IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 6, June 2026

DOI: 10.17148/IJARCCE.2026.15658

Algorithm for Traversing a Branched Contour in a Digital Image

Nikoloz Nargizashvili[1], Otar Tavdishvili[2]

Department of Applied Pattern Recognition Systems, Vl. Chavchanidze Institute of Cybernetics of the Georgian Technical University, Tbilisi, Georgia[1]

Department of Applied Pattern Recognition Systems, Vl. Chavchanidze Institute of Cybernetics of the Georgian Technical University, Tbilisi, Georgia[2]

Abstract: The retrieval of the objects extracted through digital image segmentation from image databases, as well as their classification based on shape, currently constitutes one of the most important research topics in the field of digital image analysis. Efficient retrieval of similar shapes from image databases requires highly accurate shape descriptions. At the same time, shape representations must remain invariant to translation, rotation, and scaling transformations. This paper considers the problem of contour-based shape description. In this article, an original contour tracing algorithm which is capable of handling both non-branching and branching contours is presented.

Keywords: Digital image, Shape description, Branch, Non-branch, Contour tracing algorithm.

I. INTRODUCTION

The retrieval of objects from digital image databases and their classification based on shape constitutes one of the most important areas of research in the field of digital image analysis. Applications of shape analysis include meteorology, medicine, space exploration, manufacturing, agriculture, biology, entertainment, education, law enforcement, and security. Shape representation and description methods must facilitate efficient storage, long-distance transmission, comparison with other shapes, and accurate recognition. Furthermore, effective retrieval of similar shapes from digital image databases requires highly accurate shape descriptions. For example, in medical image analysis, the shape of internal human organs plays an important role in disease diagnosis. Similarly, in astronomy, the shape of an object is essential for the classification of galaxies, stars, and planets, which constitutes a necessary step in studying and understanding the structure of the universe.

Following the extraction of an object from a digital image (typically as a result of image segmentation and binarization) image analysis tasks often require a description of the object shape, for which shape descriptors are employed.

This paper considers the problem of contour-based shape description. In this approach, the object is represented as a sequence of coordinate pairs corresponding to the contour pixels. Alternatively, characteristic points located on the contour may also be used for representation, such as points of high curvature or inflection points. The selection of such points may be performed using various methods, including approaches based on centroid distance, tangent angle, and other geometric properties.

Contour tracing algorithms are used to determine the ordered sequence of contour pixels and their corresponding coordinates. Furthermore, a consistent contour representation should ensure highly accurate contour reconstruction.

Several well-known algorithms are used for contour tracing, ranging from simple, pixel-based tracing to complex parallelization techniques [1,2,3]. Some of these include:

- **Suzuki-Abe's algorithm:** This method can efficiently trace hierarchical contours, where other contours may exist inside or outside the contour.
- **Moore's algorithm:** A simple well-known algorithm that traverses a contour by examining the eight neighbors of each pixel. It traverses only the outer part of the contour and skips the inner branches.
- **GPU (graphics processing unit)-based parallel boundary traversal algorithms:** They use the parallelism based on the GPU structure to perform real-time contour traversal at high speed.

The algorithm proposed in this article, unlike the methods mentioned, is designed to work with any type of contour. In addition, unlike some other alternatives, it is guaranteed to traverse each pixel of the contour exactly once, has a fairly flexible mechanism for choosing the direction of traversal, and has the ability to be parallelized to a certain extent.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 6, June 2026

DOI: 10.17148/IJARCCE.2026.15658

A. Definition of Branches and Junctions in the Algorithm

In real-world digital images, partial occlusion between objects is a common occurrence. Under such conditions, object contours often exhibit structural irregularities, including junction points where the contour of another object (or multiple objects) originates, as well as branching structures. To enable rigorous contour analysis, formal definitions of branches and junctions are introduced.

Before presenting the proposed algorithm, it is necessary to mention how these concepts are defined for this algorithm:

- **Branch** is defined as a directed continuation of traversal from a contour pixel A to one of its neighboring contour pixels B. Such a continuation is considered valid if traversal may proceed from A to B, or if A has previously been reached from B during contour tracing. It should be noted that not every neighboring contour pixel of A constitutes a valid branch. A neighboring contour pixel B is considered a valid branch of A if and only if there does not exist a third contour pixel that is simultaneously a common neighbor of both A and B, and orthogonally adjacent to both pixels.
- **Junction** is defined as a contour pixel from which at least three distinct branches originate. More formally, a junction consists of one incoming branch (corresponding to the traversal path by which the pixel is reached) and at least two outgoing branches along which traversal may continue.

