



Reconstruction of digitized contour lines
by Charles Glen Swenson

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in
Computer Science

Montana State University

© Copyright by Charles Glen Swenson (1992)

Abstract:

A method is proposed for the understanding and reconstruction of raster images of paper-based topographical maps. Topographical maps of mountainous regions can exhibit certain inherent problems. Among these is the confluence of contour lines when spacing between the contours becomes too confined for the resolution of the map. A method to automatically detect and resolve such junctions in raster images is proposed and tested. In this method, contours are efficiently thinned as they are extracted, without thinning cliff areas or other map symbols. This results in end points in the extracted contour lines at the junction points. The Breakpoint Method is shown to be effective for resolving these end points.

RECONSTRUCTION OF DIGITIZED CONTOUR LINES

by

Charles Glen Swenson

A thesis submitted in partial fulfillment
of the requirements for the degree

of

Master of Science

in

Computer Science

MONTANA STATE UNIVERSITY
Bozeman, Montana

January 1992

1378
Sw 419

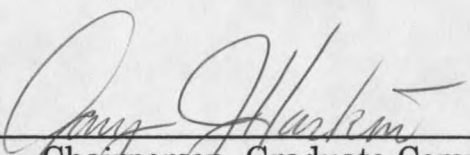
APPROVAL

of a thesis submitted by

Charles Glen Swenson

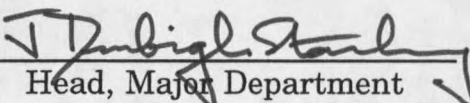
This thesis has been read by each member of the thesis committee and has been found to be satisfactory regarding content, English usage, format, citations, bibliographic style, and consistency, and is ready for submission to the College of Graduate Studies.

1/13/92
Date


Chairperson, Graduate Committee

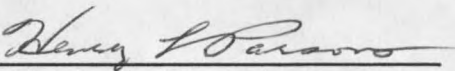
Approved for the Major Department

1/13/92
Date


Head, Major Department

Approved for the College of Graduate Studies

1/16/92
Date


Graduate Dean

STATEMENT OF PERMISSION TO USE

In presenting this thesis in partial fulfillment of the requirements for a master's degree at Montana State University, I agree that the Library shall make it available to borrowers under the rules of the Library. Brief quotations from this thesis are allowable without special permission, provided that accurate acknowledgement of source is made.

Permission for extensive quotation from or reproduction of this thesis may be granted by my major professor, or in his/her absence, by the Dean of Libraries when, in the opinion of either, the proposed use of the material is for scholarly purposes. Any copying or use of the material in this thesis for financial gain shall not be allowed without my written permission.

Signature Charles Glen Swenson

Date 1/13/92

This thesis is dedicated to my wife Giselle, who always believed in me.

TABLE OF CONTENTS

APPROVAL PAGE	ii
STATEMENT OF PERMISSION TO USE	iii
DEDICATION	iv
TABLE OF CONTENTS	v
LIST OF TABLES	vii
LIST OF FIGURES	viii
ABSTRACT	x
1. INTRODUCTION	1
Motivation	1
Problems in Vectorization of Contour Lines	3
Previous Research	9
2. THE TWINS TRACING METHOD	11
Tracing Contour Lines	11
Contour Reconstruction via Adjacency Graphs	13
Limitations of Adjacency Graphs	21
3. THE MODIFIED TWINS TRACING METHOD	25
Isolation of Extracted Contours	25
Determination of Cliff Areas	26
4. RECONSTRUCTION OF CONTOUR LINES	30
The Breakpoint Concept	30
Finding the Breakpoints	32
5. IMPLEMENTATION	34
Scanning of the Contour Lines	34
Scanning of the Cliffs	37
Reconstruction of Contour Lines	41
6. RESULTS	46
Sample Images and Commentary	46
Processing Time	54
Sensitivity to Input Parameters	55
7. CONCLUSION	57
Suggestions for Further Research	57
SELECTED BIBLIOGRAPHY	59
APPENDICES	62

TABLE OF CONTENTS - Continued

APPENDIX A	.63
Data Structures	.63 -
APPENDIX B	.68
Hardware	68

LIST OF TABLES

Table	Page
1. Initial adjacency counting for the map in Figure 14	15
2. Normalized adjacency relationships from Figure 14	16
3. Adjacency relationships from Table 2 after application of cut-off value $\theta = 0.4$	17

LIST OF FIGURES

Figure	Page
1. A digitized contour map with intersecting contour lines.	4
2. A digitized contour map with intersecting contour lines and contours that end in the center of the image.	5
3. A digitized contour map with contours that intersect other map symbols, or end to avoid intersecting them	5
4. A cliff line formed by intersecting contour lines	6
5. A possible interpretation of the cliff line in Figure 4.	7
6. Another possible interpretation of the cliff line in Figure 4.	7
7. Another possible interpretation of the cliff line in Figure 4.	8
8. Another possible interpretation of the cliff line in Figure 4.	8
9. A digitized cliff line.	10
10. The cliff line from Figure 9 after thinning.	10
11. Stopping condition one.	12
12. Stopping condition two.	12
13. Stopping condition three.	12
14. A simple contour map showing the counting for the adjacency relationship between lines A and B.	15
15. Adjacency graph constructed from Table 3.	17
16. A contour map with endpoints.	18
17. Adjacency graph for the map in Figure 16.	18
18. A contour map with endpoints.	18
19. Adjacency graph for the map in Figure 18.	18
20. A contour map with no endpoints.	19
21. Adjacency graph for the map in Figure 20, showing false adjacency relationships.	19
22. A contour map with endpoints.	19
23. Adjacency graph for the map in Figure 22.	19
24. A contour map with endpoints.	22
25. Adjacency graph for the map in Figure 24, showing false adjacency relationships.	22
26. The map from Figure 24 after lines A and B have been joined.	23
27. Adjacency graph of the map in Figure 26, now showing the correct adjacency relationships.	23
28. A simple cliff.	27
29. The cliff from Figure 28 after contour extraction and isolation.	27
30. The cliff from Figure 29 after cliff extraction and isolation.	28
31. Figure 30 with insignificant pixels removed.	28
32. Numerical values for compass directions.	34

LIST OF FIGURES - Continued

Figure	Page
33. The twins starting on the same side of the line.	36
34. A point that may be 4-connected or 8-connected.	40
35. A cliff with one misinterpreted contour line pair.	42
36. A cliff with two misinterpreted contour line pairs.	42
37. A cliff that is only partially included in the image.	43
38. A map exhibiting both a cliff and end points.	44
39. The map from Figure 38 after application of the breakpoint algorithm.	45
40. The final reconstruction of the map from Figure 39 after application of the adjacency algorithm.	45
41. A digitized contour map.	47
42. The cliffs and contours extracted from the map in Figure 41. . .	47
43. The reconstructed map after application of the breakpoint algorithm.	47
44. The final reconstruction after application of the adjacency algorithm.	47
45. A digitized contour map.	48
46. The cliffs and contours extracted from the map in Figure 45. . .	48
47. The reconstructed map after application of the breakpoint algorithm.	48
48. The final reconstruction after application of the adjacency algorithm.	48
49. A saddle.	50
50. Reconstruction of the saddle from Figure 49.	50
51. A simple cliff.	51
52. A complex cliff.	51
53. The correct path for reconstructing a contour line.	52
54. The correct paths for reconstructing multiple contour lines. . . .	53
55. Contour lines with both ends entering separate cliffs.	66

ABSTRACT

A method is proposed for the understanding and reconstruction of raster images of paper-based topographical maps. Topographical maps of mountainous regions can exhibit certain inherent problems. Among these is the confluence of contour lines when spacing between the contours becomes too confined for the resolution of the map. A method to automatically detect and resolve such junctions in raster images is proposed and tested. In this method, contours are efficiently thinned as they are extracted, without thinning cliff areas or other map symbols. This results in end points in the extracted contour lines at the junction points. The Breakpoint Method is shown to be effective for resolving these end points.

CHAPTER 1

INTRODUCTION

Motivation

The computer's ability to rapidly retrieve and manipulate large quantities of data has spurred a swift growth in the field of computer aided cartography. But to use computers to store and manipulate geographical data, the data must be in a form the computer can use; that is, digitized. Some modern data acquisition techniques, e.g. remote sensing via satellite, may capture data in a form ready for computer use [18]. However, much of the geographical information currently available is in the form of paper based maps. This has led to growing interest in techniques for digitizing these maps.

Digitization of paper based maps has traditionally been done manually with the aid of a digitization tablet. Maps may store many types of information, such as roads, buildings, property boundaries, population densities, vegetation cover, etc. A map is placed on the digitizing table, and the boundaries of these features are traced by hand using a stylus or a puck. The digitizing hardware and software save the data in a vector format convenient for computer processing.

Topographical maps are a common type of paper-based map. These maps show elevation data in the form of contour lines. Contour maps contain unusually massive amounts of data, making their digitization by traditional means an expensive and time consuming process. Digitizing a typical contour map by manually tracing all of its contour lines may consume from 40 to more than 400 tedious man-hours [5, 24]. Human error in these tracings is also a recurring problem; loops that should close don't, or do so with an over-

lap, leaving a tail on the end; lines may be inaccurately scanned, scanned twice, or omitted.

A need has therefore been recognized for effective ways of automating the process of digitization and vectorization of contour lines from paper-based maps. Three different approaches have been taken toward automating this process.

The first method attempts to mechanically imitate the manual tracing techniques. Special opto-mechanical line following hardware has been used to trace the lines directly from paper-based maps [3]. This method has seen only limited success.

A second approach uses scanning hardware to produce a raster image of the map as an end result [2]. This raster image is little more than a photograph of the map in question, as there is no implied relationship between pixels in the image; i.e. the contours are not extracted as individual entities. Although initially inexpensive, this approach is unpopular because most existing Computer Aided Cartography (CAC) software requires the data to be in vector format, and can make little use of raster images [5, 13, 18]. Another problem with this method is that the storage requirements for a raster image of a map will always be greater than that of its vector representation [5]. Considering the amounts of data involved, a reduction in storage requirements is very desirable.

Most promising are methods involving raster-scanning with subsequent processing to vectorize the image. Many methods have been proposed for extracting various shapes and features, but few can address problems particular to the correct extraction of contour lines.

Problems in Vectorization of Contour Lines

Most techniques for line extraction have concentrated on documents with highly regular lines and structures. Examples of these are algorithms for extracting lines from blueprints and engineering drafts [6, 9, 12], and for extracting buildings, roads, and certain boundaries from map images [2, 6, 10, 11, 14, 16, 19, 22, 23, 24]. Lines in these drawings tend to be straight and uniform, or composed of simple curves or dashed lines. These algorithms generally work by exploiting these known features of such lines. Unfortunately, contour lines exhibit none of these regularities, taking shapes as infinitely varied as the terrain being mapped.

It can further be proved that contour lines must follow certain strict laws not shared by other line structured data [17]. These laws include:

- 1) *Contour lines cannot intersect.*
- 2) *Contour lines cannot end except on the edges of a map.*

This causes special problems in the processing of topographical data, particularly in regions of mountainous terrain such as Montana. As the steepness of a mountain slope increases, the distance between contour lines on a map decreases. If the distance between contour line centers becomes less than the thickness of the contour lines, then rule 1 above is necessarily violated. Essentially, the contour lines will all run together into one large line. These junctions have been termed *Cliff Lines* [17].

Map makers faced with these problems may take one of 3 options when producing their paper-based maps. They first may increase the contour interval, thereby reducing the amount of data stored on the map (and subsequently its usefulness.) This is a poor solution at best, for in steeper terrain

the map would require a yet larger interval, until finally no contours at all could appear on maps of areas with vertical cliffs.

More often, map makers ignore the above rules, forming cliff lines and/or contours with endpoints on their maps at such problem areas. (Figures 1 & 2). Other problems can also arise later when text, or lines from other map symbols are inserted and overlap the contour lines. These situations are also handled by having the contour lines end before the overlapping symbol and resume on the other side, or by just imprinting the symbol on top of the contour line (Figure 3). Once again the rules for contours are violated on the map.

