposes of this paper an outline is the path along the junction between the black edge pixels and white runlengths. Black pixels are always on the left side and white pixels are always on the right side of the outline path. Such a path is shown in figure 1. This has the effect of creating anticlockwise external and clockwise internal outlines. Each change of direction in the outline path is represented by a vector, whose length is an integer number of pixels. It is possible to specify that the first vector in the loop will be the one with the lowest y value. It is described in section 4 why this type of outline gives a more consistent treatment to objects and holes. However the algorithm presented can be easily used to create other output such as freeman code or 8 connected vector chains.

Black runlengths which occur on the edge of the view area indicate that the loop created may not truly represent the desired object. In these cases the outline can be created, but it should be noted that this loop has touched the image edge.

2.3 Topology

Finding whether an object is inside a hole in a larger object has long been a problem in image processing.[8,9] Euler number counting is unreliable, and almost all previous work has concentrated on finding holes in objects and not objects within objects. By noting the order in which loops open and close, or merge, it is possible to build up a complete topology for any image, in only one pass. The topology is described here in terms of the relationships between closed loops. If loop B is enclosed within another loop A, it is a child_loop of loop A, and loop A is the parent_loop of loop B. Any other loop C, also enclosed within loop A, is the brother_loop of loop B, as it also has loop A as its' parent_loop.

3. Creating Outline Loops and Image Topology

3.1 Finding the Outline

It is obvious that the first and last black pixel of a runlength are edge pixels, but to discover whether the top and bottom edges of a black runlength lie on the outline it is necessary to find the relative positions of black runlengths on the line above and below it. Comparison of the start and end x values of black runlengths on consecutive lines in any image, indicate the position of the outline. Ten distinct cases can be found which describe any outline. Examples of these ten patterns, and the conditions for each, are given in figure 2.

An algorithm is suggested in appendix A for finding these patterns by comparison of the first and last x pixel values of black runlengths on two consecutive lines in the image. The algorithm incorporates the idea that certain patterns can never follow other patterns in the image. For example it is not possible for an external loop to open (pattern 1) immediately after an internal loop has just closed (pattern 10), so it is not tested for.

3.2 Creating the vector chain.

The outline information provided by finding these patterns needs to be related to the outline found from patterns in the rest of the image. According to the specific patterns found a number of vectors may be created, or elongated, to represent the path around the outline. They are linked together into chains, and each chain will either have two open_ends or will join to form a closed loop. As Cederberg mentions[3] the open_ends are added to in the same order for each image line, so a list of open_ends is stored in the order that they appear in a scan in the x direction of the image. When an open pattern (1 or 2) occurs two new open_ends are inserted into this list. One of the patterns 3 to 10 will always occur directly underneath an open_end. Thus an open_end will have vectors added to it with every image line processed, until it merges with another open_end. When open_ends merge they are removed from the list of open_ends, as no more vectors will be added to those ends of the vector chains.

The ten patterns, in figure 2, have been grouped. Each group has a similar effect on the open_end list and vector chains. The groups are:-

Opens, External and Internal (patterns 1 & 2), occur at the opening of a loop and initiates a vector chain. The outline found runs from the end of the runlength on the second line to the start of the runlength on the second line (BE to BS). Note that in the case of an internal open the end and start referred to belong to consecutive runlengths. Two open_ends are inserted into the list at the current position, to keep track of both the ends of this vector chain.

Start patterns, Left, Right and Vertical (patterns 3 4 & 5), always occur below the open_end of an existing loop. The result is that the outline runs from the start of the runlength on the first line to the start of the runlength on the second line (AS to BS).

End patterns, Left, Right and Vertical (patterns 6 7 & 8), always occur below the open_end of an existing loop. The result is that the outline runs from the end of the runlength on the second line to the end of the runlength on the first line (BE to AE).

Merge pattern, External and Internal (patterns 9 & 10), always occur with the start and end edges of the runlengths on the first line, (AS and AE), vertically under two open_ends and so, join these chains together. The outline runs from the start of the runlength on the first line to the end of the runlength on the first line (AS to AE). In the case of an internal merge the end and start belong to consecutive runlengths. The chains being merged are no longer open and so both open_ends are removed from the list of

open_ends, and the type of merger is tested, as explained in the next section.

