Introduction

A standard output primitive in general graphics packages is a solid-color or patterned polygon area because polygons are easier to process since they have linear boundaries.

There are two basic approaches to area filling.

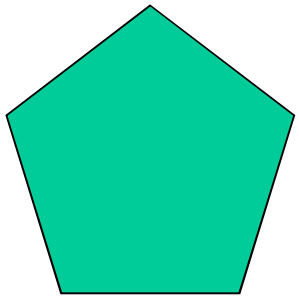
1. Determine the overlap intervals for scan lines that cross the area.
2. Start from a given interior position and paint outward from this point until encounter the specified boundary conditions.

The scan-line approach is typically used in general graphics packages to fill polygons, circles, ellipses, and other simple curves.

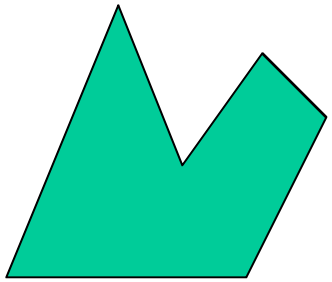
Interior points are useful with more complex boundaries and in interactive painting systems.

Introduction to Polygons

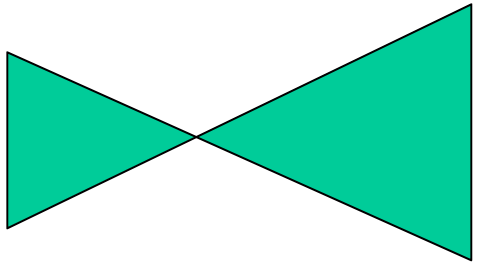
- ***Different types of Polygons***
- Simple Convex
- Simple Concave
- Non-simple : self-intersecting
- With holes



Convex



Concave



Self-intersecting

Introduction to Polygons

- Convex

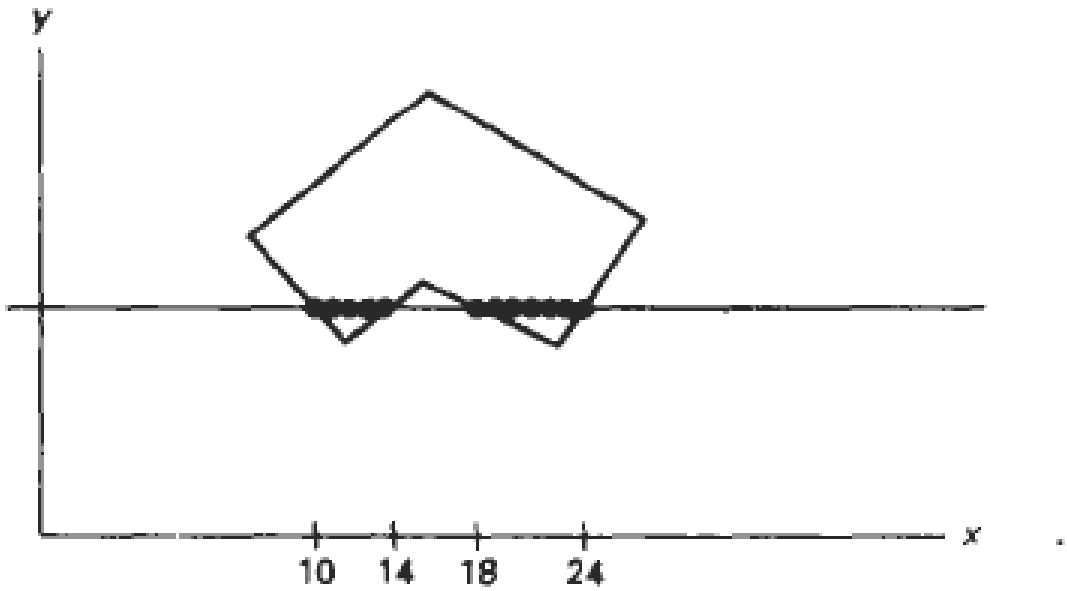
  A region S is convex iff for any x1 and x2 in S, the straight line segment connecting x1 and x2 is also contained in S. The convex hull of an object S is the smallest H such that S

Scan Line Polygon F i l l Algorithm

For each scan line crossing a polygon, algorithm locates the intersection

points of the scan line with the polygon edges.

