

# A New Polygon Based Algorithm for Filling Regions

Hoo-Cheng Liu\*, Mu-Hwa Chen\*, Shou-Yiing Hsu\*\*,  
Chaoyin Chien\*, Tsu-Feng Kuo\* and Yih-Farn Wang\*

\* *Department of Computer Science and Information Engineering,  
Tamkang University, Tamsui, Taiwan, R. O. C.*

\*\* *Department of Information Management,  
National Defense Management College,  
Taipei, Taiwan, R. O. C.*

## Abstract

Region filling is a fundamental operation in computer graphics and image processing. There are broadly two classes of region filling: polygon based and pixel based. The conventional polygon based region filling algorithm typically uses data structures of records and fields. Using these data structures, the region filling process slows down because of the time-consuming operations of records and fields. This paper proposes a new polygon based region filling algorithm by using the proposed data structures of triples. This results in use of more efficient triple operations involving arrays and elements to fill a region. Using data structures of triples, the y-coordinate modification problem that occurs in the conventional algorithm simply disappears. In addition, contrary to the conventional approach, which uses troublesome geometrical considerations in deciding the even number of elements in each linked list, the proposed triple model uses a simple criterion to meet the even number requirement. Most important of all is the fact that the proposed criterion is independent of the polygon geometry. The experimental results strongly support superiority of the proposed algorithm. It is verified that the proposed algorithm is both theoretically and experimentally better than the conventional algorithm.

**Key Words:** Region filling, Y-modification test, Active edge table, Edge table, Liu's criterion

## 1. Introduction

To fill the interior of a region is important in many applications such as computer cartography, pattern recognition, scene analysis, and computer graphics [6]. Filling the interior of a region with a given color can be classified into two broad classes: polygon based and pixel based [6]. The polygon based technique is also known variously as an "ordered edge list", a "scan conversion", or a "rasterization" technique and is applicable whenever the contour is given as a polygon. The sides or edges of a polygon are sorted according to their coordinates, and then the sorted list is scanned [5].

In general, two data structures, the Active Edge Table (AET) and the Edge Table (ET) [1, 2,

3] are used for the scan conversion algorithm of filling polygons (regions). To fill a polygon correctly, the number of edges in the AET must always be even. To meet this condition, the y coordinates of many polygon vertices under certain conditions must be properly modified. To test this y-modification of each vertex, the relations of each pair of adjacent edges and sometimes the relations of two consecutive pairs of adjacent edges should be carefully examined. Is it possible to design a new scan conversion approach for filling polygons without modifying the y coordinates? Our approach in this paper will answer this question in the affirmative. At the same time, the terms of the proposed data structures xL and xR (refer to K2 in Section 3.1) have the same merit as the terms, the left

terminal pixel and the right terminal pixel used in Nakashima et al. [4]. Moreover, the proposed approach here is also an improvement on Watt's polygon rasterization method [7] which only uses one pixel, the x coordinate, to construct its data structures. However, the proposed algorithm uses triples to construct necessary data structures. The extra element is useful in processing boundary conditions and in saving processing time. Moreover, Liu's criterion for constructing data structures is a general-purpose mechanism in helping implement the proposed algorithm.

The rest of this paper is organized as follows. Section II describes the conventional algorithm and its data structures. Section III presents the proposed data structures and the region filling process. Section IV explicates the experimental results and discussions. Finally, a set of short conclusions which summarize our contributions are provided in Section V.

### 2.A Conventional Polygon-Filling Algorithm and its Data Structures

Some assumptions are introduced in this section. A brief review of the conventional algorithm

of region filling and its data structures is also presented. For clarity and convenience, region and polygon are sometimes used interchangeably. The polygon (region) to be filled is called the input polygon.

### 2.1 Preliminary Assumptions

Vertices of the input polygon to be filled are presented as an array of integers. Each pair of integers represents a vertex, the first and second integers of an integer pair represent the x and y coordinates of a vertex, respectively. The first pair is the first vertex, the second pair is the second vertex, and so on, as shown in Fig. 1.

Also, our new method can properly handle multiple polygons with possible holes and self-intersecting edges.

A region-filling algorithm consists of two phases: constructing the data structures and processing it (filling). Two tables related to the algorithm, the Active Edge Table (AET) and the Edge Table (ET), are presented for later references and comparisons, as shown in Fig. 2 and Fig. 3, respectively [3].