These violations of basic cartography are most often not a problem for the intended audience of these maps [11]. The human mind has powerful reasoning capabilities and usually (but not always) can correctly interpret

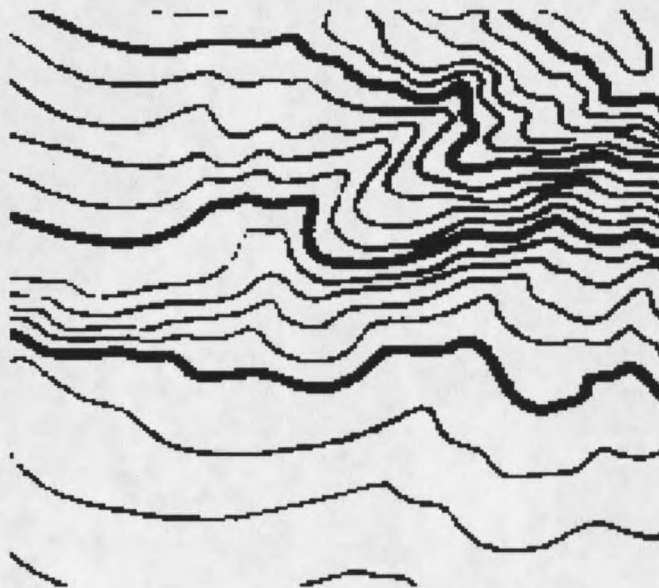


Figure 1. A digitized contour map with intersecting contour lines

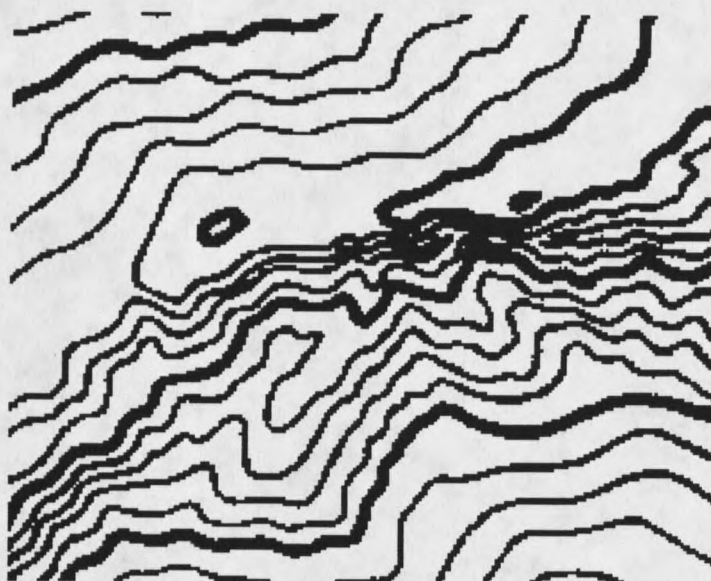


Figure 2. A digitized contour map with intersecting contour lines and contour lines that end in the center of the image.

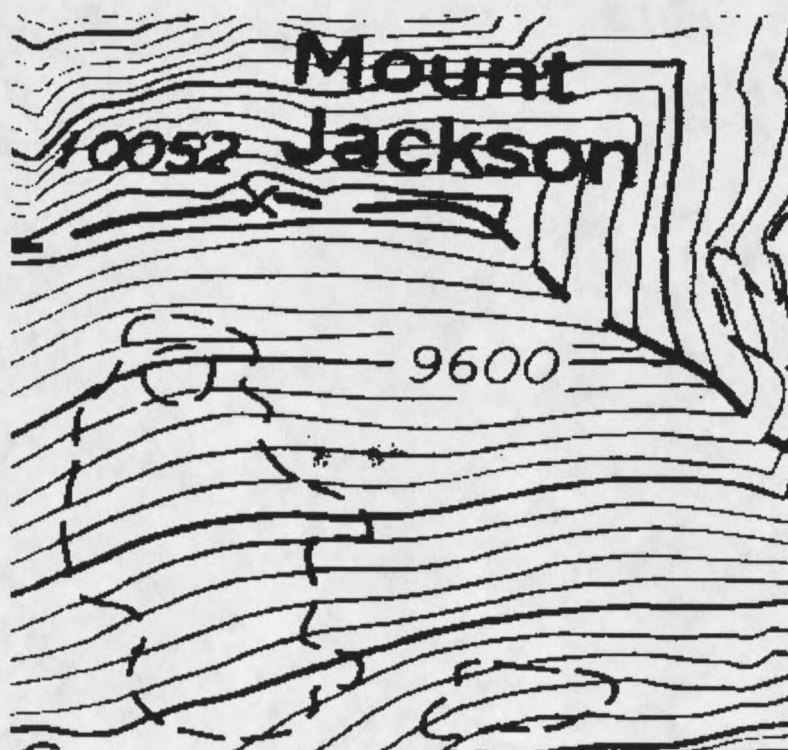


Figure 3. A digitized contour map with contour lines that intersect other map symbols, or end to avoid intersecting them.

these areas. The situation presents a greater challenge for the digital computer.

Figure 4 shows a simulated cliff line composed of intersecting contour lines. Figures 5 - 8 show some of the ways the contours could be reconstructed, none of which can be rejected as violating rules 1 or 2.

Violations of rules 1 and 2 may also result solely from the scanning process itself. The dilemma is that too high a scanning resolution results in



Figure 4. A cliff line formed by intersecting contour lines.



Figure 5. A possible interpretation of the cliff line in Figure 4.



Figure 6. Another possible interpretation of the cliff line in Figure 4.



Figure 7. Another possible interpretation of the cliff line in Figure 4.

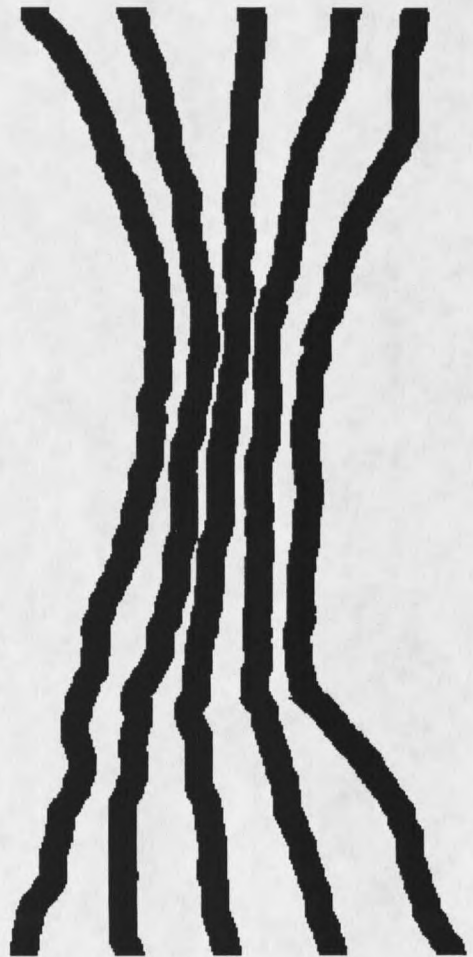


Figure 8. Another possible interpretation of the cliff line in Figure 4.

unmanageably large raster image files; yet too small a resolution can result in broken lines or cliff lines, even from a paper based map that originally exhibits none of these irregularities.

We are therefore resigned to expect violations of rules 1 and 2 in our raster images of paper-based contour maps. A method is needed for extracting the contour lines in vector format from such an image.

Previous Research

Although much work has been done toward understanding raster images of contour maps, previous works most often ignore the above mentioned problems of cliff lines or end points [2, 6, 12, 21]. When the existence of such problems is acknowledged, the solution offered is usually restricted to post-processing detection by human inspection and/or subsequent interactive correction of the problems [3, 9, 5, 13, 15, 22, 23].

As well as being more time consuming, there may be instances where interactive post-processing may prove difficult. Virtually all attempts at understanding a raster image require some form of preprocessing. This invariably includes the skeletonizing, or thinning of the image to lines of pixel thickness. Not only are lines easier to work with than areas, but they more accurately represent certain types of data. For example, in a digitized image of a contour map, a given contour line may be several pixels thick, when we know that the actual contour line should be only of point thickness.

Unfortunately, thinning is done as a preprocessing step *before* any attempt is made to understand the image. This can prove unsatisfactory for contour line data, especially when cliff lines are involved. Critical data may

be lost before understanding of the image is attempted. Figures 9 and 10 show a cliff line before and after thinning. Given Figure 10, detecting the boundaries of the cliff and correctly reconstructing the contour lines will now be difficult even with human intervention.



Figure 9. A digitized cliff line.

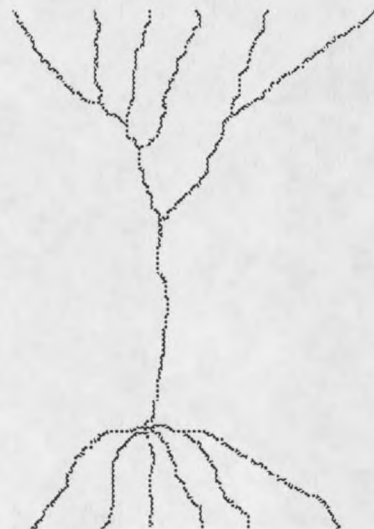


Figure 10. The cliff line from Figure 9 after thinning.

CHAPTER 2

THE TWINS TRACING METHOD

Tracing Contour Lines

The problem of contour lines that end on raster images of contour maps is addressed by Agui and Furukawa [1]. They present a method of tracing contour lines by using twin tracing points on either side of a contour line. Vectorization of a contour map is accomplished in two steps. First all contour lines are extracted. Adjacency graphs are then formed and used to reconstruct end points on the image.

Edge following techniques have been developed to trace along the boundaries of a figure [8]. In these methods, a point on the boundary of a figure rotates through its 8-neighbors to find the transition from background to foreground pixels. This transition point is taken to be the next edge point on the object. This process is repeated until the entire boundary of the image is traced.

In the twins tracing method, tracing points are established on opposite sides of a contour line. Both sides of the line are traced simultaneously by these twin tracing points. One point rotates in a clockwise direction while its twin rotates in a counter-clockwise direction. The twins are advanced in a way which minimizes the distance between them. In this way the points can advance in unison down opposite sides of the contour. This tracing continues until one of three stopping conditions is met:

- 1) The twins collide, indicating the end of a contour line. This is illustrated in Figure 11.

- 2) The twins return to their original positions, indicating a closed contour line as shown in Figure 12.
- 3) The distance between the twins becomes greater than a given threshold, indicating a collision of the contour line with some other map symbol, such as elevation data, section lines, etc. This is shown in Figure 13.

When a stopping condition is met, only one end of the line has been found. The twins are returned to the coordinates of the start of their trace, but the twins are then swapped and the algorithm repeated. This starts the twins tracing in the opposite direction. Again they proceed until a stopping condition is met.

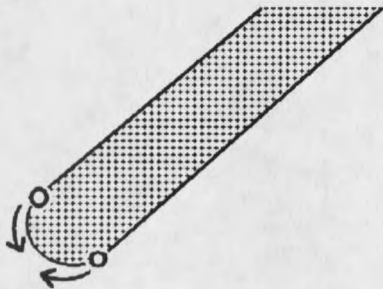


Figure 11. Stopping condition 1.

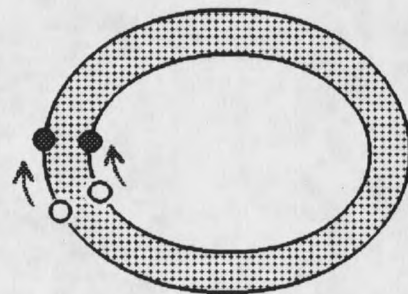


Figure 12. Stopping condition 2.

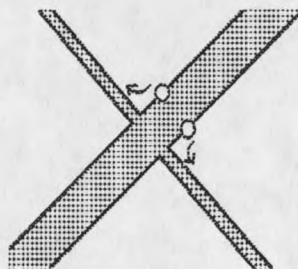


Figure 13. Stopping condition 3.

Once all contours have been extracted, the image is thinned. Information about each contour, such as type and coordinates of endpoints, is stored. Resulting contours that have end points that lie within the image boundaries then need to be reconstructed.

Contour Reconstruction using Adjacency Graphs

A method for tree representation of contour lines which relies on their topological inclusion relations has been reported [7]. In this method, two contour lines are said to be *adjacent* if a line (not necessarily straight) can be drawn that connects the two contour lines and intersects no other contour line. Unfortunately, this does not take into account contour lines that are broken due to graphical error, etc. Two contour lines that are not adjacent may be connected by a line that passes through such a broken contour line without intersecting it.

