An Efficient Algorithm for Raster-to-Vector Data Conversion

Junhua Teng[1], Fahui Wang[2], Yu Liu[3]

[1] State Key Laboratory of Satellite Ocean Environment Dynamics, Second Institute of Oceanography, State Oceanic Administration, Hangzhou,310012, China
E-mail: htgis@163.com
[2] Department of Geography and Anthropology, Louisiana State University, Baton Rouge, LA 70803, USA;
[3] Institute of Remote Sensing and Geographical Information Systems, Peking University, Beijing 100871, China

Abstract
Data conversion from raster to vector (R2V) is a key function in Geographic Information Systems (GIS) and remote sensing (RS) image processing for integrating GIS and RS data. The R2V module is available in commercial RS software packages, but there is still room to improve the computation efficiency. This paper presents an efficient R2V algorithm that processes large images and automatically builds GIS topology while scanning image lines one by one. The new algorithm, termed Two-Arm Chains Edge Tracing (TACET), has several significant advantages. First, it converts all types of area objects of RS classification in only one processing cycle. Secondly, it constructs complete area topological relationship by recording the shared edge between two polygons only once. Finally, it is scalable when processing large images. The program based on the algorithm is faster in processing large RS images with comparison to commercial software such as ENVI.

Keywords
raster to vector, data conversion, Two-Arm Chains Edge Tracing (TACET) algorithm, *LittleHuskie R2V*

I. INTRODUCTION

Raster to vector (R2V) data conversion or *vectorization* is a key function in geographical information systems (GIS) and remote sensing (RS) image processing for data integration between RS and GIS (Mattikalli, et al., 1995). In general, there are two types of R2V algorithm, namely line vectorization and polygon vectorization. *Line vectorization* is often used in CAD for digitalizing images from paper media, e.g., extracting contour lines from scanned topographic maps. During this process, linear elements on a map are automatically recognized and represented in a vector format. Numerous algorithms of line vectorization are developed, and many software packages such as AlgoLab (http://www.algolab.com/r2v.htm), RasterVect (http://www.rastervect.com) and Scan2CAD (http://www.softcover.com) are available to cater the large CAD and art design markets. *Polygon vectorization* is often used in extracting geographical features from RS imagery classification results. One application of polygon vectorization is to update land use database from RS data. This paper focuses on polygon vectorization.

Several important existing studies on polygon vectorization are highlighted here. Congalton (1997) discussed the importance of R2V or V2R in theory, and explored and evaluated the outcomes of R2V and V2R conversions. Lou et al. (2005) discussed the problem of R2V conversion based on classified remote sensing images, and recognized the difficulties of handling all classes of geographical objects simultaneously in the process. Riekert (1993) proposed an algorithm of extracting area objects from raster image data, which proved to be significant improvements over previous algorithms. At present, the R2V module is included in many RS software packages such as ENVI. However, there is still room to improve the algorithm to enhance its performance. This research builds upon the work by Riekert (1993) and makes further improvements and clarifications. Its advantages become evident in handling complex large images.

The R2V conversion is an important task for at least three reasons as explained in the following.

- R2V is a time-consuming process and demands high computing power. The faster the algorithm is, the larger image it can handle. Generally, the time complexity of a R2V algorithm is $O(n*m)$, where n and m denote the size of an image, because each pixel needs to be accessed at least once to be determined whether it is a node or a vertex.
- The speed of a R2V algorithm is also determined by spatial complexity. As the resolution of an image becomes sharper, the data volume increases. This may either significantly slow down the display of an image or prevent it from being wholly loaded due to limited computer capacity. It is critical to find an approach with reduced spatial complexity.
- In order to make R2V results ready for GIS users, it is desirable for the vector data to contain all types of classification in one file instead of multiple files with each type saved in one single file. It is also more convenient to manage the complete topological relationship among classes. RS data have become a fast and convenient source for updating spatial database. Take land use changes as an example. If all land use classifications are saved in one data set with topology, it is easy to edit one area and maintain consistency in adjacent areas.

II. BASIC CONCEPTS IN R2V ALGORITHM

As discussed in the previous section, R2V includes line vectorization and polygon vectorization. Line vectorization focuses on line thinning by converting bold lines in a digital scanned image into skeleton lines. This paper focuses on polygon vectorization. Polygon vectorization may be also related to line vectorization. For instance, one may use line vectorization to extract polygon edges first, and then export the derived edges into a GIS software package for further processing. However, this approach is rarely used because of its low transformation accuracy and efficiency. In contrast to line vectorization where line features are extracted based on center points of pixels, polygon vectorization defines boundaries by edges between pixels.

In a polygon R2V algorithm, three concepts, namely pixel, node, and topological relation, are essential components. First, a pixel is the basic element of a RS image. Second, a 2×2 window of pixels can be defined such that a node or an edge can be identified. Boundaries among polygons are created by connecting the central points of such node or edge windows. In this research, we used "edge tracing" to name this process. During edge tracing, the topological relation between two neighboring polygon is constructed simultaneously.

A. Pixels and coordinates

A pixel in a raster image is considered as a primitive element. It is treated as a polygon (square) in the vector format. Every pixel in an image has a certain resolution. For example, the resolution of Landsat TM image pixel is 30×30 meters. By transforming the raster into vector, a pixel becomes a 30×30 square. Figure 1 uses an example to show how pixel positions in an image are transformed into vector coordinates. All arcs that define polygon boundaries are orthogonal.

