Parul® University

Computer Graphics 05101301

Prof. Dampy Singh, Assistant Professor
Parul Institute of Computer Application

Attributes of Output Primitives

- In computer graphics, output primitives are the basic geometric elements used to create images or graphical representations on the screen.
- Attributes related to these primitives, as well as various algorithms to fill areas or handle special effects, are crucial in defining the visual appearance of graphics.
- Output primitives refer to the basic objects used in graphics rendering, such as points, lines, and polygons. Each of these can have associated attributes that define their appearance on the screen.

Attributes of Output Primitives cont.

Line Style: Defines how lines are drawn. It can include various types:

- Solid lines
- Dashed lines
- Dotted lines
- Dash-dot lines These line styles can be used to convey different types of information or simply for aesthetic purposes.

Colour and Intensity:

- **Colour** refers to the color used to render primitives, typically represented using RGB (Red, Green, Blue) values or a similar color model.
- **Intensity** refers to the brightness or lightness of a color. For example, a pure red color can have high intensity, while a dimmer version of the same red can have lower intensity.

Parul®
University

DIGITAL LEARNING CONTENT

Polygon Filling Algorithm

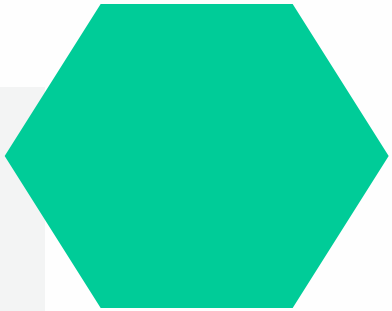
1. Scan line Fill Algorithm
2. Seed Fill Algorithm
   - -Boundary Fill Algorithm
   - -Flood Fill Algorithm

Polygon Filling Algorithm

Types of filling

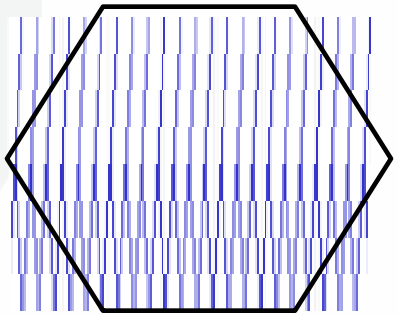
Solid-fill

All the pixels inside the polygon's boundary are illuminated.



Pattern-fill

The polygon is filled with an arbitrary predefined pattern.



Polygon Representation

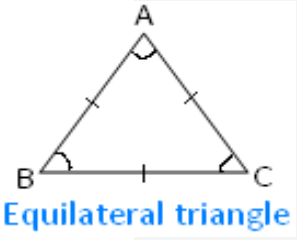
The polygon can be represented by listing its n vertices in an ordered list.

$$P = \{(x_1, y_1), (x_2, y_2), \cdots\cdots, (x_n, y_n)\}.$$

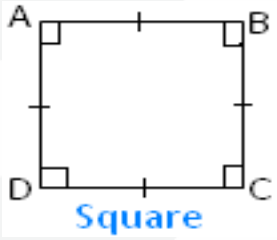
The polygon can be displayed by drawing a line between (x_1, y_1), and (x_2, y_2), then a line between (x_2, y_2), and (x_3, y_3), and so on until the end vertex. In order to close up the polygon, a line between (x_n, y_n), and (x_1, y_1) must be drawn.

Types of Polygons: Regular and Irregular

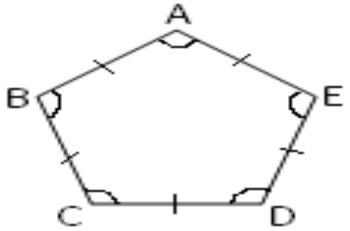
Regular Polygon :A polygon which has all its sides of equal length and all its angles of equal measures is called a **regular polygon**.



Equilateral triangle

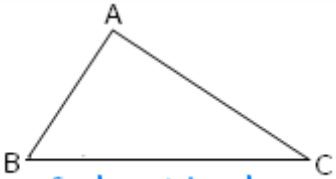


Square



Regular pentagon

Irregular Polygon :A polygon which has all its sides of unequal length and all its angles of unequal measures is called an irregular polygon.



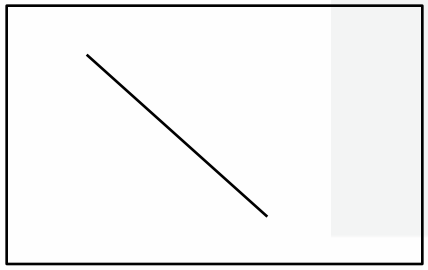
Scalene triangle

Types of Polygons: Regular and Irregular

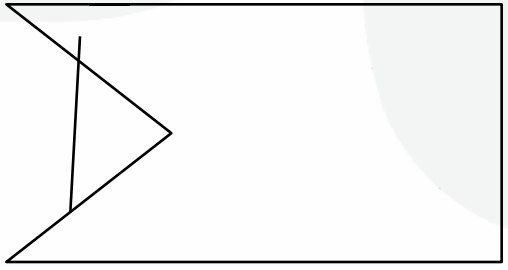
Convex Polygon - For any two points P_1, P_2 inside the polygon, all points on the line segment which connects P_1 and P_2 are inside the polygon.

Concave Polygon

- Aline segment connecting any two points may or may not lie inside the polygon.ie A polygon which is not convex.



