Computer Graphics

Module - 2

Output Primitives

CSC305

By Prof. Sunil Katkar

Department of Computer Engineering, VCET

Module -2 Output Primitives

Objective

To emphasize on implementation aspect of Computer Graphics Algorithms

.

Outcome

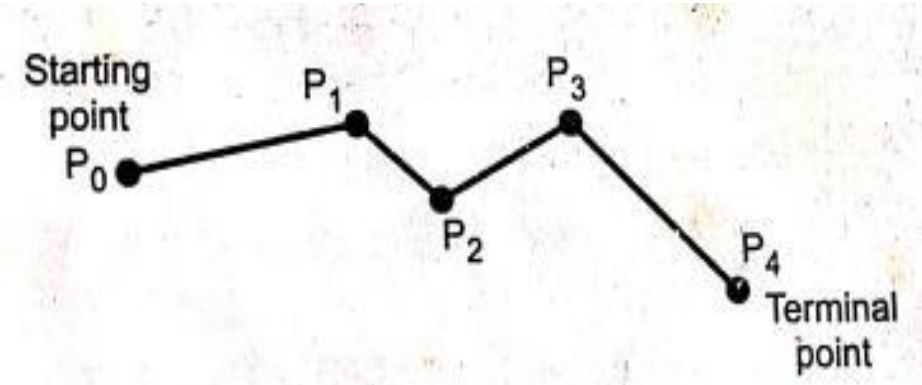
At the end of the course student will be able to:

apply scan conversions algorithms to draw point, line, circle, ellipse and compare flood fill, boundary fill algorithms

Filled Area Primitives

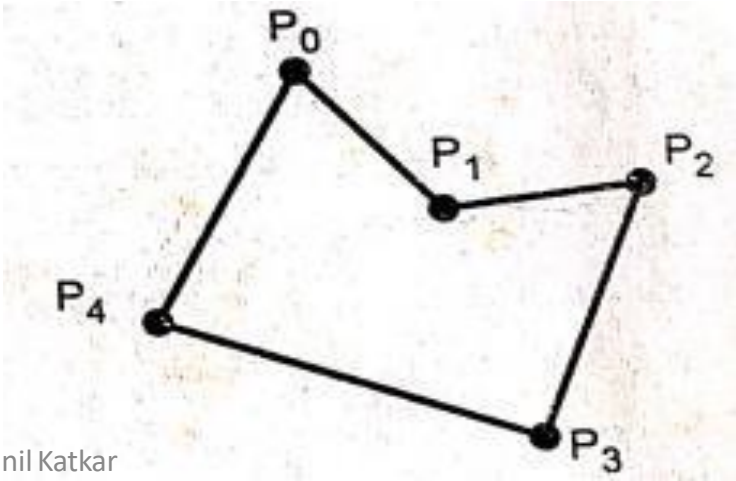
Polyline -

Polyline is a chain of connected line segments.



Polygon -

when starting point and terminal point of any polyline is same then it is called Polygon.

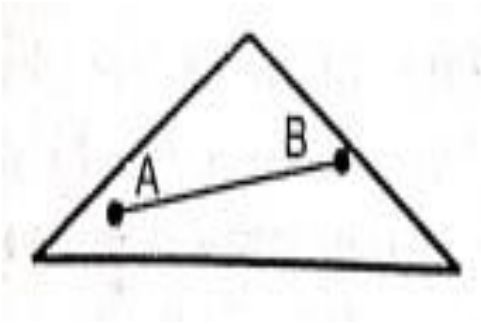


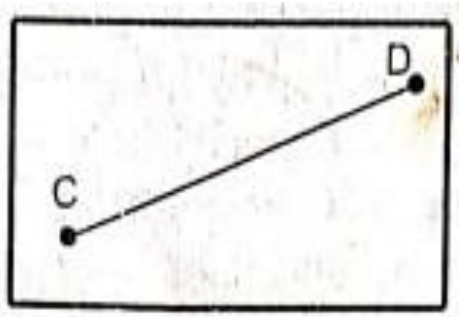
Types of Polygons -

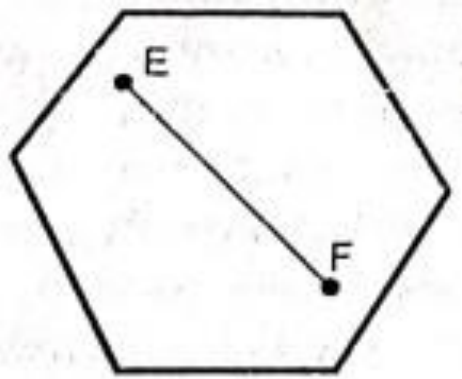
- ✓ Convex Polygon
- ✓ Concave Polygon

Convex Polygon -

Line segment joining any two points within the polygon lies completely inside the polygon.

