ISSN 2347 - 3983
Volume 4, No.7 July 2016
International Journal of Emerging Trends in Engineering Research
Available Online at http://www.warse.org/IJETER/static/pdf/file/ijeter01472016.pdf

Image Processing of Planar Digital Curves Using a Chain Code-based Technique for Edge Characterisation

M. Gooroochurn[a], D. Kerr[b], K. Bouazza-Marouf[b]
M.Gooroochurn@uom.ac.mu, d.kerr@lboro.ac.uk, k.bouazza-marouf@lboro.ac.uk
[a] Mechanical and Production Engineering Department
University of Mauritius, Reduit, Mauritius
[b] Wolfson School of Mechanical and Manufacturing Engineering
Loughborough University, Loughborough, UK

ABSTRACT

Much effort has been expended for devising solutions to analyse edges as a basis for image understanding. This paper presents image processing and analysis tools to ensure a robust operation of such edge-based algorithms. Specifically, a systematic paradigm to branching analysis and a curve segmentation technique based on chain codes are proposed. Several curve segmentation approaches are available in the literature to break edges into primitives for edge analysis. The particular solution devised is simple and very computationally efficient. The efficacy of the set of tools (branching analysis and curve segmentation) in the applications presented show that they can be effectively incorporated in low-level and intermediate level edge processing.

Keywords: Curve Segmentation, Edge Analysis, Chain Code, Branching Analysis, Polynomial Fitting.

1 INTRODUCTION

Several image processing applications use edges as basis for image understanding due to their low dimensionality. These edges are normally obtained by applying edge detectors such as Canny, Sobel and Laplacian operators. These operators work at a low-level image processing hierarchy, following which image analysis is carried out by extracting various features from these lines, culminating into image understanding. However, before the image analysis stage, it should be ascertained that the edges obtained are properly represented; this is referred to as edge conditioning henceforth. In the present context, the subsequent image analysis phase is analogous to characterising these edges by calculating geometrical attributes, which are then employed for image understanding.

One method of representing single-pixel width edges is proposed by Freeman [1], which is commonly known as chain codes. The aim of Freeman in formulating chain codes was to provide a means by which an edge discretised over a grid could be represented and transmitted effectively. He showed that his solution to edge representation gave a compact form to represent edges while having desirable properties such as easy computation of coordinates for geometrical operations, length of edge and bounded area. Similar work for 2D and 3D shapes representation has been carried out by Bribiesca [2, 3]. The analysis of edges for high-level understanding is usually done by breaking the edge representation into primitives. Ichoku et al. [4] describes the curve segmentation problem into two broad classifications: breakpoint detection and edge approximation. Algorithms in the breakpoint detection category [5] operate by first determining points separating the different segments and then (optionally) fitting lines/arcs to them, while the edge approximation category [4-8] finds the segments first by fitting these primitives and then (optionally) assign breakpoints at the end points of the lines/arcs. Ichoku et al. [4] discusses the pros and cons of these two approaches.

The selection of breakpoints is usually formulated in terms of curvature and arc lengths/orientations. The need to compute curvature along the curve and the fact that curvature cannot be exactly replicated in the digital image grid, being a mathematical concept applied to continuous curve, have been considered as shortcomings. In analysing the efficacy of these algorithms, the need to compute attributes and the requirement for iterations to converge to the breakpoints are further factors to be considered.

Having to calculate attributes at each point along the curve leads to computationally intensive solutions, which is further increased should the algorithm involve iterations. Another important factor considered in assessing the methods has been the need for setting thresholds, which in turn introduces subjectivity in the solution. This usually involves selecting a threshold level which discounts noise while preserving fine details in the image. In this view, threshold selection is considered a disadvantage and solutions with no threshold selection [9] or having little sensitivity to the selected threshold have been attempted.

The corresponding work in this paper is inspired from the discretisation nature of an 8-neighbourhood grid that is commonly used in image formation and display to propose a solution that can be used for edge characterisation. Linked with this discretised representation is the chain code, which is thus used as the basis for the algorithm. Approaches based on chain code for curve segmentation have been reported e.g. [6, 9]. The method of Arrebola et al. [9] is based on an image pyramid to analyse the curve at various scales, thus leading to high computational costs, although the authors report less computation costs compared to similar multi-scale methods.

The solution proposed by Baruch and Loew [6] is recursive and involves the computation of angles and arc lengths based on which segments are identified. Typical of these chain code-based techniques is the information provided by the end result regarding the directional attributes of the segments found as such directional information are inherent in chain code values. Furthermore, invariance to rotation can be achieved by using differential chain codes to re-formulate the chain code sequence.

Most work in curve segmentation assumes that edges do not contain branches. However, the possibility exists that single pixel width edges contain branches, e.g. due to skeletonisation. So this paper first proposes a systematic approach to branching analysis for finding the different main edge segments making up the edge. Then the edge segmentation approach is presented. The chain code solution is based on gross chain code representations derived from Freeman's chain code which run along different main directions (up/down or diagonal).

The advantages of this solution are: (1) it does not require any threshold selection, (2) it operates directly on the chain code sequence, which is a common representation of edges, (3) it does not require the computation of any attributes along the line, (4) it operates on integer numbers only, thus costly computation with floating point numbers are avoided and (5) it does not involve any iteration. Similar to the outcome of the other chain code approaches, the end result gives information about the directional attributes of the segments, which can be used to effectively fit polynomials to the segments.

One example of an application where the segmentation of an edge contour into its constituent components is the fitting of a curve along the edge points. When the points on the edge are known to lie on an ellipse or a circle, robust methods can be used to fit the appropriate polynomial to the edge points, however, in a real-world application where such assumptions cannot always be made, it would be helpful to break down the edges into its constituent components to which injective polynomials can then be applied, either using x or y as the independent variable. The proposed gross chain code representations allow this polynomial fitting as shown by the examples of Section 5.

2 Branching Analysis: A Heuristic Approach

This section formulates the different heuristic rules that have been implemented to set up the proposed branching analysis methodology. The basis and implications of these rules are then described. These rules are given in Table 1.

Rule (I) enunciates the basic method by which an edge is traced, namely by moving from edge pixel to edge pixel, while analysing the 3x3 neighbourhoods obtained. Rule (II) prevents any ambiguity in moving ahead by zeroing the edge pixel already traced. Rule (III) sets the condition for assigning an edge pixel as a branch point. This is done as a result of the analysis of the 3x3 kernel, as per Rule (I), to decide which of the eight border pixels in the 3x3 neighbourhood need(s) to be temporarily set to zero to remove ambiguity in tracing the edge. More information is given on this matter later in this section.

Rules	Description
I	A 3x3 kernel is placed over a given edge pixel and the analysis of this 3x3 neighbourhood determines how to move ahead. 8-connectivity is used.
II	In moving to the next pixel location following rule (I) to yield a new 3x3 kernel, the central position occupied by the previous 3x3 kernel is set to zero.
III	Following the analysis of the 3x3 kernel as per rule (I), the presence of more than one path signifies a branching point.
IV	In tracing an edge, priority is given to movements in the following order: horizontal, vertical and then diagonal.
V	Rule (IV) is not abided for configurations that would cause ambiguity in edge tracing at the next step.
VI	Edges are assumed to be of single pixel width.

