

# Basic Mechanisms of Brownian String Analysis and Manipulation

Krzysztof Bartyzel<sup>1</sup>

<sup>1</sup>*John Paul II Catholic University of Lublin  
Department of Computer Image Analysis  
al. Racławickie 14, 20-950 Lublin, Poland  
kbartyzel@kul.lublin.pl*

**Abstract.** *The paper presents the possibilities of using Brownian Strings as an example of Active Contour Model for linguistic description of objects in an image. Having only a notation composed of a combination of characters RULD we can calculate the statistics of the object, such as its surface area, width and height, centroid, and even image moments. The part is thoroughly discussed. The requirements for a combination of characters RULD to be called a Brownian contour are also presented and analysed carefully*

**Keywords:** *Brownian strings, Chain code, Linguistic pattern matching, Active contours, Image classification.*

## 1. Introduction

Brownian String is an example of the active contour model, have been created from a combination of the Freeman method and the Kass image segmentation technique [1]. The technique was first described by R. P. Grzeszczuk and D.N. Levin in [2]. The paper presents a description of a technique of image segmentation in which an arbitrarily sketched contour was stochastically deformed until a fit to the required shape of the object was achieved.

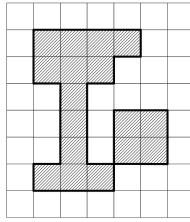



Figure 1: Brownian strings – image segmentation into object and background. 

The issue of deformable shapes has been widely analysed over the past years. There are many publications that expand the original Kass's idea. In [3] authors have proposed method where the energy function of the parametric Active Contour Module was modified by adding texture-based balloon energy and texture features of contour, object, and background points were calculated by Gabor filter bank. A novel approach to texture segmentation based on the parametric active contour model was proposed in [4] where one of the advantages of the proposed approach was that it requires only initial contour selection and no object point selection like the other variants of parametric ACM used for texture segmentation.

The segmentation of objects consists in plotting a closed vectored polygonal curve composed of so-called segments – "crack edges" [5]. The object should be within that curve. Each of the segments separates in the image two neighbouring pixels. Note should be taken of the fact that if each segment separates two pixels, the length of each segment is identical and equals the length on a pixel in the image. An example of such segmentation is presented in Fig. 1

Since segments separate pixels, there are four possible directions of segments separating pixels – from left to right, from right to left, downwards and upwards. They are symbolically referred to as *R* (Right), *L* (Left), *D* (Down) and *U* (Up).

After introducing the concept of segment direction, we can write the object contour as a string code of a polygonal curve composed of vectored segments. Figure 2 presents an object and a contour composed of vectored segments. For such an object the Brownian string code can be:

*DDRDDDLDRRRURRUULLDDL UURURULLLL*

Coding was started in the upper left corner of the object. However, as the contour is represented by a closed curve, the contour of the object in Figure 2 could equally well be described by means of string code as:

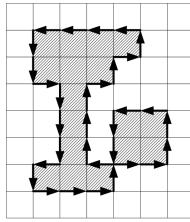


Figure 2: Brownian strings – image segmentation with the use of vectored contour.

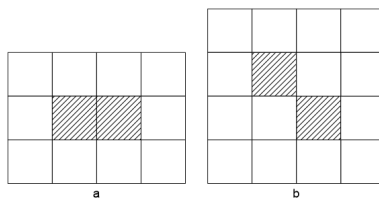


Figure 3: Vicinity of pixels.

*LLLDDRDDDLDRRRURRUULLDDL UURURU*

An additional restriction imposed on the contour by the authors of the described technique is the rule that the contour must be a Jordan curve, i.e. it must divide the plane into two discrete areas: the internal area – the object, and the external area – the background, and that it cannot have multiple points, i.e. the curve can pass through any point in the plane only once.

The second part of the restriction is very important. Two pixels can neighbour with each other in two ways – they either adhere to each other with their sides (Fig. 3a) or they touch with their apices (Fig. 3b).