B. Edges and nodes

In polygon vectorization, it is essential to delineate boundaries between polygons. Clearly, polygon boundaries may not be in homogeneous areas in an image. Hence, based on a 2×2 window, we can identify six cases of edge connections (Figure 2(a)—(f)) and eight cases of node connections (Figure 2(g)—

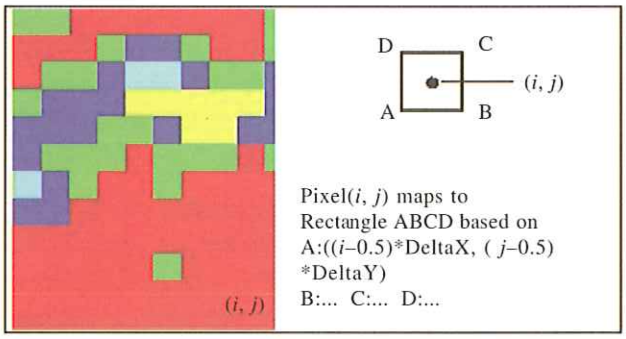


Figure 1. Pixels transformed into polygons according to their positions in an image

(n)) as shown in Figure 2. The common edge of two neighboring polygons consists of a sequence of edge connections. Two or more edges intersect and form a node, at which we should start or complete an edge tracing operation.

In the above figure, a, b, c and d stand for different values (e.g., land cover types) in a classified image. These 14 cases can be employed as templates to find edges and nodes in an image. In the proposed algorithm, the last three cases are considered the same as the 7th case (Figure 2(g)) since all result in four polygons.

C. Topological relations

Based on the extracted nodes and edges, topological relations among polygons can be built. In the proposed algorithm, the topological relation between two neighboring polygons is based on their common edge. An edge has two sides (left and right). Each corresponds to a polygon on the left or right of the edge. One special case is that a bigger polygon may contain some smaller ones. For example, A includes B in Figure 3(a). In such cases, the inner polygons are called islands or holes, whereas the outsider one is called contour. One polygon may have only one contour edge, but multiple ($N \geqslant 0$) hole edges.

III. THE TWO-ARM CHAINS EDGETRACING(TACET) ALGORITHM

A. Bitmap method

It is possible that a R2V algorithm traces all edges of different types of geo-objects in one loop. However, complex

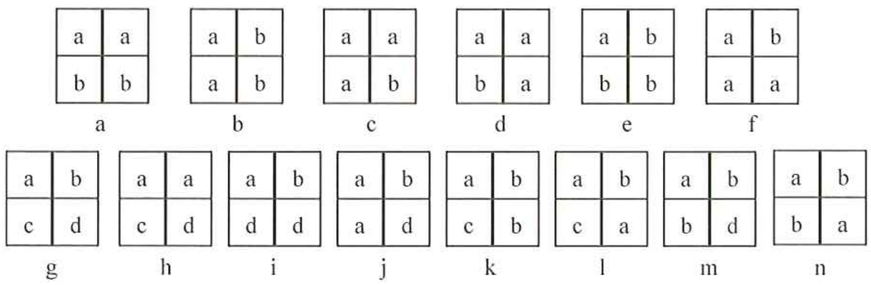
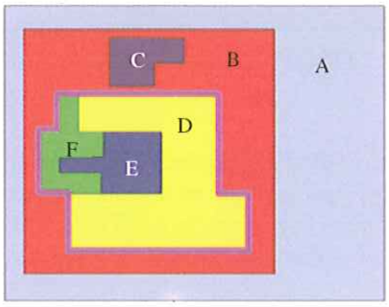


Figure 2. Six types of edge connections (top) and eight types of node connections (bottom)

relationships among polygons may complicate the process. In Figure 3(a), after single polygons (i.e., A, B, C, D, E and F) are traced, the next step is to determine the topological relationships among them. Polygon B is a complex one with one contour edge and four hole-edges of C, D, E and F.

In order to handle this type of problems, Lou et al. (2005) in ENVI 4.2 introduced a preprocessing method to create a binary image at first. In one single processing loop, it turns one particular value of pixels to 1 and all other pixels into 0 as background. By doing so, the whole image becomes a binary (0 or 1) bitmap. Since there are multiple values for pixels in the original image, it takes several loops to complete the whole process. For the case in Figure 3(a), it needs six loops to represent six values (A, B, C, D, E and F). Figure 3(b) shows the preprocessing result of class B, where polygons D, E and F in Figure 3(a) are converted into background and merged into one type. We propose an improved algorithm as discussed below.



(a) Contour and island

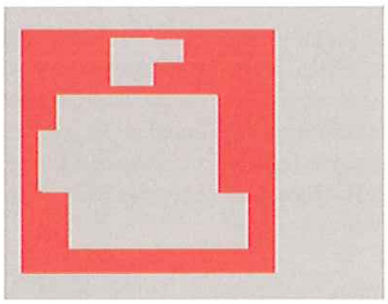

(b) Feature B extracted in a multiple-loop R2V

Figure 3. Topological relations in R2V

B. Improvements

(i) Plane-sweeping process

A scalable algorithm is developed in order to process large image files in R2V. Regardless of the image size, the algorithm loads only two lines of image at a time for tracing. Figure 4 shows that only the grey area from a whole image is need in a single iteration of R2V. A 2-line floating window is loaded to the active memory, and it moves from the top line to the bottom to complete tracing a whole image (*plane sweeping*).