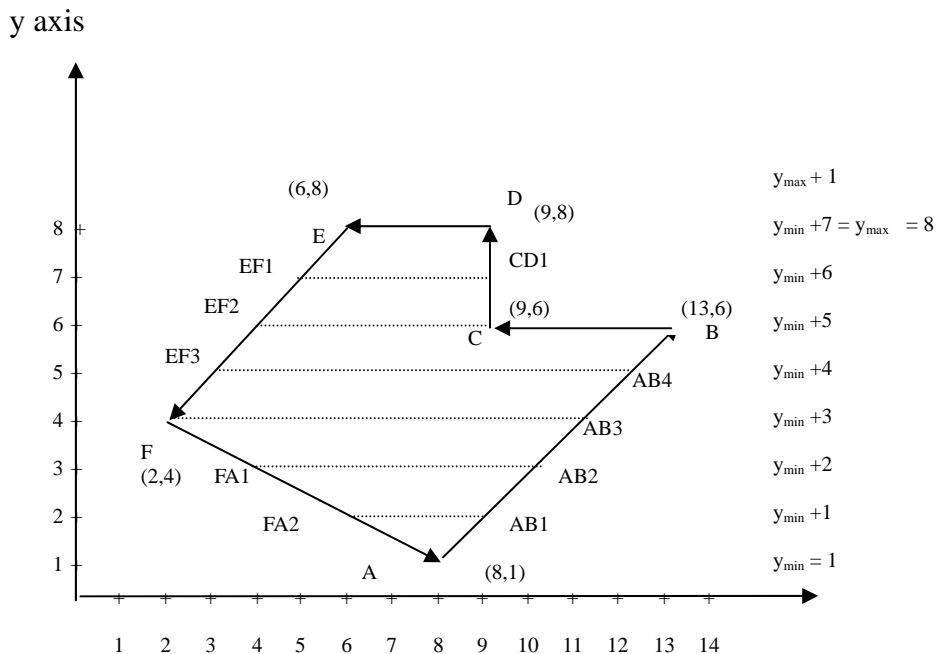


Figure 1. An input polygon with the geometrical relations of its boundary pixels.

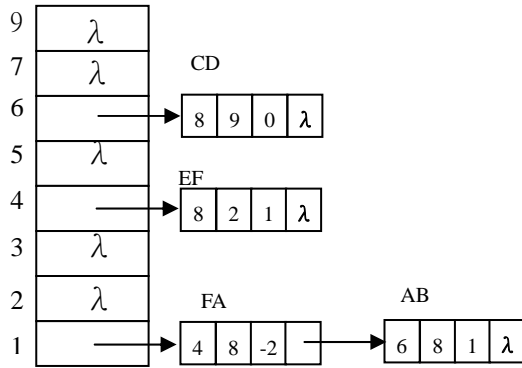


Figure 2. The Edge Table (ET) of Figure.1

## 2.2 Constructing the data structures

Geometrically, the AET represents a horizontal scan line and is a list of edges. The ET is initially formed as a list of edges and is bucket-sorted into a list of lists. For convenience, we shall refer to these lists as buckets (following the bucket sort). Buckets are indexed by their corresponding y values. The minimum indexed y value  $y_{min}$  is the minimum y coordinate of the polygon vertices, and the maximum indexed y value  $y_{max}$  is the maximum y coordinate of the polygon vertices. Each edge in any list is a node having four fields, as shown in Fig. 2. The above data structures are explained as follows:

- B1. A  $y_{top}$  field filled with the highest y coordinate value of the edge (a non-horizontal line segment).
- B2. A  $x_{val}$  field initially filled with the lowest x coordinate value of the edge, and this  $x_{val}$  value may be changed during later processing.
- B3. An increment field  $1/m$  used in stepping from one scan line to the next, where  $m$  is the slope of the edge. Note that no horizontal line edge is included, and  $1/m$  is always finite.
- B4. A link field links the next edge in the bucket. In general, singly linked list should be enough. However, for programming convenience, doubly linked list is a better choice because insertions and deletions may

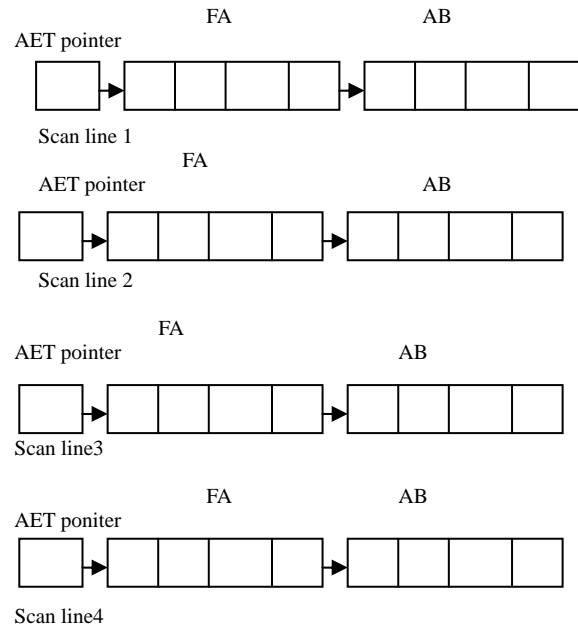


Figure 3. Different states of the Active Edge Table for different y-coordinates of Figure. 1.

occur frequently. Note also that if doubly linked structure is chosen, then two link fields should be used.

When an edge with y coordinate value  $y_{bot}$  of lower vertex is inserted into a bucket indexed by  $y_{bot}$ , the edges which include the edge newly inserted into in the bucket, are sorted by using bubble sort from bucket head to bucket tail. It follows that edges of smaller  $x_{val}$  values precede those of larger  $x_{val}$  values and for edges of equal  $x_{val}$  values, the edge with smaller  $1/m$  value is preceding.

## 2.3 The Processing Phase: Filling the region based on the above constructed data structures

Once the ET has been constructed, steps for filling the region are as follows:

- F1. Initialize the AET to be empty.
- F2. Set the index value y to the smallest y coordinate  $y_{min}$  of all polygon vertices, this y corresponds to the first nonempty bucket.
- F3. Repeat the following steps until both the AET and ET are empty.
  - F3.1 Move all edges of the ET bucket indexed by y to the AET, then sort the AET on values of the  $x_{val}$  fields, and if there is a tie, use the  $1/m$  fields for sorting.
  - F3.2 Fill in desired pixel colors of all line

segments on scan line  $y$ , if any, by using pairs of the  $x_{val}$ s from the AET's edges (nodes).

F3.3 Remove from the AET (delete nodes from the list) those edges for which  $y = y_{top}$  (edges not involved in the next scan line).

F3.4 Increment  $y$  by 1 (to the coordinate of the next scan line).

F3.5 For each nonvertical edge remaining in the AET, update the value of the  $x_{val}$  field for the new  $y$ .