In the first case there is neither an edge nor an apex of the object between the pixels, so the situation is fairly obvious and does not lead to any ambiguity. Whereas, in the second case, when the only common point of the two pixels are their apices, as many as four segments of the contour meet in the common point. Let us name that point a four-segment apex. Incorrect plotting of the contour at that point may cause that the contour will have a multiple point, which will mean that the contour will lose its property of being a Jordan curve.

The only permitted ways of plotting a contour through four-segment apices are presented in Figure 4. The presented ways are permitted because they can be interpreted as if the contour passing through the four-segment apex bypassed

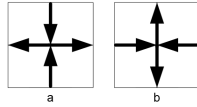


Figure 4: Permitted ways of contour plotting at four-segment apices.

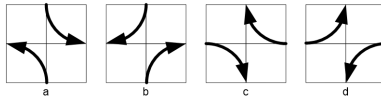


Figure 5: Interpretation of permitted ways of contour plotting through four-segment apices.

the apex and passed within a small distance from it. This means that the curve constituted by the contour has no multiple points, so it is a correctly constructed Jordan curve.

Figure 5 presents the interpretation of permitted ways of contour plotting through four-segment apices. Figures 5a and 5b correspond to the four-segment apices presented in Fig. 4a, while Figures 5c and 5d correspond to those in Fig. 4b. This problem is known as the connectivity paradox and it is fairly often discussed in the literature [6, 7, 8].

## 2. Brownian contour properties

It is easy to note the following properties of Brownian strings:

- the segments form pairs:  $R-L$  and  $D-U$ ;
- the number of segments  $R$  and  $L$  forming the contour is the same, and analogously in the case of segments  $U$  and  $D$ ;
- segment  $X$  can be followed by any other segment except for the segment which forms a pair with segment  $X$ , e.g. segment  $R$  can be followed by  $R$ ,  $D$  or  $U$ , but not by  $L$ ;
- the contour of an object described by means of a Brownian string is a cycle. Depending on the choice of the start element (point of cycle intersection) we obtain different strings corresponding to the same object;

- the form of the string depends also on the direction in which we move along the perimeter of the object.

As it is demonstrated in e.g. [8], the structure of Brownian strings allows their use for linguistic description of objects in images. They can be easily modified, and without having the original image one can achieve a reliable description of the analysed object. Having only a simple notation which is a combination of characters *RULD* we can calculate such statistics as surface area, width and height, centroid, and even image moments of the object. The string code is subject to continuous studies and even though the first paper by M. Kass was published as far back as 1988, new studies and extensions continue to appear, e.g. [9], as well as interpretations and implementations, for example an analysis of the possibility of using the chain code techniques for recognition of hand-written music symbols [10].

### 3. Basic Brownian String manipulation

It is very easy to conduct changes in Brownian Strings but we have to be aware which situations are forbidden. The Brownian String is corrupted if even one of the following conditions occurs:

- the number of segments is odd;
- the number of segments  $R$  differs from that of segments  $L$ ;
- the number of segments  $D$  differs from that of segments  $U$ ;
- two adjacent segments have opposing vectors (i.e.  $RL$ ,  $LR$ ,  $UD$ ,  $DU$ );
- there are two substrings of the contour, starting at apexes situated immediately one after another, such that the number of segments  $R$  and  $L$  and that of segments  $D$  and  $U$  in those substrings are equal.

#### 3.1. Contour reconstruction

In a situation when due to the realization of certain operations on a contour two immediately adjacent segments have opposing vectors it is necessary to correct it. This is the primary task of reconstruction. We should also keep in mind the cyclic

structure of Brownian contours. Incorrect pairs of segments can also be formed by the first and the last elements of a contour.

**Example**

We wish to correct the contour *UURDDDRRLRRULUULLDD*. It is easy to observe that it has pairs of segments with opposing vectors. After performing the sequence of modifications presented below we obtain a corrected contour.

