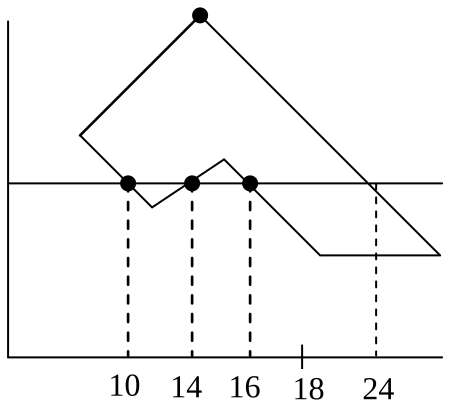
```
{
        for(int i=0;i<=3;i++)
        {
                for(int j=0;j<=5;j++)
                {
                        result[i][j]=0;
                        for(int k=0;k<=3;k++)
                                result[i][j]+=translate[i][k]*vertex[k][j];
                }
        }
}
```

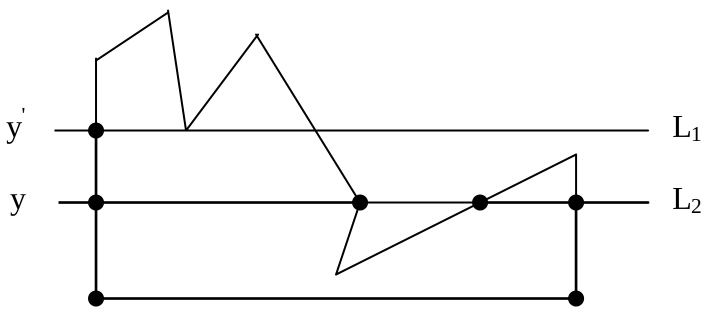
Filled Area primitives

A standard output primitive in general graphics package is solid color or patterned polygon area. Other kinds of area primitives are sometimes available, but polygons are easier to process since they have linear boundaries.
There are two basic approaches to area filling in raster systems. One way to fill an area is to determine the overlap intervals for scan lines that crosses the area. Another method for area filling is to start from a given interior position and point outward from this until a specified boundary is met.

SCAN-LINE Polygon Fill Algorithm:

In scan-line polygon fill algorithm, for each scan-line crossing a polygon, it 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 color. In the figure below, the four pixel intersection positions with the polygon boundaries defined two stretches of interior pixel from x=10 to x=14 and from x=16 to x=24.
some scan-line intersections at polygon vertices require extra special handling.
A scan-line passing through a vertex intersect two polygon edges at that position, adding two points to the list of intersection for the scan-line.