(ii) Topology

In order to build topological relations among polygons, an edge is considered to have two side skins with one central stick. In Figure 5, blue lines represent two side skins to form polygon topology, and red lines are sticks to define edges' geometry. Two side skins are connected to each other while tracing polygons. In Figure 5, four polygons are formed, namely E, E, F and the outer-line polygon. The outer-line polygon is identified with the support of two side skins.

(iii) One-looped process

This R2V algorithm scans from the top line to the bottom line by a 2-line floating window in one cycle. All polygon edges are traced, and the polygon topology is built simultaneously. It is a fast and efficient algorithm termed Two-Arm Chains Edge Tracing (TACET).

C. Key concepts of two-arm chain edge tracing

(i) Two-arc chain polygon

In this new algorithm, each polygon is completed by two arc chains. As shown in Figure 6, in the tracing processing, the top left node is defined as the start of a polygon, and the bottom right node is defined as the end of the polygon. It is coded in C++ as class CGisPolygonR2V, in which m_LeftListOfArcs and m_RightListOfArcs record the pointers of arcs and correspond to the two side skins of arcs such as:

```
Class CGisPolygonR2V
{
   long int m_PolygonSystemID
   CObArray m_LeftListOfArcs;
   CObArray m_RightListOfArcs;
};
```

(ii) Arcs

As discussed earlier, an arc in this new algorithm has one central stick with two side skins. The stick is used to trace an arc's geometrical feature by recording the coordinates of

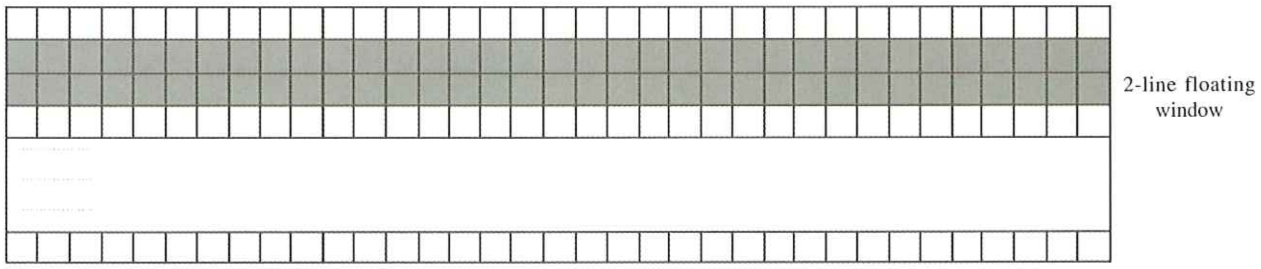


Figure 4. Two-line floating window in R2V

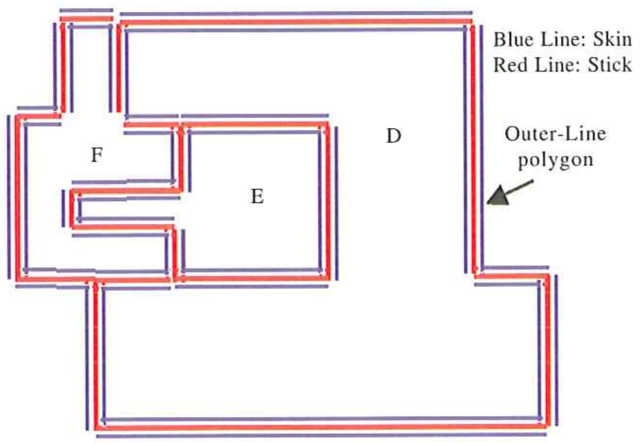


Figure 5. Two side skins tracing to form polygon topology

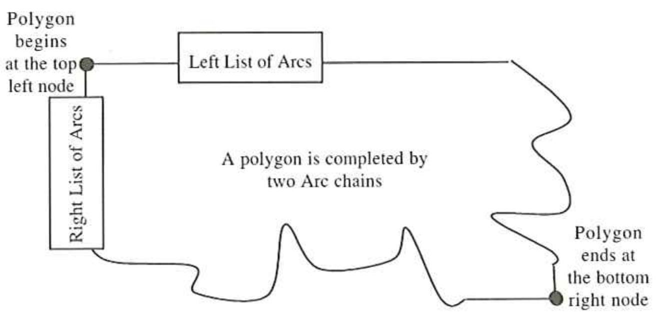

Polygon begins at the top left node
Left List of Arcs
Right List of Arcs
A polygon is completed by two Arc chains
Polygon ends at the bottom right node

Figure 6. Two arc chains are used in polygon tracing

vertices. The two side skins, defined as left skin and right skin, are used to trace polygons. When one arc ends (e.g., at a node), its pointer is added to the polygon arc chains, i.e., the left skin is added to *m_LeftListOfArcs* and the right skin to *m_RightListOfArcs*. The C++ codes are:

```
Class CcsArc
{
  long int m_ArcSystemID;
  CObArray m_ListOfVertexCorrdinate;
};
```