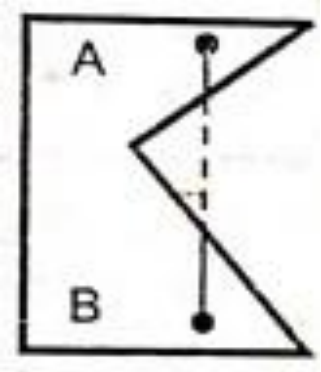


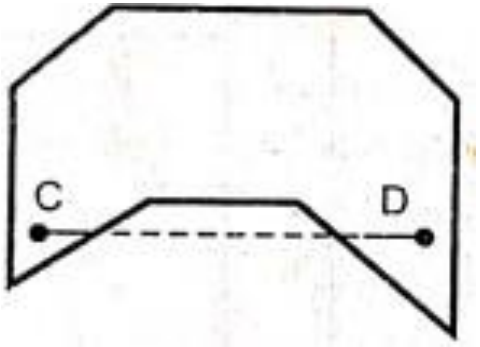


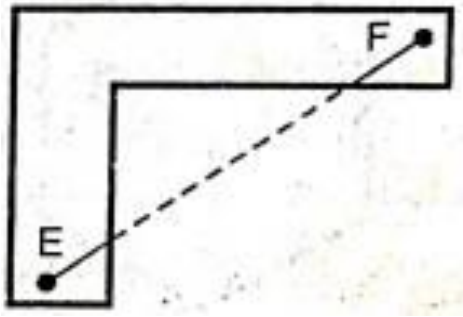


Concave Polygons -

Line segment joining any two points within the polygon may not lie completely inside the polygon.





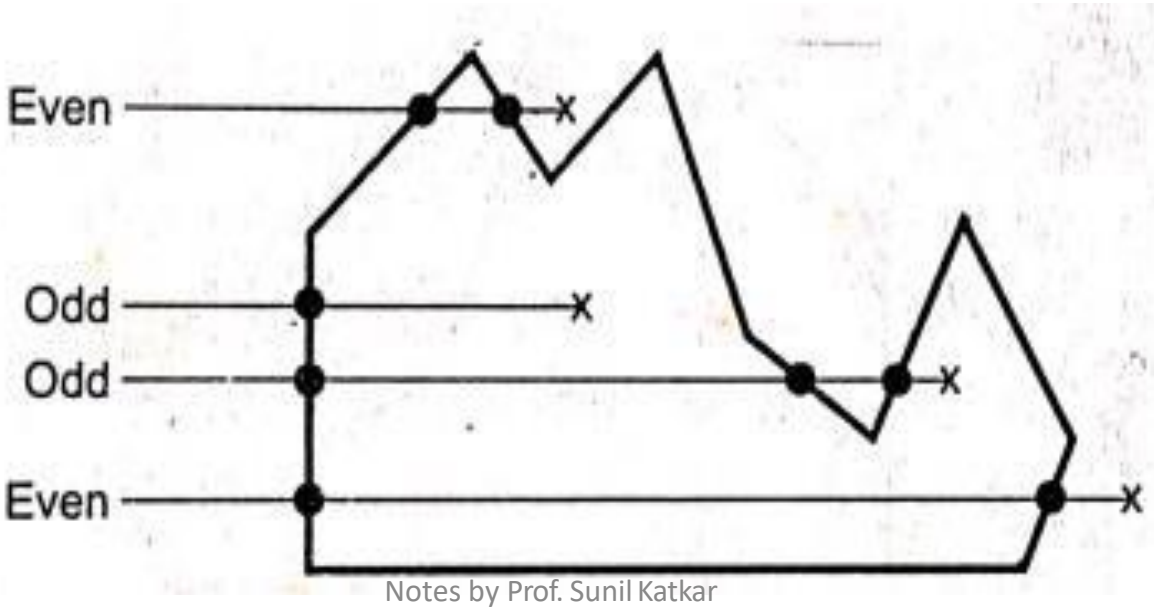


Inside Test

Determine whether or not a test point is inside of a polygon.

