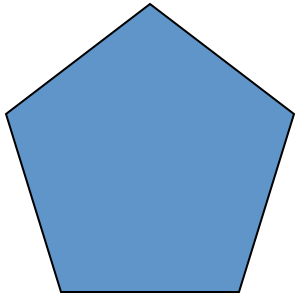
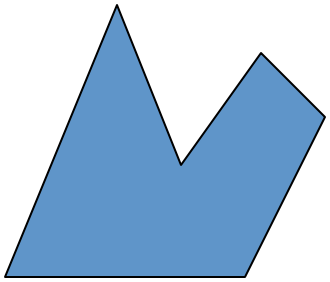
Polygon filling Methods

Introduction to Polygons

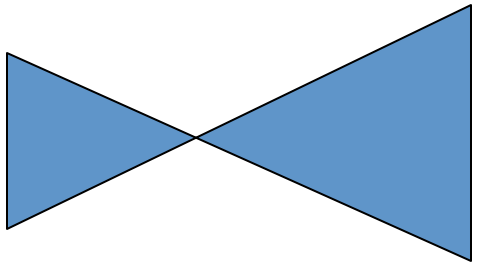
- ***Different types of Polygons***
  - Simple Convex
  - Simple Concave
  - Non-simple : self-intersecting
  - With holes



Convex



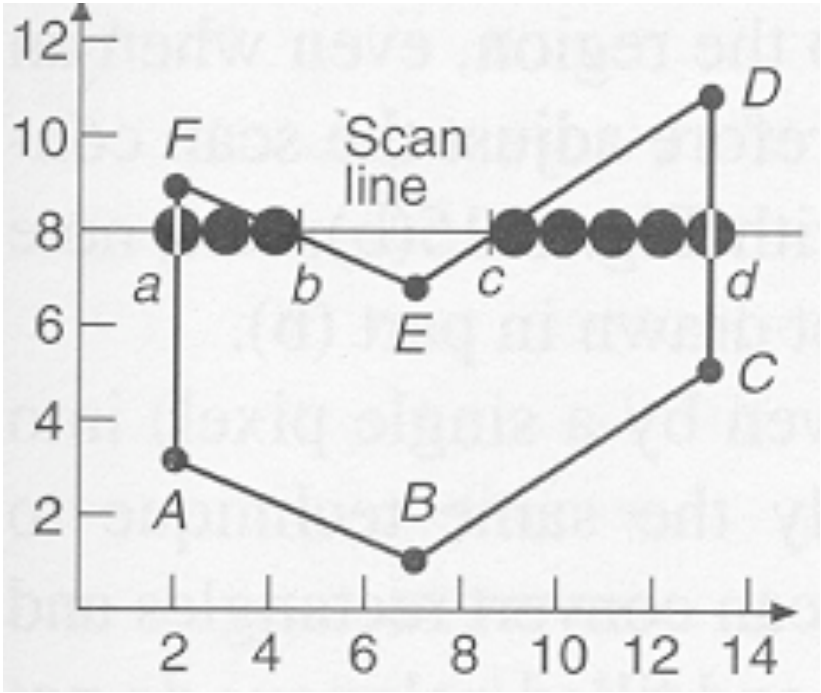
Concave



Self-intersecting

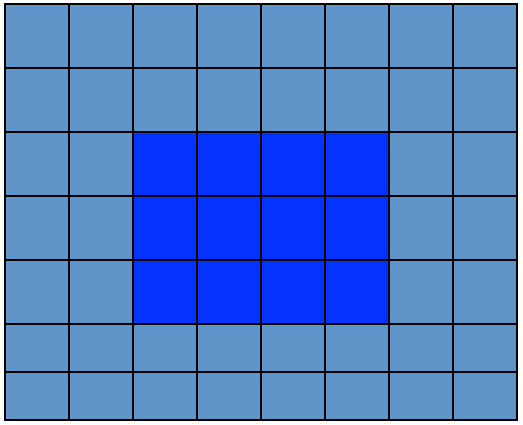
Two basic approaches

- **Scan-line approach**
  - Take successive scan lines that cross the area and fill in the spans of adjacent
  - pixels that lie inside the area from left to right
  - Typically used to fill polygons, circles, and other simple curves

