Memorial University of Newfoundland

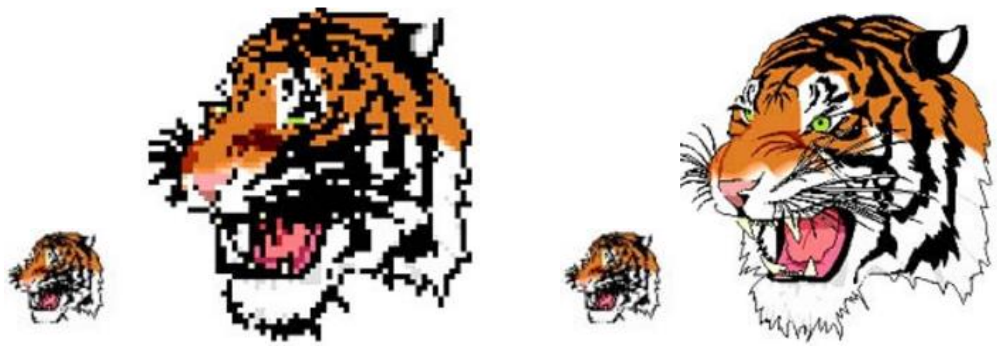
Winter 2016-2017

<u>Region Filling in Computer Graphics</u>

Zhi Yuan	201170644	zy8715@mun.ca
Thanjida Akhter	201691489	takhter@mun.ca

1. Introduction

In computer graphics, two broad categories of region filling algorithm can be classified depending on the domain of the graphics they operate in: raster filling and vector filling. In raster graphics, the images are plain matrices (bit maps) that store the color of the pixels. By this way, rich and detailed images can be created as different color can be assigned to each pixel. However, huge store space might be required because of the large number of pixels in one image and the pixels cannot retain their appearance as size increases. As shown in Figure 1(a), when the raster picture is enlarged, it becomes blurred. Unlike the raster graphics, vector graphics handles geometric information about lines, curves or other shapes called vectors. Therefore, the vector shapes, also called objects, can be scaled, and printed at various sizes without losing quality (shown in Figure1 (b)) because they are independent with each other. Another advantage of vector representation comes from the reduced amount of storage space required. However, vector graphics are time consuming and produce raster representations of the images for visualization or printing in the end [1].



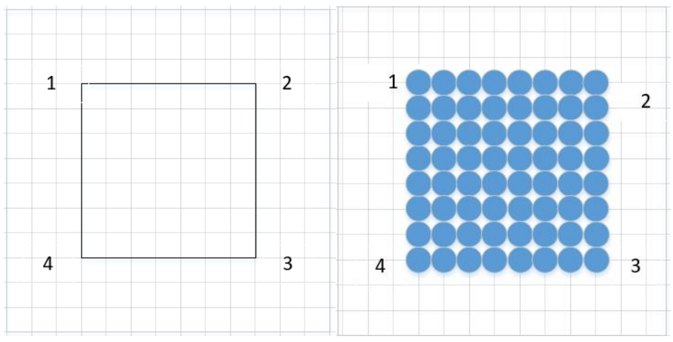
(a) Raster graphic **(b)** Vector graphic

Figure 1: Ways of representation of a polygon in graphics

Vector illustrations are widely used to produce high quality 2D drawings and figures. They are commonly based on objects that are assigned to layers, thereby allowing objects on higher layers to obscure others drawn on lower layers. Objects are typically constructed from collections of open and closed paths which are assigned to a single common layer when they are grouped together. Closed paths can be optionally filled by an opaque or semitransparent color to represent faces. A rich set of features and editing operations can then be integrated into this framework to yield powerful systems for 2D illustration.

2. Raster filling

Usually, in computer graphics a polygon can be expressed by vertexes and lattices (shown in Figure 2).



(a) Vertexes **(b)** Lattices

Figure 2: Ways of representation of a region in graphics

As Figure 2(a) shows, the square is stored as a vertex sequence and displayed the contour at a certain intensity using the vector graphics command or other available interpolating command. However, the filling algorithm cannot be applied directly because the locations of the pixels (either interior or exterior of a polygon) are unknown until scan conversion is used. For the lattice's method, a set of pixels covering the polygon is used. By this way, the fill algorithm can be applied directly to those pixels. However, the geometric information of the vertexes is missed.