Even-Odd Test -

- ✓ Draw a horizontal line passing through the test point.
- ✓ Count the number of intersections of the line segments with the polygon boundary occur.
- ✓ If the number of intersections are ODD then then the test point is inside the polygon.
- ✓ Otherwise the test point is outside the polygon.

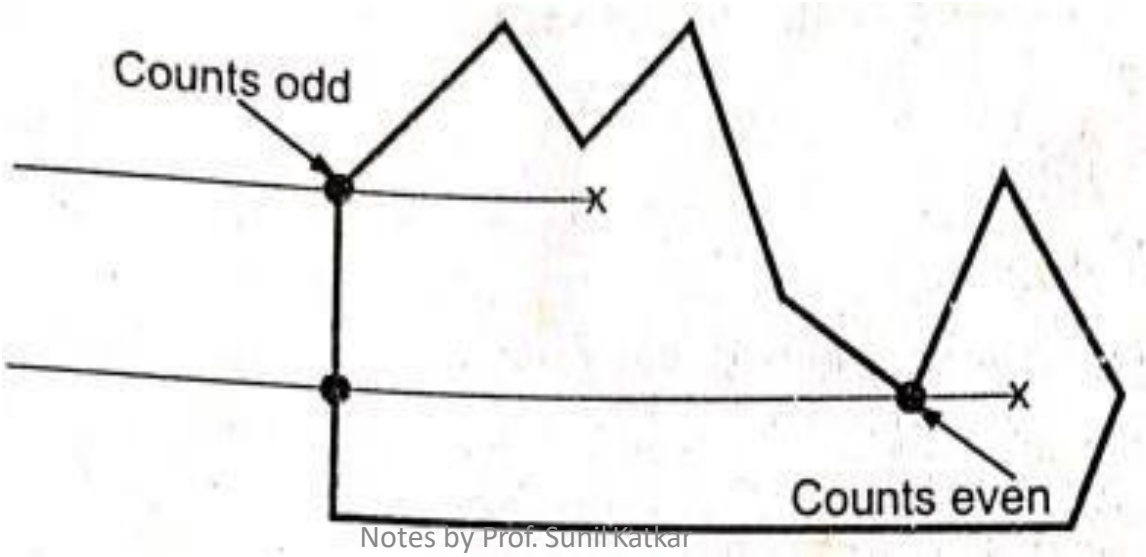


Inside Test

Note -

When a horizontal line passes through a vertex then follow the following procedure

- ✓ If the intersecting edges sharing a vertex are one side of the horizontal line (either above or below), consider **TWO** intersecting points.
- ✓ If the intersecting edges sharing a vertex are on opposite side of the horizontal line (above and below), consider **SINGLE** intersecting point.



Polygon Filling

Highlighting all the pixels which lie inside the polygon with any color other than background color.

Methods-

1. Seed Fill -

Start from a given seed point known to be inside the polygon and highlight outward from this point i.e. neighboring pixels until we encounter the boundary pixels.

- ✓ **Boundary / Edge Fill Algorithm -**
  Fill boundary-defined regions
- ✓ **Flood Fill Algorithm -**
  Fill interior-defined regions

2. Scan Line Fill -

Apply the inside test and then highlight pixels which lie inside the polygon.

Boundary / Edge Fill Algorithm

Edges of the polygon are drawn.

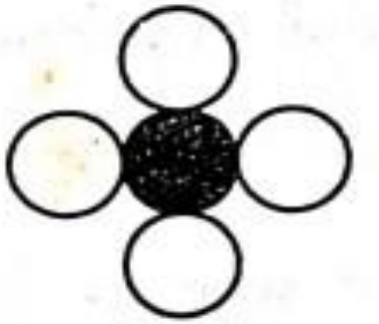
Start with some seed, any point inside the polygon.

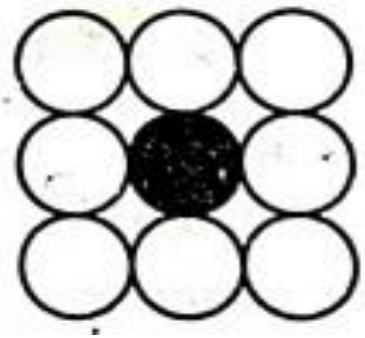
Examine the neighboring pixels to check whether the boundary pixel is reached.