From Sections 2.2~2.3 presented above, it is not clear why the  $y$  coordinates of some polygon's vertices need be modified. We shall investigate this  $y$ -modification mechanism by filling an example polygon using both Section 2.2 and Section 2.3 in the following.

#### 2.4 The Necessity of Modifying Some Y Coordinates for Correctly Filling an Example Polygon in the Conventional Algorithm

As an aid to understanding, Fig. 1 describes schematically the geometrical relations of an example input polygon's boundary pixels. Fig. 2 and Fig. 3 describe the corresponding ET and AET, respectively. Capital letters A, B, C, D, E and F represent both the labels of the vertices, the  $x$  coordinates of the corresponding vertices, and terms AB1, AB2, AB3, AB4, CD1, EF1, EF2, EF3, FA1, and FA2 represent the  $x$  coordinates on the various corresponding boundary edges. The corresponding  $y$  coordinates are labeled from  $y_{min}-1$ , through  $y_{min}$ ,  $y_{min}+1$ , ...,  $y_{min}+6$ ,  $y_{min}+7 = y_{max}$ , to  $y_{max}+1$ , where  $y_{min}=1$  and  $y_{max}=8$ .

Using Figs. 1~2, carefully tracing the first three loops step by step of the above filling phase, it is seen that

*Loop 1: Draw the pixel A ( $y = y_{min}$ ),*

*Loop 2: Draw the (horizontal) line segment from FA2 to AB1 ( $y = y_{min} + 1$ ), and*

*Loop 3: Draw the line segment from FA1 to AB2*

*( $y = y_{min} + 2$ ).*

Note that at the moment loop 3 is just finished, the AET still has two edges, edge AB and edge FA, and the  $y$  index of the AET has been changed to  $y_{min} + 3$ , and the new  $x$  values of the two edges have been obtained. With respect

to loop 4, steps are as follows.

The first step of loop 4, F3.1, is to move new bucket indexed by  $y = y_{min} + 3$ . There is one edge EF in it, we move it from the  $y_{min} + 3$  bucket to the AET, and the AET now has "three" edges. Note that "three" is not an even number.

Now when step 2 of loop 4, step F3.2, is executed, there will be an error. To avoid the above type of errors, a heuristic trick is to shorten the value of the  $y_{top}$  field of the edge FA in the bucket  $y_{min}$  by 1 (this is what we call the  $y$ -modification). The trick is very easy to understand and it succeeds.

The only problem with this trick is that it is tedious and troublesome to implement. For example, the vertices have to be classified as singular points or normal points. Moreover, for self-intersecting edges, extra sorting of the AET's edges has to be performed to determine the inside or outside problem. Suppose that the average number of edges in the AET is  $k$ , it follows that the best sorting takes  $k \cdot \ln(k)$  times. Note also that the  $y$ -coordinate modification itself wastes no time. However, to determine which  $y$  coordinate should be modified takes time.

Thus, we aim to design new data structures and propose a new scan conversion polygon filling algorithm without the  $y$ -modification, and for the case of self-intersecting edges, no extra sorting is required.

### 3. The Proposed Data Structures and the Filling Process

In this section, we will first state the proposed new data structures of arrays and elements, then explicate the proposed filling process based on the proposed data structures.

#### 3.1 The Proposed Data Structures

K1. An input polygon with vertices  $a = (x_0, y_0)$ ,  $b = (x_1, y_1)$ ,  $c = (x_2, y_2)$ ,  $d = (x_3, y_3)$ , ...,  $g = (x_6, y_6)$ ,  $h = (x_7, y_7)$  and  $(x_8, y_8) = (x_0, y_0)$  is shown in Fig. 4.

K2. A Horizontal Short Line Segment (HSLS) is a triple  $(y, xL, xR)$  which consists of an index  $y$  and two extreme  $x$  values  $xL$  and  $xR$ . Where  $xL$  is the leftmost  $x$  coordinate and  $xR$  the rightmost  $x$  coordinate of the HSLS, respectively. In particular, when  $xL = xR$ , then the HSLS is the single point  $(xL, y) =$

$(xR, y)$ . A single point is a shortest HSLs.

For example, the single point  $c = (x_2, y_2)$  is the shortest HSLs  $(y_2, x_2, x_2)$ . The line segment from  $e = (x_4, y_4)$  to  $d = (x_3, y_3)$  is also a HSLs. For this HSLs,  $y = 6$ ,  $xL = x_4 = 9$ , and  $xR = x_3 = 13$ , in this case,  $xL < xR$ , so this HSLs is not the shortest HSLs.

K3. The discrete pixel polygon is the input polygon whose boundary edges are now represented as discrete pixels which are calculated from the original edges by Bresenham's method.

K4. The HSLs polygon, the Hpolygon for short, is the input polygon whose boundary is composed of HSLs, which are calculated and collected from the discrete pixels of the discrete pixel polygon.

The input polygon is also a Hpolygon. Two HSLs elements of this Hpolygon are the two HSLs as mentioned and explained in K2. It follows that the other elements of the Hpolygon should be obvious in Fig. 4. Hence, there is no need to mention them one by one. It is found that there are actually eighteen HSLs elements in Fig. 4, as shown in Fig. 5.

K5. The new ET, the NET for short, similar to the conventional ET of Sections 1~ 2, is a list of new type of buckets called Nbuckets. Note that each node of the Nbucket consists of three fields: the  $xL$ , the  $xR$ , and the link field whose meanings can be referenced from K2 and B4 of Section 2, respectively.

Note also that nodes in every Nbucket are sorted according to the values of the  $xL$  fields.

