A Contour Tracing Algorithm that Avoids Duplicate Tracing Common Boundaries between Regions

Summary

The authors of this paper have developed a series of techniques on automatic function approximation of raster images, including references [3] and [5] that were published in this journal. In our techniques, contours are traced and their shapes are approximated by a combination of functions including straight line, arc, and quadratic curve. Because our previous works targeted mainly bi-level or two-labeled images, conventional contour tracing algorithms were sufficient. However, when these earlier algorithms were applied to images of multiple labels, the problem of duplicate tracing common boundaries between regions occurred, adversely affecting the result of function approximation.

In this paper, we focus on contour tracing and propose an algorithm that can correctly trace all boundaries including those common to neighboring regions without duplication. Our originality lies in the tracing steps that follow after boundary points where three or more labels meet are detected. Every boundary that branches from such points is recorded and traced in succession while avoiding duplicate tracing. As a result, common boundaries can be shared, which is not possible with conventional algorithms.

Keywords
Contour Tracing, Common Boundaries, Duplicate Tracing, Function Approximation
1. Introduction

The popularity of digital devices has greatly increased the chance of manipulating images on computers. In general, image data can be largely divided into raster format and function format. A raster image is composed by a group of pixels arranged in a square tessellation, with each having a certain value or label denoting the color information. This format is suitable for representing images on modern CRT and LCD displays. However, when raster images are subjected to affine transformations like enlargement, reduction, rotation, and translation, the image quality deteriorates visually in the form of jaggy-ness occurring along the region boundaries (Figure 1). Compared to this, the function format is suitable for representing images that exhibit clear region boundaries. Moreover, the function format has the added advantage that the visual quality of images can be maintained even after being subjected to affine transformations.

In our research, a series of techniques on automatic function approximation of raster images have been developed, two of which were published in this journal [1]-[5]. In our techniques, contours are traced and their shapes are approximated by a combination of functions including straight line, arc, and quadratic curve. Because our previous works had targeted bi-level or two-labeled images, conventional contour tracing algorithms that focus on a single region at a time during tracing were sufficient (Figure 2a). In other words, a region enclosed by the boundary being traced is the object of interest while everything else is background. However, when these earlier algorithms were applied to images of multiple labels, the problem of duplicate tracing common boundaries between regions occurred, adversely affecting the result of function approximation. For example, overlaps or gaps might appear in the image reconstructed from the data in function format due to approximating the common boundaries by different functions (Figure 2b). Moreover, the total length of traced contours will very likely increase, causing an increase in the overall processing time.

In this paper, we focus on contour tracing and propose an algorithm that can correctly trace all boundaries including those common to neighboring regions without duplication. Our originality lies in the tracing steps that follow after boundary points where three or more labels meet are detected. Every boundary that branches from such points is recorded and traced in succession while avoiding duplicate tracing. As a result, common boundaries can be shared, which is not possible with conventional algorithms.

The remaining of this paper is organized as follows. In Section 2, related works are discussed which is followed by a detailed description of the proposed contour tracing algorithm. In Section 3, experimental results illustrating the effectiveness of the proposed algorithm will be presented. Finally, Section 4 ends with concluding remarks.

2. Proposed Contour Tracing Algorithm

As mentioned earlier, a raster image is composed of a square tessellation of pixels, with each having a certain value of label that corresponds to the color information. For bi-level images, each pixel can have one of two possible labels such as ‘1’ and ‘0’. For color images, there will be additional labels, and are also known as multi-labeled images in literatures on region segmentation of images [6]. In this paper, we assume that region segmentation in an image has been done before contour tracing commences. The proposed algorithm can trace both 4-connected and 8-connected regions, whose definitions will be given in the following sub-section. As shown in Figure 3 on the flow of processing for function approximation, Step 2 on contour tracing is the focus of this paper.

![Fig. 1. Image quality degrades visually after a raster image is enlarged.](image-url)
2.1 Related Work

There exist many contour tracing algorithms for bi-level or two-labeled images [7]-[14]. However, when applied to images of multiple labels, these algorithms exhibit the problem of duplicate tracing common boundaries between regions as explained earlier. In certain image processing applications, the effects are not obvious. However, in function approximation of raster images, the quality of the reconstructed images is often affected.