(iii) Two-Arm Chain

A comprehensive class *CHtArms* is used to record the tracings of both arcs and polygons. Since the R2V algorithm is a plane-sweep process, the tracing is always from left to right in the horizontal direction and from top to bottom in the vertical direction. Figure 7 shows all member variables in the class *CHtArm*. Two arms are used to trace polygons and arcs: the vertical arm *m_pArcVerticalArm* and the horizontal arm *m_pArcHorizonalArm*. The concepts of virtual arm and solid arm are introduced here, if the pixel has the same value with the above one, then the horizontal arm *m_pArcHorizonalArm* is virtual, otherwise is solid, and in the same way, if the pixel has the same value with the left one, the horizontal arm *m_pArcHorizonalArm* is virtual, otherwise solid. Considering that the whole two-dimension space is divided into three parts by the two arms, three variables, namely *m_pAbovePolygon*, *m_InsidePolygon* and *m_pLeftPolygon*, are used to record

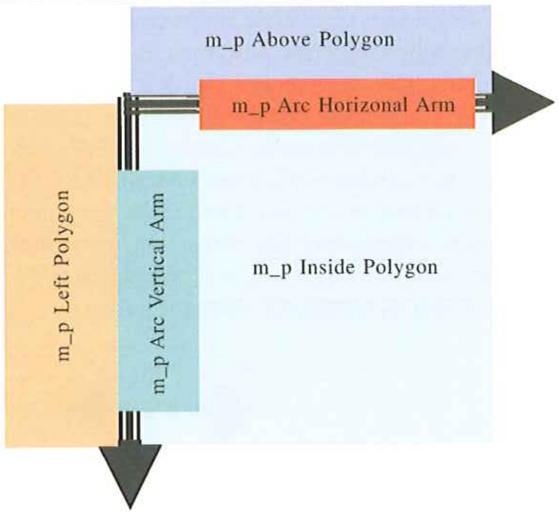


Figure 7. An instance of *CHtArm* class

the above polygon, inside polygon and left polygon respectively. Corresponding to the side skins of arcs (i.e., arms), these polygons are necessary for building the topology simultaneously with edge tracing. The variable *m_PixelValue* records a pixel's value (e.g., land use type), and the variable *m_lColPos* records the pixel's column position. The C++ definition of the Two-Arm Chain class is:

```
class CHtArm:public CObject
{
public:
     long int           m_PixelValue;
     long int           m_lColPos;
     CGisPolygonR2V     *m_pInsidePolygon;
     CGisPolygonR2V     *m_pAbovePolygon;
     CGisPolygonR2V     *m_pLeftPolygon;
     CCsArc             *m_pArcVerticalArm;
     CCsArc             *m_pArcHorizonalArm;
};
```

D. Connecting rules for two-arm chains

Since topological relations among edges and polygons are built in each arm, the whole R2V process can be implemented by connecting the arms. This was not made clear in Riekert (1993). The following explains the connecting rules for the Two-Arm Chains algorithm.

(i) Rules for creating polygons

In this new algorithm, a polygon is traced by beginning from the top left node and ending at the bottom right node. It should be noted that the outer polygon is created when the last node (i.e., the bottom right pixel) of the whole image is reached.

(ii) Rules for passing topological relations among neighboring CHtArm objects

When tracing from one arm to another from left to right and from top to bottom, topological information is passed between them. Two other two-arm objects (i.e., *pTwoArmPrevious* and *pTwoArmAbove*) are considered in connecting the arms by following the following principles. First, the vertical arm of *pTwoArmPrevious* object and the horizontal arm of *pTwoArmAbove* object are ignored because the iteration goes from left to right and from top to bottom. Secondly, the side-skins information of each arm is inherited from its neighboring solid arm and passed on to the next arm across the virtual arm. Thirdly, after connecting the arms, all horizontal arm information in each two-arm object is transferred to the next solid vertical arm, or terminated when the polygon is closed.

Figure 8 shows two typical examples out of 11 scenarios (further illustrated in Figure 9 and Figure 10). In the left graph of Figure 8, the vertical and horizontal arms of *pTwoArmAbove* object are both virtual. The *pTwoArmPrevious* object passes its left side-skin information on to *pTwoArmNext*, and *pTwoArmNext* also inherits the right side-skin information from its neighboring solid arm of *pTwoArmPrevious* object. The corresponding transferring rule is:

pTwoArmNext-> m_pAbove Polygon = pTwoArmPrevious-> m_pAbovePolygon

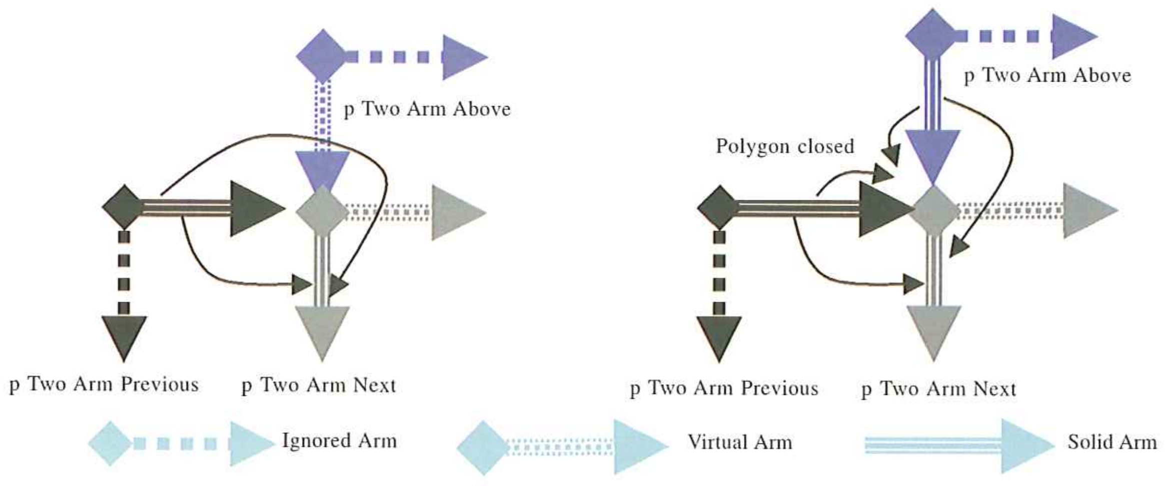


Figure 8. Two typical examples of passing topological relations between arms

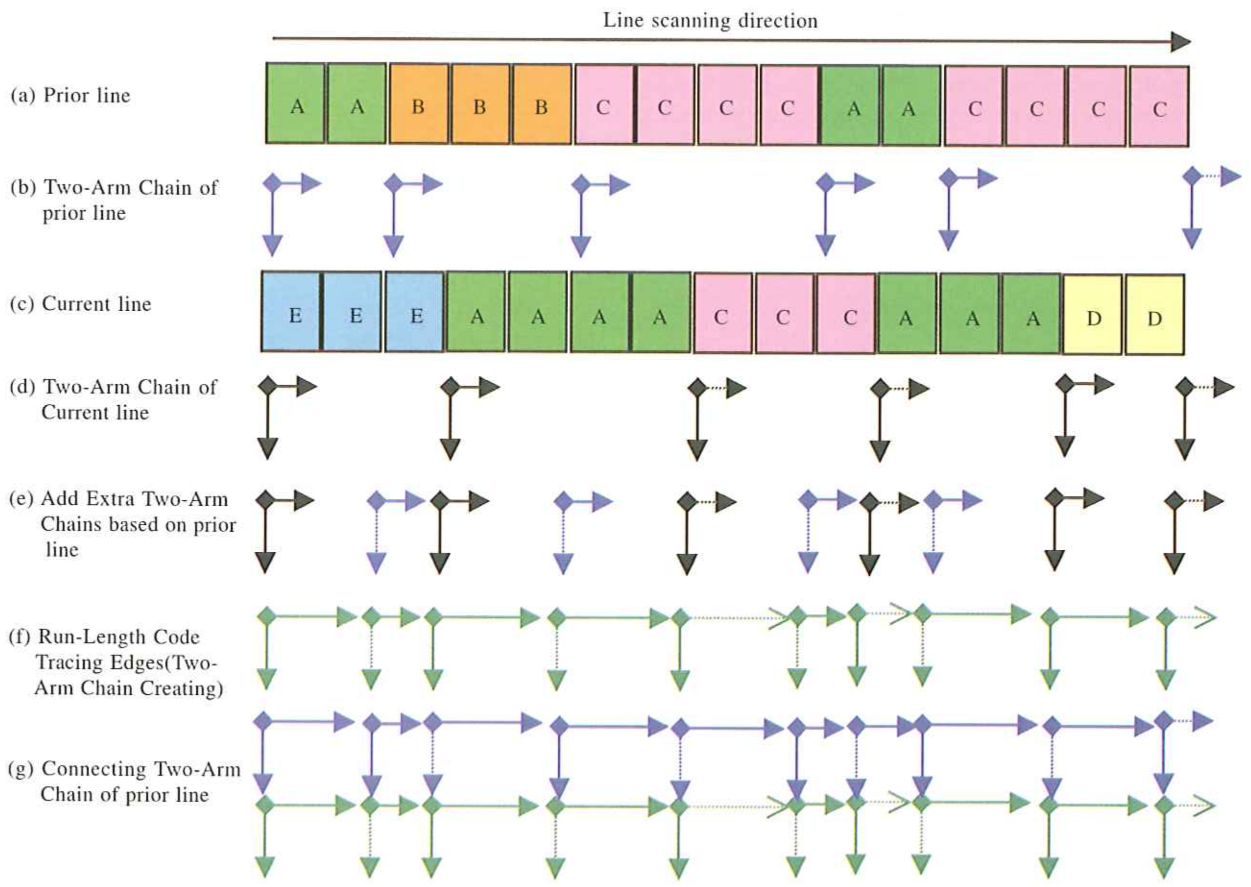

Line scanning direction
(a) Prior line
A A B B B C C C C A A C C C C
(b) Two-Arm Chain of prior line
(c) Current line
E E E A A A A C C C A A A D D
(d) Two-Arm Chain of Current line
(e) Add Extra Two-Arm Chains based on prior line
(f) Run-Length Code Tracing Edges(Two-Arm Chain Creating)
(g) Connecting Two-Arm Chain of prior line

