Unit II
Polygons And Graphical Transformations

Agenda
Part A - Polygons

- Concept
- Polygon Types
- Inside test
- Polygon filling methods

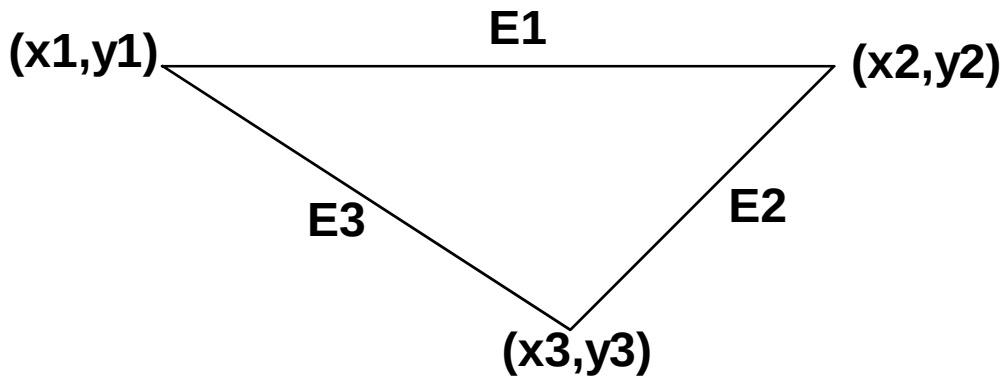
Polygons

A **polygon** is a many-sided **planar** figure composed of **vertices** and **edges**. A polygon is bounded (finite area) and closed (includes boundary).

Vertices are represented by points (x,y).

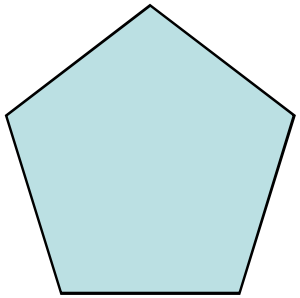
Edges are represented as line segments which connect two points, (x1,y1) and (x2,y2).

$$P = \{ (x_i , y_i) \} \quad i=1,n$$

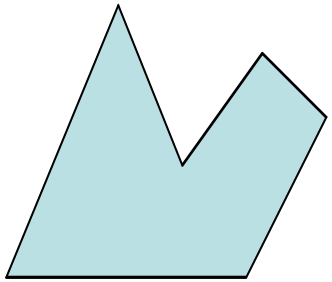


Different types of Polygons

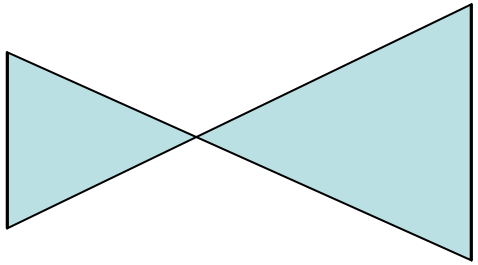
- Simple
- Complex
- Convex
- Concave
- Non-simple : self-intersecting