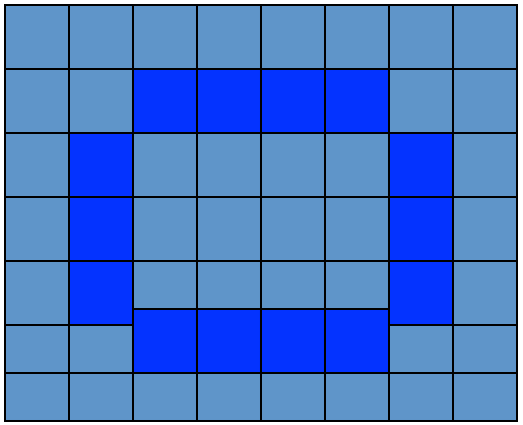


Two basic approaches

- **Seed-fill (boundary fill)**
  - These algorithms assume that at least one pixel interior to a polygon or region is known
  - Regions maybe interior or boundary defined



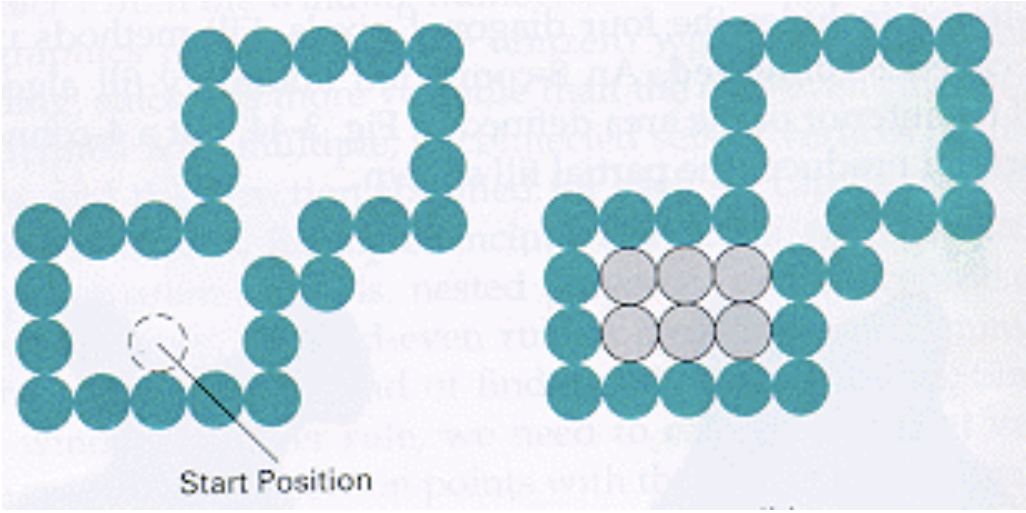
Interior-defined region



Interior-defined region

Two basic approaches

- **Seed-fill (boundary fill)**
  - Start from a given interior position and paint outward from this position until we encounter the boundary
  - Typically used to fill more complex boundaries



Scan-converting Polygons

- Spatial coherence.
- Scan line coherence.
- The characteristics of pixels on a given scan line change only where a polygon edge intersects the scan line. These intersections divide the scan line into regions.

Scan-converting Polygons

- Horizontal edges cannot intersect a scan line and are thus ignored.
- This does not mean that. horizontal edges are not formed.
- They are formed by the bottom and top edges of the rows of pixels.

A Simple Ordered Edge List Algorithm

- **A simple ordered edge list algorithm:**
- Prepare the data:
  - Determine for each polygon edge the intersections with the half interval scan lines. Horizontal edges are ignored. Store each intersection (x, y+1/2) in a list.
  - Sort the list by scan line and increasing x on the scan line; i.e., (x1, y1) precedes (X2, Y2) if Y1 >Y2 or Y1 = Y2 and x1≤ x2.
- Scan-convert the data:
  - Extract pairs of elements from the sorted list (x1,y1) and (x2,y2). The structure of the list ensures that y =y1 = y2 and X1 ≤ X2.
  - Activate pixels on the scan line y for integer values of x such that X1 ≤ X + 1/2 ≤ X2.