All the patterns have rules as to which side of external and internal loops they can occur on. For example the start patterns only occur on the left sides of external and the right side of internal loops. Being able to predict which side of a loop the pattern occurs on allows the patterns to be given an inherent "sense of direction", which creates anti-clockwise external and clockwise internal loops vector chains. The direction attributed to each pattern is shown on figure 2. Using this attribute, loops can be chained together without any need to reorder the vectors when open_ends merge.

3.3 Merging vector chains

The concept of open and merge patterns in the image was developed by the author but subsequently found to be similar to D-run and F-run patterns introduced by Kim et al.[5] Their method uses the patterns to create an intermediate stage table, called a COD table, which they then use to create output code in a second pass of the data. This requires the use of intermediate files or extra memory. It is also related to Capson's terminate_an_object[6], but his method does not keep track of the relationship between objects-within-holes in objects.

The following method overcomes these short-comings, due to the sense of direction given to each pattern, and the manner in which the merging of two open_ends is handled. The merging of vector chains results in the creation of a longer, correctly ordered, vector chain. When the two open_ends of the same vector chain merge, the result is a correctly ordered loop.

When a loop opens, the two open_ends inserted into the open_end list are given labels Left and Right. Merge patterns will be one of 4 types according to these labels.

1. Left-Left, the Left open_ends of two loops becoming joined.(Figure 3.a)
2. Right-Right, the Right open_ends of two loops becoming joined.(Figure 3.b)
3. Right-Left, the Right open_end of a loop joining onto the Left open_end of another.(Figure 3.c)
4. Left-Right, the two open_ends of the same loop becoming joined, and thus closing.(Figure 3.d)

If the merge pattern is one of the first three types, then two separate chains of vectors are becoming joined. This results in the relabelling of the other ends of one or both chains, and the correction of the topological information.

When a Left-Left merge occurs between two open_ends, their other ends remain in the list of open_ends, but are now part of the same loop. The second of the Left open_ends previously belonged to a vector chain completely surrounded by the first of the Left open_end's vector chain. In figure 3a this can be seen as loop 3 is completely encompassed by loop 2 as the Left-Left merge occurs. Thus the second of the two Left open_ends has its' other open_end relabelled as a Left end, and belonging to the same loop as the first Left open_end.

It can be seen from figure 3.b, that a Right-Left merge leaves the other end correctly labelled as Left and Right, but they now belong to the same loop. At this point it is useful to know which loop opened first in the image. If this has been stored in relation to the loop number, as an opening coordinate, then the loop with the lowest coordinate is retained, and the other end which does not have that loop number is relabelled. This is useful for character recognition techniques where it is important to know that the vector chain for each loop start at the lowest value.

For a Right-Right merge the first Right open_end belongs to a vector chain completely surrounded by the vector chain associated with the second Right open_end. In figure 3.c this can be seen as loop 4 is completely encompassed by loop 2 as the Right-Right merge occurs. The Left open_end of the inner loop is relabelled as being a Right open_end of the outer loop.

A Left-Right merge indicates that a loop has actually closed and can be thus be added to the topology tree. When a loop closes in the image the open_ends involved are each other's other ends, and so there are no open_ends still in the list that need relabelling.

3.4 The Topology of Closed Loops.

The Topology of loops is created dynamically as merge patterns are found in the image. The four types of merge each require different changes to a tree type network of closed loops. As loops close their brother_loops and the parent_loops are set to probable values. The probable parent_loop is the first loop in the open_ends list which only has its' Right open_end after the closing loop. That implies that the Left open_end must be in the list before the closing loop and thus surrounds the closed loop. When vector chains merge this labelling may prove to me wrong and so the child_loops of a merging chain are corrected as described in Appendix B, and shown in figure 3.

By labelling loops as being within other loops, or at outermost level 0, the topology of the picture is created. When a level 0 loop closes, the loop and all loops within it are correctly labelled. In order that all loops in the image open and close correctly it is necessary to create an extra blank line before the first line in the image and after the last. Thus any black runlengths on the top or bottom lines are seen to be the edges of a loop. Any loop opening or closing on the top or bottom line, or which includes the edges of runlengths which touch the sides of the image should be noted as such, as they may not correctly represent the complete outline of the object in question.