2.1 Scan Conversion Algorithm

Due all the polygons must first be decomposed into pixels lying in a regular raster grid pattern [2], the scan conversion algorithm is proposed based on scanning edges of a polygon.

2.1.1 Scanline Algorithm

This algorithm can be simply illustrated as to fill the spaces between pairs of intersections of scanline with polygon edge and usually divided into the following steps:

a. Determining the y_{max} and y_{min} of the vertexes of a polygon.
b. Scanline intersects with each edge of the polygon from y_{min} to y_{max}.
c. Sorting the intersection point in the increasing order of x coordinate. (P_0, P_1), (P_1, P_2) and (P_2, P_3) shown in Figure 3.
d. Filling all the pairs of coordinates that are inside polygons. For example, (P_0, P_1) and (P_2, P_3).

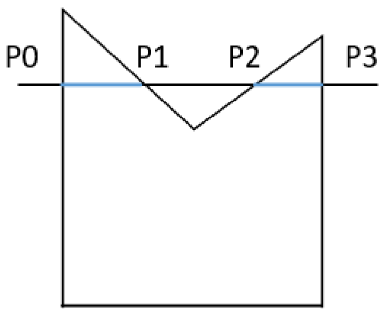


Figure 3: Scanline Algorithm

However, when the scan line intersects with the vertexes of a polygon, the number of intersections should be determined according to different situations (shown in Figure 4).

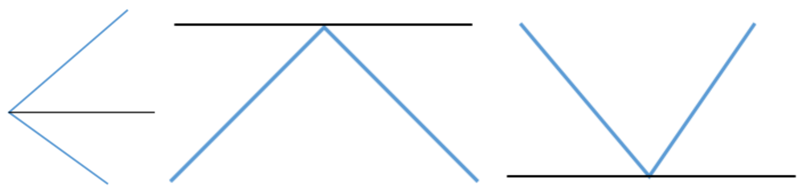


Figure 4: Various intersections of scan lines with vertexes of a polygon

Figure 4 shows three different intersections of scan lines with vertexes of a polygon. Figure 4(a) shows the edges of the intersection (in blue) locate on both sides of a scan line (in black). When counting the number of the intersection in this case, only one is considered. The (b) and (c) show the case when both edges of an intersection are on one side of a scan line. (b) shows when the scan line is on the top of the edges, in which the number of intersections will be treated as 0. However, in (c), two intersections will be counted.

This algorithm requires calculating all the intersections of scan lines with each edge of a polygon, which brings inefficiency as one scan line might cross several edges or even no intersection at all. This shortage could be overcome by active edge algorithm.

2.1.2 Active Edge Algorithm

To avoid inefficient intersection searching, only the intersections of a scan line with active edges, meaning the edges intersecting with the current scan line, are calculated taking advantage of the coherence of the scan line and polygons (shown in Figure 5) [3].

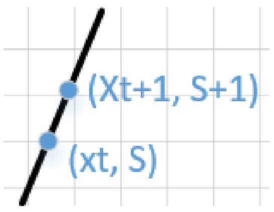


Figure 5 Edge coherence

As Figure 5 shows, the edge can be expressed as $y = kx + b$. Therefore, at the point $y = s$, $X_t = {}^{(s - b)}/_{k}$. Similarly, when $y = s + 1$, $X_{t+1} = {}^{(s + 1 - b)}/_{k} = X_t + 1/k$. By this method, the x coordinates of the intersections of edges with scan lines.

In this algorithm, two tables, called global edge table (GET) and active edge table (AET), are developed to store all the edges of a polygon and the edges in the current scan line, respectively (shown in Figure 6).