If boundary pixels are not reached, neighboring pixels are highlighted and the process is continued until boundary pixels are reached.

If boundary is specified in a single color, the algorithm proceeds outward pixel by pixel until the boundary color is encountered.

Neighboring pixels are tested using either 4-connected or 8-connected methods.



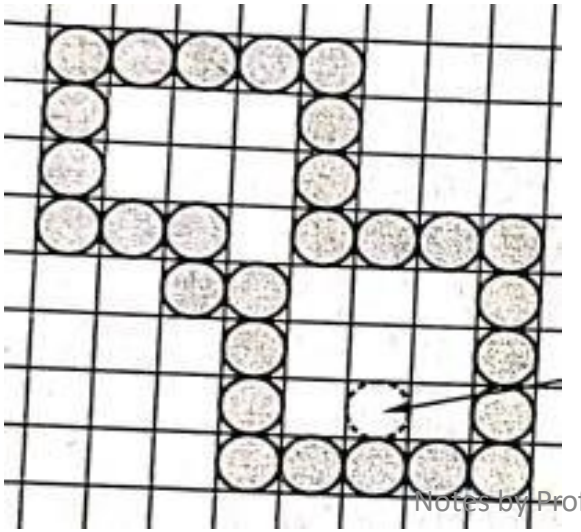


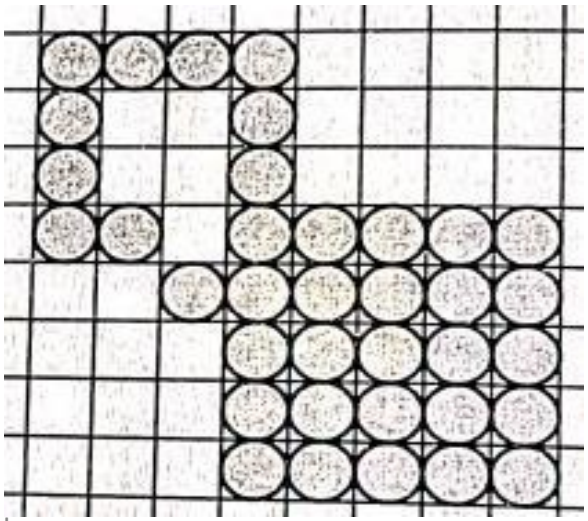
Boundary / Edge Fill Algorithm

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 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 8-connected approach is more accurate than the 4-connected approach because 4-connected algorithm produces the partial fill.





Procedure of Boundary / Edge Fill Algorithm

Fill a region with color specified in parameter fill color (f-color) up to a boundary color specified with parameter boundary color (b-color).

```
boundary_fill (x, y, f-color, b-color)
 {
if (getpixel (x, y) ! = b-color & & getpixel (x, y) ! = f-color)
        {
        putpixel (x, y, f-color)
        boundary_fill (x + 1, y, f-color, b-color)
        boundary_fill (x, y + 1, f-color, b-color)
        boundary_fill (x - 1, y, f-color, b-color)
        boundary_fill (x, y - 1, f-color, b-color)
        }
 }
```

Flood Fill Algorithm

It is used to fill an area that is not defined with a single boundary color.

Start with some seed, any point inside the polygon.

Examine the neighboring pixels of the seed pixel to check whether they are of the interior color.

If yes, they are filled with the desired fill color and their neighboring pixels are tested either 4-connected or 8-connected approach.

This process continues until all the pixels with an interior color are tested.

NOTE -

'getpixel' function gives the color of specified pixel

'putpixel' function draws the pixel with specified color

Procedure of Flood Fill Algorithm

```
flood_fill (x, y, old-color, new-color)
 {
if (getpixel (x, y) ≠ old-color) && (getpixel (x, y) ≠ new-color)
        {
        putpixel (x, y, new-color)
                flood_fill (x + 1, y, old-color, new-color)
                flood_fill (x - 1, y, old-color, new-color)
                flood_fill (x, y + 1, old-color, new-color)
                flood_fill (x, y - 1, old-color, new-color)
                flood_fill (x + 1, y + 1, old-color, new-color)
                flood_fill (x - 1, y - 1, old-color, new-color)
                flood_fill (x + 1, y - 1, old-color, new-color)
                flood_fill (x - 1, y + 1, old-color, new-color)
        }
 }
```

Limitations of Seed Fill Procedure

- ✓ This algorithm may not fill regions correctly if some interior pixels are already displayed in the fill color.
  To avoid this, we can first change the color of any interior pixels that are initially set to the fill color.
- ✓ This procedure requires considerable amount of stack memory.