4. Orthogonal vector code.

This section deals specifically with the creation of "orthogonal vector code". The definition of an outline given in section 2 is different to that used by previous authors[3,5] and leads to an outline which contains only 90 degree corners. We will refer to this type of outline as the orthogonal outline.

Orthogonal vector code is designed to represent an orthogonal outline path. It is similar to the type of output

for Freeman's 4 connected black outlines[1] , except that the outline can never double back on itself by 180 degrees. Each vector in the chain is orthogonal to the vectors before and after it in the chain, and thus by definition parallel to all vectors which are an even number of vectors away. Each vector is given a length, which is the delta x or delta y value of that line between corners found in the outline. The sign of the length is in relation to the positive x and y directions. Because there is no indication as to whether length represents a change in x or y, the first vector in the chain always represents a change in the x direction and the x,y coordinate of the start of this vector is stored in relation to the loop number.

From orthogonal vector output it is possible to estimate the quantisation effects of restricting the image to a pixel grid, and thus it is possible to fit lines to the orthogonal vector output using very simple tests. This will be the subject of a future paper.

For each pattern a specific number of vectors are added to the chain, and their length is set. All patterns which leave an open_end in the list also add vertical vectors along the ends of the black runlengths on the second line (or increment the length of an existing vertical vector). These are the chaining points for the vectors of patterns occurring on the next line in the image. The vectors added for each pattern are those shown by a dark line in figure 2, and their sign is indicated by the arrow.

This output has been chosen as it treats pixels as objects not single points. As a pixel has four sides it is possible to be an edge pixel four times, thus each side can be represented separately, not simply by a center point.

Although there are ten patterns it is possible to produce orthogonal vector code for more than one pattern with the same routine, for example Left and Right start pattern vectors. However if Freeman code or an 8 connected outline definition is being produced, then this can only be achieved by taking different action for each of the ten patterns.

5. Implementation

The algorithm presented and structures used have been implemented in C on an IBM PC and a SUN 3/140 as part of Nottingham University's Engineering Drawing Recognition Research. Successful hierarchy creation has been achieved with every image. An example image is given in figure 4, which shows the topology found as a "family-tree" of parent/child/brother relationships.

6. Conclusions

The method presented represents a simple one-pass method of creating outlines of an image from only four x pixel values lines of the image.

The method can create the correct output code on the first pass of the data, by the detection of directional patterns. The recognition of these directional patterns is very simple, and a suitable algorithm has been written to implement this.

By using a list of sides presently open at any point in the image the time taken to create the outlines is greatly reduced. This list also allows the labelling of loops to create a topology or inclusion relationship between closed loops, to any depth.

The efficiency of this method has not been quantified, but as it uses only one pass and creates correctly ordered output code and topological relationships automatically, it is considered to be an important development on previously presented methods.

References

1. H. Freeman, "On the Encoding of Arbitrary Geometric Configurations," *IRE Transactions on Electronic Computers*, pp. 260-268, June 1961.
2. Okazaki, et al, "An Automatic Circuit Diagram Reader with Loop-Structure-Based Symbol Recognition," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 10, no. 3, pp. 331-341, IEEE, May 1988.
3. R. L. T. Cederberg, "Chain-Link Coding and Segmentation for Raster Scan Devices," *Computer Graphics and Image Processing*, no. 10, pp. 224-234, Academic Press, Jan 1979.
4. M. T. Y. Lai and C. Y. Suen, "An Automatic Recognition of Charachters by Fourier Descriptors and Boundary Line Encodings," *Pattern Recognition*, vol. 14, no. 1-6, pp. 383-393, Pergamon Press, 1981.
5. Kim Jae-Kyoon, "A New Chain-Code Algorithm For Binary Images Using Run-Length Codes," *Computer Vision, Graphics and Image Processing*, vol. 41, pp. 114-128, Academic Press, 1988.
6. D. W. Capson, "An Improved Algorithm for the Sequential Extraction of Boundaries from a Raster Scan," *Computer Vision, Graphics and Image Processing*, no. 28, pp. 109-125, Academic Press, 1984.
7. A. Rosenfeld, "Connectivity in Digital Pictures," *Journal of the ACM*, vol. 17, no. 1, pp. 146-160, 1970.
8. S. B. Gray, "Local Properties of Binary Images in Two Dimensions," *IEEE Trans. Comput.*, vol. C-20, no. 5, pp. 551-561, 1971.
9. W.H.H.J. Lunscher and M. P. Beddoes, "Fast Binary-Image Boundary Extraction," *Computer Vision, Graphics and Image Processing*, vol. 38, pp. 229-257, Academic Press, 1987.

