Scan line polygon filling Boundary filling Flood filling

Department of Computer Science and Engineering
Mar Baselios College of Engineering & Technology, Nalanchira

Module - 2

Filled Area Primitives- Scan line polygon filling, Boundary filling and flood filling. Two dimensional transformations-Translation, Rotation, Scaling, Reflection and Shearing, Composite transformations, Matrix representations and homogeneous coordinates. Basic 3D transformations.

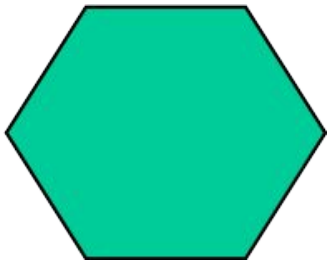
Course Outcomes

CO1	Describe the working principles of graphics devices.(Cognitive Knowl- edge level: Understand)
CO2	Illustrate line drawing, circle drawing and polygon filling algo- rithms.(Cognitive Knowledge level: Apply)
CO3	Demonstrate geometric representations, transformations on 2D & 3D objects, clipping algorithms and projection algorithms (Cognitive Knowledge level: Apply)
CO4	Summarize visible surface detection methods. (Cognitive Knowledge level: Understand)
CO5	Summarize the concepts of digital image representation, processing and demonstrate pixel relationships (Cognitive Knowledge level: Ap- ply)
CO6	Solve image enhancement and segmentation problems using

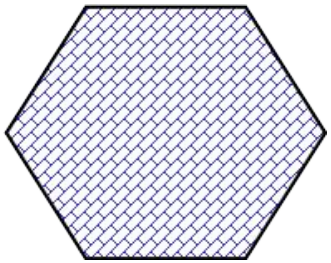
Polygon Filling

Types of polygon filling

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



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



Polygon Filling

- Approaches to area filling on raster systems ,
  1. Scan-line approach
     - Fill an area to determine the overlap intervals for scan lines that cross the area.
     - Used in general graphics packages to fill polygons, circles, ellipses, and other simple curves.
  2. Fill methods
     - Methods for area filling is to start from a given interior position and paintoutward from this point until the specified boundary conditions is encountered.

     -Used in more complex boundaries and in interactive painting.

Scan-Line Polygon Fill Algorithm

- For each scan line crossing a polygon, the area-fill algorithm
  - locates the intersection points of the scan line with the polygon
  - edges, sorts the intersection points from left to right and
  - corresponding frame-buffer positions between each intersection pair are set to the specified fill color.

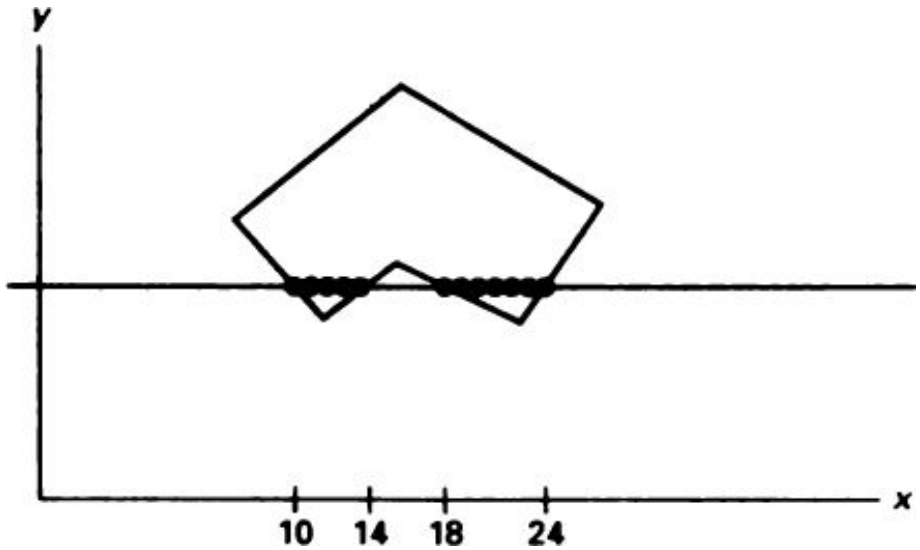


Fig. Interior pixels along a scan line passing through a polygon area

Scan-Line Polygon Fill Algorithm

Dealing with vertices

- If a scan line passing through a vertex intersects two polygon edges at that position, then add two points to the list of intersections for the scan line.

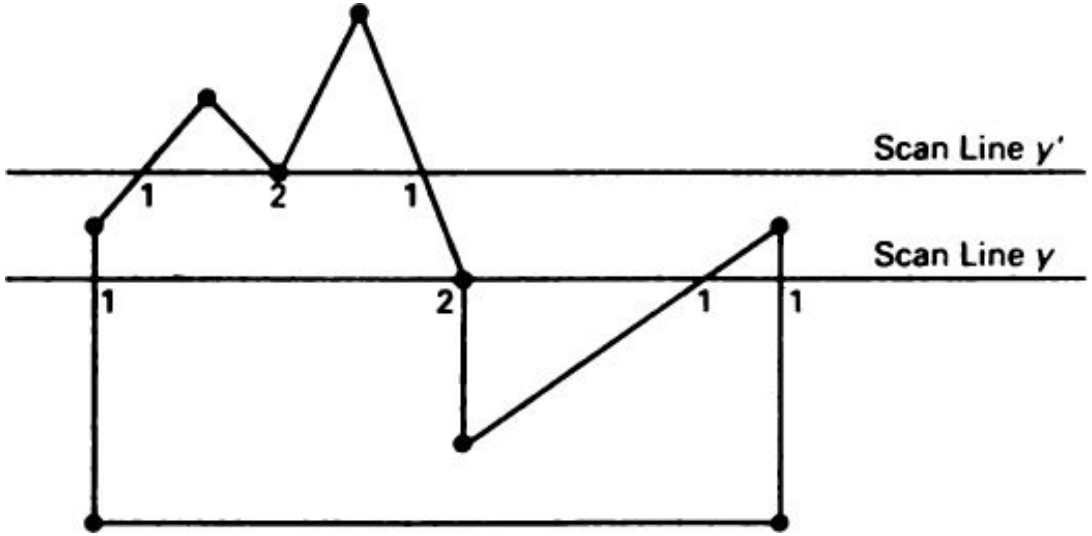


Fig. Intersection points along scan lines that intersect polygon vertices.