Agui and Furukawa address this problem with a method involving the absolute position of the contours to construct adjacency graphs for all contours in the image. Their definition of adjacency refers to whether two lines are "next to" each other, within certain parameters. Following is a summary of the method they employ.

The basic idea of the algorithm is to scan the image row by row and again column by column. During this scanning, a table is constructed storing for each pair of contours how often the scan went directly from one contour to the other. This table is then normalized, forming another table. Because of "leaks" through broken contours, it is possible for extraneous values to appear in the table, giving false adjacency relationships. By comparing the values of the normalized table against some threshold, these extraneous

values may be discarded. This results in a third table, which now holds adjacency information about the contours. The following algorithm demonstrates how this is accomplished.

Suppose a figure with R rows and C columns has been traced, and the number of individual lines extracted is n . Further assume that each line is assigned an individual number from 1 to n . A table C_{ij} ($i, j = 1, 2, \dots, n$) is constructed such that C_{pq} contains a high value if line p is very adjacent to q , and zero if there is no adjacency. Algorithm [A] below accomplishes this task:

- [A1] Initialize all elements of C_{ij} to zero.
- [A2] Scan the first row of the image. Set a variable p to the number of the first line encountered in the tracing.
- [A3] Continue scanning, setting a variable q to the number of the next encountered line.
- [A4] Increment the value of C_{pq} and C_{qp} . Set $p = q$. Repeat from step [A3] until the end of the row is reached.
- [A5] Repeat steps [A2] - [A4] for rows 2 through R .
- [A6] When all rows have been scanned, repeat steps [A2] - [A5] scanning column by column, from column 1 through column C .

Consider Figure 14 to be a contour map that has been processed using the twin tracing method. The dashed lines in Figure 14 show the counting for the adjacency relationship between lines A and B. The table C_{ij} resulting from application of algorithm A is shown in Table 1. A normalized table C'_{ij} is formed by normalizing C_{ij} as follows:

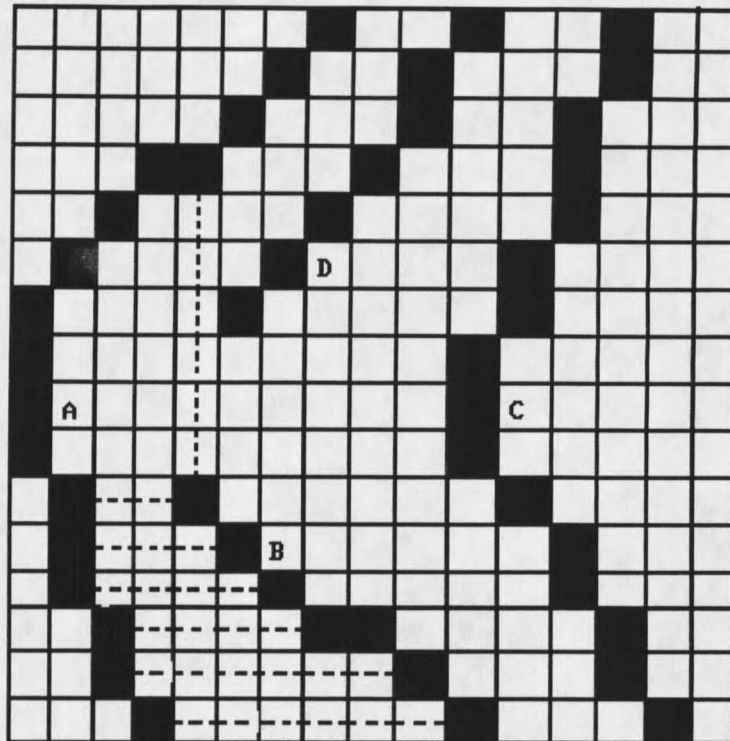


Figure 14. A simple contour map showing the counting for the adjacency relationship between lines A and B.

$$[C_{ij}] = \begin{matrix} & \begin{matrix} A & B & C & D \end{matrix} \\ \begin{matrix} A \\ B \\ C \\ D \end{matrix} & \begin{pmatrix} 0 & 7 & 3 & 10 \\ 7 & 0 & 7 & 5 \\ 3 & 7 & 0 & 8 \\ 10 & 5 & 8 & 0 \end{pmatrix} \end{matrix}$$

Table 1. Initial adjacency counting for the map in Figure 14.

$$C'_{ij} = C_{ij} / (2 \min(e_i, e_j))$$

where e_i and e_j are the lengths in pixels of lines i and j respectively. This assures that $0 \leq C'_{ij} \leq 1$ for all values of i and j . After application of this process, we have Table C'_{ij} , as shown in Table 2.

$$[C'_{ij}] = \begin{pmatrix} 0 & 7/14 & 3/32 & 10/14 \\ 7/14 & 0 & 7/14 & 5/14 \\ 3/32 & 7/14 & 0 & 8/14 \\ 10/14 & 5/14 & 8/14 & 0 \end{pmatrix}$$

Table 2. Normalized adjacency relationships from Figure 14

A third table G_{ij} is used to hold adjacency information extracted from C'_{ij} . G_{ij} is a binary table whose values are determined by:

$$G_{ij} = \begin{cases} 0 & \text{if } C'_{ij} < \text{theta} \\ 1 & \text{if } C'_{ij} \geq \text{theta} \end{cases} \quad (3)$$

where theta is a value determined by the distance between the cutoff points.

Table 3 shows the resulting table G_{ij} from using a value of theta = 0.4.

$$[G_{ij}] = \begin{pmatrix} & A & B & C & D \\ A & 0 & 1 & 0 & 1 \\ B & 1 & 0 & 1 & 0 \\ C & 0 & 1 & 0 & 1 \\ D & 1 & 0 & 1 & 0 \end{pmatrix}$$

Table 3. Adjacency relationships from Table 2 after application of cut-off value $\theta = 0.4$.

A graphical representation of Table 3 is demonstrated in Figure 15. As can be seen, there is a branch in the graph from A to B and D, which closes again to C. By examining this graph we can determine that B and D are both adjacent to A, and are also adjacent to C. This leads us to believe that B and D are possible candidates for re-connection.

Figures 16 through 23 show some possible images and resulting adjacency graphs. These figures illustrate a number of problems that must still

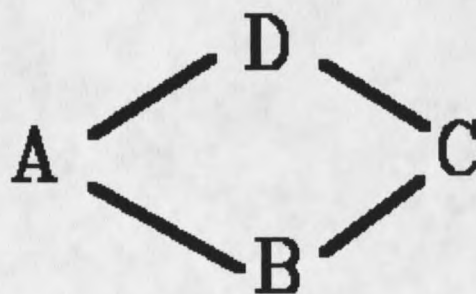


Figure 15. Adjacency graph constructed from Table 3.

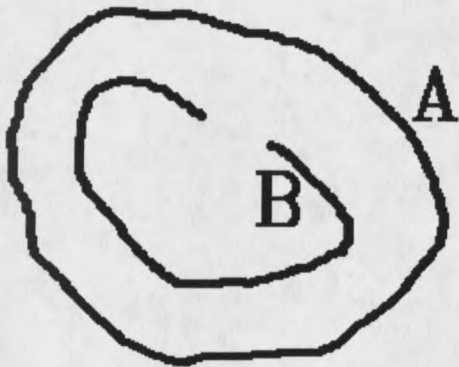


Figure 16. A contour map with endpoints.

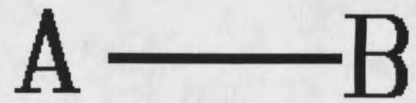


Figure 17. Adjacency graph for the map in Figure 16.

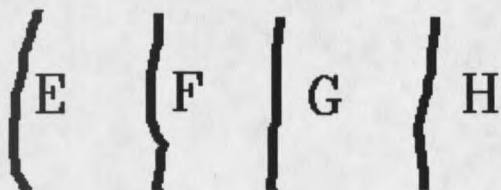
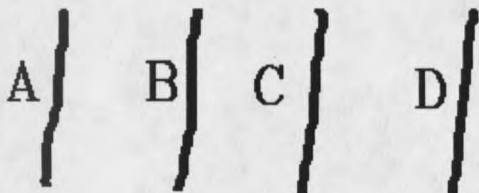


Figure 18. A contour map with endpoints.

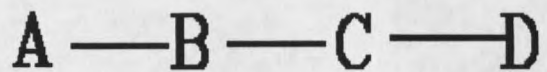


Figure 19. Adjacency graph for the map in Figure 18.

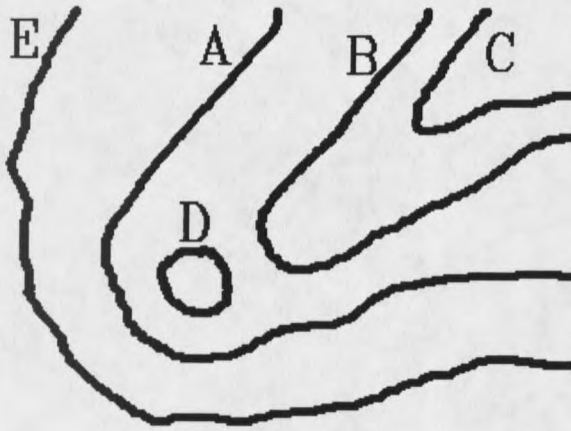


Figure 20. A contour map with no endpoints.

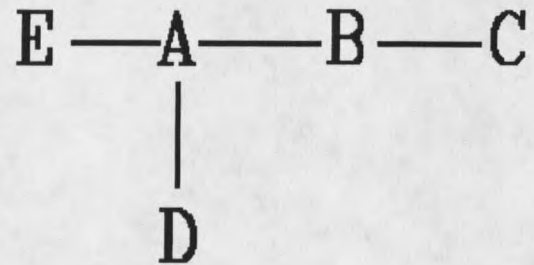


Figure 21. Adjacency graph for the map in Figure 20, showing false adjacency relationships.

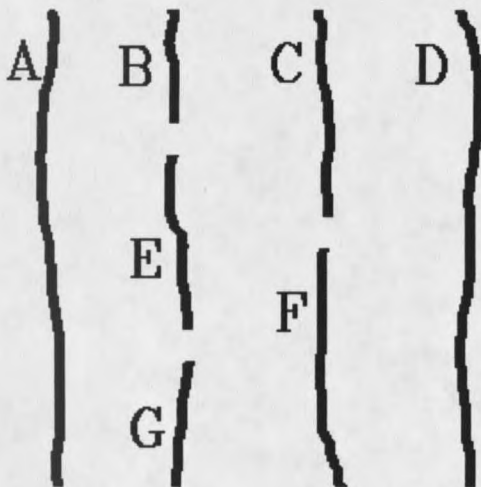


Figure 22. A contour map with endpoints.

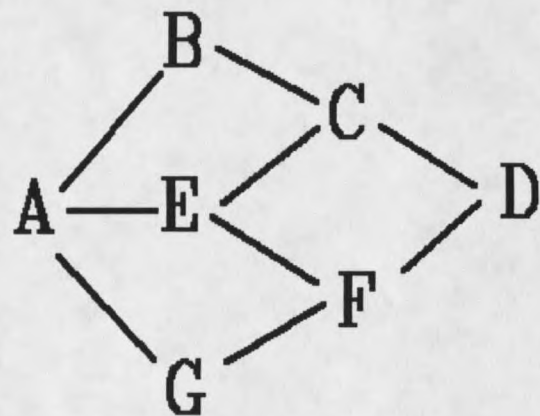


Figure 23. Adjacency graph for the map in Figure 22.

be overcome when trying to reconstruct images based on this method. Consider Figure 23. Based on this graph, which contours should be joined? And which ends of the chosen contours?

The solution offered by Agui and Furukawa is to connect the endpoints that are closest, as long as the connecting line does not intersect the line forming the root of the branch. Following is their algorithm for reconstruction of extracted contour lines based on adjacency graphs:

- [B1] Form a table $ti = (1, 2, \dots, n)$ of end points that lie within the boundaries of the image.
- [B2] Using algorithm [A], construct an adjacency graph.
- [B3] Examine the adjacency graph for branches. Store a list of all branches that satisfy the conditions:
 - (B3-1) At least two branches from a given root must have endpoints that are cut-off points.
 - (B3-2) A line connecting these cut-off points must not cross the root line.
- [B4] Connect the cut-off points that are a minimum distance from each other. Eliminate these points from t . Repeat from step [B2].