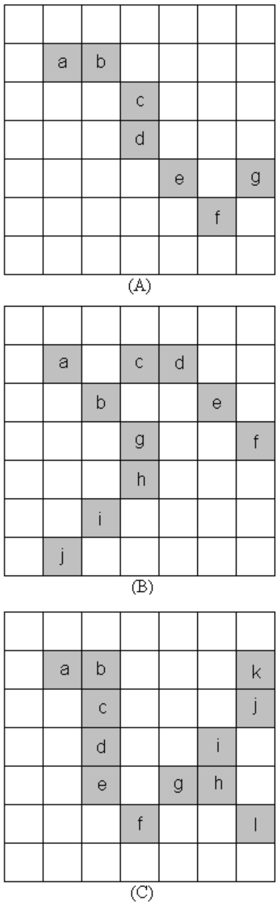
Table 1: Number of Configurations grouped by sum of border pixels

Rule (IV) allows to determine how an edge is traced in a predictable manner wherever different alternatives exist. Rule (V) has been formulated to cater for few configurations that cause ambiguity in tracing the edge as shown further. Rule (VI) is a common assumption made in edge processing applications and allows to choose which of the 256 possible combinations of the eight border pixels (the central pixel is necessarily a '1') would be encountered in tracing an edge. These 256 pixel configurations were analysed to systematically identify those which are possible in a single pixel width edge. The configurations were grouped according to the sum of the 8 border pixel locations. Table 2 summarises the number of configurations obtained in each group (second column).

Sum	Total Number of Configurations	Subset of Possible Configurations
1	8	8
2	28	28
3	56	52
4	70	28
5	56	2
6	28	0
7	8	0
8	1	0

Table 2: Number of Configurations grouped by sum of border pixels

The findings from this analysis have been used in the next section to implement the algorithm for recovering the branches along the edge in a robust manner. The need to discount edge pixels in a given 3x3 neighbourhood is illustrated by the three examples in Figure 1 where the shaded pixels represent edge points. Figure 1(A) illustrates an example where there are no branching while examples (B) and (C) involve branching. For example (A), taking pixel ‘a’ as the start point, the whole edge can be traversed by successively placing a 3x3 kernel (as per Rule (I)) over a central location and moving that centre to the location where there is an edge pixel. In moving the 3x3 kernel from ‘a’ to ‘b’, pixel ‘a’ is set to zero as per Rule (II). With this scheme, the coordinates of the points moved to can be stored or a chain code value can be allocated for each step.



(A)

(B)

(C)

Figure 1: Two examples of edges (A) without branching (B), (C) with branching

From example (B) in Figure 1, movement from ‘a’ to ‘b’ can be done in an unequivocal way as the 3x3 neighbourhood placed over ‘a’ has only one direction to move in; towards ‘b’. However, when the 3x3 kernel is placed over pixel ‘b’, two possible directions are obtained: ‘c’ and ‘g’. So pixel ‘b’ should be identified as a branch point as per Rule (III). However, applying Rule (III) directly on pixel ‘a’ of example (C) wrongly assigns it as a branch point. In the same example, point ‘g’ should not be assigned as a branch point as it is similar to point ‘b’ but point ‘h’ should be taken as a branch point.

For this reason, Rule (I) has an analysis component added to it for processing of the 3x3 neighbourhood prior to deciding whether a branch exists or not. The 3x3 neighbourhood of pixel ‘a’ in example (C) consists of two possible paths to move away from ‘a’ (points ‘b’ and ‘c’). Rule (IV) stipulates that priority is given to a horizontal movement in preference to the diagonal movement, so that an incursion is made to ‘b’ first. Doing the opposite would unduly assign a branch point at point ‘c’.

Based on Rule (IV), Rule (V) is set to cater for exceptions in pixel configurations when moving according to the set priority actually causes ambiguity in the branching analysis. Among the 118 possible configurations obtained (see Table 2), two such exceptions were found. They are shown in Figure 2. For these two configurations, moving in the horizontal directions first (to pixels ‘c’ and ‘d’) would create a situation where both branches would then come back to the same point ‘b’. This can be avoided by first moving to point ‘b’ by temporarily setting ‘c’ and ‘d’ to zero, and then from ‘b’, these two respective paths can be traced unambiguously.

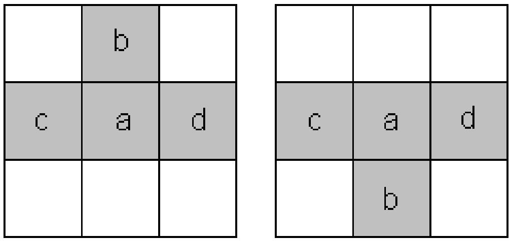


Figure 2: Configurations with higher priority given to vertical movement (edge pixels shaded)

With these two pixel configurations, it can argued why three branches, namely along ‘c’, ‘b’ and ‘d’ are not assigned when the kernel is centred at ‘a’ itself. The reason for not doing this is that it can only be ascertained that these three points are branches by moving away from the central location ‘a’ and surveying the next 3x3 neighbourhood. For example, by moving to ‘b’ first, it might be found that only two branches along {b,c} and {b,d} are present, so this scheme was adopted to yield an optimum number of branches.

Rule (VI) is a common assumption in edge analysis problems and be readily enforced in the computation of edges, e.g. in using Canny detection, in tracing the boundary of an object or in skeletonising a region. Application of Rule (VI) allowed to discard several pixel configurations, which have double pixel width edges. Specifically, the following three categories of pixel configurations were discarded.

(A) Configurations having double pixel width lines, which contravenes Rule VI. Two examples of such configurations are shown in Figure 3.

(B) Configurations in which it is impossible to move to the central pixel without causing a double pixel width line. Two instances of such configurations are shown in Figure 4. In these two configurations, the unshaded pixel locations are necessarily those via which the central pixel location was accessed, which as per Rule II is set to zero. Making any one of these pixel locations as an edge location causes a double pixel width.

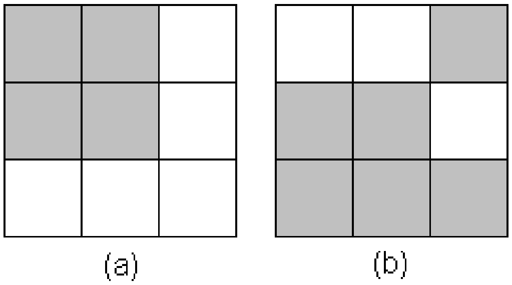


Figure 3: Examples of configurations having double pixel width

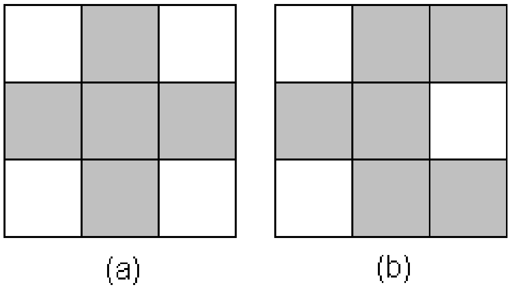


Figure 4: Examples of configurations which cause double pixel widths by creating path to central pixel

(C) Configurations in which it is impossible to access the central location due to the prioritisation of movement as per Rule (IV). Two examples are shown in Figure 5. For example, in case (a), the central location can be reached via pixels ‘a’ or ‘e’ without causing a double pixel width. But due to the higher priority of vertical displacement compared to diagonal displacement, pixel ‘h’ (and ‘d’ for location ‘e’) will be accessed first, thereby changing the 3x3 pixel configuration itself. So it is concluded that such 3x3 pixel configurations will not be encountered.

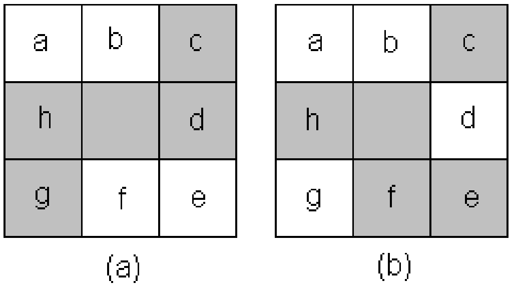


Figure 5: Examples of configurations where central pixel cannot be reached due to step prioritisation

The next section analyses the edge conditioning requirements to yield effective edge representation and branching. The requirement to analyse the 3x3 neighbourhood as given in Rule (I) is thus considered.