1. UURDDDRRLRRULUULLDD Initial contour
2. UURDDDR**R**LRRULUULLDD Incorrect pair *RL* is identified
3. **U**URDDDRRULUULL**D**D Incorrect pair *DU* is identified
4. **U**RDDDRRULUULL**D**D Incorrect pair *DU* is identified
5. **R**DDDRRULUULL Incorrect pair *LR* is identified
6. DDDRRULUUL Final contour

### 3.2. Determination of contour width and height

When calculating the width and height, it is not necessary to know the contour orientation. The algorithm works correctly for contours clockwise and counter-clockwise oriented.//

Lets start with contour height.

Let  $K$  denote a contour composed of string  $n$  of symbols belonging to set  $R,L,U,D$ . We can then write that:

$$K = k_1k_2k_3...k_n$$

where:

- $k_j \in \{R, L, U, D\}$ ,
- $j \in \{1...n\}$ ,
- $n \in \mathbb{N}$ .

**Step 1:** Based on  $K$  create string  $W_1$  that:

$$W_1 = w_1w_2w_3\dots w_m$$

where:

$$\begin{aligned} w_j &\in \{U, D\}, \\ j &\in \{1\dots m\}, \\ m &\in \mathbb{N}. \end{aligned}$$

In other words, string  $W_1$  is formed from the string  $K$  by removing elements R, L from it and maintaining the other elements order.

**Step 2:** Lets assign to the  $W_1$  elements the following values:

$$\begin{aligned} 1 &\text{ when } w_j = D, \\ -1 &\text{ when } w_j = U. \end{aligned}$$

**Step 3:** We are looking for a substring of  $W_1$ , for which the sum of values, marked  $S_w$ , assigned to particular elements is the largest.

**Step 4:** Because the contour is a closed curve, and in the presented algorithm we analyze strings, in order to calculate the correct height, we move the first element of the sequence  $W_1$  to its end, thus creating a new sequence

$$W_2 = w_2w_3\dots w_mw_1$$

Operations described in steps 3 and 4 must be repeated  $m - 1$  times.

Finally, from among the results obtained in step 3, we search for the highest value. This value is the wanted height of the object described by the contour  $K$ . Figure 6 presents a block diagram of the described algorithm.

When calculating the width, we should proceed in the same way. In step 1, on the basis of  $K$ , we create the string  $S_1$ . The string  $S_1$  formed from the string  $K$  by removing elements U, D from it and maintaining the other elements order. The rest of the algorithm remains unchanged.

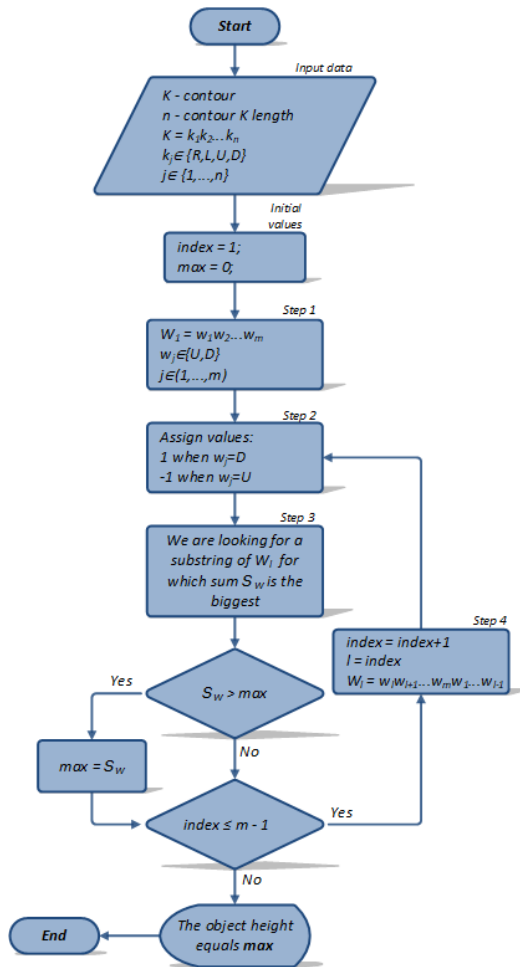


Figure 6: Block diagram describing the determination of contour width.