Convex Polygon



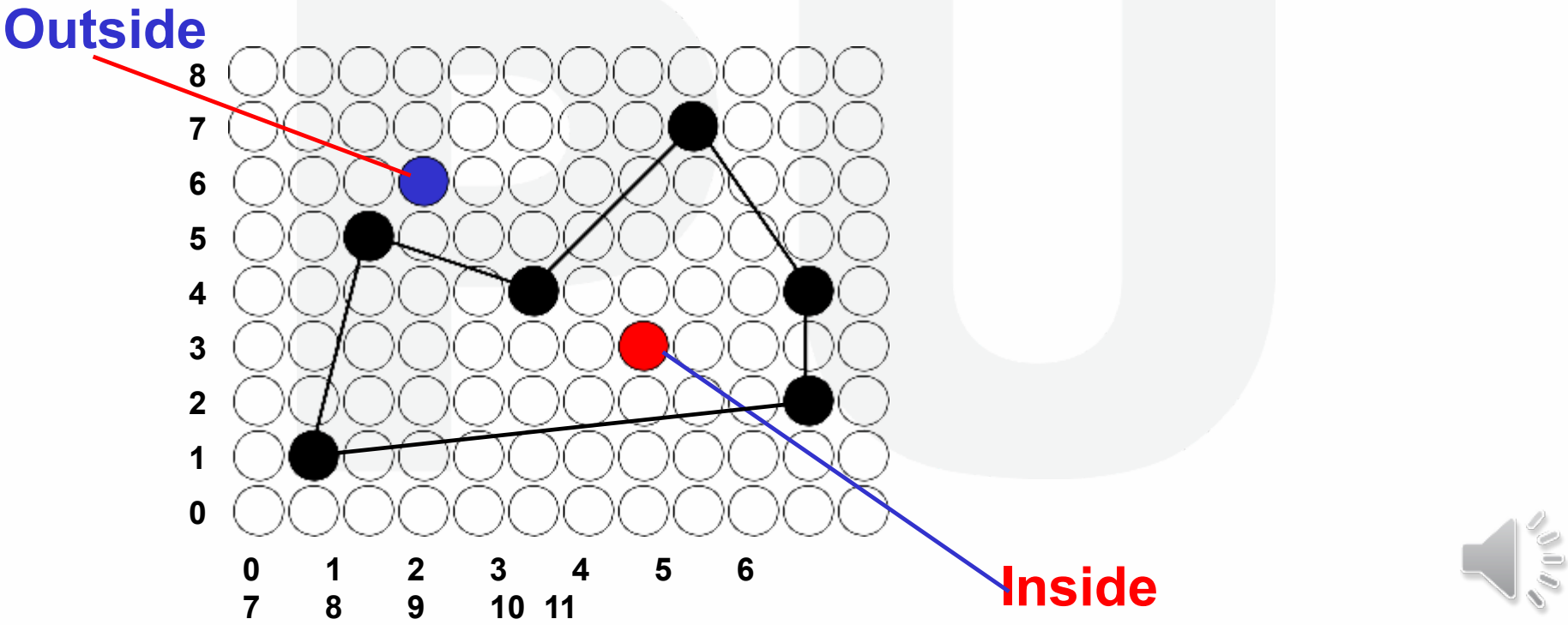
Concave Polygon

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

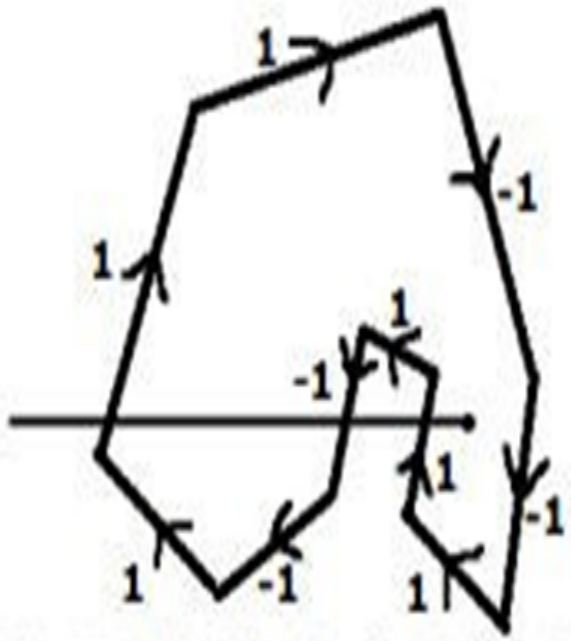
The point is considered to be **interior** if the number of intersections between the line and the polygon edges is **odd**. Otherwise, The point is exterior point.



WINDING NUMBER METHOD FOR INSIDE OUT SIDE

If the winding number is **nonzero** then *P* is defined to be an **interior** point **Else** *P* is taken to be an **exterior** point**.**

- **Sum of edge is nonzero means Inside the polygon**
- **Sum of edge is zero means outside the polygon**