These intersection points are then sorted from left to right, and the corresponding frame-buffer positions between each intersection pair are set to the specified fill color.

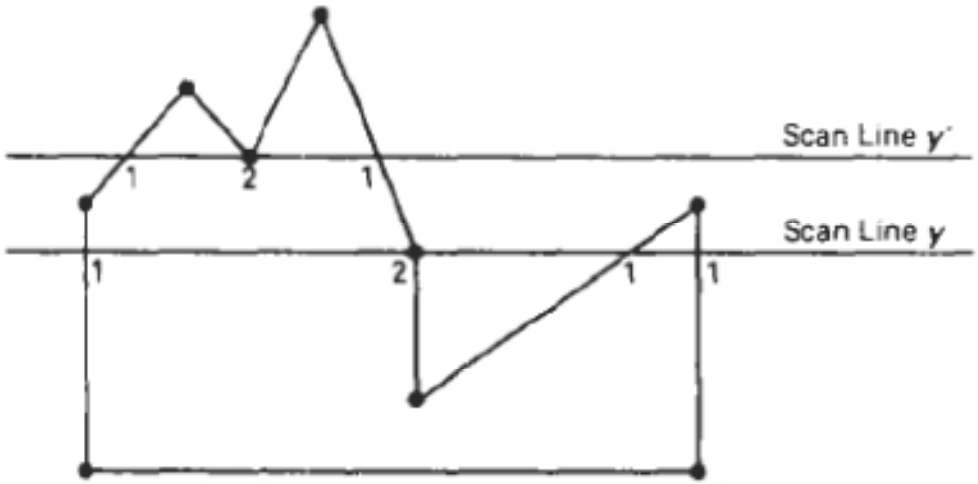


There are interior pixels from ***x*** = 10 to ***x*** = 14 and from ***x*** = 18 to **x** = 24.

Some scan-line intersections at **polygon vertices** require special handling.

A scan line passing through a vertex intersects two edges at that position,

adding two points to the list of intersections for the scan line.



Two scan lines at positions y and y' that intersect edge endpoints.

Scan line y intersects five polygon edges.

Intersection points along scan line **y'** correctly identify the interior pixel spans. But with scan line y, we need to do some additional processing to determine the correct interior points.

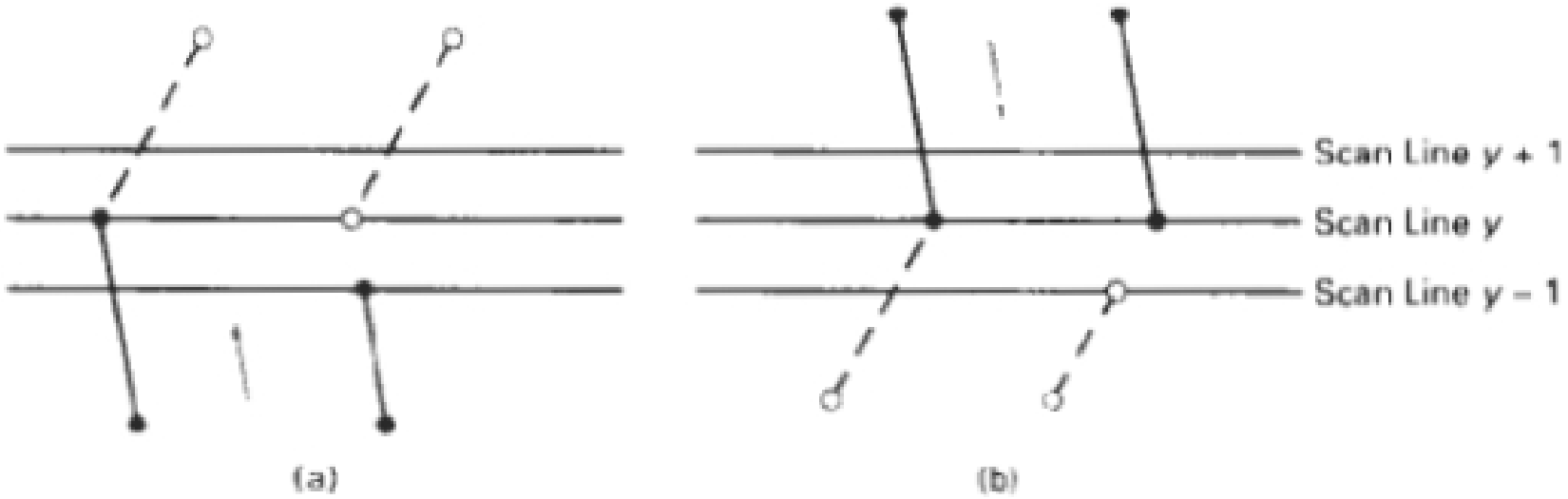
Note the position of the intersecting edges relative to the scan line.