Consider the example shown in Fig. 1. In this image, only contour pixels X and Y are classified as junctions. Pixel Y has three branches: X, B, and D. Similarly, pixel X has three branches: A, C, and Y.

Pixel D is not considered a branch of X, since X and D share a common neighboring contour pixel Y, which is orthogonally adjacent to both. Additionally, although pixel D has three neighboring contour pixels (X, Y, and B), only Y is regarded as its branch. This is because both B and X share the orthogonally adjacent common neighbor Y with D. Consequently, pixel D has only one valid branch and is therefore not classified as a junction.

The advantage of introducing these definitions of branches and junctions is that they prevent ambiguities during contour traversal. Otherwise, if two neighboring junction pixels (X and Y) share a common neighbor (D), the algorithm could potentially traverse the same pixel twice: once by moving directly from X to D, and again by traversing from X to Y and subsequently from Y to D.

Although this issue could be resolved through the introduction of additional data structures, the proposed definitions provide a significantly simpler and more elegant solution.

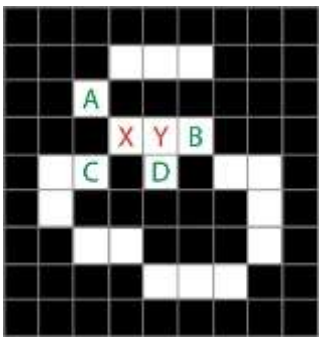

Fig. 1 Branches and junctions

B. Abbreviations and Acronyms

To formally describe the contour traversal algorithm, the following notation is introduced:

1. Let S denote the starting contour pixel from which contour tracing begins.
2. Let C denote the current pixel, i.e., the contour pixel being processed at a given step of the traversal.
3. Let L denote the previous pixel, i.e., the contour pixel visited immediately before the current pixel C.

II. ALGORITHM DESCRIPTION

This section describes the implementation details of the proposed algorithm. We begin by introducing the classes and data structures utilized in its implementation. In the following discussion, white pixels represent contour pixels, whereas black pixels represent background pixels.

A. Structures and Initialization

- **Point Class:** The `Point` class represents a pixel in the image domain and is defined by its spatial coordinates. It serves as the fundamental element for constructing a contour representation in the form of a graph-like linked structure. Each `Point` object stores its pixel coordinates and maintains a list of neighboring `Point` objects.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 6, June 2026

DOI: 10.17148/IJARCCE.2026.15658

- **Branch Class:** The `Branch` class represents a branch in the contour. It is defined by the first pixel of the branch and its preceding pixel, which corresponds to the junction from which the branch originates.
- **Junction Class:** The `Junction` class represents a junction. It consists of the corresponding pixel (a Point object) and a stack of outgoing branches (Branch objects).
- **Juncture Stack Structure:** Used to store junctions for which not all outgoing branches have yet been traced. Although alternative data structures (e.g., a queue) may also be used, a stack is selected in this example of the implementation to support a depth-first search (DFS) traversal strategy.
- **Juncture Dictionary/Map:** Used to store previously identified junctions, indexed by their spatial coordinates, enabling efficient lookup.

It is essential that the same Junction object reference is stored both in the stack and in the map, rather than creating duplicate instances. This ensures that any modification to a junction object is consistently reflected across all data structures without requiring additional synchronization.

