Polygon Filling

Filling is a process of colouring in a fixed area or region. There are two basic approaches to area filling on raster systems. One way to fill an area is to determine the overlap interval for scan lines that cross the area i.e. scan line fill algorithm

Another method for area filling is to start from a given interior position and point outward from this point, until we encounter the specified boundary conditions.

Area or region may be defined at pixel level or geometric level. When the regions are defined at pixel level, we are having different algorithms like:

- Boundary fill
- Flood fill
- Edge fill
- Fence fill

In case of geometric level, we are having scan line fill algorithm

4-Connected and 8-Connected Pixel Concept:

These are the techniques in which pixels are considered as connected to each other. In 4-Connected method, the pixels may have upto 4-neighbouring pixels as right, above, left and below of the current pixel as shown in figure below. Similarly in 8-Connected method

the pixel may have upto 8-neighbouring pixels as shown in figure.

By using any one of these techniques we can fill the interior of the polygon.

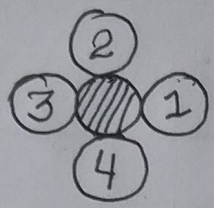


Fig: 4 Connected

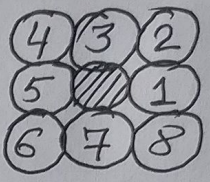


Fig: 8-Connected

Boundary fill algorithm

This is a recursive algorithm that begins with a starting pixel called a seed, inside a region and continues painting towards the boundary. The algorithm checks to see if this pixel is a boundary pixel or has already fill. If answer is no, it fills the pixel and makes a recursive call to itself using each and every neighbouring pixel as a new seed. If the answer is yes, the algorithm simply return to its caller.

A boundary fill procedure accepts as input, the coordinates of an interior points (x, y), a fill ~~colour~~ color and a boundary color starting from (x, y), the procedure test neighbouring positions to determine whether they are of boundary color, if not they are painted with the fill color and their neighbours are tested. This process continues until all pixels upto the boundary color for the area have been tested. There may be two methods for proceeding to neighbouring pixel from the seed point.

The following procedure illustrates a recursive method for filling a 4-connected area. The boundary fill method requires the coordinate of a starting point (x,y), a fill color and boundary color as argument.

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

Recursive boundary fill algorithm may not fill regions correctly if some interior pixels are already displayed in the fill color. This occurs because the algorithm checks next pixel both for boundary color and for fill color. Encountering a pixel with with the fill color can cause a recursive branch to terminate, leaving other interior pixel unfilled. To avoid this, we can first change the color of any interior piels that are initially set to the fill color before applying the boundary fill procedure

The boundary fill algorithm is having limitations. It fills the polygon having unique boundary color. If the polygon is having boundaries with different colors then this algorithm fails.

Flood Fill algorithm

This algorithm also begins with a seed (starting pixel) inside the region. It checks to see if the pixels has the regions original color. If the answer is yes, it fills the pixel with a new color and uses each of the pixels neighbours as a new seed in a recursive call. If answer is no, it returns to the caller.

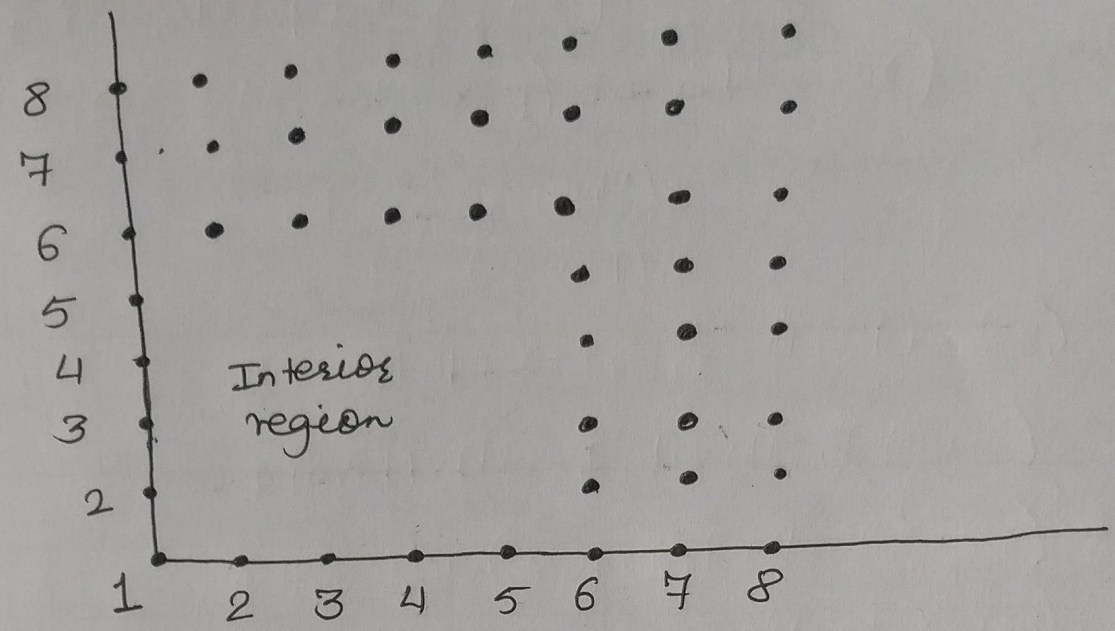
This method shares the great similarities in its operating principle with the boundary fill algorithm. It is particularly useful when the region to be filled has no uniformely coloured boundary i.e. fill in an area that is not defined within a single color boundary. So, in this case, we can paint such areas by replacing a specified interior color instead of searching for a boundary color value. If the area we want to paint has more than one interior color, we can first re-assign 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.

```
void FloodFill(int x, int y, int Fill, int oldcolor)
{  if (getpixel(x,y) == oldcolor)
   {     setcolor(Fill);
         setpixel(x,y);
         FloodFill(x+1, y, Fill, oldcolor);
         Floodfill(x-1, y, Fill, oldcolor);
         FloodFill(x, y+1, Fill, oldcolor);
         FloodFill(x, y-1, Fill, oldcolor);
   }
}
```