3 EDGE CONDITIONING

3.1 Pixel Configurations to be Permanently Removed

Figure 6 shows the different pixel configurations identified for which the central pixels can be set to zero permanently before processing the edges for branching analysis. These configurations essentially remove ‘sharp’ corners in edges. They can be programmed as a look-up table (LUT) and applied over the whole image. Figure 7 shows a binary input image and the corresponding output when such a look-up table is applied. Note that the encircled portion of the image contains a subset of the configuration of kernel 1(a) of Figure 6 but the central pixel is not removed as this will lead to breaking of the line at the branching point. This is catered for in Section 3.2.

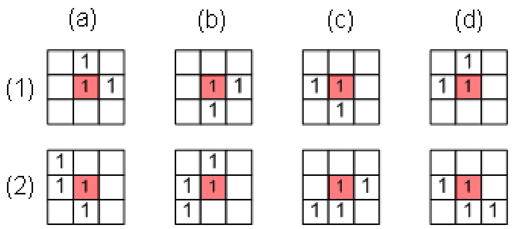


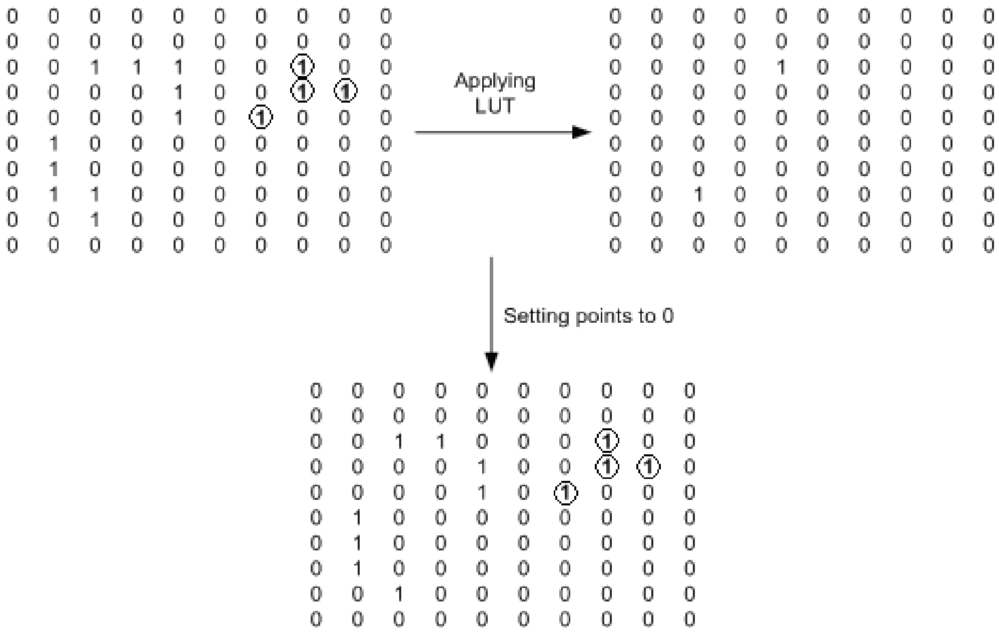
Figure 6: Central pixels are set to zero

3.2 Pixel Configurations to be Temporarily Discounted

After setting the points matching the configurations from Section 3.1 to zero, the edge can be traced to recover its different branches. The encircled part of Figure 7 was an instance of branching. Figure 8 shows a subset of the pixel configurations analysed in Section 2 for which pixel locations need to be temporarily ignored for moving unambiguously along the edge and recovering the branches as per Rule (III).

The first three rows of masks in Figure 8 correspond to situations where no branch exist on the central pixel, so setting the shaded pixels to zero temporarily enables moving to a point in a unique way according to the priority set by Rule (IV). However, the pixel moved to in the next stage is most likely a branch point as it would otherwise have been removed by the prior processing stage to remove ‘sharp’ corners by the global LUT approach described in the previous section.

The next two rows of masks are branching situations as can be seen from the immediate masks themselves. Setting the shaded pixels to zero temporarily allows to move to the non-central pixel in a unique way as per Rule (IV), and thereafter a branch point is identified and the two separate branches are traced. Some observations are relevant at this point regarding the 8 masks in the last two rows of Figure 8 {(4)a-d and (5)a-d} and the four masks in the last row of Figure 6 {(2)a-d} as they belong to the same family. First, there are only four masks in Figure 6 since the other four masks completing the configuration family of kernels would simply break off the configuration into two segments if they are all applied. Second, the masks may look similar, but their appearance in Figure 8 means that their neighbourhoods were not similar prior to moving to the present state, since as per Rule (II), the central pixel is set to zero.



$$
\begin{array}{cccccccccc}
0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0\\
0&0&1&1&1&0&0&\textcircled{1}&0&0\\
0&0&0&0&1&0&0&\textcircled{1}&\textcircled{1}&0\\
0&0&0&0&1&0&\textcircled{1}&0&0&0\\
0&1&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0\\
0&1&1&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0
\end{array}
\xrightarrow{\text{Applying LUT}}
\begin{array}{cccccccccc}
0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&1&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0
\end{array}
$$

$$
\xrightarrow{\text{Setting points to 0}}
\begin{array}{cccccccccc}
0&0&0&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0\\
0&0&1&1&0&0&0&\textcircled{1}&0&0\\
0&0&0&0&1&0&0&\textcircled{1}&\textcircled{1}&0\\
0&0&0&0&1&0&\textcircled{1}&0&0&0\\
0&1&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0\\
0&1&0&0&0&0&0&0&0&0\\
0&0&1&0&0&0&0&0&0&0\\
0&0&0&0&0&0&0&0&0&0
\end{array}
$$

Figure 7: Example with configurations from Figure 6

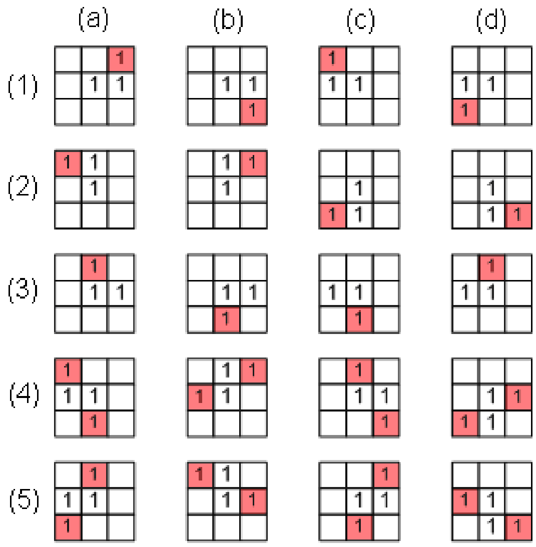


Figure 8: Subset of Pixel Configurations showing pixels to be temporarily zeroed (in red)

For the 118 possible configurations identified (see Table 2), logical expressions were derived for setting the temporary state of each of the 8 border pixels during branching. This is actually the kernel analysis stipulated in Rule (I) and would enable the subsequent application of Rule (III) to identify branching points. Logical variables were assigned to each of the 8 pixel locations and given a true state for configurations where they need to be set to zero (see Figure 9).

For example, if ‘a’ is given a true state, then ‘A’ should be set to 0 and if ‘f’ is given a true state, ‘F’ should be set to 0. The pixel configurations deemed impossible were set as ‘don’t cares’ in the truth table for each output variable.

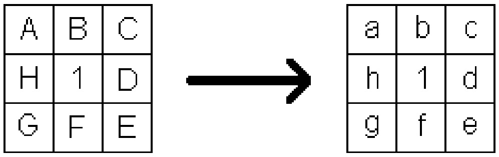


Figure 9: Logical Mask computed from 3x3 mask. A to H are edge pixels and a-h are the output variables.

