An Algorithm for Extracting Contour Lines Based on Interval Tree from Grid DEM

WANG Tao

Abstract This paper proposes a new algorithm for determining the starting points of contour lines. The new algorithm is based on the interval tree. The result improves the algorithm’s efficiency remarkably. Further, a new strategy is designed to constrain the direction of threading and the resulting contour bears more meaningful information.

Keywords algorithm; contour; grid DEM; threading; interval tree

CLC number P208

Introduction

The contour line is an elegant tool for representing geomorphology on analogue media and in a digital environment (Mark, 1997). It has significant advantages over Grid and TIN in that it can convey qualitative and quantitative information in parallel. Due to the fast development and maturation of geographic data acquisition technology such as InSAR and LIDAR, digital elevation data in matrix form is becoming the most popular data source for representing terrain. As a primary function in computer-aided cartography and GIS, extraction of contour lines from Grid DEM has been studied for a long time.

The first type of existing algorithms extracting contour lines from a grid involves approximating a polynomial surface in a local or global manner. This method has the benefit that the resulting contour lines do not need smoothing further. However, the mathematical surface is hard to define and the implementation is not direct (McCullagh, 1988). The second type interpolates points at given heights between grid cells and connects them sequentially. This has been widely accepted in practice (Watson, 1992; Jones, et al, 2000); hence, there are various implementations of this method. The most often used has about two steps: find a starting cell and trace in the grid until going back or reaching the boundary. There exist a lot of results on how to improve the efficiency of the tracing process. This paper investigates the first problem, which has not been recognized carefully in existing research. It also puts forward new ideas on how to find the starting cell more efficiently. Another new result is that the contour lines extracted by our algorithm are in the same direction, which provides more information for further terrain analysis based on contour.

1 New algorithm based on interval tree

When extracting a continuous contour line of a given height from an elevation grid matrix, we first need to find a starting point which is located on some
edge bounded by two adjacent grid points. The given height must be higher than one grid point and lower than the other. In order to find the starting point, traditional algorithms scan the edges of the elevation matrix one by one, horizontally or vertically. This strategy is very inefficient in both theory and practice.

1.1 Extraction of directed contour

In a regular elevation grid, an edge between two neighboring grid points can be treated as an interval in the elevation dimension. If we assume that the x-axis is leftward and the y-axis downward, there are three kinds of elevation intervals in the x positive direction: the left point is higher, equal or lower than the right one. Considering the traversal of a contour line in the elevation grid, we can see that a closed contour line must intersect with at least two edges paralleling with the x-axis and that there are half of them with the characteristic that the left point of the interval is lower than its right. So these in the half subset can be selected as candidates to find starting points of contour lines. The condition clause to determine if there is a point with elevation \(Z\) on a given interval of \([Z_1, Z_2]\) is revised from the commonly used as:

\[(Z - Z_1) \times (Z - Z_2) < 0\]

to:

\[Z > Z_1 \text{ and } Z < Z_2\]  \hspace{1cm} (1)

or to:

\[Z < Z_1 \text{ and } Z > Z_2\]  \hspace{1cm} (2)

Now there are only comparative operators left and the candidates are reduced by roughly 50 percent. If Eq.(1) is used and the threading direction is further restricted downwards, the left adjacent region of all resulting contour lines is higher than itself and the right lower. As indicated in Fig.1, when looking for the heading point of the contour line with height 8, we can choose the one located between edges 7 and 10, according to Eq.(1). Further, downward tracing makes the new contour line higher than the right neighboring grid points and lower than the left neighboring grid points.

There is a further expectation to improve the efficiency of the querying for massive grids. We propose that the index on the edges of the elevation dimension be used to accelerate the first operation. Here, the interval tree suggested in the Kreveld’s algorithm for extracting isolines from TIN (Kreveld, 1996) is adapted. Preprocessing of building the index from the elevation matrix produces the intermediate result, which can be employed when determining the contour lines’ starting point.

1.2 Extracting of contour lines based on interval tree

Interval tree is a binary search tree structure which stores sorted intervals in each node and can improve the efficiency of querying on intervals (Preparata and Shamos, 1985). An interval stored in the tree is a pair of values in which the first is smaller than the second. Here, one horizontal edge in a grid which is bounded by two elevation values is treated as an interval. As indicated in the above analysis, only part of the set of edges whose left is lower than the right is stored in the interval tree for further querying. These are named as candidate intervals.

The construction of an interval tree with an elevation interval is similar to that of a binary tree. Fig.2 shows nine intervals stored within the interval tree. From the tree at the right, we can see that the key values are important to the result. The selection of key values should be considered because it affects the balance and depth of the resulting tree and the number of intervals on each node. The efficiency of querying depends on the latter three factors. There are five steps in our implementation, as follows:

1. Get the histogram of the elevation of the left (or right or middle) points of candidate intervals;

2. Set a value \(M\) as the maximum number of intervals that can be stored in each tree node. Break the histogram into parts where each part has \(M\) intervals at most. Get the series of elevation values that breaks the parts;
(3) Build up an ordered binary tree based on the series of elevation values which is taken as a result of the in-order traversal;

(4) Insert all candidate intervals into the binary tree. For each interval, take the root of the tree as current node. If the left value of the interval is smaller than the current node’s key value while the right is greater, or the current node is a leaf, store this interval in the current node. If the left and right values are both smaller (greater), take the left (right) child node as current node, then perform the previous operation until the interval is inserted into the tree; and,

(5) Sort the interval list in each tree node into ascending order by the left points’ elevation. A typical interval tree may store another interval list ordered by the right points’ elevation; however, this doubles the storage space; Hence, only the first one is kept with some sacrifice of query efficiency.