We can modify procedure floodfill to reduce the storage requirement of the stack by filling horizontal pixels spans as boundary fill algorithm. In this approach, we stack only the begining position for those pixel spans having the value oldcolor. The steps in this modified algorithm are similar to those for boundary fill. Starting at the first position of each span, the pixel values are replaced until a value other than old color is encountered.

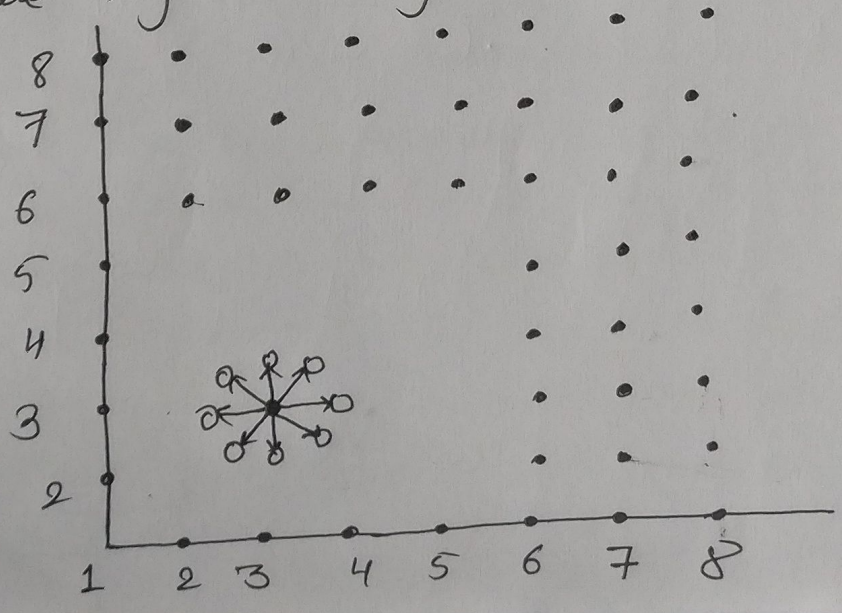
Q. How would a flood fill algorithm fill the region as shown in the figure using the 8-Connected for region pixel?



Solution

Assume that a seed pixel is (3,3). The flood fill algorithm, will inspect the eight points surrounding the seed i.e. (4,4), (3,4), (2,4), (2,3), (2,2), (3,2), (4,2) and (4,3). Since all the points surrounding the seed have the regions original colors each point will be filled.

Now each of eight points becomes a new seed and the points surrounding each new seed are inspected and filled. This process continues until all the points surrounding all the seeds are of the region's original color.



Scan-line Polygon Fill Algorithm

Following figure illustrates the scan-line procedure for solid filling of polygon areas. For each scan line crossing a polygon, the area-fill 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.