Scan Line Algorithm

This algorithm proceeds scan line by scan line.

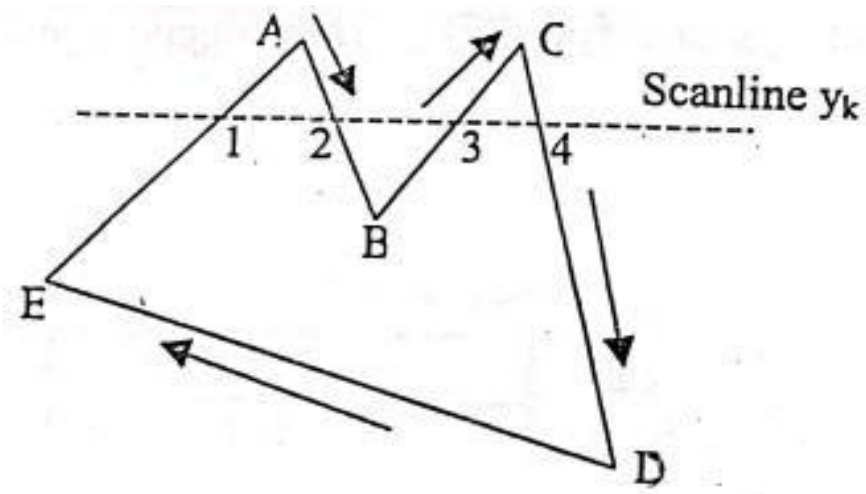
For the scan line crossing the polygon, area fill algorithm locates the intersection points with the polygon edges.

These intersection points are then stored in the increasing order of their x-values.

The corresponding location in the frame buffer are set to the specified fill color.

Scan Line Algorithm

1. Accepts the number of vertices in the polygon and coordinates of the vertices.



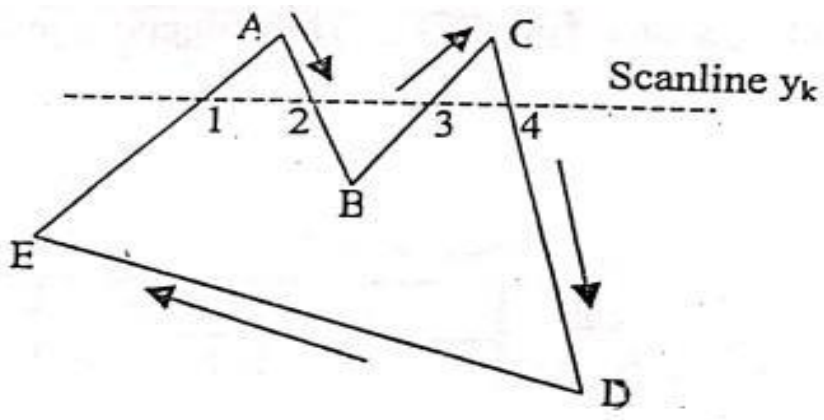
2. Active edge list is prepared for each scan line intersecting the polygon that contains the list of edges intersecting the scan line.

AB
BC
CD
EA

Scan Line Algorithm

3.For each polygon edge in the AEL, the intersecting points are calculated.

AB	(x_1, y_k)
BC	(x_2, y_k)
CD	(x_3, y_k)
EA	(x_4, y_k)