B. Algorithm Step Description

1. Begin searching for the first white pixel starting from the upper-left corner of the image. Examine the first row from left to right. If no white pixel is found in the current row, proceed to the next row and continue the search until the first contour (white) pixel is encountered. Once found, create the corresponding `Point` class object for this pixel (its neighbor list will initially remain empty). Designate this pixel as both S (the starting pixel) and C (the currently processed pixel).
2. Traverse the neighborhood of pixel S in the clockwise direction until the second contour pixel is found. Begin with the right neighbor (Fig. 2). If it is not white, examine the lower-right neighbor, then the middle lower neighbor, and finally the lower-left neighbor. Examination of the remaining neighboring pixels is unnecessary, since all pixels located above the first pixel and to its left within the same row are known to be black: otherwise, another white pixel would have been encountered earlier during Step 1. After locating the second pixel, connect the first and second pixels by adding each to the other neighbor list. Designate the second pixel as C and the first pixel as L.
3. If the currently processed pixel C coincides with the starting pixel S, proceed to Step 10. Otherwise, proceed to Step 4.
4. Determine whether pixel C exists in the map of previously processed junctions. If it does, retrieve the corresponding previously created `Junction` object from the map and continue execution from Step 6. Otherwise, proceed to Step 5.
5. Connect pixels C and L by adding them to each other's neighbor lists. Then determine whether C is a junction: traverse all neighbors of C in the clockwise direction, beginning from L, and determine how many of them constitute valid branches. Connect C to all of its branches by adding the corresponding pixels to each other's neighbor lists if they have not already been connected. If the number of branches is greater than or equal to three, construct the corresponding Junction object for pixel C, add it both to the junction stack and to the map of previously processed junctions, and proceed to Step 6. Otherwise, proceed to Step 8.
6. If pixel C exists in the map of processed junctions, remove pixel L from the branch stack of the corresponding `Junction` object.
7. If the branch stack corresponding to C is empty, proceed to Step 10. Otherwise, designate C as L, remove the first branch from the stack and designate it as C. If additional branches remain in the stack, reinsert the corresponding `Junction` object into the junction stack and proceed to Step 3.
8. For the current pixel C, examine the four orthogonally adjacent neighboring pixels. If any of them are white and differ from L, designate the pixel currently marked as C as L, assign the newly found pixel to C, and return to Step 3. Otherwise, proceed to Step 9.
9. If no white orthogonal neighbor is different from L is found, examine the diagonal neighboring pixels and search for a white pixel different from L that is not orthogonally adjacent to L. If such a pixel is found, designate the pixel currently marked as C as L, assign the discovered diagonal neighbor to C, and return to Step 3. Otherwise, proceed to Step 10.
10. Connect pixels C and L. Then search the junction stack for the first `Junction` object whose branch stack is not empty. If no such object exists, terminate the algorithm and return S as the result. Otherwise, proceed to Step 11.
11. Designate the pixel currently marked as C as L. Remove the first branch from the branch stack of the `Junction` object identified in the previous step and designate it as C. If additional branches remain in the stack after removal, reinsert the corresponding `Junction` object into the junction stack. Proceed to Step 4.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 6, June 2026

DOI: 10.17148/IJARCCE.2026.15658

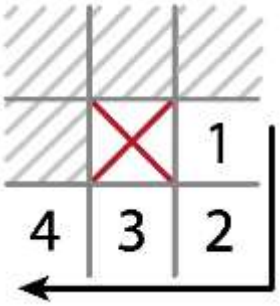

Fig. 2 Clockwise second point selection

C. *Regions with Thickness Greater than One Pixel*

Although skeletonization algorithms generally reduce an object to a contour with a thickness of one pixel, exceptional regions may occur near junction pixels. In such regions, the skeletonization algorithm may determine that removing a particular pixel would compromise the connectivity or structural integrity of the contour. Consequently, regions in which the contour thickness remains equal to two pixels may be preserved.

This situation typically arises when all pixels within such a region are classified as junction pixels. Therefore, it is necessary to account for the possibility of encountering such cases during contour traversal.

The proposed algorithm explicitly handles these situations and ensures that no traversal ambiguities or processing issues occur when the contour is traversed using the described method.

D. Next Pixel Selection Outside Junction Regions

After the first two contour pixels have been identified, it is necessary to continuously maintain the coordinates of L, representing the previously visited white pixel. This allows the algorithm to determine both the direction from which the traversal originated and the direction in which it should continue.

When the current pixel C is not a junction pixel, several properties of the contour can be exploited to reduce the number of candidate pixels considered during the selection of the next contour pixel.

- At any given moment, traversal may continue in at most two directions along the contour, since C is not a junction pixel.
- Among the eight neighbors of C, the number of white pixels is at least two and at most four. Otherwise, either the contour thickness would exceed one pixel in some region, or C would itself be classified as a junction pixel.
- Among the white neighbors of C, at most two may be orthogonally adjacent and at most two may be diagonally adjacent. Otherwise, C would constitute a junction pixel.
- No white neighboring pixel of C may be orthogonally adjacent to more than one other white neighbor. Otherwise, the contour thickness would exceed one pixel within some interval, or C would become a junction pixel.

To determine the next contour pixel, the algorithm examines the eight neighboring pixels of the current white pixel. If the current pixel has two adjacent white neighbors, one of them must necessarily correspond to L. The other neighboring pixel is then selected as the next contour pixel.

Consider the case in which the previously selected white pixel has three or four white neighbors. Fig. 3 illustrates a situation in which the algorithm must choose between two candidate pixels: an orthogonally adjacent neighbor and a diagonally adjacent neighbor. In this case, the orthogonally adjacent white pixel different from L is selected, namely pixel 1. The same principle applies even when L is orthogonally adjacent rather than diagonally adjacent.