For scan line y, the two intersecting edges sharing a vertex are on opposite sides of the scan line But for scan line **y'** the two intersecting edges are both above the scan line.

So the vertices that require additional processing are those that have connecting edges on opposite sides of the scan line.

Identify the vertices by tracing the polygon boundary either in clockwise or counterclockwise to observe the relative changes in vertex y coordinates.

If the Y values of two consecutive edges increase or decrease, count the middle vertex as a single intersection point Otherwise, the shared vertex represents a local extremum (minimum or maximum).

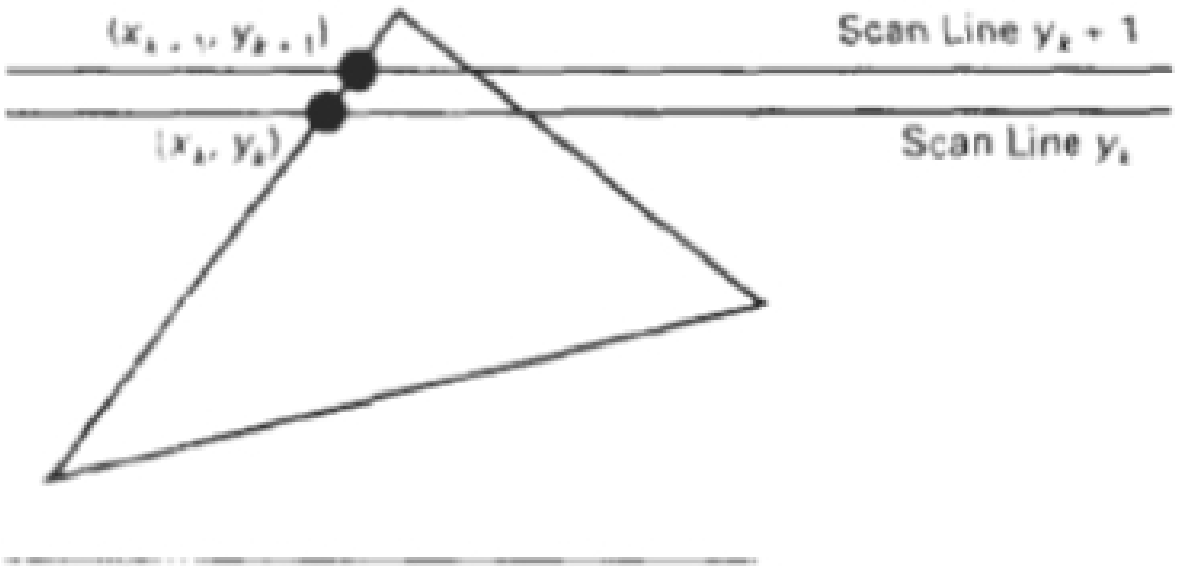


When the endpoint y coordinates of the two edges are increasing, the y value of the upper endpoint for the current edge **1s** decreased by 1, as in Fig. (a).

When the endpoint y values are decreasing, Fig.(b), we decrease the coordinate of the upper endpoint of the edge following the current edge.