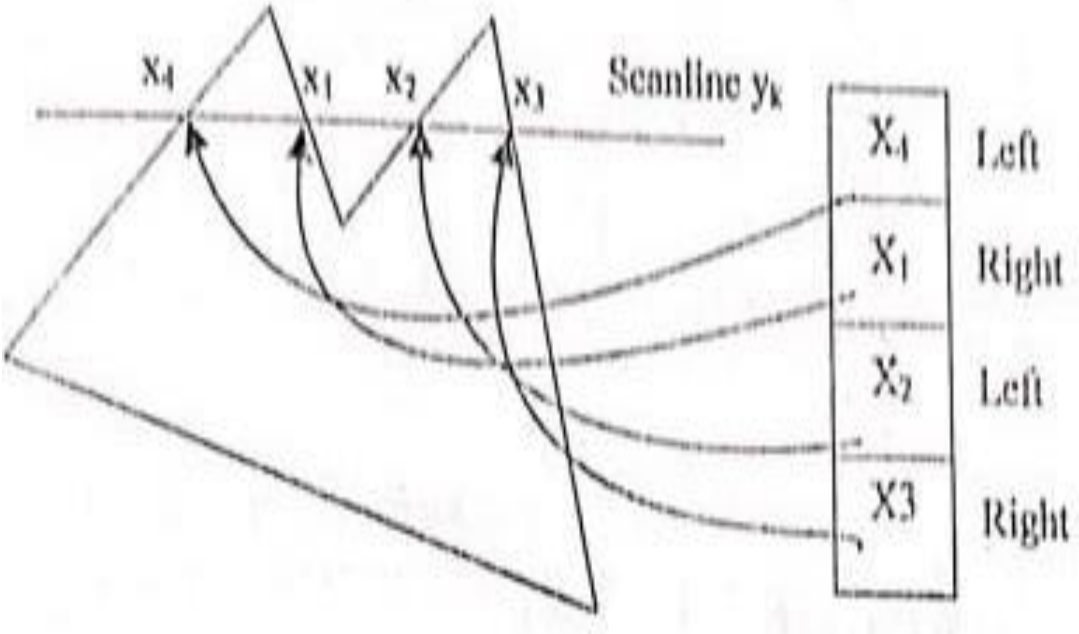
4. Sort all the edge intersecting points (EIP) in the increasing order of x-values.

Sorted ERP

EA	(x_4, y_k)
AB	(x_1, y_k)
BC	(x_2, y_k)
CD	(x_3, y_k)

Scan Line Algorithm

The first x-value gives the left boundary, second x-value gives the right boundary.

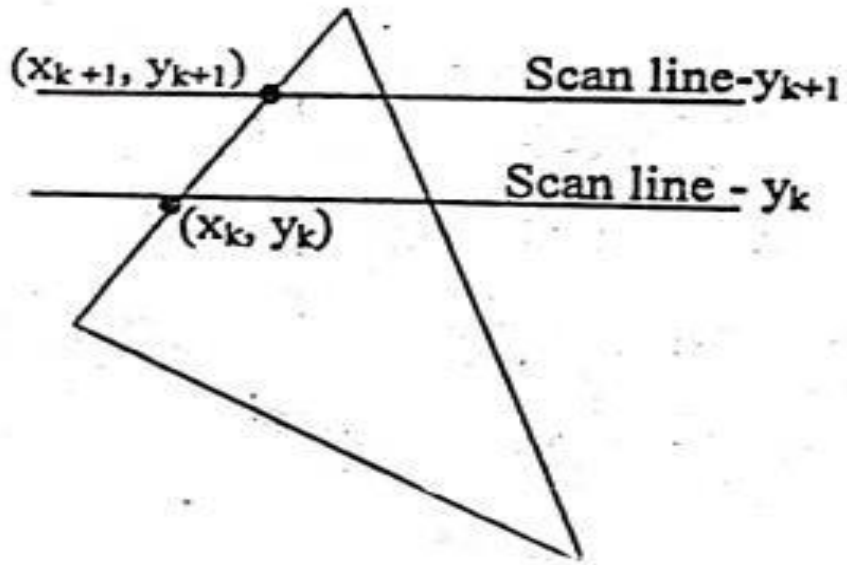


5.Fill all the pixels positions between left and right boundary with the specified color value.

6. Repeat the steps 2 to 5 for all scan lines.

7. Stop

Coherence



Two successive scan lines intersecting the left edge of the polygon.

The slope of this polygon boundary is

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

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

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

Each successive x-intercept can be calculated by $x_k = x_0 + k_0/m$

Where x_0 is the first intersection of the edge.

Scan Line Algorithm

1. Read n, the number of vertices of polygon.
2. Read x and y coordinates of all vertices in array x[n] and y[n].
3. Find y_{min} and y_{max}.
4. Store the initial x value (x_1) y values y_1 and y_2 for two endpoints and x increment Δ_x from scan line to scan line for each edge in the array edges [n] [4] .

   while doing this check that $y_1 > y_2$, if not interchange y_1 and y_2 and corresponding x_1 and x_2 so that for each edge, y_1 represents its maximum y coordinate and y_2 represents it minimum y coordinate.

5. Sort the rows of array, edges [n] [4] in descending order of y_1, descending order of y_2 and ascending order of x_2.
6. Set $y = y_{max}$.