The logical expressions obtained are:

$$a = AB + AH$$

$$b = BD\overline{H} + B\overline{D}H$$

$$c = BC + CD$$

$$d = BDH + DFH$$

$$e = \overline{A}EF + \overline{C}EF + DE$$

$$f = \overline{D}FH + DF\overline{H}$$

$$g = ACEF + FG + GH$$

$$h = BDH + DFH$$

4 BRANCHING ANALYSIS AND EDGE CHARACTERISATION

The objective in analysing branching in these lines is to develop a method for uniquely and predictably tracing the edge and returning the chain codes along the different branches traced. Although not presented in this paper, the recovery of the different branches along an edge can be followed by further analysis to identify which branches are relevant for analysis for a given application, e.g. the longest branch.

Furthermore, the directional attributes given by the gross chain code representations presented later in this section can be used to effectively select branches along certain directions. The algorithm developed is based on two functions, one that records the chain code values obtained along an edge up to a branch point, and returns the chain code obtained till that point as well as the branch points.

The second function tracks the paths followed and the corresponding chain code sets, passing the first function successively the branch point it returned previously. This is done until all the branches are covered and the output is a history of the different path numbers followed. Paths are numbered as shown in Figure 10, with 1 given to the starting segment.

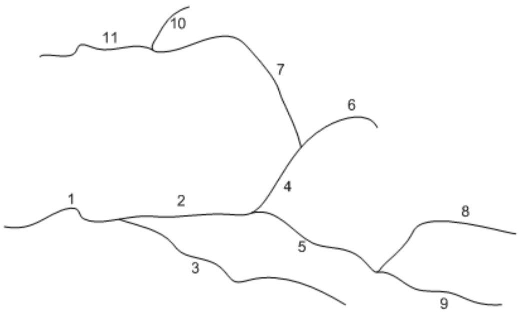
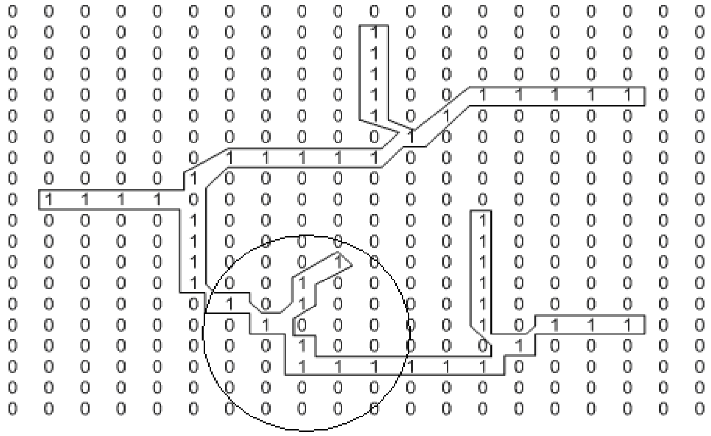


Figure 10: Branching Analysis Example

Upon reaching a branch point, the resulting branches are given labels 2 and 3, thereafter path 2 is followed and upon arriving at a branch, the branch points are labelled as 4 and 5 and path 3 is processed next, returning to paths 4 and 5 subsequently. In this way, the paths traced by the algorithm are: (1 3), (1 2 5 8), (1 2 5 9), (1 2 4 6), (1 2 4 7 10) and (1 2 4 7 11).

Figure 11 shows an example of applying this branching analysis. As seen, there are five paths covered, and hence the output has five chain code sets. In this particular case, the branching algorithm has been applied directly on this edge without any pre-processing to remove ‘sharp’ corners. To illustrate the application of the logical expressions obtained and the rules set formulated earlier, the neighbourhoods of all the edge points and the resulting logical output for the section of the edge encircled in Figure 11 are shown in Figure 12.

The arrows show the progression of the edge tracing process, which as per Rule (I) involves placing a 3x3 kernel over the edge pixel in question and analysing its neighbourhood. These 3x3 kernels are numbered from (i) to (ix) in Figure 12. Analysis of the 3x3 kernels using the previously derived logical expressions yields masks, numbered from (1) to (9). The 3x3 kernels along the edge (kernels (i) to (ix)) have the previous edge pixel set at zero as per Rule (II).



```
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 0 0 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 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 0 0 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 1 0 0 1 1 1 1 1 0 0
0 0 0 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0
0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0
0 0 0 0 0 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 1 0 0 1 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 1 0 1 0 0 0 0 1 0 0 0 0 0 0
0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 1 1 1 0 0
0 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0
0 0 0 0 0 0 0 0 1 1 1 1 1 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 0
0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
```

Line 1: 0 0 0 1 1 0 0 0 0 1 3 2 2 2 2

Line 2: 0 0 0 1 1 0 0 0 0 1 1 1 0 0 0 0

Line 3: 0 0 0 7 6 6 6 7 7 1 2 1

Line 4: 0 0 0 7 6 6 6 7 7 7 6 0 0 0 0 0 1 3 2 2 2 2 2

Line 5: 0 0 0 7 6 6 6 7 7 7 6 0 0 0 0 0 1 1 0 0

Figure 11: Branching Example

Analysis of the output masks and accordingly setting the corresponding pixels’ states temporarily allow the application of Rule (III) to determine the presence of branching. In this way, analysis of kernel (ii) gives an output mask (2) where all the eight boundary locations are zero so that the original kernel is used itself to test for branching. The presence of more than one pixel in the neighbourhood of the central pixel signifies a branching point, which is shown by the two arrows originating from kernel (ii). All the other kernels have a similar zero output mask except output mask (6) corresponding to kernel (vi), where the location corresponding to the high state needs to be set to zero to advance downwards first according to Rule (V) in preference to moving to moving diagonally.

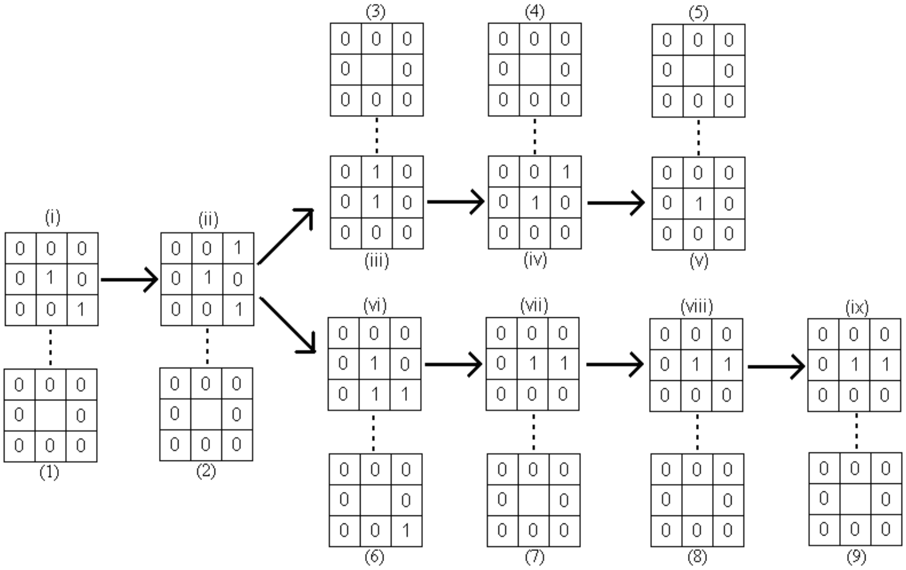


Figure 12: Application of logical expressions during branching

4.1 Characterising Lines Using Chain codes

This section describes the tools developed to compute directional attributes for edges. The approach to developing these tools has been to cluster them according to their main orientation. Two schemes are proposed to pre-process the Freeman’s chain code (Figure 13) set from which segments of the chain code corresponding to a particular gross orientation are segregated.

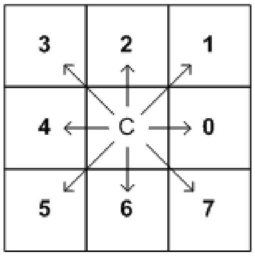


Figure 13: Freeman's Chain code

One of the scheme attempts to find those segments with gross orientations in the North-East, North-West, South-East and South-West directions, while the other scheme finds segments with North, South, East and West orientations. These two tools can be used for decomposing an edge into segments for further processing.