Coherence : properties of one part of a scene are related in some way to other parts of the scene so that the relationship can be used to reduce processing.

Coherence methods often involve incremental calculations applied along a single scan line or between successive scan lines.



The slope of the polygon boundary line

$$\frac{y_{k+1}-y_k}{X_{k+1}-x_k}$$

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

And $X_{k+1}=x_k + 1/m$

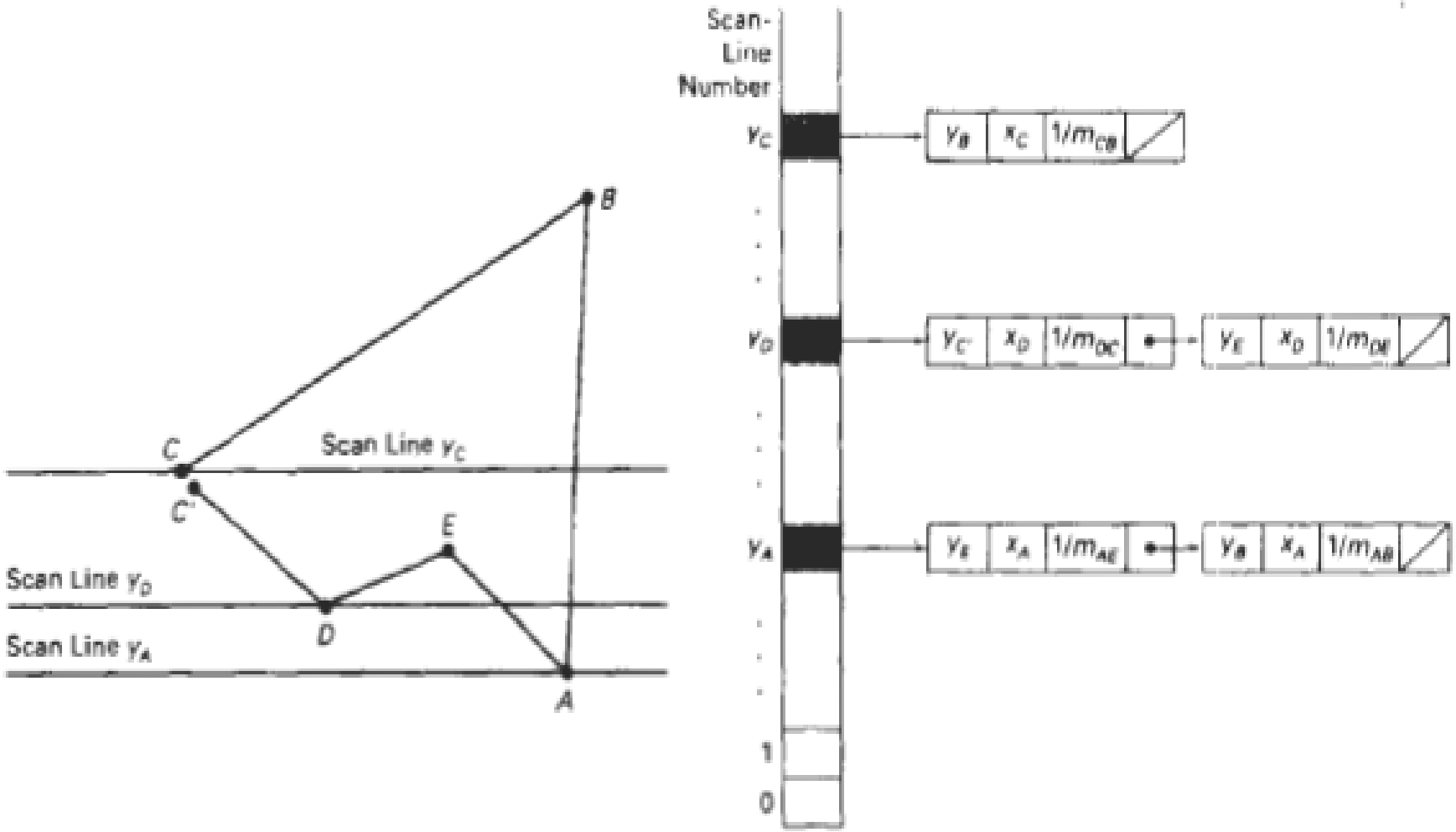


Figure 3-39
A polygon and its sorted edge table, with edge $\overline{DC}$ shortened by one unit in the y direction.