This algorithm proves ineffective when many endpoints lie in close proximity to each other. Furthermore, in cases such as Figures 16 and 18, cut-off points do not show up in the corresponding adjacency graphs. And in Figure 21 a branch shows up where no cut-off points occur.

Limitations of Adjacency Graphs

The adjacency graph method of reconstruction is effective for resolving many situations involving cut-off points in a contour map. However, it is quite sensitive to the parameter θ ; too large a value will result in missed contours, while too small a value may result in joining wrong endpoints. More importantly, the authors do not address or even mention the problem of cliff lines. If such cliff lines could be extracted, they would produce adjacency graphs similar the one in Figure 19. These are of the type that the adjacency algorithm has the most trouble with. Moreover, the image is thinned before any attempt is made at understanding the lines it contains. As mentioned in the first chapter, this can lead to loss of data critical to later reconstruction.

The adjacency graph approach is also computationally expensive. Creation of the adjacency graph requires multiple calculations on a large two-dimensional array of real number, as well as two complete scanings of the entire image. More importantly, a new adjacency graph must be constructed after each joining of two lines. The reason for this is illustrated in Figures 24 - 27.

Figure 24 shows a contour map, and Figure 25 shows the adjacency graph for this map. The information about the adjacency of line *D* is missed by the algorithm. However, lines *A* and *B* have the closest endpoints. If these two are connected as in Figure 26, and the adjacency recalculated, a graph showing the new and more correct adjacency of line *D* results, as shown in Figure 27.

Finally, if you can determine which lines enter a cliff, it is intuitive that these are exactly the lines that must be joined together when recon-

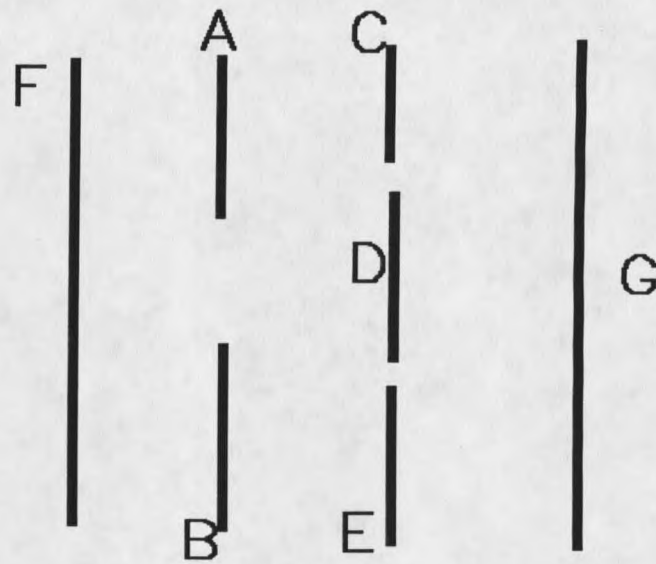


Figure 24. A contour map with end points.

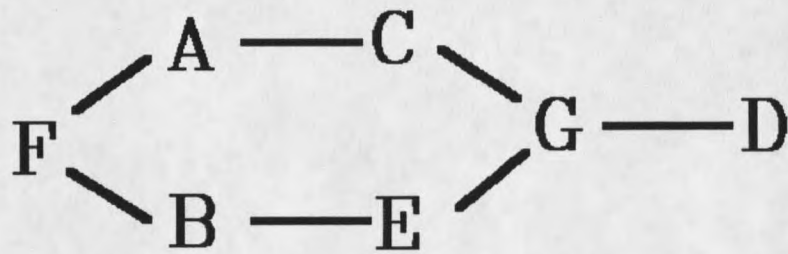


Figure 25. Adjacency graph for the map in Figure 24, showing false adjacency relationships.

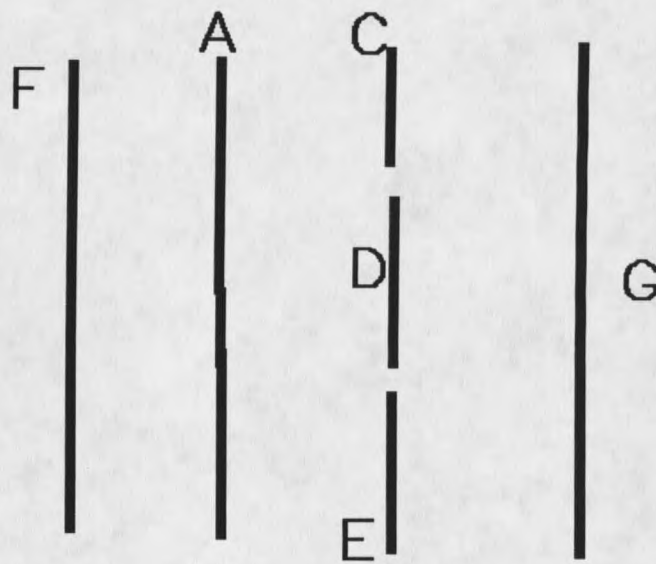


Figure 26. The map from Figure 24 after lines A and B have been joined.

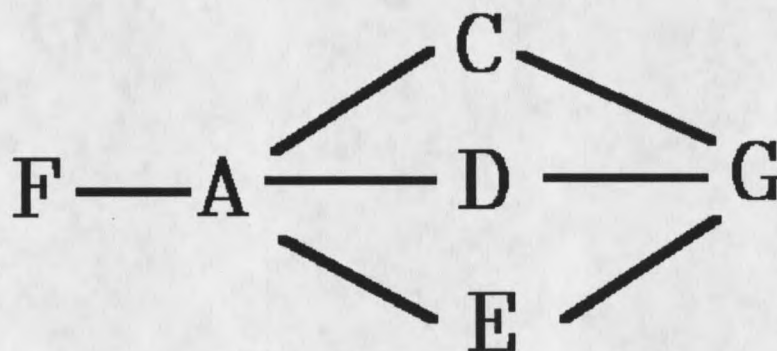


Figure 27. Adjacency graph of the map in Figure 26, now showing the correct adjacency relationships.

structing the contour map. Yet the adjacency method makes no attempt to extract or use this information. It may therefore consider incorrect contours for reconstruction of such areas.

It was decided that a variation of the twins tracing method could be used to extract cliff lines. A list could be kept of all contour lines entering and leaving a cliff. This list could then be consulted when choosing which endpoints to reconnect.

CHAPTER 3

THE MODIFIED TWINS TRACING METHOD

Isolation of Extracted Contours

A modification of the twins tracing algorithm is used to extract the lines of the image. This method assumes that each pixel can be set to one of a range of values, and that the initial image is composed of pixels of background and foreground values only.

It has been shown that for two points set a minimum distance from each other on opposite sides of a line, the midpoint between the points gives an accurate indication of the center of the line [8]. In the modified tracing method, as the twins are advanced the pixels midway between them is calculated and set to a value n uniquely identifying the current line being extracted. These are taken to be pixels on the center of the line itself, i.e. the skeleton of the line. In this way the contour lines are thinned on the fly at the time of extraction.

As the edge point pixels are traced, they are set to a predetermined value v indicating that they are now neither foreground nor background values, but are edges of a line that has already been extracted. This in effect isolates an extracted contour line from consideration in future scans.

The tracing continues as in the original method, with a linked list of center points resulting. The tracing is stopped when one of the following conditions is met:

- 1) The twins intersect one another, indicating an endpoint.
- 2) The twins return to their original positions, indicating a closed contour line.

3) The distance between the twins becomes greater than some threshold, indicating intersection with another contour line or with some other map symbol.

4) The twins encounter a value other than foreground color or background color, indicating an intersection with a previously extracted contour line.

When a stopping condition such as 3) or 4) above is encountered, the twins are connected by a straight line of pixels of value n . This, in combination with the resetting of the values of the contour's perimeter pixels, assures the complete isolation of the contour. It also assures that the portion of the perimeter of a cliff where a contour line enters will have its pixels set to a value that identifies that contour. The importance of this is demonstrated later.

Figure 28 shows an image of a contour map consisting of two contour lines that intersect in a cliff. Foreground pixels have a value of 1. For clarity, background pixels have been omitted. Figure 29 shows the image after extraction of the lines. In this case v has a value of 9. A value n of 2 or higher uniquely identifies a pixel as belonging to an extracted contour line n . As can be seen, each of the four extracted contours is completely isolated from the rest of the image, being surrounded either by the pixels of value 9 which define its original edges, or by pixels which identify the contour.

Determination of Cliff Areas

As described above, the contour lines are thinned and isolated at the time of tracing. Examination of Figure 29 shows that after the tracing step the only foreground-valued pixels which would be exposed to subsequent

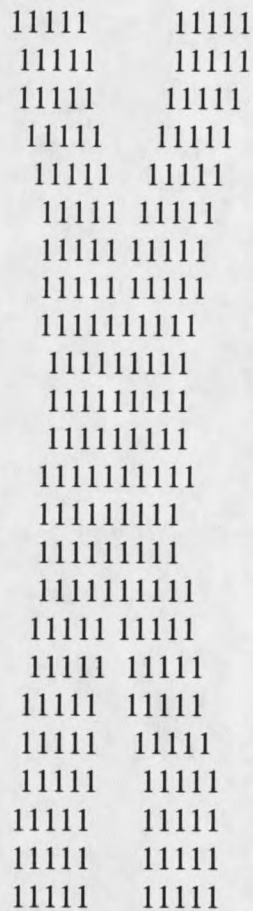


Figure 28. A simple cliff.

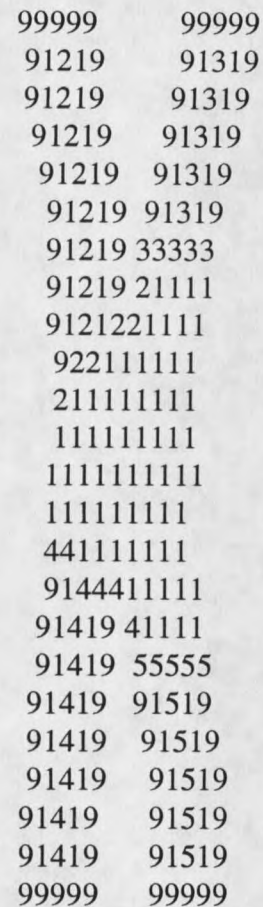


Figure 29. The cliff from Figure 28 after contour extraction and isolation.

scanning belong to the problem area, which is the cliff itself. Since the thinning process is selective, that is, it only thins lines while they fall within certain parameters, the cliff has not yet been thinned. Therefore all original data is still available for use in the reconstruction process.

The image is re-scanned in a manner similar to that described above in order to search for such cliff areas. When a cliff area is encountered, its perimeter is traced in a clockwise direction. Each pixel on the perimeter of the cliff is set to a value i that marks it as having been visited. In this way, the cliff too is isolated from further scanning.

When the tracing of the cliff perimeter is completed, the perimeter will be composed entirely of runs of pixels of value i interspersed with runs of pixels that identify not only the contour lines that enter the cliff, but where in the cliff they enter. The scanning process then continues searching the rest of the image for more cliffs.

Figure 30 shows the image after the cliff scanning is completed. In this case i has a value of 6. Figure 31 shows this same figure with the insignificant pixels removed for clarity.

```

99999      99999
91219      91319
91219      91319
91219      91319
91219      91319
91219 91319
91219 33333
91219 21116
9121221116
922111116
211111116
611111116
611111116
611111116
444111116
914441116
91419 41116
91419 55555
91419 91519
91419 91519
91419 91519
91419 91519
91419 91519
99999      99999

```

Figure 30. Cliff from Figure 29 after cliff extraction and isolation.

```

2          3
2          3
2          3
2          3
2          3
2 33333
2 2 6
2 22 6
22 6
2 6
6 6
6 6
6 6
444 6
444 6
4 4 6
4 55555
4 5
4 5
4 5
4 5
4 5

```

Figure 31. Figure 30 with insignificant pixels removed.

During the scanning of the cliffs, data is tabulated for each cliff encountered. In particular, when a pixel that identifies an extracted contour is encountered, its value is added to a list of extracted lines that enter this cliff. This list of entering contours is kept *in the order in which each value is encountered during the tracing*. Assuming a clockwise tracing, such a list for Figure 31 may look like:

{ 2, 3, 5, 4 }

CHAPTER 4

RECONSTRUCTION OF CONTOUR LINES

After the contour and cliff extraction from chapter 3, we have a list of contour ends, each of which must be connected to another contour in the list. But how to connect them?