Convex



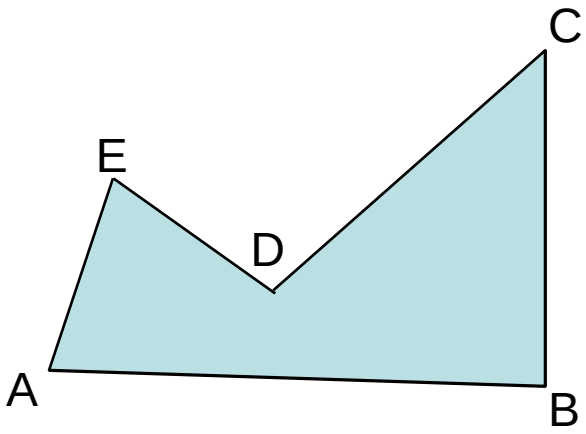
Concave

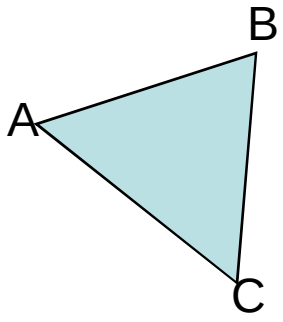


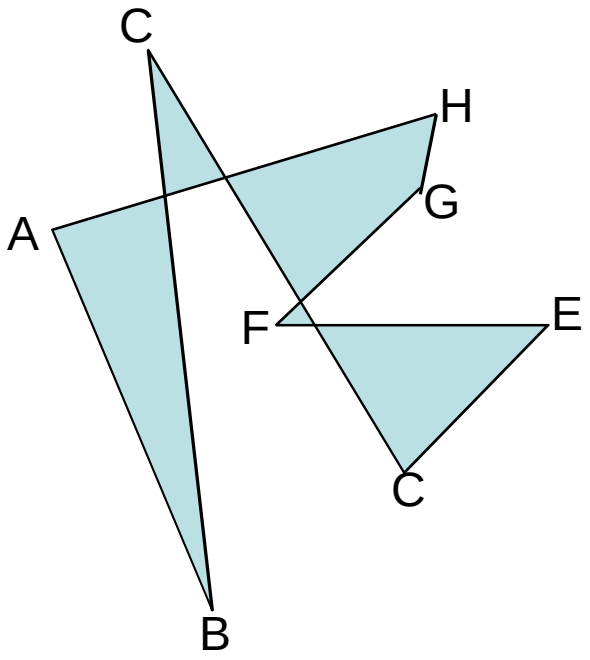
Self-intersecting

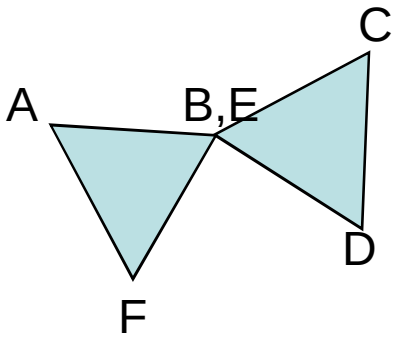
Polygons: Complex vs Simple

- A simple polygon – edges only intersect a vertices, no coincident vertices
- A complex polygon – edges intersect and/or coincident vertices







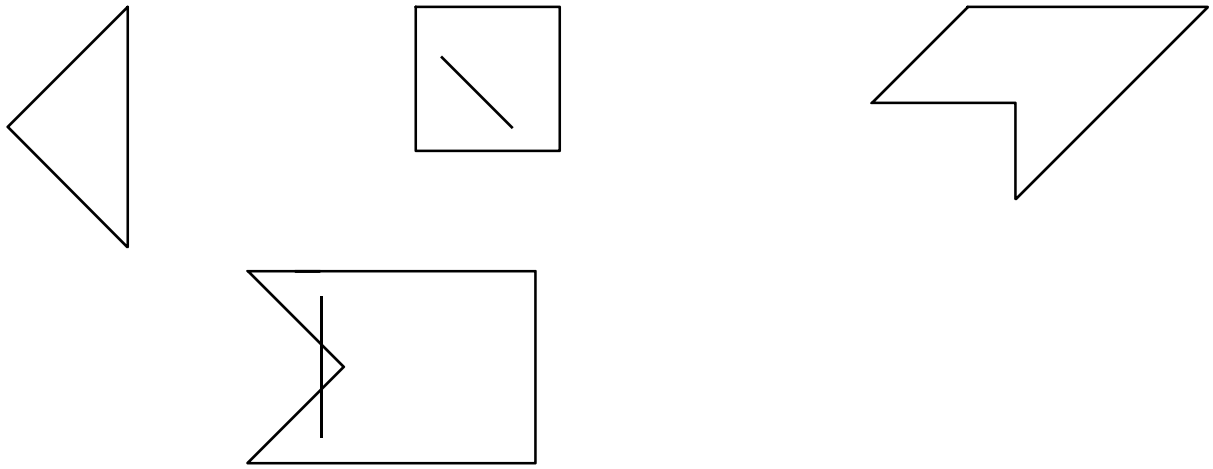


Simple Polygons: Convex and Concave

Convex Polygon - For any two points P_1, P_2 inside the polygon, all points on the line segment which connects P_1 and P_2 are inside the polygon.

- All points $P = uP_1 + (1-u)P_2$, u in [0,1] are inside the polygon provided that P_1 and P_2 are inside the polygon.

Concave Polygon - A polygon which is not convex.