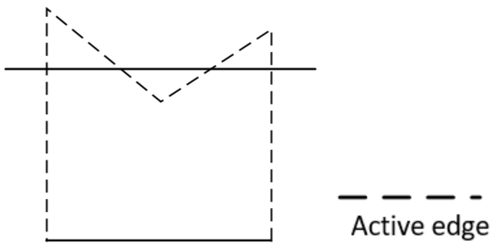


Figure 6: Active edges

In the global edge table, edges are bucket sorted according to their minimum y coordinates. When the scan line reaches the lower endpoint of an edge, it will be active and added into the active edge table. It should be notice that only the edges intersecting with current scan line are input in the AET. A typical structure of a node in an AET can be shown as

X	y_{max}	1/k	Next

Figure 7: A nodes in AET

Where x is the intersection of the edge with the current scan line, y_{max} is the maximum y of the current edge. When the current scan line moves above the upper endpoint (y_{max}), the edge becomes inactive and should be removed from the active edge table. Therefore, the algorithm can be performed as following steps:

a. Develop the global edge table and set the y_{min} as the input of GET.
b. Initialize AET to be empty.

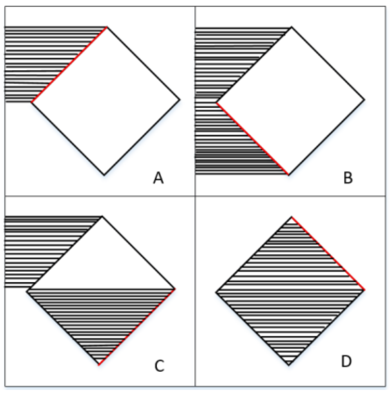
c. Move from GET bucket y to the AET the edges whose y_{min} = y.
d. Remove from AET the edges for which y = y_{max} (not involved in next scan line). Then, sort AET.
e. Fill desired pixel values on scan line y by using pairs of x coordinates from AET.
f. Increment y by 1.
g. For each no vertical edge remaining in AET, update x for new y.
h. Repeat the steps from c to g until AET and GET are empty.

By using this algorithm, all the pixels are visited only once which can greatly reduce the calculation. Therefore, it has become a widely used algorithm in polygon filling.

2.1.3 Edge Fill Algorithm

Edge fill algorithm is to invert the color of all pixels on the left (right) side of the intersections of scan lines with edges. Then the target polygon can be filled after the same process for all the edges of the polygon [4].

The intuitive idea can be illustrated in the following Figure 8. As the Figure shows, all pixels from the point, where the polygon boundary crosses a scan line, to the left edge of the screen are inverted. By this way, the points outside the polygon will be inverted an even number of times, which means these pixels will keep their original colors.



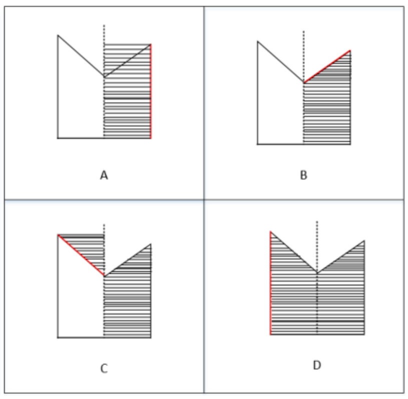


Figure 8: Edge Fill Algorithm **Figure 9: Fence fill algorithm**

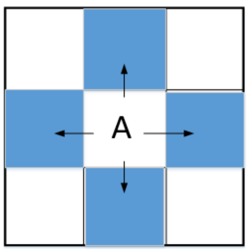
One of the advantages of this algorithm is it does fill in the same order as rendering's and therefore can easily be pipelined. Besides the former advantage, it is also a very simple algorithm. However, the pixels outside of the polygon might be visited various times for a complex polygon. Therefore, huge times of inputs/outputs might be required.

Based on above algorithm, another improved one, called Fence Fill, and is developed to reduce the repeat visits [5]. In this algorithm, an arbitrary boundary could be used, named a fence which is chosen from the X value of some point on the boundary of the polygon. By using the fence, only the pixels on scan lines between a boundary of the polygon and the fence, instead of all the pixels on the left of the boundary in former algorithm, will be inverted. It can be illustrated as follows (the dotted line represents the fence) (Fundamentals of Computer Graphics and Multimedia, D.P. Mukherjee. 2006).