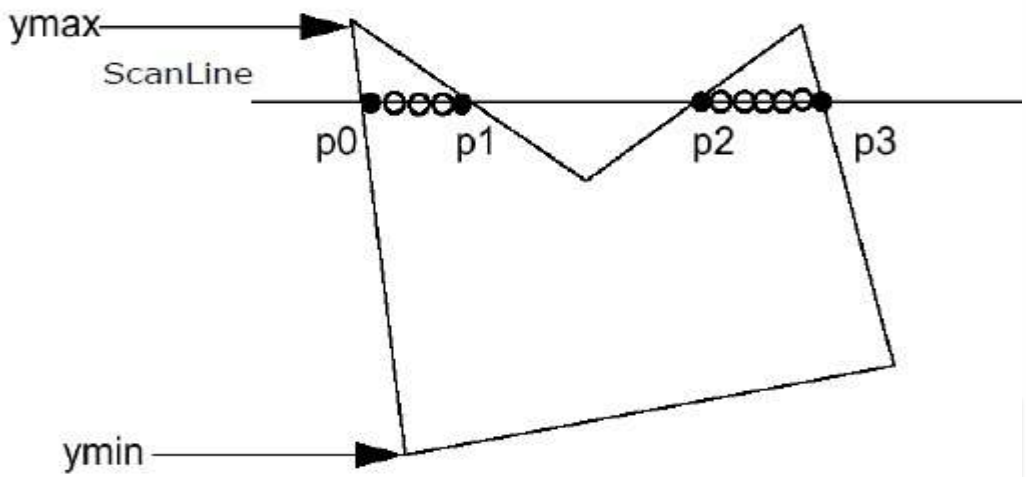


Winding Number Method

The Scan-Line Polygon Fill Algorithm

This algorithm works by intersecting scanline with polygon edges and fills the polygon between pairs of intersections. The following steps depict how this algorithm works.

Step 1 – Find out the Ymin and Ymax from the given polygon.



Scan Line Drawing Algorithm

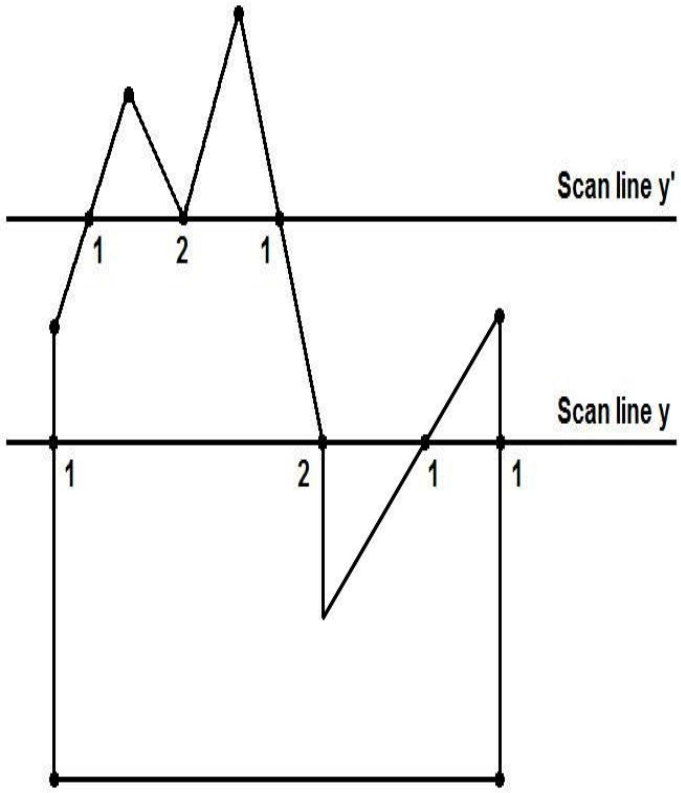
- **Step 2** – ScanLine intersects with each edge of the polygon from Ymin to Ymax. Name each intersection point of the polygon. As per the figure shown above, they are named as p0, p1, p2, p3.

- **Step 3** – Sort the intersection point in the increasing order of X coordinate i.e. (p0, p1), (p1, p2), and (p2, p3).

- **Step 4** – Fill all those pair of coordinates that are inside polygons and ignore the alternate pairs.

The Scan-Line Polygon Fill Algorithm

Dealing with Vertices



If the edges having a common intersection point lies on the same side of the scan line then the intersection point is considered two.

But if the edges having a common intersection point lies on the different side of the scan line then the intersection point is considered one.

Here please count number of intersection 1.

Problem:
Calulating the Intersection Points is a slow process.

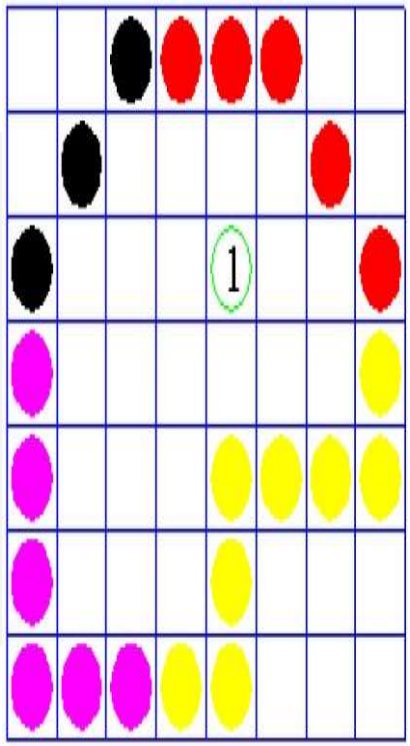
The Scan-Line Polygon Fill Algorithm

Step-04:

Keep repeating Step-03 until the end point is reached or number of iterations equals to (ΔX-1) times.