A related work on a contour tracing algorithm that preserves common boundaries between regions was introduced in [15]. The author reported that if the conventional boundary definition were adopted in tracing contours of an image of multiple labels, the common boundaries between regions might not “overlap” properly. A conventional boundary is defined as the set of pixels of a region R that have at least one neighbor not in R, in the sense of either 4-connected or 8-connected region. Refer to Fig. 4 for these definitions. To overcome this problem, the algorithm was developed based on an extended boundary definition. An extended boundary of a region R consists of pixels that belong to R along a defined set of traversal directions and those that are in the complement of R. Refer to Fig.5a for an example of an extended boundary defined in [15].

Compared to [15], our proposed algorithm differs in several important aspects. First, the algorithm of [15] was designed to trace boundaries of 4-connected regions only, while ours has no such restriction. Second, for the algorithm of [15] to work, the number of regions has to be known in advance. In our algorithm, such pre-condition is unnecessary.
Because of these differences, a direct comparison between [15] and our algorithm using multi-labeled images found commonly in applications will not be meaningful. As such, our algorithm was compared with a conventional algorithm used in Ref. [3] that was also able to trace contours in multi-labeled images with multiple, disjoint regions of same label.

Moreover, to facilitate representation of common boundaries, in our algorithm the corners of pixels instead of their centers are used as coordinates, which was the case of [15] (Refer to Figure 6). The reason for using corners of pixels as coordinates is that if centers were used, a common boundary would be included in both regions, leading to ambiguity. For a comparison, consider the extended boundary as defined by our algorithm in Fig.5b with Fig.5a.

2.2 Algorithm Description

The proposed algorithm is composed of one major routine and two subroutines, as illustrated by the Nassi-Shneiderman Charts in Figures 7(a)-(c) [16]. For interested readers, C-style pseudo codes of these routines are given in Appendix A. Furthermore, as an important requirement for a practical algorithm, a proof of the termination of the proposed algorithm is briefly sketched and given in Appendix B.

Input to the main routine, “Contour_Tracing”, is a raster image assumed to have segmented to multiple labels. Traversal begins from left to right, top to bottom. That is, the coordinate (0, 0) is at the top left corner, while (m-1, n-1) is at the bottom right corner, with \( m \times n \) being the size of the image.

If a difference between the current pixel and the one to its right is detected, the subroutine “Trace_Boundaries” is called by passing it the current position and traversal direction as input parameters. It is expected to trace a set of common boundaries, and will in turn call subroutine “Trace_One_Boundary” with the current position and trace direction (initially same as traversal direction) as parameters. The result is that a traced boundary and the set of branches (each having a direction) at its end point are returned to “Trace_Boundaries”. These branches will be traced in succession until all the region boundaries of the image are covered.

Throughout the preceding discussion, the next traversal or trace direction is decided by searching for a matching target pattern from a lookup table that involves the current trace position and the current tracking direction. For 4-connected regions only, Table 1 is sufficient. To handle both 4-connected and 8-connected regions that do not have topological ambiguity in the form of leaks across a region, three more patterns as in Table 2 are used. In such cases, the distribution of labels in the neighborhood of an ambiguous region can be used to decide its boundary.

In addition, the subroutine “Compose_Regions” composes a set of closed regions from the list of traced boundaries. Finally, when all the pixels in the image have been traversed, the main “Contour_Tracing” routine returns as result both a list of traced boundaries and a list of closed regions to the next processing step.
Main routine: Contour-tracing

Input an image.

Repeat for all pixels in image

Is the right edge of pixel a part of untraced boundary?

Yes

Call “Trace-boundaries”.

No

Stop (neither of the earlier patterns)

Table 1. Lookup Table

<table>
<thead>
<tr>
<th>Target pattern</th>
<th>Next direction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Right</td>
</tr>
<tr>
<td></td>
<td>Forward</td>
</tr>
<tr>
<td></td>
<td>Left</td>
</tr>
<tr>
<td></td>
<td>Stop</td>
</tr>
</tbody>
</table>

Routine “Trace-one-boundary”

Add current position to “boundary”.

Advance current position in the current direction.

Refer to lookup-table and update current direction.

Does the boundary has branches?

Yes

Return with branches found.

No

Repeat until boundary becomes a loop.

Return with no branch detected.

“Trace-boundaries”

Initial the trace from the given start point.

Is the initial trace looped?

Yes

Return with a region composed by traced boundary.

No

Add all branches on end point to “non-traced” set.

Remove a branch from “non-traced” set and trace it.

Is end-point an undiscovered branch?