### 3.2 The Proposed Filling Process

Once the NET is formed, the proposed filling process is as follows:

NF1. Using Bresenham's method [3], calculate all the discrete pixels of all the input polygon's edges one by one, and save the discrete pixels into the discrete pixel polygon. The discrete pixel polygon can be implemented as an array of integers.

NF2. Starting from the first pixel of the pixel polygon till the last, test all pixels sequentially, find groups of consecutive

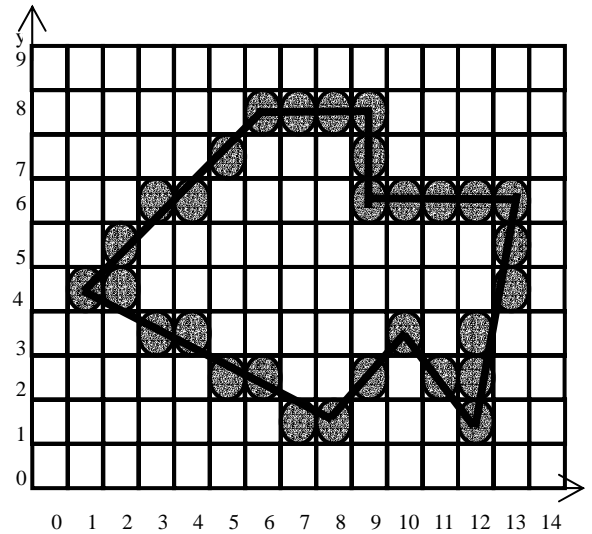


Figure 4. The vertices of the input polygon IP are  $a = (x_0, y_0)$ ,  $b = (x_1, y_1)$ ,  $c = (x_2, y_2)$ ,  $d = (x_3, y_3)$ ,  $e = (x_4, y_4)$ ,  $f = (x_5, y_5)$ ,  $g = (x_6, y_6)$ ,  $h = (x_7, y_7)$ , and  $i = (x_8, y_8)$ , whose coordinates values are  $(8,1)$ ,  $(10,3)$ ,  $(12,1)$ ,  $(13,6)$ ,  $(9,6)$ ,  $(9,8)$ ,  $(6,8)$ , and  $(1,4)$ , respectively, where  $(x_0, y_0) = (x_8, y_8)$ .

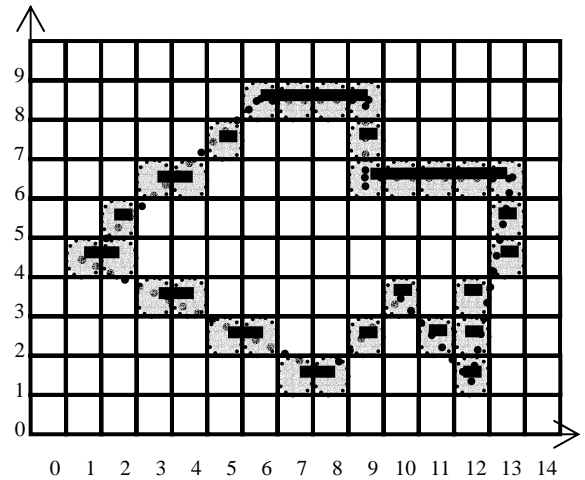


Figure 5. There are eighteen HSLs elements in Figure 4.

pixels having the same  $y$  coordinate. For each group of pixels, find the left-most  $x$  valued pixel,  $xL$ , and the rightmost  $x$  valued pixel,  $xR$ . Having found  $xL$  and  $xR$ , adding the common  $y$ , the three form a triple which is what we call a HSLs. Append this newly found HSLs to the Hpolygon, which is initially empty, and which can be considered as an array of HSLs. We refer to it as Harray.

NF3. Assume that there are  $h$  HSLs in the Hpolygon and let  $Hel$  denote a HSLs element of the Hpolygon. To use any  $Hel$

of the Hpolygon, by the conventional usage of array subscripts, any valid subscript value must be an integer between 0 and h-1. We will use  $H_i$  as the  $i$ 'th Hel, and say that for all integers  $i$ ,  $0 < i < h-1$ , if  $H_i$  is the current Hel, then  $H_{i+1}$  is the next Hel, and  $H_{i-1}$  is the previous Hel.

If  $H_0$  is the current Hel, then  $H_1$  is the next Hel, and  $H_{h-1}$  is the previous Hel. If  $H_{h-1}$  is the current Hel, then  $H_0$  is the next Hel, and  $H_{h-2}$  is the previous Hel. Carefully studying these current, next, and previous relations, it is seen that a modulus mathematics dividing by h for handling subscripts will easily solve the above relational problem.

*NF4. Using modulus mathematics (dividing by h) for processing subscripts, we now propose a criterion for constructing the NET, which is a list of Nbuckets as mentioned in K5. The NET is initially empty, and all its Nbuckets are indexed from  $y_{min}$  to  $y_{max}$  ( $y_{min} = 1$ ,  $y_{max} = 8$ , as shown in Fig. 1) as the ET in Section 2.*

Starting from the first Hel of the Harray, for each current Hel, we can use its y index to find its previous Hel and next Hel. For simplicity, we will use  $C_y$ ,  $P_y$ , and  $N_y$  to denote the y indices of the current Hel, the previous Hel, and the next Hel, respectively, we also denote  $x_L$  and  $x_R$  as the  $x_L$  and  $x_R$  values of the current Hel.