In following example, the four pixel intersection positions with the polygon boundaries define two stretches of interior pixels from $x = 10$ to $x = 14$ and from $x = 18$ to $x = 24$

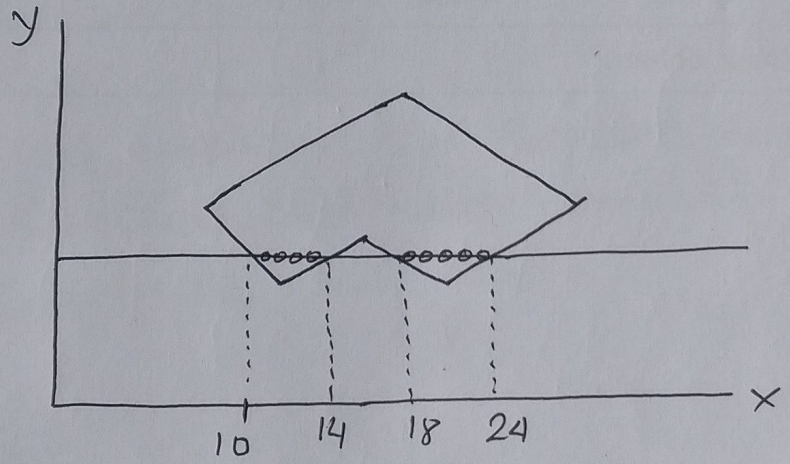


Fig: Interior pixels along a scan line passing through a polygon area.

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. Following figure shows two scan lines at positions y and y' that intersect edge ~~points~~ end points.

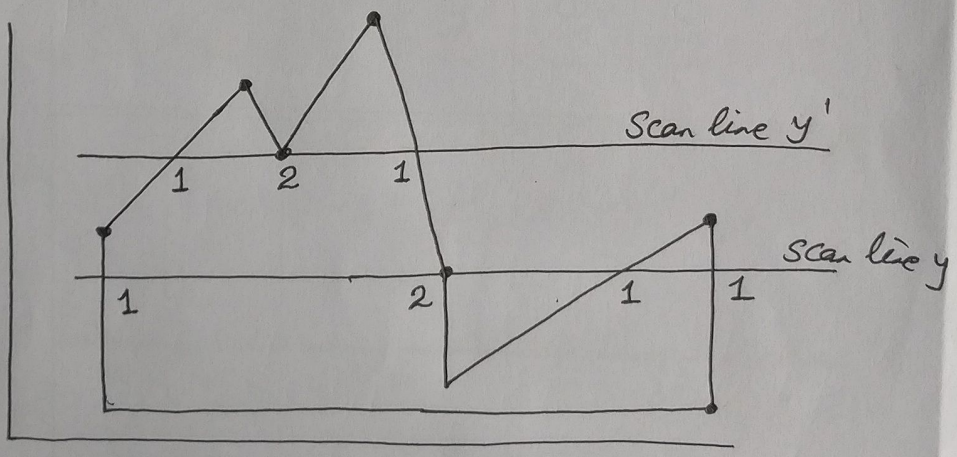
Scan line y intersects five polygon edges. Scan line y', however, intersects an even number of edges although it also passes through a vertex. Intersection points along scan line y' correctly identify the interior pixel spans. But with scan line y, we need to do some additional processing to determine the correct interior points.

The topological difference between scan line y and y' is identified by noting the position of the intersecting edges relative to the scan line.

For scan line y:
- two intersecting edges sharing a vertex are on opposite sides of the scan line

For scan line y':
- two intersecting edges are both above the scan line

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



The slope of this polygon boundary line can be expressed in terms of the scan-line intersection coordinates:

$$m = \frac{y_{k+1} - y_k}{x_{k+1} - x_k}$$

Since the change in y coordinates between the two scan lines is simply:

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

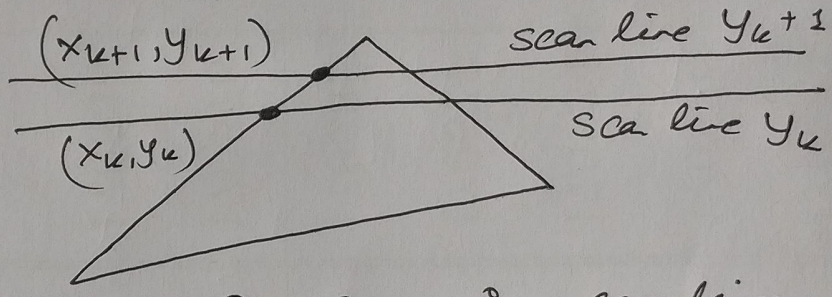


Fig: Two successive scan lines intersecting a polygon boundary

The x-intersection value x_{k+1} on the upper scan line can be determined from the x-intersection value x_k on the preceding scan line as:

$$x_{k+1} = x_k + 1/m$$

In sequential fill algorithm the increment of x values by the amount $1/m$ along an edge can be accomplished with integer operations by recalling that the slope m is the ratio of two integers:

$$m = \frac{\Delta y}{\Delta x}$$

The incremental calculations of x intercepts along an edge for successive scan lines can be expressed as:

$$x_{k+1} = x_k + \frac{\Delta x}{\Delta y}$$