For further information contact:-
Dr D. G. Elliman or P. J Connor
Computer Science Department,
University of Nottingham,
Nottingham. NG7 2RD.
England.

email: dge@Cs.Nott.AC.UK (janet)
telephone:(England) 0602-484848

Figure 1. Outline of a Bilevel Image

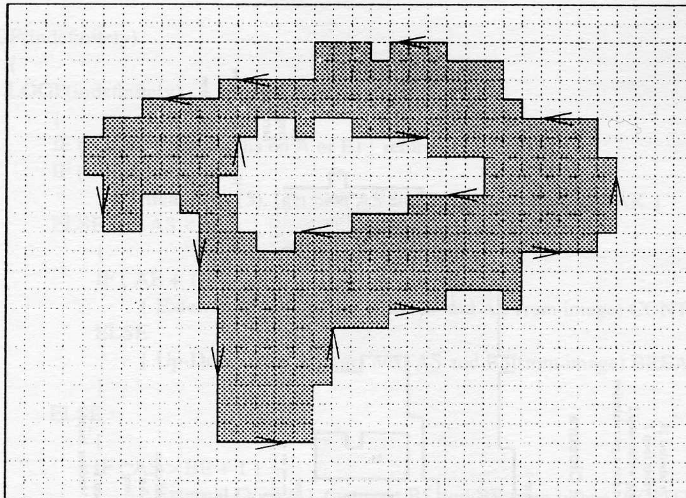

Figure 2. Outline patterns and conditions

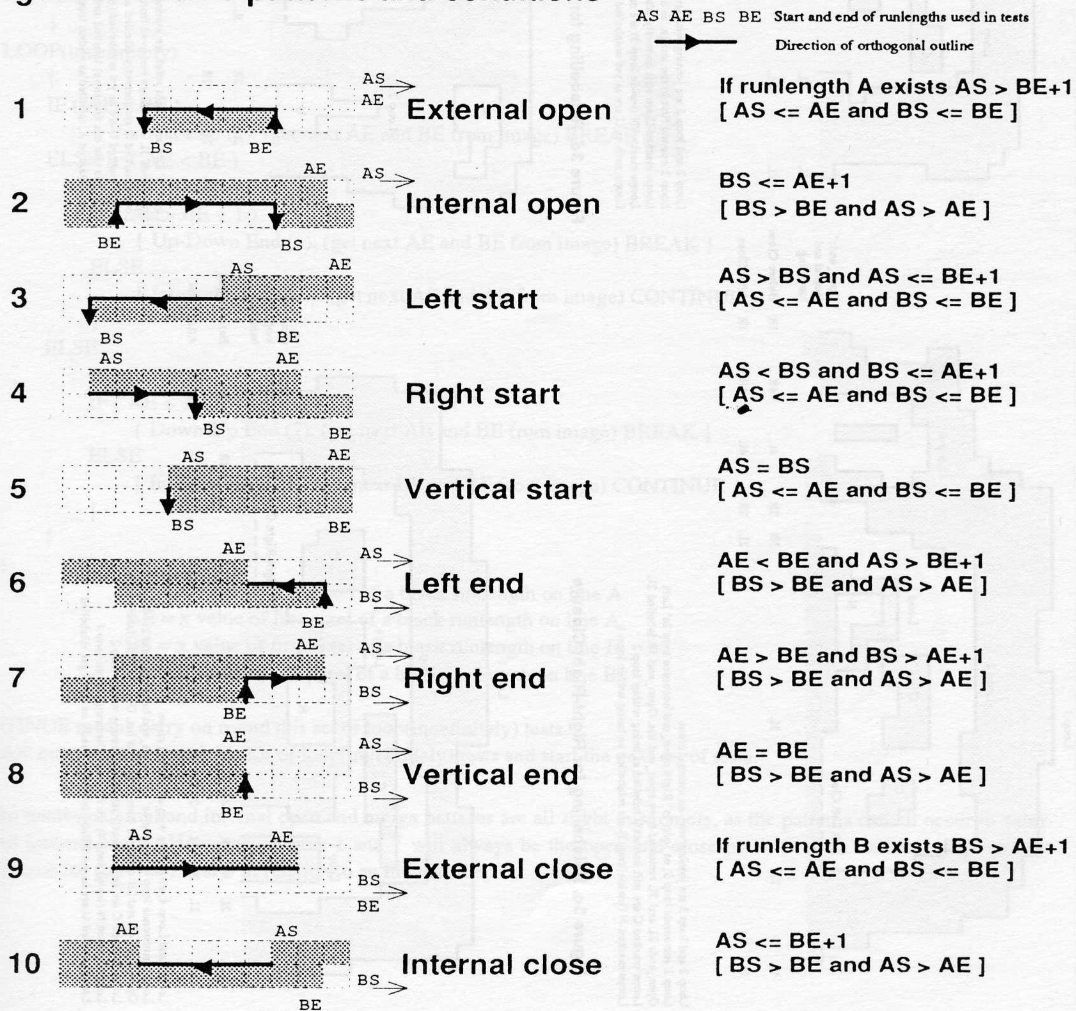

Figure 3a. Relabelling For Left-Left Closure

Loop 2 and loop 3 are joining with an internal close
Loop 2 surrounds loop 3, so loop 3s' Right open_end becomes loop 2s' Left
Open_ends 2L and 3L removed from open_end list. Open_end 3R labelled 2L
Loops marked C are still correctly labelled as being within loop 2
Loops marked N (inside loop 3) are relabelled as being within loop 1

Figure 3b. Relabelling for Right-Left Closure

Loop 2 and loop 3 are joining with an external close
Loop 2 opened first in the image, so open_end 3R is relabelled 2R
Loops marked C are still correctly labelled as being within loop 2
Loops marked N (inside loop 3) are relabelled as being within loop 2
Loops such as U are as yet unlabelled (not closed), so don't get relabelled

Figure 3c. Relabelling for Right-Right Closure

Loop 2 and Loop 4 are joining with an internal close
Loop 2 surrounds loop 4, so loop 4s' Left open_end becomes loop 2s' Right
Open_ends 4R and 2R removed from open_end list. Open_end 4L labelled 2R
Loops marked C are still correctly labelled as beong within loop 2
Loops marked N (inside loop 4) are relabelled as being within loop 1
Loops marked U are as yet unlabelled (not closed), so don't get relabelled

Figure 3d. Relabelling for Left-Right Closure

Loop 3s' Left and Right sides are joining with an internal close
Open_ends 3L and 3R are removed from the open_end list
Loops marked C are correctly labelled as being within loop 3
Loops marked U are as yet unlabelled (not closed)