Flood Fill Algorithm

- Sometimes we come across an object where we want to fill the area and its boundary with different colors.
- In other words, it replaces the interior color of the object with the fill color. When no more pixels of the original interior color exist, the algorithm is completed.
- This algorithm relies on the Four-connect or Eight-connect method of filling in the pixels.
- But instead of looking for the boundary color, it is looking for all adjacent pixels that are a part of the interior.

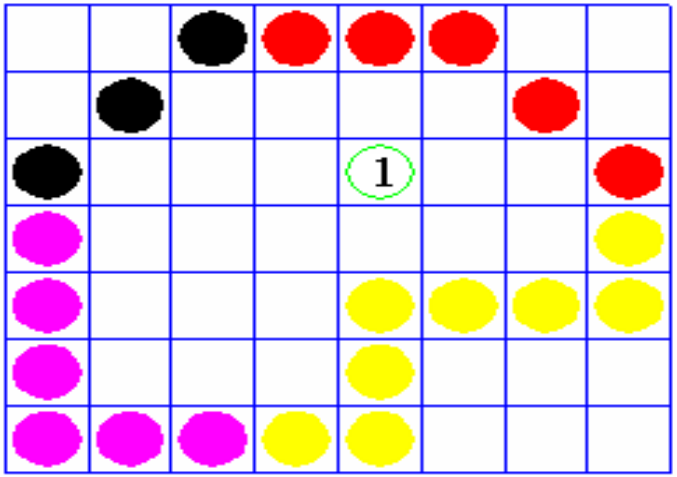


Flood Fill Algorithm

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**.



Flood Fill Algorithm

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.

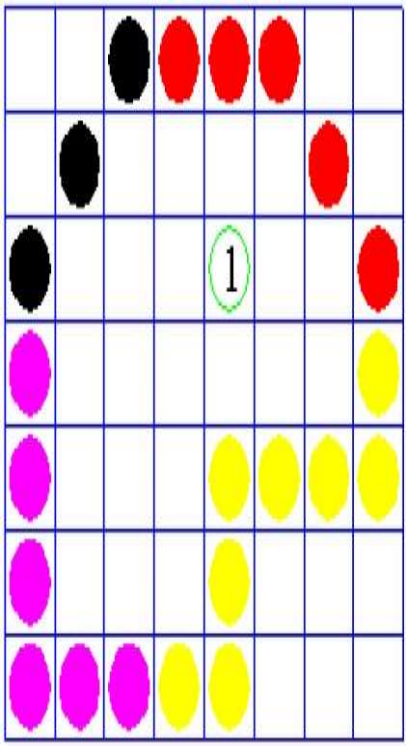
If the area has **more than one** interior color, we can first **reassign pixel values** so that all interior pixels have the same color.

Using either **4-connected** or **8-connected** approach, we then step through pixel positions until all interior pixels have been repainted.

In this algorithm we match color of a pixel with background color of the object. We stop color fill until we does not find a pixel where color is not same as background color.

Flood Fill Algorithm

```
Procedure floodfill (x, y,fill_ color, old_color: integer)
    If (getpixel (x, y)=old_color)
   {
    setpixel (x, y, fill_color);
    fill (x+1, y, fill_color, old_color);
     fill (x-1, y, fill_color, old_color);
    fill (x, y+1, fill_color, old_color);
    fill (x, y-1, fill_color, old_color);
     }
}
```



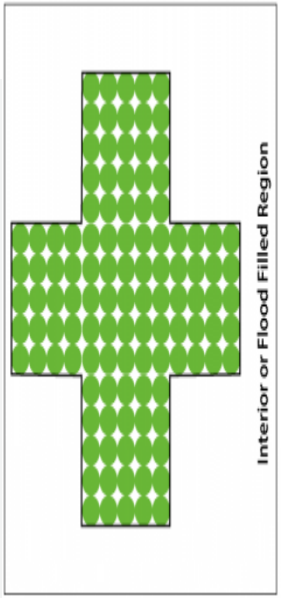
Flood Fill Algorithm

Advantages of Flood-fill algorithm

It provides an easy way to fill color in graphics.

The Flood-fill algorithm colors the whole area through interconnected pixels by a single color.

The algorithm fills the same color inside the boundary.



Disadvantages of Flood-fill Algorithm

It is a more time-consuming algorithm.

Sometimes it does not work on large polygons.

Boundary Fill Algorithm

- Another approach to area filling is to 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**.

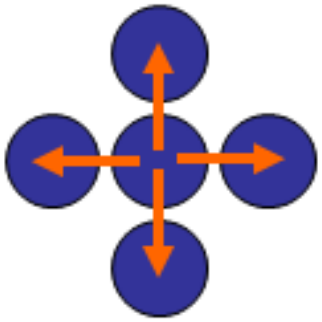
Boundary Fill Algorithm

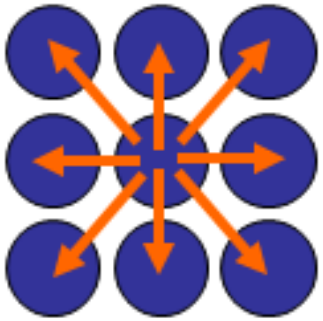
The following steps illustrate the **recursive** boundary-fill algorithm:

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**) in the stack for processing. 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**.

