Filled Area Primitives:

Note: For filling a given picture or object with color's, we can do it in two ways in C programming. The two ways are given below:

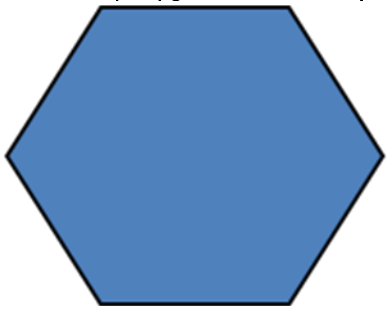
i. Using filling algorithms such as Floodfill algorithm, Boundary fill algorithm and scanline polygon fill algorithm, we can color the objects.
ii. Using inbuilt graphics functions such as floodfill(),setfillstyle() we can fill the object with color's directly without using any filling algorithm.

Here we will see the **filling algorithms**

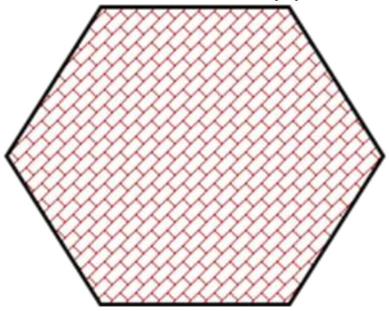
Polygon Filling

Types of filling

- Solid-fill
  All the pixels inside the polygon's boundary are illuminated.



- Pattern-fill
  the polygon is filled with an arbitrary predefined pattern.