### 3.3. Determination of object surface area

Surface area is an important feature of an object. In the case of an object in an image, the determination of its surface area is not a major problem, as it is sufficient to count all pixels of the image, classified as belonging to the object. In the case of contours described by means of Brownian strings the problem is more difficult. There is no simple method allowing to determine whether a pixel with specific coordinates is inside an object or outside of it.

The algorithm presented below for surface area calculation is based on the observation that an object with rectangular shape has a characteristic contour form, that is:

$$'D'*height+'R'*width+'U'*height+'L'*width$$

Obviously, the surface area of such an object equals:

$$width*height$$

For an object constructed in this way, in its contour we can identify at most the following two-element substrings:

- straight sections:  $DD, RR, UU, LL$ ;
- 'bends':  $DR, RU, UL, LD$ .

Now it is enough to modify the contour of any object so that it "fits" the above characteristic model of rectangular objects. The modification consists in "filling gaps", i.e. adding individual pixels in suitable places in the object. When adding pixels, we cannot increase neither the width nor the height of the original object. This operation is continued until the contour of the modified object corresponds to the model contour of rectangular objects. The surface area of the original object will equal:

$$width*height-number\ of\ modifications$$

The areas that need to be filled in for the needs of this algorithm will be called gaps. We can distinguish four kinds of gaps:  $RD, UR, LU, DL$ . Figure 7 presents a sample object with two marked gaps.

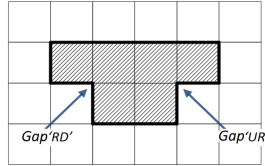


Figure 7: Sample object with marked gaps, the contour oriented anticlockwise.

### 3.4. Algorithm of surface area determination

Let  $K$  denote a contour oriented anticlockwise, composed of string  $n$  of symbols belonging to set  $R,L,U,D$ . We can then write that:

$$K = k_1 k_2 k_3 \dots k_n$$

where:

$$\begin{aligned} k_j &\in \{R, L, U, D\}, \\ j &\in \{1 \dots n\}, \\ n &\in \mathbb{N}. \end{aligned}$$

**Step 1:** On the basis of  $K$  we calculate the height and width of the object. The algorithm for the calculation of those values has been presented in the preceding subsection.

**Step 2:** In string  $K$  we search for gaps  $RD$ ,  $UR$ ,  $LU$ ,  $DL$ . As mentioned before, they indicate places in which we need to insert individual pixels. Insertion of a single pixel in front of  $RD$  can be interpreted as preceding that gap with string  $DRUL$ . This produces  $DRUL RD$ . Unfortunately, such a modification may result in an unacceptable situation, i.e. the neighbourhood of two opposing elements ( $L$  and  $R$  or  $D$  and  $U$ ). Those incorrect pair of elements must be eliminated. In the situation under discussion those will be element pairs ' $LR$ ' and ' $UD$ ' which will be formed after the elimination of the first error. Ultimately, we obtain string ' $DR$ '. Summing up, the insertion of a single pixel in the place of a gap causes the substitution of gaps  $RD$ ,  $UR$ ,  $LU$ ,  $DL$  with  $DR$ ,  $RU$ ,  $UL$ ,  $LD$  respectively. As we can see, the elements which replace the gaps are allowable two-element substrings of rectangular shaped objects.

**Step 3:** The modifications made in the previous step may lead to interference with the contour properties through the appearance of opposing elements next to each other. That may happen in the immediate vicinity of a detected gap. The contour must then be reconstructed in conformance with the algorithm presented earlier.

**Step 4:** The moment when no gaps are detected in string  $K$ , we must perform a cyclic translation of the contour by one half of its length. This is necessary, as the contour is a closed curve, and in the adopted representation of the contour a situation may occur when gaps appear precisely at the point where the start and the end of string  $K$  meet.

Figure 8 presents a block diagram of the described algorithm.

### 3.5. Determination of contour vector

The determination of contour vector is very important, as many methods require the contour orientation to be anticlockwise, and only then their operation is correct. An example of such an algorithm can be the algorithm of surface area determination.