Inside-Outside Tests : Identify interior regions of objects.

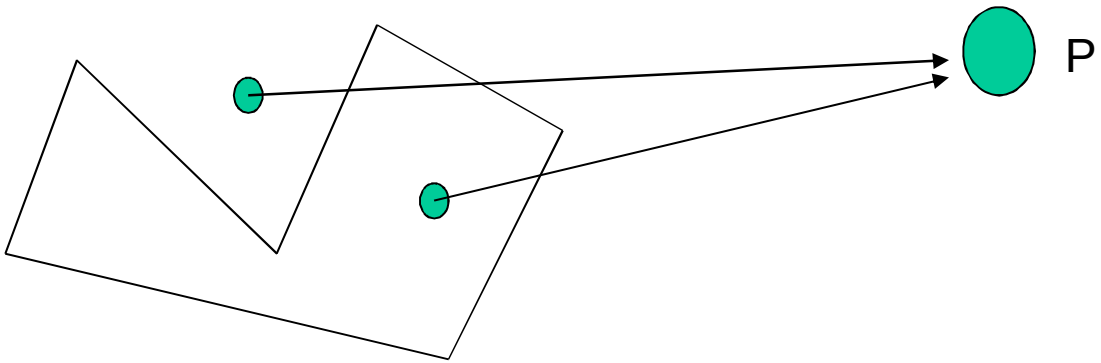
In elementary geometry, a polygon is usually defined as having no self-intersections. Standard polygons include triangles, rectangles, octagons, and decagons.

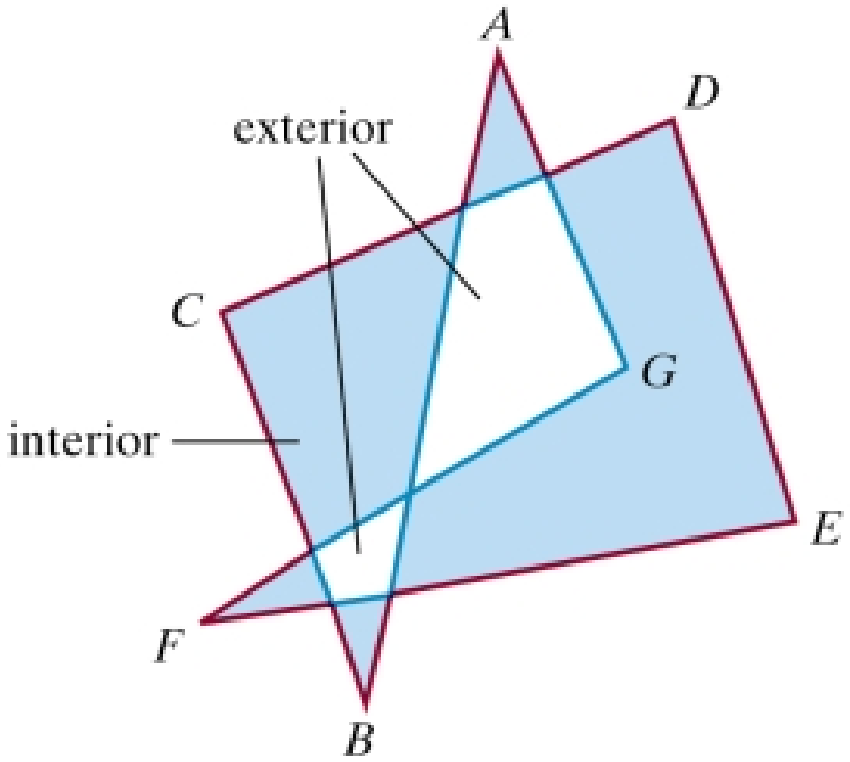
The component edges of these objects are joined only at the vertices, and otherwise the edges have no common points in the plane.

