Scan-Line Polygon Fill Algorithm

- Figure below shows the procedure for scan-line filling algorithm.

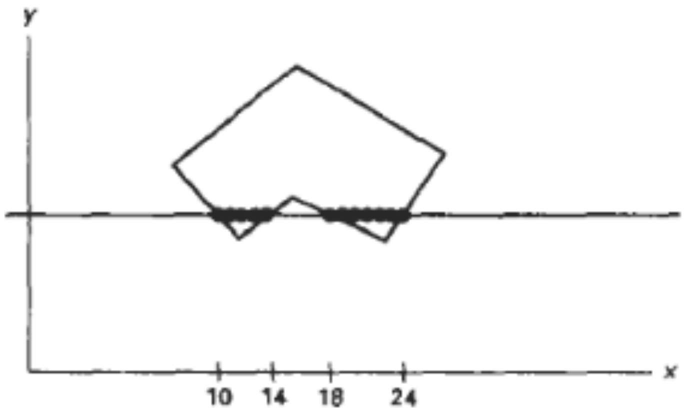


Fig. 2.17: - Interior pixels along a scan line passing through a polygon area.

- For each scan-line crossing a polygon, the algorithm locates the intersection points are of scan line with the polygon edges.
- This intersection points are stored from left to right.
- Frame buffer positions between each pair of intersection point are set to specified fill color.
- Some scan line intersects at vertex position they are required special handling.
- For vertex we must look at the other endpoints of the two line segments of the polygon which meet at this vertex.
- If these points lie on the same (up or down) side of the scan line, then that point is counts as two intersection points.
- If they lie on opposite sides of the scan line, then the point is counted as single intersection.
- This is illustrated in figure below

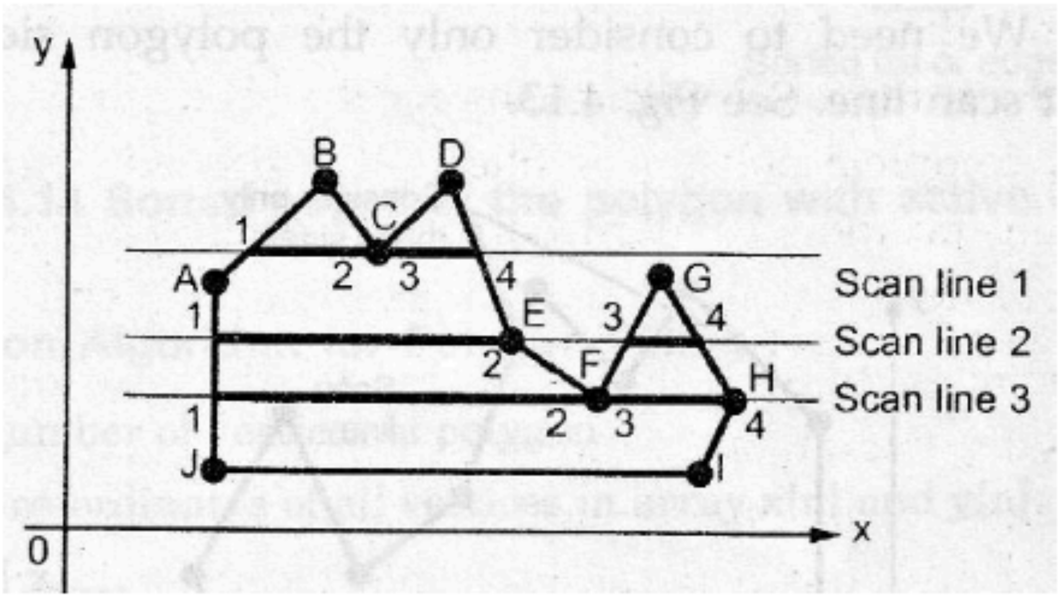


Fig. 2.18: - Intersection points along the scan line that intersect polygon vertices.