When there is a need to find starting points of contour lines with given elevation, the first step is to search for eligible intervals from the interval tree by the value. The querying process is mostly similar to the construction as follows:

(1) Take the root of the tree as current node;

(2) Identify intervals which contain given value in current node if the given value is not smaller than the value of the left end of the first interval;

(3) If the current node is a leaf or given value is identical with the key value of the current node, stop querying; and,

(4) Take the left (right) child as current node if the given value is smaller (greater) than the key value of the current node then go back to the second step.

All the identified intervals which have not been visited during the threading of contour lines of given elevation can be taken as heading intervals to interpolate starting points. In our implementation, only horizontal edges whose left elevation is lower than its right are indexed. As such, the process of extraction should check up the boundary edges of the elevation grid for the open contour lines ahead of extracting closed contour lines.

2 Conclusions and discussion

The traditional strategy of finding starting points of contour lines needs $O(N)$ time. The algorithm based on interval tree needs $O(N\log N)$ time for preprocessing and $O(\log N)$ time for query during contour lines threading.

Our program is implemented using Visual Basic 6.0 and is compiled under Windows XP. Three datasets (indicated in Table 1) have been used for testing the straightforward algorithm and the proposed algorithm with an interval tree. Table 2 gives the CPU time costs of the program on Pentium 4 2.5G, RAM 512MB with three datasets. Each cell in the last three rows with different $M$ values has two values. The one before the slash is time needed for preprocessing and extracting, which is generally greater than the time of the traditional algorithm without index as shown in Table 2. The one after the slash is time needed for extracting based on the existing index, which is obviously reduced compared to that of the traditional algorithm. The value in PpT column is preprocessing time needed for constructing the interval tree.

| Table 1 Three testing datasets |
|-------------------------------|
| Grid size | Elevation range/m | 50 m | 100 m |
|------------|--------------------|------|-------|
| A | 1 201 by 1 201 | 1 732-5 757 | 4 202 lines, 1 386 482 points | 2 111 lines, 693 133 points |
| B | 1 201 by 1 201 | 791-1 568 | 13 781 lines, 959 326 points | 6 789 lines, 478 911 points |
| C | 2 402 by 3 603 | 16-3 059 | 40 437 lines, 3 990 461 points | 18 352 lines, 1 977 028 points |
Table 2 Comparison of efficiency of algorithms without index and with interval tree/D.S

| Elevation interval | A     | B     | C     |
|--------------------|-------|-------|-------|
|                    | 100 m | 50 m  | PpT   | 100 m | 50 m  | PpT   | 100 m | 50 m  | PpT   |
| No index           | 1.51  | 3.0   | 0.41  | 0.79  | 6.58  | 13.37 |
| M=10               | 1.98/0.46 | 2.53/0.92 | 1.57 | 2.17/0.24 | 2.47/0.48 | 1.96 | 20.65/1.71 | 21.98/3.57 | 18.67 |
| Interval tree      | 2.53/0.92 | 1.57 | 2.12/0.25 | 2.39/0.51 | 1.86 | 19.78/1.89 | 21.57/3.80 | 17.83 |
| M=1000             | 1.84/0.51 | 2.18/0.98 | 1.26 | 2.15/0.26 | 2.40/0.54 | 1.88 | 19.11/2.00 | 19.59/4.14 | 16.28 |
| M=5000             |       |       |       |       |       |       |       |       |       |

*PpT denotes preprocessing time.

The dimension of testing datasets A and B are equal. However, the elevation range and the count of nodes of the resulting contour lines of dataset B, which locates in the loess area, are much less than those of dataset A, which locates in the alpine area. However, the count of contour lines of B is much more than A. Table 2 shows that the preprocessing of B needs more time than A. Table 2 also shows that when the maximum number accommodated in each tree node (M) is greater, the preprocessing time is reduced and the extracting time is raised slightly.

Our new algorithm-based interval tree remarkably improves efficiency. However, the index structure consumes a lot of memory because the ordinary elevation grid is massive. The next step should investigate the relation of terrain characteristics with contour lines. A quick improvement can consider the roles of terrain-specific lines, which must intersect with every contour line. The candidate intervals can be reduced remarkably. For example, the candidate intervals of dataset B can be reduced from 48% to 11% but it needs more preprocessing time to extract terrain-specific lines. Another characteristic that the elevation difference of two neighboring contour lines is smaller than one elevation interval may be used to improve the algorithm’s efficiency.

References

[1] Wu H H, Gong J Y (1997) Spatial data structure and processing of GIS[M]. Beijing: Surveying and Mapping Houseing (in Chinese)
[2] Mark D (1997) The history of geographic information systems: invention and re-invention of triangulated irregular networks (TINS)[C]. Proceedings of GIS/LIS’97, Cincinnati
[3] McCullagh M (1988) Terrain and surface modeling systems: theory and practice[J]. Photogrammetric Record, 12(72): 747-779
[4] Paul B (1987) CONREC: A contouring subroutine[J]. Byte: The Small Systems Journal, 12(6): 143-150
[5] van Kreveld M (1996) Efficient methods for isoline extraction from a TIN[J]. International Journal of GIS, 10: 523-540
[6] Jones N, Kennard M (2000) Fast algorithm for generating sorted contour strings[J]. Computers and Geosciences, 26: 831-837
[7] Preparata F, Shamos M (1985) Computational geometry-an introduction[M]. New York: Springer-Verlag
[8] Hennig T A, Kretsch J L (2001) The shuttle radar topography mission[C]. Proceedings of DEM, Manno