The possibility of determination of contour vector is based on the same observation that was necessary for the calculation of surface area. In objects oriented anticlockwise the pairs of elements  $RD, UR, LU, DL$  are substituted by  $DR, RU, UL, LD$ . This results from the observation that in contours of rectangular objects the only allowable bends are of type  $DR, RU, UL, LD$ . Whereas, if a contour is oriented clockwise the substitution of elements  $RD, UR, LU, DL$  by  $DR, RU, UL, LD$  will initially cause that the object will shrink, and its surface area will get reduced until ultimately it will disappear.

It suffices to note, as was the case with surface area determination, that in the contour of a rectangular object oriented clockwise we will only be able to distinguish the following two-element substrings:

- straight sections:  $DD, RR, UU, LL$ ;
- 'bends':  $RD, UR, LU, DL$ .

For example, for the object presented in Figure 9 and oriented anticlockwise we would obtain a contour in the form:

$DRDRRURULLLL$

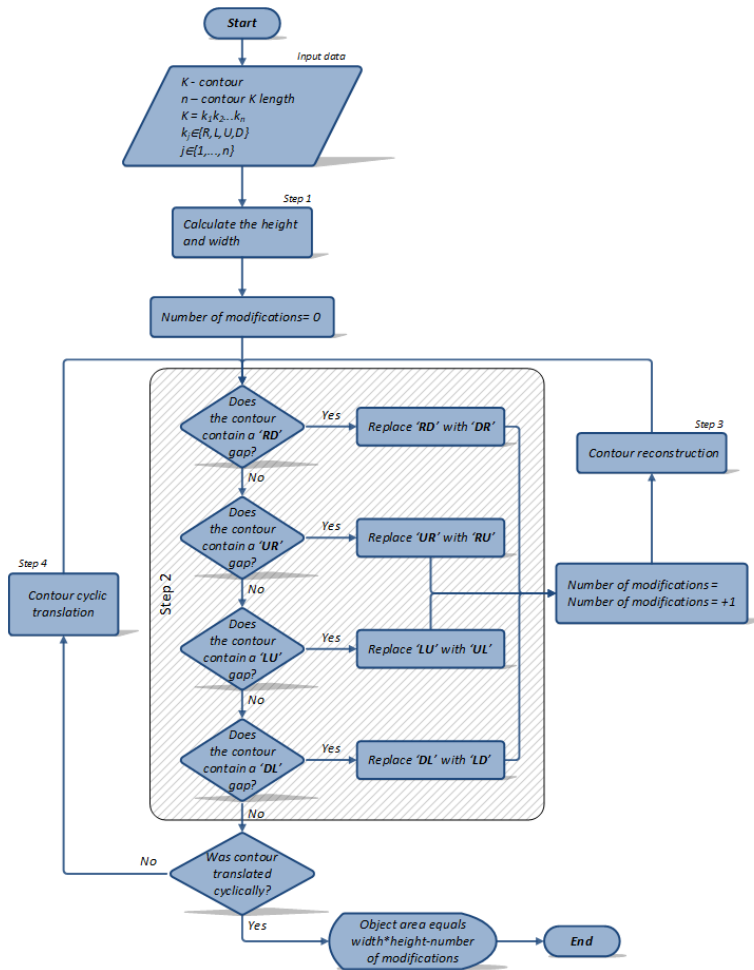


Figure 8: Block diagram describing the determination of contour surface area.

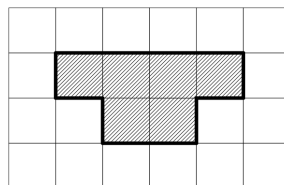


Figure 9: Example of an object.