- As shown in the Fig. 2.18, each scan line intersects the vertex or vertices of the polygon. For scan line 1, the other end points (B and D) of the two line segments of the polygon lie on the same side of the scan line, hence there are two intersections resulting two pairs: 1 -2 and 3 - 4. Intersections points 2 and 3 are actually same Points. For scan line 2 the other endpoints (D and F) of the two line segments of the Polygon lie on the opposite sides of the scan line, hence there is a single intersection resulting two pairs: l - 2 and 3 - 4. For scan line 3, two vertices are the intersection points"

- For vertex F the other end points E and G of the two line segments of the polygon lie on the same side of the scan line whereas for vertex H, the other endpoints G and I of the two line segments of the polygon lie on the opposite side of the scan line. Therefore, at vertex F there are two intersections and at vertex H there is only one intersection. This results two pairs: 1 - 2 and 3 - 4 and points 2 and 3 are actually same points.
- Coherence methods often involve incremental calculations applied along a single scan line or between successive scan lines.
- In determining edge intersections, we can set up incremental coordinate calculations along any edge by exploiting the fact that the slope of the edge is constant from one scan line to the next.
- Figure below shows three successive scan-lines crossing the left edge of polygon.

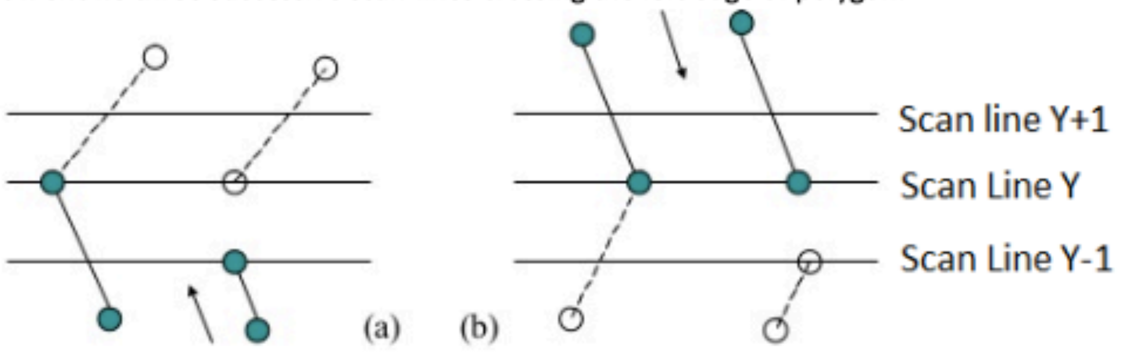


Fig. 2.18: - adjacent scan line intersects with polygon edge.

- For above figure we can write slope equation for polygon boundary as follows.

$$m = \frac{y_{k+1} - y_k}{x_{k+1} - x_k}$$

- Since change in y coordinates between the two scan lines is simply

$$y_{k+1} - y_k = 1$$

- So slope equation can be modified as follows

$$m = \frac{y_{k+1} - y_k}{x_{k+1} - x_k}$$

$$m = \frac{1}{x_{k+1} - x_k}$$

$$x_{k+1} - x_k = \frac{1}{m}$$

$$x_{k+1} = x_k + \frac{1}{m}$$

- Each successive x intercept can thus be calculated by adding the inverse of the slope and rounding to the nearest integer.
- For parallel execution of this algorithm we assign each scan line to separate processor in that case instead of using previous x values for calculation we use initial x values by using equation as.

$$x_k = x_0 + \frac{k}{m}$$

- Now if we put $m = \frac{\Delta y}{\Delta x}$ in incremental calculation equation $x_{k+1} = x_k + \frac{1}{m}$ then we obtain equation as.

$$x_{k+1} = x_k + \frac{\Delta x}{\Delta y}$$

- Using this equation we can perform integer evaluation of x intercept by initializing a counter to 0, then incrementing counter by the value of Δx each time we move up to a new scan line.
- When the counter value becomes equal to or greater than Δy, we increment the current x intersection value by 1 and decrease the counter by the value Δy.
- This procedure is seen by following figure.