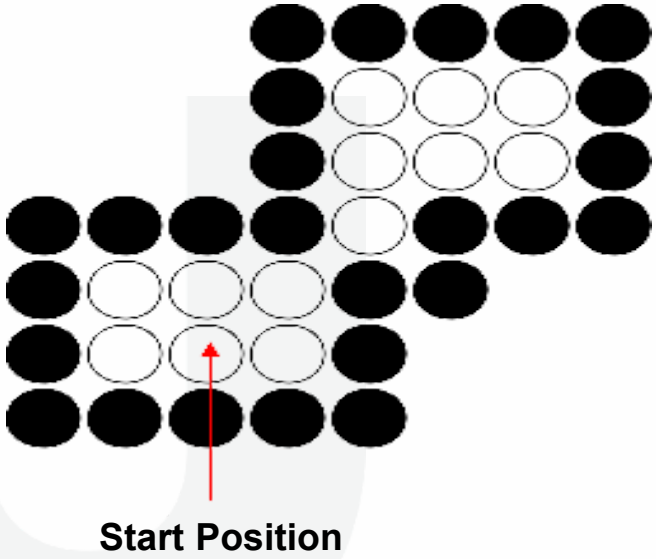
Boundary Fill Algorithm

The order of pixels that should be added to stack using **4-connected** is above, below, left, and right. For **8-connected** is above, below, left, right, above- left, above-right, below-left, and below-right.

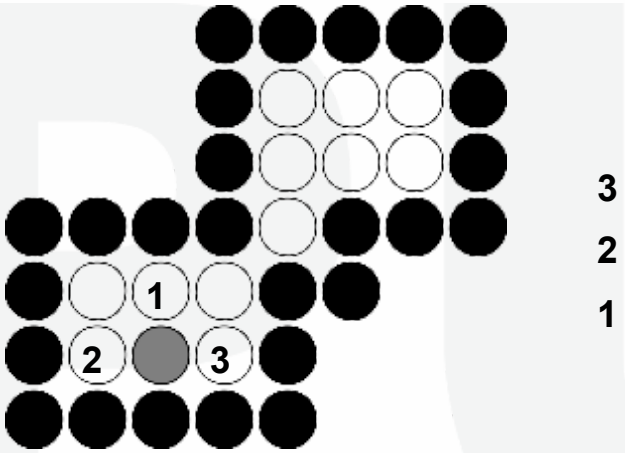




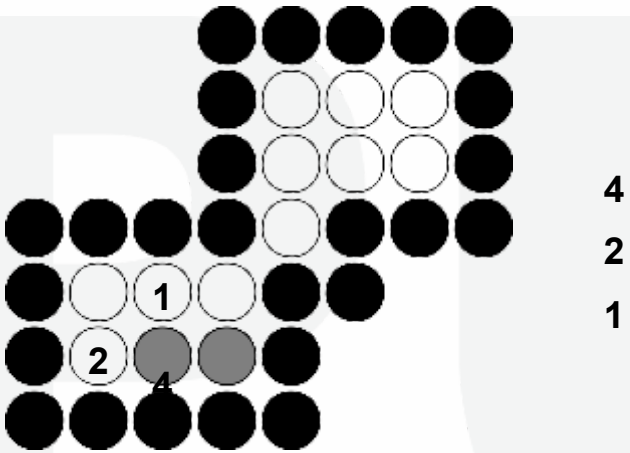
Boundary Fill Algorithm : 4-connected (Example)



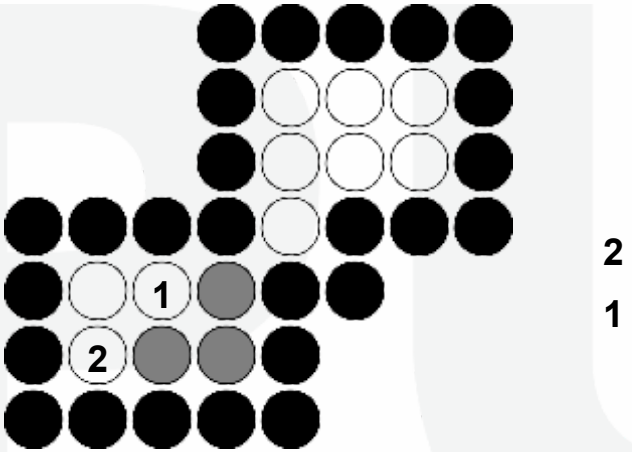
Boundary Fill Algorithm : 4-connected (Example)



Boundary Fill Algorithm : 4-connected (Example)



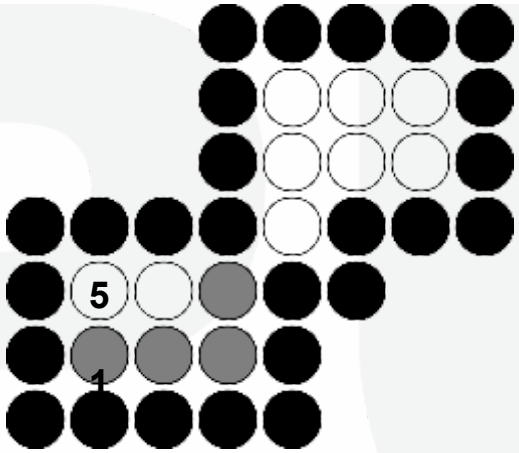
Boundary Fill Algorithm : 4-connected (Example)