Graphics packages normally use either the odd-even rule to identify interior regions of an object.

Odd-Even rule:

- Let (x,y) be the point we are trying to determine if it is inside or outside of an object. Draw a line between this point and a distant point P. If the number of edges it crosses is odd then it is an interior point otherwise exterior point.





Odd-Even Rule

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 proceeds outward pixel by pixel until the boundary color is encountered.

- It is useful in interactive painting packages, where interior points are easily selected.

- The inputs of the this algorithm are:
  - Coordinates of the interior point (x, y)
  - Fill Color
  - Boundary Color

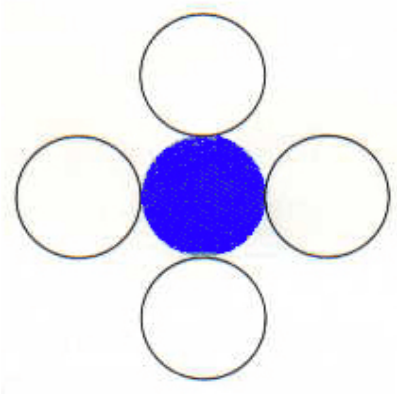
Boundary-Fill Algorithm (cont.)

- Starting from (x, y), the algorithm tests neighboring pixels to determine whether they are of the boundary color. If not, they are painted with the fill color, and their neighbors are tested. This process continues until all pixels up to the boundary have been tested.

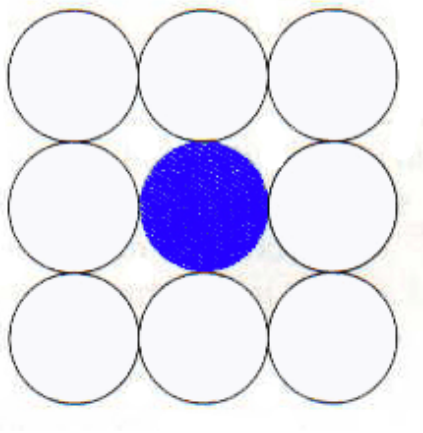
- There are two methods for proceeding to neighboring pixels from the current test position:

Boundary-Fill Algorithm (cont.)

