Documentation for a Contouring Strategy

Problem Statement:

To extract a region (with a set of hole cycles) coordinates from a 2D grid. A grid is in ArcGrid format. A file in ArcGrid format first has header(attribute) information about the 2D grid and then followed by a grid itself. Header (attribute) information has each dimensional length, bottom left starting coordinates, a cell size and no data value. In our case, a cell represents some square area. The value in each cell is the intensity of the area represented by that cell. If this value is zero then the area represented by that cell represent an empty area. Each connected set of cells with same intensity represents a region of that intensity. A region can have holes, this means that in an interior of a region there can be a cells of other intensity or intensity value zero. So, problem is extract each such region with a set of hole cycles.

Few Definitions:

Pair Flag: This flag is set while looping around the outer cycle. It is a flag meant for boundary cell at each vertical level. A Flag is true if there exists an interior at that vertical level. So, we care about this flag only when we are going down or up.

Hole flag: The flag is set to true for all upward and downward cells of the hole. A cell is an upward cell (downward cell) if it is encountered while going up (down) during hole boundary traversal.

Figure1: Pair flag for first region in black **Figure2**: Hole flag for first region in black

Boundary Flag: For each boundary cell this flag is true.

Visited Flag: This is for boundary cells only. This flag is false for each boundary cell initially. As we visit through the boundary cells this flag is set to true.

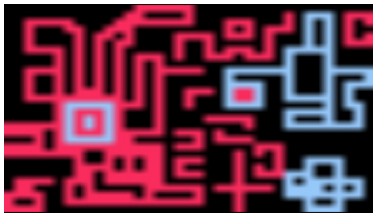

Figure3: Boundary flag for whole grid in respective intensity color

Strategy:

Input we have for an algorithm is a 2D grid and all associated attributes. First we figure out the boundary cell in the input and set the boundary flag for them and set the visited flag for each of the boundary cells to false. This is a first pass. In second pass we stop at cell which has boundary flag set to true and visited flag set to false. Whenever we encounter such type of cell we apply our two step algorithm at that cell otherwise we just skip through the cells. The two step algorithm is described below.

A Two Step Strategy for Every Encountered Outer Cycle:

We come to this stage when the cell encountered is a starting point of the outer cycle. So we do the following things during first step:

Step 1:

We are at a starting point of an outer cycle.

a) We loop around (counter-clockwise order) the boundary cells (Outer Cycle) and write to file the coordinates of end of segments.

b) While looping we set the pair flags for each boundary cells. A pair flag is set for all the boundary cells that are encountered while going down and up. And whenever the cell is encountered twice while going up and down we toggle this flag. The pair flag is shown for first region in figure 1.

c) And we also set the visited flag for each boundary cell to true. This is to avoid reporting half regions.

The second step is to find the holes inside the outer cycle. We loop around the outer cycle of the region again and do the following:

Step 2:

a) At each boundary cell that has pair flag set to true and direction of traversal is downward we stop at that cell and go to next step.

b) We scan the horizontal extent of the region at that vertical level. How do we know what is the extent of the region at that vertical level? For that we traverse in horizontal direction till we find the cell that has a pair flag set to true. While scanning the extent if we encounter a boundary cell that has same intensity as outer cycle we stop at that cell and go to next step(c).
If while scanning through the extent we find the cell with hole flag set to true we skip the horizontal extent of the hole. Extent of a hole is bounded by cells with hole flag set to true. After skipping the hole extent we continue scanning again.

c) We know that this cell is a starting cell of the hole cycle. We loop around (counter clock wise order)the boundary cell of the hole and write to file the coordinates of the end of segments. While looping we set the hole flag for all the cells that are encountered while going up and down. Also we set the visited flag for all cell in loop to true. After reporting a hole we continue with step above(b)

d) If the horizontal extent of the region is complete. Means a scanning in step (b) is complete we start with the step (a) all over again for all the next encountered cells.

Complexity:

1) To figure out the boundary cells one pass through a whole grid. O (n)
2) I will talk about the worst case: A concentric cycles of alternate or different intensity is one of the worst cases. In This case except the inner most cycle every other cycle is traversed twice. Since these cycles are reported once as a hole and once as an outer cycle. So the worst case would have complexity less than O (2n).
3) So the overall complexity is O (n) +O (2n) = O (n).