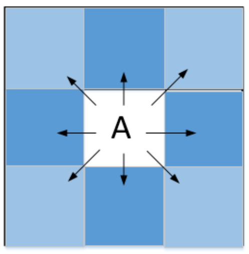
2.2 Region Fill

Region fill algorithm is another important algorithm in computer graphics. Usually, a region can be defined as a group of adjacent, connected pixels where pixels stand for picture element [6]. In a region filling, a target color is given to a pixel locating in the region and spread out to all contiguous pixels. The target region can be stored by reserving the pixels either on or in its boundary, based on which two algorithms, called boundary fill algorithm and flood fill algorithm respectively are developed.

The neighbor pixels of the target one can be defined as 4 neighbors and 8 neighbors, shown in Figure 9.



(a)

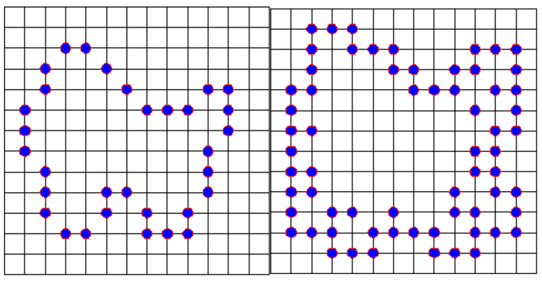


(b)

Figure 10: 4-neighbor and 8-nerghbor

As Figure 10(a) shows, starting from the pixel A, determining the four pixels next to A (in blue) whether they are filled with the desired color and changing the colors of the target pixels. Like 4-neighbor fill method, 8-neighbor fill method (Figure 10(b)) follows the same algorithm but searching 4 more near pixels.

Similarly, the region represented in the boundary and inner points by using 4 neighbors and 8 neighbors can be shown as,



(a)

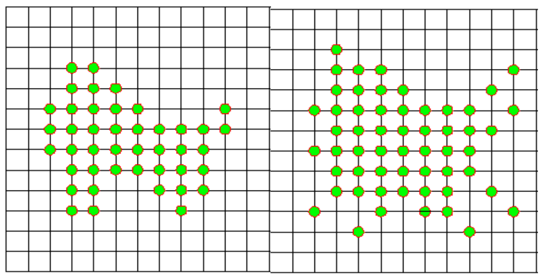

(b)

Figure 11: Region represented by (a) boundary and inner points
(b) The left sides are 4 neighbors and the right sides are 8 neighbors

In a fill algorithm, a starting pixel, also called the seed, provides a filling color, which is one of the input parameters. The other two inputs for a fill algorithm are the coordinates of the seed and the color of the boundary. Based on these inputs, the color of the current pixel is checked with the seed's color and change the default color with the fill color if they are not the same. Then recursively follow the procedure with its neighborhood points until the boundary pixels reached.

2.3 Square Visibility fill

In the method, a big ring around the image should be created firstly and chopped up into little pieces which stand for a small solid angle. As following figure shows, it centered as the seed, if the piece holds a 1, which means the range of angles is visible from the seed. The visible part on the ring is shown in green and the invisible shown in red.

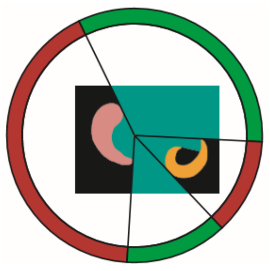


Figure 12: Square visibility fill algorithm [7]

The process can be shown as follows. At the beginning of each step, a square of the current side length should be drawn and start at one corner and walk clockwise around the square to look for shadow casters (pixels of any color other than the target color) and the appropriate cell on the ring. Then a line from the center of the seed through the center of the blocker and extend will be drawn until it hits the ring. Repeating this step, every pixel currently set to the target color can be considered as a candidate for being filled in. Then check the visibility cell, if the value is 1 which stands for the pixel is visible and the pixel's color should be changed to the new color.