Loop 3 is (for the first time) labelled as being within loop 1
Loops marked A, labelled as within loop 1, are loop 3s' brother loops

Appendix A. Pattern finding algorithm called for each pair of lines.

```
LOOP(indefinitely)
  {
  LOOP(indefinitely)
    {
    IF (No more runlengths on A or B) { return to get next line }
    IF ( AS = BS )
        { Vertical Start (5). (get next AS and BS from image) BREAK }
    ELSE IF ( AS < BS )
        {
        IF ( AE + 1 < BS )
            { External Close (9). (get next AS and AE from image) CONTINUE }
        ELSE
            { Up-Down Start (4). (get next AS and BS from image) BREAK }
        }
    ELSE
        {
        IF ( AS > BE + 1 )
            { External Open (1). (get next BS and BE from image) CONTINUE }
        ELSE
            { Down-Up Start (3). (get next AS and BS from image) BREAK }
        }

    LOOP(indefinitely)
    {
    IF ( AE = BE )
        { Vertical End (8). (get next AE and BE from image) BREAK }
    ELSE IF ( AE < BE )
        {
        IF ( AS > BE + 1 )
            { Up-Down End (6). (get next AE and BE from image) BREAK }
        ELSE
            { Internal Close (10). (get next AS and AE from image) CONTINUE }
        }
    ELSE
        {
        IF ( AE + 1 < BS )
            { Down-Up End (7). (get next AE and BE from image) BREAK }
        ELSE
            { Internal Open (2). (get next BS and BE from image) CONTINUE }
        }

    }
}
```

AS = x value of first pixel of a black runlength on line A
AE = x value of last pixel of a black runlength on line A
BS = x value of first pixel of a black runlength on line B
BE = x value of last pixel of a black runlength on line B

CONTINUE means carry on round this set of loop(indefinitely) tests
BREAK means jump out of this set of loop(indefinitely) tests and start the next set of tests.