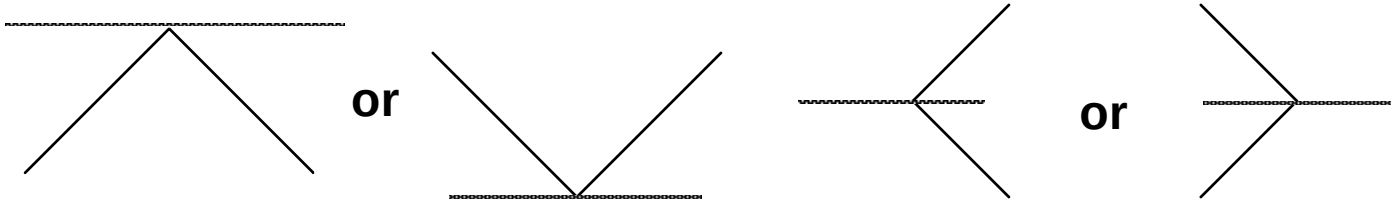
Inside vs Outside Polygon: Odd-Parity Test

-Is point P inside polygon?

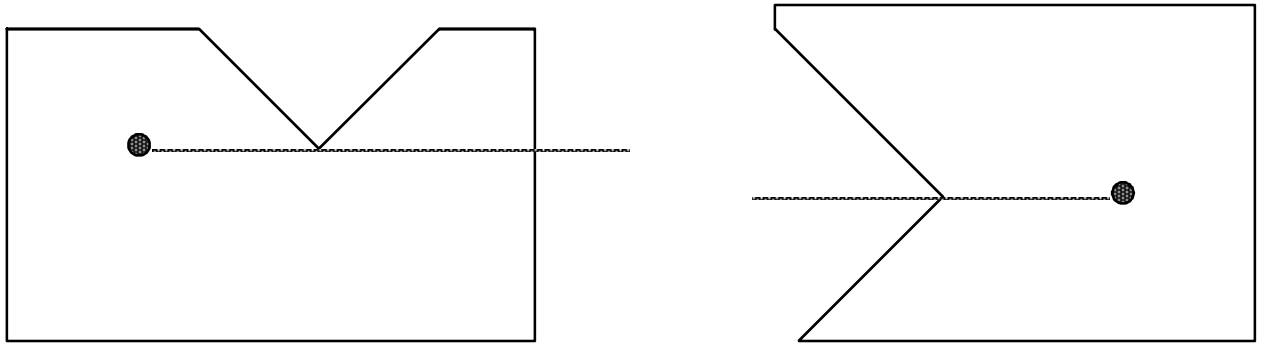
-Count # times a ray from P intersects edge of polygon

-odd count implies inside

-When crossing a vertex, if the vertex's two edges are on same side of line then count it twice, else count it once.