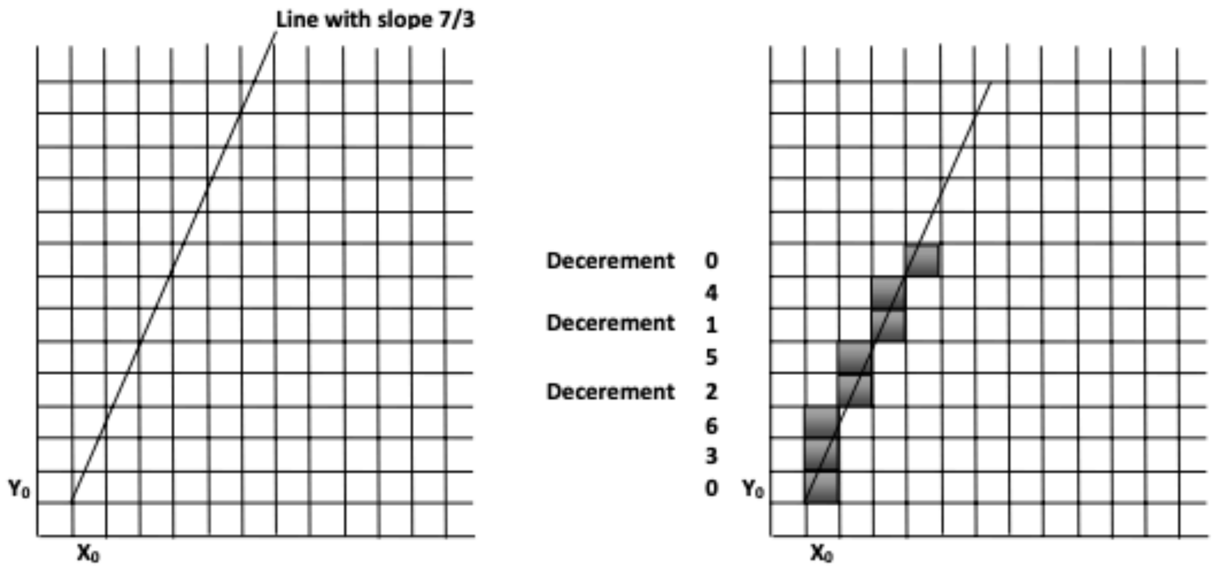
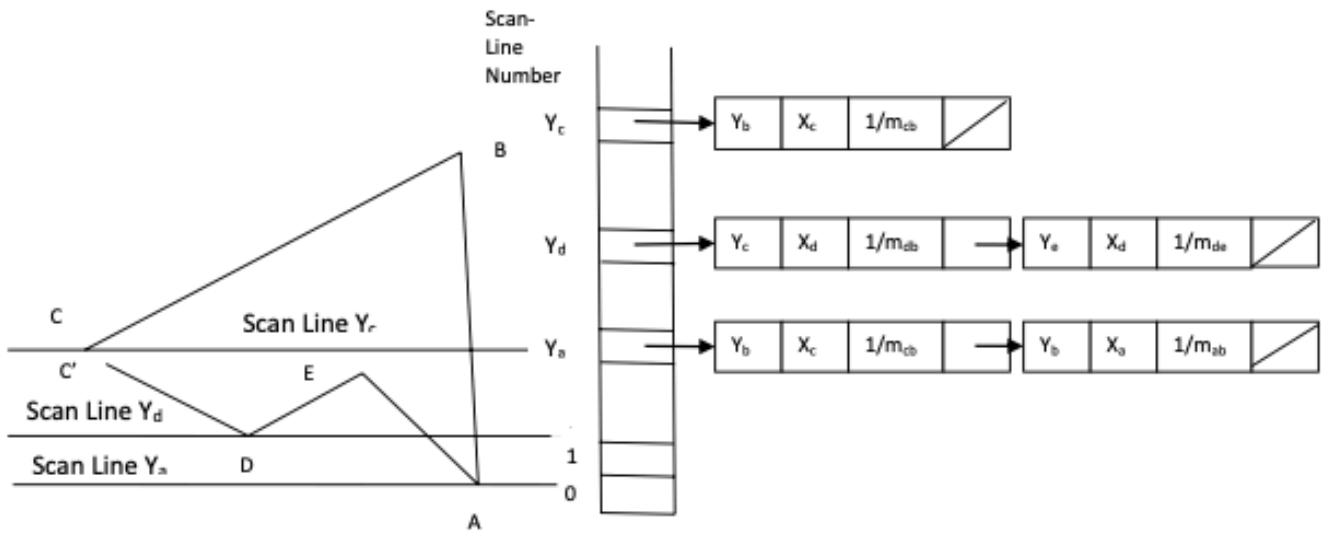


Fig. 2.19: - line with slope 7/3 and its integer calculation using equation $x_{k+1} = x_k + \frac{\Delta x}{\Delta y}$.