Yes

Add all branches around end-point to “non-traced” set.

No

Delete the end-point from “non-traced”.

Add the beginning-point and end-point to “traced” set.

Repeat until “non-traced” set is empty.

Remove a branch from “traced” set and assign it to “current-branch”.

Set “current-region” to “current-branch”.

Remove a branch that is nearest counter-clockwise neighbor of end-point of boundary from the “traced” set and assign it as the new “current-branch”.

Add “current-branch” to the “current-region”.

Repeat until “current-region” becomes closed.

Repeat until “traced” set becomes empty.

Input an image.

Repeat for all pixels in image

Is the right edge of pixel a part of untraced boundary?

Yes

Call “Trace-boundaries”.

No

Stop (neither of the earlier patterns)

Fig. 7a. NS chart of “Contour_Tracing” routine

Fig. 7b. NS Chart of “Trace_Boundaries” routine

Fig. 7c. NS Chart of “Trace_One_Boundary”
Table 2. Additional Lookup Table

<table>
<thead>
<tr>
<th>Target pattern</th>
<th>Next direction</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Right (because (A = D, B \neq C))</td>
</tr>
<tr>
<td></td>
<td>Left (because (A \neq D, B = C))</td>
</tr>
<tr>
<td></td>
<td>Stop (because (A = D, B = C))</td>
</tr>
</tbody>
</table>

(*) Decide next direction based on the major label of neighbor pixels would be preferred than simply terminating the tracing.

2.3 An Illustration

In this section, we describe the contour tracing steps for a simple example in Fig.8 that consists of two regions and a background, which is also considered a region on its own. At the end, two closed regions are composed, with one common boundary traced and shared.

Tracing steps:
1. Scan input from left to right, top to bottom.
2. Start contour trace if the label of right pixel not equals the current.
3. Add branches at stop point to “non_traced_branches” = \{A, B, C\}
4. Remove branch A and start tracing.
5. At stop point, update “traced_boundaries” = \{A, D\}, “non_traced_branches” = \{B, C, E, F\}.
6. Remove branch B and start tracing.
7. At stop point, update “traced_boundaries” = \{A, D, B, E\}, “non_traced_branches” = \{C, F\}.
8. Remove branch C and start tracing.
9. At stop point, update “traced_boundaries” = \{A, D, B, E, C, F\}, “non_traced_branches” = \{\}.
10. “traced_boundaries” = \{A, B, C, D, E, F\}, and new_region X = \{\}.
11. Remove A, and add to X = \{AD\}, and “trace_boundaries” = \{B, C, D, E, F\}.
12. From end point at D, nearest and counter clockwise branch is E.
13. Remove E, and add to X = \{AD, EB\}, and “trace_boundaries” = \{B, C, D, F\}.
14. From end point at B, nearest and counter clockwise branch is A.
15. AD is already in X, and region X is closed. Add X to regions list.
16. “traced_boundaries” = \{B, C, D, F\}, and new_region Y = \{\}.
17. Remove B, and add to Y = \{BE\}, and “trace_boundaries” = \{C, D, F\}.
18. From end point at E, nearest and counter clockwise branch is F.
19. Remove F, and add to Y = \{BE, FC\}, and “trace_boundaries” = \{C\}.
20. From end point at C, nearest and counter clockwise branch is B.
21. BE is already in Y, and region Y is closed. Add Y to regions list.
22. “traced_boundaries” = \{C, D\}, and new_region Z = \{\}.
23. Remove C, and add to Z = \{CF\}, and “trace_boundaries” = \{D\}.
24. From end point at F, nearest and counter clockwise branch is D.
25. Remove D, and add to Z = \{CF, DA\}, and “trace_boundaries” = \{\}.
26. From end point at A, nearest and counter clockwise branch is C.
27. CF is already in Z, and region Z is closed. Add Z to regions list.
28. Region X is outermost region, and is removed from the final regions list.

Fig.8. An example to illustrate contour tracing steps.
3. Experimental Evaluation

The effectiveness of the proposed contour tracing algorithm is examined through a set of experiments. For evaluation, images that are reconstructed from coded images in function format to which a conventional contour tracing algorithm applied in [3] are compared with those that were traced by the proposed algorithm based on the following three criteria:

1. Overlaps and gaps are not observed at common boundaries between regions in the reconstructed image, particularly after an enlargement operation.
2. Reduction in the total length (in pixels) of all traced boundaries.
3. Reduction in the overall processing time for function approximation.