The proposed criterion for constructing NET is easy to implement and it is also independent of input polygons. Note also that the proposed criterion is powerful enough to cope with concave, convex polygon with self-intersecting edges, or holes. We refer to it as the Liu's Criterion for even number decision which consists of the following two conditions:

**Condition one:**

*If (  $P_y > C_y \ \&\& \ N_y < C_y \ // \ P_y < C_y \ \&\& \ N_y > C_y$  ) then sort and insert a node (a triple, details in NF2) with field values  $x_L$  and  $x_R$  of  $C_y$  into a proper position(indexed by y) in the y Nbucket.*

**Condition two:**

*If (  $P_y > C_y \ \&\& \ N_y > C_y \ // \ P_y < C_y \ \&\& \ N_y < C_y$  ) then sort and insert a pair of identical nodes with field values  $x_L$  and  $x_R$  of  $C_y$  into two proper consecutive positions in the y Nbucket respectively.*

For each current Hel, in turn, find and compare the three corresponding y values  $C_y$ ,  $P_y$ , and  $N_y$ . For example, in Fig. 4, let  $C_y=6$ , then  $P_y=5$ ,  $N_y=7$ . These three Hels meet Condition one. Also, in Fig. 4, let  $C_y=8$ , then  $P_y=7$ ,  $N_y=7$ . These three Hels obviously meet Condition two.

Note that the number of edges in the AET of the conventional algorithm must always be even as mentioned in Section 1. Similarly, for a new data structures and algorithm, the number of elements of our analogous HSLs AET should also always be even, then the region (the input polygon) can be correctly filled.

Observing Fig. 4, the HSLs AET identified by  $y = 8$ , there is only one HSLs. To make it is even, this HSLs should be doubled, and this explains why Condition Two is to be processed in this way. By contrary, the HSLs AET identified by  $y = 4$ , has two HSLs, so that Condition One is applicable, i.e., each HSLs of the two is counted only once.

As an aid to understanding, Fig. 6 applies schematically the Liu's criterion to a polygon. Note that Fig. 6 is only designed for explanation purpose. Actually, implementation of this structure doesn't adopt conventional edge-list approach. Instead, a more efficient and faster array-based approach is used.

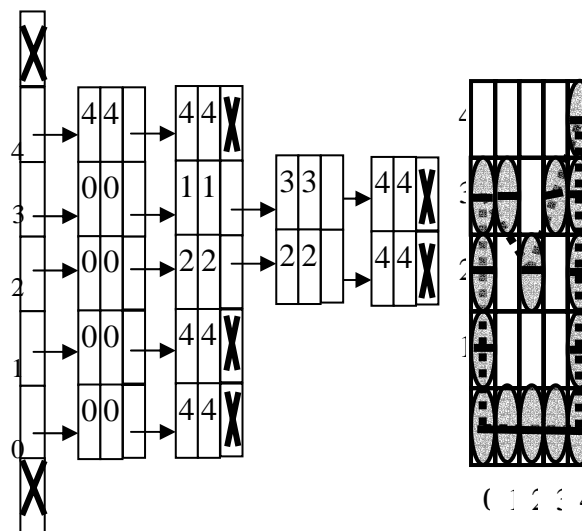


Figure 6. Apply schematically the Liu's criterion to a polygon.

Section 2.4 explains that to suitably modify many y coordinates is to make the number of edges in the AET of the conventional algorithm always even. It follows that the number of elements of the proposed analogous

HSLs AET should also always be even. However, the problem is easily resolved if the above-mentioned two Conditions are followed.

To complete the proposed algorithm, perform the following filling operations:

```

NF5. Once all the Nbuckets in the NET are
formed, fill the interior of the original
input polygon with a given pixel color by
the following loop.
  For  $y = y_{min}$  to  $y = y_{max}$  do
  {while there is still a pair of nodes in the y
  bucket,
    do {
    for the first node of the pair, using the value
    of  $xL$ ,
    for the second node of the pair, using
    the value of  $xR$ , and using the value of
     $y$ , draw a horizontal line segment
    from
     $(xL,y)$  to  $(xR,y)$  with the given pixel
    color.
    }
  }
  
```

From the data-structure point of view, each of the steps, from F3.1 to F3.5 in Section 2.3, involves the time-consuming operations of records and their fields. In comparison, each step of the proposed algorithm involves more efficient (faster) operations of arrays and their elements.

The time superiority of our new algorithm is really based on the proposed data structures in Section 3. In other words, the slow execution time of the conventional algorithm comes from the data structures in Section 2. It is seen that different data structures state different time efficiency. Hence, the superiority of new algorithm is largely based on the superiority of its data structures. The following experimental results will justify the claim.

#### 4. Experimental Results

This section compares the processing times of conventional algorithm and the proposed algorithm. Our experimental results include both the case of drawing no line (F3.2 and NF5 are skipped) and the case of drawing all lines (F3.2 and NF5 are performed). Fig. 7 describes the input convex polygon with 4, 6, 17, and 68 edges, respectively. Fig. 8 describes the input polygon with self-intersecting edges of 4, 6, 17, and 68, respectively. Fig. 9 presents the

cases of drawing all lines.

The case of drawing no line is reported in Fig. 10, Table 1, and Fig. 11 and Table 2. The line-drawing case is reported in Fig. 12, Table 3, and Fig. 13 and Table 4.

Both cases include two types of comparison. Type 1 presents those polygons without self-intersection, as shown in Fig. 7. Type 2 presents those polygons with self-intersection, as shown in Fig. 8. The experimental results for Type 1 comparison of both cases are shown in Fig. 10, Table 1, and Fig. 12 and Table 3. While that for Type 2 are shown in Fig. 11, Table 2, and Fig. 13, and Table 4.