- Steps for above procedure

1. Suppose m = 7/3
2. Initially, set counter to 0, and increment to 3 (which is Δx).
3. When move to next scan line, increment counter by adding Δx
4. When counter is equal or greater than 7 (which is Δy), increment the x-intercept (in other words, the x-intercept for this scan line is one more than the previous scan line), and decrement counter by 7(which is Δy).

- To efficiently perform a polygon fill, we can first store the polygon boundary in a sorted edge table that contains all the information necessary to process the scan lines efficiently.
- We use bucket sort to store the edge sorted on the smallest y value of each edge in the correct scan line positions.
- Only the non-horizontal edges are entered into the sorted edge table.
- Figure below shows one example of storing edge table.



Scan-Line Number
Y_c
Y_d
Y_a
1
0
Y_b | X_c | $1/m_{cb}$
Y_c | X_d | $1/m_{db}$
Y_e | X_d | $1/m_{de}$
Y_b | X_c | $1/m_{cb}$
Y_b | X_a | $1/m_{ab}$
B
C
Scan Line Y_{c}
C'
E
Scan Line Y_d
Scan Line Y_a
D
A

Fig. 2.20: - A polygon and its sorted edge table.

- Each entry in the table for a particular scan line contains the maximum y values for that edges, the x intercept value for edge, and the inverse slope of the edge.
- For each scan line the edges are in sorted order from left to right.
- Than we process the scan line from the bottom to top for whole polygon and produce active edge list for each scan line crossing the polygon boundaries.
- The active edge list for a scan line contains all edges crossed by that line, with iterative coherence calculation used to obtain the edge intersections.

Inside-Outside Tests

- In area filling and other graphics operation often required to find particular point is inside or outside the polygon.
- For finding which region is inside or which region is outside most graphics package use either odd even rule or the nonzero winding number rule.

Odd Even Rule

- It is also called the odd parity rule or even odd rule.
- By conceptually drawing a line from any position p to a distant point outside the coordinate extents of the object and counting the number of edges crossing by this line is odd, than p is an interior point. Otherwise p is exterior point.
- To obtain accurate edge count we must sure that line selected is does not pass from any vertices.
- This is shown in figure 2.21(a).

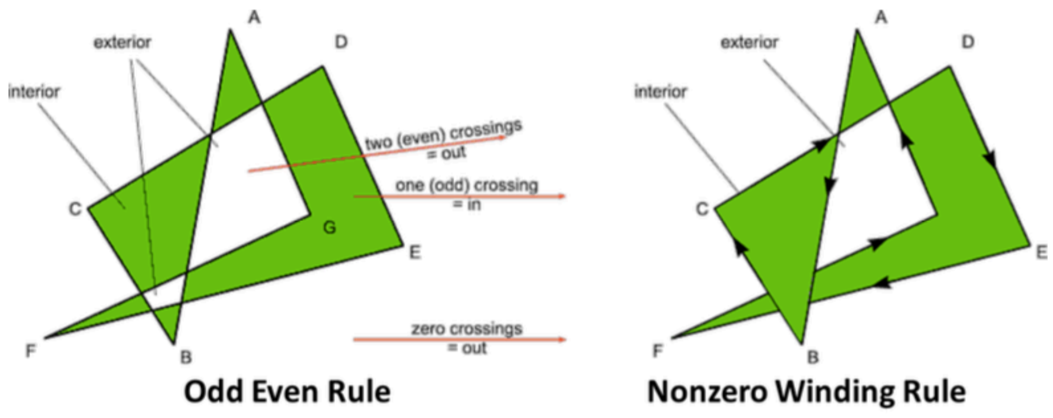


Fig. 2.21: - Identifying interior and exterior region for a self-intersecting polygon.