4.1.1 Gross Clustering for Direction of Movement Determination

Due to discretisation of the image formation process into pixels, changes in direction are brought about by combinations of horizontal and vertical displacements coupled with diagonal ones. For example, an edge with a gentle positive slope contains more of {0} chain codes, one with moderate positive slope contains {0}, {1} and {2} chain codes in significant proportions while one with a steep slope contains more of {2} values. This fact is exploited in this section and the next to derive a grosser representation of the chain code to aid in segregating an edge into smaller segments with different gross directions.

Figure 14 shows the scheme used to derive the gross representation of the chain code for finding orientations along the diagonals. A displacement along the {0} direction can signify movement along the {1} or {7} direction, as these are normally combined to bring about a given movement due to discretisation. Similarly, a movement along {2} can mean either movement along {1} or {3} directions, a {4} can be used to bring about displacement along the {3} and {5} directions and finally a {6} may mean incursion into the {5} or {7} directions. The aim here is to find the segments of the line that contain smooth transitions. For example, a {0} may mean displacement along the {1} or {7} directions, but a {0} preceded or followed by a {1} point to a displacement along the {1} direction. The same rule applies for the other orientation dualities mentioned before.

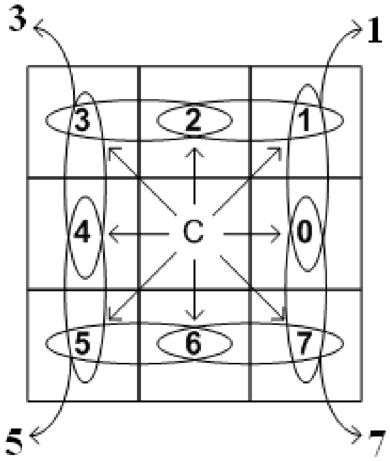


Figure 14: Gross codes for finding diagonal movement

The algorithm proposed to implement such an identification of the segments of a chain code set with similar diagonal movement is as follows:

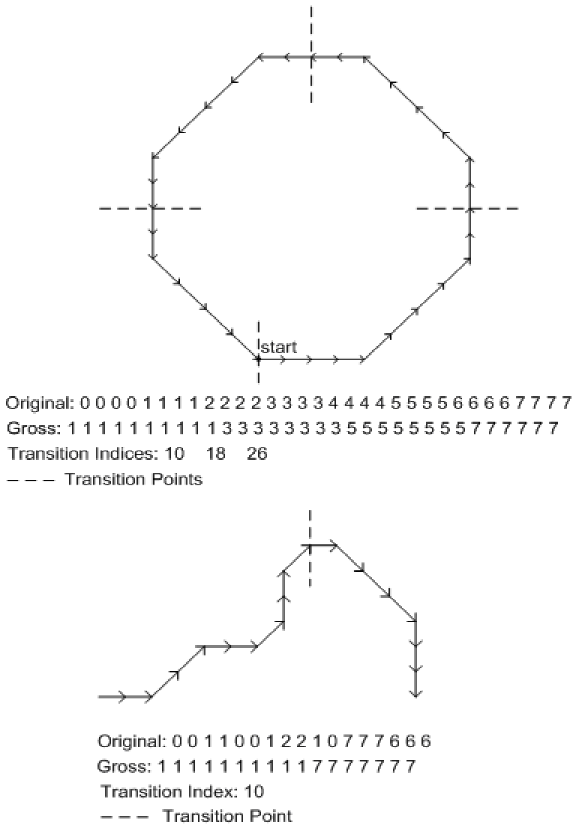
1. The odd chain code values are found.
2. The intermediate even values are given values equal to the odd number nearest to them as shown by the arrows in Figure 14.
3. The indices at the points of transition are found and returned. Referring to these in the original chain code helps to revert back to the initial direction changes.

Figure 15 shows examples of chain code sequences processed using the proposed method. The broken lines show the demarcation between the segments that represent different diagonal displacements. For the first case, all the four possible diagonal movements are present while the second case represents a more realistic example of the type of edge obtained.

4.1.2 Gross Clustering for Finding Vertical and Horizontal Lines

A similar approach as described in section 4.1.1 for finding diagonal segments can be used for finding horizontal and vertical segments in a chain code set. The proposed algorithm is as follows:

1. The even chain code values are found.
2. The intermediate values are found by the closest even chain code values as shown in Figure 16.
3. Recording the indices of transition again allows reverting back to the original chain code set and retrieving the different segments.



Original: 0 0 0 0 1 1 1 1 2 2 2 2 3 3 3 3 4 4 4 4 5 5 5 5 6 6 6 6 7 7 7 7
Gross: 1 1 1 1 1 1 1 1 1 1 3 3 3 3 3 3 3 3 5 5 5 5 5 5 5 5 7 7 7 7 7 7
Transition Indices: 10 18 26
– – – Transition Points

Original: 0 0 1 1 0 0 1 2 2 1 0 7 7 7 6 6 6
Gross: 1 1 1 1 1 1 1 1 1 1 7 7 7 7 7 7 7
Transition Index: 10
– – – Transition Point

Figure 15: Gross chain code representation for diagonal displacement

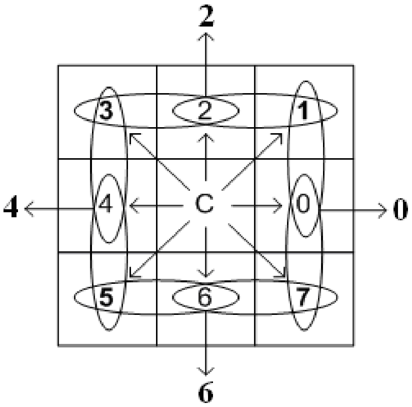


Figure 16: Gross codes for finding vertical and horizontal movements

Figure 17 shows results obtained by applying this algorithm. The broken lines again show the transition between segments, but here the demarcation is that between segments with different horizontal or vertical components. The first example contains all the four types of vertical and horizontal displacements while an arbitrary line segment taken in the second case shows that the algorithm rightly separates the initial horizontal displacement from the subsequent vertical movement and thereafter the right and final downward displacements are correctly singled out as segments. Using the indices returned from the function, it is possible to extract these segments from the original chain code set.

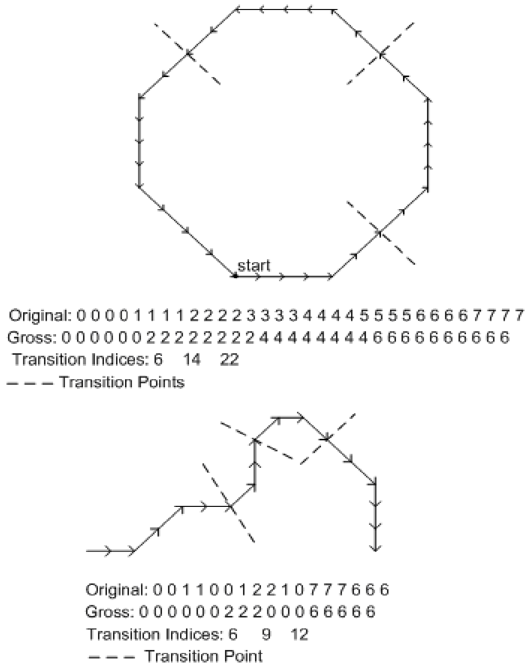

start
Original: 0 0 0 0 1 1 1 1 2 2 2 2 3 3 3 3 4 4 4 4 5 5 5 5 6 6 6 6 7 7 7 7
Gross: 0 0 0 0 0 0 2 2 2 2 2 2 2 2 4 4 4 4 4 4 4 4 6 6 6 6 6 6 6 6 6 6
Transition Indices: 6 14 22
– – – Transition Points
Original: 0 0 1 1 0 0 1 2 2 1 0 7 7 7 6 6 6
Gross: 0 0 0 0 0 0 2 2 2 0 0 0 6 6 6 6 6
Transition Indices: 6 9 12
– – – Transition Point

Figure 17: Chain code representation for horizontal and vertical displacement

5 APPLICATION EXAMPLES

This section first gives an example of the application of the edge characterisation technique described above with a mathematical problem. Two examples of edge processing are then given for a contour obtained from a fluid propagation problem and a line obtained by applying the Canny operator to a face image.