Three raster images in Microsoft bitmap format in 8bits/pixel are used for experimental evaluation and are shown in Figure 9. The programs used for our experiments (one using the conventional tracing algorithm and another the proposed algorithm) were written in C. The computer used in the experiments is a Pentium III 1.1GHz PC with 632MB memory running Microsoft Windows XP operating system.

Figure 10 shows respectively the outlines of two reconstructed images related to test image #1 given in Figure 9. Both of these outlines were enlarged by 3 times the original size, with the one on the left being traced by the proposed algorithm while the one on the right by the conventional algorithm as applied in [3]. It is obvious that overlaps and gaps are observed in the reconstructed image on the right whereas did not occur in the one on the left.

Similarly, Figure 11 shows the outlines of an area in the reconstructed images for test image #2 given in Figure 9. Both were being enlarged by 5 times the original size, with the one on the left being traced by the proposed algorithm while the one on the right by the conventional algorithm. Again, overlaps and gaps are observed in a number of areas in the reconstructed image on the right whereas did not occur in the one on the left. Lastly, Figure 12 shows respectively an area in the reconstructed images for test image #3 in Figure 9 being enlarged by 5 times the original size. By comparing these two images with the original image of similar size, overlaps and gaps are clearly observed at the places that are highlighted in the reconstructed image on the right whereas did not occur in the one on the left.

Next, the proposed algorithm is compared with the conventional algorithm using the second and the third evaluation criteria, namely the reduction in total length of traced contours and the amount of overall processing time. As noted earlier, the only difference lies in the stage of contour tracing. The results are summarized in Table 3.

From Table 3, it is observed that the total length of traced boundaries is smaller when the proposed algorithm is applied than when the conventional approach is used. In terms of overall processing times, the experimental results show that when the proposed algorithm is applied, the reductions are 66%, 37% and 59% respectively. These results confirm the merits of the proposed algorithm.
Fig. 10. Compare the proposed and conventional algorithm on reconstructed outlines of Test image #1 (3x enlarged)

Fig. 11. Compare the proposed and conventional algorithm on the outline of a reconstructed area in Test image #2 (5x enlarged, an area in reconstructed image)

Fig. 12. Compare the proposed and conventional algorithm on a reconstructed area in Test image #3 (5x enlarged)
Table 3. Compare the total length of traced boundaries (in pixels) and the processing time between proposed and conventional algorithm

<table>
<thead>
<tr>
<th>Method</th>
<th>Total length of traced boundaries (in pixels)</th>
<th>Overall Processing Time (seconds)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Image #1</td>
<td>Image #2</td>
</tr>
<tr>
<td>Conventional algorithm</td>
<td>955</td>
<td>6,767</td>
</tr>
<tr>
<td>Proposed algorithm</td>
<td>770</td>
<td>5,536</td>
</tr>
</tbody>
</table>

4. Conclusions

A contour tracing algorithm for function approximation of color raster images is introduced that solves the problems caused by duplicate tracing the common boundaries of adjoining color regions when conventional algorithms designed for bi-level images are inappropriately used. The effectiveness of the proposed algorithm was evaluated by a set of experiments that confirmed that overlaps and gaps did not occur at the common boundaries in the reconstructed images. Furthermore, by applying the proposed contour tracing algorithm, reductions in both the processing time and the size of function approximated images are observed.