Scan-Line Fill of Curved Boundary Areas

- Scan-line fill of region with curved boundary is more time consuming as intersection calculation now involves nonlinear boundaries.
- For simple curve such as circle or ellipse scan line fill process is straight forward process.
- We calculate the two scan line intersection on opposite side of the curve.
- This is same as generating pixel position along the curve boundary using standard equation of curve.
- Then we fill the color between two boundary intersections.
- Symmetry property is used to reduce the calculation.
- Similar method can be used for fill the curve section.

Boundary Fill Algorithm/ Edge Fill Algorithm

- In this method, edges of the polygons are drawn. Then starting with some seed, any point inside the polygon we examine the neighbouring pixels to check whether the boundary pixel is reached.
- If boundary pixels are not reached, pixels are highlighted and the process is continued until boundary pixels are reached.
- Boundary defined regions may be either 4-cormected or 8-connected. as shown in the Figure below

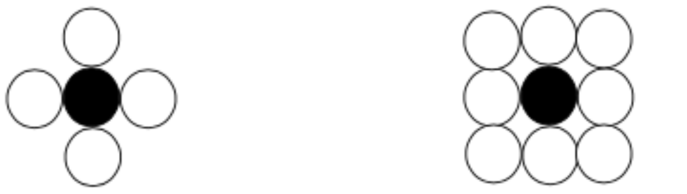


Fig. 2.22: - Neighbor pixel connected to one pixel.

- If a region is 4-connected, then every pixel in the region may be reached by a combination of moves in only four directions: left, right, up and down.
- For an 8-connected region every pixel in the region may be reached by a combination of moves in the two horizontal, two vertical, and four diagonal directions.
- In some cases, an 8-connected algorithm is more accurate than the 4-connected algorithm. This is illustrated in Figure below. Here, a 4-connected algorithm produces the partial fill.

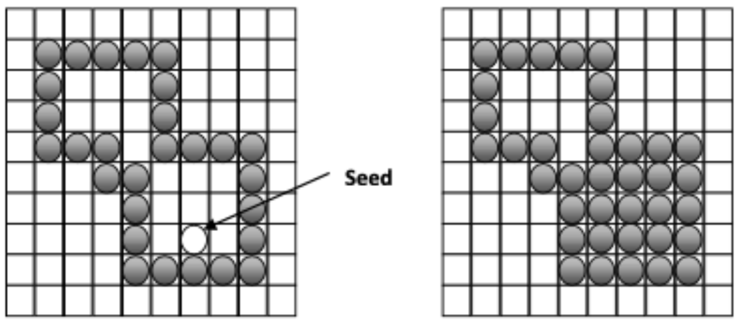


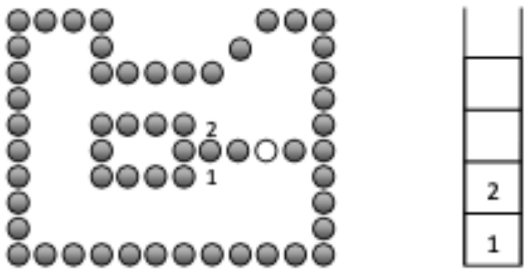
Fig. 2.23: - partial filling resulted due to using 4-connected algorithm.

- The following procedure illustrates the recursive method for filling a 4-connected region with color specified in parameter fill color (f-color) up to a boundary color specified with parameter boundary color (b-color).
- Procedure :

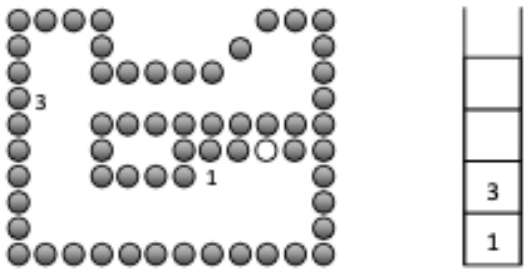
```
boundary-fill4(x, y, f-color, b-color)
{
    if(getpixel (x,y) ! = b-color&&gepixel (x, y) ! = f-color)
    {
            putpixel (x, y, f-color)
            boundary-fill4(x + 1, y, f-color, b-color);
            boundary-fill4(x, y + 1, f-color, b-color);
            boundary-fill4(x - 1, y, f-color, b-color);
            boundary-fill4(x, y - l, f-color, b-color);
    }
}
```