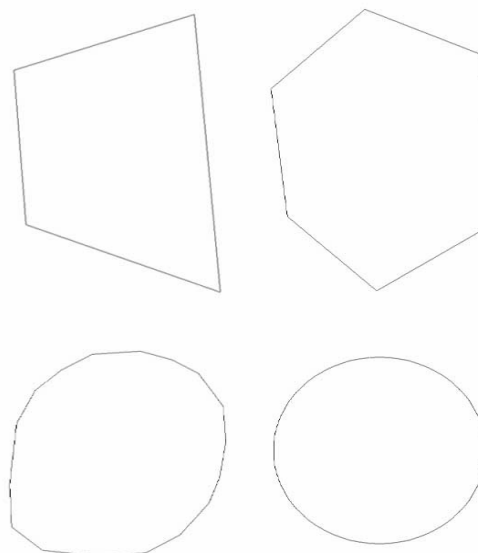


Figure 7. The input convex polygon with 4, 6, 17, and 68 edges, respectively.

All algorithms are implemented on a Pentium-II Intel CPU running Microsoft Visual Basic V.5 under the MS Windows 98 operating system.

Here, "execution times" or "times" mean the total times in which the program executes. This is because filling regions only takes a few milliseconds, however, the time unit of display function in Visual Basic supports only measurement in terms of second. To get the obvious execution time, the experiment uses same program to repeat the execution more than once such as 10, 20, 30 times, and so on, for purpose of comparison.

The term "second" in Figs. 10~13 means that different algorithms take time in seconds for processing various input polygons.

Under each experimental sample, the units of execution time are in seconds. Since Visual Basic supports no millisecond display function.

Consequently, the "0" value of execution time in Table 1 comes from rounding function when the execution time is less than 0.5 seconds.

All figures give comparison results in graphic form, and all tables give comparison results in tabular form. Sample (4), sample (6), sample (17) and sample (68) correspond to polygons' edges 4, 6, 17, and 68, respectively. The first column of Tables 1~4 gives a total times of program execution. For example, if the number of program execution times is ten, such as Tables 1~4, this means that the same program has run a total of 10 times.

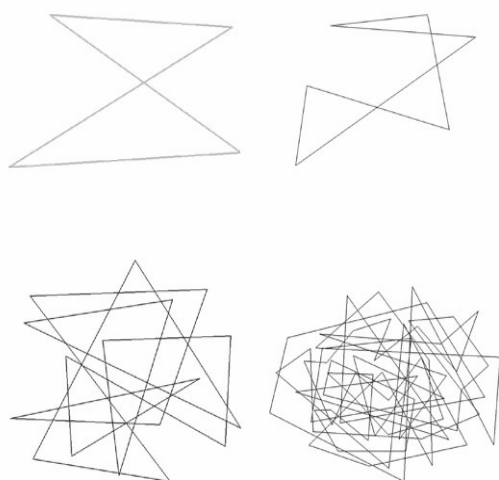


Figure 8. The input polygons with self-intersecting edges of 4, 6, 17, and 68, respectively.

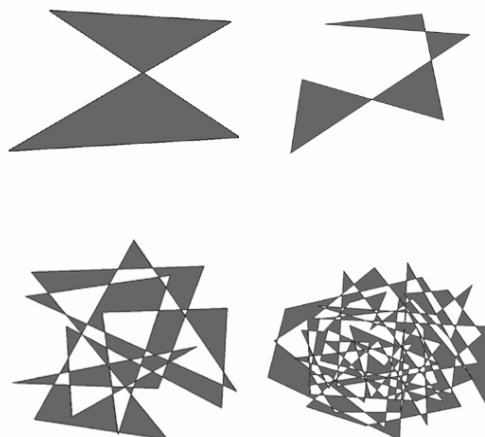


Figure 9. The cases of drawing lines.

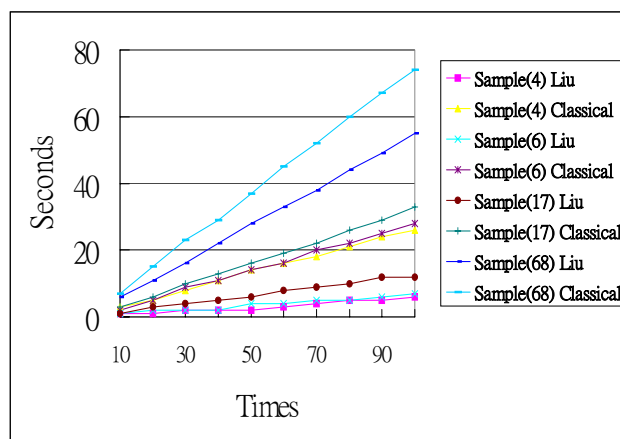


Figure 10. Comparison results for Liu's algorithm and the conventional algorithm of drawing no line in graphical form.

Table 1. Comparison of execution time for Liu's algorithm and the conventional algorithm using different edges as input polygon, and execution times in Seconds of drawing no Line in tabular form.