Otherwise, skip it. It should be noticed that the number of cell in the ring should be big enough; all the blocked angles should be accurately marked.

3. Vector Filling Algorithm

Vector graphics systems use a combination of stacked layers of paths and planar, which are the two most common representations in both academic and commercial systems. However, stacked layers of paths are fundamentally limited in their ability to model even basic topological constructs, such as joining an edge (or path) to the middle of another edge, or sharing an edge between two faces.

3.1 Vector Graphics Complex (VGC)

Basic topological modeling, such as the ability to have several faces share a common edge, has been largely absent from vector graphics. We introduce the vector graphics complex (VGC) as a simple data structure to support fundamental topological modeling operations for vector graphics illustrations. The VGC can represent any arbitrary non-manifold topology as an immersion in the plane

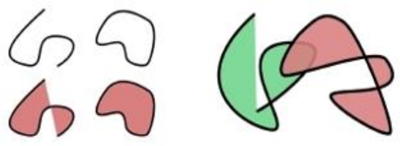


Figure 13: The "SVG" representation

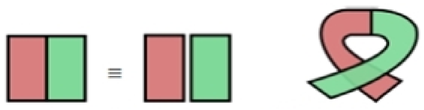


Figure 14: Limitations of existing representation

3.2 Scalable Vector Graphic (SVG)

The traditional vector graphics representation is Scalable Vector Graphics (SVG) file specification. With SVG, a drawing is represented using building blocks called paths. A path is typically a list of Bézier control points that can be either closed or open. It has drawing attributes that indicate how it must be rendered, such as stroke width, stroke color, and fill color, as illustrated Figure 13. Paths are defined independently of each other, which mean that if one path is dragged and dropped by the artist on top of another one, they freely overlap and do not interact as shown in Figure 13.

This basic overlapping capability is a very desirable feature since it allows the artist to freely edit the geometry of the paths and move the objects without any constraints. Nonetheless, there are cases where it would be desirable to model interaction between paths. A canonical example also described in occurs when the illustration represents two shapes that share a common partial contour or edge. In SVG, this must be represented as two independent closed paths, where the common section has the same geometry, as illustrated in Figure 14 (left).

4. Conclusion

For photographs and photo-realistic images, raster graphics are more applied compared to vector graphic. Besides above-mentioned algorithms widely used in raster graphics, other algorithms, such as line sweep fill, based on the classic methods are also developed. Although more complex regions can be dealt with and time can be saved, their disadvantages, such as being blur when enlarged, are not overcome. Therefore, in some situations, for example graphic design, vector graphics are commonly used.

VGC is a superset of multi-layer vector graphics, planar maps and stroke graphs, which significantly extends the range of objects that can be drawn with vector graphics, including 2D projections of 3D objects with imprecise or incomplete geometry, non-manifold surfaces of arbitrary genus, non-orient able surfaces, and overlapping faces.

5. Reference

[1] Dalstein B., Ronfard R. and Panner V.M. Vector graphic complexes. ACM Transaction of Graphics 2014; 33(4).
[2] Badler N.I. Scan conversion. Advances in Computer Graphics I. EurographicSeminars (Tutorials and Perspectives in Computer Graphics) 1986; Springer, Berlin, and Heidelberg.
[3] Agkland D.B., Weste H. N. The edge flag algorithm – A fill method for raster scan displays. IEEE Transactions on Computers 1981; 1(1): 41-48.
[4] Dunlavey R.M. Efficient polygon-filling algorithms for raster displays. ACM Transactions on Graphics 1983; 4(10): 264-273.
[5] Anitha S., Evangeline D. An efficient fence fill algorithm using inside-outside test. International Journal of Advanced Research in Computer Science and Software Engineering 2013; 3(11): 605-609.
[6] Lee T. E., Pan J. Y. and Chu P. An Algorithm for Region Filling Using Two-Dimensional Grammars. International Journal of Intelligent Systems 1987; 9(2): 255-263.
[7] Glassner A. Fill 'er up! [graphics filling algorithms]. IEEE Computer Graph. Application 2001; 1(21): 78-85.