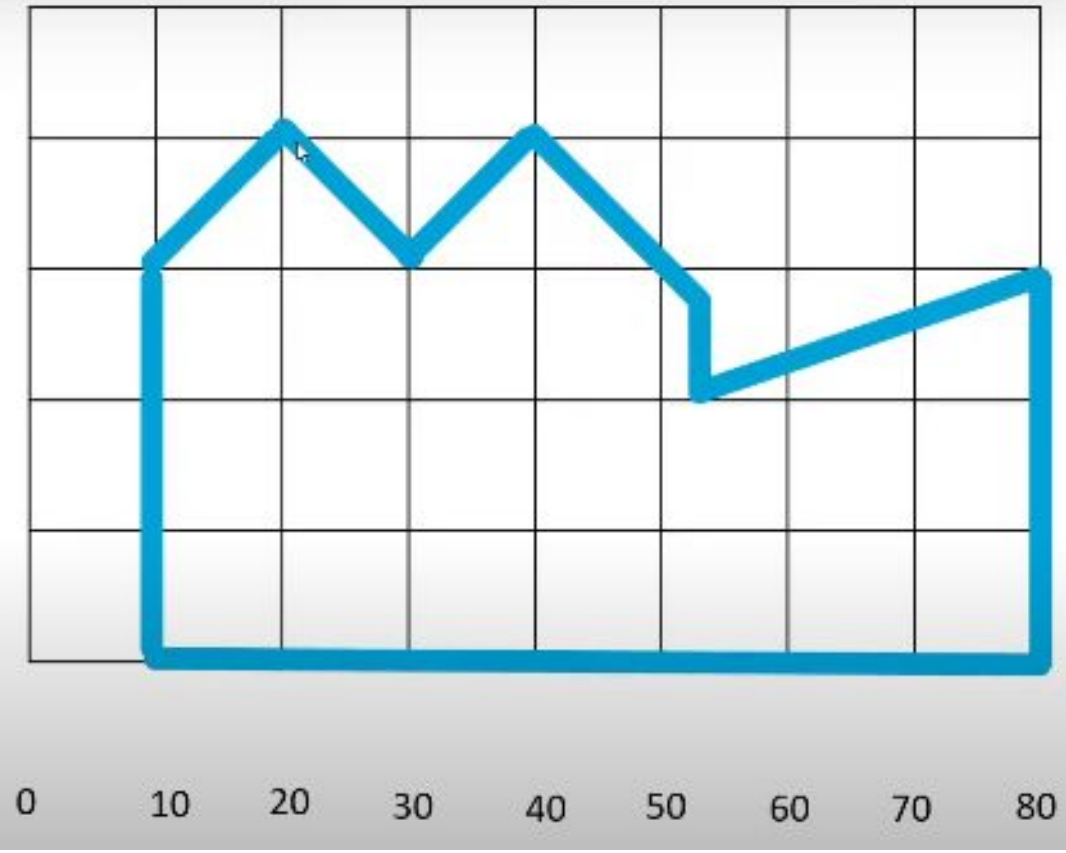
SCAN LINE A

SCAN LINE B

SCAN LINE C

SCAN LINE D

SCAN LINE E



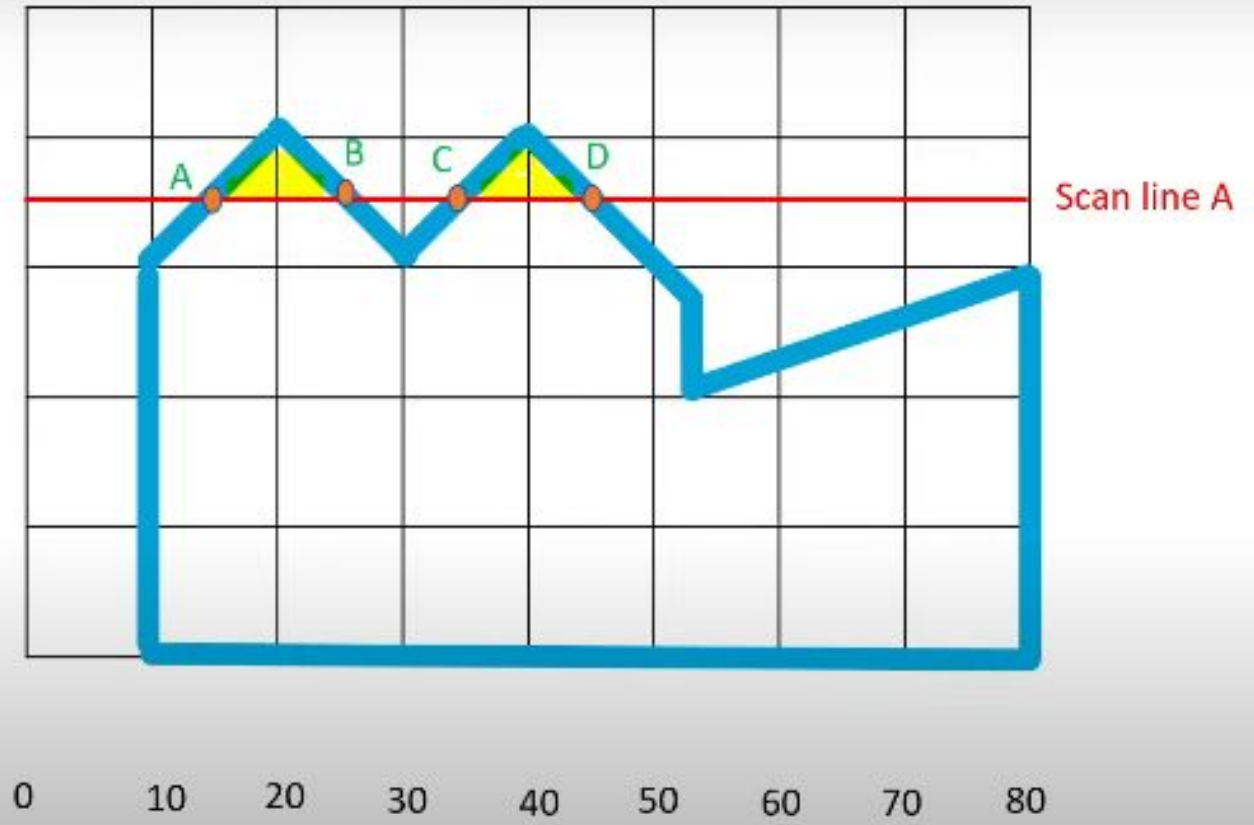
SCAN LINE A {A , B} {C, D}

SCAN LINE B

SCAN LINE C

SCAN LINE D

SCAN LINE E



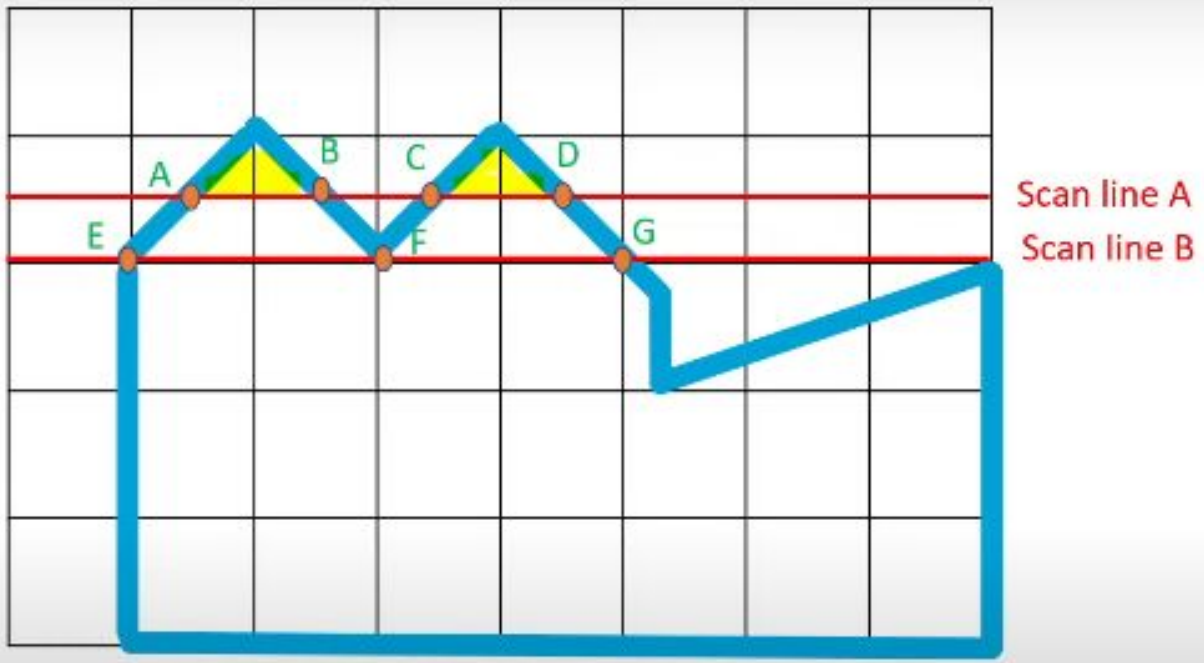
SCAN LINE A {A , B} {C , D}

SCAN LINE B {E , F} {F, G}

SCAN LINE C

SCAN LINE D

SCAN LINE E



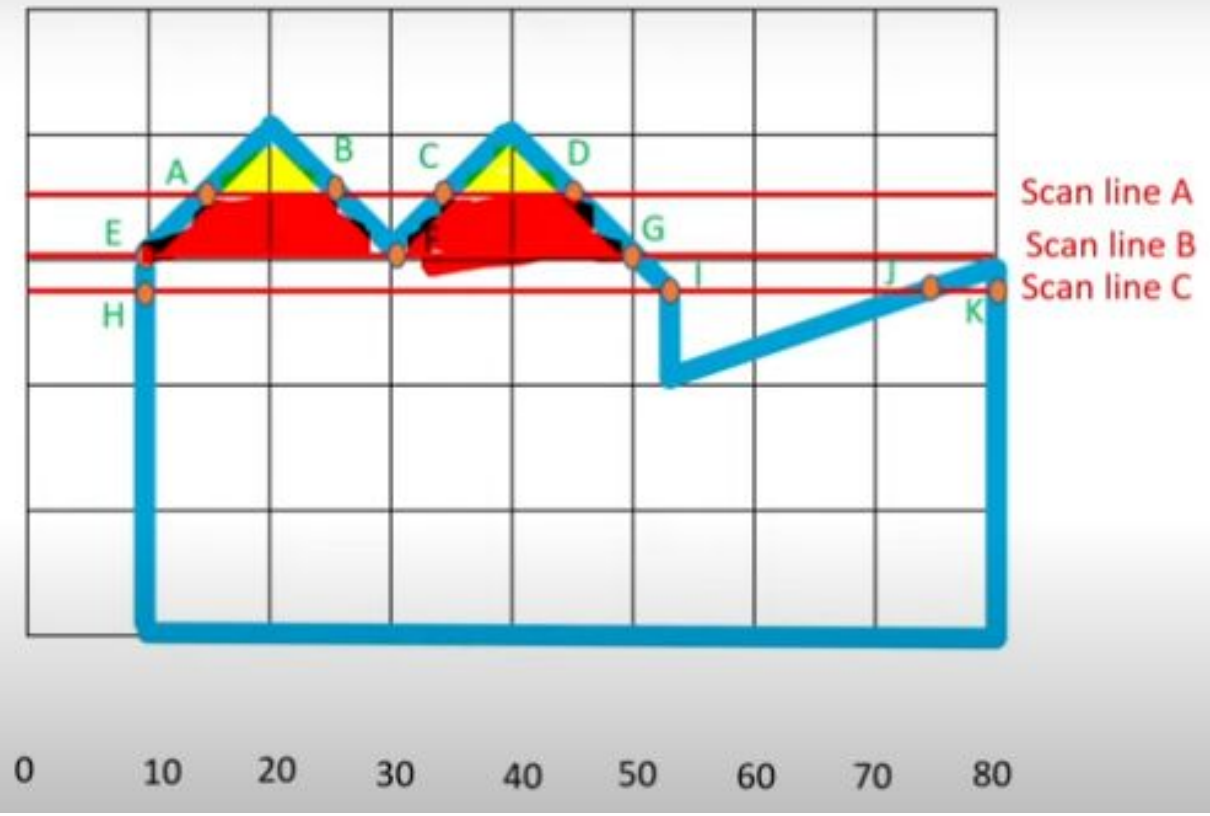
SCAN LINE A {A , B} {C , D}

SCAN LINE B {E , F} {F, G}

SCAN LINE C {H, I} {J , K}

SCAN LINE D

SCAN LINE E



in polygons.

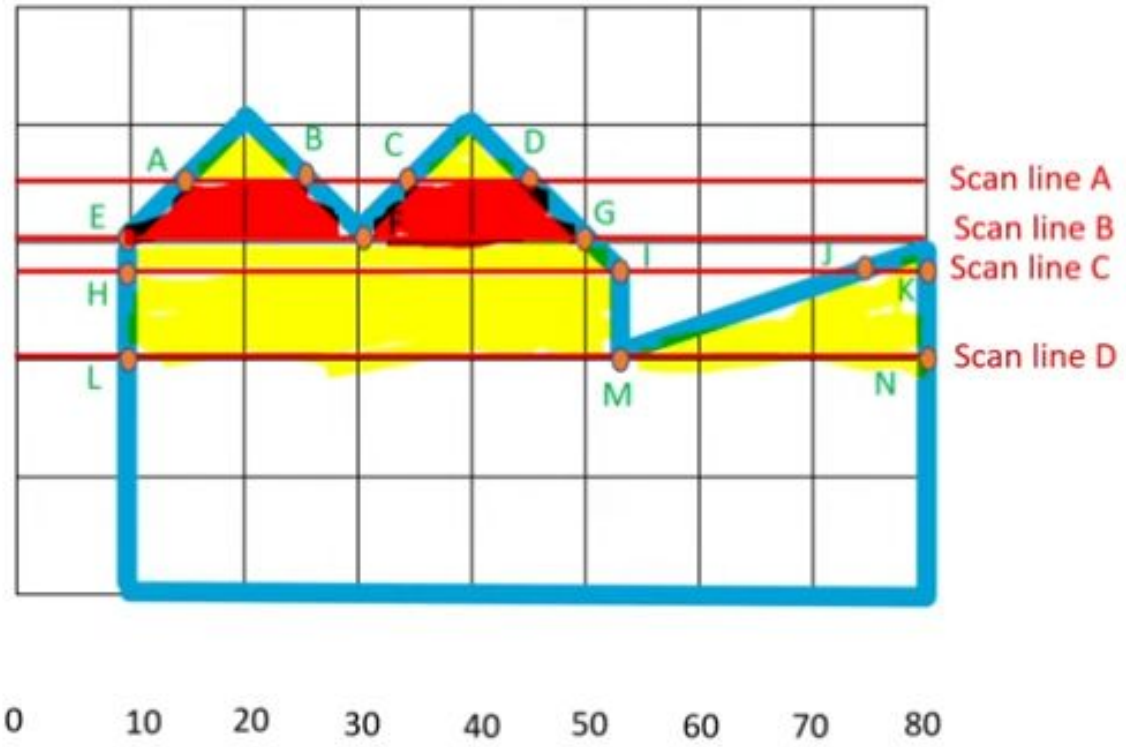
SCAN LINE A {A , B} {C, D}

SCAN LINE B {E , F} {F, G}

SCAN LINE C {H, I} {J , K}

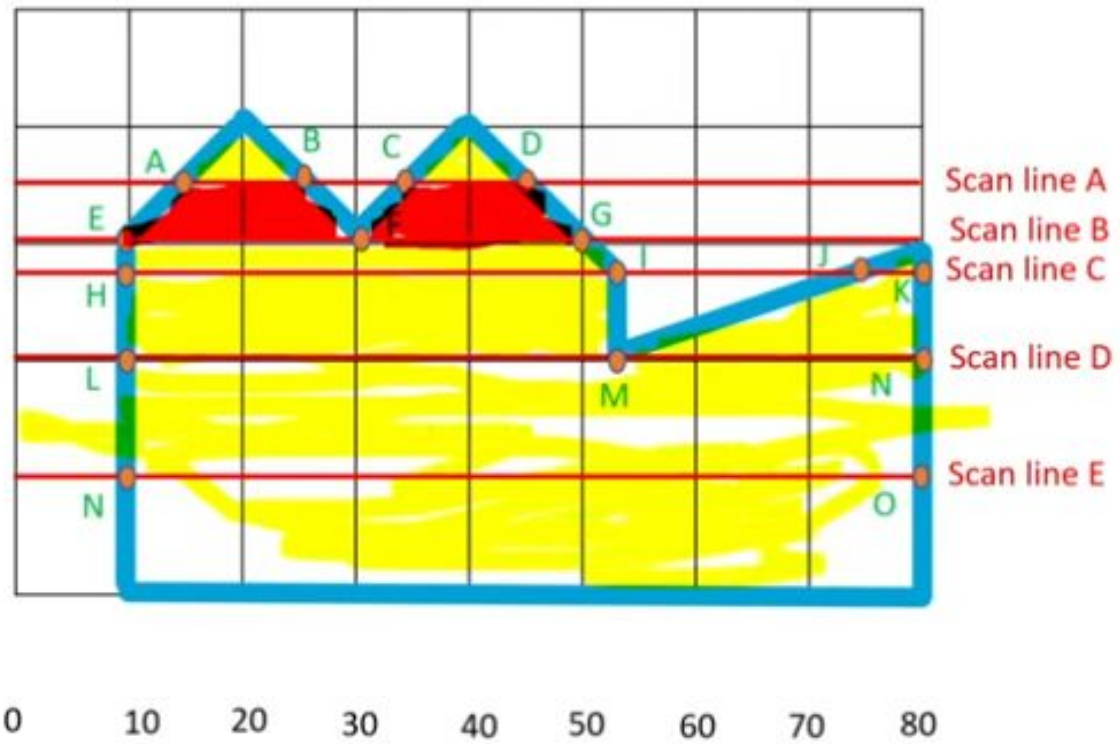
SCAN LINE D {L , M} {M, N}

SCAN LINE E



Scan line polygon filling algorithm is used for solid color filling in polygons.