- Note: 'getpixel' function gives the color of .specified pixel and 'putpixel' function draws the pixel with specified color.
- Same procedure can be modified according to 8 connected region algorithm by including four additional statements to test diagonal positions, such as $(x + 1, y + 1)$.
- This procedure requires considerable stacking of neighbouring points more, efficient methods are generally employed.
- This method fill horizontal pixel spans across scan lines, instead of proceeding to 4 connected or 8 connected neighbouring points.
- Then we need only stack a beginning position for each horizontal pixel span, instead of stacking all unprocessed neighbouring positions around the current position.
- Starting from the initial interior point with this method, we first fill in the contiguous span of pixels on this starting scan line.
- Then we locate and stack starting positions for spans on the adjacent scan lines, where spans are defined as the contiguous horizontal string of positions bounded by pixels displayed in the area border color.
- At each subsequent step, we unstack the next start position and repeat the process.
- An example of how pixel spans could be filled using this approach is illustrated for the 4-connected fill region in Figure below.

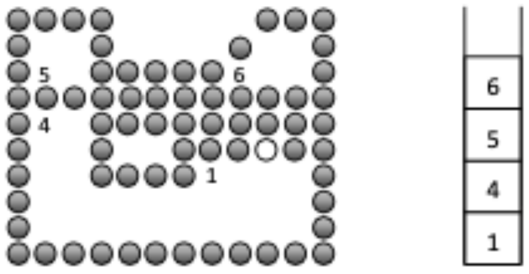
(a)



(b)



(c)



(d)

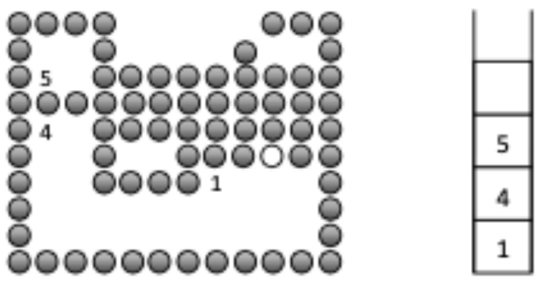


Fig. 2.24: - Boundary fill across pixel spans for a 4-connected area.

Flood-Fill Algorithm

- Sometimes it is required to fill in an area that is not defined within a single color boundary.
- In such cases we can fill areas by replacing a specified interior color instead of searching for a boundary color.
- This approach is called a flood-fill algorithm. Like boundary fill algorithm, here we start with some seed and examine the neighbouring pixels.
- However, here pixels are checked for a specified interior color instead of boundary color and they are replaced by new color.
- Using either a 4-connected or 8-connected approach, we can step through pixel positions until all interior point have been filled.
- The following procedure illustrates the recursive method for filling 4-connected region using flood-fill algorithm.
- Procedure :

```
flood-fill4(x, y, new-color, old-color)
{
    if(getpixel (x,y) = = old-color)
    {
            putpixel (x, y, new-color)
            flood-fill4 (x + 1, y, new-color, old -color);
            flood-fill4 (x, y + 1, new -color, old -color);
            flood-fill4 (x - 1, y, new -color, old -color);
            flood-fill4 (x, y - l, new -color, old-color);
    }
}
```

- Note: 'getpixel' function gives the color of .specified pixel and 'putpixel' function draws the pixel with specified color.

Character Generation

- We can display letters and numbers in variety of size and style.
- The overall design style for the set of character is called typeface.
- Today large numbers of typefaces are available for computer application for example Helvetica, New York platino etc.
- Originally, the term font referred to a set of cast metal character forms in a particular size and format, such as 10-point Courier Italic or 12- point Palatino Bold. Now, the terms font and typeface are often used interchangeably, since printing is no longer done with cast metal forms.
- Two different representations are used for storing computer fonts.

Bitmap Font/ Bitmapped Font

- A simple method for representing the character shapes in a particular typeface is to use rectangular grid patterns.
- Figure below shows pattern for particular letter.

1	1	1	1	1	1	0
0	1	1	0	0	1	1
0	1	1	0	0	1	1
0	1	1	1	1	1	0
0	1	1	0	0	1	1
0	1	1	0	0	1	1
1	1	1	1	1	1	0

Fig. 2.25: - Grid pattern for letter B.