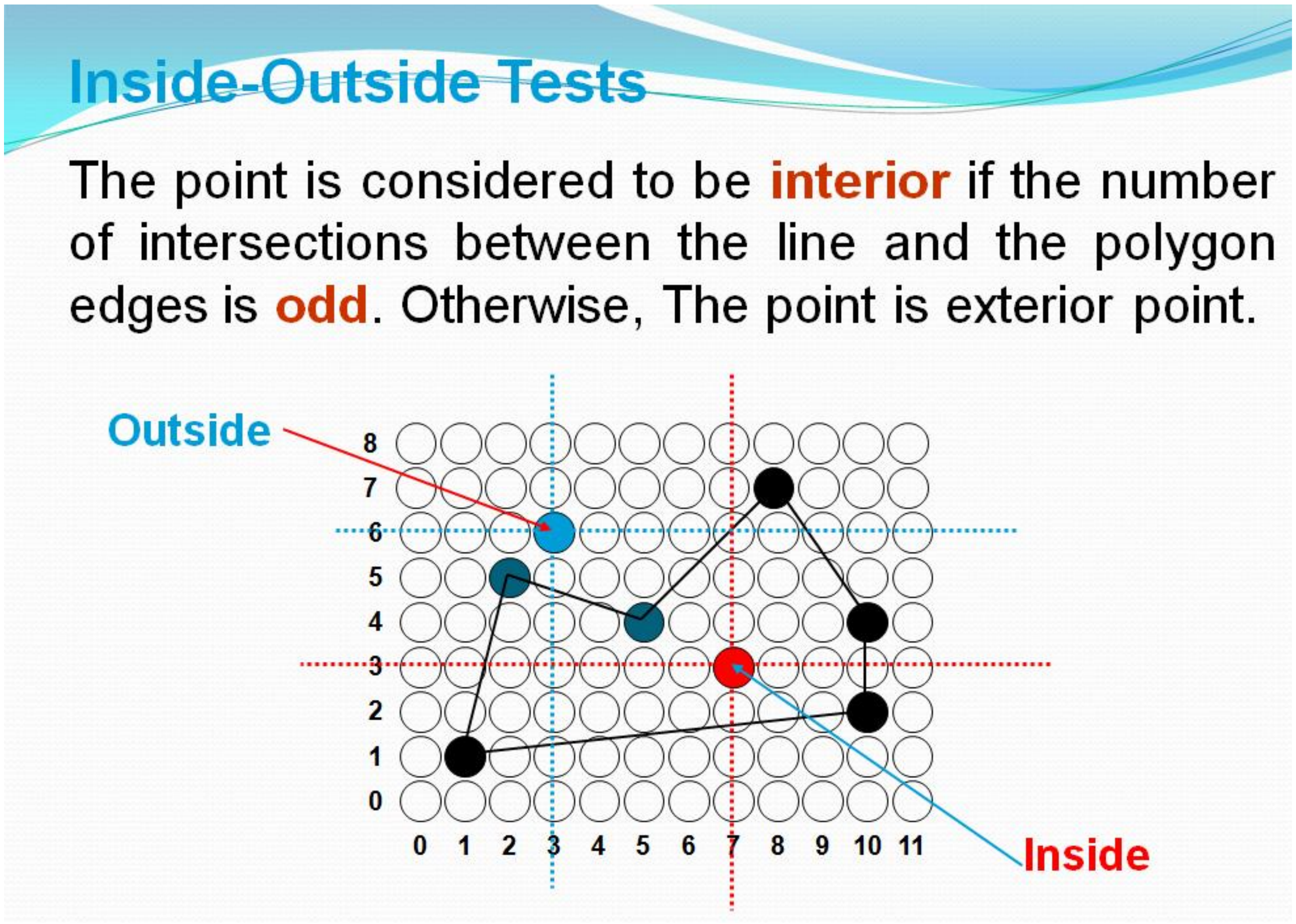
Polygon Representation

- The polygon can be represented by listing its n vertices in an ordered list.
  P = {(x_1, y_1), (x_2, y_2), ……., (x_n, y_n)}.
- The polygon can be displayed by drawing a line between (x_1, y_1), and (x_2, y_2), then a line between (x_2, y_2), and (x_3, y_3), and so on until the end vertex. In order to close up the polygon, a line between (x_n, y_n), and (x_1, y_1) must be drawn.
- One problem with this representation is that if we wish to translate the polygon, it is necessary to apply the translation transformation to each vertex in order to obtain the translated polygon.

Inside-Outside Tests

- When filling polygons we should decide whether a particular point is interior or exterior to a polygon.
- A rule called the odd-parity (or the odd-even rule) is applied to test whether a point is interior or not.
- To apply this rule, we conceptually draw a line starting from the particular point and extending to a distance point outside the coordinate extends of the object in any direction such that no polygon vertex intersects with the line.

Inside-Outside Tests



Inside-Outside Tests

The point is considered to be **interior** if the number of intersections between the line and the polygon edges is **odd**. Otherwise, The point is exterior point.

Outside
8
7
6
5
4
3
2
1
0
0 1 2 3 4 5 6 7 8 9 10 11
Inside

The Filling Algorithms are 3:

They are:

Three Algorithms for filling areas:

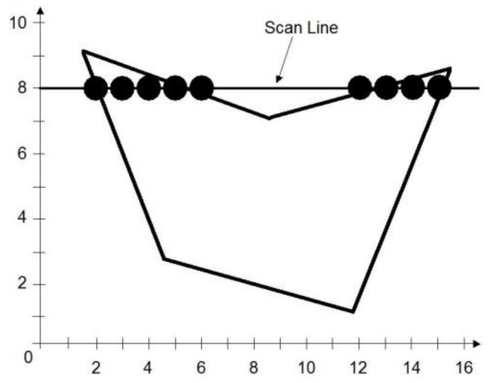
- 1) Scan Line Polygon Fill Algorithm
- 2) Boundary Fill Algorithm
- 3)Flood fill Algorithm

Scan Line Polygon Fill Algorithm:

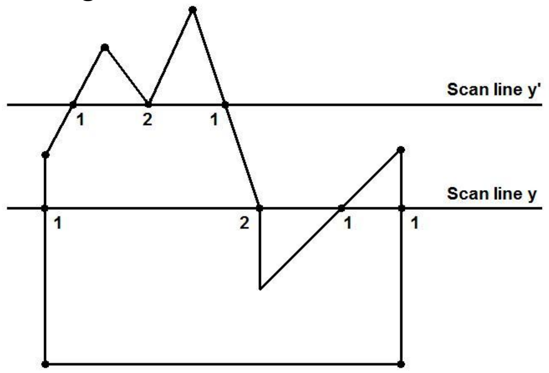
- The basic scan-line algorithm is as follows:
    - Find the intersections of the scan line with all edges of the polygon
    - Sort the intersections by increasing x coordinate
    - Fill in all pixels between pairs of intersections that lie interior to the polygon