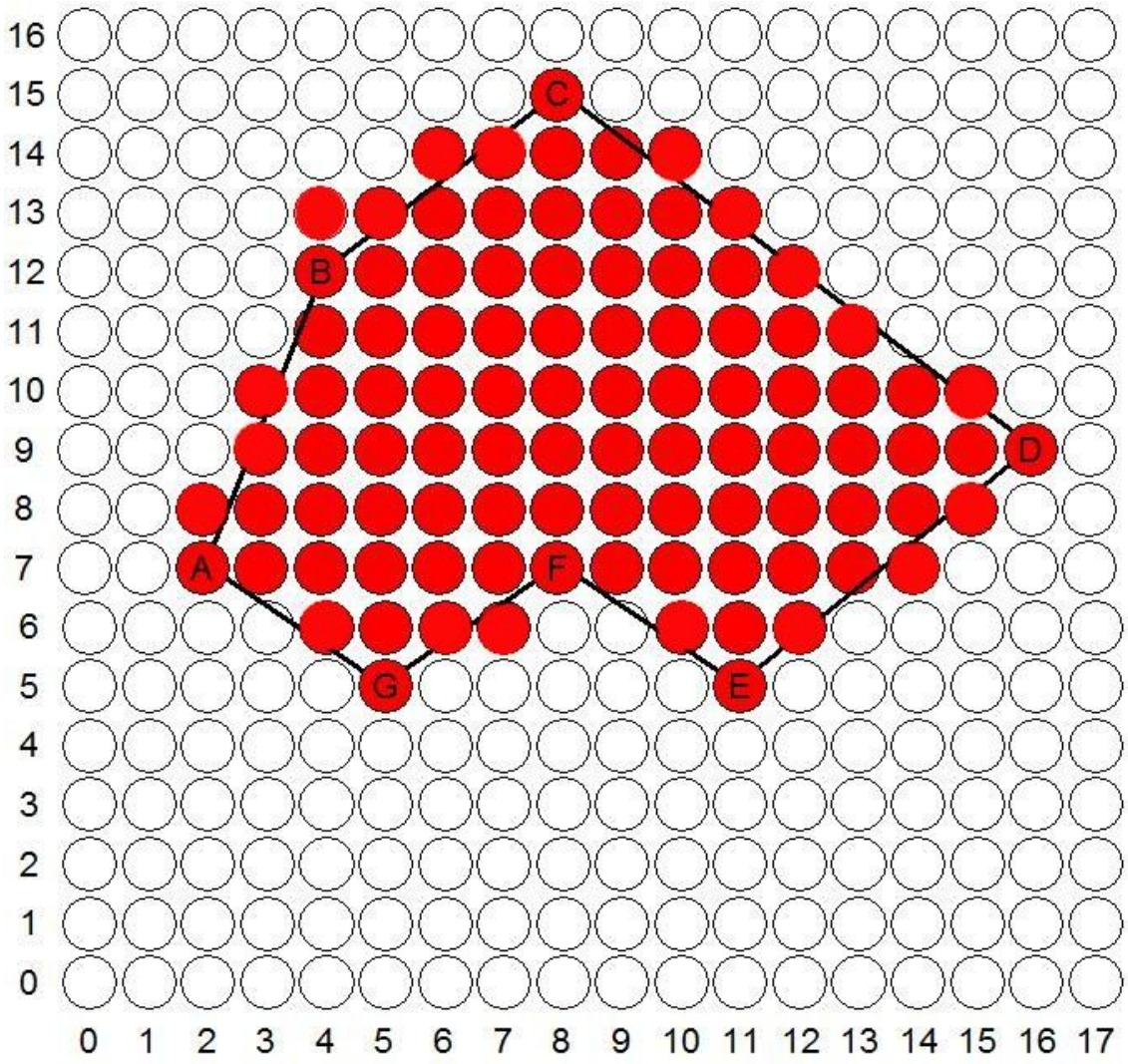
SCAN LINE A {A , B} {C, D}
SCAN LINE B {E , F} {F, G}
SCAN LINE C {H, I} {J , K}
SCAN LINE D {L , M} {M, N}
SCAN LINE E {N , O}



Special Cases:

- Some scan-line intersections at polygon vertices require special handling.
- A scan line passing through a vertex intersects two polygon edges at that position, adding two points to the list of intersections for the scan line.

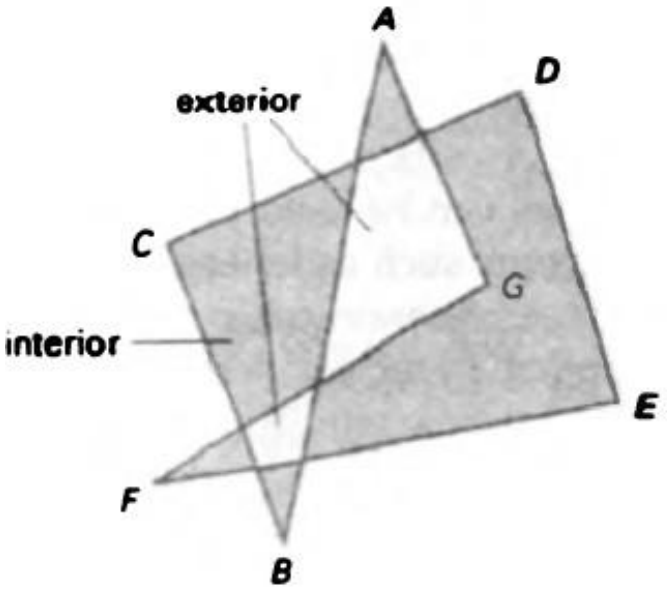
Scan-Line Polygon Fill Algorithm



Inside-outside Test

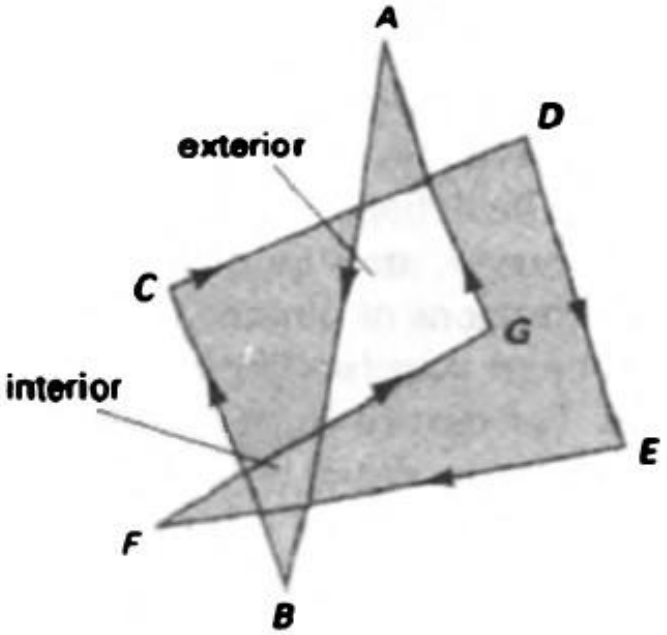
- When an object is filled, it is essential to identify whether the specific point is inside the object or outside the object.
- This is done by Inside-outside test also known as counting number method.
- An object can be identified whether inside or outside by two methods:
  - Odd-Even Rule
  - Nonzero winding number rule

Inside-outside Test



Odd Even rule

(a)



Nonzero Winding Number Rule

(b)

Inside-outside Test

Odd-Even Rule

- Drawing a line from any position P to a distant point outside the coordinate extents of the object and counting the number of edge crossings along the line.
- If the number of polygon edges crossed by this line is odd, then P is an interior point. Otherwise, P is an exterior point.
- To obtain an accurate edge count, we must be sure that the line path we choose does not intersect any polygon vertices.

Inside-outside Test

Non zero winding number rule

- Counts the number of times the polygon edges wind around a particular point in the counter clockwise direction. This count is called the winding number.
- The interior points of a two-dimensional object are defined to be those that have a nonzero value for the winding number.
- We apply the nonzero winding number rule to polygons by initializing the winding number to 0 and again imagining a line drawn from any position P to a distant point beyond the coordinate extents of the object.

Inside-outside Test

- The line we choose must not pass through any vertices. As we move along the line from position P to the distant point, we count the number of edges that cross the line in each direction. We add 1 to the winding number every time we intersect a polygon edge that crosses the line from right to left, and we subtract 1 every time we intersect an edge that crosses from left to right.
- The final value of the winding number, after all edge crossings have been counted, determines the relative position of P. If the winding number is nonzero, P is defined to be an interior point. Otherwise, P is taken to be an exterior point.

Inside-outside Test

- For standard polygons and other simple shapes, the nonzero winding number rule and the odd-even rule give the same results.
- But for more complicated shapes, the two methods may yield different interior and exterior regions.

Boundary-Fill Algorithm

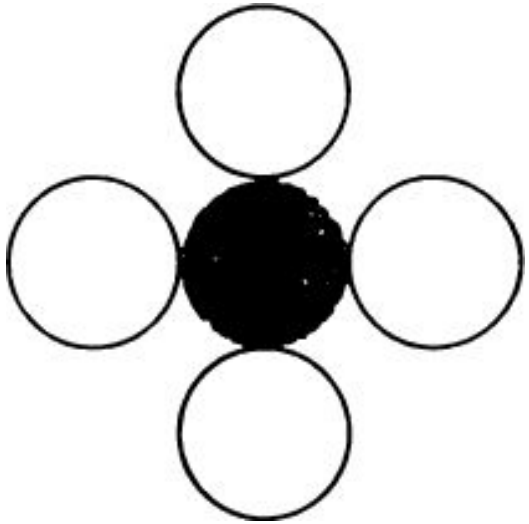
- Another approach to area filling is to start at a point inside a region and paint the interior outward toward the boundary.
- If the boundary is specified in a single color, the fill algorithm proceeds outward pixel by pixel until the boundary color is encountered.
- This method, called the boundary-Fill algorithm, is particularly useful in interactive painting packages, where interior points are easily selected.
- To display a solid color region (with no border), the designer can choose the fill color to be the same as the boundary color.

Boundary-Fill Algorithm

- A boundary-fill procedure:
  1. Accepts as input the coordinates of an interior point (x, y), a fill color, and a boundary color.
  2. Starting from (x, y), the procedure tests neighbouring positions to determine whether they are of the boundary color.
  3. If not, they are painted with the fill color, and their neighbours are tested.
  4. This process continues until all pixels up to the boundary color for the area have been tested.