2

1

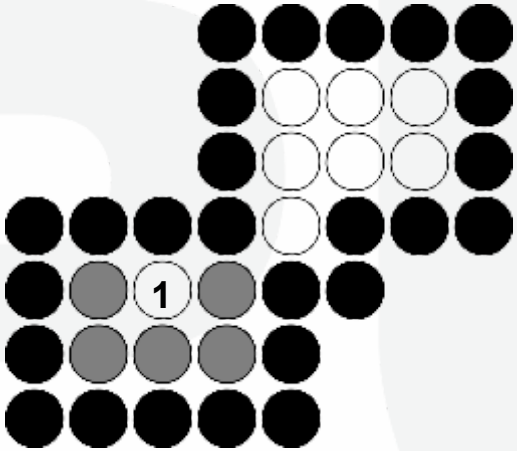
Boundary Fill Algorithm : 4-connected (Example)



5

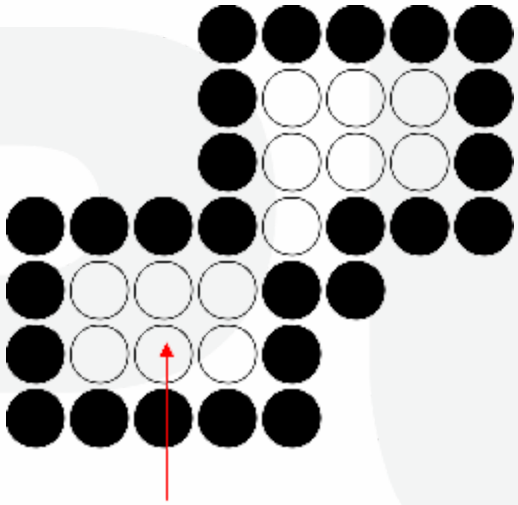
1

Boundary Fill Algorithm : 4-connected (Example)

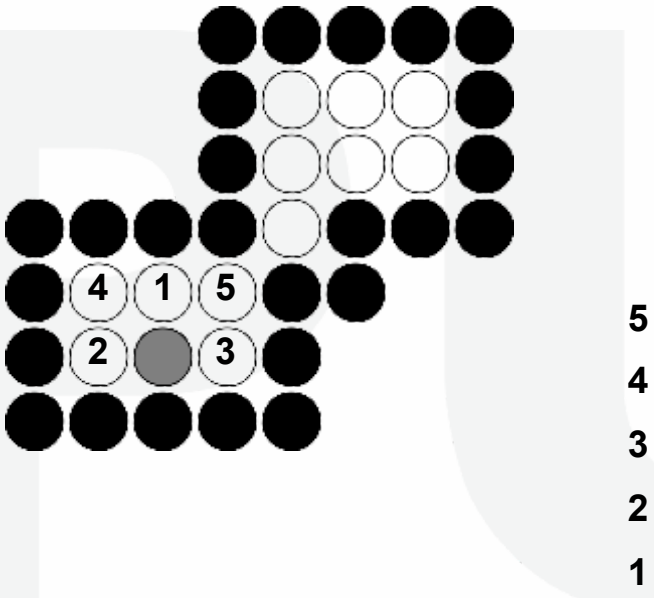


1

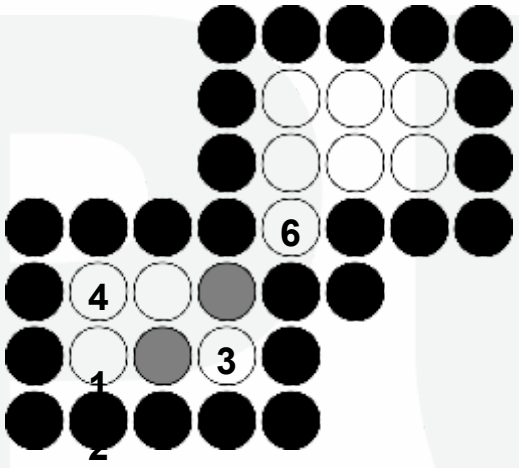
Boundary Fill Algorithm(8-connected Example)



Boundary Fill Algorithm(8-connected Example)

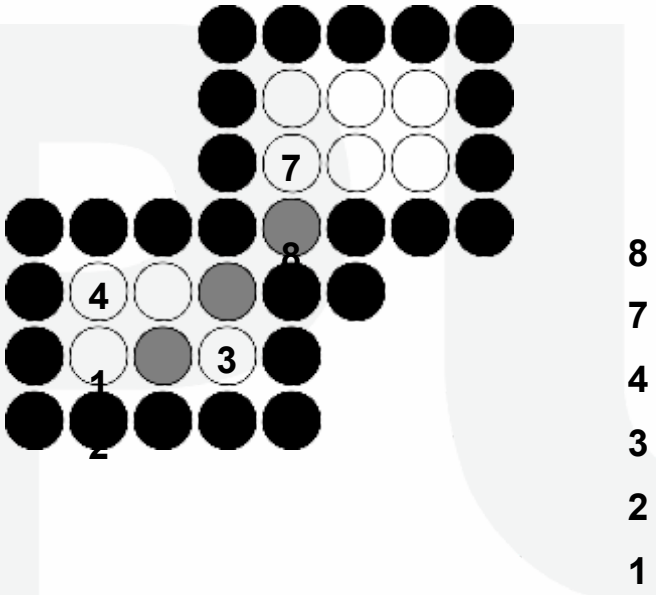


Boundary Fill Algorithm(8-connected Example)