The contour of the same object but oriented in the opposite direction might have the form:

$$RRRRDLLLULU$$

The above observations, and knowledge of the algorithm of surface area calculation, allow the formulation of the following conclusions:

- If the contour is oriented anticlockwise and the element pairs  $RD$ ,  $UR$ ,  $LU$ ,  $DL$  are substituted by pairs  $DR$ ,  $RU$ ,  $UL$ ,  $LD$  the object will initially expand and ultimately will attain a rectangular shape;
- If the contour is oriented clockwise and the element pairs  $RD$ ,  $UR$ ,  $LU$ ,  $DL$  are substituted by pairs  $DR$ ,  $RU$ ,  $UL$ ,  $LD$  the object will initially shrink and ultimately it will disappear;
- If, for a given contour, we substitute the element pairs  $RD$ ,  $UR$ ,  $LU$ ,  $DL$  with pairs  $DR$ ,  $RU$ ,  $UL$ ,  $LD$  and observe that the contour shrinks, that contour is oriented clockwise;
- If, however, for a given contour, we substitute element pairs  $RD$ ,  $UR$ ,  $LU$ ,  $DL$  with pairs  $DR$ ,  $RU$ ,  $UL$ ,  $LD$  and observe that the contour does not shrink and ultimately attains a rectangular shape, that contour is oriented anticlockwise.

### 3.6. Algorithm of contour vector direction determination

Let  $K$  denote a contour composed of string  $n$  of symbols belonging to set  $R, L, U, D$ . We can then write that:

$$K = k_1 k_2 k_3 \dots k_n$$

where:

$$\begin{aligned} k_j &\in \{R, L, U, D\}, \\ j &\in \{1 \dots n\}, \\ n &\in \mathbb{N}. \end{aligned}$$

**Step 1:** On the basis of  $K$  we calculate the height and width of the object. The algorithm for the calculation of those values is presented in one of the earlier subsections.

**Step 2:** As in the case of surface area calculation, we search string  $K$  for pairs of elements  $RD$ ,  $UR$ ,  $LU$ ,  $DL$ . If such a pair is found, we substitute it with the corresponding pair of elements  $DR$ ,  $RU$ ,  $UL$ ,  $LD$ .

**Step 3:** The modifications performed in the preceding step may interfere with the properties of the contour through the appearance of opposing elements next to each other. This may occur in the immediate vicinity of the detected gap. The contour must then be repaired in conformance with the algorithm presented earlier. If the modification results in a decrease in the size of the object (its height or width), it means that contour  $K$  is oriented clockwise.

**Step 4:** When in string  $K$  we do not find any pairs sought in step 2, we must perform the rotation of the contour by one half of its length. This is necessary, as the contour represents a closed curve and the sought pair of elements may occur at the point joining the start and end of string  $K$ . After performing the rotation we go back to step 2.

Ultimately, if the size of the object (its height and width) has not changed, it means that contour  $K$  is oriented anticlockwise.

Figure 10 presents a block diagram illustrating the operation of the above algorithm.

## 4. Summary

The objective of this study was to develop methods allowing the use of Brownian strings for automatic description and identification of objects in digital images. Although we are dealing with a simple concept, the possibilities of construction and analysis of complex structures are significant. In a similar manner we can also obtain more advanced statistics of objects described with a Brownian string, e.g. their width and height, centroid, and image moments. As it is demonstrated in references [6, 7], there are theoretical foundations permitting the comparison of objects. The use of the Levenstein distance retains all the formal properties of distance, and as such can be successfully applied for comparing the contours of objects. This means that Brownian strings can be efficiently applied not only for linguistic description of objects, but also permit credible and reliable comparison of objects.