1. The 4-connected method.

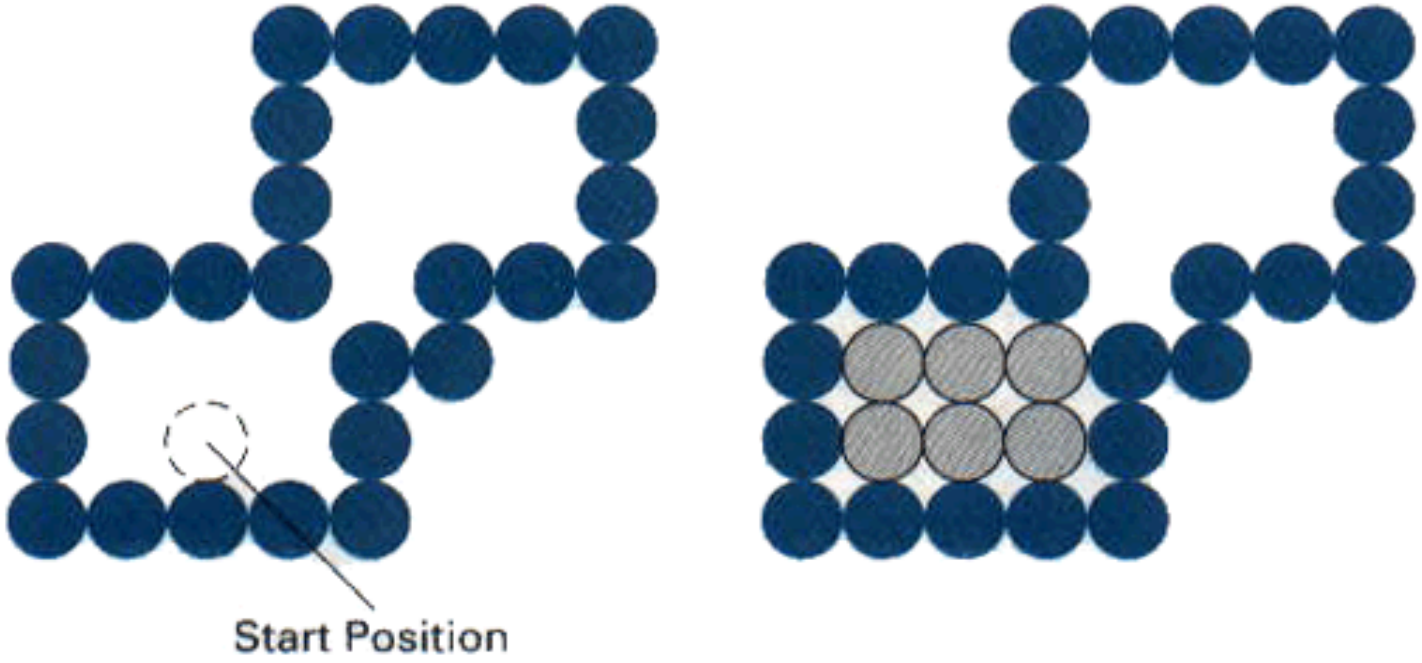


2. The 8-connected method.



```
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+l, y, fill, boundary);
boundaryfill4 (x-1, y, fill, boundary) :
boundaryfill4 (x, y+l, fill, boundary);
boundaryfill4 (x, y-1, fill, boundary) ;
}
}
```

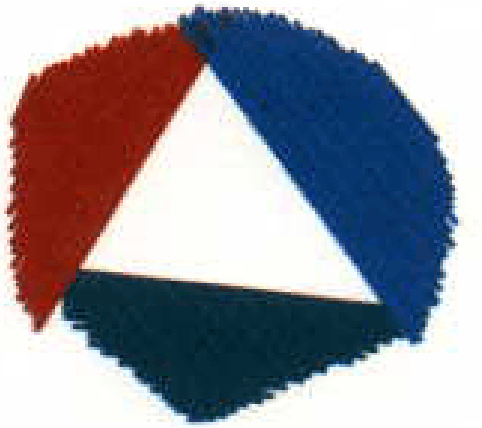
Boundary-Fill Algorithm (cont.)



The area defined within color boundary in (a) is only partially filled in (b)
using a 4-connected boundary-fill algorithm

Flood-Fill Algorithm

- Sometimes we want to fill in (or recolor) an area that is not defined within a single color boundary. We can 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 (cont.)

- We start from a specified interior point (x, y) and reassign all pixel values that are currently set to a given interior color with the desired fill color.

- If the area we want to paint has more than one interior color, we can first reassign pixel values so that all interior points have the same color. Using either a 4-connected or 8-connected approach, we then step through pixel positions until all interior points have been repainted.

Flood-Fill Algorithm (cont.)

```
void floodFill4 (int x, int y, int fillColor, int interiorColor)
{     int color;
      /* set current color to fillColor, then perform following operations. */
      getPixel (x, y, color);
      if  (color = interiorColor)
      {
            setPixel (x, y);   // set color of pixel to fillColor
            floodFill4  (x + 1, y, fillColor, interiorColor);
            floodFill4 (x - 1,   y, fillColor, interiorColor);
            floodFill4 (x, y + 1,  fillColor, interiorColor);
            floodFill4 (x, y - 1,   fillColor, interiorColor);
      }
}
```

Application

Flood fill and Boundary fill is an algorithm that determines the area connected to a given node in a multi-dimensional array.

It is used in the "bucket" fill tool of paint programs to determine which parts of a bitmap to fill with color, and in games such as Go and **Minesweeper** for determining which pieces are cleared.

Scope of research

- A hybrid between scan line and z buffer that it with the active edge table sorting, and instead rasterizes one scanline at a time into a Z-buffer, maintaining active polygon spans from one scanline to the next.