Figure 9. Edge tracing while creating two-arm chain

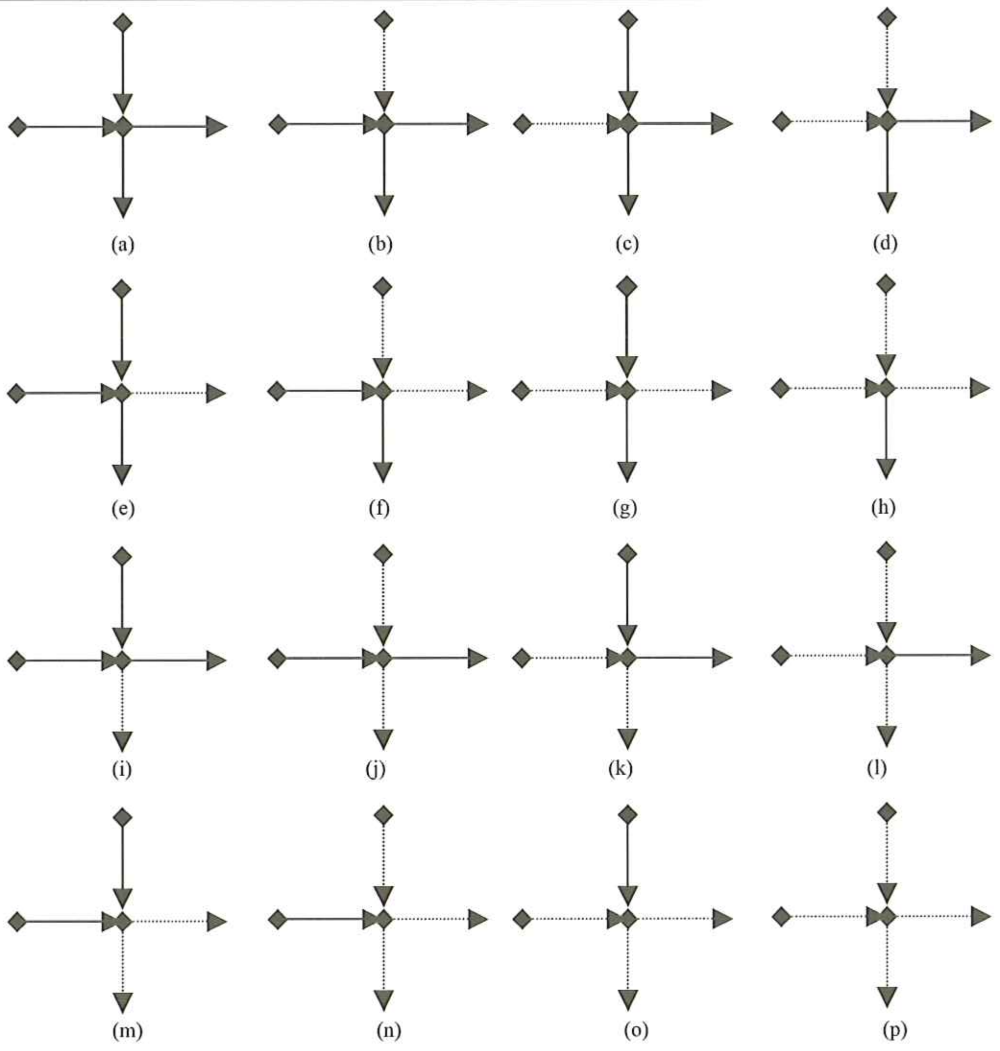


Figure 10. 16 possible scenarios of Two-Arm Chains connections

pTwoArmNext-> m_pInsidePolygon = pTwoArmPrevious-> m_pAbovePolygon
pTwoArmNext-> m_pLeftPolygon = pTwoArmPrevious-> m_pInsidePolygon;
In the right graph of Figure 8, the left side-skin information of *pTwoArmPrevious* object and the right side-skin information of *pTwoArmAbove* are terminated because of closing of the polygon at the bottom right, i.e., the end of polygon tracing. Its left side-skin information is inherited from *pTwoArmAbove*, and its right side-skin information from its neighboring *pTwoArmPrevious* object. Here all horizontal arms information is transferred into vertical arms.

E. Rules for connecting arcs to form a polygon

If nodes are met during the process of connecting two-arm objects, the corresponding arc tracing is completed. At this time, the *CCsArc* object should be saved and assigned to the resulting polygon. If the ending arc object is a horizontal arm, the left side skin is appended to the left arc chain of the *m_pAbovePolygon* object of current two-arm object, and the right side skin is appended to the right arc chain of the *m_pInsidePolygon* object of current two-arm object. If the ending arc object is a vertical arm, the right side skin is appended to the right arc chain of the *m_pLeftPolygon* object of current two-arm object, and the left side skin is appended to the left arc chain of the *m_pInsidePolygon* object of current two-arm object (also see Figure 7).

