Assignment:

Boundary Fill Method:

Boundary fill algorithm starts at a pixel inside of polygon to be filled & paints the interior proceeding outwards towards the boundary. This algorithm works only if the colour with which the region are different. If the boundary is one of single colours, this approach proceeds outwards pixel by pixel until it hits the boundary of the region.

Boundary fill algorithm is recursive in nature. It takes an interior point (x, y) a fill colour & a boundary colour as the input. The algorithm starts by checking the colour of (x, y). If its colour is not equal to fill colour and function is called for all neighbours of (x, y). If a point is found to be fill

colours, or of boundary colour the function does not call its neighbours & returns. This process continues until all points upto boundary colour for the region have been tested.

The boundary fill algorithm can be implemented by 4-connected pixels or 8-connected pixels.

4-connected:

```
void boundaryFill4(int x, int y, int fill_colour, int boundary_colour)
{
    if(getpixel(x,y) != boundary_colour && getpixel(x,y) != fillcolour)
    {
        putpixel(x, y, fill_color);
        boundaryFill4(x+1, y, fill_color, boundary_color);
        boundaryFill4(x, y+1, fill_color, boundary_color);
        boundaryFill4(x-1, y, fill_color, boundary_color);
        boundaryFill4(x, y-1, fillcolor, boundary_color);
    }
}
```

8-Connected:

```
void boundaryFill8(int x, int y, int fill_color, int boundary_color)
{
    if(getpixel(x,y) != boundary_colour && getpixel(x,y) != fillcolour)
    {
        putpixel(x, y, fill_color);
        boundaryFill8(x+1, y, fillcolor, boundary_color);
        boundaryFill8(x, y+1, fill_color, boundary_color);
        boundaryFill8(x-1, y, fillcolor, boundary_color);
        boundaryFill8(x, y-1, fillcolor, boundary_color);
        boundaryFill8(x-1, y-1, fill_color, boundary_color);
        boundaryFill8(x-1, y+1, fill_color, boundary_color);
        boundaryFill8(x+1, y-1, fillcolor, boundary_colour);
        boundaryFill8(x+1, y+1, fill_color, boundary_color);
    }
}
```

Flood fill algorithm:

In this method, a point are fixed, which is inside region is selected. This point is called a seed point. The flood fill algorithm has many character similar to boundary fill.

4-Connected:

```
Procedure  Floodfill (x, y, fill_color, old_color, integer).
{
    if (getpixel (x, y) = old_color)
    {
        setpixel (x, y, fillcolor).
        fill (x+1, y, fill_color, old_color).
        fill (x-1, y, fill_color, old_color)
        fill (x, y+1, fill_color, old_color)
        fill (x, y-1, fill_color, old_color).
    }
}
```

8-Connected:

```
Procedure  Floodfill (x, y, fill_color, old_color, integer).
{
    if (getpixel (x, y) = old_color)
    {
        setpixel (x, y, fill_colour).
        fill (x+1, y, fill_color, old_color).
        fill (x-1, y, fill_color, old_color).
        fill (x, y+1, fill_color, old_color).
        fill (x-1, y-1, fill_color, old_color).
        fill (x-1, y+1, fill_colour, old_colour)
        fill (x+1, y-1, fill_colour, old_colour).
        fill (x+1, y+1, fill_colour, old_colour).
    }
}
```

Scanline polygon fill algorithm:

Scanline algorithm is a process of filling regions of a polygon from geometrically defined by the coordination of waiting of this polygon graph. This algorithm is specifically used for region filling just like the boundary fill & flood-fill algorithm.

For a scanline polygon filling there are 3 steps to perform in following order.

1. Find intersections of the scan-line with all edges of polygon.
2. Sort intersections by increasing x-coordinates from left to right.
3. Make pairs of the intersection & fill in colours within all pixels inside the pair.

The slope of the polygon boundary line can be expressed in terms of the scan-line intersection co-ordinates.

$$m = \left(y_{k+1} - y_k\right)\left(x_{k+1} - x_k\right).$$

Since the change in y-coordinates b/w 2 scan lines is simply $y_{k+1} - y_k = 1$.

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

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

Each successive intercept can be calculated by adding the inverse of the slope and rounding to nearest integer.