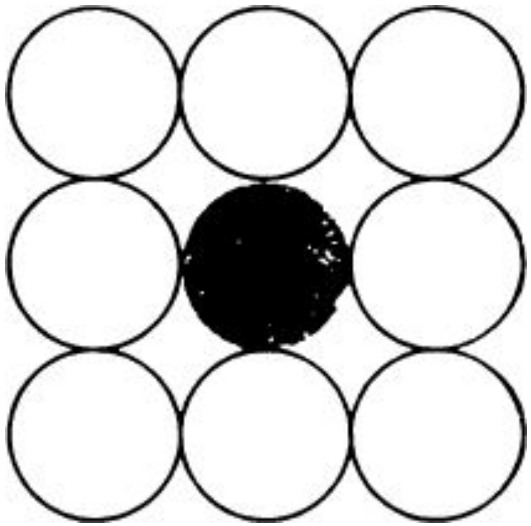
Boundary-Fill Algorithm

- Two methods are used for proceeding to neighbouring pixels from the current test position.
  - 4-connected pixels



Boundary-Fill Algorithm

- 8-connected pixels



Boundary-Fill Algorithm

The following procedure illustrates a recursive method for filling a 4-connected area with an intensity specified in parameter fill up to a boundary color specified with parameter boundary.

```
void boundaryFill4 (int x, int y, int fill, int boundary)
{
        int current;
        current = getPixel (x, y);
        if ((current != boundary) && (current != fill))
        {
                setColor (fill) ;
                setPixel (x, y);
                boundaryFill4 (x+1, y, fill, boundary);
                boundaryFill4 (x-1, y, fill, boundary);
```

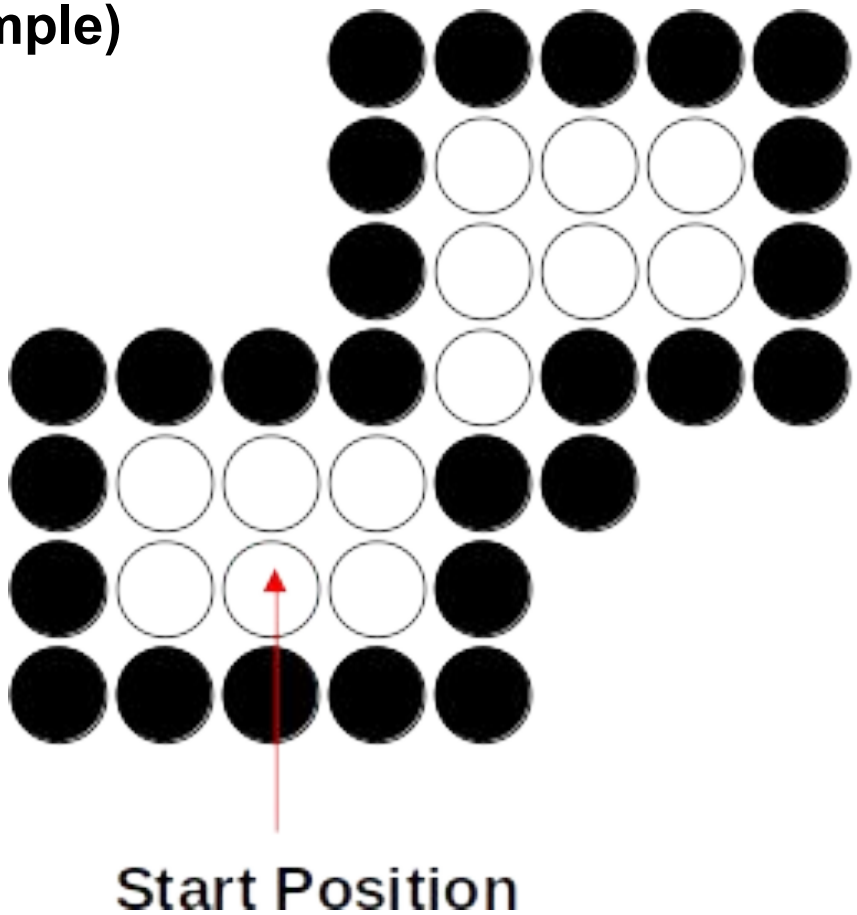
Boundary-Fill Algorithm

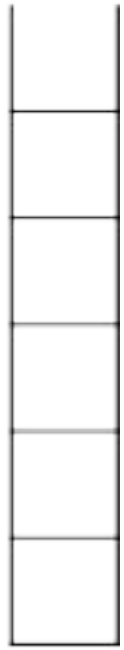
```
            boundaryFill4 (x, y+1, fill, boundary);
            boundaryFill4 (x, y-1, fill, boundary);


        }
    }
```

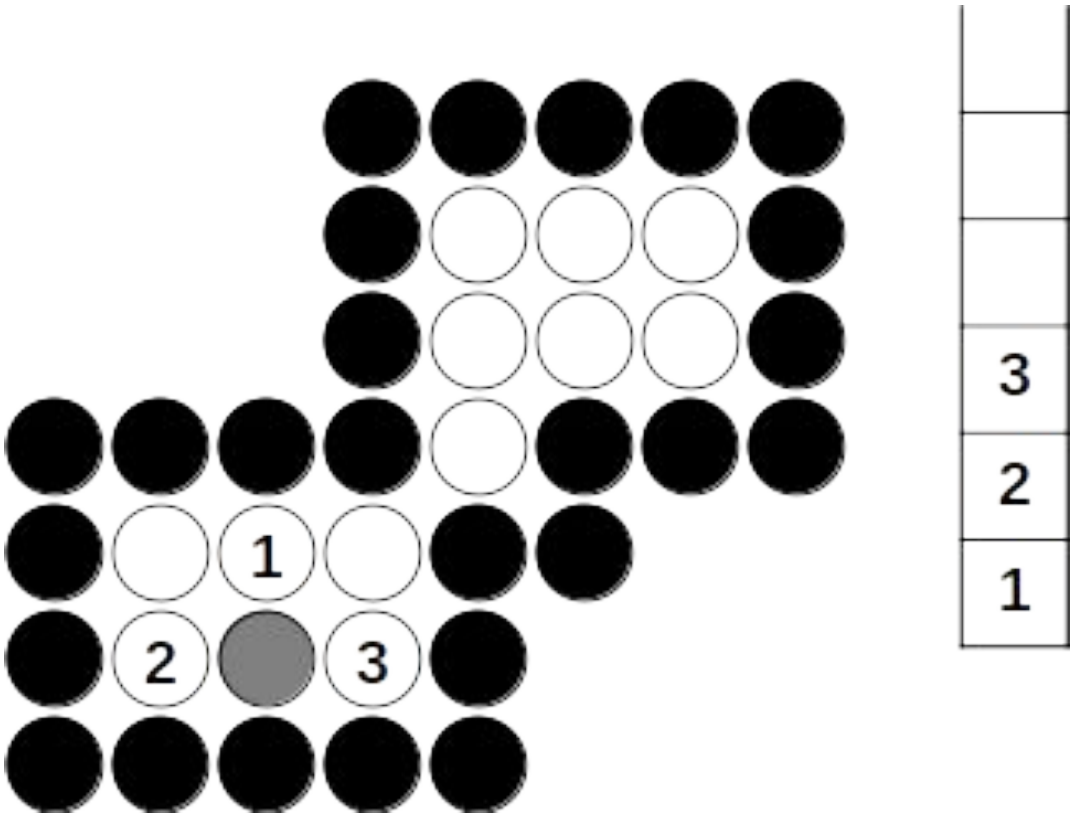
Boundary-Fill Algorithm

4-connected (Example)