F. Implementing the two-arm chain edge tracing (TACET) algorithm

(i) Edge tracing based on the two-arm chain algorithm

Figure 9 shows the edge tracing procedures based on the Two-Arm Chain algorithm. Figure 9(a) illustrates the distribution of pixels in the top line of an image. Because of the map boundary, the corresponding arcs are all solid except for the last one (as shown in Figure 9(b)). Figure 9(c) shows the distribution pattern of pixels in the next line, and Figure 9 (d) is the derived Two-Arm Chains. In Figure 9(d), two Two-Arm objects have virtual horizontal arms because of the same pixel values C and A as in the prior line. As the Two-Arm objects in the prior line(in Figure 9(b)) have to be inherited, extra Two-Arm objects are added into current line (Figure 9 (d)) though these objects have the virtual vertical arms. As shown in Figure 9(e), the blue ones are inherited from the prior chain. As a pixel's position in each arm (represented by class *CHtArm*) is saved in the member variable *m_lColPos*, the horizontal edges can be formed by recording the running-length codes according to its contiguous *CHtArm* objects. For example, *m_lVolPos* is 0 in the first *CHtArm* object, and *m_lColPos* is 2 in the second *CHtArm* object. Therefore, the horizontal edge is from 0 to 2 along the horizontal axis. Figure 9(f) shows the complete chains' information, where solid arms are represented as solid lines and virtual arms are dashed lines. As discussed in the next subsection, solid and virtual arms play different roles in connecting arms. Figure 10(g) shows how two Two-Arm Chains (one shown in Figure 9(b) and another in Figure 9(f)) are connected. If no corresponding Two-Arm objects exist between Figures 9(b) and 9(f), extra Two-Arm objects with virtual vertical arms are created similarly.

(ii) Two-Arm Chains Connection Analysis

During the above process of connecting the Two-Arm Chains, polygons are also created. 11 valid cases can be identified from 2^4 (=16) possible scenarios. In Figure 10, solid lines represent cases when two neighboring pixels have different values, and dashed lines represent two neighboring pixels with the same values.

In Figure 10(a), all arms in four directions are solid (also refer to the 7th case (Figure 2(g)) in Figure 2). In this case, one polygon (top left of the node) is closed, and it begins tracing a new polygon (bottom right of the node). In Figure 10(b), the top connection is a virtual arm, and the polygon's information is thus transferred from left to right. Since the arms of right and bottom are both solid, it starts tracing a new polygon. In Figure 10(c), the polygon's information is transferred from top to bottom, it also starts tracing a new polygon. In Figure 10(d), it starts tracing two polygons. In Figure 10(e), one traced polygon is closed, and other polygons' information is transferred from top to bottom. Figure 10(f) shows the connection by a vertex where the left arm is extended into the bottom arm, and Figure 10(g) is similar as the top arm extends to the bottom arm. Figure 10(h) is an invalid case because there is only one solid arm. In Figure 10(i), one traced polygon is closed, and the polygon's information is transferred from left to right. Figure 10(j) and Figure 10(k) are both connections by a vertex, and Figure 10(l) is also an invalid case. In Figure 10(m), one traced polygon is closed. Figures 10(m)—(p) are three invalid cases.

(iii) Ridge and island detections

Since this algorithm traces polygons while scanning lines one by one, it cannot detect whether it is an island (as shown in the left of Figure 11) or a ridge (as shown in the right of Figure 11). A solution to this problem is provided below.

In the TACET algorithm, topological relationships are generated at the left-top corner of polygons when a polygon tracing process is completed at the right-bottom corner. Each polygon only consists of two arm-chains. For an island, we need to decide the parent polygon that contains the island. Unfortunately, this issue was not described by Riekert (1993) in detail. According to the definition of a *CHtArm* object, the member *m_lColPos* records the order of this *CHtArm* object along a horizontal scan. Hence, it is straightforward to find the polygon associated with the arm prior to the current one. The previous polygon is a parent polygon if the current traced polygon is an island.

As shown in Figure 11(a), at point A as the starting node of Two-Arm Chains, it creates two polygons #1 and #2. When it meets point B, it creates two new polygons #3 and #4, and detects that #3 is an island of #2. When it meets point C as an ending node, polygons #3 and #4 are closed. While it meets point D, polygons #1 and #2 are closed. In Figure 11(b), the same results are derived at points E and F. However, when it reaches point G as a bottom right node, the polygons are closed. What is the difference between points C and G? At point G, the arm pointing to polygon #2 connects with the arm pointing to a different polygon #3. This means that assuming polygon #3 as an outer polygon at the beginning point F is not valid. Therefore, polygon #3 does not exist, and the edges of polygon #3 are combined with those of polygon #2. On the contrary, at point C, the vertical arm pointing to polygon #4 connects with the horizontal arm also pointing to polygon #4 on the inner side. On the outer side, they point to the same

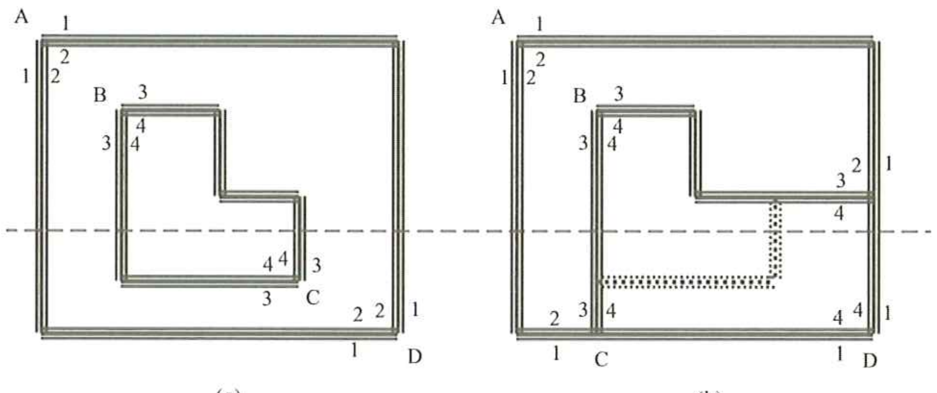


Figure 11. Ridge and island detection in polygon tracing

polygon #3 as well. This leads to closing two polygons #3 and #4, and generating an island.