- When the pattern in figure 2.25 is copied to the area of frame buffer, the 1 bits designate which pixel positions are to be displayed on the monitor.
- Bitmap fonts are the simplest to define and display as character grid only need to be mapped to a frame-buffer position.
- Bitmap fonts require more space because each variation (size and format) must be stored in a font cache.
- It is possible to generate different size and other variation from one set but this usually does not produce good result.

Outline Font

- In this method character is generated using curve section and straight line as combine assembly.
- Figure below shows how it is generated.

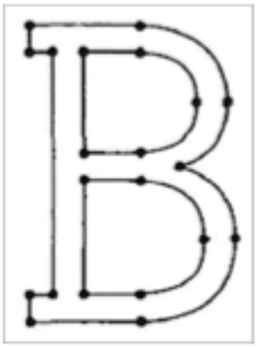


Fig. 2.26: - outline for letter B.

- To display the character shown in figure 2.26 we need to fill interior region of the character.
- This method requires less storage since each variation does not required a distinct font cache.
- We can produce boldface, italic, or different sizes by manipulating the curve definitions for the character outlines.
- But this will take more time to process the outline fonts, because they must be scan converted into the frame buffer.

Stroke Method

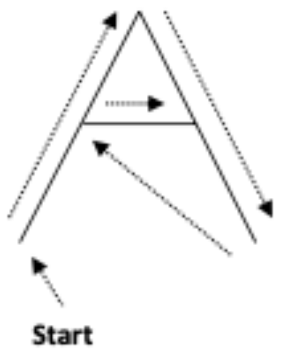
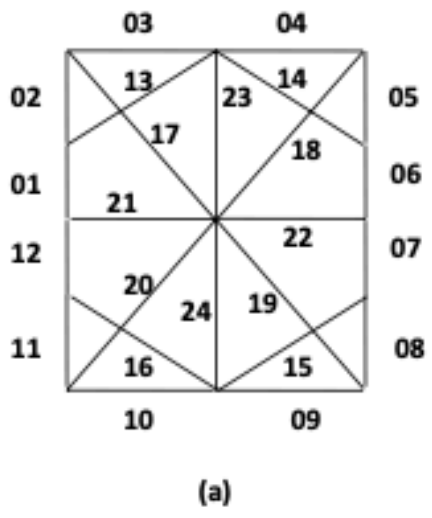


Fig. 2.27: - Stroke Method for Letter A.

- It uses small line segments to generate a character.
- The small series of line segments are drawn like a stroke of a pen to form a character as shown in figure.
- We can generate our own stroke method by calling line drawing algorithm.
- Here it is necessary to decide which line segments are needs for each character and then draw that line to display character.
- It support scaling by changing length of line segment.

Starbust Method



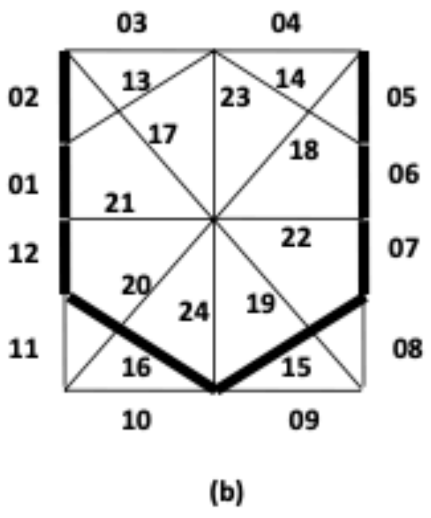


Fig. 2.28: - (a) Starbust Method. (b) Letter V using starbust method

- In this method a fix pattern of line segments are used to generate characters.
- As shown in figure 2.28 there are 24 line segments.
- We highlight those lines which are necessary to draw a particular character.
- Pattern for particular character is stored in the form of 24 bit code. In which each bit represents corresponding line having that number.
- That code contains 0 or 1 based on line segment need to highlight. We put bit value 1 for highlighted line and 0 for other line.
- Code for letter V is
  1 1 0 0 1 1 1 0 0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 0
- This technique is not used now a days because:
  1. It requires more memory to store 24 bit code for single character.

2. It requires conversion from code to character.
3. It doesn’t provide curve shapes.