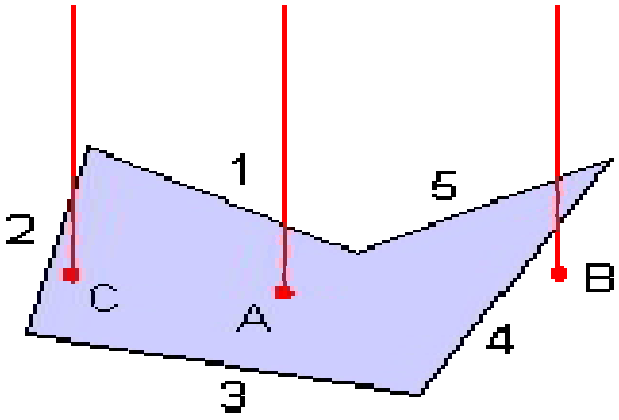


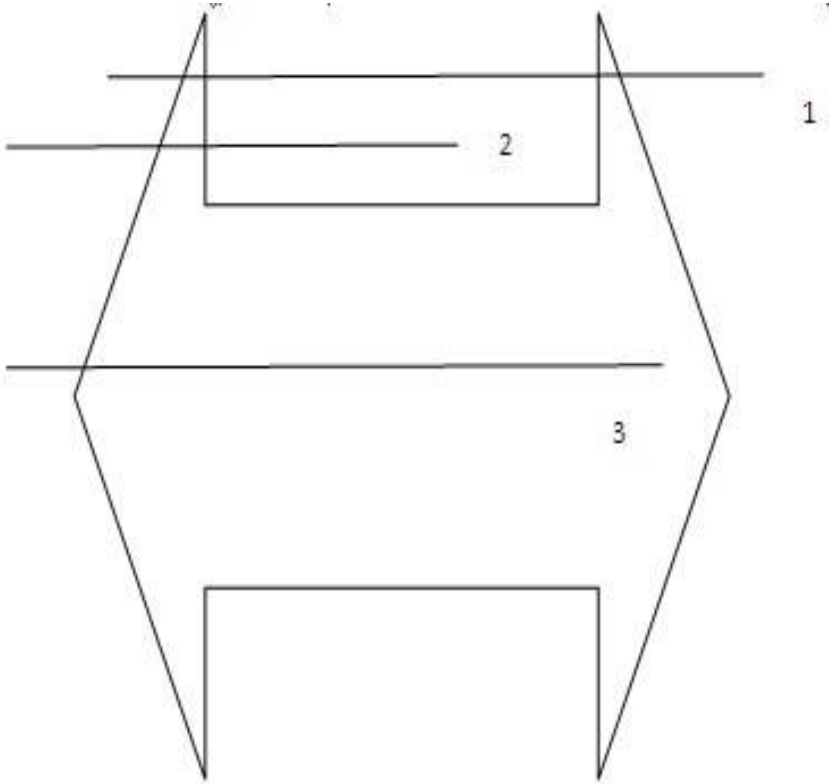
Count twice **Count once**



Inside Test

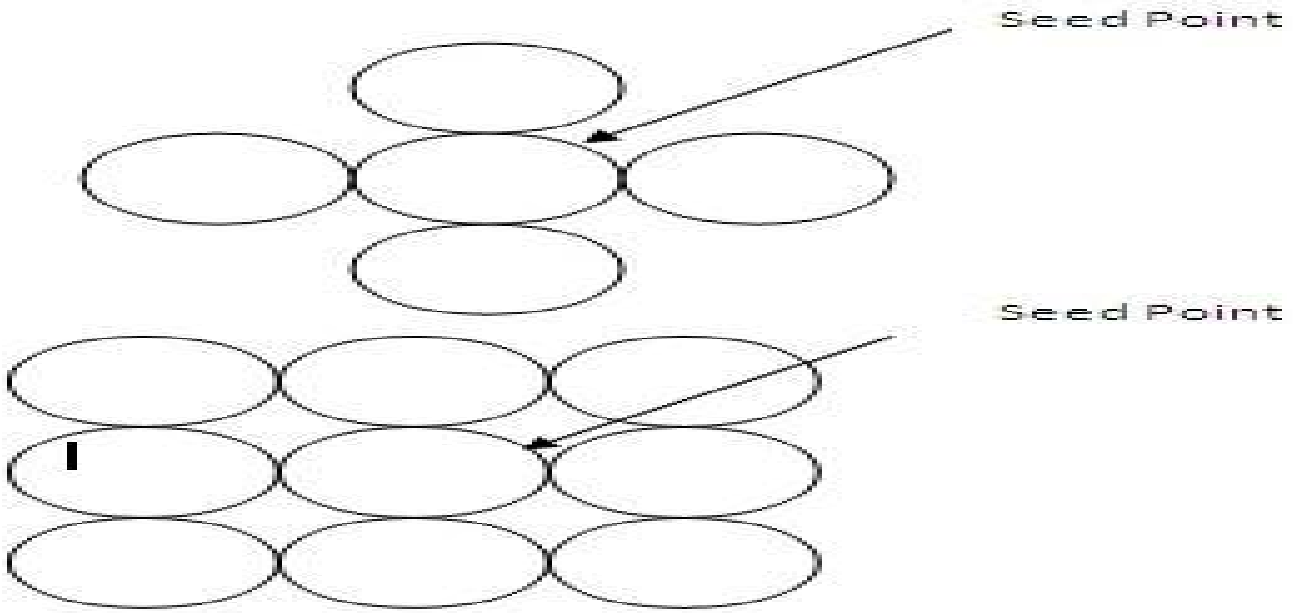
- A calculation that must be performed billions of times in every movie that uses computer graphics is to check if one object is hidden by another object in front of it. This calculation in turn is based on another calculation that checks whether or not a test point lies inside or outside the boundary of an object.
- **Even odd method** : This method is used to determine whether a point is inside the polygon or outside.
- - For this, simply construct a line segment from the point in question to point outside the polygon.
- Then draw a line segment from the point to outside and count no. of times line intersects the polygon.
- If it is even then the point is outside the polygon otherwise inside