In summary, the Two-Arm Chain algorithm leads to two different outcomes in these two cases and detects ridges and islands. According to the above discussions, the two cases depicted by Figure 11 are similarly dealt with in the proposed TACET algorithm. The only difference exists in the way of connecting two arms. Since all possible connections are enumerated in Simply Section III, it indicates the completeness of the TACET algorithm. It should be noted that we assume every polygon consists of two arms in this research. However, polygon #4 in Figure 11(a) can be simply represented by one arc. In other words, pseudo-nodes may be created in this algorithm. These pseudo-nodes are "harmless" in managing topological relationships of the resulting map. Hence, the two-arm assumption makes the proposed method simpler and more elegant than that of Riekert (1993).

IV. CASESTUDYANDEVALUATION

The program implementing the Two-Arm Chain Edge Tracing algorithm is nicknamed "*LittleHuskie R2V*". Huskie is the mascot of Northern Illinois University (NIU), where the primary author, supported by the Zhejiang Association for International Exchange of Personnel (ZAIEP), was visiting in 2006 and the secondary author was at the time affiliated with. Bulk of the programming work was done from July to December 2006. The performance of LittleHuskie R2V is evaluated and validated by comparing with ENVI 4.2, a widely used commercial RS image processing software package.

The original sample images came from TM of Landsat 5 (Figure 12(a)). Three different sizes were used: 500×500, 1000×1000 and 2000×2000. Based on the images, land use classifications (as shown in Figure 12(b)) were derived. The classification files were used as the input raster file for LittleHuskie and ENVI to perform the task of R2V conversion.

Table 1 lists the testing results of LittleHuskie and ENVI 4.2. It was carried out in a PC with a 2.4GHz CPU and a 512MB memory. The results demonstrate that LittleHusike was 8 times faster than ENVI in large image processing.

(a) Original TM image

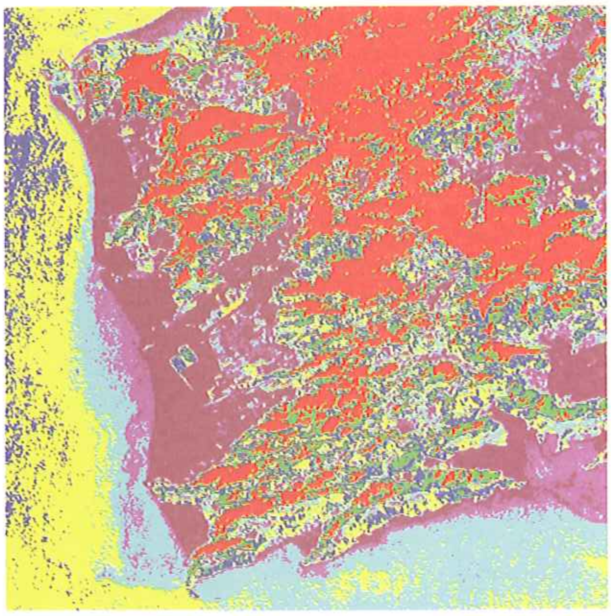

(b) Classification result of TM image

Figure 12

Table 1. Comparison of LittleHuskie vs. ENVI 4.2

Image size	Polygons	Time by ENVI4.2(s)	Time by LittleHuskie (seconds)
500×500	22,619	38	5
1000×1000	90,181	417	31
2000×2000	230,824	4,840	282

V. CONCLUSION

Data conversion from raster to vector (R2V) is a key function included in commercial RS software packages. However, R2V is currently a time-consuming process in software such as ENVI, and calls for the need to improve the computation efficiency. This paper presents an efficient R2V algorithm that processes large images and automatically builds GIS topology while scanning image lines one by one. The new algorithm, termed Two-Arm Chains Edge Tracing (TACET), has several significant advantages. First, it converts all types of area objects of RS classification in only one processing cycle. Secondly, it constructs complete area topological relationship by recording the shared edge between two polygons only once. Finally, it is scalable when processing large images. The program based on the algorithm, called "LittleHuskie R2V", is faster in processing large RS images with comparison to ENVI.

ACKNOWLEDGEMENTS

Financial supports from the Natural Science Foundation of Zhejiang Province (No. Y506185) and two National Technical Support Projects of China (No. 2006BAC03B02 and No. 2006BAC03B01) are gratefully acknowledged.

REFERENCES

[1] Congalton R. G., 1997, Exploring and evaluating the consequences of vector-to-raster and raster-to-vector conversion. *Photogrammetric Engineering & Remote Sensing* 63: 425—434.

[2] Lou X., Huang W., Shi A., Teng J., 2005, Raster to vector conversion of classified remote sensing image. *Geoscience and Remote Sensing Symposium IGARSS-05 Proceedings. IEEE International* 5: 3656—3658.

[3] Mattikalli N. M., Devereux B. J., Richards K. S., 1995, Integration of remotely sensed satellite images with a geographical information system. *Computers and Geosciences* 21: 947—956.

[4] Riekert W. F., 1993, Extracting area objects from raster image data. *IEEE Computer Graphics & Applications*, March 1993: 68—73.