The names external and internal open and merge patterns are all slight misnomers, as the patterns can all occur in external and internal loops. However patterns 1 and 7 will always be the open and close patterns of an external loop, and the same is true for patterns 2 and 8 in relation to an internal loop.

Appendix B. Creation and Correction to Topology

These relabelling schemes treat external loops and internal loops identically. Thus the child_loop/parent_loop relationships may be taken to any depth. The following is a description of the changes to the parent_loop, child_loop and brother_loop lists for the loops involved in a merging of open_ends.

If both open_ends of the same loop are joined (a Left-Right merger) then that loop has been completed and can be added to a tree of closed loops thus.

a. As an initial guess the parent_loop is set to the first vector chain with its' Left open_end to the left of the closed loop and its' Right open end to the right of the closed loop. This can be found by inspection of the open_end list as the one which **only** has its' Right open_end after those of the closed loop.

b. The closed loop's brother_loop is the loop currently held as this parent_loop's child_loop, if it has one.

c. The parent_loop's child_loop becomes the newly closed loop.

Figure 3.d shows an example of a Left-Right merger. The closed loop, 3, should be labelled as being within loop 1, in the manner described above; i.e. loop 1 is loop 3's parent_loop, loops marked A are loop 3's brother_loops and loop 3 is loop 1's child_loop. (This may later prove to be incorrect if loop 1 does not close around loop 3. Loops such as 4 which have both their Left and Right open_ends to the right of the closed loop are ignored as the open_end list is searched to find loop 3's parent_loop. If the search shows that a parent_loop does not exist then the closed loop is labelled as being at the outermost level 0. This may be achieved by having a dummy loop which is the parent_loop of all level 0 loops. After processing the whole image, all level 0 loops are contained within the brother_loop list of the dummy's child_loop.

When a loop closes, any loops for which it is the parent i.e. child_loop or child_loop's brothers, are correct, and there will not be any other loops inside the loop which has just closed. Examples of correctly labelled loops are marked C in figure 3.d.

When a Left-Left merge occurs between two open_ends, the child_loop and its' brother_loops for the second vector chain have the wrong parent_loop label. They are relabelled as being the child_loops of the vector chain which has only its' Right open_end after the Right open_end of the first vector chains involved in the merge. This can be found by inspection of the open_end list. They are then placed on the end of the list of brother_loops for the correct parent's child_loop. If such a loop does not exist the wrongly labelled loops are labelled as being at level 0 and within the dummy loop. An example of a Left-Left merger is given in figure 3.a. The loop marked N is incorrectly labelled as being within loop 3, and so is relabelled as being a child_loop of loop 1.

When a Right-Right merge occurs, the child_loop, and its' brother_loops, for the first vector chain have the wrong parent_loop label. They are relabelled as having the parent_loop of the vector chain which has only its' Right open_end after the Right open_end of the second vector chains involved in the merge. This can be found by inspection of the open_end list. They are then placed on the end of the list of brother_loops for the correct parent's child_loop. If no suitable loop exists for the parent_loop, the wrongly labelled loops are labelled as being at level 0 and within the dummy loop. An example of a Right-Right merge is given in figure 5.c. The loop marked N. is incorrectly labelled as being within loop 4, and so is relabelled as being the child_loop of loop 1.

When a Right-Left merge occurs, and it has been established which loop is to be kept as described above, any loops which are within the second loop to open are relabelled as child_loops of the first loop to open. An example of a Right-Left merger is shown in figure 3.b. Loop 2 opened with a smaller y value than loop 3, so loops within loop 3, marked N, are incorrectly labelled and should be relabelled as being within loop 2.

Figure 4. Topological Relationships, after closure of all loops