This strategy prevents contour pixels from being skipped and generalizes naturally to cases in which an additional fourth white pixel exists to the right or below L (Fig. 4). In such cases, the fourth pixel is guaranteed to correspond to L: since orthogonal neighbors are always examined before diagonal neighbors, traversal from the upper-left pixel to the center pixel could not occur without first passing through this fourth pixel. Consequently, traversal would proceed along the direction indicated by the green arrow rather than the red arrow.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 6, June 2026

DOI: 10.17148/IJARCCE.2026.15658

Fig. 3 Three contour pixel neighbor example

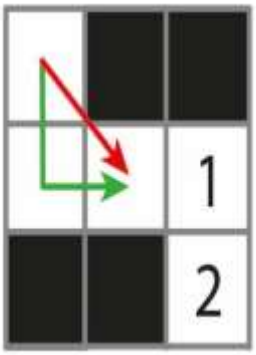

Fig. 4 Ways of reaching the center from top left corner

A special case must also be considered in which the current pixel has three neighboring white pixels, one of which corresponds to the orthogonally adjacent pixel L, while the remaining two are positioned diagonally (Fig. 5).

In this situation, it is evident that pixel 1 must already have been traversed before reaching L. Consequently, pixel 2 should be selected as the next contour pixel. However, without additional condition, the algorithm cannot reliably determine this.

Specifically, the algorithm first determines that the current pixel has no orthogonally adjacent white neighbor other than L and therefore proceeds to examine the diagonal neighbors. Since neither diagonal neighbor corresponds to L, the algorithm cannot uniquely determine which of the two diagonal pixels should be selected as the next contour pixel. To ensure that pixel 2 is selected, it is sufficient to determine which of the two diagonal neighbors is not adjacent to L and continue tracing in that direction.

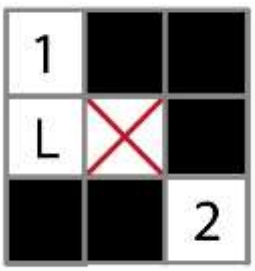

Fig. 5 Three neighbor special cases

E. Algorithm Termination Criteria

When returning to the starting pixel, reaching a dead-end, or encountering an already visited junction pixel, the algorithm checks if there are any junctions present in the junction stack whose branch stack is non-empty and terminates if no such junctions are found.

IJARCCE ISSN (O) 2278-1021, ISSN (P) 2319-5940

International Journal of Advanced Research in Computer and Communication Engineering

Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 6, June 2026

DOI: 10.17148/IJARCCE.2026.15658

III. TEST DATA

The proposed algorithm was tested on a set of manually created images covering simple, complex, and edge cases, including contours with multiple branches, junctions, and non-uniform thickness. These controlled tests were designed to evaluate correctness and robustness under varying conditions. The results show that the algorithm consistently reconstructs contours accurately and handles all cases without loss of connectivity or traversal ambiguity. Fig. 6 displays one of the simpler contour shape test images the algorithm was tested on. The size of the image is 870 x 822 and contains a total contour pixel count of 4002. Fig. 7 displays one of the more complex images the algorithm was tested on. The size of this image is 1366 x 768 and contains a total contour pixel count of 9731.

Fig. 6 A simpler contour test image example

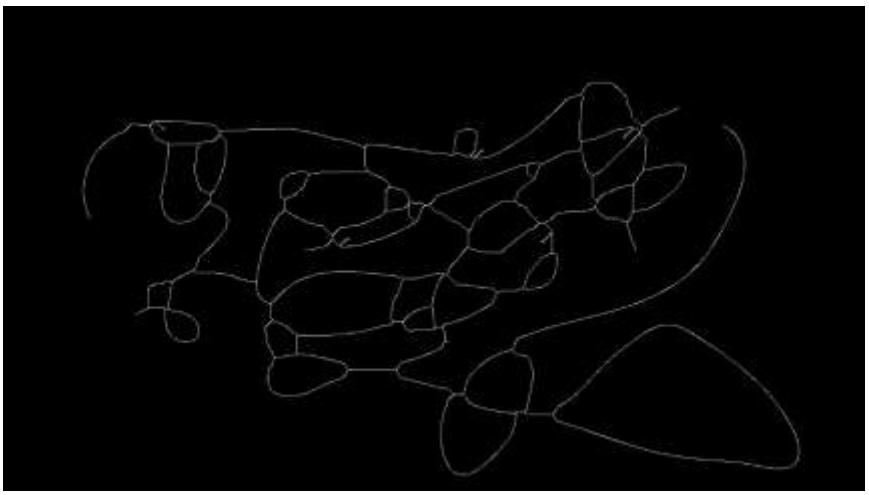

Fig. 7 Complex multi-junction contour test image example

IV. PERFORMANCE ANALYSIS

This section presents a performance analysis of the proposed algorithm in terms of time and space complexity, as well as its structural flexibility. The algorithm is examined to determine its computational efficiency, memory requirements, and ability to handle varying contour configurations.