The scan-line polygon-filling algorithm involves

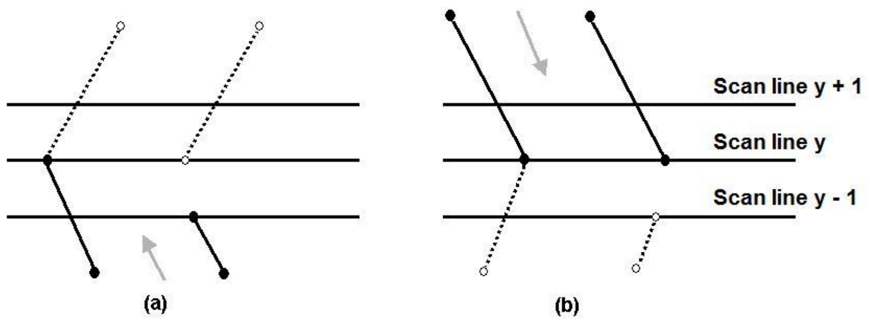
- the **horizontal scanning** of the polygon from its **lowermost** to its **topmost** vertex,
- identifying which edges intersect the scan-line,
- and finally drawing the interior horizontal lines with the specified fill color. process.



Dealing with vertices:



- When the endpoint **y** coordinates of the two edges are **increasing**, the **y** value of the upper endpoint for the **current edge** is decreased by one (a)
- When the endpoint **y** values are **decreasing**, the **y** value of the **next edge** is decreased by one (b)

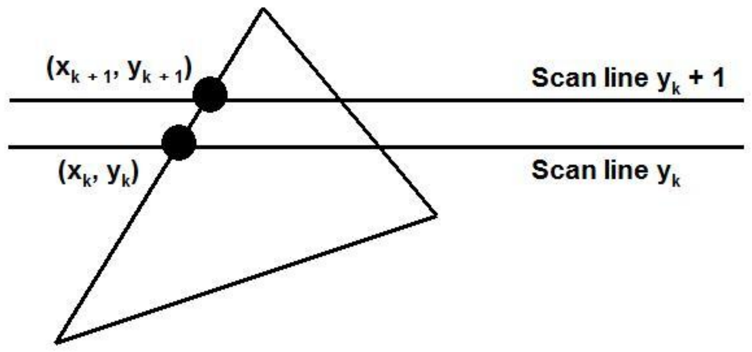


- **Determining Edge Intersections**

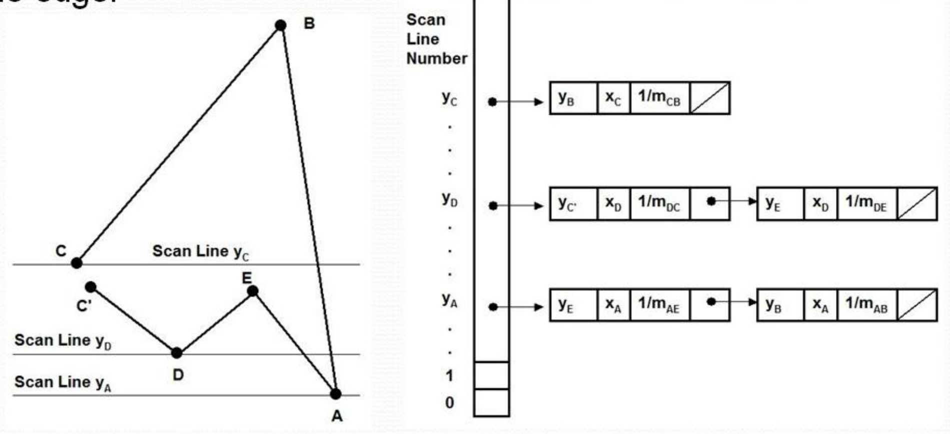
  $m = (y_{k+1} – y_k) / (x_{k+1} – x_k)$

  $y_{k+1} – y_k = 1$

  $x_{k+1} = x_k + 1/m$



• Each **entry** in the table for a particular scan line contains the **maximum y** value for that edge, the **x-intercept** value (**at the lower vertex**) for the edge, and the **inverse slope** of the edge.