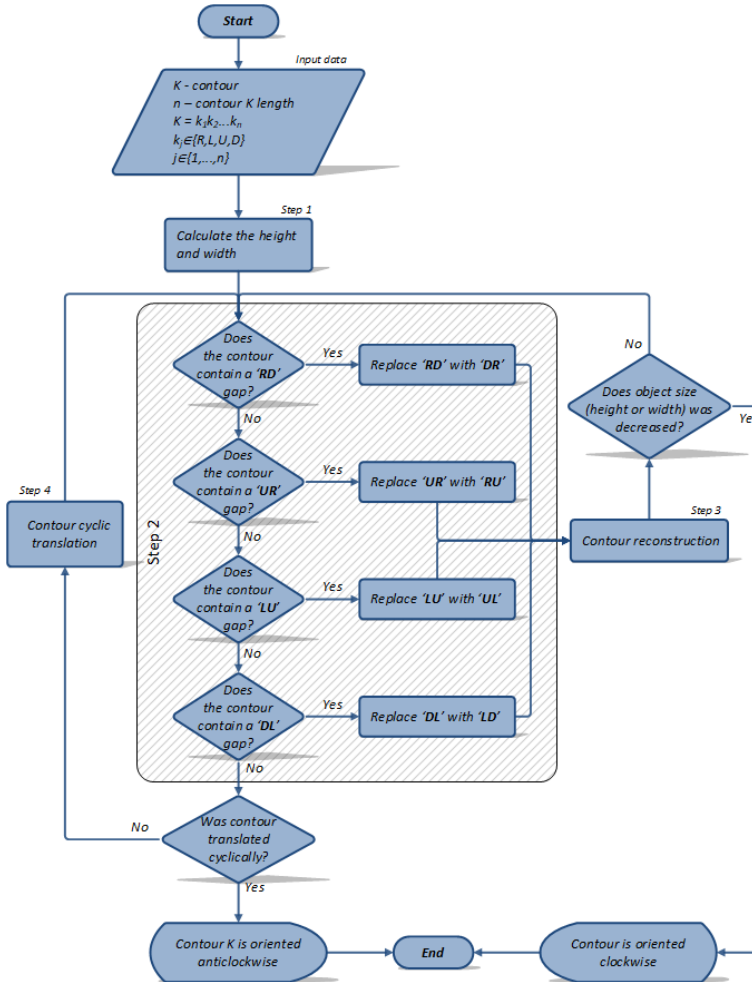


Figure 10: Block diagram describing the determination of vector direction of a given contour.

## References

- [1] Kass, M., Witkin, W., and Terzopoulos, D., *Snakes: Active Contour Models*, International Journal of Computer Vision, 1988.
- [2] Grzeszczuk, R. P. and Levin, D. N., "*Brownian strings*": *Segmenting images with stochastically deformable contours*, IEEE Transactions on Pattern Analysis and Machine Intelligence, 1997, pp. 1100–1114.
- [3] Moallem, P., Tahvilian, H., and Monadjemi, S., *Parametric active contour model using Gabor balloon energy for texture segmentation*, Signal, Image and Video Processing, vol. 10, 2015, pp. 351–358.
- [4] Subudhi, P. and Mukhopadhyay, S., *A novel texture segmentation method based on co-occurrence energy-driven parametric active contour model*, Signal, Image and Video Processing, 2017, pp. 1–8.
- [5] Grzeszczuk, R. P., Eddins, S., and DeFanti, T., *A Region Fill Algorithm Based on Crack Contour Boundaries*, Univ. of Illinois Technical Report UIC-EECS-92-6, 1992.
- [6] Bartyzel, K., *Invariant Levenstein Distance for Comparison of Brownian Strings*, Journal of Applied Computer Science, vol 18, 2010, pp. 7–17.
- [7] Bartyzel, K., *Invariant Levenstein Distance as an example of the Hausdor Distance*, Journal of Applied Computer Science, vol 22, 2014, pp. 7–17.
- [8] Szczepaniak, P. S., *Brownian strings: A flexible linguistic description of images*, International Journal of Knowledge-based and Intelligent Engineering Systems, vol. 14, 2010, pp. 153–158.
- [9] Hasan, H., Haron, H., Hashim, S., and Syakirin, O. F., *Logical Heuristic Algorithm in Extracting 2D Structure Thinned Binary Image into Freeman Chain Code (FCC)*, Visual Informatics: Bridging Research and Practice. IVIC 2009. Lecture Notes in Computer Science, vol 5857, 2009.
- [10] Oh, J., Son, S., and Lee, S., *Online recognition of handwritten music symbols*, International Journal on Document Analysis and Recognition (IJ DAR), vol. 20, 2017, pp. 79–89.