Execution Times	Sample(4)		Sample(6)		Sample(17)		Sample(68)	
	Liu [sec.]	Conventional [sec.]	Liu [sec.]	Conventional [sec.]	Liu [sec.]	Conventional [sec.]	Liu [sec.]	Conventional [sec.]
10	0	3	1	2	1	2	1	3
20	1	5	1	6	2	5	1	5
30	2	8	1	9	2	8	3	8
40	2	11	2	11	3	10	3	10
50	2	13	3	13	4	13	4	14
60	4	15	4	16	4	16	4	16
70	4	19	4	18	4	19	5	19
80	4	21	5	21	5	22	6	22
90	5	24	6	24	6	24	6	25
100	5	26	6	26	6	27	7	27

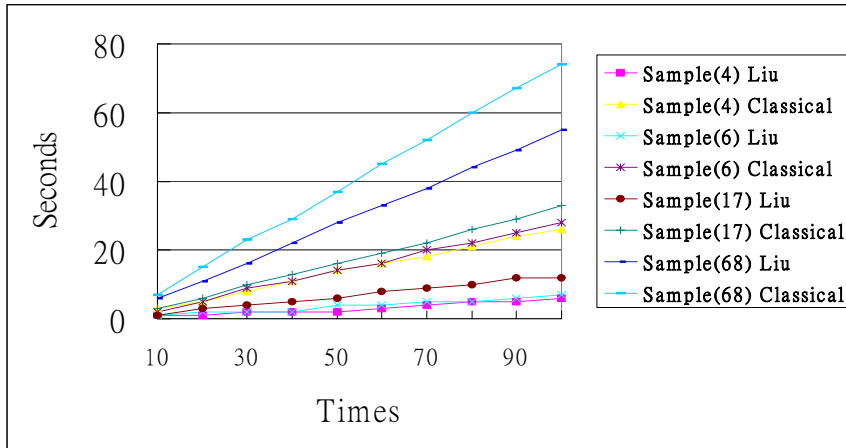


Figure 11. Comparison results of drawing no line using polygons with self-intersection edges in graphical form.

Table 2. Comparison results of drawing no line using polygons with self-intersection edges in tabular form.

Execution Times	Sample(4)		Sample(6)		Sample(17)		Sample(68)	
	Liu [sec.]	Conventional [sec.]	Liu [sec.]	Conventional [sec.]	Liu [sec.]	Conventional [sec.]	Liu [sec.]	Conventional [sec.]
10	1	3	1	2	1	3	6	7
20	1	5	2	5	3	6	11	15
30	2	8	2	9	4	10	16	23
40	2	11	2	11	5	13	22	29
50	2	14	4	14	6	16	28	37
60	3	16	4	16	8	19	33	45
70	4	18	5	20	9	22	38	52
80	5	21	5	22	10	26	44	60
90	5	24	6	25	12	29	49	67
100	6	26	7	28	12	33	55	74

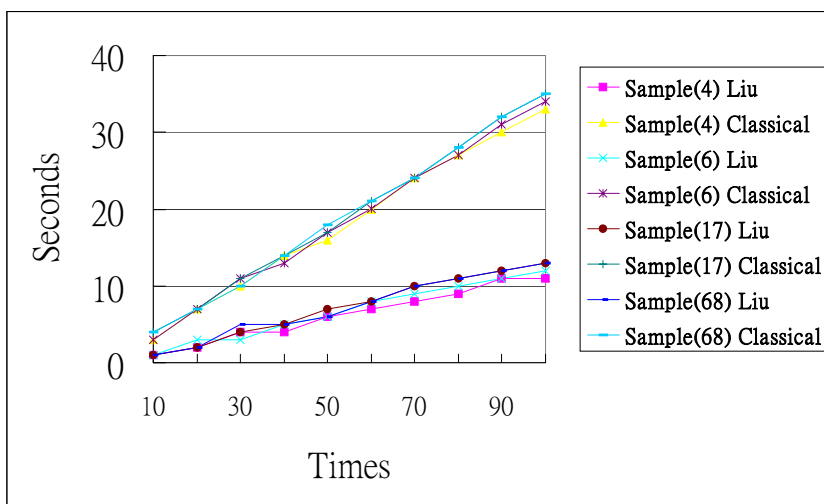


Figure 12. Comparison results of drawing lines using polygons with no self-intersection edge in graphical form.

Table 3. Comparison results of drawing lines using polygons with no self-intersection edge in tabular form.

Execution Times	Sample(4)		Sample(6)		Sample(17)		Sample(68)	
	Liu [sec.]	Conventional [sec.]	Liu [sec.]	Conventional [sec.]	Liu [sec.]	Conventional [sec.]	Liu [sec.]	Conventional [sec.]
10	1	3	1	3	1	4	1	4
20	2	7	3	7	2	7	2	7
30	4	10	3	11	4	11	5	10
40	4	14	5	13	5	14	5	14
50	6	16	6	17	7	17	6	18
60	7	20	8	20	8	21	8	21
70	8	24	9	24	10	24	10	24
80	9	27	10	27	11	28	11	28
90	11	30	11	31	12	32	12	32
100	11	33	12	34	13	35	13	35

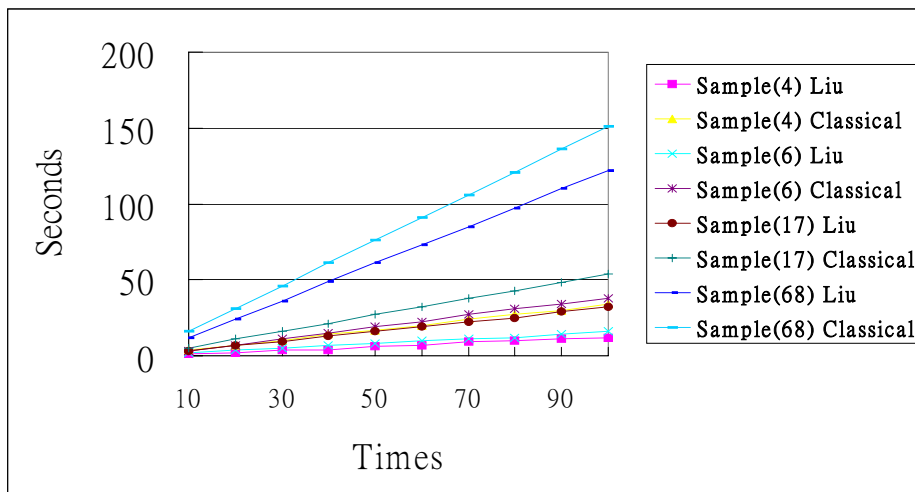


Figure 13. Comparison results of drawing lines using polygon with self-intersection edges in graphical form

Table 4. Comparison results of drawing lines using polygon with self-intersection edges in tabular form.