B
C
Scan Line y_C
E
C'
Scan Line y_D
D
Scan Line y_A
A
Scan Line Number
y_C
y_D
y_A
1
0
y_B | x_C | $1/m_{CB}$
$y_{C'}$ | x_D | $1/m_{DC}$
y_E | x_D | $1/m_{DE}$
y_E | x_A | $1/m_{AE}$
y_B | x_A | $1/m_{AB}$

Steps in algorithm:

1. Find the minimum enclosed rectangle
2. Here the number of scanlines are equal to (ymax-ymin+1)
3. For each scanline, do

   a. Obtain the intersection points of scanline with polygon edges
   b. Sort the intersection edges from left to right
   c. Form the pairs of intersection points from the obtained list
   d. Fill with colors with in each pair if intersection points
   e. Repeat above procedure until ymax

Algorithm Steps:

1. the horizontal scanning of the polygon from its lowermost to its topmost vertex
2. identify the edge intersections of scan line with polygon
3. Build the edge table
    a. Each entry in the table for a particular scan line contains the maximum y value for that edge, the x-intercept value (at the lower vertex) for the edge, and the inverse slope of the edge.
4. Determine whether any edges need to be splitted or not. If there is need to split, split the edges.
5. Add new edges and build modified edge table.
6. Build Active edge table for each scan line and fill the polygon based on intersection of scanline with polygon edges.

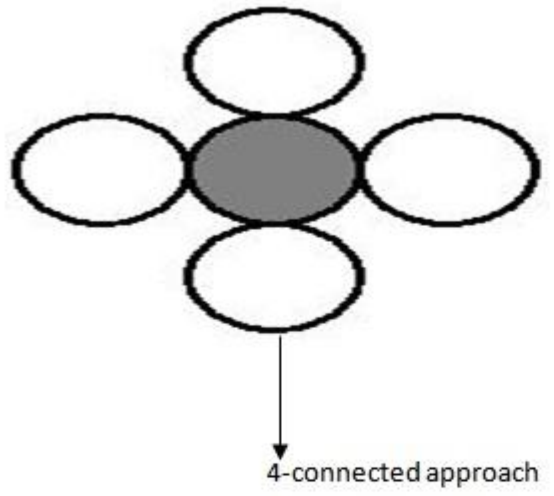
Boundary Fill Algorithm:

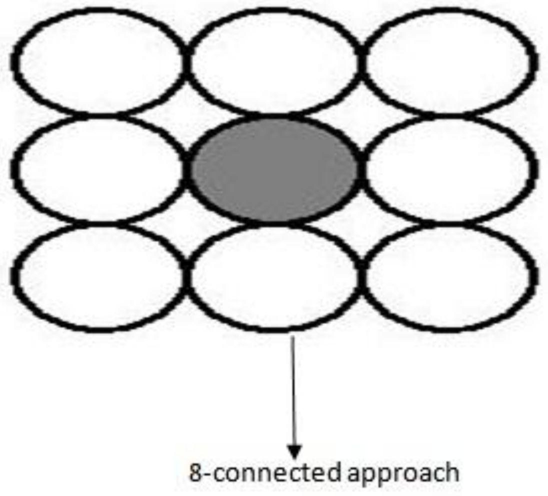
- Start at a point inside a region and paint the interior outward toward the boundary.
- If the boundary is specified in a single color, the fill algorithm processed outward pixel by pixel until the boundary color is encountered.
- A boundary-fill procedure accepts as input the coordinate of the interior point (x, y), a fill color, and a boundary color.

Algorithm:

The following steps illustrate the idea of the recursive boundary-fill algorithm:

1. Start from an interior point.
2. If the current pixel is not already filled and if it is not an edge point, then set the pixel with the fill color, and store its neighboring pixels (**4 or 8-connected**). Store only neighboring pixel that is not already filled and is not an edge point.
3. Select the next pixel from the stack, and continue with step 2.





In 4 connected approach, we can fill an object in only 4 directions. We have 4 possibilities for proceeding to next pixel from current pixel.

In 8 connected approach, we can fill an object in 8 directions. We have 8 possibilities for proceeding to next pixel from current pixel.

Function for 4 connected approach:

```
void boundary_fill(int x, int y, int fcolor, int bcolor)
{
   if ((getpixel(x, y) != bcolor) && (getpixel(x, y) != fcolor))
    {                delay(10);
                    putpixel(x, y, fcolor);
                    boundary_fill(x + 1, y, fcolor, bcolor);
                    boundary_fill(x - 1, y, fcolor, bcolor);
                    boundary_fill(x, y + 1, fcolor, bcolor);
                    boundary_fill(x, y - 1, fcolor, bcolor);
   }
}
```