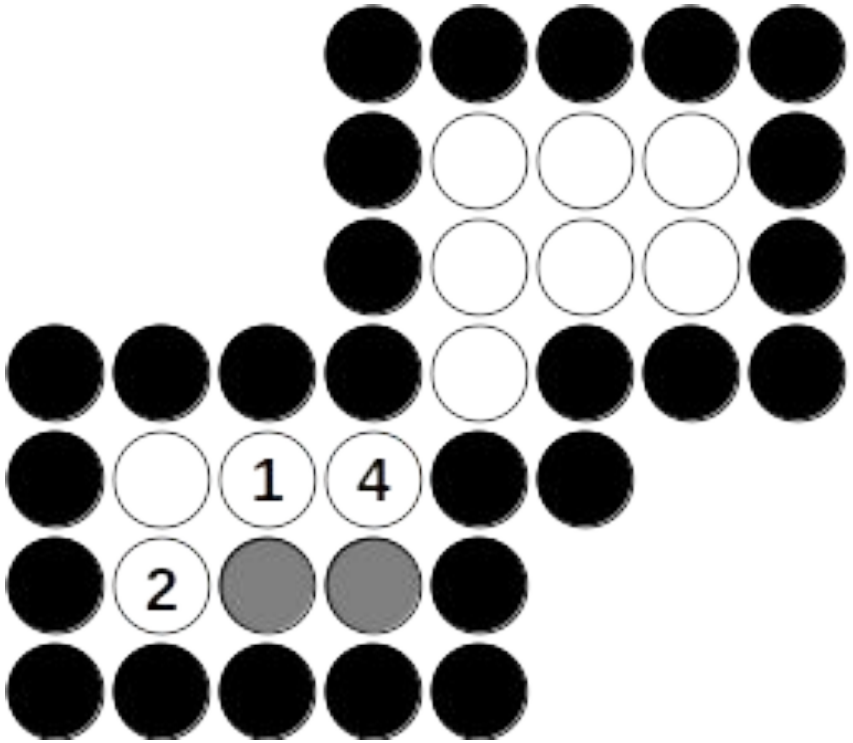




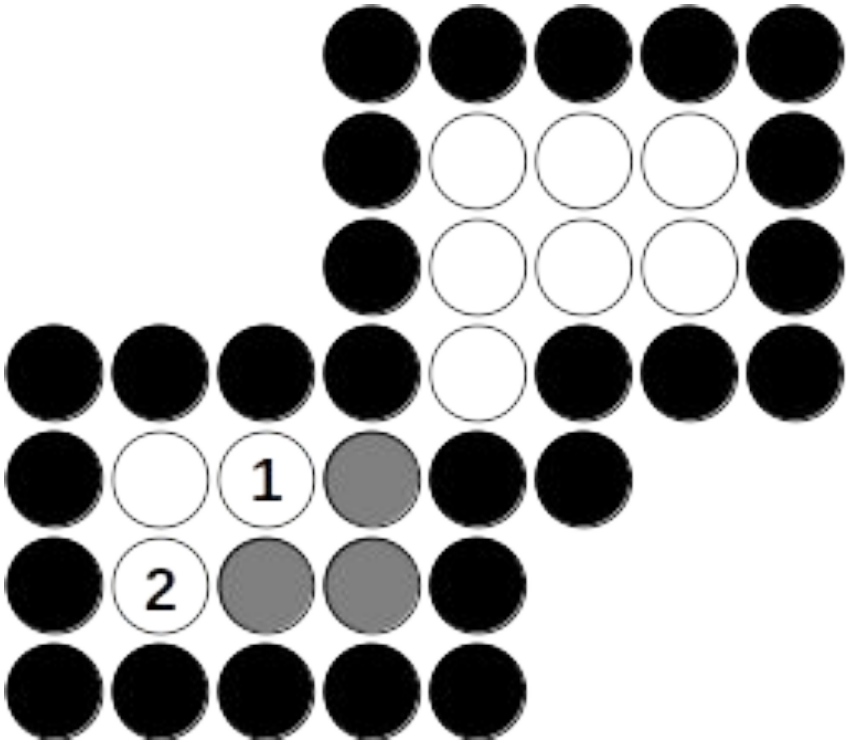
Boundary-Fill Algorithm



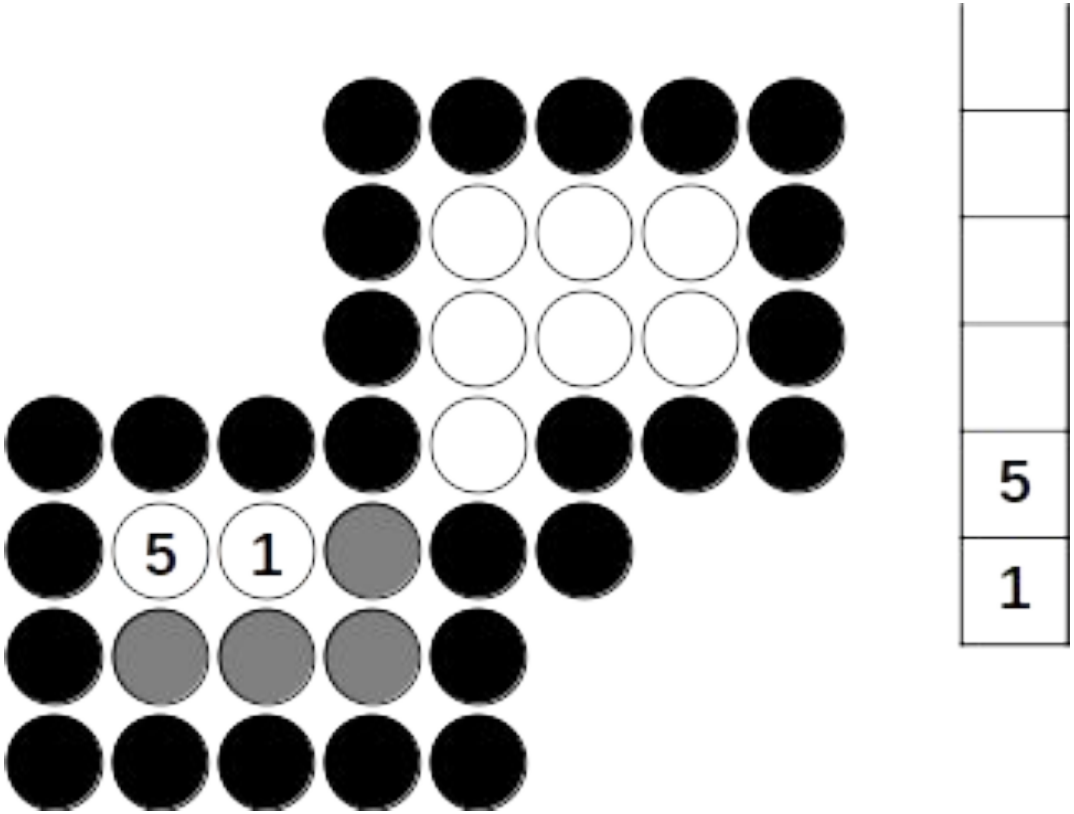
Boundary-Fill Algorithm



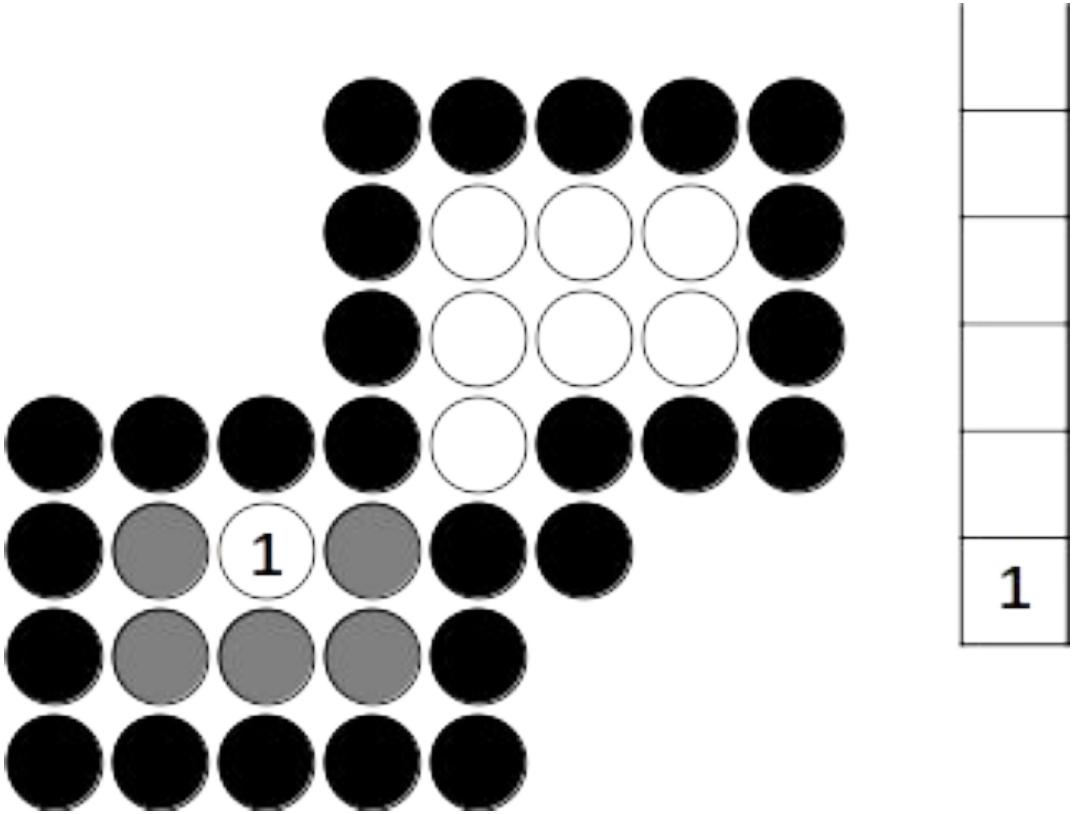
Boundary-Fill Algorithm



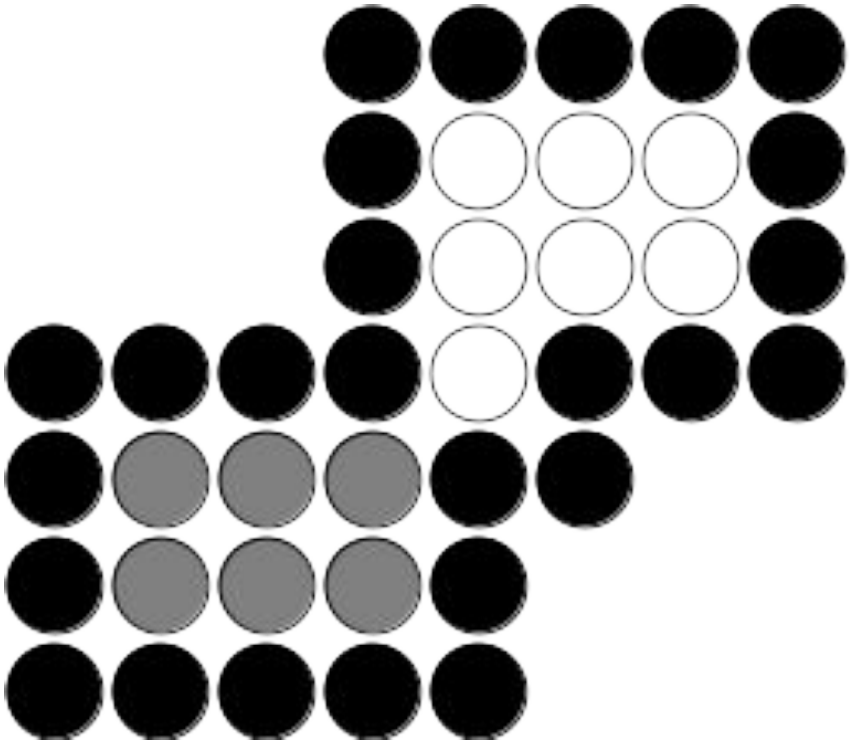
Boundary-Fill Algorithm



Boundary-Fill Algorithm

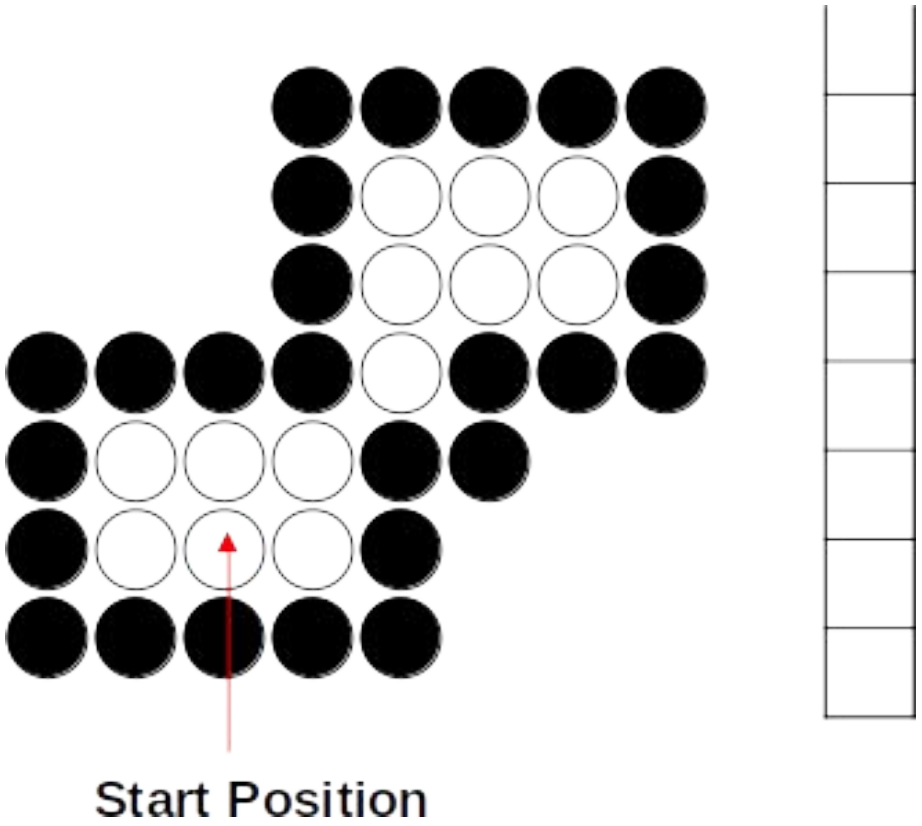


Boundary-Fill Algorithm

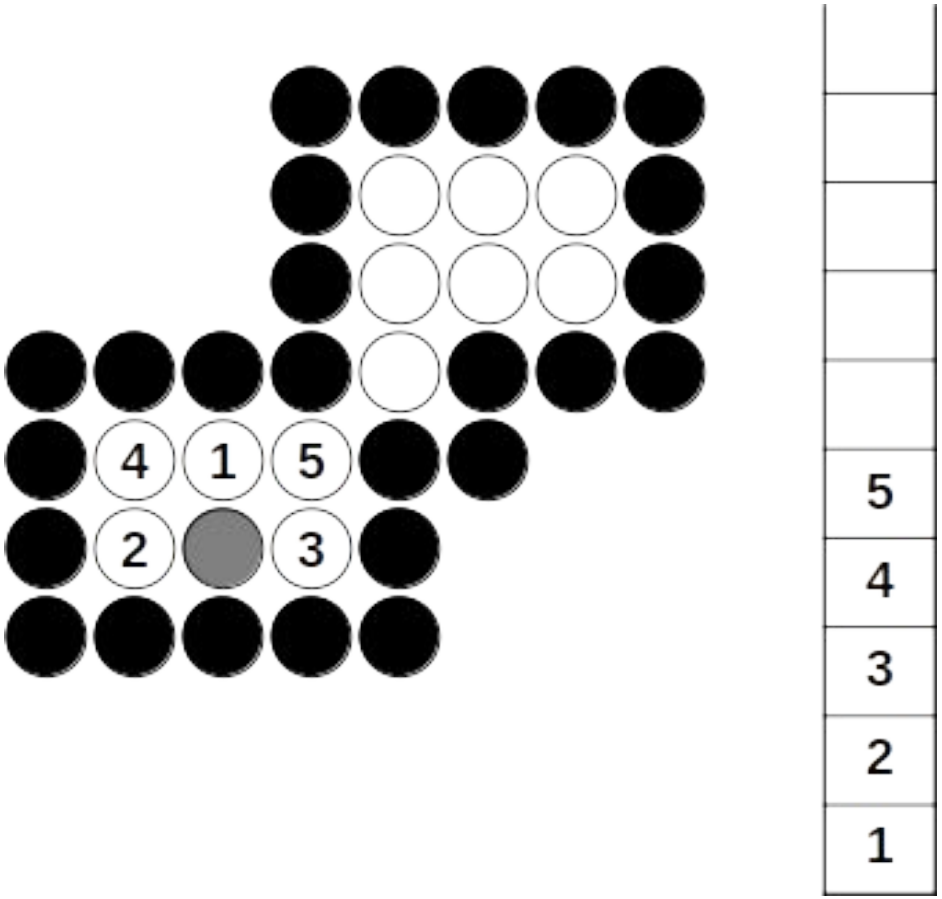


Boundary-Fill Algorithm

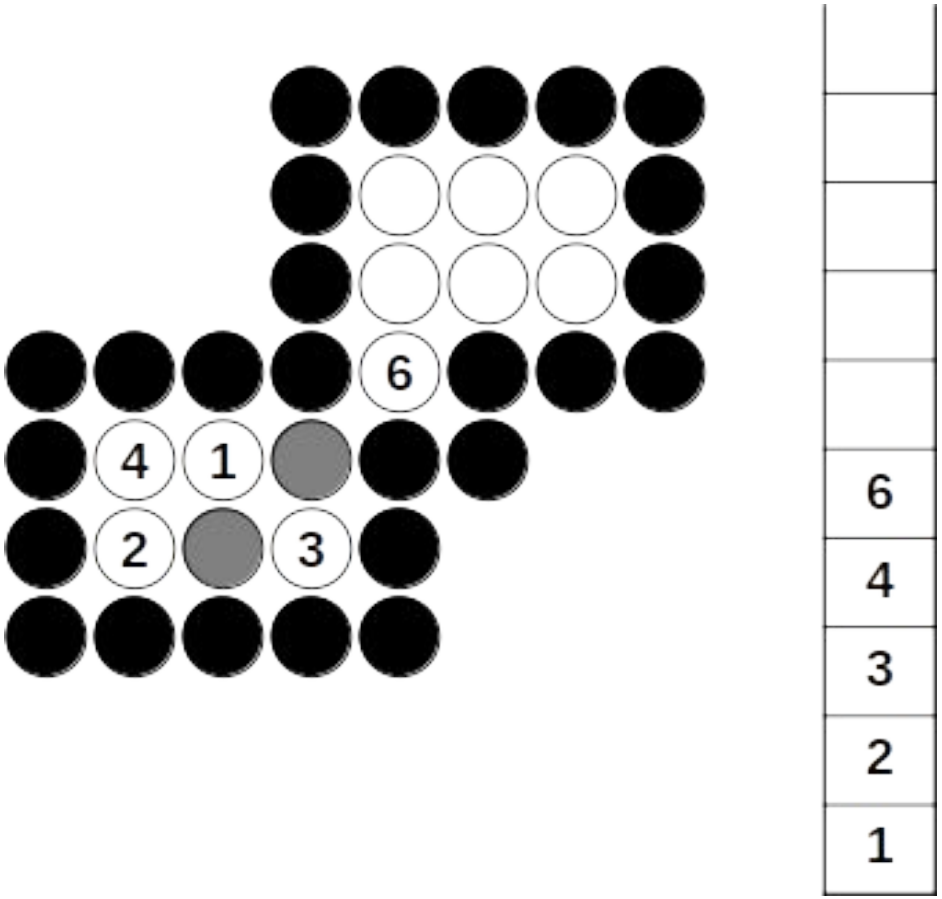
8-connected (Example)