Conclusion:

I had manually checked the output for the rohit.ArcGrid file. The output is correct. I think this is a simple and most intuitive approach.

Pseudo Code:

These algorithms are member functions of the contour class therefor they have access to the input grid and all its attributes.

Algorithm 1:

get_outercycle (int x, int y, ofstream* outfile)
INPUT: Coordinates of the cell where the outer cycle starts (x, y). And pointer to an outfile where we write the list of coordinates.
OUTPUT: Write to file the end point coordinates of the segments. And Also set the pair flag (an attribute of the cell) for the boundary cells of the outer cycle.

```
1)  Int i =y, j=x, pdirection=0, cdirection=2;
2)  do
3)      if(cdirection==1)
4)        While(last cell in direction 1 is not reached)
5)            Set cell. Visited flag = true;
6)            Toggle the cell. Pair flag;
7)             j++;
8)      else if(cdirection==2)
9)        While(last cell in direction 2 is not reached)
10)           Set cell. Visited flag = true;
11)           Toggle the cell. Pair flag;
12)           i++;
13)     else if(cdirection==3)
14)       While(last cell in direction 3 is not reached)
15)           Set cell. Visited flag = true;
16)           Toggle the cell. Pair flag;
17)           j--;
18)     else if(cdirection==4)
19)       While(last cell in direction 4 is not reached)
20)           Set cell. Visited flag = true;
21)           Toggle the cell. Pair flag;
22)            i--;
23)
24)     pdirection=cdirection;
25)     If(pdirection==1)
26)       if(lower neighbor is boundary cell of same intensity )
27)           output to file the coordinated of lower left corner of the cell.
28)           cdirection = 2;
29)       else if(upper neighbor is boundary cell of same intensity)
30)           output to file the coordinated of lower right corner of the cell.
31)           cdirection = 4;
32) // similarly set cdirection for all other pdirections (2, 3, 4) and output to file the respective coordinates.
33)
34)     If(pdirection==cdirection)    // means U turn
35)       If(pdirection==1)
36)           output to file the coordinated of lower right corner and upper right corner of the cell.
37)           cdirection = 3;
38) // similarly set cdirection for all other pdirections (2, 3, 4) and output to file the respective coordinates.
39)
40)
41) while(i!=y&&j!=x )
```

Algorithm 2:

Algorithm is same to algorithm 1 except the output to file is not there and there is a modification from line number 10 an 11. I am just providing the modification to these two lines only. The following set of lines should be embedded into algorithm 1 to get a full algorithm 2.

get_holecycles (int x, int y)
INPUT: Coordinates of the cell where the outer cycle starts (x , y). And pointer to an outfile where we write the list of coordinates.
OUTPUT: Write to file the end point coordinates of the segments. And Also set the pair flag (an attribute of the cell) for the boundary cells of the outer cycle.
// INTENSITYLEVEL is an intensity of outer cycle // holeid take care of hole count

```
1)      else if(cdirection==2)
2)        While(last cell in direction 2 is not reached)
3)          If(cell[y][x]. Pair == true)
4)          int q=x;
5)            do
6)              If(cell[y][q].hole==true)
7)                do
8)                  q++;
9)                while(cell[y][q].hole!=true)
10)             if(cell[y+1][q+1].intensity!=INTENSITYLEVEL&&cell[y][q+1].pair!=true)
11)               get_holecycle (y, q, holeid, outfile)          // Get hole cycle function
12)               holeid++;
13)               while(q==x||!(cell[y][q+1].hole==true||cell[y][q+1].pair==true))
14)                 q++;
15)           q++;
16)           while(cell[y][q].pair!=true)
```

Algorithm 3:

get_holecycle (int x, int y, int holeid, ofstream* outfile)
INPUT: Coordinates of the cell where the boundary of hole cycle starts (x, y). And pointer to an outfile where we write the list of coordinates.
OUTPUT: Write to file the end point coordinates of the segments. And also set the hole flag (an attribute of the cell) for the boundary cells of the hole cycle

This algorithm is also similar to get_outercycle. The difference is that this provide the inner cycle of the cells whereas get_outercycle provide the outer cycle of the boundary cells.
Another function it serves is that is sets the hole flags whole looping the hole.

In a following image red color is the output of get_holecycle and green color id the output of get_outercycle.