Consider again Figure 4 which shows a cliff and the contours that enter it. After extraction of the contour lines, we are faced with deciding which ends to reconnect. Figures 5 - 8 show only some of the possible ways of connecting the contours which do not violate rules 1 and 2 for contour lines as stated in Chapter 1.

Out of all the possibilities, most people will agree that Figure 8 is most likely to be the correct way of rejoining these contours. This is because the directions of the connected endpoints in Figure 8 are more nearly opposite each other in direction than are the ones in the other images. The eye tends to connect such lines. This fact is exploited by the map makers who produce such cliff areas, and is the criteria used in this paper for reconstruction of these cliffs.

The Breakpoint Concept

In mountainous terrain, there is a certain local similarity between contour lines that are in close proximity to each other. This similarity includes the concept of adjacency exploited by Agui and Furukawa [1]. Yet it also includes features such as the shape, and hence the direction, of the contours. This similarity is most pronounced in the areas of steepest terrain, where one is most likely to encounter cliffs.

Consider again Figure 4. Upon examination, we can intuitively divide the lines entering the cliff into two groups; those coming in from the bottom and those coming in from the top. These groups have an equal number of members. Furthermore, the members of each group are adjacent to each other. This will almost always be the case, due to the local similarity described above [1].

The following properties have been demonstrated for general contour lines entering a cliff [17]:

1. *All contour lines that enter a cliff line from the {left, right} and leave towards the {right, left} do so on a first-in-first-out basis.*
2. *If one contour line enters a cliff line from the left and leaves towards the right and another contour line enters the cliff line from the right and leaves towards the left, one contour line must both enter and leave before the other contour line can enter.*

Suppose we are given an ordered list of the contour lines one would encounter in making a complete traverse of the edge of the boundary of the cliff shown in Figure 4. Such a list may look like:

$$L = \{ b, c, d, e, f, g, h, i, j, a \}$$

In traversing this list, we will come to a place where in going from one line to another, one goes from the group of lines entering the cliff from the top to the group entering from the bottom. I call this point the *breakpoint*. In the ex-

ample given, this breakpoint comes between e and f . The lines on either side of this breakpoint, e and f , should be joined together. In examining the list further, we find that if we take the pair of lines that immediately precede and follow this pair in the list, d and g , we come to another pair which should be connected. Proceeding in this manner, we can connect all pairs of contours in the list: c to h , b to i , and a to j .

Finding the BreakPoint

It has thus been established that in most cases a breakpoint exists in an ordered list of lines entering a cliff, and that the correct way of rejoining contour lines is likely to be to join the ends that are most nearly opposite each other in direction. The breakpoint in a list should therefore be established at the point where the largest change in endpoint direction occurs when traversing the list.

The following algorithm is suggested for finding the breakpoint in an ordered list $L = \{1, 2, 3, \dots, n\}$ of lines entering a cliff:

[C1] Set left breakpoint $bl = 1$; right breakpoint $br = n$; max = the magnitude of the difference in direction between line 1 and line n .

[C2] Set left pointer $l = 1$; right pointer $r = 2$. If the magnitude of the difference in direction between lines l and $r > max$ then set $bl = l$ and $br = r$.

[C3] If $r < n$ then increment r and l and execute [C2].

At this point you have established the breakpoint as being between lines bl and br . Following is the algorithm for pairing up the correct endpoints.

[D1] Connect lines bl and br . Set $counter = 1$;

[D2] Increment br ; decrement bl ; if $br > n$ set $br = 1$; if $bl < 1$ set $bl = n$;

[D3] If $counter < n / 2$, go to [D1].

The combination of algorithms [C] and [D] will reconstruct the contours that comprise the cliff.

CHAPTER 5

IMPLEMENTATION

Scanning of the Contour Lines

Once read into memory, the image is scanned from left to right, pixel by pixel, starting with the first row. When a foreground pixel is encountered, a routine is called to analyze the situation. If it appears that this is part of a legitimate contour line, and that edge points have been established on opposite sides of the line, then the line is traced in its entirety. The scanning then continues for the rest of the first row, and then is repeated for each subsequent row.

A problem exists in that a near horizontal line may be missed by this scanning [1]. For this reason, the scanning process is then repeated in a vertical fashion, with the image being re-scanned column by column.

A pixel in a two dimensional image may have neighbors in any of 8 different directions. In all subsequent discussion and algorithms, when a direction is given it will be in a range of $1 \leq \text{direction} < 9$. A direction of 1 is taken to be "straight up", with 3 being "straight right," etc. as illustrated in Figure 32. This same convention will be used not only to discuss the rela-

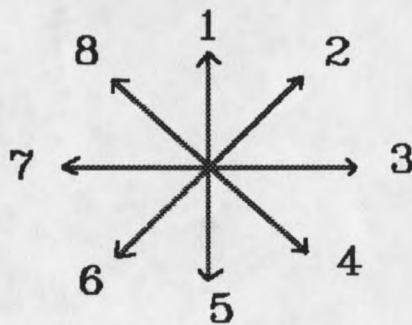


Figure 32. Numeric values for compass directions.

tionship of neighboring pixels, but also for the direction of endpoints of lines, in which case these values may be real numbers instead of integers.

The algorithm used for advancing the left twin along the boundaries of a line is as follows:

[E1] Set the previous direction to 7.

[E2] Starting at the neighboring pixel in the previous direction, keep rotating in a counter-clockwise direction until the 8-connected neighbor in the direction you are pointing is a non-background pixel. This pixel marks the transition from outside to inside the contour. Set the twin's coordinates to this new value, and set the previous direction to the direction from the twin's new position to the last background pixel checked.

[E3] If you have returned to the position first occupied at the start of the algorithm, or if you intersect with the other twin, or if the distance between you and your twin becomes too great, then go on to [E4]. Otherwise, repeat from [E2].

[E4] If you have not returned to the original starting place, then set the coordinates of the right twin to the original coordinates of the left twin, set the previous direction to 3, and repeat one more time from step [E2].

The algorithm for the right twin is symmetrical.

A problem exists in that the twin points might be established on the same side of a line. This can happen at the top of a curve in a contour, as illustrated in Figure 33. This situation is checked for by advancing the twins a set number of times before the real tracing begins [1]. Since the twins

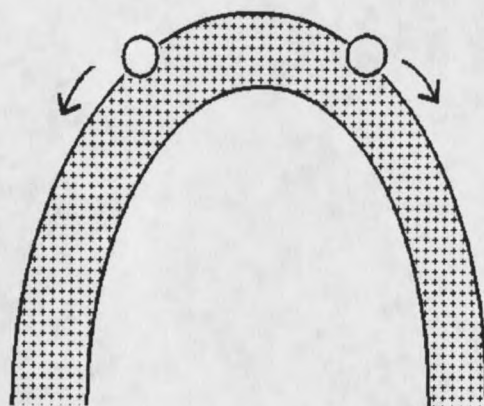


Figure 33. The twins starting on the same side of the line.

rotate in opposite directions, if they happen to be on the same side of the line they will quickly either collide or move farther and farther apart. If either of these cases occur, the points are abandoned.

Following is the algorithm for scanning the contour lines.

- [F1] Set the column index to point to column 0 and the row index to row 0. Initialize a contour ID number c to some low value.
- [F2] Scan the current row until a foreground pixel is encountered. Set the first twin tl to this position.
- [F3] Continue scanning until a background pixel is reached or the edge of the image is reached. In the first case set the second twin tr to this position. In the latter case, go to [F2], restarting the scanning at the beginning of the next row. (Unless, of course, you are on the last row).
- [F4] Test the two points tl and tr using algorithm [E]. If the points fail the test, go to [F2] and restart the scanning from $tr + 1$.
- [F5] Begin tracing along the edges of the contour, marking each pixel visited

by tl and tr to a pre-defined value indicating that this pixel has been visited. Always advance either tr , tl , or both, whichever will result in a minimum distance between the two. At each move, find the pixel midway between tl and tr and set it to c . Add this value to the end of a chain which defines the extracted contour. Continue until a stopping condition is reached. If the end of the line was not found, connect tl to tr with a straight line, setting all pixels on this line equal to c .

[F6] Set tl to the original location of tr and vice versa. Restart the tracing one more time from [F4].

By this method, when a contour line is discovered, it is immediately traced from the point of discovery to both of its ends before the scanning continues. The line is both thinned and vectorized during this process. However, since the twins can detect when they are entering a cliff, the thinning is selective, and none of the cliff data is lost. The entire contour is also effectively sealed off during the tracing, and will not be considered when encountered during subsequent scanning.

If a foreground pixel is found when scanning for a line, and before finding the next background pixel you then encounter a pixel other than a foreground pixel or else come to the edge of the image, the points are abandoned and the scanning continues.

Scanning the Cliffs

After all contours have been scanned, all exposed foreground pixels are assumed to be parts of cliffs. The image is scanned again to find these cliff

areas. Once a cliff is found, its boundary is traced and the cliff is isolated from further scanning in a manner similar to that used for contour lines. When a pixel is encountered that belongs to an extracted contour line, that contour line is added to the end of a list of contour lines that enter this cliff. When the entire cliff perimeter is traced, the scanning continues. Again, certain shapes and configurations may make a cliff invisible to a horizontal scan, so the image is scanned twice, once top to bottom, then again left to right.

Any parts of the image which were not extracted as contour lines will be extracted by the cliff scanning. This includes the cliffs themselves, of course, but also includes map symbols, short or broken lines, contour elevation data, broken contour lines that were too short to be accepted by the twins tracing algorithm, etc. These "cliffs" are handled by discarding any cliff that was traced without encountering a contour line entering it. In a practical implementation, pre-processing could discard many of these symbols on the basis of original pixel color.

The perimeter of a given cliff will consist of exposed foreground pixels and Contour Identification Pixels that were used to seal off the ends of the contours that enter the cliff. Due to the nature of the scanning algorithm, the cliff tracing algorithm always begins its trace on an exposed foreground pixel. It then rotates through its 8-neighbor pixels to find the transition from outside the cliff to inside, and hence the next pixel on the cliff perimeter.

The edge tracing algorithm is similar to the perimeter tracing used for contour lines, except for two important differences. First, any pixel value less than n is considered to be outside the perimeter of the cliff area (i.e., are "background" pixels), where n is the highest contour identification value used

to mark extracted contours. This is because the outside border of the cliff is composed not only of true background pixels, but of the ends of the already extracted contour lines.

The second way in which the tracing is different is that the cliff perimeter is sometimes considered to be four-connected instead of eight-connected. Examination of Figure 30 shows why this is necessary. In this instance, the interior of the cliff is eight-connected to the interior of the upper left extracted contour line (line 2). If the perimeter of the cliff was always considered to be eight-connected, the edge tracing would "leak" through from the perimeter of the cliff to the interior of this extracted contour line. However, four-connectivity would lose some of the detail of the cliff perimeter, and may also result in missed contour identification pixels.

This dilemma is handled as follows: the algorithm considers accepting a perimeter pixel in the direction of 2, 4, 6, or 8, only if the pixel in the direction immediately before and immediately after the pixel are not both background pixels. (Background being any pixel of value less than n , as explained above). For example, in Figure 34, if point p is the current edge pixel, point b (direction 8) would only be considered if points a (direction 7) and c (direction 1) are not both background pixels. This allows the cliff perimeter to retain its eight-connectivity without penetrating the extracted contours. It also allows the scanning process to examine all 8 neighbors to ensure encountering any contour identification pixels that border the perimeter.

During the tracing, if a Contour Identification Pixel is encountered, the algorithm checks to see if this contour has already been added to the list

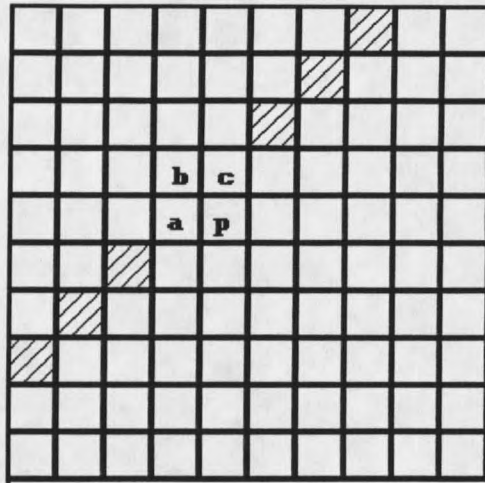


Figure 34. A point that may be 4-connected or 8-connected.

of contours entering this cliff. If it is not in the list, it is added to the end, and the tracing continues to the next perimeter point. The algorithm stops when it returns to the original starting pixel.