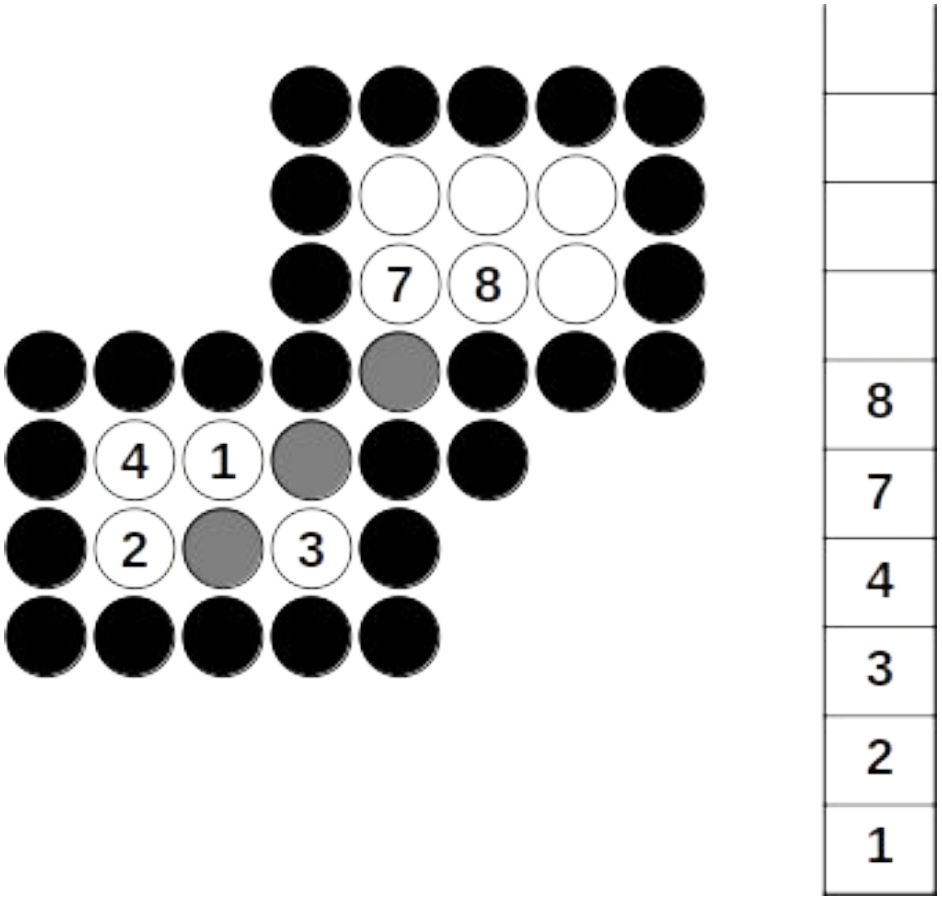
Boundary-Fill Algorithm



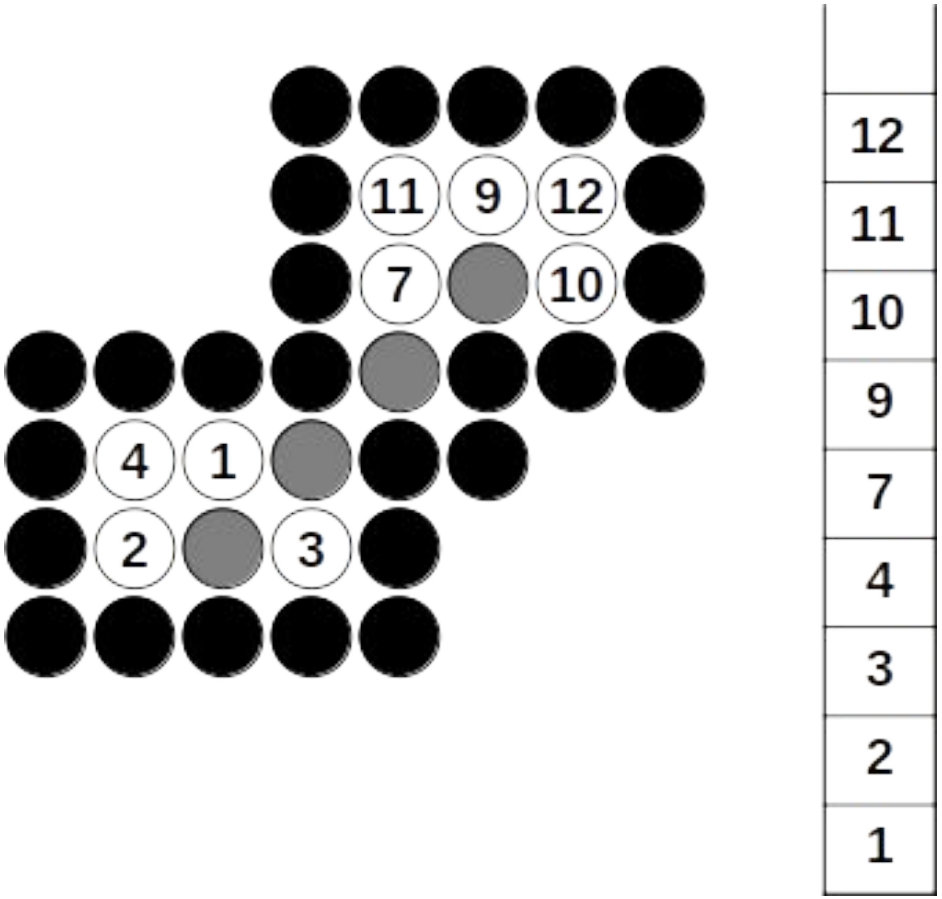
Boundary-Fill Algorithm



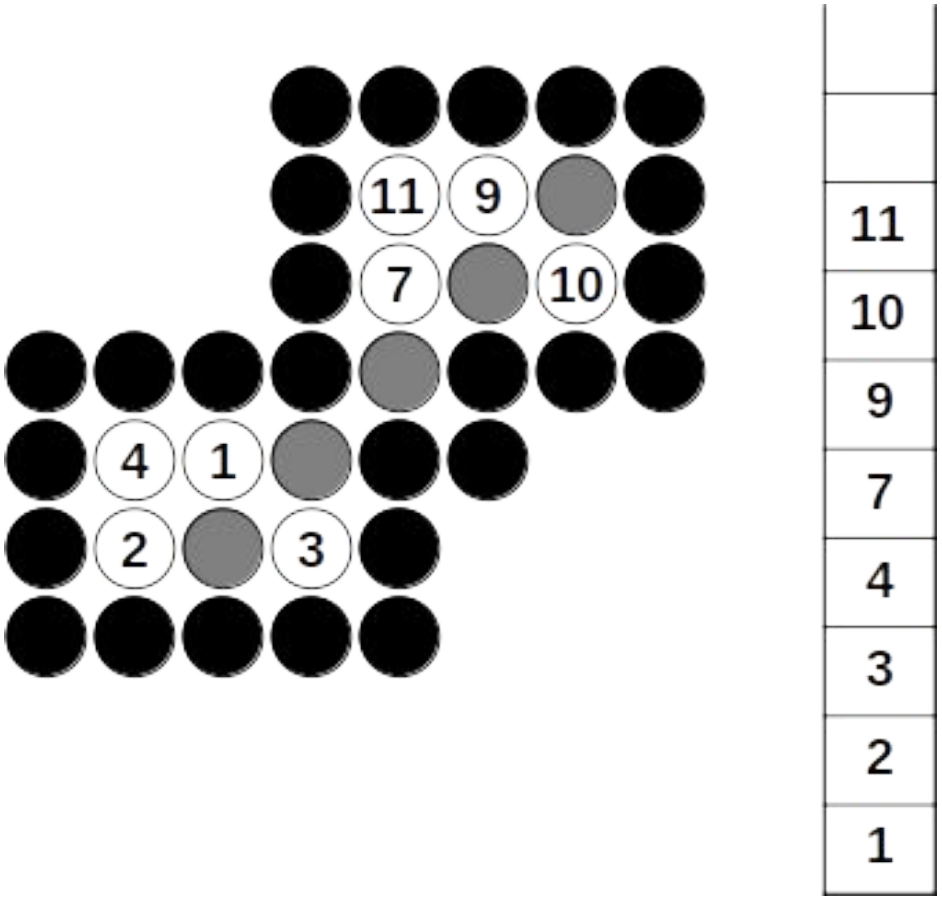
Boundary-Fill Algorithm



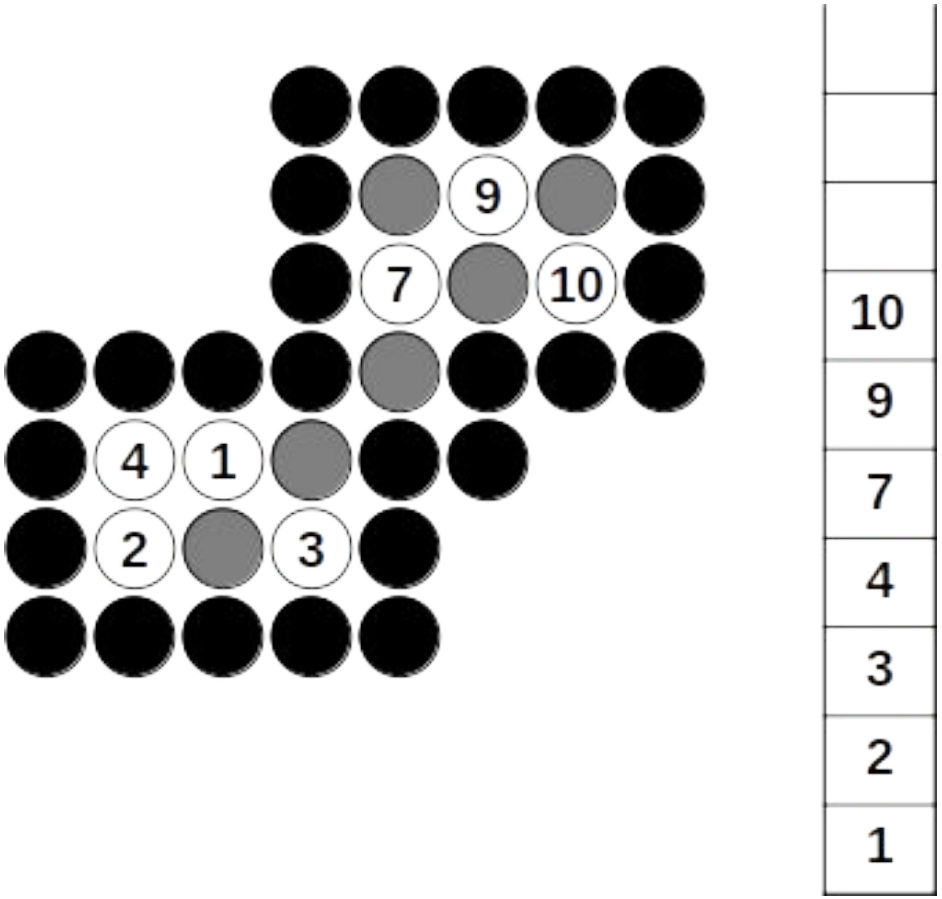
Boundary-Fill Algorithm



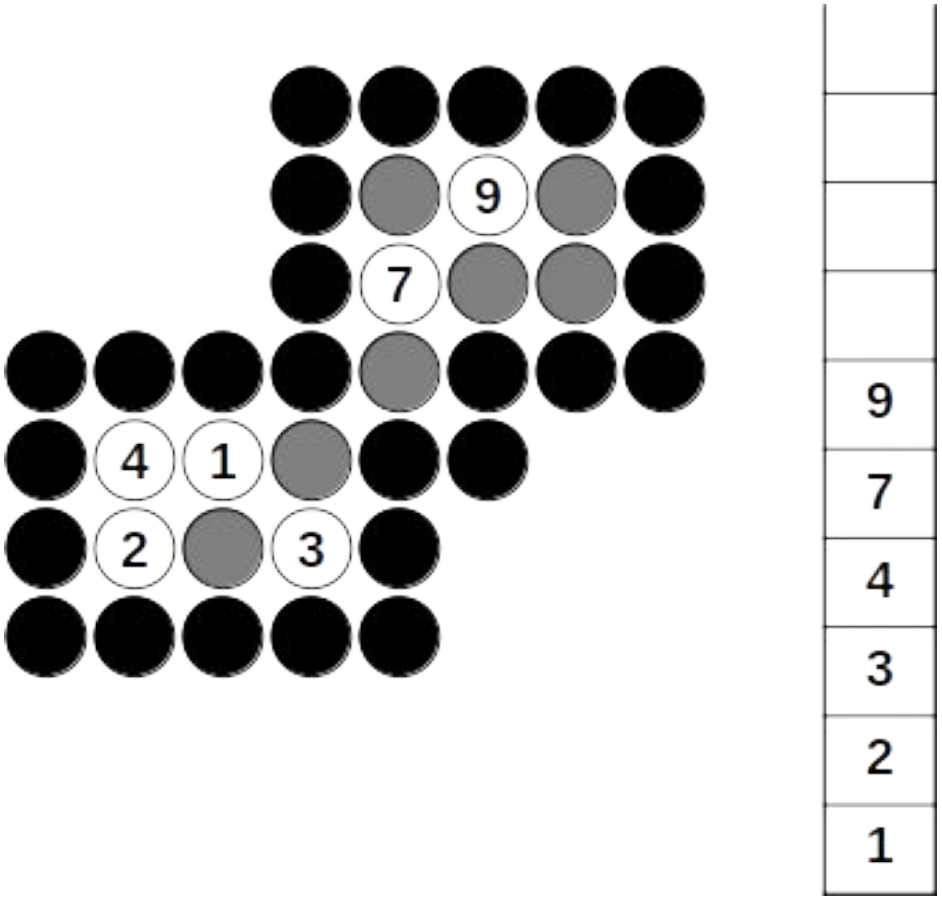
Boundary-Fill Algorithm



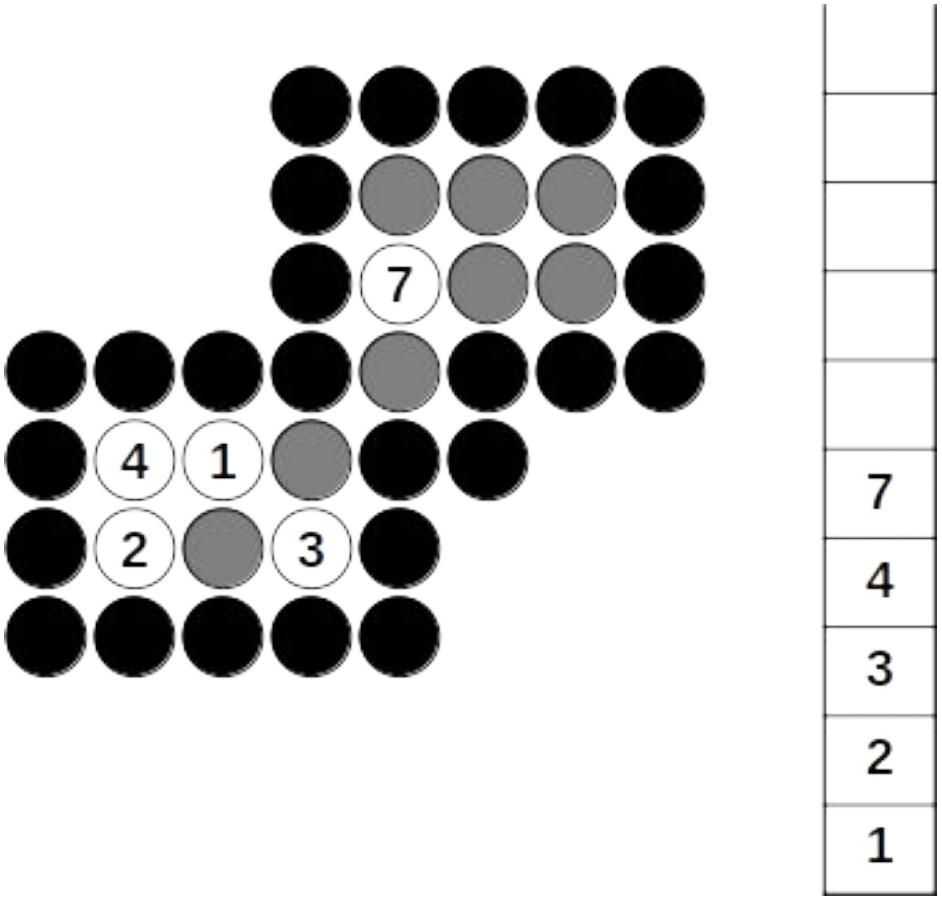
Boundary-Fill Algorithm



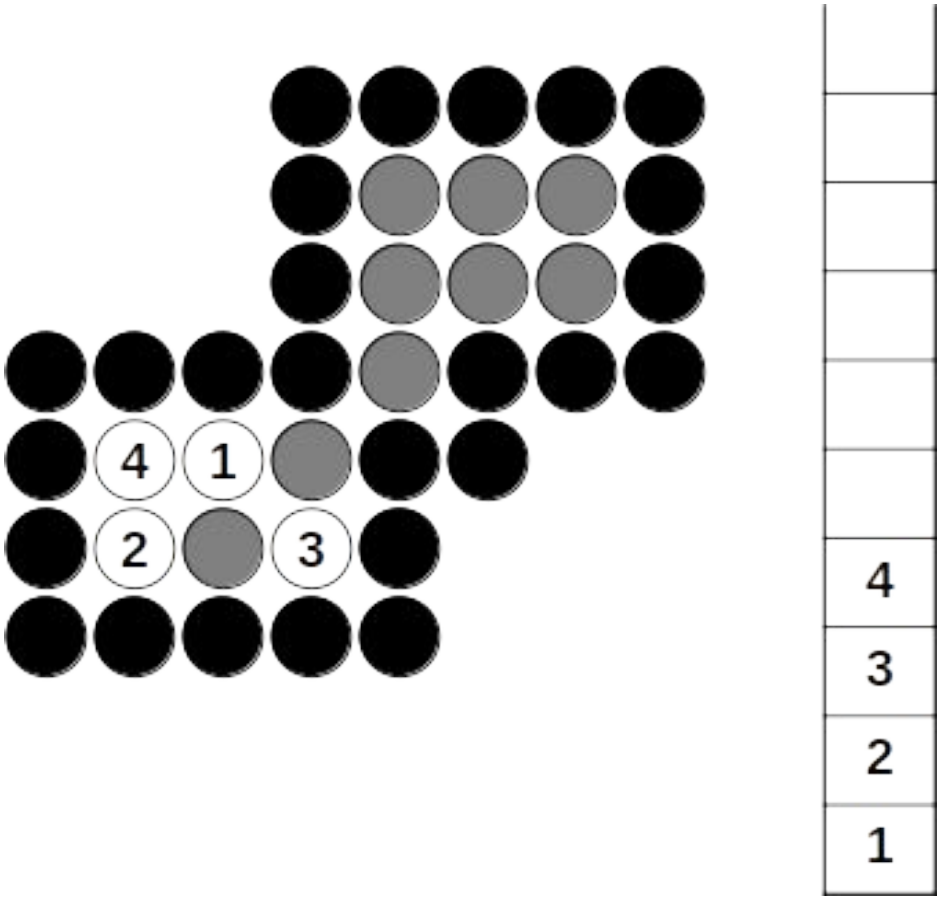
Boundary-Fill Algorithm



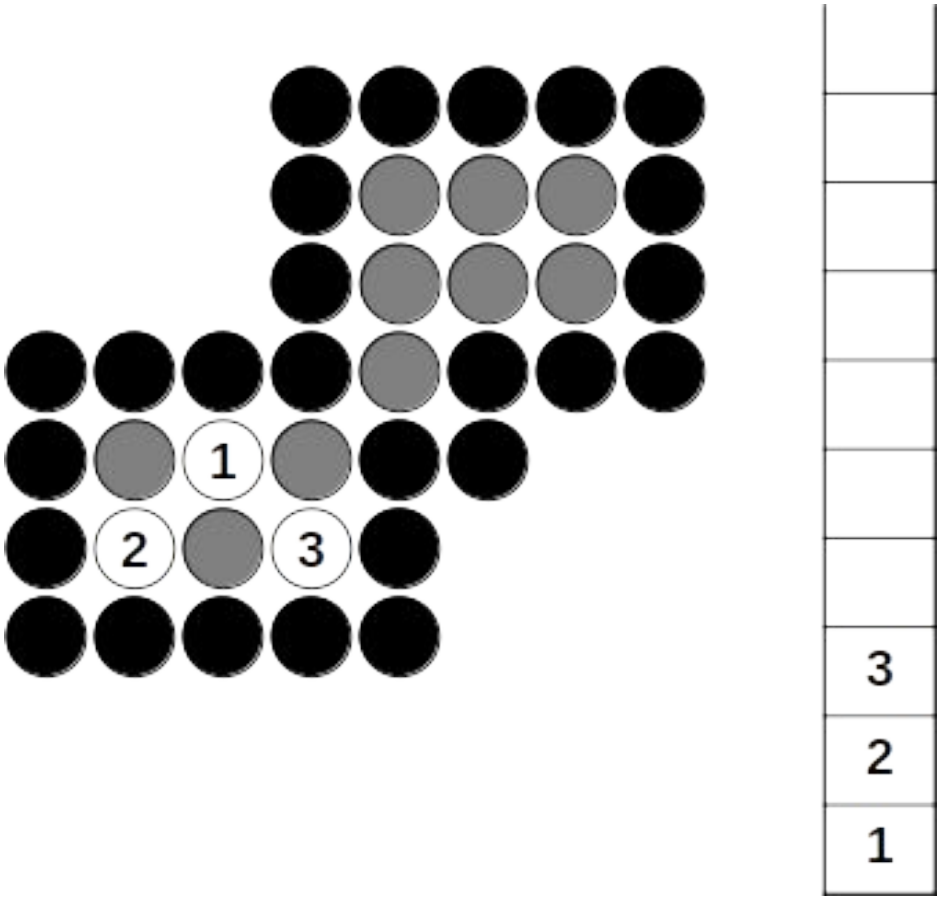
Boundary-Fill Algorithm



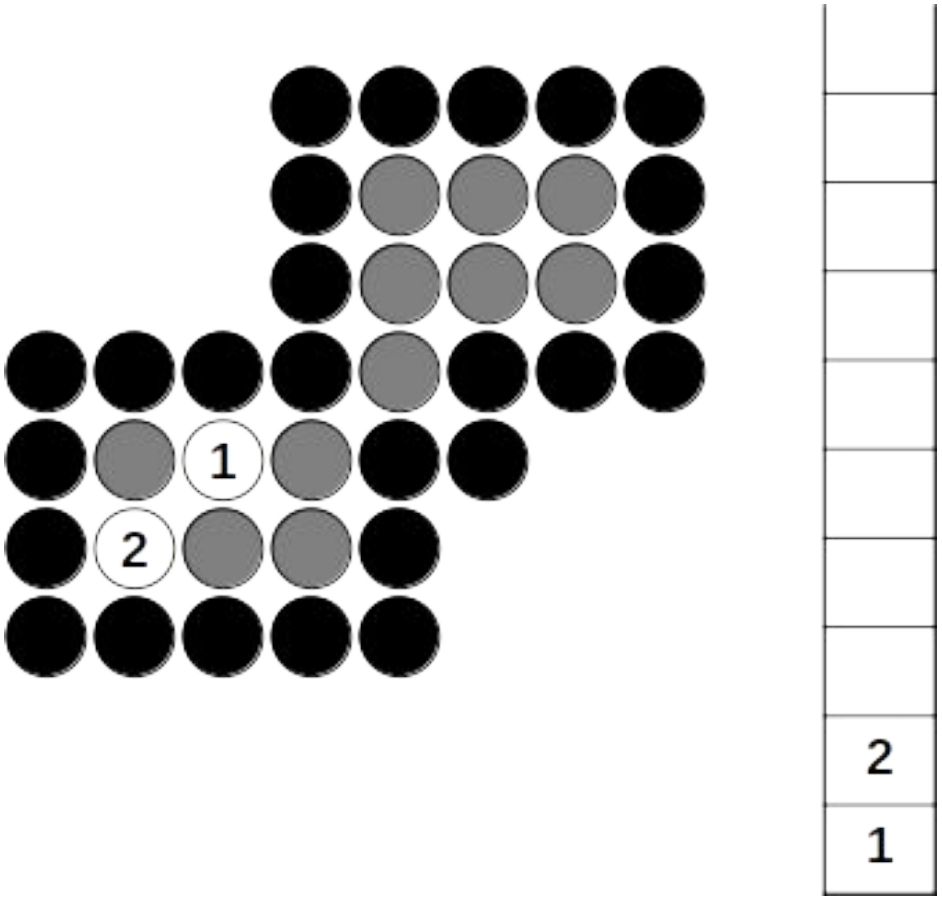
Boundary-Fill Algorithm



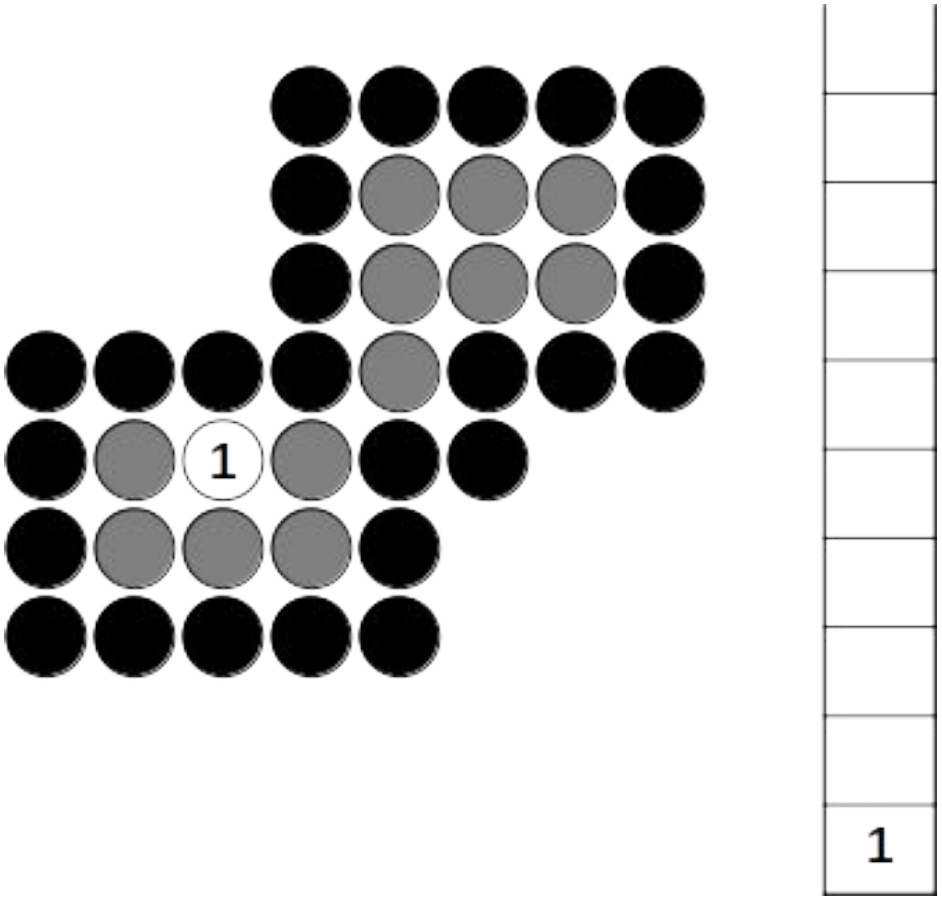
Boundary-Fill Algorithm