5.1 Curve Fitting

Consider the curves $y = x^2$ and $y = \pm\sqrt{x}$ as depicted in Figure 18 . These two graphs are speculative edges found in an image. While it is customary to use x as the independent variable to approximate a function, clearly using such an approach for the square root edge over the whole curve leads to a non-injective case and hence fitting a general quadratic or cubic function will not be successful. The application of the gross chain code representations for vertical/horizontal directions is demonstrated next using unity intervals representative of the resolution obtained in images at pixel level (although negative coordinates are used). Fitting a polynomial curve to these points gives sub-pixel accuracy. Figure 19 shows spatial discretisations of these two curves, which would be typically obtained in an image array.

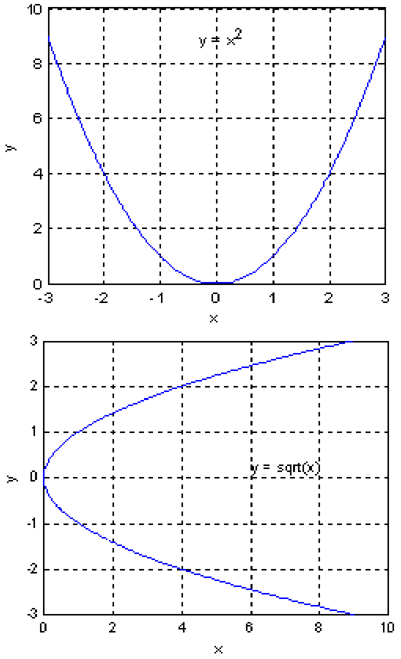


Figure 18: Quadratic and Square root functions

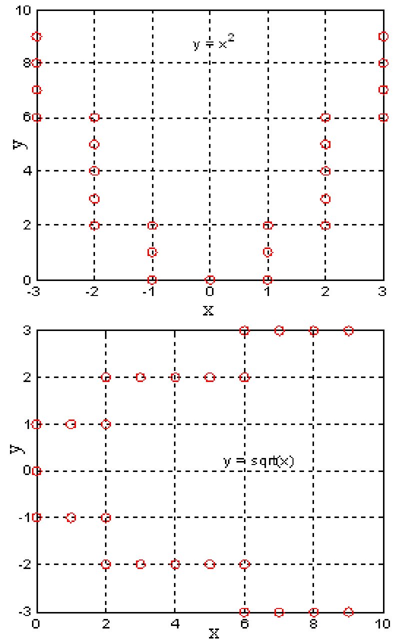


Figure 19: Pixel level representation

Using the same convention that a chain code value of 0 is in the direction of increasing x (and 4 in the direction of decreasing x) and that a chain code value of 2 is in the direction of decreasing y (and 6 in the direction of increasing y), the chain code sequence obtained for these coordinates is:

[2 2 2 0 2 2 2 2 0 2 2 0
0 6 6 0 6 6 6 6 0 6 6 6]

However, applying the edge conditioning described before removes the right angles from the

edge resulting in the following chain code sequence:
[2 2 2 1 2 2 1 2 1 7 6 7
6 6 7 6 6 6]

Applying the vertical/horizontal gross chain code representations gives the following:
[2 2 2 2 2 2 2 2 2 6 6 6
6 6 6 6 6 6]

Clearly, the gross chain codes show that the edge is predominantly vertical, one branch going up (2) and one branch going down (6), hence y can be effectively used as the independent variable for curve fitting for each of the two segments. A similar analysis for the square root data is carried out next. The chain code sequence for these coordinates is:
[4 4 4 6 4 4 4 4 6 4 4 6
6 0 0 6 0 0 0 0 6 0 0 0]

With edge conditioning to remove the right angles, the chain code sequence obtained is:
[4 4 4 5 4 4 5 4 5 7 0 7
0 0 7 0 0 0].

The corresponding gross chain code representation for vertical/horizontal components is:
[4 4 4 4 4 4 4 4 2 0 0 0
0 0 0 0 0 0]

This time, the gross representation shows movement only in the horizontal direction, with a branch going to the right (0) and one going to the left (4). Hence for the square root curve, it would be concluded that the components of the edge are predominantly horizontal, and hence x is to be as the independent variable for curve fitting on these horizontal segments.

5.2 Contour analysis in Fluid Flow

This section describes the application of the branching analysis and gross chain code representations as a component in the processing of fluid contour so as to track the propagation of the fluid by generating flow vectors at the boundary of the contours. This exercise needed an analysis of the fluid contours at consecutive frames. Figure 20(a) shows the fluid contours over two frames, Figure 20(b) and (c) show the separate contours obtained. For a robust analysis of these contours, a branching analysis proved to be necessary, as some pixel configurations were obtained which could cause erroneous interpretations. The result from the branching analysis was typically used to choose the longest branch.

The next step of using the gross chain codes was applied to separate the second fluid contour so that points could be paired from the first contour. By segregating the second fluid contour into horizontal and vertical segments (see Figure 21), sub-pixel accuracy could be obtained in pairing points between the two boundaries, whereas using pixel locations themselves did not give an effective pairing. The segmentation of the contour into parts enable fitting of a low order polynomial to the segments, which lend to either analytical analysis or the generation of sub-pixel data.

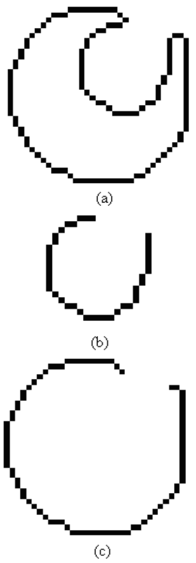


Figure 20: Fluid Fronts over two consecutive frames

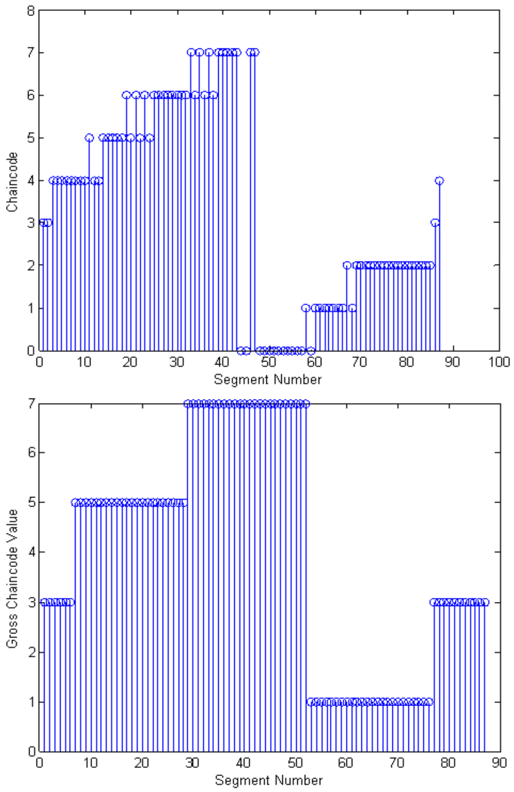


Figure 21: Conventional Chain code (top) and gross chain code (bottom) representation of fluid contour

It is clear that the results obtained from the gross chain code representation yield a better understanding of the general trend of the contour, and provide a more robust means to design algorithms for edge characterisation. Figure 22 shows the different segments obtained by horizontal/vertical gross chain code representation. This version of the gross chain code gives a clear indication of which variable to use as the independent variable for an effective polynomial fitting (a vertical segment would be effectively characterised by fitting a function x = f(y) whereas a horizontal segment would be processed as a function y = f(x)). An explicit example of polynomial fitting is presented next.