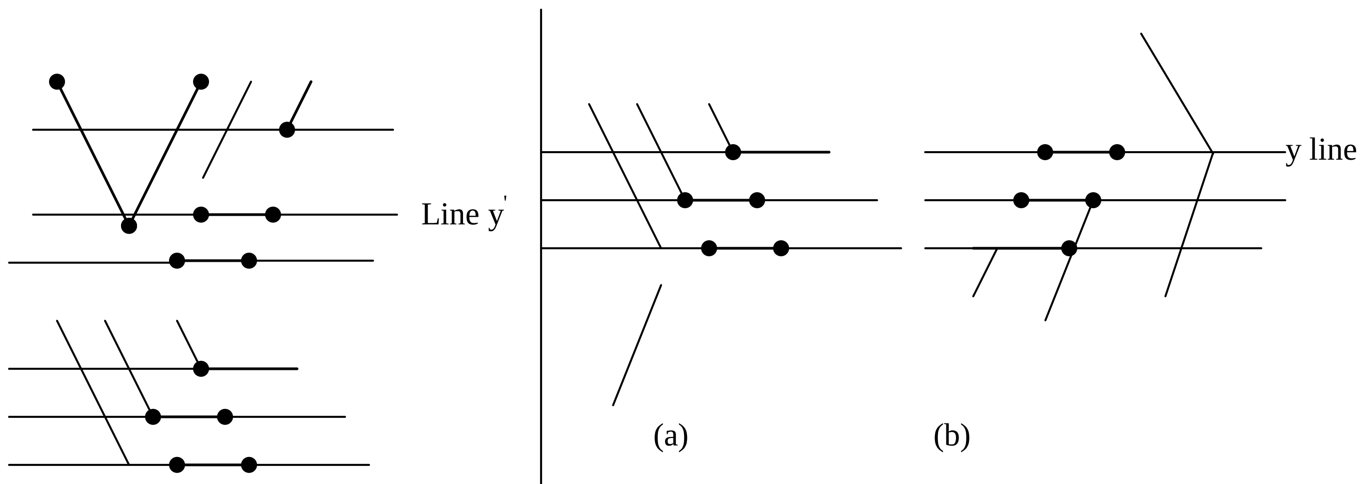


This figure shows two scan lines at position y and y' that intersect the edge points. Scan line at y intersects five polygon edges. Scan line at y' intersects 4 (even numbers) of edges though it passes through vertex.

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.

For scan line y, the two edges sharing the intersecting vertex are on opposite side of the scan-line. But for scan-line y' the two edges sharing intersecting vertex are on the same side (above) the scan line position. So the vertices those are on opposite side of scan line require extra processing.

We can identify these vertices by tracing around the polygon boundary either in clockwise or counter clockwise order and observing the relative changes in vertex y coordinates as we move from one edge to next. If the endpoint y values of two consecutive edges monotonically increases or decrease, we need to count the middle vertex as a single intersection point for any scan line passing through that vertex. Otherwise the shared vertex represents a local extreme (minimum or maximum) on the polygon boundary, and the two edge intersections with the scan-line passing through that vertex.

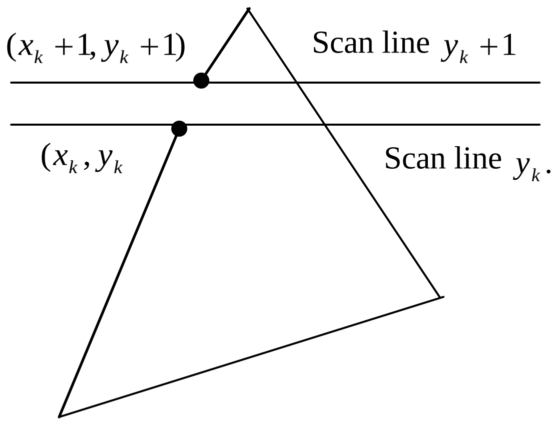


Line y', in which on tracing along edges, the y co-ordinate value is monotonically increasing so the vertex is count as the two intersecting points.

In line y, tracing along edge the y-coordinate of one edge increasing in (a) and the other edge is decreasing in (b) so this vertex is count as a single intersection point for the scan-line fill algorithm.

In successive scan lines crossing a left edge of a polygon, The slope of this polygon boundary line can be expressed in terms of scan-line intersection co-ordinates:

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



Since the change between two scan line in y co-ordinates is 1,

$$y_k + 1 - y_k = 1$$

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

$$x_k + 1 = x_k + \tfrac{1}{m}$$

Each successive x intercept can thus be calculated by x values by the amount of $\tfrac{1}{m}$ along an edge can be accomplished with integer operation by recalling that the slope m is the ratio to two integers

$$m = \Delta y / \Delta x$$

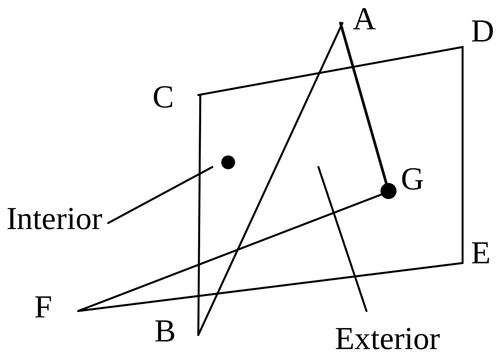
Where $\Delta x \,\&\, \Delta y$ are the differences between the edge endpoint x and y co-ordinate values. Thus incremental calculations of x intercepts along an edge for successive scan lines can be expressed as

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

Inside-Outside Test:

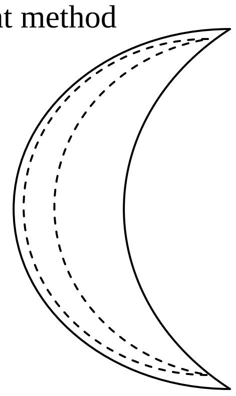
Area filling algorithms and other graphics package often need to identify interior and exterior region for a complex polygon in a plane. For ex. in figure below, it needs to identify interior and exterior region.