1
2
5
C
A
B
4
3


1
2
3

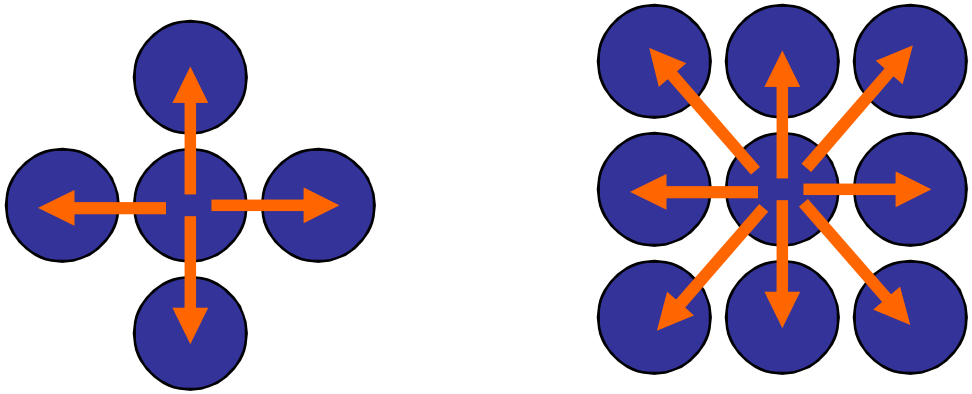
Polygon Filling Methods

- Seed Filling
- Scan Line Filling
- Seed Filling: In this method a particular seed point is picked and used as starting point to fill upwards and downwards pixels until boundary is reached.



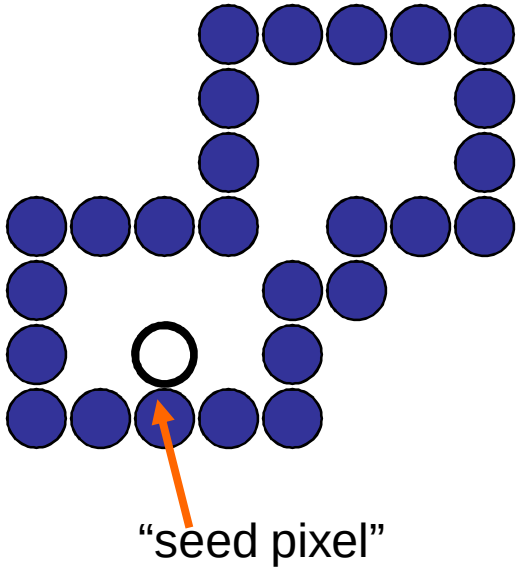
4 vs 8 connected

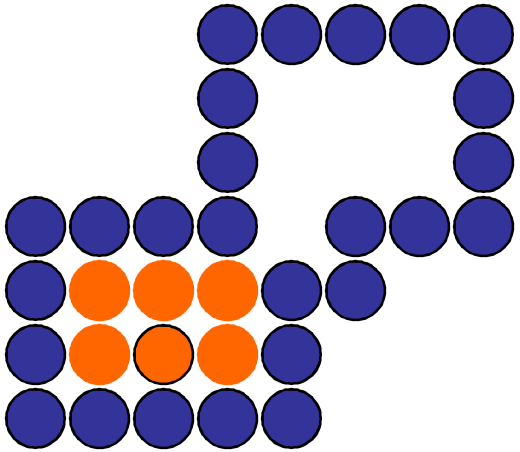
- Define: 4-connected versus 8-connected, its about the neighbors