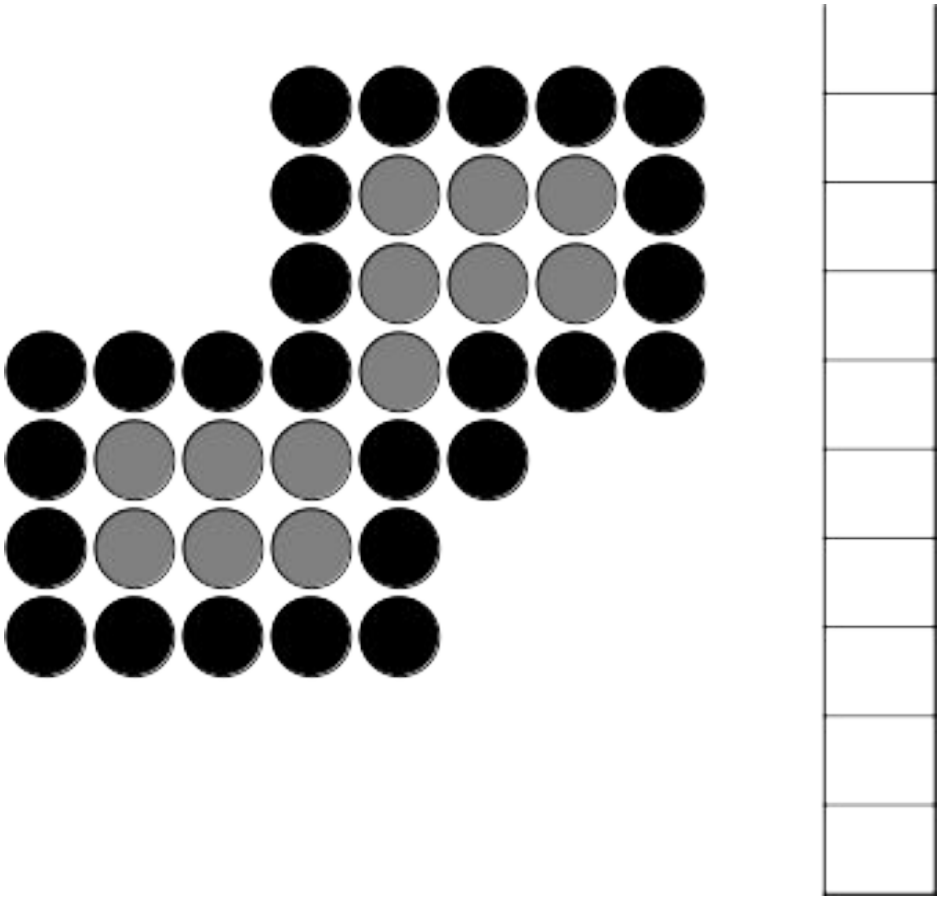
Boundary-Fill Algorithm



Boundary-Fill Algorithm



Boundary-Fill Algorithm

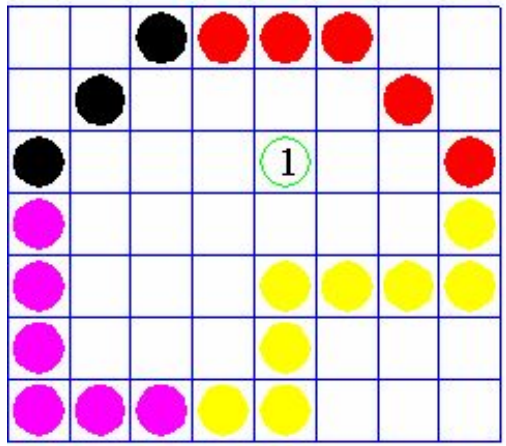


Boundary-Fill Algorithm

- It requires considerable stacking of neighboring pixels, more efficient methods are generally employed.
- Span Flood-Fill fill horizontal pixel spans across scan lines, instead of proceeding to 4-connected or 8-connected neighboring pixels.
- It needs only stack a beginning position for each horizontal pixel spans, instead of stacking all unprocessed neighboring positions around the current position.

Flood Fill Algorithm

- Used to fill (or recolor) an area that is not defined within a single color boundary.
- Paint such areas by replacing a specified interior color instead of searching for a boundary color value.



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.

Flood Fill Algorithm

The following procedure flood fills a 4-connected region recursively, starting from the input position.

```
void floodFill4 (int x, int y, int fillColor, int oldColor)
{
    if (getPixel (x, y) == oldColor)
    {
        setColor (fillColor) ; setPixel
        (x, y);
        floodFill4 (x+1, y, fillColor, oldColor); floodFill4 (x-1,
        y, fillColor, oldColor); floodFill4 (x, y+1, fillColor,
        oldColor);
```

Flood Fill Algorithm

```
            floodFill4 (x, y-1, fillColor, oldColor);

        }

}
```