Scan Line Algorithm

7. Find the active edges and update active edge list :

if ($y > y_2$ and $y \leq y_1$)

{ edge is active }

else

{ edge is not active}

8.Compute the x increments for all active edges for current y value [initially x-intersect is x_1 and x intersects for successive y values can be given as

$$x_{i+1} \leftarrow x_i + \Delta_x$$

where $\Delta_x = -1/m$ and $m = (y_2 - y_1) / (x_2 - x_1)$ i.e. slope of a line segment

9.If x intersect is vertex i.e. x-intersect = x_1 and $y = y_1$ then apply vertex test to check whether to consider one intersect or two intersects. Store all x intersects in the x-intersect [ ] array.

Scan Line Algorithm

10. Sort x-intersect [ ] array in the ascending order.

11. Extract pairs of intersects from the sorted x-intersect [ ] array.

12.Pass pairs of x values to line drawing routine to draw corresponding line segments.

13. Set $y = y - 1$.

14. Repeat steps 7 through 13 until $y \geq y_{min.}$

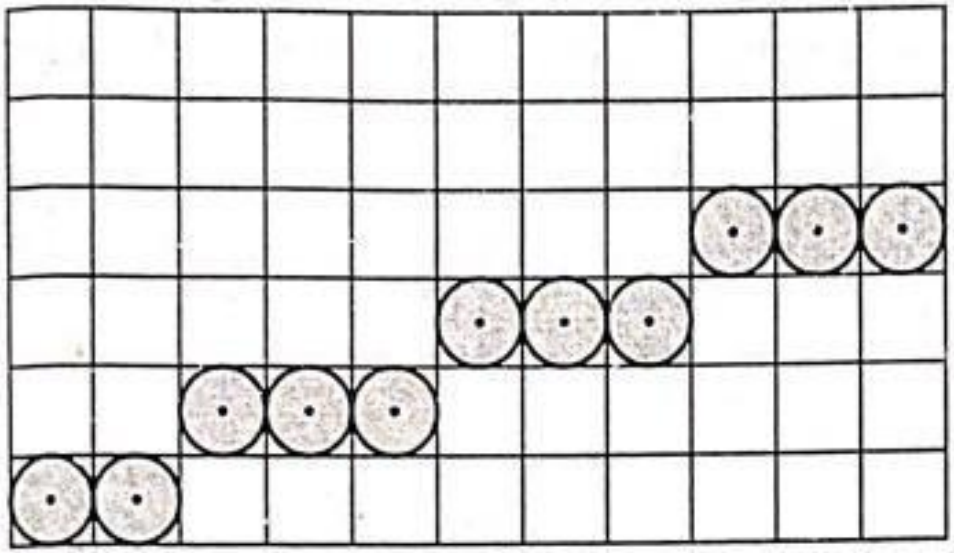
15. Stop

In step 7, we have checked for $y \leq y_1$ and not simply $y < y_1$. Hence step 9 becomes redundant.

Features of Scan Line Algorithm

- Used for filling the polygon.
- Solves the problem of hidden surfaces while generating display scan line.
- Used in orthogonal projection.
- Non recursive algorithm.
- Stack only a beginning position for each horizontal pixel scan, instead of stacking all unprocessed neighboring positions around the current position. Therefore, it is efficient algorithm.

Antialiasing of Lines



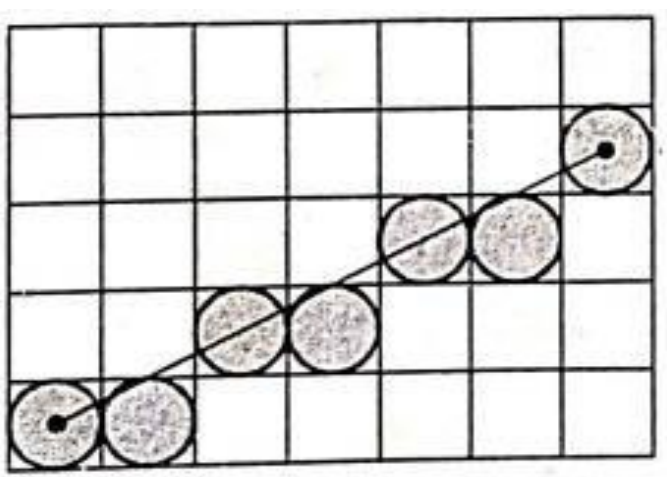
In low resolution screen line appears like a stair-step, this effect is known as **aliasing.**