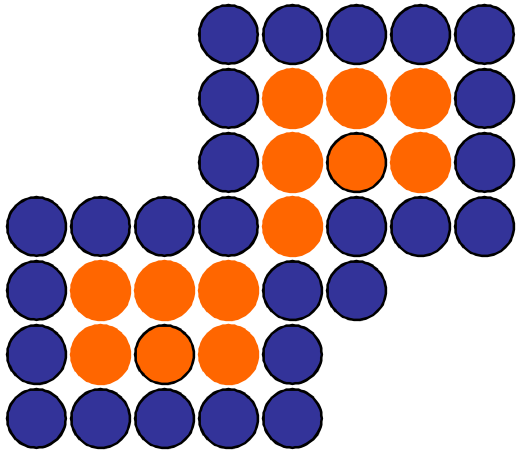


4 vs 8 connected

- Fill Result: 4-connected versus 8-connected



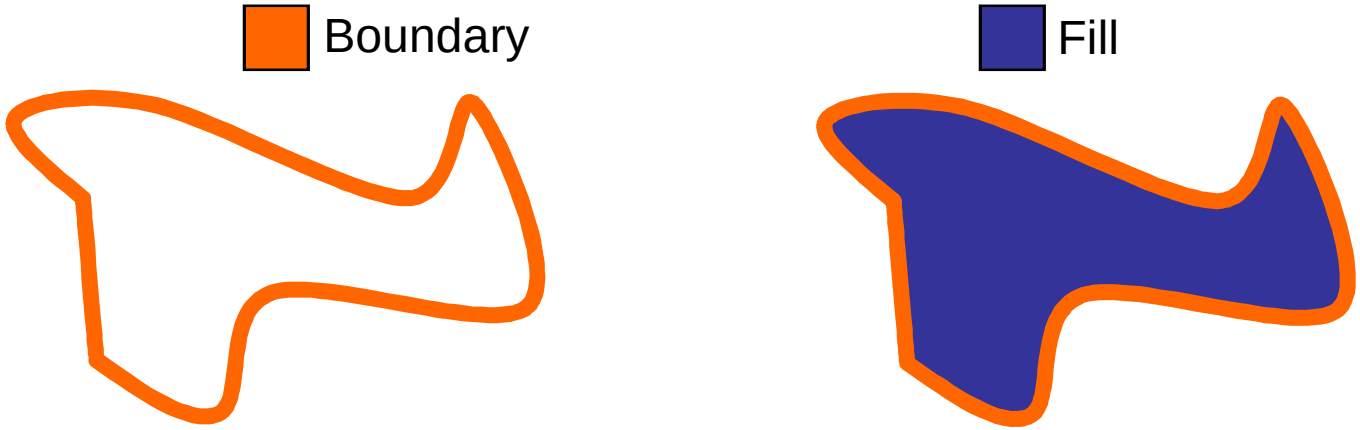




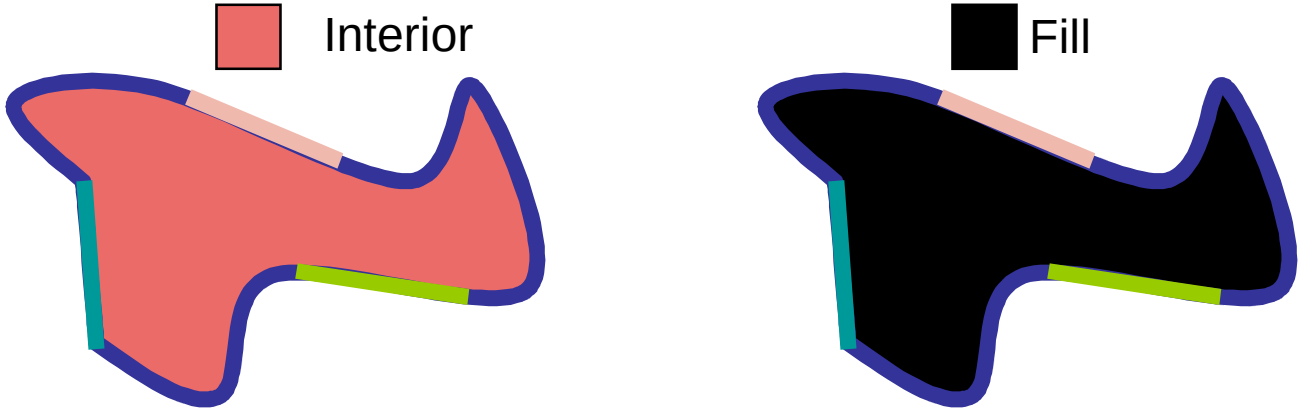
- Seed fill method is of two types: Boundary fill and Flood fill.
- Boundary fill algorithm: In Boundary filling a seed point is fixed, and then neighboring pixels are checked to match with the boundary color. Then, color filling is done until boundary is reached. A region may be 4 connected or 8 connected.