After tracing the entire perimeter of the cliff, its perimeter will be composed solely of pixels that mark it as having been processed, and by the pixels that identify the lines entering it. The cliff is therefore effectively isolated from further scanning. An ordered list of the contours that enter the cliff has also been made.

The following algorithm describes the scanning and tracing of the cliffs:

- [G1] Set the column index to point to column 0 and the row index to row 0.
- [G2] Scan the current row from the current position until a foreground pixel is encountered. This pixel will be on the perimeter of an unextracted cliff. If the end of the row is reached during the scanning, increment the row index and set the col index to 0. Quit when you have scanned every row.

[G4] Begin tracing along the edges of the contour as follows: Examine every eight-neighbor in a clockwise direction, starting with the pixel that was examined just prior to discovering the current edge pixel (which will necessarily be a background pixel). When you encounter a foreground pixel that passes the 4-8 connectivity test described above, set the current pixel to i , add it to the end of a linked list which defines the perimeter of the cliff, and make the newly found pixel the current pixel. If during the tracing you see a pixel whose original value indicates that it is part of an extracted contour line, and this line is not in the list of lines for this contour, add this contour line to the end of the list. Continue until you reach and process the original starting point.

[G5] Repeat this process scanning column by column instead of row by row.

Reconstruction of Contour Lines

Once all contours and cliffs have been extracted, we have all of the contours in vector format. We further have a list of cliffs, each with a list of contours that enter it. The cliffs in this list are then examined one by one, and an attempt is made to connect the contour ends that enter the cliffs.

For a given cliff, a count is made of the number of contours entering it. If the count is not an even number, it is assumed that some kind of miss-scanning occurred during the extraction of the contour lines. An example of this is shown in Figure 35, where line b is actually two contour lines that form a sort of mini-cliff attached to the main cliff, yet are thin enough to fool the tracing algorithm into thinking they are a single line.

If an even number of such miss-scannings occur for a given cliff, then the



Figure 35. A cliff with one misinterpreted contour line pair.



Figure 36. A cliff with two misinterpreted contour line pairs.

algorithm will not notice and will attempt to join the lines. Again, the principle of local similarity of contours means that in such cases the proper endpoints will usually be joined, albeit some of the endpoints will be of two contours instead of one. This is illustrated in Figure 36, where lines a and f, b and e, and c and d will all be joined, with b and e being the miss-scanned lines.

When it has been established that the cliff has an even number of contours entering it, a search is made for the breakpoint. The ordered list of contours entering the cliff is traversed, and the difference in direction of the endpoints of contours adjacent to each other in the list is tracked. The pair

with the greatest difference is taken to be the breakpoint, and this pair is joined. Then the pair made from the contour immediately preceding and following this pair are joined. This is continued until the list is exhausted. The process is then repeated for the next cliff.

Cliffs that border on the edge of the image are given somewhat special handling. In such cases, critical information may not be available, and an incorrect joining of the contour lines may result. This is illustrated in Figure 37, where lines A and B might be joined, as no knowledge about the existence of lines C and D is available to the program. Therefore, the lines in such cliffs are only joined if the difference of the direction of their endpoints is greater than a certain threshold.

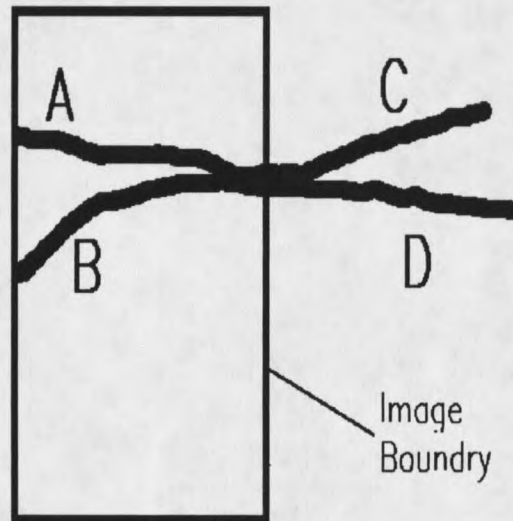


Figure 37. A cliff that is only partially included in the image.

The above method is only designed to handle the problem of cliffs, i.e. lines that violate rule 1 from chapter 1. It does not address rule 2 from chapter 1, namely contour lines that end on a map. Moreover, it is common for

certain problem areas on a real map to exhibit both problems simultaneously. Therefore the adjacency graph method is used in conjunction with the cliff solving method to process the map. The cliff solving method is used first in an attempt to satisfy all cliff problems on the map. Only then is the adjacency method called upon to work on endpoints left over.

Figure 38 shows a cliff problem area that violates both rules 1 and 2 from chapter 1. Figure 39 shows what the cliff would look like after reconstruction using the cliffs method. Figure 40 represents the final reconstruction after subsequent application of the adjacency graph method.



Figure 38. A map exhibiting both a cliff and end points.

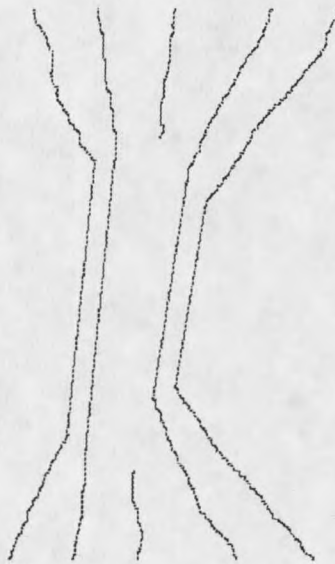


Figure 39. The map from Figure 38 after application of the breakpoint algorithm.

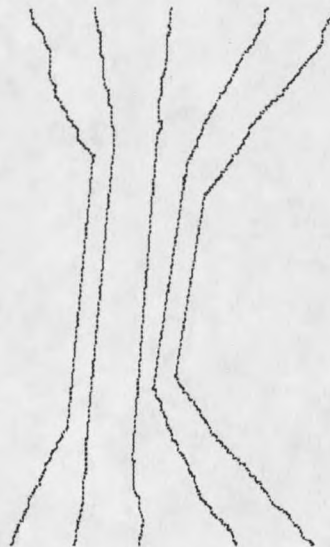


Figure 40. The final reconstruction of the map from Figure 39 after application of the adjacency algorithm.

CHAPTER 6

RESULTS

Sample Images and Commentary

This chapter illustrates some of the results achieved from application of the combined cliffs and adjacency algorithms. The sample images are digitized from contour maps of mountainous areas in the vicinity of Montana State University. Each image is 240 by 244 pixels. Figures 41 and 45 show the original digitized images. Figures 42 and 46 show the contours and cliffs extracted from these images. Figures 43 and 47 show the result of applying the breakpoint algorithm. Figures 44 and 48 show the result of applying the breakpoint algorithm in conjunction with the adjacency algorithm, which gives us the final reconstructions of the images.

In Figure 41 we see one major cliff area, as well as several endpoints. More problems arise when extracting the thicker lines, as they become too wide in areas where sharp curves occur. This results in a series of trivial cliffs on these lines when extracted, as is evident in Figure 42.

In Figure 43 we see that the cliff areas have been successfully resolved, leaving only the endpoints. Most of the latter are resolved by the adjacency method, giving Figure 44 for a final result.

In the resolved cliff area in the upper left quadrant of the image, it appears that two of the three resulting contours still overlap. However, the correct endpoints of all six original contours have indeed been joined, and the resulting three contours are now stored in the computer as separate entities.

In Figure 45, we see a more complicated image. In addition to the many cliff areas in the upper right side of the image, there is a section line

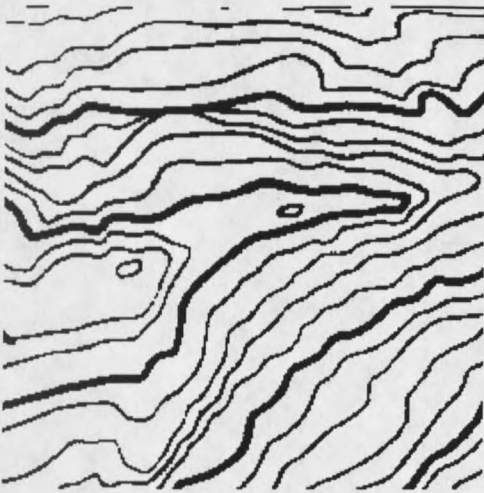


Figure 41. A digitized contour map.

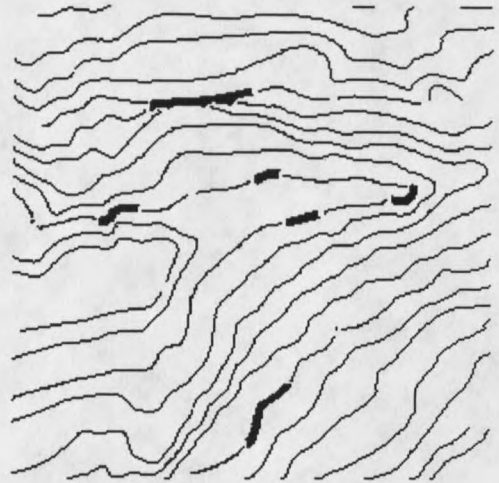


Figure 42. The cliffs and contours extracted from the map in Figure 41.

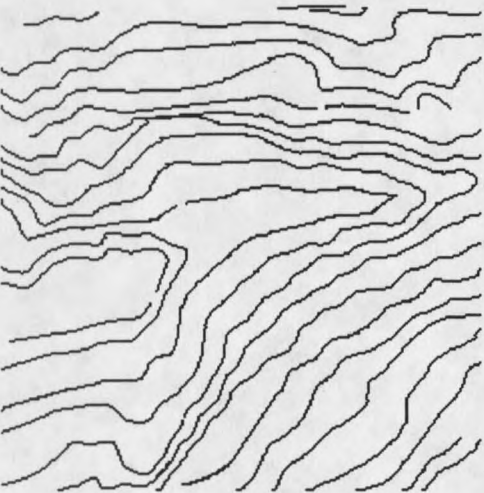


Figure 43. The reconstructed map after application of the breakpoint algorithm.

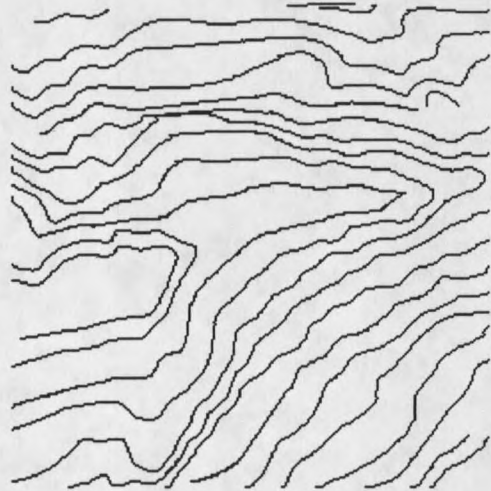


Figure 44. The final reconstruction after application of the adjacency algorithm.

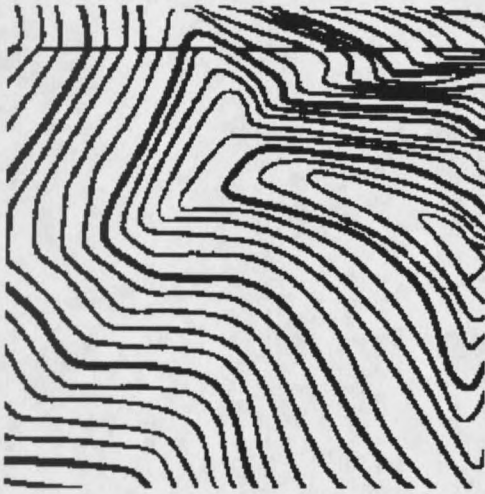


Figure 45. A digitized contour map.

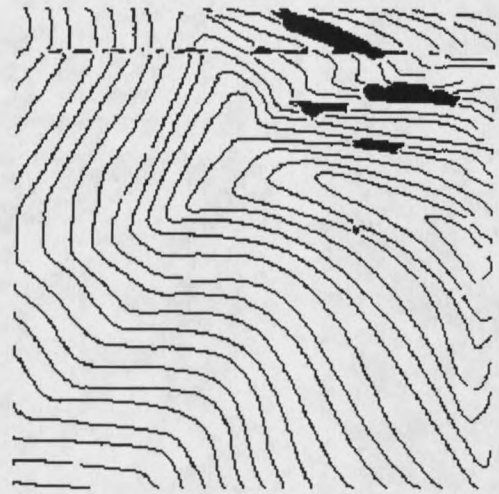


Figure 46. The cliffs and contours extracted from the map in Figure 45.