Appendix A. Pseudo code of major routines

As mentioned in Section 2.2, the proposed algorithm is composed of three major routines. Pseudo codes of these routines are given below.

```
Contour_Tracing (In: Image, Out: Result) {
(1) list_of_boundaries = {}; 
(2) list_of_regions = {}; 
(3) x = y = 0; cp = pixel(x,y); /* cp: current pixel */ 
(4) for(y=0 ; y < Image.height ; y++) { 
(5) for(x=0 ; x < Image.width-1 ; x++){
(6) cp = pixel(x,y); 
(7) if (not marked(cp) &&
    label(cp) != label(right_of(cp))) {
(8) start_branch.position = right_up_corner(cp); 
(9) start_branch.direction = downward; 
(10) Trace_Boundaries(in: start_branch,
                out: new_bnds, new_regs);
(11) add_to(list_of_boundaries, new_bnds); 
(12) add_to(list_of_regions, new_regs); 
(13) }
(14) } 
(15) }
(16) Result = {list_of_boundaries, list_of_regions}; 
}

Trace_Boundaries (In: start_branch, 
                Out: new_bnds, new_regs) {
(1) traced_boundaries = {}; 
(2) non_traced_branches = {}; 
(3) Trace_One_Boundary(In: start_branch, 
                        Out: branches,traced_boundary); 
(4) if (branches == {}) { /* a closed boundary */ 
(5) new_bnds = new_regs = traced_boundary; 
(6) return; 
(7) } 
(8) add_to(non_traced_branches, branches); 
(9) while (non_traced_branches != {}) { 
(10) br = remove_first(non_traced_branches) 
     /* 'br' has start position and direction*/ 
(11) Trace_One_Boundary( In: br, 
                        Out: branches,traced_boundary); 
(12) rev_br = endpoint_of(traced_boundary); 
(13) if (belong(rev_br, non_traced_branches) 
(14) delete_from(non_traced_branches,rev_br); 
(15) } else { 
(16) delete_from(branches, rev_br); 
(17) add_to(non_traced_branches, branches); 
(18) } 
(19) add_to(traced_bnds, traced_boundary); 
(20) rev_bnd = reverse_of(traced_boundary); 
(21) add_to(traced_bnds, rev_bnd); 
(22) } /* end while */
```
Appendix B. Proof of algorithm’s termination

The proposed algorithm traces all boundaries that belong to the contours that compose the regions in a raster image. A raster image can be considered as a connected planar graph $G$, where each boundary is an edge. There are two branches in opposite direction associated with each boundary. The notations below are defined:

$\text{adj}(b) =$ the set of all edges incident by the vertices of branch ‘$b$’

$B(G) =$ the union set of branches from all edges of graph $G$

$n =$ the number of trace steps at any time

$X_n =$ the set of branches that have not been discovered up to step $n$

$Y_n =$ the set of branches that have not been traced up to step $n$

$Z_n =$ the set of branches that have been traced up to step $n$

Because every branch can be classified as belonging to either $X_n$, $Y_n$ or $Z_n$ the following condition holds at any $n$

$X_n \cup Y_n \cup Z_n = B(G)$

That is, at $n = 0$, contour tracing on the graph $G$ begins. The first branch $b$ that is considered at the first termination point, the set of pre-conditions below hold:

$X_0 = B(G) \setminus \text{adj}(b),$

$Y_0 = \text{adj}(b),$

$Z_0 = \emptyset$

Consider the following cases:

1. If $Y_n \neq \emptyset$, $a \in Y_n$ is taken from the list of untraced branches and start tracing. At the termination point of this trace, if the branch $b_n$ is not an element of $Y_n$, then both $X_n$ and $Y_n$ will be updated by the following rule:

$X_{n+1} = X_n \setminus \text{adj}(b)$

$Y_{n+1} = Y_n \cup \text{adj}(b) \setminus \{a,b\}$

$Z_{n+1} = Z_n \cup \{a,b\}$
(2) If the branch $b_i$ is an element of $Y_n$, then both $X_n$ and $Y_n$ will be updated by the following rule.

\[
\begin{align*}
X_{n+1} &= X_n \\
Y_{n+1} &= Y_n \setminus \{a, b\} \\
Z_{n+1} &= Z_n \cup \{a, b\}
\end{align*}
\]

When $Y = \emptyset$, in the case of $X_n \neq \emptyset$, take any branch $b$ for which $b \in X_n$, in such case the edge to which $b$ belongs to has not been traced, implying that the opposite branch of $b$ is also either not yet traced or not yet discovered, further implying that $Y = \emptyset$ is not true. Therefore, all branches that are included in $X_n$ are not connected with the graph $G$ that has been traced, implying that they are not connected. This contradicts the set of pre-conditions above.

As a result, when $Y_n = \emptyset$, $X_n = \emptyset$ is also satisfied, so that $B(G) = X_n \cup Y_n \cup Z_n$ implying that $Z_n = B(G)$.

Thus, at the termination of the algorithm, all the connected boundaries have been traced.

Finally, if the number of trace steps until $Y = \emptyset$ is $N$, if we consider that any branch that should be traced must somehow be connected to another branch, then $N = |B(G)|/2$.

Acknowledgments

This research has received support from the Core Research for Evolutional Science and Technology Program in the Japan Science and Technology Corporation for Paul W.H. Kwan and from the Center of Tsukuba Advanced Research Alliance of the University of Tsukuba for Tetsuo Sugiymama.

References