More Efficient Ordered Edge List Algorithms

- The simple algorithm given in the previous section generates a large list which must be sorted.
- Making the sort more efficient improves the algorithm. This is accomplished by separating the vertical scan line sort in y from the horizontal scan line sort in x; by using a y bucket sort, the algorithm is now:
- A more efficient ordered edge list algorithm:
  - Prepare the data:
    - Determine for each polygon edge the intersections with the half interval scan lines, i.e., at y + 1/2. Ignore horizontal edges. Place the x; coordinate of the intersection in the bucket corresponding to y.
    - As each scan line is addressed, i.e., for each y bucket, sort the list of x intersections into increasing order; i.e., x1 precedes x2 if $x1 \leq x2$.
- Scan-convert the data:
  - For each scan line, extract pairs of intersections from the x-sorted list. Activate pixels on the scan line y corresponding to that bucket for integer values of x such that $x1 \leq x + 1/2 \leq x2$.

The Edge Fill Algorithm

Algorithm :

- For each scan line intersecting a polygon edge at (x1, y1) complement all pixels whose midpoints lie to the right of (x1, y1), i.e., for (x, y1), X + 1/2 > X1.
- The order in which in which the polygon edges are considered is unimportant.

The fence fill algorithm

Algorithm:

- For each scan line intersecting a polygon edge:
- If the intersection is to the left of the fence, complement all pixels having a midpoint to the right of the intersection of the scan line and the edge, and to the left of the fence.
- If the intersection is to the right of the fence, complement all pixels having a midpoint to the left of or on the intersection of the scan line and the edge, and to the right of the fence.

The Edge Flag Algorithm

Contour outline:

- Using the half scan line convention for each edge intersecting the scan line, set the leftmost pixel whose midpoint lies to the right of the intersection, i.e., for $x + 1/2 > X_{interaection}$, to the boundary value.

Fill:

- For each scan line intersecting the polygon

```
Inside = FALSE
    for x = 0 (left) to X = Xmax (right)
     if the pixel at x is set to the boundary value then
        negate Inside
     end if
    if Inside = TRUE then
        set the pixel at x to the polygon value
    else
        reset the pixel at x to the background value
    end if
    next x
```

Seed Fill Method

- The algorithms discussed in the previous sections fill the polygon in scan line order.
- A different approach is used in the seed fill algorithms. The seed fill algorithms assume that at least one pixel interior to a polygon or region is known.
- The algorithm then attempts to find and color or fill all other pixels interior to the region. Regions may be either inter or boundary-defined.

A Simple Seed Fill Algorithm

- Simple seed fill algorithm using a stack:
  - Push the seed pixel onto the stack
  - While the stack is not empty, pop a pixel from the stack.
  - Set the pixel to the required value
  - For each of the 4-connected pixels adjacent to the current pixel, check if it is a boundary pixel, or if it has already been set to the required value.
  - In either case, ignore it. Otherwise, push it onto the stack.
- The algorithm can be modified for 8-connected regions by looking at the 8- connected pixels rather than only the 4-connected pixels.

A Scan Line Seed Fill Algorithm

Algorithm:

- A seed pixel on a span is popped from a stack containing the seed pixel.
- The span containing the seed pixel is filled to the right and left of the seed pixel along a scan line, until a boundary is found.
- The algorithm remembers the extreme left and the extreme right pixels in the span as X*left* and X*right*.
- In the range of X*left* ≤ x ≤ X*right*, the scan lines immediately above and immediately below the current scan line are examined to see if they completely contain either boundary pixels or previously filled pixels.
- If these scan lines do not contain either boundary or previously filled pixels, then in the range X*left* ≤ x ≤ X*right* the extreme right pixel in each span is marked as a seed pixel and pushed onto the stack.
- The algorithm is initialized by pushing a single seed pixel onto the stack and is complete when the stack is empty.
- The algorithm jumps holes and indentations in the region boundary,

Flood fill algorithm

- Sometimes we want to fill in or recolor an area that is not defined within single color boundary.
- We can paint those areas by replacing a specified interior color instead of searching for a boundary color value.
- This approach is called flood fill algorithm.
- We start from a specified interior point (x. y) and reassign all pixel value 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.
- The following procedure flood fills a 4-connected region recursively starting from the input position.

Flood fill algorithm

```
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 )
            floodFill4 ( x ,   y-1,  fillcolor , oldcolor )
    }
}
```