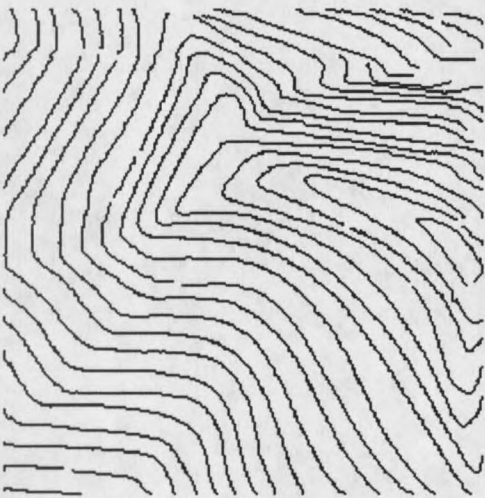


Figure 47. The reconstructed map after application of the breakpoint algorithm.

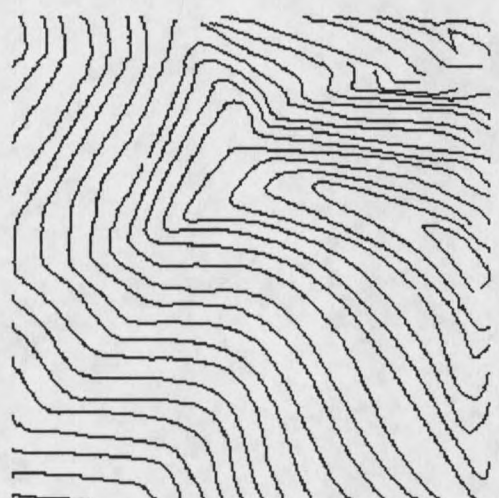


Figure 48. The final reconstruction after application of the adjacency algorithm.

running across the top of the image. This results in a series of small cliffs, as can be seen in the upper left hand portion of Figure 46. In Figure 47 we see that the cliff routine has resolved many of the cliff areas, but has missed a few in the upper left hand corner. This is because many of the contours in this area belong to more than one cliff, and the algorithm is not yet advanced enough to handle this situation. It does recognize when this occurs, however, and defers processing to the adjacency method.

The final result is shown in Figure 48. Although all problems associated with the section line have been resolved, there are still a number of other endpoints that the adjacency method missed. In addition, there is an incorrect joining of two contour lines in the upper right hand corner.

It should be noted that in this example, preprocessing could have easily been used to remove the section line on the basis of color (assuming a color scanner had been used). However, we would then need to rely totally on the adjacency method to reconstruct the resulting endpoints.

In general, the breakpoint algorithm can correctly reconstruct simple cliffs. In cases where the algorithm is not sure of a correct reconstruction, (and in all cases of endpoints instead of cliffs), the decision is deferred to the adjacency algorithm. If this too fails to find a reasonable solution, interactive human input would be the next option. This is as of yet not implemented.

One problem not addressed by the breakpoint algorithm is that of saddle contours in a cliff. Although these areas can lead to mistakes in reconstruction, these mistakes are often not serious. In Figure 49 an example of some saddle contours is shown, with the cliff area represented by the dotted line. In this instance, the breakpoint might be found between lines *d* and *e*, and the reconstructions might be made as shown in Figure 50.

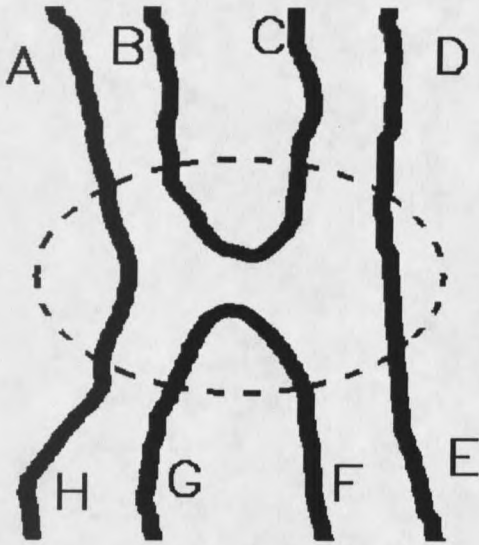


Figure 49. A saddle.

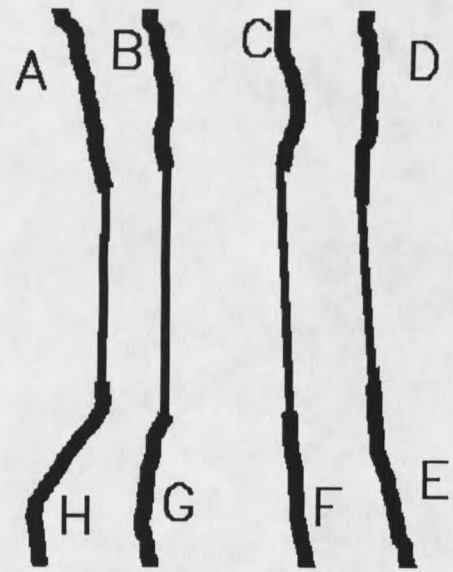


Figure 50. Reconstruction of the saddle from Figure 49.

This is obviously not the correct way to reconnect the lines; however this mistake is not as serious as it first appears. Because the original lines $b-c$ and $g-f$ are both between the original lines $a-h$ and $d-e$, they must necessarily be of the same elevation. Therefore the extent of the mistake is confined to only a slight exaggeration in the elevation of the saddle area itself, an error of less than one contour interval. Future implementations may use other extracted features to detect and process such instances.

The tracing algorithm works best when the maximum thickness parameter is chosen just slightly larger than the average thickness of the contour lines to be extracted. This is true for Agui and Furukawa's original algorithm as well as for the current variation. However, when processing a noisy line whose thickness varies beyond this value, the algorithm will halt at each place where noise causes the line to exceed the maximum thickness

parameter. Each of these places will result in a trivial cliff with only two contour lines entering it. These situations are handled much more efficiently by the breakpoint algorithm than the adjacency method, and with much less chance of error.

One problem that occurs frequently is that both ends of an extracted contour line may enter the same cliff. This situation may result in one of two situations. In the first, we have the endpoints of the line coming in from opposite ends of the cliff. This is illustrated in Figure 51. In this case we have a simple cliff that can be handled easily by the breakpoint method.

The second possibility is illustrated by line A in Figure 52. This results in what I call a complex cliff, a situation the breakpoint method is unable to process. Occurrences of these complex cliffs can be minimized by



Figure 51. A simple cliff.



Figure 52. A complex cliff.

assigning a higher value to the minimum acceptable length for a contour line.

The purpose of this thesis was to correctly identify which endpoints of contours entering a cliff should be joined, not to necessarily correctly join them. Therefore the endpoints in this implementation are simply joined with straight lines. In a practical implementation, a number of methods could be utilized to more correctly connect the endpoints - that is, to have the connecting line more accurately follow where the original connecting line went.

As previously noted, the principle of similarity between proximal lines in areas of steep terrain means that a line may have a roughly similar shape to a line adjacent to it. Agui and Furukawa [1] have suggested that the connecting line may be interpolated between the lines adjacent to the two endpoints. This is illustrated in Figure 53, where the dashed line illustrates the most probable choice for connecting the two endpoints.

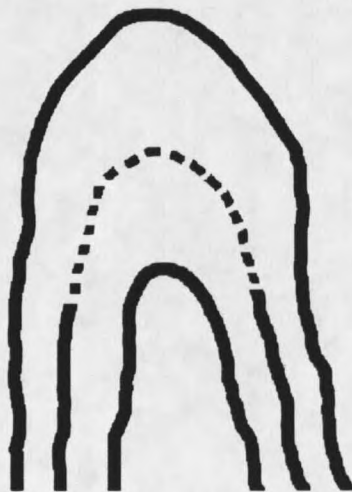


Figure 53. The correct path for reconstructing a contour line.

In cliff areas, additional information on how to connect the endpoints is available, and the selective thinning used by the cliffs method of processing leaves this information intact. Specifically, the connecting line should not include any pixels that were originally background pixels. Therefore, the following method could be used to connect the lines: The pair that are on either side of the breakpoint should join along the edge of the perimeter of the cliff joining them, at a distance of $1/2$ the average diameter of the lines. The last pair of lines connected, i.e, those on the opposite side of the cliff, should be joined similarly along the opposite edge of the cliff. The position of the other lines could be interpolated between these two. This is illustrated in Figure 54.

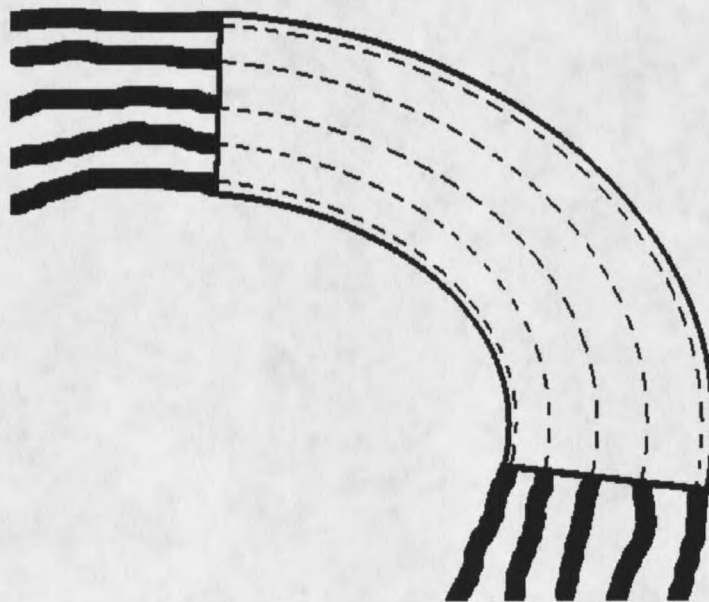


Figure 54. The correct paths for reconstructing multiple contour lines.

Processing Time

As the image only needs to be scanned twice (once horizontally and once vertically) to extract all cliff information, and as this information involves few floating point calculations, the cliff method of reconstruction requires comparatively little compute time.

As mentioned in chapter 2 however, the adjacency graph method requires two complete scanings of the image and subsequent processing and scanning of a two-dimensional array of floating point values for each pair of lines connected. In the current implementation it takes approximately twice as long as the cliffs method per contour line reconstructed .

Processing of the map in Figure 45 took 210 seconds on the target machine. This time was partially distributed as follows:

Reading the image from disk:	17.1 seconds
Extracting the contour lines	
using the Twins method:	11.1 seconds
Extracting the cliff areas:	3.2 seconds
Resolution of cliffs, involving	
reconstruction of 25 lines:	30.5 seconds
Resolution of endpoints, involving	
reconstruction of 12 lines:	66.7 seconds

A further savings in time over the original method is realized by the modification of the twins tracing method wherein the image is thinned during the tracing process. Depending on the alternative thinning method chosen, this can amount to a substantial reduction in total process time.

Sensitivity to Input Parameters

The program as a whole is sensitive to a number of parameters. The most crucial of these are listed below.

The maximum line thickness: Too great of a value will result in distorted endpoints and in the miss-interpretation of non-contour map symbols. Too small a value will result in missed lines, or severely broken lines. A good value will be just slightly larger than the average contour line thickness.

The number of trial steps: Too small a value may sometimes result in the twins starting their tracing on the same side of a line, or in the extraction of non-contour map symbols. Too large a value will result in the missing of shorter contour lines. It was found that a good value tends to be approximately 3 times the average line thickness.

The value θ : Too large or too small a value results in an incorrect adjacency graph. The value used to process Figures 41 and 45 was 0.15.

The value used to determine how far to back off from a contour endpoint: Too small a value will allow the direction to be influenced by the distortion experienced as the twins encounter a cliff. Too large a value may result in lost accuracy in the true direction of the endpoint. A value of approximately the thickness of a contour line was used.

The value used to determine the direction of an endpoint: Too great a value loses detail, too small a value is susceptible to local aberrations in the thinning process. A value of approximately twice the average line thickness was used.