A. Time Complexity

Contour traversal requires O(c) time, where c denotes the length of the contour (i.e., the total number of contour pixels). However, the algorithm can be partially parallelized to a certain extent by tracing each found branch in a separate thread. The effectiveness of this parallelization depends on the number of junction pixels in the contour and their spatial distribution.

B. Space Complexity

The algorithm requires O(c) memory to store the coordinates of all pixels visited during contour traversal, where c denotes the length of the contour. However, if it is not necessary to retain all visited pixels for the final output—for example, when each contour pixel is processed only once and not needed for later use—the data structure representing pixel connectivity can be omitted from the algorithm. In this case, the memory requirement is reduced to O(j), where j denotes the number of junction pixels in the contour, since only the junction stack and map structures will be used.

International Journal of Advanced Research in Computer and Communication Engineering
Impact Factor 8.471 ⁜ Peer-reviewed & Refereed journal ⁜ Vol. 15, Issue 6, June 2026
DOI: 10.17148/IJARCCE.2026.15658

C. Tracing Flexibility

Three criteria are used to determine the traversal direction of the algorithm: the order in which neighboring pixels are examined, the type of data structure used to store junction pixels (specifically the element insertion and removal logic of the structure), and the type of data structure used to store branches. The ability to modify these three criteria provides a simple and flexible mechanism for selecting the traversal direction.

D. Evaluation Based on Core Contour Tracing Algorithm Criteria

- **Algorithm accuracy:** Accuracy constitutes one of the biggest advantages of the proposed algorithm. No contour pixels are skipped or traversed redundantly; each contour pixel is visited exactly once. The process of locating the starting pixel, as well as the traversal order of the pixels, is fully deterministic, enabling an accurate and consistent representation of the contour structure.
- **Time complexity:** The time complexity of the algorithm is O(c), where c denotes the contour length. This represents the minimum possible traversal time complexity for contour tracing algorithms in the absence of parallelization. Consequently, the proposed method achieves performance comparable to other efficient approaches, such as Moore's algorithm. In addition, the algorithm supports partial parallelization, since each branch may be traversed independently within a separate execution thread. The effectiveness of the parallelization depends on the number of junctions and branches present in the contour - as their number increases, the potential performance gain from parallel execution also increases.
- **Memory consumption:** The algorithm requires O(c) memory if all traversed contour pixels must be stored and returned as the final result. Otherwise, only the junction stack, the map of previously visited junctions, and several auxiliary variables must be maintained, reducing the memory complexity to O(j), where j denotes the number of junction pixels in the contour. Consequently, the proposed algorithm is highly memory-efficient and compares favourably with many alternative approaches that require additional O(c) memory for storing visited pixels instead of O(j). Examples include conventional graph traversal methods based on DFS or BFS principles.
- **Accuracy of contour rescaling and reconstruction:** The algorithm preserves all contour pixels, as well as their traversal order and traversal direction. This facilitates accurate contour rescaling, reconstruction, comparison with other contours, and related shape-processing operations.

V. CONCLUSION

The proposed algorithm is characterized by a flexible mechanism for predefined traversal direction selection, the ability to preserve the ordering of contour pixels—which simplifies contour reconstruction, scaling, and comparison with other contours—low memory consumption and computational complexity, partial parallelization capability, and robust operation on contours of arbitrary structure. In addition, the ability to detect junction pixels during traversal may facilitate contour processing directly during the execution of the algorithm.

The results obtained demonstrate that the proposed method provides an efficient and reliable approach for contour traversal and shape description, making it suitable for further applications in digital image analysis and related research areas.

REFERENCES

[1]. S. Suzuki and K. Abe, "Topological Structural Analysis of Digitized Binary Images by Border Following", *Computer Vision, Graphics and Image Processing*, vol. 30, Issue 1, pp. 32-46, 1985.

[2]. Victor M. Garcia-Molla, Pedro Alonso-Jorda, Ricardo Garcia-Laguia, " Parallel border tracking in binary images using GPUs ", *The Journal of Supercomputing*, vol. 78, pp. 9817-9839, 2022. DOI: https://doi.org/10.1007/s11227-021-04260-y.

[3]. S. Gupta, "Algorithms to Speed Up Contour Tracing in Real Time Image Processing Systems". *IEEE Access*, vol.10, pp. 127365-127376, 2022. DOI: 10.1109/ACCESS.2022.3226943.