Filling Irregular Boundaries

- boundary fill: expand and fill region until you reach boundary color



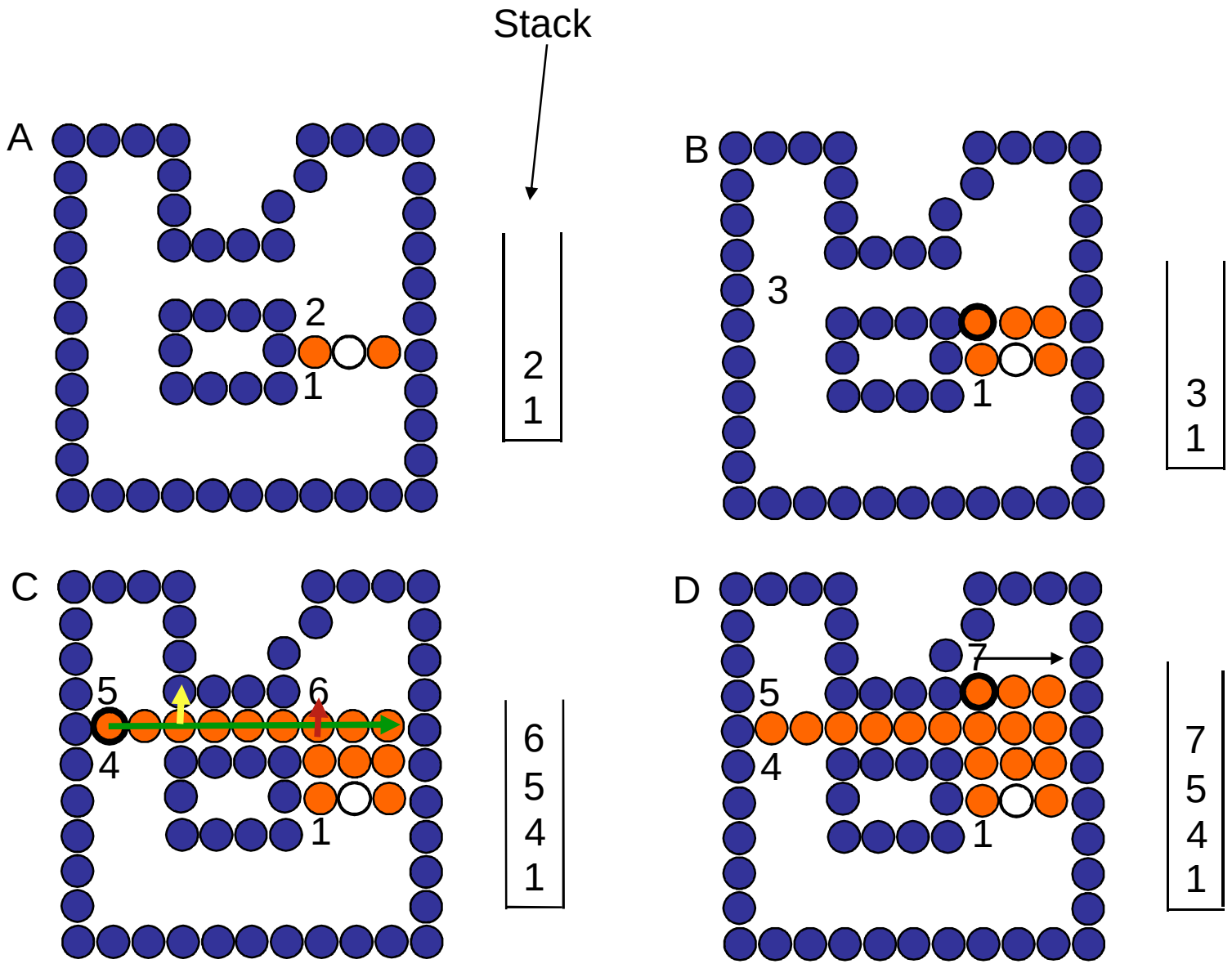
- flood fill: expand and fill region while you find interior color



Recursive Pixel Boundary Fill

```
void boundaryFill4(int x, int y){
  if(getPixel(x,y)!=borderColor &&
getPixel(x,y)!=fillColor ){
   setPixel(x,y);
   boundaryFill4(x+1,y);
   boundaryFill4(x-1,y);
   boundaryFill4(x,y+1);
   boundaryFill4(x,y-1);
   }
}
```

Recursive Scan-line (boundary) fill



Flood Fill

- Fill can be done recursively if we know a seed point (x,y) located inside (WHITE)
- Scan convert edges into buffer in edge/inside color (BLACK)

```
flood_fill(int x, int y) {
    if(read_pixel(x,y)= = WHITE) {
       write_pixel(x,y,BLACK);
       flood_fill(x-1, y);
       flood_fill(x+1, y);
       flood_fill(x, y+1);
       flood_fill(x, y-1);
}   }
```