The minimum allowable direction difference before edge-cliffs are processed: This is to take into account the fact that critical information about a cliff, such as which contours enter it, may not be available because part of the cliff is not included in the image, as explained in chapter 5. This parameter is therefore set such that a cliff that borders on the edge of an image is processed only if the lines at the breakpoint are nearly opposite each other in direction. This may result in more unresolved cliff lines, but it is probably better to miss connecting a few lines altogether than to connect them incorrectly.

CHAPTER 7.

CONCLUSION

The breakpoint method has been shown to be an efficient means for resolving simple cliff areas in digitized contour maps. When used in conjunction with the adjacency graph method, many of the common problems encountered in the automatic vectorization of raster images of contour maps can be resolved.

However, the vast number of special cases means that interactive editing will still be necessary in any practical implementation. Further research could refine the process to keep these manual steps to a minimum.

Suggestions for Further Research

The only feature used in finding the breakpoint is the direction of the contour lines - the pair with the greatest difference in direction wins. Although this method is often successful, it may be beneficial to use other features to determine the breakpoints, utilizing statistical pattern recognition methods. Adjacency is one such feature: lines at the break point will likely be adjacent to a common line. Another useful feature of paper-based contour maps that is not exploited is the fact that every 5th contour line is usually slightly thicker than the others. The extraction of this feature will be complicated by instances wherein two contour lines form a long cliff of slightly greater than average thickness, yet still thin enough to be extracted as a thick contour line. Yet another possible feature would be *a priori* topological information about the map as a whole, built and updated as the contour lines are restored.

These same features could also be used to detect and process other special cases, such as the saddle points and complex cliffs previously mentioned.

SELECTED BIBLIOGRAPHY

1. Agui, T. and Furukawa, T. **A Method for Extracting and Restoring of Contour Lines on a Contour Map.** *The Transactions of the IECE of Japan*: Vol. E 63, No 8 (August, 1980).
2. Amin, T. and Kasturi, R. **Map Data Processing: Recognition of Lines and Symbols.** *Optical Engineering*, 26, 354-358 (1987).
3. Baker, D., Gogineni, B., Sheppird, G., Williams, L. and Bethel, J. **LINETRAC - A Hardware Processor for Scanning and Vectorizing Contours, and Associated Systems Software for Automatically Generating Digital Terrain Elevation Data.**
4. Baruch, O. **Line Thinning by Line Following.** *Pattern Recognition Letters*, 8: 271-276, (1988).
5. Boyle, A. **Raster Scan Techniques in Computer Aided Cartography.** *XVI International Congress, Montreux, Switzerland* 504.4/1-504.4/10 (1981).
6. Ejiri, M., Miyatake, S., Matsushima, H., Kakumoto, S. and Shimada, T. **Automatic Recognition of Design Drawings and Maps.** *Proceedings of 7th International Conference on Pattern Recognition, Montreal*, 1296-1305 (1984).
7. Freeman, H. **On Searching a Contour Map for a Given Terrain Elevation Profile.** *Journal of The Franklin Institute*, 284:1, 1-25 (1967)
8. Jimenez, J., and Navalon, J. **Some Experiments in Image Vectorization.** *IBM J. of Research and Development*, 26, No. 6: 724-734 (1982).
9. Karima, M., Sadhai, K. S., and McNeil, T. O. **From Paper Drawings to Computer Aided Design.** *IEEE Computer Graphics and Applications*: 24-39 (1985).
10. Kasturi, R. and Alemany, J. **Information Extraction from Images of Paper-Based Maps.** *IEEE Trans. Software Engineering (Image Databases)*, 14: 671-675 (1988)
11. Kasturi, R., Fernandez, R., Amlani, M. and Feng, W. **Map Data Processing in Geographic Information Systems.** *Computer* (1989).
12. Kittler, J., Fu, K., and Pau, L. **Interpreting Binary Pixel Representations of Line-Structured Data.** *Pattern Recognition Theory and Applications. Proc of the NATO Advanced Study Institute held at St. Anne's College, Oxford. March 29-April 10* (1981).
13. Kong, T. Y. and Rosenfeld, A. **Digital Topology: Introduction and Survey.**
14. Leberi, F. and Olson, D. **Raster Scanning for Operational Digitizing of Graphical Data.** *Photogrammetric Eng. and Remote Sensing*, Vol 48, No. 4, 615-627 (Apr. 1982).

15. Martin, F., Cook, J. and Becker, S. **Automated Systematic Method for Forest Land Classification and Mapping.** *Montana Timber-Water Co-op Study* (1983).
16. Morean, O. and Kasturi, R. **Application of Pattern Recognition Techniques to Process Cartographic Data.** *Applications of Artificial Intelligence*, J. F. Gilmore, ed., *Proc SPIE* 485, 211-213 (1984).
17. Morse, S. **A Mathematical Model for the Analysis of Contour-Line Data.** *JACM* V15 No2, 205-220 (April 1968).
18. Peuquet, D. **An Examination of Techniques for Reformatting Cartographica** 18: 34-48 (1980).
19. Smith, R. **Computer Processing of Line Images: A Survey.** *Pattern Recognition*, 20, 7-15, (1987).
20. Suzuki, S., and Abe, K. **Topological Structure Analysis of Digitized Binary Images by Border Following.** *Computer Vision Graphics Image Process.* 30, 32-46 (1985).
21. Suzuki, S., Kosugi, M. and Hoshino, T. **Automatic Line Drawing Recognition of Large-Scale Maps.** *Optical Engineering*, 26, no. 7:642-649.
22. Suzuki, S., Yamada, T. **Maris: Map Recognizing Input System.** *Proc. Int. Workshop on Industrial Applications of Machine Vision and Machine Intelligence.* 214-219 (1987).
23. Suzuki, S. and Yamada, T. **Maris: Map Recognition Input System.** *Pattern Recognition*, 23:8, 919-933 (1990).
24. Weber, W., **Computer-Assisted Cartography Research and Development Report of the Federal Republic of Germany 1984-1986.** *Nachrichten aus dem Karten und Vermessungswesen.* 46: 123-38 (1987).

APPENDICES

APPENDIX AData Structures

Following are some of the data structures used in the implementation of the program:

```
typedef struct {  
    BYTE x;  
    BYTE y;  
} point;
```

```
typedef struct {  
    point pt;  
    void * next;  
} cchain;
```

```
typedef struct {  
    BYTE end1type;  
    BYTE end2type;  
    point end1coords;  
    point end2coords;  
    float end1dir;  
    float end2dir;  
    int length;  
    cchain * chain;  
} cont;
```

```
typedef cont * contptr;
```

```
typedef cchain * cchainptr;
```

```
typedef struct {  
    int contid;  
    point intrsectpt;  
    void * next;  
} contlist;
```

```
typedef struct {  
    cchain mycontour;  
    contlist conts;  
    int edge;  
} cliff;
```

```
typedef contlist * contlistptr;
```

```

typedef cliff * cliffptr ;

typedef struct {
    int contid ;
    int end ;
    } contend ;

typedef struct {
    BYTE e1 ;
    BYTE e2 ;
    } endchd ;

endchd contendchd [ MAX_CONTS ] = { 0 , 0 } ;
unsigned char image [ 256 ] [ 256 ] ;
int adj [ MAX_CONTS ] [ MAX_CONTS ] = { 0 } ;
cliff cliffs [ MAX_CLIFFS ] ;
contptr contours [ MAX_CONTS ] = { NULL } ;

```

The structure *conts* is used to hold information about a given contour line. *Length* holds the length in pixels of the contour line. *Chain* is itself a structure of type *cchain*. It consists of a singly-linked list of *points*, which again is a structure consisting of the x and y coordinates of a given pixel. *Chain* contains the vector representations of the extracted contour line. *End1type* and *end2type* hold information about what condition stopped the tracing for this end of the contour. This condition will be either that a given endpoint runs off the edge of the image, ends in the middle of the image, or becomes too wide (enters a cliff). *End1* is always the head of the linked list *chain* and *end2* is always the tail. *End1coords* and *end2coords* hold the coordinates of these two endpoints, while *end1dir* and *end2dir* hold the direction of the respective ends of the contour line. Finally, an array *contours* is used to hold all of the *cont* structures, one for each extracted contour line.

The above mentioned directions of the endpoints of the lines is critical for the algorithm, as it is the sole criteria for finding the breakpoints. The

direction is determined by backing off a set number of pixels from the end of a line, and taking the direction of the end to be the direction from this pixel to the end. Because of possible distortion when the twins encounter a cliff, the last few pixels on the ends of contour lines may not accurately represent the center of the contour line. Therefore the last few pixels of any contour line end that enters a cliff are rejected.

The choice of using a singly-linked list to represent the extracted contours was based on memory limitations in the target machine. An even more memory efficient, although somewhat less versatile method would be to use chain codes instead of the linked list. Conversely, a doubly-linked list would facilitate contour manipulations, such as joining two contour lines, at the expense of more memory. As implemented, the algorithm must make special cases of instances wherein the heads of two lists or the tails of two lists must be joined. In particular, in the above two cases, one of the lists is reversed, making the head the tail. Then the two contours can be joined as normal; the tail of one to the head of the other.

The type *cliff* holds information about each cliff. *Mycontour* contains the vectorized perimeter of the cliff, and *edge* is a boolean that is set if the cliff borders on the edge of the image. *Conts* is itself a linked list of structures used to hold information about each contour that enters the cliff. Each of these structures store the number that uniquely identifies the cliff and the coordinates of approximately where the contour enters the cliff. Again, an array *cliffs* is used to hold the cliff structures, one for each cliff detected.

The structure *endchd* stores information about whether the endpoints of a given contour line have changed or not. This is necessary because information about the endpoints of a contour may change between the time when

it is stored in a given *cliff* structure and when it is processed. This can be illustrated in Figure 55, which shows a series of cliffs with contour lines entering them. In this figure, cliff I has recorded the fact that end 2 of contour A enters it. Cliff II knows that end 1 of both contour A and of contour B enter it. Cliff III knows that end 2 of contour B enter it. After processing cliff II, line A will be joined with line B. This joining will result in lines A and B pointing to the same structure, a single line that runs from the original end 2 of line A to end 2 of line B. However, in joining the lines, either A or B must first be reversed, which involves swapping its endpoints. Therefore, either cliff I or cliff II will now have some misinformation about the contour entering it -namely it will think the wrong endpoint enters it. The array *contendchd* therefore must keep track of when this situation occurs, to

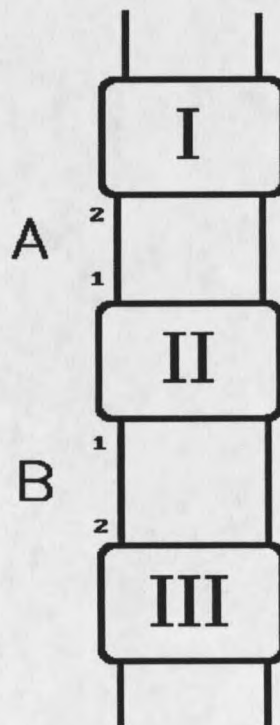


Figure 55. Contour lines with both ends entering separate cliffs.

keep wrong ends of a reconstructed line from being joined during further reconstruction.

The array *adj* is used to store adjacency information about the contours for use in the adjacency method. Since I am storing information about the length of each contour in a separate structure, this is the only array needed. The normalizing of the adjacency table can be done in place, thereby avoiding the two separate arrays, one of them of type float, used in the original algorithm of Agui and Furukawa [1].

APPENDIX B

Hardware

The program is implemented in Turbo C on a DOS based system. The target machine is a 16 Mhz 80286 with VGA graphics capabilities, 2 Mega-bytes of random access memory, and a 80287 math co-processor. Although the program is rather compute intensive, most computations involve only integers, so the co-processor offers only a slight increase in performance.

Images are digitized with a simple hand-held digitizing device, and saved initially in raster format, 1 bit per pixel. A utility program expands this format to one byte per pixel, setting all background pixels to 0 and all foreground pixels to 255. The image is read from disk into a two dimensional array $I[maxX][maxY]$. The current program will only handle images of size up to 256 X 256 pixels due to memory limitations of the system. However, the only non-ANSI portions of the program are the calls to the Turbo-C graphic routines to initialize the display device and to display a pixel; hence the program could easily be ported to other platforms.

MONTANA STATE UNIVERSITY LIBRARIES



3 1762 10182522 0

HOUCHEN
BINDERY LTD
UTICA/OMAHA
NE.