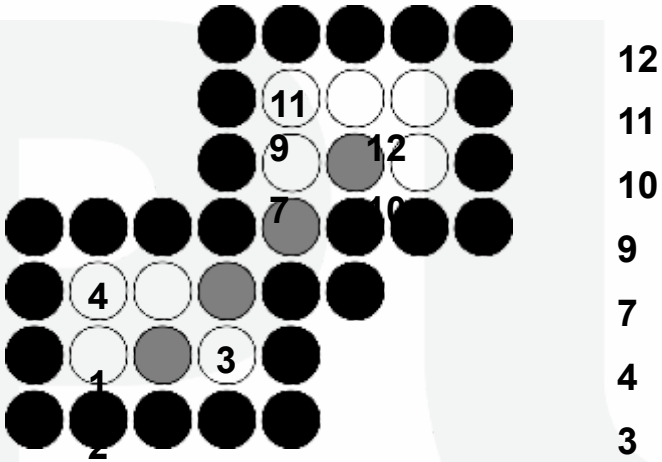


6
4
3
2
1

Boundary Fill Algorithm(8-connected Example)

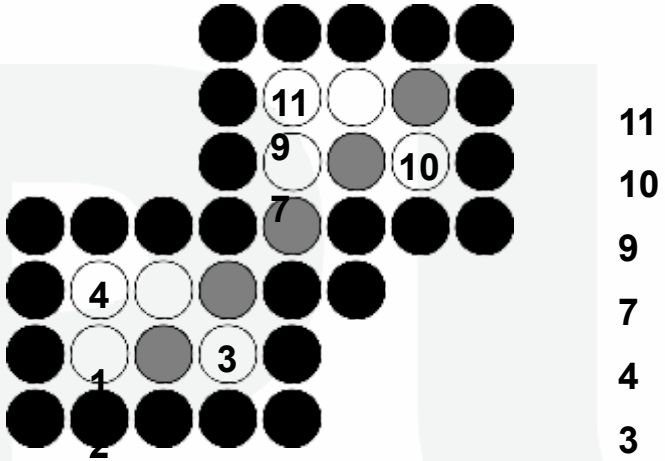


Boundary Fill Algorithm(8-connected Example)



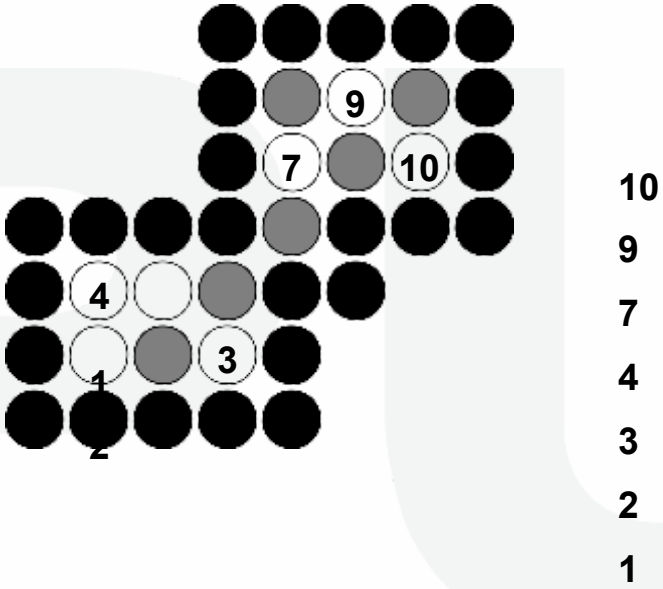
12
11
10
9
7
4
3
2
1

Boundary Fill Algorithm(8-connected Example)

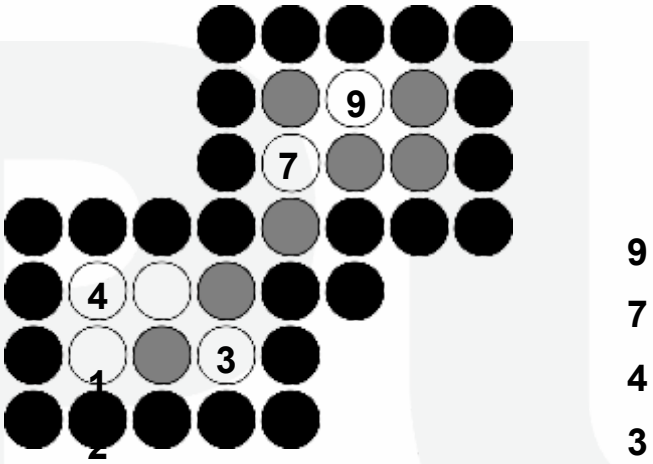


11
10
9
7
4
3
2
1

Boundary Fill Algorithm(8-connected Example)

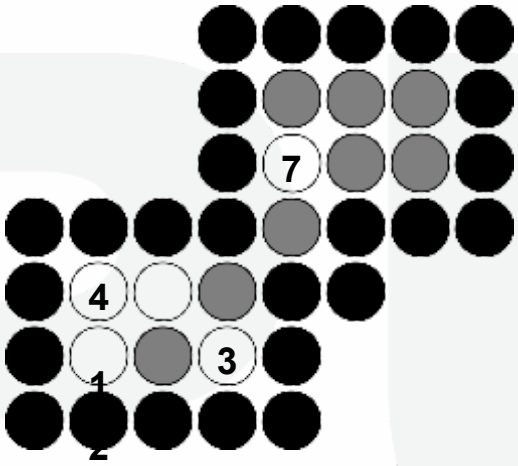


Boundary Fill Algorithm(8-connected Example)