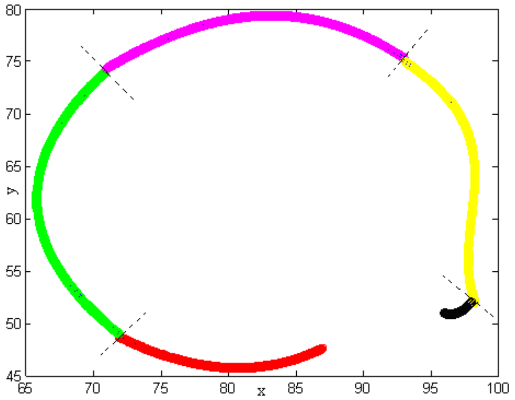


Figure 22: Segments found and fitted with cubic polynomial after horizontal/vertical segmentation

5.3 *Polynomial Fitting of Shape Segments*

The last example considered uses a simple sketch of a car (Figure 23) to show how the proposed gross chain code can be used to first break up a shape into its constituent parts to fit a polynomial to them. Since this is an ideal diagram drawn in a graphics package, branching analysis is not required. However, processing to ensure effective chain coding may be required in this case, especially at the corners.

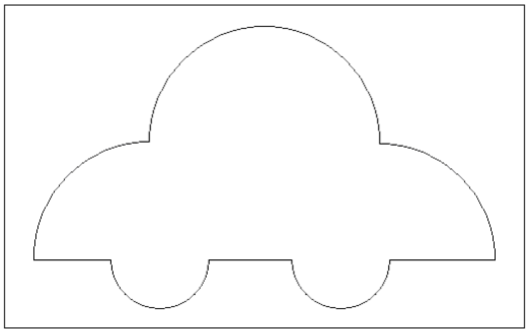


Figure 23: Simple diagram with piecewise lines/curves (image inverted)

The Freeman's chain code for this car outline is shown in Figure 24 (top). The segmentation of this figure into parts is performed by the horizontal/vertical gross chain code as opposed to the diagonal version because the aim is to fit a polynomial to these parts, and as discussed before, the derivation of the horizontal/vertical parts bears directly on the choice of the independent variables. Figure 24 (bottom) thus shows the gross horizontal/vertical chain codes. The clear demarcation of the segments shows that 8 vertical (2 and 6) and 8 horizontal (0 and 4) segments are obtained.

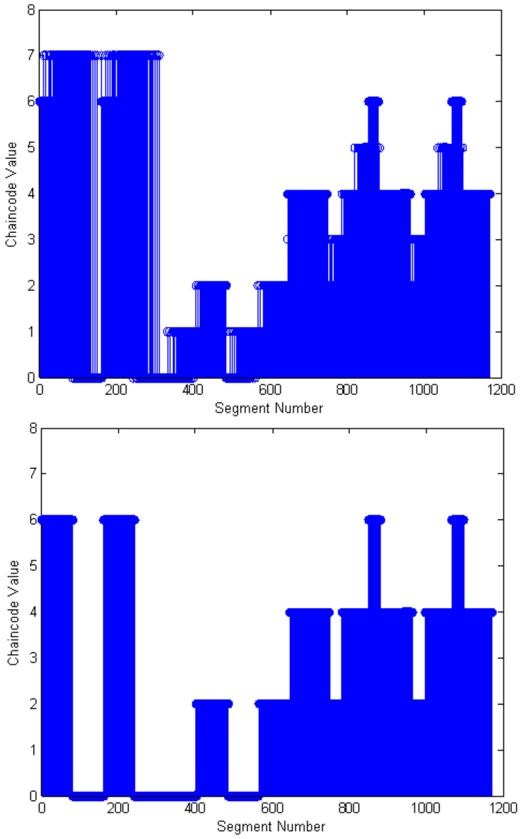


Figure 24: Normal (Top) and Gross (Left) Chain code representation for simple car diagram

Once these segments are derived, polynomials can be fitted to them based on the gross chain code value of a given set of data points. For example, the group having a gross chain code value of 6 (or 2) is fitted with a polynomial where y is the independent variable, while x will be used as the independent variable for the group with a gross chain code value of 0 (or 4). The result of this segmentation and polynomial fitting exercise is shown in Figure 25.

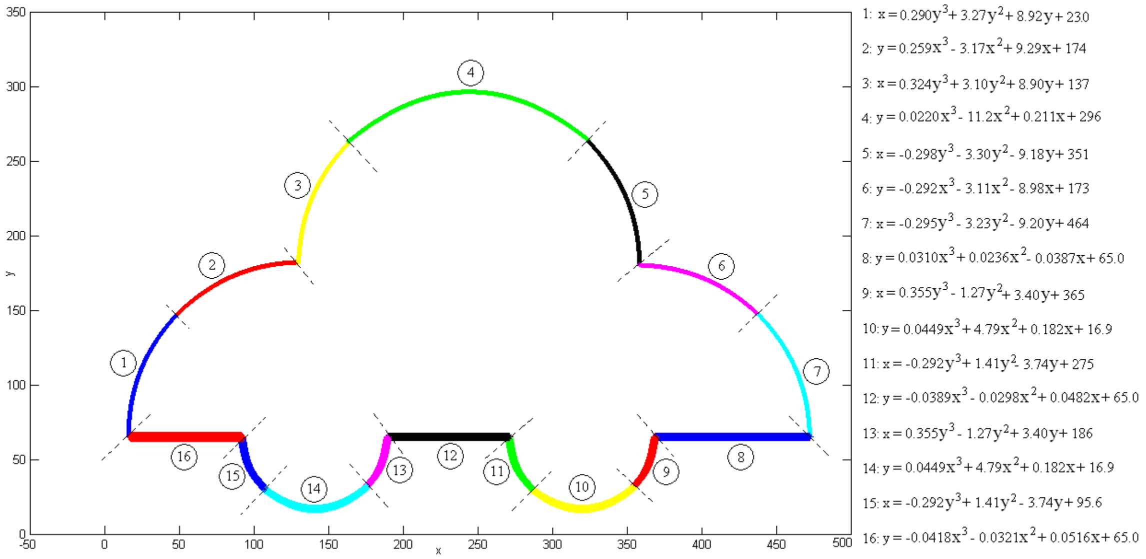


Figure 25: Polynomials obtained from horizontal and vertical segments

5.4 *Edge Analysis in Face Processing*

The example presented in this section relates to edges obtained by processing a face image by the canny operator. Figure 26 shows an edge image obtained. Branches are typically obtained when canny operators are applied for edge detection.

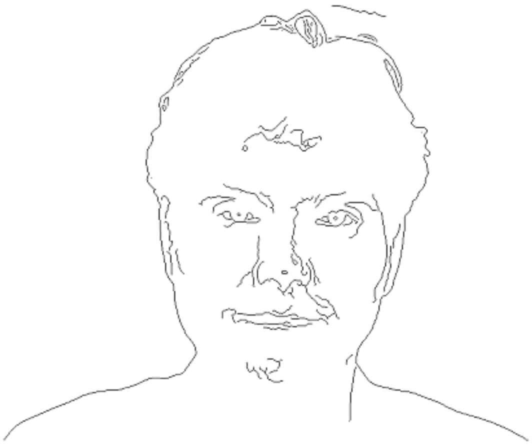


Figure 26: Edges obtained from Canny Operator (image inverted) [10]

Figure 27 shows one of the edges selected on the left side of the person's face with two regions of interests (ROIs) enclosed in squares to show regions where branching was obtained. Figure 28 and Figure 29 show magnified versions of these ROIs. Figure 28(b) shows the result of applying the edge conditioning algorithms described earlier for ensuring an effective chain coding. Branching analysis is applied on this edge image and four branches are obtained, as illustrated in Figure 30(a)-(d).