Execution Times	Sample(4)		Sample(6)		Sample(17)		Sample(68)	
	Liu [sec.]	Conventional [sec.]	Liu [sec.]	Conventional [sec.]	Liu [sec.]	Conventional [sec.]	Liu [sec.]	Conventional [sec.]
10	1	4	2	3	3	5	12	16
20	2	7	4	7	7	11	24	31
30	4	10	5	11	9	16	36	46
40	4	14	7	15	13	21	49	61
50	6	17	8	19	16	27	61	76
60	7	20	10	22	19	32	73	91
70	9	24	11	27	22	38	85	106
80	10	27	12	31	25	43	97	121
90	11	30	14	34	29	48	110	136
100	12	34	16	38	32	54	122	151

From the above experimental results (Figs. 10 ~ 13 and Tables 1~ 4), it is verified that the proposed scan conversion algorithm is better than the conventional scan conversion algorithm for filling polygons under the given comparison types.

## 5. Conclusion

The y-modification problem arises from considering the geometrical boundary problem of adjacent polygon edges. To meet the criterion of having an even number of nodes in each list (even number criterion, for short) during the filling process, the conventional record-based approach has to consider the geometrical relations of adjacent polygon edges. This implies that to meet the even number requirement, the conventional approach in solving the y-modification is totally dependent on polygon geometry. In comparison, the proposed array-based triple approach uses the simpler Liu's criterion to solve the even number problem. Note also that Liu's criterion is wholly independent of polygon geometry. This is also one of main contributions in this paper.

From the graphical representation of record-based data structures of the conventional region-filling algorithm shown in Fig. 1 ~ Fig. 3, it is obvious that operations of records and fields are very time-consuming. For example, to find the next element of any Edge Table of Fig. 1, the time-consuming linked field operations of finding the next element would have to be adopted. By contrast, to find the next element of our data structures of arrays and elements represented in Fig. 4, the faster increment operations of array subscripts can be employed.

From the above discussion on the time efficiency of finding next elements, the implication is that there is a time superiority for array-based data structures relative to record-based data structures. This is applicable to other operations of region filling algorithms. This explains why our array-based region filling algorithm is time superior to the conventional record-based region filling algorithms.

Contrary to the conventional scan-conversion algorithm that is based on data structures of fields and records, the proposed conversion algorithm is based on data structures of triples involving arrays and elements for filling polygons. The proposed algorithm has faster execution time and uses no y-modification

and no extra sorting, at all. It is seen that these advantages come from data structures of triples.

It follows that more efficient data structures can usually result in more efficient algorithms. Experimental results of the proposed region-filling algorithm provide strong justification for our claim.

This paper has made some contributions in this regard. First, using data structures of triples, the proposed array-based algorithm effectively solves the y-coordinate modification problem of the conventional record-based counterpart. Next, the proposed algorithm is easy to implement because it uses more efficient triple operations involving arrays and elements to fill a region. Finally, it is verified that the proposed region filling algorithm is time superior to conventional region filling algorithms. Experiments were developed on MS Windows 98 to illustrate the validity of the proposed algorithm. Moreover, in solving any particular problem, the decision to adopt an array-based or a record-based data structure is of great importance. For the region filling algorithm, the experimental results strongly show that an array-based approach is far more efficient than the conventional record-based approach. Finding the appropriate approach to improve execution time is also one of the contributions of this paper.

## Acknowledgement

Many individuals deserve mention where this manuscript is concerned. First of all, Dr. David Kleykamp, who revises the English writing in this manuscript many times; next, Dr. Timothy K. Shih, for helping research on this paper.

## Reference

- [1] Ackland, B. D. and Weste, N. H., "The edge flag algorithm-A fill method for raster scan displays," *IEEE Transactions on Computers*, Vol. C-30 No. 1, pp. 41-47 (1981).
- [2] Distanto A. and Veneziani, N., "A Two-Pass Algorithm for Raster Graphics," *Computer Graphics and Image Processing* 20, pp. 288-295 (1982).
- [3] Foley, J. D., Andries, V. D., Feiner, S. K. and Hughes, J. F., *Computer Graphics*

- Principles and Practice*, Second Ed., Addison-Wesley, Reading, MA. (1990).
- [4] Nakashima, K., Massashi, K., Katsumi, M., Yoshihiro, S., and Yasuaki, N., "A contour fill method for alpha-numeric character image generation," *Proceedings of the Second International Conference on Document Analysis and Recognition*, pp. 722-725 (1993).
- [5] Pavlidis, T., "Contour filling in raster graphics," *ACM computer Graphics*, Vol. 15, No. 3., pp. 29-36 (1981).
- [6] Pavlidis, T., "Filling algorithms for raster graphics," *Computer Graphics and Image Processing* 10, pp.126-141 (1979).
- [7] Watt, A., *3D computer graphics*, 2nd Ed., Addison-Wesley, Reading, MA. (1993).

***Manuscript Received: Aug. 26, 1999***

***Revision Received: Oct. 22, 1999***

***And Accepted: Dec. 06, 1999***