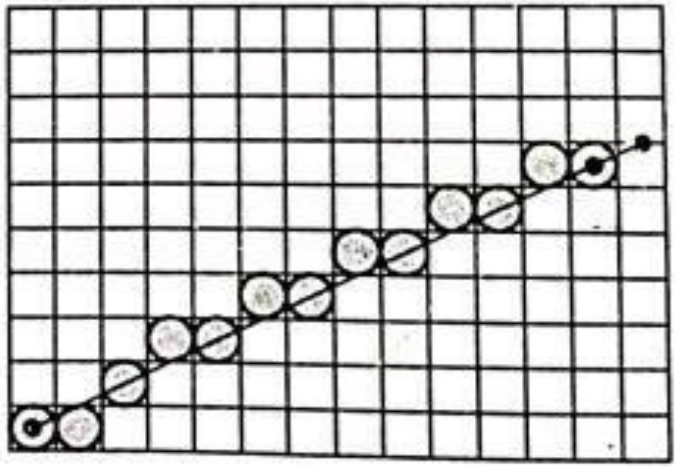
It is dominant for lines having slopes less than 20 or greater than 70 degree.

The process of adjusting intensities of the pixels along the line to minimize the effect of aliasing is called **antialiasing.**

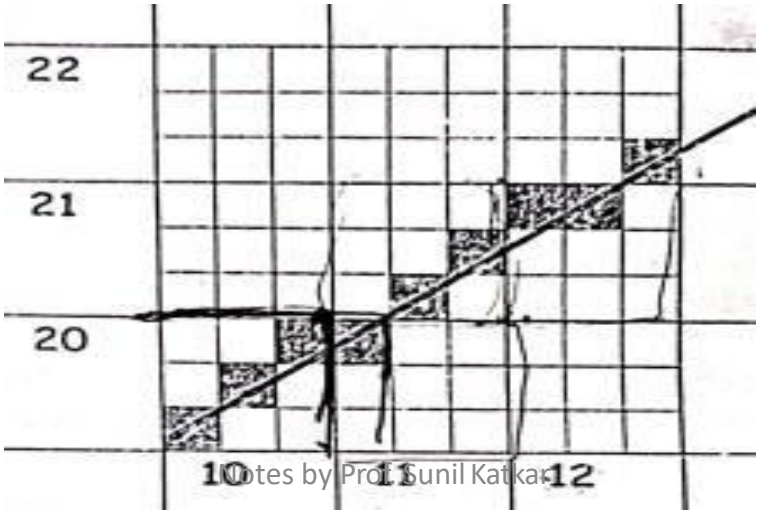
Methods of Antialiasing

- ✓ **Increasing resolution**



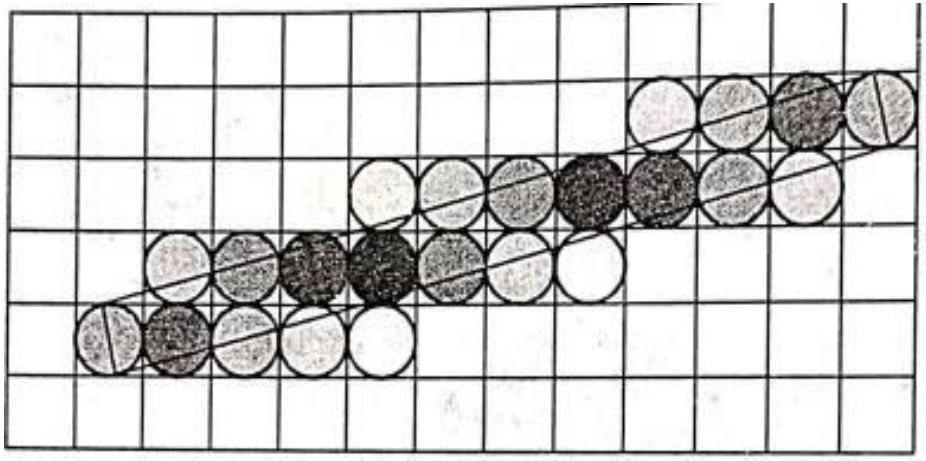


- ✓ **Super sampling straight line segments**



Methods of Antialiasing

- ✓ **Unweighted area sampling**



- ✓ **weighted area sampling**