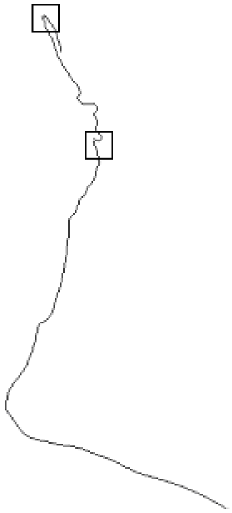


Figure 27: Selected Line for Processing

The Freeman's chain code, the diagonal and horizontal/vertical gross chain codes were computed and the results show the ability of the proposed gross chain code to give a better indication of the direction of an edge. The gross representations would enable to interpret branch (d) as a predominantly downward going edge whereas those for branch (a) would clearly show that there are two main directions of the edge, one going down and one going to the right. This could effectively be used to separate the shoulder and neck regions of the face image.

Additionally, the percentage of horizontal and vertical regions as well as the lengths of these segments can be used as criteria for edge selection from the several edges obtained from Canny operator. The segments of the edge obtained as a result of finding the gross vertical/horizontal chain code representation are shown in Figure 31.

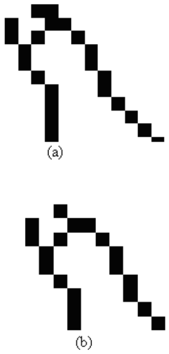


Figure 28: Close-up view of possible branching (a) before (b) after pre-processing

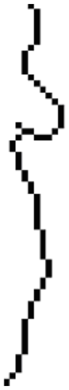


Figure 29: Close-up view of branching

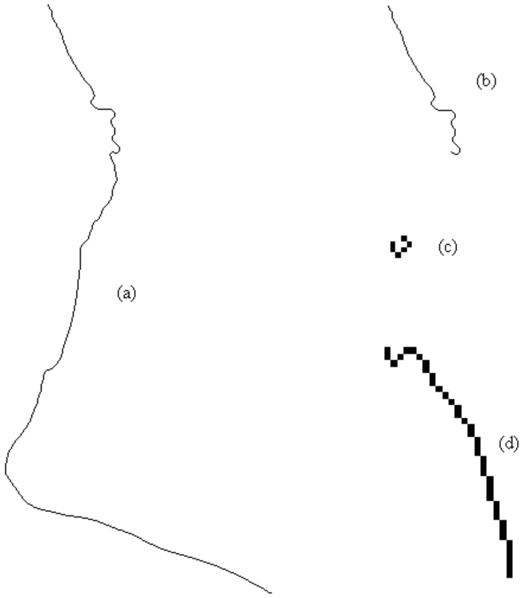


Figure 30: Branches obtained

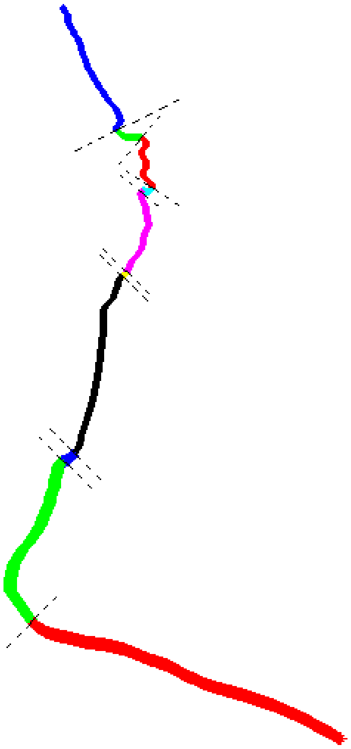


Figure 31: Segments obtained for longest branch with vertical/horizontal gross chain codes

6 DISCUSSION AND CONCLUSION

The image processing and analysis tools proposed in this paper aim at pre-processing edges to remove pixel configurations which both cause sub-optimal representation and more importantly lead to wrongly concluding that a branching point exist in the edge. Such image processing tools were initially developed in the framework of face processing applications, for image understanding specifically related to finding the vertical cheek lines based on their orientations and concavity/convexity. The edges obtained during face processing using Canny operators contained branches and the success of the algorithm relied heavily on detecting these branching points and correctly accounting for them. The whole framework of edge conditioning, branching analysis and gross orientation representation stem from this research work.

In addition, during the synthesis of a method to represent the motion of a fluid patch using flow vectors, this framework proved helpful. The flow vectors indicate by their magnitude and orientation how the fluid contour moved between two consecutive image frames to allow a precise reconstruction of these flow vectors between the fluid boundaries over consecutive frames, curve fitting was deemed necessary to achieve sub-pixel accuracy. The movement of the fluid boundary occurs in complex fashion, such that a single polynomial could not be fitted to the whole fluid contour. The framework enabled to decompose the contour into vertical and horizontal segments, to which appropriate polynomials could be fitted, and thereafter used for finding the flow vectors.

The two gross chain code representations proposed can be effectively used in conjunction to ensure a robust operation. While the horizontal/vertical version was found to be suitable for segmenting a curve into separate horizontal and vertical parts, to which polynomials can be fitted, the diagonal version would allow to guard against certain configurations of pixels which the horizontal/vertical version would not be able to process. For example, a purely diagonal line with chain code [1 1 1 1 1] would not be properly processed by the horizontal/vertical version as there are no even chain code values.

However, used in combination, the results from the two versions can be used to guard against such occurrences and in the event horizontal/vertical version fails and the diagonal version returns an output, it can be implied that either of x or y can be used as the independent variable. The output obtained from the edge segmentation can be further processed to merge adjacent segments, e.g. by checking if they belong to an ellipse [11] or whether they can be merged into clusters [12].

One of the main contribution of this research work presented relates to curve segmentation. Through its simplicity and operation of chain code sequence itself, without any iteration, high execution speeds can be achieved in edge processing applications, e.g. for applications in product line inspection. The application of the proposed gross chain code method to polynomial fitting was the theme of the examples presented.

The break points obtained from the algorithm can be effectively used in conjunction with conventional line/arc fitting techniques in the search for such primitives. The segments obtained from the proposed curve segmentation approach would represent an effective starting point for such paradigms. Used in conjunction with the branching analysis method presented, robust edge processing and characterisation functionalities can be designed.

References

1. Freeman, H., On the encoding of arbitrary geometric configurations, IRE Trans. Electron. Comput 10 (1961) 260–268.
2. Bribiesca, E., A geometric structure for two-dimensional shapes and three-dimensional surfaces, Pattern Recognit 25 (1992) 483-496.
3. Bribiesca, E., A chain code for representing 3D curves, Pattern Recognit 33 (2000) 755-765.
4. Ichoku, C., Deffontaines, B. and Chorowicz, J., Segmentation of digital plane curves: a dynamic focusing approach" Pattern Recog. Lett. 17 (1996) 741-750.
5. West, G. and Rosin, P., Techniques for segmenting image curves into meaningful descriptions, Pattern Recognit 24 (1991) 643-652.
6. Baruch, O. and Loew, M. H., Segmentation of two-dimensional boundaries using the chain code, Pattern Recognit 21 (1988) 581-589.
7. Phillips, T. Y. and Rosenfeld, A., A method of curve partitioning using arc-chord distance, Pattern Recog. Lett. 5 (1987) 285-288.
8. Pham, T. D. and Yan, H., An effective algorithm for the segmentation of digital plane curves—the isoparametric formulation, Pattern Recog. Lett. 19 (1998) 171-176.
9. Arrebola, F. and Sandoval, F., Corner detection and curve segmentation by multiresolution chain-code linking, Pattern Recognit 38 (2005) 1596-1614.
10. Martinez, A. M. and Benavente, R., The AR face database, CVC Technical report 1998.
11. Hahn, K. et al., A new algorithm for ellipse detection by curve segments, Pattern Recognition Letters 29 (2008) 1836-1841.
12. Tai, C. L., Hu, S. M. and Huang, Q. X., Approximate merging of B-spline curves via knot adjustment and constrained optimization, Comput. - Aided Des. 35 (2003) 893-899.