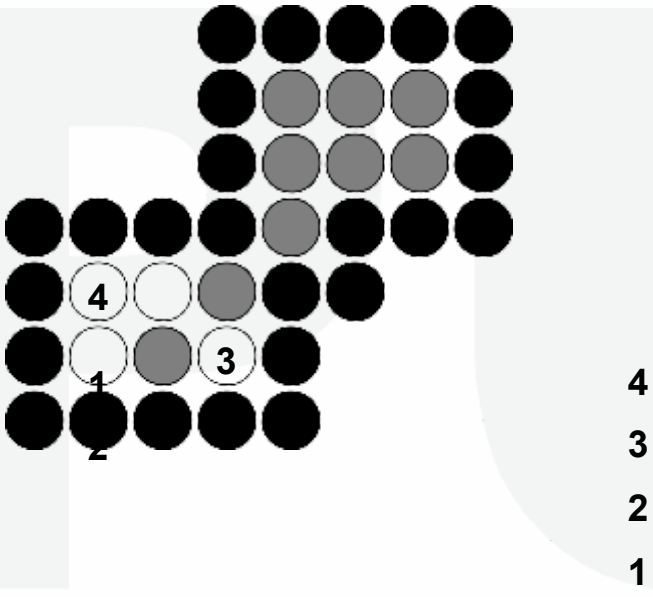
9
7
4
3
2
1

Boundary Fill Algorithm(8-connected Example)

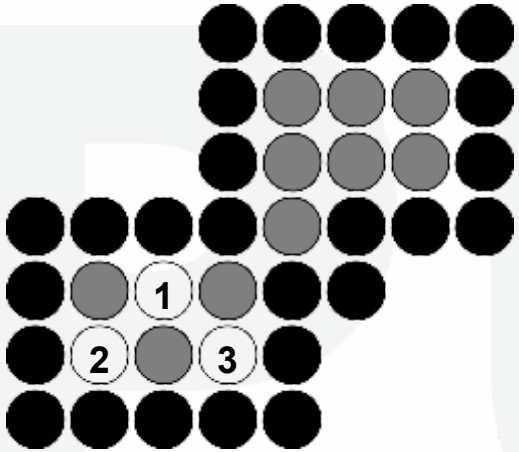


7
4
3
2
1

Boundary Fill Algorithm(8-connected Example)

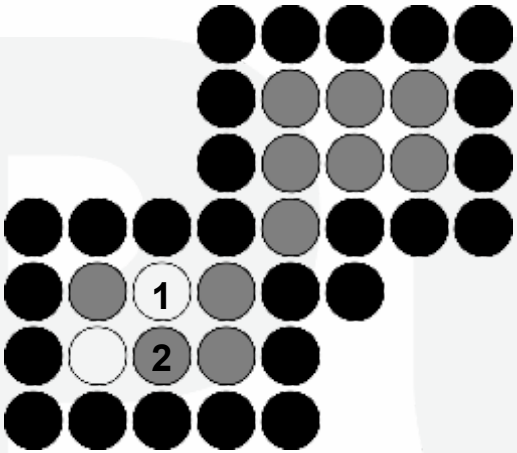


Boundary Fill Algorithm(8-connected Example)



3
2
1

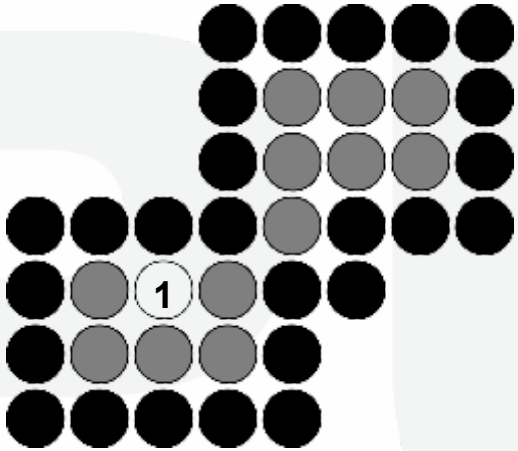
Boundary Fill Algorithm(8-connected Example)



2

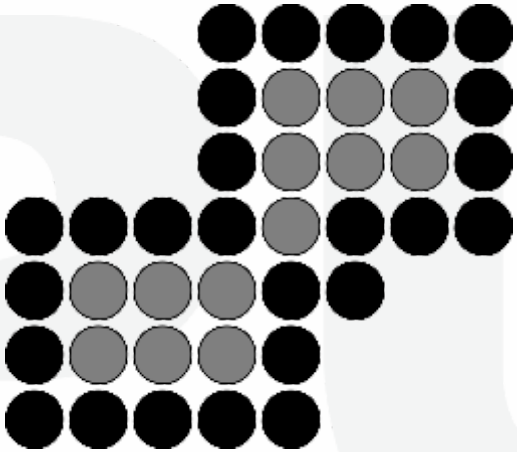
1

Boundary Fill Algorithm(8-connected Example)



1

Boundary Fill Algorithm(8-connected Example)



Boundary Fill Algorithm

In boundary fill algorithm we start from a seed pixel and start fill the color until boundary color does not encountered. In this algorithm if color of seed pixel is already fill color then we have to make this pixel as background color and then start fill color.

Functions used in this algorithm

Getpixel( )- This function is used to fetch color of a pixel on the screen.

Putpixel( ) – To give color on a particular pixel.

DIGITAL LEARNING CONTENT

Parul® University

www.paruluniversity.ac.in