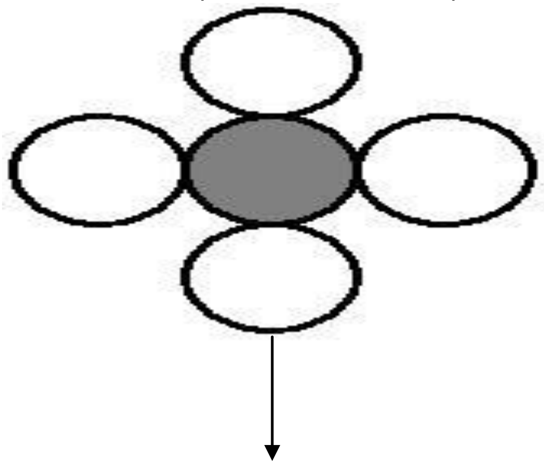
Function for 8 connected approach:

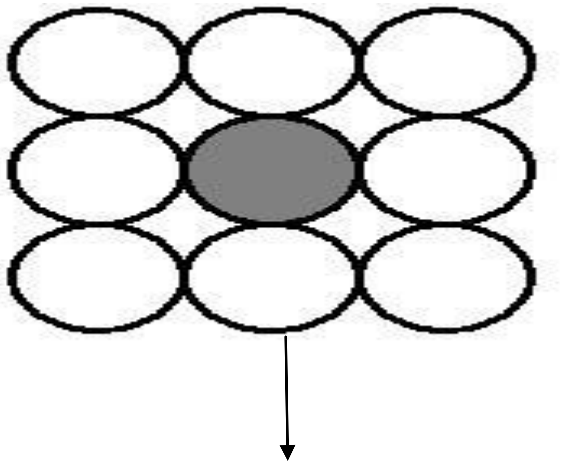
```
void boundary_fill(int x, int y, int fcolor, int bcolor)
{

   if ((getpixel(x, y) != bcolor) && (getpixel(x, y) != fcolor))
    {                         delay(10);
          putpixel(x, y, fcolor);
          boundary_fill(x + 1, y, fcolor, bcolor);
          boundary_fill(x , y+1, fcolor, bcolor);
          boundary_fill(x+1, y + 1, fcolor, bcolor);
          boundary_fill(x-1, y - 1, fcolor, bcolor);
          boundary_fill(x-1, y, fcolor, bcolor);
          boundary_fill(x , y-1, fcolor, bcolor);
          boundary_fill(x-1, y + 1, fcolor, bcolor);
          boundary_fill(x+1, y - 1, fcolor, bcolor);

     }
}
```

Flood Fill Algorithm:

Sometimes we want to fill in (recolor) an area that is not defined within a single color boundary. We paint such areas by replacing a specified interior color instead of searching for a boundary color value. This approach is called a flood-fill algorithm.

1. We start from a specified interior pixel (x, y) and reassign all pixel values that are currently set to a given interior color with the desired fill color.
2. If the area has more than one interior color, we can first reassign pixel values so that all interior pixels have the same color.
3. Using either 4-connected or 8-connected approach, we then step through pixel positions until all interior pixels have been repainted.


4-connected approach


8-connected approach

4-connected Flood Fill approach:

1. We can implement flood fill algorithm by using recursion.
2. First all the pixels should be reassigned to common color. here common color is black.
3. Start with a point inside given object, check the following condition:

   if(getpixel(x,y)==old_col)---old_col is common color
4. If above condition is satisfied, then following 4 steps are followed in filling the object.

```
//Recursive 4-way floodfill.

putpixel(x,y,fill_col);
flood(x+1,y,fill_col,old_col);
flood(x-1,y,fill_col,old_col);
flood(x,y+1,fill_col,old_col);
flood(x,y-1,fill_col,old_col);
```

8-connected Flood fill approach:

1. We can implement flood fill algorithm by using recursion.
2. First all the pixels should be reassigned to common color. here common color is black.
3. Start with a point inside given object, check the following condition:

   if(getpixel(x,y)==old_col)---old_col is common color
4. If above condition is satisfied, then following 8 steps are followed in filling the object.

   //Recursive 4-way floodfill.

```
putpixel(x,y,fill_col);
flood(x+1,y,fill_col,old_col);
flood(x-1,y,fill_col,old_col);
flood(x,y+1,fill_col,old_col);
flood(x,y-1,fill_col,old_col);
flood(x + 1, y - 1, fill_col, old_col);
flood(x + 1, y + 1, fill_col, old_col);
flood(x - 1, y - 1, fill_col, old_col);
flood(x - 1, y + 1, fill_col, old_col);
```