We apply odd-even rule, also called odd-parity rule. To identify the interior or exterior point, we can draw a line from a point p to a distant point outside the co-ordinate extents of the object and count the number of intersecting edge crossed by this line. If the intersecting edge crossed by this line is odd, P is interior otherwise P is exterior.



Scan-Line Fill of Curved Boundary area

It requires more work then polygon filling, since intersection calculation involves nonlinear boundary for simple curves as circle, eclipses, performing a scan line fill is straight forward process. We only need to calculate the two scan-line intersection on opposite sides of the curve. then simply fill the horizontal spans of pixel between the boundary points on opposite side of curve. Symmetries between quadrants are used to reduce the boundary calculation we can fill generating pixel position along curve boundary using mid point method

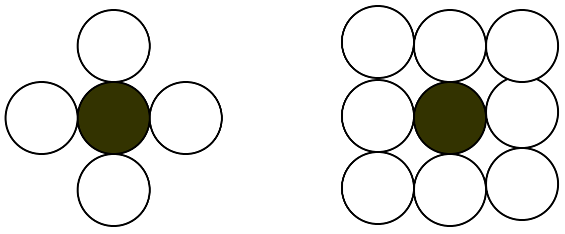


Boundary-fill Algorithm:

In Boundary filling algorithm starts at a point inside a region and paint the interior outward the boundary. If the boundary is specified in a single color, the fill algorithm proceeds outward pixel by until the boundary color is reached.

A boundary-fill procedure accepts as input the co-ordinates of an interior point (x,y), a fill color, and a boundary color. Starting from (x,y), the procedure tests 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 continue until all pixel up to the boundary color area have tested.

The neighbouring pixels from current pixel are proceeded by two
method: 4- connected if they are adjacent horizontally and vertically.
8- connected if they adjacent horizontally, vertically and diagonally.


4- Connected 8- Connected

- Fill method that applies and tests its 4 neighbouring pixel is called 4- connected.
- Fill method that applies and tests its 8 neighbouring pixel is called 8- connected.

The out line of this algorithm is:

```
void   Boundary_fill4(int   x,int   y,int   b_color,   int
fill_color)
{
     int value=get pixel (x,y);
     if (value! =b_color&&value!=fill_color)
     {
          putpixel (x,y,fill_color);
          Boundary_fill 4 (x-1,y, b_color, fill_color);
          Boundary_fill  4  (x+1,y,  b_color,  fill_color);
               Boundary_fill     4     (x,y-1,     b_color,
fill_color);
          Boundary_fill  4  (x,y+1,  b_color,  fill_color);

      }
 }
```

Boundary fill 8- connected:

```
void   Boundary-fill8(int   x,int   y,int   b_color,   int
fill_color)
{
     int current;
     current=getpixel (x,y);
     if (current !=b_color&&current!=fill_color)
     (    putpixel (x,y,fill_color);
          Boundary_fill8(x-1,y,b_color,fill_color);
          Boundary_fill8(x+1,y,b_color,fill_color);
          Boundary_fill8(x,y-1,b_color,fill_color);
          Boundary_fill8(x,y+1,b_color,fill_color);
          Boundary_fill8(x-1,y-1,b_color,fill_color);
          Boundary_fill8(x-1,y+1,b_color,fill_color);
          Boundary_fill8(x+1,y-1,b_color,fill_color);
          Boundary_fill8(x+1,y+1,b_color,fill_color);
     }
}
```

Recursive boundary-fill algorithm not fill regions correctly if some interior pixels are already displayed in the fill color. Encountering a pixel 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 pixels that are initially set to the fill color before applying the boundary fill procedure.

Flood-fill Algorithm:

Flood_fill Algorithm is applicable when we want to fill an area that is not defined within a single color boundary. If fill area is bounded with different color, we can paint that area by replacing a specified interior color instead of searching of boundary color value. This approach is called 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 desired fill_color.

Using either 4-connected or 8-connected region recursively starting from input position, The algorithm fills the area by desired color.

Algorithm:

```
void   flood_fill4(int   x,int   y,int   fill_color,int
old_color)
{
```